Databáze

Luboslav Lacko

Datové sklady analýza OLAP a dolování dat

s příklady v SQL Serveru a Oracle

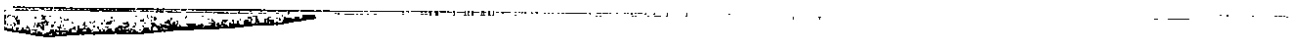
Přiložené CD obsahuje cvičnou verzi SQL Serveru 2000, nástroje Easy Olap od firmy Peyo a soubory k příkladům

computer press

www.knihy.cpress.cz

22049/9788024622049 L

+ CD



Luboslav Lacko

Databáze: datové sklady, OLAP a dolování dat s příklady v Microsoft SQL Serveru a Oracle

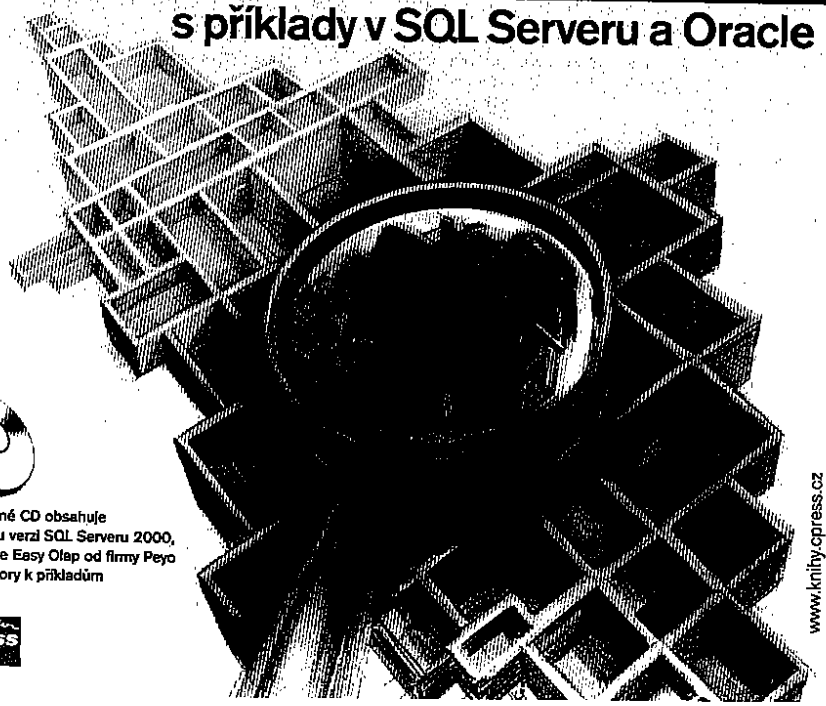
Computer Press
Brno
2003

Databáze

Luboslav Lacko

Datové sklady analýza OLAP a dolování dat

s příklady v SQL Serveru a Oracle



Přiložené CD obsahuje cvičnou verzi SQL Serveru 2000, nástroje Easy Olap od firmy Peyo a soubory k příkladům

computer press

www.knihy.cpress.cz

Databáze: datové sklady, OLAP a dolování dat s příklady v Microsoft SQL Serveru a Oracle

Luboslav Lacko

Vydalo nakladatelství Computer Press jako svou 1262. publikaci.

Vydavatelství a nakladatelství Computer Press[a],
Nám. 28. dubna 48, 635 00 Brno, http://www.cpress.cz

ISBN 80-7226-969-0

Prodejní kód: K0788

Překlad: Tomáš Eder
Odborná korektura: Tomáš Eder
Jazyková korektura: Petra Láníčková
Sazba: Vladimír Ludva
Rejstřík: Luboslav Lacko
Obálka: Martin Sodomka

Komentář na zadní straně obálky: Ivo Magera
Technická spolupráce: Jiří Matoušek, Pavlína Bauerová, Petr Klíma, Petr Chládek
Odpovědný redaktor: Ivo Magera
Technický redaktor: Petr Klíma
Produkce: Petr Baláš

Computer Press, a.s., Nám. 28. dubna 48, 635 00 Brno
tel.: 546 122 111, fax: 546 122 112
Objednávejte na: **www.knihy.cpress.cz**
distribuce@cpress.cz

Bezplatná telefonní linka: **800 555 513**

Dotazy k vydavatelské činnosti směřujte na: **knihy@cpress.cz**
Objednávat můžete na adresách vydavatelství nebo přímo na: **www.knihy.cpress.cz**

Máte-li zájem o pravidelné zasílání informací o knižních novinkách do Vaší e-mailové schránky, zašlete nám jakoukoli, i prázdnou zprávu na adresu **novinky@cpress.cz.**

Obsah

Úvod

Tato publikace je určena čtenářům, kteří se již orientují v problematice databází a potřebují se seznámit s pokročilejšími databázovými technologiemi, jako jsou datové sklady (data warehouse) analýzy OLAP a dolování údajů (data mining). Věta: „...orientují se v problematice databází..." neznamená, že je nutné mít zkušenosti s administrací databázových serverů, případně zkušenosti s vývojem databázových aplikací. Problematika databází a datových skladů zahrnuje nejen technologické aspekty, ale i management, plánování, návrh, zavedení a řízení konkrétního projektu databáze nebo datového skladu. Publikace je koncipována spíše jako praktická příručka, rozhodně nenahradí v této oblasti učebnici ekonomie ani učebnici managementu, ale může být jejich vhodným doplňkem.

...šedá je teorie a zelený je strom praxe...

Teoretické základy a užívané principy jsou vysvětlovány převážně na platformě Microsoft SQL Server (pro tento produkt budeme v publikaci používat i zkrácený název MS SQL Server), neboť k publikaci je přibaleno CD se 120denní Trial verzí databázového serveru Microsoft SQL Server 2000 včetně analytických služeb. Proto jsou z důvodu maximální dostupnosti všechny ukázky a příklady vytvořeny pro tuto databázovou a analytickou platformu. Celá jedna poměrně rozsáhlá kapitola je věnována i platformě Oracle 9i, a to z hlediska přípravy údajů pro analýzu OLAP.

Důležité jsou však principy, ne technologie. Když porovnáme například vytvoření krychle OLAP pomocí průvodce na obou platformách, tak zjistíme, že jednotlivé dialogy jsou až na designové úpravy prakticky totožné. Všechny příklady v knize jsou koncipovány tak, aby je bylo možné vyzkoušet na běžném PC nebo notebooku. Pro zajímavost uvádíme, že všechny příklady z této knihy byly realizovány a odzkoušeny na notebooku s procesorem Pentium III taktovaným na 1 GHz a 384 megabajty operační paměti, což je v době vydání knihy hardwarová konfigurace hluboko pod optimálním poměrem cena/výkon.

Nástrojů a aplikací pro budování datových skladů, analýz OLAP a Data Mining je na trhu nepřeberné množství a nebylo by v možnostech rozsahu této publikace je všechny představit. A kdybychom některé z nich, například Cognos, do publikace zahrnuli, logicky by vyvstala otázka, proč právě tento produkt, a ne jiný. Proto jsme výběr cíleně zúžili na analytické nástroje integrované do databázových serverů. Přece vždy analyzujeme a zpracováváme údaje uložené v databázích. Výběr platformy Microsoft SQL Server a hlavně jeho analytických služeb nebyl náhodný. I jiné databázové platformy, například Oracle 9i nebo IBM DB2, sice obsahují analytické nástroje, ale tyto databáze a analytické služby lépe běží na podnikových serverech než na běžných desktopech a noteboocích. Naproti tomu o hardwarových požadavcích platformy Microsoft SQL server by se dalo říci, že jsou na dnešní dobu opravdu minimální. Postačí 128 MB paměti RAM a pro zajímavost uvádíme, že instalace analytických služeb zabere na disku jen okolo 80 megabajtů.

Takže předpokládáme, že i čtenáři, kteří používají jiné databázové platformy, sáhnou při prvních pokusech právě po analytických službách MS SQL Serveru. A 120denní trial verze na výuku bohatě vystačí. Nepředpokládáme sice, že této knize bude někdy konkurovat kniha z populární série „Naučme se ... za 21 dní". Ale pro zvládnutí jednotlivých částí, například pro analýzu OLAP nebo Data Mining, je 21 dní dostačujících.

Co v této publikaci nenajdete

Účelem publikace je představení technologií datových skladů, analýz OLAP a data miningu, ne porovnání jednotlivých produktů a platforem.

Kapitola 1

Úvod do problematiky datových skladů, analýzy OLAP a data miningu

Intenzivní nasazení informačních technologií do rozličných odvětví lidské činnosti přináší sebou jeden zákonitý jev, kterým je shromažďování velkého množství různorodých údajů. Shromažďují se údaje z technologických zařízení, firemní administrativy, odbytu a podobně. Množství údajů mají shromážděno i velké supermarkety ze skladových karet, elektronických pokladen a podobně. Výsledek je lehko předvídatelný. Za kratší či delší dobu se podaří shromáždit obrovské množství údajů. Moderní databázové servery umožňují nejen bezpečnou a rychlou práci s takovým množstvím údajů, ale umožňují nám z těchto údajů získat i informace.

Na první, povrchní pohled by se mohlo zdát, že mezi pojmy údaje a informace můžeme postavit znaménko rovnosti. Ale jen na první pohled. Proces transformace údajů na informace a převod těchto informací na poznatky prostřednictvím objevování nazýváme **Business Intelligence**. Jinými slovy účelem Business Intelligence je konvertovat velké objemy údajů na poznatky, které jsou potřebné pro koncové uživatele. Tyto poznatky můžeme potom efektivně využít například v procesu rozhodování.

Pod pojmem informace nerozumíme jen konkrétní záznam nebo množinu záznamů. Často potřebujeme sledovat trend nějaké veličiny, například při obchodování s cennými papíry, nebo potřebujeme najít mezi údaji určité závislosti. Proto moderní databázové servery obsahují rozsáhlou podporu pro budování datových skladů (**data warehouse**), **analýzy OLAP** (Online Analytical Processing) a **data mining** (dolování, odkrývání dat).

Samozřejmě poznatky z obchodování se zpracovávaly i předtím, neboť plány a studie se pro manažery vypracovávaly odjakživa, byly to samozřejmě reporty v papírové podobě. Určitý posun nastal v období 1980 až 1990, když se pro vyhodnocování údajů z obchodování začaly používat různé tabulkové kalkulační programy. Éra produktů pro Business Intelligence začala okolo roku 1990. Dostupnost produktů se postupně zlepšovala, kvalitní analytický server ve 120denní časově omezené verzi najdete i na přibaleném CD, takže toto téma se otevřelo i pro studentské práce a studentské projekty. Nesmíme samozřejmě podléhat iluzím, databázové a analytické servery patří k nejdražším produktům. Ale jak se říká, údaje, ale hlavně kvalitní a dostupné informace jsou jednoduše k nezaplacení.

Přechod od transakčních databází k analytickým

Ještě předtím, než se dostaneme k typickým analytickým systémům, nebude na škodu stručný přehled databázových systémů od transakčních systémů, přes operační taktické systémy, strategické systémy pro podporu rozhodování až po informační systémy pro vrcholové řízení. Obecně je možné říci, že systémy vyšší úrovně v sobě zahrnují i systémy nižší úrovně.

Systémy OLTP (Online Transaction Processing)

Oblast použití transakčních databázových systémů (TPS Transaction Processing Systems) je skoro neomezená. Primárním cílem při jejich návrhu je umožnit klientům databázového serveru vykonávání velkého množství transakcí online, například bankovních, obchodních a podobně. Cílem transakčních databázových systémů je automatizace každodenních činností, které jsou předmětem našeho podnikání, například skladové hospodářství, mzdy, nákup a prodej, případně řízení a monitorování technologických procesů v reálném čase.

Transakční systémy jsou ve firemní praxi velmi často používané a jsou velmi oblíbené, jednak pro svoje výhody, ale i z důvodu existence množství specialistů, ať už administrátorů, nebo vývojářů. V případě, že transakční databázový systém pokrývá většinu podnikových aktivit, nazýváme ho systém **ERP (Enterprise Resource Planning)**. Ke zdroji údajů tedy ve stejném čase přistupuje velké množství uživatelů, kteří údaje z databáze čtou, jiní do něj zapisují, případně někteří vykonávají i jednodušší analýzy. Ano i nad databázemi OLTP je možné vybudovat analytické aplikace OLAP, ale jak se lidově říká: „Není to pravé ořechové." Nejen z teorie, ale i z praktických zkušeností totiž vyplývá, že údaje v transakčních databázích by měly být uloženy v normalizovaných tabulkách, které by měly vyhovovat podmínkám druhé nebo třetí normální formy. To znamená mnoho atomických, relačně svázaných tabulek. Analýza velkého množství takto uložených údajů by byla proto velmi neefektivní a značně pomalá. Také návrh analytické aplikace, hlavně tabulek dimenzí nad databází OLTP není právě jednoduchá záležitost.

Systémy MIS – (Management Information Systems)

Rozhraní mezi relačními a transakčními systémy není ani ostře ohraničené, ani jednoznačné. Od klasických transakčních systémů se dostáváme k systémům pro podporu řízení a rozhodování. Hlavní úlohou těchto systémů je poskytování kvalitních informací pro řídící pracovníky. Informace slouží jako podklad pro řízení. Z úrovně okamžitých transakcí se dostáváme na jakousi firemní „operativní taktickou úroveň". Systémy MIS sice poskytují řídícím pracovníkům různé komplexní přehledy a sestavy, agregované podle různých hledisek, například časových, geografických, organizačních a jiných. Do systémů MIS vstupují údaje z transakčních systémů. Původní manažerské informační systémy poskytovaly vedoucím pracovníkům sestavy poměrně komplikovaným způsobem. Požadavky na sestavu se odeslaly vývojovému týmu MIS, který vytvořil sestavu a poskytl ji manažerům až po určitém čase, zpravidla po několika dnech, týdnech, nebo dokonce až po několika měsících. O užitečnosti takových informací v dnešním uspěchaném podnikatelském světě můžeme oprávněně pochybovat.

Systémy DSS – (Decision-Support Systems)

Jak vyplývá z jejich názvu, jedná se o systémy pro podporu rozhodování. Zpravidla jsou nadstavbou systémů MIS. Na rozdíl od systémů typu MIS, které se nasazují na operativně taktické úrovni, systémy DSS jsou už na rozhraní taktického a strategického rozhodování. Poskytují řídícím pracovníkům například výsledky poměrně složitých analýz. Kromě nasazení DSS na rozhraní operativně taktické a strategické úrovně byl důležitým aspektem i nástup osobních počítačů v osmdesátých letech minulého století, hlavně řady IBM PC. Tento aspekt umožnil přímý přístup k operačním údajům i pro obchodní uživatele.

Systémy EIS – (Executive Information Systems)

Přes operačně taktickou úroveň jsme se dostali v našem přehledu až na strategickou úroveň, k informačním systémům pro vrcholové řízení. Tyto systémy uvádíme v našem přehledu jen informativně, stále častěji jsou totiž nahrazovány systémy DSS pro podporu rozhodování a pravděpodobněji než s termínem EIS se setkáme s termínem Business Intelligence.

OLAP (Online Analytical Processing)

Typické využití systémů OLAP je pro analýzu velkého množství údajů. Výsledkem analýzy jsou souhrny a reporty, které slouží manažerům jako podklady pro jejich rozhodování, ať už v oblasti řízení firmy, řízení ekonomických a technologických procesů a podobně. Pro výpočet krychlí OLAP je potřebné vykonat velké množství výpočtů a agregací, a to skoro v reálném čase.

Kvalita údajů pro analýzy

V současnosti už každý využívá pro svoji činnost nějaký druh ekonomického softwaru, například účetnictví, skladové hospodářství, evidence pohybu zboží a podobně. A co přitom chtě nechtě dělají? Shromažďují údaje, zčásti samozřejmě bezcenné, ale i velmi cenné, které však zůstávají nevyužity, protože jsou uloženy ve formě, která je činí nedostupnými pro účely získávání informací. A tak i přes existenci údajů se řídící pracovníci k potřebným informacím nemusí dostat. A potom ani nejlepší manažer nemůže vykonat odpovídající rozhodnutí.

Můžeme to ukázat na jednoduché analogii. Máme k dispozici tým špičkových architektů a urbanistů a ukážeme jim fotografie náměstí s tím, že by bylo potřeba vylepšit současné architektonické řešení. Po předložení snímků se tým pustí do práce a řešení je zanedlouho k dispozici. Ale představme si, že bychom týmu architektů a urbanistů předložili krabici puzzle, a to by ještě kousky z fotografií náměstí byly promíchány s kousky puzzle náměstí Svatého Marka a puzzle Myšáka Mikiho. I v tomto případě jsou kompletní údaje k dispozici, ale rozdrobené na množství promíchaných mozaikových kousků. Abychom viděli fotografie náměstí, tak bychom museli nejdříve vzpomínanou mozaiku poskládat. To je poměrně zdlouhavá práce i tehdy, když výsledný obraz známe, to ale není je náš případ, my totiž nevíme, jak má vypadat výsledný obraz. Kdybychom to věděli, nemuseli bychom skládat mozaiku.

Stejné je to i v obchodní a firemní praxi. Když máme obchodní firmu třeba i jen střední velikosti, která prodává 1 000 druhů zboží prostřednictvím 10 prodejních kanálů 100 odběratelům, tak se lehce dopočítáme milionu možných kombinací vzpomínaných položek. A kdybychom chtěli sledovat obchodní život firmy po měsících, dostáváme dvanáct milionů možných kombinací. A to jsme stále jen u jednoho ukazatele, například tržeb. Kdybychom chtěli sledovat více ukazatelů, například zisky, výsledky kampaní a podobně museli bychom vzpomínaných dvanáct milionů ještě vynásobit počtem ukazatelů.

Jak tedy řešit takovou situaci? Dvanáct milionů je obrovské číslo, ale vzpomeňme si, jak jsme se k němu dopracovali. Násobením poměrně malých čísel, tedy dimenzí. Když tedy vzpomínané údaje uspořádáme do multidimenzionální struktury, budeme pracovat s o mnoho menšími rozměry dimenzí, a údaje tedy budou mít o mnoho větší vypovídací schopnost. Multidimenzionální princip totiž umožňuje přistupovat k údajům tak, že se můžeme podívat na pro nás ještě obsáhnutelnou množinu údajů. Multidimenzionální informace je navíc v čisté podobě, což znamená, že je orientovaná na předmět podnikání a není vázaná na konkrétní systém, který je určený na sběr základních údajů.

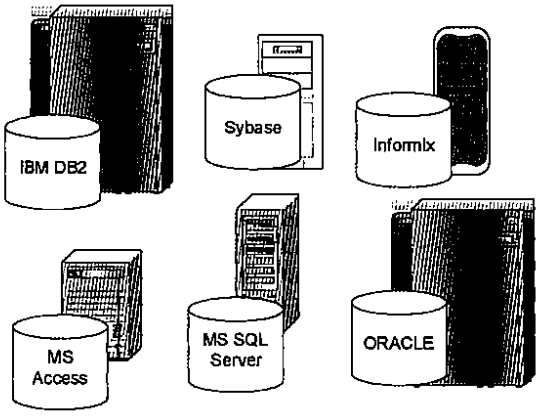
Nevhodnost transakčních databází pro analýzy

Transakční databáze označované i jako databáze OLTP, tak jak je známe z běžné podnikové praxe, jsou určeny pro ukládání operačních údajů. Výsledkem dotazování jsou databázové tabulky, souhrny získané pomocí agregačních funkcí, různé sestavy a podobně. Databáze OLTP jsou z důvodu jednoduchého dotazování a vyloučení redundance zpravidla normalizovány, což znamená, že transakční databáze vyhovují pravidlům tzv. normálních forem.

Struktura operačních údajů v databázích OLTP je ve většině případů komplexní a vysoce strukturovaná (3NF). Tyto systémy dosahují vysokých výkonů spíše při transakcích online než při složitých analýzách, které jsou velmi náročné na výpočetní kapacitu procesorů. Komplexní analýza vyžaduje jiné techniky návrhu databází, například použití multidimenzionálních a hvězdicových schémat s tabulkami faktů, které obsahují měrné jednotky obchodování a vysoce denormalizované tabulky dimenzí. Databázové systémy typu OLTP jsou optimalizované pro obchodní transakce, ale pro získání informací pro podporu rozhodování nejsou příliš vhodné i z jiných důvodů.

Decentralizovanost systémů OLTP

Největší překážkou použití databázových systémů OLTP pro analýzy je skutečnost, že tyto systémy nemají k dispozici integrovaný zdroj údajů ze všech operačních systémů v rámci podniku tak, aby umožnily tvorbu komplexních analýz, což znamená, že potřebné údaje nebo údaje, které by měly sloužit jako podklady pro analýzy, jsou roztroušené v různých zpravidla heterogenních systémech OLTP a musí se pracně integrovat dříve, než je možné získat požadované informace. Časová náročnost případných analýz je u ne příliš složité ani příliš komplexní analýzy poměrně vysoká. Někdy se dokonce ani nepodaří integrovat data mezi jednotlivými systémy, takže vlastně ani nemůžeme získat globální obraz o stavu podnikání.



Obr. 1.1 Decentralizace systémů OLTP

Samozřejmě z technického hlediska nám nic nebrání provádět analýzy údajů z transakčních databází, ale u velkého objemu údajů, který se nahromadí za určitý čas, vytěžení těchto databází není příliš dobré řešení.

Nevýhody použití transakčních databází pro analytické účely bychom mohli shrnout do několika bodů:

- **není jednoduché najít příčiny a vysvětlení problémů**
- **obtížné hledání závislosti jednotlivých veličin**
- **příliš rozsáhlé výstupy z transakčních systémů**

  Někdy se používají standardní výstupní sestavy z produkčních systémů, tyto výstupy jsou při širším zadání parametrů, které pro analýzy potřebujeme, příliš rozsáhlé.
- **dlouhý čas výpočtu a degradace výpočetního výkonu databázového stroje transakční databáze**

  Když používáme pro online zpracování transakcí a analýzy pro podporu rozhodování stejný server, tak tento přístup degraduje výkon použitého hardwaru i operačního systému. Důsledkem je prodloužení času odezvy, a to jednak uživateli vykonávajícímu transakci, jednak i uživateli čekajícímu na výsledek analýzy. Problémy mohou být i s výkonem sítě. Výpočet potřebných agregací souhrnů, predikcí a podobně v transakčních systémech trvá velmi dlouho a v mnohých případech neúměrně zatěžují databázový stroj produkční databáze, takže dochází ke značné degradaci výkonu produkčního systému. Prodloužení doby odezvy transakčního systému je o mnoho větší nevýhoda, než by se na první pohled mohlo zdát, protože transakční systém je primární a přímo životně důležitý. Kdyby tento systém selhal nebo byl zatížen nad únosnou míru, potom by už nebyl žádný důvod pro analýzu údajů. Jednoduše by nebylo z čeho údaje analyzovat.

- **transakční systém neuchovává historické údaje**

  Další typickou vlastností transakčních databází je, že neumožňují uchovávání údajů po delší dobu, jinými slovy řečeno, tyto systémy neukládají v dostatečném rozsahu historické údaje potřebné ke komplexní analýze nebo predikci. Jednoduše disková kapacita provozních počítačů na to nestačí. Mnoho údajů se v některých případech neuchovává vůbec, například období platnosti některých údajů, například valutových kurzů, ceníků a podobně.

- **nehomogenní struktura údajů**

  Ty samé údaje mohou být v různých decentralizovaných systémech uloženy v různých tvarech a formátech. Například záznam o telefonním čísle zákazníka může být v jednom systému uložen jako 15místné číslo, v jiném jako 18znakový řetězec, a to nehovoříme o předvolbách, ať už místních nebo mezinárodních, případně o mezerách mezi skupinami číslic, pomlčce za předvolbou a podobně. Podobné problémy bývají s rodným číslem, někde je uvedeno jako desetimístný řetězec, jinde je po prvních šesti číslicích lomítko a až za ním následují poslední čtyři číslice.

- **dlouhý čas přípravy údajů**

  Ve většině případů se vstupní údaje pro různé analýzy nacházejí v různých zpravidla nehomogenních zdrojích. Postupné připojení k nim není sice ve většině případů z technického hlediska velký problém, ale vyžaduje to nemálo času a námahy. Ovšem analyzovat tyto údaje z více transakčních systémů najednou je z technického hlediska velký problém. A to jsme stále jen u technického hlediska. Kolik špičkových analytiků a marketingových odborníků ovládá jazyk SQL? Narážíme na další bariéru dostupnosti údajů, tentokrát způsobenou lidským faktorem. Analytici proto musí při vypracovávaní analýzy spolupracovat s databázovými specialisty. A to trvá dlouho a také to významně zvyšuje náklady. I když se to podaří, tak po změně struktury některé z transakčních databází jsme tam, kde jsme byli na začátku, a musíme definice příslušných dotazů přeprogramovat.

- **není jednoduché najít příčiny a vysvětlení problémů a obtížné hledání závislostí jednotlivých veličin**

  Ani po překonaní všech naznačených problémů stále nemáme vyhráno. Je totiž problém najít konfliktní hodnoty sledovaných veličin, hlavně když ty závisí na více faktorech. U složitějších jevů není vždy jasné na čem a v jaké míře je sledovaná veličina závislá. Platí to samé jako u Data Miningu – nezahrnutí důležité závislosti může často vést k podstatnému nebo úplnému zkreslení výsledků analýzy.

Přechod od relačních databází k multidimenzionálním

Relační databázový model

Relační databáze, ve kterých vykonáváme velké množství transakcí v reálném čase, například v bankách, supermarketech a podobně, nebo změn údajů při monitorovaní nějaké-

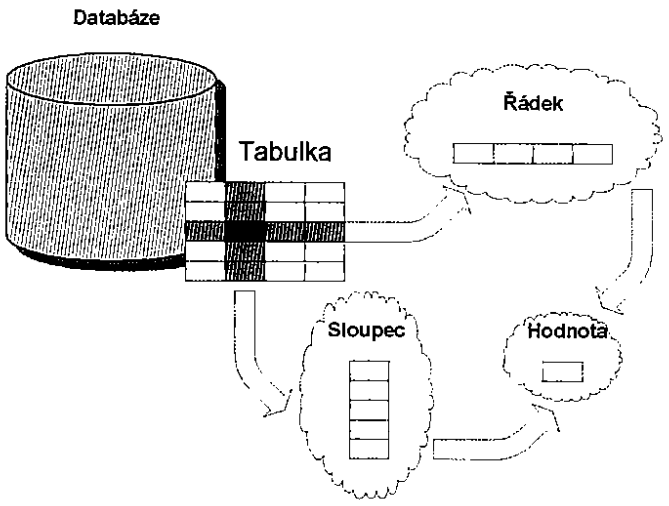
ho procesu, například zápis záznamů o hovorech při monitorovaní provozu mobilního operátora, nazýváme databáze **OLTP** (Online Transaction Processing) nebo podle jiné terminologie i operační databáze, protože se v nich vykonává velké množství operací, nebo někdy i aktivní databáze. Transakční databáze, do kterých se ukládají aktuální operační údaje, jsou organizovány jako relační, což znamená, že údaje jsou uloženy v databázových tabulkách, mezi kterými jsou relační vztahy vyplývající z aplikační logiky. Teorie relačních databází je stará více než 30 let, přičemž základy této teorie položil Dr. E. F. Codd, který, jak v uvidíme v dalších kapitolách, byl i průkopníkem teorie analytických databází. Protože v konečném důsledku údaje, které se ukládají do datových skladů, pocházejí z relačních databází, je důležité znát alespoň základy teorie relačních databází. Nejdříve uvedeme stručné definice základních pojmů.

Databáze

Databázi chápeme jako úložiště údajů, které jsou uloženy a zpracovávány nezávisle na aplikačních programech. Databáze zapouzdřují jednak vlastní údaje, ale i relační vztahy mezi jednotlivými prvky a objekty v databázi, schémata popisující struktury údajů a integritní omezení.

Databázová tabulka

V prostředí relačních databází jsou údaje, v kterých se skrývají potenciální informace, uloženy v klasických dvojrozměrných databázových tabulkách, které se skládají z hodnot umístěných na průsečíku řádků a sloupců. Každý řádek (záznam) v tabulce je zpravidla identifikován pomocí nějakého identifikátoru.



Obr. 1.2 Princip relační databáze

Teorie relačních databází

V úvodu kapitoly byl vzpomenutý průkopník oblasti teorie relačních databází Dr. E. F. Codd. Mezi jeho nejznámější počiny v této oblasti patří definování tří **podmínek minimální relačnosti**:

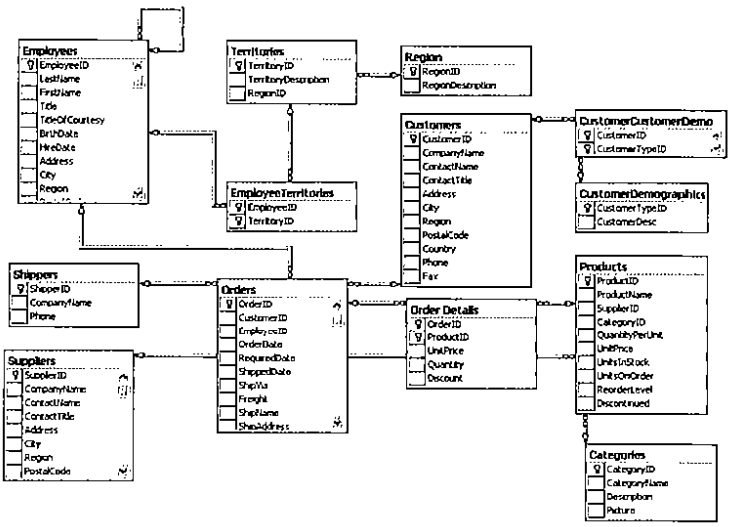
1. *Všechny údaje v databázi jsou uloženy v tabulkách.*
2. *Fyzická struktura údajů a uložení dat je nezávislé a úplně od uživatele odstíněné, což znamená, že neexistují žádné uživateli viditelné přístupové cesty (včetně indexů).*
3. *Pro práci s údaji v databázi předpokládáme existenci databázového jazyka, který umožňuje realizovat minimálně operace selekce, restrikce, projekce a spojení.*

a dvanácti **pravidel pro relační databázové systémy**:

1. *Údaje v relační databázi musí být reprezentovány explicitně na logické úrovni pomocí relačních tabulek.*
2. *Údaje uložené v relační databázi musí být přístupné kombinací názvu tabulky, názvu sloupce a hodnoty primárního klíče. V síťovém nebo distribuovaném databázovém prostředí obvykle udáváme i název databáze.*
3. *Musí existovat indikátor chybějící hodnoty nebo hodnoty, kterou neznáme (rozdílný od čísla nula a prázdného řetězce). Tento indikátor označujeme NULL.*
4. *Databáze musí umožňovat autorizovaným uživatelům přístup nejen k údajům, ale i k jejich popisům (metadatům) pomocí stejného databázového jazyka.*
5. *Databázový jazyk musí být jednoduchý a uživatelsky přívětivý, přičemž musí umožňovat interaktivní i programový režim. Kromě toho musí umožňovat definici údajů, entitních omezení, manipulaci s údaji, definici transakcí a definici přístupových práv.*
6. *Relační databázový systém musí poskytovat způsob definování pohledů a musí umožnit pro tyto pohledy povolit nebo zakázat vkládání a rušení řádků nebo aktualizaci sloupců v základních tabulkách, nad kterými je pohled vytvořen.*
7. *Relační databázový systém musí umožňovat množinové operace s celými tabulkami nejen při vyhledávání, ale i při vkládání, aktualizaci a rušení dat.*
8. *Aplikační logika nesmí vyžadovat modifikaci v případě změny interního uložení nebo metody přístupu k údajům. Tento bod tedy definuje fyzickou datovou nezávislost.*
9. *Aplikační logika nesmí vyžadovat modifikaci v případě změn základních tabulek nevyvolávajících ztrátu informace (zrušení nebo přidání sloupce do tabulky). Tento bod definuje logickou datovou nezávislost.*
10. *Aplikační logika nesmí vyžadovat modifikaci v případě změn integritních omezení definovaných pomocí databázového jazyka a uložených v katalogu dat.*
11. *Aplikační logika nesmí vyžadovat modifikaci v případě, kdy jsou data distribuována na různých počítačích.*
12. *Když má databázový systém nízkoúrovňový (procedurální) programovací jazyk, tak tomuto jazyku nesmí být umožněno rušit nebo měnit omezení definovaná databázovým jazykem.*

Relační vztahy

Aby bylo možné aplikovat relační teorii, je zpravidla nutné vytvořit více databázových tabulek a definovat relační vztahy mezi nimi.



Obr. 1.3 Příklad relační databáze složené z více tabulek (cvičná databáze fiktivní firmy Northwind)

Primární klíč je jednoznačný identifikátor každého záznamu. Může to být sloupec, případně kombinace více sloupců, které slouží pro jednoznačnou identifikaci každého řádku tabulky. Hodnota pole primárního klíče musí být v rámci tabulky jedinečná. Pole primárního klíče musí obsahovat konkrétní hodnotu, tedy nesmí nikdy nabýt hodnoty NULL. Bez primárního klíče není možné definovat relace mezi tabulkami. Při výběru atributů, které budou tvořit primární klíč, je potřeba dbát na to, aby vybrané atributy skutečně plnily úlohu identifikačního klíče, ale na druhou stranu i na to, aby jejich použití bylo efektivní i z časového a paměťového hlediska.

Cizí klíč je sloupec, případně kombinace více sloupců, které jsou propojeny na primární klíč v jiné tabulce.

Vztahy mezi entitami

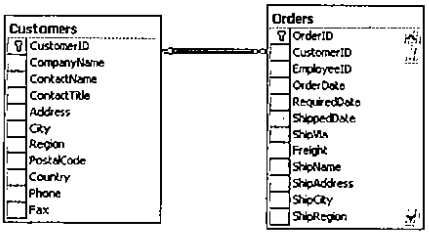
Relace mezi tabulkami vlastně popisují vztahy mezi objekty reálného světa, které tyto tabulky představují. Vzhledem ke kardinalitě budeme entity nazývat první a druhá. Mezi nimi bude vztah (relace).

Vztah jedna k jedné (1:1, one-to-one)

V tomto vztahu první entitě, tedy záznamu v databázové tabulce odpovídá maximálně jedna druhá entita, záznam z jiné databázové tabulky. Tedy každý řádek primární tabulky je možné svázat právě s jedním řádkem sekundární tabulky. Relaci one-to-one zajistíme pomocí unikátních klíčů v obou dvou tabulkách. Do této kategorie vztahů patří i tzv. částečné vztahy 1:0 a 0:1.

Vztah jedna k mnoha (1:N, one-to-many)

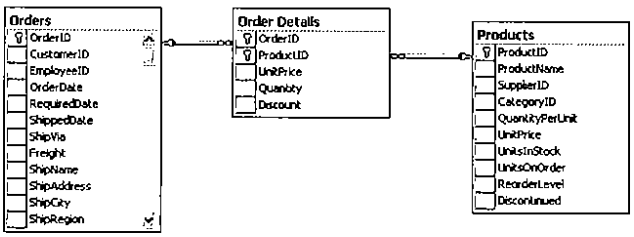
Každý řádek primární tabulky je možné svázat s jedním nebo více řádky sekundární tabulky. U vztahu 1:N je významný směr. Na obrázku je příklad vztahu mezi zákazníky a objednávkami. Jeden zákazník může mít více objednávek.



Obr. 1.4 Vztah jedna k mnoha

Vztah mnoho k mnoha (N:M, many-to-many)

U tohoto vztahu více řádků primární tabulky může být svázáno s více řádky sekundární tabulky. Protože většina databázových systémů nedokáže přímo pracovat se vztahy typu N:M, v praxi se používá dekompozice, což znamená, že tento vztah se implementuje pomocí spojovací tabulky. To znamená, že vztah N:M můžeme rozložit na dva vztahy typu N:1.



Obr. 1.5 Vztah jedna k mnoha

Normalizace databází

U návrhu relačního schématu je důležité dodržovat určitá pravidla, která nazýváme normální formy. Normalizace obvykle vede k odstranění redundancí a značně zefektivňuje

práci s databázovými tabulkami. Pro relační databáze zpravidla platí, že čím jsou tabulky ve vyšších normálních formách, tím lépe by se s nimi z hlediska aplikační logiky mělo pracovat. POZOR, hovoříme o relačních databázích. Databázové tabulky multidimenzionálních databází jsou zpravidla nenormalizované, hlavně tabulky dimenzí.

První normální forma (1NF)

Tabulka splňuje podmínku první normální formy tehdy, když všechny atributy (sloupce) jsou atomické, což znamená dále nedělitelné. Jeden sloupec nesmí obsahovat více druhů údajů, nebo řečeno matematickou terminologií – musí obsahovat skalární hodnotu. Hodnotou sloupce také nesmí být relace. Když databázová tabulka tyto podmínky nesplňuje, je v takzvané nulté normální formě NFNF (non-first normal form) a je potřebné ji rozložit. Například tuto tabulku v nulté normální formě,

Jméno	Telefony
Jan Novák	0112345678, 0602161234, 0905317212
Linda Dvořáková	0298765432, 0602333444
Libor Kudláček	0602111222

kde sloupec TELEFONY obsahuje více druhů údajů, tedy číslo na pevné linky a mobilní telefony, bychom mohli upravit tak, aby splňovala podmínky pro 1NF například:

Příjmení	Jméno	Telefon 1	Telefon 2	Telefon 3
Novák	Jan	0112345678	0602161234	0905317212
Dvořáková	Linda	0298765432	0602333444	
Kudláček	Libor	0602111222		

Druhá normální forma (2NF)

Tabulka splňuje podmínku pro zařazení do druhé normální formy tehdy, když splňuje podmínku 1NF a každý atribut kromě primárního klíče musí být úplně závislý na celém primárním klíči. Druhá normální forma se proto týká jen tabulek, které mají více primárních klíčů. Když má tabulka jen jeden primární klíč, tak podmínka pro 2NF je splněna automaticky.

Zbozi	Dodavatel	Email_dodavatele	Cena
cement	Cementárny Podolí	cementarny@podoli.cz	270
cement	Stavebniny Teplice	stavebniny@teplice.cz	295
písek	Pískárny u Vody	piskarny@voda.cz	85
písek	Stavebniny Teplice	stavebniny@teplice.cz	78

Tabulka je navržena tak, že její primární klíč je tvořen sloupci ZBOZI a DODAVATEL. Tabulka by byla v 2NF, pokud by neobsahovala sloupec `mail_dodavatele`. Sloupec **cena** je z hlediska zařazení tabulky do 2NF v pořádku, protože cena závisí na kombinaci zboží – dodavatel. Naproti tomu sloupec `mail_dodavatele` závisí jen na části primárního klíče (DODAVATEL), ale vůbec nezávisí na jeho druhé části (ZBOZI).

Úprava tabulky, aby splňovala podmínky 1NF, je poměrně jednoduchá. Stačí rozdělit původní tabulku na více tabulek:

Zboží	Kod_dodavatele	Cena
cement	004	270
cement	007	295
písek	002	85
písek	007	78

Kod_dodavatele	Dodavatel	Email
004	Cementárny Podolí	cementarny@podoli.cz
007	Stavebniny Teplice	stavebniny@teplice.cz
002	Pískárny u Vody	piskarny@voda.cz

Třetí normální forma (3NF)

Tabulka je v třetí normální formě tehdy, když je v druhé normální formě a zároveň neexistují závislosti nekličových sloupců tabulky. Pro příklad se můžeme vrátit k tabulce, kterou jsme uvedli jako příklad správně navržené tabulky pro 1NF.

Prijmeni	Rodne_cislo	PSC	Mesto
Novák	121265/1234	056 12	Praha
Dvořáková	065371/4444	073 11	Brno
Kudláček	010154/3333	022 14	Plzeň

Vidíme, že je tu závislost mezi neklíčovými sloupci PSC a Mesto. Sloupec PSC bychom nemuseli zadávat, dal by se vyhledat pro dané město z tabulek poštovních směrovacích čísel. Otázka je, zda to bude, nebo nebude výhodné. Kdybychom chtěli vždy dodržet pravidla pro třetí normální formu, tak by to mohlo vést ke zbytečně složitým a nepřehledným tabulkám a v některých případech dokonce ke ztrátě výkonu. Když má firma 20 zaměstnanců, jejichž adresy eviduje v databázi, je zbytečné mít v databázi tabulku několika tisíců směrovacích čísel měst a obcí.

Takto organizované databáze umožňují:

- velký počet paralelních přístupů,
- rychlé a operativní změny údajů v tabulkách,
- využití indexů, transakcí,
- optimalizovat strukturu údajů s využitím normalizovaných tabulek.

Výhody a nevýhody relačních databází

Relační databáze mají svoje nesporné výhody, mezi které patří možnost využití potenciálu jednak odborníků ve firmách, kteří relační databázový model mnoho roků rutinně používají, jednak potenciál softwaru a vývojových nástrojů pro vývoj a ladění aplikací a pro generování výstupních reportů. Princip relačních databází je využíván v transakčních databázích i datových skladech. Mezi hlavní omezení relačních databází patří

- absence komplexních analytických nástrojů,
- potenciální omezení objemu údajů, ke kterým je možné v rozumném čase přistoupit.

Oblast použití transakčních databází je skoro neomezená. Primárním cílem při jejich návrhu je umožnit klientům databázového serveru vykonávání velkého množství transakcí online, například bankovních, obchodních a podobně. Manipulace s údaji vyplývající z obchodní logiky, například převod peněžních prostředků z jednoho bankovního účtu na druhý, který se skládá z postupnosti určitých kroků, je vykonávána jako transakce, která převede databázi z jednoho konzistentního stavu do druhého. Zjednodušeně řečeno, buď všechny operace v transakci proběhnou úspěšně, nebo neproběhnou vůbec, tedy databáze i v případě neúspěšného vykonání některé operace bude uvedena do konzistentního stavu, v jakém byla před transakcí.

Nevhodnost transakčních databází pro analýzy OLAP

Ke zdroji údajů v transakčních databázích ve stejném čase přistupuje velké množství uživatelů, kteří údaje z databáze čtou, jiní do ní zapisují, případně někteří vykonávají i jednodušší analýzy. Ano, i nad databázemi OLTP je možné vybudovat analytické aplikace OLAP, ale jak se lidově říká: „Není to pravé ořechové." Nejen z teorie, ale i z praktických zkušeností totiž vyplývá, že údaje v transakčních databázích by měly být uloženy v normalizovaných tabulkách, které by měly vyhovovat podmínkám druhé nebo třetí normální formy. To znamená mnoho atomických, relačně svázaných tabulek. Analýza velkého množství takto uložených údajů by byla proto velmi neefektivní a značně pomalá. Také návrh analytické aplikace, hlavně tabulek dimenzí nad databází OLTP, není právě jednoduchá záležitost.

Multidimenzionální databáze

Východiskem pro překonání dvou hlavních omezení relačních databází je zavedení organizace údajů do multidimenzionálních struktur. Takto vytvořené databáze slouží jako podklad pro získání sumarizovaných a agregovaných údajů, tedy vlastně informací. Jak uvidíme později, do multidimenzionálních databází ukládáme upravené a „vyčištěné" údaje. Na rozdíl od relačních databází používáme převážně nenormalizované tabulky, které můžeme rozdělit na dva druhy: na tabulky faktů a tabulky dimenzí.

I multidimenzionální databáze mají svoje výhody a nevýhody. K hlavním výhodám patří:

- rychlý komplexní přístup k velkému objemu údajů,
- přístup k multidimenzionálním a relačním datovým strukturám,
- možnost komplexních analýz,
- silné schopnosti pro modelování a prognózy.

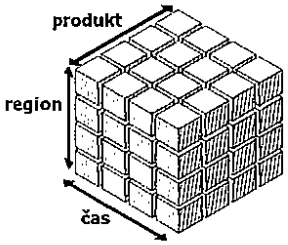
Nevýhodou jsou například vyšší nároky na kapacitu úložiště, problémy při změně dimenzí bez přizpůsobení časové dimenze a podobně. Analytické databáze označujeme i pojmem OLAP (Online Analytical Processing).

OLAP (Online Analytical Processing) – tato zkratka zahrnuje struktury údajů a analytické služby, které slouží pro analýzu velkého množství údajů.

Multidimenzionální databázový model

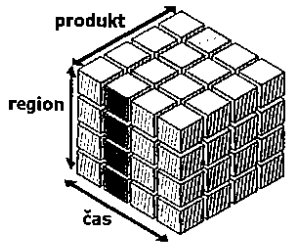
Převážná většina údajů je organizovaná v relační databázi v dvojrozměrných relačních tabulkách. Každý řádek takovéto tabulky se vztahuje k nějakému předmětu, události nebo k jejich části. Výsledkem agregace a analýzy údajů bývá obvykle multidimenzionální datová struktura – **krychle**. Zjednodušeně by se dalo říci, že krychle je v multidimenzionálním datovém modelu jakýmsi ekvivalentem tabulky v relační databázi. Typické využití systémů OLAP je pro analýzu velkého množství údajů. Výsledkem analýzy jsou souhrny a reporty, které slouží manažerům jako podklady pro jejich rozhodnutí, ať už v oblasti řízení firmy, řízení ekonomických a technologických procesů a podobně. Pro výpočet krychlí OLAP je potřebné vykonat velké množství výpočtů a agregací, a to v reálném čase. Každá krychle OLAP má několik dimenzí. Na rozdíl od geometrické krychle může mít multidimenzionální databázový model i více dimenzí než tři. MS SQL Server 2000 umožňuje použití až 64 dimenzí. Příkladem typického třídimenzionálního modelu může být krychle s dimenzemi:

- čas,
- region,
- produkt.

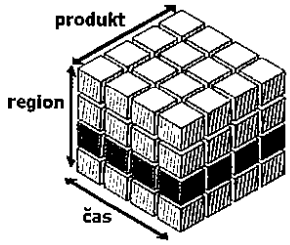


Obr. 1.6 Princip multidimenzionální krychle

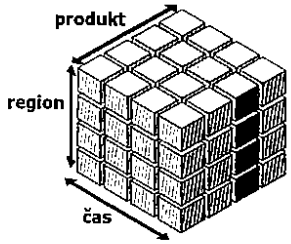
Údaje se nacházejí v průnicích jednotlivých dimenzí. Můžeme analyzovat údaje jen za určité časové období, například abychom vyhodnotili výsledky reklamní kampaně nebo sledovanost webové stránky za určité období a podobně. Jiným příkladem mohou být údaje z určitého regionu, ke kterým má přístup regionální ředitel pro potřeby svého rozhodování. Případně marketingové údaje pro jednotlivé produkty nebo skupiny produktů mohou být k dispozici pro produktové manažery.



Obr. 1.7 Analýza údajů za určité časové období



Obr. 1.8 Analýza údajů podle regionálních kritérií



Obr. 1.9 Analýza údajů pro určitý produkt nebo skupinu produktů

Porovnání relačního a multidimenzionálního modelu

Kdybychom měli heslovitě porovnat výhody a nevýhody relačních a multidimenzionálních databází, došli bychom k následujícím závěrům:

Relační model

výhody

- potenciál odborníků ve firmách, kteří tento model mnoho roků rutinně používají
- potenciál softwaru a vývojových nástrojů pro vývoj a ladění aplikací a pro generování reportů
- použitelnost v transakčních databázích i datových skladech

nevýhody

- absence komplexních analytických nástrojů
- potenciální omezení objemu údajů, ke kterým je možné v rozumném čase přistoupit

Multidimenzionální model

výhody

- rychlý komplexní přístup k velkému objemu údajů
- přístup k multidimenzionálním a relačním datovým strukturám
- možnost komplexních analýz
- silné schopnosti pro modelování a prognózy

nevýhody

- problémy při změně dimenzí, bez přizpůsobení časové dimenzi
- vyšší nároky na kapacitu úložiště

Nástroje pro administraci a práci s analytickými službami

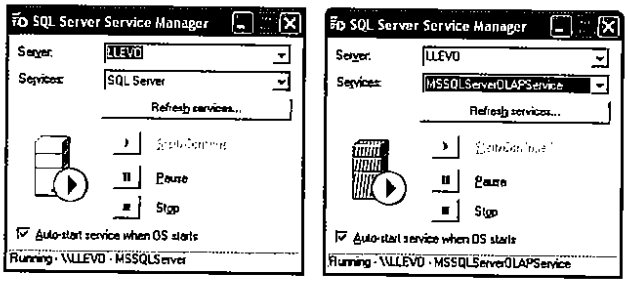
Vzhledem k přiloženému CD předpokládáme, že čtenáři-začátečníci budou dělat svoje první pokusy s analýzami OLAP, dataminingem a přípravou a zavedením údajů do datových skladů s Microsoft SQL Serverem 2000 a jeho integrovanými analytickými službami a datovou pumpou DTS (Data Transformation Services).

MS SQL Server 2000 je výkonný databázový server dostupný jen pro platformu Windows. Je přímo integrovaný s uživatelskými konty a bezpečnostními prvky operačních systémů řady Windows NT/ 2000/XP2003 Server. Tento databázový server pokrývá celou škálu hardwarových platforem od clusterových serverů po přenosné a kapesní počítače s operačním systémem Pocket PC. Tam zatím analytické služby implementovány nejsou, ale kapesní počítače této třídy můžeme využít jako klienty pro přístup k analytickým údajům. Po nainstalování analytických služeb MS SQL Serveru 2000 máme na serveru, případně lokálním počítači sloužícím pro vývoj, případně pro výuku ihned k dispozici analytický server,

tedy službu na pozadí operačního systému. Navenek se nic neděje, ale při pozornějším zkoumání zjistíme, že v menu MS SQL Serveru přibyla nová složka. V podobné situaci ale může být i člověk, který najde instalační CD s trial verzí MS SQL Serveru v odborném časopise, a nevěděl, o co se jedná, analytické služby si nainstaluje. Přijde tak o necelých 100 MB diskového prostoru a nějaké megabajty operační paměti, ale užitek z toho nebude mít žádný. Proto je potřebné představit nástroje pro administraci databázového a analytického serveru a také nástroje pro vytváření analytických databází a práci s nimi

SQL Server Service Manager

Aplikace SQL Server Service Manager slouží pro start, pozastavení, případně zastavení databázového serveru (Služba *SQL Server*) a analytického serveru (Služba *MSSQLServerOLAPService*).



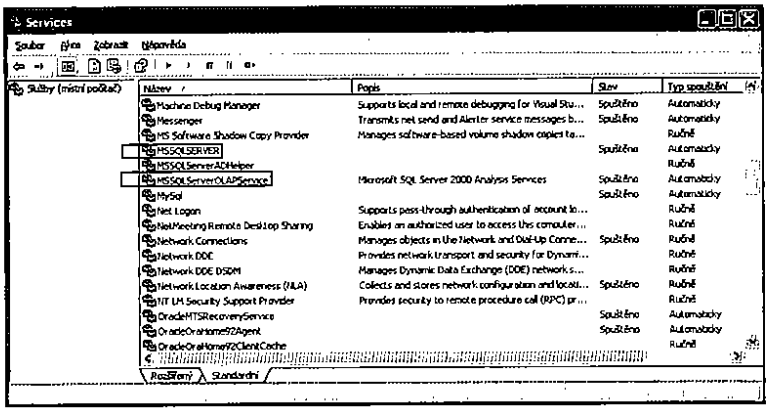
Obr. 1.10 SQL Server Service Manager

Příslušná služba se přepíná pomocí ovládacího prvku Services. Zaškrtnutím volby *Autostart service when OS Starts* nastavíme automatické spouštění příslušných služeb při startu operačního systému. Stejné úkony, tedy start, pozastavení, případně zastavení databázového serveru, je možné vykonávat i pomocí správce služeb operačního systému Windows.

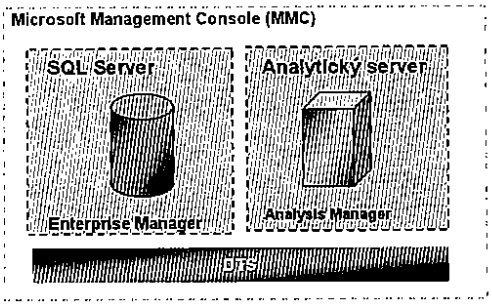
Když si pozorněji prohlédneme fragment seznamu služeb na obrázku, tak vidíme, že jde o typický příklad počítače (konkrétně notebooku) a vývojáře databázových aplikací a datových skladů v nehomogenním prostředí, protože kromě MS SQL Serveru tam v dokonalé souhře běží i jiné databázové servery.

Microsoft Management Console (MMC)

Všechny tři základní funkční bloky MS SQL Serveru a Analytických služeb, tedy administrace databázového serveru, analytické služby a datová pumpa DTS se ovládají jednotně pomocí grafického uživatelského rozhraní (GUI) s názvem Microsoft Management Console (MMC). Toto uživatelské prostředí slouží nejen pro MS SQL Server, ale i pro přístup a správu mnoha služeb v operačním systému Windows typové řady NT/2000/XP/Windows Server 2003. Blokové schéma zapouzdření MS SQL Serveru, analytického serveru a DTS je na obrázku.



Obr. 1.11 Správce služeb operačního systému Windows



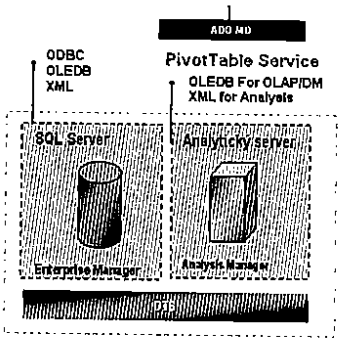
Obr. 1.12 Blokové schéma Microsoft Management Console

Když toto schéma mírně upravíme, služby zůstanou k dispozici pro aplikace, ale namísto klientského rozhraní je na dalším obrázku rozhraní technologické. Ne každému vyhovují aplikace dodávané spolu s databázovým a analytickým serverem a často je výhodné vytvářet vlastní aplikace tak, aby „zapadly" do existující podnikové informační infrastruktury.

Technologické rozhraní se skládá z:

- OLE DB pro OLAP
- OLE DB pro Data Mining
- XML pro analýzy

Vývoji klientských aplikací se podrobně věnuje kapitola 7.

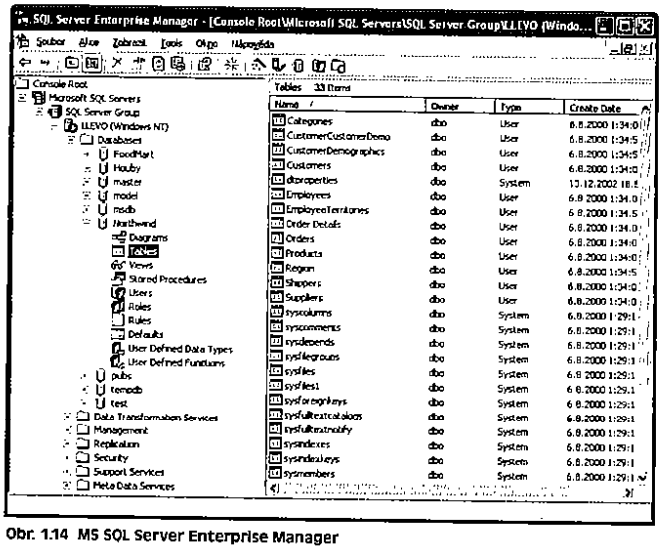


Obr. 1.13 Blokové schéma technologického rozhraní

SQL Server Enterprise Manager

Z hlediska uživatelského rozhraní, můžeme prostřednictvím MMC vykonávat administraci databázového serveru Microsoft SQL Server 2000, když spustíme položku menu **Enterprise Manager**.

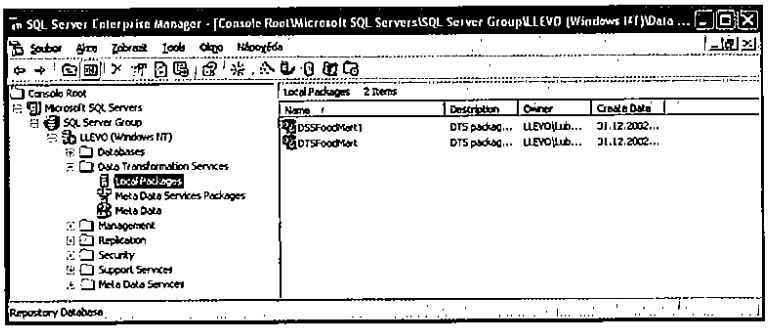
Pomocí této administrátorské aplikace můžeme například vytvářet nové databáze, vytvářet a spravovat účty jednotlivých uživatelů, přidělovat a rušit jejich oprávnění pro práci



Obr. 1.14 MS SQL Server Enterprise Manager

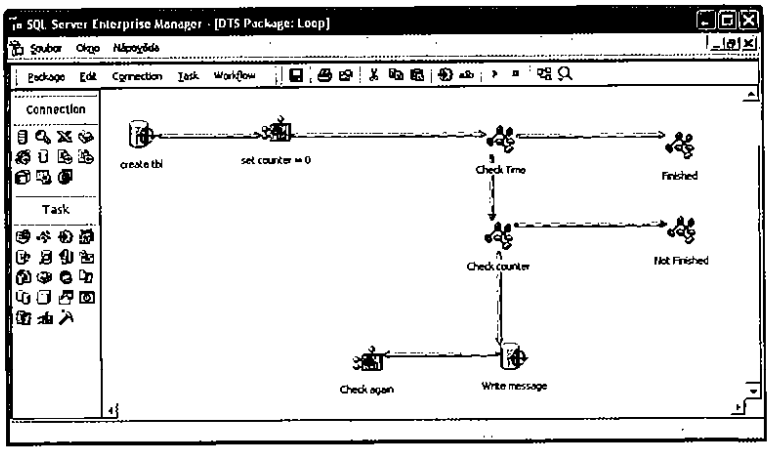
s jednotlivými objekty databáze. Také je možné nastavit strategii údržby a zálohování údajů v databázi a podobně.

Pomocí aplikace SQL Server Enterprise manager můžeme vytvářet spravovat a editovat i balíčky datové pumpy DTS. Pro tento účel slouží složka **Data Transformation Services**.



Obr. 1.15 MS SQL Server Enterprise Manager

V pravém okně aplikace jsou zobrazeny jednotlivé balíčky DTS. Po výběru některého z nich se MMC přepne na zobrazení nástroje DTS Designer, kde můžeme pomocí grafického uživatelského rozhraní navrhovat, případně editovat postupnost kroků pro transformaci dat.



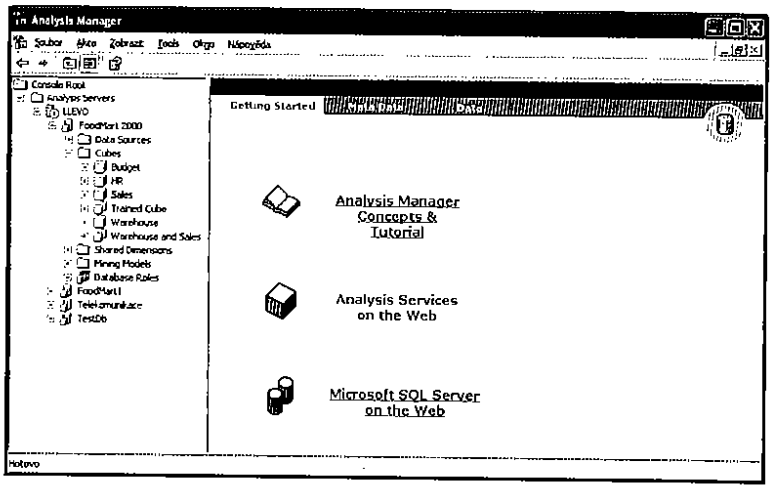
Obr. 1.16 SQL Server Enterprise Manager

Analysis Manager

Klíčovou aplikací pro práci s analytickými službami MS SQL Serveru a multidimenzionálními databázemi je Analysis Manager. I tato aplikace je zapouzdřena do Microsoft Management Console. Jako skoro každá aplikace MMC i Analysis manager se skládá z hierarchické stromové struktury složek a objektů v levé části pracovního okna obrazovky. Pravá část pracovní obrazovky je rozdělena na tři záložky:

- Getting Started
- Meta Data
- Data

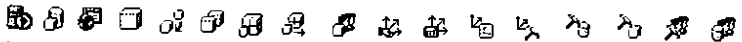
V záložce **Getting Started** je jednoduchý tutoriál pro vytvoření krychle OLAP a odkazy na materiály ohledně analytických služeb MS SQL Serveru, které jsou umístněny na webu.



Obr. 1.17 Analysis Manager

Levé okno prohlížeče objektů

Toto okno může obsahovat hierarchickou strukturu těchto objektů:



Obr. 1.18 Ikony v okně Object Browser

1. *Analysis Servers* – Složka obsahuje ikony pro každý analytický server, který je registrovaný v aplikaci Analysis Manager. Název serveru je vypsán vedle ikony.

2. *Databases* – Každý analytický server spravuje jednu nebo více analytických databází. Ikony příslušných databází jsou seřazeny pod ikonami analytických serverů.

   Pod ikonami jednotlivých analytických databází jsou ikony složek:

   - Data Sources
   - Cubes
   - Shared Dimensions
   - Mining Models

3. *Data Sources* – Složka obsahuje ikony datových zdrojů, tedy relačních databází, na základě kterých byla analytická databáze vytvořena.

Krychle

Složka Cubes může obsahovat tři typy krychlí OLAP:

1. *Regulární krychle*
2. *Přilinkovaná krychle*
3. *Virtuální krychle*

Partitions (rozdělení)

Každá krychle může obsahovat *Partitions*. V hierarchické struktuře prohlížeče objektů můžeme zobrazit dva typy *Partitions:*

1. *Local Partitions*
2. *Remote Partitions*

Cube Roles

Ikona Cube Roles slouží pro administraci uživatelského přístupu ke krychlím.

Sdílené dimenze

Jednotlivé dimenze můžeme používat ve více krychlích. Složka Shared Dimension může obsahovat ikony těchto typů dimenzí:

1. *Regular Dimensions*
2. *Virtual Dimensions*
3. *Parent-Child Dimensions*
4. *Data Mining Dimensions*

Modely pro datamining

Složka Mining Models obsahuje ikony data miningových modelů:

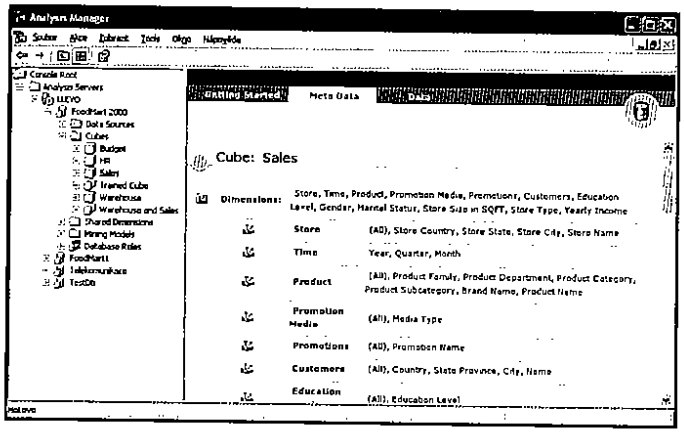
1. *Relational Mining Models*
2. *OLAP Mining Models*
3. *Mining Model Roles*

Database Roles

Ikona Database Roles slouží k administraci uživatelského přístupu k analytickým databázím.

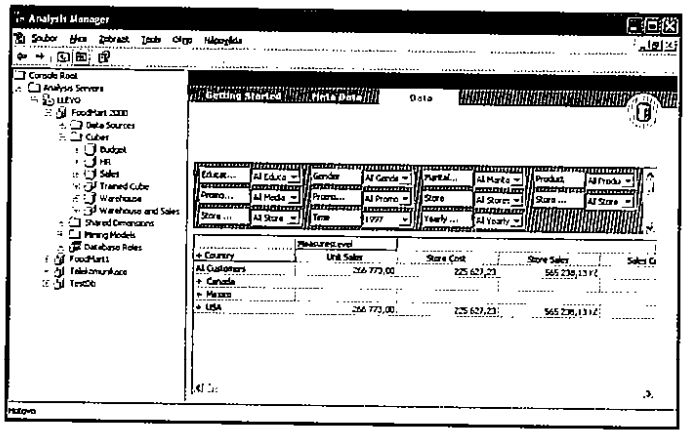
Pravé pracovní okno

Na záložce **Meta Data** máme v přehledné formě vypsány údaje o objektu, který je označen v levém okně. Může to být krychle OLAP, dimenze, dataminingový model a podobně.



Obr. 1.19 Analysis Manager

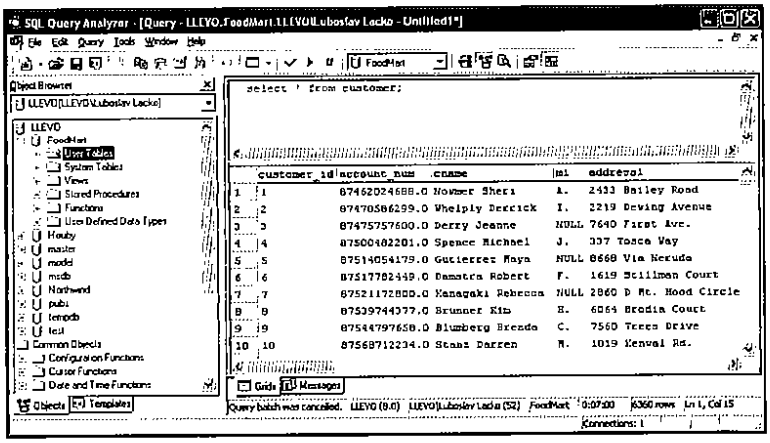
V záložce **Data** si můžeme vytvářet návrhové zobrazení kontingenčních tabulek pro prohlížení údajů.



Obr. 1.20 Analysis Manager

SQL Server Query Analyzer

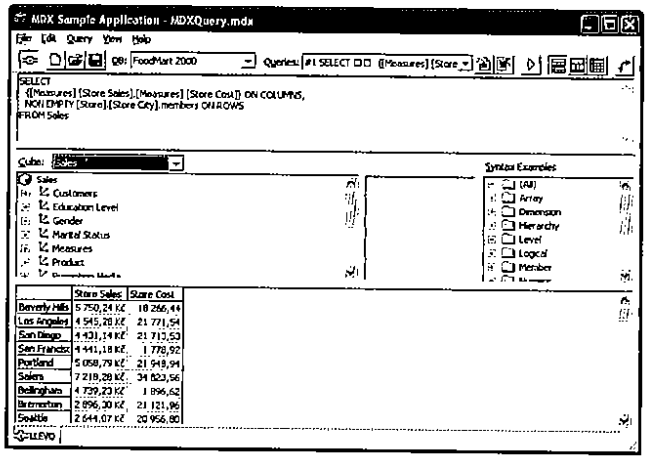
Databázovým specialistům to možná přijde směšné, ale častým problémem začátečníků je triviální věc. Vytvoří svůj první dotaz SQL pravděpodobně chybný, jsou začátečníci, ale co s ním, kam ho zadat? Když se pokročilejším čtenářům bude zdát tento úvod směšný, pokuste se někam nějak zadat příkaz typu CREATE CUBE a uvidíte, že s tím budou docela problémy. Ale vraťme se k jazyku SQL. Pro práci s údaji prostřednictvím jazyka SQL, například pro ladění příkazů v etapě vývoje používáme klientskou konzolovou aplikaci, prostřednictvím které můžeme jednak zadávat příkazy, jednak zobrazovat odezvy databázového serveru. Spolu s MS SQL Serverem je dodávaná konzolová aplikace SQL Query Analyser.



Obr. 1.21 MS SQL Query Analyser

MDX Sample Application

Konzolovou aplikaci MDX Sample Application, která se nainstaluje spolu s analytickými službami MS SQL Serveru 2000, můžeme používat pro ladění a testování příkazů pro přístup k údajům v multidimenzionálních databázích v jazyce MDX. Tímto tématem se podrobně zabývá kapitola 6. Tato konzolová aplikace nám podstatně zjednodušuje zadávání dimenzí do těla příkazů MDX. Stačí ve středním okně rozvinout stromovou strukturu příslušné dimenze a klepnout na konkrétní úroveň nebo na konkrétní prvek. Jeho název se potom přenese do těla příkazu MDX na aktuální pozici kurzoru.



Obr. 1.22 MDX Sample Application

Kapitola 2

Datové sklady

Intenzivní nasazování informačních technologií do prakticky všech odvětví přináší s sebou shromažďování velkého množství různorodých údajů. Shromažďují se údaje z technologických zařízení, firemní administrativy, odbytu a podobně. Množství údajů mají shromážděno i velké obchodní domy a obchodní řetězce například ze skladových karet, elektronických pokladen a podobně.

Tato situace by se dala s trochou nadsázky přirovnat k uživateli Internetu, který ze sítě sítí bezhlavě stahuje obrovské množství údajů, které ho zajímají (a pro jistotu i ty údaje, které ho momentálně nezajímají, ale v budoucnosti by ho zaujímat mohly). Nashromážděné informace skladuje nejdříve na pevném disku svého počítače a potom je vypálí na CD. Tímto postupem se mu podaří za určitou dobu shromáždit vzhledem k jeho poměrům a potřebám obrovské množství údajů, řádově gigabajty až desítky gigabajtů. A do hry vstupují vysokorychlostní sítě a zapisovatelná média DVD...

V předcházejícím odstavci se příliš často objevovalo slovo **údaje**, ale úmyslně se tam ani jednou neobjevilo slovo **informace**. Tyto dva pojmy totiž ani zdaleka nevyjadřují to samé. Výstižnou definici rozdílu mezi údaji a informacemi můžeme najít ve firemní literatuře firmy Oracle [1]. Podle této definice se údaje stávají informacemi, když:

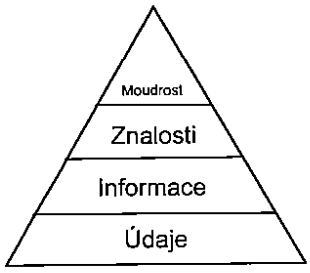
- máme údaje,
- víme, že máme údaje,
- víme, kde tyto údaje máme,
- máme k nim přístup,
- zdroji údajů můžeme důvěřovat.

Když se zamyslíme nad příkladem s fanatikem shromažďujícím údaje z Internetu, tak jsou splněny jen dvě podmínky. Vzpomínaný shromažďovač skutečně má údaje a dokonce i tuší, že je někde má. Ale to je vše. Nevíme přesně, kde tyto údaje má, když nepovažujeme za směrodajný údaj, že je má na některém CD ve skříni. Nemá k těmto údajům ani operativní přístup a o důvěryhodnosti většiny volně dostupných údajů na Internetu také víme svoje. Takže náš fanatik má mnoho údajů, ale nemá prakticky žádné informace. Je to sice příklad trochu přitáhnutý za vlasy, ale podobná situace může nastat ve firmách, obchodních domech a podobně. Může dojít k paradoxu, že sice mají shromážděno velké objemy údajů, ale tyto údaje obsahují jen malé množství užitečných informací a tyto informace nejsou operativně k dispozici. Vysvětlením rozdílu mezi údaji a informacemi se zaobírá i pojem Business Intelligence.

Business Intelligence

Tento pojem můžeme definovat jako proces transformace údajů na informace a převod těchto informací na poznatky prostřednictvím objevování. Účelem Business Intelligence

je tedy konverze velkých objemů údajů na poznatky, které jsou potřebné pro koncové uživatele. Tyto poznatky můžeme potom efektivně využít například v procesu rozhodování. Pod pojmem informace nerozumíme jen konkrétní záznam nebo množinu záznamů. Často potřebujeme sledovat trend nějaké veličiny, například při obchodování s cennými papíry, nebo potřebujeme najít mezi údaji určité závislosti. Proto moderní databázové servery obsahují rozsáhlou podporu pro OLAP (Online Analytical Processing), Data Mining (dolování, odkrývání dat) a Data Warehouse (datové sklady). Proměnu údajů na informace, informací na znalosti a budování „moudrosti" na základě znalostí můžeme zobrazit na hierarchické pyramidě informačních úrovní. Základem všeho jsou údaje. Údaje obsahují jen jednoduchá fakta, přičemž tušíme, že někde ve vnitřní množině údajů jsou ukryty určité informace. Tyto informace však odhalíme až tehdy, když přidáme k datům souvislosti. Když do hry vstoupí kromě informací i tvořivá inteligence, tak získáme znalosti. Když tyto znalosti zevšeobecníme, získáme „moudrost", což znamená schopnost přesného zhodnocení znalostí a jejich následné uplatnění v reálné praxi.

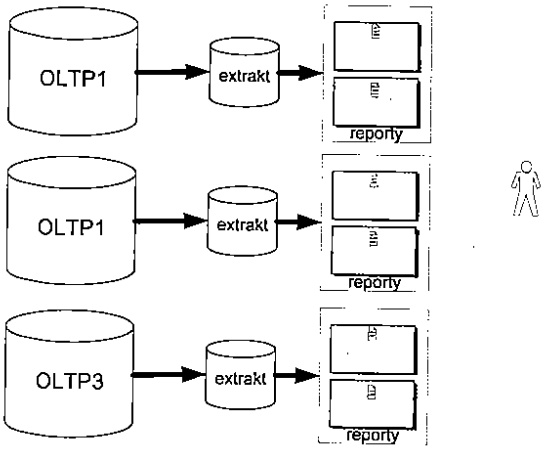


Obr. 2.1 Hierarchie informačních úrovní

Zpracování údajů z operačního prostředí

Údaje je samozřejmě možné „přetavit" na informace a následně analyzovat i v operačním prostředí, kde tyto údaje vznikají. Takovýto postup je však možný jen v málo vytížených transakčních systémech. Jinak dochází k neúměrnému snižování výkonu těchto systémů. Problém snižování výkonu je možné částečně vyřešit výběrem (extrakcí) a přenesením údajů z jednoho prostředí do prostředí jiného, částečně se vyřešil použitím technik *zpracování extraktu*, které vybírají data z jednoho prostředí a přenášejí je do jiného prostředí, kde se na jiném hardwaru zpracují.

Údaje pro extrakci se vybírají podle určitých kritérií a následně se umístí zpravidla do souborů nebo databází v jiném operačním prostředí. Tyto vyextrahované údaje a výsledky analýz získaných nad těmito údaji jsou potom k dispozici analytikům a pracovníkům, kteří řídí a rozhodují. Proces extrakce byl logickým krokem od systémů OLTP k systémům na podporu rozhodování. Údaje se přesouvají z transakčních systémů na klientské systémy určené pro analýzu.



Obr. 2.2 Extrakce údajů

Zdálo by se, že extrakce údajů a zpracování takto získaných extraktů je ideální řešení, ale dochází k mnoha problémům. Jednak může docházet k mnohonásobnému větvení tím způsobem, že se extrahované údaje stanou zdrojem pro další extrakci. Extrakce údajů může úplně zaměstnat kapacitu oddělení IT podniku, což je také nežádoucí. Dochází také k duplicitám, kdy se při extrakci a zpracovávaní extraktů pokaždé přistupuje k těm stejným údajům. Také flexibilita extrakce je velmi omezená. Protože extrakty obsahují jen údaje, a ne metadata, tedy údaje o údajích nebo jiným způsobem řečeno údaje o tom, jakým způsobem byly dané údaje získány, je těžké přizpůsobit extrakci změnám v předmětu a způsobu podnikání.

Po přečtení statí o datových skladech bychom našli i další nevýhody extrakce údajů. Především chybí jednotná časová základna, jednotné algoritmy pro transformaci údajů a výpočet požadovaných hodnot. Přístup k externím údajům pravděpodobně bude nekonzistentní a nebude správně definovaná ani granularita externích údajů. Sestavy vygenerované na základě extrahovaných údajů tak ve většině případů obsahují spíše údaje než informace.

Dalším závažným důvodem nevhodnosti operačních databází pro analýzy je nedostatečná historie údajů způsobená tím, že v transakčních systémech jsou z kapacitních důvodů udržována jen data za krátké časové období, maximálně několik měsíců. V některých systémech se starší údaje archivují, ale tyto archivy mají jen velmi omezený rozsah použití. Pravděpodobně už z názvu této kapitoly tušíte, že řešením naznačených problémů by mohl být datový sklad. K tomuto řešení se dostaneme postupně po etapách, přibližně tak, jak probíhal chronologický vývoj datových skladů.

Skladování údajů

Když se zamyslíme nad tím, co je to datový sklad, odpověď na tuto otázku je jednoduchá. Je to určitým způsobem strukturované úložiště údajů. Kdybychom ale hledali nějaké analogie mezi klasickým skladem a datovým skladem, tak to není tak jednoduché, jak to na první pohled vypadá.

V klasickém skladu skladujeme buď materiály, součástky a polotovary, které vstupují do výrobního procesu, nebo naopak skladujeme výrobky předtím, než se budou expedovat. Nikdo totiž nemá zájem skladovat dlouhou dobu polotovary a už vůbec ne hotové výrobky. Čím rychleji je dokážeme vyexpedovat a prodat, tím lépe pro ekonomiku firmy.

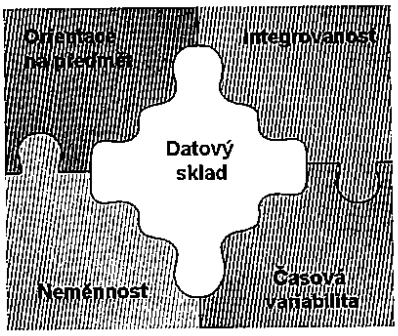
V datovém skladu naproti tomu chceme shromažďovat a uchovávat informační bohatství firmy za co nejdelší období. Spíše než ke klasickým skladům můžeme datové sklady přirovnat k depozitářům muzeí. I v tomto případě se snažíme shromažďovat exponáty, třídit je časově, geograficky, podle druhů a podobně.

Datový sklad

Nejznámější definice datového skladu pochází od Billa Inmona.

Datový sklad je podnikově strukturovaný depozitář subjektově orientovaných, integrovaných, časově proměnlivých, historických dat použitých na získávání informací a podporu rozhodování. V datovém skladu jsou uložena atomická a sumární data.

Údaje se získávají a ukládají do produkčních (operačních) databází, které mohou být v různých odděleních firem nebo dokonce v rozličných geografických lokalitách. Tyto údaje v pravidelných intervalech sesbíráme, předzpracujeme a zavedeme do datového skladu. Datový sklad je v podstatě též databáze, jen je organizovaná podle trochu jiných pravidel, tabulky například nemusí být normalizovány a podobně. Datový sklad je tedy soubor technologií pro efektivní skladování údajů, tak aby tyto údaje po jejich přeměně na informace sloužily podpoře rozhodování.



Obr. 2.3 Grafické vyjádření definice datového skladu

Definice Billa Inmona je velmi stručná a výstižná. Pravděpodobně ale bude potřebné si tuto definici přečíst několikrát a zamyslet se nad jednotlivými pojmy, které definici tvoří.

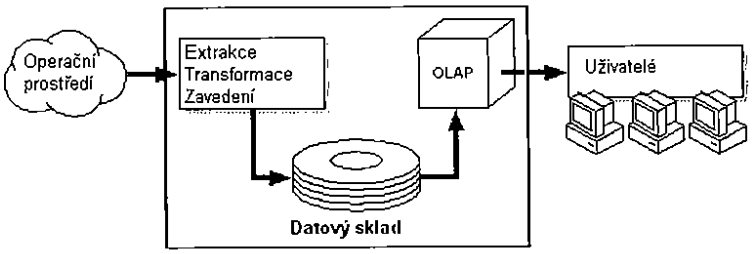
- **subjektová orientace:** Údaje se do datového skladu zapisují spíše podle předmětu zájmu než podle aplikace, ve které byly vytvořeny. Při orientaci na subjekt jsou data v datovém skladu kategorizována podle subjektu, kterým může být např. zákazník, dodavatel, zaměstnanec, výrobek a podobně. Orientace na aplikaci naproti tomu znamená, že údaje jsou v systému uloženy podle jednotlivých aplikací, například údaje aplikace pro odbyt, údaje aplikace pro fakturaci, údaje aplikace pro personalistiku.
- **integrovanost:** Datový sklad musí být jednotný a integrovaný. To znamená, že údaje týkající se konkrétního předmětu se do datového skladu ukládají jen jednou. Proto musíme zavést jednotnou terminologii, jednotné a konzistentní jednotky veličin. Není to jednoduchá úloha, protože údaje přicházejí do datového skladu z nekonzistentního a neintegrovaného operačního prostředí. Proto musí být údaje v etapě přípravy a zavedení upraveny, vyčištěny a sjednoceny. Když údaje nejsou konzistentní a důvěryhodné, datový sklad ztrácí význam.
- **časová variabilita:** Údaje se ukládají do datového skladu jako série snímků, z nichž každý reprezentuje určitý časový úsek. Na rozdíl od operačního prostředí, kde jsou údaje platné v okamžiku přístupu, v datových skladech jsou údaje platné pro určitý časový moment, časový snímek. Zatímco v operačním databázovém prostředí se údaje ukládají za kratší časové období, nejčastěji dní, maximálně měsíců, v datovém skladě jsou údaje za delší časové období, typicky několik roků. Klíčové atributy v datovém skladě obsahují čas, který v operačních databázích nemusí být uváděn. Jakmile je v datovém skladu zaznamenán konkrétní snímek dat z operativní databáze, nemohou být už tyto údaje v datovém skladě modifikovány.
- **neměnnost:** V operačních transakčních databázích jsou údaje do databáze jednak vkládány, jednak modifikovány, a i mazány. Údaje v datovém skladě se obvykle nemění ani neodstraňují, jen se v pravidelných intervalech přidávají nové údaje. Proto je manipulace s údaji mnohem jednodušší v datových skladech. V zásadě můžeme připustit jen dva typy operací. Zavedení údajů do datového skladu a přístup k těmto údajům. Žádné změny údajů nejsou přípustné. Z toho vyplývá, že většina metod pro optimalizaci a normalizaci údajů a transakční přístup k údajům je v datovém skladě nepotřebná.

Údaje se získávají a ukládají do produkčních (operačních) databází, které mohou být v různých odděleních firem, nebo dokonce v rozličných geografických lokalitách. Tyto údaje v pravidelných intervalech sebereme, předzpracujeme a zavedeme do datového skladu. Datový sklad je v podstatě též databáze, jen je organizovaná podle vzpomínaných pravidel. Rozdíly mezi produkční databází a datovým skladem přehledně zobrazuje následující tabulka.

Vlastnost	Operační databáze	Datový sklad
Čas odezvy	Zlomky sekund až sekundy	Sekundy až hodiny
Operace	DML (data manipulation language)	Primární jen na čtení
Původ dat	30-60 dní	Série snímků za časový úsek
Organizace dat	Podle aplikace	Podle předmětu, času…
Velikost	Malá až velká	Velká až velmi velká
Zdroje dat	Operační, interní	Operační, interní, externí
Činnosti	Procesy	Analýza

V datovém skladu můžeme vykonávat různé analýzy pro potřeby rozhodování manažerů, obchodníků. Nástroje pro budování a provoz datových skladů však představují poměrně velkou počáteční investici za hardware a software, takže datové sklady využívají většinou banky, pojišťovny, mobilní a telekomunikační operátoři, velké obchodní řetězce a podobně. K údajům v datovém skladě a výsledkům analýz nad těmito údaji mohou mít prostřednictvím webu přístup manažeři firmy po celém světě a obchodní partneři na základě přidělených přístupových práv. Informace jsou totiž zbožím.

Tím, že údaje z datového skladu můžeme poskytovat i svým obchodním partnerům, se nám může postupně část vynaložených nákladů na datový sklad vrátit. Například dodavatelé velkého obchodního domu určitě ocení informace o tom, jaké zboží jde nejvíce na odbyt, případně o jaké zboží bude pravděpodobně zájem v nejbližších týdnech. Mýtus o tom, že datové sklady jako oblast informatiky je záležitost složitá a náročná na čas i finance, by se zdánlivě mohl prolomit jednak po přečtení a pochopení definice a jednak při pohledu na principiální schéma datového skladu. Problém je formulovatelný do jednoho krátkého odstavce. Stačí získat údaje z operačního prostředí a po úpravě je zavést do datového skladu, nad těmito údaji vykonat analýzy a jejich výsledky zpřístupnit uživatelům, tedy manažerům a analytikům.



Obr. 2.4 Datový sklad

A ještě dodáme, že datový sklad v malém je možné simulovat i na trochu lépe vybaveném desktopovém počítači. Co je potom na datových skladech tak náročné a složité?

Návrh a koncepce

Asi největší procento z hodnoty datového skladu se skrývá v jeho návrhu a koncepci. Know-how a to nejen z oblasti datových skladů, ale hlavně z oblasti podnikání naší firmy, je zkrátka to nejcennější, co ve firmě máme, když samozřejmě nepočítáme firemní značku zavedené firmy. V tomto případě si kvalitu na rozdíl od hardwaru a softwaru nemůžeme koupit, kvalitu musíme vytvořit, případně ji pro nás vytvoří dobře placený vývojový tým.

Hardware

Co se týká nákladů, ani hardware se nedá zahanbit, ale z hlediska filosofie datového skladu se jedná jen o technické prostředky, které jsou nahraditelné. Pozor, technické prostředky, ne údaje, které jsou v nich uloženy. A tak musíme uvážit, kam se v přímé úměře výkonu a spolehlivosti na jedné straně a ceny na druhé straně chceme pohybovat. Samozřejmě pro rychlý přístup k obrovskému množství údajů potřebujeme výkonné servery nebo serverové farmy.

Software

Nástroje pro vytváření datových skladů a analýzy údajů jsou také velmi drahou záležitostí. Stále více se prosazuje trend integrace analytických služeb přímo do instalací databázových serverů, přičemž v některých případech se za analytické služby platí licenční poplatky, v některých případech je to zahrnuto v ceně databázového serveru.

Datové trhy

Datové trhy (anglicky datamart) jsou určité přesně specifikované podmnožiny datového skladu, které jsou určeny pro menší organizační složky firmy. Datový sklad je totiž z hlediska investic i objemu prací velmi náročný projekt. Proto se v některých případech přistupuje k budování datového skladu po částech, což znamená, že pro některé důležité organizační složky se vytvořily jakési podmnožiny datového skladu – datové trhy. Kromě ekonomického efektu má tento postup i psychologický efekt, protože fungující podmnožina datového skladu prohlubuje důvěru v úspěšnost a potřebnost datového skladu jako celku.

Datové trhy mohou vzniknout i opačným postupem, to jest nejdříve se vytvoří centrální integrovaný datový sklad a z něho se potom vytvoří několik datových trhů. Toto řešení je flexibilnější a klade menší nároky na provoz a údržbu. Datové trhy tedy mohou existovat jako subsystémy datových skladů nebo i samostatně jako jednoduchý datový sklad.

Metody budování datového skladu

Pravděpodobně nejdůležitějším krokem při budování datového skladu je výběr nejvhodnější metody. Musíme brát do úvahy nejen organizační strukturu a informační „kulturu" firmy, ale předvídat i možné potíže, které se během budování datového skladu nevyhnutelně objeví. Nejznámější a nejčastěji používané metody jsou:

- **Metoda „velkého třesku"**
- **Přírůstková metoda**

Metoda „velkého třesku"

Mnohé firmy a vývojáři a možná i někteří konzultanti se domnívají, že je možné realizovat implementaci datového skladu pomocí jediného projektu. Ale vývoj datového skladu je náročná záležitost a pravděpodobně se ji nepodaří vyřešit najednou a už vůbec ne v rozumném čase. To je ale největší slabina, protože kdybychom i nakonec projekt datového skladu metodou velkého třesku zrealizovali, mezitím se mohou změnit nejen technologie, ale i požadavky uživatelů. Metoda „velkého třesku" se skládá z tří etap:

- Analýza požadavků podniku
- Vytvoření podnikového datového skladu
- Vytvoření přístupu buď přímo, nebo přes datové trhy

Jedinou výhodou metody „velkého třesku" je skutečnost, že můžeme celý projekt kompletně vypracovat ještě před začátkem jeho realizace. Budování datového skladu je dynamický proces, při kterém je skoro jisté, že se změní nejen technologie, ale i požadavky uživatelů, není možné tento fakt považovat za skutečnou výhodu. Takže převažují spíše nevýhody, které se dají vyjmenovat velmi stručně a jsou velmi závažné. Je tu jednak velké riziko změny požadavků a hlavně trvá velmi dlouhý čas, než se projeví první výsledky obrovských investic do datového skladu, jinak řečeno, než se dostaví „hmatatelný" obchodní zisk.

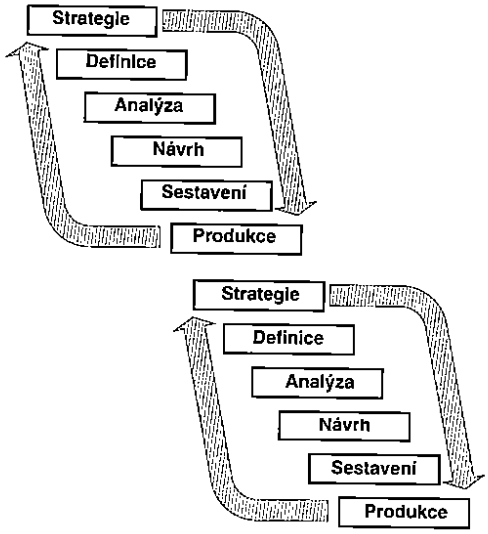
Přírůstková metoda

Přírůstková metoda, jinak nazvaná i evoluční předpokládá budování datového skladu po jednotlivých etapách, tedy místo vybudování celého datového skladu postupně přibývají přírůstková řešení, která samozřejmě zapadají do celkové architektury datového skladu. Začneme tedy budováním několika málo předmětných oblastí, typicky jednou nebo dvěma. Toto částečné řešení implementujeme například jako škálovatelný datový trh a poskytneme ho koncovým uživatelům. Tím se částečně uspokojí „hlad" managementu po návratnosti investicí, když první subsystémy začnou fungovat a přinášet výhody v krátkém čase po zahájení projektu. Když se částečné řešení, navíc důsledně otestované skutečnými uživateli osvědčí, můžeme do systému přidat další předmětnou oblast, čímž získáme novou funkcionalitu. A takto bychom mohli pokračovat až do úplného vytvoření datového skladu.

Budování datového skladu přírůstkovou metodou je tedy iterativní proces, který udržuje neustálou spojitost mezi datovým skladem a potřebami uživatelů. U přírůstkové metody na rozdíl od metody „velkého třesku" převažují výhody nad nevýhodami. K hlavním výhodám přírůstkové metody patří:

- Přírůstkové budování datového skladu zachovává kontinuitu budovaného projektu s požadavky a potřebami uživatelů.
- Umožňuje implementovat škálovatelnou, tedy rozšiřitelnou architekturu.
- Zabezpečí rychlejší zisk a tedy i rychlejší návratnost investic.

Ideu budování přírůstkové metody bychom mohli znázornit pomocí následujícího schématu.



Obr. 2.5 Schéma přírůstkové metody (zahrnuje jen dva cykly iterace)

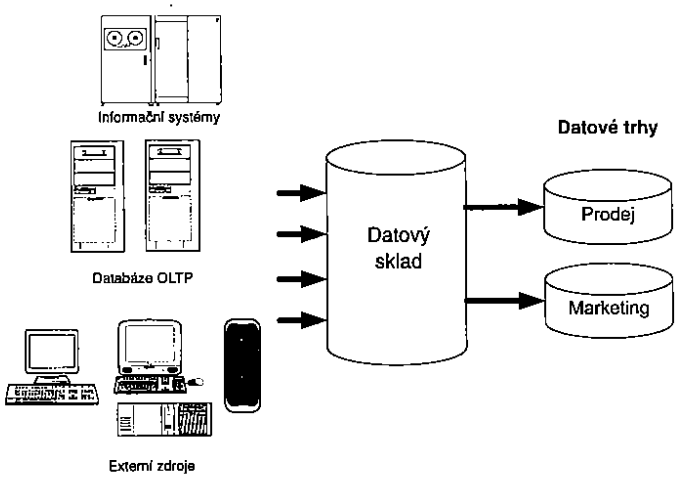
Když se rozhodneme pro budování datových skladů přírůstkovou metodou, můžeme si vybrat jednu z dvou variant

- Přírůstková metoda směrem „shora dolů".
- Přírůstková metoda směrem „zdola nahoru".

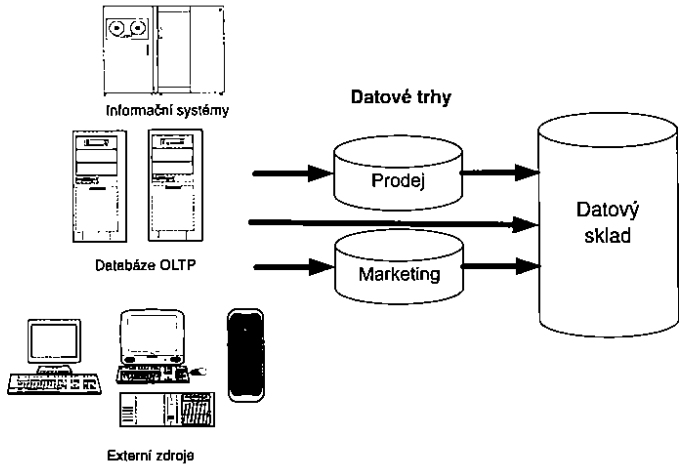
Přírůstková metoda směrem „shora dolů"

U této metody nejdříve na základě požadavků uživatelů vytvoříme konceptuální model datového skladu, přičemž stanovíme hierarchii předmětných oblastí. Následně sestavíme konceptuální modely jednotlivých předmětných oblastí. Jinými slovy řečeno, postupně vytváříme datové trhy jednotlivých předmětných oblastí v rámci struktury datového skladu.

Tato metoda poskytuje poměrně rychlou implementaci jednotlivých datových trhů a tím i návratnost investic. V porovnání s metodou „velkého třesku" přírůstková metoda „shora dolů" je zatížena podstatně menším rizikem, neboť není tak náročná na analýzu. Mezi hlavní nevýhody přírůstkové metody směrem „shora dolů" patří zvýšené vstupní náklady dříve, než je možné předvídat návratnost investic.



Obr. 2.6 Schéma přírůstkové metody „shora dolů"



Obr. 2.7 Schéma přírůstkové metody „zdola nahoru"

Přírůstková metoda směrem „zdola nahoru"

Tato přírůstková metoda je velmi podobná metodě „shora dolů", ale prioritu mají údaje před obchodním ziskem. Jak je zřejmé ze schématu, budujeme nejdříve datové trhy předmětných oblastí v rámci struktury datového skladu.

U této metody vystupuje do popředí oddělení IT podniku. Co se týká výhod a nevýhod, převažují nevýhody. Nakolik se konceptuální model odvíjí od zdrojových systémů, je celková rozšiřitelnost v některých případech značně problematická. Oddělení IT se v mnohých podnicích nepovažuje zrovna za „lídra" v oblasti strategie a marketingu, proto se o mnohých připravovaných změnách a strategických záměrech dovídá oddělení IT zpravidla jako poslední. Proto mohou navrhnout a v některých případech i zrealizovat něco, co je vzhledem ke strategickým záměrům podniku už neaktuální. A navíc oddělení IT je zvyklé pracovat spíše s údaji než s informacemi, proto úloha lídra pro oddělení IT není vždy šťastné řešení.

Fáze přírůstkové metody

Když si prohlédneme schéma přírůstkové metody, tak zjistíme, že se skládá z následujících kroků:

- Strategie
- Definice
- Analýza
- Návrh
- Sestavení
- Produkce

Strategie

Úloha fáze strategie je v podstatě jen jediná, o to je ale důležitější, ba dalo by se říci, že je v rámci budování datového skladu klíčová. Je potřebné definovat cíle. Jednak cíl podnikání, jednak účel řešení datového skladu. Proto by fáze strategie měla být plně v rukách vrcholového managementu. Cíle jako takové, v našem případě konkrétní cíle podnikové strategie, mohou být krátkodobé a dlouhodobé. Dlouhodobý cíl projektu datového skladu počítá nejen s jeho vybudováním, ale definuje i strategii správy datového skladu, přičemž počítá i s takovými důležitými věcmi, jako je dokumentace datového skladu a školení jeho uživatelů. V této fázi se definuje i základ architektury podnikového datového skladu.

Definice

Během fáze definice se snažíme jasně definovat rozsah a cíl přírůstkového vývoje. Vytvoří se počáteční přírůstek, konceptuální modely, zdokumentují se zdroje dat a přesně se vymezí rozsah kvality těchto údajů. V této fázi se navrhuje jednak architektura datového skladu, jednak architektura technických prostředků. V této fázi se také zaměříme na pochopení struktury operačních a externích zdrojů údajů. Stanovíme krátkodobé a dlouhodobé obchodní cíle, pro jejichž podporu datový sklad budujeme.

Analýza

Cílem fáze analýzy je zaměřit se na informace o uživatelích, získávání dat a požadavků na přístup k datům na obchodní analýzu a přijímání rozhodnutí. Relační a multidimenzionální modely se vytvářejí pro datový sklad, metadata, a když je to vhodné, tak i pro datové trhy. Během této fáze se dokončí i výběr nástroje pro všechny komponenty datového skladu. V této fázi řešíme problémy kvality dat a stanovujeme požadavky na metadata.

Návrh

Cílem fáze návrhu je transformovat požadavky získané během fáze analýzy do detailních podmínek návrhu a dokončit instalaci technické architektury.

Sestavení

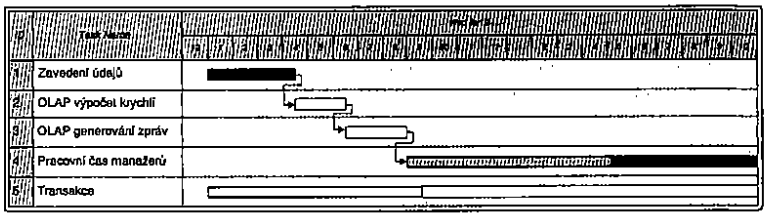
Cílem fáze sestavení je vytvořit a otestovat navržené databázové struktury, moduly získávání dat, moduly správy datového skladu, moduly metadat, moduly přístupu k datům a sestavy a dotazy.

Produkce

Dostáváme se do závěrečné fáze, do fáze přechodu do produkce. V této fázi nainstalujeme datový sklad, začneme ho používat a zároveň začneme řídit růst a údržbu datového skladu.

Provoz datového skladu

Při návrhu koncepce řešení musíme vycházet z každodenního provozu datového skladu a zabezpečit, aby manageři a analytici měli údaje k dispozici skoro v reálném čase. Typickým příkladem pro výraz „skoro v reálném čase" je následující časová tabulka, která zobrazuje časovou souslednost procesů analýzy. **Pozor, údaje v prvních třech řádcích se týkají předcházejícího dne**, což znamená, že v našem případě se po půlnoci vždy analyzují údaje z předcházejícího dne.



Obr. 2.8 Časová návaznost procesu OLAP v rámci jednoho dne.

U slovního popisu celého procesu znázorněného v časové tabulce bude lepší postupovat zdola nahoru.

Transakce

V posledním řádku (5) vidíme, že transakce dnešního dne běží nepřetržitě 24 hodin. Může se jednat o výrobní proces, elektronický obchod, bankovní transakce, prostě proces OLTP, který je zdrojem údajů pro analýzy. Tyto analýzy dnešních údajů se budou vykonávat až po půlnoci, aby zítra na začátku pracovní doby byly k dispozici. Kdybychom se rozhodli využít systém OLTP i pro analýzy OLAP v čase od 00:00 do 09:00, tak budou běžet oba dva procesy souběžně, tzn. dnešní transakce a analýza včerejších údajů.

Pracovní čas manažerů

Začátek pracovního času manažerů (v naší tabulce 09:00) je rozhodujícím okamžikem, kdy by měly být výsledky analýzy a příslušné reporty těmto manažerům k dispozici. Na našem grafu jsme dokonce znázornili fakt, že pracovní doba manažerů obvykle nekončí standardním firemním „padla" v 16:00, to proto, že právě tento pracovní čas je pro mnohé manažery charakteristický.

Proces přípravy údajů a analýza

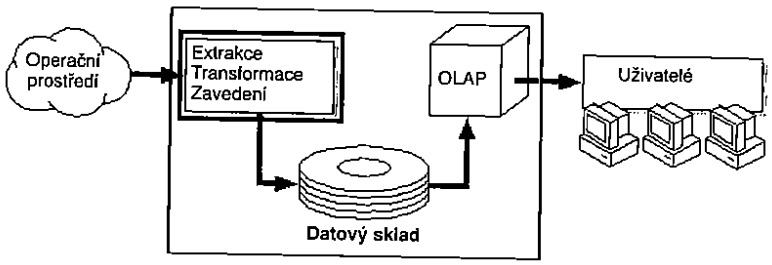
Aby mohli manažeři získat ve stanoveném čase kvalitní podklady pro podporu rozhodování, musí proběhnout příprava údajů pro analýzu, jejich zavedení do datového skladu a následná analýza. V našem příkladě proces ETL (*řádek 1 časové tabulky*) začne ihned po půlnoci o 00:00. Po zhruba třech hodinách je tato etapa ukončena a může začít přepočet krychlí a ostatní agregační a analytické výpočty (*řádek 2 časové tabulky*). Na tento proces naváže generování sestav, reportů a podobně (*řádek 3 časové tabulky*). Samozřejmě je potřebné počítat i s určitou rezervou, například pro nepředvídatelné situace a podobně.

Kapitola 3

Příprava údajů – etapa ETL

S nasazením technologie business intelligence a data warehouse (BI/DW) prakticky nikdy nezačínáme, jak se říká, „na zelené louce". Obvykle se před zavedením těchto technologií pro procesy rozhodování používají údaje získávané z primárních transakčních systémů **OLTP** (Online Transaction Processing), v lepším případě zpracované do sestav. Tyto sestavy jsou potom (zpravidla ručně nebo pomocí softwaru typu MS Office) zpracovávány do manažerských podkladů pro účely rozhodování.

Údaje pro proces business intelligence a data warehouse tedy pocházejí z různých nehomogenních zdrojů. Mohou to být údaje ze souborových databází (Access, dBase...), údaje z databází spravovaných některým databázovým serverem (Oracle, Informix, Microsoft SQL Server, Sybase, Interbase, Ingres...). Mohou to být údaje vyexportované nějakou databázovou platformou do tzv. flat file a podobně. Příprava a zavedení údajů je důležitou součástí každého řešení datového skladu. Údaje z operačního prostředí je potřebné před zavedením do datového skladu vyextrahovat, vyčistit, upravit a až následně ve vhodné formě do datového skladu zavést.



Obr. 3.1 Datový sklad

Extrakce, transformace a zavedení

Nástroje a postupy **ETL** (Extraction, Transformation, Loading), případně vyjádřené jinou terminologií **ETT** (Extraction, Transformation, Transport), jsou velmi důležitou součástí každého projektu datového skladu. Celý proces ETL je komplexní a ve většině případů

časově poměrně náročný. U některých implementací může zabrat i více než polovinu celkového času, úsilí a tedy i velkou část nákladů potřebných na vytvoření datového skladu.

Když se blíže podíváme na jednotlivé etapy procesu ETL:

- **Extrakce** – výběr dat prostřednictvím různých metod,
- **Transformace** – ověření, čištění, integrování a časové označení dat,
- **Loading** – přemístění dat do datového skladu.

zjistíme, že hlavním cílem etapy ETL je centralizace údajů, tzn. jejich shromáždění z mnoha zpravidla nehomogenních a různorodých zdrojů z databází OLTP, a naplnění datového skladu určenými údaji v požadovaném čase. Tyto dva ekvivalentní pojmy (ETT nebo ETL) popisují řadu procesů, jejichž úlohou je extrakce údajů ze zdrojových systémů, jejich transformace a vyčistění a přenos údajů do datového skladu. Údaje se tedy v této etapě nejen přenášejí, ale i zpracovávají, například indexují, sumarizují, zjišťují se případné změny struktur zdrojových údajů potřebných pro datový sklad, podle potřeby se mění struktura klíčů a udržují se metadata, tedy data o datech, v tomto případě předpisů a definicí pro přenos a zpravování údajů.

Počátečním naplněním datového skladu údaji z operačních databází úloha ETL samozřejmě nekončí, datový sklad se přece v pravidelných intervalech plní aktualizovanými daty. Je potřebné si uvědomit, že na úrovni ETL se pracuje s údaji, z kterých se později stanou informace. I tak by ale údaje zaváděné do datového skladu měly být kvalitní, přesné, k věci a aktuální, a tedy užitečné pro uživatele. Kromě kvality údajů je samozřejmě důležitá i jejich dostupnost, aby mohli jednotliví uživatelé datového skladu, například manažeři, obchodníci a analytici, používat datový sklad účinně a efektivně.

Oblast vynášení údajů

Ještě předtím, než se budeme podrobněji věnovat jednotlivým etapám ETL, je potřebné upozornit na jakési „staveniště" datového skladu, kterým je oblast pro vynášení údajů. Z hlediska implementace se může jednat například o paměť operačních dat, adresář textových nebo flat souborů, tabulky v relační databázi nebo vlastní struktury údajů, které používají nástroje určené pro vynášení dat. Z hlediska principu můžeme použít dva modely vynášení:

- Model lokálního vynášení.
- Model vzdáleného vynášení.

Výběr modelu závisí na uživatelských požadavcích a samozřejmě na objemu údajů a kvalitě a rychlosti připojení.

Model lokálního vynášení

Procesy úpravy a transformace údajů se v tomto případě vykonávají nejdříve, a to lokálně v operačním prostředí a až potom se přenášejí do vynášecí oblasti. Tento způsob může značně zatěžovat operační paměť počítače, takže je ho možné využívat jen u méně zatížených systémů.

Model vzdáleného vynášení

V tomto případě se „surová" data nejdříve přenesou z operačního prostředí do vynášecí oblasti, nebo dokonce přímo do prostředí datového skladu, a až tady se zpracují.

Extrakce

Údaje, které chceme přenést do datového skladu, jsou jednak umístěny v různých nehomogenních operačních prostředích, hardwarových platformách (PC, mainframe, iMac...), operačních systémech (Windows, Unix, Linux, Sun, Solaris...), databázových systémech (MS SQL Server, Oracle, Informix, IBM DB2...), archivních systémech, podnikových systémech (SAP, PeopleSoft, Baan...), a co je ještě horší, mohou se vyskytovat v rozličných formátech. Úlohou extrakce je právě získat údaje právě z takových zdrojů.

Kapitolu samu pro sebe tvoří archivní data, která obsahují historické údaje. Hlavní rozdíl mezi datovým skladem a archivem je v tom, že údaje v datovém skladu se pravidelně obnovují. Údaje z archivů jsou nezastupitelným zdrojem historických dat při prvním naplnění datového skladu. Naproti tomu archivní data nepoužíváme pro obnovu údajů v datovém skladu.

Kromě interních údajů z našeho podnikatelského prostředí je někdy potřebné pracovat i s externími údaji. Tyto údaje můžeme získat analýzou konkurenčního prostředí, zakoupením údajů o zákaznících, nebo i stáhnutím údajů volně přístupných na Internetu. Z toho vyplývá, že tady nemůžeme periodicky odebírat vzorky, jako jsme zvyklí u interních údajů. Externí údaje proto vyžadují nepřetržité monitorování za účelem určení, kdy jsou dostupné.

Pro extrakci jsou k dispozici různé postupy, nástroje a technologie. Můžeme vytvářet vlastní aplikace ve vyšších procedurálních programovacích jazycích, C++, C# nebo v procedurálních nadstavbách jazyka SQL (T-SQL, PL/SQL...). Pro menší množství údajů je výhodné vytvořit přístupovou bránu (gateway). Tato metoda ale pro větší objemy údajů zatěžuje síť. Někdy můžeme použít výstupy z vlastních podnikových systémů, které umožňují konverzi a vyčištění údajů. Při správně navrhnuté etapě ETL máme k dispozici metadata pro všechny fáze této etapy. Tato metadata obsahují informace o místě, typu, přístupových privilegiích a struktuře zdroje údajů.

Transformace

Říká se, že je lepší nemít žádná data, než mít data nekvalitní. Když jsou data v datovém skladu nekvalitní, snižuje to důvěru v taková řešení a datový sklad se oprávněně nevyužívá. Jádro problému je v tom, že nekvalitní data se dost často vyskytují ve zdrojových systémech. Použití nekvalitních údajů vede k chybným nebo minimálně nepřesným sestavám a následně k chybným obchodním rozhodnutím. Představme si, že se na základě analýz, například na základě výsledků Data Miningu, rozhodneme oslovit marketingovou kampaní určitou skupinu zákazníků a chceme to realizovat formou poštovních zásilek, z některé pobočky nebo od obchodního zástupce získáme adresy zákazníků v této podobě:

```
name                 address1
-------------------- ----------------------------------------
Nowmer Sheri         Bailey Road 2433 Tlaxiaco 15057 Mexico
Whelply Derrick      Dewing Avenue Sooke 2219 Canada
Derry Jeanne         7640 First Ave. Issaquah USA 73980
Spence Michael       Tosca Way Burnaby 337 74674  Canada
Gutierrez Maya       8668 Via Neruda Novato USA
Damstra Robert       1619 Stillman Court Lynnwood 90792 USA
Kanagaki Rebecca     D Mt. Hood Circle 2860 Tlaxiaco Mexico
```

O automatickém vyplňování adres na poštovních obálkách si můžeme jen nechat snít. I přes vynaloženou námahu, když to udělám pracně „ručně", se nám dost zásilek vrátí jako nedoručené. Když máme databázi zákazníků v takovém tvaru, jako v následujícím výpisu, není co řešit, tak jakýkoliv kancelářský software adresy z takto organizované databázové tabulky nebo textového souboru natáhne a tisk obálek je úplně bezproblémový.

```
cname              address1              city          postal_code   country
------------------ --------------------- ------------- ------------- --------
Nowmer Sheri       2433 Bailey Road      Tlaxiaco      15057         Mexico
Whelply Derrick    2219 Dewing Avenue    Sooke         17172         Canada
Derry Jeanne       7640 First Ave.       Issaquah      73980         USA
Spence Michael     337 Tosca Way         Burnaby       74674         Canada
Gutierrez Maya     8668 Via Neruda       Novato        57355         USA
Damstra Robert     1619 Stillman Court   Lynnwood      90792         USA
Kanagaki Rebecca   2860 D Mt. Hood Circle Tlaxiaco     13343         Mexico
...
```

A to jsme uvedli jen primitivní příklad. Dobře navržený databázový sklad musí umožňovat o mnoho složitější dotazy, například najít správnou cílovou skupinu uživatelů pro marketingovou kampaň, zjistit, zda určitý konkrétní zákazník nebo skupina zákazníků kupuje příslušné produkty, pod pojmem skupina zákazníků můžeme rozumět například rodinu, komunitu (teenagerů, seniorů...). A to se stále jedná jen o typické úlohy. V životě firem ale nepanuje jen stereotyp, dochází například k akvizicím jiných společností a tím i převzetí jejich zákazníků a podobně. S postupem času bude pravděpodobně potřebné uskutečňovat potřebné změny v systémech OLTP, aby se zlepšila kvalita dat datového skladu.

Čištění údajů

Údaje z externích zdrojů mají určitou kvalitu, která je buď postačující, nebo nepostačující pro jejich zavedení do datového skladu. Často dokonce bývá kvalita údajů značně proměnlivá, například když pracovník vyplňoval údaje po „veselé" noci a podobně. Pro čištění údajů je v anglické terminologii užito několik rovnocenných termínů, například cleaning (čištění), scrubbing (vyčištění) a cleansing (pročištění).

Čištění údajů může být někdy velmi náročné, a tedy i nákladné. V některých případech dokonce ani nemá smysl čistit údaje s vysokými náklady, když je přínos pro podnikání zanedbatelný. Dokonce když i systémy OLTP obsahují kvalitní údaje, tyto údaje nemusí být zárukou kvalitního datového skladu. Systémy OLTP totiž neobsahují historické údaje.

Transformace

Transformace samotná je soubor úloh a úkonů, které vedou ke zvýšení kvality údajů, hlavně k odstranění anomálií. Anomálie nejsou v systémech OLTP zpravidla na závadu, když to řekneme trochu ironicky, tak se tyto anomálie v systémech OLTP roky budují. Vývoj některých systémů trvá poměrně dlouho, obměňují se verze softwaru, mění se vývojová prostředí a technologické platformy, na kterých se tento software vyvíjí. Mění se operační systémy.

Jako příklad můžeme uvést přechod z operačního systému MS DOS na Windows. V DOSu se zdála být pozice kódové stránky pro češtinu a slovenštinu od bratrů Kamenických neotřesitelná. V ní se ukládaly skoro všechny textové údaje, jména, adresy a podobně. Kdo z uživatelů Windows si ale dnes vzpomene, co to ta „stránka Kamenických" vlastně byla? Snad jen pokud bude postaven před úlohu zpracovávat textové údaje, kde byla tato kódová stránka použita.

Do hry ale kromě technických záležitostí vstupuje i lidský faktor, například pravopisné chyby, pořadí zadávání atd. Během čištění dat, tedy sjednocuje formátování údajů, sjednocuje přiřazení datových typů, jednotek míry a peněžních měn. Údaje v databázích OLTP často obsahují různě kódované údaje nebo i primární klíče, které se skládají z více částí. Například kód výrobku, ze kterého se dá vyextrahovat například datum výroby, číslo závodu, ve kterém byl výrobek vyroben, a podobně. Při zavádění těchto údajů do datových skladů je potřebné rozložit tyto údaje na atomické hodnoty.

Pokusíme se upozornit na nejčastěji se vyskytující problémy při transformaci údajů.

Nejednoznačnost údajů

Jedním z problémů, které musíme při transformaci údajů řešit, je i nejednoznačnost údajů. Například údaje o pohlaví zákazníka mohou být uloženy různým způsobem.

```
name                 gender
-------------------- -------
Nowmer Sheri         Female
Whelply Derrick      male
Derry Jeanne         F
Spence Michael       man
Gutierrez Maya       woman
Damstra Robert       F
Kanagaki Rebecca     female
```

Po transformaci by mohl být tento sloupec v jednotném tvaru:

```
name                 gender
-------------------- -------
Nowmer Sheri         F
Whelply Derrick      M
Derry Jeanne         F
Spence Michael       M
Gutierrez Maya       F
Damstra Robert       F
Kanagaki Rebecca     F
```

Chybějící hodnoty a duplicitní záznamy

Starosti nám mohou způsobovat i chybějící hodnoty (sloupce relačních databází obsahující hodnotu NULL), případně otevřená nebo skrytá duplicita záznamů. Duplicita údajů je menší problém, když je něco navíc, tak se to dá vždy odstranit. V některých případech to ale může být dost časově náročné. Větší problém představují chybějící údaje. V takovém případě máme více možností. U malého objemu chybějících údajů je můžeme ignorovat. Někdy mů-

žeme chybějící hodnoty doplnit z jiných zdrojů. Když nemáme jinou možnost, můžeme je s vhodným příznakem dočasně ponechat v systému OLTP a zapracovat později.

Konvence názvů pojmů a objektů

Když slučujeme údaje z různých zdrojů, které v podstatě popisují stejný jev, ale mají jednotlivé entity vedeny pod různými názvy, tak musíme sloučit terminologii a vytvořit jednotnou konvenci názvů.

Různé peněžní měny

Kapitolou samou o sobě jsou hodnoty měny. Suma 200,50 znamená něco úplně jiného v dánských korunách než ve forintech. Tato problematika je velmi aktuální například při přechodu na euro. V přechodném období se uváděly ceny v místních měnách i v EUR.

Formáty čísel a textových řetězců

Údaje jsou v relačních databázích a souborech uloženy v různých druzích formátů. Největší problém je s ukládáním číselných údajů. Nejčastěji se pro tyto údaje používají numerické a řetězcové datové typy. Do numerického datového formátu se ukládá číslo jako numerická hodnota, do řetězcového datového typu se číslo ukládá jako postupnost číslic a jiných znaků, například desetinných čárek a mezer.

V čem je tedy problém? Když máme například rodné číslo ve tvaru 6808214321, můžeme ho v této podobě uložit jako číslo, ale i jako textový řetězec. V častěji uváděné podobě 680821/4321 můžeme toto rodné číslo uložit jen jako textový řetězec. Podobně je to i s poštovními směrovacími čísly. V podobě 12345 ho můžeme uložit i jako číslo, i jako textový řetězec. Ve tvaru 123 45 ho můžeme uložit jen jako textový řetězec. Numerické formáty se pro ukládání PSČ nepoužívají i z jiného důvodu. Mnoho z nich totiž začíná nulou, a tak se pětimístné PSČ 03483 změní na čtyřmístné 3483, protože nuly před první platnou číslicí se v numerických formátech vynechávají.

Referenční integrita

Kromě hodnot jsou v údajích skryty i různé vztahy, například master – detail, organizační struktura firmy, hierarchická struktura zaměstnanců a podobně. Ale údaje jsou dynamické, organizační struktura se mění, často bez dokumentace a adekvátních změn v databázích OLTP. Když se zruší nějaké oddělení a zůstanou po něm nějaké záznamy, mohou tito „údajoví sirotkové" zkreslit údaje a tedy nepříznivě ovlivnit kvalitu údajů.

Chybějící datum

Čas plní v datovém skladě významnou úlohu. Od něj se vše odvíjí a skoro každá analytická databáze má časovou dimenzi. V mnohých transakčních systémech se údaje neoznačují časovými údaji, v jiných je naopak čas důležitou veličinou. Například datum objednávky, transakce a podobně. Časový údaj musí být přítomný v datech před jejich zavedením do datového skladu, nebo se musí určit a přidat při zavádění dat.

Je potřeba dobře uvážit, kdy a kde se transformace uskuteční. Můžeme ji vykonávat sériově nebo paralelně se zaváděním údajů. U sériového způsobu se transformace vykoná před zavedením dat do datového skladu. U paralelní metody se tento proces vykonává souběžně se zaváděním.

Přenos

Završením etapy ETL je přenos údajů z paměti zdrojových dat nebo přechodné vynášecí oblasti do datového skladu. Přenos spočívá v přesunu údajů a jejich uložení do databázových tabulek. Přenos by měl být plánovaný a automatizovaný. Při prvotním naplnění datového skladu může jít o obrovské množství údajů. Potom se už údaje zavádějí v pravidelných časových obdobích, například každý den, a to v takových objemech, kolik údajů za dané období v databázích OLTP vznikne.

Zavádění dat by mělo být plánované a hierarchizované. Stejně by mělo být automatizované na nejvyšší možnou míru. Po zavedení údajů zpravidla probíhá jejich indexování, aby byl přístup k nim optimalizovaný. Pro jednoznačnou identifikaci údajů se používají i uměle vytvářené klíče, s jejichž pomocí zajistíme jednoznačnost každého řádku v tabulce. Data datového skladu jsou totiž často kombinací mnoha transformovaných záznamů, které nemají žádné přirozené klíče, které by se daly použít pro jednoznačnou identifikaci.

Chyby a problémy etapy ETL

Proces ETL vždy neproběhne úspěšně. Problémy mohou být se spolehlivostí úložišť údajů (disky jsou mechanická zařízení, která se opotřebovávají), může dojít k výpadkům spojení, zdroje údajů se mohou měnit, například při upgradu systémů OLTP, které se nedokumentují v metadatech. Důležité je ověření údajů, protože když údaje nejsou ověřeny, tak může dojít k problémům při extrakci a transformaci. Podle závažnosti selhání je potřebné začít nanovo nebo můžeme pokračovat od místa selhání. Nepřesné nebo neúplné údaje mohou být příčinou nepřesnosti výsledků analýzy, což následně může vést k nesprávným obchodním, anebo ještě hůře k strategickým rozhodnutím.

Testování etapy ETL

U etapy ETL se asi nejvíce projeví pravdivost trochu ironické věty: „Nikdy není čas udělat něco pořádně, ale vždy je dost času udělat to potom ještě jednou." Proto je třeba etapu ETL důkladně otestovat nejdříve na simulovaných a později i na ostrých údajích. V etapě testování se už poprvé projeví i to, zda jsme etapu ETL dobře a podrobně zdokumentovali.

Ani po otestování a plném nasazení ETL nemáme jistotu, že vše bude stále fungovat k naší spokojenosti. Entropie (zjednodušeně řečeno, že věci ponechané samy sobě spějí od deseti k pěti) je totiž neúprosná. Objem datového skladu roste rychle a metrika zavádění a granularity dat vyžaduje pravidelnou revizi. Na vykonání procesů ETL je možné použít jednak specializované nástroje ať už od externích dodavatelů, nebo vyvinuté pro konkrétní projekt, různé brány a datové pumpy mezi databázovými systémy a interně vyvinuté nebo dodavatelské nástroje. Z konkrétních nástrojů můžeme vzpomenout Microsoft DTS (Data Transformation Services), DPS (Data Pipeline Services) od české firmy Adastra, Oracle Warehouse Builder (OWB) a mnohé další.

Jednoduchý ilustrační příklad transformace

Poměrně jednoduchou úlohou je „přesvědčit" prakticky libovolný databázový server, aby vyexportoval údaje z databázové tabulky do souboru flat. Pod tímto pojmem se skrývá textový soubor, kde jsou údaje odděleny nějakým zpravidla dopředu dohodnutým oddělovačem (čárka, mezera, tabulátor), například:

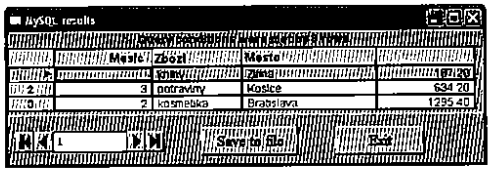
```
E,003715,4,153,09061987,0140000.00,"IRENE, HIRSH"    ,1,085.00,2,066.00,3,088.00,4,125.00
P,003715,01152000,01162000,00101,0005000.00,0007000.00,150.00,200.00,133.00,075.00,055.00,066.00,077.00
P,003715,02152000,02162000,00102,0003000.00,0008000.00,120.00,180.00,120.00,065.00,044.00,075.00,055.00
```

Oddělovač není nutnou podmínkou, můžeme použít i delimitované soubory, kde vycházíme z pevné struktury údajů

```
E0037154153090619870140000001RENE HIRSH     1085002066003088004125 00
P0037150115200001162000001010005000000007000001500020000133000750005500066000770 0
P0037150215200002162000001020003000000008000001200018000120000650004400075000550 0
```

Jako příklad uvedeme export z databáze spravované volně šiřitelným (a proto i populárním) databázovým serverem MySQL. Tento server nemá implementované žádné analytické nástroje, takže ani nemáme jinou možnost. Když máme údaje pod správou moderních výkonných (a drahých) databázových serverů typu Oracle, MS SQL Server, IBM DB2, můžeme tyto servery využít i pro analýzu a data warehouse. Export se bude týkat tabulky market, která má velmi jednoduchou strukturu

```
MESIC   ZBOZI       MESTO     CENA
------- ---------- -------- -------
      1 kosmetika  Zilina     194.9
      1 potraviny  Zilina      84.2
      1 knihy      Zilina     187.2
```



Obr. 3.2 Struktura údajů jednoduchého příkladu exportu

Už při letmém pohledu na tlačítka konzoly MySQL vidíme, že s exportem do souboru zřejmě nebudou větší problémy. Z uvedených údajů nám vznikne textový soubor, který, jak jsme už uvedli, dokáže načítat jakýkoliv databázový server.

```
# mesic       zbozi        mesto        cena
    1,'      knihy'  ,'      Zilina' ,'   187.20'
    3,'  potraviny'  ,'      Kosice' ,'   634.20'
    2,'  kosmetika'  ,'  Bratislava' ,'  1295.40'
```

K dispozici máme samozřejmě i jiné možnosti, například zaregistrovat tento zdroj údajů a přistupovat k němu přes rozhraní ODBC a podobně.

Příklad pro import cvičné databáze FoodMart

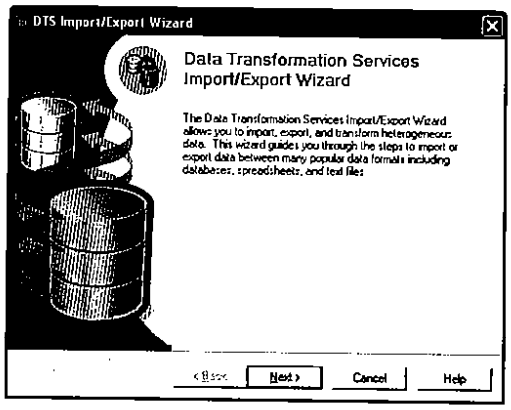
Cvičná databáze FoodMart se nainstaluje při instalaci analytických služeb MS SQL Serveru 2000. Tuto databázi využijeme v kapitolách věnovaných analýze OLAP a data miningu. Databáze je standardně dodávaná ve formátu MS Access, tj. soubor s příponou MDB. Má přibližně 28 MB a při standardní instalaci analytických služeb tento soubor najdeme v adresáři *C:\Program Files\Microsoft Analysis Services\Samples*.

Abychom si tento příklad zjednodušili, můžeme soubor FoodMart 2000.mdb překopírovat do nově vytvořeného adresáře *C:\FoodMart*. Naší úlohou je přenést údaje včetně struktur do nově vytvořené databáze MS SQL Serveru 2000 s názvem *FoodMart*. Pro úplnost, novou databázi vytvoříme například pomocí aplikace Enterprise Manager, pomocí položky menu New Database. Příslušné menu se zobrazí po klepnutí pravým tlačítkem myši na složku Databases v levém okně aplikace. Co se týká uživatelského rozhraní a logiky přístupu k transformaci, máme k dispozici dvě možnosti, přičemž fyzicky půjde stále o ten samý proces.

- **Import and Export Data** – tato utilita je dostupná z hlavního Windows menu MS SQL Serveru (*Start – MS SQL Server – Import and Export Data*).
- **DTS Package** – tato složka je dostupná například z aplikace MS SQL Server Enterprise Manager. (Windows menu: *Start – MS SQL Server – Enterprise Manager*).

Protože budeme přenášet více databázových tabulek, které potřebujeme přenést včetně jejich struktur, relačních vztahů a restrikcí, přičemž však nepředpokládáme žádné složitější transformace, tak bude jednodušší použít datovou pumpu jako aplikaci **Import and Export Data**.

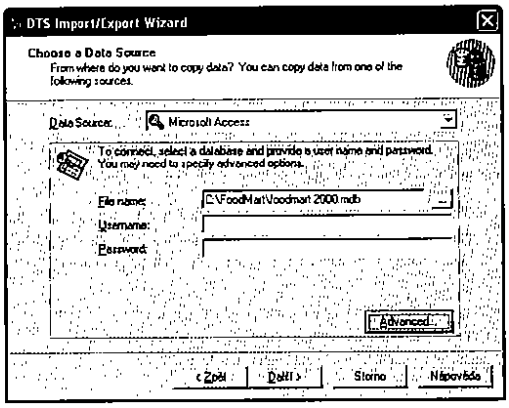
Aplikace *Import and Export Data* je navržena jako posloupnost dialogů, které tvoří průvodce exportem a importem údajů. První dialog s názvem *Data Transformation Services Import/Export Wizard* je jen jakýmsi představením funkcionality této aplikace.



Obr. 3.3 Úvodní obrazovka DTS Import/Export Wizard

Po stisku tlačítka *Next* se dostaneme do prvního interaktivního dialogu pro výběr zdroje údajů s názvem *Choose a Data Source*. Dialog se skládá z prvku pro výběr typu zdroje údajů a z formuláře pro zadání parametrů pro definování přístupu k tomuto zdroji. V našem případě specifikujeme jako zdroj údajů databázi *Microsoft Access*.

Při výběru datového zdroje si všimněme široké palety různých souborových databází, ale i databázových platforem, které jsou službou DTS podporovány. Další možnosti poskytuje rozhraní ODBC (Open Database Connectivity). Ve formuláři pro zadání parametrů zdroje údajů zadáme v případě souborové databáze nejdůležitější parametr, tedy fyzické umístnění souboru, v našem případě například ***C:\FoodMart\foodmart 2000.mdb***.

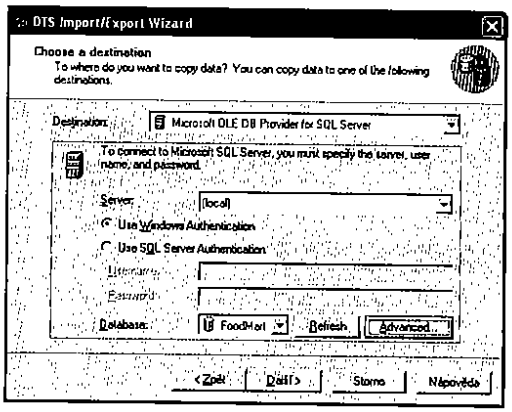


Obr. 3.4 Výběr zdroje údajů

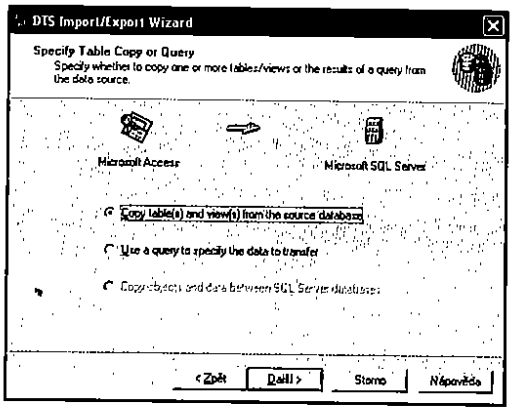
V dalším logickém kroku specifikujeme cílovou destinaci pro transformované údaje, v našem případě to bude nově vytvořená databáze *FoodMart* pod správou MS SQL Serveru 2000. V reálné aplikaci bychom museli zadat i parametry pro přístup k údajům. V našem cvičném příkladě jsme všechno ponechali na autentizaci operačního systému Windows. Pro výběr cílové databáze musíme znovu nastavit potřebné parametry v dialogu *Choose a Destination*.

Po stisku tlačítka *Next* následuje dialog s názvem *Specify Table Copy or Query*. V tomto dialogu máme možnost zvolit si metodu pro výběr množiny údajů, které potřebujeme přenést a případně přetransformovat. Máme na výběr tři možnosti:

- kopírovat vybrané tabulky a pohledy jako celky ze zdroje dat do cílové databáze,
- pomocí dotazu Query specifikovat podmnožiny údajů, které potřebujeme přetransformovat,
- kopírovat objekty a údaje mezi databázemi pod správou SQL Serveru.



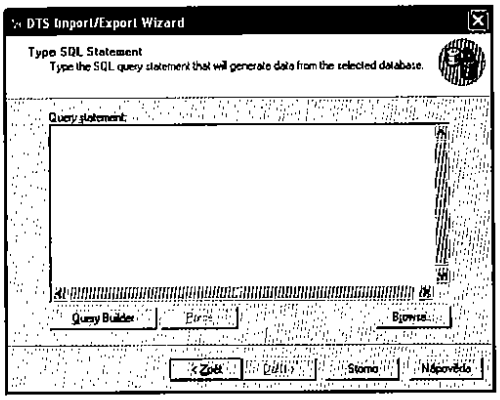
Obr. 3.5 Výběr destinace



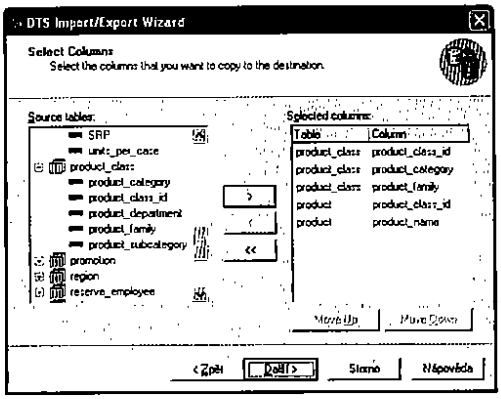
Obr. 3.6 Výběr objektů pro přenos údajů

V našem příkladě potřebujeme přenést struktury a údaje ze souborové databáze Access do MS SQL Serveru, takže třetí možnost do úvahy vůbec nepřipadá (v dialogu je zakázaná). Naším cílem je skoro lineární přenos (s minimem úprav a transformací) všech tabulek ze zdrojové databáze do cílové, takže ponecháme označenou první variantu *Copy table(s) and view(s) from the source database.*

Často však nejsme postavení před takovou jednoduchou úlohu jako v tomto případě, kdy cvičná databáze FoodMart obsahuje rozumné množství údajů, a to dokonce velmi vhodně strukturovaných. Pro proces ETL jsou spíše charakteristické různé transformace a očišťování údajů, abychom získali údaje vhodné pro analýzu OLAP. Proto nahlídneme i do druhé větve, kdy můžeme vytvořit dotaz SQL pro výběr údajů, které se budou transformovat. Tento postup se hodí například v případě, kdy už máme vytvořený dotaz SQL pro výběr údajů, které nás zajímají, nebo víme, jak ho na základě příslušných požadavků jednoduše vytvořit. Po potvrzení volby *Use a query to specify the data to transfer* můžeme zadat dotaz SQL přímo do jednoduchého editačního okna nebo využít možnosti skupiny dialo-



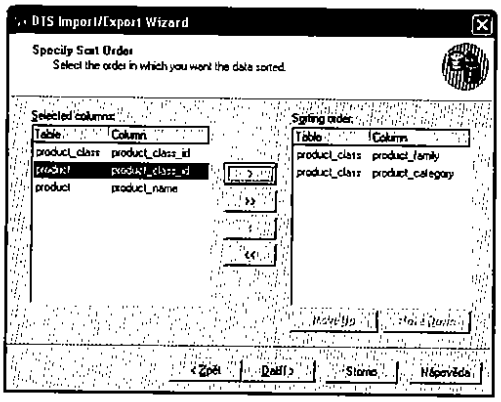
Obr. 3.7 Dialog pro přímé zadání příkazu SQL



Obr. 3.8 Query Builder – výběr sloupců

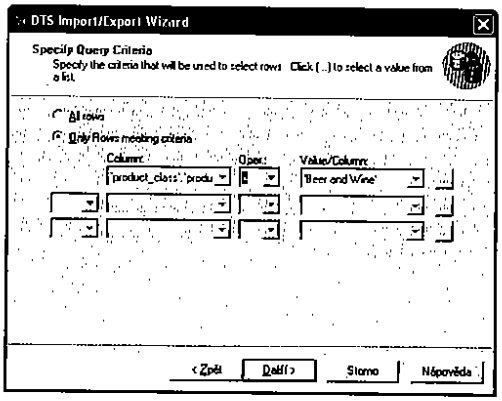
gů s názvem **Query Builder**, které slouží pro interaktivní vytvoření dotazu SQL. V prvním kroku nástroje Query Builder v dialogu *Select Column* vybereme požadované sloupce z příslušných tabulek.

Následuje dialog s názvem *Specify Sort Order* pro případné setřídění hodnot.



Obr. 3.9 Query Builder – dialog pro setřídění

Vytvoření dotazu SQL dokončíme zadáním omezujících podmínek pro výběr údajů a v případě potřeby stanovíme i pořadí sloupců pro setřídění záznamů.

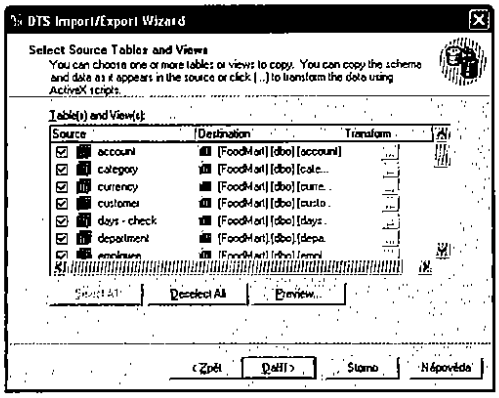


Obr. 3.10 Query Builder – vytvoření podmínky pro výběr údajů

Výsledkem našeho jednoduchého pokusu byl například dotaz SQL:

```
select `product_class`.`product_class_id`, `product_class`.`product_category`, `pro-
duct_class`.`product_family`, `product`.`product_class_id`, `product`.`product_name`
from `product_class`,`product`
where `product_class`.`product_category`='Beer and Wine'
order by `product_class`.`product_family`, `product_class`.`product_category`
```

Vraťme se však k cvičnému příkladu, ve kterém potřebujeme přenést celé tabulky.

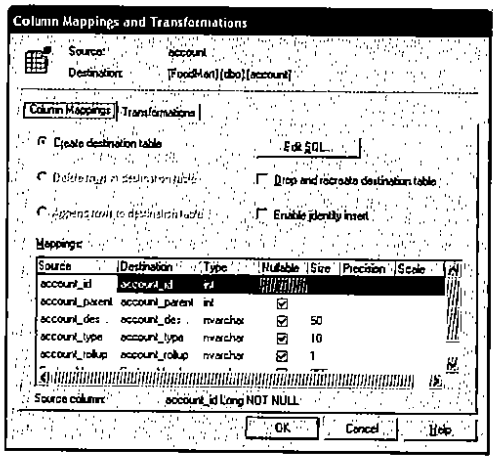


Obr. 3.11 Výběr tabulek a pohledů

Všimněme si třetího sloupce tabulky s názvem **Transform**. Po klepnutí na tlačítko s třemi tečkami můžeme definovat pravidla pro transformaci příslušné tabulky.

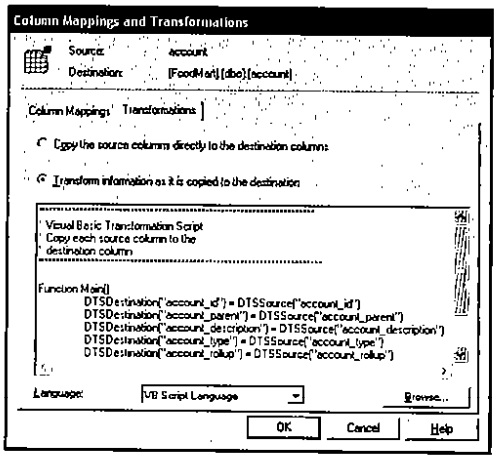
Aby bylo možné přenášet údaje ze zdrojové databáze do cílové, musí se v cílové databázi vytvořit identické objekty, například databázové tabulky, jako ve zdrojové databázi. Návrh cílové tabulky se samozřejmě přizpůsobí výběru sloupců. Když ze zdrojové tabulky vybereme jen některé sloupce, bude skript pro vytvoření nové tabulky v cílové databázi obsahovat jen tyto sloupce. Příkaz pro vytvoření tabulky v cílové databázi si můžeme prohlédnout, případně upravit v editačním dialogu, který se zobrazí po stisku tlačítka Edit SQL. Například když vybereme pro transformaci všechny sloupce z tabulky Account, navrhne průvodce příkaz pro vytvoření cílové tabulky ve tvaru:

```
CREATE TABLE [FoodMart].[dbo].[account]
(
  [account_id] int NOT NULL,
  [account_parent] int NULL,
  [account_description] nvarchar (50) NULL,
  [account_type] nvarchar (10) NULL,
  [account_rollup] nvarchar (1) NULL,
  [Custom Members] nvarchar (255) NULL
)
```



Obr. 3.12 Výběr jednotlivých sloupců

Druhá záložka dialogu umožňuje definovat skript pro transformaci údajů.



Obr. 3.13 Skript pro transformaci údajů

Na výběr máme dokonce k dispozici více skriptovacích jazyků, například **VBScript** (default) nebo **JScript**. Ukážeme si pro zajímavost příklad jednoduché transformace typu 1:1.

```
'**********************************************************************
'  Visual Basic Transformation Script
'  Copy each source column to the
'  destination column
'**********************************************************************

Function Main()
        DTSDestination("account_id") = DTSSource("account_id")
        DTSDestination("account_parent") = DTSSource("account_parent")
        DTSDestination("account_description") = DTSSource("account_description")
        DTSDestination("account_type") = DTSSource("account_type")
        DTSDestination("account_rollup") = DTSSource("account_rollup")
        DTSDestination("Custom Members") = DTSSource("Custom Members")
        Main = DTSTransformStat_OK
End Function
```

Zdálo by se, že cvičná databáze FoodMart neposkytuje žádné možnosti pro transformaci. Když se ale pozorněji podíváme na tabulku *customer*, kterou při analýze OLAP v další kapitole využijeme jako tabulku dimenzí regionálního typu, budeme mít mírné problémy, protože jméno a příjmení zákazníka máme uloženy ve dvou sloupcích tabulky a my budeme potřebovat specifikovat oba atributy jméno i příjmení jako jednu dimenzi.

Příklad údajů v tabulce **customer**:

```
customer_id lname               fname          city          state_province
----------- ------------------- -------------- ------------- ------------------
1           Nowmer              Sheri          Tlaxiaco      Oaxaca
2           Whelply             Derrick        Sooke         BC
3           Derry               Jeanne         Issaquah      WA
4           Spence              Michael        Burnaby       BC
5           Gutierrez           Maya           Novato        CA
6           Damstra             Robert         Lynnwood      WA
```

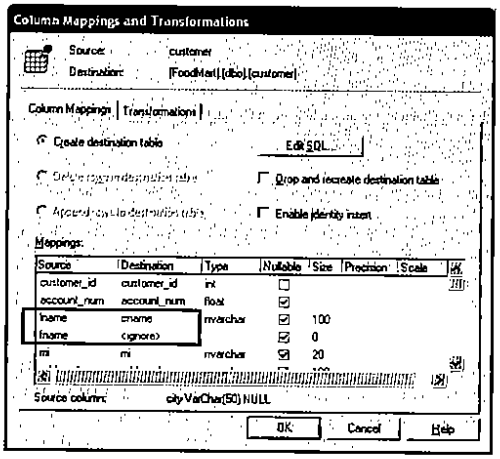
Proto napíšeme jednoduchý skriptový kód, pomocí kterého sloučíme oba dva sloupce do jednoho s názvem **cname**. Dříve než přistoupíme k vytvoření skriptu pro sloučení dvou sloupců, musíme modifikovat skript pro vytvoření tabulky v cílové databázi. Namísto původního příkazu CREATE TABLE:

```
CREATE TABLE [FoodMart].[dbo].[customer]
(
  [customer_id] int NOT NULL,
  [account_num] float NULL,
  [lname] nvarchar (100) NULL,
  [fname] nvarchar (50) NULL,
  [mi] nvarchar (20) NULL,
  [address1] nvarchar (100) NULL,
  [address2] nvarchar (100) NULL,
...
)
```

Použijeme tedy příkaz mírně modifikovaný, ve kterém namísto sloupců **lname a fname** vytvoříme jeden sloupec s názvem **cname** a samozřejmě s dostatečnou dimenzí rozsahu. Vidíme, že sloupec **lname** měl povolený rozsah 100 znaků a sloupec fname 50. Když k tomu připočteme jeden znak na mezeru mezi příjmením a jménem, zjistíme, že pro bezpečnou transformaci potřebujeme, aby sloupec cname měl minimálně 151 znaků. Modifikovaná část příkazu tedy bude ve tvaru:

```
CREATE TABLE [FoodMart].[dbo].[customer]
(
  [customer_id] int NOT NULL,
  [account_num] float NULL,
  [cname] nvarchar (151) NULL,
  [mi] nvarchar (20) NULL,
  [address1] nvarchar (100) NULL,
  [address2] nvarchar (100) NULL,
...
)
```

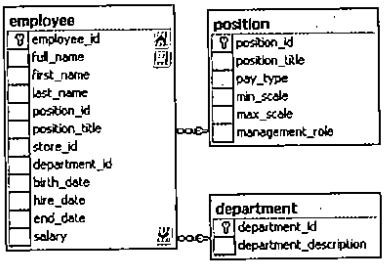
Také je potřebné změnit mapování sloupců například takto (změny jsou v obrázku vyznačené):



Obr. 3.14 Úprava mapování sloupců

Skript pro samotné sloučení sloupců potom bude velmi jednoduchý, změníme jen tučně označený řádek

```
Function Main()
        DTSDestination("customer_id") = DTSSource("customer_id")
        DTSDestination("account_num") = DTSSource("account_num")
```

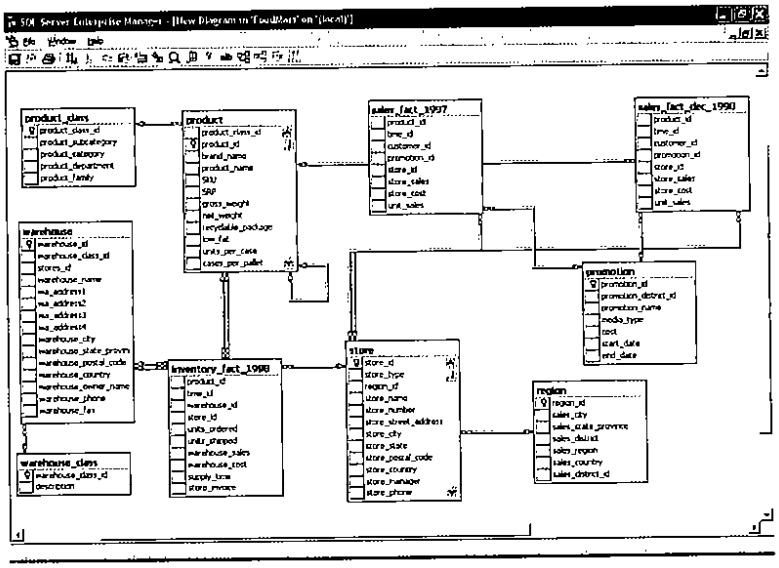


Obr. 3.18 Relační vazby mezi tabulkami EMPLOYEE, POSITION a DEPARTMENT

Skript pro vytvoření cizích klíčů pro definování vazeb mezi databázovými tabulkami.

```
ALTER TABLE employee ADD FOREIGN KEY(position_id) references position(position_id)
ALTER TABLE employee ADD FOREIGN KEY(department_id) references department(department_id)
```

Důležité jsou hlavně vazby mezi ostatními tabulkami, které jsou obrazem předmětu podnikání.



Obr. 3.19 Relační vazby mezi tabulkami

Skript pro vytvoření cizích klíčů pro definování vazeb mezi databázovými tabulkami.

```
ALTER TABLE inventory_fact_1998 ADD FOREIGN KEY(product_id) references product(product_id)
ALTER TABLE inventory_fact_1998 ADD FOREIGN KEY(warehouse_id) references warehoUSE(warehouse_id)
ALTER TABLE inventory_fact_1998 ADD FOREIGN KEY(store_id) references store(store_id)
ALTER TABLE sales_fact_1997 ADD FOREIGN KEY(store_id) references store(store_id)
ALTER TABLE sales_fact_dec_1998 ADD FOREIGN KEY(product_id) references product(product_id)
ALTER TABLE sales_fact_dec_1998 ADD FOREIGN KEY(promotion_id) references promotion(promotion_id)
ALTER TABLE sales_fact_dec_1998 ADD FOREIGN KEY(store_id) references store(store_id)
ALTER TABLE product ADD FOREIGN KEY(product_class_id) references product_class(product_class_id)
ALTER TABLE product ADD FOREIGN KEY(product_id) references product(product_id)
ALTER TABLE sales_fact_1997 ADD FOREIGN KEY(promotion_id) references promotion(promotion_id)
ALTER TABLE store ADD FOREIGN KEY(region_id) references region(region_id)
ALTER TABLE warehouse ADD FOREIGN KEY(warehouse_class_id) references
warehouse_class(warehouse_class_id)
```

Příklad pro přenos údajů z cvičné databáze NorthWind pro OLAP

Na rozdíl od cvičné databáze FoodMart, jejíž struktura už byla přímo předurčena pro OLAP, což znamená, že obsahovala tabulky faktů a dimenzí, cvičná databáze Northwind je typickým příkladem transakční databáze, tedy databáze OLTP. Abychom mohli nad jejími údaji vytvářet krychle OLAP a dělat analýzy, je potřebné ji v etapě ETL přetransformovat, tedy vytvořit tabulky, pomocí nichž budeme moci vytvářet dimenze a podobně.

Teorií OLAPu, tabulek faktů a tabulek dimenzí se podrobněji zabývá další kapitola, na tomto místě jen stručně připomeneme, že **fakta** jsou numerické měrné jednotky obchodování. Prvotní fakta, například objem prodeje, se mohou kombinovat nebo vypočítat pomocí jiných faktů a vytvořit tak měrné jednotky. Měrné jednotky se mohou uložit v tabulce faktů, případně vyvolat, když je to nevyhnutelné, na účely vykazování. Naproti tomu **dimenze** jsou textové popisy obchodování. Tabulky dimenzí nám umožňují zjistit, „kdo", „kdy", „proč" a „jak", pokud jde o obchodování a transakce prvků. Nejčastěji se používají časové, produktové a geografické dimenze.

Na základě údajů z databáze Northwind tedy vytvoříme a naplníme jednoduchý datový sklad, databázi Northwind_DW. Nad touto databází budeme potom vytvářet krychle OLAP.

Novou databázi na platformě MS SQL Serveru vytvoříme buď pomocí konzolové aplikace SQL Query Analyser příkazem

```
CREATE DATABASE Northwind_DW
```

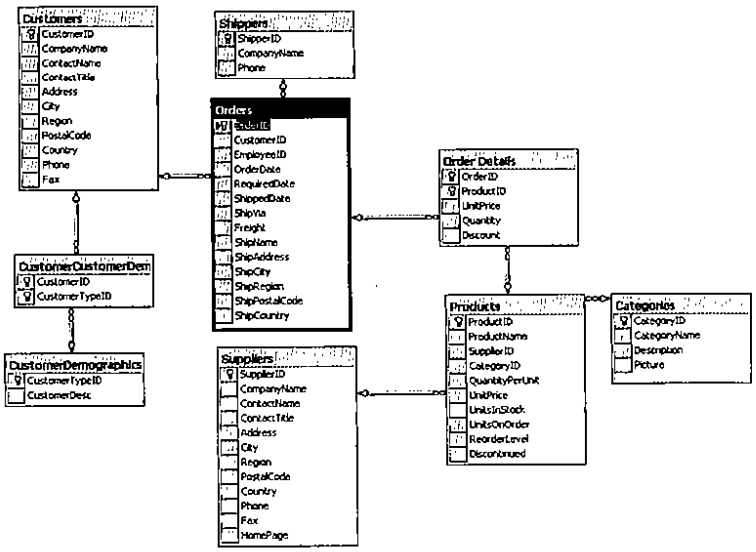
nebo interaktivně pomocí návrhového dialogu v aplikaci Enterprise Manager. Před jejím naplněním se musíme důkladně seznámit se strukturou zdrojové databáze Northwind.

Struktura cvičné databáze Northwind

Cvičná databáze, která obsahuje údaje z fiktivní firmy **Northwind**, se nainstaluje spolu s MS SQL Serverem 2000. Firma Northwind se zabývá importem a exportem potravinářských komodit. Databáze obsahuje 13 tabulek.

Orders	Region	Customers
Products	Territories	Shippers
Order Details	EmployeeTerritories	Suppliers
CustomerCustomerDemo	Employees	
CustomerDemographics	Categories	

Protože se chceme zabývat obratem firmy jako celku, nebudou nás zajímat výkony jednotlivých zaměstnanců. Proto tabulky Employees, EmployeeTerritories, Territories a Region nebudeme brát v úvahu. Z databázového diagramu je zřejmé, že vše se točí okolo objednávek, a tedy tabulka Orders bude tabulkou faktů. Vidíme, že můžeme sestavit zákaznickou (nebo demografickou) dimenzi, produktovou dimenzi, případně můžeme zkoumat i jednotlivé dopravce. I když jsme vyloučili tabulky, které se týkaly zaměstnanců, stále vidíme množství nadbytečných údajů a také nám chybí tabulka, která by posloužila jako podklad pro vytvoření časové dimenze.



Obr. 3.20 Struktura databáze Northwind

Tabulka faktů

Pro vytvoření tabulky faktů použijeme dvě tabulky, ORDERS a ORDER DETAILS.

Tabulka ORDERS

Jméno sloupce	Datový typ	Klíče
OrderID	int	PK.
CustomerID	nchar(5)	FK Customers(CustomerID)
EmployeeID	int	FK Employees(EmployeeID)
OrderDate	datetime	
RequiredDate	datetime	
ShippedDate	datetime	
ShipVia	int	FK Shippers(ShipperID)
Freight	money	
ShipName	nvarchar(40)	
ShipAddress	nvarchar(60)	
ShipCity	nvarchar(15)	
ShipRegion	nvarchar(15)	
ShipPostalCode	nvarchar(10)	
ShipCountry	nvarchar(15)	

Tabulka ORDER DETAILS

Jméno sloupce	Datový typ	Klíče
OrderID	int	PK FK Orders(OrderID)
ProductID	int	PK, FK Products(ProductID)
UnitPrice	money	
Quantity	smallint	
Discount	real	

Do nové databáze Northwind_DW budeme potřebovat sloupce (označili jsme je tučným fontem):

- **ID_PRODUCT**
- **ID_CUSTOMER**
- **ID_TIME**
- **QUANTITY**
- **PRICE**

Sloupce ID_PRODUCT, ID_CUSTOMER a ID_TIME jsou přitom cizí klíče do tabulek dimenzí a sloupce QUANTITY (množství) a PRICE (cena) obsahují měrné jednotky obchodování. Hodnotu do sloupce PRICE musíme vypočítat na základě hodnot ve sloupcích podle vztahu

```
PRICE = QUANTITY * UNITPRICE * (1-DISCOUNT).
```

Na základě uvedené analýzy můžeme vytvořit dotaz SQL, s jehož pomocí vypíšeme údaje ze zdrojové databáze v takovém tvaru, jak je budeme ukládat do tabulky faktů v cílové databázi.

```
SELECT ProductID, CustomerID, CAST(OrderDate AS INT) AS ID_Time,
       Quantity, Quantity * UnitPrice * (1-Discount) AS Price
FROM Orders join [Order Details] ON
   Orders.OrderID = [Order Details].OrderID
```

```
ProductID   CustomerID ID_Time     Quantity Price
----------- ---------- ----------- -------- ----------------------
11          VINET      35248       12       168.0
42          VINET      35248       10       98.0
72          VINET      35248       5        174.0
14          TOMSP      35249       9        167.39999
51          TOMSP      35249       40       1696.0
41          HANAR      35252       10       77.0
51          HANAR      35252       35       1261.4
65          HANAR      35252       15       214.20001
22          VICTE      35252       6        95.760002
57          VICTE      35252       15       222.3
...
```

Tabulka má celkem 2 155 řádků. Podle výsledků dotazu SQL vidíme, že je vše v pořádku, a můžeme tedy v databázi Northwind_DW vytvořit novou tabulku a naplnit ji údaji. Pro jednoduchost a názornost nebudeme při návrhu vytvářet omezení. Tabulku F_ORDERS vytvoříme příkazem

```
CREATE TABLE f_orders
(
    id_product   INT,
    id_customer  NCHAR(5),
    id_time      INT,
    quantity     REAL,
    price        REAL,
)
```

a naplníme ji pomocí příkazu INSERT INTO, přičemž využijeme mírně modifikovaný příkaz SELECT, který jsme použili při ladění návrhu.

```
INSERT INTO Northwind_DW..f_orders (id_product, id_customer, id_time, quantity, price)
   SELECT ProductID, CustomerID, CAST(OrderDate AS INT) AS ID_Time,
```

```
        Quantity, Quantity * UnitPrice * (1-Discount) AS Price
  FROM Northwind..Orders join Northwind..[Order Details] ON
       Northwind..Orders.OrderID = Northwind..[Order Details].OrderID
```

Tabulka pro vytvoření časové dimenze

V tabulce ORDERS zdrojové databáze je u každé objednávky uvedeno datum jako datový typ DATETIME. Při transformaci do nové databáze jsme toto datum pomocí příkazu CAST(OrderDate AS INT) překonvertovali na celočíselný datový typ INTEGER. K tomuto cizímu klíči se bude vztahovat tabulka D_TIME, kterou budeme muset vytvořit a naplnit vygenerovanými údaji. Tabulku D_TIME vytvoříme pomocí příkazu

```
CREATE TABLE d_time
(
    id_time INT,
    datum   DATETIME,
    den     INT,
    mesiac  INT,
    kvartal INT,
    rok     INT,
    CONSTRAINT pk_d_time PRIMARY KEY (id_time)
)
```

Při generování hodnot tabulky bychom měli brát v úvahu i dostatečnou „zásobu" údajů. Když chceme mít „živou" analytickou aplikaci několik let, nic nám nebrání naplnit tabulky pro časovou dimenzi na dostatečný počet záznamů dopředu. Ročně je to typicky 365 záznamů, takže z hlediska datového skladu je to naprosto zanedbatelné. Není samozřejmě problém kdykoliv přidat nové záznamy, ale člověk zapomíná a najít potom takovou triviální logickou chybu nebývá vždy jednoduché. Skript v jazyce T-SQL generuje data ve stanoveném časovém intervalu. Pro generování částí data (den, měsíc, kvartál, rok) použijeme funkce DAY, MONTH, YEAR a DATEPART.

```
DECLARE @den DATETIME
SET @den = '1995-01-01'
WHILE @den < '2004-01-01'
  BEGIN
    INSERT d_time(id_time, datum, den, mesiac, kvartal, rok)
    VALUES (CAST(@den AS INT), @den, DAY(@den), MONTH(@den), DATEPART(qq, @den), YEAR(@den))
   SET @den = DATEADD(d,1,@den)
  END
```

Tabulka pro vytvoření produktové dimenze

Tabulku pro produktovou dimenzi vytvoříme z údajů v tabulkách PRODUCTS a CATEGORIES, přičemž z hlediska hierarchie půjde o dvě úrovně: produkty a kategorie produktů.

Tabulka PRODUCTS

Jméno sloupce	Datový typ	Klíče
ProductID	int	PK
ProductName	nvarchar(40)	
SupplierID	int	FK Suppliers(SupplierID),
CategoryID	int	FK Categories(CategoryID),
QuantityPerUnit	nvarchar(20)	
UnitPrice	money	
UnitsInStock	smallint	
UnitsOnOrder	smallint	
ReorderLevel	smallint	
Discontinued	bit	

Tabulka CATEGORIES

Jméno sloupce	Datový typ	Klíče
CategoryID	int	PK
CategoryName	nvarchar(15)	
Description	ntext	
Picture	image	

I v tomto případě vytvoříme dotaz SQL, s jehož pomocí vypíšeme údaje ze zdrojové databáze v takovém tvaru, jak je budeme ukládat do tabulky produktové dimenze v cílové databázi.

```
SELECT ProductID, ProductName, CategoryName
FROM  Products join Categories on
      Products.CategoryID = Categories.CategoryID
```

```
ProductID   ProductName                              CategoryName
----------- ---------------------------------------- ---------------
1           Chai                                     Beverages
2           Chang                                    Beverages
3           Aniseed Syrup                            Condiments
4           Chef Anton's Cajun Seasoning             Condiments
5           Chef Anton's Gumbo Mix                   Condiments
6           Grandma's Boysenberry Spread             Condiments
7           Uncle Bob's Organic Dried Pears          Produce
8           Northwoods Cranberry Sauce               Condiments
9           Mishi Kobe Niku                          Meat/Poultry
10          Ikura                                    Seafood
...
```

V databázi Northwind_DW vytvoříme novou tabulku D_PRODUCTS a naplníme ji údaji.

```
CREATE TABLE d_products
(
    id_product      INT,
    productname     NVARCHAR(40),
    categoryname    NVARCHAR(15),
    CONSTRAINT pk_d_products PRIMARY KEY (id_product)
)

INSERT INTO Northwind_DW..d_products (id_product, productname, categoryname)
  SELECT ProductID, ProductName, CategoryName
  FROM  Northwind..Products join Northwind..Categories on
        Northwind..Products.CategoryID = Northwind..Categories.CategoryID
```

Tabulka pro vytvoření zákaznické (regionální) dimenze

Tabulku pro produktovou dimenzi vytvoříme z údajů v tabulce CUSTOMERS, přičemž využijeme jen tyto sloupce.

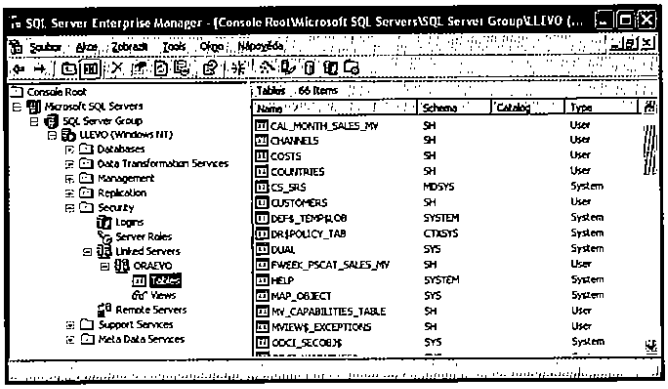
Tabulka CUSTOMERS

Jméno sloupce	Datový typ	Klíče
CustomerID	nchar(5)	PK Cust...
CompanyName	nvarchar(40)	
ContactName	nvarchar(30)	
ContactTitle	nvarchar(30)	
Address	nvarchar(60)	
City	nvarchar(15)	
Region	nvarchar(15)	
PostalCode	nvarchar(10)	
Country	nvarchar(15)	
Phone	nvarchar(24)	
Fax	nvarchar(24)	

V databázi Northwind_DW vytvoříme novou tabulku D_CUSTOMERS a naplníme ji údaji.

```
CREATE TABLE d_customers
(
    customerid  NCHAR(5),
    companyname NVARCHAR(40),
    contactname NVARCHAR(30),
    city        NVARCHAR(15),
    region      NVARCHAR(15),
    country     NVARCHAR(15),
    CONSTRAINT pk_d_customers PRIMARY KEY (customerid)
)
```

```
PROD_CATEG PROD_SUBCATEGORY                PROD_NAME
---------- ------------------------------- ----------------------------------
Women      Trousers - Women                Gurfield& Murks Pleated Trousers
Women      Trousers - Women                Gurfield& Murks Pleated Trousers
Girls      Trousers And Jeans - Girls      Coin Pocket Twill Cargo Trousers
Boys       Shirts - Boys                   And 2 Crosscourt Tee Kids
Boys       Shirts - Boys                   And 2 Crosscourt Tee Kids
Boys       Shirts - Boys                   And 2 Crosscourt Tee Kids
Men        Shorts - Men                    Kahala Pleated Chino Short
Boys       Shirts - Boys                   And 2 Crosscourt Tee Kids
Women      Shoes - Women                   Vunelli Bb-956
```

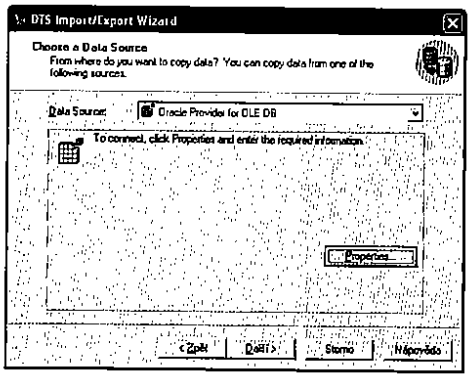


Obr. 3.27 Připojení k databázovému serveru ORACLE

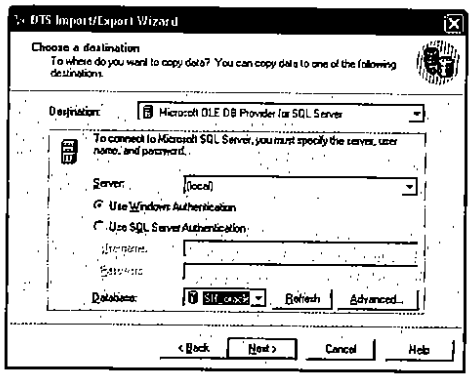
Import údajů z cvičného schématu SCOTT

Pro přenos údajů použijeme datovou pumpu DTS, přičemž parametry přenosu nastavíme pomocí aplikace *Data Transformation Services Import/Export Wizard*.

Pokračujeme výběrem typu zdroje údajů a zadáváním parametrů pro definování přístupu k tomuto zdroji. V našem případě jsme specifikovali jako zdroj údajů Oracle Provider for OLE DB, případně můžeme použít připojení k databázi Oracle přes rozhraní ODBC. Jako typ zdroje údajů zvolíme název databáze a nastavíme přístupové parametry k údajům ve schématu SH. V našem případě jsme předtím pomocí administrátorských nástrojů Oracle 9i nastavili přístupové heslo a odemkli schéma uživatele SH. Po nastavení všech parametrů můžeme právě vytvořené připojení vyzkoušet. Jako cílovou destinaci vybereme nově vytvořenou databázi na MS SQL Serveru s názvem SH_oracle.

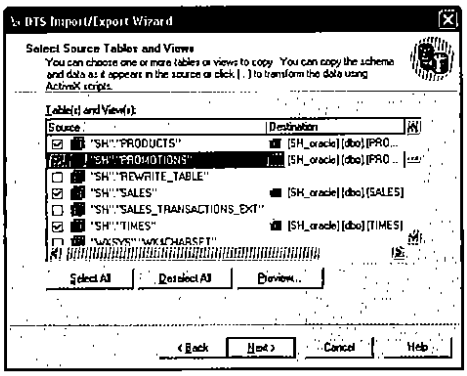


Obr. 3.28 Připojení k databázi Oracle 9i jako zdroji údajů



Obr. 3.29 Nastavení cílové destinace pro přenášené údaje

Z nabízených způsobů přenosu zvolíme možnost překopírování tabulky a pohledy ze zdrojové do cílové databáze. Schéma SH obsahuje 15 tabulek a materializovaných pohledů. Pro naší potřebu přeneseme tabulku SALES, kterou nejdříve použijeme jako tabulku faktů, a tabulky TIMES, PRODUCTS, CUSTOMERS, COUNTRIES, CHANNELS a PROMOTIONS, které později využijeme jako tabulky dimenzí.



Obr. 3.30 Výběr tabulek a pohledů ze zdrojové databáze, které chceme přenést

Na závěr nám průvodce zobrazí souhrn dosud nastavených parametrů.

```
Source: Oracle
Using Oracle Provider for OLE DB

Destination: Microsoft SQL Server
Using Microsoft OLE DB Provider for SQL Server
Location: (local)
Database: SH_oracle

Tables
"SH"."CHANNELS" -> [SH_oracle].[dbo].[CHANNELS]
"SH"."COUNTRIES" -> [SH_oracle].[dbo].[COUNTRIES]
"SH"."CUSTOMERS" -> [SH_oracle].[dbo].[CUSTOMERS]
"SH"."PRODUCTS" -> [SH_oracle].[dbo].[PRODUCTS]
"SH"."PROMOTIONS" -> [SH_oracle].[dbo].[PROMOTIONS]
"SH"."SALES" -> [SH_oracle].[dbo].[SALES]
"SH"."TIMES" -> [SH_oracle].[dbo].[TIMES]
```

Zůstává poslední krok – spuštění přenosu, případně jeho naplánování na později.

Příklad přípravy údajů pro data mining – databáze hub

V páté kapitole se budeme věnovat praktické ukázce data miningu nad příkladem z reálné praxe – databází jedlých a nejedlých hub. Úlohou data miningu bude predikovat na základě vlastností houby, zda je tato houba jedlá, nebo jedovatá. Ale nepředbíhejme, databázi pro data mining potřebujeme nejdříve překonvertovat a připravit. Databázi hub jsme

získali z knihovny katedry univerzity, konkrétně *Department of Information and Computer Science University of California, Irvine CA.*

Co je naším cílem, je víceméně jasné. Potřebujeme jednoduchou tabulku hub, ve které bude pro každou houbu jeden záznam, který bude obsahovat vlastnosti houby včetně toho, zda je jedlá, nebo nejedlá. Takže tabulku přibližně v takovém tvaru (reálná tabulka má až 23 sloupců s jednotlivými vlastnostmi).

id	Pozivatelnost	TvarKlobouku	PovrchKlob	BarvaKlob	Ox.ZmenaBarvy	Zapach
1	nejedlá, jedovatá	vypouklý	hladký	hnědá	ano	štiplavý, čpavý
3	jedlá	zvonovitý	hladký	bílá	ano	anýzový
5	jedlá	vypouklý	hladký	šedá	ne	žádný
7	jedlá	zvonovitý	hladký	bílá	ano	po mandlích
9	nejedlá, jedovatá	vypouklý	šupinatý	bílá	ano	štiplavý, čpavý
11	jedlá	vypouklý	šupinatý	žlutá	ano	anýzový

Zdrojová databáze se skládá ze dvou souborů **agaricus-lepiota.data** a **agaricus-lepiota.names.**

Soubor **agaricus-lepiota.data** je klasickým souborem flat-file, kde jsou jednotlivé atributy odděleny čárkami a jednotlivé záznamy jsou odděleny přechodem na nový řádek (ASCII kódy CR a LF, hexadecimální „OD" a „OA") Celý soubor má 8 124 řádků, což představuje celkový objem údajů přibližně 382 kilobajtů. V našem výpise je jen ilustrační ukázka.

```
p,x,s,n,t,p,f,c,n,k,e,e,s,s,w,w,p,w,o,p,k,s,u
e,x,s,y,t,a,f,c,b,k,e,c,s,s,w,w,p,w,o,p,n,n,g
e,b,s,w,t,l,f,c,b,n,e,c,s,s,w,w,p,w,o,p,n,n,m
p,x,y,w,t,p,f,c,n,n,e,e,s,s,w,w,p,w,o,p,k,s,u
e,x,s,g,f,n,f,w,b,k,t,e,s,s,w,w,p,w,o,e,n,a,g
e,x,y,y,t,a,f,c,b,n,e,c,s,s,w,w,p,w,o,p,k,n,g
e,b,s,w,t,a,f,c,b,g,e,c,s,s,w,w,p,w,o,p,k,n,m
e,b,y,w,t,l,f,c,b,n,e,c,s,s,w,w,p,w,o,p,n,s,m
p,x,y,w,t,p,f,c,n,p,e,e,s,s,w,w,p,w,o,p,k,v,g
e,b,s,y,t,a,f,c,b,g,e,c,s,s,w,w,p,w,o,p,k,s,m
e,x,y,y,t,l,f,c,b,g,e,c,s,s,w,w,p,w,o,p,n,n,g
e,x,y,y,t,a,f,c,b,n,e,c,s,s,w,w,p,w,o,p,k,s,m
e,b,s,y,t,a,f,c,b,w,e,c,s,s,w,w,p,w,o,p,n,s,g
p,x,y,w,t,p,f,c,n,k,e,e,s,s,w,w,p,w,o,p,n,v,u
....
```

Na první pohled z tohoto souboru nevydolujeme ani nevydedukujeme, ba dokonce ani nezjistíme vůbec nic. Legenda k takto uspořádanému flat-file je v souboru **agaricus-lepiota.names.**

```
classes: edible=e, poisonous=p
     1. cap-shape:                bell=b,conical=c,convex=x,flat=f,
                                  knobbed=k,sunken=s
     2. cap-surface:              fibrous=f,grooves=g,scaly=y,smooth=s
     3. cap-color:                brown=n,buff=b,cinnamon=c,gray=g,green=r,
                                  pink=p,purple=u,red=e,white=w,yellow=y
     4. bruises:                  bruises=t,no=f
```

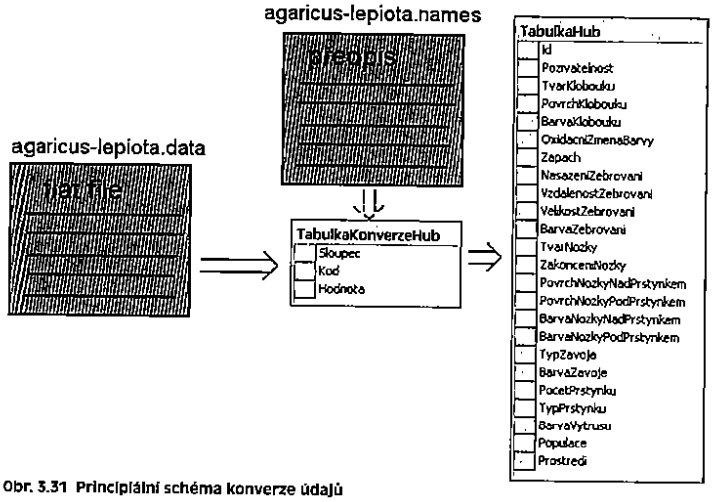
```
 5. odor:                      almond=a,anise=l,creosote=c,fishy=y,foul=f,
                               musty=m,none=n,pungent=p,spicy=s
 6. gill-attachment:           attached=a,descending=d,free=f,notched=n
 7. gill-spacing:              close=c,crowded=w,distant=d
 8. gill-size:                 broad=b,narrow=n
 9. gill-color:                black=k,brown=n,buff=b,chocolate=h,gray=g,
                               green=r,orange=o,pink=p,purple=u,red=e,
                               white=w,yellow=y
10. stalk-shape:               enlarging=e,tapering=t
11. stalk-root:                bulbous=b,club=c,cup=u,equal=e,
                               rhizomorphs=z,rooted=r,missing=?
12. stalk-surface-above-ring:  fibrous=f,scaly=y,silky=k,smooth=s
13. stalk-surface-below-ring:  fibrous=f,scaly=y,silky=k,smooth=s
14. stalk-color-above-ring:    brown=n,buff=b,cinnamon=c,gray=g,orange=o,
                               pink=p,red=e,white=w,yellow=y
15. stalk-color-below-ring:    brown=n,buff=b,cinnamon=c,gray=g,orange=o,
                               pink=p,red=e,white=w,yellow=y
16. veil-type:                 partial=p,universal=u
17. veil-color:                brown=n,orange=o,white=w,yellow=y
18. ring-number:               none=n,one=o,two=t
19. ring-type:                 cobwebby=c,evanescent=e,flaring=f,large=l,
                               none=n,pendant=p,sheathing=s,zone=z
20. spore-print-color:         black=k,brown=n,buff=b,chocolate=h,green=r,
                               orange=o,purple=u,white=w,yellow=y
21. population:                abundant=a,clustered=c,numerous=n,
                               scattered=s,several=v,solitary=y
22. habitat:                   grasses=g,leaves=l,meadows=m,paths=p,
                               urban=u,waste=w,woods=d
```

Kromě požívatelnosti máme tedy k dispozici ještě 22 dalších charakteristických vlastností pro každou houbu.

Když si myslíme, že obsah těchto dvou souborů překonvertujeme pomocí nějakého konverzního wizarda, už asi tušíme, že tudy cesta nevede. Budeme se muset pustit do konverze po etapách. Princip je jasný ze schématu. Potřebujeme překonvertovat a přenést údaje ze souboru typu flat-file do databázové tabulky.

Nejdříve mírně upravíme a do češtiny přeložíme obsah souboru **agaricus-lepiota.names**, Takto dostaneme základní předpis pro dekódování obsahu.

```
Pozivatelnost:          jedlá=e, nejedlá,jedovatá =p
TvarKlobouku:           zvonovitý=b, kuželovitý=c, vypouklý=x, plochý=f,
                        hrbolek uprostřed=k,
                        propadený, vmáčknutý=s
PovrchKlobouku:         vláknitý=f, rýhovaný=g, šupinatý=y, hladký=s
BarvaKlobouku:          hnědá=n, hnědožlutá=b, skořicová=c, hnědá=g, zelená=r,
                        růžová=p, purpurová=u, červená=e, bílá=w, žlutá=y
OxidacniZmenaBarvy:     ano=t, ne=f
Zapach:                 po mandlích=a, anýzový=l, kreozot=c, po rybině=y,
                        hnilobný=f, zatuchlý, plesnivý=m, žádný=n, štiplavý,
                        čpavý=p, kořenitý =s
NasazeniZebrovani:      pevně připojené=a, svažující se=d, volné=f, s vrubem=n
```



Obr. 3.31 Principiální schéma konverze údajů

```
VzdalenostZebrovani:    těsné=c, velmi těsné=w, vzdálené=d
VelikostZebrovani:      široké=b, úzké=n
BarvaZebrovani:         černá=k, hnědá=n, hnědožlutá=b, čokoládová=h, hnědá=g,
                        zelená=r, oranžová=o, růžová=p, purpurová=u, červená=e,
                        bílá=w, žlutá=y
TvarNozky:              rozšiřující se dolů=e, zužující se dolů=t
ZakonceniNozky:         cibulovité=b, hůl, pálka=c, kalich=u, stejný, rovný=e,
                        oddenkovitý=z, kořeny=r, (neurčený)=?
PovrchNozkyNadPrstynkem: vláknitý=f, šupinatý=y, hedvábný=k, hladký=s
PovrchNozkyPodPrstynkem: vláknitý=f, šupinatý=y, hedvábný=k, hladký=s
BarvaNozkyNadPrstynkem: hnědá=n, hnědožlutá=b, skořicová=c, hnědá=g, oranžová=o,
                        růžová=p, červená=e, bílá=w, žlutá=y
BarvaNozkyPodPrstynkem: hnědá=n, hnědožlutá=b, skořicová=c, hnědá=g, oranžová=o,
                        růžová=p, červená=e, bílá=w, žlutá=y
TypZavoje:              částečný=p, úplný=u
BarvaZavoje:            hnědá=n, oranžová=o, bílá=w, žlutá=y
PocetPrstynku:          žádný=n, jeden=o, dva=t
TypPrstynku:            síťovitý, pavučina=c, rozplývající se=e,
                        rozšiřující se=f, velký=l,
                        chybí=n, přivěskovitý=p, hladký, pokrývkovitý=s,
                        pásmovitý, více částí=z
BarvaVytrusu:           černá=k, hnědá=n, hnědožlutá=b, čokoládová=h, zelená=r,
                        oranžová=o, purpurová=u, bílá=w, žlutá=y
Populace:               velmi početná=a, shluky=c, početná=n, roztroušená=s,
                        několik hub=v, osamělá=y
Prostredi:              tráva=g, listí=l, louka=m, cesta=p, město=u, odpadky=w,
                        dřevo=d
```

Tento pro člověka celkem srozumitelný předpis musíme převést do podoby srozumitelné pro stroj, tedy do databázové tabulky nebo souboru XML. V našem případě použijeme konverzní tabulku. Navrhneme tedy jednoduchou databázovou tabulku, která bude obsahovat předpis pro konverzi.

Název	Typ
Sloupec	varchar(50)
Kód	char(1)
Hodnota	varchar(50)

Příkaz pro vytvoření konverzní tabulky potom bude:

```
CREATE TABLE TabulkaKonverzeHub
(
        [Sloupec] varchar(50),
        [Kod] char(1),
        [Hodnota] varchar(50)
)
```

Tabulku naplníme údaji, bude se jednat o celkově 128 záznamů.

```
INSERT INTO TabulkaKonverzeHub (Sloupec, Kod, Hodnota)
       VALUES ('Pozivatelnost', 'e', 'jedlá');
INSERT INTO TabulkaKonverzeHub (Sloupec, Kod, Hodnota)
       VALUES ('Pozivatelnost', 'p', 'nejedlá, jedovatá');
...
...
```

Příkazy pro naplnění tabulky samozřejmě nemusíme generovat ručně, můžeme si na to napsat nějaký šikovný podprogram, například ve Visual Basicu, ale pro 128 záznamů to budeme mít manuálně v nějakém šikovném editoru pomocí kopírování bloků vytvořeno zřejmě rychleji. Tabulka bude obsahovat údaje v tomto tvaru (z každé skupiny hodnot sloupce jsme vypsali jen jeden záznam).

```
Sloupec                  Kód Hodnota
------------------------ --- ------------
BarvaKlobouku            b   hnědožlutá
...
BarvaNozkyNadPrstynkem   b   hnědožlutá
...
BarvaNozkyPodPrstynkem   b   hnědožlutá
...
BarvaVytrusu             b   hnědožlutá
...
BarvaZavoje              n   hnědá
...
BarvaZebrovani           b   hnědožlutá
...
NasazeniZebrovani        a   pevně připojené
...
```

```
OxidacniZmenaBarvy        f   ne
...
PocetPrstynku             n   žádný
..
Populace                  a   velmi početná
...
PovrchKlobouku            f   vláknitý
...
PovrchNozkyNadPrstynkem   f   vláknitý
...
PovrchNozkyPodPrstynkem   f   vláknitý
..
Pozivatelnost             e   jedlá
..
Prostredi                 d   dřevo
...
TvarKlobouku              b   zvonovitý
...
TvarNozky                 e   rozšiřující se dolů
...
TypPrstynku               c   síťovitý, pavučina
...
TypZavoje                 p   částečný
...
VelikostZebrovani         b   široké
...
VzdalenostZebrovani       c   těsné
...
ZakonceniNozky            ?   (neurčený)
...
Zapach                    a   po mandlích
...
```

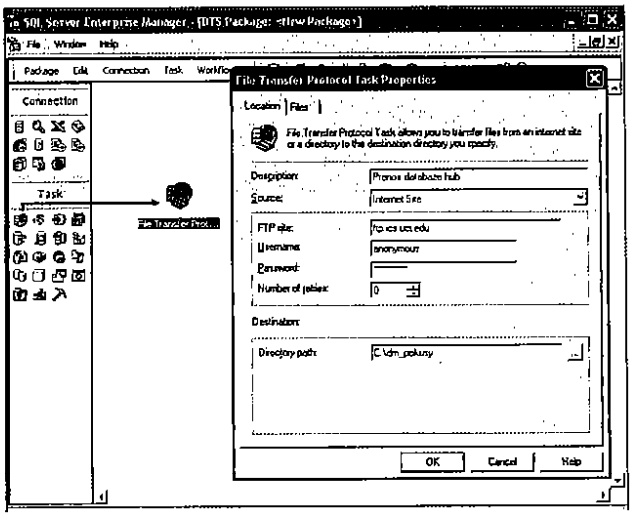
Dalším nevyhnutelným krokem bude vytvoření cílové tabulky s názvem například `TabulkaHub` v cílové databázi. Počet a názvy sloupců této tabulky také vyplývají z obsahu souboru **agaricus-lepiota.names**.

```
CREATE TABLE TabulkaHub
(
        [id] int IDENTITY (1, 1) NOT NULL,
        [Pozivatelnost] nvarchar(50),
        [TvarKlobouku] nvarchar(50),
        [PovrchKlobouku] nvarchar(50),
        [BarvaKlobouku] nvarchar(50),
        [OxidacniZmenaBarvy] nvarchar(50),
        [Zapach] nvarchar(50),
        [NasazeniZebrovani] nvarchar(50),
        [VzdalenostZebrovani] nvarchar(50),
        [VelikostZebrovani] nvarchar(50),
        [BarvaZebrovani] nvarchar(50),
        [TvarNozky] nvarchar(50),
```

```
    [ZakonceniNozky] nvarchar(50),
    [PovrchNozkyNadPrstynkem] nvarchar(50),
    [PovrchNozkyPodPrstynkem] nvarchar(50),
    [BarvaNozkyNadPrstynkem] nvarchar(50),
    [BarvaNozkyPodPrstynkem] nvarchar(50),
    [TypZavoje] nvarchar(50),
    [BarvaZavoje] nvarchar(50),
    [PocetPrstynku] nvarchar(50),
    [TypPrstynku] nvarchar(50),
    [BarvaVytrusu] nvarchar(50),
    [Populace] nvarchar(50),
    [Prostredi] nvarchar(50)
)
```

Pozor. Je důležité navrhnout pořadí sloupců tak, jak jsou uspořádány v souboru flat, abychom potom neměli problémy při transformaci. Dosud jsme se věnovali obsahu souboru **agaricus-lepiota.names,** tedy vlastně textu legendy, která vysvětlovala význam jednotlivých atributů v hlavní databázi, v souboru **agaricus-lepiota.data**

Konverze a úprava tohoto souboru se už po dosud vykonaných přípravných krocích dá velmi lehko zautomatizovat. Začneme tím, že máme někde uložený soubor **agaricus-lepiota.data.** Když se však blíže podíváme na možnosti datové pumpy (DTS Services), můžeme dokonce přeskočit i tento krok a údaje stahovat přímo ze serveru FTP, kde jsou primárně uloženy pomocí úlohy File Transport Protocol Task.

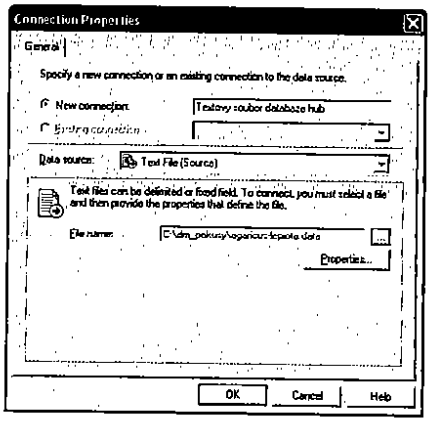


Obr. 3.32 Přenos údajů pomocí úlohy File Transport Protocol Task

Ve složce Files si po připojení k požadovanému serveru v klasickém dvojoknovém dialogu vybereme, které soubory budeme potřebovat přenášet.

Definování zdroje údajů

V našem případě se zdrojová databáze měnit nebude, takže si ji stáhneme jednou a konverze bude probíhat ze souboru na disku lokálního počítače. Jako zdroj údajů tedy zvolíme textový soubor.

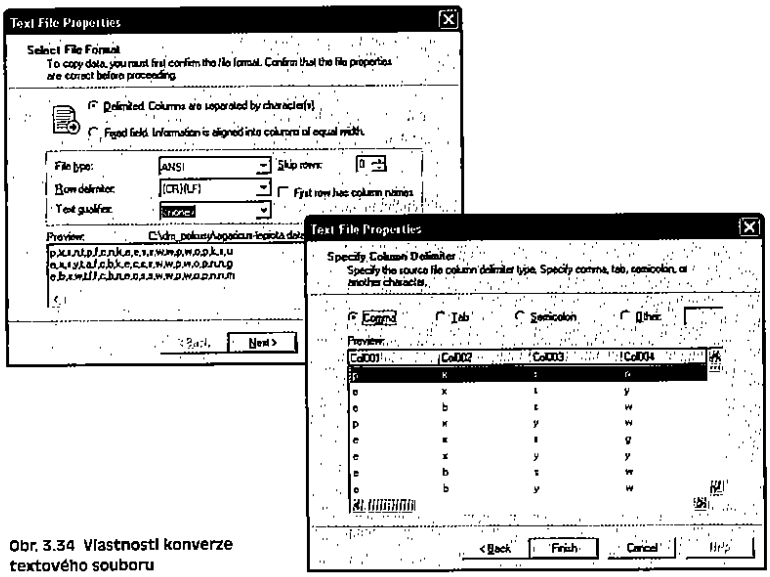


Obr. 3.33 Zdroj údajů – textový soubor

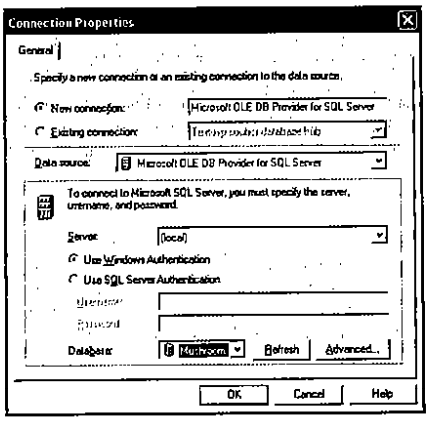
V dialozích aktivovaných tlačítkem Properties definujeme parametry konverze textového souboru. V našem případě jsou jednotlivé záznamy odděleny znaky přechodu na nový řádek (CR LF v kódu ASCII) a jednotlivé sloupce záznamu jsou odděleny čárkami.

Definování cílové destinace

Jako cílovou destinaci pro údaje vybereme databázi, kterou jsme předtím vytvořili, například Mushrooms, nebo můžeme použít i cvičnou databázi PUBS. Důležité je, abychom údaje dostali ze souboru flat do databázové tabulky pod správou MS SQL Serveru. Nejdříve ukážeme jen jednoduchý přenos do pomocné tabulky bez transformace. Když se podíváme na výsledný tvar tabulky po přenosu do databáze (na obrázku vpravo), zjistíme, že nám vcelku vyhovuje.

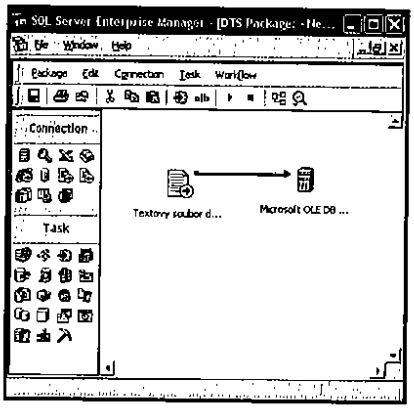


Obr. 3.34 Vlastnosti konverze textového souboru



Obr. 3.35 Definování cílové destinace

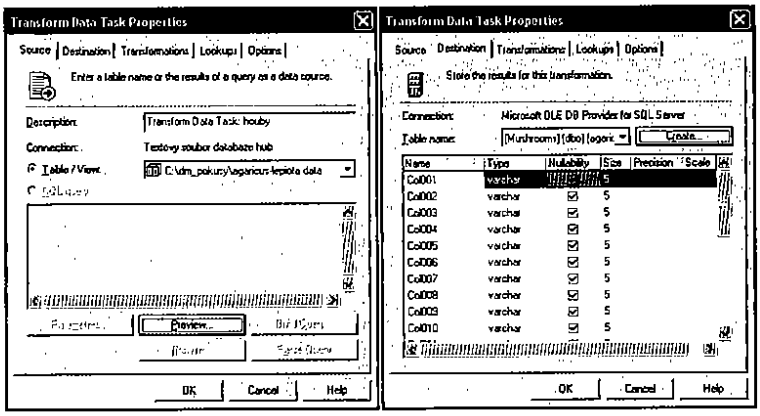
Po přidání transformace dostaneme kostru jednoduchého balíčku pro DTS.



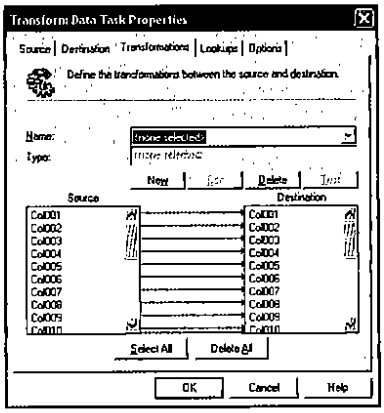
Obr. 3.36 Balíček DTS pro přenos a transformaci údajů

Připomeňme si, že do cílové tabulky `TabulkaHub` chceme dostat už přetransformované údaje v textové podobě. Průvodce nám vygeneroval skript pro vytvoření nové tabulky s názvem `agaricus-lepiota`. Původně navrhl šířku každého sloupce na 255 znaků (`[Col001] varchar (255) NULL,`). Protože víme, že v jednotlivých polích tabulky budou jen jednoznakové údaje, zkrátili jsme šířku sloupce na pět znaků (aby se nám do případného výpisu vešly názvy sloupců).

```
CREATE TABLE [agaricus-lepiota]
(
  [Col001] varchar (5) NULL,
  [Col002] varchar (5) NULL,
  [Col003] varchar (5) NULL,
  ...
  [Col023] varchar (5) NULL
)
```



Obr. 3.37 Definování transformace (záložky Source a Destination)

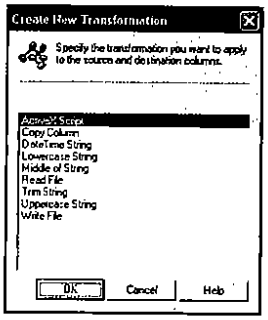


Obr. 3.38 Definování transformace 1:1

Po spuštění a úspěšném ukončení transformace se můžeme o skutečném stavu cílové tabulky přesvědčit jejím výpisem.

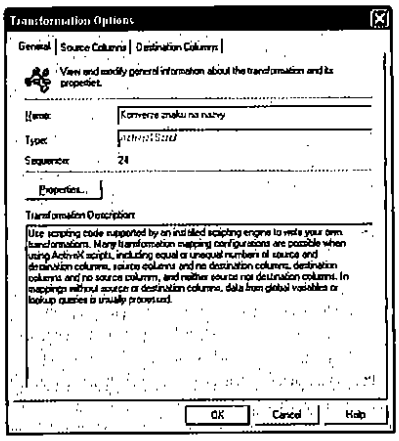
```
SELECT * FROM [agaricus-lepiota];

Col001 Col002 Col003 Col004 Col005 Col006 Col007 Col008 Col009 Col010 ...
------ ------ ------ ------ ------ ------ ------ ------ ------ ------ ------
p      x      s      n      t      p      f      c      n      k      ...
e      x      s      y      t      a      f      c      b      k      ...
e      b      s      w      t      l      f      c      b      n      ...
...
...
(8124 row(s) affected)
```

Naším cílem je však přenos a transformace údajů do srozumitelné textové podoby do tabulky `TabulkaHub`. Proto vytvoříme novou transformaci, která bude plně vyhovovat našim požadavkům. Když nastavíme tabulku `TabulkaHub` jako cílovou destinaci, průvodci vytvořením transformace to vyhovuje, tabulka má správný počet sloupců správného datového typu, dokonce s velkou rezervou 50 znaků pro každý sloupec. Tentokrát ale o transformaci 1:1 zájem nemáme, chceme překonvertovat údaje do textové podoby s využitím převodní tabulky `TabulkaKonverzeHub`. Nejdříve vymažeme všechny průvodcem navržené transformace pro jednotlivé sloupce. Následně tlačítkem Select ALL vybereme všechny sloupce a tlačítkem New vytvoříme novou transformaci jako ActiveX script.



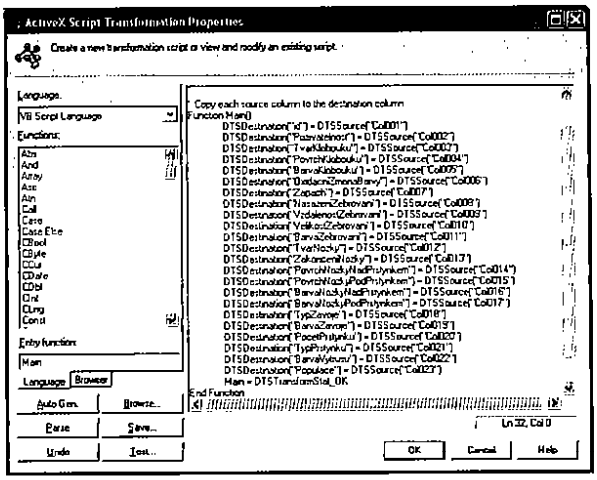
Obr. 3.39 Nová transformace jako script ActiveX

Transformaci nějak pojmenujeme a pustíme se do tvorby skriptů.



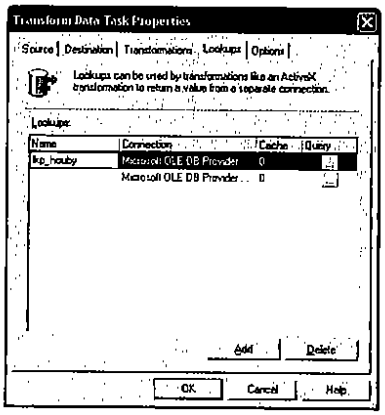
Obr. 3.40 Nastavení vlastností transformace

Znovu opakujeme, že záleží na pořadí sloupců v cílové tabulce. Abychom neměli problémy, navrhli jsme pořadí sloupců tak, jak jsou uspořádány v souboru flat. V dialogu můžeme toto přiřazení ještě naposledy zkontrolovat a upravit.



Obr. 3.41 Nová transformace jako ActiveX script

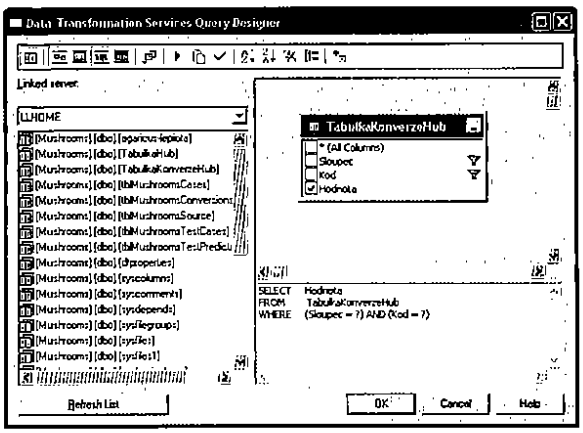
Zde je třeba znovu upozornit na platnost jednoho velmi moudrého Murphyho zákona *„Nikdy není dost času na to, aby se to udělalo, ale vždy je dost času na to, aby se to udělalo znovu."* Mnohokrát jde o důležité ekonomické údaje, případně o lékařské vzorky a podobně. Kdybychom v tomto případě pomíchali údaje v jednotlivých sloupcích a později se na základě výsledků data miningové analýzy vydali na houby, tak by pravděpodobně naše tělesné tekutiny posloužily jako další vzorky, ale tentokrát v nemocniční laboratoři. Ale teď už znovu vážně. Po kontrole přiřazení jednotlivých řádků dialog s ActiveX skriptem zavřeme. Teď je ten správný čas, aby na scénu vstoupila konverzní tabulka `TabulkaKonverzeHub`.



Obr. 3.42 Záložka Lookups

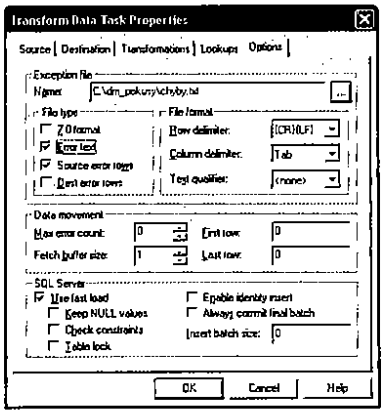
V dialogu DTD Query Designer navrhneme parametrický dotaz SQL pro konverzi údajů s využitím tabulky `TabulkaKonverzeHub`. Parametry jsou zadány jako otazník „?".

```
SELECT     Hodnota
FROM       TabulkaKonverzeHub
WHERE      (Sloupec = ?) AND (Kod = ?)
```



Obr. 3.43 Dialog pro návrh parametrického dotazu SQL pro konverzi údajů

Nakonec ve složce Options definujeme textový soubor, kam budeme ukládat protokol o případné neúspěšné transformaci.



Obr. 3.44 Záložka Options

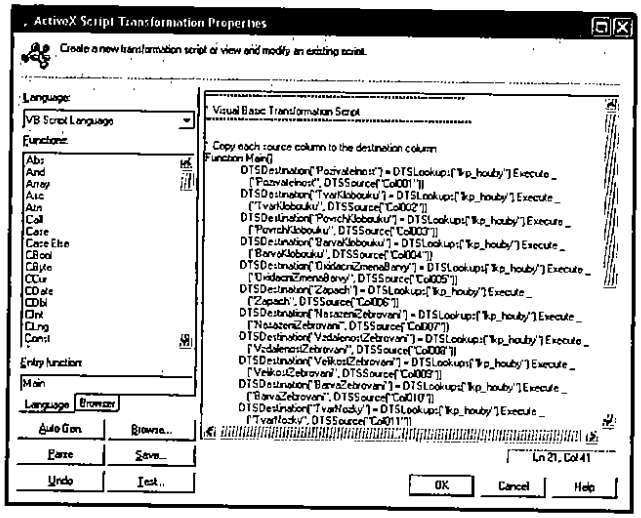
Teď se musíme vrátit do záložky Transformations, konkrétně do dialogu se skriptem ActiveX. Původní řádky kódu převodního skriptu, například

```
DTSDestination("Pozivatelnost") = DTSSource("Col001")
DTSDestination("TvarKlobouku") = DTSSource("Col002")
DTSDestination("PovrchKlobouku") = DTSSource("Col003")
```

musíme upravit tak, abychom mohli využít parametrický dotaz SQL pro konverzi s pomocí konverzní tabulky, tedy:

```
DTSDestination("Pozivatelnost") = DTSLookups("lkp_houby").Execute _
        ("Pozivatelnost", DTSSource("Col001"))
    DTSDestination("TvarKlobouku") = DTSLookups("lkp_houby").Execute _
        ("TvarKlobouku", DTSSource("Col002"))
    DTSDestination("PovrchKlobouku") = DTSLookups("lkp_houby").Execute _
        ("PovrchKlobouku", DTSSource("Col003"))

...
```

Na první pohled to vypadá jako velmi pracné editování, ale s využitím kopírování řádků a hodnot v uvozovkách je to práce tak na pět minut.



Obr. 3.45 Upravený transformační skript

I v tomto případě po spuštění a úspěšném ukončení transformace se o úspěchu přesvědčíme dotazem SQL. Je třeba však dodat, že v tomto případě bude transformace trvat i několik desítek minut, neboť pro každý z 8 124 řádků musí DTS engine položit 23 dotazů SQL do převodní tabulky.

```
SELECT * FROM [TabulkaHub];
```

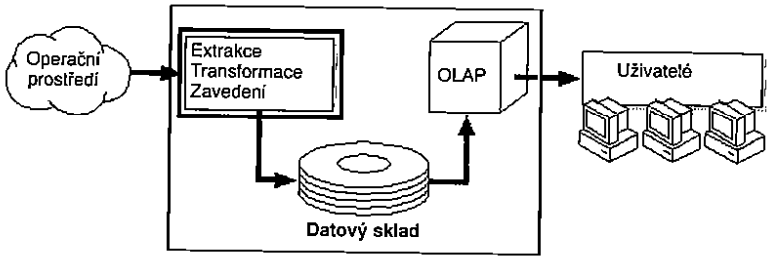
id	Pozivatelnost	TvarKlobouku	PovrchKlob	BarvaKlob	Ox.ZmenaBarvy	Zapach
1	nejedlá, jedovatá	vypouklý	hladký	hnědá	ano	štiplavý, čpavý
3	jedlá	zvonovitý	hladký	bílá	ano	anýzový
5	jedlá	vypouklý	hladký	šedá	ne	žádný
7	jedlá	zvonovitý	hladký	bílá	ano	po mandlích
9	nejedlá, jedovatá	vypouklý	šupinatý	bílá	ano	štiplavý, čpavý
11	jedlá	vypouklý	šupinatý	žlutá	ano	anýzový

```
...
(8124 row(s) affected)
```

Kapitola 4

Analýza OLAP

V předcházejících kapitolách jsme už stručně naznačili základní rozdíly mezi systémy typu **OLTP a OLAP**. Pochopení teoretického rozdílu není pro aplikace typu BI/DW až tak důležité. O mnoho důležitější je správně navrhnou strukturu databáze pro danou oblast použití. Analýza OLAP slouží ke zpracování údajů uložených v datovém skladu do podoby pro koncové uživatele, tedy manažery a analytiky.



Obr. 4.1 Schéma datového skladu

K této kapitole je možné přistupovat dvěma způsoby, buď si nejdříve prostudovat (nebo zopakovat) teoretický úvod do problematiky OLAP, nebo si nejdříve pročíst část o OLAPu pro začátečníky, kde je problematika OLAPu popsaná na základě jednoduchého příkladu domácího rozpočtu, a k podrobnější teorii se vrátit později.

Teoretický úvod do problematiky OLAP

Termín **OLAP** zavedl Dr. E . F. Codd na popsání technologie, která by pomohla překlenou mezery mezi využitím osobních počítačů a řízením podnikových dat. Pro OLAP existuje více různých definic, například:

OLAP je volně definovaný řád principů, které poskytují dimenzionální rámec pro podporu rozhodování.

Pojem OLAP se poměrně často zaměňuje s jiným pojmem **DSS** (Decision Support Systems) – systémy na podporu rozhodování. Tyto systémy umožňují pracovníkům přijímajícím rozhodnutí přístup k údajům potřebným na „tvorbu" takových rozhodnutí.

Kromě jedné z definic pojmu OLAP je poměrně známé i tzv. **„dvanáctero" původních pravidel OLAP od Dr. E . F. Codda.**

Pravidlo první: **Multidimenzionální konceptuální pohled**: OLAP by měl poskytovat uživateli multidimenzionální model odpovídající jeho podnikatelským potřebám tak, aby tento model mohl využívat pro analýzu shromážděných údajů.

Pravidlo druhé: **Transparentnost**: Technologie systému OLAP, podřízená databáze a architektura výpočtů by měly být pro uživatele transparentní, aby mohl naplno využívat svoji produktivitu a odbornost při použití nástrojů front-end a prostředí. Pro platformu Microsoft můžeme jako klientský nástroj použít například program MS Excel z balíku MS Office, který bude napojený na Server OLAP. Důležitá je samozřejmě i heterogennost vstupních dat, kterou zajistíme v procesu ETL.

Pravidlo třetí: **Dostupnost**: Systém OLAP by měl přistupovat jen k těm údajům, které jsou potřebné pro analýzu. Systém by měl být navíc schopen přistupovat ke všem údajům potřebným pro analýzu nezávisle na tom, z kterého heterogenního podnikového zdroje tyto údaje pocházejí, jak často jsou obnovovány a podobně. Znovu zdůrazňujeme význam etapy ETL, ve které se údaje očistí, zahustí a přetransformují.

Pravidlo čtvrté: **Konzistentní vykazování**: I když počet záznamů a tedy i velikost databází postupem času roste, uživatel by neměl pocítit žádné podstatné snížení výkonu.

Pravidlo páté: **Architektura klient-server**: Systém OLAP musí odpovídat principu architektury klient-server s přihlédnutím k maximální ceně a výkonu, flexibilitě a interoperatibilitě.

Pravidlo šesté: **Generická dimenzionalita**: Každá dimenze údajů musí být ekvivalentní ve struktuře i operačních schopnostech.

Pravidlo sedmé: **Dynamické ošetření řídkých matic**: Systém OLAP by měl byt schopen adaptovat své fyzické schéma na konkrétní analytický model, který optimalizuje ošetření řídkých matic, přičemž dosáhne a udrží požadovanou úroveň výkonu.

Pravidlo osmé: **Podpora pro více uživatelů**: Systém OLAP musí být schopen podporovat pracovní skupinu uživatelů pracujících současně na konkrétním modelu.

Pravidlo deváté: **Neomezené křížové dimenzionální operace**: Systém OLAP musí dokázat rozeznat dimenzionální hierarchie a automaticky vykonat asociované kumulované kalkulace v rámci dimenzí i mezi nimi.

Pravidlo desáté: **Intuitivní manipulace s údaji**: Pravidlo definuje konsolidované přeorientování cest na detailní úroveň a zpět (drill down, drill up). Uživatelské rozhraní by mělo umožňovat všechny manipulace způsobem „ukázat a klepnout, případně zachytit a přemístit v buňkách krychle".

Pravidlo jedenácté: **Flexibilní vykazování**: Musí existovat schopnost uspořádat řádky, sloupce a buňky způsobem, který umožní analýzu a intuitivní vizuální prezentaci analytických sestav.

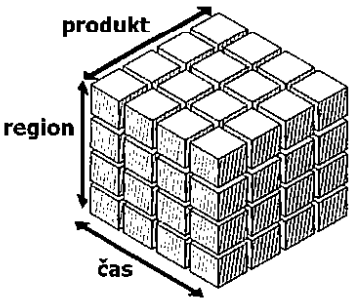
Pravidlo dvanácté: **Neomezené dimenze a úrovně agregace**: V závislosti na požadavcích podnikání může mít analytický model více dimenzí, přičemž každá z nich může mít vícenásobné hierarchie. Systém OLAP by neměl zavádět (například z technických důvodů) žádné umělé omezení počtu dimenzí nebo úrovní agregace.

Fakta a dimenze

Každá krychle OLAP byla vytvořena na základě dvou druhů údajů: faktů a dimenzí.

Fakta jsou numerické měrné jednotky obchodovaní. Tabulka faktů je pochopitelně největší tabulka v databázi a obsahuje velký objem dat. Jak se dozvíme později, tabulky faktů a dimenzí mohou vytvářet určitá schémata, například hvězdicové schéma (star schema) nebo schéma sněhové vločky (snowflake schema). Hvězdicové schéma obyčejně obsahuje jen jednu tabulku faktů, jiné, hlavně schémata DSS, mohou obsahovat i více tabulek faktů. Prvotní fakta, například objem prodeje, se mohou kombinovat nebo vypočítat pomocí jiných faktů a vytvořit tak měrné jednotky. Měrné jednotky se mohou uložit v tabulce faktů, případně vyvolat, když je to nevyhnutelné, na účely vykazování.

Dimenze obsahují logicky nebo organizačně hierarchicky uspořádané údaje. Jsou to vlastně textové popisy obchodování. Tabulky dimenzí jsou obyčejně menší než tabulky faktů a data v nich se nemění tak často. Tabulky dimenzí vysvětlují všechna „proč" a „jak", pokud jde o obchodování a transakce prvků. Dimenze obecně obsahují relativně stabilní data, dimenze zákazníků se aktualizují častěji. Jak už bylo vzpomenuto, velmi často se používají časové, produktové a geografické dimenze.



Obr. 4.2 Příklad dimenzí

Tabulky dimenzí obvykle obsahují stromovou strukturu. Například dimenze vytvořená na základě geografických informací, tedy regionální dimenze se člení na jednotlivé úrovně podle konkrétního územně-správního členění dané geografické oblasti, například:

```
Region
  • Kontinent
  • • Země
  • • • Územní celek
  • • • • Město
```

Počet teček znamená úroveň vnoření jednotlivých atributů dimenze.

Také se používá i produktová klasifikace. Produkty se začleňují do druhů, kategorií, skupin a podobně, například:

```
Produkt
  • Druh produktu
  • • Kategorie
  • • • Subkategorie
  • • • • Název produktu
```

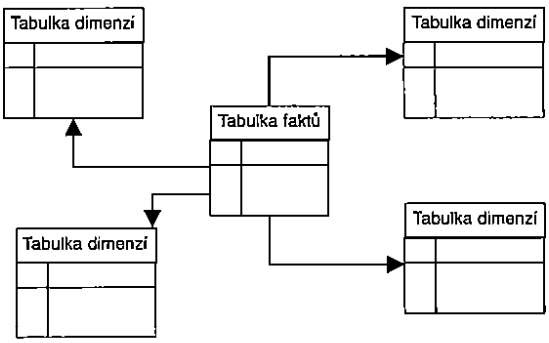
Skoro vždy se používá jako dimenze čas, kde jsou jednotlivé úrovně definovány podle kalendářních nebo fiskálních zvyklostí, například:

```
Čas
  • Rok
  • • Kvartál
  • • • Měsíc
  • • • • Týden
```

Schémata tabulek dimenzí

Krychli vytváříme na základě dimenzionálního modelu, který má určité topologické uspořádání, kterému říkáme schéma. Nejčastěji používáme **hvězdicové schéma** (star schema), nebo **schéma „sněhové vločky"** (snowflake schema).

Hvězdicové schéma se skládá z tabulky faktů obsahující cizí klíče, které se vztahují k primárním klíčům v tabulkách dimenzí. Hvězdicové schéma nemá normalizované dimenze ani relační propojení mezi tabulkami dimenzí, proto je velmi lehce pochopitelné, ale v důsledku nenormalizovaných dimenzí je vytvoření takového modelu relativně pomalé, ale na druhé straně tento model poskytuje vysoký „dotazovací výkon". (*Poznámka: jak jsme už uvedli, normalizace v tomto případě znamená, že tabulka vyhovuje zařazení do některé z tzv. normálních forem.*)



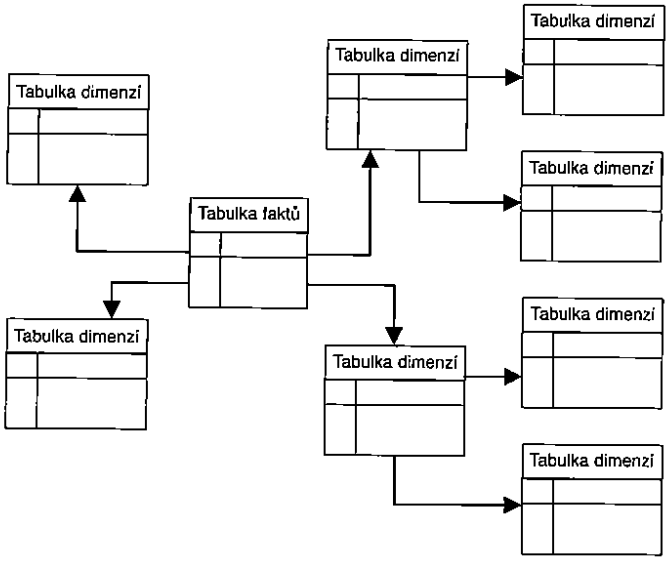
Obr. 4.3 Hvězdicové schéma (star schema)

Jako příklad takové nenormalizované tabulky dimenzí ukážeme několik záznamů z tabulky **time_by_day** z databáze FoodMart.

time_id	the_date	the_day	the_month	the_year	day_of_mon	week_of_y	month_of_y	quarter
367	1997-01-01	Wednesday	January	1997	1	2.0	1	Q1
368	1997-01-02	Thursday	January	1997	2	2.0	1	Q1
369	1997-01-03	Friday	January	1997	3	2.0	1	Q1
370	1997-01-04	Saturday	January	1997	4	2.0	1	Q1
371	1997-01-05	Sunday	January	1997	5	3.0	1	Q1
372	1997-01-06	Monday	January	1997	6	3.0	1	Q1

Vidíme, že o nějaké normalizaci nemůže být ani řeč o úspoře místa ani nemluvě. Jen těžko si můžeme představit větší nadbytečnost. Sestavit takovou tabulku vyžaduje i určité úsilí, ale pro každý jeden den okamžitě víme jeho pořadí v týdnu (vlastně název dne), v měsíci, v kvartálu, v roce a podobně. Teď už chápeme příčinu vysokého dotazovacího výkonu, protože všechny údaje získáme najednou a nemusíme je skládat z relačních tabulek.

Schéma „sněhové vločky" obsahuje některé dimenze složené z mnoha relačně svázaných tabulek. Tento model umožňuje rychlejší zavedení údajů do normalizovaných tabulek, ale má podstatně nižší dotazovací výkon, neboť obsahuje větší množství spojení tabulek.



Obr. 4.4 Schéma sněhové vločky (snowflake schema)

Úložiště multidimenzionálních údajů MOLAP, ROLAP, HOLAP

Pojednání o jednotlivých druzích multidimenzionálních modelů databází začneme zdůrazněním rozdílů mezi relačními a multidimenzionálními (OLAP) databázovými modely.

Relační databázový model. Údaje jsou uloženy v dvoudimenzionálních tabulkách. Každý řádek v tabulce obsahuje data, která jsou zpravidla obrazem reálného světa, tedy data, která se vztahují k nějaké věci nebo k její části. Sloupce dvoudimenzionálních databázových tabulek obsahují údaje týkající se atributů.

Multidimenzionální databázový model. Datový model multidimenzionální databáze je možné zobrazit jako vícerozměrnou krychli. Tato krychle je vlastně ekvivalent tabulky v relační databázi. Každá krychle má několik dimenzí (ekvivalent indexových polí v relačních tabulkách). Nejlépe si dokážeme představit klasickou trojrozměrnou krychli, ale počet dimenzí v reálných multidimenzionálních databázích je zpravidla větší. Znalci programování ve vyšších programovacích jazycích určitě namítnou, že multidimenzionální krychle není nic jiného než vícerozměrné pole. Prostor pro celou krychli je už dopředu rozvržen. Jednotlivé záznamy se v multidimenzionálních krychlích nacházejí na průsečících dimenzí. Například objem prodeje chladniček (produktová dimenze) za první kvartál roku 2001 (časová dimenze) v Severomoravském kraji (geografická nebo zákaznická dimenze).

S rostoucím počtem rozměrů multidimenzionální databáze velmi rychle rostou i požadavky na úložnou kapacitu. Ale ne na všech průsečících dimenzí se vždy nacházejí údaje. Takovou krychli nazýváme i řídkou krychlí. V praxi se u multidimenzionálních databází používají různé technologie na kompresi objemu použitého diskového prostoru.

Multidimenzionální OLAP (MOLAP)

Pro multidimenzionální online analytické zpracování (MOLAP) se získávají data buď z datového skladu, nebo z operačních zdrojů. Mechanismus MOLAP potom uloží analytická data ve vlastních datových strukturách a sumářích. Během tohoto procesu se spočítá tolik předběžných výsledků, kolik je z technického a časového hlediska možné. Údaje v úložišti typu MOLAP se tedy budou ukládat jako dopředu vypočítaná pole. Hodnoty dat i indexů se uchovávají v jednotlivých polích multidimenzionální databáze. Databáze je organizována tak, aby umožnila rychlé získávání příslušných údajů z více dimenzí. Část údajů se může zavést ze serveru ke klientovi, což umožňuje rychlé analýzy bez velkého zatížení sítě.

Hlavní výhodou MOLAP je maximální výkon vzhledem k dotazům uživatelů, nevýhodou je redundance údajů, neboť tyto údaje jsou uloženy jednak v relační databázi, jednak v multidimenzionální databázi. Požadavky na úložnou kapacitu mohou v případě použití více dimenzí extrémně narůstat.

Relační databázový OLAP (ROLAP)

Relační online analytické zpracování údajů (ROLAP) získává údaje pro analýzy z relačního datového skladu. Tyto údaje z relačních databází se po zpracování předkládají uživateli jako multidimenzionální pohled. Data a metadata se v úložišti ROLAP ukládají jako záznamy v relační databázi. Server OLAP dynamicky používá tato metadata na generování příkazů SQL, které jsou potřebné na získávání dat požadovaných uživateli. U tohoto způsobu zůstávají data uložena v relačních databázích, takže nevzniká problém s redundancí.

Hybridní OLAP (HOLAP)

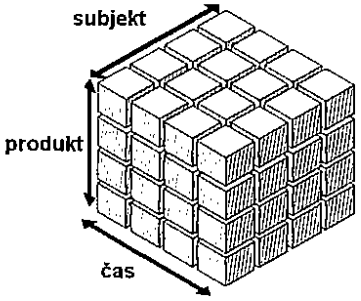
Hybridní OLAP je kombinací úložišť MOLAP a ROLAP, přičemž se využívají výhody jednotlivých typů úložišť a do značné míry se eliminují nevýhody. Údaje zůstávají v relačních databázích a spočítané agregace se ukládají do multidimenzionálních struktur. Při dotazování se údaje vybírají do multidimenzionální paměti cache. U hybridního řešení relační databáze ukládá množství detailních dat a multidimenzionální model ukládá sumární data.

OLAP pro začátečníky

Po krátkém teoretickém úvodu se tyto pojmy dají lehce pochopit a na rozdíl od mnoha jiných témat i lehce představit. Budeme tedy pokračovat krátkým úvodem pro začátečníky, kde bude vše podřízené srozumitelnosti.

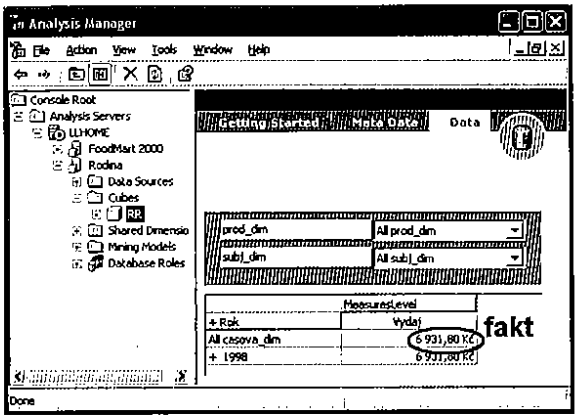
Jak uvidíme na konci této kapitoly pro začátečníky, popisujeme problematiku OLAP vlastně „reverzně", tedy od konce do začátku. Máme pro to dobré důvody. Když chceme ukázat princip něčeho, je dobré, když to něco, v našem případě krychli OLAP už máme k dispozici. Na základě jakých údajů a jak byla krychle OLAP vytvořena se tedy dozvíme na konci kapitoly.

Námětem pro naši jednoduchou krychli OLAP bude téma každému blízké a srozumitelné – rodinný rozpočet. Naší rodinu tvoří čtyři členové. Rodiče Jan a Marta a dvě děti Mirek a Zuzka. A celá rodina podobně jako ostatní rodiny vesele utrácí finance ze svého rodinného rozpočtu za různé zboží a služby. Když jsme si už krychli dopředu vytvořili, můžeme začít oním klasickým: „Máme krychli představující výdej rodinného rozpočtu, která má tři dimenze, čas, produkt a subjekt."



Obr. 4.5 OLAP krychle rodinného rozpočtu

Už po této první větě můžeme očekávat otázky typu „Kde máme krychli?" „Jak s ní budeme pracovat?" a podobně. Nuže krychle se skrývá někde na disku, kde a jak je uložena, ví jen speciální program, který běží na pozadí na našem počítači, který se nazývá Server OLAP. Tento server se nainstaluje například při instalaci analytických služeb MS SQL Serveru 2000. A krychli můžeme prohlížet pomocí různých klientských aplikací, například aplikace **Analysis Manager** (také součást instalace analytických služeb), nebo dokonce i tabulkového programu Microsoft Excel, který je součástí kancelářského balíku Microsoft Office. V naší ukázce budeme používat Analysis Manager, takže krychli uvidíme v záložce Data.

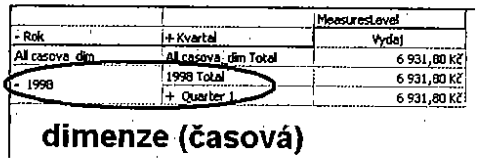


Obr. 4.6 Krychle OLAP zobrazená pomocí aplikace Analysis Manager

Tušíme další námitku. „V záložce Data není žádná krychle, jen nějaká tabulka." A to se už dostáváme k problematice zobrazování krychlí a práce s nimi. Na obrázku je totiž skutečně zobrazena krychle OLAP ve formě tabulky. Zatím tam vidíme jen jeden údaj (vypsaný dvakrát pod sebou, ale k tomu se ještě dostaneme). Není to ona pověstná hodnota 42, kterou mnozí znají ze známé sci-fi publikace *Stopařův průvodce po galaxii* (pro ty, kteří to nečetli, tuto hodnotu vrátil superpočítač po výpočtech, které trvaly 7,5 milionů roků jako odpověď na otázku o významu vesmíru a života), je tam suma 6 931,80 Kč. Tady nám aplikační logika trochu pomohla v tom, že za číslem je zobrazen symbol měnové jednotky. Kdyby tam nebyl, mohlo by číslo 6 931,80 znamenat cokoliv. Například vzdálenost v mílích do USA, počet kubických stop batožinového prostoru, měsíční výdaje průměrné rodiny v curech, prostě cokoliv. V našem případě víme, že tato suma je součtem výdajů za sledované období. Je sice nereálná, neboť vznikla na základě tabulky, která obsahovala jen několik cvičných údajů, ale pro naše účely postačí stejně dobře jako jakékoliv jiné číslo.

A tady se dostáváme k prvnímu pojmu z oblasti OLAP, k faktům. Suma 6 931,80 Kč je totiž **fakt**, který v našem případě udává, kolik bylo utraceno prostředků z rodinného rozpočtu. Místo pojmu fakt se někdy používá i pojem měrná jednotka obchodování (v našem případě spíše utrácení). Fakt je tedy číselná veličina, která je předmětem našeho zájmu. Fakty může být počet prodaných kusů, obrat, náklady, zisk a podobně. Ale vraťme se k neurčitosti našeho faktu, k sumě 6 931,80 Kč. Když se pozorněji podíváme na tabulku v okně aplikace, tak zjistíme, že se jedná o výdaj a že se tato suma nějakým způsobem vztahuje k roku 1998. Nevíme ani přesně kdy, ani kým a ani za co byly tyto peníze utraceny.

Údaj 1998 nám aspoň přibližně charakterizuje, kdy byla vzpomínaná suma utracena. Je to časová **dimenze**. Když klepneme kurzorem myši na znamínko + před údajem 1998, tak zjistíme, že se jedná o údaje za první kvartál. Z úrovně roků jsme se dostali na nižší úroveň časové dimenze, na kvartály.



- Rok	+ Kvartal	MeasuresLevel Vydaj
All casova_dim	All casova_dim Total	6 931,80 Kč
- 1998	1998 Total	6 931,80 Kč
	+ Quarter 1	6 931,80 Kč

dimenze (časová)

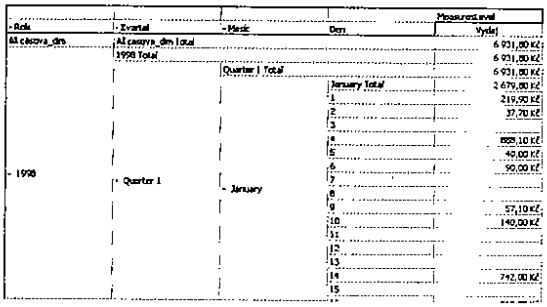
Obr. 4.7 Časová dimenze na úrovni kvartálů

Stále ale nevíme, zda tato suma byla minuta prvního ledna, nebo naopak po 76 korunách každý den. Ale i při označení Quarter 1 je znaménko plus, což znamená, že i tato úroveň je ještě dělitelná, zřejmě na měsíce a potom na dny. Zkusme tedy dále rozvinout časovou dimenzi.

- Rok	+ Kvartal	+ Mesic	MeasuresLevel Vydaj
All casova_dim	All casova_dim Total		6 931,80 Kč
	1998 Total		6 931,80 Kč
- 1998	- Quarter 1	Quarter 1 Total	6 931,80 Kč
		+ January	2 679,80 Kč
		+ February	2 026,80 Kč
		+ March	2 225,20 Kč

Obr. 4.8 Časová dimenze na úrovni měsíců

Na této úrovni jsme zjistili sumy výdajů za každý měsíc a souhrn za první kvartál a celý rok 1998. Když v tomto roce údaje za jiné kvartály nemáme, tak je souhrn za kvartál a rok logicky stejný.

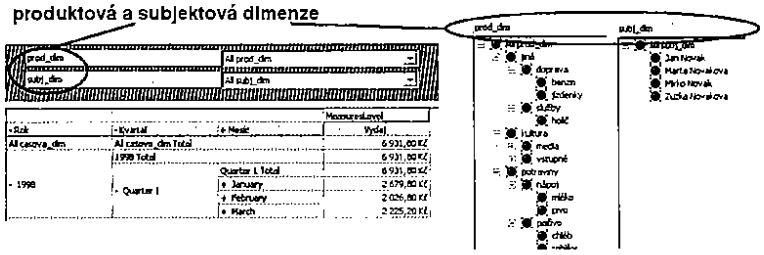


Obr. 4.9 Časová dimenze na úrovni dní

Teď už přesně víme, kolik peněz bylo utraceno za který den, a vidíme i hierarchickou strukturu časové dimenze.

Časová dimenze
- Rok
  - Kvartál
    - Měsíc
      - Den

Ale zatím nevíme, za co a ani kým byly tyto finanční prostředky utraceny. Vraťme se na úroveň měsíců a všimněme si na obrázku dalších dvou dimenzí: produktové a subjektové. Když je rozvineme (na obrázku vpravo) uvidíme i jejich hierarchické úrovně.



Obr. 4.10 Produktová a subjektová dimenze (v pravé části jsou obě dvě dimenze rozvinuty)

Produktová dimenze nám udává, za jaké druhy produktů jsme utráceli. Dimenze má tři všeobecné hierarchické úrovně: druh, skupina a produkt.

Produktová dimenze
- Druh
- • Skupina
- • • Produkt

Podle toho, jaké údaje jsme vkládali do tabulky, ze které byla produktová dimenze vytvořena, je konkrétní hierarchie produktové dimenze například následující:

Produktová dimenze
- Jiné
- • Doprava
- • • Benzín
- • • Jízdenky
- • Služby
- • • Holič
- Kultura
- • Vstupné
- • • Kino
- • • Divadlo
- • Media
- • • Kazety VHS
- Potraviny
- • Nápoje
- • • Pivo
- • • Mléko
- • Pečivo
- • • Chléb.
- • • Rohlíky

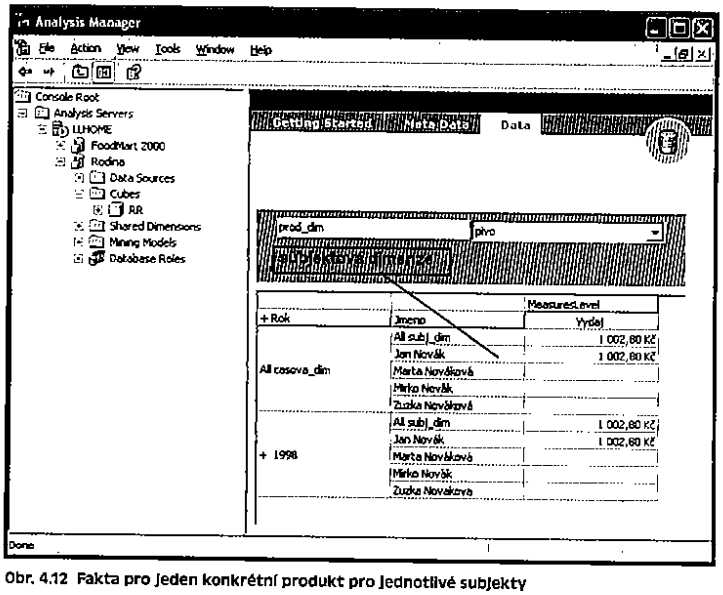
Když nás bude zajímat, kolik se utratilo například za pivo, stačí, když nastavíme úroveň omezení produktové dimenze na hodnotu pivo.

subj_dim	All subj_dim
prod_dim	pivo

			MeasuresLevel
- Rok	- Kvartal	+ Mesic	Vydaj
All casova_dim	All casova_dim Total		1 002,80 Kč
- 1998	1998 Total		1 002,80 Kč
	- Quarter 1	Quarter 1 Total	1 002,80 Kč
		+ January	302,30 Kč
		+ February	201,30 Kč
		+ March	499,20 Kč

Obr. 4.11 Fakta omezená jen pro jeden konkrétní produkt

Z takového přehledu víme, kolik piva bylo nakoupeno v jednotlivých měsících, i když zatím nevíme, kdo za tím stojí (samozřejmě, že to tušíme). A co můžeme z takto prezentovaných faktů vydedukovat? Třeba podle spotřeby piva například to, že zřejmě byl měsíc březen teplejší než předcházející. Přece jen nás zajímá, kdo konkrétně se za tou spotřebou piva skrývá. K tomu nám poslouží subjektová dimenze.



Obr. 4.12 Fakta pro jeden konkrétní produkt pro jednotlivé subjekty

Subjektová dimenze má v našem případě jen jednu hierarchickou úroveň, kde jsou jména jednotlivých členů rodiny.

Subjektová dimenze
- Jan Novák
- Marta Nováková
- Mirek Novák
- Zuzka Nováková

Když pomocí metody drag and drop přesuneme ikonu subjektové dimenze do levé části tabulky, tak můžeme zjistit výdaje za sledovaný produkt (pivo) pro jednotlivé subjekty. Časovou dimenzi jsme stáhli na úroveň roků, aby tabulka nebyla příliš velká. Teď stačí jen letmý pohled na tabulku a hned nám je jasné, jak je to s pivem v naší domácnosti.

Databázové tabulky pro cvičný příklad

V předcházející stati jsme představili jednoduchou krychli včetně popisu faktů a dimenzí. Věříme, že jsme poskytli odpovědi na mnohé otázky hlavně začátečníkům v této oblasti. Až na jednu, jak a na základě čeho to celé vzniklo. V této stati proto ukážeme vytvoření a naplnění databází a krychle, kterou jsme použili v úvodu pro začátečníky.

Relační databáze pro cvičný příklad

Běžná databáze pro evidování stavu a pohybů rodinného rozpočtu by mohla být navržena tak, že by se skládala ze dvou jednoduchých tabulek. V první z nich s názvem například **clenove_rodiny** budou informace o členech rodiny. Tabulka by mohla být navržena přibližně takto:

Název sloupce	Datový typ
kdo	int primary key
jmeno	Varchar(30)
limit	decimal(9,2)
cerpal	decimal(9,2)

Tabulka obsahuje právě čtyři záznamy, když se jedná o čtyřčlennou rodinu. Při pohledu do této tabulky jsou hned zřejmá některá fakta a souvislosti.

kdo	jmeno	limit	cerpal
1	Jan Novák	50000.00	.00
2	Marta Nováková	90000.00	.00
3	Mirek Novák	5000.00	.00
4	Zuzka Nováková	5000.00	.00

Hlavou rodiny je Jan Novák, nebo alespoň deklarovanou hlavou rodiny dle identifikátoru kdo. Při pohledu na roční limity spotřeby financí vidíme, že skutečnou hlavou rodiny je jeho manželka Marta. Mirek a Zuzka jsou jejich děti. Tato tabulka je relativně statická, neboť každý přírůstek do ní by byl v této rodině nepochybně velkou událostí. O mnoho častěji se při dnešním konzumním způsobu života budou přidávat nové údaje do druhé tabulky s názvem **rodinny_rozpocet**. Tabulka může být navržena například takto:

Název sloupce	Datový typ
datum	datetime
kdo	int
polozka	varchar(20)
prijem	decimal(9,2)
vydaj	decimal(9,2)

Tabulka obsahuje sloupce pro datum, název položky, identifikaci člena rodiny, jehož se daná položka týká (jinými slovy, kdo vydělal, případně spotřeboval část rodinných financí), a samozřejmě sloupce pro příjem a výdaje. Její obsah, až na to, že neobsahuje jméno, ale jen odkaz do tabulky členů rodiny, připomíná jednoduchý účetní sešit.

datum	kdo	polozka	prijem	vydaj
...	..	..	..	..
04.03.2002	2	Kabelka	.00	1700.00
09.03.2002	3	Autíčko	.00	320.00
12.03.2002	4	Panenka	.00	550.00
22.03.2002	1	Boty	.00	2000.00
...	..	..	..	..

Když se na tuto tabulku podíváme z hlediska jejího možného použití pro analýzu OLAP, v tabulce členů rodiny jsou údaje pro vytvoření subjektové dimenze, v tabulce rodinný rozpočet jsou údaje pro vytvoření časové dimenze (sloupec datum) a produktové dimenze (sloupec položka). A tabulka rodinný rozpočet obsahuje i měrné jednotky obchodování, tedy fakta. Kdybychom ale chtěli vytvořenou krychli použít pro rodinné analýzy, tak by bylo potřebné časovou a produktovou dimenzi „rozměnit na drobné".

Multidimenzionální databáze pro cvičný příklad

Kdybychom chtěli dvě tabulky databáze rodinného rozpočtu použít jako podklad pro vytvoření krychle OLAP, bude výhodnější relační strukturu převést na multidimenzionální. V našem případě by tato transformace byla velmi jednoduchá, ale protože jde jen o cvičné tabulky, které neobsahují reálné údaje, tak vytvoříme a naplníme multidimenziální databázi znovu, tedy jako se lidově říká „na zelené louce". Databáze bude obsahovat tabulky pro vytvoření časové, produktové a subjektové dimenze a tabulku faktů. Protože tato kapitola je určena hlavně pro začátečníky, ukážeme i příkazy jazyka SQL pro vytvoření a naplnění jednotlivých tabulek.

Tabulka pro časovou dimenzi

Tabulku pro vytvoření časové dimenze `datumova_tab` jsme úmyslně navrhli tak, abychom do ní později mohli zkopírovat údaje z už vyplněné tabulky `time_by_day` z cvičné databáze FoodMart.

```
CREATE TABLE datumova_tab
(
        dat_id int NOT NULL,
        datum smalldatetime NULL,
        den smallint NULL,
        mesic smallint NULL,
        kvartal nvarchar(2),
        rok smallint NULL
)
```

Samozřejmě bychom mohli „rozměnit na drobné" sloupec DATUM z tabulky rodinny_rozpocet. Pomocí vhodného příkazu v jazyce SQL to není žádný technický problém.

```
SELECT DAY(datum) AS den,
       MONTH(datum) AS mesic,
       DATEPART(qq, datum) AS kvartal,
       YEAR(datum) AS rok
  FROM rodinny_rozpocet
```

```
den         mesic       kvartal     rok
----------- ----------- ----------- -----------
2           3           1           2002
4           3           1           2002
9           3           1           2002
12          3           1           2002
22          3           1           2002
28          3           1           2002
```

Problém je v něčem jiném. Když vygenerujeme tabulku, která má posloužit jako podklad pro vytvoření časové dimenze, na základě jedné tabulky, tak se může stát, že tato tabulka bude lidově řečeno „děravá". V naší tabulce jsou jen některé dny. Proto je výhodnější generovat tabulku pro časovou dimenzi uměle nebo překopírovat údaje z jiné vhodné časové tabulky. Čas totiž plyne stejně. Stejně pro naši fiktivní rodinu a stejně pro fiktivní firmu FoodMart. Takže z ní můžeme překopírovat vhodné sloupce za vhodný časový úsek. Z tabulky `time_by_day` do tabulky `datumova_tab` potom přeneseme údaje například za první tři měsíce roku 1998 příkazem

```
INSERT INTO datumova_tab (dat_id, datum, den, mesic, kvartal, rok)
    SELECT time_id - 731, the_date, day_of_month, month_of_year, quarter, the_year
      FROM time_by_day WHERE the_year = 1998 AND month_of_year < 4;
```

Tabulka bude obsahovat 90 řádků, jeden řádek pro každý den prvního čtvrtletí 1998. 1. leden 1998 má ale v původní tabulce `time_by_day` číslo 732. Abychom si zjednodušili situaci a 1. 1. 1998 nám začínal od jednotky, odečteme od hodnoty sloupce `dat_id` hodnotu 731.

```
UPDATE datumova_tab SET dat_id = dat_id-731;
```

Potom bude tabulka `datumova_tab` obsahovat údaje:

```
dat_id      datum       den    mesic  kvartal rok
----------- ----------- ------ ------ ------- ------
1           1998-01-01  1      1      Q1      1998
2           1998-01-02  2      1      Q1      1998
3           1998-01-03  3      1      Q1      1998
4           1998-01-04  4      1      Q1      1998
5           1998-01-05  5      1      Q1      1998
6           1998-01-06  6      1      Q1      1998
7           1998-01-07  7      1      Q1      1998
8           1998-01-08  8      1      Q1      1998
9           1998-01-09  9      1      Q1      1998
10          1998-01-10  10     1      Q1      1998
...
...
(90 row(s) affected)
```

Tabulka pro subjektovou dimenzi

Pod pojmem subjekt budeme v tomto případě myslet člena rodiny. Ke skriptům pro vytvoření a naplnění tabulky není třeba žádné vysvětlování. Dva rodiče Jan a Marta a dvě děti Mirek a Zuzka.

```
CREATE TABLE clenove_rodiny
(
   sub_id int NOT NULL,
   jmeno varchar(20)
);

INSERT INTO clenove_rodiny VALUES (1, 'Jan Novák');
INSERT INTO clenove_rodiny VALUES (2, 'Marta Nováková');
INSERT INTO clenove_rodiny VALUES (3, 'Mirek Novák');
INSERT INTO clenove_rodiny VALUES (4, 'Zuzka Nováková');
```

Tabulka pro produktovou dimenzi

Jako podklad pro vytvoření produktové dimenze, přičemž pod pojmem produkt v domácím rozpočtu budeme uvažovat předmět, za který některý člen rodiny utratil peníze. Pro jednoduchost nebudeme uvažovat o nákupech pro jiné členy rodiny (typický případ, kdy děti koupí otci ponožky k narozeninám nebo matka koupí zákusky a děti se už postarají, aby se nezkazily).

```
CREATE TABLE produkty
(
        prod_id int NOT NULL ,
        druh varchar(15),
        skupina varchar(15),
        produkt varchar(15)
)
```

Tabulku samozřejmě naplníme několika cvičnými údaji

```
INSERT INTO produkty VALUES (1, 'potraviny', 'nápoj', 'mléko');
INSERT INTO produkty VALUES (2, 'potraviny', 'nápoj', 'pivo');
INSERT INTO produkty VALUES (3, 'potraviny', 'pečivo', 'chléb');
INSERT INTO produkty VALUES (4, 'potraviny', 'pečivo', 'rohlíky');
INSERT INTO produkty VALUES (5, 'jiné', 'doprava', 'benzín');
INSERT INTO produkty VALUES (6, 'jiné', 'doprava', 'jízdenky');
INSERT INTO produkty VALUES (7, 'jiné', 'služby', 'holič');
INSERT INTO produkty VALUES (8, 'kultura', 'vstupné', 'kino');
INSERT INTO produkty VALUES (9, 'kultura', 'vstupné', 'divadlo');
INSERT INTO produkty VALUES (10, 'kultura', 'media', 'kazety VHS');
```

Tabulka faktů – výdaje rodinného rozpočtu

Fakta jsou v našem případě rodinného rozpočtu vlastně rodinné výdaje. V tabulce faktů jsou samozřejmě klíče do tabulek dimenzí.

```
CREATE TABLE rozpocet
(
        sub_id int NOT NULL,
        prod_id int NOT NULL,
        dat_id int NOT NULL,
        vydaj money
)
```

I tuto tabulku musíme naplnit vzorky údajů za tři měsíce. Připomínáme, že ve sloupci `dat_id` je pořadové číslo dne, tedy za první čtvrtletí jsou to hodnoty od 1 do 90.

– první měsíc

```
INSERT INTO rozpocet VALUES (1, 2, 1, 155.20);
INSERT INTO rozpocet VALUES (1, 2, 6, 90);
INSERT INTO rozpocet VALUES (1, 2, 9, 57.10);
INSERT INTO rozpocet VALUES (1, 5, 4, 888.10);
INSERT INTO rozpocet VALUES (1, 5, 14, 742);
INSERT INTO rozpocet VALUES (2, 1, 1, 64.70);
INSERT INTO rozpocet VALUES (2, 1, 2, 37.70);
INSERT INTO rozpocet VALUES (3, 6, 5, 40);
INSERT INTO rozpocet VALUES (3, 8, 10, 140);
INSERT INTO rozpocet VALUES (4, 7, 16, 210);
INSERT INTO rozpocet VALUES (4, 9, 18, 100);
```

– druhý měsíc

```
INSERT INTO rozpocet VALUES (1, 2, 41, 78.20);
INSERT INTO rozpocet VALUES (1, 2, 56, 55);
INSERT INTO rozpocet VALUES (1, 2, 59, 68.10);
INSERT INTO rozpocet VALUES (1, 5, 34, 324.10);
INSERT INTO rozpocet VALUES (1, 5, 34, 698);
INSERT INTO rozpocet VALUES (2, 1, 41, 99.70);
INSERT INTO rozpocet VALUES (2, 1, 52, 77.70);
```

```
INSERT INTO rozpocet VALUES (3, 6, 45, 66);
INSERT INTO rozpocet VALUES (3, 8, 30, 155);
INSERT INTO rozpocet VALUES (4, 7, 46, 244);
INSERT INTO rozpocet VALUES (4, 9, 58, 255);
```

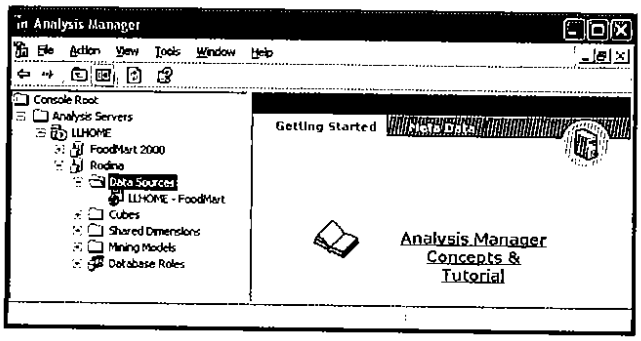
– třetí měsíc

```
INSERT INTO rozpocet VALUES (1, 2, 71, 154.20);
INSERT INTO rozpocet VALUES (1, 2, 86, 212);
INSERT INTO rozpocet VALUES (1, 2, 79, 133);
INSERT INTO rozpocet VALUES (1, 5, 64, 92);
INSERT INTO rozpocet VALUES (1, 5, 54, 61);
INSERT INTO rozpocet VALUES (2, 1, 81, 813);
INSERT INTO rozpocet VALUES (2, 1, 82, 298);
INSERT INTO rozpocet VALUES (3, 6, 75, 25);
INSERT INTO rozpocet VALUES (3, 8, 80, 118);
INSERT INTO rozpocet VALUES (4, 7, 76, 291);
INSERT INTO rozpocet VALUES (4, 9, 68, 89);
```

Teď, když máme připraveny údaje, můžeme vytvořit krychli OLAP.

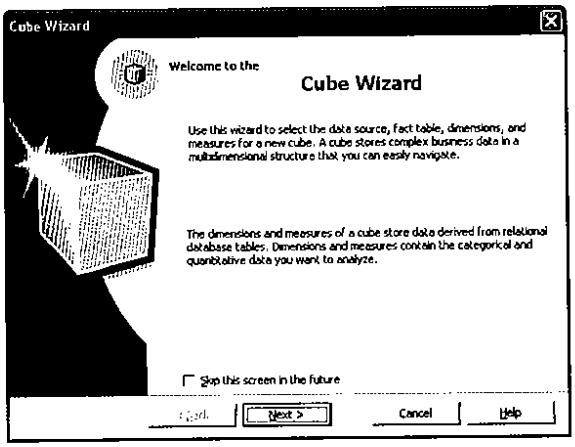
Vytvoření jednoduché krychle pomocí průvodce

Pro vytvoření nové krychle pomocí průvodce a pro práci s analytickým serverem vůbec budeme využívat aplikaci **Analysis Manager**, kterou najdeme v menu MS SQL Serveru. Pro naši cvičnou krychli jsme vytvořili novou analytickou databázi s názvem Rodina. Ve složce dataSources jsme ji napojili na databázi FoodMart, kde máme tabulky cvičné miniaplikace rodinný rozpočet.



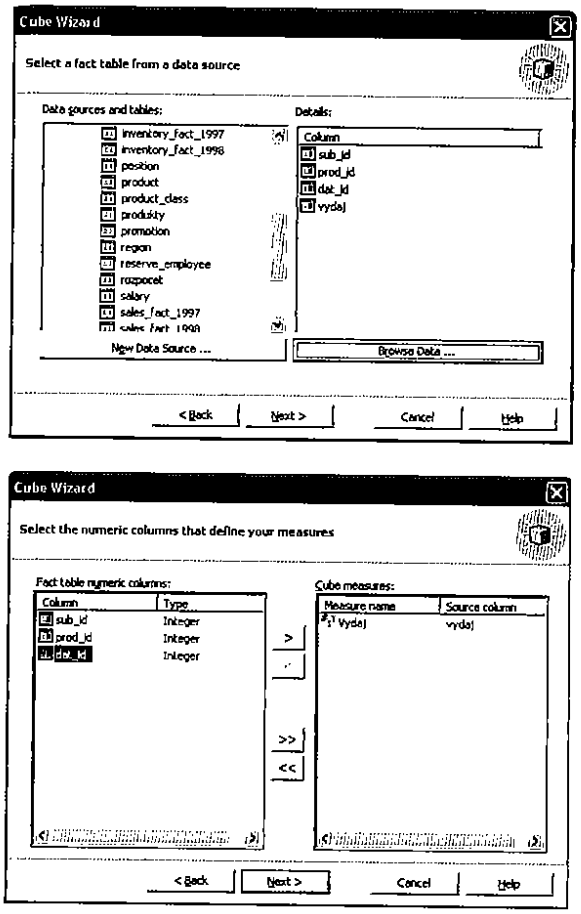
Obr. 4.13 Analysis Services – vytvoření nové analytické databáze a její napojení na zdroj údajů

V záložce **Cubes** vytvoříme novou krychli prostřednictvím průvodce vytvoření krychle. (Cube Wizard). Celý postup je tímto průvodcem dobře navigován, takže ho zvládne úspěšně napoprvé i začátečník.



Obr. 4.14 Cube Wizard – úvodní dialog

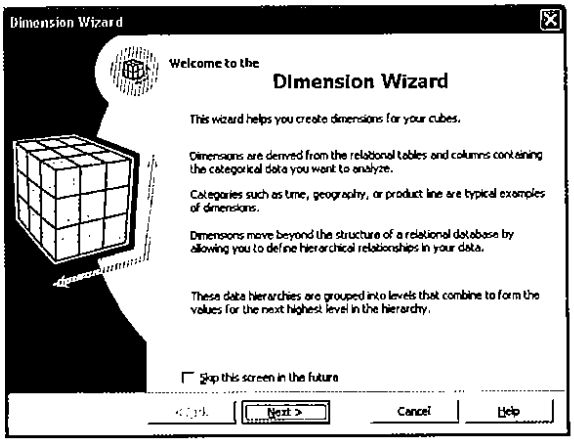
V následujícím kroku vybereme tabulku faktů, které tvoří měrné jednotky obchodování (v rodinném rozpočtu spíše utrácení). V našem případě to bude tabulka rozpočet. V následujícím kroku (na obrázku vpravo) vybereme konkrétní sloupec, který bude obsahovat numerická fakta, v našem případě to bude sloupec výdaj.

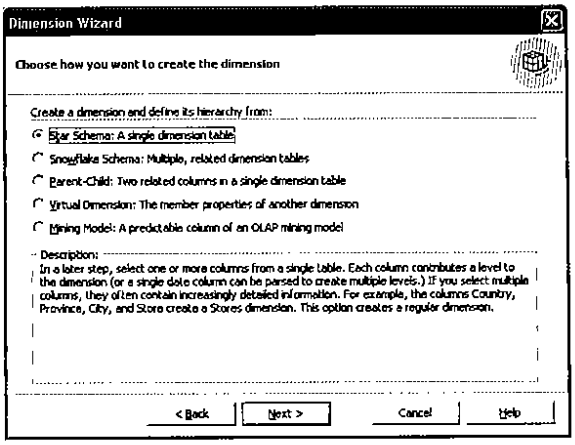


Obr. 4.15 Cube Wizard – výběr tabulky faktů a měrných jednotek obchodování

Po výběru tabulky faktů se prostřednictvím nástroje **Cube Wizard** dostaneme k vytvoření jednotlivých dimenzí. V našem případě budeme mít tři dimenze: subjektovou, předmětovou a časovou. I pro vytvoření dimenzí nám bude nabídnut další užitečný průvodce –

Dimension Wizard, který na chvíli „převezme žezlo od Cube Wizarda". V následujícím dialogu si můžeme zvolit vhodná schémata dimenzí. Všechny naše dimenze budou typu „star schema".

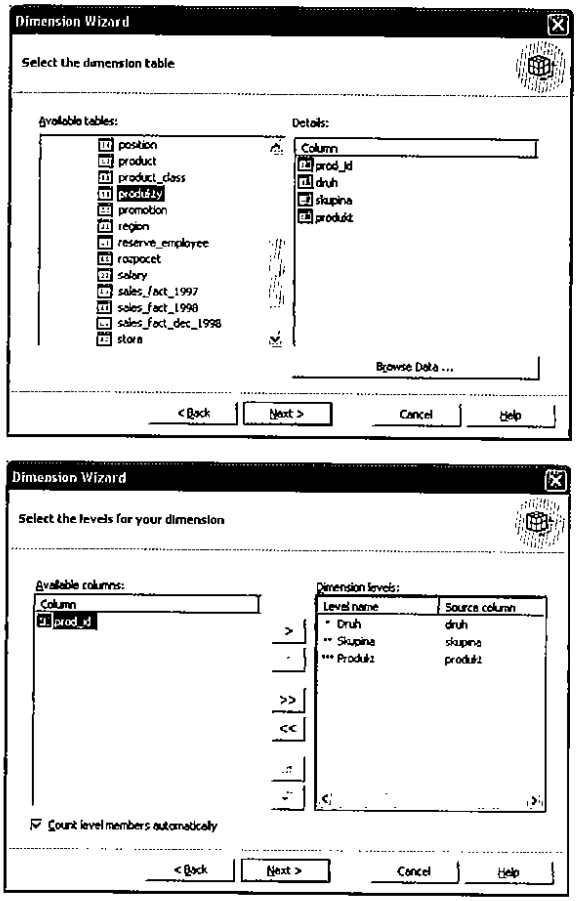




Obr. 4.16 Dimension Wizard – úvodní dialog a výběr typu schématu

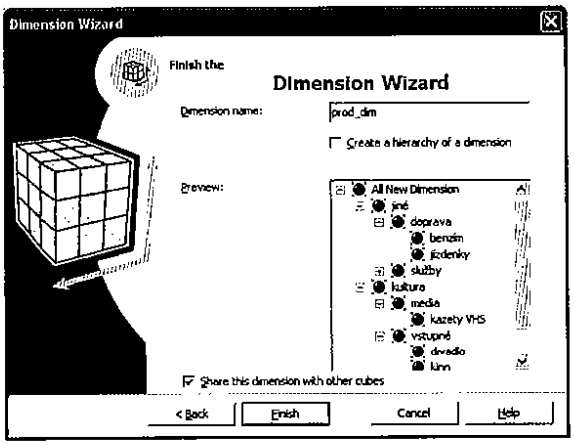
Předmětová dimenze

Tato dimenze bude obsahovat hierarchické členění produktů, které jsou v našem případě předmětem utrácení prostředků rodinného rozpočtu. Nejdříve vybereme databázovou tabulku produkty, která bude základem pro vytvoření produktové dimenze. V dalším dialogu vytvoříme přesouváním ikon sloupců hierarchickou strukturu dimenze.



Obr. 4.17 Dimension Wizard – vytvoření produktové dimenze

V následujících dvou dialozích nic neměníme. Důležitý je poslední dialog, ve kterém dimenzi pojmenujeme a kde si můžeme prohlédnout i hierarchickou strukturu jejích úrovní. Nezapomeňme zaškrtnout možnost sdílení právě vytvořené dimenze v jiných krychlích.



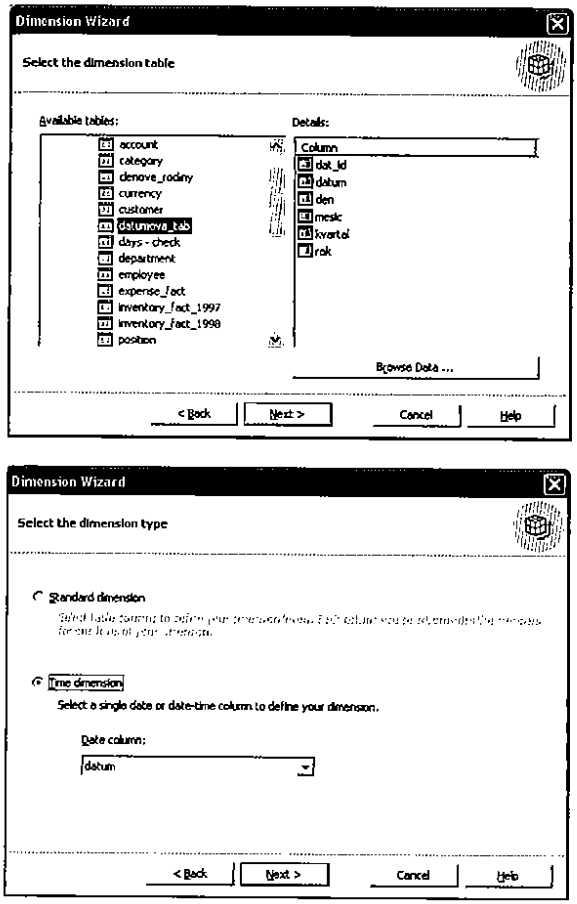
Obr. 4.18 Dimension Wizard – pojmenování dimenze

Subjektová dimenze

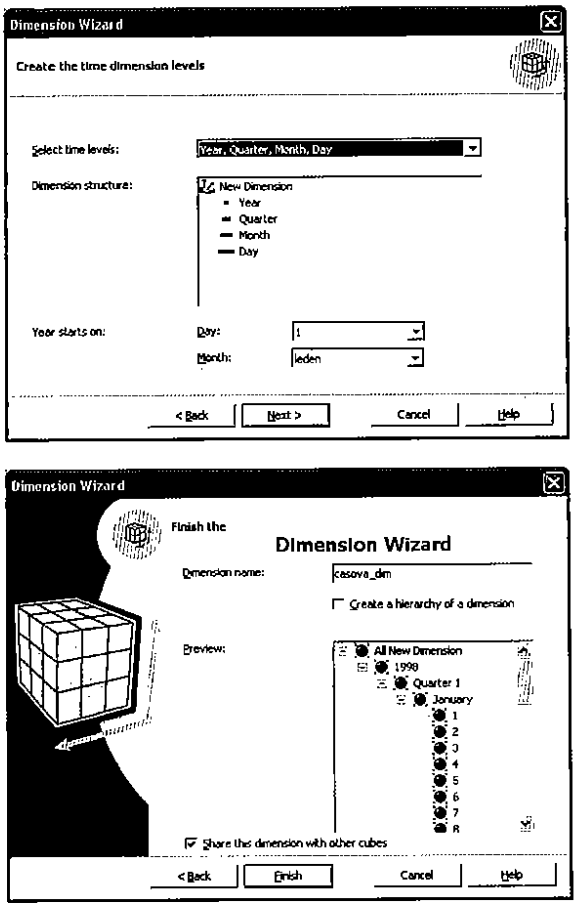
Podobným způsobem vytvoříme i subjektovou dimenzi. Tato dimenze bude obsahovat jména členů rodiny. Vytvoříme ji nad tabulkou clenove_rodiny a bude mít jen jednu úroveň – jméno. Dimenzi pojmenujeme subj_dim.

Časová dimenze

Časová dimenze bude obsahovat hierarchicky uspořádaná datua podle kalendářních zvyklostí. Vytvoříme ji nad tabulkou `datumova_tab`. Protože průvodce vytvořením dimenze rozpoznal, že tabulka obsahuje i datové a časové hodnoty, tak nám nabídne možnost vytvořit časovou dimenzi automaticky. Dimenzi pojmenujeme casova_dim.

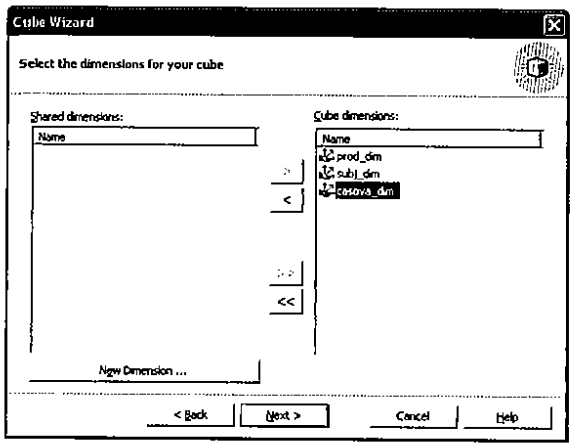


Obr. 4.19 Dimension Wizard – vytvoření časové dimenze



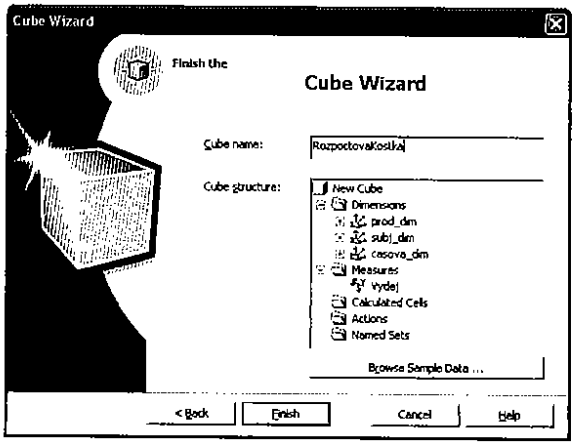
Obr. 4.20 Dimension Wizard – návrh hierarchie časové dimenze

Po vytvoření všech tří dimenzí můžeme zase pokračovat v dialozích průvodce Cube Wizard.



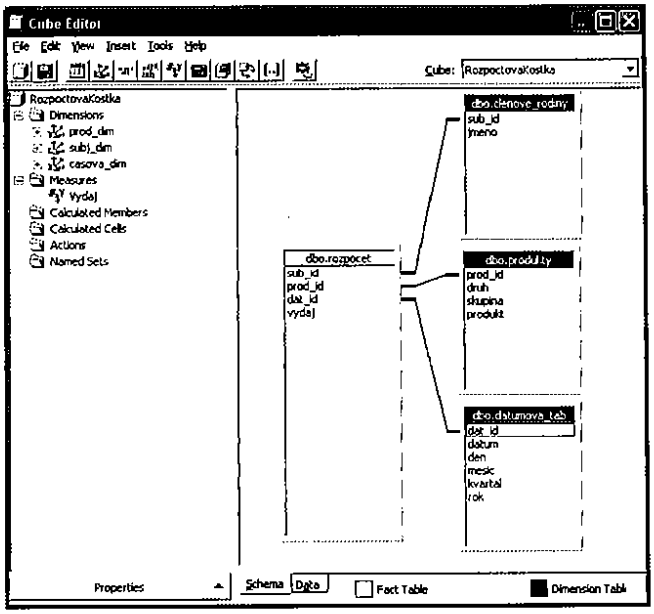
Obr. 4.21 Cube Wizard – stav po vytvoření dimenzí

Po přepočítaní tabulky faktů se dostáváme do finále, kde můžeme naši krychli pojmenovat a zkontrolovat některé její parametry.



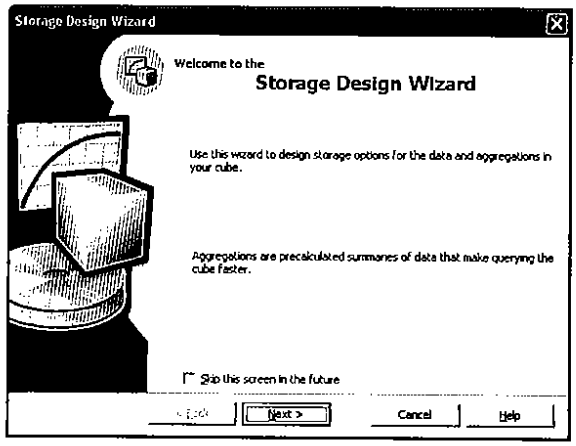
Obr. 4.22 Cube Wizard – závěrečný dialog

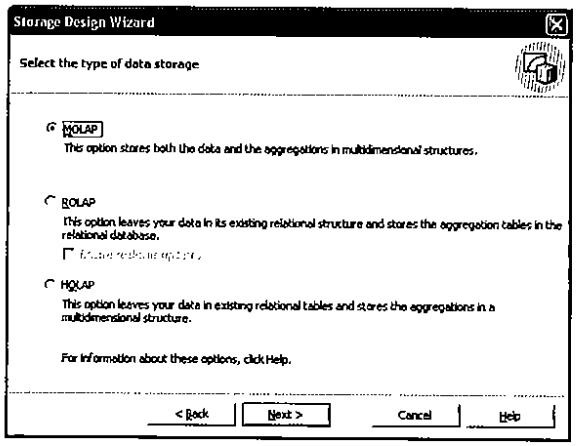
Po ukončení návrhu krychle nás Analysis Manager přepne do utility **Cube Editor**. Tento nástroj má v pravé části dvě záložky: **Schema** a **Data**. Záložka **Schema** obsahuje schéma tabulek faktů a dimenzí. Pozor, v záložce Data máme zatím jen cvičné údaje. Důvod je jednoduchý. Výpočet složitější krychle obvykle trvá i několik hodin, a kdybychom chtěli návrh a zobrazení takovéto krychle demonstrovat například na nějaké prezentaci, ocenili bychom schopnost ukázat strukturu údajů, aniž bychom museli přitom čekat na dopočítání celé krychle.



Obr. 4.23 Cube Editor – přehled topologie krychle

My máme krychli poměrně jednoduchou, takže můžeme pokračovat výpočtem krychle a uložením vypočtených hodnot. Po aktivaci tlačítka **Process Cube** v nástroji Cube Editor nám nabídne pomocnou ruku další průvodce, **Storage Design Wizard**, který nám pomůže s návrhem úložiště pro krychli.





Obr. 4.24 Storage Design Wizard

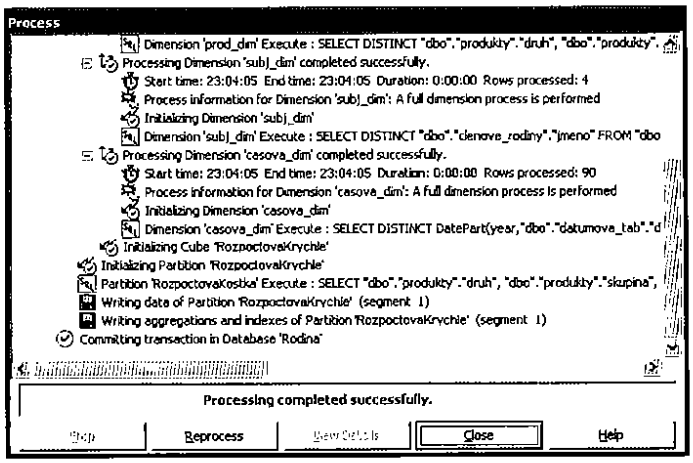
Hlavně u vícerozměrných velkých krychlí je důležité stanovit optimum mezi prostorem, který naše krychle zabere na disku, a rychlostí výpočtu a přístupu k údajům. Průvodce nám nabídne tři typy úložišť.

MOLAP je úložiště optimalizované pro multidimenzionální struktury.

ROLAP je úložiště optimalizované pro relační struktury.

HOLAP je úložiště MOLAP a ROLAP, kdy údaje zůstanou v relačních strukturách a vypočítané agregace budou uloženy v multidimenzionálních strukturách.

Zvolíme si například úložiště údajů typu MOLAP a ve spolupráci s průvodcem vypočítáme optimalizovanou velikost úložiště. V našem případě, přestože jsme byli velkorysí, tak jsme nastavili hranici pro úložiště krychle na 1 MB. Nemusíme nic počítat ani optimalizovat.

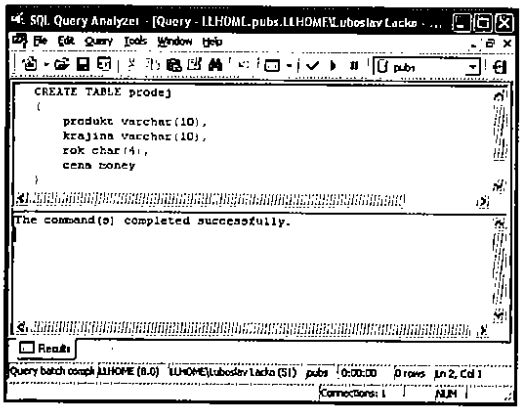


Obr. 4.25 Průběh výpočtu krychle

Tímto krokem jsme tvorbu krychle ukončili a můžeme s ní pracovat.

Krychle OLAP v jazyce SQL – klauzule CUBE

Pomocí klauzule CUBE získáme multidimenzionální přehled všech možných kombinací dle vybraných dimenzí. Můžeme tak vykonat analýzu a sumarizaci údajů v tabulce současně podle více kritérií. Praktické použití pro vytvoření multidimenzionální krychle si ukážeme na tabulce prodeje produktů. Pro názornost nepoužijeme tabulky faktů a dimenzí, ale všechny údaje budou v jedné jednoduché tabulce. Námětem je prodej čtyř potravinářských produktů (mléko, pivo, chléb a rohlíky) v Čechách a na Slovensku za roky 2000 a 2001. Protože jsme ještě stále blízko kapitoly OLAP pro začátečníky, ukážeme i příkazy SQL pro vytvoření cvičné tabulky `prodej`. Příkazy zadáváme pomocí konzolové aplikace Query Analyser. Cvičnou tabulku naplníme v cyklu pomocí generátoru náhodných čísel.



Obr. 4.26 Query Analyser

```
CREATE TABLE prodej
(
    produkt varchar(10),
    zeme varchar(10),
    rok char(4),
    cena money
)
```

```
DECLARE @N int
SET @N=10000
WHILE (@N >0)
BEGIN
   INSERT INTO prodej VALUES ('mléko','Čechy', '2000', RAND(@N)*390);
   INSERT INTO prodej VALUES ('pivo', 'Čechy', '2000', RAND(@N)*975);
   INSERT INTO prodej VALUES ('chléb','Čechy', '2000', RAND(@N)*180);
   INSERT INTO prodej VALUES ('rohlíky', 'Čechy', '2000', RAND(@N)*79);

   INSERT INTO prodej VALUES ('mléko', 'Slovensko', '2000', RAND(@N)*270);
   INSERT INTO prodej VALUES ('pivo', 'Slovensko','2000', RAND(@N)*600);
   INSERT INTO prodej VALUES ('chléb', 'Slovensko', '2000', RAND(@N)*120);
   INSERT INTO prodej VALUES ('rohlíky', 'Slovensko', '2000', RAND(@N)*55);

   INSERT INTO prodej VALUES ('mléko','Čechy', '2001', RAND(@N)*440);
   INSERT INTO prodej VALUES ('pivo', 'Čechy', '2001', RAND(@N)*1300);
   INSERT INTO prodej VALUES ('chléb','Čechy', '2001', RAND(@N)*245);
   INSERT INTO prodej VALUES ('rohlíky', 'Čechy', '2001', RAND(@N)*110);
```

```
    INSERT INTO prodej VALUES ('mléko', 'Slovensko', '2001', RAND(@N)*310);
    INSERT INTO prodej VALUES ('pivo', 'Slovensko','2001', RAND(@N)*720);
    INSERT INTO prodej VALUES ('chléb', 'Slovensko', '2001', RAND(@N)*199);
    INSERT INTO prodej VALUES ('rohlíky', 'Slovensko', '2001', RAND(@N)*90);

SET @N=@N-1
END
```

Některé informace zjistíme i pomocí klasických dotazů SQL, například počet záznamů nebo celkovou inkasovanou sumu.

```
SELECT COUNT(*) FROM prodej
SELECT SUM(cena) FROM prodej
```

V naší tabulce máme 160 000 záznamů a prodali jsme zboží za 49 074 452,43 Kč. Jsou to dvě fakta, která zatím nedokážeme zjemnit a analyzovat. Na základě těchto informací můžeme jen konstatovat, zda nám tento dosahovaný obrat postačuje, nebo nepostačuje. Nevíme ale nic například o prodeji jednotlivých druhů zboží v jednotlivých zemích ani za jednotlivé roky.
Pomocí klauzule GROUP BY můžeme údaje seskupit podle jednotlivých kritérií:

```
SELECT produkt, zeme, rok, SUM(cena) AS Obrat
FROM prodej
GROUP BY produkt, zeme, rok
```

```
produkt    zeme      rok  Obrat
---------- --------- ---- ---------------------
mléko      Čechy     2000 3146315.3778
chléb      Čechy     2000 1452145.5596
pivo       Slovensko 2001 5808582.2377
rohlíky    Slovensko 2000 443711.1434
chléb      Čechy     2001 1976531.4560
chléb      Slovensko 2001 1605427.5906
mléko      Čechy     2001 3549689.1444
mléko      Slovensko 2000 2178218.3391
pivo       Slovensko 2000 4840485.1982
rohlíky    Čechy     2000 637330.5510
pivo       Čechy     2000 7865788.4467
rohlíky    Čechy     2001 887422.2868
mléko      Slovensko 2001 2500917.3523
pivo       Čechy     2001 10487717.9290
rohlíky    Slovensko 2001 726072.7798
chléb      Slovensko 2000 968097.0397
```

Takový výpis nám poskytne už více informací. Ale stále bychom potřebovali tyto údaje buď zjemnit, nebo sumarizovat. Proto se pustíme do analýzy, v našem případě si vytvoříme takzvanou krychli, což znamená multidimenzionální přehled všech možných kombinací podle vybraných dimenzí. Použijeme tedy příkaz SQL obsahující klauzuli CUBE.

```
SELECT produkt, zeme, rok, SUM(cena) AS Obrat
FROM prodej
GROUP BY produkt, zeme, rok WITH CUBE
```

```
produkt    zeme     rok         Obrat
---------- -------- ----------- --------------------
chléb      Čechy    2000        1452145.5596
chléb      Čechy    2001        1976531.4560
chléb      Čechy    NULL        3428677.0156
chléb      Slovensko 2000       968097.0397
chléb      Slovensko 2001       1605427.5906
chléb      Slovensko NULL       2573524.6303
chléb      NULL     NULL        6002201.6459
mléko      Čechy    2000        3146315.3778
mléko      Čechy    2001        3549689.1444
mléko      Čechy    NULL        6696004.5222
mléko      Slovensko 2000       2178218.3391
mléko      Slovensko 2001       2500917.3523
mléko      Slovensko NULL       4679135.6914
mléko      NULL     NULL        11375140.2136
pivo       Čechy    2000        7865788.4467
pivo       Čechy    2001        10487717.9290
pivo       Čechy    NULL        18353506.3757
pivo       Slovensko 2000       4840485.1982
pivo       Slovensko 2001       5808582.2377
pivo       Slovensko NULL       10649067.4359
pivo       NULL     NULL        29002573.8116
rohlíky    Čechy    2000        637330.5510
rohlíky    Čechy    2001        887422.2868
rohlíky    Čechy    NULL        1524752.8378
rohlíky    Slovensko 2000       443711.1434
rohlíky    Slovensko 2001       726072.7798
rohlíky    Slovensko NULL       1169783.9232
rohlíky    NULL     NULL        2694536.7610
NULL       NULL     NULL        49074452.4321
NULL       Čechy    2000        13101579.9351
NULL       Čechy    2001        16901360.8162
NULL       Čechy    NULL        30002940.7513
NULL       Slovensko 2000       8430511.7204
NULL       Slovensko 2001       10640999.9604
NULL       Slovensko NULL       19071511.6808
chléb      NULL     2000        2420242.5993
mléko      NULL     2000        5324533.7169
pivo       NULL     2000        12706273.6449
rohlíky    NULL     2000        1081041.6944
NULL       NULL     2000        21532091.6555
chléb      NULL     2001        3581959.0466
mléko      NULL     2001        6050606.4967
pivo       NULL     2001        16296300.1667
rohlíky    NULL     2001        1613495.0666
NULL       NULL     2001        27542360.7766

(45 row(s) affected)
```

Když z tabulky odstraníme řetězec NULL (nahradíme ho čtyřmi mezerami), tak bude výpis trochu přehlednější. Všimněme si, že tabulka obsahuje jednak souhrn za všechny produkty, v našem případě řádek,

```
NULL        NULL        NULL        49074452.4321
```

jednak souhrny za chléb, mléko, pivo a rohlíky v řádcích

```
chléb       NULL        NULL        6002201.6459
mléko       NULL        NULL        11375140.2136
pivo        NULL        NULL        29002573.8116
rohlíky     NULL        NULL        2694536.7610
```

a také souhrny pro jednotlivé země

```
NULL        Čechy       NULL        30002940.7513
NULL        Slovensko   NULL        19071511.6808
```

a roky

```
NULL        NULL        2000        21532091.6555
NULL        NULL        2001        27542360.7766
```

Přehlednější výpis dostaneme, když použijeme funkci GROUPING, která vrací 0, když daná hodnota vypočítaná klauzulí CUBE obsahuje nějaká fakta, tedy konkrétní hodnotu a 1, když je tam HODNOTA NULL, která byla vygenerovaná pomocí CUBE. Musíme se pochopitelně pojistit i pro případ, že by v některém sloupci byla hodnota NULL jako taková, tedy nevygenerovaná klauzulí CUBE.

```
SELECT CASE WHEN (GROUPING(produkt) = 1) THEN 'S'
            ELSE ISNULL(produkt, '???')
       END AS produkt,
       CASE WHEN (GROUPING(zeme) = 1) THEN 'S'
            ELSE ISNULL(zeme, '???')
       END AS zeme,
       CASE WHEN (GROUPING(rok) = 1) THEN 'S'
            ELSE ISNULL(rok, '???')
       END AS rok,
       SUM(cena) AS obrat
FROM prodej
GROUP BY produkt, zeme, rok WITH CUBE
```

```
produkt    zeme       rok     obrat
---------- ---------- ------- ---------------------
chléb      Čechy      2000    1452145.5596
chléb      Čechy      2001    1976531.4560
chléb      Čechy      S       3428677.0156
chléb      Slovensko  2000    968097.0397
chléb      Slovensko  2001    1605427.5906
chléb      Slovensko  S       2573524.6303
chléb      S          S       6002201.6459
```

```
mléko       Čechy       2000    3146315.3778
mléko       Čechy       2001    3549689.1444
mléko       Čechy       S       6696004.5222
mléko       Slovensko   2000    2178218.3391
mléko       Slovensko   2001    2500917.3523
mléko       Slovensko   S       4679135.6914
mléko       S           S       11375140.2136
...
...
chléb       S           2000    2420242.5993
mléko       S           2000    5324533.7169
pivo        S           2000    12706273.6449
rohlíky     S           2000    1081041.6944
S           S           2000    21532091.6555
chléb       S           2001    3581959.0466
mléko       S           2001    6050606.4967
pivo        S           2001    16296300.1667
rohlíky     S           2001    1613495.0666
S           S           2001    27542360.7766

(45 row(s) affected)
```

Pro účely například obchodních prezentací je možné zobrazit tyto údaje v trochu zhuštěném a přehlednějším formátu. Nejdříve tomu budeme říkat „řídká krychle". Naše „krychle" má dimenze produkt, zeme a rok (dále jen PKR). Pomocí funkce GROUPING můžeme vytvořit masku jednotlivých dimenzí.

```
SELECT produkt, zeme, rok, SUM(cena) AS obrat,
GROUPING (produkt)AS P,
GROUPING (zeme)AS K,
GROUPING (rok)AS R
FROM prodej
GROUP BY produkt, zeme, rok WITH CUBE;
```

```
produkt     zeme        rok     obrat                P    K    R
----------- ----------- ------ --------------------- ---- ---- ----
chléb       Čechy       2000   1452145.5596          0    0    0
chléb       Čechy       2001   1976531.4560          0    0    0
chléb       Čechy       NULL   3428677.0156          0    0    1
chléb       Slovensko   2000   968097.0397           0    0    0
chléb       Slovensko   2001   1605427.5906          0    0    0
...
NULL        Slovensko   2001   10640999.9604         1    0    0
NULL        Slovensko   NULL   19071511.6808         1    0    1
chléb       NULL        2000   2420242.5993          0    1    0
mléko       NULL        2000   5324533.7169          0    1    0
pivo        NULL        2000   12706273.6449         0    1    0
rohlíky     NULL        2000   1081041.6944          0    1    0
NULL        NULL        2000   21532091.6555         1    1    0
chléb       NULL        2001   3581959.0466          0    1    0
mléko       NULL        2001   6050606.4967          0    1    0
```

```
pivo        NULL        2001    16296300.1667          0    1    0
rohlíky     NULL        2001    1613495.0666           0    1    0
NULL        NULL        2001    27542360.7766          1    1    0

(45 row(s) affected)
```

Všimněme si „masky" jednotlivých dimenzí. Když má množina atributů **PKR** pro daný záznam hodnotu **(0,0,0)** a chceme vytvořit řídkou krychli, která bude obsahovat jen souhrny za jednotlivé kombinace dimenzí, tak můžeme tento řádek vynechat. Záznamy, jejichž masky obsahují alespoň jednu jednotku, představují souhrny. Například záznam s maskou **PKR (1,1,0)** vypovídá o celkovém prodeji za rok 2001.

```
produkt     zeme     rok     obrat                  P    K    R
----------- ---------- ------ --------------------- ---- ---- ----
NULL        NULL        2001   27542360.7766         1    1    0
```

Představuje souhrny prodeje nápojů a pečiva. Necháme vypsat tedy jen záznamy, kde maska **MKT** obsahuje alespoň jednu jednotku.

```
SELECT produkt, zeme, rok,
SUM(cena) AS obrat
FROM prodej
GROUP BY produkt, zeme, rok WITH CUBE
HAVING (GROUPING(produkt)=1)OR(GROUPING (zeme)=1)OR(GROUPING (rok)=1);
```

```
produkt     zeme     rok      obrat
----------- ---------- ------- ---------------------
chléb       Čechy       NULL    3428677.0156
chléb       Slovensko   NULL    2573524.6303
chléb       NULL        NULL    6002201.6459
mléko       Čechy       NULL    6696004.5222
mléko       Slovensko   NULL    4679135.6914
mléko       NULL        NULL    11375140.2136
pivo        Čechy       NULL    18353506.3757
pivo        Slovensko   NULL    10649067.4359
pivo        NULL        NULL    29002573.8116
rohlíky     Čechy       NULL    1524752.8378
rohlíky     Slovensko   NULL    1169783.9232
rohlíky     NULL        NULL    2694536.7610
NULL        NULL        NULL    49074452.4321
NULL        Čechy       2000    13101579.9351
NULL        Čechy       2001    16901360.8162
NULL        Čechy       NULL    30002940.7513
NULL        Slovensko   2000    8430511.7204
NULL        Slovensko   2001    10640999.9604
NULL        Slovensko   NULL    19071511.6808
chléb       NULL        2000    2420242.5993
mléko       NULL        2000    5324533.7169
pivo        NULL        2000    12706273.6449
rohlíky     NULL        2000    1081041.6944
```

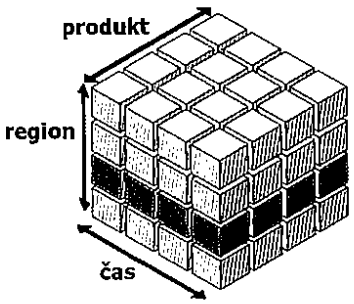
```
NULL        NULL        2000    21532091.6555
chléb       NULL        2001    3581959.0466
mléko       NULL        2001    6050606.4967
pivo        NULL        2001    16296300.1667
rohlíky     NULL        2001    1613495.0666
NULL        NULL        2001    27542360.7766

(29 row(s) affected)
```

Výpis bude obsahovat už jen 29 záznamů. Když nás zajímají údaje jen pro jednu krajinu, můžeme toto omezení realizovat podmínkou v klauzuli HAVING.

```
SELECT zeme, produkt, rok, SUM(cena) AS obrat
FROM prodej
GROUP BY zeme,produkt, rok WITH CUBE
HAVING ((GROUPING (zeme)=1)OR(GROUPING(produkt)=1)OR(GROUPING (rok)=1))
          AND (zeme = 'Slovensko');


zeme      produkt     rok  obrat
---------- ---------- ---- ---------------------
Slovensko  chléb      NULL 2573524.6303
Slovensko  mléko      NULL 4679135.6914
Slovensko  pivo       NULL 10649067.4359
Slovensko  rohlíky    NULL 1169783.9232
Slovensko  NULL       NULL 19071511.6808
Slovensko  NULL       2000 8430511.7204
Slovensko  NULL       2001 10640999.9604
```

Výstup je sice správný, ale není příliš přehledný, je tam nadbytečný sloupec. Proto můžeme odstranit sloupec země, čímž zredukujeme „krychli" o jeden rozměr a pomocí podmínky v klauzuli WHERE zařídíme, že se bude pracovat jen s údaji ze Slovenska.

```
SELECT produkt, rok,
SUM(cena) AS obrat
FROM prodej WHERE zeme = 'Slovensko'
GROUP BY produkt, rok WITH CUBE
HAVING (GROUPING(produkt)=1)OR(GROUPING (rok)=1)

produkt    rok  obrat
---------- ---- ---------------------
chléb      NULL 2573524.6303
mléko      NULL 4679135.6914
pivo       NULL 10649067.4359
rohlíky    NULL 1169783.9232
NULL       NULL 19071511.6808
NULL       2000 8430511.7204
NULL       2001 10640999.9604
```

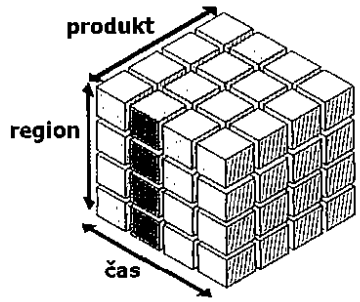


Obr. 4.27 Údaje omezené jen na jednu geografickou oblast

Podobně si můžeme nechat vypsat údaje za jeden rok, v našem případě za rok 2000.

```
SELECT produkt, zeme,
SUM(cena) AS obrat
FROM prodej WHERE rok = 2000
GROUP BY produkt, zeme WITH CUBE
HAVING (GROUPING(produkt)=1)OR(GROUPING (zeme)=1);
```

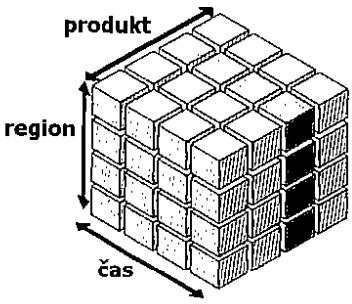
```
produkt    zeme       obrat
---------- ---------- ---------------------
chléb      NULL       2420242.5993
mléko      NULL       5324533.7169
pivo       NULL       12706273.6449
rohlíky    NULL       1081041.6944
NULL       NULL       21532091.6555
NULL       Čechy      13101579.9351
NULL       Slovensko  8430511.7204
```



Obr. 4.28 Údaje omezené jen na určitý časový úsek

Samozřejmě to stejné můžeme udělat pro určitý produkt, například pro chléb.

```
SELECT zeme, rok,
SUM(cena) AS obrat
FROM prodej WHERE produkt = 'chléb'
GROUP BY zeme, rok WITH CUBE
HAVING (GROUPING (zeme)=1)OR(GROUPING (rok)=1)
```



Obr. 4.29 Údaje omezené jen na určitý produkt

Protože v našem příkladě nemáme tabulky dimenzí, nemůžeme se do krychle „zavrtat" (drill down) trochu hlouběji a udělat například analýzu pro jednotlivé dny.

V některých případech trvá výpočet všech hodnot krychle poměrně dlouhý čas. Proto se v praxi vypočítají krychle za určité časové období, například za každý den jen jednou (například z údajů aktualizovaného datového skladu), a uloží se do samostatné databáze. A kdyby tento proces probíhal několik hodin, můžeme ho automaticky odstartovat po půlnoci pro údaje za předcházející den. Ráno, když manažeři a analytici přijdou do práce, tak mají všechny údaje za předcházející den okamžitě k dispozici. Když porovnáme čas odezvy serveru, předtím trval výpočet krychle několik desítek sekund až hodiny, oproti tomu výběr údajů z dopředu vypočítané tabulky je prakticky okamžitý. Použitím klauzule CUBE získáme přehledný výpis přepočítaných souhrnů sum pro dané rozměry.

Na první pohled jednoduché, ale v praxi málo využitelné. Krychle se v našem jednoduchém příkladě vytvářela na základě jednoduché tabulky, která obsahuje dimenze i údaje. Také zjistíme, že jednotlivé dimenze krychle (`rok`, `zeme`, `zbozi`) jsou už „dále nedělitelné". V reálných krychlích se totiž často využívá tzv „drilování", což znamená, že některou dimenzi zjemníme (*drill down*), nebo nás naopak zajímají i globálnější údaje, tak použijeme hrubší nastavení dimenze (*drill up*). Lehce to pochopíme například na časové dimenzi. Někdy potřebujeme údaje za měsíc, jindy za týden, nebo dokonce za jeden den, jindy zase potřebujeme údaje za jednotlivé roky. A to už nehovoříme o tom, kolik řádků by v takovéto interpretaci obsahovala i značně řídká krychle, kdyby měla více dimenzí.

Vytvoření jednoduché krychle OLAP nad podnikovou databází

Cvičná databáze **FoodMart** dodávána a nainstalována s analytickými službami MS SQL Serveru 2000 je z aplikačního hlediska databáze velkého potravinářského obchodního řetězce, který má svoje prodejny v USA, Mexiku a Kanadě. Jeho podrobná struktura je v příloze číslo 2. V zásadě jsou dva možné přístupy, jak pracovat s touto databází. Buď ji ponecháme jako soubor MDB a budeme se k ní připojovat přes ODBC nebo OLEDB. Tato možnost je výhodná pro ty, kteří pracují s programem Microsoft Access.

Jak uvidíme v dalších kapitolách, krychle OLAP můžeme vytvářet a pracovat s nimi i pomocí programu Microsoft Excel. Když použijeme MS SQL Server, tak je výhodné migrovat tuto databázi pod správu databázového serveru, kde s ní můžeme pohodlně pracovat pomocí klientské konzolové aplikace. Postup etapy ETL pro databázi FoodMart je popsán v třetí kapitole.

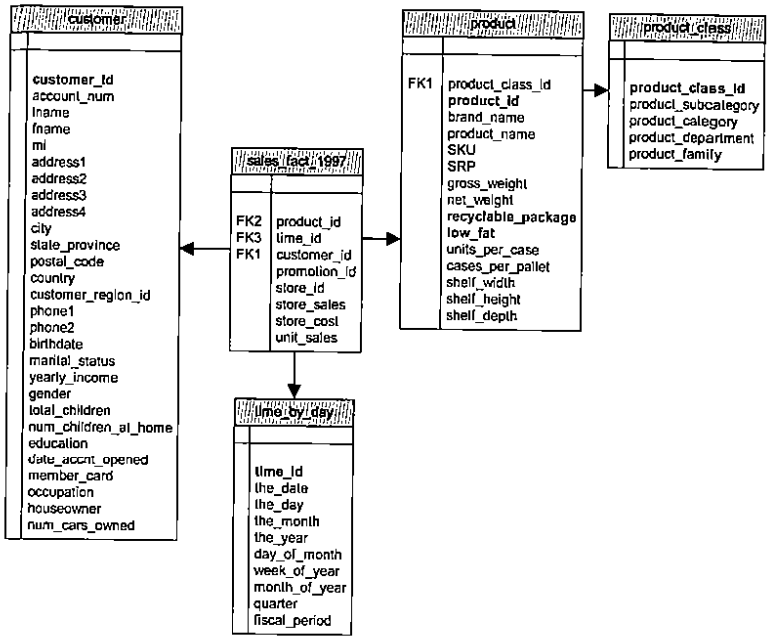
Při instalaci analytických služeb se kromě zkopírování transakční databáze OLPT vytvoří i analytická databáze Foodmart a v ní tři cvičné krychle (Marketing, Human Resources (HR) a Expense Budget) a i všechny potřebné dimenze, ale z pedagogického hlediska bude rozhodně zajímavější začít od začátku a projít kompletně celý příklad od připojení ke zdroji údajů, přes vytvoření dimenzí, výpočet krychle a připojení k analytickým údajům. Základy postupu práce s průvodcem vytvoření krychle jsou ve stati OLAP pro začátečníky, takže v tomto příkladě se zaměříme na konkrétní řešení.

Abychom mohli efektivně navrhovat multidimenzionální strukturu OLAP, je výhodné připravit si diagram příslušné databáze OLTP. V tomto konkrétním případě můžeme pro zobrazení, úpravy a tisk relačního diagramu použít buď program Microsoft Access, případně program MS Visio 2002 , nebo když jsme databázi migrovali na MS SQL Server, můžeme pro vykreslení diagramu použít aplikaci MS SQL Serveru 2000 Enterprise Manager. V našem příkladě budeme pracovat s tabulkami: **sales_fact_1997** (tabulka faktů), **customer, time_by_day, product** a **product_class** (tabulky dimenzí).

Tabulka faktů (sales_facts_1997) obsahuje záznamy o jednotlivých prodejních transakcích.

product_id	time_id	customer_id	promotion_id	store_id	store_sales	store_cost	unit_sales
869	367	3449	0	6	10.6000	3.8160	5.0
1472	367	3449	0	6	6.6000	2.5080	3.0
76	367	3449	0	6	6.7600	3.2448	4.0
320	367	3449	0	6	9.7800	3.6186	3.0
4	367	3449	0	6	14.5600	4.8048	4.0
952	367	3449	0	6	11.6400	5.8200	4.0
1222	367	3449	0	6	9.8000	3.7240	4.0
517	367	7859	0	6	13.6000	5.8480	4.0

Když se podíváme na tabulku **time_by_day**, tak víme, že první záznam z našeho výpisu je z prvního ledna 1997.



Obr. 4.30 Příklad schématu dimenzí typu „STAR" a „SNOWFLAKE" (diagram byl vytvořený pomocí Microsoft Visio 2002, postup je uvedený v příloze č. 1)

Tabulky dimenzí – jako dimenze pro naši jednoduchou cvičnou krychli jsme vybrali tabulky Product, Store, Customer a Time_by_day. Zatím ukážeme jen schéma stromové struktury jednotlivých tabulek. Podrobněji si o jednotlivých tabulkách a jejich obsahu povíme při návrhu jednotlivých dimenzí.

```
Product
  • Product Family
  • • Product Department
  • • • Product Category
  • • • • Product Subcategory
  • • • • • Product Name

Customers
  • Country
  • • State Province
  • • • City
  • • • • Name

Store
  • Store Country
  • • Store State
  • • • Store City
  • • • • Store Name

Time
  • Year
  • • Quarter
  • • • Month
```

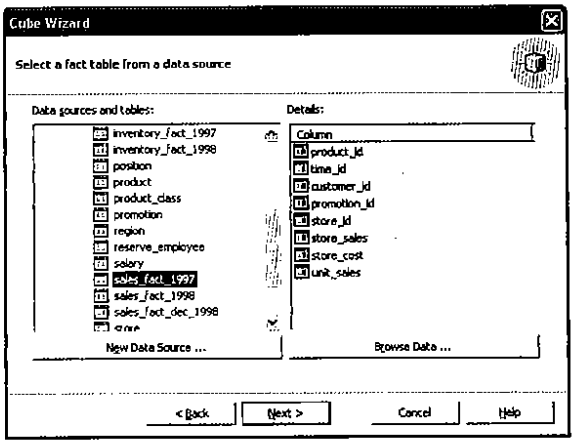
Po těchto příkladech by měla být pozornějšímu čtenáři vazba mezi tabulkou faktů a tabulkami dimenzí poměrně jasná. Nástroje pro návrh krychlí a dimenzí obvykle umožňují i grafické zobrazení vazeb mezi tabulkami faktů a dimenzí.

Vytvoření krychle OLAP pomocí průvodce

Po předběžném seznámení s databází FoodMart a hrubém návrhu topologie krychle můžeme přikročit k jejímu vytvoření pomocí aplikace *Analysis Manager*, která je součástí konzoly *Microsoft Management Console*. Vytvoříme novou analytickou databázi s názvem například *SuperMarket*. Do složky *Data Sources* analytické databáze přidáme odkaz na relační databázi pod správou MS SQL Serveru s názvem *Supermarket*. Můžeme se samozřejmě připojit i přímo k databázi FoodMart 2000 ve formátu MS Access například prostřednictvím rozhraní ODBC (Open Database Connectivity). Novou krychli vytvoříme na záložce **Cubes** pomocí průvodce vytvoření krychle. (Cube Wizard).

Tabulka faktů

V prvním kroku zvolíme **tabulku faktů**, v našem případě to bude tabulka *sales_fact_1997*. Není těžké už podle názvu uhodnout, že tato tabulka obsahuje údaje o obchodování v roce 1997. Ukážeme zjednodušený výpis tabulky faktů, který obsahuje tři sloupce `product_id`, `time_id` a `customer_id`, které představují vazby na tabulky dimenzí. Další tři sloupce `store_sales`, `store_cost` a `unit_sales` obsahují měrné jednotky obchodování.

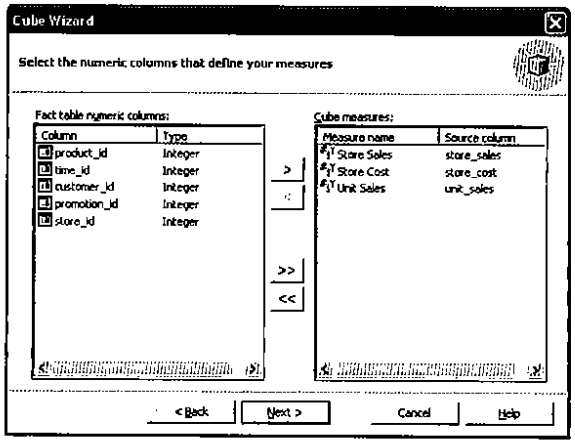


Obr. 4.31 Výběr tabulky faktů

Příklad údajů uložených v tabulce faktů. Sloupce obsahující fakta jsou vypsány tučným fontem.

```
product_id  time_id   customer_id store_sales   store_cost   unit_sales
----------- --------- ----------- ------------- ------------ -----------
173         748       2094        4.2900        1.8447       3.0
1119        748       2094        9.5100        3.5187       3.0
1242        748       2094        7.9200        2.8512       4.0
460         748       2094        6.4400        2.7048       4.0
104         748       2094        11.6700       3.9678       3.0
...
```

V následujícím kroku vybereme sloupce z tabulky faktů, které budou **měrnými jednotkami pro analýzu**, v našem případě to budou logicky sloupce `store_sales`, `store_cost` a `unit_sales`.



Obr. 4.32 Výběr měrných jednotek v tabulce faktů

Tabulky dimenzí

Dvě dimenze tvoří jednoduché tabulky *customer* a *time_by_day*. První z nich charakterizuje zákazníky, v našem příkladě nás bude zajímat hlavně jejich regionální charakteristika. Druhá tabulka představuje časovou dimenzi. Třetí dimenzi budou tvořit dvě relačně svázané tabulky *product* a *product_class*. Dimenze vytvoříme pomocí průvodce vytvořením dimenzí (Dimension Wizard).

Zákaznická (regionální) dimenze

Dimenzi budou tvořit jen atributy (sloupce) z jedné tabulky, z tabulky CUSTOMERS.

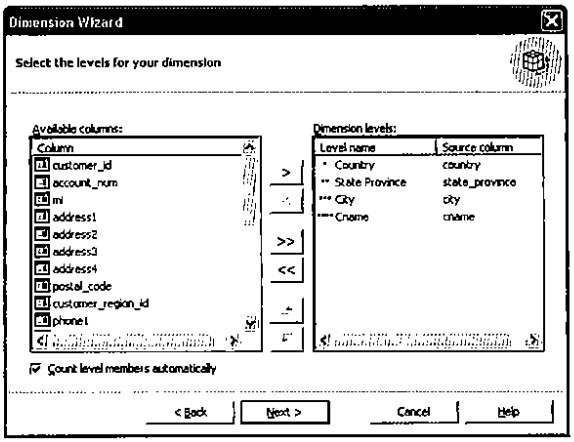
```
customer_id cname           city          state_province
----------- --------        ------------- --------------
1           Nowmer Sheri    Tlaxiaco      Oaxaca
2           Whelply Derrick Sooke         BC
```

```
3          Derry Jeanne      Issaquah      WA
4          Spence Michael    Burnaby       BC
...
```

Každý z atributů tvoří jednu z hierarchických úrovní stromové struktury dimenze, například sloupce stát, kraj, region, město pro dimenzi regionálního typu. Hierarchie zákaznické dimenze bude:

```
Zákazníci
• Country
• • State Province
• • • City
• • • • Name
```

V jednom z návrhových dialogů se můžeme rozhodnout pro standardní, nebo časovou dimenzi. Dimenze **Zákazníci** je samozřejmě typickým příkladem standardní dimenze. Postupně navrhneme jednotlivé úrovně této dimenze jednoduchým přemístěním sloupců z levé strany dialogu na pravou.



Obr. 4.33 Návrh dimenze Zákazníci

Časová dimenze

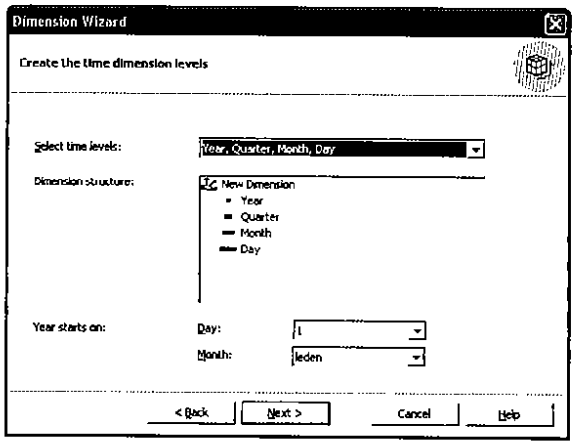
Časovou dimenzi vytvoříme nad tabulkou **time_by_day.**

```
time_id  the_date    the_day     the_month   the_year day_of_month month_of_year quarter
-------- ----------- ----------- ----------- -------- ------------ ------------- -------
738      1998-01-07  Wednesday   January     1998     7            1             Q1
739      1998-01-08  Thursday    January     1998     8            1             Q1
740      1998-01-09  Friday      January     1998     9            1             Q1
```

Podobně jako v předcházejícím případě šlo o schéma Star, ale tentokrát v příslušném dialogu potvrdíme, že se jedná o časovou dimenzi. Podstatně nám to ulehčí její návrh. Hierarchie dimenze bude logicky vycházet z kalendářních zvyklostí.

```
Čas
• Year
• • Quarter
• • • Month
```

Návrh jednotlivých úrovní časové dimenze nad takto logicky uspořádanou tabulkou zvládne průvodce sám a nabídne nám výsledný dialog.



Obr. 4.34 Hierarchická úroveň automaticky navržené časové dimenze

Dimenzi pojmenujeme **Čas**.

Časová dimenze

Zatím se každá dimenze něčím lišila od předcházející. Produktová dimenze bude standardní, ale použijeme schéma Snowflake, u kterého jsou dimenze tvořeny jedním nebo více sloupci z několika relačně svázaných tabulek. V našem případě to budou tabulky PRODUCT

```
product_class_id product_id  product_name                    ...
---------------- ----------- ------------------------------
30               1           Washington Berry Juice
19               6           Washington Cola
35               12          Jeffers Oatmeal
```

```
35              13          Jeffers Corn Puffs
62              16          Blue Label Canned Beets
...
```

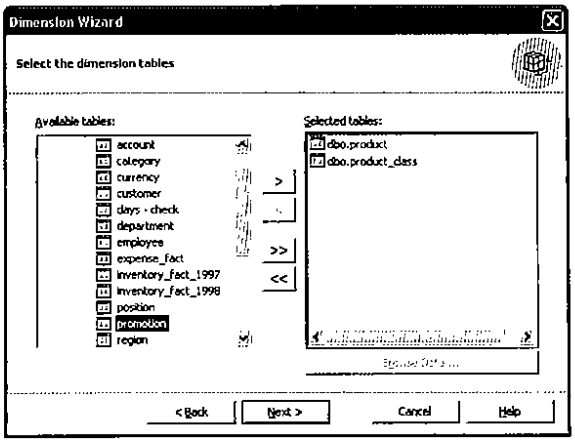
a PRODUCT_CLASS.

```
product_class_id product_subcategory product_category  product_department  product_family
---------------- ------------------- ----------------- -------------------- --------------
1                Nuts                Specialty         Produce              Food
2                Shellfish           Seafood           Seafood              Food
3                Canned Fruit        Fruit             Canned Products      Food
4                Spices              Baking Goods      Baking Goods         Food
5                Pasta               Starchy Foods     Starchy Foods        Food
6                Yogurt              Dairy             Dairy                Food
...
```

Podobně jako u hvězdicového schématu každý z atributů tvoří jednu z hierarchických úrovní stromové struktury dimenzí.

```
Produkty
  • Product Family
  • • Product Department
  • • • Product Category
  • • • • Product Subcategory
  • • • • • Product Name
```

Dimenzi **Produkty** můžeme podle našich představ vytvořit jen z obou dvou relačně svázaných tabulek. Postup je stejný jako u schématu Star s tím rozdílem, že si vybereme dvě tabulky: *product* a *product_class*.

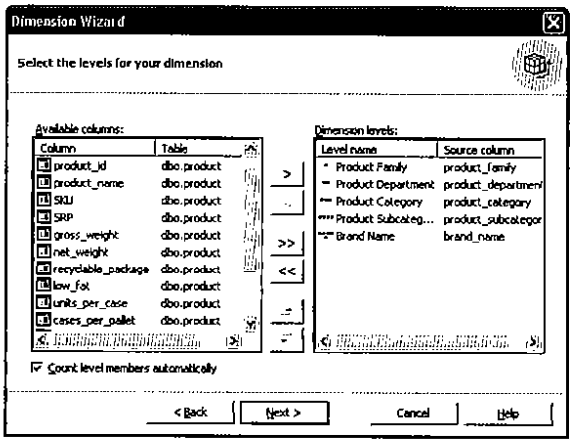


Obr. 4.35 Výběr tabulek pro dimenze Produkty

S výjimkou úrovně dimenze *product_name*, která je z tabulky ***product***, ostatní dimenze

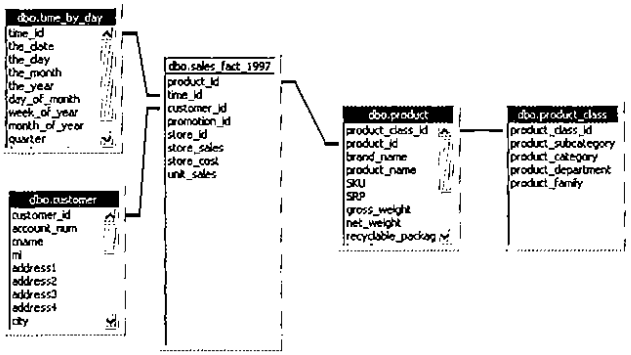
- *product_family*
- *product_department*
- *product_category* a *product_subcategory*

jsou z tabulky ***product_class***.



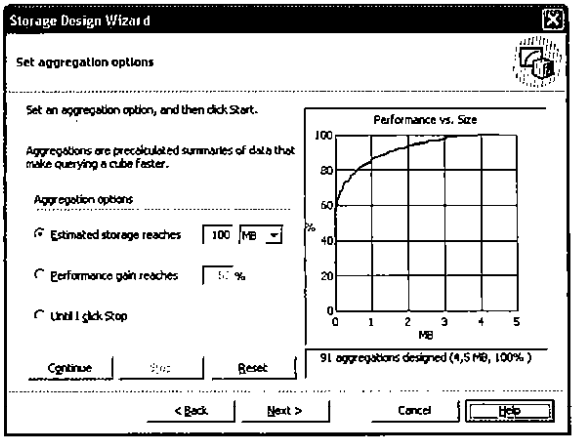
Obr. 4.36 Návrh hierarchie produktové dimenze

Pojmenováním dimenze Produkty jsme ukončili návrh dimenzí a budeme pokračovat dalšími kroky Cube Wizardu, kde pojmenujeme vytvářenou krychli, například Analýza1. Po ukončení návrhu krychle nás Analysis Manager přepne do utility **Cube Editor**, kde si na záložce Schéma můžeme prohlédnou topologii vytvořené krychle.



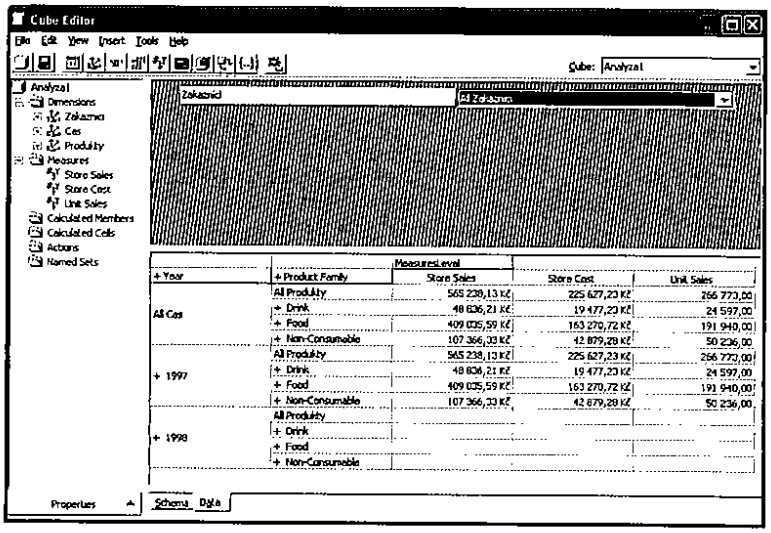
Obr. 4.37 Cube Editor

Po aktivaci tlačítka **Process Cube** v nástroji Cube Editor nám nabídne pomocnou ruku další průvodce – **Storage Design Wizard**, který nám pomůže s návrhem úložiště pro krychli. Hlavně u vícerozměrných velkých krychlí je důležité stanovit optimum mezi prostorem, který naše krychle zabere na disku, a rychlostí výpočtu a přístupu k údajům. Průvodce nám nabídne tři typy úložišť MOLAP, ROLAP a HOLAP. Zvolíme například úložiště údajů typu MOLAP a ve spolupráci s průvodcem vypočítáme optimalizovanou velikost úložiště. Naše krychle zabere jen 4,4 megabajtů.



Obr. 4.38 Optimalizace úložiště pro krychli

Po stisku tlačítka Continue jdeme do finále a zahájíme výpočet krychle. Jak jsme už uvedli, tento proces může trvat i několik hodin. O jeho průběhu jsme informováni v přehledném dialogu. Proces je náročný na paměť, proto před jeho odstartováním doporučujeme ukončit všechny ostatní aplikace. Po ukončení výpočtu máme v záložce Data utility Cube Editor skutečné údaje, které můžeme metodou drag-and-drop libovolně v tabulce přeskupovat a organizovat. Můžeme se zavrtávat stále hlouběji do jednotlivých dimenzí, nebo naopak zkoumat jen globální souhrny údajů.



Obr. 4.39 Výsledek analýzy

Vytvoření „reálné" krychle OLAP

V předcházejícím příkladě jsme vytvářeli cvičnou krychli OLAP, která byla koncipována pro seznámení s jednotlivými variantami průvodců vytvořením dimenzí, tedy pro schéma star a schéma snowflake. Tento příklad bude z reálné praxe. Stále pracujeme s cvičnou databází.

Scénář cvičného příkladu

Představme si, že jsme fiktivní databázový administrátor (DBA) fiktivní firmy FoodMart a právě jsme se vrátili z porady od finančního ředitele. Předmětem porady bylo téma ve firmách celkem obvyklé, řešení vysokých nákladů. Řešení však musí předcházet podrobná analýza výdajů. Potřebujeme sledovat výdaje pro jednotlivá oddělení, obchody seskupené geograficky v časovém horizontu.

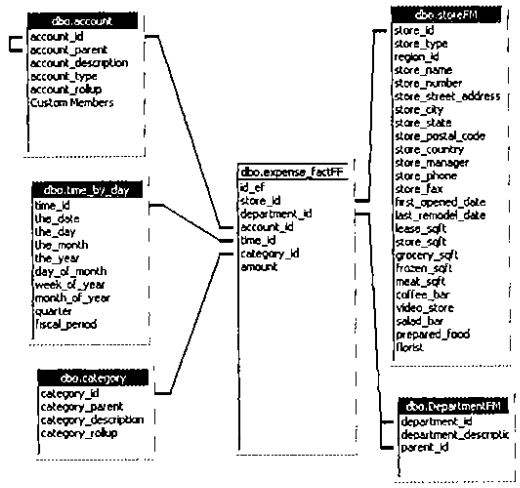
Na první pohled by se mohl zdát tento příklad pro návrh krychle OLAP podobný tomu předcházejícímu, ale podobá se mu skutečně jen do té míry jak se sobě podobají různé krychle OLAP. Jednak topologie krychle bude jiná, výchozí situace budou jiné a ukážeme i některé nové postupy, například použití editoru dimenzí vytvořením dimenzí typu Parent – Child, dimenze s nepravidelnou hierarchickou strukturou a podobně. Předcházející příklad by se dal charakterizovat jako přímočará „sterilní" cvičná aplikace, tento příklad se bude navzdory použití cvičné databáze o mnoho více přibližovat reálné praxi.

Rozbor řešení

Předpokládejme standardní situaci po nainstalování Analytických služeb MS SQL Serveru. Cvičnou analytickou databázi FoodMart 2000 sice máme nainstalovanou, ale pro názornost začneme vytvářet novou analytickou databázi s názvem například FoodMart1. Jako zdroj údajů ve složce DataSources zadáme relační databázi OLTP FoodMart. Máme dvě možnosti, které byly probrány v předcházejících kapitolách. Buď pomocí DTS přeneseme obsah souborové databáze Access *FoodMart 2000.mdb* (nachází se v adresáři *C:\Program Files\Microsoft Analysis Services\Samples*) do databáze MS SQL Serveru, nebo se přes rozhraní ODBC připojíme přímo k souborové databázi *FoodMart 2000.mdb*. V tomto příkladě budeme navrhovat nejdříve dimenze a až potom krychli.

Konceptuální model řešení

Kdybychom navrhli vhodné schéma multidimenzionální databáze skládající se z tabulky faktů EXPENSE_FACTS z dimenzí SCENAR, DEPARTMENT, STORES, ACCOUNTS a TIME, přičemž bychom nehleděli na momentální strukturu tabulek v databázi FoodMart 2000, ale jen na literu zadání, schéma by bylo:



Obr. 4.40 Hvězdicové schéma analytické databáze

Logický model řešení

Kdybychom chtěli vytvořit návrh multidimenzionální databáze na základě konceptuálního návrhu, tak narazíme na několik problémů, které ve zkratce vyjmenujeme.

Tabulka faktů EXPENSE_FACTS

Sloupec	Typ
store_id	Long Integer
account_id	Long Integer
exp_date	Date/Time
Time_id	Long Integer
category_id	Text
currency_id	Long Integer
amount	Currency

V tabulce faktů není žádný odkaz na dimenzi DEPARTMENT, úplně v ní chybí sloupec department_id. Budeme ho muset doplnit na základě znalosti organizační struktury firmy. Dalším potenciálním problémem je skutečnost, že suma je uvedena v měně té země, kde se příslušný obchod nachází, v našem případě v Amerických dolarech (USD), Kanadských dolarech (CAD) a Mexických Pesos. Kurs kanadského dolaru a mexického pesos je navíc klouzavý a závisí na konkrétním období. Proto budeme muset kurs přepočítávat pomocí tabulky CURRENCY na základě sloupce `currency_id`.

Dimenze SCENAR

Dimenzi vytvoříme na základě tabulky CATEGORY.

Sloupec	Typ
category_id	Text
category_parent	Text
category_description	Text
category_rollup	Text

Dimenze bude mít jen jednu úroveň.

Scenar
- category_description

Dimenze DEPARTMENT

Dimenzi vytvoříme na základě tabulky DEPARTMENT.

Sloupec	Typ
department_id	Long Integer
department_description	Text

```
Department
  • department_description
```

Původní tabulka však neobsahuje hierarchickou úroveň oddělení firmy, proto do tabulky pro tuto dimenzi musíme přidat sloupec `parent_id` a implementovat hierarchickou strukturu.

Dimenze STORES

Dimenzi vytvoříme na základě tabulky STORE.

Sloupec	Typ
store_id	Long Integer
store_name	Text
store_city	Text
store_state	Text
store_country	Text
...	...

(V tabulce jsou vypsány jen sloupce, které použijeme pro vytvoření dimenze STORES.)

```
Stores
  • store_country
  • • store_state
  • • • store_city
  • • • • store_name
```

V tomto případě není zdánlivě žádný problém, všechny databázové struktury na sebe ideálně navazují. Potíže jsou však s něčím jiným. Tato geografická dimenze je napasovaná na územně správní členění Spojených států amerických. Tedy jedna země (USA) je rozčleněná na 49 států a v těchto státech se nacházejí města a v nich prodejny. Ale svět nekončí na hranicích USA (i když přesvědčit o tom některé Američany není zrovna jednoduché ☺). Kdybychom tuto strukturu chtěli aplikovat například na Francii nebo jinou evropskou zemi, máme úplně zbytečnou jednu úroveň geografické dimenze. Členění na svazové státy se v Evropě příliš nevyužívá. Takže by bylo výhodné úroveň STATE dimenze STORES podle potřeby skrývat.

Dimenze ACCOUNTS

Dimenzi vytvoříme na základě tabulky ACCOUNT.

Sloupec	Typ
account_id	Long Integer
account_parent	Long Integer
account_description	Text
account_type	Text
account_rollup	Text
Custom Members	Text

Tabulka account má implementovanou hierarchickou strukturu, takže s její hierarchií při návrhu dimenze typu parent – child problémy nebudou. Budeme však potřebovat definovat sumární operace.

Dimenze TIME

Časovou dimenzi vytvoříme na základě tabulky TIME_BY_DAY.

Všechny dosud naznačené problémy budeme muset řešit při přípravě údajů a návrhu multidimenzionální databáze.

Návrh dimenzí

Budeme postupně navrhovat dimenze SCENAR, DEPARTMENT, STORES, ACCOUNTS a TIME.

Vytvoření dimenze SCENAR (star schema)

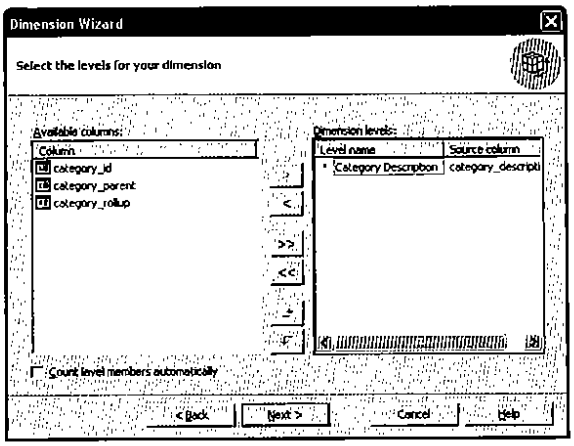
Pomocí dimenze *Scénář* můžeme vybrat jeden ze „scénářů" pro omezení údajů v analytické OLAP databázi. Dimenzi vytvoříme na základě tabulky Category, přičemž podle jejího obsahu do úvahy připadají čtyři scénáře:

- Current Year's Actuals
- Adjustment for Budget input
- Current Year's Budget
- Forecast

Dimenzi SCENAR vytvoříme standardním postupem pomocí průvodce vytvořením dimenze. Průvodce aktivujeme pomocí kontextového menu položkou *„Shared Dimension – New Dimension – Wizard"*. V prvním kroku průvodce vybereme volbu *„Star Schema: A single dimension table"*. Pokračujeme výběrem tabulky pro navrhovanou dimenzi. Bude to tabulka Category. V tomto kroku si můžeme pro zajímavost prohlédnout její obsah, tabulka má jen čtyři záznamy.

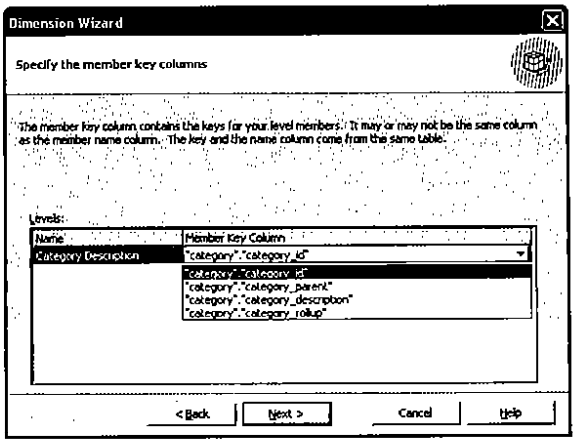
```
category_id     category_parent category_description          category_rollup
--------------- --------------- ------------------------------ ---------------
ACTUAL          NULL            Current Year's Actuals         NULL
ADJUSTMENT      NULL            Adjustment for Budget input    NULL
BUDGET          NULL            Current Year's Budget          NULL
FORECAST        NULL            Forecast                       NULL
```

Dimenze *Scénář* bude mít jen jednu hierarchickou úroveň `category_description`. Vybereme ji pomocí dvojitého poklepání v levém sloupci dialogu. Pokračujeme tlačítkem *„Next"*.



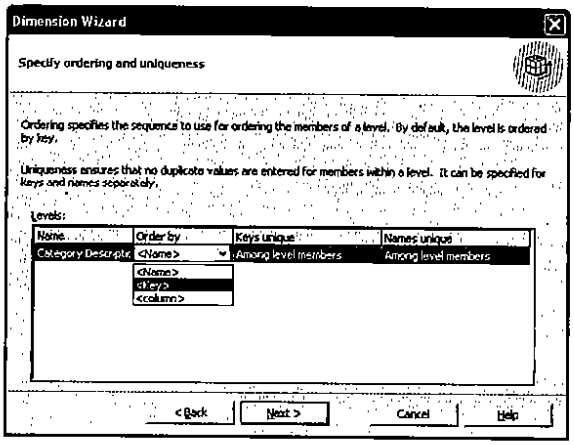
Obr. 4.41 Vytvoření jednoúrovňové dimenze Kategorie

Následuje dialog pro výběr klíčových sloupců. Rozvineme položky sloupce „Member Key Column" a vybereme příslušné sloupce. V našem případě jsme logicky jako klíčový vybrali sloupec CATEGORY.CATEGORY_ID, tedy unikátní identifikátor tabulky Category.



Obr. 4.42 Nastavení Member Key Column

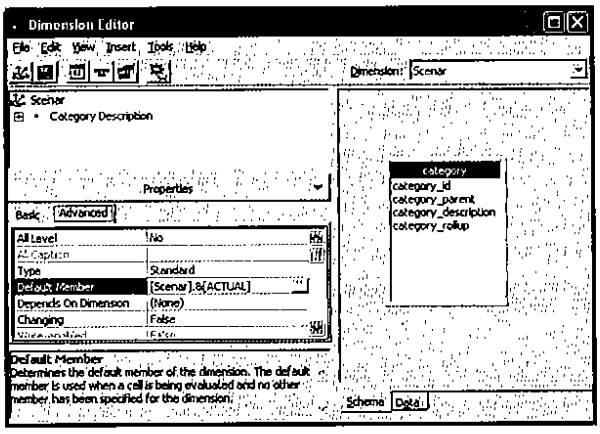
Uvedeným postupem zajistíme, že se jednotlivé položky úrovně dimenze budou zobrazovat v popisné textové podobě (`Current Year's Actuals`, `Adjustment for Budget Input`, `Current Year's Budget`, `Forecast`), ale pracovat se bude ve skutečnosti s hodnotou sloupce CATEGORY_ID. Největšího dotazovacího výkonu totiž dosáhneme, když položky úrovní dimenze budou jednoduché, krátké, nejlépe numerické hodnoty. A naopak při analýzách se nám z hlediska uživatelského komfortu a přehlednosti bude nejlépe pracovat s názornými textovými popisy. V následujícím dialogu specifikujeme, že položky této dimenze se budou třídit podle sloupce CETEGORY_ID, ne podle sloupce CATEGORY_DESCRIPTION. Takže v následujícím dialogu označíme volbu *„Ordering and uniqueness of members"* a po stisknutí tlačítka Next nastavíme třídění podle hodnoty klíče, tedy ve sloupci *„Order by"* z možností <Name>,<Key>,<Column> vybereme položku <Key>.



Obr. 4.43 Nastavení třídění

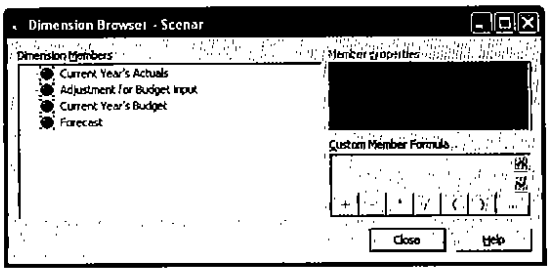
Dimenzi pojmenujeme jako „Scénář" a tlačítkem *„Finish"* ukončíme její předběžný návrh. Tímto úkonem se dostáváme do dialogu nástroje *„Dimension Editor"*. V hlavním menu editoru dimenzí zaškrtneme položku *„Properties"*, čímž se nám v levé části pracovní obrazovky zobrazí okno *Properties* s dvěma záložkami *„Basic"* a *„Advanced"*. Budeme pracovat se záložkou *„Advanced"*.

Vzhledem k jednoúrovňové hierarchické struktuře dimenze vytvořené na základě úplně jednoduché databázové tabulky se čtyřmi sloupci, která obsahuje čtyři záznamy, není potřebná speciální systémem generovaná nejvyšší hierarchická úroveň ALL pro tuto dimenzi. Proto nastavíme hodnotu položky *„All Level"* na NO.



Obr. 4.44 Zrušení úrovně ALL a nastavení „Default Member"

V tomto případě je výhodné nastavit implicitní hodnotu výběru scénáře. Proto parametr *„Default Member"* nastavíme na hodnotu *„Current Year's Actuals"*. Nakonec uložíme změny v návrhu dimenze a práci s editorem dimenze můžeme ukončit. Návrh dimenze můžeme na závěr zkontrolovat tak, že v kontextovém menu *„Shared Dimensions"* vybereme položky *„Browse Dimension Data"*.



Obr. 4.45 Zrušení úrovně ALL a nastavení „Default Member"

Všimněme si, že položky dimenze nejsou setříděny v abecedním pořadí podle textu položky, ale když si prohlédneme obsah tabulky Category, na jejímž základě je dimenze vytvořena,

```
category_id     category_description
--------------- ------------------------------
ACTUAL          Current Year's Actuals
ADJUSTMENT      Adjustment for Budget input
BUDGET          Current Year's Budget
FORECAST        Forecast
```

zjistíme, že jsou uspořádány přesně tak, jak jsme stanovili, tedy podle klíče `category_id`.

Vytvoření dimenze DEPARTMENT (dimenze typu Parent – Child)

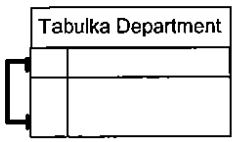
Dimenze Department je typickým příkladem dimenze vyjadřující hierarchickou organizační strukturu firmy. Když si necháme zobrazit obsah databázové tabulky Department,

```
department_id department_description
------------- ----------------------------------------
1             HQ General Management
2             HQ Information Systems
3             HQ Marketing
4             HQ Human Resources
5             HQ Finance and Accounting
11            Store Management
14            Store Information Systems
15            Store Permanent Checkers
16            Store Temporary Checkers
17            Store Permanent Stockers
18            Store Temporary Stockers
19            Store Permanent Butchers
```

zjistíme, že tabulka Department má v původní databázi FoodMart dva sloupce.

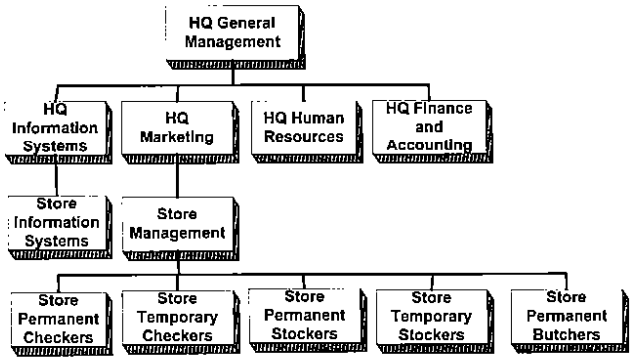
Jméno sloupce	Datový typ
department_id	int
department_description	nvarchar(100)

Aby s hierarchickou strukturou dokázal pracovat i databázový a analytický server musíme v etapě ETL doplnit do tabulky unární relaci, tedy relační vazbu tabulky na sebe samu. Nejvyšší hierarchická úroveň je v takové vazbě tvořena jedním prvkem, který je svázaný s prvky o jednu úroveň níže. Tyto prvky mohou mít vazbu na další prvky z nižší úrovně. Každý prvek kromě prvku na nejvyšší úrovni má tedy vazbu na jeden prvek z vyšší úrovně a žádnou, jednu nebo více vazeb na prvky nižší úrovně. Pomocí unární relace se často vyjadřuje hierarchický vztah **nadřízený – podřízený**. Sloupec tabulky může obsahovat vazbu na primární klíč jeho nejbližšího nadřízeného.



Obr. 4.46 Unární relace

Jedno políčko ve vzpomínaném sloupci v některém řádku tabulky je obvykle prázdné. Jedná se o prvek na nejvyšší hierarchické úrovni. V případě databáze pracovníků to bude nejvyšší nadřízený, v případě organizační struktury firmy to bude generální ředitelství a podobně. Hierarchická struktura oddělení fiktivní firmy FoodMart by mohla být taková.



Obr. 4.47 Hierarchická struktura oddělení fiktivní firmy FoodMart

Hierarchickou strukturu implementujeme do databáze FoodMart 2000 (do relační databáze, ne analytické) tak, že na základě databázové tabulky Department vytvoříme novou tabulku DepartmentFM. Tabulka DepartmentFM bude mít přidaný další sloupec Parent_id, s jehož pomocí vyjádříme hierarchickou strukturu oddělení.

Jméno sloupce	Datový typ
department_id	int
department_description	nvarchar(100)
parent_id	int

Hierarchická struktura bude implementovaná následovně:

```
department_id department_description         parent_id
------------- ------------------------------------------
1             HQ General Management
2             HQ Information Systems          1
3             HQ Marketing                    1
4             HQ Human Resources              1
5             HQ Finance and Accounting       1
11            Store Management                3
14            Store Information Systems       2
15            Store Permanent Checkers        11
16            Store Temporary Checkers        11
17            Store Permanent Stockers        11
18            Store Temporary Stockers        11
19            Store Permanent Butchers        11
```

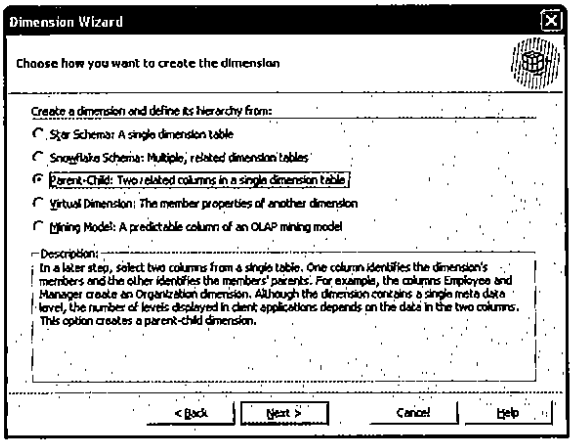
Novou tabulku vytvoříme pomocí příkazu jazyka SQL

```
CREATE TABLE DepartmentFM
(
    [department_id] [int] NOT NULL,
    [department_description] [nvarchar] (100) COLLATE Czech_CI_AS NULL,
    [parent_id] [int]
) ON [PRIMARY]
```

a naplníme pomocí sekvence příkazů:

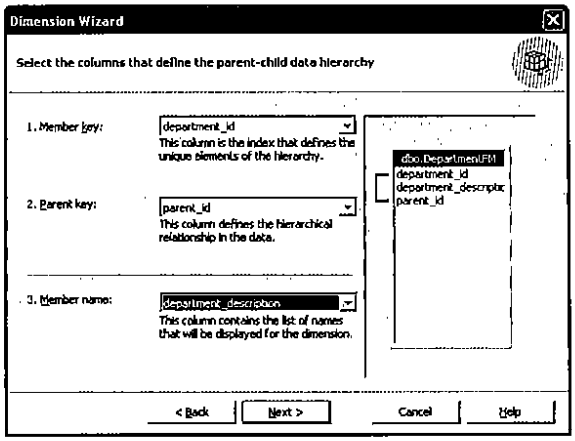
```
INSERT INTO DepartmentFM VALUES( 1,'HQ General Management', NULL );
INSERT INTO DepartmentFM VALUES( 2,'HQ Information Systems', 1);
INSERT INTO DepartmentFM VALUES( 3,'HQ Marketing', 1);
INSERT INTO DepartmentFM VALUES( 4,'HQ Human Resources', 1);
INSERT INTO DepartmentFM VALUES( 5,'HQ Finance and Accounting', 1);
INSERT INTO DepartmentFM VALUES( 11,'Store Management', 3);
INSERT INTO DepartmentFM VALUES( 14,'Store Information Systems', 2);
INSERT INTO DepartmentFM VALUES( 15,'Store Permanent Checkers', 11);
INSERT INTO DepartmentFM VALUES( 16,'Store Temporary Checkers', 11);
INSERT INTO DepartmentFM VALUES( 17,'Store Permanent Stockers', 11);
INSERT INTO DepartmentFM VALUES( 18,'Store Temporary Stockers', 11);
INSERT INTO DepartmentFM VALUES( 19,'Store Permanent Butchers', 11);
```

Dimenzi Department vytvoříme pomocí průvodce vytvořením dimenze (kontextové menu *„Shared Dimension – New Dimension – Wizard"*). V prvním kroku průvodce vybereme volbu *„Parent – Child: Two related columns in a single dimension table"*.



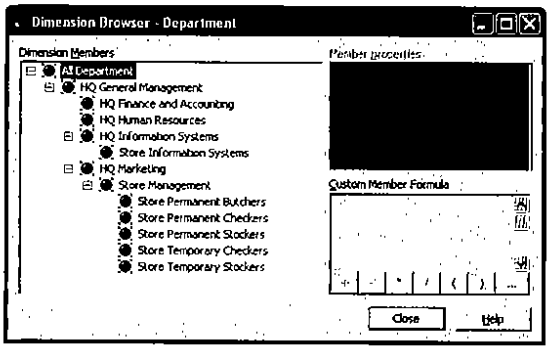
Obr. 4.48 Vytvoření dimenze typu Parent – Child

Pokračujeme výběrem tabulky pro navrhovanou dimenzi. Bude to tabulka Departments. V následujícím dialogu průvodce definujeme hierarchii typu parent – child. V našem případě je tato vazba definovaná pomocí sloupců `department_id` a `parent_id`.



Obr. 4.49 Definování sloupců pro dimenzi Parent – Child

Po dokončení návrhu si můžeme prohlédnout hierarchickou úroveň dimenze Department.



Obr. 4.50 Hierarchie dimenze Department

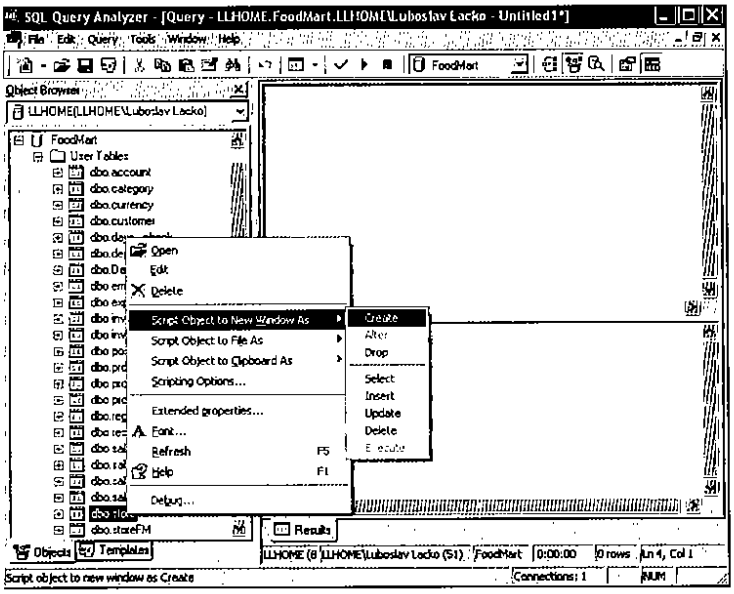
Vytvoření dimenze STORES (star schema – dimenze s nepravidelnou hierarchií)

V některých případech není možné hierarchickou strukturu dimenze definovat pro každý případ stejně. Typickým příkladem je geografická dimenze. Územněsprávní členění v USA je Country – State – City. V Evropě sice některé státy mají krajské uspořádání, ale používá se jen hierarchie Country – City. Proto v takovém případě je potřebné úroveň State skrýt. V tabulce stores databáze FoodMart 2000 (vypsali jsme jen sloupce `store_id`, `store_country`, `store_state`, `store_city` a `store_name`) jsou „napasované" údaje na územněsprávní členění USA, a tedy i pro Mexiko jsou uvedené „svazové státy".

store_id	store_country	store_state	store_city	store_name
0	USA	CA	Alameda	HQ
1	Mexico	Guerrero	Acapulco	Store 1
2	USA	WA	Bellingham	Store 2
3	USA	WA	Bremerton	Store 3
4	Mexico	Zacatecas	Camacho	Store 4
5	Mexico	Jalisco	Guadalajara	Store 5
6	USA	CA	Beverly Hills	Store 6
7	USA	CA	Los Angeles	Store 7
8	Mexico	Yucatan	Merida	Store 8
9	Mexico	DF	Mexico City	Store 9
10	Mexico	Veracruz	Orizaba	Store 10
11	USA	OR	Portland	Store 11
12	Mexico	Zacatecas	Hidalgo	Store 12
13	USA	OR	Salem	Store 13
14	USA	CA	San Francisco	Store 14
15	USA	WA	Seattle	Store 15

16	USA	WA	Spokane	Store 16
17	USA	WA	Tacoma	Store 17
18	Mexico	Zacatecas	Hidalgo	Store 18
19	Canada	BC	Vancouver	Store 19
20	Canada	BC	Victoria	Store 20
21	Mexico	DF	San Andres	Store 21
22	USA	WA	Walla Walla	Store 22
23	USA	WA	Yakima	Store 23
24	USA	CA	San Diego	Store 24

Podobnou strukturu by určitě Američané „napasovali" i na územněsprávní členění Čech a Slovenska, ale na Slovensku by asi nikdo (zatím) neřekl, že má svoje prodejny v Žilinském vyšším územním celku. Abychom situaci v databázi FoodMart zreálnili, tak vytvoříme v databázi FoodMart novou tabulku StoreFM se stejnou strukturou. Pro tento účel můžeme použít DTS nebo v konzolové aplikaci Query Analyser získat skript pro vytvoření tabulky Store.



Obr. 4.51 Získání kódu příkazu CREAT TABLE pro vytvoření tabulky s identickou strukturou, jako má tabulka Store

Ve skriptu SQL upravíme název tabulky na StoreFM.

```
CREATE TABLE [storeFM] (
        [store_id] [int] NOT NULL ,
        [store_type] [nvarchar] (255) COLLATE Czech_CI_AS NULL ,
        [region_id] [int] NULL ,
        ...
        [florist] [bit] NOT NULL
) ON [PRIMARY]
```

Tabulku po vytvoření naplníme údaji z původní tabulky příkazem

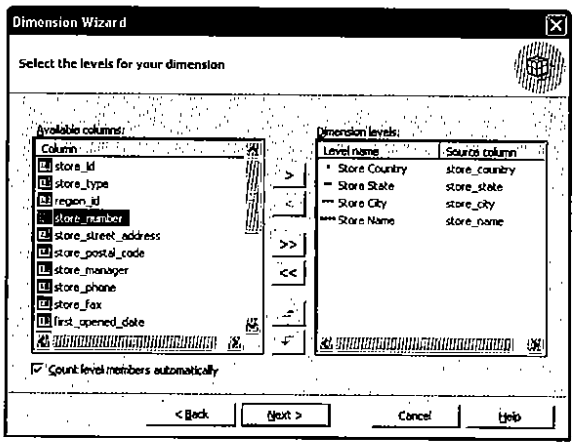
```
INSERT INTO storeFM SELECT * FROM store;
```

a nakonec pro všechny mexické záznamy nastavíme hodnoty sloupce store_state na hodnotu 'Mexico' příkazem

```
UPDATE storeFM SET store_state = 'Mexico' WHERE store_country = 'Mexico';
```

Po těchto úpravách zdrojové databáze můžeme přistoupit k samotnému návrhu dimenze. Pro dimenzi Stores vybereme typ dimenze *Star Schema*, vytvoříme ji na základě databázové tabulky StoresFM a navrhneme typickou geografickou hierarchickou úroveň:

- Store Country
- • Store State
- • • Store City
- • • • Store Name

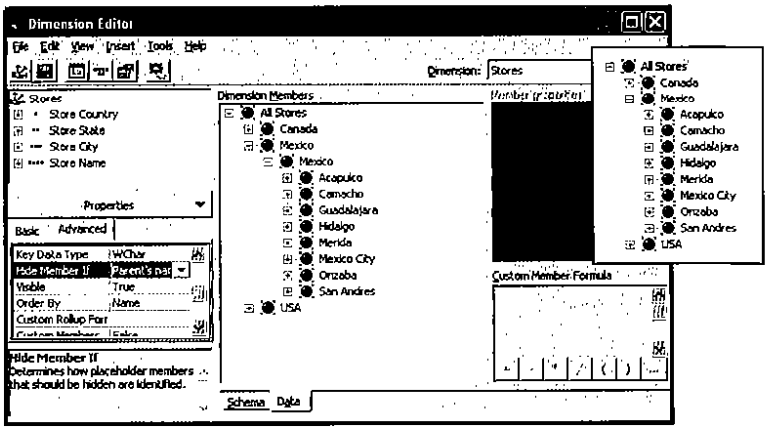


Obr. 4.52 Hierarchie dimenze Stores

Podobně jako u dimenze Scénář nastavíme pro úroveň Store Name klíčový sloupec store_id. S numerickými hodnotami se lépe pracuje analytickému serveru a analytik, případně manažer zase uvítá název prodejny v „lidské řeči".

Skrytí některé úrovně dimenze

V nástroji Dimension Editor ve složce *Advanced* nastavíme pro úroveň dimenze *Store State* parametr *Hide Member if* na hodnotu *Parent's Name*. To znamená, že tato úroveň dimenze se skryje v případě, že hodnoty příslušného sloupce databázové tabulky, v našem případě sloupce *Store State*, budou mít hodnotu NULL. V tabulce StoreFM jsou všechny záznamy pro Mexico. Když porovnáme hierarchii před úpravou, pro Mexico se tam vyskytuje navíc hierarchická úroveň svazového státu, která má také hodnotu „Mexico", takže pošťák by takovou adresu v pohodě našel, ale v multidimenzionální databázi by to byla zbytečná a nefunkční hierarchická úroveň. Když si prohlédneme hierarchii úrovní pro USA, zjistíme, že tam státy zůstaly.



Obr. 4.53 Skrytí úrovně dimenze store_state; vpravo je hierarchická úroveň po skrytí

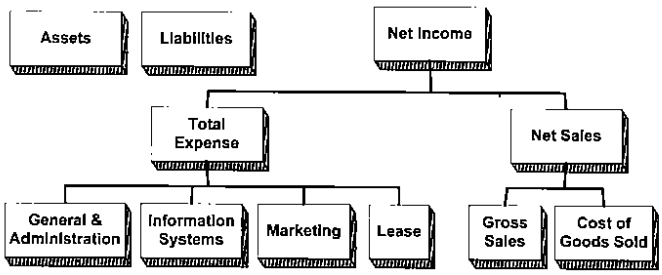
Vytvoření dimenze Accounts (dimenze typu Custom Rollups)

V relačních databázích se velmi často používají agregační funkce. Agregační funkce operují nad množinou řádků, přičemž vracejí jeden výsledek pro celou vstupní množinu údajů. Tyto funkce pomocí matematických a statistických operací zpracovávají agregované hodnoty v celých sloupcích. Nejpoužívanější agregační funkce jsou COUNT, SUM, MIN a MAX.

Dimenzi Accounts vytvoříme jako typ *Parent Child* na základě databázové tabulky Account. Tentokrát nemusíme v původní databázi nic upravovat, neboť tabulka Account má implementovanou hierarchickou strukturu pomocí sloupců account_id a account_parent.

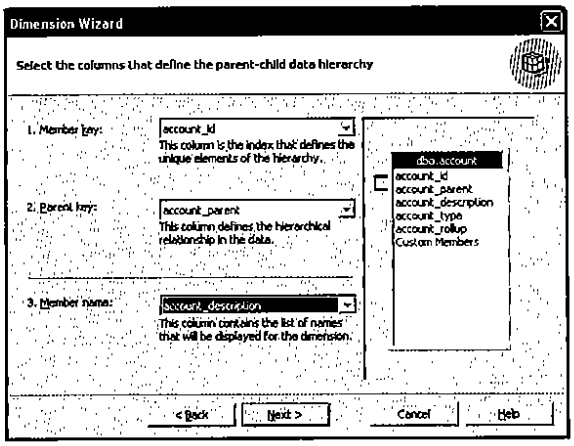
```
account_id  account_parent account_description    account_type account_rollup Custom Members
----------- -------------- ---------------------- ------------ ------------- --------------
1000        NULL           Assets                 Asset        -             NULL
2000        NULL           Liabilities            Liability    -             NULL
3000        5000           Net Sales              Income       +             NULL
3100        3000           Gross Sales            Income       +     LookUpCube("[Sales]",
                                                                  "(Measures.[Store Sales],
                                                          "+time.currentmember.UniqueName+",
                                                       "Store.currentmember.UniqueName+")")
3200        3000           Cost of Goods Sold     Income       -             NULL
4000        5000           Total Expense          Expense      -             NULL
4100        4000           General & Administrati Expense      +             NULL
4200        4000           Information Systems    Expense      +             NULL
4300        4000           Marketing              Expense      +             NULL
4400        4000           Lease                  Expense      +             NULL
5000        NULL           Net Income             Income       +             NULL
```

Pravděpodobně názornější bude blokové schéma hierarchie.



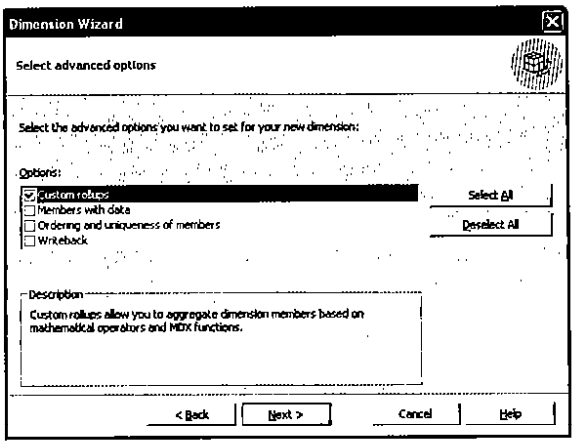
Obr. 4.54 Hierarchická struktura parent – child v databázové tabulce Account

Pokračujeme v návrhu dimenze. Vazbu parent – child definujeme pomocí sloupců Member key: `account_id` a Parent key" `Account_parent`.



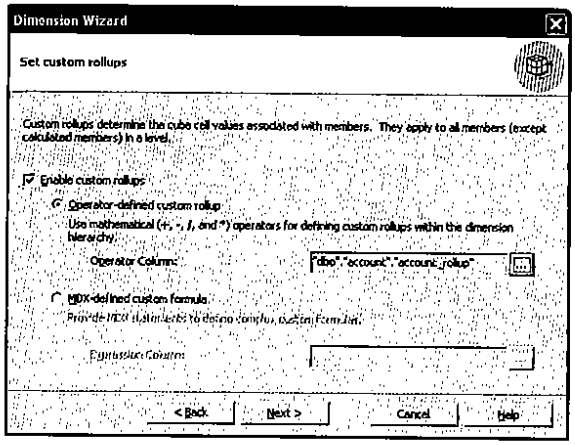
Obr. 4.55 Dimenze Accounts typu parent – child

Důležitý je dialog „*Select advanced options*", ve kterém zaškrtneme volbu „Custom rollups", čímž povolíme agregování úrovní dimenze na základě matematických operátorů a funkcí jazyka MDX.



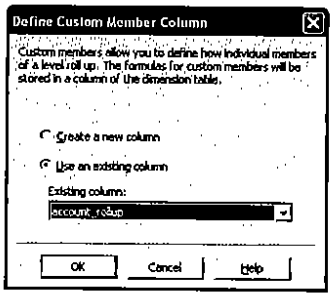
Obr. 4.56 Dimenze Accounts typu parent – child

Návrh dimenze pokračuje dialogem „*Set Custom Rollups*", kde vybereme, zda budeme používat matematické operátory, nebo funkce v jazyce MDX.



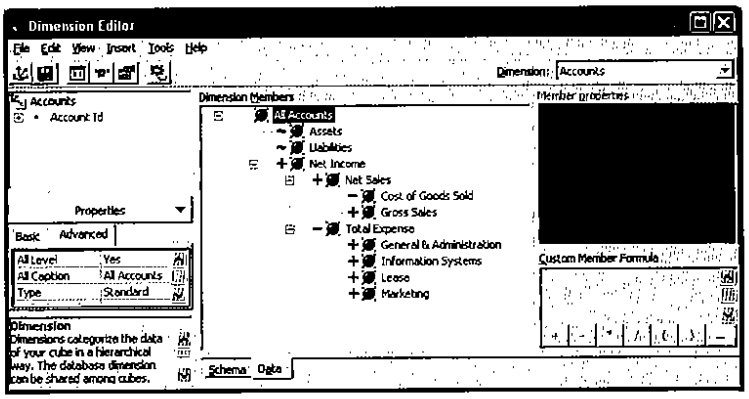
Obr. 4.57 Dimenze Accounts typu parent – child

Pomocí tlačítka (...) zobrazíme dialog pro definování uživatelského sloupce. V našem příkladě budeme používat matematické operátory nad existujícím sloupcem, konkrétně nad sloupcem account_rollup.



Obr. 4.58 Dimenze Accounts typu parent – child

V záložce Data Dimension Editoru si můžeme prohlédnout hierarchii dimenzí a operátory.



Obr. 4.59 Dimenze Accounts typu parent – child

Vytvoření dimenze Time (časová dimenze)

Časová dimenze bude prakticky totožná s časovou dimenzí jako v předcházejícím příkladě. I postup návrhu je prakticky totožný. Proto její vytvoření popíšeme jen heslovitě. Vybereme typ schématu „*Star Schema*", tabulku *time_by_day* a v dialogu pro výběr typu dimenze označíme volbu „*Time dimension*". Úrovně dimenze navrhneme Year – Quarter – Month.

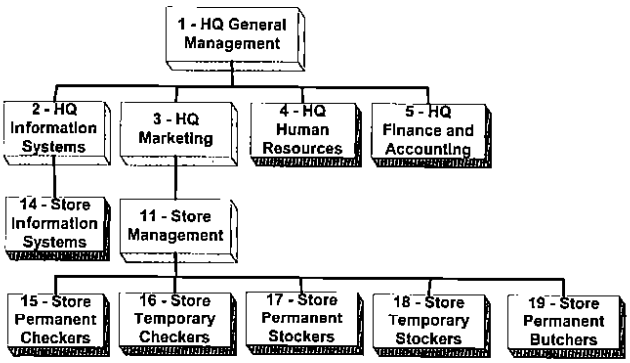
Návrh tabulky faktů

Zatím máme navržené dimenze SCENAR, DEPARTMENT, STORES, ACCOUNTS a TIME. Jako tabulku faktů vybereme tabulku expense_factFM. Přestože taková tabulka ve cvičné databázi FoodMart 2000 není, předpokládáme, že v reálné aplikaci bychom ji mít měli. V databázi FoodMart 2000 máme tabulku expense_fact, která má následující strukturu:

Sloupec	Typ
store_id	Long Integer
account_id	Long Integer
exp_date	Date/Time
time_id	Long Integer
category_id	Text
currency_id	Long Integer
amount	Currency

Tuto tabulku ale přímo bez úprav použít nemůžeme, když se podíváme na její strukturu, tak zjistíme, že sice obsahuje jednu míru (amount) a cizí klíče pro čtyři tabulky dimenzí

SCENAR, STORES, ACCOUNTS a TIME, neobsahuje ale odkaz do tabulky dimenze DEPARTMENT. Tento odkaz jednoduše neznáme, a protože se jedná o cvičnou databázi, doplníme ho pomocí převodní tabulky, kterou vytvoříme na základě znalosti organizační struktury firmy. Firma má oddělení 1, 2, 3, 4, 5, 11, 14, 15, 16, 17, 18, 19, přičemž jejich hierarchie je následující:



Obr. 4.60 Hierarchická struktura oddělení fiktivní firmy FoodMart

Oddělení 1, 2, 3 a 11 tedy nejsou na konci hierarchické struktury. Takže v převodní tabulce budou jen oddělení 4, 5, 14, 15, 16, 17, 18, 19.

Převodní tabulka v našem případě bude například takováto:

SHORT_ID	DEPARTMENT_ID
0	4
1	5
2	14
3	15
4	16
5	17
6	18
7	19
8	4
9	5
10	14
11	15
12	16
13	17

store_id	department_id
14	18
15	19
16	4
17	5
18	14
19	15
20	16
21	17
22	18
23	19
24	4

Vytvoříme ji příkazem

```
CREATE TABLE store_to_dep
(
        store_id int NOT NULL ,
        department_id int NOT NULL
);
```

a naplníme sekvencí příkazů

```
INSERT INTO store_to_dep VALUES(0,4);
INSERT INTO store_to_dep VALUES(1,5);
INSERT INTO store_to_dep VALUES(2,14);
...
INSERT INTO store_to_dep VALUES(23,19);
INSERT INTO store_to_dep VALUES(24,4);
```

Převodní tabulku máme vytvořenou a naplněnou, takže nám nic nebrání vytvořit a naplnit tabulku `expense_factFM`. Na základě příkazu pro vytvoření tabulky `expense_fact`

```
CREATE TABLE [expense_fact]
(
        [store_id] [int] NOT NULL ,
        [account_id] [int] NOT NULL ,
        [exp_date] [smalldatetime] NOT NULL ,
        [time_id] [int] NULL ,
        [category_id] [nvarchar] (15) COLLATE Slovak_CI_AS NULL ,
        [currency_id] [int] NULL ,
        [amount] [money] NULL
) ON [PRIMARY]
```

vytvoříme novou tabulku `expense_factFM`, do které doplníme sloupec `department_id` a pro pořádek i unikátní identifikátor `id_ef`. Tento identifikátor budeme při vkládání záznamů generovat automaticky inkrementálně.

```
CREATE TABLE [expense_factFM]
(
        [id_ef] [int] IDENTITY (1, 1) NOT FOR REPLICATION PRIMARY KEY,
        [store_id] [int] NOT NULL ,
        [department_id] [int] NOT NULL,
        [account_id] [int] NOT NULL ,
        [exp_date] [smalldatetime] NOT NULL ,
        [time_id] [int] NULL ,
        [category_id] [nvarchar] (15) COLLATE Slovak_CI_AS NULL ,
        [currency_id] [int] NULL ,
        [amount] [money] NULL
) ON [PRIMARY]
```

Tabulku po vytvoření naplníme údaji z původní tabulky, přičemž z převodní tabulky store_to_dep doplníme sloupec department_id. Pro názornost to udělám ve dvou krocích. V prvním kroku vypíšeme hodnoty, které budeme vkládat do nové tabulky z tabulek expense_fact a store_to_dep.

```
SELECT expense_fact.store_id, account_id, exp_date, time_id, category_id,
         currency_id, amount, store_to_dep.department_id
   FROM expense_fact, store_to_dep
    WHERE expense_fact.store_id = store_to_dep.store_id;
```

Výpis je přesně podle našich představ,

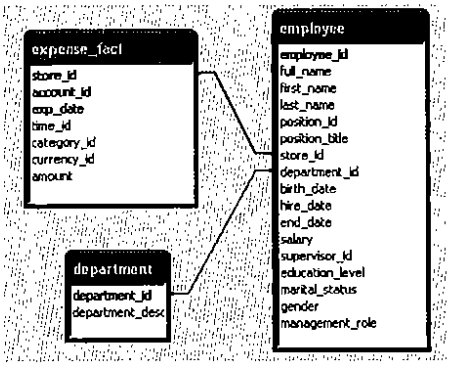
```
store_id    account_id  exp_date     time_id  category_id  currency_id amount   department_id
----------- ----------- ------------ -------- ------------ ----------- -------- -------------
0           4100        1997-01-01   367      ACTUAL       1           942.00   2
0           4100        1997-02-01   398      ACTUAL       1           942.00   2
0           4100        1997-03-01   426      ACTUAL       1           942.00   2
0           4100        1997-04-01   457      ACTUAL       1           942.00   2
```

takže můžeme odladěný příkaz SELECT doplnit do příkazu INSERT INTO.

```
INSERT INTO expense_factFM (store_id, account_id, exp_date, time_id, category_id,
                            currency_id, amount, department_id)
  SELECT expense_fact.store_id, account_id, exp_date, time_id, category_id,
         currency_id, amount, store_to_dep.department_id
   FROM expense_fact, store_to_dep
    WHERE expense_fact.store_id = store_to_dep.store_id;
```

!! Krátké odbočení, neboť ne vždy jsou věci tak jednoduché, jak se na první pohled jeví !!

Při podrobnějším pohledu na tabulky databáze FoodMart 2000, hlavně na tabulky expense_fact, department a employee se zdá, že předcházející postup byl nejen zbytečný, ale i nesprávný. Neboť sloupec department_id do tabulky expense_factFM můžeme jednoduše doplnit na základě spojovací tabulky employee, kde je vazba mezi store_id a department_id. Názorně to vidíme na následujícím schématu.



Obr. 4.61 Relační vazby pro doplnění cizího klíče na dimenzi DEPARTMENT.

Potom bychom mohli naplnit tabulku `expense_factFM` příkazem

```
INSERT INTO expense_factFM (store_id, account_id, exp_date, time_id, category_id,
                            currency_id, amount, department_id)
   SELECT expense_fact.store_id, account_id, exp_date, time_id, category_id,
          currency_id, amount, employee.department_id
     FROM expense_fact, employee
      WHERE expense_fact.store_id = employee.store_id;
```

Z hlediska databázové logiky je vše v úplném pořádku, ale po zamyšlení se nad faktem, že jako spojovací tabulka mezi tabulkami oddělení a obchodů by měla sloužit tabulka zaměstnanců si uvědomíme, že z hlediska aplikační logiky toto řešení nebude dobré. Svědčí o tom i hlášení databázového serveru po vykonání příkazu INSERT INTO ... SELECT.

```
110880 row(s) affected
```

Místo 2 400 záznamů se uložilo až 110 880 záznamů, což je nesmysl. Vysvětlení je jednoduché, stačí, když si necháme vypsat sloupce `store_id,department_id` z tabulky `employee`.

```
SELECT DISTINCT store_id,department_id FROM employee;

store_id    department_id
----------- -------------
0           1
0           2
0           3
0           4
0           5
....
....
24          11
24          14
```

```
24          15
24          16
24          17
24          18
24          19

(149 row(s) affected)
```

Tento „odstrašující" příklad jsme ukázali záměrně, abychom ukázali význam návaznosti struktury databázových struktur na aplikační logiku.

Úpravy tabulky faktů

Vraťme se však po krátkém odbočení k linii příkladu. Zdálo by se, že tabulku expense_factFM máme vytvořenou a upravenou tak, že ji můžeme použít jako tabulku faktů. Podívejme se ale pozorněji na její aktuální stav, tedy na údaje, které momentálně obsahuje.

```
id_ef  store_id  department_id account_id  exp_date    time category_id  currency_id amount
------ --------- ------------- ----------- ----------- ---- ------------ ----------- ------
1      0         2             4100        1997-01-01  367  ACTUAL       1           942.00
2      0         2             4100        1997-02-01  398  ACTUAL       1           942.00
3      0         2             4100        1997-03-01  426  ACTUAL       1           942.00
...
1823   18        17            4400        1998-11-01  1036 ACTUAL       2           780.00
1824   18        17            4400        1998-12-01  1066 ACTUAL       2           780.00
1825   19        18            4100        1997-01-01  367  ACTUAL       3           .0000
1826   19        18            4100        1997-02-01  398  ACTUAL       3           .0000
...
```

Všimněme si, že sloupec částka (amount) se vztahuje ke sloupci identifikátor druhu měny (currency_id). Aby to nebylo tak jednoduché, tabulka currency, do které ukazuje sloupec, vyjadřuje nejen kurs peněžních jednotek, tedy mexického pesos a kanadského dolaru vůči USD, ale i kursovní pohyby těchto měn v závislosti na datu.

```
currency_id date            currency        conversion_ratio
----------- --------------- --------------- --------------------
1           1997-01-01      USD             1.0000
...
2           1997-10-01      Mexican Peso    .1100
2           1997-11-01      Mexican Peso    .1200
...
3           1997-01-01      Canadian Dollar .7000
...
3           1997-04-01      Canadian Dollar .6900
...
3           1997-12-01      Canadian Dollar .6800
..
3           1998-12-01      Canadian Dollar .6700
```

Ve výpisu tabulky kursů jsme pro úsporu místa zobrazili jen hodnoty, kdy se kurz měnil. Takže kdybychom bezmyšlenkovitě ponechali tabulku expense_factFM v takovém stavu, v jakém je, výsledky by byly značně zkreslené, v případě mexického pesos by byla chyba

skoro jeden řád. I tuto úpravu, tedy vynásobení sumy aktuálním kursem, ukážeme rozloženou na dvě etapy. V první etapě vytvoříme dotaz SQL SELECT, s jehož pomocí vypíšeme sloupce `currency_id`, `amount`, `conversion_ratio`, a hlavně vypočítaný sloupec `amountUSD`, který vypočítáme jako součin sloupce `amount` z tabulky `expense_factFM` a aktuálního kursu `conversion_ratio` z tabulky `currency`. Vzorec pro výpočet tohoto sloupce potom bude

```
amountUSD = (amount * conversion_ratio)
```

Abychom byli přesní, uvedeme kompletní názvy sloupců v rámci databáze, tedy ve tvaru tabulka.sloupec.

```
amountUSD = (expense_factFM.amount * currency.conversion_ratio)
```

Podmínka pro definování vztahu mezi tabulkami bude:

```
expense_factFM.currency_id = currency.currency_id
```

Zůstává nám definovat časovou podmínku, abychom násobili aktuálním kursem podle data.

```
expense_factFM.exp_date = currency.date
```

Na základě uvedeného rozboru lehce sestavíme dotaz SQL SELECT pro výpočet výdajů přepočítaných aktuálním kursem na USD.

```
SELECT expense_factFM.currency_id, amount, conversion_ratio,
       (amount * conversion_ratio) AS amountUSD
  FROM expense_factFM, currency
  WHERE expense_factFM.currency_id = currency.currency_id AND
        expense_factFM.exp_date = currency.date;
```

Možná bude tento příkaz přehlednější po zavedení aliasů pro názvy tabulek.

```
expense_factFM AS EF, currency AS CU
```

Potom bude předcházející příkaz SELECT v tomto tvaru:

```
SELECT EF.currency_id, amount, conversion_ratio,
       (amount * conversion_ratio) AS amountUSD
  FROM expense_factFM AS EF, currency AS CU
  WHERE EF.currency_id = CU.currency_id AND
        EF.exp_date = CU.date;
```

Jako reakci databázového serveru na tento příkaz získáme výpis, ve kterém můžeme lehce zkontrolovat, že dotaz SQL byl zformulovaný dobře a že se skutečně vždy násobí správným kursem.

```
currency_id amount                conversion_ratio      amountUSD
----------- --------------------- --------------------- ------------
1           942.0000              1.0000                942.0000
...
2           .0000                 .1200                 .0000
2           .0000                 .1200                 .0000
```

```
2          743.6200          .1200          89.2344
2          743.6200          .1200          89.2344
2          743.6200          .1200          89.2344
2          743.6200          .1200          89.2344
2          743.6200          .1100          81.7982
2          743.6200          .1100          81.7982
...
```

Zůstává nám upravit tabulku `expense_factFM` tak, aby v sloupci byla hodnota v USD, a samozřejmě do sloupce `currency_id` uložíme hodnotu 1 jako odkaz na měnu USD.

Samozřejmě by to byl správný postup, ale kdybychom se po takovém přepočtu podívali na tabulku `expense_factFM` jako na tabulku faktů, tak zjistíme, že jsou tam minimálně dva sloupce `exp_date` a `currency_id`, které sloužily na kursovní přepočet úplně zbytečně.

```
id_ef  store_id  department_id account_id  exp_date    time category_id  currency_id amount
------ --------- ------------- ----------- ----------- ---- ------------ ----------- ------
1      0         2             4100        1997-01-01  367  ACTUAL       1           942.00
2      0         2             4100        1997-02-01  398  ACTUAL       1           942.00
3      0         2             4100        1997-03-01  426  ACTUAL       1           942.00
...
```

Z důvodu vyššího dotazovacího výkonu (samozřejmě ne v tomto konkrétním cvičném příkladě, kdy tabulka faktů obsahuje 2 400 záznamů, ale obecně) je výhodnější vytvořit novou tabulku faktů, ze které po etapě ETL zbytečné údaje jednoduše vynecháme. Nová tabulka `expense_factFF`, kterou vytvoříme na základě „pracovní" tabulky `expense_factFM`, bude obsahovat jen sloupce.

```
id_ef  store_id  department_id account_id   time category_id  amount
------ --------- ------------- -----------  ---- ------------ ------
1      0         2             4100         367  ACTUAL       942.00
2      0         2             4100         398  ACTUAL       942.00
3      0         2             4100         426  ACTUAL       942.00
```

Bude to míra `amount` a odkazy `store_id`, `department_id`, `account_id`, `time_id` a `category_id` na jednotlivé dimenze STORES, DEPARTMENT, ACCOUNTS, TIME a SCENAR. Postup pro vytvoření nové tabulky `expense_factFF` a její naplnění údaji z tabulky `expense_factFM` už známe.

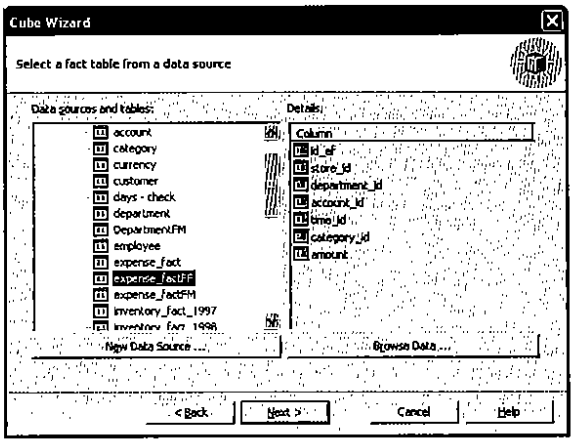
```
CREATE TABLE [expense_factFF]
(
        [id_ef] [int] PRIMARY KEY,
       [store_id] [int] NOT NULL ,
        [department_id] [int] NOT NULL,
       [account_id] [int] NOT NULL ,
        [time_id] [int] NULL ,
        [category_id] [nvarchar] (15) COLLATE Slovak_CI_AS NULL ,
        [amount] [money] NULL
) ON [PRIMARY]
```

```
INSERT INTO expense_factFF (id_ef, store_id, account_id, time_id, category_id,
                            department_id, amount)
  SELECT id_ef, store_id, account_id, time_id, category_id, department_id,
         (amount * conversion_ratio)
   FROM expense_factFM, currency
     WHERE expense_factFM.currency_id = currency.currency_id AND
        expense_factFM.exp_date = currency.date;
```

Samozřejmě znalci jazyka SQL napíší přímo příkaz pro přenos údajů z tabulky `expense_fact` do finální tabulky `expense_factFF`.

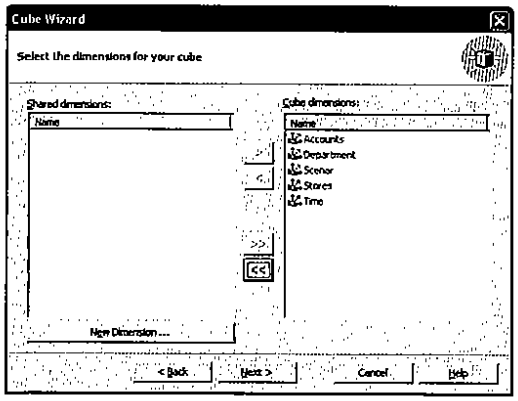
Vytvoření krychle

Po poměrně náročné přípravě dimenzí a tabulky faktů následuje paradoxně nejjednodušší část celého příkladu – vytvoření samotné krychle. Jako tabulku faktů vybereme pracně vytvořenou tabulku `expense_factFF`.



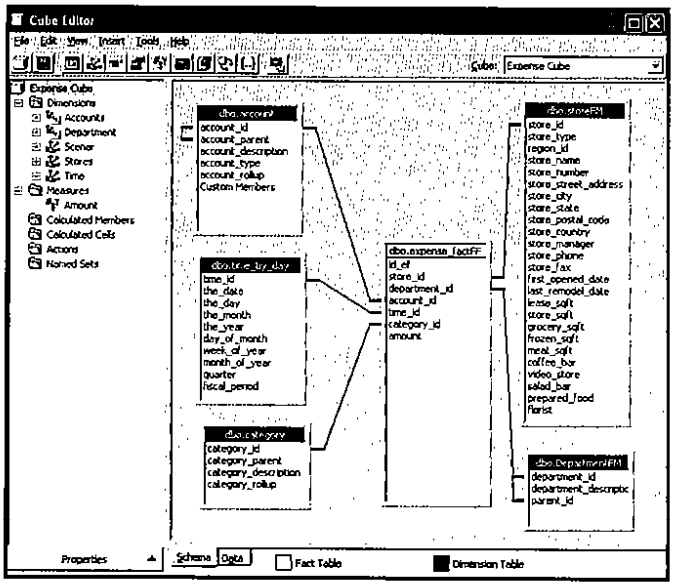
Obr. 4.62 Výběr tabulky faktů

Jako míru vybereme sloupec `amount`. Vybereme všechny navržené dimenze (SCENAR, DEPARTMENT, STORES, ACCOUNTS a TIME) a krychli pojmenujeme, například Expense Cube.



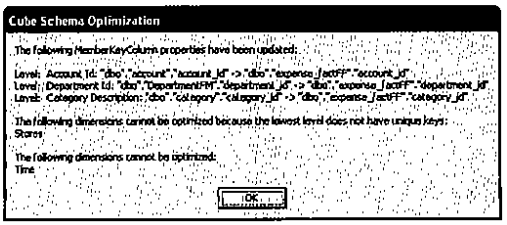
Obr. 4.63 Výběr dimenzí

Abychom zachovali v tomto cvičném příkladě určitý nádech „reality", po návrhu krychle v nástroji Cube Editor přistoupíme nejdříve k optimalizaci.



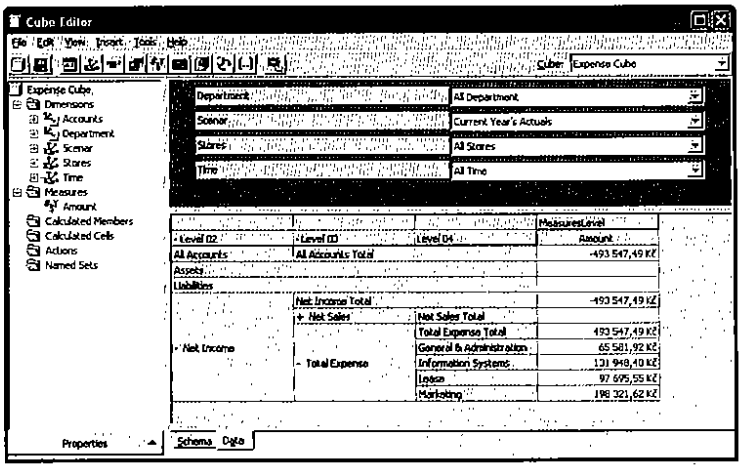
Obr. 4.64 Topologie navržené krychle

V menu Tools vybereme volbu Optimize Schema. Bude nám nabídnuta optimalizace tří dimenzí. Pro dimenzi Store není optimalizace možná, neboť nejnižší úroveň dimenze neobsahuje unikátní klíč.



Obr. 4.65 Dialog Cube Schema Optimalization

Abychom nastavili tento klíč, ukončíme zatím Cube Editor a ve složce Shared Dimension budeme editovat dimenzi Stores. V dialogu Dimension Editor ve složce Advanced nastavíme parametr Member Keys Unique na hodnotu True. Změnu uložíme, ukončíme Dimension Editor a znovu spustíme Cube Editor. Když znovu aktivujeme volbu menu Tools – Optimize Schema, zobrazí se dialog ohledně optimalizace dimenze Stores včetně hlášení, že tři ostatní dimenze už byly optimalizovány.



Obr. 4.66 Krychle Expense Cube v okně Cube Editor

Vytvoření krychle OLAP nad údaji z databáze Oracle

V kapitolách o přípravě údajů a datových skladech jsme si objasnili, že homogenní databázové a platformové prostředí je v podnikové sféře spíše vzácností, tedy když si odmyslíme plánovanou budovanou informační infrastrukturu moderních firem. Střetneme se s různými databázovými platformami a budeme postavení před úlohu pracovat s údaji z různých zdrojů. Databázová platforma Oracle 9i obsahuje analytické nástroje a při dostatku paměti i nejnovější verze Release 2 běží i na vývojářském počítači nebo notebooku.

Hardwarové požadavky by se daly shrnout do dvou pojmů: 256 MB paměti RAM a minimálně 5 GB volného místa na pevném disku. Takto vybavený počítač může dobře sloužit pro vývoj databázových aplikací, ale když začneme používat analytické služby, tak zjistíme, že databáze Oracle je koncipována spíše do prostředí podnikových serverů než na notebook studenta. Takže předpokládáme, že i čtenáři, kteří používají jiné databázové platformy, sáhnou při prvních pokusech právě po analytických službách MS SQL Serveru.

MS SQL Server Analysis Services versus databáze Oracle – první pokusy

Ukázat připojení a analýzu údajů z nejpoužívanějších databázových platforem by bylo nad rámec této publikace, proto se zaměříme na jednu z nejpoužívanějších databázových platforem v podnikovém prostředí – Oracle 9.

Znalci prostředí Oracle vědí, že během instalace databázového serveru se vytvoří několik cvičných schémat a jedno z nich, SH (Sales History), je určeno právě pro analýzu OLAP. Když začneme tuto část konstatováním, že pro první pokus s analytickými službami postačí tabulky z cvičného schématu SCOTT, pravděpodobně si i začínající „oraclisti" budou myslet, že se jedná o tiskovou chybu nebo nepodařený vtip. Toto schéma obsahuje dvě jednoduché tabulky EMP a DEPT. Pro naše účely vystačíme s tabulkami **EMP**, **DEPT** a **SALGRADE** ze schématu cvičného klienta **SCOTT** (implicitní heslo TIGER). Je to jakási zjednodušená personalistika firmy. Tabulka **EMP** obsahuje základní údaje o zaměstnancích firmy a tabulka **DEPT** obsahuje informace o odděleních firmy, tabulka **SALGRADE** obsahuje informace o tarifním ohodnocení zaměstnanců, řečeno naší terminologií o platových třídách.

Tabulka **EMP** obsahuje osm sloupců a následující návrhovou strukturu (můžeme ji zjistit příkazem `DESCRIBE emp;`).

NÁZEV	TYP
EMPNO	NUMBER(4)
ENAME	VARCHAR2(10)
JOB	VARCHAR2(9)
MGR	NUMBER(4)
HIREDATE	DATE
SAL	NUMBER(7,2)
COMM	NUMBER(7,2)
DEPTNO	NUMBER(2)

Tabulka **EMP** byla vytvořena pomocí příkazu (zjednodušeno):

```
CREATE TABLE emp
(
  empno    NUMBER(4) NOT NULL,
  ename    VARCHAR2(10),
  job      VARCHAR2(9),
  mgr      NUMBER(4),
  hiredate DATE,
  sal      NUMBER(7,2),
  comm     NUMBER(7,2),
  deptno   NUMBER(2)
);
```

Tabulka obsahuje tyto cvičné údaje o 14 zaměstnancích firmy:

```
EMPNO ENAME   JOB       MGR    HIREDATE  SAL    COMM   DEPTNO
----- ------- --------- ------ --------- ------ ------ -------
7369 SMITH    CLERK     7902 17-DEC-80   800           20
7499 ALLEN    SALESMAN  7698 20-FEB-81   1600   300     30
7521 WARD     SALESMAN  7698 22-FEB-81   1250   500     30
7566 JONES    MANAGER   7839 02-APR-81   2975           20
7654 MARTIN   SALESMAN  7698 28-SEP-81   1250   1400    30
7698 BLAKE    MANAGER   7839 01-MAY-81   2850           30
7782 CLARK    MANAGER   7839 09-JUN-81   2450           10
7788 SCOTT    ANALYST   7566 19-APR-87   3000           20
7839 KING     PRESIDENT       17-NOV-81   5000          10
7844 TURNER   SALESMAN  7698 08-SEP-81   1500           30
7876 ADAMS    CLERK     7788 23-MAY-87   1100           20
7900 JAMES    CLERK     7698 03-DEC-81   950            30
7902 FORD     ANALYST   7566 03-DEC-81   3000           20
7934 MILLER   CLERK     7782 23-JAN-82   1300           10
```

Tabulka **DEPT** má strukturu:

Název	Typ
DEPTNO	NUMBER(2)
DNAME	VARCHAR2(14)
LOC	VARCHAR2(13)

Byla vytvořena pomocí příkazu:

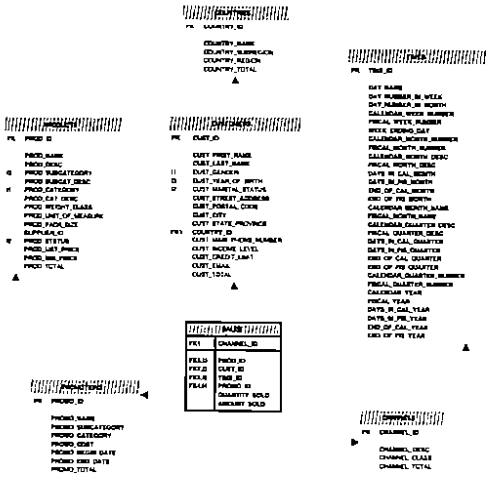
```
CREATE TABLE dept
(
  deptno NUMBER(2) NOT NULL,
  dname VARCHAR2(14),
  loc VARCHAR2(13)
);
```

a obsahuje tyto cvičné údaje:

```
DEPTNO   DNAME      LOC
------   ---------- ---------
10       ACCOUNTING NEW YORK
20       RESEARCH   DALLAS
30       SALES      CHICAGO
40       OPERATIONS BOSTON
```

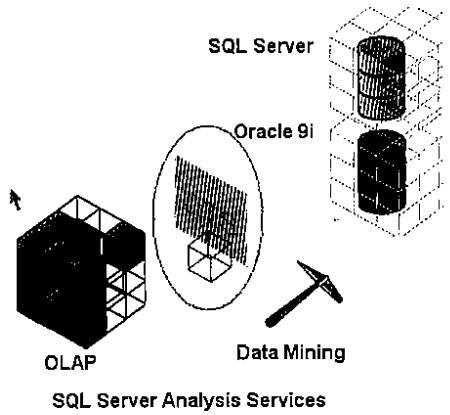
Vytvoření krychle OLAP nad údaji z databáze Oracle

V třetí kapitole je uveden postup připojení k jinému databázovému serveru, konkrétně k Oracle 9*i* a také ukázka postupu importu údajů z databáze Oracle 9i do databáze MS SQL Serveru. Jednalo se o přenos databázových tabulek z cvičného schématu SH (Sales History). Tato databáze je podrobně popsaná v kapitole 10. Pro zajímavost je v desáté kapitole vytvořena podobná krychle OLAP, jako v tomto příkladě, ale v prostředí Oracle. Schéma Sales History obsahuje celkem 15 databázových tabulek a materializovaných pohledů. Pro naši ukázku bude stačit znát strukturu tabulky faktů **SALES** a tabulky dimenzí **TIMES, PRODUCTS, CUSTOMERS** a **COUNTRIES**.



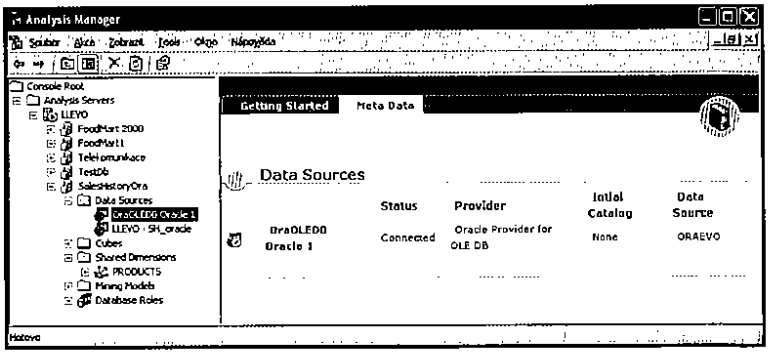
Obr. 4.67 Vytvoření krychle OLAP nad údaji z databáze ORACLE

V aplikaci Analysis Manager vytvoříme novou analytickou databázi například s názvem SalesHistoryOra. Pro výběr zdroje údajů máme dvě možnosti. Jednak použít již vytvořenou tabulku z příkladu z třetí kapitoly ze schématu Sales History uloženou v databázi MS SQL Serveru. Zajímavější ale bude možnost ponechat údaje tam, kde jsou, tedy v bezpečném a spolehlivém úložišti databáze Oracle, a připojit se k nim pomocí Analytických služeb MS SQL Serveru. Ve složce „Data Sources" tedy vytvoříme připojení na externí datový zdroj.



Obr. 4.68 Vytvoření krychle OLAP nad údaji z databáze ORACLE

Jako poskytovatele jsme použili Oracle Provider pro OLE DB. V záložce „Metadata" si můžeme prohlédnout parametry připojení. Nejdůležitějším parametrem je pochopitelně „Status", kde hodnota „Connected" tohoto parametru svědčí o úspěšném připojení.



Obr. 4.69 Vytvoření krychle OLAP nad údaji z databáze ORACLE

Po nastavení a otestování parametrů pro připojení k Oracle 9i můžeme přistoupit k vytváření jednotlivých dimenzí a následně k vytvoření krychle.

Vytvoření dimenzí

Krychle SH_CUBE bude obsahovat celkem pět dimenzí:

- časovou dimenzi, vytvořenou na základě údajů z tabulky SH.TIMES,
- produktovou dimenzi vytvořenou nad tabulkou SH.PRODUCT,
- regionální dimenzi vytvořenou nad tabulkami SH.CUSTOMERS a SH.COUNTRIES.

Postup je úplně analogický jako při vytváření krychle z údajů uložených v databázi MS SQL Serveru s jedinou výjimkou, a tou je časová dimenze.

Časová dimenze

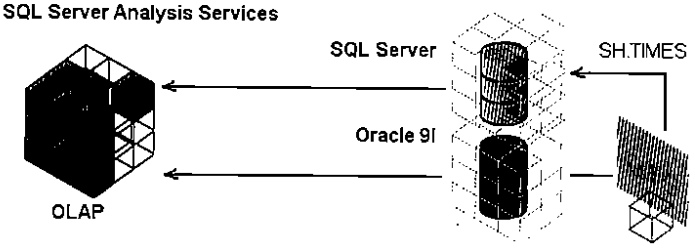
Tuto dimenzi vytvoříme na základě údajů z tabulky SH.TIMES. Jako datový sloupec vybereme sloupec TIME_ID a dimenze bude mít hierarchickou strukturu

- `Year` sloupec `CALENDAR_YEAR`
  - `Quarter` sloupec `CALENDAR_QUARTER_DESC`
    - `Month` sloupec `CALENDAR_MONTH_DESC`
      - `Day` sloupec `TIME_ID`

Kdybychom chtěli vygenerovat časovou dimenzi automaticky na základě tabulky SH.TIMES v databázi Oracle, tak bychom nepochodili. Průvodce totiž na základě názvů sloupců a datových typů v původní tabulce uložené v databázi Oracle nedokáže automaticky vygenerovat hierarchické úrovně časové dimenze. Tento problém se dá řešit různě, nejjednodušší se nám zdálo přenesení celé tabulky TIMES do databáze MS SQL Serveru. Po tomto kroku máme pro analytickou databázi „SalesHistoryOra" ve složce „Data Sources" dva datové zdroje (názvy byly vygenerovány automaticky při definování datových zdrojů).

OraOLEDB Oracle1 – data uložená v databázi Oracle 9i.

LLEVO-SH_oracle – tabulka SH.TIMES přenesená pomocí datové pumpy DTS z databáze Oracle do databáze MS SQL Serveru.



Obr. 4.70 Dva zdroje údajů pro analytickou databázi SalesHistoryOra

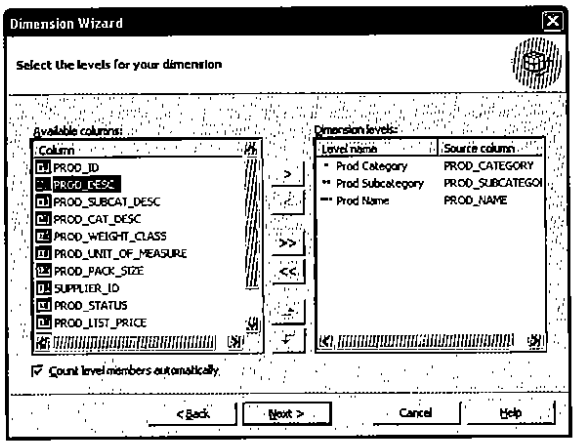
Dobrá otázka by byla, když už přenášíme jednu tabulku, proč nepřeneseme na databázový server všechny tabulky. Tabulka faktů má ale v tomto případě více než milion záznamů, a to se jedná o cvičnou databázi. Reálné tabulky faktů jsou o mnoho větší. Kdyby tabulka faktů nebo tabulky sloužící pro její vytvoření byly v nějaké souborové databázi, například v souborech MDB databáze Access, kde se o nějaké spolehlivosti dá jen těžko hovořit, bylo by přenesení údajů otázkou nanejvýš aktuální a potřebnou. Ale když jsou údaje uloženy v databázích pod správou moderních spolehlivých a bezpečných databázových serverů, tak můžeme údaje ponechat tam, kde jsou. Kdybychom budovali větší dílo, tedy datový sklad, údaje samozřejmě zavedeme do jednotného úložiště.

Vytvoření ostatních dimenzí je už jednoduchá rutina.

Produktová dimenze

Produktová dimenze je vytvořena nad tabulkou SH.PRODUCTS jako Schéma Star a má tyto úrovně:

- Prod Category — sloupec PROD_CATEGORY
  - Prod Subcategory — sloupec PROD_SUBCATEGORY
    - Prod Name — sloupec PROD_NAME

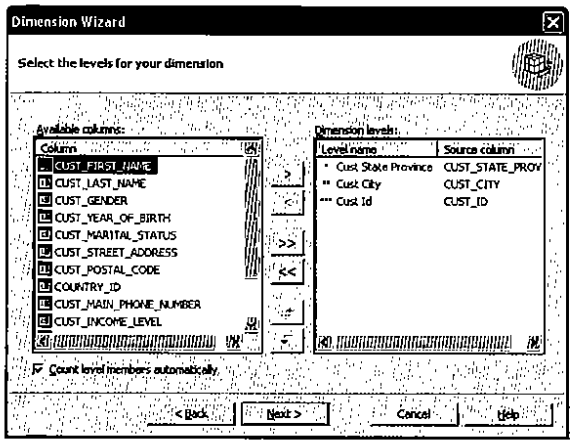


Obr. 4.71 Produktová dimenze v analytické databázi SalesHistoryOra

Zákaznická dimenze

Geografická nebo zákaznická dimenze je vytvořena nad tabulkou SH.CUSTOMERS.

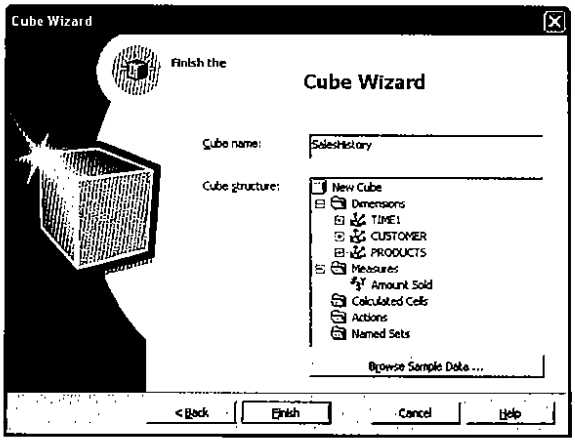
CUSTOMERS
- Cust State Province — sloupec CUST_STATE_PROVINCE
  - Cust City — sloupec CUST_CITY
    - Cust Id — sloupec CUST_ID



Obr. 4.72 Zákaznická (geografická) dimenze v analytické databázi SalesHistoryOra

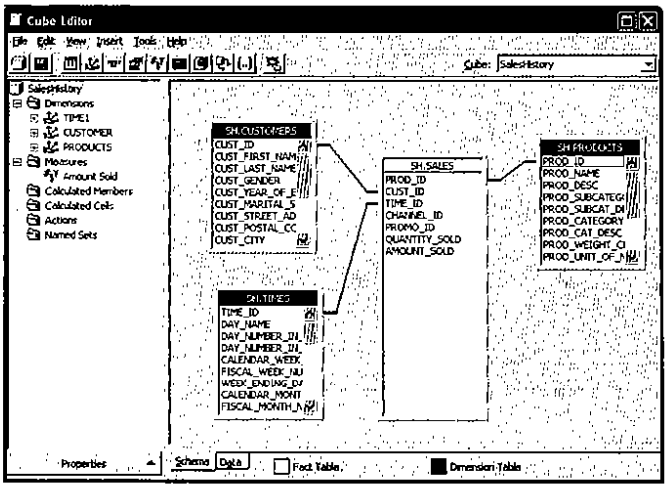
Vytvoření krychle

Jako tabulku faktů zvolíme tabulku SH.SALES, přičemž využijeme jen jednu míru, konkrétně sloupec Amount_sold.



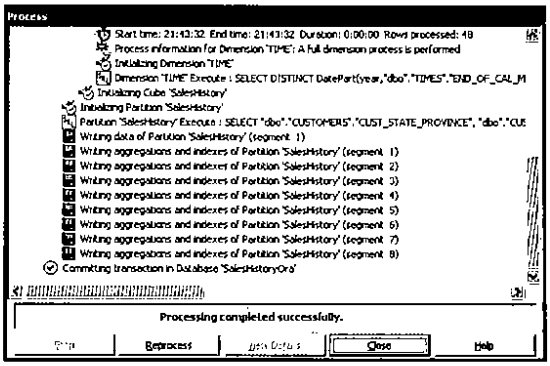
Obr. 4.73 Návrh krychle SalesHistory v analytické databázi SalesHistoryOra

Při návrhu krychle použijeme doteď vytvořené dimenze, v našem případě TIME, CUSTOMERS a PRODUCTS.



Obr. 4.74 Tabulka faktů a tabulky dimenzí

Základní funkcionalitu krychle OLAP vytvořené nad cvičným schématem Sales History v databázi Oracle můžeme vyzkoušet pomocí nástroje Cube Editor ve složce Data.



Obr. 4.75 Zákaznická (geografická) dimenze v analytické databázi SalesHistoryOra

Administrace přístupu k databázím OLAP

O administraci přístupu k údajům v klasických transakčních databázích se popsalo už mnoho knih a článků v odborných časopisech. Zamyslete se však nad tím, proč je vlastně důležitá administrace přístupu k transakčním databázím a jaké údaje tyto databáze zpravidla obsahují. Transakční databáze jsou typickým příkladem víceuživatelského prostředí. Proto je velmi důležité definovat přístupová práva jednotlivých uživatelů k databázovým objektům a specifikovat rozsah jejich oprávnění. Někteří uživatelé mohou údaje do databáze zapisovat, případně je mazat, jiní mají zpřístupněné jen čtení údajů. Uživatelský přístup má na starosti správce databáze většinou ve spolupráci se správcem systému. Kdybychom měli shrnout obsah tohoto odstavce jednoduše, je potřeba zabránit přístupu neoprávněných osob k údajům v databázích.

Kdybychom ale měli porovnat „citlivost" a obchodní hodnotu údajů uložených v transakčních databázích s informacemi v analytických databázích, rozdíl je propastný. Když by se někomu podařilo „ukořistit" několik tabulek, nebo i celou transakční databázi, užitek by z toho měl jen po složitém vyhodnocování. Kdežto po získaní přístupu do analytické databáze má narušitel všechny informace naservírované jako na stříbrném tácu. Proto je ochrana údajů v analytických databázích o mnoho důležitější, jde zpravidla o nejtajnější know-how firmy.

Role

Když začneme vytvářet uživatele pro složitější projekt a přidělovat jim přístupová práva, dříve či později přijdeme na to, že se nám vytvářejí skupinky uživatelů, kteří mají stejná práva. Je to analogie s běžnou hierarchií ve firmách, kde pracuje více pracovníků na stejné pozici. Proto se při administrování databází používají role. Role umožňují sdružovat uživatele do skupin. Mezi uživateli a rolemi je vztah M:N, což znamená, že nejen více uživatelů je přiřazeno jedné roli, ale i opačně, jeden uživatel může mít více rolí. V analytických službách serveru MS SQL můžeme definovat dva typy rolí.

- Databázové role pro přístup k analytické databázi jako celku.
- Role pro přístup ke konkrétním krychlím.

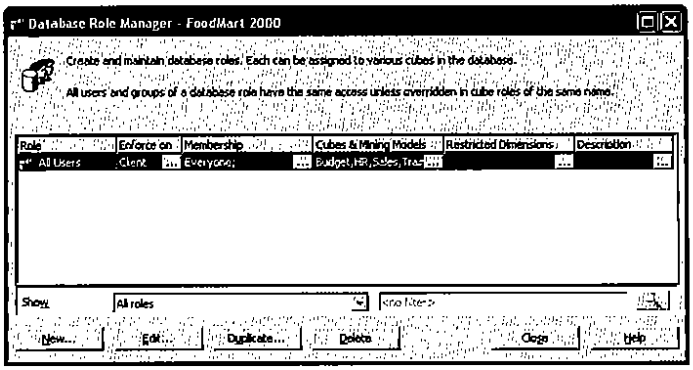
Role vytváříme pomocí aplikace Analysis Manager, kde v prohlížeči objektů aktivujeme kontextové menu ikony Database Roles anebo Cube roles.

Role pro přístup k analytickým databázím

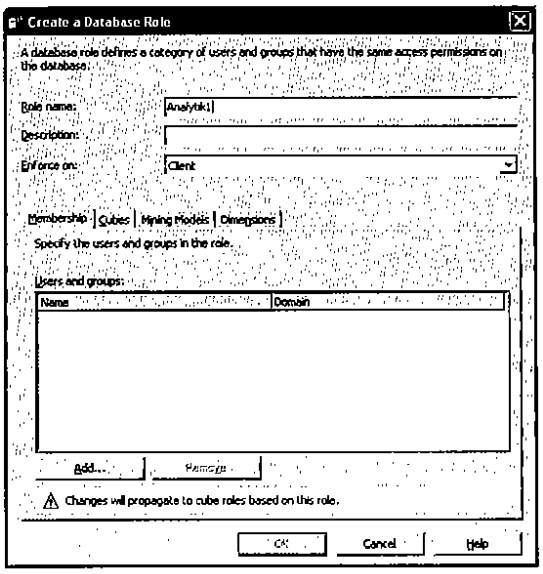
V kontextovém menu ikony Database Roles vybíráme položku Manage roles. Zobrazí se dialog nástroje Database Role Manager.

V okně Database Role manager jsou zobrazeny všechny doteď vytvořené role. Pro cvičnou databázi FoodMart 2000 je předdefinovaná role All Users. Pomocí tlačítka New otevřeme dialog pro přidání nové role. Dialog „Create a Database Role" obsahuje kromě pole pro zadání názvu role i čtyři záložky:

- Membership
- Cubes
- Mining Models
- Dimensions



Obr. 4.76 Database Role Manager

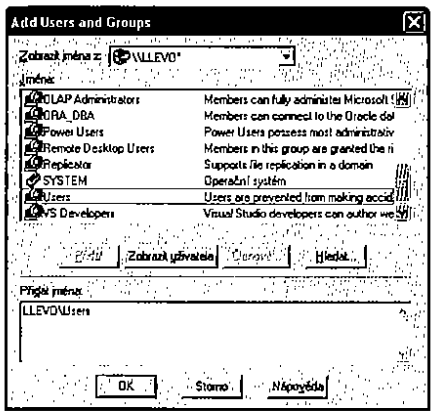


Obr. 4.77 Vytvoření nové databázové role

Pomocí combo boxu „Enforce on" můžeme určit, zda se nastavená privilegia budou uplatňovat na serveru, nebo na klientských počítačích.

Záložka Membership

Nového uživatele nebo skupinu uživatelů můžeme přidat na záložce Membership pomocí dialogu „Add Users and Groups" aktivovaného tlačítkem „Add...".



Obr. 4.78 Přidání uživatele nebo uživatelské skupiny

Protože nejen Server MS SQL, ale samozřejmě i analytické služby jsou úzce svázané s operačním systémem Windows, zobrazí se nám nabídka jeho uživatelů.

Záložka Cubes

V této složce můžeme zaškrtnutím příslušného políčka povolit přístup k údajům jednotlivých krychlí OLAP.

Záložka Mining Models

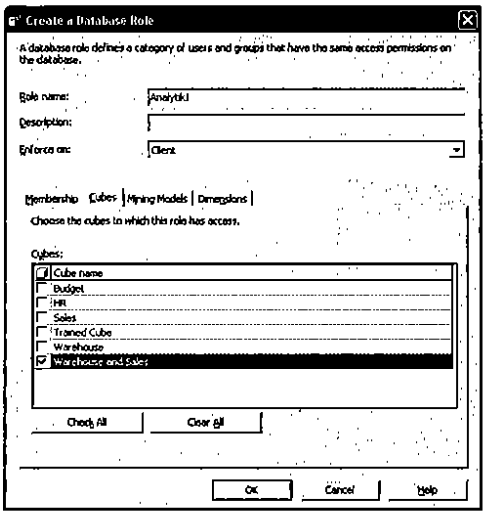
Podobným způsobem definujeme přístup k jednotlivým dataminingovým modelům.

Záložka Dimensions

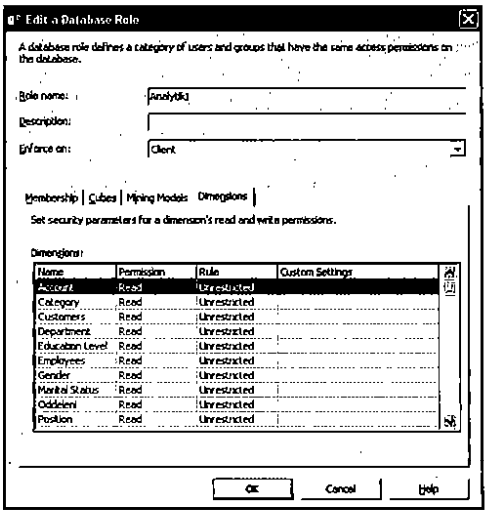
V této záložce definujeme způsob přístupu k údajům podle jednotlivých dimenzí.

Role pro přístup k jednotlivým krychlím analytické databáze

Kromě přístupových práv pro celou analytickou databázi, kde můžeme nastavit přístupová práva jednotlivých uživatelů k jednotlivým krychlím jako celku, můžeme nastavovat přístupová práva i vzhledem k objektům uvnitř krychle, tedy vzhledem k dimenzím a buňkám.



Obr. 4.79 Create a Database Role – záložka Cubes



Obr. 4.80 Create a Database Role – záložka Dimensions

V předcházející části jsme ukázali nastavení přístupových práv pro celou analytickou databázi. Přístupová privilegia můžeme nastavovat i pro jednotlivé krychle. Postup je analogický jako při nastavování přístupových práv pro analytickou databázi. Aktivujeme kontextové menu na ikoně *„Cube Roles"*. Tato ikona se nachází ve složce každé krychle. Pro nastavování se používa nástroj *„Cube Role Manager"*. Jeho vzhled a použití je podobné jako u nástroje *„Database Role Manager"*.

Podobně i dialog pro vytvoření nové role pro přístup k údajům v krychli obsahuje kromě pole pro zadání názvu role i čtyři záložky:

- Membership
- Dimensions
- Cells
- Options

Záložka Membership

Slouží pro přidání nového uživatele nebo skupiny uživatelů.

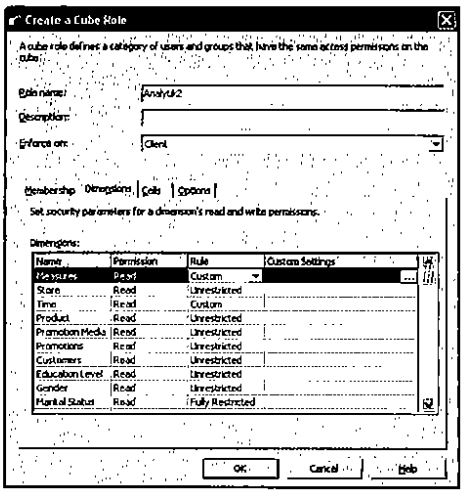
Záložka Dimensions

Na této záložce nastavujeme omezení pro jednotlivé dimenze. Členy dimenze můžeme buď jen číst, případně když na to máme oprávnění, i modifikovat.

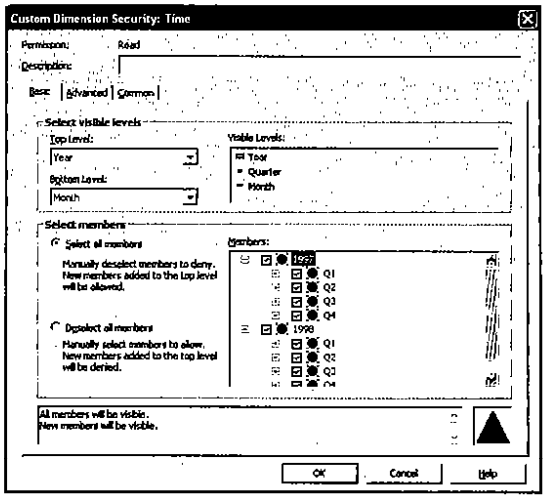
Můžeme použít přístup bez omezení (Unrestricted), můžeme přístup k vybrané dimenzi úplně zakázat (Fully Restricted), případně můžeme přístupové parametry nastavit podle našich požadavků (Custom).

Oprávnění	Popis
Read	Oprávnění specifikuje, které členy dimenzí jsou viditelné.
Read/write	Oprávnění specifikuje, které členy dimenzí je možné modifikovat.

Když sloupec Role pro vybranou dimenzi nastavíme na hodnotu „Custom", zobrazí se ve sloupci „Custom Settings" tlačítko (...) pro nastavení přístupových parametrů pro jednotlivé úrovně dimenze.

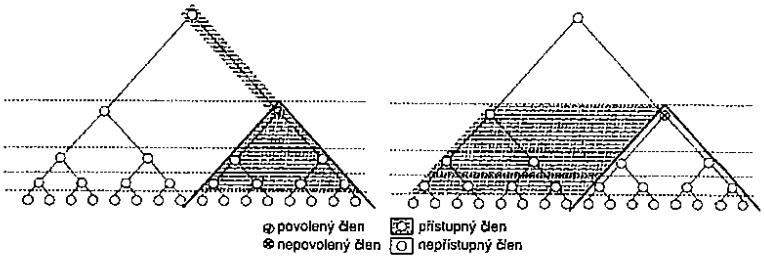


Obr. 4.81 Create a Cube Role – záložka Dimension



Obr. 4.82 Dialog pro nastavení přístupových parametrů pro jednotlivé úrovně vybrané dimenze

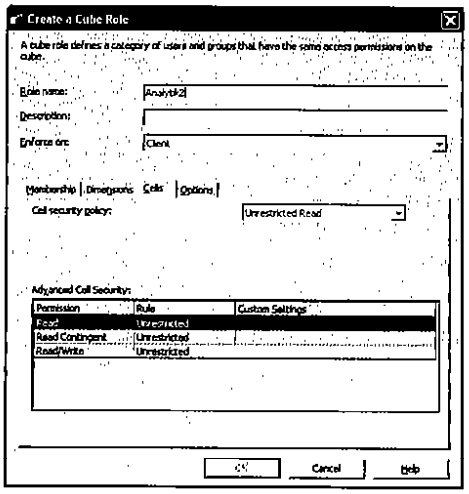
Filosofie přístupu může být řízena pomocí povolených a zakázaných členů dimenzí. Mohou totiž nastat v zásadě dva případy. Někdy potřebujeme zpřístupnit skoro všechno kromě malé části. V takovém případě je výhodné specifikovat zákazy na úrovni jednotlivých členů dimenze (Denied Members). Jindy je potřebné zpřístupnit jen určitou malou část a v takovém případě je zase výhodnější specifikovat povolení (Allowed Members). Takto je možné určit „výřez" z hierarchické struktury, který bude přístupný, nebo naopak výřez, který bude nepřístupný.



Obr. 4.83 Povolené a zakázané členy dimenzí

Záložka Cells

Výsledky analýz jsou natolik citlivé, že bylo potřebné specifikovat dokonce i přístupová pravidla pro jednotlivé buňky.

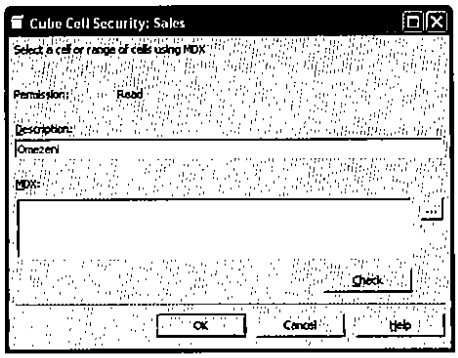


Obr. 4.84 Create a Cube Role, záložka Cells

Pomocí Combo Boxu „Cell security policy" můžeme nastavit úroveň bezpečnostní politiky pro přístup k buňkám krychlí analytické databáze. Úrovně bezpečnostní politiky jsou v této tabulce:

Odvolání	Popis
Unrestricted Read	Přístupné jsou všechny hodnoty buněk. Tato volba je definována implicitně.
Unrestricted Read/Write	Hodnoty všech buněk jsou přístupné nejen pro čtení, ale i pro zápis.
Advanced	Jsou zpřístupněny jen ty hodnoty, které jsou defnovány pomocí příkazů jazyka MDX.

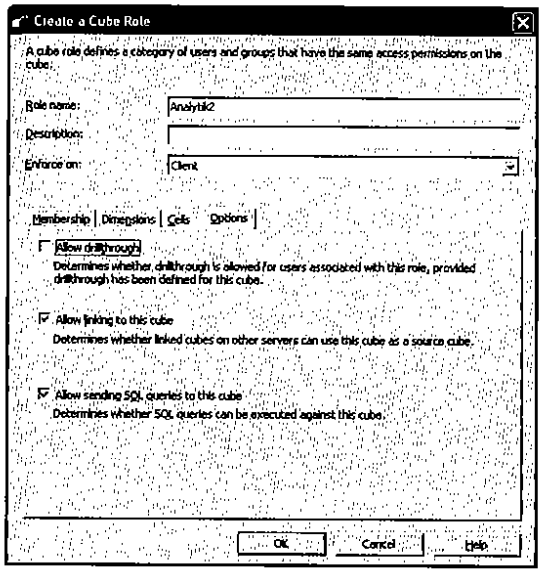
Kromě základního nastavení můžeme nastavovat přístup k jednotlivým buňkám pomocí ovládacích prvků ve skupině „Advanced Cell Security". Pravidla se definují v jazyce MDX. Příkaz jazyka MDX nemusíme pracně psát do editačního okna. K dispozici máme pro tento účel vizuální nástroj MDX Builder. Aktivuje se tlačítkem (...) vedle editačního okna příkazu MDX.



Obr. 4.85 Definování pravidel pro přístup k buňkám v jazyce MDX

Záložka Options

Na této záložce můžeme nastavit další parametry pro přístup k údajům v krychlích OLAP.



Obr. 4.86 Create a Cube Role, záložka Options

Kapitola 5

Přístup k databázím OLAP z klientských aplikací

Už z názvů jednotlivých komponentů jako Server OLAP a analytické služby vyplývá, že se jedná o služby běžící na serveru. Samozřejmě, k těmto službám se dodávají různé aplikace a utility pro jejich správu, ladění a v některých případech i na prohlížení výsledků. Ovšem datové sklady a analytické databáze jsou v podnikovém prostoru efektivní jen tehdy, když je mohou využívat příslušní klienti, například manažeři, analytici a podobně. Analytické služby jsou proto k dispozici jako systém klient-server. Pozornost je proto kromě návrhu analytických databází potřebné věnovat i klientské části a právě o tom budou následující tři kapitoly.

V každém prostředí datového skladu se uživatelé a jejich požadavky na aplikační prostředí a typy dotazů liší v závislosti na jejich přístupu k předmětu podnikání nebo na jejich postavení ve firemní hierarchii. Mohou to být manažeři, obchodníci, analytici, externisti. Každý z nich potřebuje jiný segment údajů, jiný typ přístupu a jinak prezentované údaje. Například analytici mají přístup k údajům v datovém skladě jen prostřednictvím zobrazených nebo vytištěných sestav. Při výběru existující nebo nové klientské aplikace musíme brát v úvahu požadavky a potřeby jejich budoucích uživatelů.

V současném průmyslu IT existuje mnoho architektur. V prostředí datového skladu dominují architektury typu klient-server a přístup k údajům přes web. Procesy pro přístup k údajům a jejich zpracování se tak rozdělí mezi servery a klientské počítače, na kterých se vykonává lokální zpracování.

Filosofie klientského přístupu k analytickým údajům

Základní rozdělení klientských aplikací je podle kritéria, zda jádro aplikace běží na serveru, nebo na klientském počítači, tyto aplikace rozdělujeme na takzvaného „tenkého" a „tlustého" klienta.

Tenký klient

Pod pojmem **„tenký klient"** myslíme zpravidla počítač, kde se pro přístup k údajům používá prohlížeč webových stránek. To znamená, že celá aplikační logika běží na serveru a výpočetní možnosti klientského počítače víceméně leží ladem, využívají se hlavně pro zobrazení. Tenký klient je zpravidla navržen tak, aby byl malý, takže hromadné zpracování dat se vykonává na serveru.

Jednou z vizí budoucnosti byl a stále je tenký klient jako síťový počítač bez pevného disku. Tato vize se naplňuje například nástupem mobilních zařízení (Pocket PC 2002, PALM, Smartphone). Pod pojmem tenký klient můžeme tedy chápat nejen jakýkoliv počítač, od nejvýkonnějšího serveru, až po low-endové sestavy, ale i kapesní počítače třídy Pocket PC, Palm, ba dokonce i mobilní telefony s podporou protokolu WAP. Potom mluvíme o „mobilním tenkém klientovi".

Výhodou použití tenkého klienta je hlavně podpora nejširšího spektra zařízení a platforem a přístup prostřednictvím prohlížeče webových stránek. Hlavní nevýhodou je, že takové aplikace z principu nepracují v režimu offline, ve většině případů málo využívají výkon lokálního zařízení, jsou tu ale i jistá omezení týkající se uživatelského rozhraní, která vyplývají například z toho, že protokol HTTP je bezstavový. Výhodou tenkého klienta jsou jednak nízké náklady na hardware na jednoho uživatele, jednak nízké licenční náklady na software, protože „drahé" jádro softwaru je na serveru.

Při nástupu nových technologií pravidlo o laciném hardware chvíli neplatí, první počítače Pocket PC byly poměrně drahé, a tak zpočátku nacházely uplatnění hlavně jako luxusní hračky v rukách manažerů. Dnes je ale využívá o mnoho širší spektrum uživatelů a i ti vzpomínaní manažeři už svoje luxusní hračky využívají hlavně pro přístup k informacím.

Tlustý klient

Pod pojmem **„tlustý klient"** (v anglické terminologii rich client) si budeme představovat klientskou aplikaci, která běží na lokálním počítači, a tedy může naplno využívat výkon jeho hardwaru a možnosti operačního systému. Pro svoji činnost ale využívá údaje a službu nějakého serveru.

V 80. letech s nástupem éry PC se na většinu pracovišť začaly zavádět poměrně výkonné počítače s grafickým rozhraním. Jejich výpočetní a paměťové možnosti postupně rostly, takže na nich bylo možné provozovat i výkonné databázové servery. Podniky s nástupem této éry zpravidla zredukovaly počítačovou architekturu sálových počítačů. V současnosti je PC základem moderního podnikového systému a poskytuje mnohým uživatelům možnosti zpracování údajů a mnohých jiných úkonů a činností, které při své práci potřebují.

Toto řešení začalo mít také určité nevýhody. PC vyžadovaly stále výkonnější hardware a stále více softwaru. Ale správa více kopií softwaru je složitá a jeho licencování je drahé. Nákup a provoz PC je nákladný ani ne vzhledem k hardwaru, ale hlavně vzhledem k množství softwaru potřebného na každém konkrétním PC. Tlustý klient tedy na klientském počítači vykoná operace zpracování a vyhodnocení údajů. Samotné údaje se ukládají na serveru.

Výhodou takového řešení je, že aplikace pracují v obou dvou režimech offline i online, přičemž se během připojení může klientský počítač synchronizovat, případně replikovat údaje se serverem. Nevýhodou je, že pro každou platformu (Windows, Linux, iMAC, Pocket PC 2002...) musíme aplikaci vyvíjet zvlášť. Tato nevýhoda se částečně stírá použitím tzv. řízené-

ho (managed) kódu, který je pro více platforem stejný. Takto můžeme například vyvíjet aplikace v programovacích jazycích vb.net a C# pro platformu Windows i Pocket PC 2002.

Připojený tlustý klient

Když aplikace typu tlustý klient pracuje v síťovém prostředí a má neustálou konektivitu na server, tak v takovém případě mluvíme o připojeném tlustém klientovi. Když pro připojení použijeme protokol HTTP, můžeme aplikace tohoto typu využívat prakticky všude, kde je připojení na Internet.

Odpojený tlustý klient

Filosofie aplikací typu odpojený tlustý klient je založena na poznatcích z praxe, kdy často potřebujeme výsledky analýz, hlavně sumární a parciální, a to i v takových situacích, kdy nemáme možnost připojení k síti, například na prezentacích, u zákazníka a podobně. V takovém případě pracujeme s údaji v „lokálních" krychlích, uložených v souborech, takže je můžeme stejně jako ostatní soubory přenášet a kopírovat, případně stáhnout z webu, poslat elektronickou poštou a podobně.

Libovolný všeobecný klient

Mnozí se možná nad tímto pojmem pousmějí jako nad nesplnitelným snem, ale opak je pravdou. Dnes je stále více rozšířený formát pro výměnu údajů mezi nehomogenními a nekompatibilními aplikacemi v nehomogenním prostředí. Je to formát XML. Proto když analytický server dokáže poskytnout údaje v tomto formátu, tak máme prakticky neomezené možnosti zobrazení a využívání těchto údajů v libovolných aplikacích.

Programy balíku Microsoft Office jako klienti analytických služeb

Z pohledu uživatele je nejlepší taková klientská aplikace, kterou uživatel důvěrně zná, tedy ideální případ je integrace přístupu k analytickým službám do aplikací, s nimiž klienti doposud pracovali. Mohou to být různé speciální jednoúčelové aplikace, ale když se zamyslíme nad tím, jaký typ softwaru se na klientských počítačích, tedy na běžných pracovních stanicích, které máme ve svých kancelářích, nejčastěji používá, bezpochyby zvítězí kancelářské balíky. Tuto filosofii si uvědomila i firma Microsoft a v poměrně velkém rozsahu integrovala podporu analytických služeb do svého kancelářského balíku Microsoft Office.

Když se připojíme k serveru OLAP pomocí klientské aplikace, například pomocí programu Microsoft Excel, sumarizované hodnoty vypočítává server OLAP, klientská aplikace tyto údaje jen zobrazuje, případně je dále zpracovává nebo například ukládá jako pohledy. Při zobrazení nebo změně sestavy se proto mezi serverem OLAP a klientskou aplikací posílá poměrně málo údajů.

Nejen tabulkový procesor MS Excel a kancelářský databázový program MS Access ale mohou pracovat s analytickými údaji. Vynikající myšlenka je možnost nainstalovat a používat Smart Tags (chytré značky). Potom když například píšeme nějakou zprávu nebo analýzu v textovém editoru Microsoft Word a napíšeme slovo, které se vyskytuje v analytické databázi, například píšeme o svých produktech nebo filiálkách firmy, případně geografických lokalitách, toto slovo se zvýrazní, když na něj aktivujeme Smart Tags, tak můžeme do

textu vložit údaj, tabulku, nebo dokonce graf, který je výsledkem analýzy pro danou entitu. Nebo si jen otevřeme výsledky analýzy v programu MS Excel a na základě těchto podkladů napíšeme do textu zprávy vlastní vyhodnocení analyzované situace.

Kontingenční tabulka (Pivot Table)

Předtím než začneme využívat jako klientské aplikace programy z kancelářského balíku Microsoft Office, hlavně program Microsoft Excel, nebo webové komponenty balíku Microsoft Office s názvem Office Web Components (sedmá kapitola), tak je potřebné vysvětlit pojem kontingenční tabulka (anglicky Pivot Table). Ti čtenáři, kteří s kontingenčními tabulkami například v MS Excel pracují, mohou tuto část klidně přeskočit.

Kontingenční tabulka má na rozdíl od klasické tabulky několik speciálních vlastností. Například umožňuje určitou rotaci, tedy výměnu řádků a sloupců, kombinaci a hierarchickou strukturu řádků a sloupců. Nejlépe to vysvětlíme na klasické databázové tabulce. Když vytvoříme jednoduchou tabulku

```
CREATE TABLE PivotTable
( rok      INT,
  kvartal  INT,
  zisk     money
)

INSERT INTO PivotTable VALUES (2000, 1, 6300)
INSERT INTO PivotTable VALUES (2000, 2, 6900)
INSERT INTO PivotTable VALUES (2000, 3, 7100)
INSERT INTO PivotTable VALUES (2000, 4, 7150)
INSERT INTO PivotTable VALUES (2001, 1, 7300)
INSERT INTO PivotTable VALUES (2001, 2, 7700)
INSERT INTO PivotTable VALUES (2001, 3, 8000)
INSERT INTO PivotTable VALUES (2001, 4, 8100)
```

a pomocí dotazu SQL necháme vypsat její obsah, dostaneme výpis klasické tabulky.

```
SELECT * FROM PivotTable
```

rok	kvartal	zisk
2000	1	6300.0000
2000	2	6900.0000
2000	3	7100.0000
2000	4	7150.0000
2001	1	7300.0000
2001	2	7700.0000
2001	3	8000.0000
2001	4	8100.0000

Když ale potřebujeme výstup, kde budou řádky a sloupce uspořádány trochu jinak, například pro jednotlivé kvartály příslušného roku, musíme pro vygenerování takového výstupu použít poměrně složitý dotaz SQL.

```
SELECT rok,
    SUM(CASE kvartal WHEN 1 THEN zisk ELSE 0 END) AS 'První kvartál',
    SUM(CASE kvartal WHEN 2 THEN zisk ELSE 0 END) AS 'Druhý kvartál',
    SUM(CASE kvartal WHEN 3 THEN zisk ELSE 0 END) AS 'Třetí Kvartál',
    SUM(CASE kvartal WHEN 4 THEN zisk ELSE 0 END) AS 'Čtvrtý kvartál'
FROM PivotTable GROUP BY rok
```

Rok	První kvartál	Druhý kvartál	Třetí kvartál	Čtvrtý kvartál
2000	6300.0000	6900.0000	7100.0000	7150.0000
2001	7300.0000	7700.0000	8000.0000	8100.0000

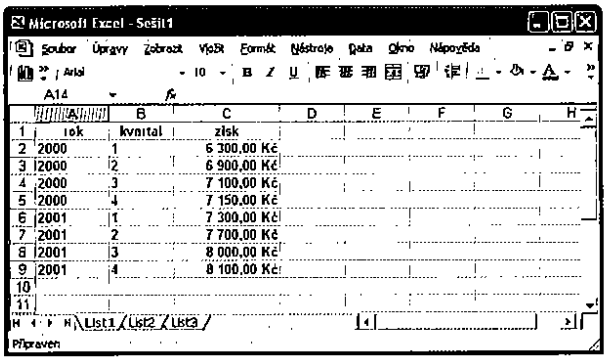
V této tabulce jsou některé údaje z řádků přeskupeny do sloupců. Případně když potřebujeme získat ještě i sumární údaje, tak zase musíme vytvořit pro daný účel ještě složitější dotaz SQL.

```
SELECT P1.*, (P1.Q1 + P1.Q2 + P1.Q3 + P1.Q4) AS 'Spolu za rok'
FROM (SELECT rok,
             SUM(CASE P.kvartal WHEN 1 THEN P.zisk ELSE 0 END) AS Q1,
             SUM(CASE P.kvartal WHEN 2 THEN P.zisk ELSE 0 END) AS Q2,
             SUM(CASE P.kvartal WHEN 3 THEN P.zisk ELSE 0 END) AS Q3,
             SUM(CASE P.kvartal WHEN 4 THEN P.zisk ELSE 0 END) AS Q4
      FROM PivotTable AS P
      GROUP BY P.rok) AS P1
```

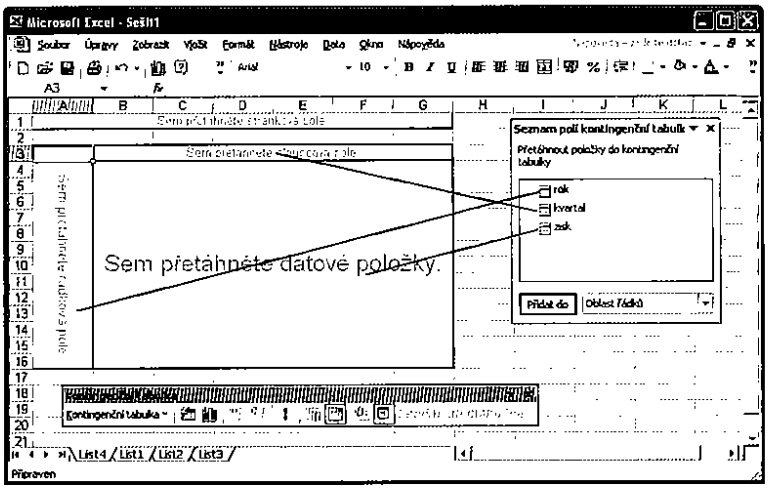
Rok	Q1	Q2	Q3	Q4	Spolu za rok
2000	6300.0000	6900.0000	7100.0000	7150.0000	27450.0000
2001	7300.0000	7700.0000	8000.0000	8100.0000	31100.0000

Po podrobnějším prostudovaní předcházejících příkladů a porovnání s postupem v programu MS Excel nejlépe pochopíme význam kontingenční tabulky. Původní klasickou tabulku překonvertujeme na kontingenční (menu *Údaje –Sestava kontingenční tabulky a kontingenčního grafu*).

Pro vytvoření kontingenční tabulky použijeme příslušného průvodce, přičemž jako zdroj údajů využijeme původní tabulku v listu programu MS Excel. Tu stejnou sestavu, kterou jsme předtím pracně vytvářeli pomocí dotazu SQL, vytvoříme přesunutím tří položek na příslušná pole tabulky.

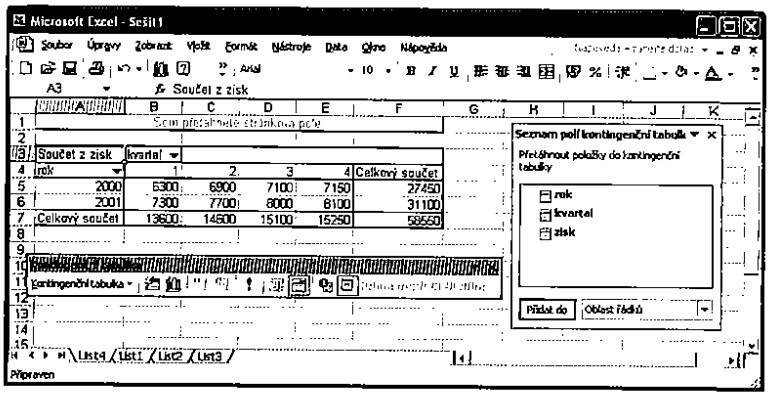


Obr. 5.1 Původní tabulka v programu Microsoft Excel



Obr. 5.2 Vytvoření kontingenční tabulky

V našem případě jsme atribut ROK, tedy jeho ikonu přesunuli do obdélníka pro pole řádků, atribut kvartál do obdélníka pro pole sloupců. Nakonec přesuneme atribut zisk do obdélníka pro pole údajů. Po těchto třech přesunech ikon máme okamžitě k dispozici výslednou sestavu.



Obr. 5.3 Údaje v kontingenční tabulce

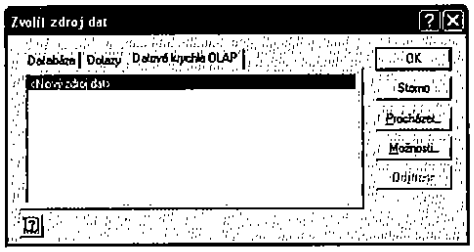
Pozornému čtenáři je už zřejmé, jak se dá kontingenční tabulka použít pro zobrazení údajů z multidimenzionálních databází. Do obdélníka pro pole údajů umístíme fakta a do obdélníků pro pole řádků a sloupců umístíme příslušné dimenze. V sestavě kontingenční tabulky nebo kontingenčního grafu se z každé dimenze stává množina polí, ve kterých je možno rozbalit a sbalit podrobnosti na jednotlivých úrovních hierarchií. Údajová pole jsou pole ze zdrojového seznamu, tabulky nebo databáze, která obsahují údaje sumarizované v sestavě kontingenční tabulky nebo kontingenčního grafu. Údajová pole obyčejně obsahují číselné údaje, například statistické údaje nebo částky prodeje.

Microsoft Excel jako klient analytických služeb

Pro prohlížení výsledků analýzy můžeme použít různé klientské programy. Ve firemním prostředí bude pravděpodobně nejdostupnějším programem **MS Excel 2000** (nebo XP) z balíku MS Office 2000 nebo Office XP. Připojit se můžeme k analytickým službám z libovolného klientského počítače, který má povolený přístup k analytickému serveru a ke konkrétním výsledkům analýzy.

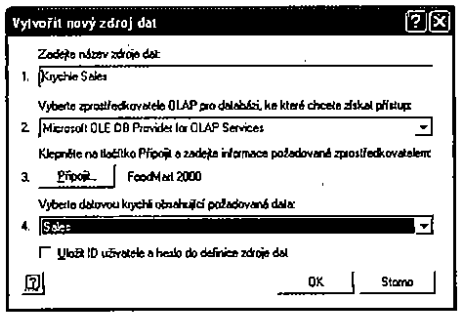
Připojení ke zdroji údajů

V hlavním menu programu MS Excel zvolíme položku menu *Data* → *Načíst externí data* → *Nový Databázový dotaz*. Zobrazí se dialog pro připojení k externímu zdroji údajů se záložkami **Databáze**, **Dotazy** a **Datové krychle OLAP**.



Obr. 5.4 Výběr zdroje údajů

Vybereme záložku **Datové krychle OLAP** a po stisku tlačítka OK budeme vytvářet připojení k analytickému serveru, konkrétně se připojíme ke cvičné krychli SALES.

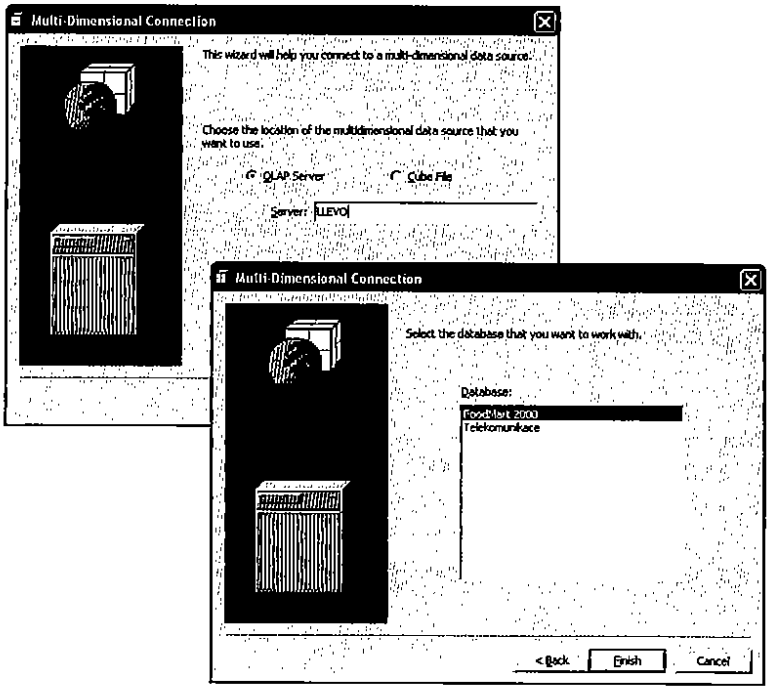


Obr. 5.5 Připojení ke krychli Analyza1 z programu MS Excel 2000

Po pojmenování zdroje údajů a výběru zprostředkovatele, v našem případě jsme ponechali předvolenou možnost Microsoft OLE DB Provider for OLAP Services, stiskneme tlačítko PŘIPOJIT, čímž se dostaneme do soustavy dialogů pro zadání údajů potřebných pro vybraného zprostředkovatele. Můžeme se připojit přímo na server OLAP nebo můžeme načítat údaje krychle OLAP ze souboru. Podle toho, kterou z těchto možností se chystáme využít, nastavíme i parametry pro připojení.

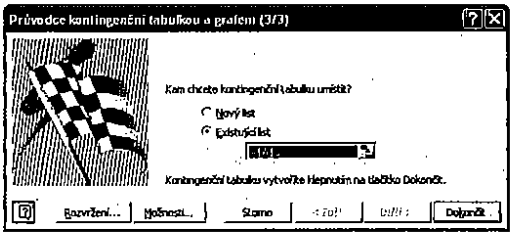
V našem případě, když se chystáme připojit k cvičné krychli SALES, kterou spravuje server OLAP, se připojíme k serveru OLAP. Definování parametrů v takovém případě spo-

čívá v zadání názvu serveru a ve výběru analytické databáze. Název analytického serveru na lokálním počítači nejjednodušeji zjistíme tak, když klepneme na sdruženou ikonku MS SQL a analytického serveru, která je zobrazena v pravém dolním rohu pracovní plochy Windows vedle symbolu hodin. Když se připojíme ke vzdálenému serveru, tak zjistíme příslušné parametry u jeho administrátora.



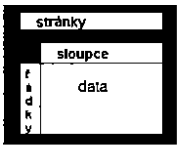
Obr. 5.6 Výběr druhu datového zdroje

Po vytvoření zdroje údajů přijde na řadu jejich zobrazení. Program MS Excel 2000 nás nenechá, abychom se trápili se zobrazováním výsledků analýzy, a nabídne nám **Průvodce kontingenční tabulkou**. O kontingenčních tabulkách (pivot table) se více dočteme v dokumentaci k MS Excel, případně z odborné literatury věnované tomuto tabulkovému procesoru. Novou kontingenční tabulku umístíme na existující list sešitu MS Excel nejčastěji do levé horní části.



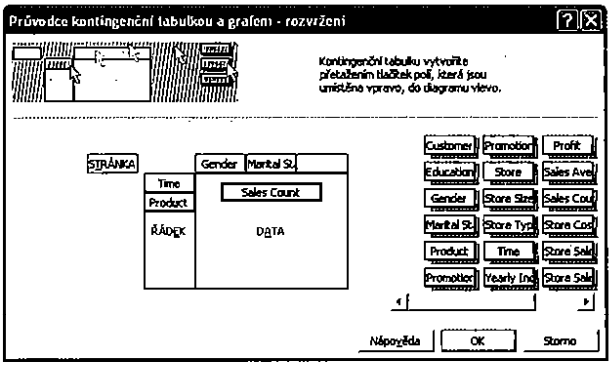
Obr. 5.7 Průvodce kontingenční tabulkou a grafem

V této fázi máme dvě možnosti. Můžeme po stisku tlačítka **Rozložení** využít dalšího průvodce, s jehož pomocí vytvoříme konkrétní strukturu kontingenční tabulky. Naše činnost při návrhu tabulky je zredukovaná na přesouvání ikonek s názvy faktů a dimenzí z levé části dialogu do jednotlivých polí návrhového formuláře. V návrhovém formuláři, ať už průvodci, nebo přímo v kontingenční tabulce máme pole pro stránky, řádky, sloupce, tedy položky z tabulek dimenzí, a pole pro údaje, tedy položky z tabulky faktů.

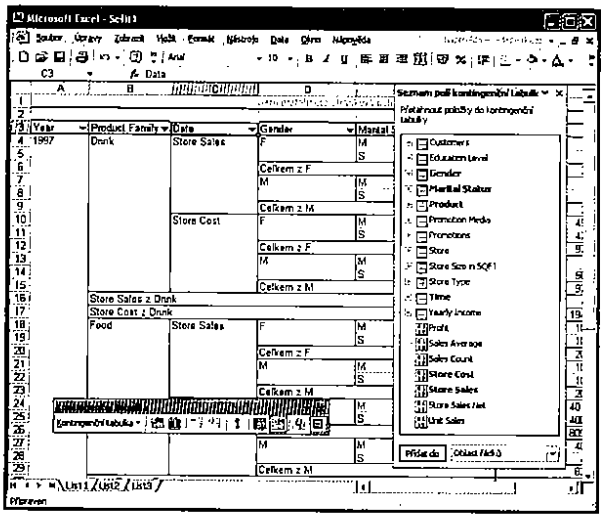


Obr. 5.8 Pole pro stránky, řádky, sloupce a data

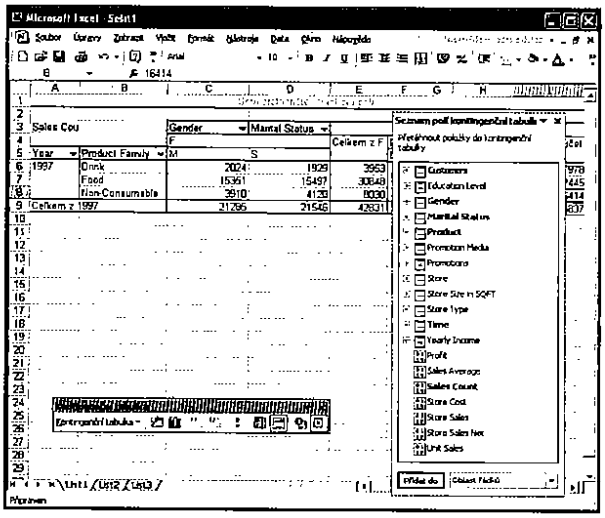
Po připojení ke cvičné krychli SALES, která se nainstaluje při instalaci analytických služeb MS SQL Serveru 2000, můžeme vytvořit například takový jednoduchý návrh.



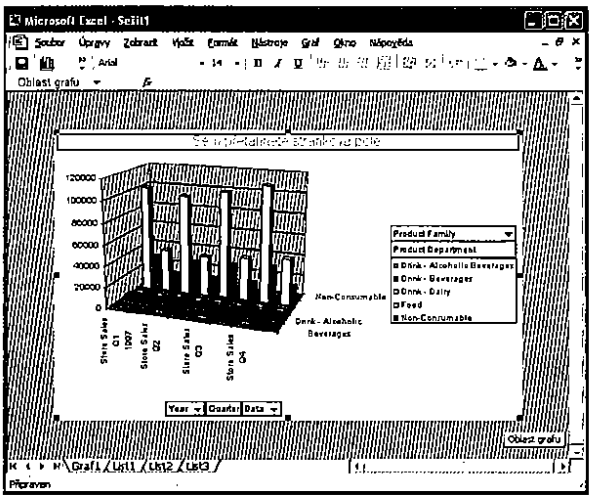
Obr. 5.9 Průvodce rozložením polí kontingenční tabulky



Obr. 5.10 Výsledek návrhu kontingenční tabulky

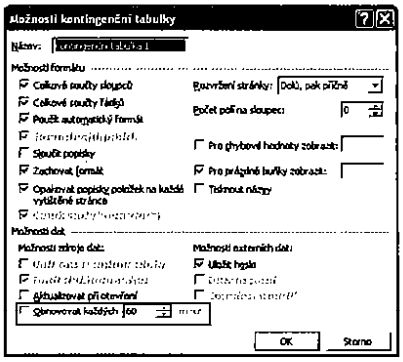


Obr. 5.11 Interaktivní návrh kontingenční tabulky



Obr. 5.15 Zobrazení výsledků analýzy ve formě grafu

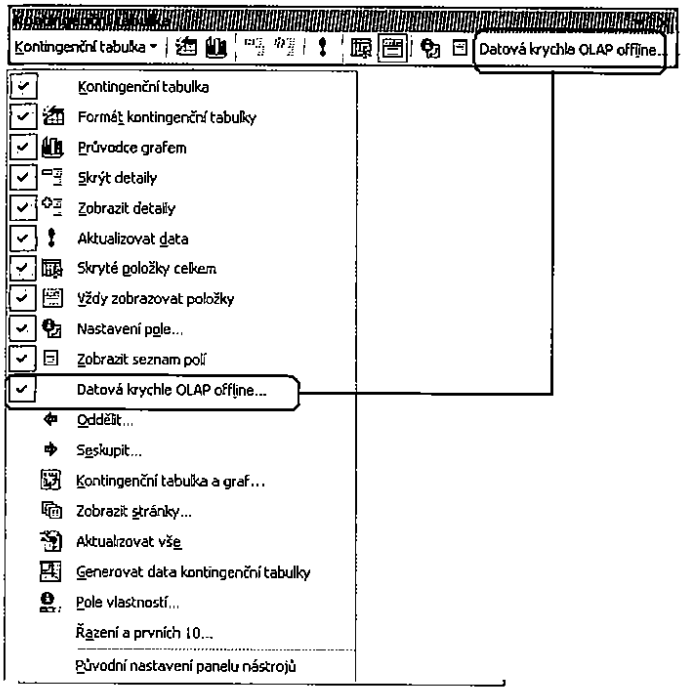
Musíme si uvědomit, že Excel vystupuje vůči serveru OLAP jako klient, což znamená, že když uložíme pracovní list a později ho otevřeme a budeme chtít například měnit rozsah nebo přesnost dimenzí, podaří se nám to jen za předpokladu, že budeme mít spojení na server OLAP. Když si zobrazíme po stisku levého tlačítka myši dialog pro nastavení parametrů, můžeme například nastavit, aby se po každém otevření listu jeho obsah aktualizoval nebo automatickou periodickou aktualizaci v nastaveném časovém intervalu.



Obr. 5.16 Nastavení periodické aktualizace kontingenční tabulky

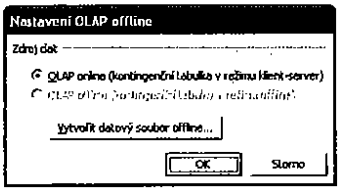
Práce s údaji OLAP v režimu offline

Krychle OLAP uložené v souborech s příponou .CUB na pevném disku počítače nám umožní pracovat v režimu offline, tedy i po odpojení od serveru OLAP. Na základě těchto údajů můžeme vytvořit sestavu kontingenční tabulky nebo kontingenčního grafu. Když chceme pracovat s OLAP v režimu offline, tak je výhodné přidat tlačítko do toolbaru kontingenční tabulky. Klepneme na šipku směřující dolů v pravém horním rohu panelu kontingenční tabulky (přidat nebo odstranit tlačítka) a přidáme tlačítko *„OLAP v režimu offline"*.



Obr. 5.17 Přidání tlačítka OLAP v režimu offline do toolbaru kontingenční tabulky

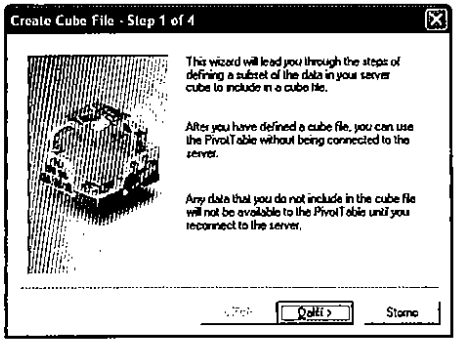
Po aktivování tlačítka *„OLAP v režimu offline"* se zobrazí úvodní dialog pro vytvoření souboru s údaji.



Obr. 5.18 Dialog pro nastavení OLAP v režimu offline

Když soubor s údaji ještě nemáme, klepneme na tlačítko „*Vytvořit soubor s údaji v režimu offline*". Když soubor údajových krychlí v režimu offline pro sestavu už existuje, tak klepneme na tlačítko „*Upravit soubor s údaji v režimu offline*".

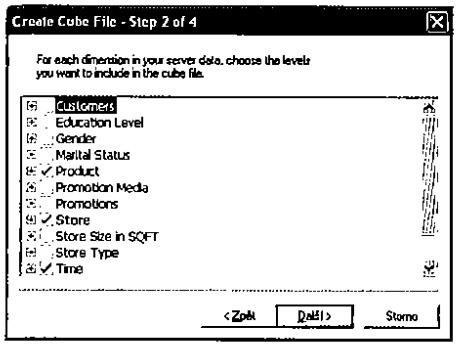
Následně se aktivuje průvodce vytvořením souborů krychle pro režim offline.



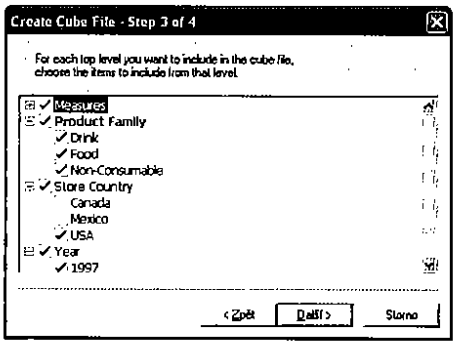
Obr. 5.19 Dialog pro nastavení OLAP v režime offline

V prvním kroku průvodce Offline Cube Wizard si případně přečteme pokyny a klepneme na tlačítko „Dále".

V druhém kroku průvodce vybereme všechny dimenze, tedy struktury OLAP, ve kterých jsou údaje uspořádány do hierarchických úrovní z krychle serveru s údaji, které chceme zahrnout do souboru údajových krychlí v režimu offline. Dimenzi vybereme označením checkboxu. Pomocí tlačítek (+) můžeme rozvinout jednotlivé dimenze na požadované hierarchické úrovně. Není samozřejmě možné vynechat střední úrovně v rámci dimenze. Když potřebujeme zmenšit velikost souboru údajových krychlí nižší úrovně, které není potřebné zobrazit v sestavě, jednoduše je vynecháme. Do údajové krychle ale musíme zahrnout všechny dimenze, ve kterých jsou seskupeny položky, abychom se později mohli bez problémů přepínat mezi databází serveru a souborem v režimu offline.

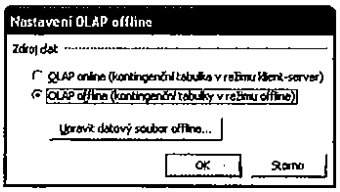


Obr. 5.20 Výběr dimenzí



Obr. 5.21 Výběr měřítek a úrovní dimenzí

V třetím kroku průvodce můžeme rozvinout měřítko (Measures) a vybrat pole, která se mají v sestavě použít jako údajová pole. Musíme vybrat alespoň jednu položku. Potom vybereme konkrétní položky jednotlivých dimenzí, které se mají zahrnout do souboru údajových krychlí v režimu offline. V posledním dialogu necháme potvrzenou volbu OLAP v režimu offline (v čase bez připojení použít sestavu kontingenční tabulky).



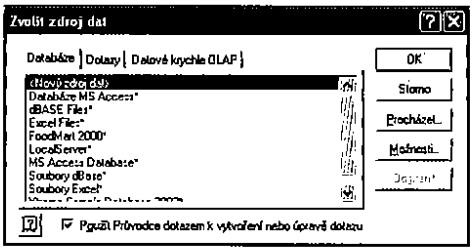
Obr. 5.22 Dialog pro nastavení OLAP v režimu offline

V posledním, čtvrtém kroku průvodce zadáme název a umístění pro soubor typu .cub a klepneme na tlačítko Finish. V našem případě bylo umístění souboru krychle *C:\Documents and Settings\Luboslav Lacko\My Documents\Sales.cub*. Nakonec uložíme příslušný sešit programu MS Excel.

Vytvoření krychle ze záznamů v dotazu

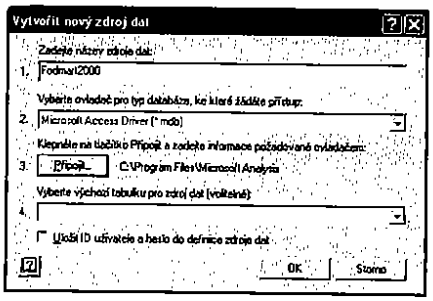
Pro některé typy zdrojových údajů je možné vytvořit krychli OLAP ze záznamů vrácených dotazem SQL. Vytvořením krychle OLAP z dotazu SQL se hierarchicky nečleněná množina záznamů změní na soubor s hierarchickou strukturou, která umožňuje soustředit se v sestavě na požadovanou úroveň detailů. V našem příkladě ukážeme vytvoření krychle OLAP nad databází FoodMart 2000, konkrétně se připojíme k databázi MS Access, tedy souboru FoodMart 2000.MDB. Tento soubor se po nainstalování analytických služeb nachází v adresáři *C:\Program Files\Microsoft Analysis Services\Samples*.

V programu MS Excel otevřeme menu *Údaje – Importovat externí údaje – Nový databázový dotaz*. V zobrazeném dialogu vybereme záložku Databáze. V dolní části dialogu ponecháme zaškrtnutou volbu Použít průvodce dotazem k vytvoření nebo úpravě dotazu.



Obr. 5.23 Výběr zdroje údajů

V dialogu pro vytvoření nového zdroje údajů zadáme název zdroje údajů, například Foodmart2000. V druhém poli dialogu vybereme ovladač pro Microsoft Access. Po stisku tlačítka Připojit zadáme cestu k databázovému souboru, v našem případě C:\Program Files\Microsoft Analysis Services\Samples\foodmart 2000.mdb.



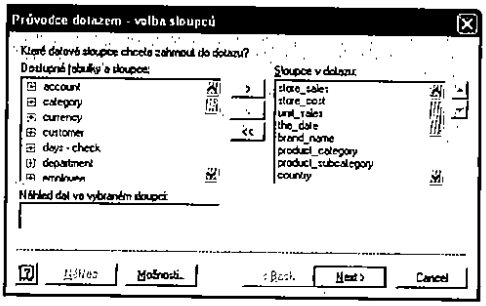
Obr. 5.24 Vytvoření nového databázového zdroje

Vraťme se ale do dialogu pro výběr zdroje údajů. Po klepnutí na název Foodmart2000 aktivujeme průvodce vytvořením dotazu.

Průvodce vytvořením dotazu SQL

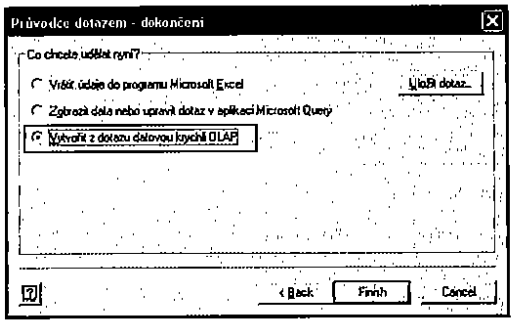
V prvním dialogu průvodce vybereme tabulky a sloupce, které chceme zahrnout do dotazu. V této fázi vybereme sloupce budoucích tabulek faktů a tabulek dimenzí, které přesouváme do okna „Sloupce v dotazu". Můžeme vybrat například tyto sloupce:

TABULKA	SLOUPCE
sales_fact_1998	store_sales, store_cost, unit_sales
time_by_day	the_date
Product	brand_name
Product_class	product_category, product_subcategory
customer	country, state_province, city, lname
Store	store_country, store_state, store_city, store_name



Obr. 5.25 Výběr sloupců pro vytvoření dotazu

V následujících dvou dialozích můžeme případně nastavit filtrování a setřídění údajů. Když filtrování nebo setřídění nepožadujeme, přeskočíme příslušné dialogy tlačítkem Next. Důležitý je poslední dialog průvodce vytvořením dotazu, kde zaškrtneme volbu „Vytvořit z dotazu datovou krychli OLAP".

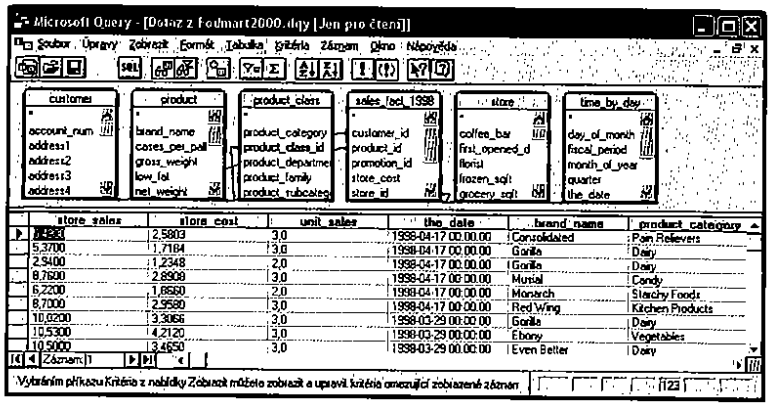


Obr. 5.26 Výběr sloupců pro vytvoření dotazu

Když si necháme zobrazit vytvořený dotaz, tak vidíme, že průvodce nám ušetřil poměrně mnoho námahy a samozřejmě eliminoval i případné chyby z nepozornosti při jeho „ručním" vytvoření.

```
SELECT sales_fact_1998.store_sales, sales_fact_1998.store_cost,
       sales_fact_1998.unit_sales, time_by_day.the_date, product.brand_name,
       product_class.product_category, product_class.product_subcategory,
       customer.country, customer.state_province, customer.city, customer.lname,
       store.store_country, store.store_state, store.store_city, store.store_name
FROM customer customer, product product, product_class product_class,
     sales_fact_1998 sales_fact_1998, store store, time_by_day time_by_day
WHERE customer.customer_id = sales_fact_1998.customer_id AND
      product.product_class_id = product_class.product_class_id AND
      product.product_id = sales_fact_1998.product_id AND
      sales_fact_1998.time_id = time_by_day.time_id AND
      sales_fact_1998.store_id = store.store_id
```

Dotaz je možné vytvořit i v programu Microsoft Query. Tento program zahrnuje všechna pole, která v krychli chcete použít, a pro případ možných budoucích změn ukládá dotaz do souboru s příponou .dqy. Krychli potom vytvoříme spuštěním průvodce OLAP Cube Wizard v menu Soubor programu Microsoft Query.



Obr. 5.27 Vytvoření dotazu v programu Microsoft Query

Průvodce vytvořením datové krychle OLAP

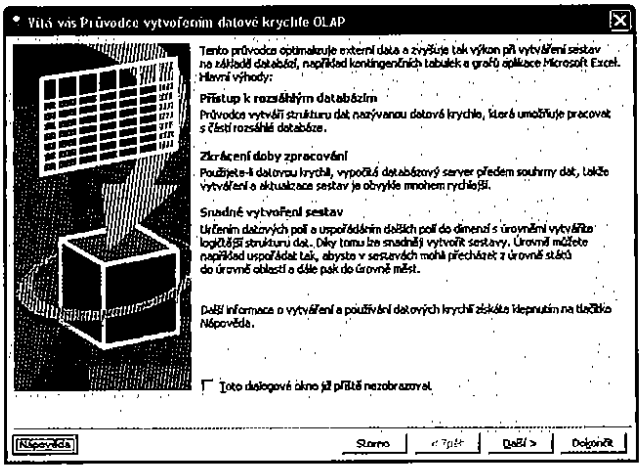
Na základě dotazu SQL je možné vytvořit krychli OLAP pomocí průvodce vytvořením datové krychle OLAP. Průvodce umožňuje vytváření dvou typů krychlí.

Prvním typem je definice údajové krychle, která potom bude uložena do souboru s příponou .OGY. Po otevření sestavy založené na tomto typu souboru se údajová krychle dočasně vytvoří v paměti.

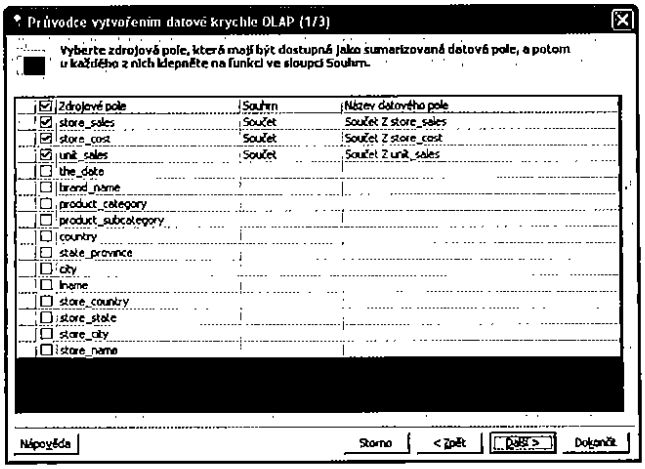
Druhým typem je samostatný soubor údajových krychlí v režimu offline, který umožňuje pokračovat v práci s údaji i po odpojení ze sítě. Při aktualizaci původní databáze je možné obnovit oba dva typy krychlí a zahrnout do nich nové nebo změněné údaje, které splňují kritéria původního dotazu. Do krychle ale není možné přidat další pole z původní databáze nebo dotazu. Je potřeba přidat další pole, neboť musíme otevřít a upravit původní soubor dotazu SQL s příponou .DGY v programu Microsoft Query a následně opětovným spuštěním průvodce OLAP Cube Wizard vytvořit novou krychli.

Fakta

V prvním kroku průvodce vybíráme pole, která budou používána jako datová pole (fakta), a definujeme agregační funkce (Maximum, Minimum, Počet a Součet) pro výpočty hodnot v polích. Před vytvořením krychle OLAP si musíme samozřejmě promyslet, které sloupce obsahují hodnoty, ze kterých budeme vytvářet souhrny, a tyto sloupce umístíme do datového pole. Pole, která obsahují popisné údaje, použijeme jako dimenze. Všechna vytvořená pole je potřeba v tomto kroku pojmenovat. Jako datová pole jsme v našem příkladě vybrali sloupce store_sales, store_cost, a unit_sales. Jako agregační funkce byla v každém případě vybrána funkce Souhrn (SUM).



Obr. 5.28 Průvodce vytvořením datové krychle OLAP



Obr. 5.29 Výběr datových polí

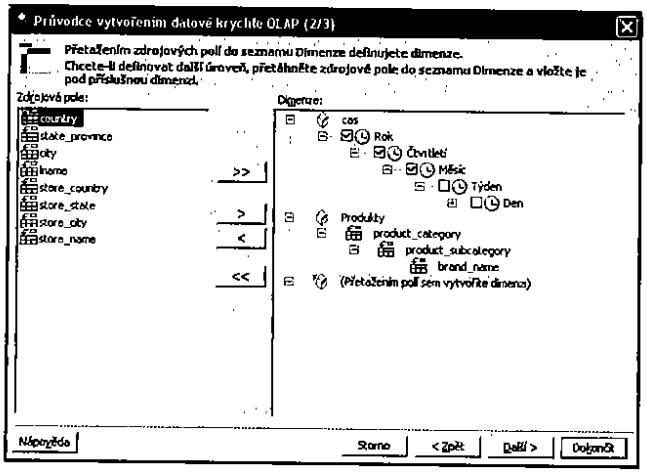
V druhém kroku průvodce uspořádáme ostatní pole do dimenzí. Navrhneme jejich hierarchickou strukturu a jednotlivé dimenze a úrovně pojmenujeme.

Časová dimenze

Přemístíme sloupec *the_date* ze zdrojového pole do pole dimenzí. Ve vzniklé hierarchii časové dimenze vymažeme zaškrtávací pole u hierarchických úrovní Týden a Den. Název dimenze můžeme přejmenovat například na *Čas*.

Produktová dimenze

Hierarchie produktové dimenze budou tvořit sloupce *product_category, product_subcategory,* a *brand_name*. Dimenzi přeměníme například na *Produkty*.



Obr. 5.30 Návrh hierarchií dimenzí; na obrázku je stav po návrhu časové a produktové dimenze

Zákaznická dimenze

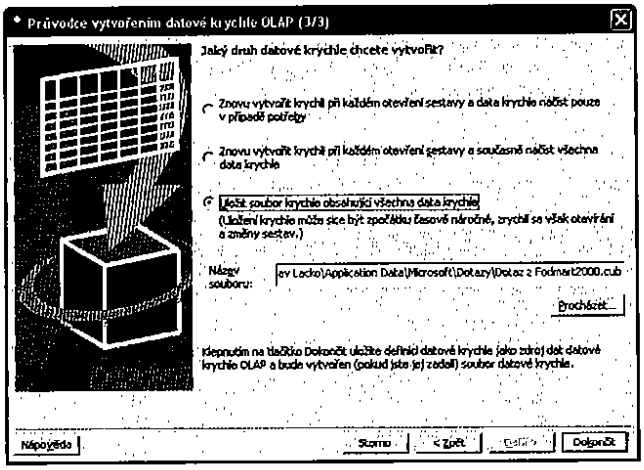
Do zákaznické dimenze přesuneme ikony sloupců *country, state_province, city* a *lname*. Dimenzi pojmenujeme *Zákazníci*.

Dimenze prodeje

Dimenzi struktury prodejní sítě budou tvořit sloupce store_country, *store_state, store_city* a *store_name*. Dimenzi pojmenujeme například *Prodej*.

V třetím kroku průvodce můžeme určit, zda bude krychle OLAP umístěna v paměti, nebo v souboru.

Vytvořená krychle OLAP bude v našem případě umístěna v souboru *C:\Documents and Settings\Luboslav Lacko\Application Data\Microsoft\Dotazy\Dotaz z Fodmart2000.cub*.



Obr. 5.31 Závěrečný dialog průvodce vytvořením krychle OLAP

Po ukončení práce s průvodcem je možné návrh krychle OLAP kdykoliv změnit. Můžeme například přidat nebo odebrat pole, změnit způsob vytváření souhrnů z datových polí nebo změnit dimenze.

Když si prohlédneme obsah souboru vytvořeného průvodcem (v našem případě *Dotaz z Fodmart2000.oqy*), tak zjistíme, že tento textový soubor obsahuje metadata a skládá se ze čtyř částí.

V první části je specifikovaný typ dotazu a řetězec pro připojení.

```
QueryType=OLEDB
Version=1
CommandType=Cube
Connection=Provider=MSOLAP;
Initial Catalog=[OCWCube];
Data Source=C:\Documents and Settings\Luboslav Lacko\Application
Data\Microsoft\Dotazy\Dotaz z Fodmart2000.cub;
```

V druhé části je příkaz pro vytvoření krychle.

```
CREATE CUBE [OCWCube]
(  DIMENSION [cas] TYPE TIME, LEVEL [Vše] TYPE ALL, LEVEL [Rok] TYPE YEAR,
             LEVEL [Čtvrtletí] TYPE QUARTER, LEVEL [Měsíc] TYPE MONTH,
   DIMENSION [Produkty], LEVEL [Vše] TYPE ALL,  LEVEL [product_category],
             LEVEL [product_subcategory], LEVEL [brand_name],
   DIMENSION [Zakaznici], LEVEL [Vše] TYPE ALL, LEVEL [country],
             LEVEL [state_province], LEVEL [city], LEVEL [lname],
```

```
   DIMENSION [Predaj],   LEVEL [Vše] TYPE ALL,  LEVEL [store_country],
             LEVEL [store_state], LEVEL [store_city], LEVEL [store_name],
   MEASURE [Součet Z store_sales] FUNCTION SUM,
   MEASURE [Součet Z store_cost]  FUNCTION SUM,
   MEASURE [Součet Z unit_sales]  FUNCTION SUM
);
```

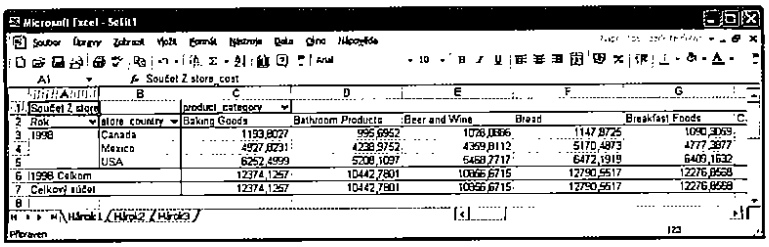
V třetí části je příkaz pro naplnění krychle údaji. Použita je konstrukce INSERT INTO – SELECT.

```
INSERT INTO OCWCube
 ([Součet Z store_sales], [Součet Z store_cost], [Součet Z unit_sales],
  [cas],
  [brand_name], [product_category], [product_subcategory],
 [country], [state_province], [city], [lname],
 [store_country], [store_state], [store_city], [store_name])
OPTIONS ATTEMPT_ANALYSIS
SELECT sales_fact_1998.store_sales,
       sales_fact_1998.store_cost,
       sales_fact_1998.unit_sales,
       time_by_day.the_date,
       product.brand_name, product_class.product_category,
       product_class.product_subcategory,
       customer.country, customer.state_province, customer.city, customer.lname,
       store.store_country, store.store_state, store.store_city, store.store_name
  FROM customer customer, product product, product_class product_class,
       sales_fact_1998 sales_fact_1998, store store, time_by_day time_by_day
  WHERE customer.customer_id = sales_fact_1998.customer_id AND
        product.product_class_id = product_class.product_class_id AND
        product.product_id = sales_fact_1998.product_id AND
        sales_fact_1998.time_id = time_by_day.time_id AND
        sales_fact_1998.store_id = store.store_id;
```

Ve čtvrté části je definované připojení ke zdrojové databázi.

```
Source_DSN="DBQ=C:\Program Files\Microsoft Analysis Services\Samples\foodmart
2000.mdb;DefaultDir=C:\Program Files\Microsoft Analysis
Services\Samples;Driver={Microsoft Access Driver (*.mdb)};DriverId=25;FIL=MS
Access;MaxBufferSize=2048;MaxScanRows=8;PageTimeout=5;SafeTransactions=0;Threads=3
;UID=admin;UserCommitSync=Yes;":
UseExistingFile=True
```

Po vytvoření krychle OLAP pokračuje program MS Excel nám už známým průvodcem vytvořením kontingenční tabulky.



Součet z store_cost		product_category					
Rok	store_country	Baking Goods	Bathroom Products	Beer and Wine	Bread	Breakfast Foods	C...
1998	Canada	1193,8027	995,6952	1076,0886	1147,8725	1090,3069	
	Mexico	4927,8231	4238,9752	4359,8112	5170,4873	4777,3977	
	USA	6252,4999	5208,1097	5468,7717	6472,1918	6409,1632	
1998 Celkem		12374,1257	10442,7801	10866,6715	12790,5517	12276,8588	
Celkový součet		12374,1257	10442,7801	10866,6715	12790,5517	12276,8588	

Obr. 5.32 Kontingenční tabulka v programu MS Excel vytvořená ze souboru OLAP krychle

Inteligentní značky (SmartTags) v programech balíku Microsoft Office

Některé dosud probrané typy klientských aplikací jsou dost složité a nemělo tedy smysl je poskytnout analytikům a manažerům, což jsou většinou lidé zběhlí například v ekonomice, financích a obchodu, ale ne příliš zběhlí v informačních technologiích; „inteligentní značky" (Smart Tags) si tito lidé zaručeně oblíbí. Samozřejmě nemá smysl, aby se administrátor nebo specialista na analýzy OLAP ptal manažerů a analytiků, zda nechtějí Smart Tags. Odpověď, které bychom se dočkali, by byla ve většině případů typu: „Smart co...?" Ale když tuto funkcionalitu příslušnému okruhu manažerů například na pracovní poradě předvedeme, úspěchem si můžeme být skoro jisti.

Inteligentní značky

Inteligentní značky se používají při psaní a práci s textem v programech kancelářského balíku MS Office. V praxi to funguje následovně. Když píšeme nebo editujeme text, označí se některá slova nebo části slov podtržením fialovou tečkovanou čárou. Mohou to být například jména z adresáře, názvy programů a podobně. Potom můžeme pomocí inteligentních značek vykonávat jednoduchým způsobem úkony, které bychom jinak pracně realizovali v jiných aplikacích, například posílání pošty, přidávání adresářů do kontaktů a podobně.

Nás bude vzhledem k zaměření publikace zajímat hlavně použití inteligentních značek v oblasti Business Intelligence. Použití je až triviálně jednoduché. Komponentu Smart Tags máme napojenou na vhodnou analytickou databázi (to zajistí manažerovi databázový administrátor) a můžeme začít pracovat s textem. V námi vytvářeném nebo editovaném textu budeme mít označené všechny pojmy, které jsou zároveň uloženy v analytické databázi, ať už jako dimenze, měřítka nebo údaje. Když například píšeme v textovém editoru MS Word zprávu o ekonomických výsledcích naší firmy, případně o prodeji produktů a podobně, budeme mít označeny všechny názvy produktů, filiálek firmy, jména manažerů a zaměstnanců, samozřejmě jen za předpokladu, že se nacházejí v databázi. Potom stačí nad takovým pojmem klepnout a nechat si zobrazit příslušnou kontingenční tabulku nebo graf. Výsledek analýzy potom podle svého uvážení můžeme do připravované zprávy opsat či vysvětlit, což bude nejčastější v případě, když máme problémy a potřebujeme je

zdůvodnit. Názornější a jednodušší je druhá možnost – příslušnou tabulku nebo ještě lépe graf přímo vložit do připravovaného textu. Ale nepředbíhejme, nejdříve musíme pro tento účel nainstalovat balík Business Intelligence Smart Tag Wizard.

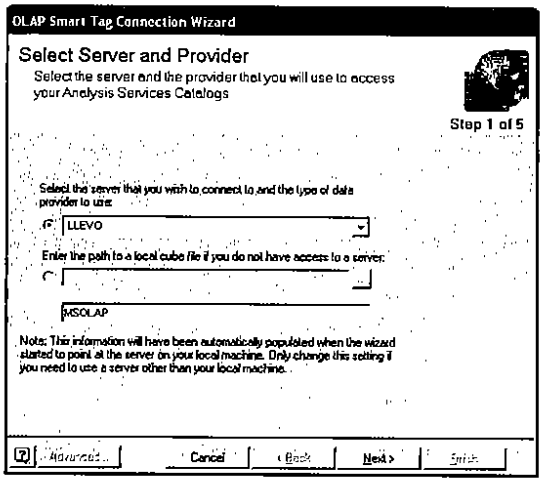
Business Intelligence Smart Tag

Tento doplněk ke kancelářskému balíku MS Office můžeme stáhnout z webu z adresy:

```
http://msdn.microsoft.com/downloads/sample.asp?url=/msdn-files/027/002/096/
MsdnCompositeDoc.xml
```

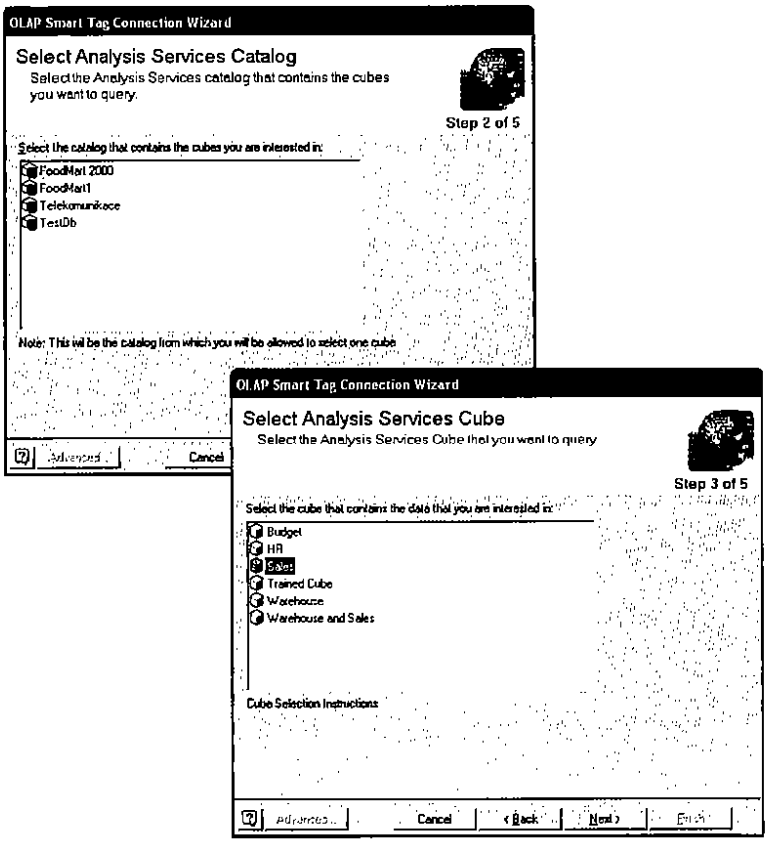
Protože se jedná o komponentu, kterou mohou nainstalovat a používat i manažeři a analytici, kterým pojem administrace databází nic neříká, tak popíšeme jednotlivé kroky instalace podrobněji. Instalace spočívá ve stáhnutí a spuštění instalačního souboru bi_st.msi na příslušném lokálním počítači. Po nainstalování je potřebné napojit komponentu na příslušnou analytickou databázi, například na FoodMart 2000. V hlavním menu operačního systému Windows zvolíme položku *Programs, – Microsoft BI Smart Tag – BI Smart Tag Wizard.*

Jak vidíme v záhlaví, konfigurační průvodce se skládá z pěti kroků. V prvním kroku nastavujeme připojení na analytický server, případně na údaje v lokální krychli (soubory s příponou .CUB).



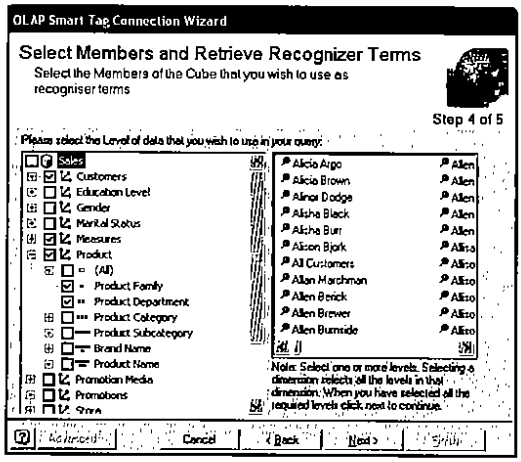
Obr. 5.33 BI Smart Tag Wizard – připojení k analytické databázi

Následuje dialog pro výběr příslušné analytické databáze a krychle ze zvolené analytické databáze, na kterou bude komponenta Smart Tag napojena a odkud bude čerpat klíčová slova a výsledky analýz.



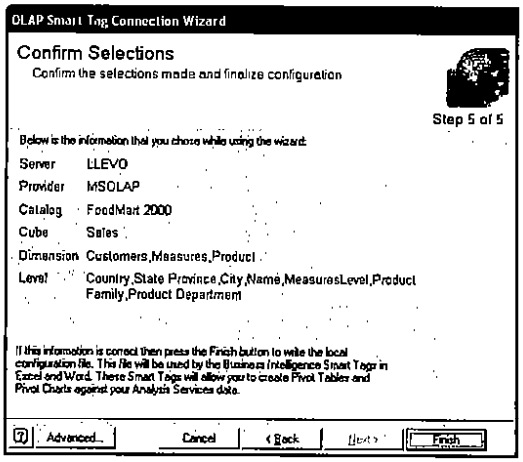
Obr. 5.34 BI Smart Tag Wizard – výběr krychle z analytické databáze

Největší pozornost je potřeba věnovat čtvrtému konfiguračnímu dialogu, kde označujeme jednotlivé prvky dimenzí, jejichž názvy se budou ve zpracovávaném textu označovat inteligentními značkami.



Obr. 5.35 BI Smart Tag Wizard – výběr dimenzí, na jejichž základě se budou označovat slova v textu

Poslední, pátý dialog je v podstatě potvrzovací. Před definitivní změnou konfigurace můžeme zadané údaje zkontrolovat a případně se vrátit zpět do příslušného dialogu a změnit nesprávně zadané údaje.



Obr. 5.36 BI Smart Tag Wizard – potvrzení nastavené konfigurace

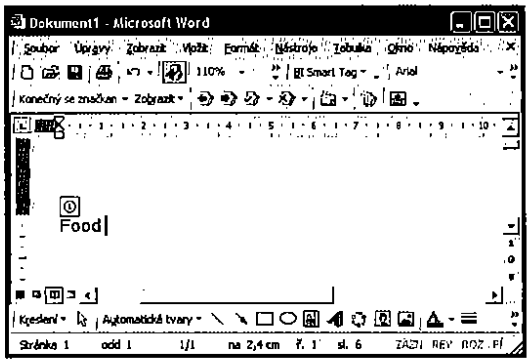
Konfigurační údaje budou uloženy do souboru XML BITerms.XML ve tvaru

```
<rs:data>
  <rs:insert>
    <z:row Server='LLEVO' Provider='MSOLAP' Catalog='FoodMart 2000' Cube='Sales'
    Dimension='Customers' Level='Country,State Province,City,Name'/>
    <z:row Server='LLEVO' Provider='MSOLAP' Catalog='FoodMart 2000' Cube='Sales'
    Dimension='Measures' Level='MeasuresLevel'/>
    <z:row Server='LLEVO' Provider='MSOLAP' Catalog='FoodMart 2000' Cube='Sales'
    Dimension='Product' Level='Product Family,Product Department'/>
    </rs:insert>
</rs:data>
```

Nakonec budeme ještě upozorněni, že příslušné změny konfigurace se projeví až po ukončení a opětovném spuštění programů MS Word a MS Excel, když jsme je měli během konfigurace spuštěné.

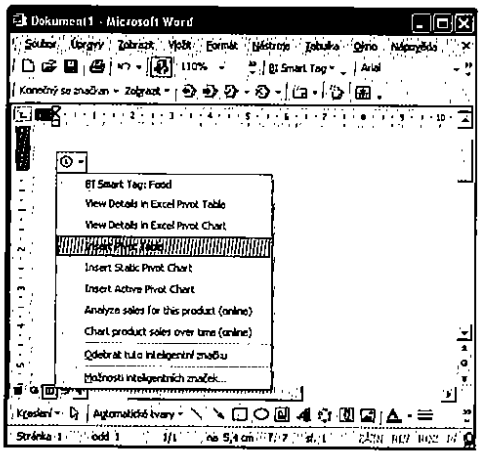
Použití BI Smart Tag v programu Microsoft Word

Když v textovém editoru Microsoft Word napíšeme slovo, které je názvem některé dimenze, nebo když taková slova text už obsahuje, jsou podtržené tenkou tečkovanou fialovou čárou. Pokud nás pro některý takto označený pojem zajímají výsledky analýzy, tak přesuneme na označené slovo kurzor myši a zobrazí se ikona pro inteligentní značky (písmeno i v kroužku). Další postup je následující. Přesuneme kurzor myši ze slova na ikonu inteligentní značky. Následně se vedle ikony inteligentní značky zobrazí ikona combo boxu (šipka dolů).



Obr. 5.37 Inteligentní značka nad slovem Food, které představuje jednu z dimenzí

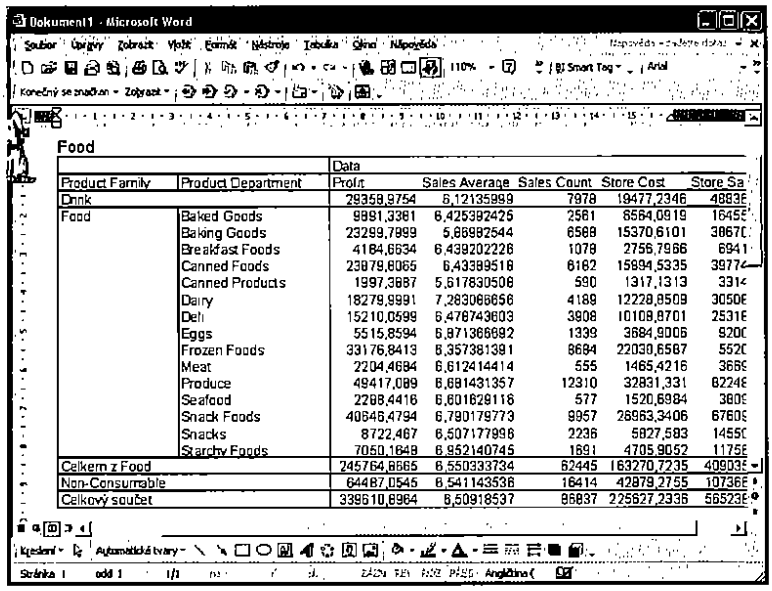
Po klepnutí na tuto ikonu se zobrazí kontextové menu inteligentních značek. V našem případě jsme poklepali na slovo Food, které představuje jednu z dimenzí v krychli Sales.



Obr. 5.38 Kontextové menu pro inteligentní značku

Nabízené možnosti jsou poměrně pestré. Můžeme si prohlédnout kontingenční tabulku nebo kontingenční graf v programu MS Excel. Tuto možnost využijeme tehdy, když nepotřebujeme uvádět fakta přímo do textu v „surové" formě, ale vypracovat vyhodnocení nebo analýzu aktuální situace. Zjednodušeně řečeno – prohlédneme si graf nebo tabulku a na základě takto získaných informací napíšeme příslušný blok textu týkající se dané problematiky. V některých případech je potřebné vložit příslušnou tabulku nebo graf přímo do textu.

Všimněme si na obrázku nástrojové lišty programu MS Word. Po nainstalování komponenty BI Smart Tag sem přibyl combo box Smart Tag. Kromě nápovědy obsahuje položku menu pro konfiguraci BI Smart Tag a položku Current Term List, s jejíž pomocí aktivujeme dialog se seznamem všech slov, na které se budou potenciálně aktivovat inteligentní značky.

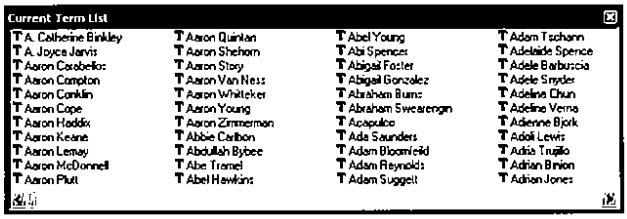


Dokument1 - Microsoft Word

Food

		Data				
Product Family	Product Department	Profit	Sales Average	Sales Count	Store Cost	Store Sa
Drink		28358,9754	6,12135999	7978	19477,2346	48835
Food	Baked Goods	9891,3381	6,425392425	2561	6564,0919	16455
	Baking Goods	23299,7999	5,86982544	6568	15370,6101	38670
	Breakfast Foods	4184,6634	6,439202228	1078	2756,7966	6941
	Canned Foods	23879,8065	6,43389518	6182	15894,5335	39774
	Canned Products	1997,3887	5,617830508	590	1317,1313	3314
	Dairy	18279,9991	7,283086656	4189	12228,8509	30508
	Deli	15210,0599	6,476743603	3908	10108,8701	25318
	Eggs	5515,8594	6,871366982	1339	3684,9006	9200
	Frozen Foods	33176,8413	8,357381381	8684	22038,6587	5520
	Meat	2204,4684	6,612414414	555	1465,4216	3669
	Produce	49417,089	6,681431357	12310	32831,331	82248
	Seafood	2288,4416	6,601628118	577	1520,6984	3809
	Snack Foods	40646,4794	6,790179773	9957	26963,3406	67609
	Snacks	8722,467	6,507177998	2236	5827,593	14550
	Starchy Foods	7050,1648	6,952140745	1691	4705,9052	11756
Celkem z Food		245764,8665	6,550333734	62445	163270,7235	409035
Non-Consumable		64487,0545	6,541143536	16414	42879,2755	107366
Celkový součet		339610,8964	6,50918537	86837	225827,2336	565238

Obr. 5.39 Vložení kontingenční tabulky do textu v programu MS Word pomocí menu Inteligentní značky



Current Term List

A. Catherine Binkley	Aaron Quinlan	Abel Young	Adam Tschann
A. Joyce Jarvis	Aaron Shehorn	Abi Spencer	Adelaide Spence
Aaron Carabello	Aaron Story	Abigail Foster	Adele Barbuscia
Aaron Compton	Aaron Van Ness	Abigail Gonzalez	Adele Snyder
Aaron Conklin	Aaron Whiteker	Abraham Burns	Adeline Chun
Aaron Cope	Aaron Young	Abraham Swearengen	Adeline Verna
Aaron Haddix	Aaron Zimmerman	Acapulco	Adienne Bjork
Aaron Keane	Abbie Carlson	Ada Saunders	Adoli Lewis
Aaron Lemay	Abdullah Bybee	Adam Bloomfield	Adria Trujillo
Aaron McDonnell	Abe Tramel	Adam Reynolds	Adrian Brion
Aaron Plutt	Abel Hawkins	Adam Suggett	Adrian Jones

Obr. 5.40 Dialog Current Term List

V našem případě mají drtivou převahu jména zákazníků. Je to výhodné, když například budeme vybavovat stížnost zákazníka nebo jeho žádost o věrnostní nebo kreditní kartu, tak si lehce zjistíme, s kým máme co do činění. Když do textu napíšeme jméno Aaron Quinlan (musí být kompletní, tedy jméno i příjmení), tak si můžeme při vybavování jeho záležitostí nechat v programu Excel zobrazit tabulku s jeho „parametry", případně ji vložit přímo do textu.

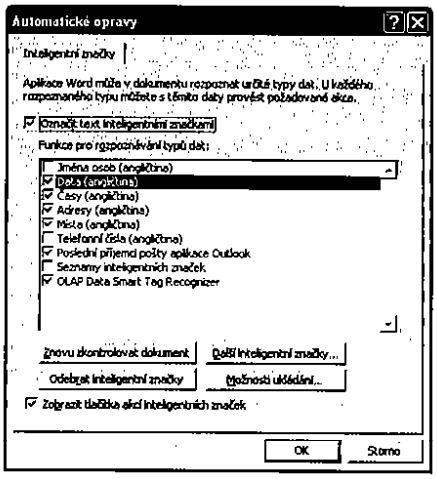
PT(1) For Aaron Quintan

Country	State	Sales Amount	Sales Average	Sales Count	Store Cost	Store Sales
USA	CA	95637.4149	6.512063	24442	63530.4	159167.8
	OR	85504.5694	6.583549	21611	56772.5	142277.1
	WA	158468.9121	6.468057	40784	105324	263793.2
Celkem z USA		339610.8964	6.509185	86837	225627	565238.1
Celkový součet		339610.8964	6.509185	86837	225627	565238.1

Když plánujeme používat inteligentní značky, tak bychom měli na tuto skutečnost myslet už při návrhu krychle OLAP, hlavně při návrhu názvů dimenzí. Například v krychli Sales je regionální nebo zákaznická dimenze navržena takto:

USA
CA
OR
WA

Těm, co znají názvy a zkratky 49 států USA, nebude takto navržená dimenze dělat žádné problémy. Potíž je s něčím jiným. Když inteligentní značky fungují tak, že se příslušné řetězce označují i tehdy, když tvoří části slov, slabiky CA OR a v určitém rozsahu i slabika



Obr. 5.41 Dialog pro nastavení parametrů inteligentních značek

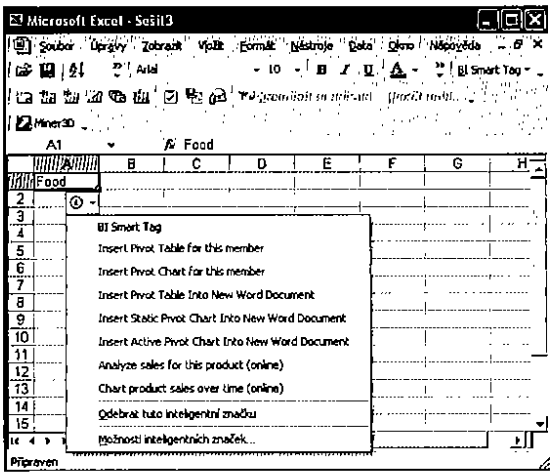
WA se vyskytují v běžném textu velmi často. Proto je z hlediska použití inteligentních značek o mnoho lepší návrh:

USA
California
Oregon
Washington

Inteligentní značky můžeme podle potřeby zapínat a vypínat v dialogu pro nastavení parametrů inteligentních značek.

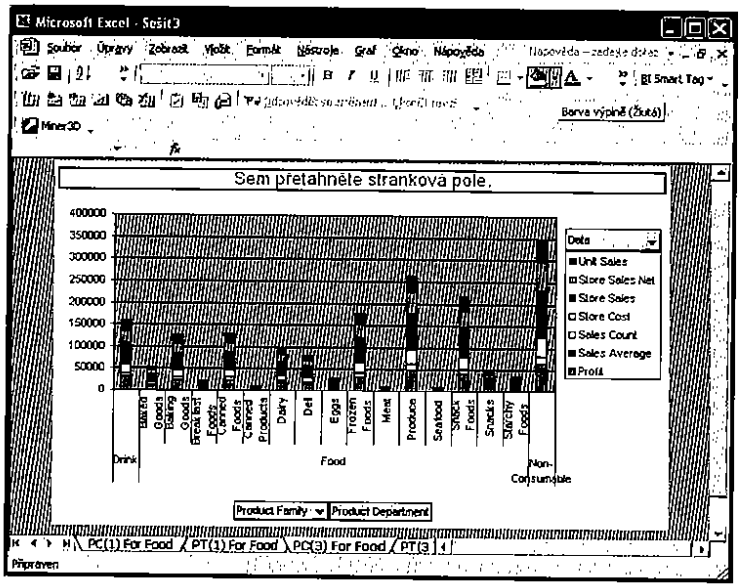
Použití BI Smart Tag v programu Microsoft Excel

Použití je úplně analogické jako v programu Microsoft Word, inteligentní značky se v tomto případě vztahují ke slovům, která jsou napsaná v jednotlivých buňkách tabulkového procesoru. Když do některé buňky tabulky napíšeme slovo, které se vyskytuje v některé dimenzi, v našem případě slovo Food, tak se zobrazí v pravém dolním rohu buňky malý fialový trojúhelník. Když na něj přesuneme kurzor myši, zobrazí se ikona inteligentní značky spolu s ikonou combo boxu pro kontextové menu. Situace s nabídkou menu je analogická jako v textovém editoru MS Word, ale jak jsme ve Wordu mohli zobrazovat tabulky a grafy, tak v Excelu je to analogické, neboť pomocí menu MS Excelu můžeme vkládat kontingenční tabulky a grafy do otevřených nebo nových dokumentů.



Obr. 5.42 Menu inteligentních značek v programu Excel

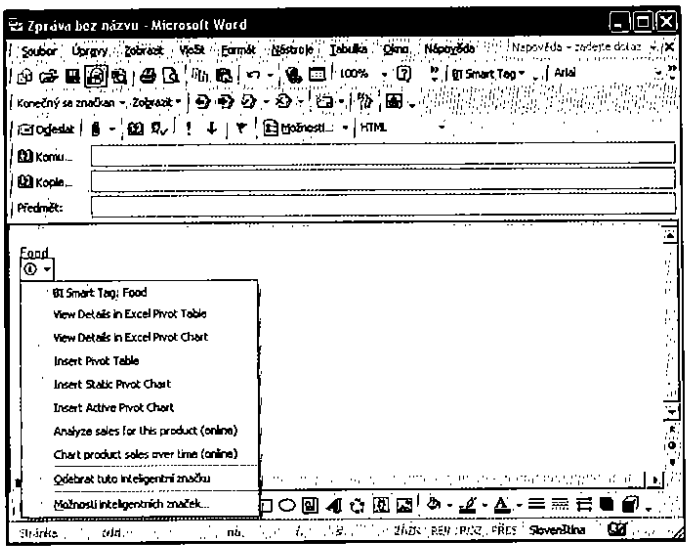
Možnost vložit do listu programu MS Excel kontingenční tabulku nebo graf s výsledky analýzy pochopitelně zůstává.



Obr. 5.43 Vložení grafu do pracovního listu programu Excel

Použití BI Smart Tag v programu Microsoft Outlook

Kromě programů MS Word a MS Excel můžeme inteligentní značky úplně podobně využívat i v programu Microsoft Outlook. Asi nejvíce tuto možnost využijeme při práci s elektronickou poštou. Když čteme e-mail, v němž se nás někdo dotazuje na určitý produkt, stačí využít inteligentní značku a výsledky analýzy pro daný produkt máme skoro okamžitě k dispozici.



Obr. 5.44 Menu inteligentních značek při psaní nového e-mailu

Praktická ukázka použití inteligentních značek

Mějme například za úlohu vypracovat posudky na činnost dvou naposledy přijatých pracovníků fiktivní firmy FoodMart. Abychom zjistili, kteří to jsou, nejdříve položíme dotaz SQL na tabulku employee.

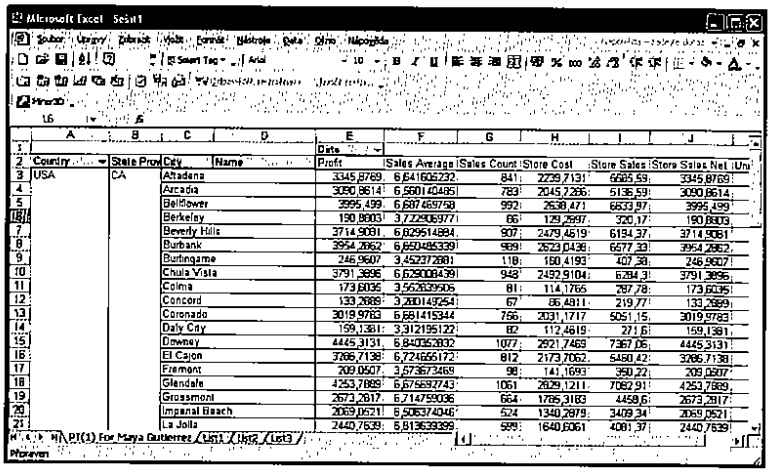
```
SELECT TOP 2  full_name, position_title, hire_date, salary
   FROM employee ORDER BY hire_date DESC:

full_name          position_title           hire_date                salary
---------------- ------------------------ ------------------------- -----------
Michael Spence     VP Country Manager       1998-01-01 00:00:00      40000.0000
Maya Gutierrez     VP Country Manager       1998-01-01 00:00:00      35000.0000
```

Když překopírujeme obě dvě jména Michael Spence i Maya Gutierrez do připravovaného textu, tak se obě jména zobrazí podtržené fialovou tečkovanou čárou. Můžeme vkládat kontingenční tabulku přímo do textu.

Michael Spence

Country	Profit	Sales Average	Sales Count	Store Cost	Store Sales	Store Sales Net
USA	339610,8964	6,50918537	86837	225627,2336	565238,13	339610,8964	266773
Celkový součet	339610,8964	6,50918537	86837	225627,2336	565238,13	339610,8964	266773

Podobně bychom získali tabulku pro Maya Gutierrez. Tato zaměstnankyně má mnohem širší rozsah působnosti, takže si necháme zobrazit kontingenční tabulku v programu MS Excel a napíšeme vyhodnocení.



Obr. 5.45 Kontingenční tabulka pro konkrétního zaměstnance zobrazená v programu MS Excel

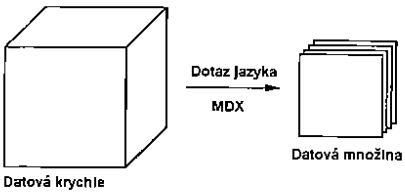
Případně do připravované zprávy vložíme jen menší fragment tabulky.

Country	State Prov	Profit	Sales Average	Sales Count	Store Cost	Store Sales
USA	CA	95637,4149	6,512062843	24442	63530,4251	159167,84
	OR	85504,5694	6,583548656	21611	56772,5006	142277,07
	WA	158468,9121	6,468056591	40784	105324,3079	263793,22
Celkem z USA		339610,8964	6,50918537	86837	225627,2336	565238,13
Celkový součet		339610,8964	6,50918537	86837	225627,2336	565238,13

Kapitola 6

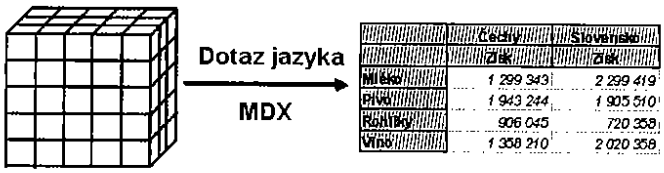
Jazyk MDX pro přístup ke krychlím OLAP

MDX je zkratka ze slovního spojení Multidimensional Expresions (multidimenzionální výrazy). Ale pěkně po pořádku. Jazyk MDX pro multidimenzionální databáze je určitým ekvivalentem jazyka SQL v relačních databázích. Jeho hlavním cílem je navigace v multidimenzionálních údajích.



Obr. 6.1 Datová krychle versus datová množina

Pomocí dotazu v MDX vypíšeme vybranou podmnožinu údajů z multidimenzionální struktury (krychle OLAP) do dvojrozměrné tabulky, která obsahuje množinu buněk, proto této struktuře říkáme Cellset.



Obr. 6.2 Princip výběru údajů pomocí dotazu v MDX

Podobně jako jazyk SQL ani jazyk MDX není cílem, ale jen prostředkem pro přístup k multidimenzionálním údajům. A to jednak prostřednictvím nějaké klientské konzoly, ale hlavně také pro přístup k multidimenzionálním údajům z různých aplikací.

V úvodu kapitoly o jazyku MDX alespoň stručně nadefinujeme některé základní pojmy. Tyto definice budeme postupně rozvíjet a demonstrovat na praktických příkladech.

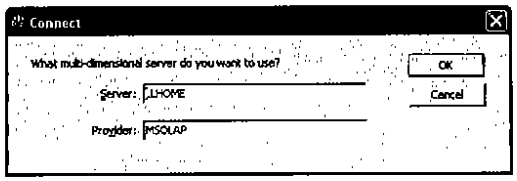
Analogie jazyka MDX a SQL oblastí použití nekončí. I struktura příkazu pro výběr údajů SELECT je podobná. Potěšitelné je to, že je jazyk MDX ve srovnání s jazykem SQL podstatně jednodušší. Syntaxe příkazu SELECT (podotýkáme, že úplná) se skládá z klauzulí FROM a WHERE.

```
SELECT [<specifikace_osy>
       [, <specifikace_osy>...]]
  FROM [<specifikace_krychle>]
[WHERE [<specifikace_řezu>]]
```

Pokračovat v rozvíjení syntaxe příkazů jazyka MDX by nemělo význam. Abychom mohli sestavovat a testovat příkazy MDX, potřebujeme znát dvě věci. Jednak know-how, jak a s pomocí jaké aplikace zadávat příkazy MDX a zobrazovat jejich výsledky a rovněž topologii krychle, se kterou pracujeme.

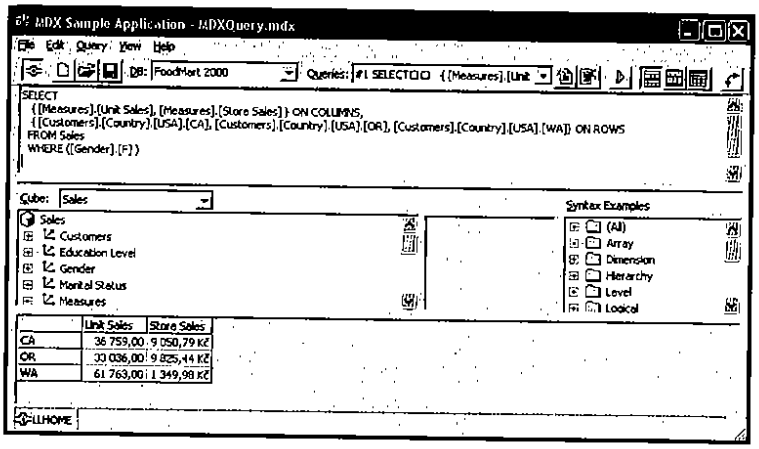
MDX Sample Application

První pokusy s příkazy jazyka MDX můžeme provádět pomocí aplikace MDX Sample Application, která se nainstaluje společně s analytickými službami SQL Serveru 2000. Po spuštění aplikace se přihlásíme k příslušnému serveru OLAP pomocí přihlašovacího dialogu.



Obr. 6.3 Přihlášení se k serveru OLAP

Po úspěšném přihlášení máme k dispozici poměrně komfortní nástroj pro práci s multidimenzionálními strukturami, jednoduchou konzolovou aplikaci, pomocí které můžeme zadávat příkazy a prohlížet si výsledky. Určitým omezením je možnost použití maximálně dvou os. Je to logické, protože se výsledek zobrazí jako dvojrozměrná tabulka. MDX Sample application nám zjednodušuje zadávání dimenzí do těla příkazů jazyka MDX. Stačí v prostředním okně rozvinout stromovou strukturu příslušné dimenze a klepnout na konkrétní úroveň nebo na konkrétní prvek. Jeho název se pak přenese do těla příkazu MDX na aktuální pozici kurzoru.



Obr. 6.4 MDX Sample Application

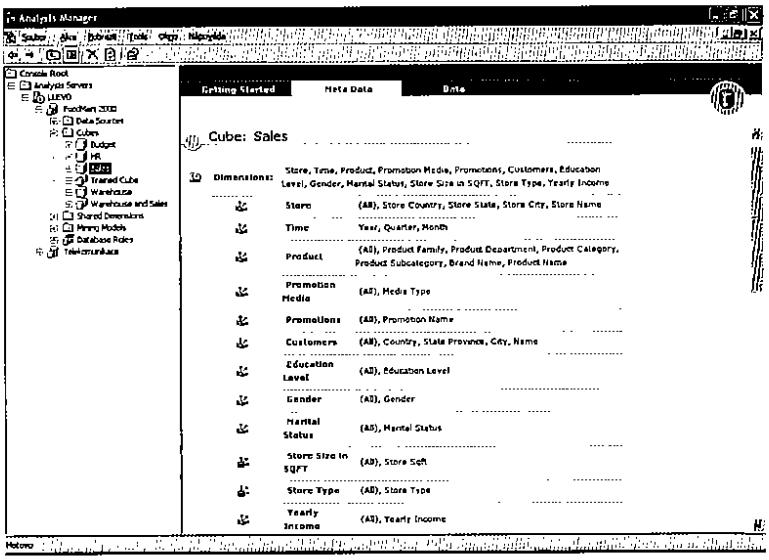
V další kapitole ukážeme příklady vývoje aplikací pro přístup k multidimenzionálním údajům. Pro některé aplikace budeme potřebovat vývojové prostředí, ale pro aplikaci ASP nám postačí jen samotné Windows.

Cvičná krychle Sales

Dříve než se pokusíme přejít ke konkrétní krychli OLAP pomocí jazyka MDX, musíme se v topologii dané krychle trochu zorientovat. Paradoxně je to pravděpodobně jednodušší úloha, než se zorientovat ve složitější relační databázi, hlavně pokud obsahuje více tabulek. O krychli OLAP víme, že má několik dimenzí, z nichž jedna bude s největší pravděpodobností časová, a na průsečících dimenzí budou fakta, tedy měrné jednotky obchodování, takže nejčastěji finanční položky nebo počty kusů.

Dimenze krychle Sales

Cvičná krychle Sales má 12 dimenzí. Nejlepší přehled získáme ze složky metadata v aplikaci Analysis Manager.



Obr. 6.5 Dimenze cvičné krychle SALES

Nebo z naší přehledné tabulky.

Name	Levels	Popis
Customers	Country, State or Province, City, Name	Geografická dimenze, která umožňuje zkoumat, odkud pocházejí zákazníci.
Education Level	Education Level	Úroveň vzdělání zákazníků "Graduate Degree" nebo "High School Degree".
Gender	Gender	Pohlaví zákazníka: "M" (zákazník) nebo "F" (zákaznice).
Marital Status	Marital Status	Rodinný stav zákazníka: "S" (single - svobodný) nebo "M" (ženatý, vdaná).
Product	Product Family Product Department Product Category Product Subcategory Brand Name Product Name	Produkty, s kterými společnost FoodMart obchoduje.
Promotion Media	Media Type	Typ média používaného pro propagaci, například denní tisk, rozhlas, televize...
Promotions	Promotion Name	Která reklama byla důvodem k nákupu.

Název dimenze	Úrovně	Popis
Store	Store Country Store State Store City Store Name	Geografická dimenze hierarchie obchodního řetězce.
Store Size in SQFT	Store Square Feet	Velikost prodejny (ve čtverečních stopách).
Store Type	Store Type	Typ prodejny, například "Deluxe Supermarket" nebo "Small Grocery".
Time	Years, Quarters, Months	Čas, ve kterém se obchod uskutečnil.
Yearly Income	Yearly Income	Roční příjem zákazníka.

Měrné jednotky krychle Sales

V měrných jednotkách je situace přehlednější. Krychle SALES obsahuje 6 měrných jednotek.

Název měrné jednotky	Popis
Unit Sales	Počet prodaných kusů.
Store Cost	Hodnota prodaného zboží.
Store Sales	Hodnota obchodních transakcí.
Sales Count	Počet obchodních transakcí.
Store Sales Net	Hodnota transakcí za sníženou cenu.
Sales Average	Prodané množství/počet obchodů (vypočítaná měrná jednotka).

Základy jazyka MDX

Po alespoň letmém seznámení se se strukturou cvičné krychle SALES a seznámení se s konzolovou aplikací MDX Sample Application se můžeme vrátit k syntaxi základního příkazu pro dotaz MDX SELECT.

```
SELECT [<specifikace_osy>
       [, <specifikace_osy>...]]
  FROM [<specifikace_krychle>]
[WHERE [<specifikace_řezu>]]
```

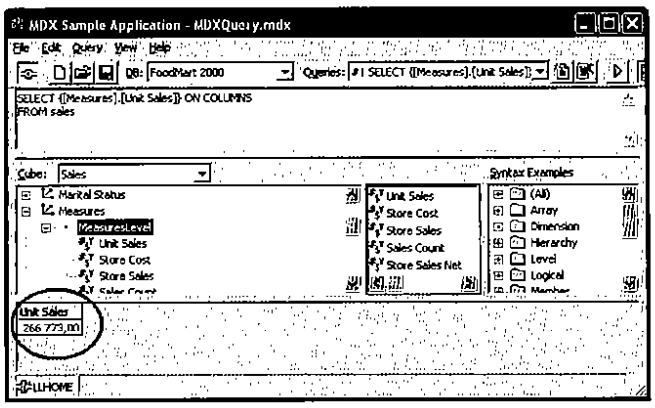
Pravděpodobně srozumitelnější bude syntaktický zápis v tomto tvaru:

```
SELECT  <<set>> ON COLUMNS,
       <<set>> ON ROWS,
       <<set>> ON PAGE
  FROM  <<cube>>
  WHERE <<tuple>>
```

Syntaxi příkazu SELECT ukážeme na praktických příkladech, přičemž budeme postupně vysvětlovat a demonstrovat jednotlivá pravidla. Jako nejjednodušší příkaz jazyka MDX pro

cvičnou krychli SALES bychom mohli aplikovat příkaz pro výpis jedné hodnoty v jednom sloupci, celkovou sumu, kterou všichni zákazníci utratili za nápoje

```
SELECT [Measures].[Unit Sales] ON COLUMNS
FROM sales
```



Obr. 6.6 Výsledek jednoduchého dotazu MDX

Jména objektů

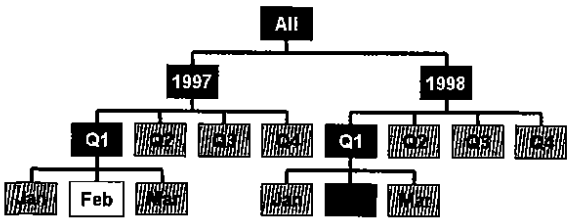
Jména objektů se doporučuje uzavřít do hranatých závorek [ ]. Je to jen doporučení, náš předchozí příkaz by mohl klidně mít tvar:

```
SELECT [Measures.[Unit Sales]] ON COLUMNS
FROM sales
```

ale na přehlednosti by mu to moc nepřidalo. Některé názvy objektů, například ty, které se skládají z více slov oddělených mezerou, v našem případě například `[Unit Sales]`, nebo názvy objektů, které jsou totožné s klíčovými slovy jazyka MDX, například `[SELECT]`, nebo čísla, například `[2003]`, v hranatých závorkách být jednoduše musí.

Dalším důležitým doporučením je používat plně kvalifikované názvy objektů, například první kvartál `[Q1]` může být v roce 2000, ale také v roce 2002, 2003... Proto je potřeba používat úplné názvy objektů, například `[2002].[Q1]`. Pokud plně kvalifikované názvy nepoužijeme, bude při interpretaci příkazu použit první výskyt daného názvu objektu v krychli.

I pro správné definování hierarchické úrovně členů platí určité zásady. Nejlépe to předvedeme na příkladu časové dimenze, jejíž hierarchická struktura je z běžného života každému známa.



Obr. 6.7 Hierarchická struktura časové dimenze

Správně definované jsou například názvy objektů

```
SELECT {[Time].[Year].[1997].[Q1].[2]} ON COLUMNS
FROM sales

SELECT {[Year].[1997].[Q1].[2]} ON COLUMNS
FROM sales

SELECT {[1997].[Q1].[2]} ON COLUMNS
FROM sales
```

a naopak s nesprávně definovanými názvy objektů budou pravděpodobně problémy, například:

```
SELECT {[Time].[Q1].[2]} ON COLUMNS
FROM sales

SELECT {[2003].[2]} ON COLUMNS
FROM sales
```

Terminologie jazyka MDX

Zatím jsme se setkali s názvy objektů v dotazu MDX. V každé oblasti je snaha používat jednotnou terminologii, proto i v jazyce MDX je potřeba alespoň stručně definovat základní pojmy.

Prvek (Member): Definice tohoto pojmu je jednoduchá a stručná. Může se jednat o kterýkoliv prvek (objekt) v hierarchii, například:

```
[Drink]
[2001]
[2001].[Q1].[Jan]
```

N-tice (tuple): Jeden prvek, případně průnik dvou, nebo i více prvků. Například:

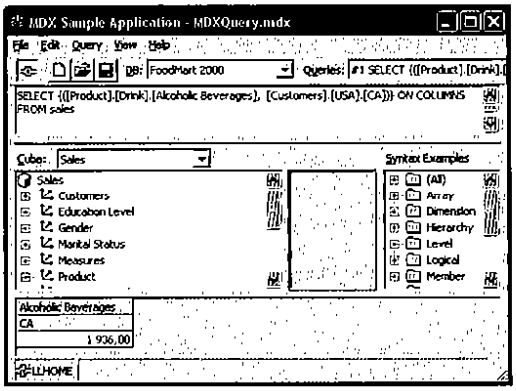
```
([Product].[Drink].[Alcoholic Beverages], [Customers].[USA].[CA])
[Product].[Food]
```

Můžeme zjistit průnik prodaných alkoholických nápojů a zákazníků z USA.

```
SELECT {([Product].[Drink].[Beverages], [Customers].[USA]} ON COLUMNS
FROM sales
```

nebo můžeme aplikovat užší výběr, například průnik prodaných alkoholických nápojů a zákazníků z amerického státu Kalifornie. Přesněji bychom to mohli nazvat jako tržbu za alkoholické nápoje prodané zákazníkům z Kalifornie.

```
SELECT {([Product].[Drink].[Beverages], [Customers].[USA].[CA])} ON COLUMNS
FROM sales
```



Obr. 6.8 Průnik prodaných alkoholických nápojů zákazníkům z Kalifornie

Množina (set): Tímto pojmem označujeme množinu n-tic nebo prvků. Prakticky je to vlastně množina buněk ve výsledné tabulce.

```
SELECT {[William Wong], [Andrea Donlon]} ON COLUMNS
FROM sales
```

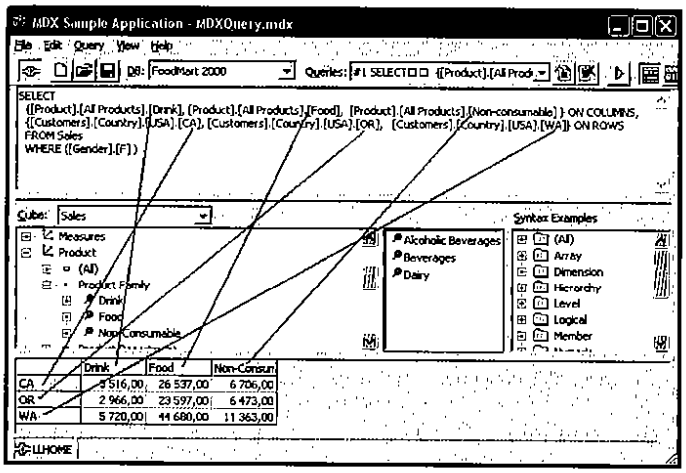
```
SELECT {([Non-Consumable].USA ), (Beverages, Mexico)} ON COLUMNS
FROM sales
```

Množiny nám vracejí i některé funkce. Platí pravidlo, že každý prvek množiny musí mít stejný počet dimenzí.

Dimenze (osy tabulky buněk)

Dimenze je množina elementů stejného typu. Předpokládáme, že po prostudování kapitoly OLAP máme základní terminologii zvládnutou. Zde je příklad výpisu „dvojrozměrné" tabulky údajů.

```
SELECT
  {[Product].[All Products].[Drink], [Product].[All Products].[Food],
   [Product].[All Products].[Non-consumable] } ON COLUMNS,
  {[Customers].[Country].[USA].[CA], [Customers].[Country].[USA].[OR],
   [Customers].[Country].[USA].[WA]} ON ROWS
  FROM Sales
  WHERE ([Gender].[F] )
```



Obr. 6.9 Výsledek dotazu MDX

Pomocí klauzulí ON COLUMNS a ON ROWS jsme se dostali na úroveň dvou dimenzí výsledku. Na obrázku máme čárami vyznačené souvislosti mezi dotazem MDX a Cellsetem neboli množinou buněk v tabulce. Rozeberme si tento příkaz podrobněji po jednotlivých řádcích. Klíčové slovo SELECT, kterým dotaz MDX začíná, vysvětlovat nemusíme, je to analogie příkazu SELECT pro výběr údajů v „klasickém" jazyce SQL. Pomocí části příkazu

```
<<set>> ON COLUMNS,
```

v našem případě konkrétně

```
{[Product].[All Products].[Drink], [Product].[All Products].[Food],
   [Product].[All Products].[Non-consumable] } ON COLUMNS,
```

definujeme první dimenzi, tj., že ve výsledné tabulce budou tři sloupce `[Drink]`, `[Food]` a `[Non-consumable]` z dimenze `[Product].[All Products]`. Druhou dimenzi neboli řádky jsme definovali pomocí části příkazu

```
<<set>> ON ROWS,
```

konkrétně

```
{ [Customers].[Country].[USA].[CA], [Customers].[Country].[USA].[OR],
[Customers].[Country].[USA].[WA]} ON ROWS
```

Tabulka bude mít v našem případě tři řádky.

	Drink	Food	Non-consumable
CA	3 516,00	26 537,00	6 706,00
OR	2 966,00	23 597,00	6 473,00
WA	5 720,00	44 680,00	11 363,00

Povinnou částí příkazu SELECT jazyka MDX je výběr příslušné krychle pomocí klauzule FROM. V našem případě vybíráme údaje z krychle SALES.

```
FROM Sales
```

Příkaz SELECT jazyka MDX také umožňuje definovat podmínku pro omezení nebo v případě krychle by byl vhodnější termín výřez (omezení množiny údajů). V našem případě jsme se zaměřili na ženy.

```
WHERE ([Gender].[F] )
```

Pomocí klauzulí ON COLUMNS a ON ROWS jsme definovali dvě dimenze. Jazyk MDX umožňuje práci až se 128 dimenzemi. Tyto můžeme definovat buď slovně (`COLUMNS`, `ROWS`, `PAGES`, `SECTIONS`, `CHAPTERS`), nebo číselně jako `AXIS(<index>)`.

Syntaktický předpis pro definování osy je:

```
<specifikace_osy> ::= <set> ON <jméno_osy>

<jméno_osy> ::= COLUMNS | ROWS | PAGES | SECTIONS | CHAPTERS | AXIS(<index>)
```

Příkaz

```
SELECT
  {[Product].[All Products].[Drink], [Product].[All Products].[Food],
   [Product].[All Products].[Non-consumable] } ON COLUMNS,
  {[Customers].[Country].[USA].[CA], [Customers].[Country].[USA].[OR],
   [Customers].[Country].[USA].[WA]} ON ROWS
FROM Sales
```

bychom tedy mohli zapsat pomocí čísel os i následovně:

```
SELECT
  {[Product].[All Products].[Drink], [Product].[All Products].[Food],
   [Product].[All Products].[Non-consumable] } ON AXIS(0),
  {[Customers].[Country].[USA].[CA], [Customers].[Country].[USA].[OR],
   [Customers].[Country].[USA].[WA]} ON AXIS(1)
 FROM Sales
```

Výsledná tabulka buněk je v obou případech úplně stejná

	Beer	Wine	Non-Consumable
CA	7 102,00	53 656,00	13 990,00
OR	6 106,00	48 537,00	13 016,00
WA	11 389,00	89 747,00	23 230,00

přičemž osa 0 je podle našich zvyklostí osou X, osa 1 je osou Y, osa 2 je osou Z...

Kromě absolutního definování os buněk můžeme buňky definovat i relativně pomocí klauzulí PrevMember a NextMember. Pokud máme například dotaz MDX pro údaje za druhý kvartál,

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
       {[Time].[1997].[Q2] } ON ROWS
FROM sales
```

	Unit Sales
Q2	62 610,00

můžeme jednoduše zjistit údaje za předchozí, tedy první kvartál

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
       {[Time].[1997].[Q2].PrevMember} ON ROWS
FROM sales
```

	Unit Sales
Q1	66 291,00

nebo následující třetí kvartál.

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
       {[Time].[1997].[Q2].NextMember} ON ROWS
FROM sales
```

	Unit Sales
Q3	65 848,00

Ty samé příkazy můžeme zapsat i pomocí klauzulí LAG (zpoždění) a LEAD (předstih).

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
       {[Time].[1997].[Q2].LAG(1)} ON ROWS
FROM sales
```

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
       {[Time].[1997].[Q2].LEAD(1)} ON ROWS
FROM sales
```

Dimenze os můžeme měnit pomocí klauzule MEMBERS na určité hierarchické úrovni, například:

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
       {[Time].[Year].MEMBERS } ON ROWS
FROM sales
```

	Unit Sales
1997	266 773,00
1998	

Nebo na vyšší hierarchické úrovni:

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
       {[Time].MEMBERS } ON ROWS
FROM sales
```

	Unit Sales
1997	266 773,00
Q1	66 291,00
1	21 628,00
2	20 957,00
3	23 706,00
Q2	62 610,00
4	20 179,00
5	21 081,00
6	21 350,00
Q3	65 848,00
7	23 763,00
8	21 697,00
9	20 388,00
Q4	72 024,00
10	19 958,00
11	25 270,00
12	26 796,00
1998	

Řezy – klauzule WHERE

Na rozdíl od jazyka SQL, kde se klauzule WHERE používá pro definování omezení, tj. pro omezení množiny údajů, v jazyce MDX se klauzule WHERE používá pro definování výrazu údajů. Jak uvidíme dále, pro omezení množiny údajů se v jazyce MDX používá operátor Filter().

Použití klauzule WHERE předvedeme na praktickém příkladu.

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
       {[Product].[Product Family].Members} ON ROWS
FROM Sales
```

	Unit Sales
Drink	24 597,00
Food	191 940,00
Non-Consumable	50 236,00

Pokud chceme omezit údaje jen na americký stát Kalifornii, definujeme tuto skutečnost v klauzuli WHERE.

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
       {[Product].[Product Family].Members} ON ROWS
  FROM Sales
  WHERE [Customers].[All Customers].[USA].[CA]
```

	Unit Sales
Drink	7 102,00
Food	53 656,00
Non-Consumable	13 990,00

Výřez můžeme vytvářet ve více rovinách, naši podmínku výřezu údajů pro Kalifornii můžeme zkombinovat s výřezem pro třetí kvartál roku 1997.

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
       {[Product].[Product Family].Members} ON ROWS
  FROM Sales
  WHERE ([Customers].[All Customers].[USA].[CA], [Time].[1997].[Q3])
```

	Unit Sales
Drink	1 792,00
Food	13 135,00
Non-Consumable	3 443,00

Specifikace intervalu osy – operátor:

Když potřebujeme na některé ose vymezit určitý interval, například časový, nadefinujeme ho pomocí operátoru.

Nejprve vypíšeme údaje za celý rozsah časové dimenze ([Time].MEMBERS).

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
       {[Time].MEMBERS } ON ROWS
FROM sales
```

	Unit Sales
1997	266 773,00
Q1	66 291,00
1	21 628,00
2	20 957,00
3	23 706,00
Q2	62 610,00
4	20 179,00
5	21 081,00
6	21 350,00
Q3	65 848,00
7	23 763,00
8	21 697,00
9	20 388,00
Q4	72 024,00
10	19 958,00
11	25 270,00
12	26 796,00
1998	

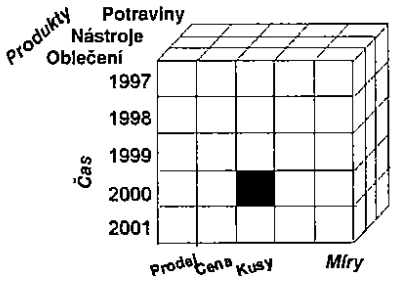
Pokud nás zajímají jen údaje od ledna do května 1997, použijeme příkaz, kde definujeme požadovaný interval.

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
       {[Time].[1997].[Q1].[1] :[Time].[1997].[Q2].[5] } ON ROWS
FROM sales
```

	Unit Sales
1	21 628,00
2	20 957,00
3	23 706,00
4	20 179,00
5	21 081,00

Fakta – měrné jednotky obchodování

Doposud jsme pro jednoduchost v některých příkladech nastavovali dimenze pro řádky a sloupce, nicméně jsme se příliš nezajímali, jaká čísla, tedy fakta nebo měrné jednotky obchodování, jsou vlastně v buňkách cellsetu na průsečících dimenzí znázorněna.



Obr. 6.10 Buňky obsahující měrné jednotky obchodování

Tato fakta můžeme vypsat například do řádků tabulky.

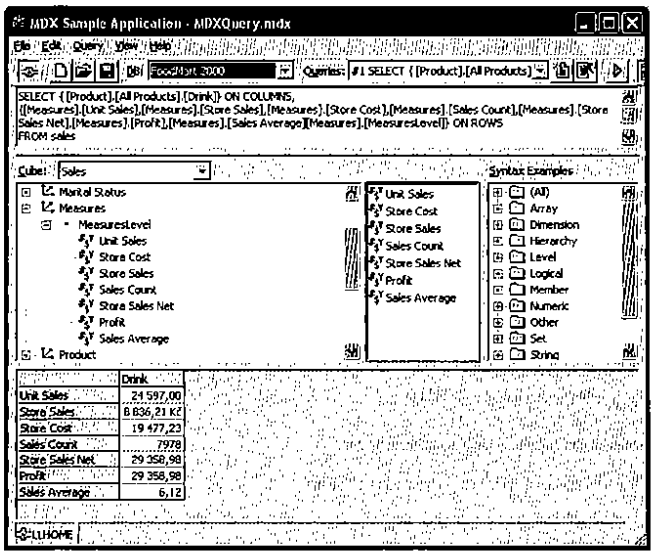
```
SELECT {[Product].[All Products].[Drink]} ON COLUMNS,
       {[Measures].[Unit Sales],
        [Measures].[Store Sales],
        [Measures].[Store Cost],
        [Measures].[Sales Count],
        [Measures].[Store Sales Net],
        [Measures].[Profit],
        [Measures].[Sales Average]} ON ROWS
FROM sales
```

Případně můžeme měrnou jednotku specifikovat výřezem v klauzuli WHERE, například:

```
SELECT { [Product].[All Products].[Drink]} ON COLUMNS
FROM sales
WHERE ([Measures].[Sales Count])
```

Vypočtené prvky – klauzule WITH

Ne vždy se nám pro vytvoření os hodí aktuální hierarchie dimenzí a jindy potřebujeme vypočítat některé hodnoty pomocí aritmetických operátorů a funkcí. Proto jazyk MDX obsahuje klauzuli WITH, pomocí které můžeme potřebné hodnoty prvků vypočítat.



Obr. 6.11 Měrné jednotky obchodování

Například v cvičné krychli SALES je hierarchie časové dimenze navržena systémem ROK – KVARTÁL – MĚSÍC. Pokud potřebujeme údaje za první a druhé pololetí, musíme je vypočítat. Všimněme si použití klauzule NON EMPTY, pomocí které se vypíší jen neprázdné buňky.

```
WITH
   MEMBER [Time].[1 polrok] AS 'Sum({[Time].[Q1], [Time].[Q2]})'
   MEMBER [Time].[2 polrok] AS 'Sum({[Time].[Q3], [Time].[Q4]})'
SELECT
   NON EMPTY { [Store].[Store Name].Members} ON COLUMNS,
   { [Time].[1 polrok], [Time].[2 polrok]} ON ROWS
FROM Sales
WHERE [Measures].[Store Sales]
```

	Store 8	Store 9	Store 24	Store 24	
1. pololetí	20 801,04 Kč	25 421,41 Kč	26 275,11 Kč	2 074,39 Kč	…
2. pololetí	24 949,20 Kč	29 123,87 Kč	28 156,03 Kč	2 366,79 Kč	…

Vyloučení prázdných buněk – klauzule NON EMPTY

V předchozím příkladu si všimněme použití klauzule NON EMPTY, pomocí které se vypíší jen neprázdné buňky. Pokud potřebujeme vypsat státy, kde se prodává dietní kola, můžeme aplikovat dotaz:

```
SELECT
   {[Store].[Store State].Members} ON COLUMNS,
   {[Product].[Excellent Diet Cola]} ON ROWS
FROM Sales
WHERE [Measures].[Unit Sales]
```

Excellent Diet Cola								56,00	61,00	67,00

Pokud se trochu zamyslíme nad zadáním, tak jsme ho vlastně nesplnili, protože jsme zbytečně vypsali i buňky států, kde se dietní kola neprodává. Použijeme proto klauzuli NON EMPTY.

```
SELECT
   NON EMPTY {[Store].[Store State].Members} ON COLUMNS,
   {[Product].[Excellent Diet Cola]} ON ROWS
FROM Sales
WHERE [Measures].[Unit Sales]
```

Excellent Diet Cola	56,00	61,00	67,00

Podmínka IIF

Zavedením podmínky IIF se podstatně rozšíří možnosti jazyka MDX.

```
WITH MEMBER [Measures].[Super obrat] AS
   'IIf(Measures.[Store Sales] > 20000, "Yes", "No")'

SELECT
   {[Store].[Store State].members} ON COLUMNS,
   {[Measures].[Store Sales], [Measures].[Super obrat]} ON ROWS
FROM Sales
```

Store Sales								159 167,84 Kč	142 277,07 Kč	263 793,22 Kč
Super obrat	No	No	No	No	No	No	No	Yes	Yes	Yes

Křížové spojení – operátor CrossJoin()

Při práci s multidimenzionálními strukturami často narazíme na požadavek spojení dvou různých dimenzí. Operátor CrossJoin() slouží ke spojení dvou různých množin a výpisu všech možných kombinací. Syntaxe křížového spojení je:

```
CrossJoin(«Set1», «Set2»)
```

V našem příkladu vypíšeme hodnoty pro kombinaci produktů [Ebony Oranges] a [Ebony Plums] v letech 1997 a 1998.

```
SELECT
   CROSSJOIN ({[Product].[Ebony Oranges], [Product].[Ebony Plums]},
              {[Time].[1997],[Time].[1998]}) ON COLUMNS,
   {[Store].[Store State].Members} ON ROWS
FROM Sales
WHERE (Measures].[Unit Sales]
```

	[Ebony Oranges]		[Ebony Plums]	
	1997	1998	1997	1998
BC				
DF				
Guerrero				
Jalisco				
Veracruz				
Yucatan				
Zacatecas				
CA	38,00		43,00	
OR	52,00		36,00	
WA	72,00		89,00	

Pokud chceme vypsat pouze ty buňky, které obsahují nějakou hodnotu, použijeme klauzuli NON EMPTY.

```
SELECT
   NON EMPTY CROSSJOIN ({[Product].[Ebony Oranges], [Product].[Ebony Plums]},
                        {[Time].[1997],[Time].[1998]}) ON COLUMNS,
   NON EMPTY {[Store].[Store State].Members} ON ROWS
FROM Sales
WHERE [Measures].[Unit Sales]
```

	[Ebony Oranges]	[Ebony Plums]
	1997	1997
CA	38,00	43,00
OR	52,00	36,00
WA	72,00	89,00

Případně můžeme přidat kombinaci dimenzí i do řádků a například zkoumat, jak v příslušných státech nakupují dané produkty muži a jak ženy.

```
SELECT
   NON EMPTY CROSSJOIN ({[Product].[Ebony Oranges], [Product].[Ebony Plums]},
   {[Time].[1997],[Time].[1998]}) ON COLUMNS,
   NON EMPTY CROSSJOIN({[Store].[Store State].Members},{[Gender].[M],
[Gender].[F]}) ON ROWS
FROM Sales
WHERE [Measures].[Unit Sales]
```

		Ebony Oranges	
		1997	1997
CA	M	18,00	28,00
	F	20,00	15,00
OR	M	26,00	13,00
	F	26,00	23,00
WA	M	43,00	42,00
	F	29,00	47,00

Alternativním zápisem operátoru CrossJoin() je znak * (asterix). Používá se podobně jako operátor násobení. Náš příklad můžeme zapsat i takto:

```
SELECT
   NON EMPTY ({[Product].[Ebony Oranges], [Product].[Ebony Plums]}*
   {[Time].[1997],[Time].[1998]}) ON COLUMNS,
   NON EMPTY ({[Store].[Store State].Members}*{[Gender].[M], [Gender].[F]}) ON
ROWS
FROM Sales
WHERE [Measures].[Unit Sales]
```

Redukce údajů – operátor Filter()

V závislosti na požadavcích zobrazení údajů je často potřeba omezit množinu údajů. Pro tento účel se používá operátor Filter(), jehož syntaxe je:

```
Filter (<<set>>, bool podminka)
```

Při konstrukci podmínky můžeme používat tyto operátory porovnávání:

Operátor porovnávání	Význam operátoru
=	rovnost
<>	nerovnost
>	větší
<	menší
>=	větší nebo roven
<=	menší nebo roven

Dále můžeme používat logické operátory pro sestavení složitější podmínky na základě kombinace více jednoduchých podmínek.

NOT Kombinace se uplatní, pokud je následující podmínka nepravdivá.

AND Kombinace se uplatní, pokud jsou obě podmínky pravdivé.

OR Kombinace se uplatní, pokud je alespoň jedna z podmínek pravdivá.

Ukážeme si to na jednoduchém příkladu, kde vypíšeme všechny (neprázdné) hodnoty Unit Sales pro jednotlivé obchody.

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
       NON EMPTY {[Store].[Store Name].members} ON ROWS
 FROM [Sales]
```

	Unit Sales
Store 6	21 333,00
Store 7	25 663,00
Store 24	25 635,00
Store 14	2 117,00
Store 11	26 079,00
Store 13	41 580,00
Store 2	2 237,00
Store 3	24 576,00
Store 15	25 011,00
Store 16	23 591,00
Store 17	35 257,00
Store 22	2 203,00
Store 23	11 491,00

Pokud nás zajímají pouze vyšší obraty, například nad 20 000, můžeme aplikovat filtr.

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
    FILTER ({[Store].[Store Name].members}, [Measures].[Unit Sales]>20000) ON ROWS
 FROM [Sales]
```

	Unit Sales
Store 6	21 333,00
Store 7	25 663,00
Store 24	25 635,00
Store 11	26 079,00
Store 13	41 580,00
Store 3	24 576,00
Store 15	25 011,00
Store 16	23 591,00
Store 17	35 257,00

Setřídění množiny – operátor Order()

Obvykle nás zajímají údaje v pořadí od nejdůležitějších po nejméně důležité. Pokud chceme pochválit pobočky za vysoké obraty, setřídíme údaje sestupně (DESC), tj. od nejvyšší hodnoty po nejmenší. Pokud naopak hledáme skryté rezervy, je výhodnější setřídit údaje od nejmenších po největší. V takovém případě použijeme vzestupné třídění (ASC). Pro setřídění se používá operátor Order(). Syntaktický předpis pro tento operátor je

```
Order(«Set», [«Řetězcový výraz» | «Numerický výraz»] [, ASC | DESC | BASC |
BDESC])
```

Setřídění může být hierarchické (ASC, DESC) nebo nehierarchické (BASC, BDESC). Rozdíl si ukážeme na příkladu. Nejprve vypíšeme nesetříděné údaje:

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
    NON EMPTY {[Store].[Store Name].members} ON ROWS
 FROM [Sales]
```

	Unit Sales
Store 6	21 333,00
Store 7	25 663,00
Store 24	25 635,00
Store 14	2 117,00
Store 11	26 079,00
Store 13	41 580,00
Store 2	2 237,00
Store 3	24 576,00
Store 15	25 011,00
Store 16	23 591,00
Store 17	35 257,00
Store 22	2 203,00
Store 23	11 491,00

a nyní je setřídíme hierarchicky (obchody jsou geograficky rozmístěné v městech a státech).

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
       NON EMPTY ORDER({[Store].[Store Name].members},
          [Measures].[Unit Sales],DESC) ON ROWS
 FROM [Sales]
```

	Unit Sales
Store 17	35 257,00
Store 15	25 011,00
Store 3	24 576,00
Store 16	23 591,00
Store 23	11 491,00
Store 2	2 237,00
Store 22	2 203,00
Store 7	25 663,00
Store 24	25 635,00
Store 6	21 333,00
Store 14	2 117,00
Store 13	41 580,00
Store 11	26 079,00

Lépe to pochopíme, pokud si necháme vypsat celou hierarchii dimenze STORES.

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
       NON EMPTY ORDER({[Store].members},
          [Measures].[Unit Sales],DESC) ON ROWS
 FROM [Sales]
```

(Tabulka je pro úsporu místa rozdělená na tři sloupce.)

	Unit Sales
All Stores	266 773,00
USA	266 773,00
WA	124 366,00
Tacoma	35 257,00
Store 17	35 257,00
Seattle	25 011,00
Store 15	25 011,00
Bremerton	24 576,00
Store 3	24 576,00
Spokane	23 591,00

Store 16	23 591,00
Yakima	11 491,00
Store 23	11 491,00
Bellingham	2 237,00
Store 2	2 237,00
Walla Walla	2 203,00
Store 22	2 203,00
CA	74 748,00
Los Angeles	25 663,00
Store 7	25 663,00
San Diego	25 635,00

Store 24	25 635,00
Beverly Hills	21 333,00
Store 6	21 333,00
San Francisco	2 117,00
Store 14	2 117,00
OR	67 659,00
Salem	41 580,00
Store 13	41 580,00
Portland	26 079,00
Store 11	26 079,00

Nakonec setřídíme údaje absolutně, bez ohledu na hierarchii.

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
       NON EMPTY ORDER({[Store].[Store Name].members},
          [Measures].[Unit Sales],BDESC) ON ROWS
 FROM [Sales]
```

	Unit Sales
Store 13	41 580.00
Store 17	35 257.00
Store 11	26 079.00
Store 7	25 663.00
Store 24	25 635.00
Store 15	25 011.00
Store 3	24 576.00
Store 16	23 591.00
Store 6	21 333.00
Store 23	11 491.00
Store 2	2 237.00
Store 22	2 203.00
Store 14	2 117.00

Lokální krychle OLAP

V předchozí kapitole jsme si ukázali vytvoření lokální krychle OLAP pomocí programu Microsoft Excel. Metadata pro definici lokální krychle jsou uložená v souboru s příponou .CUB.

Proces definování lokální krychle se skládá ze známých kroků:

- definování dimenzí,
- definování měrných jednotek obchodování,
- definování případných vypočítaných členů,
- definování dalších objektů, například úrovní dimenzí, vlastností členů...,
- naplnění dimenzí,
- naplnění měrných jednotek a vypočítaných členů,
- mapování dimenzí a měrných jednotek na strukturu krychle OLAP.

Příkaz CREATE CUBE pro vytvoření krychle

Úplná syntaxe příkazu CREATE CUBE pro vytvoření krychle OLAP je velmi rozsáhlá a členitá, proto si ukážeme zjednodušenou syntaktickou definici.

```
CREATE CUBE <název krychle>
(
  DIMENSION <název> [TYPE <Typ>]
     LEVEL <název>[TYPE <Typ>],[LEVEL <název> [TYPE <Typ]....
  MEASURE <název> FUNCTION <název>...
...
)
```

Například:

```
CREATE CUBE SalesNW
(
   DIMENSION [Country],
     LEVEL [All] TYPE ALL,
     LEVEL [Country],
     LEVEL [City],
     LEVEL [CustomerID],
   DIMENSION [Salesperson],
     LEVEL [All] TYPE ALL,
     LEVEL [Salesperson],
   DIMENSION [ShipperName],
     LEVEL [All] TYPE ALL,
     LEVEL [ShipperName],
   DIMENSION [CategoryName],
     LEVEL [All] TYPE ALL,
     LEVEL [CategoryName],
     LEVEL [ProductName],
 DIMENSION [OrderDate] TYPE TIME,
     LEVEL [All] TYPE ALL,
     LEVEL [Year] TYPE YEAR,
     LEVEL [Quarter] TYPE QUARTER,
     LEVEL [Month] TYPE MONTH,
     LEVEL [Day] TYPE DAY,
  MEASURE [Sum Of ExtendedPrice] FUNCTION SUM,
  MEASURE [Sum Of Quantity]      FUNCTION SUM
)
```

Definování dimenzí

Už při letmém pohledu na předchozí jednoduchý příklad příkazu CREATE CUBE vidíme, že syntaxe části tohoto příkazu pro definování dimenzí nebude složitá, pouze to může být při příliš hierarchicky strukturovaných dimenzích trochu pracné. Zjednodušená syntaxe definování dimenze je:

```
DIMENSION [název dimenze] [TYPE TIME
```

Například:

```
...
DIMENSION [Country],
...
```

Nebo pro definování úrovní časové dimenze:

```
...
DIMENSION [OrderDate] TYPE TIME,
...
```

Definování úrovní dimenzí

Úrovně dimenze (LEVELS) definujeme jako vnořené řádky příkazu CREATE table pro jednotlivé dimenze. Důležité je pořadí podle hierarchické struktury dané dimenze. To znamená, že úrovně vypisujeme od nejvyšší hierarchické úrovně, tedy od kořene. Syntaxe části příkazu CREATE CUBE pro definování jednotlivých úrovní dimenze je:

```
LEVEL [název úrovně] [TYPE ][FORMAT_NAME název_formátu [FORMAT_KEY klíč_formátu]]
[OPTIONS volby]....
```

Jako nejvyšší úroveň zpravidla definujeme úroveň TYPE ALL.

```
LEVEL [All] TYPE ALL,
```

U časové dimenze musíme definovat pro jednotlivé atributy i standardní kalendářní a časovou hierarchickou úroveň.

```
YEAR | QUARTER | MONTH | WEEK | DAY | DAYOFWEEK | DATE | HOUR | MINUTE | SECOND
```

Například:

```
DIMENSION [Country],
     LEVEL [All] TYPE ALL,
     LEVEL [Country],
     LEVEL [City],
     LEVEL [CustomerID],

...
```

Nebo pro definování úrovní časové dimenze:

```
...
DIMENSION [OrderDate] TYPE TIME,
     LEVEL [All] TYPE ALL,
     LEVEL [Year] TYPE YEAR,
     LEVEL [Quarter] TYPE QUARTER,
     LEVEL [Month] TYPE MONTH,
     LEVEL [Day] TYPE DAY,
...
```

Vlastnosti úrovní dimenzí

Pro definování úrovní dimenzí platí určitá pravidla, která vyplývají z aplikační logiky a zvyklostí běžného života. Například že geografická dimenze vychází z adresy a ta zase z územněsprávního členění, že produkty mají určité standardní vlastnosti jako rozměry, hmotnost, cenu a podobně. Proto pro každý člen hierarchické úrovně můžeme definovat standardní vlastnosti.

```
PROPERTY [název vlastnosti] [TYPE PropTye]
```

Jako typ vlastností členů hierarchických úrovní můžeme použít tyto typy:

REGULAR	GEO_BOUNDARY_FRONT
ID	GEO_BOUNDARY_REAR
RELATION_TO_PARENT	GEO_BOUNDARY_POLYGON
ORG_TITLE	PHYSICAL_SIZE
CAPTION	PHYSICAL_COLOR
CAPTION_SHORT	PHYSICAL_WEIGHT
CAPTION_DESCRIPTION	PHYSICAL_HEIGHT
CAPTION_ABREVIATION	PHYSICAL_WIDTH
WEB_URL	PHYSICAL_DEPTH
WEB_HTML	PHYSICAL_VOLUME
WEB_XML_OR_XSL	PHYSICAL_DENSITY
WEB_MAIL_ALIAS	PERSON_FULL_NAME
ADDRESS	PERSON_FIRST_NAME
ADDRESS_STREET	PERSON_LAST_NAME
ADDRESS_HOUSE	PERSON_MIDDLE_NAME
ADDRESS_CITY	PERSON_DEMOGRAPHIC
ADDRESS_STATE_OR_PROVINCE	PERSON_CONTACT
ADDRESS_ZIP	QTY_RANGE_LOW
ADDRESS_QUARTER	QTY_RANGE_HIGH
ADDRESS_COUNTRY	FORMATTING_COLOR
ADDRESS_BUILDING	FORMATTING_ORDER
ADDRESS_ROOM	FORMATTING_FONT
ADDRESS_FLOOR	FORMATTING_FONT_EFFECTS
ADDRESS_FAX	FORMATTING_FONT_SIZE
ADDRESS_PHONE	FORMATTING_SUB_TOTAL
GEO_CENTROID_X	DATE
GEO_CENTROID_Y	DATE_START
GEO_CENTROID_Z	DATE_ENDED
GEO_BOUNDARY_TOP	DATE_CANCELED
GEO_BOUNDARY_LEFT	DATE_MODIFIED
GEO_BOUNDARY_BOTTOM	DATE_DURATION
GEO_BOUNDARY_RIGHT	VERSION

Definování měrných jednotek

Při vytváření krychle musíme kromě dimenzí definovat i měrné jednotky. Na jednotlivé měrné jednotky potom aplikujeme agregační funkce SUM, COUNT, MAX a MIN. Zjednodušený syntaktický předpis části příkazu CREATE CUBE pro definování měrných jednotek je:

```
MEASURE [název měrné jednotky][FORMAT formátovací_řetězec][TYPE OLEDB definice_typu]
  FUNCTION agregační_funkce,
```

Jako datové typy OLEDB můžeme použít tyto:

DBTYPE_I1	DBTYPE_UI8
DBTYPE_I2	DBTYPE_R4
DBTYPE_I4	DBTYPE_R8
DBTYPE_I8	DBTYPE_CY
DBTYPE_UI1	DBTYPE_DECIMAL
DBTYPE_UI2	DBTYPE_NUMERIC
DBTYPE_UI4	DBTYPE_DATE

V našem příkladu jsme měli dvě měrné jednotky, kde byla v obou případech aplikována agregační funkce SUM.

```
MEASURE [Sum Of ExtendedPrice] FUNCTION SUM,
MEASURE [Sum Of Quantity]      FUNCTION SUM
```

Příkaz INSERT INTO SELECT pro naplnění krychle

Příkazem CREATE CUBE se krychle OLAP jen vytvoří a následně je potřeba ji naplnit údaji. Pro tento účel se používá konstrukce INSERT INTO – SELECT, jejíž zjednodušená syntaxe je

```
INSERT INTO <název krychle>
    (
     <název dimenze>.<název úrovně>, [<název dimenze>.<název úrovně>,…]
     <název měrné jednotky>.[<název měrné jednotky>,… ]
    )
    OPTIONS <option list>
SELECT <Column list>
FROM <table list>
WHERE <joins list>
```

Například pro naplnění krychle z databáze Northwind:

```
INSERT INTO SalesNW
 [Country].[Country], [City], [CustomerID],
 [Sum Of ExtendedPrice], [Sum Of Quantity],
 [ShipperName].[ShipperName], [Salesperson].[Salesperson],
 [CategoryName].[CategoryName],
 [ProductName], [OrderDate])
OPTIONS ATTEMPT_ANALYSIS
SELECT Invoices.Country, Invoices.City, Invoices.CustomerID,
       Invoices.ExtendedPrice, Invoices.Quantity, Invoices.ShipperName,
       Invoices.Salesperson, Categories.CategoryName,
       Products.ProductName, Invoices.OrderDate
FROM Northwind.dbo.Categories Categories,
     Northwind.dbo.Invoices Invoices,
     Northwind.dbo.Products Products
WHERE Categories.CategoryID = Products.CategoryID AND
      Invoices.ProductID = Products.ProductID
```

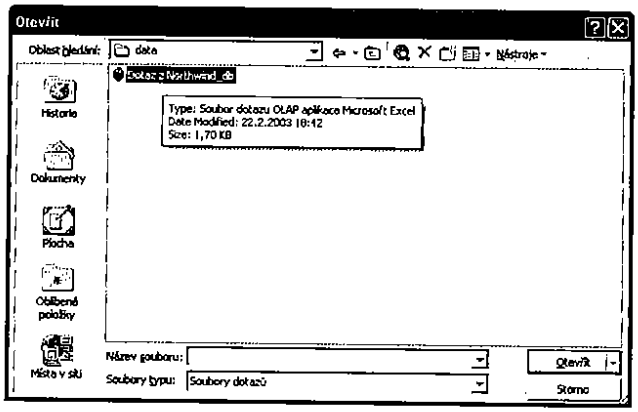
Otevření souboru s definicí lokální krychle v programu Microsoft Excel

Pro použití v programu Excel můžeme příkazy CREATE CUBE a INSERT INTO SELECT umístit do souboru s příponou .OQY (OLAP Query). Struktura tohoto souboru je popsána v předchozí kapitole, kde jsme ho v programu Excel i vytvořili.

```
QueryType=OLEDB
Version=1
CommandType=Cube
Connection=Provider=MSOLAP;
Initial Catalog=[NWCube];
Data Source=C:\Data\Northwind_db.cub;

CreateCube=CREATE CUBE [NWCube]
(DIMENSION [Country],
   LEVEL [Vše] TYPE ALL, LEVEL [Country], LEVEL [City], LEVEL [CustomerID],
DIMENSION [Salesperson],
   LEVEL [Vše] TYPE ALL, LEVEL [Salesperson],
DIMENSION [ShipperName],
   LEVEL [Vše] TYPE ALL, LEVEL [ShipperName],
DIMENSION [CategoryName],
   LEVEL [Vše] TYPE ALL, LEVEL [CategoryName], LEVEL [ProductName],
DIMENSION [OrderDate] TYPE TIME,
   LEVEL [Vše] TYPE ALL,  LEVEL [Rok] TYPE YEAR,  LEVEL [Čtvrtletí] TYPE QUARTER,
   LEVEL [Měsíc] TYPE MONTH,  LEVEL [Den] TYPE DAY,
MEASURE [Sum Of ExtendedPrice]    FUNCTION SUM,
MEASURE [Sum Of Quantity]    FUNCTION SUM  );
```

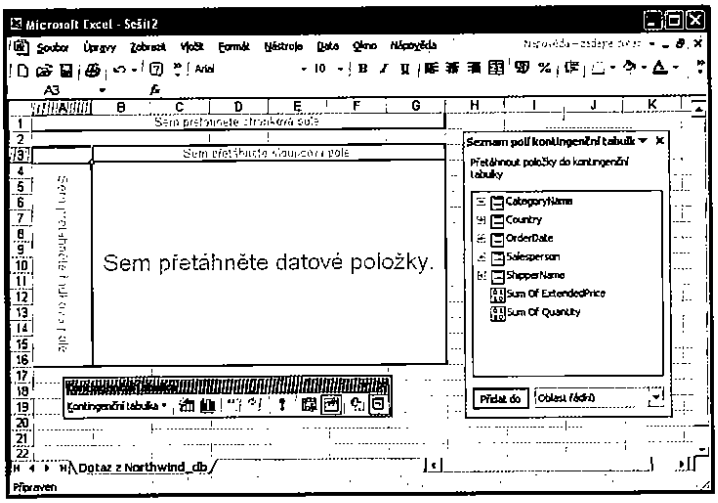
```
InsertInto=INSERT INTO OCWCube([Country].[Country], [City], [CustomerID],
    [Sum Of ExtendedPrice], [Sum Of Quantity], [ShipperName].[ShipperName],
    [Salesperson].[Salesperson], [CategoryName].[CategoryName], [ProductName],
    [OrderDate]) OPTIONS ATTEMPT_ANALYSIS
  SELECT Invoices.Country, Invoices.City, Invoices.CustomerID,
    Invoices.ExtendedPrice, Invoices.Quantity, Invoices.ShipperName,
    Invoices.Salesperson, Categories.CategoryName, Products.ProductName,
    Invoices.OrderDate
  FROM Northwind.dbo.Categories Categories, Northwind.dbo.Invoices Invoices,
    Northwind.dbo.Products Products
  WHERE Categories.CategoryID = Products.CategoryID AND
        Invoices.ProductID = Products.ProductID;

Source_DSN="DRIVER=SQL Server;SERVER=LLEVO;UID=sa; DATABASE=Northwind;
Trusted_Connection=Yes";
UseExistingFile=True
CommandText=OCWCube
```

Soubor s příponou OQY je možné otevřít v programu Microsoft Excel. Soubor otevřeme pomocí standardního dialogu pro otevírání souborů, kde nastavíme typ otvíraného souboru „Soubory dotazů".

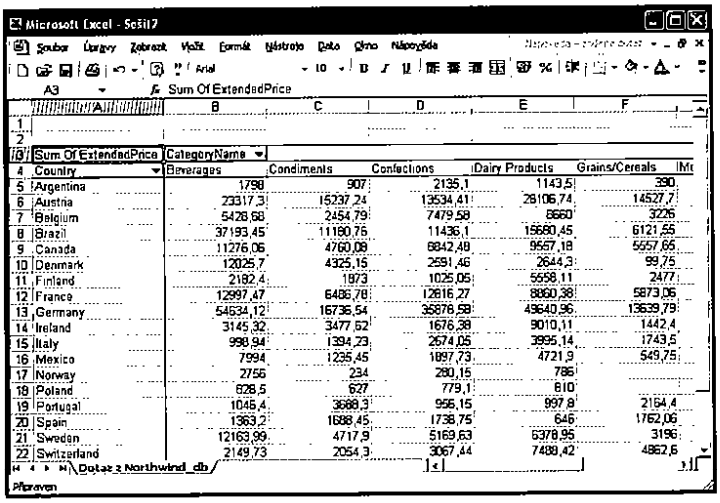


Obr. 6.12 Otevření souboru lokální krychle

Po otevření souboru můžeme pokračovat v interaktivním návrhu kontingenční tabulky. Výsledek návrhu můžeme zobrazit ve formě tabulky, případně grafu, a uložit jako standardní sešit programu Excel.



Obr. 6.13 Otevření souboru lokální krychle



Sum Of ExtendedPrice	CategoryName				
Country	Beverages	Condiments	Confections	Dairy Products	Grains/Cereals
Argentina	1798	907	2135,1	1143,5	390
Austria	23317,3	15237,24	13534,41	28106,74	14527,7
Belgium	5428,68	2454,79	7479,58	8660	3226
Brazil	37193,45	11190,76	11436,1	15660,45	6121,55
Canada	11276,06	4760,08	8842,48	9567,18	5557,65
Denmark	12025,7	4325,15	2591,46	2644,3	99,75
Finland	2182,4	1973	1025,05	5558,11	2477
France	12997,47	6486,78	12816,27	8860,38	5873,06
Germany	54634,12	16736,54	35878,58	49640,96	13639,79
Ireland	3145,32	3477,52	1676,38	9010,11	1442,4
Italy	998,94	1394,23	2674,05	3995,14	1743,5
Mexico	7994	1235,45	1897,73	4721,9	549,75
Norway	2756	234	280,15	786	
Poland	628,5	627	779,1	810	
Portugal	1046,4	3688,3	956,15	997,8	2164,4
Spain	1363,2	1688,45	1738,75	646	1762,06
Sweden	12163,99	4717,9	5169,63	6378,95	3196
Switzerland	2149,73	2054,3	3067,44	7488,42	4862,6

Obr. 6.14 Příklad kontingenční tabulky ze souboru lokální krychle

Vytvoření lokální krychle pomocí aplikačního kódu

Tento příklad použití příkazu CREATE CUBE a INSERT INTO SELECT můžeme zapsat také v libovolném programovacím jazyce, například ve Visual Basicu. V našem případě je celá rutina realizována jako obsluha události (klepnutí na tlačítko aplikačního formuláře).

```
Private Sub Command1_Click()
Dim cnCube As ADODB.Connection
Dim s As String
Dim Prov As String
Dim DS As String
Dim SourceDSN As String
Dim SourceDSNSuffix As String
Dim sCC As String
Dim sII As String
Dim sSel As String

 Prov = "PROVIDER=MSOLAP"
 DS = "DATA SOURCE=c:\data\nw.cub"
 SourceDSN = "SOURCE_DSN=""DRIVER=SQL Server; SERVER=LLEVO; UID=sa;DATABASE=Northwind"""
 sCC = "CREATECUBE=CREATE CUBE SalesNW( "
 sCC = sCC & " DIMENSION [Country], LEVEL [All] TYPE ALL, LEVEL [Country], LEVEL [City],
LEVEL [CustomerID],"
 sCC = sCC & " DIMENSION [Salesperson], LEVEL [All] TYPE ALL, LEVEL [Salesperson],"
 sCC = sCC & " DIMENSION [ShipperName], LEVEL [All] TYPE ALL, LEVEL [ShipperName],"
 sCC = sCC & " DIMENSION [CategoryName],LEVEL [All] TYPE ALL, LEVEL [CategoryName],
LEVEL [ProductName],"
 sCC = sCC & " DIMENSION [OrderDate] TYPE TIME, LEVEL [All] TYPE ALL, LEVEL [Year] TYPE
YEAR, LEVEL [Quarter] TYPE QUARTER, LEVEL [Month] TYPE MONTH,  LEVEL [Day] TYPE DAY,"
 sCC = sCC & " MEASURE [Sum Of ExtendedPrice] FUNCTION SUM, MEASURE [Sum Of Quantity]
FUNCTION SUM )"

sII = sII & "InsertInto=INSERT INTO Sales ("
sII = sII & "[Country].[Country], [City], [CustomerID], [Sum Of ExtendedPrice],
[Sum Of Quantity], [ShipperName].[ShipperName], [Salesperson].[Salesperson],"
sII = sII & "[CategoryName].[CategoryName], [ProductName], [OrderDate]) OPTIONS
ATTEMPT_ANALYSIS "

sSel = " SELECT Invoices.Country, Invoices.City, Invoices.CustomerID,
Invoices.ExtendedPrice, Invoices.Quantity, Invoices.ShipperName,"
sSel =sSel + "Invoices.Salesperson, Categories.CategoryName, Products.ProductName,
Invoices.OrderDate "
sSel = sSel + "FROM Northwind.dbo.Categories Categories, Northwind.dbo.Invoices
Invoices, Northwind.dbo.Products Products "
sSel = sSel + "WHERE Categories.CategoryID = Products.CategoryID AND
Invoices.ProductID = Products.ProductID"

sII = sII & sSel

Set cnCube = New ADODB.Connection
s = Prov & ";" & DS & ";" & SourceDSN & ";" & sCC & ";" & sII & ";"
cnCube.Open s

End Sub
```

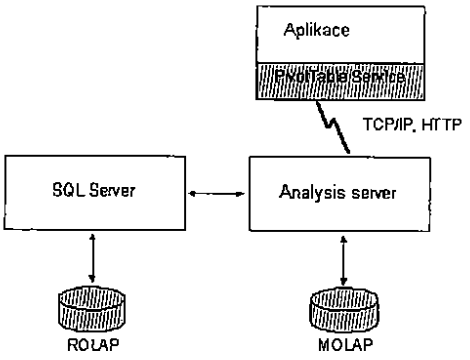
Kapitola 7

Vývoj aplikací pro přístup ke krychlím OLAP

Tématem předchozí kapitoly byl jazyk MDX pro přístup k multidimenzionálním databázím. Podobně jako příkazy jazyka SQL můžeme i příkazy jazyka MDX zadávat pomocí konzolové aplikace (se serverem SQL je dodávána jednoduchá klientská aplikace MDX Sample Application). Pro některé aplikace budeme potřebovat vývojové prostředí, ale pro aplikaci ASP nám budou stačit samotné Windows.

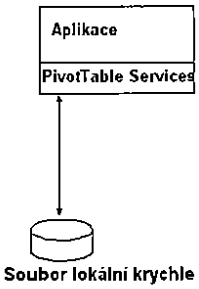
Pivot Table Services

PTS (Pivot Table Services) je primární rozhraní pro aplikace, které potřebují pracovat s analytickými službami, takže s krychlemi pod správou serveru OLAP, ale i s lokálními (off-line) krychlemi. Primárním cílem PTS je získávání údajů v multidimenzionálním nebo tabulkovém tvaru, umožňuje také editování údajů a změny lokálních krychlí.



Obr. 7.1 Koncepce Pivot Table Services pro připojení k analytickému serveru

V případě, že se připojujeme k souboru lokální krychle, nepotřebujeme žádné síťové připojení.



Obr. 7.2 Koncepce Pivot Table Services pro připojení k lokální krychli

Pomocí PTS můžeme vytvářet, měnit a aktualizovat krychle OLAP pomocí příkazů CREATE CUBE, ALTER CUBE, UPDATE CUBE.

Instalace a distribuce

Pokud distribuujeme PivotTable Services na jiný počítač, například klientský nebo na počítač zákazníka, pro kterého jsme vyvíjeli analytickou aplikaci, je potřeba nejdříve nainstalovat komponenty *Microsoft Data Access Components (MDAC)*.

PTS můžeme nainstalovat z instalačních souborů Ptslite.exe nebo Ptsfull.exe. Instalační soubory se nachází na instalačním CD Serveru SQL a analytických služeb v adresáři

\Msolap\Install\Pts.

Pomocí instalačního souboru *Ptslite.exe* nainstalujeme pouze komponentu PTS. Instalační soubor *Ptsfull.exe* kromě PivotTable Service files nainstaluje i komponenty Microsoft Data Access Components (MDAC).

PTS můžeme nainstalovat i „manuálně". I když soubory PTS nezabírají na disku moc místa, můžeme provést tři typy instalace. Soubory s jednotlivými komponentami se standardně instalují do adresáře

```
C:\Program Files\Common Files\System\OLE DB.
```

Základní instalace

Základní instalace je potřeba pro přístup k analytickému serveru přes TCP/IP nebo HTTP, případně pokud chceme číst údaje z lokální krychle. Při základní instalaci je potřeba nainstalovat a zaregistrovat soubory *Msolap80.dll, Msolui80.dll, Msolap80.rll* a *Olapuir.rll*.

Instalace pro vytvoření a aktualizaci lokálních krychlí

V takovém případě je kromě základní instalace potřeba navíc nainstalovat a zaregistrovat soubory

Msmdcb80.dll a *Msmdgd80.dll* .

Instalace pro čtení data miningových modelů v databázích OLAP nebo v relační databázi

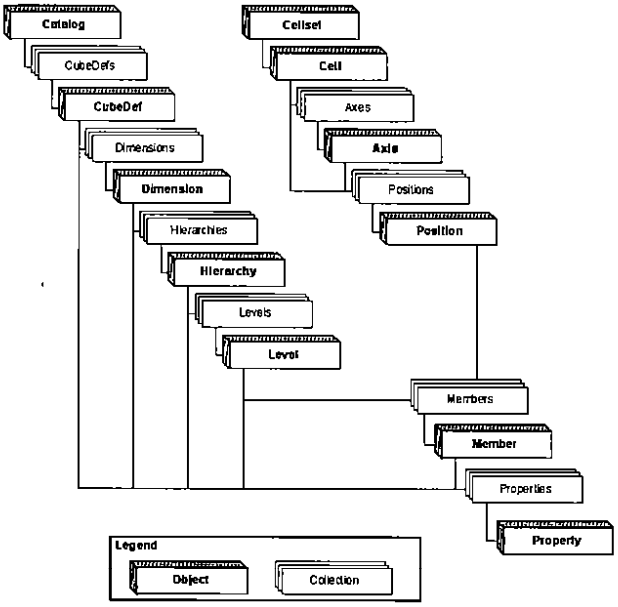
Kromě základní instalace je navíc potřeba nainstalovat a zaregistrovat soubory *Msdmine.dll, Msmdun80.dll, Msdmine.rll* a *Msdmeng.dll.*

Rozhraní ADOMD

Pro vytvoření klientské aplikace, která bude přistupovat k údajům v krychli OLAP, potřebujeme znát i rozhraní pro přístup k těmto údajům. Na platformě Serveru SQL 2000 plní tuto úlohu rozhraní ADOMD (ActiveX Data Objects Multidimensional). Jedná se o rozšíření rozhraní ADO o podporu práce s multidimenzionálními údaji. Rozšíření spočívá v zavedení objektů CATALOG a CELLSET.

Objekt CATALOG slouží k získávání metadat, tedy informací o multidimenzionální databázi.

Objekt CELLSET obsahuje výsledky dotazů jazyka MDX v multidimenzionální formě. Jde o obdobu objektu ADO.RecordSet pro relační údaje. Hierarchická struktura ADOMD je zřejmá ze schématu.



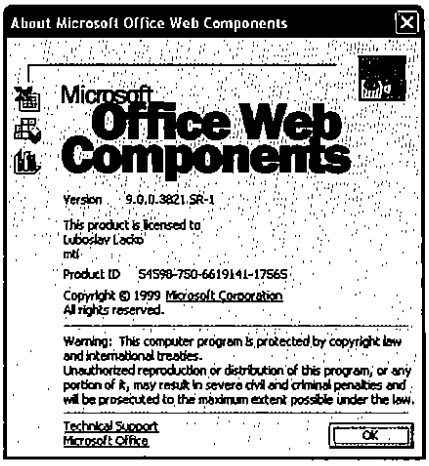
Obr. 7.3 Hierarchická struktura objektů ADOMD

Aplikace pro přístup k údajům v analytických databázích typu tenký klient

Při vývoji aplikací typu tenký klient máme v zásadě dvě možnosti. Buď použijeme hotové komponenty Office Web Components (OWC), nebo budeme vyvíjet webovou aplikaci kompletně sami. Pokud se rozhodneme vyvíjet klasickou webovou aplikaci pro přístup k analytickým údajům, tedy bez využití Office Web Components, máme znovu několik možností výběru serverového skriptovacího systému, například JSP, PHP, ASP, ASP.NET, PERL.... Jelikož Microsoft SQL Server včetně analytických služeb běží jen na platformě Windows, nabízí se jako typické „nativní" skriptovací systémy stránky ASP a ASP.NET. Pro stránky ASP.NET je potřeba nainstalovat technologickou platformu .NET Framework, pro stránky ASP postačí Windows a Internet Information Server. Proto s ohledem na co největší dostupnost použijeme pro příklady v této kapitole stránky ASP.

Zobrazení výsledků analýzy pomocí Office Web Components

Poměrně komfortní a na vývojářské práce velmi nenáročný způsob zobrazování výsledků analýzy OLAP je využití komponent Office Web Components (OWC).



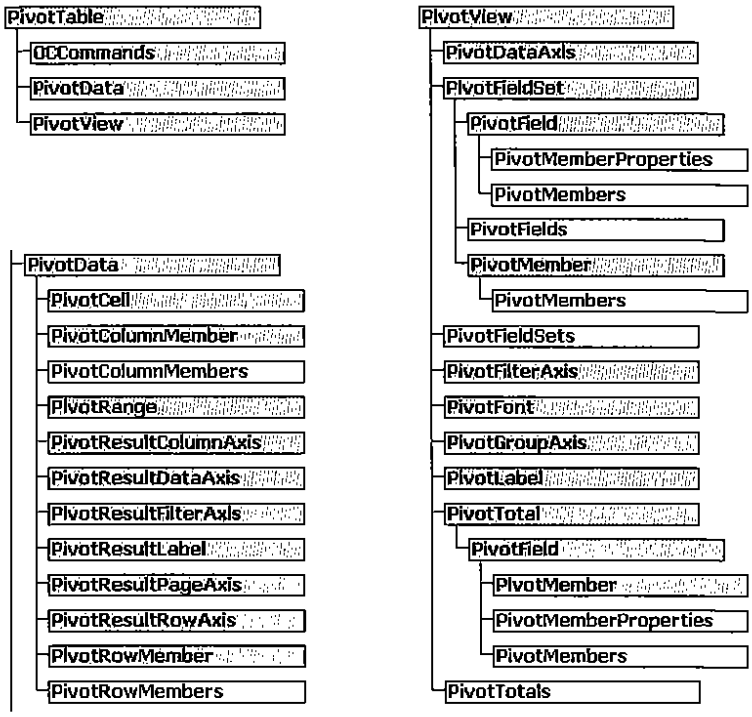
Obr. 7.4 Office Web Components

Tyto komponenty, konkrétně PivotTable, Chart a Spreadsheet je možné využívat v novějších verzích prohlížečů (IE 4.01 a vyšší), ale i ve formulářích aplikací vyvíjených v programovacích jazycích Visual Basic a Visual C#. Mohou být používány dokonce i bez uživatel-

ského rozhraní například na stránkách ASP. Výhodou těchto komponent je také skutečnost, že mohou generovat statický obsah, například obrázky ve formátu GIF, stránky HTML, případně soubory XML.

Komponenta OWC PivotTable

Komponenta OWC PivotTable je prakticky shodná s kontingenční tabulkou v programu Microsoft Excel. Objektová struktura komponenty PivotTable je poměrně složitá a skládá se ze tří objektů OCCommands, PivotData a PivotView, které se dále hierarchicky člení.



Obr. 7.5 Objektová struktura PivotTable

Použití komponenty Pivot Table ukážeme na třech praktických příkladech.

Připojení komponenty OWC PivotTable k serveru OLAP

V prvním příkladu ukážeme připojení komponenty OWC PivotTable k serveru OLAP. Komponentu umístíme na stránku HTML.

```
<object classid="clsid:0002E520-0000-0000-C000-000000000046"
  id="OWC1" width="640" height="480">
</object>
```

Její parametry nastavíme pomocí kódu skriptu v JavaScriptu. Tento kód skriptu se bude interpretovat na počítači klienta. Komponentu jsme v našem případě pojmenovali OWC1. Nejdříve získáme z dokumentu HTML referenci

```
  var objPT = document.all.OWC1;
```

a nastavíme parametry pro připojení, tedy připojovací řetězec na server OLAP a název krychle.

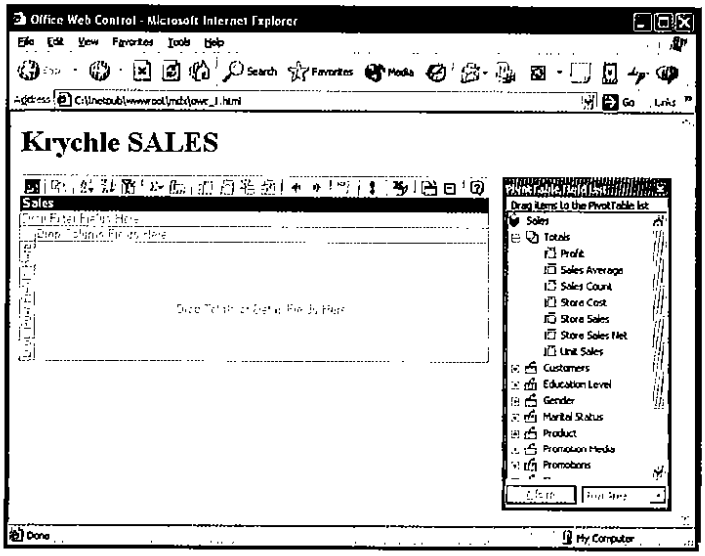
```
objPT.ConnectionString = "Provider=msolap;Data Source=localhost;Initial
                         Catalog=FoodMart 2000";
objPT.DataMember = "Sales";
```

Protože zatím nijak nedefinujeme to, které měrné jednotky a dimenze se kde zobrazí a zda se vůbec zobrazí, zobrazíme v kontextu stránky HTML klienta i seznam polí, které můžeme poté interaktivně přesunout na příslušná pole kontingenční tabulky.

```
  objPT.DisplayFieldList = 1;
```

Kompletní výpis kódu stránky OWC_1.HTML.

```
<html><head><title>Office Web Control</title></head><body>

<h1>Krychle SALES</h1>
<p><object classid="clsid:0002E520-0000-0000-C000-000000000046"
  id="OWC1" width="640" height="480">
  </object></p>

<script language="JavaScript">
// Odkaz na objekt Pivot Table.
   var objPT = document.all.OWC1;
   objPT.DisplayFieldList = 1;

// Připojení Pivot Table na server OLAP.
   objPT.ConnectionString = "Provider=msolap;Data Source=localhost;Initial
Catalog=FoodMart 2000";
   objPT.DataMember = "Sales";
</script></body></html>
```

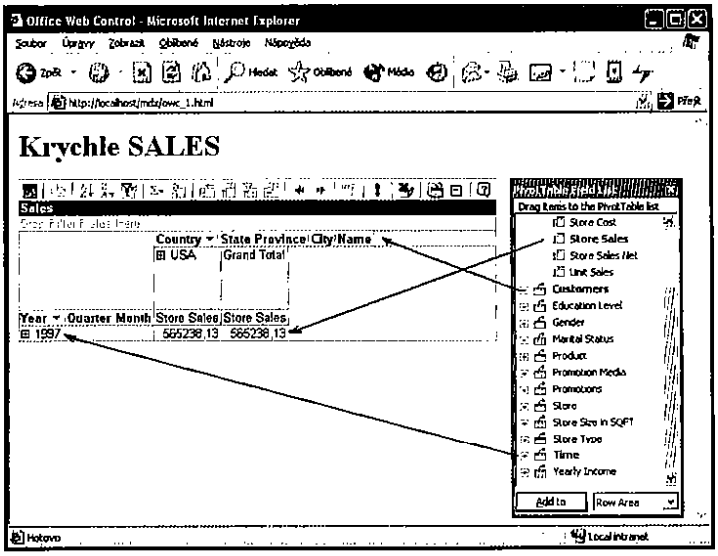


Obr. 7.6 Office Web Components – Pivot Table na stránce HTML klienta

Po načtení stránky HTML se zobrazí kontingenční tabulka a seznam měrných jednotek a dimenzí v okně **PivotTable Field List**. V této etapě můžeme metodou drag-and-drop přesunout ikonky měrných jednotek a úrovní dimenzí na příslušná pole kontingenční tabulky. Výhodnější je přesunout nejdříve dimenze a až pak měrné jednotky, abychom neztratili orientaci a přehled v návrhu tabulky.

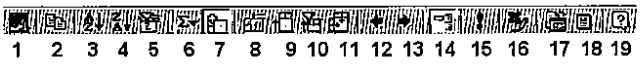
Všimněme si panelu tlačítek zobrazené komponenty. Jednotlivá tlačítka nám poskytují velmi široké a komfortní možnosti práce s komponentou a režimem jejího zobrazování. Jejich význam je v tabulce.

Číslo	Význam
1	Zobrazení dialogu „About Microsoft Office Web Components".
2	Kopírování vybraných údajů do schránky.
3, 4	Setřídění podle řádků nebo sloupců.
5	Zobrazení zaškrtávacích políček pro výběr úrovní řádků a sloupců.
6	Sumarizační funkce Sum, Count, Min nebo Max.
7	Zobrazení mezisouhrnů pro vybraná pole.
8	Přesun vybraného pole do oblasti řádků.
9	Přesun vybraného pole do oblasti sloupců.



Obr. 7.7 Office Web Components – definování faktů a dimenzí

Číslo	Význam
10	Přesun vybraného pole do oblasti filtru.
11	Přesun vybraného pole do datové oblasti.
12	Výběr následující vnější úrovně.
13	Výběr následující vnitřní úrovně.
14	Zobrazení detailů v datové oblasti.
15	Aktualizace údajů podle zdrojové databáze.
16	Spuštění programu Excel a zkopírování tabulky do jeho pracovního sešitu.
17	Zobrazení Panelu vlastností kontingenční tabulky.
18	Zobrazení hierarchické úrovně polí zdrojových údajů.
19	Zobrazení nápovědy.



Obr. 7.8 Panel tlačítek OWC komponenty Pivot Table

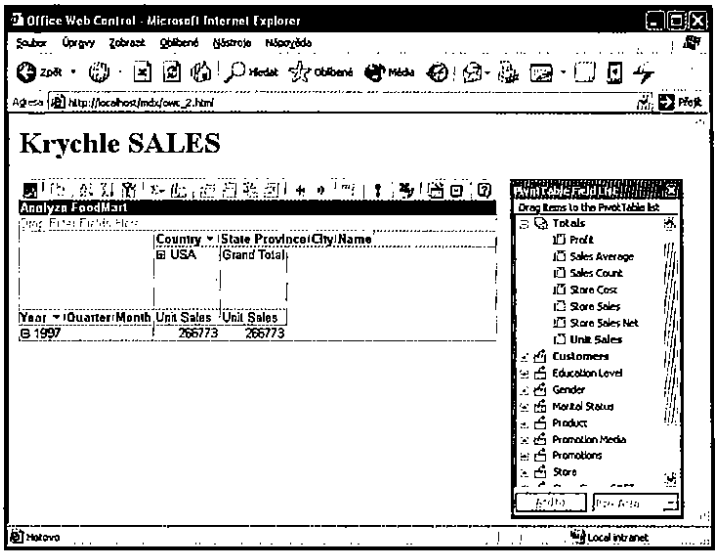
Definování polí OWC komponenty PivotTable

V prvním příkladu jsme u klienta zobrazili komponentu OWC PivotTable sice připojenou k serveru OLAP, ale „prázdnou", tzn., že si klient musel sám nadefinovat zobrazovací strukturu kontingenční tabulky. Jak by pochodil vývojář nebo analytik, kdyby takovou aplikaci v „surovém stavu" poskytl svému manažerovi, raději ani nebudeme uvažovat. Podstatným krokem vpřed bude zobrazení kontingenční tabulky v kontextu stránky HTML u klienta už s předdefinovanou návrhovou strukturou. Proto v tomto příkladu nadefinujeme objekty `objView.RowAxis`, `objView.ColumnAxis` a `objView.DataAxis`. Aby byl dojem ze zobrazené stránky co nejlepší, nadefinujeme i nadpis v záhlaví kontingenční tabulky `Analyza FoodMart`.

```
var objView = objPT.ActiveView;
objView.TitleBar.Caption = "Analyza FoodMart";
var objFlds = objPT.ActiveView.FieldSets;
objView.RowAxis.InsertFieldSet(objFlds("[Time]"));
objView.ColumnAxis.InsertFieldSet(objFlds("[Customers]"));
objView.DataAxis.InsertTotal(objView.Totals("Unit Sales"));
```

Kompletní výpis kódu stránky OWC_2.HTML bude:

```
<h1>Krychle SALES</h1>
<p><object classid="clsid:0002E520-0000-0000-C000-000000000046"
  id="OWC1" width="640" height="480">
</object></p>


<script language="JavaScript">

// reference na objekt Pivot Table
   var objPT = document.all.OWC1;

// připojení PivotTable na server OLAP
   objPT.ConnectionString = "Provider=msolap;Data Source=localhost;Initial
Catalog=FoodMart 2000";
   objPT.DataMember = "Sales";

//Nastavení vlastností PivotTable
   objPT.AllowFiltering = 1;
   objPT.DisplayFieldList = 1;
   objPT.DisplayExpandIndicator = 1;
   objPT.DisplayToolbar = 1;


 //definice polí
     var objView = objPT.ActiveView;
     objView.TitleBar.Caption = "Analyza FoodMart";
     var objFlds = objPT.ActiveView.FieldSets;
     objView.RowAxis.InsertFieldSet(objFlds("[Time]"));
     objView.ColumnAxis.InsertFieldSet(objFlds("[Customers]"));
     objView.DataAxis.InsertTotal(objView.Totals("Unit Sales"));

</script></body></html>
```



Obr. 7.9 Office Web Components – předdefinovaná fakta a dimenze

Vypočtené souhrny

Kromě hodnot, které jsou uložené v multidimenzionálních databázích nebo lokálních krychlích (soubory s příponou .CUB), můžeme pomocí kontingenční tabulky z balíku Office Web Components zobrazit i různé vypočtené souhrny z měrných jednotek.

Ukážeme si jednoduchý příklad stránky HTML, která umožňuje výběr předdefinovaných vypočtených souhrnů, například

Roční prodej	`Sum(YTD(),[Measures].[Store Sales]`
Prodej v předchozím období	`([Measures].[Store Sales], Time.PrevMember)`
Prodej v paralelním období	`(ParallelPeriod(, [Measures].[Store Sales])`
...	...

Použijeme funkci

```
ptable.ActiveView.AddCalculatedTotal(Název, zahlavi,výraz)
```

kde jako parametr zadáme požadovaný výraz.

Kompletní kód stránky HTML je:

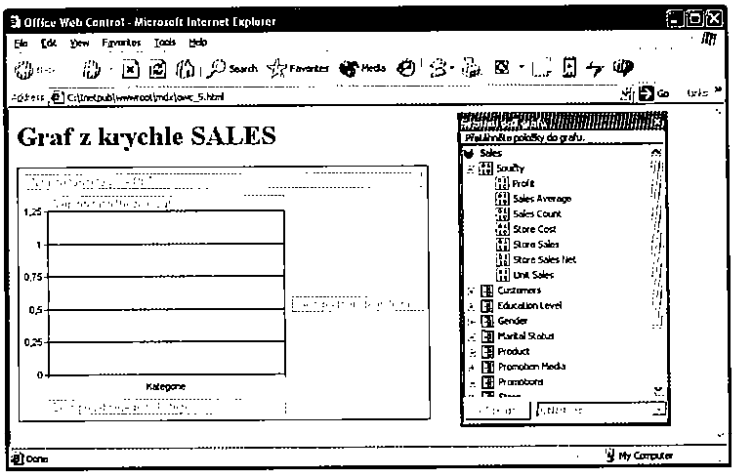
```
<html><head></head><body>
<h1>Vypocitane souhrny</h1>

<SELECT id=cbxExprs>
    <OPTION selected value="">(--Vypocitany souhrn--)</OPTION>
        <option value="Sum(YTD(),[Measures].[Store Sales])">Roční prodej</option>
        <option value="([Measures].[Store Sales], Time.PrevMember)">
        Prodej v predchozim obdobi</option>
        <option value="(ParallelPeriod(), [Measures].[Store Sales])">
        Prodej v soubeznem obdobi</option>
</SELECT>

<br><TEXTAREA rows=4 id=txtExpr wrap="off" style="width=100%"></TEXTAREA>
<br><INPUT type="button" value="Vypocet souhrnu" id=btnCreate></p>
<hr>

<OBJECT CLASSID="clsid:0002E552-0000-0000-C000-000000000046"
  id=ptable>
</OBJECT>

<script language=vbscript>
Option Explicit

Sub Window_onLoad()
    ptable.ConnectionString = "provider=MSOLAP;data source=LOCALHOST;initial cata-
log=FoodMart 2000"
    ptable.DataMember = "Sales"
    ptable.ActiveView.DisplayCalculatedMembers = True

    Dim view
    set view = ptable.ActiveView
    view.RowAxis.InsertFieldset view.Fieldsets("Time")
    view.DataAxis.InsertTotal view.Totals("Store Sales")
End Sub

Sub btnCreate_onClick()
    Dim sName
    Dim ttl

    sName = cbxExprs.options(cbxExprs.selectedIndex).text
    set ttl = ptable.ActiveView.AddCalculatedTotal(sName, sName,txtExpr.value)
    ptable.ActiveView.DataAxis.InsertTotal ttl
End Sub

Sub cbxExprs_onChange()
    txtExpr.value = cbxExprs.value
End Sub

</script></body></html>
```

Komponenta OWC Chart

I další užitečná komponenta OWC Chart, kterou můžeme hojně využívat pro zobrazování multidimenzionálních údajů, je totožná s kontingenčním grafem programu Excel, ale co je vynikající, na stránkách HTML se s ní pracuje velmi podobně jako s komponentou OWC PivotTable. Když si v předchozí stati prohlédneme první příklad stránky HTML s komponentou OWC PivotTable, stačí v kódu stránky zaměnit ID komponenty za `classid=CLSID:0002E556-0000-0000-C000-000000000046`.

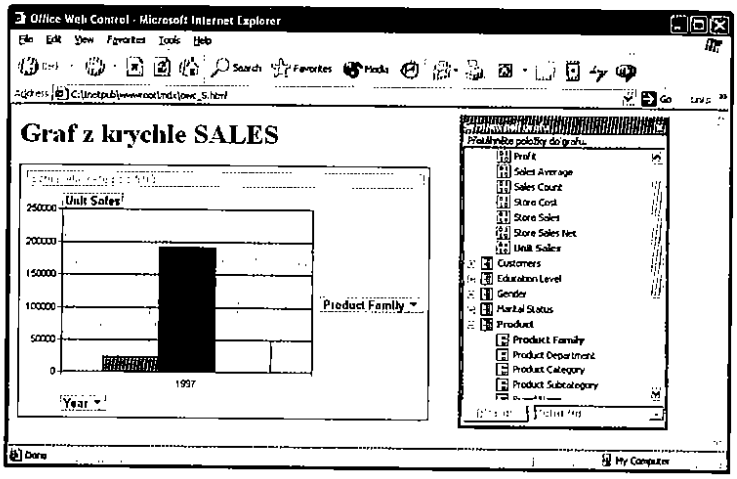
Kompletní výpis kódu stránky OWC_5.HTML je:

```
<html><head><title>Office Web Control</title></head><body>

<h1>Graf z krychle SALES</h1>
<p><object id=Chart1 classid=CLSID:0002E556-0000-0000-C000-000000000046
   style="width:60%;height:300"></object></p>

<script language="JavaScript">
// odkaz na objekt Chart
   var objChart = document.all.Chart1;
   objChart.DisplayFieldList = 1;

// pripojeni na server OLAP
   objChart.ConnectionString = "Provider=msolap;Data Source=localhost;Initial
Catalog=FoodMart 2000";
   objChart.DataMember = "Sales";

</script></body></html>
```



Obr. 7.10 OWC Chart

Po načtení stránky HTML můžeme podobně jako u komponenty OWC PivotTable metodou drag-and-drop přesunout ikonky měrných jednotek a úrovní dimenzí na příslušná pole formuláře grafu.



Obr. 7.11 OWC Chart po návrhu zobrazení

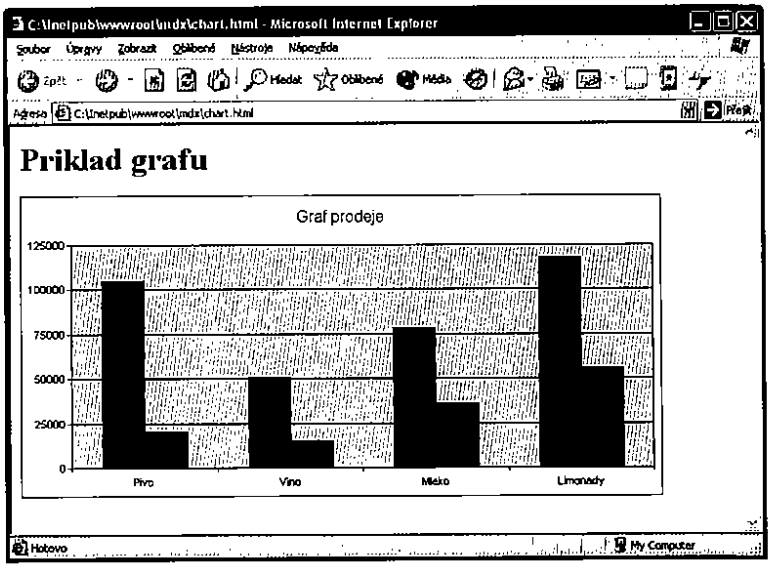
Teoretické a praktické minimum ke grafům

Pokud bychom chtěli navrhnout graf podle druhého příkladu z OWC PivotTable, tady bychom neuspěli. Aby bylo zřejmé, jak se definují parametry a odevzdávají údaje pro graf založený na komponentě OWC Chart, ukážeme si dva příklady jednoduchého grafu se dvěma sériemi. Použijeme implicitní hodnoty zadané přímo v kódu. Oba dva příklady jsou na stránce HTML a je možné je otevřít v prohlížeči i v režimu off-line. Pro zadávání parametrů je potřeba použít skriptovací jazyk interpretovaný u klienta. Pro první příklad je použit skriptovací jazyk JavaScript a pro druhý příklad skriptovací jazyk VisualBasic Script (VBScript). V jednotlivých příkladech si ukážeme i různé způsoby zobrazování série.

Zdrojový kód stránky **CHART.HTML** (použitý JavaScript):

```
html><head></head><body>
<h1>Priklad grafu</h1>
<p><object id=Chart1 classid=CLSID:0002E556-0000-0000-C000-000000000046
   style="width:60%;height:300"></object>
</p></p>

<script language="JavaScript">

// odkaz na objekt chart
   var objChart = document.all.Chart1;
```

```
    xValues = new Array("Pivo", "Vino", "Mleko", "Limonady");
    yValues1 = Array(104737, 50952, 78128, 117797);
    yValues2 = Array(20000, 15000, 36000, 56000);
    var c = objChart.constants;

  //Zahlavi grafu
    objChart.HasChartSpaceTitle = 1;
    objChart.ChartSpaceTitle.Caption = "Graf prodeje";

  //Pridame dve serie pro radky a sloupce
   objChart.Charts.Add();
   objChart.Charts(0).SeriesCollection.Add();
   objChart.Charts(0).SeriesCollection.Add();


  //Prvni serie
     objChart.Charts(0).SeriesCollection(0).SetData(c.chDimCategories,
c.chDataLiteral, xValues);
     objChart.Charts(0).SeriesCollection(0).SetData(c.chDimValues,
c.chDataLiteral, yValues1);
```



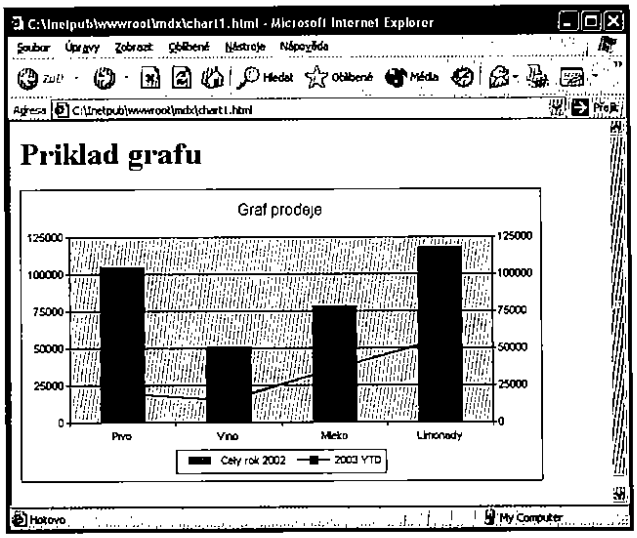
Obr. 7.12 OWC Chart – příklad jednoduchého grafu

```
  //Druha serie
      objChart.Charts(0).SeriesCollection(1).SetData(c.chDimCategories,
c.chDataLiteral, xValues);
      objChart.Charts(0).SeriesCollection(1).SetData(c.chDimValues,
c.chDataLiteral, yValues2);

</script></body></html>
```

Zdrojový kód stánky **CHART1.HTML** (použitý VBScript):

```
<html><head></head><body>
<h1>Priklad grafu</h1>
<p><object id=Chart1 classid=CLSID:0002E556-0000-0000-C000-000000000046
   style="width:90%;height:300"></object>
</p></p>

<script language="VBScript">

   'Deklarace promennych
    Dim c
    Set c = Chart1.constants
    Dim oSeries

   Dim xValues, yValues1, yValues2
       xValues = Array("Pivo", "Vino", "Mleko", "Limonady")
       yValues1 = Array(104737, 50952, 78128, 117797)
       yValues2 = Array(20000, 15000, 36000, 56000)

       'Inicializace grafu
       Dim oChart
       Chart1.Clear
       Set oChart = Chart1.Charts.Add

     'Zahlavi grafu
       oChart.HasTitle = True
       oChart.Title.Caption = "Graf prodeje"


      'První serie pro rok 2002 - sloupcovy graf
       Set oSeries = oChart.SeriesCollection.Add
       oSeries.Caption = "Cely rok 2002"
       oSeries.SetData c.chDimCategories, c.chDataLiteral, xValues
       oSeries.SetData c.chDimValues, c.chDataLiteral, yValues1
       oSeries.Type = c.chChartTypeColumnClustered

       'Druha serie 2003 YTD
       'typ LineMarkers ve stejne souradnicov soustave
       Set oSeries = oChart.SeriesCollection.Add
       oSeries.Caption = "2003 YTD"
       oSeries.SetData c.chDimCategories, c.chDataLiteral, xValues
       oSeries.SetData c.chDimValues, c.chDataLiteral, yValues2
```

```
        oSeries.Type = c.chChartTypeLineMarkers

        'osa Y pro druhou serii vpravo
         Dim axisScale
         Dim axRightAxis
         Set axisScale = oChart.Axes(c.chAxisPositionLeft).Scaling
         Set axRightAxis = oChart.Axes.Add(axisScale)
         axRightAxis.Position = c.chAxisPositionRight

       'Legenda v dolni casti grafu
        oChart.HasLegend = True
        oChart.Legend.Position = c.chLegendPositionBottom

</script></body></html>
```



Obr. 7.13 OWC Chart – příklad jednoduchého grafu

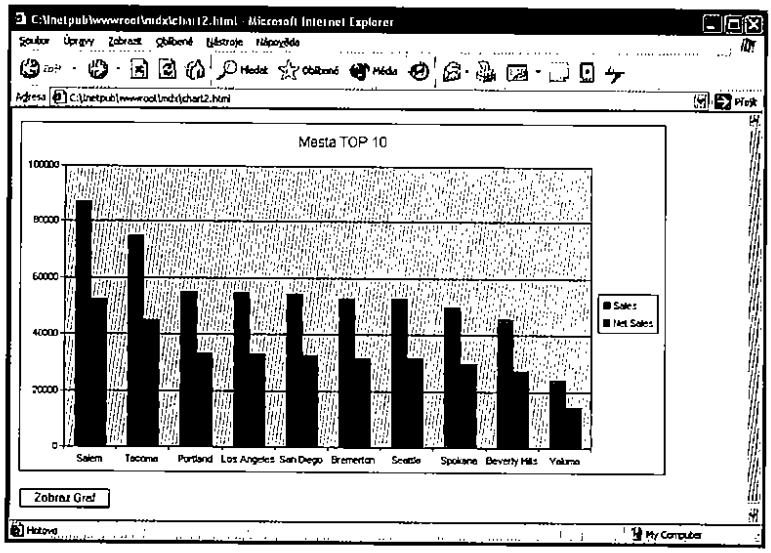
Zobrazení údajů přes OWC Chart pomocí ADODB

Pokud chceme využít komponentu OWC Chart pro zobrazení údajů získaných z multidimenzionální databáze na základě dotazu jazyka MDX, můžeme komponentu Chart připojit na dva druhy zdrojů údajů:

- ADODB Recordset
- ADOMD Cellset

Připojení komponenty OWC Chart na ADODB Recordset se provede pomocí kódu

```
Dim oConn, oRS
Set oConn = CreateObject("ADODB.Connection")
Set oRS = CreateObject("ADODB.Recordset")
oConn.Open sConn
oRS.Open sMDX, oConn, 3 'adOpenStatic

'Graf na zaklade udaju z rekordsetu
 Chart1.Clear
 Set Chart1.DataSource = oRS
```



Obr. 7.14 OWC Chart – zobrazení údajů z dotazu MDX přes ADODB recordset

Kompletní zdrojový kód stránky HTML **CHART2.HTML:**

```
<HTML><BODY>
<OBJECT classid=clsid:0002E500-0000-0000-C000-000000000046
id=Chart1 style="WIDTH:90%; HEIGHT:400"></OBJECT>
<P><P>
<BUTTON ID=btnZobraz>Zobraz Graf </BUTTON>

<SCRIPT Language="VBScript">
```

```
Dim sMDX, sConn, c, a

sMDX = "SELECT Measures.MEMBERS ON COLUMNS, " & _
       "TOPCOUNT([Store].[Store City].MEMBERS), 10, " & _
       "Measures.[Store Sales Net]) ON ROWS From [Sales]"

sConn = "Provider=msolap;Initial Catalog=FoodMart 2000;Data Source=localhost"

Set c = Chart1.Constants

Function btnZobraz_OnClick()

    'Rekordset na zaklade dotazu MDX
    Dim oConn, oRS
    Set oConn = CreateObject("ADODB.Connection")
    Set oRS = CreateObject("ADODB.Recordset")
    oConn.Open sConn
    oRS.Open sMDX, oConn, 3 'adOpenStatic

    'Graf na zaklade udaju z rekordsetu
     Chart1.Clear
     Set Chart1.DataSource = oRS
     Dim oChart
     Set oChart = Chart1.Charts.Add
     oChart.HasTitle = True
     oChart.Title.Caption = "Mesta TOP 10"
     oChart.HasLegend = True

   'Serie pro Sales
     Set a = oChart.SeriesCollection.Add
       a.Caption = "Sales"
       a.SetData c.chDimCategories, 0, oRS.Fields(2).Name
       a.SetData c.chDimValues, 0, oRS.Fields(5).Name

   'Serie pro Net Sales
     Set a = oChart.SeriesCollection.Add
        a.Caption = "Net Sales"
        a.SetData c.chDimCategories, 0, oRS.Fields(2).Name 'City
        a.SetData c.chDimValues, 0, oRS.Fields(7).Name     'Net Sales

    Chart1.Refresh

    'Zavreni objektu
    oRS.Close
    oConn.Close

End Function
</SCRIPT></BODY></HTML>
```

Stručné minimum stránek ASP

Pro stránky ASP vlastně kromě operačního systému Windows a webového serveru, který je též k dispozici na instalačním CD Windows, nepotřebujeme nutně žádné vývojové prostředí. Stránkami ASP, které tvoří projekt, jsou soubory umístěné v příslušném adresáři na serveru s příponou ASP. V našich příkladech bude celý projekt tvořen vždy jen jednou stránkou ASP. Soubory ASP jsou v principu stránky HTML (s příponou ASP) doplněné o kód ve skriptovacím jazyce (JScript, VBScript), který je od kódu HTML oddělený oddělovači **<% ... %>**. Pokud řádek na stránce ASP obsahuje kód skriptu, interpretuje se a jeho výstupy jsou vloženy do stránky HTML pro klienta.

Častá otázka začátečníků je: „Kam je potřeba tento soubor umístit, aby ho aplikační server interpretoval." Při standardní instalaci Personal Web Serveru, případně Internet Information Serveru, se na disku vytvoří adresář **C:\INETPUB**. Tento adresář obsahuje další podadresáře, například **AdminScripts, MailRoot, Scripts, Webpub, wwwroot** a podobně. Nejjednodušší je umístit soubor s příponou ASP buď přímo do podadresáře wwwroot, případně v něm vytvořit další podadresář. Hlavní dialog Personal Web Serveru, případně Internet Information Serveru, nám prozradí, jaká je webová adresa našeho serveru, například:

```
Publikování ve www je zapnuto. Vaše domovská stránka je k dispozici na adrese:
http://llhome
Domovský adresář: C:\inetpub\wwwroot
```

Pokud tedy do adresáře **C:\inetpub\wwwroot** umístíme soubor mdx.asp, aktivujeme ho zadáním adresy

```
http://llhome/mdx.asp
```

Pokud umístíme soubor **mdx.asp** do podadresáře, například:

```
C:\inetpub\wwwroot\projekt\pokus.asp
```

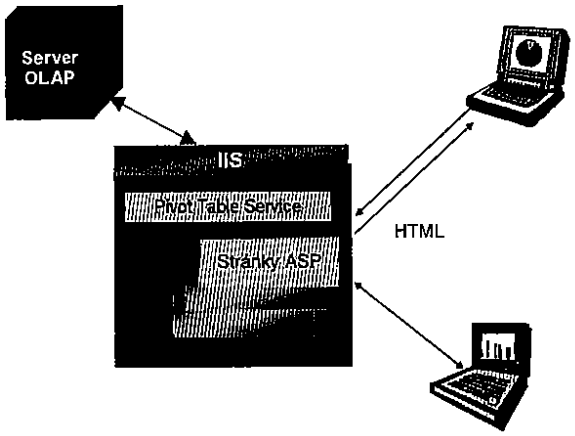
aktivujeme ho v našem případě zadáním adresy:

```
http://llhome/projekt/mdx.asp
```

Stránky ASP provádí přístup k údajům z analytických databází přes komponentu Pivot Table Service prostřednictvím Internet Information Serveru (IIS).

K údajům v multidimenzionálních databázích budeme provádět přístup přes rozhraní ADOMD, proto si musíme vytvořit objekty

```
Set cat = Server.CreateObject("ADOMD.Catalog")
Set cst = Server.CreateObject("ADOMD.CellSet")
```



Obr. 7.15 Principiální schéma analytické aplikace ASP

Pro výběr údajů použijeme například tento příkaz SELECT pro výběr údajů z krychle SALES.

```
SELECT
   { [Measures].[Unit Sales], [Measures].[Store Sales] } ON COLUMNS,
   { [Customers].[Country].[USA].[CA], [Customers].[Country].[USA].[OR],
[Customers].[Country].[USA].[WA]} ON ROWS
  FROM Sales
  WHERE ([Gender].[F] )
```

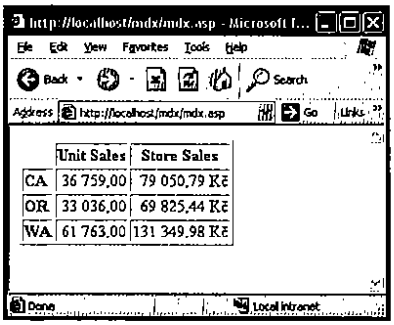
Údaje zobrazíme na stránce HTML u klienta ve formě tabulky. Počet a názvy dimenzí pro vytvoření záhlaví tabulky nemusíme navrhovat „natvrdo". Získáme je z objektu ADOMD.Catalog. Kompletní kód stránky ASP pro náš dotaz potom bude:

```
<%@ Language=VBScript %>
<HTML><HEAD></HEAD><BODY>

<%
Dim cat,cst,i,j,sSelect,csw,iDC0,iDC1,iPC0, iPC1

'Pripojeni se k objektum ADOMD.Catalog a ADOMD.CellSet
  Set cat = Server.CreateObject("ADOMD.Catalog")
  Set cst = Server.CreateObject("ADOMD.CellSet")

'Pripojeni se k serveru OLAP
        OLAPServerName = "LOCALHOST"
        cat.ActiveConnection = "Data Source=" & OLAPServerName & ";Initial
Catalog=FoodMart 2000;Provider=msolap;"
```

```
'Vytvoreni prikazu SELECT jazyka MDX a nastaveni CellSet Source
  sSelect = sSelect & "SELECT "
  sSelect = sSelect & " { [Measures].[Unit Sales], [Measures].[Store Sales] } ON
COLUMNS,"
  sSelect = sSelect & " ( [Customers].[Country].[USA].[CA],
[Customers].[Country].[USA].[OR], [Customers].[Country].[USA].[WA]) ON ROWS"
  sSelect = sSelect & " FROM Sales WHERE ([Gender].[F] )"
  cst.Source = sSelect

'Nastaveni a otevreni aktivniho pripojeni
  Set cst.ActiveConnection = cat.ActiveConnection
  cst.Open

'Pocty dimenzi v jednotlivych osach
  iDC0 = cst.Axes(0).DimensionCount-1
  iDC1 = cst.Axes(1).DimensionCount-1
  iPC0 = cst.Axes(0).Positions.Count - 1
  iPC1 = cst.Axes(1).Positions.Count - 1

'Vypis nazvu dimenzi vodorovne do zahlavi tabulky
  Response.Write "<Table border=1>"
  For h=0 to iDC0
    Response.Write "<TR>"
    For c=0 to iDC1
      Response.Write "<TD></TD>"
    Next

   For i = 0 To iPC0
     Response.Write "<TH>"
     Response.Write cst.Axes(0).Positions(i).Members(h).Caption
     Response.Write "</TH>"
   Next
  Response.Write "</TR>"

Next

'Vypis nazvu dimenzi svisle do prvnich sloupcu tabulky
  For j = 0 To iPC1
    Response.Write "<TR>"
    For h=0 to iDC1
      Response.Write "<TD><B>"
      Response.Write cst.Axes(1).Positions(j).Members(h).Caption
      Response.Write "</B></TD>"
    Next

'Vypis hodnot do tabulky
    For k = 0 To iPC0
     Response.Write "<TD align=right>"
     Response.Write cst(k, j).FormattedValue
     Response.Write "</TD>"
   Next
```

```
    Response.Write "</TR>"
  Next
  Response.Write "</Table>"

%>
</BODY></HTML>
```

U klienta se zobrazí výsledky na stránce HTML ve formě tabulky.



Obr. 7.16 Tabulka výsledků aplikace ASP

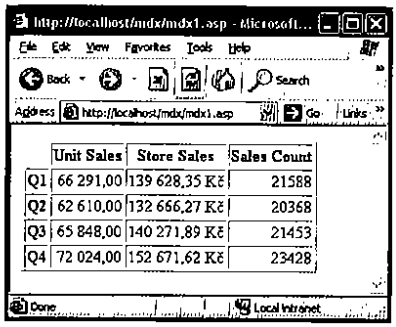
Výhody „dynamického" návrhu tabulky se ukáží, když změníme příkaz SELECT, například na

```
SELECT
   { [Measures].[Unit Sales], [Measures].[Store Sales], [Measures].[Sales Count] } ON COLUMNS,
   { [Time].[1997].[Q1], [Time].[1997].[Q2], [Time].[1997].[Q3], [Time].[1997].[Q4] } ON ROWS
  FROM Sales
```

V kódu stránky ASP se změní řádky.

```
'Vytvoreni prikazu SELECT jazyka MDX a nastaveni CellSet Source
  sSelect = sSelect & "SELECT "
  sSelect = sSelect & "{[Measures].[Unit Sales], [Measures].[Store Sales],
[Measures].[Sales Count]} ON COLUMNS,"
  sSelect = sSelect & "{[Time].[1997].[Q1], [Time].[1997].[Q2],
[Time].[1997].[Q3], [Time].[1997].[Q4]} ON ROWS"
  sSelect = sSelect & " FROM Sales"
  cst.Source = sSelect
```

Výsledkem na stránce HTML u klienta bude požadovaná tabulka.



	Unit Sales	Store Sales	Sales Count
Q1	66 291,00	139 628,35 Kč	21588
Q2	62 610,00	132 666,27 Kč	20368
Q3	65 848,00	140 271,89 Kč	21453
Q4	72 024,00	152 671,62 Kč	23428

Obr. 7.17 Tabulka výsledků aplikace ASP pro jiný dotaz

Od aplikace tohoto typu je jen krůček k vytvoření „univerzální" konzolové aplikace MDX. Stačí doplnit formulář pro zadání dotazu MDX a vypsat výsledek ve formě tabulky na stránce HTML. Asi nejvíce námahy budeme muset vynaložit na správně hierarchicky strukturovaný výpis názvů dimenzí v záhlaví sloupců a řádků. Kompletní kód konzolové aplikace MDX realizované jako stránka ASP je (soubor MDXkonzola.asp):

```
<%@ Language=VBScript %>
<%
Response.Buffer=True
Response.Expires=0
%>
<html><head></head><body>
<%

Dim cat,cst,i,j,sSelect,LevelValue,iDC0,iDC1,iPC0, iPC1

Set cat = Server.CreateObject("ADOMD.Catalog")
Set cst = Server.CreateObject("ADOMD.CellSet")
cat.ActiveConnection = "Data Source=LOCALHOST:Initial Catalog=FoodMart
2000;Provider=msolap;"

'Prikaz SELECT bud z okna formulare nebo ze sesion
Session("MDXQuery")=Request.Form("MDXQuery")
If Len(Session("MDXQuery")) < 5 Then
        sSelect = "SELECT [ [Measures].MEMBERS ] ON COLUMNS, [ [Time].MEMBERS ] ON
ROWS FROM Sales"
Else
        sSelect = Session("MDXQuery")
End if
cst.Source = sSelect

Set cst.ActiveConnection = cat.ActiveConnection
cst.Open
```

```
%>
<form action="MDXkonzola.asp" method="POST" id="form1" name="form1">
<b>Konzola MDX</b><br>
<textarea rows="8" cols="80" id="textareaMDX" name="MDXQuery" wrap="soft">
<%=Session("MDXQuery")%></textarea>
<p><input type="submit" value="Dotaz MDX" id="submit1" name="submit1"></p>
</form>
<p><%=strSource%></p>
<%

'Pocty dimenzi
 iDC0 = cst.Axes(0).DimensionCount-1
 iDC1 = cst.Axes(1).DimensionCount-1
 iPC0 = cst.Axes(0).Positions.Count - 1
 iPC1 = cst.Axes(1).Positions.Count - 1

'HTML tabulka, pripravime kostru tabulky
 Response.Write "<Table  border=1>"
 For h=0 to iDC0
    Response.Write "<TR>"
    For c=0 to iDC1
        Response.Write "<TD></TD>"
    Next

 'je aktualni dimenze shodna s predchozi
    If h = iDC0 then

  'Vypis nazvu dimenzi do zahlavi sloupcu
    For i = 0 To iPC0
      Response.Write "<TH>"& cst.Axes(0).Positions(i).Members(h).Caption &"</TH>"
    Next
    Else

'Preskakovani nazvu dimenzi pokud uz byly vypsany
    CaptionCount = 1
    LastCaption = cst.Axes(0).Positions(0).Members(h).Caption
    Response.Write "<TH"
    For t=1 to iPC0

'Pokud je predchozi text je shodny s aktualnim inkrementujeme pocitadlo
        If LastCaption = cst.Axes(0).Positions(t).Members(h).Caption then
          CaptionCount = CaptionCount+1

       'jinak ho zapiseme jako colspan do tabulky HTML
          If  t = iPC0 then
            Response.Write " colspan=" & CaptionCount & ">" & LastCaption & "</TH>"
          End if
          Else
           Response.Write " colspan=" & CaptionCount & ">" & LastCaption & "</TH><TH"
           CaptionCount=1
           LastCaption=cst.Axes(0).Positions(t).Members(h).Caption
```

```
        End if
    Next
  End if
      Response.Write "</TR>"
Next

 'Pozice pro vypis do pole intArray
 Dim arRiadky(), intArray, Poloha, sDimenzia
 intArray=0
 Poloha=0
 For a=1 To iDC1
      intArray = intArray+(iPC1+1)
 Next
 intArray = intArray-1
 ReDim arRiadky(intArray)

'Formatovani zahlavi radku
  For j = 0 To iPC1
      Response.Write "<TR>"
      For h=0 to iDC1
    sDimenzia = cst.Axes(1).Positions(j).Members(h).Caption
    Response.Write "<TD><B>"
        If h=iDC1 then
            Response.Write sDimenzia
        Else
            arRiadky(Poloha) = sDimenzia
            If Poloha < iDC1 then
            Response.Write sDimenzia
            Else
         'vyssi hierarchicky prvek dimenze se vypise jen jednou
         'v ostatnich bunkach tabulky jsou mezery
            If arRiadky(Poloha) = arRiadky(Poloha - iDC1) then
                Response.Write " "
            Else
                Response.Write sDimenzia
            End if
        End if
        Response.Write "</B></TD>"
        Poloha = Poloha + 1
      End if
  Next

 'Vypis hodnot do bunek tabulky
 'pre vetsi nazornost stridame barvy susednich bunek
For k = 0 To iPC0
 If (j+1) Mod 2 = 0 Then
      Response.Write "<TD align=right bgcolor=cccccc>"
 Else
      Response.Write "<TD align=right bgcolor=dddddd>"
 End If
     Response.Write cst(k, j).FormattedValue &"</TD>"
```

```
   Next
    Response.Write "</TR>"
Next
Response.Write "</Table>"

%>
</font></body></html>
```

Microsoft XML for Analysis SDK

Instalace balíku XML for Analysis SDK spočívá ve stažení přibližně jednomegabajtového instalačního souboru MSXAINST.EXE z webu www.microsoft.com/download a jeho spuštění. Po nainstalování je potřeba nastavit konfigurační parametry.

Dokument XML

Výhodou technologie XML je její nezávislost na použité softwarové platformě, ať už se jedná o operační systém, databázi a podobně. XML definuje jednak formát údajů a jednak i operace s těmito údaji. Při pohledu na obsah souboru XML otevřený například v textovém editoru si určitě uvědomíme, že je tento formát pro výměnu údajů stejně dobře čitelný pro počítače jako pro lidi. Tyto údaje můžeme lehce načítat a identifikovat pomocí poměrně jednoduché rutiny v libovolném programovacím jazyce.

Pod pojmem dokument XML rozumíme dokument, například soubor s příponou XML, jehož obsah splňuje pravidla syntaxe značkovacího jazyka XML. Dokument se skládá z elementů. Každý element obsahuje počáteční a koncovou značku, jinými slovy počáteční a koncový tag. Oba dva tagy obsahují název elementu a koncový tag obsahuje navíc před názvem elementu lomítko. Typickým příkladem elementu XML může být například element obsahující jméno a příjmení, například z naší tabulky PRACOVNICI.

```
<jmeno>Novak Alfonz</jmeno>
```

Každý element může v sobě obsahovat další, takzvané vnořené elementy.

```
<pracovnik>
   <jmeno>Novak Alfonz</jmeno>
   <funkce>ředitel</funkce>
   <mzda>50000</mzda>
</pracovnik>
```

Už na základě tohoto jednoduchého příkladu můžeme usoudit, že dokument XML má hierarchickou stromovou strukturu. Úroveň vnoření jednotlivých elementů není omezená, ale je vyžadováno, aby měl každý dokument XML jeden kořenový element, například

```
<?xml version="1.0"?>
<pracovnici>
  <pracovnik>
     <jmeno>Novak Alfonz</jmeno>
     <funkce>ředitel</funkce>
     <mzda>50000</mzda>
  </pracovnik>
```

```
  <pracovnik>
     <jmeno>Plha Jan</jmeno>
     <funkce>vedoucí marketingu</funkce>
     <mzda>32000</mzda>
  </pracovnik>
</pracovnici>
```

Konfigurace XML for analysis SDK

Po krátkém představení technologie XML se vraťme k XML for analysis SDK. Samozřejmě, jelikož se jedná o technologii XML, budeme i parametry nastavovat v souboru XML. Při standardní instalaci v souboru *C:\Program Files\Microsoft XML for Analysis SDK\config\datasources.xml*. Struktura konfiguračního souboru je uvedena v komentáři na začátku konfiguračního souboru.

```
<DataSources>
    <DataSource>
    <DataSourceName>Jmeno</DataSourceName>
    <DataSourceDescription>Popis</DataSourceDescription>
    <URL>URL</URL>
    <DataSourceInfo>pripojovaci retezec</DataSourceInfo>
    <ProviderName>jmeno providera</ProviderName>
    <ProviderType><TDP/><MDP/><DMP/></ProviderType>
    <AuthenticationMode>Unauthenticated/Authenticated/Integrated</AuthenticationMode>
    </DataSource>
  ...
</DataSources>
```

Vysvětlení si zaslouží snad jen řádek

```
<ProviderType><TDP/><MDP/><DMP/></ProviderType>
```

kde zkratky pro typ providera mají význam

TDP – Tabular Data Provider,

MDP – Multidimensional Data Provider,

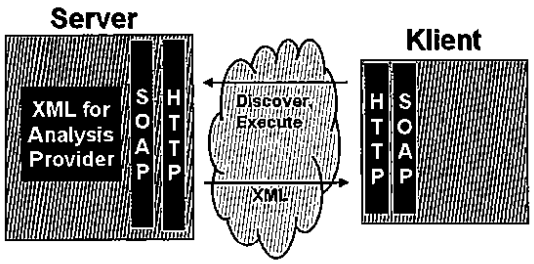
DPM – OLEDB pre DM data provider.

Údaje pro konkrétní konfiguraci byly v našem případě tyto:

```
<DataSources UnnamedSessionsTimeout="300" NamedSessionsTimeout="3600">
 <DataSource>
    <DataSourceName>Local Analysis Server</DataSourceName>
    <DataSourceDescription>MS Analysis Server 2000</DataSourceDescription>
    <URL>http://localhost/xmlanalysis/msxisapi.dll</URL>
    <DataSourceInfo>Provider=MSOLAP;Data Source=localhost</DataSourceInfo>
    <ProviderName>Microsoft XML for Analysis</ProviderName>
    <ProviderType><TDP/><MDP/><DMP/></ProviderType>
    <AuthenticationMode>Unauthenticated</AuthenticationMode>
 </DataSource>
```

Na závěr ještě musíme nakonfigurovat přístup k souboru ISAPI DLL `msxisapi.dll`. Máme dvě možnosti – buď vytvořit virtuální adresář nad adresářem, kde je tento soubor fyzicky umístěný při instalaci, implicitně C:\Program Files\Microsoft XML for Analysis SDK\Isapi, nebo vytvoříme nový adresář v pracovním adresáři webového serveru IIS C:\inetpub\www.root\xmlanalysis a soubor ISAPIDLL do tohoto adresáře zkopírujeme. Potom bude odkaz na adresu URL ve tvaru

```
<URL>http://localhost/xmlanalysis/msxisapi.dll</URL>
```



Obr. 7.18 Schéma XML for Analysis SDK

Metody

Discover

Metoda Discover umožňuje získat informace o webové službě, dostupných datových zdrojích, případně o dalších metadatech.

Execute

Metoda Execute umožňuje zadání příkazu pro XML for Analysis. Údaje jsou samozřejmě také vráceny ve formátu XML.

Můžeme to předvést na jednoduchém příkladu. Webové službě předáme

```
SELECT { [Measures].[Unit Sales]} ON COLUMNS FROM sales
```

Příklad je opět realizován jako obsluha klepnutí na tlačítko ve formuláři aplikace napsané ve Visual Basicu.

```
private void button1_Click(object sender, System.EventArgs e)
{
     localhost1.MsXmlAnalysis xml = new localhost1.MsXmlAnalysis ();
     string mdxCmd="SELECT{[Measures].[Unit Sales]} ON COLUMNS FROM sales";
     XmlDocument doc = new XmlDocument();
     XmlElement command=doc.CreateElement("Statement");
     command.InnerText=mdxCmd;
     XmlElement properties=doc.CreateElement("PropertyList");
     properties.InnerXml="<DataSourceInfo>Provider=MSOLAP;
```

```
                        Data Source=localhost</DataSourceInfo>
                         <Catalog>Foodmart 2000</Catalog>";
    XmlElement result = xml.Execute(command,properties);
    txt.Text = result.OuterXml;
}
```

Aplikace pro přístup k údajům v analytických databázích typu tlustý klient

Ladicí aplikace pro výpis údajů z krychle na základě dotazu MDX

Pokud vytváříme reálnou aplikaci, věnujeme velkou pozornost návrhu aplikačního rozhraní. Navrhujeme různé formuláře pro zadávání údajů a tabulky nebo textová pole pro jejich výstup. Cíl tohoto příkladu je trochu jiný – vytvořit jednoduchou ladicí aplikaci pro první pokusy s rozhraním pro přístup k multidimenzionálním databázím ADOMD, konkrétně s objekty `ADOMD.Catalog()` a `Cellset()`. Konkrétní realizaci výstupu ve formuláři aplikace a jeho formátování zatím nebudeme věnovat pozornost. Proto se bude formulář aplikace skládat jen z jediného tlačítka, kterým aktivujeme odevzdání dotazu MDX analytickému serveru a následný výpis výsledků.

Obr. 7.19 Jednoduchý formulář ladicí aplikace v jazyce Visual Basic.NET

Při pohledu na navržený formulář se logicky skýtá otázka, kam chceme výsledky vypisovat. Abychom s tím v etapě ladění neměli skutečně žádné starosti, ještě jednou si připomeneme, že jsme v „etapě ladění", a tak budeme výsledky dotazu MDX vypisovat do ladicího okna vývojového prostředí. Pro textový výpis použijeme příkaz

```
System.Diagnostics.Debug.WriteLine(...)
```

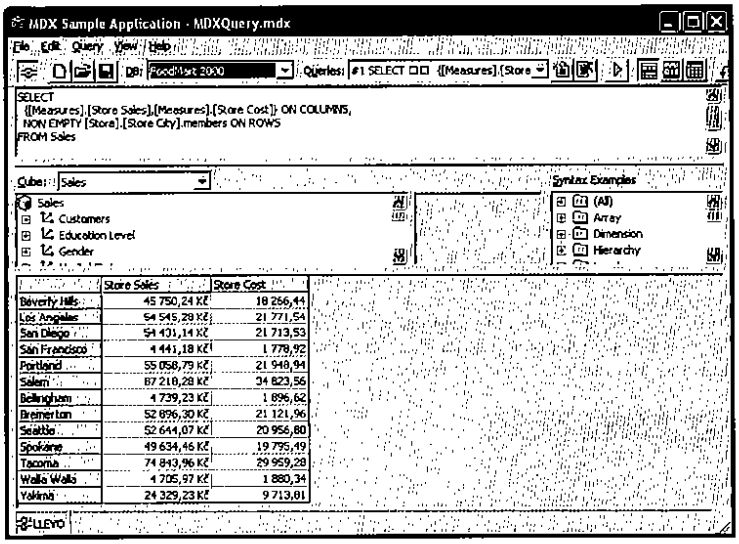
Celý aplikační kód je v obsluze události klepnutí na tlačítko „Výpis do ladicího okna". Nejdříve deklarujeme objekty `ADOMD.Catalog()` a `Cellset()`.

```
  Dim cat As New ADOMD.Catalog()
  Dim cst As New ADOMD.Cellset()
```

Do řetězcové proměnné uložíme dotaz MSX, v našem případě:

```
SELECT
  {[Measures].[Store Sales],[Measures].[Store Cost]} ON COLUMNS,
  NON EMPTY [Store].[Store City].members ON ROWS
FROM Sales
```

Pokud je tato aplikace naším prvním pokusem s ADOMD, je výhodné vyzkoušet si syntaktickou správnost dotazu MDX v MDX Sample Application, která se nainstaluje při instalaci analytických služeb SQL Serveru 2000. Potom při ladění aplikace zažijeme mnohem méně „překvapení".



Obr. 7.20 Otestování syntaktické správnosti dotazu MDX v MDX Sample Application

V našem případě je vše v pořádku, takže pokud se v ladicí aplikaci vyskytnou problémy, nebude to vinou špatné syntaxe příkazu MDX.

V dalším kroku aktivujeme objekty `ADOMD.Catalog()` a `Cellset()`.

```
cat.let_ActiveConnection("Data Source=LOCALHOST;Provider=msolap; initial
catalog=FoodMart 2000")
cst.let_Source(strSrc)
cst.ActiveConnection = cat.ActiveConnection
cst.Open()
```

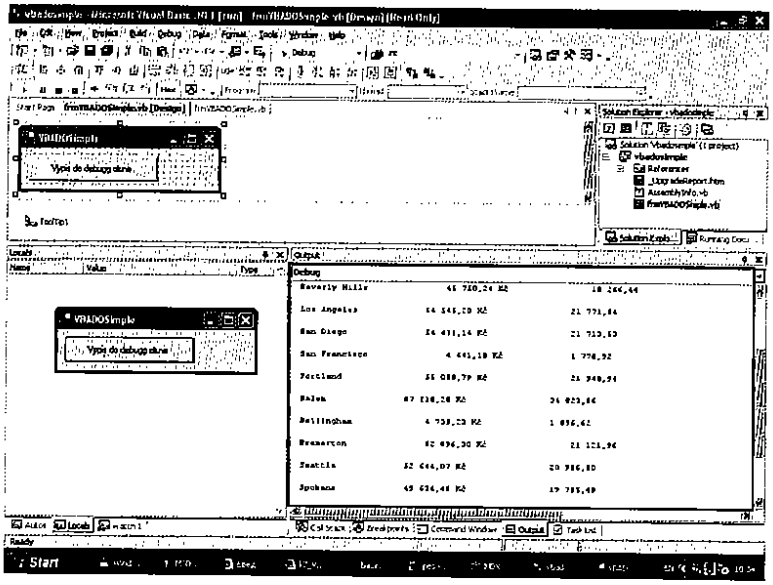
Po úspěšném otevření připojení k serveru OLAP vypíšeme výsledky do ladicího okna. Tuto etapu můžeme rozdělit do dvou logických kroků. Nejdříve vypíšeme záhlaví jednotlivých sloupců.

```
For i = 0 To cst.Axes(0).Positions.Count - 1
    strHead = strHead & cst.Axes(0).Positions(i).Members(0).Caption & vbTab & vbTab
  Next
```

A následně všechny řádky.

```
strLine = ""
For j = 0 To cst.Axes(1).Positions.Count - 1
  strLine = strLine & cst.Axes(1).Positions(j).Members(0).Caption & vbTab & vbTab
  For k = 0 To cst.Axes(0).Positions.Count - 1
      strLine = strLine & cst.Item(k, j).FormattedValue & vbTab & vbTab
  Next
  System.Diagnostics.Debug.WriteLine(strLine & vbCrLf)
  strLine = ""
Next
```

Po spuštění aplikace a stisku tlačítka „Výpis do ladicího okna" si můžeme prohlédnout buď chybové hlášení vývojového prostředí, pokud jsme se dopustili nějaké chyby, nebo pokud je vše v pořádku, budou v ladicím okně vývojového prostředí (v našem případě Visual Studio.NET) zobrazeny výsledky.



Obr. 7.21 Výpis výsledků do ladicího okna

Samozřejmě není problém výsledky v ladicím okně naformátovat, ale v našem případě jsme preferovali jednoduchost zdrojového kódu, formátování textů až nakonec. Výsledky můžeme z ladicího okna zkopírovat a v textovém editoru upravit.

```
                  Store Sales   Store Cost
Beverly Hills      45 750,24 Kč   18 266,44
Los Angeles        54 545,28 Kč   21 771,54
San Diego          54 431,14 Kč   21 713,53
San Francisco       4 441,18 Kč    1 778,92
Portland           55 058,79 Kč   21 948,94
Salem              87 218,28 Kč   34 823,56
Bellingham          4 739,23 Kč    1 896,62
Bremerton          52 896,30 Kč   21 121,96
Seattle            52 644,07 Kč   20 956,80
Spokane            49 634,46 Kč   19 795,49
Tacoma             74 843,96 Kč   29 959,28
Walla Walla         4 705,97 Kč    1 880,34
Yakima             24 329,23 Kč    9 713,81
```

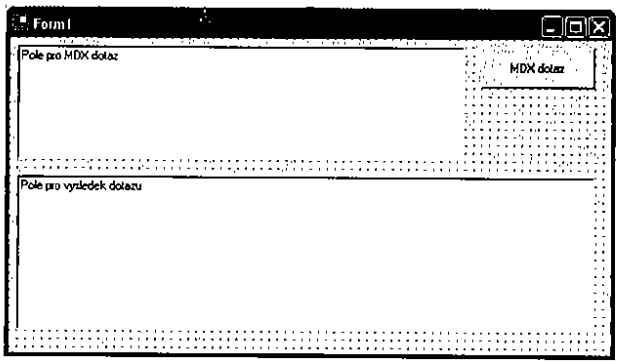
Kompletní výpis kódu obsluhy události klepnutí na tlačítko „Výpis do ladicího okna" :

```
Private Sub cmdVypis_Click(ByVal eventSender As System.Object, ByVal eventArgs As
System.EventArgs) Handles cmdVypis.Click
  Dim k As Object

 'Deklarace ADOMD objektu
  Dim cat As New ADOMD.Catalog()
  Dim cst As New ADOMD.Cellset()

  Dim i As Short
  Dim j As Short
  Dim strSrc As String
  Dim strHead As String
  Dim strLine As String

  'Dotaz MDX
  strSrc = strSrc & "SELECT "
  strSrc = strSrc &"{[Measures].[Store Sales],[Measures].[Store Cost]} ON COLUMNS,"
  strSrc = strSrc & "NON EMPTY [Store].[Store City].members ON ROWS"
  strSrc = strSrc & " FROM Sales"

  'Parametry ADOMD pripojeni
   cat.let_ActiveConnection("Data Source=LOCALHOST;Provider=msolap; initial
catalog=FoodMart 2000")
   cst.let_Source(strSrc)
   cst.ActiveConnection = cat.ActiveConnection
   cst.Open()

 'Zahlavi vypisu
  strHead = vbTab & vbTab & vbTab & vbTab
  For i = 0 To cst.Axes(0).Positions.Count - 1
    strHead = strHead & cst.Axes(0).Positions(i).Members(0).Caption & vbTab & vbTab
  Next
```

```
'Radky vypisu
 strLine = ""
 For j = 0 To cst.Axes(1).Positions.Count - 1
   strLine = strLine & cst.Axes(1).Positions(j).Members(0).Caption & vbTab & vbTab
   For k = 0 To cst.Axes(0).Positions.Count - 1
       strLine = strLine & cst.Item(k, j).FormattedValue & vbTab & vbTab
   Next
   System.Diagnostics.Debug.WriteLine(strLine & vbCrLf)
   strLine = ""
  Next

End Sub
```

Konzolová aplikace MDX

Asi nejuniverzálnějším příkladem pro přístup k multidimenzionálním databázím bude jednoduchá konzolová aplikace. Aplikace je napsaná v programovacím jazyce Visual Basic (vývojové prostředí Visual Studio.NET). Aplikační formulář je jednoduchý. Skládá se ze dvou textových polí, kde musíme povolit vlastnost Multiline = TRUE.

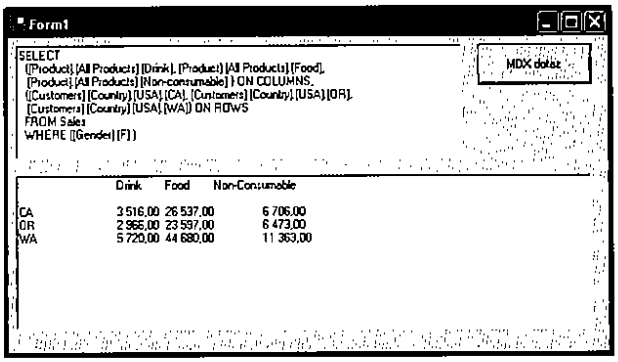


Obr. 7.22 Formulář konzolové aplikace MDX v jazyce Visual Basic.NET

Po zadání dotazu MDX do horního zadávacího okna aplikace, například:

```
SELECT
  {[Product].[All Products].[Drink], [Product].[All Products].[Food],
   [Product].[All Products].[Non-consumable] } ON COLUMNS,
  {[Customers].[Country].[USA].[CA], [Customers].[Country].[USA].[OR],
   [Customers].[Country].[USA].[WA]} ON ROWS
  FROM Sales
  WHERE ([Gender].[F] )
```

a aktivování tlačítka „Dotaz MDX" se ve spodním okně zobrazí výsledek



Obr. 7.23 Výsledek dotazu MDX

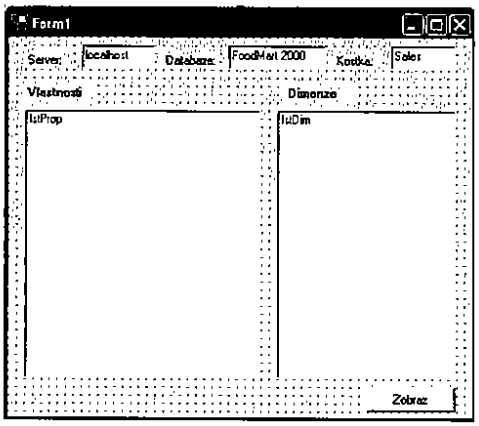
Kompletní výpis těla procedury pro obsluhu události klepnutí na tlačítko „Dotaz MDX".

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

  'Deklarace a inicializace objektu ADOMD
    Dim Cst As New ADOMD.Cellset()
    Dim Cnn As New ADODB.Connection()
    Dim i, j, k As Integer

    Cnn.ConnectionString = "Provider=MSOLAP;Data Source=LOCALHOST;Initial Catalog=FoodMart 2000"
    Cnn.Open()
    Cst.ActiveConnection = Cnn

  'Ve vysledku odsuneme radek se zahlavim
    TextBox2.Text = vbTab & vbTab

  'Dotaz MDX ze zadavaciho formulare
    Cst.let_Source(TextBox1.Text)

  'Osetreni chyby pokud se CellSet nepodari otevrit
    On Error GoTo Err
       Cst.Open()

   'Vypis radku zahlavi sloupcu
       For i = 0 To Cst.Axes(0).Positions.Count - 1
         TextBox2.Text = TextBox2.Text & Cst.Axes(0).Positions(i).Members(0).Caption & vbTab
       Next i

       TextBox2.Text = TextBox2.Text & vbCrLf & vbCrLf
```

```
        For j = 0 To Cst.Axes(1).Positions.Count - 1
            TextBox2.Text = TextBox2.Text & Cst.Axes(1).Positions(j).Members(0).Caption & vbTab
            For k = 0 To Cst.Axes(0).Positions.Count - 1
                TextBox2.Text = TextBox2.Text & vbTab & Cst(k, j).FormattedValue
            Next k
            TextBox2.Text = TextBox2.Text & vbCrLf
        Next j
        Exit Sub
Err:
        MsgBox("Nespravny dotaz MDX")

    End Sub
```

Aplikace pro výpis vlastností a dimenzí krychle

Pro výpis vlastností a dimenzí krychle použijeme objekt CubeDef. Aplikace je vytvořena v programovacím jazyce Visual Basic ve vývojovém prostředí Visual Studio.NET. Samozřejmě je možno použít i starší verzi Visual Basic 6. Abychom mohli nastavovat parametry a zobrazovat údaje, v tomto případě vlastnosti a dimenze vybrané krychle, vytvoříme jednoduchý uživatelský formulář (Form1). V jeho horní části jsou pole pro zadání názvu serveru, názvu databáze a názvu krychle. Ve spodní části jsou dvě pole typu ListView pro výpis vlastností a dimenzí.



Obr. 7.24 Návrh formuláře aplikace

Činnost aplikace spočívá v připojení se k analytickému serveru a získání požadovaných údajů.

Nejdříve vytvoříme připojovací řetězec na základě údajů z polí SERVER a DATABAZE ze zadávacího formuláře.

```
sConnect = "Data Source=" & Me.sServer.Text & ";" & "Initial Catalog="
sConnect = sConnect + Me.sDatabaze.Text & ";" & "Provider=MSOLAP;"
```

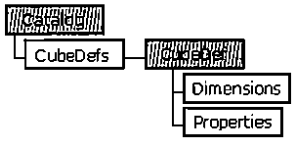
V našem případě byl připojovací řetězec

```
"Data Source=LOCALHOST;Initial Catalog= FoodMart 2000;Provider=MSOLAP;"
```

V následujícím kroku deklarujeme a vytvoříme objekt ADOMD.Catalog a nastavíme jako parametr připojovací řetězec.

```
Dim adomdCat As ADOMD.Catalog
...
adomdCat.let_ActiveConnection(sConnect)
```

Potom deklarujeme a vytvoříme objekt ADOMD.CubeDef.



Obr. 7.25 Objekt CubeDef

Jako parametr metody CubeDefs zadáme název krychle.

```
Dim adomdCD As ADOMD.CubeDef
adomdCD = adomdCat.CubeDefs(Me.sKrychle.Text)
```

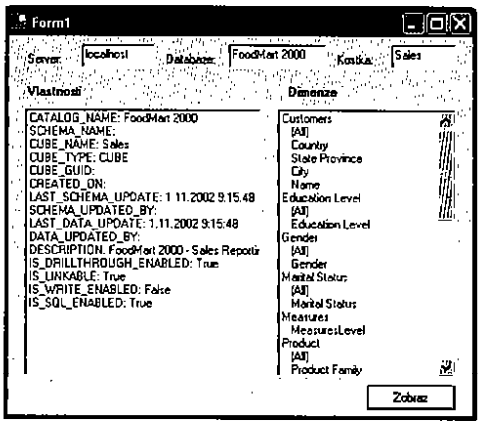
Naším úkolem je vypsat vlastnosti a dimenze. Názvy a hodnoty jednotlivých vlastností zjistíme pomocí metod `adomdCD.Properties(nVl).Name`, a `adomdCD.Properties(nVl).Value`

Proměnná nVL je index, počet vlastností zjistíme pomocí metody `adomdCD.Properties.Count`. Podobně postupujeme při výpisu jednotlivých dimenzí, kde zjišťujeme název dimenze pomocí metody `adomdCD.Dimensions(nDim).Name` a její hierarchické postavení pomocí metody `adomdCD.Dimensions(nDim).Hierarchies.Count`.

Kompletní výpis kódu aplikace potom bude:

```
Friend Class Form1
        Inherits System.Windows.Forms.Form

Private Sub Command1_Click(ByVal eventSender As System.Object, ByVal eventArgs As
System.EventArgs) Handles Command1.Click

  'Deklarujeme objekty ADOMD.Catalog a ADOMD.CubeDef
    Dim adomdCat As ADOMD.Catalog
    Dim adomdCD As ADOMD.CubeDef

    Dim nVl As Short
    Dim nDim As Short
```

```
    Dim nHier As Short
    Dim nLev As Short
    Dim sConnect As String
    Dim sRadek As String

'Vytvorime objekt ADOMD.Catalog
      adomdCat = New ADOMD.Catalog

'Podle zadanych udaju SERVER, DATABAZE sestavime pripojovaci retezec
    sConnect = "Data Source=" & Me.sServer.Text & ";" & "Initial Catalog="
    sConnect = sConnect + Me.sDatabaze.Text & ";" & "Provider=MSOLAP;"
    adomdCat.let_ActiveConnection(sConnect)

 'Podle parametru KRYCHLE vybereme krychli
     adomdCD = adomdCat.CubeDefs(Me.sKrychle.Text)

  'Vypis vlastnosti (Name, Value)
      For nVl = 0 To adomdCD.Properties.Count - 1
          sRadek = adomdCD.Properties(nVl).Name & ": "
          sRadek = sRadek & adomdCD.Properties(nVl).Value
          Me.lstProp.Items.Add(sRadek)
      Next nVl

   'Vypis dimenzi
    For nDim = 0 To adomdCD.Dimensions.Count - 1
      Me.lstDim.Items.Add(adomdCD.Dimensions(nDim).Name)
      For nHier = 0 To adomdCD.Dimensions(nDim).Hierarchies.Count - 1
        For nLev=0 To adomdCD.Dimensions(nDim).Hierarchies(nHier).Levels.Count- 1
```



Obr. 7.26 Výpis údajů

```
            Me.lstDim.Items.Add("    " &
adomdCD.Dimensions(nDim).Hierarchies(nHier).Levels(nLev).Name)
          Next nLev
        Next nHier
      Next nDim

  'Odstraníme objekty
     adomdCat = Nothing
     adomdCD = Nothing
   End Sub
End Class
```

Kapitola 8

Data mining

Tento termín můžeme volně přeložit jako způsob získávání, dolování, odkrývání dat (analogie těžby), informací pro podporu rozhodování z existujících datových zdrojů. I když jde jen o slovní hříčku, můžeme tento proces přirovnat např. k těžbě drahých kovů.

Postup je principiálně analogický. V úvodní fázi obvykle na základě určitých indicií a nekomplexních poznatků jen předpokládáme, že se v daném teritoriu nějaký drahý kov nachází. Vyslovenou hypotézu musíme na vybraném vzorku průzkumem ověřit. Pak je na geolozích, aby takto vyslovenou hypotézu na základě průzkumů zamítli, nebo nezamítli. Ze statistických definic z oblasti testování hypotéz totiž vyplývá, že hypotéza se může buď zamítnout, nebo nezamítnout. Hypotéza, jak uvidíme v našem stručném statistickém minimu, se totiž testováním nedá potvrdit. V případě, že se naše domněnka potvrdí, spustíme těžbu. Ale tím jsme ještě drahý kov nezískali. Vytěženou horninu musíme nejdříve zbavit nepotřebné hlušiny a její zbytek potom musí projít dalším procesem zhodnocování. Výsledkem tohoto procesu by měl být měřitelný ekonomický efekt.

Data mining nám všeobecně může pomoct při identifikaci problémů a identifikaci existujících nebo pravděpodobných vzájemných vztahů mezi jednotlivými entitami. Na základě těchto faktů můžeme uvést stručnou a jednoduchou definici data miningu.

Data mining je proces analýzy dat z různých perspektiv a jejich přeměna na užitečné informace. Z matematického a statistického hlediska jde o hledání korelací, tedy vzájemných vztahů nebo vzorů v údajích.

Data mining je principiálně založený na heuristických algoritmech, neuronových sítích a jiných pokročilých softwarových technologiích a metodách umělé inteligence. Pomáhá sledovat a analyzovat trendy a předvídat události. Jako příklad můžeme uvést operátory mobilních sítí. Všichni tito operátoři získávají velké množství údajů z provozu svých sítí. Pokud by chtěli, mohou monitorovat a analyzovat nejen informace o telefonních hovorech, ale také o geografické poloze každého účastníka. Časem by se jim určitě podařilo zjistit různé souvislosti a závislosti a vypracovat jakýsi model chování každého účastníka mobilní sítě.

V ekonomické oblasti se data mining používá například při analýze a predikci úvěrového rizika, predikci rizika při vydávání kreditních karet a podobně. Jako příklad můžeme uvést společnosti, které vydávají kreditní karty. Na základě transakcí, nákupů a jiných operací, které vykonává majitel kreditní karty, je možno získat velké množství údajů o chování klienta a i o jeho pohybech v geografických dimenzích. I v tomto případě by se postupem času zřejmě podařilo zjistit různé souvislosti a závislosti a vypracovat jakýsi model chování každého majitele kreditní karty.

Nepopulární důsledky zneužití takových modelů si dokáže každý domyslet sám (směrování marketingových reklamních akcií, tajná služba, soukromý detektiv najatý žárlivou manželkou, konkurenční firma...). Málokdo si uvědomí, že případný model chování majitele kreditní karty může být v mnoha případech užitečný i pro samotného majitele. Představme si situaci, kdy mu bude karta ukradena a někdo se naučí napodobit jeho podpis. V případě mnoha karet to není žádný problém, podpis majitele je přímo na kartě. Provozovatel kreditní karty na základě modelu dosavadního chování klienta může zjistit, že se klient najednou chová „nějak divně", a pokud usoudí, že je toto chování s nejvyšší pravděpodobností způsobené odcizením kreditní karty, může kartu sám zablokovat a klientovi zachránit nemalou sumu. Algoritmus analýzy a vyhodnocení však musí být velmi přesný, protože se majitel kreditní karty může začít chovat „nějak divně" i z jiných důvodů, například se ožení.

Co data mining neumožňuje

Po přečtení úvodu by se zdálo, že by data mining mohl být univerzální a téměř „všemocnou" metodou. Samozřejmě tomu tak není. Jednak to není proces jednoduchý, a dokonce by se dalo říci, že je někde na rozhraní vědy a magie, což znamená, že někdy můžeme na základě víceméně náhodně vybraných vstupů získat cenné informace, tedy přijdeme k nim, jak se lidově říká, „jako slepý k houslím", jindy může být výsledek data miningu triviální, v praxi nevyužitelný, ba dokonce žádný. Nemá totiž smysl dokonce ani při procvičování data miningu aplikovat data miningový model na databázi naplněnou náhodnými údaji. Kde žádné údaje nejsou, nemůžeme je samozřejmě ani vydolovat. I při reálných údajích musí být tyto údaje kvalitní, tedy kvalitně připravené. Ve většině případů bude potřeba upravit hlavně vstupy, například jejich seskupením do intervalu, případně snížit počet diskrétních hodnot ({1,2,3,4} -> {ano, ne}) ...

Může nastat i jiný vážný problém. Pokud v našich vstupních údajích nemáme podchycený některý důležitý údaj, může být výsledek nesprávný. Jako příklad můžeme uvést zkoumání denního prodeje piva bez zahrnutí vnější teploty a podobně. Samozřejmě musíme mít jasno i v „místě" data miningu v konkrétním informačním systému, případně v podsystému, který slouží pro podporu procesu rozhodování. Data mining musíme chápat jen jako prostředek k získávání informací pro podporu rozhodování. Samotné rozhodování musí provést příslušný odpovědný pracovník. Úlohu data miningu tedy chápeme jako prostředek k poskytnutí kvalitních vstupů do procesu rozhodování. Nezastupitelnou úlohou managementu je pak vypracovávání a definování vizí, koncepcí a cílů a také i vyslovování hypotéz. Data mining pak může posloužit jako nástroj pro ověření jednotlivých hypotéz, tedy zda je management nakonec zamítne, nebo nezamítne.

Z uvedeného vyplývá, že se data mining prakticky nedá automatizovat, protože musíme vytvářet příslušné modely, přizpůsobovat je konkrétním podmínkám, přizpůsobovat je konkrétním požadavkům managementu a podobně. Proto sice nemůžeme definovat konkrétní postupy, ale je možno vytvořit určitou metodiku.

Teoretický úvod

Data mining je z hlediska matematické a statistické teorie založený hlavně na hledání korelací a na testovaní hypotéz.

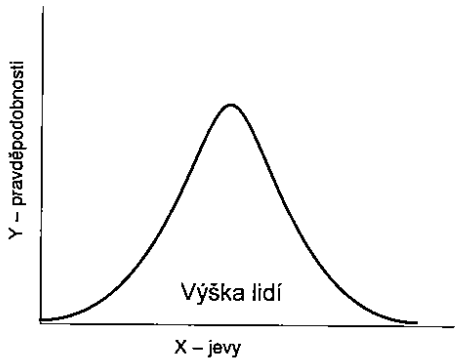
Stručné statistické minimum

Poznámka autora: Tuto stať mohou sice čtenáři bez matematického talentu a statistických znalostí bez nějakých větších problémů přeskočit, ale byla by to škoda. Budu se snažit, aby bylo i toto exaktní téma popsáno co možná nejsrozumitelněji, a slibuji, že tu nenajdou žádný vzorec.

Rozdělení pravděpodobnosti a testování hypotéz

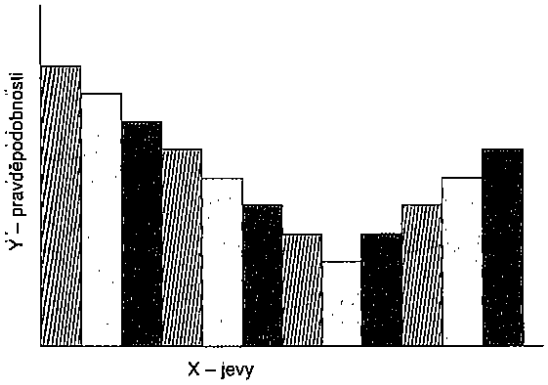
Nejdříve nadefinujeme pojem „rozdělení pravděpodobnosti". Je to jednoduchá funkce, jejíž graf má na ose x jednotlivé jevy a na ose y pravděpodobnosti výskytů těchto jevů. Tato funkce může být spojitá, nebo diskrétní. Jako jednoduchý příklad spojité funkce můžeme uvést rozdělení pravděpodobnosti výšky mužské populace. Pro každou hodnotu výšky, a protože je to funkce spojitá, může to být i desetinné číslo, dokážeme určit pravděpodobnost jejího výskytu v mužské populaci.

Vidíme, že tato závislost má určitý tvar a rozdělení, které má křivku takového tvaru, říkáme Gaussovo normální rozdělení. Toto rozdělení je aplikovatelné na mnoho jevů v přírodě. V našem případě, někde uprostřed, kde je vrchol křivky, je nejčastěji se vyskytující výška mužů okolo 175 cm. Na okrajích křivky vidíme nízkou pravděpodobnost výskytu lidí extrémně nízkých (vlevo) a extrémně vysokých.



Obr. 8.1 Rozdělení pravděpodobnosti pro spojité hodnoty

Pro vyjádření rozdělení pravděpodobnosti mnoha diskrétních jevů, například házení kostkou, ruletu a podobně, používáme obvykle sloupcový diagram.



Obr. 8.2 Příklad rozdělení pravděpodobnosti pro diskrétní hodnoty

U hypotézy se v technickém i statistickém smyslu soustřeďujeme na charakteristický způsob rozdělení pravděpodobnosti určitého jevu. Potřebujeme vlastně určit nějaký speciální průběh tohoto rozdělení. Přitom nemusí jít o nějaký nový způsob, o kterém se domníváme, že je správný. Někdy to může být i způsob, který pokládáme za nesprávný, ale máme v úmyslu najít právě pomocí tohoto testu něco, co by nám umožnilo navržený způsob rozdělení pravděpodobnosti zamítnout. Při statistickém testování hypotéz musíme vždy mít předem nějaké poznatky, nikdy nezačínáme z ničeho. Volba ověřované hypotézy proto samozřejmě nemůže být zcela náhodná. Ověřovaná hypotéza (někdy ji označujeme také jako nulovou hypotézu) musí patřit do skupiny takzvaných přípustných hypotéz, které jsou alespoň částečně specifikované. To znamená, že rozdělení pravděpodobnosti obsahuje alespoň jeden parametr, který není přesně stanovený.

Z matematického hlediska bychom ověřovanou hypotézu mohli stanovit například takto: Máme přesně definované rozdělení pravděpodobnosti, které však obsahuje některé neurčené parametry. Testovanou hypotézu budeme často testovat proti jiné, alternativní hypotéze z množiny přípustných hypotéz. Testování hypotézy tedy znamená, že se na základě statistických pozorování pokusíme určit, zda testovanou hypotézu zamítneme nebo nezamítneme. Na tomto místě je potřeba vysvětlit rozdíl mezi pojmy „nezamítnuté" a „potvrzené". To, že hypotézu při jejím ověřování nezamítneme, ještě neznamená, že ji potvrdíme, tedy uznáme za správnou. Nejsnáze to pochopíme na hypotéze: „Ve vesmíru žijí i jiné civilizace." Tuto hypotézu vědci nezamítli, ale to ještě neznamená, že je pravdivá. Až po objevení se prvního mimozemšťana budeme moci tuto hypotézu potvrdit. A aby to nebylo tak jednoduché, při testování hypotéz se můžeme dopustit dvou druhů chyb: chyby zamítnutí a chyby nezamítnutí.

Chyby zamítnutí se dopustíme tehdy, když hypotézu zamítneme, i když tato hypotéza zamítnutá být neměla. **Chyby nezamítnutí** se dopustíme tehdy, když testovací hypotézu nezamítneme v případě, že bychom ji měli zamítnout. Navzdory tomu, že se v tomto odstav-

ci neobjevily žádné sumy a ani integrály, vidíme že testování hypotéz vůbec není jednoduchá záležitost.

Z dosud uvedeného vyplývá, že stojíme před dvěma problémy. Jednak bychom chtěli provést ověření tak, aby vznikla jen nepatrná možnost chybného zamítnutí, ale chtěli bychom zmenšit i možnost chyby nezamítnutí. A protože jsme jen lidé, kromě dosud vyjmenovaných exaktních, objektivních kritérií existují i kritéria subjektivní. Pokud totiž někdo vysloví nějakou hypotézu, bude se podvědomě snažit shromažďovat fakta, která podporují její nezamítnutí, a naopak bude kriticky zkoumat všechna fakta vedoucí k zamítnutí jím vyslovené hypotézy.

Statistické metody využívané data miningovými modely

Data miningové modely využívají některé statistické metody, hlavně korelaci, lineární a logistickou regresi, diskriminantní analýzu a metody předpovídání. Složitější postupy jsou někdy realizovány pomocí neuronových sítí a genetických algoritmů.

Korelace je míra závislosti mezi dvěma proměnnými. Někdy je tato korelace zřejmá a jednoduše vysvětlitelná, například společné nákupy mouky a cukru jsou motivované nákupem uvedených komodit ve velkém balení tehdy, když máme k dispozici auto, případně se chystáme péct na svatbu... Vždy to tak jasné není. Například se zjistilo, že se v supermarketech často nakupují společně dětské plenky a pivo. Ne, v tomto případě je spotřebitel plenek, tedy batole, mimo podezření. Tuto korelaci mají na svědomí nakupující. Velká balení plenek obvykle nakupují otcové, protože se maminka věnuje potomkovi a po identifikování tohoto faktu už nákup piva společně s plenkami není žádnou záhadou. Například vysoká korelace mezi nákupy určitých produktů jako sýru a keksů prozrazuje, že je zvykem tyto produkty nakupovat společně.

Korelace může být pozitivní a negativní. Pozitivní korelace udává, že vysoká úroveň jedné proměnné bude provázena vysokou úrovní korelační proměnné. Naopak negativní korelace udává, že vysoká úroveň jedné proměnné bude provázena nízkou úrovní korelační proměnné. Zjištění pozitivní korelace je důležité například při rozmisťování zboží v supermarketu, případně při marketingových rozhodnutích, které zboží je možno prodávat společně.

Ne vždy je samozřejmě prodej dvou druhů zboží způsoben pouze korelací jejich současného prodeje. Například když se před Vánoci společně s balením tří videokazet prodává i miniaturní diář. Tady není přímý korelační vztah mezi prodejem videokazet a minidiářů, ale komplikovanější, ale víceméně snadno pochopitelná časová souvislost. Důležité je v tomto případě slovní spojení „před Vánoci". Vánoci je vysvětlená motivace k nákupu trojitého balení videokazet, a protože po Vánocích následuje těsně nový rok, vstupuje do hry poptávka po mini diářích.

Možná se to na první pohled nezdá, ale význam má i negativní korelace, například při výměně skladby firemní produkce. Můžeme to demonstrovat například zájmem výrobců klasických fotoaparátů o fotoaparáty digitální nebo zájmem automobilových koncernů o automobily s alternativním pohonem, například elektromobily. Pokud z různých důvodů, například ekologických nebo z důvodu vyčerpání zásob ropy a tím zvýšení její ceny poklesne poptávka po klasických automobilech, logicky stoupne poptávka po automobilech využívajících alternativní zdroje energie. Proto znalost a využívání negativní korelace může pomoci stanovit produktovou strategii tak, aby případné změny trhu výrazně neohrozily ekonomiku firmy.

Lineární regrese je statistická metoda, která kvantifikuje závislost mezi dvěma spojitými proměnnými: závislou proměnnou, kterou se snažíme predikovat, a nezávislou, tedy prediktivní proměnnou. V průběhu lineární regrese se snažíme najít přímku procházející mezi jednotlivými body, pro kterou platí, že součet druhých mocnin odchylek od každého bodu je minimální. Pokud je vztah mezi závislou a nezávislou proměnnou nelineární, zpravidla je potřeba transformovat prediktivní proměnnou tak, aby umožnila najít lepší proložení.

Logistická regrese je velmi podobná lineární regresi s tím rozdílem, že závislá proměnná není spojitá, ale diskrétní. Proto ji transformujeme na spojitou hodnotu, která je funkcí pravděpodobnosti výskytu události. Tuto regresi je možno použít k predikování výsledků dvou nebo více úrovní, například proč nastane jev nesplácení půjček a podobně.

Diskriminantní analýza měří důležitost faktorů určujících zařazení do kategorie. Například zkoumáme vztah mezi platiči a neplatiči nájemného. Pokud dokážeme identifikovat správné faktory, měl by být náš model, vytvořený na základě takové analýzy, schopný využít tyto faktory na rozlišení (diskriminování) mezi těmi nájemníky, kteří platí, a těmi, kteří neplatí.

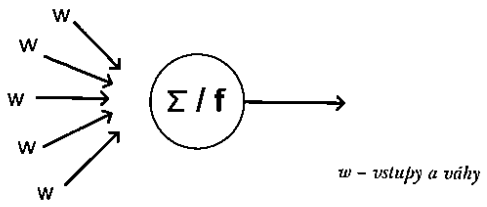
Předpovědi trendů pro nějakou proměnnou, například obrat, jsou založené na analýze údajů z minulosti, na základě kterých se definují určitá pravidla, pomocí kterých předpovídáme budoucí trend dané proměnné. Můžeme použít například regresi časových úseků nebo neuronové sítě. Předpovědi trendů zahrnují i krátkodobé cyklické fluktuace, například pokud predikujeme trend vývoje nezaměstnanosti na příští rok na základě údajů z minulých let, určitě se v dobré předpovědi objeví i logické fluktuace, například nárůst nezaměstnanosti po ukončení školní docházky, její pokles při sezónních pracích a podobně.

Neuronové sítě

Neuronové sítě jsou modely, které určitým způsobem simulují strukturu lidského mozku. Jak všichni dobře víme, lidský mozek se od dětství postupně učí, získává poznatky z množiny podnětů z okolního světa. Podobně i neuronové sítě získávají poznatky z množiny vstupů a na základě nich doladují své parametry modelu vzhledem k novým znalostem.

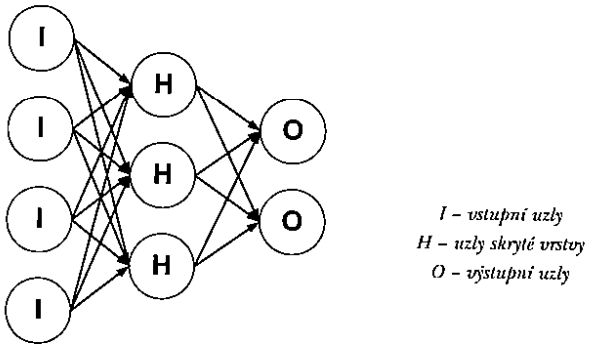
Zpracování pomocí neuronových sítí tedy nevychází z žádného statistického rozdělení, ale pracuje podobně jako lidský mozek na principu rozpoznávání vzorů a minimalizace chyb. Tento proces si můžeme představit jako přijímání informací a učení se z každé zkušenosti tak, aby byly nalezené v údajích určitého schématu. Neuronovou síť tvoří uzly uspořádané do vrstev. Dříve, než začne vlastní proces, se údaje rozdělí do tréninkové a testovací množiny. V průběhu každé iterace jsou vstupy zpracovávány systémem a jsou porovnávány se skutečnou hodnotou. Změří se chyba a odevzdá se ke zpracování systému, aby upravil původní váhy. Proces končí zpravidla v okamžiku dosažení předem dané minimální chyby.

Základní logickou jednotkou neuronové sítě je neuron, tedy buňka, do které vstupují vstupní údaje, ať už z vnějšího vstupu sítě nebo z výstupu jiného neuronu. Neuron údaje částečně zpracuje pomocí sumarizace a aktivačních funkcí a údaje postupují do dalšího neuronu nebo na výstup neuronové sítě.



Obr. 8.3 Schéma neuronu

Na schématu jednoduché neuronové sítě už na první pohled vidíme velkou výhodu této architektury. Údaje jsou totiž v jednotlivých vrstvách zpracovávány paralelně mnoha neurony.



Obr. 8.4 Schéma neuronové sítě s jednou skrytou vrstvou

Genetické algoritmy předpokládají vliv evolučního procesu, kdy se porovnává více modelů, které se v jednotlivých krocích upravují křížením, mutací a klonováním, náhodnými výměnami hodnot, znamének a funkcí. Tato metoda je velmi náročná na výpočetní kapacitu.

Model procesu data miningu

Jako každý projekt informačního systému i proces data miningu můžeme analyzovat a modelovat.

Konceptuální model

Konceptuální model představuje komplexní, ve většině případů uživatelský pohled na předmětnou oblast. V průběhu jednotlivých cyklů procesu, ve většině případů ekonomického, sesbíráme obvykle velké množství informací. Každá obchodní transakce, objednávka a podobně je přece podchycená ve firemním informačním systému. Momentálně nás

nebude zajímat, v jakých databázových platformách a v jakých formátech se předmětné údaje nachází. Jak jsme už uvedli v úvodu této publikace, s nárůstem kvantity údajů zpravidla nepřichází automaticky kvalita, a dokonce by se dalo říct, že zde existuje nepřímo úměrná závislost mezi kvantitou údajů a kvalitou informace. Takže kvalitu, v tomto případě informace, musíme z údajů v procesu data miningu pracně „vydolovat".

Logický model

Logický model je implementačně nezávislý pohled jednak na údaje v systému, jednak na požadované výsledky. Vychází jednosměrně z konceptuálního modelu, to znamená, že při změně logického modelu se konceptuální model nemění. Při vytváření logického modelu také nemusíme brát v úvahu konkrétní počítačovou reprezentaci. Z teorie vyplývá, že logický model na rozdíl od fyzického modelu může být minimální, to znamená, že nemusí obsahovat žádnou redundanci, která je ve fyzickém modelu někdy nutná z implementačního hlediska.

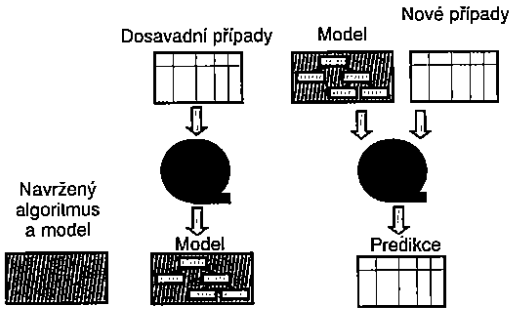
Fyzický model

Fyzický model představuje konkrétní realizaci daného projektu v počítači.

Procesní schéma data miningu

Typický příklad procesního schématu data miningu je zobrazený na obrázku. Celý postup je možné rozčlenit do tří etap.

- výběr algoritmu a modelu,
- fáze učení aplikovaná na dosud existující případy,
- analýza a predikce nových případů.



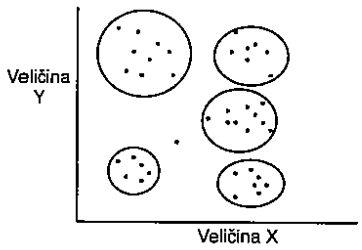
Obr. 8.5 Procesní schéma data miningu

Výběr algoritmu a modelu

Základem dobrých výsledků analýz je výběr vhodných algoritmů. Uvedeme přehled algoritmů, které se nejčastěji používají pro data mining a jsou tedy implementovány v data miningových nástrojích.

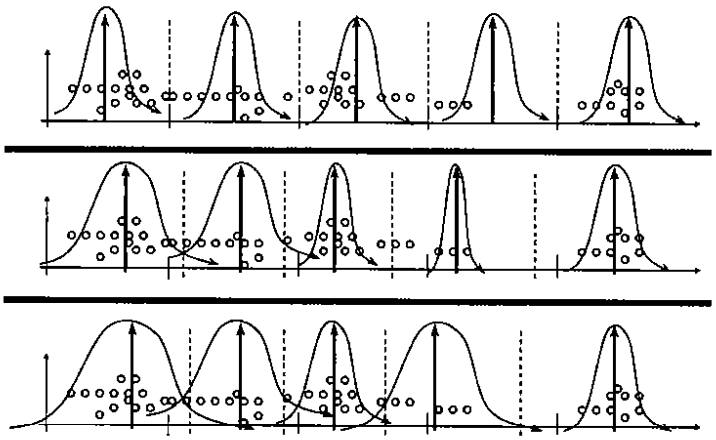
Vícerozměrné shlukové diagramy

Při analýze údajů na základě shluků seskupujeme údaje podle podobných charakteristik. Používá se například pro identifikaci zákaznických segmentů, které jsou založené na společných charakteristikách, například demografických, sociálních, profesních a podobně. Tento algoritmus se používá i pro odhalování shluků dat v multidimenzionálních prostorech. Pomocí něj můžeme rozdělit množiny případů na co nejohraničenější skupiny, takzvané „ostrovy podobnosti".



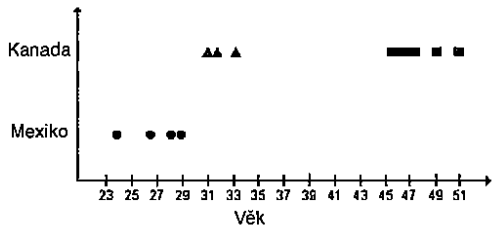
Obr. 8.6 Odhalování shluků

Shluky jsou odhalovány na základě aplikování a analýzy křivek pravděpodobnosti, přičemž se zkoumá, zda jednotlivé údaje nebo skupiny údajů nesplňují podmínku statistického rozložení, například Gaussova normálního rozložení a podobně.



Obr. 8.7 Princip vyhledávání segmentů

Jako příklad je možno uvést výsledky analýzy, do kterých sousedních zemí nejčastěji cestují lidé z USA v určitých věkových kategoriích. Výsledkem analýzy je grafické znázornění.

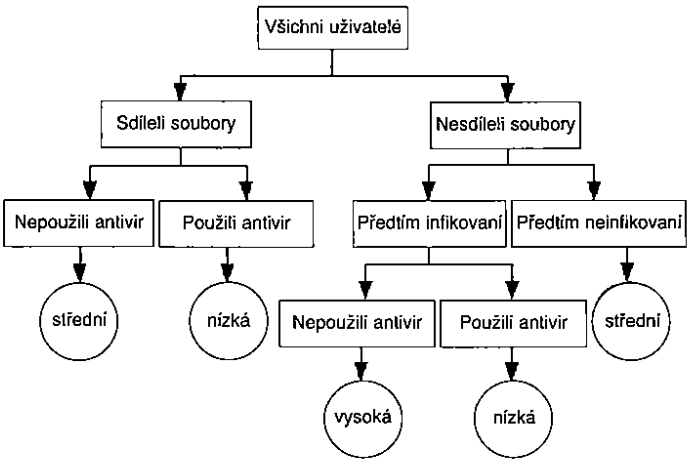


Obr. 8.8 Příklad výsledků shlukového algoritmu

Vidíme, že do Mexika cestují nejvíc lidé ve věku od 23 do 29 let a do Kanady ve dvou věkových skupinách 31-34 a 45-51 let.

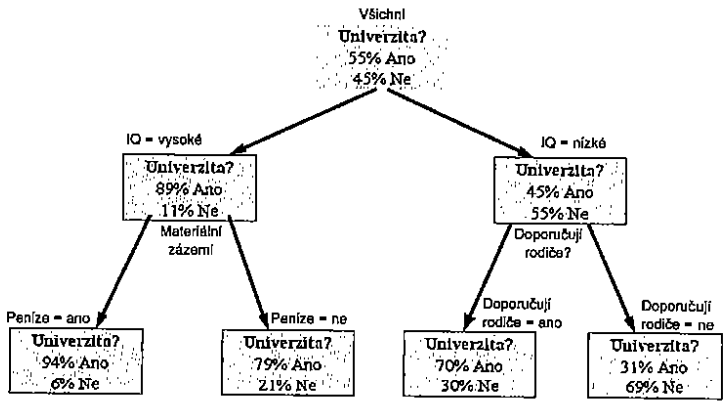
Nevyvážené rozhodovací stromy

Tento typ algoritmů odhaluje závislosti a vyhledává specifické vlastnosti, které potom poslouží pro predikci. Jako jednoduchý příklad takového nevyváženého rozpadového stromu je možné uvést diagram závislosti, který predikuje riziko zavlečení počítačového viru do počítače v závislosti na tom, zda využíváme, nebo nevyužíváme sdílené soubory a zda používáme, nebo nepoužíváme antivirový program. V úvahu se brala i skutečnost, zda byl zkoumaný počítač už někdy předtím infikovaný.



Obr. 8.9 Příklad použití nevyváženého rozpadového stromu

Výsledkem je predikce možnosti nakažení počítačovým virem pro jednotlivé větve rozpadového stromu.



Obr. 8.10 Příklad použití nevyváženého rozpadového stromu

Fáze učení

V této fázi se vybraný data miningový model učí a upřesňuje své parametry na množině údajů získaných z dosud existujících případů. Jako podklad pro fázi učení slouží dosud sesbírané (naměřené) a vyhodnocené údaje. Výběru a přípravě údajů, které budou sloužit jako podklad pro učící fázi, musíme věnovat velkou pozornost, aby byly dostatečně reprezentativní a očištěné od případných chyb. Nejvýhodnější je připravit a předzpracovat tyto údaje v procesu ETL. Vidíme, že učící fáze je velmi náročná na výběr správného modelu a také na výběr a předzpracování údajů.

Analýza a predikce nových případů

V etapě analýzy a predikce aplikujeme už naučený data miningový model na množinu vstupních údajů, ze kterých potřebujeme získat souvislosti. Na rozdíl od fáze učení je tato fáze spíše náročná na výpočetní kapacitu použitého hardwaru a softwaru. Praktický příklad fáze učení a fáze analýzy a predikce nových případů bude předveden v praktických příkladech.

Výběr modelu

Pro vytvoření data miningového modelu je potřeba definovat vstupní sloupce – atributy z databázových tabulek, které obsahují shromážděné nebo naměřené údaje včetně definování spojitosti/diskrétnosti sloupců a distribuce hodnot.

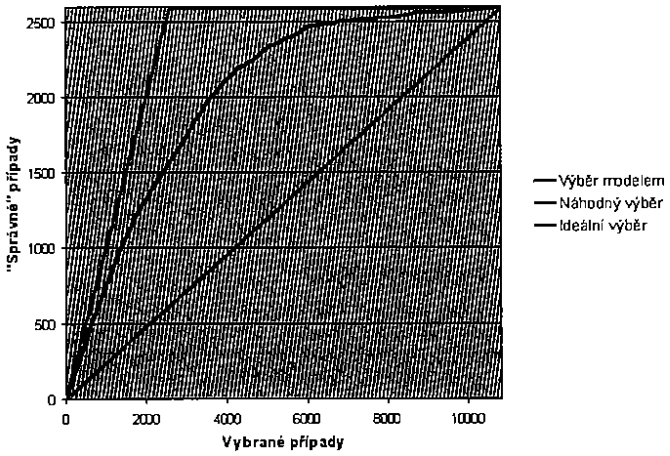
Zpracování, vyhodnocení a ověření modelu

Pomocí vývojového prostředí pro data mining je obvykle možno zjišťovat jednotlivé závislosti včetně zobrazení jejich váhy vzhledem ke zkoumané problematice.

Ověření modelu

V etapě vývoje a testování libovolného data miningového modelu obvykle testujeme predikci pro evaluační množinu. Z pohledu statistiky se jedná o testování hypotéz. Jednou z metod testování data miningového modelu je tzv. liftchart. Nejdříve seřadíme výsledky podle klesající pravděpodobnosti předpovědí a porovnáme je s náhodným výběrem (nejhorší možný případ) a s nejlepším teoreticky možným výběrem. Křivka pro správně navržený model by samozřejmě měla být co nejbližší ideální křivce a její sklon by se měl plynule snižovat. V opačném případě máme nesprávně zvolený model. Situace je zobrazena na diagramu.

- Lomená čára představuje ideální případ. Skládá se v podstatě ze dvou přímek. První přímka je strmá, protože jsou z množiny údajů vybrány všechny správné případy. Potom následuje zlom, protože žádný další případ už logicky nemůže být vybrán.
- Rovná přímka představuje teoreticky nejhorší případ – náhodný výběr.
- Křivka vyjadřuje výběr pomocí data miningového modelu. Čím je křivka blíže k ideálnímu průběhu, tím je testovaný model kvalitnější



Obr. 8.11 Graf pro ověření data miningového modelu

Praktické příklady data miningu

Jako praktické příklady se velmi často používají příklady nad naplněnými demonstračními databázemi, v případě SQL Serveru 2000 je to například databáze fiktivní firmy Northwind a databáze obchodního řetězce Foodmart.

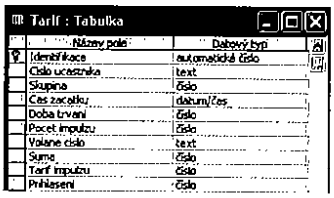
Než uvedeme příklady nad uvedenými demonstračními databázemi, musíme pochopit jejich strukturu (viz přílohy) a také ne každý čtenář bez ekonomického vzdělání pochopí strukturu firemní databáze, proto jako první použijeme jednoduchý příklad, dalo by se říci „ze života", tedy databázovou tabulku záznamů hovorů z telefonní ústředny. Vždyť telefonujeme (a platíme účty) prakticky všichni, takže údaje o telefonních hovorech, navíc koncentrované do jedné tabulky, není třeba zvlášť komentovat.

Už v teoretické části jsme upozorňovali, že se data mining dá provádět a tedy i zkoušet a procvičovat jen s reálnými údaji. Protože kde žádné souvislosti nejsou, například v testovací databázi, která byla vytvořená pomocí pseudonáhodného generátoru, logicky ani žádné souvislosti nevygenerujeme. Cenné reálné údaje pro data mining můžeme získat v knihovně univerzity *Department of Information and Computer Science University of California, Irvine CA* na adrese http://www.ics.uci.edu/~mlearn/MLRepository.html. Z tohoto zdroje pochází i naše cvičná databáze hub, kterou jsme použili v druhém příkladu.

Příklad data miningu s údaji z telefonní ústředny (MS SQL Server 2000)

Tento příklad je koncipován pro seznámení se s analytickým serverem dodávaným společně s databázovým serverem MS SQL Server 2000 a jeho nástroji. Provedeme jej celý včetně etapy ETL. Zdrojem údajů pro tento příklad jsou reálné záznamy z provozu telefonní ústředny.

Protože je tu určitý rozpor, když na jedné straně potřebujeme reálné údaje, a na druhé straně mnohé údaje podléhají utajení, jsou v našem případě konkrétní údaje zamaskované z důvodu zachování telekomunikačního tajemství. Sběr údajů v tomto případě spočíval v tom, že telefonní ústředna zaznamenává do databáze údaje o každém uskutečněném hovoru. Několikadenní záznam byl uložen do 20MB souboru typu MDB (soubor aplikace Microsoft Access) a obsahoval 175 400 záznamů o telefonních hovorech. Několik desítek tisíc údajů je už slušný balík na „vydolování" určitých souvislostí a možná i na předpovídání nějakého budoucího trendu. Struktura původní databázové tabulky je jasná z návrhového zobrazení.



Obr. 8.12 Struktura údajů z telefonní ústředny

Databázová tabulka, která bude sloužit jako podklad pro jednoduchý data mining, obsahuje sloupce

- Identifikátor
- Číslo volajícího
- Skupina
- Čas začátku
- Doba trvání
- Počet impulsů
- Volané číslo
- Suma
- Tarif impulsů
- Přihlášení

Abychom si utvořili ještě lepší představu o údajích v databázi, uvedeme několik ilustračních záznamů (konkrétní telefonní čísla jsou zamaskována).

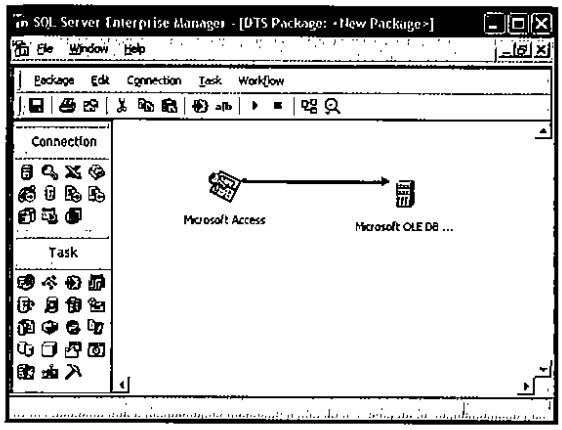
Volající	Skup	Čas začátku	Trvání	Impulsů	Volané číslo	Přihlášení
8x-951xx	1	2000-10-11 14:50:00	16	1	63451xx	17
8x-951xx	3	2000-10-11 15:02:00	12	2	090319xxxx	51
8x-952xx	1	2000-10-11 15:25:00	15	2	6345xxx	20

Vysvětlit je potřeba asi pouze sloupec **Skupina**. Tento údaj zařazuje uskutečněný telefonní hovor do jedné ze skupin. Nás budou zajímat skupiny:

0 Místní hovory
1 Meziměsto
3 Hovor na mobilní telefon
4 Hovor na audiotextovou službu
5 Mezinárodní hovor

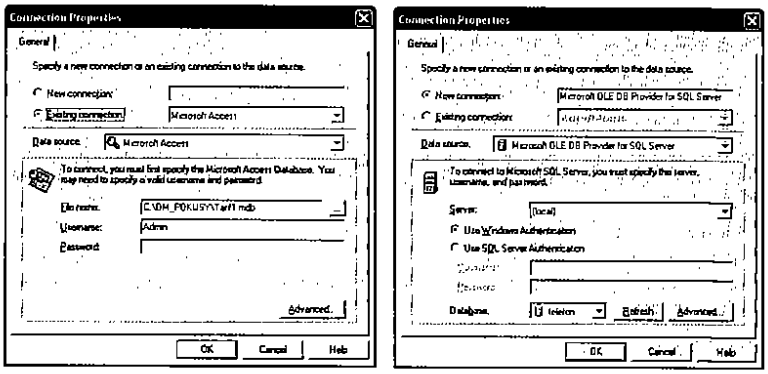
Příprava údajů

Údaje jsme přenášeli do nově vytvořené databáze pod správou SQL Serveru 2000 s názvem telefon. Použili jsme nástroj DTS (Data Transformation Services), který je součástí SQL Serveru 2000. Transformace byla v tomto případě velmi jednoduchá a spočívala ve vynechání některých záznamů, které nenesly informační hodnotu, a také i některých sloupců, které neobsahovaly podstatné informace nebo se daly odvodit z jiných sloupců, například protelefonovaná suma. Celý tento jednoduchý proces jsme nadefinovali v grafickém uživatelském prostředí **SQL Server Enterprise Manager**. Nejdříve metodou drag-and-drop přesuneme na pracovní plochu aplikace ikonu databáze, která obsahuje zdrojové údaje, v našem případě Access.



Obr. 8.13 Grafický návrh transformace údajů pomocí nástroje DTS

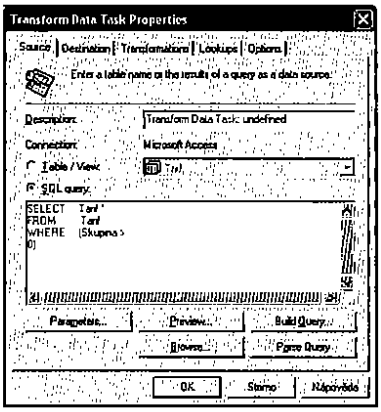
Parametry zdroje údajů a cílové databáze, kam jsme chtěli transportovat upravené údaje, jsme nadefinovali pomocí dialogů:



Obr. 8.14 Parametry zdroje údajů (vlevo) a cílové databáze (vpravo)

Pokud se pozorněji podíváme na předchozí obrázek, zjistíme, že máme nadefinovanou zdrojovou a cílovou databázi, ale samotnou transformaci máme sice znázorněnou šipkou, ale zatím není definovaná. Transformace se v našem případě bude skládat ze dvou kroků, restrikce a projekce.

Restrikce spočívá ve výběru jen platných údajů. Telefonní ústředna totiž z důvodů diagnostiky zaznamenává i údaje o spojeních, které se z různých důvodů neuskutečnily, například měl volaný obsazeno a podobně. Takové záznamy o hovorech mají ve sloupci skupina hodnotu menší než 0. Proto vybereme jen záznamy, které mají ve sloupci skupina nenulovou hodnotu. Dotaz SQL zadáme na záložce Source.



Obr. 8.15 Záložka Source – výběr vhodných údajů

Pro omezení tedy použijeme příkaz jazyka SQL:

```
SELECT * FROM Tarif WHERE (Skupina >= 0)
```

Po této úpravě nám z celkového počtu 175 400 záznamů zůstane 164 800 platných záznamů, tedy záznamů o reálně uskutečněných hovorech, na kterých telekomunikační společnost vydělala.

Projekce bude spočívat ve výběru potřebných sloupců v cílové databázové tabulce.

V dialogu *Create Destination Table* nám průvodce definováním transformace nabízí vytvoření nové tabulky na základě tabulky původní. Implicitní návrh zahrnuje všechny sloupce původní tabulky.

```
CREATE TABLE [Tarif]
(
  [Identifikace] int NOT NULL,
  [Cislo ucastnika] nvarchar (25) NOT NULL,
  [Skupina] tinyint NULL,
  [Cas zacatku] smalldatetime NULL,
  [Doba trvani] int NULL,
  [Pocet impulzu] int NULL,
  [Volane cislo] nvarchar (20) NULL,
  [Suma] float NULL,
```

```
  [Tarif impulzu] int NULL,
  [Prihlaseni] tinyint NULL
)
```

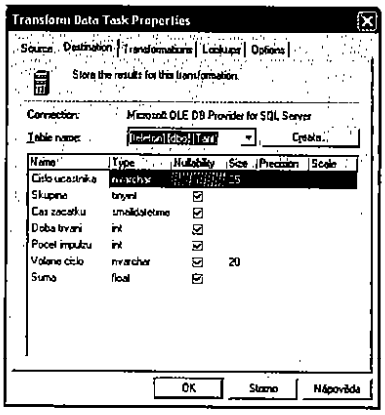
Jelikož jsme se rozhodli počet sloupců redukovat a ponechat jen sloupce

- Číslo volajícího
- Skupina
- Čas začátku
- Doba trvání
- Počet impulsů
- Volané číslo
- Suma

v příkazu pro vytvoření tabulky ponecháme jen řádky vyznačené tučným fontem. Takže upravený příkaz pro vytvoření tabulky v cílové databázi bude:

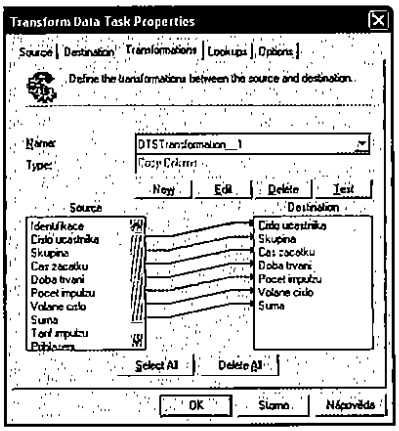
```
CREATE TABLE [Tarif]
(
  [Cislo ucastnika] nvarchar (25) NOT NULL,
  [Skupina] tinyint NULL,
  [Cas zacatku] smalldatetime NULL,
  [Doba trvani] int NULL,
  [Pocet impulzu] int NULL,
  [Volane cislo] nvarchar (20) NULL,
  [Suma] float NULL,
 )
```

Po vytvoření nové tabulky se nám její struktura zobrazí na záložce Destination.



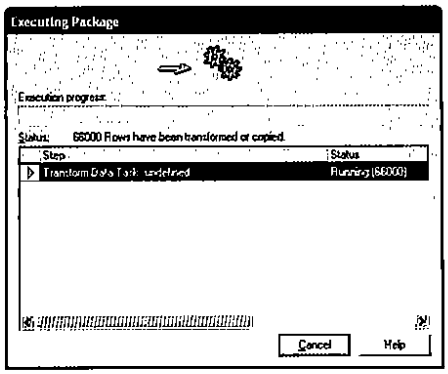
Obr. 8.16 Záložka Destination

V našem jednoduchém případě, kdy jsme jen vynechali některé sloupce, nám dokonce průvodce navrhl celou transformaci úplně automaticky.



Obr. 8.17 Záložka Transformation, definice parametrů a transformace údajů z databáze Access do databáze MS SQL Serveru 2000

Klepnutím na zelenou šipku můžeme přenos a transformaci spustit. O jejím průběhu budeme informováni v dialogu:



Obr. 8.18 Průběh přenosu a transformace údajů

Připojení analytického serveru k analyzovaným údajům

Po předchozí etapě přípravy údajů máme údaje v databázi *telefon*, konkrétně v jediné tabulce *Tarif*. Pokud chceme na těchto údajích provádět data mining, nemusíme je už nikam přesouvat. Dokonce jsme je mohli upravovat a ponechat v původním databázovém souboru s příponou MDB, ale pod správou databázového serveru budou tyto údaje rozhodně bezpečněji uložené a rychleji přístupné, než by tomu bylo v souborové databázi.

Pro práci s analytickým serverem integrovaným v MS SQL Serveru 2000, například pro vytváření a prohlížení modelů, slouží nástroj **Analysis Manager**.

Pokud si všimneme stromové struktury v levém okně aplikace, hierarchie začíná složkou analytických serverů. Pro každý server je vytvořená záložka, která na nižším stupni hierarchie obsahuje jednotlivé analytické databáze. Implicitně je nainstalována analytická databáze fiktivního obchodního řetězce FoodMart 2000 (relační databáze, na základě které byla táto analytická databáze vytvořena, je v souboru FoodMart 2000.MDB). Každá analytická databáze je v nástroji Analysis Manager reprezentována složkou obsahující následující položky (složky nižší hierarchie).

Data Sources (datové zdroje) – každá analytická databáze obsahuje připojení na jednu, případně i více relačních nebo multidimenzionálních databází, které fyzicky obsahují analyzované údaje.

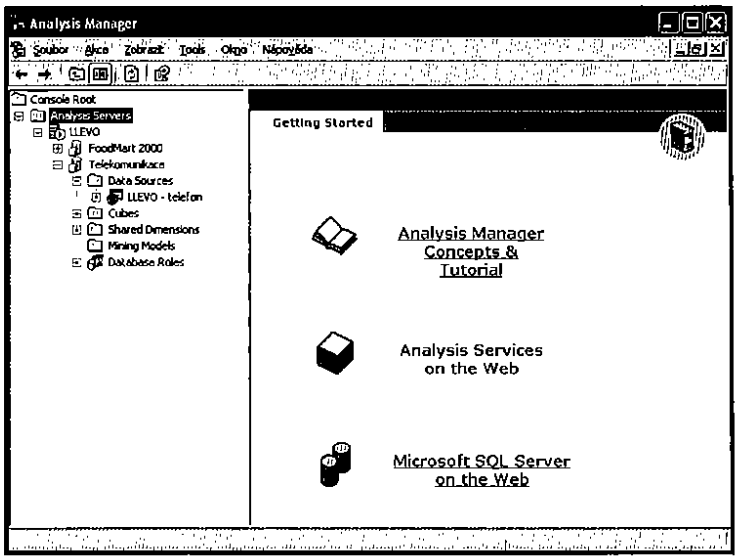
Cubes (krychle) – složka obsahuje krychle OLAP.

Shared Dimensions (sdílené dimenze zdroje) – složka obsahuje dimenze, které může společně sdílet více krychlí OLAP.

Mining Models (modely pro data mining) – každý data miningový model je reprezentován ikonkou, přičemž ikonka ve tvaru krychle znázorňuje data miningový model nad multidimenzionální databází OLAP a ikonka ve tvaru válce model nad relační databází.

Database Roles (databázové role) – jednotlivým krychlím OLAP a data miningovým modelů mohou být přiřazené databázové role. Výsledky analýz mohou mít totiž v mnoha případech doslova strategický význam.

Naší další úlohou bude tedy připojit analytický server k naší databázi. Pomocí nástroje Analysis Manager vytvoříme novou analytickou databázi a nazveme ji například *Telekomunikace*. Ve složce Data Sources vytvoříme propojení na příslušnou databázi, která obsahuje údaje, v našem případě na databázi *telefon*.



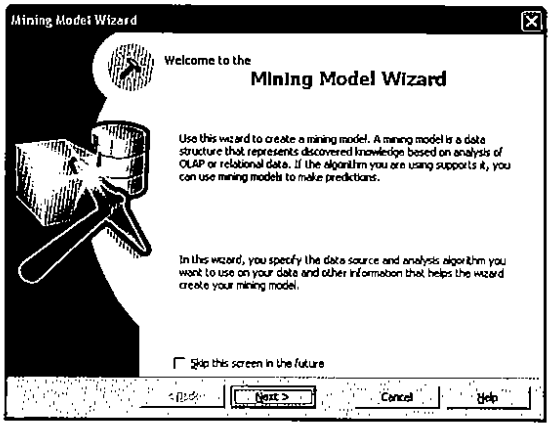
Obr. 8.19 Připojení analytického serveru ke zdroji údajů

Návrh modelu

Po úspěšné transformaci údajů můžeme přejít k návrhu modelu. Pomocí nástroje **Analysis Manager** ve složce *Mining Models* vytvoříme nad údaji z databáze „mining model" s názvem například *Odhad1*. Při vytváření nám bude asistovat Průvodce vytvořením data miningového modelu.

Jednotlivé dialogy průvodce vytvořením data miningového modelu jsou koncipovány tak, že logicky rozdělují tuto návrhovou etapu do šesti kroků (pokud nepočítáme spuštění vytvořeného modelu), které jsou téměř shodné s rozdělením etapy návrhu modelu, které doporučují teoretické prameny.

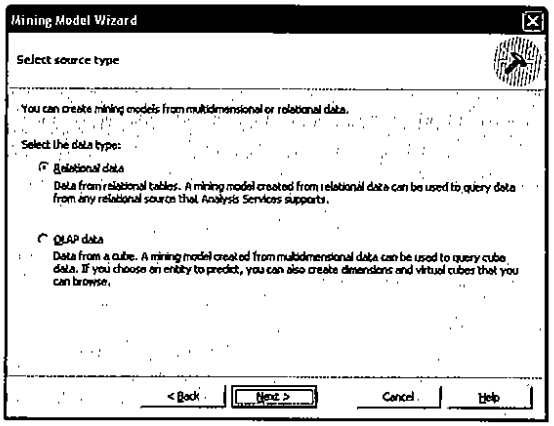
- Výběr zdroje údajů.
- Výběr případové tabulky nebo tabulek, které budou sloužit jako podklad pro vytvoření modelu.
- Výběr vhodného algoritmu.
- Editování propojení v případě více podkladových tabulek.
- Výběr klíčů v případové tabulce.
- Výběr vstupních a predikovaných sloupců.
- Spuštění modelu.



Obr. 8.20 Průvodce vytvořením data miningového modelu

Výběr zdroje údajů

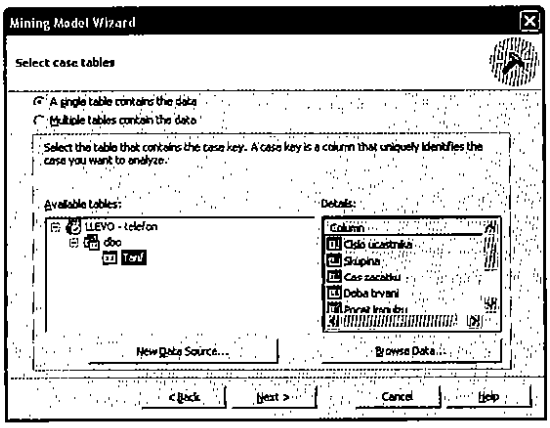
Těžit údaje můžeme jednak z relačních databází, ale také z multidimenzionálních datových struktur OLAP. Proto je od nás požadována specifikace typu zdroje údajů. Naše databáze je samozřejmě relační, takže volba v následujícím dialogu bude jednoznačná:



Obr. 8.21 Výběr typu zdrojových údajů

Výběr případové tabulky nebo tabulek, které budou sloužit jako podklad pro vytvoření modelu

Všimněme si přepínače v horní části dialogu, pomocí kterého zvolíme, zda zdrojem údajů bude jedna databázová tabulka, nebo více relačně provázaných tabulek. V našem případě máme situaci ohledně zdroje údajů relativně jednoduchou. Údaje máme v jedné databázové tabulce.



Obr. 8.22 Výběr konkrétní databázové tabulky

Výběr vhodného algoritmu

Následuje dialog pro výběr algoritmu pro data mining. Máme na výběr dva algoritmy:

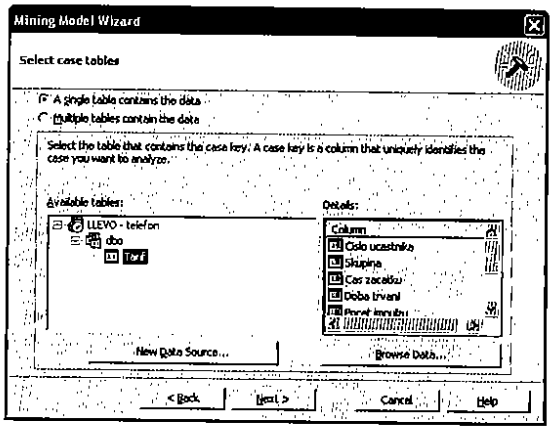
- Nevyvážené rozpadové stromy (Microsoft Decision Trees).
- Shlukový algoritmus (Microsoft Clustering).

Výběr klíče v případové tabulce

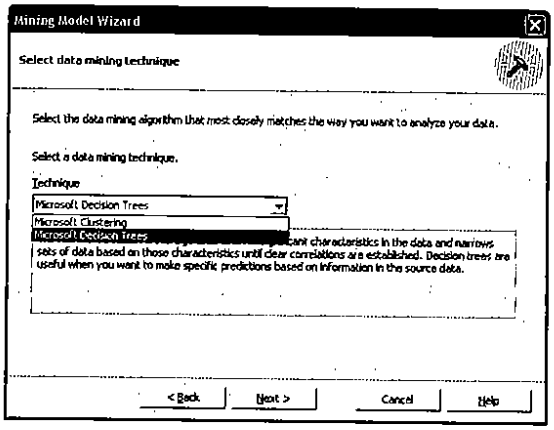
Tento krok slouží k výběru klíče, pomocí kterého bude každý analyzovaný případ identifikován. V našem případě jsme vybrali číslo volajícího.

Výběr vstupních a predikovaných sloupců

V dalším dialogu vybereme sloupce, jejichž hodnoty nebo průběh budeme predikovat, a sloupce, které budou vstupní. Dialog je rozdělen na tři části. Vlevo jsou sloupce zdrojové tabulky nebo tabulek, pravá část dialogu je rozdělena na dvě okna, na predikované a vstupní sloupce. Výběr spočívá v přesouvání sloupců mezi levým a pravým oknem.



Obr. 8.23 Výběr algoritmu pro data mining

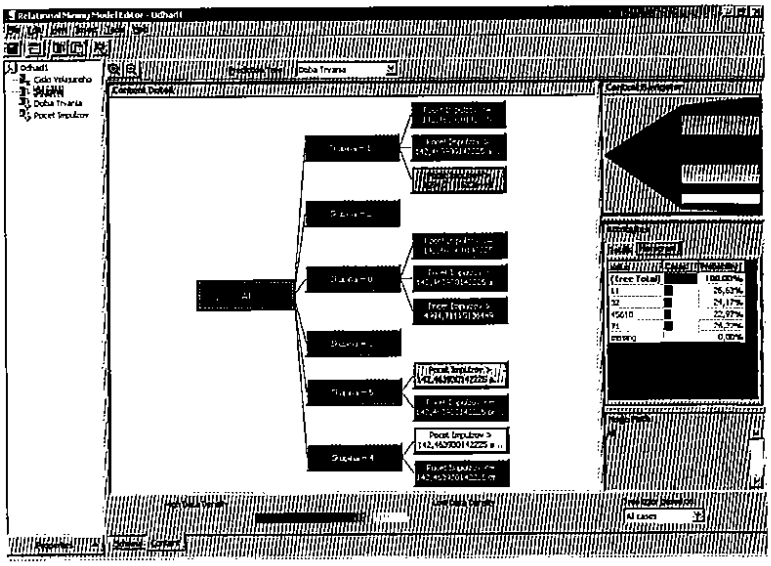


Obr. 8.24 Výběr vstupních a predikovatelných sloupců

Vyznačíme vstupní a predikované sloupce a můžeme celý proces spustit. Všimněme si ještě v dolní části dialogu, že máme možnost data miningový model dokončit pomocí editoru.

Spuštění modelu

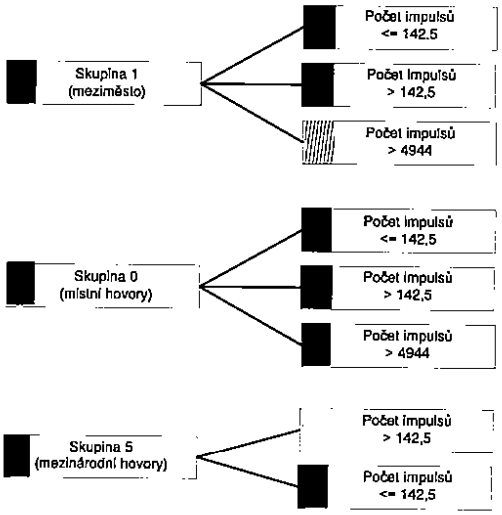
Po vytvoření data miningového modelu ho můžeme spustit. Po několika minutách máme výsledek dataminingu k dispozici v přehledné formě. Už základ stromu nám odhaluje souvislosti, na jaké skupiny telefonních čísel (viz tabulka výše) lidé nejdéle telefonují.



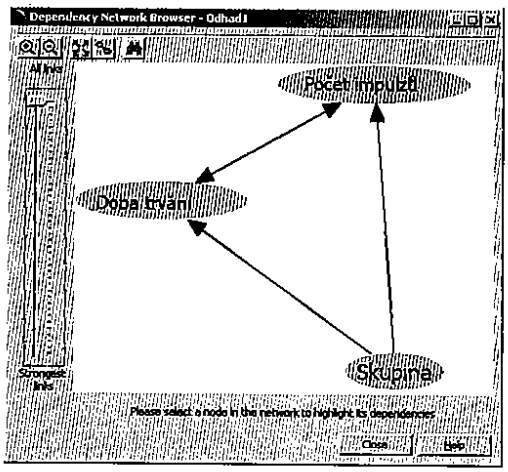
Obr. 8.25 Výsledek jednoduchého data miningu – závislost doby trvání hovoru na druhu hovoru

Hustota dat je v diagramu znázorněna barevně – čím tmavší barva, tím je v našem případě delší doba trvání hovoru. Pro větší názornost část grafu překreslíme jako samostatný obrázek.

Ve skupině 1 (meziměstské hovory) lidé telefonují o něco déle než ve skupině 0 (místní hovory) a v obou těchto skupinách telefonují podstatně déle než ve skupině 5 (mezinárodní hovory). V hlavním okně vidíme jen část stromu, v pravé horní části máme navigační okno, pomocí kterého se můžeme orientovat v celém stromu a zacházet do větších podrobností. Můžeme zobrazit i diagram vazeb mezi zkoumanými veličinami.

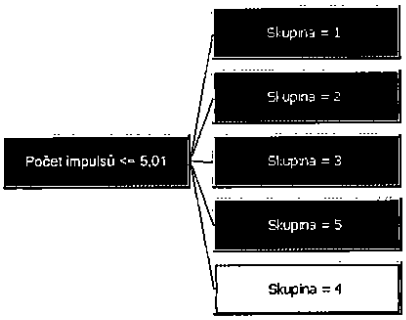


Obr. 8.26 Výsledek data miningu – závislost doby trvání hovoru na druhu hovoru (detail)

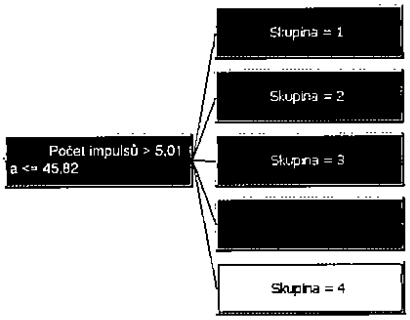


Obr. 8.27 Grafické znázornění vazeb

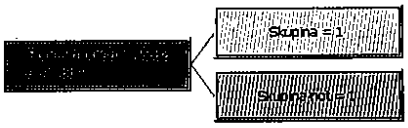
Je jen logické, a snadno pochopitelné, že mezi délkou trvání hovoru a počtem impulsů je vzájemná souvislost. Oba dva uvedené parametry však závisí na druhu hovoru, v našem příkladu vyjádřeném parametrem *Skupina*. Můžeme zkoumat i jiné souvislosti, například jak jsou zastoupené jednotlivé kategorie telefonních hovorů v určitých intervalech počtů impulsů a podobně.



Obr. 8.28 Zkoumání jiných závislostí (počet impulsů < 5,01)



Obr. 8.29 Zkoumání jiných závislostí (počet impulsů z intervalu 5,02 až 45,82)



Obr. 8.30 Zkoumání jiných závislostí (počet impulsů z intervalu 45,83 až 354)

Všimněme si, že v intervalu impulsů 45,83 až 354 je výsledek analýzy zobrazený předběžně jen ve dvou oknech, pro skupinu 1 a pro ostatní skupiny dohromady. Samozřejmě můžeme toto políčko v hierarchické struktuře dále rozložit.

V „dolování" bychom samozřejmě mohli pokračovat. Pokud bychom v etapě přípravy údajů například konkrétně specifikovali dny v týdnu a přesněji specifikovali jednotlivé skupiny telefonních čísel, na které se volá, zjistili bychom, ve které dni nebo v kterou dobu se nejvíce telefonuje na mobilní telefony nebo do zahraničí, jak se činí audiotextové firmy, kdy jsou telefonické soutěže v televizi nebo rozhlase, mohli bychom statisticky zkontrolovat, zda se nepodvádí v hlasovacích soutěžích (na Slovensku je to například relace SITO), mohli bychom všechno predikovat... ale z důvodu zachování telekomunikačního tajemství to raději necháme na operátorech pevných a mobilních sítí.

Příklad data miningu z reálné praxe – databáze hub

V další praktické ukázce data miningu ukážeme celý postup včetně rozdělení databáze údajů na učící a testovací množinu, vytvoření modelu pro predikci a také si ukážeme i samotnou predikci včetně jejího otestování. Námět je velmi jednoduchý. Podle určitých příznaků zkoumáme, zda je příslušná houba jedlá, nebo nejedlá. Databáze hub pro tento už klasický příklad data miningu byla získána z knihovny univerzity *Department of Information and Computer Science University of California, Irvine CA*.

Příprava údajů

Pro zajímavost, databáze hub byla původně umístěná ve dvou souborech agaricus-lepiota.data a agaricus-lepiota.names. Na tomto místě předvedeme jen ukázku původních „databázových" souborů, aby si čtenář uvědomil, jaké transformace bylo potřeba v etapě přípravy údajů provést. Podrobně to popisujeme v třetí kapitole věnované přípravě údajů. Překlad názvů do češtiny byl tedy jen nepodstatnou částí celého procesu.

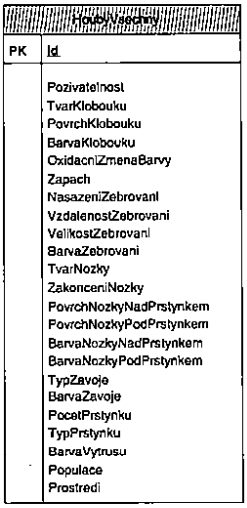
Soubor **agaricus-lepiota.data** (jen ukázka, celý soubor má 382 kilobajtů):

```
p,x,s,n,t,p,f,c,n,k,e,e,s,s,w,w,p,w,o,p,k,s,u
e,x,s,y,t,a,f,c,b,k,e,c,s,s,w,w,p,w,o,p,n,n,g
e,b,s,w,t,l,f,c,b,n,e,c,s,s,w,w,p,w,o,p,n,n,m
p,x,y,w,t,p,f,c,n,n,e,e,s,s,w,w,p,w,o,p,k,s,u
e,x,s,g,f,n,f,w,b,k,t,e,s,s,w,w,p,w,o,e,n,a,g
e,x,y,y,t,a,f,c,b,n,e,c,s,s,w,w,p,w,o,p,k,n,g
e,b,s,w,t,a,f,c,b,g,e,c,s,s,w,w,p,w,o,p,k,n,m
e,b,y,w,t,l,f,c,b,n,e,c,s,s,w,w,p,w,o,p,n,s,m
p,x,y,w,t,p,f,c,n,p,e,e,s,s,w,w,p,w,o,p,k,v,g
e,b,s,y,t,a,f,c,b,g,e,c,s,s,w,w,p,w,o,p,k,s,m
e,x,y,y,t,l,f,c,b,g,e,c,s,s,w,w,p,w,o,p,n,n,g
e,x,y,y,t,a,f,c,b,n,e,c,s,s,w,w,p,w,o,p,k,s,m
e,b,s,y,t,a,f,c,b,w,e,c,s,s,w,w,p,w,o,p,n,s,g
p,x,y,w,t,p,f,c,n,k,e,e,s,s,w,w,p,w,o,p,n,v,u
...
```

Soubor **agaricus-lepiota.names**:

```
classes: edible=e, poisonous=p
     1. cap-shape:                bell=b,conical=c,convex=x,flat=f,
                                  knobbed=k,sunken=s
     2. cap-surface:              fibrous=f,grooves=g,scaly=y,smooth=s
     3. cap-color:                brown=n,buff=b,cinnamon=c,gray=g,green=r,
                                  pink=p,purple=u,red=e,white=w,yellow=y
     4. bruises?:                 bruises=t,no=f
     5. odor:                     almond=a,anise=l,creosote=c,fishy=y,foul=f,
                                  musty=m,none=n,pungent=p,spicy=s
     6. gill-attachment:          attached=a,descending=d,free=f,notched=n
...
...
    19. ring-type:                cobwebby=c,evanescent=e,flaring=f,large=l,
                                  none=n,pendant=p,sheathing=s,zone=z
    20. spore-print-color:        black=k,brown=n,buff=b,chocolate=h,green=r,
                                  orange=o,purple=u,white=w,yellow=y
    21. population:               abundant=a,clustered=c,numerous=n,
                                  scattered=s,several=v,solitary=y
    22. habitat:                  grasses=g,leaves=l,meadows=m,paths=p,
                                  urban=u,waste=w,woods=d
```

Pochopení struktury tabulek databáze **Houby**, která obsahuje údaje o houbách ani v tomto případě nebude pravděpodobně nikomu činit problémy.



Obr. 8.31 Tabulka charakteristických vlastností hub

Tabulka *HoubyVsechny*, jak vyplývá z názvu, obsahuje údaje o houbách. Pro názornost několik údajů z této tabulky vypíšeme. Protože má tabulka mnoho sloupců, rozdělili jsme výpis do několika bloků.

id	Pozivatelnost	TvarKlobouku	PovrchKlob	BarvaKlob	Ox.ZmenaBarvy	Zapach
1	nejedlá, jedovatá	vypouklý	hladký	hnědá	ano	štiplavý, čpavý
3	jedlá	zvonovitý	hladký	bílá	ano	anýzový
5	jedlá	vypouklý	hladký	šedá	ne	žádný
7	jedlá	zvonovitý	hladký	bílá	ano	po mandlích
9	nejedlá, jedovatá	vypouklý	šupinatý	bílá	ano	štiplavý, čpavý
11	jedlá	vypouklý	šupinatý	žlutá	ano	anýzový
13	jedlá	zvonovitý	hladký	žlutá	ano	po mandlích
15	jedlá	vypouklý	vláknitý	hnědá	ne	žádný
17	jedlá	plochý	vláknitý	bílá	ne	žádný
19	nejedlá, jedovatá	vypouklý	šupinatý	bílá	ano	štiplavý, čpavý
21	jedlá	zvonovitý	hladký	žlutá	ano	po mandlích

...vodorovné pokračování tabulky...

NasazeniZebrovani	VzdalenostZebrovani	VelikostZebr	BarvaZebr	TvarNozky
volné	těsné	úzké	černá	rozšiřující se dolů
volné	těsné	široké	hnědá	rozšiřující se dolů
volné	velmi těsné	široké	černá	zužující se dolů
volné	těsné	široké	šedá	rozšiřující se dolů
volné	těsné	úzké	růžová	rozšiřující se dolů
volné	těsné	široké	šedá	rozšiřující se dolů
volné	těsné	široké	bílá	rozšiřující se dolů
volné	velmi těsné	široké	hnědá	zužující se dolů
volné	velmi těsné	široké	černá	zužující se dolů
volné	těsné	úzké	hnědá	rozšiřující se dolů
volné	těsné	široké	černá	rozšiřující se dolů

...vodorovné pokračování tabulky...

ZakonceniNozky	PovrchNadPrstynkem	PovrchNozkyPod	BarvaNozkyNad	BarvaNozkyPod	TypZavoje
stejný, rovný	hladký	hladký	bílá	bílá	částečný
hůl, pálka	hladký	hladký	bílá	bílá	částečný
stejný, rovný	hladký	hladký	bílá	bílá	částečný
hůl, pálka	hladký	hladký	bílá	bílá	částečný
stejný, rovný	hladký	hladký	bílá	bílá	částečný
hůl, pálka	hladký	hladký	bílá	bílá	částečný
hůl, pálka	hladký	hladký	bílá	bílá	částečný
stejný, rovný	hladký	vláknitý	bílá	bílá	částečný
stejný, rovný	hladký	hladký	bílá	bílá	částečný
stejný, rovný	hladký	hladký	bílá	bílá	částečný
hůl, pálka	hladký	hladký	bílá	bílá	částečný

...vodorovné pokračování tabulky...

BarvaZavoje	PocetPrstynku	TypPrstynku	BarvaVytrusu	Populace	Prostredi
bílá	jeden	přívěskovitý	černá	roztroušená	město
bílá	jeden	přívěskovitý	hnědá	početná	louka
bílá	jeden	rozplývající se	hnědá	velmi početná	tráva
bílá	jeden	přívěskovitý	černá	početná	louka
bílá	jeden	přívěskovitý	černá	několik hub	tráva
bílá	jeden	přívěskovitý	hnědá	početná	tráva
bílá	jeden	přívěskovitý	hnědá	roztroušená	tráva
bílá	jeden	rozplývající se	černá	velmi početná	tráva
bílá	jeden	rozplývající se	hnědá	velmi početná	tráva
bílá	jeden	přívěskovitý	hnědá	roztroušená	město
bílá	jeden	přívěskovitý	hnědá	roztroušená	louka

Tabulku **HoubyVsechny** rozdělíme systémem sudá, lichá na dvě tabulky, učební a testovací:

- **HoubyUcici**
- **HoubyTestovaci**

Každá tabulka obsahuje 4 062 záznamů, přičemž tabulka HoubyUcici obsahuje záznamy se sudými indexy a tabulka HoubyTestovaci obsahuje záznamy s lichými indexy. Celková statistika zastoupení jedlých a jedovatých hub je následující:

```
jedlé:  4 208 (51,8 %)
jedovaté: 3 916 (48,2 %)
celkem:  8 124 hub
```

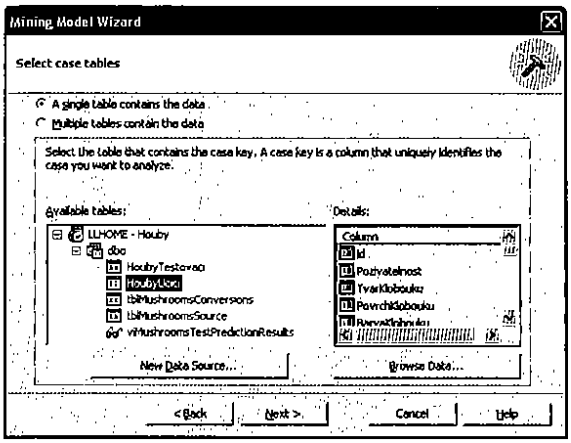
Vytvoření modelu pro Data Mining

Tabulka ***HoubyUcici*** obsahuje 4 062 záznamů. Několik tisíc údajů je už poměrně slušný balík na „vydolování" souvislostí a v některých aplikacích i na předpověď budoucího trendu. Přistupme pro ilustraci k vlastnímu data miningu. Pomocí nástroje **Analysis Manager** vytvořme nad údaji v databázi „mining model" s názvem například *JedleHouby*.

V jednom z prvních dialogů od nás Průvodce vytvořením data miningového modelu požaduje zadat typ zdroje údajů, buď typu relační databáze, nebo typu krychle OLAP. Naše databáze hub je samozřejmě relační. Následuje výběr tabulky nebo více tabulek, které se stanou podkladem pro proces data miningu. V našem případě to bude tabulka *HoubyUcici*.

V následujícím kroku je potřeba vybrat typ algoritmu pro analýzu údajů a predikci. Microsoft v analytických službách SQL Serveru 2000 nabízí dvě možnosti:

- **Microsoft clustering** (Vícerozměrné shlukové diagramy) – algoritmus se používá pro odhalování shluků dat v multidimenzionálních prostorech.
- **Microsoft decision Trees** (Nevyvážený rozpadový strom) – odhaluje závislosti a vyhledává specifické vlastnosti, které potom poslouží pro predikci.



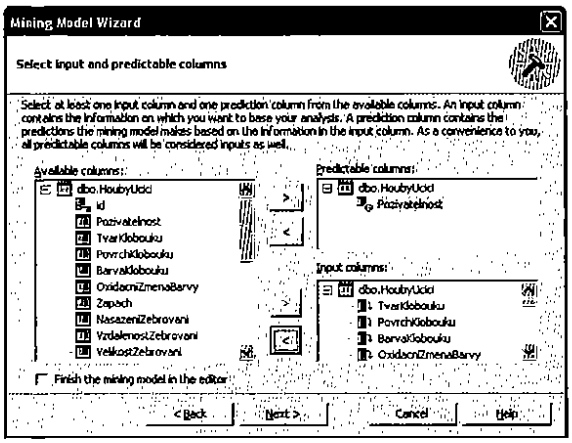
Obr. 8.32 Model data miningu jedlých hub – výběr tabulek

Microsoft decision Trees

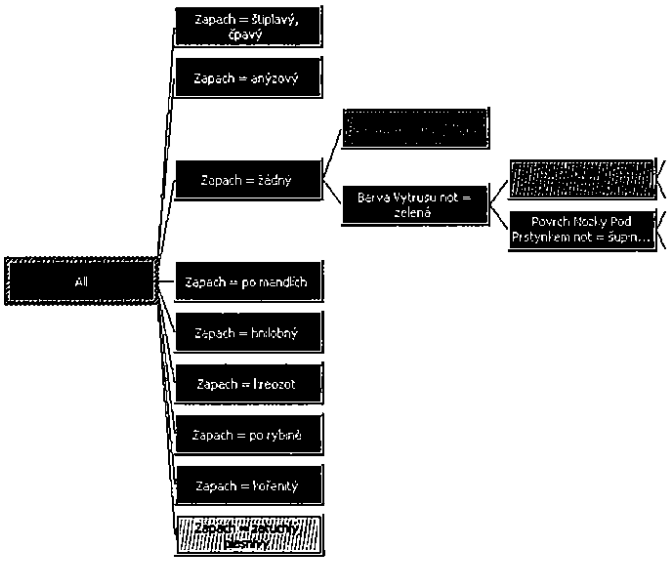
Zkušení houbaři by určitě dokázali takový diagram sestavit i pro náš příklad jen **na základě svých zkušeností**. A jsme vlastně doma. Houbař získal své poznatky na základě dlouholetých zkušeností, proto se na jeho odhad můžeme s vysokou mírou jistoty spolehnout. Zavání to možná i troškou černého humoru, ale tato míra jistoty je o to větší, že houbaři, kteří se při realizaci své záliby dopustili závažného omylu, už mezi námi nejsou.

SQL Server 2000 na houby nechodí a žádné poznatky o nich tedy nemá (stejně jako nemá poznatky o vývoji dámské módy, počasí a podobně). Na základě zaznamenaných údajů však tyto poznatky může získat. V následujícím jednoduchém dialogu je potřeba vybrat klíčový sloupec (unikátní identifikátor pro každý záznam), v našem případě to bude ID. Pokračujeme dalším dialogem, nejdůležitějším, ve kterém specifikujeme sloupce, které chceme predikovat, a sloupce, na základě jejichž hodnot chceme predikci uskutečnit. V našem příkladu budeme predikovat požívatelnost na základě všech ostatních znaků.

Výsledkem bude přehledný diagram pro všechny hodnoty sloupce **požívatelnost**, tedy pro jedlé i jedovaté houby. Jedním slovem pro všechny druhy hub. Čím je barva políčka tmavší, tím je pravděpodobnost výskytu této vlastnosti vyšší.

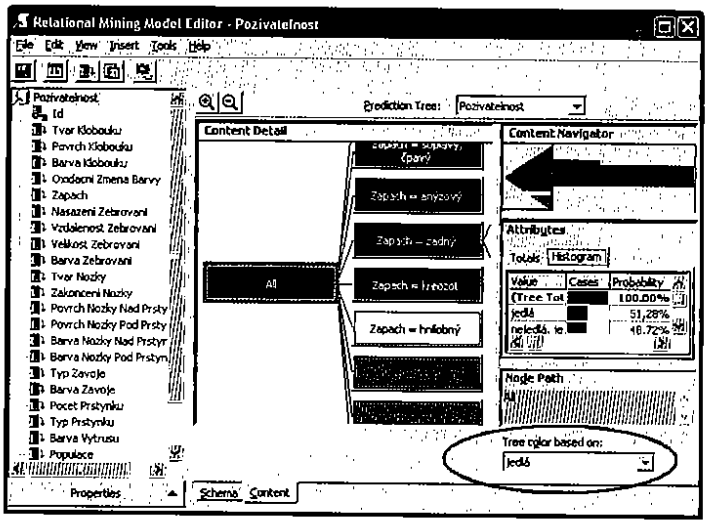


Obr. 8.33 Model data miningu jedlých hub – výběr vstupních a predikovatelných sloupců



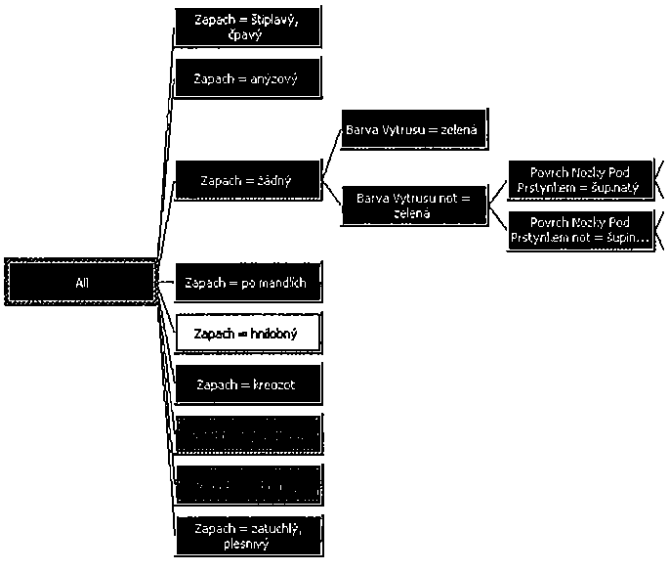
Obr. 8.34 Odhalené závislosti pro všechny houby

Nejčastěji se tedy na základě našich údajů v přírodě vyskytují houby, které pod prstýnkem nejsou šupinaté, nemají barvu výtrusu zelenou a nijak nezapáchají. Na základě tohoto diagramu samozřejmě nemůžeme určit vlastnosti hub jedlých a nejedlých. Všimněme si v levé dolní části okna aplikace **Relational Mining Model editor** ovládacího prvku s názvem **Tree color based on**. Pomocí tohoto prvku můžeme nastavit, k jaké hodnotě sloupce požívatelnost (jedlé, nejedlé) se budou intenzity barev v diagramu vztahovat.



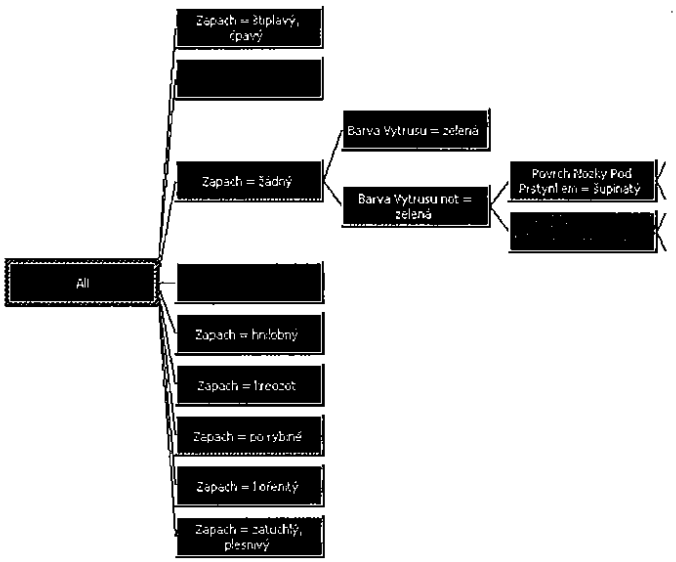
Obr. 8.35 Relational Mining model editor

Podívejme se podrobněji na diagram pro jedlé houby.



Obr. 8.36 Odhalené závislosti pro jedlé houby

Z výsledků analýzy vyplývá, že jedním z charakteristických znaků jedlých hub je to, že nemají zelený výtrus, ale hlavním kritériem pro určení konzumovatelnosti je jejich zápach. Můžeme tedy akceptovat houby bez zápachu nebo takové, co jsou cítit po anýzu nebo po mandlích. A naopak vidíme, že houby, které mají hnilobný zápach, se s největší pravděpodobností dají sníst jen jednou. Jelikož sloupec požívatelnost obsahuje jen dvě hodnoty, bude diagram pro nejedlé a jedovaté huby jakýmsi přibližným negativem hub jedlých. Pokud má houba zelený výtrus, případně má štiplavý zápach, hnilobný zápach nebo čpí po kreozotu (ropný produkt), rybách, případně má kořenitý nebo zatuchlý, plesnivý zápach, na konzumaci se určitě nehodí.

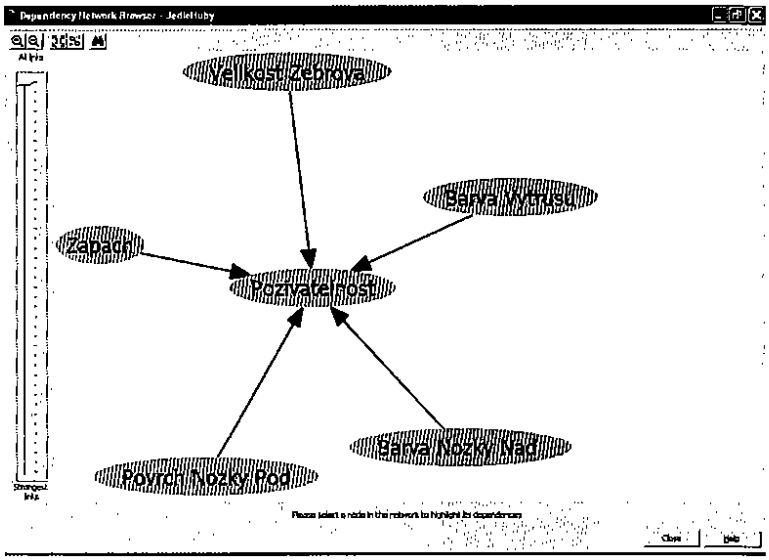


Obr. 8.37 Odhalené závislosti pro nejedlé houby

Všeobecně jsme odhalili, že konzumovatelnost hub, teda zda jsou jedlé, případně nejedlé nebo jedovaté, závisí v první řadě na jejich zápachu. Potvrdí nám to i nástroj **Dependency Network Browser**.

Práce s tímto nástrojem je velmi jednoduchá. Po jeho spuštění se ve formě přehledného diagramu zobrazí, na čem a do jaké míry závisí predikovatelná veličina. V našem případě závisí poživatelnost hub na těchto vlastnostech (shora ve směru hodinových ručiček):

- velikost žebrování,
- barva výtrusu,
- barva nožičky nad prstýnkem,
- povrch nožičky pod prstýnkem,
- zápach.



Obr. 8.38 Dependency Network Browser

Zatím nám chybí informace, která vlastnost nejvíce ovlivňuje konzumovatelnost hub. Všimněme si však posuvného ovládacího prvku levé části okna aplikace. Pokud ho budeme posouvat směrem dolů, postupně nám odpadnou méně významné vlivy. Pořadí vlastností, které ovlivňují jedlost hub, bude tedy v pořadí od nejcharakterističtější po nejméně charakteristickou vlastnost následovně:

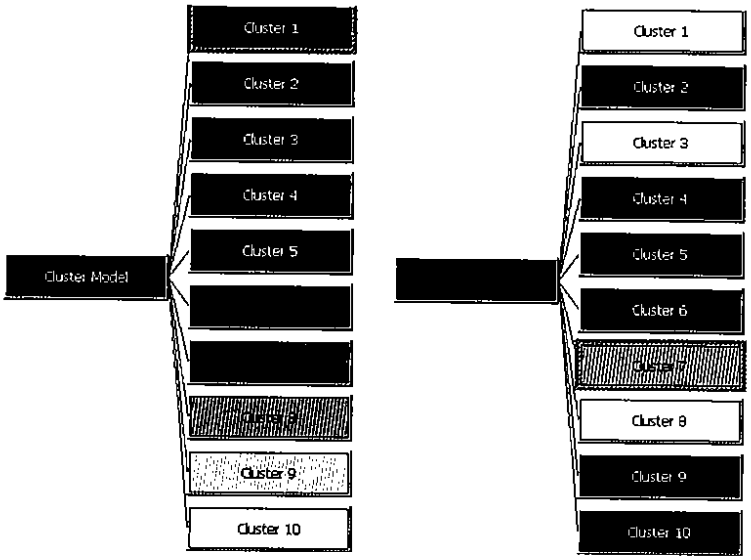
- **zápach,**
- **barva výtrusu,**
- **povrch nožičky pod prstýnkem,**
- **velikost žebrování,**
- **barva nožičky nad prstýnkem.**

Tento příklad je velmi vhodný pro první pokusy s data miningem. Souvislosti totiž odhalíme jen tam, kde se opravdu nějaké nachází. Proto se pro data mining hodí právě takové příklady z reálného života. Z uměle vytvořených tabulek toho pravděpodobně moc nevydolujeme.

Alternativní algoritmus – Microsoft clustering

Ukážeme si i případ, kdy budeme muset náš prvotní model mírně upravit. Tentokrát použijeme alternativní algoritmus. Postup při vytváření alternativního modelu, který využívá

algoritmus pro odhalování shluků, je téměř stejný jako pro rozhodovací strom. Výsledky, které dostaneme budou, v tomto případě prezentovány odlišným způsobem.



Obr. 8.39 Výsledek algoritmu Microsoft Clustering

Na první pohled se zdá být dosažený výsledek triviální, houby byly rozdělené do deseti skupin (diagram vlevo). Ale pokud si prohlédneme charakteristické vlastnosti pro jednotlivá seskupení, uvidíme, že data miningový stroj nemrhal strojovým časem počítače a na základě výpisu (My jsme ho ovlivnění prvním dojmem z důvodu úspory místa zjednodušili. Vždyť ani reálný výpis bychom zřejmě nečetli detailně pro všechny případy. Zajímal by nás asi jen jeho začátek a konec, ale za chvíli **uvidíme, jaké chyby bychom se dopustili.**) by se možná daly definovat i nějaké houbařské teorie.

```
Cluster1 (870 případů)
Barva Zebrovani = hnědožlutá,
Zapach = po rybině, kořenitý,
Barva Vytrusu = bílá,
Zakonceni Nozky = (neurčený),
Velikost Zebrovani = úzké,
Typ Prstynku = rozplývající se,
Tvar Klobouku = hrbolek uprostřed,
Prostredi = listí,
Barva Klobouku = červená, hnědá
```

```
Pozivatelnost = nejedlá, jedovatá,
Populace = několik hub,
Tvar Nozky = zužující se dolů,
Barva Nozky Pod Prstynkem = růžová,
Oxidacni Zmena Barvy = ne,
Barva Nozky Nad Prstynkem = růžová,
Povrch Nozky Pod Prstynkem = hedvábný,

Cluster2 (855 případů)
Barva Nozky Nad Prstynkem = šedá,
Barva Zebrovani = purpurová,
Prostredi = dřevo,
Oxidacni Zmena Barvy = ano,
Zapach = žádný,
Zakonceni Nozky = cibulovité,
Typ Prstynku = přívěskovitý,
Pozivatelnost = jedlá,
Populace = osamělá,
Tvar Nozky = zužující se dolů,
Barva Vytrusu = černá, hnědá
Povrch Nozky Pod Prstynkem = hladký,
Velikost Zebrovani = široké,
Povrch Klobouku = vláknitý,
Barva Klobouku = červená,
Barva Zebrovani = hnědá, bílá

Cluster3 (655 případů)
Barva Nozky Nad Prstynkem = hnědá, hnědožlutá,
Typ Prstynku = velký,
Barva Vytrusu = čokoládová,
Zapach = hnilobný,
Povrch Nozky Nad Prstynkem = hedvábný,
Barva Klobouku = žlutá, šedá
Barva Zebrovani = čokoládová, šedá
Tvar Nozky = rozšiřující se dolů,
Zakonceni Nozky = cibulovité,
Pozivatelnost = nejedlá, jedovatá,
Populace = osamělá,
Oxidacni Zmena Barvy = ne,
Prostredi = cesta,
Velikost Zebrovani = široké

..

Cluster9 (127 případů Pozivatelnost = jedlá,
Cluster10 (117 případů Pozivatelnost = jedlá,
```

Nás však zajímá hlavně poživatelnost hub. Přepneme proto parametr a vygenerujeme nový diagram, tentokrát bude barva okna reprezentovat poživatelnost. V pravé části našeho obrázku vidíme takový diagram rozdělení na shluky jedlých a nejedlých hub. Černá barva

obdélníku znamená jedlé huby. Shluky 2, 4, 5, 9 a 10 představují jedlé houby, shluky 1, 3 a 8 jsou reprezentovány obdélníky bílé barvy, představují tedy nejedlé a jedovaté houby. Zajímavý je v tomto případě shluk 6 (tmavě modrý) a shluk 7 (světle modrý), pokud si necháme vypsat vlastnosti, podle kterých byly jednotlivé případy rozdělené do shluků.

```
Cluster6 (258 případů)
Typ Prstynku = rozšiřující se,
Povrch Klobouku = rýhovaný, vláknitý
Barva Klobouku = purpurová, zelená, růžová, skořicová
Tvar Klobouku = propadený, vmáčknutý,
Zapach = kreozot,
Velikost Zebrovani = úzké,
Barva Nozky Nad Prstynkem = bílá,
Vzdalenost Zebrovani = velmi těsné,
Tvar Nozky = rozšiřující se dolů,
Prostredi = město, dřevo,
Barva Nozky Pod Prstynkem = žlutá, bílá,
Povrch Nozky Nad Prstynkem = hladký

Cluster7 (235 případov)
Barva Vytrusu = žlutá, oranžová, hnědožlutá, čokoládová
Barva Zebrovani = oranžová, žlutá
Barva Nozky Nad Prstynkem = oranžová,
Barva Zavoje = oranžová, hnědá,
Nasazeni Zebrovani = pevně připojené,
Barva Klobouku = hnědožlutá,
Povrch Klobouku = hladký,
Prostredi = město, listí,
Populace = shluky,
Typ Prstynku = přívěskovitý,
Povrch Nozky Nad Prstynkem = vláknitý,
Zapach = hnilobný
```

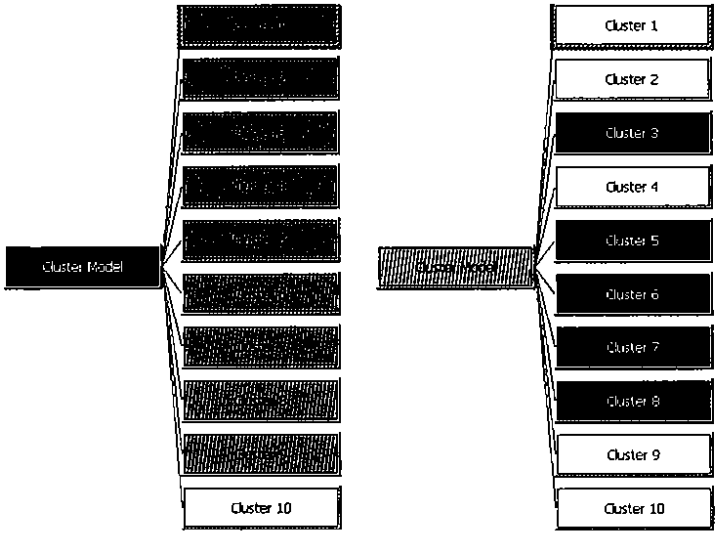
S překvapením zjistíme, že poživatelnost v těchto dvou shlucích vůbec nefiguruje. Podívejme se na histogramy v pravé části okna aplikace. Zatímco pro shluky 2, 4, 5, 9 a 10, které představují jedlé houby, je pravděpodobnost výskytu 100% a pro nejedlé a jedovaté houby je pravděpodobnost výskytu 0%, pro shluky 1, 3 a 8 jedovatých hub je to přesně naopak, pro shluky 6 a 7 jsou poměry téměř stejné.

Value	Cases	Probability
(Cluster 6 Total)		100,00%
jedlá		53,10%
nejedlá, jedovatá		46,90%
missing		0,00%

Value	Cases	Probability
(Cluster 7 Total)		100,00%
jedlá		44,68%
nejedlá, jedovatá		55,32%
missing		0,00%

Obr. 8.40 Histogramy pro shluky 6 a 7

Na základě takového modelu bychom tedy možná vytvořili krásnou teorii z hlediska přírodopisného, ale z hlediska kulinářského by to byla teorie mírně morbidní. Proto musíme model shlukového algoritmu mírně upravit, přičemž využijeme poznatky z modelu na principu rozpadového stromu, kde jsme zjistili, že hlavní vliv na poživatelnost hub má zápach a barva výtrusu.



Obr. 8.41 Výsledek algoritmu Microsoft Clustering při jiných vstupních podmínkách

V pravé části podobně jako v předchozím případě vidíme rozdělení na shluky jedlých a nejedlých hub. Černá barva obdélníka znamená jedlé houby. V tomto případě jsou houby do shluků rozdělené s výjimkou posledního desátého shluku téměř rovnoměrně. První shluk obsahuje 436 případů, devátý shluk 403 případů a poslední desátý shluk obsahuje 279 případů. Ale rozdělení na shluky jedlých a jedovatých hub je v tomto případě úplně jednoznačné.

Analogie cvičného příkladu s jinými oblastmi – stanovení úvěrového rizika...

Někdo by se mohl zeptat: „A k čemu je to dobré?"

Poznámka autora: Přesně na toto se mě ptala zkušební komise při disertační zkoušce, kdy jsem tento příklad použil ve své práci na ilustraci možností data miningu.

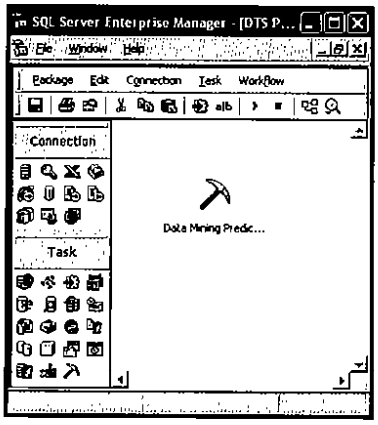
Lidé, kteří na téma „otrava houbami" už leželi v nemocnici, by určitě věděli. Průnik množin houbařů a analytiků používajících SQL Server asi nebude velký. SQL Server, a hlavně jeho analytické služby skutečně více používají finanční a marketingoví odborníci a jiní spe-

cialisté převážně z této branže. Náš příklad s určením konzumovatelnosti hub má velmi dobrou analogii k jiným veličinám, které na základě dosud nasbíraných údajů potřebujeme predikovat. Je to například stanovení úvěrového rizika při poskytování úvěru fyzickým osobám. Podobně jako jsme o houbách, se kterými jsme dosud měli zkušenosti, dokázali shromáždit různé údaje, přičemž nás v konečném důsledku jako správné konzumenty zajímá hlavně konzumovatelnost, i o zákaznících, kterým jsme dosud poskytli úvěr nebo kreditní kartu, máme shromážděno poměrně hodně údajů. Jen stačí najít souvislosti mezi ostatními vlastnostmi a mírou úvěrového rizika.

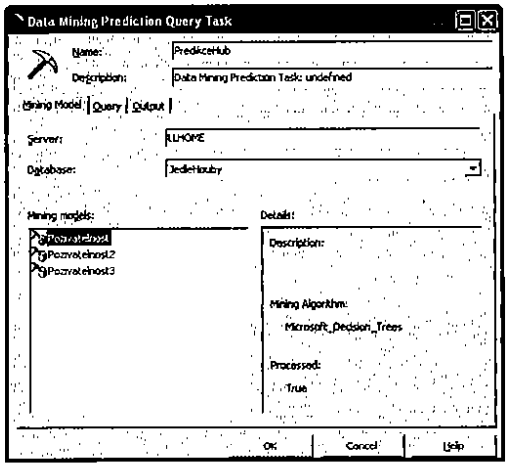
Dokonce bychom se mohli s určitou dávkou humorného nadhledu zamyslet i nad analogií výsledků analýzy konzumovatelnosti hub a mírou úvěrového rizika. U hub byl určujícím kritériem zápach. Asi je každému jasné, jaká je míra úvěrového rizika u záhazníka, který voní drahou kolínskou, a u zákazníka, ze kterého je silně cítit jablečné víno.

Příklad predikce na základě výsledků analýzy

Tento příklad je natolik komplexní a názorný, že jsme na něm dosud demonstrovali použití obou algoritmů implementovaných v analytickém serveru dodávaném s produktem Microsoft SQL Server 2000 a rovněž nutnost úpravy původního modelu. Na závěr příkladu nám ale zůstává jeho nejzajímavější část – ukážeme ještě příklad predikce nad databází hub. Jak jsme už uvedli, tato databáze obsahuje dvě tabulky, které vznikly rozdělením jedné větší tabulky na dvě části. Tabulku ***HoubyUcici*** už známe, na ní jsme prováděli analýzu. Predikci provedeme na tabulce ***HoubyTestovaci***, která má přesně stejnou strukturu a obsahuje druhou polovinu údajů z původní tabulky. Algoritmus pro rozdělení tabulky byl velmi jednoduchý. Záznamy se sudým ID byly přesunuty do jedné tabulky a záznamy s lichým ID do druhé. Jako nástroj použijeme **Data Mining Prediction Quey Task**, což je vlastně úloha v DTS. Pokud si prohlédneme obrázek aplikace **DTS Package**, je to poslední ikona v okně tasks (symbol kladiva).

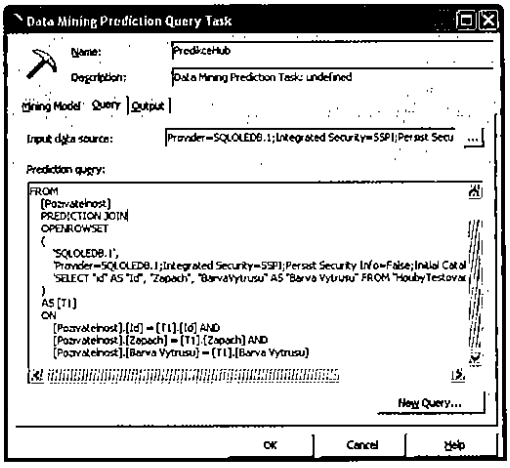


Obr. 8.42 DTS Package



Obr. 8.43 Prediction Query Task

Parametry pro tento nástroj nastavíme na třech záložkách. Na první z nich (Mining Model) se připojíme na příslušnou databázi analytického serveru. Na druhé záložce specifikujeme, kterou veličinu predikujeme na základě vstupních veličin.

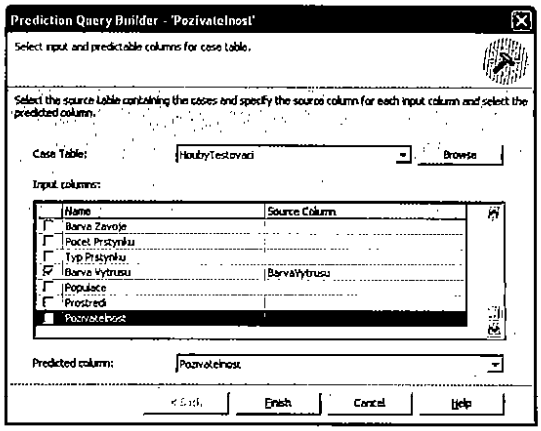


Obr. 8.44 Prediction Query Task

Nelekejme se poměrně složitého příkazu v tomto okně:

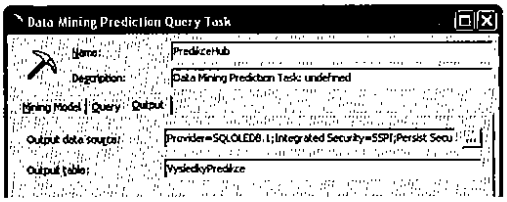
```
SELECT FLATTENED
    [T1].[Id], [T1].[Zapach], [T1].[Barva Vytrusu],
[Pozivatelnost].[Pozivatelnost]
FROM
    [Pozivatelnost]
    PREDICTION JOIN
    OPENROWSET
    (
        'SQLOLEDB.1',
        'Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security
Info=False;Initial Catalog=Houby;Data Source=LLHOME',
        'SELECT "id" AS "Id", "Zapach", "BarvaVytrusu" AS "Barva Vytrusu" FROM
"HoubyTestovaci" ORDER BY "id"'
    )
    AS [T1]
    ON
        [Pozivatelnost].[Id] = [T1].[Id] AND
        [Pozivatelnost].[Zapach] = [T1].[Zapach] AND
        [Pozivatelnost].[Barva Vytrusu] = [T1].[Barva Vytrusu]
```

Nemusíme ho totiž psát sami. Pod tlačítkem New Query se skrývá průvodce s názvem **Prediction Query Builder**. Zadáme tabulku, kterou budeme na základě vybraného modelu zkoumat, ***HoubyTestovaci***, vybereme příslušné sloupce a zbytek udělá tento nástroj sám. Víme, že hlavní vliv na konzumovatelnost hub má jejich zápach a barva výtrusu, proto budeme predikovat konzumovatelnost hub z druhé tabulky jen na základě těchto dvou vlastností.



Obr. 8.45 Prediction Query Builder

Na poslední záložce určíme, kam se uloží výsledky predikce. V našem příkladu jsme zvolili tabulku **VysledkyPredikce** v té samé databázi.



Obr. 8.46 Výsledná tabulka

Výsledkem naší predikce bude jednoduchá databázová tabulka (vypsaná příkazem `select * from VysledkyPredikce`):

```
Tl_Id   Tl_Zapach            Tl_Barva Vytrusu   Pozivatelnost_Pozivatelnost
------- -------------------- ------------------ ------------------------------
1       štiplavý, čpavý      černá              nejedlá, jedovatá
3       anýzový              hnědá              jedlá
5       žádný                hnědá              jedlá
7       po mandlích          černá              jedlá
9       štiplavý, čpavý      černá              nejedlá, jedovatá
```

Výsledky predikce jsou v posledním sloupci `Pozivatelnost_Pozivatelnost`.

Logicky nás zajímá, nakolik je tato predikce úspěšná. V tomto případě to můžeme jednoduše zjistit. Ve sloupci **Pozivatelnost** testovací databáze máme totiž uloženou informaci o tom, jestli daná houba ve skutečnosti je, nebo není konzumovatelná. Vytvoříme proto novou tabulku `PorovnaniPredikce`, která bude mít jen tři sloupce:

- ID
- Predikci (tedy předpověď provedenou pomocí Data Mining Prediction Task)
- Realitu (skutečný údaj o tom, zda daná houba je, nebo není konzumovatelná)

```
CREATE TABLE PorovnaniPredikce
(
        ID INT,
        Predikce VARCHAR (255),
        Realita VARCHAR (255)
);
```

Tabulku naplníme „sesypáním" údajů z obou tabulek:

```
INSERT INTO PorovnaniPredikce (ID, Predikce, Realita)
       SELECT   VysledkyPredikce.Tl_Id,
                VysledkyPredikce.Pozivatelnost_Pozivatelnost,
                HoubyTestovaci.Pozivatelnost
       FROM HoubyTestovaci, VysledkyPredikce
       WHERE VysledkyPredikce.Tl_Id = HoubyTestovaci.ID;
```

Výsledná tabulka bude potom obsahovat údaje v tomto tvaru:

```
ID          Predikce                          Realita
----------- --------------------------------- ------------------------------
1           nejedlá, jedovatá                 nejedlá, jedovatá
3           jedlá                             jedlá
5           jedlá                             jedlá
7           jedlá                             jedlá
9           nejedlá, jedovatá                 nejedlá, jedovatá
11          jedlá                             jedlá
13          jedlá                             jedlá
15          jedlá                             jedlá
17          jedlá                             jedlá
19          nejedlá, jedovatá                 nejedlá, jedovatá
21          jedlá                             jedlá
23          jedlá                             jedlá
25          jedlá                             jedlá
...
```

V prvních třinácti případech se tedy náš predikční mechanismus nezmýlil ani jednou. Samozřejmě, že nás zajímá, jak to dopadlo v celé tabulce, která obsahuje 4 062 záznamů. Abychom to zjistili, napíšeme jednoduchý skript v jazyce SQL, který bude obsahovat dva jednoduché příkazy SELECT:

```
SELECT count(*) AS [Celkovy pocet] FROM PorovnaniPredikce;

SELECT count(*) AS [Chybna predikce] FROM PorovnaniPredikce
WHERE Predikce <> Realita;
```

Výsledek potvrzuje téměř úplnou přesnost odhadů.

```
Celkovy pocet
-------------
4 062


Chybna predikce
---------------
23

(1 row(s) affected)
```

Výsledek, tedy v tomto případě procento správně predikovaných případů můžeme pomocí příkazu SQL zobrazit i procentuálně.

```
SELECT 100-(100* CONVERT(float, (SELECT COUNT(*) FROM PorovnaniPredikce
        WHERE Predikce <> Realita)) / CONVERT(float, COUNT(*))) AS Procento
FROM  PorovnaniPredikce;

Procento
------------------
99.43377646479567
```

Vidíme, že z celkového počtu 4 062 hub bylo nesprávně určených 23, což je jen zanedbatelných **0,56 %**. V tomto případě není však omyl jako omyl. Pokud necháme jedlou houbu, kterou algoritmus nesprávně určil jako jedovatou, v lese, nestane se nic. Opačný případ, kdy algoritmus určí jedovatou houbu jako jedlou, však může mít tragické následky. Ověřme si tedy, jaký je poměr uvedených případů, které mohou nastat při nesprávném určení houby:

```
SELECT count(*) AS [Jedovata urcena jako jedla] FROM PorovnaniPredikce
WHERE Predikce ='jedlá' AND Predikce <> Realita;

SELECT count(*) AS [Jedla urcena jako jedovata] FROM PorovnaniPredikce
WHERE Predikce ='nejedlá, jedovatá' AND Predikce <> Realita;
```

Výsledek nás v tomto případě jemně zamrazí.

```
Jedovata urcena jako jedla
--------------------------
23

Jedla urcena jako jedovata
--------------------------
0
```

Potvrzuje to platnost Murphyho zákonů (pokud se dá něco pokazit, provede se to důsledně). Určité riziko **0,56 %** tu zůstává. Vzpomeňme si, že náš model nebyl až tak přesný, jako by teoreticky mohl být. Na základě diagramu, který byl výsledkem data miningu, jsme věděli, že hlavní vliv na konzumovatelnost hub má jejich zápach a barva výtrusu, proto jsme predikovali jen na základě těchto dvou vlastností. Pokud budeme predikovat na základě všech vlastností, získáme jen **7** nesprávně určených hub, což představuje přesnost 99,82 %, což představuje pravděpodobnost chyby jen **0,17 %**. Přesnost se tedy zvýšila více než trojnásobně, ale pokud bychom vzali v úvahu absolutní čísla, byl i předchozí model úplně vyhovující. Bohužel i teď jsou nesprávně určené houby tím horším způsobem, tedy jedovaté jsou určené jako jedlé.

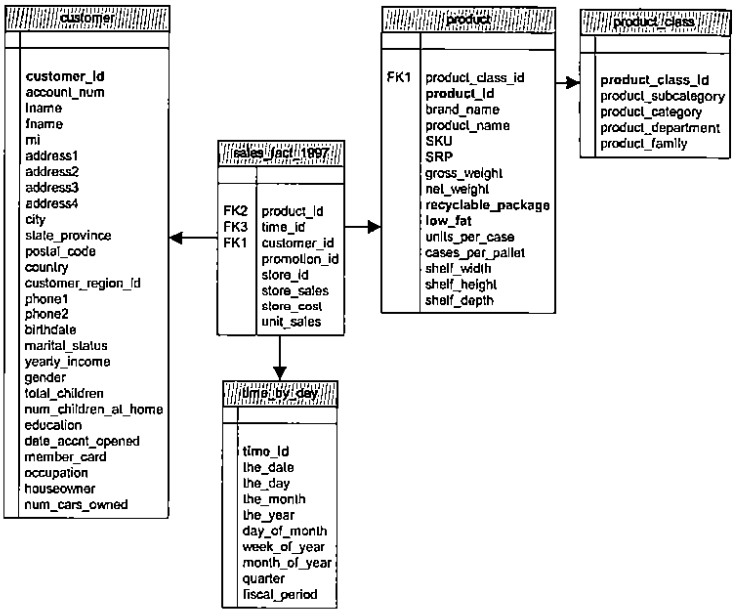
Pro zajímavost provedeme predikci i podle modelu vytvořeného na základě shlukového algoritmu. V tomto případě byl počet nesprávně určených hub 217, což odpovídá přesnosti „jen" **94,75 %**. Nepřesnost se v tomto případě rozděluje na obě strany, takže.

```
Jedovata urcena jako jedla
--------------------------
130

Jedla urcena jako jedovata
--------------------------
87
```

V tomto případě je tedy o hodně přesnější data miningový model založený na nevyvážených rozpadových stromech.

Příklad data miningu z cvičné databáze FoodMart

Předchozí příklad byl velmi vděčný z důvodu jednoduchosti a názornosti, ale ukážeme si i příklad s ekonomickým námětem. V tomto příkladu budeme vykonávat analýzu údajů z cvičné databáze FoodMart, která se dodává s MS SQL Serverem 2000.



Obr. 8.47 Struktura cvičné databáze Foodmart (zjednodušená)

Cvičná databáze FoodMart je dodávána jako soubor MDB (pro databázový program Microsoft Access), proto je výhodné ji přenést pomocí DTS do databáze pod správu databázového serveru Microsoft SQL Server 2000. Pro tuto cvičnou databázi jsou už v databázi analytického serveru připravené dva data miningové modely **Customer Pattern Discovery** a **Member card RDBMS**.

Příprava údajů

Tabulka customer obsahuje 10 281 záznamů. I tuto tabulku rozdělíme na učící a testovací množinu údajů. Tabulka ZakazniciUcici bude obsahovat záznamy se sudými indexy a tabulka ZakazniciTestovaci bude obsahovat záznamy s lichými indexy. Tento příklad si může vyzkoušet úplně každý, protože CD se 120denní verzí SQL Serveru 2000 je přibalené k publikaci.

Věta: „Rozdělíme tabulku na dvě tabulky podle sudých a lichých indexů." se hlavně začátečníkovi snadněji poví, než udělá. Navrhnout tuto transformaci pomocí grafického návrhového prostředí DTS není problém, ale abychom tento postup mohli realizovat i v jiných databázových platformách, ukážeme raději „manuální" postup pomocí příkazů jazyka SQL.

Rozdělení tabulky pomocí příkazů jazyka SQL.

Na základě analýzy budeme predikovat sloupec, který nám udává druh zákaznícky karty [member_card]. Tabulka se skládá z téměř třiceti sloupců, přičemž ze zkušenosti víme, že na to, jakou úvěrovou (zákaznickou) kartu zákazník vlastní, určitě nemá vliv poštovní směrovací číslo ani telefonní číslo a dokonce ani adresa, pokud ji nemáme zadanou tak, aby udávala typ čtvrti, ve které zákazník bydlí (například kontrast mezi Manhattanem a Bronxem). Takže největší vliv na to, jakou má zákazník kartu, mají hlavně tyto sloupce:

```
marital_status
yearly_income
gender
total_children
num_children_at_home
education
```

Proto v procesu úpravy údajů, který v tomto případě můžeme začlenit do přenosu údajů z databáze Access do databáze pod správou MS SQL Serveru vytvoříme tabulku Zakaznici Ucici.

```
CREATE TABLE ZakazniciUcici
(
        customer_id int NOT NULL ,
        marital_status nvarchar (1),
        yearly_income nvarchar (50),
        gender nvarchar (1),
        total_children smallint NULL,
        num_children_at_home smallint NULL ,
        education nvarchar (30),
        member_card nvarchar (50)
);
```

Stejnou strukturu bude mít i tabulka ZakazniciTestovaci.

```
CREATE TABLE ZakazniciTestovaci
(
        customer_id int NOT NULL ,
        marital_status nvarchar (1),
        yearly_income nvarchar (50),
        gender nvarchar (1),
        total_children smallint NULL,
        num_children_at_home smallint NULL ,
        education nvarchar (30),
        member_card nvarchar (50)
);
```

Abychom naplnili tabulku *ZakazniciUcici* záznamy se sudými indexy, použijeme příkaz:

```
INSERT INTO ZakazniciUcici
  SELECT customer_id ,marital_status, yearly_income, gender,total_children,
       num_children_at_home, education, member_card
  FROM customer WHERE (customer_id div % 2 = 0);
```

Pro naplnění tabulky *ZakazniciTestovaci* analogicky použijeme příkaz:

```
INSERT INTO ZakazniciTestovaci
  SELECT customer_id ,marital_status, yearly_income, gender,total_children,
       num_children_at_home, education, member_card
  FROM customer WHERE (customer_id div % 2 = 1);
```

O obsahu tabulky, například ZakazniciUcici se můžeme přesvědčit příkazem:

```
SELECT * from ZakazniciUcici ORDER BY customer_id;
```

customer_id	marital_	yearly_income	gender	total_children	num_childr	education	member_card
2	S	$70K - $90K	M	1	0	Partial High School	Bronze
4	M	$10K - $30K	M	4	4	Partial High School	Normal
6	S	$70K - $90K	F	3	0	Bachelors Degree	Bronze
8	M	$50K - $70K	M	2	2	Bachelors Degree	Bronze
10	S	$30K - $50K	M	4	0	Bachelors Degree	Golden
12	S	$30K - $50K	F	1	0	High School Degree	Bronze
14	M	$50K - $70K	F	2	2	Bachelors Degree	Bronze
16	M	$50K - $70K	F	2	2	High School Degree	Bronze
18	M	$30K - $50K	M	3	2	Partial College	Bronze
20	S	$10K - $30K	F	2	0	Partial High School	Normal

Návrh modelu

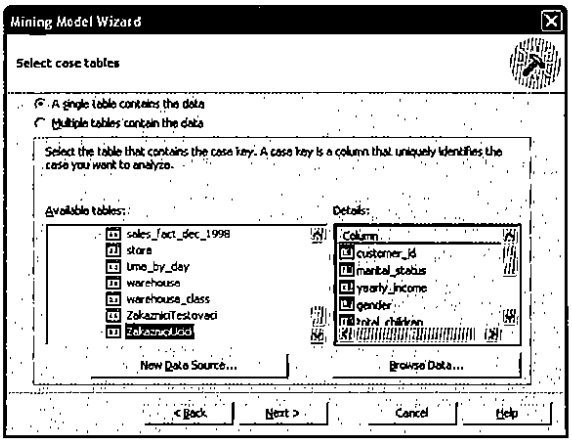
V naší analýze budeme zkoumat, jak závisí druh kreditní karty (zlatá, stříbrná...) na ročním příjmu jejího majitele, počtu dětí, rodinném stavu (ženatý, svobodný) a podobně, takže příklad velmi podobný už vytvořenému cvičnému modelu Member card RDBMS. Postup pro Analysis Manager už známe, proto jen stručně v bodech:

Z cvičné databáze FoodMart vybereme jako typ zdroje relační databázi a jako zdroj údajů tabulku **Customer**.

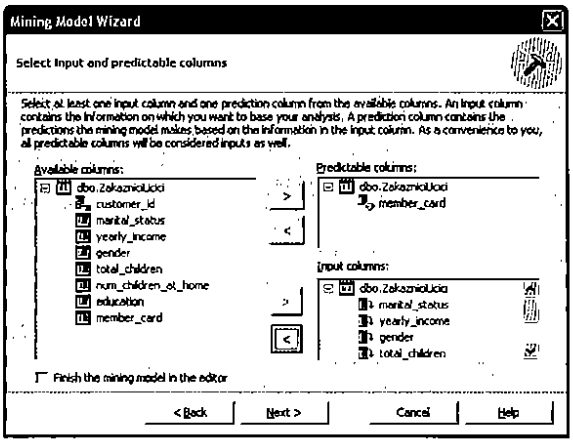
Vybereme typ algoritmu **Microsoft decision Trees** (Nevyvážený rozpadový strom) a unikátní identifikátor každého záznamu **customer_id**.

Predikujeme sloupec `member_card` na základě vstupních údajů:

```
marital_status
yearly_income
gender
total_children
num_children_at_home
education
```



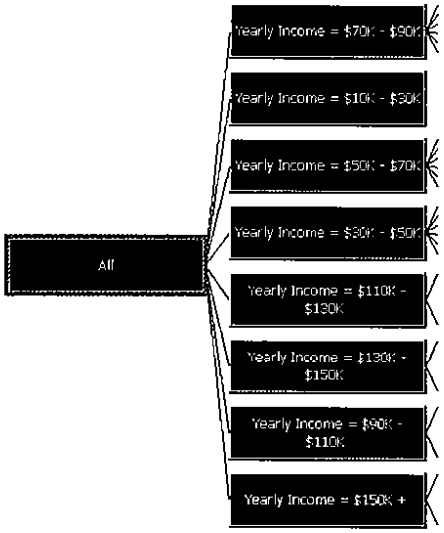
Obr. 8.48 Výběr tabulek



Obr. 8.49 Výběr vstupních a predikovaných údajů

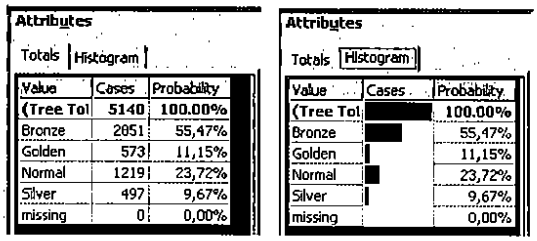
Výsledkem bude i v tomto případě přehledný diagram pro všechny hodnoty sloupce **member_card**, tedy pro zlaté, stříbrné, bronzové a normální karty a i pro případ, že zákazník

kartu nevlastní. Čím je barva políčka tmavší, tím je pravděpodobnost výskytu této vlastnosti u držitele příslušné karty vyšší. Samozřejmě byl jako určující kritérium správně vybrán roční příjem.



Obr. 8.50 Výsledek analýzy

Všimněme si i procentuálního rozložení jednotlivých druhů karet mezi zákazníky. Můžeme si zvolit buď formu výpisu počtu případů, nebo názornější formu histogramu.

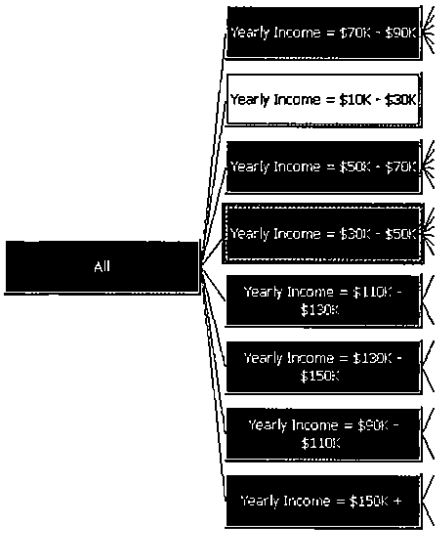


Obr. 8.51 Procentuální zastoupení jednotlivých druhů karet mezi zákazníky

O dost zajímavější budou výsledky analýzy pro jednotlivé druhy karet.

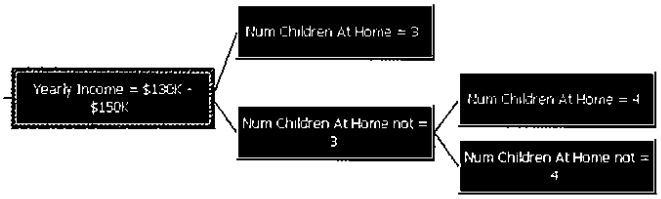
Zlaté karty

Nejvíc majitelů zlatých karet se rekrutuje ze skupiny zákazníků s ročním příjmem nad 150 tisíc USD. Zákazníci (kromě několika výjimek, kteří možná něco zdědili) s příjmem 10–30 tisíc USD na vydání takové karty pravděpodobně nemají ani nárok.



Obr. 8.52 Diagram pro zlaté karty

Jako hlavní faktor pro vydání příslušného druhu karty byl na základě vstupních údajů vyhodnocen roční příjem. Dalším rozhodujícím faktorem je počet vyživovaných dětí.



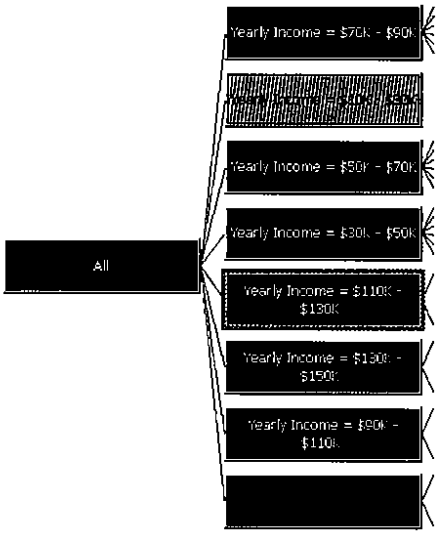
Obr. 8.53 Diagram pro zlaté karty, další faktory

Stříbrné karty

V tomto segmentu je situace prakticky stejná jako u zlatých karet. Nejvíc majitelů stříbrných karet se i zde rekrutuje ze skupiny zákazníků s ročním příjmem nad 150 tisíc USD. Zákazníci s příjmem 10–30 tisíc USD jsou i zde prakticky bez šance.

Bronzové karty

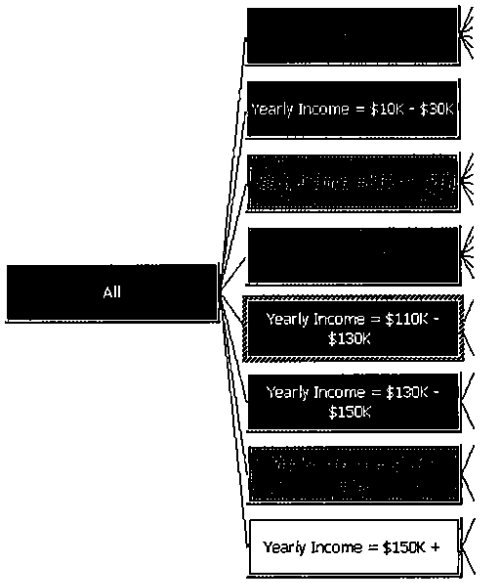
V tomto segmentu je situace úplně opačná. Ke slovu se dostávají ostatní příjmové skupiny s výjimkou zákazníků s příjmem 10–30 tisíc USD. Zákazníci s ročním příjmem nad 150 tisíc USD na druhé straně o takové karty už zájem nemají.



Obr. 8.54 Diagram pro bronzové karty

Standardní karty

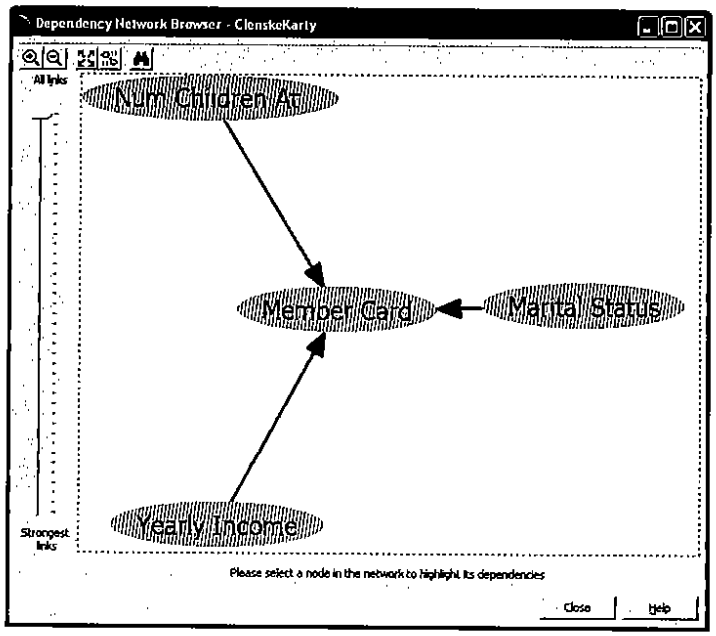
Tyto karty jsou prakticky doménou zákazníků s příjmem 10–30 tisíc USD.



Obr. 8.55 Diagram pro standardní karty

Vyzkoušejme ještě sílu vlivu jednotlivých atributů pomocí **Dependency Network Browser**. To, že na druh karty má největší vliv roční příjem, jsme určitě věděli, na to datamining nepotřebujeme, ale co ostatní vlivy?

V našem případě závisí druh kreditní karty na ročním příjmu, počtu dětí v domácnosti a na tom, zda je zákazník (zákaznice) svobodný nebo ve „stavu manželském".



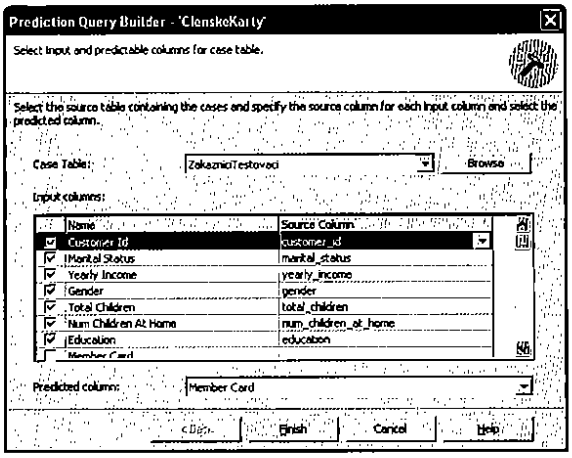
Obr. 8.56 Vliv atributů

Predikce na základě výsledků analýzy

I v tomto případě bude velmi zajímavá predikce, kterou aplikujeme na tabulku Zakaznici-Testovaci. Jako nástroj i v tomto případě použijeme **Data Mining Prediction Quey Task**, jako úloha v DTS.

Na základě námi zadaných údajů průvodce vygeneruje příkaz pro predikci.

```
SELECT FLATTENED
    [T1].[Customer Id], [T1].[Marital Status], [T1].[Yearly Income],
[T1].[Gender], [T1].[Total Children], [T1].[Num Children At Home],
[T1].[Education], [ClenskeKarty].[Member Card]
FROM
    [ClenskeKarty]
    PREDICTION JOIN
    OPENROWSET
    (
```



Obr. 8.57 Výběr vstupních sloupců a predikovaného sloupce

```
        'SQLOLEDB.1',
        'Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security
Info=False;Initial Catalog=Supermarket;Data Source=LLHOME',
        'SELECT "customer_id" AS "Customer Id", "marital_status" AS "Marital
Status", "yearly_income" AS "Yearly Income", "gender" AS "Gender",
"total_children" AS "Total Children", "num_children_at_home" AS "Num Children At
Home", "education" AS "Education" FROM "ZakazniciTestovaci" ORDER BY "customer_id"'
    )
    AS [T1]
    ON
        [ClenskeKarty].[Customer Id] = [T1].[Customer Id] AND
        [ClenskeKarty].[Marital Status] = [T1].[Marital Status] AND
        [ClenskeKarty].[Yearly Income] = [T1].[Yearly Income] AND
        [ClenskeKarty].[Gender] = [T1].[Gender] AND
        [ClenskeKarty].[Total Children] = [T1].[Total Children] AND
        [ClenskeKarty].[Num Children At Home] = [T1].[Num Children At Home] AND
        [ClenskeKarty].[Education] = [T1].[Education]
```

Výsledek predikce bude uložen do tabulky PredikceZakazniku.

```
Id   Marital Yearly Income Gender Total Child Num Children Education           Member Card
---- ------- ------------- ------ ----------- ------------ ------------------- -----------
1    M       $30K - $50K   F      4           2            Partial High School Bronze
3    M       $50K - $70K   F      1           1            Bachelors Degree    Bronze
```

```
5   S       $30K - $50K   F      3          0          Partial College      Bronze
7   M       $30K - $50K   F      2          1          Partial High School  Bronze
9   M       $10K - $30K   M      5          3          Partial High School  Normal
11  S       $50K - $70K   M      4          0          High School Degree   Bronze
13  S       $30K - $50K   M      4          0          High School Degree   Bronze
15  S       $90K - $110K  M      3          0          Graduate Degree      Bronze
17  S       $90K - $110K  M      3          0          High School Degree   Bronze
19  S       $70K - $90K   F      1          0          High School Degree   Bronze
21  S       $30K - $50K   M      1          0          High School Degree   Bronze
23  S       $30K - $50K   F      2          0          High School Degree   Bronze
```

I v tomto případě nás bude zajímat úspěšnost predikce. Vytvoříme proto novou tabulku `PorovnaniPredikce`, která bude mít tyto sloupce:

- ID
- Predikci (tedy předpověď provedenou pomocí Data Mining Prediction Task)
- Realitu (skutečný údaj o tom, jakou kreditní kartu má daný zákazník)

```
CREATE TABLE PorovnaniPredikce
(
        ID INT,
        Predikce VARCHAR (25),
        Realita VARCHAR (25)
);
```

Tabulku naplníme „sesypáním" údajů z obou tabulek:

```
INSERT INTO PorovnaniPredikce (ID, Predikce, Realita)
       SELECT   [PredikceZakazniku].[T1_Customer Id],
                [PredikceZakazniku].[ClenskeKarty_Member Card],
                ZakazniciTestovaci.[member_card]
       FROM ZakazniciTestovaci, [PredikceZakazniku]
       WHERE [PredikceZakazniku].[T1_Customer Id] = ZakazniciTestovaci.customer_ID;
```

Výsledná tabulka bude potom obsahovat údaje v tomto tvaru:

```
ID          Predikce                 Realita
----------- ------------------------ ------------------------
1           Bronze                   Bronze
3           Bronze                   Bronze
5           Bronze                   Silver
7           Bronze                   Bronze
9           Normal                   Normal
11          Bronze                   Bronze
13          Bronze                   Bronze
15          Bronze                   Bronze
17          Bronze                   Bronze
19          Bronze                   Bronze
21          Bronze                   Normal
23          Bronze                   Silver
```

```
25          Bronze                  Bronze
27          Bronze                  Bronze
29          Normal                  Normal
31          Golden                  Golden
33          Bronze                  Bronze
35          Bronze                  Bronze
37          Bronze                  Bronze
39          Bronze                  Bronze
41          Bronze                  Bronze
```

V prvních dvaceti případech se náš predikční mechanismus spletl třikrát. Nebude to tedy zřejmě téměř absolutní přesnost predikce jako v předchozím příkladu (vždyť v tomto případě nejde o život, ale jen o byznys). Samozřejmě, že i v tomto případě nás zajímá, jak to dopadlo v celé tabulce, která obsahuje 5 141 záznamů.

```
SELECT count(*) AS [Celkovy pocet] FROM PorovnaniPredikce;

SELECT count(*) AS [Chybna predikce] FROM PorovnaniPredikce
WHERE Predikce <> Realita;

SELECT 100* CONVERT(float, (SELECT COUNT(*) FROM PorovnaniPredikce WHERE
                            predikce <>Realita)) / CONVERT(float, COUNT(*)) AS
Percento
FROM PorovnaniPredikce;
```

Výsledek je v souhrnu celkem uspokojivý.

```
Celkovy pocet
-------------
5141

(1 row(s) affected)

Chybna predikce
---------------
914

(1 row(s) affected)

Procento
----------------------------------------------------------
17.778642287492705
```

Téměř osmnáct procent bylo určeno chybně, ale **82,2 %** bylo predikováno správně.

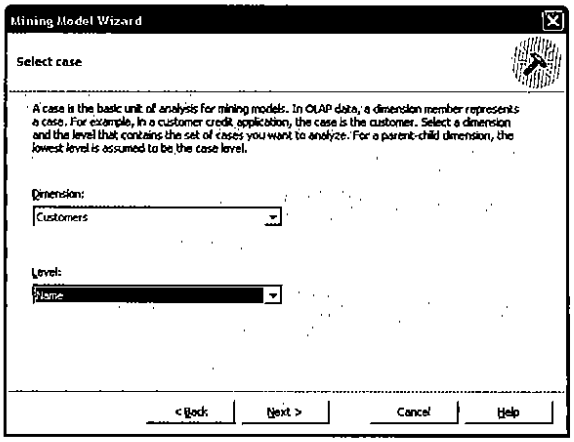
Příklad Data Miningu z databáze OLAP

V dosavadních příkladech jsme aplikovali Data Mining na údaje v transakčních, tedy relačních databázích. Zajímavé závislosti však skrývají i analytické databáze. Proto ukážeme příklad aplikování Data Miningu na analytickou databázi. Použijeme cvičnou analytickou databázi FoodMart 2000.

Námět dataminingu v obchodních řetězcích bývá téměř vždy stejný. Přímočaře bychom to řekli jednoduchou holou větou: „Zvýšit obrat." Pokud za tuto větu přidáme vykřičník, dostaneme typický námět mnoha porad manažerů. Cíl je jasný, ale potřebujeme prostředky k jeho dosažení. Jednou z možností zvýšení obratu je zvýšení spokojenosti zákazníků. Každému je jasné, že zvýšením cen spokojnosti zákazníků nedosáhneme. Co však zákazníky přitahuje, jsou různé kreditní úvěrové a nákupní karty. Proto i Data Mining zaměříme na tento cíl. Zajímají nás různé souvislosti, například geografické, demografické (rodinný stav, roční příjem...).

Data Miningový model budeme aplikovat na krychli Sales. V nástroji Analysis Manager rozvineme v levém okně hierarchickou strukturu analytické databáze FoodMart 2000. Pravým tlačítkem klepneme na ikonu krychle OLAP Sales a v kontextové nabídce vybereme volbu „New Mining Model". Zahájíme tím činnost průvodce „Mining Model Wizard".

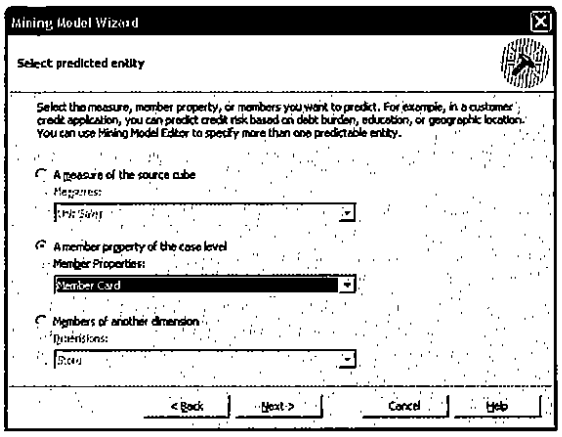
V prvním kroku průvodce vybereme algoritmus „Microsoft Decision Trees". Tento dialog je ještě shodný s dialogem průvodce pro vytvoření Data Miningového modelu z relační databáze. V dalším kroku průvodce se už postup liší, následuje totiž výběr dimenze pro analyzovaný případ.



Obr. 8.58 Mining Model Wizard – Dialog Select case

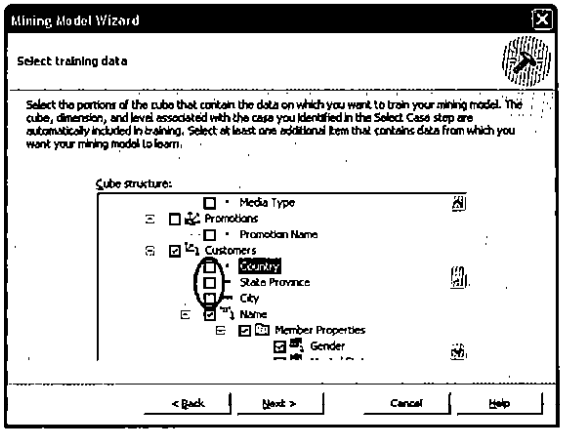
V našem případě jsou předmětem zájmu zákazníci, takže vybereme dimenzi „Customers" a úroveň „Name".

Predikovanou entitou bude „Member Card".



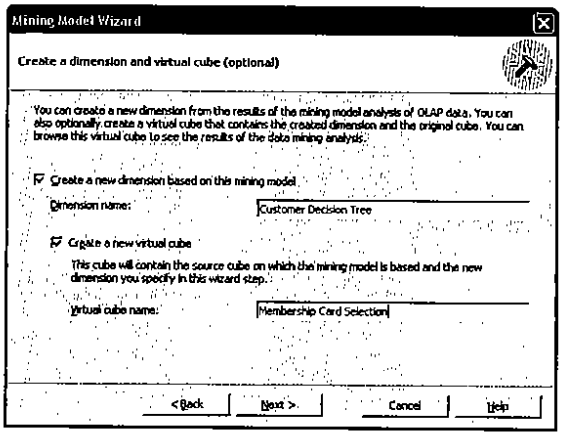
Obr. 8.59 Mining Model Wizard – dialog pro výběr predikované entity

Následující krok logicky vyplývá z průniku teorie OLAPu a Data Miningu. Pro Data Mining potřebujeme jen „případy" na nejnižší úrovni dimenze. Sumarizované údaje jsou pro Data Mining totiž bezcenné. Proto zrušíme zaškrtnutí polí vyšších úrovní dimenze „Customers", tedy „Country", „State Province" a „City".



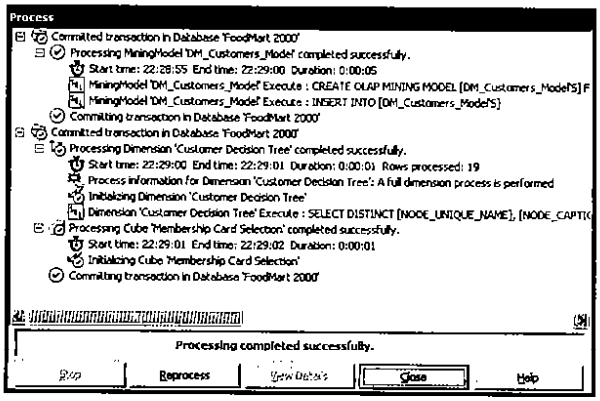
Obr. 8.60 Mining Model Wizard – dialog pro výběr úrovní dimenze

V dalším dialogu vytvoříme novou dimenzi na základě právě vytvářeného data miningového modelu s názvem například „Customer Decision Tree". Nová dimenze bude aplikována ve virtuální krychli s názvem „Membership Card Selection".

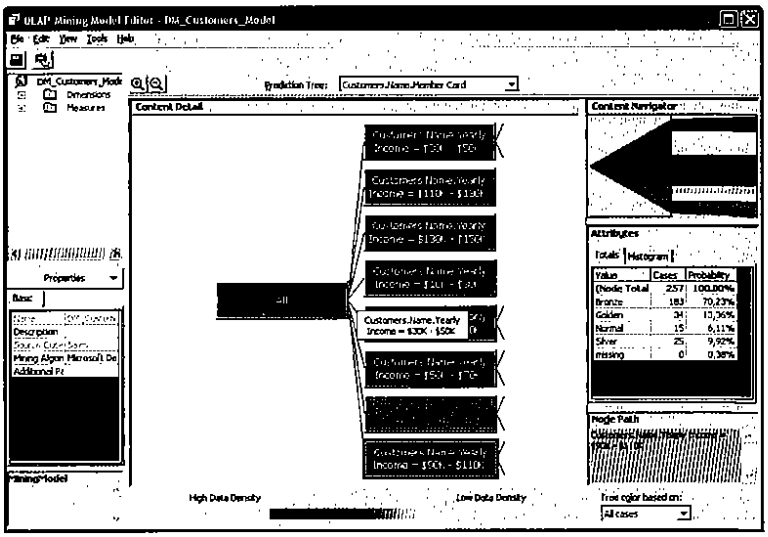


Obr. 8.61 Mining Model Wizard – dialog Select case

V posledním dialogu průvodce vytvářený data miningový model pojmenujeme, například „DM_Customers_Model". Ponecháme zaškrtnutou volbu „Process now", takže po stisknutí tlačítka „Next" se spustí výpočet data miningového modelu. Jeho průběh je zobrazen v dialogu „Process".



Obr. 8.62 Průběh vytváření modelu



Obr. 8.63 Výsledky data miningu

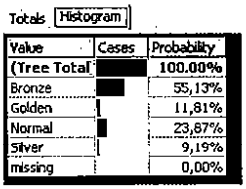
Pokud klepneme na obdélník ALL, který reprezentuje všechny případy, zjistíme, že ve sledovaném období (záleží na krychli, v případě krychle SALES se jednalo o rok 1998) vykonalo transakce 5 581 zákazníků. Pokud chceme zjistit, kolik zákazníků máme celkem zahrnutých v úrovni Name dimenze Customers, stačí položit dotaz SQL v transakční databázi na tabulku CUSTOMER.

```
SELECT COUNT(*) FROM customer;

10281
```

V roce 1998 tedy vykonalo nějaké transakce jen více než polovina evidovaných zákazníků. V tabulce nebo v histogramu vidíme zastoupení jednotlivých druhů zákaznických karet.

Totals | Histogram

Value	Cases	Probability
(Tree Total	5581	100,00%
Bronze	3077	55,13%
Golden	659	11,81%
Normal	1332	23,87%
Silver	513	9,19%
missing	0	0,00%

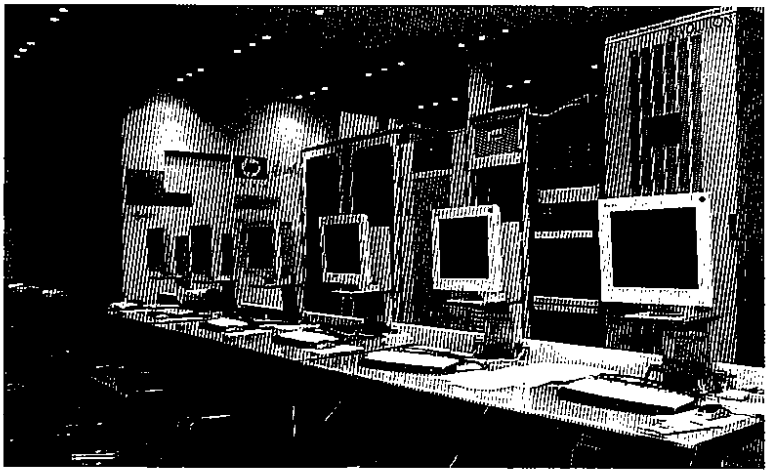


Obr. 8.64 Výsledky analýzy pro všechny případy (ve formě tabulky i histogramu)

Kapitola 9

Příklad řešení dvouterabajtového datového skladu

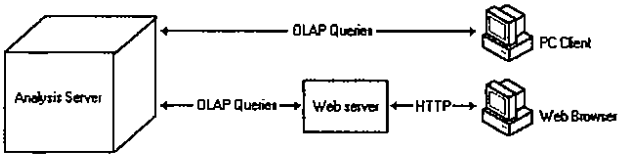
Jako příklad implementace reálného datového skladu představíme technické řešení dvouterabajtového datového skladu, se kterým se mnozí čtenáři, zejména z České republiky, měli možnost seznámit na mezinárodní výstavě Invex 2002. Cílem řešení bylo prezentovat technologické možnosti databáze Microsoft SQL Server a jejích analytických nástrojů a rovněž možnosti serverů firmy Hewlett-Packard. Projekt vznikl z iniciativy společnosti Microsoft za spolupráce firem Dataware a Hewlett-Packard. Námětem řešení bylo vytvořit datový sklad, který by věrně imitoval reálnou aplikaci obchodu z oblasti retail. Důraz byl kladen nejen na serverová řešení, ale i na přístup z klientských aplikací, a to i na mobilních zařízeních. Potom není problém zobrazit výsledek analýzy nad dvouterabajtovým datovým skladem na zařízení třídy PocketPC.



Obr. 9.1 Realizace dvouterabajtového datového skladu na Invexu

Jako databázový stroj pro uložení 2TB údajů byl použit MS SQL Server 2000 ve verzi Enterprise Edition s nainstalovaným servisním balíčkem Service Pack 2. Databázový server běžel na operačním systému Windows 2000 Datacenter Server.

Pro vytváření a spravování krychlí OLAP byly použity dva servery Microsoft Analysis Services (service pack 2) na operačním systému Windows 2000 Datacenter Server. Zhruba třetina údajů v krychlích OLAP byla vytvořena jako remote partition.



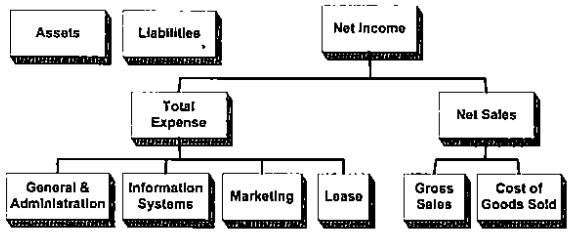
Obr. 9.2 Schéma dvouterabajtového datového skladu

Zdrojové údaje

Pro vytvoření databáze byly použity fiktivní údaje obchodního řetězce, který prodával 2 300 produktů 3 000 zákazníků na 12 obchodních místech po dobu 12 let. Pro všechny tabulky dimenzí byla použita reálně vypadající jména zákazníků, názvy produktů i obchodních míst. Pro názvy produktů, jejich kategorizaci a názvy obchodních míst byly použity reálné údaje od společnosti Sony. Aby bylo zachováno zdání reality, byly algoritmy pro generování údajů vytvořené tak, že prodej postupně, ale kolísavě rok od roku roste. Algoritmy zohledňují i poměrné zastoupení prodeje v jednotlivých segmentech, takže pokud vytvoříme souhrny a agregace a ty následně zobrazíme například ve formě koláčových grafů, budou tyto grafy dokonce působit hodnověrně. Údaje (fakta) za uvedených dvanáct let byly uložené ve 141 databázových tabulkách (12 x 12 = 144, takže tři měsíce z posledního roku chybí, Invex bývá na podzim...). Každá tabulka obsahuje 240 milionů záznamů, což dohromady představuje úctyhodných 33,8 miliardy záznamů.

Struktura databáze

Kromě měsíčních tabulek faktů, které obsahují záznamy o obchodování, obsahuje datový sklad i tabulky pro vytvoření dimenzí **Produkty, Zákazník, Prodejny** a **Datum**.



Obr. 9.3 Struktura tabulek datového skladu

Dimenze „**Produkty**" je tvořena 2 hierarchiemi o 2 úrovních – Skupina, název (resp. Part Number).

Dimenze „**Prodejny**" je tvořena 3 úrovněmi – Kraj, město, sklad.

Dimenze „**Datum**" je tvořena 4 úrovněmi – Rok, kvartál, měsíc, den.

Dimenze „**Zákazník**" je tvořena 3 úrovněmi – Kraj, město, zákazník.

Typ dimenze Zákazník byl nastaven na „Geography" a jednotlivé úrovně dimenze měly nastavený parametr Level type na region (level kraj), country (level okres) a city (level město). Toto nastavení se potom dá s výhodou použít v klientském nástroji „NovaView" od společnosti Panorama Software, pro napojení výstupů z krychle OLAP na Microsoft MapPoint (zobrazování grafů s hodnotami z krychle přímo v mapě).

Produktová dimenze

Produktová dimenze je vytvořena nad tabulkou produktů. Tabulka PRODUKTY má sedm sloupců.

Název	Datový typ
PRODUKT_ID	int
SORTIMENT	varchar(100)
VYROBCE	varchar (100)
TYP_VYROBKU	varchar(100)
NAZEV	varchar(100)
CENA	int

Příkaz jazyka SQL pro vytvoření tabulky produktů je

```
CREATE TABLE PRODUKTY

(
        PRODUKT_ID int IDENTITY (1, 1) NOT NULL ,
        SORTIMENT varchar (100) COLLATE Czech_CI_AS NULL ,
        VYROBCE varchar (100) COLLATE Czech_CI_AS NULL ,
        TYP_VYROBKU varchar (100) COLLATE Czech_CI_AS NULL ,
        NAZEV varchar (100) COLLATE Czech_CI_AS NULL ,
        CENA int NULL
)
```

Tabulka obsahuje 5 000 záznamů.

	PRODUKT_ID	SORTIMENT	VYROBCE	TYP_VYROBKU	NAZEV	CENA
1	1	Baterie	VARTA	Super	tužka	5
2	2	Baterie	VARTA	Super	plochá	20
3	3	Baterie	VARTA	Super	malé mono	8
4	4	Baterie	VARTA	Super	velké mono	11
5	5	Baterie	VARTA	Super	9V blok	20
6	6	Baterie	VARTA	Longlife	tužka	6
7	7	Baterie	VARTA	Longlife	válcová	27
8	8	Baterie	VARTA	Longlife	plochá	34
9	9	Baterie	VARTA	Longlife	malé mono	10
10	10	Baterie	VARTA	Longlife	velké mono	17
11	11	Baterie	VARTA	Longlife	9V blok	29

Obr. 9.4 Příklad údajů uložených v tabulce PRODUKTY

Dimenze Produkty má 2 úrovně a 2 hierarchie:

Hierarchie Produkty.název

Level	Members
Skupina	6
Název	2300

Hierarchie Produkty.PartNo

Level	Members
Skupina	6
Part Number	2300

Zákaznická dimenze

Tato dimenze je vybudována nad tabulkou Odberatele. Tabulka má 12 sloupců

Název sloupce	Datový typ
ODBERATEL_ID	int
NAZEV	varchar(100)
SKUPINA	varchar(2)
PLOCHA	int
ZAMESTNANCI	int
ZAMEST_KAT	varchar(50)
PLOCHA_KAT	varchar(50)
STAT	varchar(50)
KRAJ	varchar(50)
OKRES	varchar(50)
PSC	varchar(40)
MESTO	varchar(50)

Příkaz jazyka SQL pro vytvoření tabulky odběratelů bude:

```
CREATE TABLE ODBERATELE
(
        ODBERATEL_ID int IDENTITY (1, 1) NOT NULL ,
        NAZEV varchar (100) COLLATE Czech_CI_AS NULL ,
        SKUPINA varchar (2) COLLATE Czech_CI_AS NULL ,
        PLOCHA int NULL ,
        ZAMESTNANCI int NULL ,
        ZAMEST_KAT varchar (50) COLLATE Czech_CI_AS NULL ,
        PLOCHA_KAT varchar (50) COLLATE Czech_CI_AS NULL ,
        STAT varchar (50) COLLATE Czech_CI_AS NULL ,
        KRAJ varchar (50) COLLATE Czech_CI_AS NULL ,
        OKRES varchar (50) COLLATE Czech_CI_AS NULL ,
        PSC varchar (40) COLLATE Czech_CI_AS NULL ,
        MESTO varchar (50) COLLATE Czech_CI_AS NULL
)
```

Tabulka obsahuje 3 000 záznamů.

	ODBERATEL_ID	NAZEV	SKUPINA	PLOCHA	ZAMEST...	ZAMEST_KAT	PLOCHA_KAT	STAT	KRAJ
1	1	2112 ELECTRONICS spol. ...	B	132	18	11-20	101-150m2	ČR	Praha
2	2	3C SERVIS	A	132	15	11-20	101-150m2	ČR	Pardubický
3	3	3M ČESKO spol. s r. o.	A	127	19	11-20	101-150m2	ČR	Praha
4	4	A - ELEKTRO ŠTĚPBA	C	44	7	06-10	000-050m2	ČR	Zlínský
5	5	A.A. ELEKTRA sdružení HO...	B	60	11	11-20	051-100m2	ČR	Budějovický
6	6	A.M.D.	B	60	7	06-10	051-100m2	ČR	Praha
7	7	A.P. SERVIS s.r.o.	A	80	8	06-10	051-100m2	ČR	Pardubický
8	8	A+B ELEKTRO	B	134	14	11-20	101-150m2	ČR	Praha
9	9	AB CONTROL spol.s r.o.	A	128	10	06-10	101-150m2	ČR	Brněnský
10	10	AB ELMONT	A	71	8	06-10	051-100m2	ČR	Ústecký

Obr. 9.5 Příklad údajů uložených v tabulce ODBERATELE

Dimenze Zákazník má tři úrovně.

Level	Members
Kraj	Počet krajů
Město	157
Zákazník	3 000

Dimenze prodejny

Dimenze prodejny byla vytvořena nad databázovou tabulkou SKLADY. Tabulka má 7 sloupců.

Název	Datový typ
SKLAD_ID	int
NAZEV	varcha(100)
STAT	varchar(50)
KRAJ	varchar(50)
OKRES	varchar(50)
MESTO	varchar(50)
PSC	varchar(50)

Příkaz jazyka SQL pro vytvoření tabulky skladů bude:

```
CREATE TABLE SKLADY
(
        SKLAD_ID int IDENTITY (1, 1) NOT NULL ,
        NAZEV varchar (100) COLLATE Czech_CI_AS NULL ,
        STAT varchar (50) COLLATE Czech_CI_AS NULL ,
        KRAJ varchar (50) COLLATE Czech_CI_AS NULL ,
        OKRES varchar (50) COLLATE Czech_CI_AS NULL ,
        MESTO varchar (50) COLLATE Czech_CI_AS NULL ,
        PSC varchar (50) COLLATE Czech_CI_AS NULL
)
```

Tabulka obsahuje 1 000 záznamů.

	SKLAD_ID	NAZEV	STAT	KRAJ	OKRES	MESTO	PSC
1	1	AGRONORD, a. s.	ČR	Ústecký	Litoměřice	Roudnice nad Labem	41301
2	2	ANDRES MILAN	ČR	Ústecký	Litoměřice	Roudnice nad Labem	41301
3	3	ASTALOŠ PETR	ČR	Liberecký	Česká Lípa	Nový Bor	47301
4	4	AUDIO	ČR	Ústecký	Chomutov	Kadaň	43201
5	5	AUTO LAUDA DOPRAVA	ČR	Liberecký	Česká Lípa	Nový Bor	47301
6	6	AUTOBAZAR MERTLÍK	ČR	Ústecký	Chomutov	Kadaň	43201
7	7	AUTOBAZAR MM	ČR	Liberecký	Česká Lípa	Nový Bor	47301
8	8	AUTOCENTRUM - AUTOBAZAR	ČR	Liberecký	Česká Lípa	Nový Bor	47301
9	9	AUTOCENTRUM DORDA	ČR	Liberecký	Česká Lípa	Nový Bor	47301
10	10	AUTOCENTRUM PECINKA	ČR	Ústecký	Litoměřice	Roudnice nad Labem	41301

Obr. 9.6 Příklad údajů uložených v tabulce SKLADY

Dimenze Prodejny má tři úrovně:

Název	Členství
Kraj	10
Město	11
Sklad	12

Časová dimenze

Dimenze prodejny byla vytvořena nad databázovou tabulkou DATUM. Tabulka má dva sloupce.

Název	Datový typ
DATUM_ID	int
DATUM	smalldatetime

Příkaz jazyka SQL pro vytvoření tabulky skladů bude:

```
CREATE TABLE [DATUM]
(
        DATUM_ID int IDENTITY (1, 1) NOT NULL,
        DATUM smalldatetime NULL
)
```

Tabulka obsahuje 2 844 záznamů.

	DATUM_ID	DATUM
1	1	2001-10-14 00:00:00
2	2	2001-10-13 00:00:00
3	3	2001-10-12 00:00:00
4	4	2001-10-11 00:00:00
5	5	2001-10-10 00:00:00
6	6	2001-10-09 00:00:00
7	7	2001-10-08 00:00:00
8	8	2001-10-07 00:00:00
9	9	2001-10-06 00:00:00
10	10	2001-10-05 00:00:00
11	11	2001-10-04 00:00:00

Obr. 9.7 Příklad údajů uložených v tabulce DATUM

Zavedení údajů do datového skladu

Po ukončení generování datových souborů jsme se vlastně dostali do výchozí situace, kdy máme údaje získané z firemní činnosti a začínáme budovat datový sklad. Tyto údaje potřebujeme zavést do tabulek faktů. Tabulky faktů měly názvy PRODEJ_rok_mesic, tedy například PRODEJ_2001_1.

Tabulka má 14 sloupců.

Sloupec	Datový typ
ODBERATEL_ID	int
PRODUKT_ID	int
SKLAD_ID	int
DATUM_ID_PRODEJ	int
DATUM_ID_SPLATNOST	int
DATUM_ID_OBJEDNANO	int
DATUM_ID_ZAPLACENO	int
KUSY	int
N_CENA	int
P_CENA	int
POCET_PRODEJU	int
SPLATNOST	int
ZAPLACENO	int
OBJEDNANO	int

Příkaz jazyka SQL pro její vytvoření je:

```
CREATE TABLE PRODEJ_2001_1
(
        N_COUNT int,
       ODBERATEL_ID int NULL ,
        PRODUKT_ID int NULL ,
        SKLAD_ID int NULL ,
        DATUM_ID_PRODEJ int NULL ,
        DATUM_ID_SPLATNOST int NULL ,
        DATUM_ID_OBJEDNANO int NULL ,
        DATUM_ID_ZAPLACENO int NULL ,
        KUSY int NULL ,
        N_CENA int NULL ,
        P_CENA int NULL ,
        POCET_PRODEJU int NULL ,
        SPLATNOST int NULL ,
        ZAPLACENO int NULL ,
        OBJEDNANO int NULL
)
```

Vzhledem k formátu textového souboru, ve kterém je pořadové číslo záznamu, byl i do tabulek faktů přidán sloupec `N_COUNT int`.

Import údajů z textových souborů do tabulky faktů se prováděl pomocí příkazu jazyka T-SQL BULK INSERT.

```
BULK INSERT prodej_2001_1
FROM 'c:\dataware\data_2001_1_1.txt'
WITH (FIELDTERMINATOR = ':', ROWTERMINATOR = '\n', ROWS_PER_BATCH = 1000000, TABLOCK)
```

Naplnění jednoho souboru trvalo zhruba 20 minut a běželo současně s generováním souborů. Import údajů za jeden měsíc tak trval přibližně tři hodiny.

	N_COUNT	ODBERATEL_ID	PRODUKT_ID	SKLAD_ID	DATUM_ID_PRODEJ	DATUM_ID_SPLATNOST	DATUM_ID_OBJEDNANO
1	1	1721	1757	651	286	300	279
2	2	225	2506	541	274	288	268
3	3	509	992	117	266	266	261
4	4	2395	2747	384	261	275	256
5	5	1903	142	462	267	281	260
6	6	1721	17	789	266	296	261
7	7	1079	2684	586	279	293	274
8	8	1843	1104	167	287	301	286
9	9	1098	4366	434	282	312	275
10	10	2940	2431	779	270	360	268
11	11	508	1229	106	270	284	263

Obr. 9.8 Příklad údajů uložených v tabulce faktů

Vygenerování údajů pro naplnění datového skladu

Údaje byly generovány pomocí aplikace napsané v programovacím jazyce Visual Basic. Aplikace GenerovaniDat.exe generovala textové soubory ve tvaru:

```
1;1721;332;354;277;277;272;278;1;278;306;1;0;1;5
2;1991;1111;244;265;295;261;298;1;17053;17906;1;30;3;4
3;2340;4289;819;287;301;284;306;1;1600;1920;1;14;5;3
4;1459;1375;86;282;372;279;384;3;3900;4485;1;90;12;3
5;434;4699;165;258;318;253;592;2;19622;21192;1;60;274;5
6;943;4494;492;257;271;254;328;1;5099;5813;1;14;57;3
7;535;207;289;270;270;267;274;2;3340;3841;1;0;4;3
8;1763;4653;95;258;318;257;321;3;168066;201679;1;60;3;1
...
```

Je to klasický flat-soubor, kde každý řádek představuje jeden záznam. Jednotlivé položky v rámci záznamu jsou oddělené středníkem (;). Význam jednotlivých položek je v tabulce.

	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11	K12	K13	K14	K15
1	1721	332	354	277	277	272	278	1	278	306	1	0	1	5
2	1991	1111	244	265	295	261	298	1	17053	17906	1	30	3	4
3	2340	4289	819	287	301	284	306	1	1600	1920	1	14	5	3
4	1459	1375	86	282	372	279	384	3	3900	4485	1	90	12	3
5	434	4699	165	258	318	253	592	2	19622	21192	1	60	274	5
6	943	4494	492	257	271	254	328	1	5099	5813	1	14	57	3
7	535	207	289	270	270	267	274	2	3340	3841	1	0	4	3
8	1763	4653	95	258	318	257	321	3	168066	201679	1	60	3	1

Význam jednotlivých názvů sloupců je:

Číslo sloupce	Význam	Název sloupce v tabulce
(1)	pořadové číslo záznamu	
(2)	odběratel	ODBERATEL_ID
(3)	produkt	PRODUKT_ID
(4)	sklad	SKLAD_ID
(5)	datum prodeje	DATUM_ID_PRODEJ
(6)	datum splatnosti	DATUM_ID_SPLATNOST
(7)	datum objednávky	DATUM_ID_OBJEDNANO
(8)	datum úhrady	DATUM_ID_ZAPLACENO
(9)	počet kusů	KUSY
(10)	cena	N_CENA
(11)	cena	P_CENA
(12)	počet prodejů	POCET_PRODEJU
(13)	splatnost	SPLATNOST
(14)	zaplaceno	ZAPLACENO
(15)	objednáno	OBJEDNANO

Pro každý měsíc se generovalo dohromady 8 souborů. Vygenerování jednoho souboru trvalo přibližně 40 minut. Z toho vyplývá, že při takovém tempu se údaje za jeden měsíc generovaly přibližně 6 hodin. Server, na kterém se generovaly údaje, měl 8 procesorů, takže generování běželo paralelně v osmi procesech a za 6 hodin bylo možno vygenerovat údaje pro osm měsíců. Jednoduchou matematikou zjistíme, že pro vygenerování 141 měsíců se spotřebovalo pro 18 cyklů čistého času 108 hodin, tedy přibližně čtyři a půl dne. Samozřejmě celý proces obsluhovali a řídili lidé, takže byl celkový čas okolo 6 dní.

Vstupní parametry pro program **GenerovaniDat.exe** jsou:

-log c:\log.txt ... název souboru protokolu pro záznam událostí

-od 1 ... od kterého měsíce generovat údaje

-do 1 ... do kterého měsíce generovat údaje

-rok 2001 ... pro který rok

-cesta c:\ ... kam ukládat vygenerované soubory

-pocet 1000 ... počet záznamů v jednom souboru

Například :

```
GenerovaniDat.exe -rok 2001 -od 1 -do 2 -log c:\log.txt -pocet 10000 -cesta c:\dataware\
```

O každém vygenerovaném souboru je v protokolu (v našem případě `c:\log.txt`) záznam ve tvaru:

```
...
akce : generovani datovych souboru
zacatek : 08.03.2003 17:12:37
```

```
konec : 08.03.2003 17:12:37
rozdil : 00:00:00
zaznamu : 1250
soubor : data_2002_1_1.txt
...
```

Vytvoření analytické databáze

Vytvořená krychle obsahovala 141 oddílů. Jednotlivé oddíly se vytvářely VB skriptem pomocí knihovny DSO (Decision Support Objects). Řez oddílů byl nastavený na 1 měsíc (141 prodejních měsíců představovalo 141 oddílů krychle). Výsledná velikost krychle byla po vytvoření všech agregací přibližně 320 GB v primárních oddílech a v remote oddílech dalších 140 GB. Dohromady tedy krychle měla přibližně 460 GB.

Krychle byla vytvořena jako MOLAP, přičemž při jejím prvním výpočtu byly nastaveny agregace na 80 % výkonu. Oddíly se zpracovávaly paralelně ve čtyřech procesech.

Následně byly vytvořeny další agregace na základě nastaveného parametru „Usage-Based optimization". Nové agregace se vytvářely jen pro dotazy, které trvaly více než 10 sekund.

Krychle obsahovala tyto ukazatele:

- Kusy – počet prodaných kusů.
- Nákupní ceny – nákupní ceny výrobků.
- Prodejní ceny – prodejní ceny výrobků.
- Počet prodejů.

V krychli nebyly použity žádné vypočítávané ukazatele.

Výpočet krychlí běžel paralelně na šesti procesech. Celkový čas na vytvoření 2TB datového skladu byl 3 týdny.

Klientské přístupy k analytické databázi

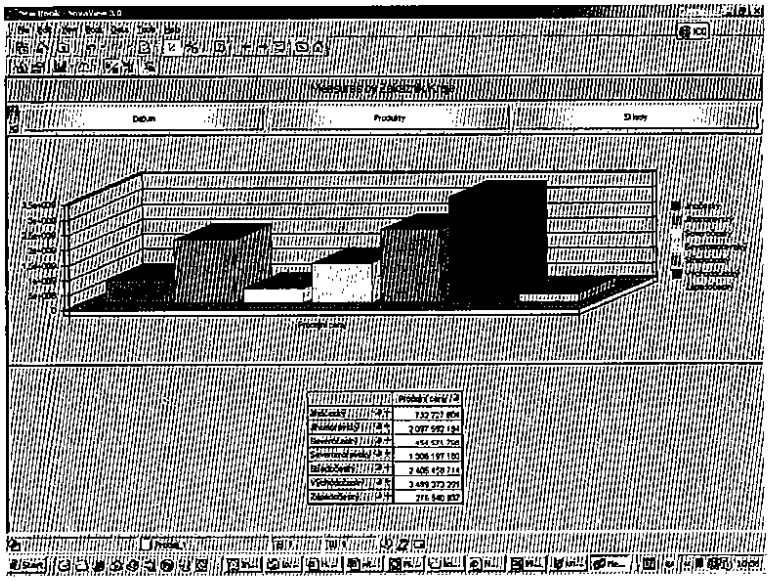
Řešení předpokládalo klientský přístup z různých typů klientských zařízení. Pro PC klientů byly použity nástroje Microsoftu a izraelské firmy Panorama Software. Jedním z cílů řešení bylo ukázat, že 2 TB údajů nejsou problém ani pro mobilní zařízení. Samozřejmě jako tenké klienty. Proto byly vytvořeny i uživatelské aplikace pro PocketPC / MDA.

Microsoft Office Web Component

Sada uživatelských pohledů byla vytvořena v Microsoft Office Web Component (Office XP). Z pivot table existuje jednoduchý export údajů do programu MS Excel, přičemž se zachová připojení na Analysis Services.

Panorama Software

Nástroje e-BI a Nova View firmy Panorama Software poskytovaly další sadu uživatelských pohledů. Tyto nástroje poskytují svým uživatelům velmi rychlou odezvu při přístupu k 2 TB v OLAP.



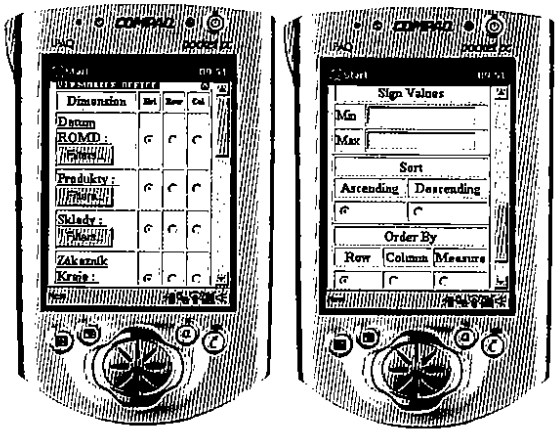
Obr. 9.9 Aplikace Nova View

Aplikace HTML

Klient s „čistým" HTML byl připravený pro ukázku faktu, že i s minimálními technologickými nároky je možno vytvořit kvalitního klienta pro analýzu 2 TB údajů v OLAP.

VIP Mobile Office

Mobilní klient pro platformu Pocket PC byl originálním řešením firmy Dataware pro online přístup k údajům OLAP. Řešení umožňuje vytvářet sady předpřipravených tabulí a grafů. Přímo z mobilního zařízení je možno zadávat ad-hoc dotazy, filtry, provádět rozpad drill-down a další funkce. Pro online připojení k serveru datového skladu je možno použít připojení GPRS nebo GSM. V podnikovém prostředí se používá zpravidla Wireless LAN.



Obr. 9.10 VIP Mobile Office

Testy výkonnosti řešení

Účelem testů bylo zjistit dobu odezvy analytického serveru při zpracovaní více dotazů MDX současně a modelovat tak reálný provoz. Testovací dotazy MDX byly generovány automaticky a do databáze byl zapisován čas před spuštěním každého dotazu MDX a po jeho ukončení. Dimenze v řádcích a sloupcích (produkty, zákazníci, prodejny) byly rozložené do náhodné hloubky úrovně. Test byl vykonáván současným přístupem z více klientských počítačů. Dohromady bylo ve 4 testech vygenerováno 3 156 dotazů.

Test 1 – Na 4 PC byl spuštěn skript, který postupně spustil 9 náhodných dotazů MDX.

Test 2 – Na 10 PC byl spuštěn skript, který postupně spustil 22 náhodných dotazů MDX.

Test 3 – Na 20 PC byl spuštěn skript, který postupně spustil 66 náhodných dotazů MDX.

Test 4 – Na 20 PC byl spuštěn skript, který postupně spustil 79 náhodných dotazů MDX.

Výsledky testů

Test 4 (20 PC po 3 dotazech)	
Průměrná doba odezvy	4,6 sec
Maximální doba odezvy	2:01
Minimální doba odezvy	0 sec

Test 1: 40 PC po 12 dotazech	
Průměrná doba odezvy	4,3 sec
Maximální doba odezvy	1:52 min
Minimální doba odezvy	0 sec

Test 3: 120 PC po 100 dotazech	
Průměrná doba odezvy	2,9 sec
Maximální doba odezvy	2:32 min
Minimální doba odezvy	0 sec

Test 4: 250 PC po 75 dotazech	
Průměrná doba odezvy	2,1 sec
Maximální doba odezvy	2:19 min
Minimální doba odezvy	0 sec

Na první pohled je zřejmá určitá nelogičnost – čím více dotazů, tím kratší doba odezvy. Může to být způsobeno i tím, že pro zhruba 20 procent uměle vygenerovaných dotazů není v analytické databázi žádný výsledek. Firma X totiž nekoupila v daném období zboží Y. Se zvyšujícím se počtem dotazů úměrně stoupá i počet nulových průniků, kdežto náročných dotazů je poměrně málo. Proto byly v druhém pokusu údaje vyčištěné od nulových průniků, tj. byly vygenerovány jen takové dotazy MDX, které vrátily vždy neprázdný průnik dat.

Test 1a: 40 PC po 9 dotazech	
Průměrná doba odezvy	4,9 sec
Maximální doba odezvy	2:01 min
Minimální doba odezvy	0 sec

Test 2a: 120 PC po 22 dotazech	
Průměrná doba odezvy	5,1 sec
Maximální doba odezvy	1:52 min
Minimální doba odezvy	0 sec

Test 3a: 120 PC po 100 dotazech	
Průměrná doba odezvy	7,4 sec
Maximální doba odezvy	2:32 min
Minimální doba odezvy	0 sec

Test A: 128 paralelních dotazů	
Průměrná doba odezvy	7.6 sec
Maximální doba odezvy	2:19 min
Minimální doba odezvy	0 sec

Minimální doba odezvy signalizuje, že se některé dotazy generovaly ze serveru cache. Server byl před testem totiž týden v provozu. Výsledky testů jsou velmi optimistické a dá se předpokládat, že v reálném provozu by byly tyto výsledky ještě lepší. Uživatelé totiž často spouštějí podobné typy dotazů.

Příklad identického řešení na lokálním počítači

Na první pohled se nadpis této statě bude zdát mírně nesmyslný, vždyť realizace dvouterabajtového datového skladu na lokálním vývojářském počítači, tedy desktopu nebo notebooku nedává smysl. Ale kapacita dnešních disků optimálních z hlediska cena – výkon se pohybuje už v hranicích 100 až 200 gigabajtů, tedy 0,1 až 0,2 terabajtu. A taktovací frekvence procesorů v rozmezí 2 až 4 GHz poskytují v této oblasti netušené možnosti. Pokud omezíme množství údajů, můžeme toto řešení vyzkoušet na libovolném dobře vybaveném PC. To svědčí nejen o vynikající flexibilitě, ale i o horizontální a vertikální škálovatelnosti nejen konkrétního řešení, ale všeobecně i technologií datových skladů a analytických databází.

Konkrétní popis a návod na realizaci uvádíme ze stejného důvodu, kvůli kterému toto vystavované prototypové řešení vzniklo. Možnost zkusit to v praxi závisí na hardwarových kapacitách a na tom, pro jaké řešení se z hlediska škálovatelnosti rozhodneme. Pokud si vygenerujeme méně údajů, uložíme je do menšího skladu, pokud vygenerujeme údajů víc, budou se nároky na hardwarovou realizaci zvyšovat. Další výhodou popisovaného řešení je skutečnost, že simulované je jen generování údajů. Po jejich vygenerování jsme sice v trochu idealizované, ale realitě se blížící situaci. Málokdy jsou v praxi totiž všechny údaje uložené v souborech flat stejného formátu.

Vytvoření databáze datového skladu

V předchozí statí jsou popsané struktury tabulky faktů a dimenzí. Relační databáze Dataware, ve které jsou uložené, je pod správou Serveru SQL. Tabulky pro vytvoření dimenzí **Produkty**, **Zákazník**, **Prodejny** a **Datum** obsahují jen po několika tisících údajích, proto tyto tabulky nijak nezatíží úložnou kapacitu databáze. Na CD je soubor se zálohou databáze, kde jsou už tyto tabulky vytvořené a naplněné. Proto pomocí funkce Restore obnovíme takto zazálohované údaje do databáze s názvem Dataware. Záloha se nachází na CD v souboru DW.BACKUP.

Postup obnovy databáze ze zálohy

Prvním krokem je vytvoření nové databáze s názvem „Dataware". Novou databázi vytvoříme pomocí aplikace SQL Server Enterprise Manager v nabídce *Databases – New Database*. Zkušení administrátoři samozřejmě v tomto kroku mohou blíže specifikovat parametry vytvářené databáze, například takto:

1. Data files

- *automatically grow file*
- *file grow in megabytes (40MB)*
- *10MB space allocated při vytvoření databáze*

2. Transaction log

- *automatically grow file*
- *file grow in megabytes (10 MB)*
- *maximum file size unrestricted*

3. Options

- *simple recovery model*

Následně je potřeba přenést do nově vytvořené databáze tabulky dimenzí ze zálohy. V nabídce *Databases – All Tasks – Restore Database* aktivujeme dialog pro obnovu databáze ze zálohy. Jako název obnovované databáze zadáme „Dataware" a zaškrtneme položku „From device".

	N_COUNT	ODBERATEL_ID	PRODUKT_ID	SKLAD_ID	DATUM_ID_PRODEJ	DATUM_ID_SPLATNOST	DATUM_ID_OBJEDNANO
1	1	1721	1757	651	286	300	279
2	2	225	2506	541	274	288	268
3	3	509	992	117	266	266	261
4	4	2395	2747	384	261	275	256
5	5	1903	142	462	267	281	260
6	6	1721	17	789	266	295	261
7	7	1379	2684	586	279	293	274
8	8	1843	1104	167	287	301	286
9	9	1098	4366	434	282	312	275
10	10	2940	2431	779	270	360	268
11	11	508	1229	106	270	284	263

Obr. 9.11 Obnovení databáze ze zálohy

Musíme dát pozor, aby nám v záložce data seděly fyzické názvy databázových souborů skutečné databáze Dataware na konkrétním počítači. Logické názvy souborů jsou T1Prodej_Data a T1Prodej_Log. Fyzické názvy souborů, které byly v našem řešení,

T1Prodej_Data	*E:\MSSQL2000_XP\MSSQL\Data\Dataware_Data.MDF*
T1Prodej_Log	*E:\MSSQL2000_XP\MSSQL\Data\Dataware_Log.LDF*

je potřeba změnit podle konkrétní realizace, například při implicitní instalaci SQL Serveru na:

T1Prodej_Data	*C:\Program Files\Microsoft SQL Server\MSSQL\Data\Dataware_Data.MDF*
T1Prodej_Log	*C:\Program Files\Microsoft SQL Server\MSSQL\Data\Dataware_Log.LDF*

Vygenerování údajů o obchodování

Po obnovení databáze ze zálohy máme vytvořené a naplněné tabulky dimenzí. Tabulky faktů jsou vytvořené pro všech 12 měsíců roku 2002 a jsou zatím prázdné.

Pro jejich naplnění použijeme aplikaci **GenerovaniDat.exe**. Kolik údajů vygenerujeme, závisí na plánované velikosti a kapacitě datového skladu. Každý musí zvážit své požadavky a hardwarové možnosti. Vygenerujeme například 10 milionů záznamů pro prvních šest měsíců roku 2002 příkazem:

```
GenerovaniDat.exe -rok 2002 -od 1 -do 6 -log c:\log.txt -pocet 10000000 -cesta c:\data\
```

Význam parametrů je následující:

-rok 2001 ... pro který rok

-od 1 ... od kterého měsíce se mají generovat údaje

-do 1 ... do kterého měsíce se mají generovat údaje

-log c:\log.txt ... název souboru protokolu pro záznam událostí

-pocet 10000000 ... počet záznamů v jednom souboru

-cesta c:\ ... kam ukládat vygenerované soubory

Vygenerování souborů pro jeden měsíc na PC s 2GHz Pentiem 4 trvalo přibližně 15 minut a údaje zabraly na pevném disku 559 megabajtů.

```
Directory of C:\data

12.03.2003  20:04        68 890 589 data_2002_1_1.txt
12.03.2003  20:06        69 994 243 data_2002_1_2.txt
12.03.2003  20:08        69 992 768 data_2002_1_3.txt
12.03.2003  20:09        69 986 131 data_2002_1_4.txt
12.03.2003  20:11        69 986 129 data_2002_1_5.txt
12.03.2003  20:13        69 977 697 data_2002_1_6.txt
12.03.2003  20:14        69 983 000 data_2002_1_7.txt
12.03.2003  20:16        69 986 481 data_2002_1_8.txt
               8 File(s)    558 797 038 bytes
```

Celkem bylo pro šest měsíců vygenerováno 3,3 gigabajtu údajů. Pro zajímavost, pomocí kompresního algoritmu ZIP se dají vygenerované soubory flat zkomprimovat asi 7x.

O generování údajů se vytváří dva druhy protokolů. Jeden do souboru (`c:\log.txt`).

```
akce : generovani datovych souboru
zacatek : 12.03.2003 20:47:35
konec : 12.03.2003 20:48:46
rozdil : 00:01:11
zaznamu : 1250000
soubor : data_2002_1_1.txt

...
...

akce : generovani datovych souboru
zacatek : 12.03.2003 21:46:24
```

```
konec : 12.03.2003 21:47:37
rozdil : 00:01:13
zaznamu : 1250000
soubor : data_2002_6_8.txt
```

Druhý do databázové tabulky LOG v databázi Dataware. V našem výpisu jsme ukázali jen jeho první a poslední blok. Celkově bylo při generování údajů za šest měsíců 48, jelikož se pro každý měsíc generuje vždy osm datových souborů.

Tabulka LOG se podobně jako tabulky dimenzí obnoví ze zálohy na CD. Struktura tabulky je zřejmá z příkazu pro její vytvoření, generování hodnot data a času probíhá podle systémového času PC v daném okamžiku.

```
CREATE TABLE LOG
(
        LOG_ID int IDENTITY (1, 1) NOT NULL ,
        MSG varchar (6000),
        ACTIONTYPE varchar (100),
        MESIC int NULL ,
        ROK int NULL ,
        SOUBOR int NULL ,
        STARTED bit NULL CONSTRAINT DEFAULT (1),
        FINISHED bit NULL DEFAULT (0),
        STARTTIME datetime NULL DEFAULT (getdate()),
        ENDTIME datetime NULL ,
        CREATED datetime NULL DEFAULT (getdate())
)
```

Zavedení údajů do datového skladu

Pro zavedení údajů ze souborů flat do databáze datového skladu byla použita tato sekvence příkazů:

```
BULK INSERT prodej_2002_1
FROM 'c:\data\data_2002_1_1.txt'
WITH (FIELDTERMINATOR  = ';', ROWTERMINATOR  = '\n', ROWS_PER_BATCH = 1000000, TABLOCK)
```

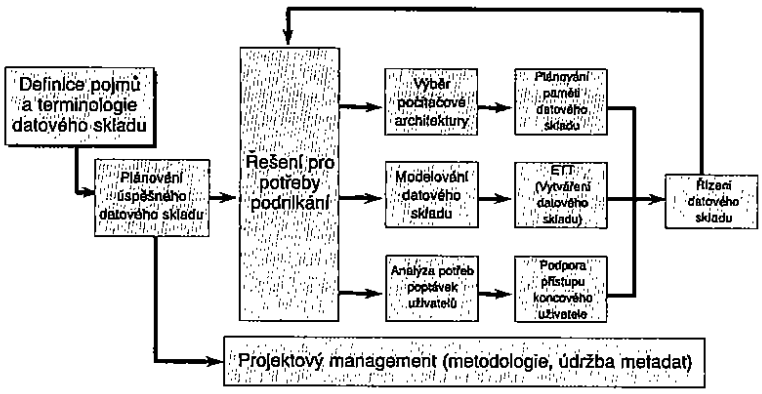
Zavedení jednoho, zhruba 70megabajtového souboru trvalo zhruba 15 sekund. Po zavedení všech datových souborů do databáze datového skladu můžeme přejít k vytváření krychlí OLAP. Postup je popsaný ve čtvrté kapitole a struktura datového skladu v předchozí stati.

Kapitola 10

Oracle

Pro budování datových skladů existují různé návody a metodiky. Nejnáročnějším aspektem vývoje datového skladu není technická obtížnost, ale výběr metody budování datového skladu, která nejvíc vyhovuje potřebám konkrétní firmy. V minulosti se oddělení IT pokoušela zabezpečit celopodnikovou implementaci datového skladu pomocí jediného projektu. Tato metoda se nazývá „Metoda velkého třesku". Vývoj datového skladu je však náročná úloha, ale samotný vývoj nesmí trvat neúměrně dlouho, protože se pak technologie a požadavky uživatele změní ještě před dokončením projektu.

V této kapitole si stručně představíme metodiku, kterou vypracovali specialisté firmy Oracle. Hlavním důvodem je její nezávislost na platformě. Při vypracování této metodiky měli její autoři samozřejmě na zřeteli použití nástrojů pro budování datových skladů od firmy Oracle, například OWB (Oracle Warehouse Building), DISCOVERER, EXPRESS a podobně. Klidně ale můžeme tuto metodiku plnohodnotně využít, i když používáme nástroje jiných firem. Princip budování datového skladu totiž není závislý na platformě. Celou metodiku můžeme shrnout do jednoho blokového schématu:



Obr. 10.1 Metodika Oracle pro budování datových skladů

Integrace těchto postupů přímo do databázového serveru právě takové paralelní zpracování umožňuje.

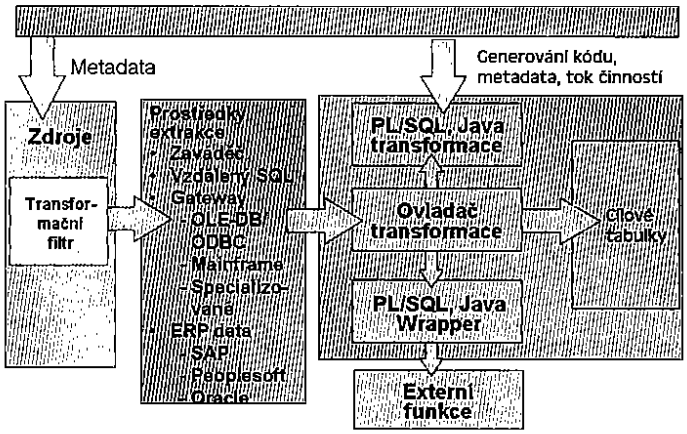
Výhody jsou zřejmé:

- postačí jeden databázový stroj,
- údaje jsou integrované,
- celý proces může být prakticky libovolně škálovatelný.

Kromě zrychlení a zefektivnění celého postupu analýz a dataminingu vystupuje do popředí i značná úspora nákladů, která vyplývá z nižšího počtu potřebných (poměrně drahých) softwarových produktů a nižšího počtu potřebných specialistů.

Praktický příklad ETL pomocí nástroje Oracle OWB (Oracle Warehouse Builder)

Pro přípravu a zavedení údajů se používá komplexní nástroj Oracle Warehouse Builder. Jeho architektura je na obrázku.



Obr. 10.4 Architektura Oracle Warehouse Builder

Jako námět pro praktickou ukázku použijeme stejný příklad jako v třetí kapitole. Budeme transformovat databázi hub získanou z knihovny katedry univerzity, konkrétně *Department of Information and Computer Science University of California, Irvine CA*.

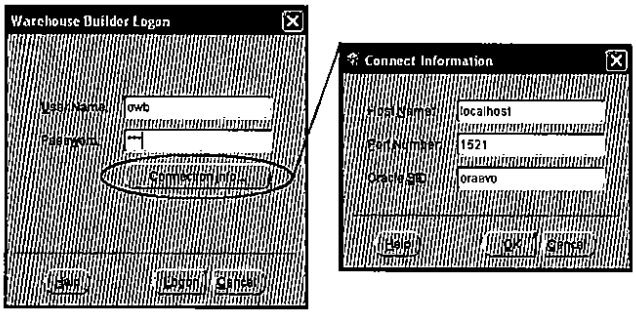
Zdrojová databáze se skládá ze dvou souborů **agaricus_lepiota.data** a **agaricus_lepiota.names**.

Soubor **agaricus_lepiota.data** je klasickým příkladem flat-souboru, kde jsou jednotlivé atributy oddělené čárkami a jednotlivé záznamy jsou oddělené přechodem na nový řádek. Soubor tedy obsahuje údaje ve tvaru:

```
p,x,s,n,t,p,f,c,n,k,e,e,s,s,w,w,p,w,o,p,k,s,u
e,x,s,y,t,a,f,c,b,k,e,c,s,s,w,w,p,w,o,p,n,n,g
e,b,s,w,t,l,f,c,b,n,e,c,s,s,w,w,p,w,o,p,n,n,m
p,x,y,w,t,p,f,c,n,n,e,e,s,s,w,w,p,w,o,p,k,s,u
...
```

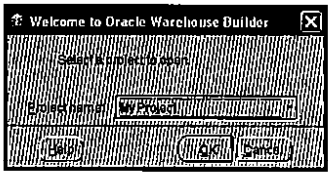
Legenda k takto uspořádanému flat-souboru je potom v souboru **agaricus_lepiota.names.** Oba dva soubory jsou umístěné v adresáři Houby.

Transformaci cvičné databáze uskutečníme pomocí aplikace OWB Client. Při spuštění této aplikace se přihlásíme pomocí přístupových parametrů, které zadal správce při instalaci produktu Oracle Warehouse Builder. Při prvním přihlášení musíme zadat i parametry pro připojení se k databázi. Dialog pro nastavení těchto parametrů (na obrázku vpravo) aktivujeme tlačítkem *Connection Info...*



Obr. 10.5 Přihlášení se k aplikaci OWB Client

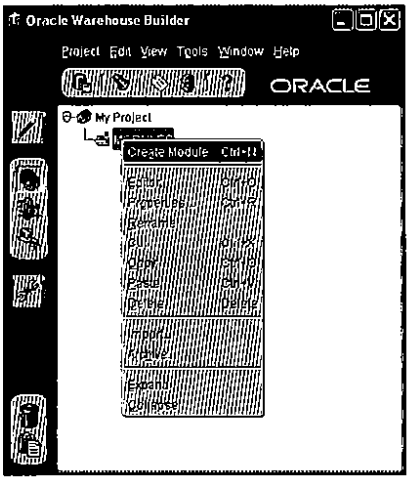
Prvním krokem, který musíme v aplikaci OWB učinit, je výběr projektu. Pro náš cvičný příklad můžeme otevřít projekt *My Project*.



Obr. 10.6 OWB – výběr projektu

Po výběru projektu se otevře hlavní pracovní obrazovka aplikace OWB Client, která obsahuje hierarchickou strukturu modulů. Složka Modules je zatím prázdná. Postupně v ní vytvoříme dva moduly. Jeden se bude týkat zdroje údajů a druhý cílové databáze, kam údaje po transformaci uložíme.

Pokud vyvoláme pro tuto složku kontextovou nabídku (pravé tlačítko myši ve Windows), můžeme volbou položky Create Module vytvořit nový modul. Modul vytvoříme pomocí průvodce vytvořením modulu (New Module Wizard).



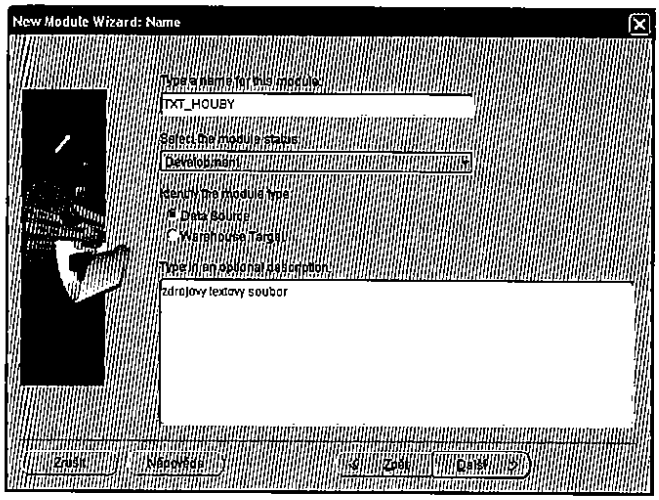
Obr. 10.7 OWB – vytvoření nového modulu

Definování modulu pro zdroj údajů

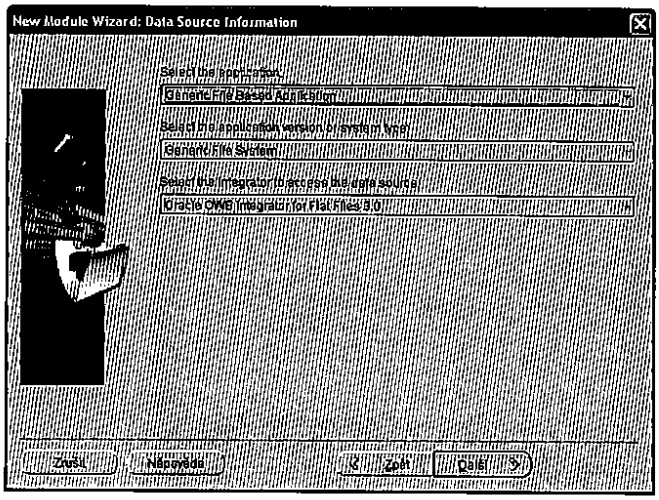
Po úvodním dialogu průvodce se zobrazí první „pracovní" dialog průvodce, ve kterém zadáme název nového modulu (v našem případě TXT_HOUBY) a označíme ho za zdrojový.

V následujícím dialogu upřesníme zdroj údajů. Jako zdroj údajů můžeme zvolit databázi (*Generic Oracle Database Application*) nebo soubory (*Generic File Based Application*). V našem případě zvolíme jako zdroj údajů soubory. Po nastavení této volby se v dalších políčkách automaticky nastaví *Generic File system* a *Oracle OWB Integrator for Flat Files 3.0*.

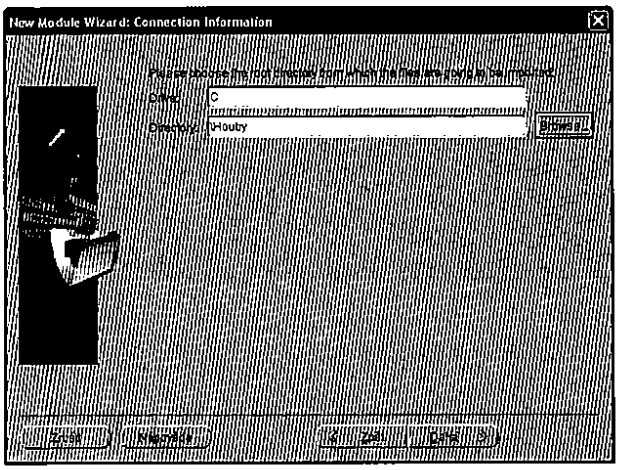
V případě souboru je volba v dialogu Connection Information jednoduchá. Zadáme diskovou jednotku a adresář, ve kterém se zdrojový soubor nachází.



Obr. 10.8 OWB – zadání názvu zdrojového modulu

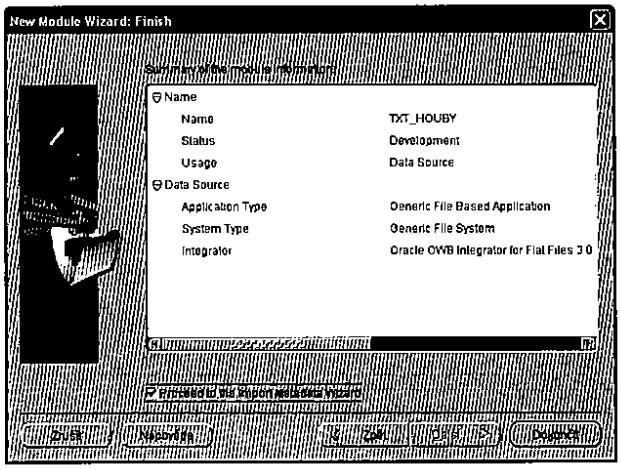


Obr. 10.9 OWB – nastavení informací o typu zdroje údajů



Obr. 10.10 OWB – zadání názvu zdrojového modulu

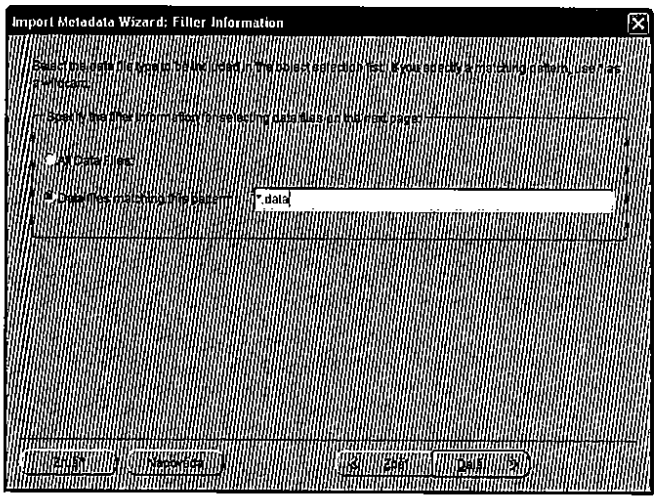
V konečném dialogu průvodce vytvořením nového modulu se zobrazí souhrnné informace o zdroji údajů. V tomto dialogu zaškrtneme volbu Proceed to the Import Metadata Wizard, která aktivuje průvodce importem vstupních metadat. Pokud tuto volbu ponecháme nezaškrtnutou, budeme samozřejmě moci vstupní metadata importovat později.



Obr. 10.11 OWB – finální dialog průvodce vytvořením nového modulu

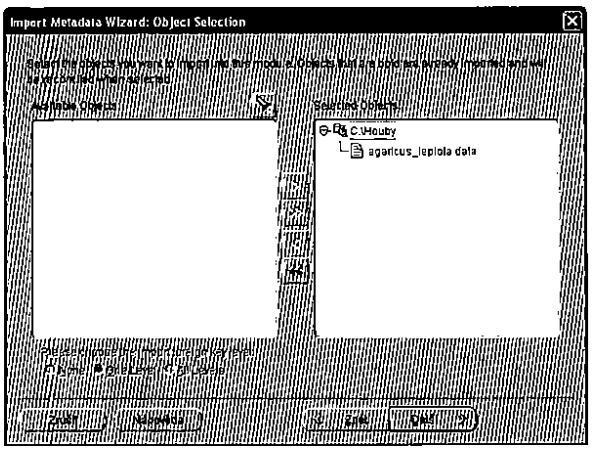
Průvodce importem metadat

Průvodce importem Metadat se skládá ze tří kroků. V prvním kroku nastavíme případný filtr pro vstupní údaje. V našem případě máme v nastaveném adresáři dva soubory s názvy agaricus_lepiota.data a agaricus_lepiota.names. Pro jednoduchou transformaci, kterou plánujeme uskutečnit, potřebujeme jen soubor s příponou data, proto aktivujeme volbu *Data Files Matching this pattern* a nastavíme filtr *.data.



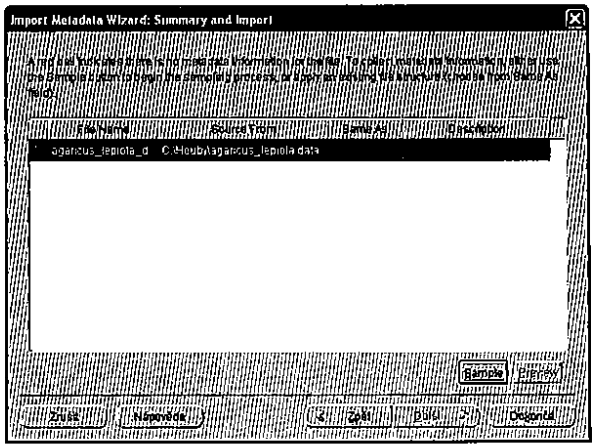
Obr. 10.12 OWB – průvodce importem metadat, nastavení filtru

V našem případě, kdy máme ve zdrojovém adresáři jen dva soubory, bychom si poradili i bez filtru. V následujícím dialogu totiž přesouváme potřebné soubory z okna dostupných souborů do okna vybraných souborů.



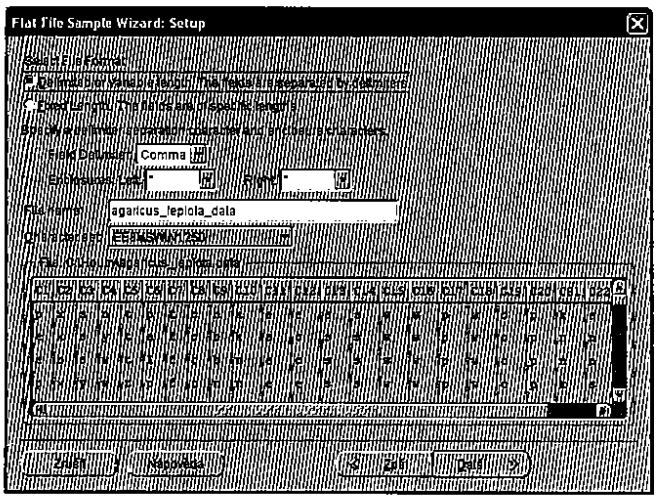
Obr. 10.13 OWB – průvodce importem metadat, výběr souborů

V posledním souhrnném dialogu průvodce vytvořením metadat se zobrazí seznam vybraných souborů. Pokud je nalevo od názvu souboru červený kroužek, znamená to, že pro tento soubor ještě nebyla vytvořena metadata. Tato metadata vytvoříme po aktivování tlačítka Sample. V našem případě se automaticky aktivuje průvodce vytvořením metadat pro flat-soubory.



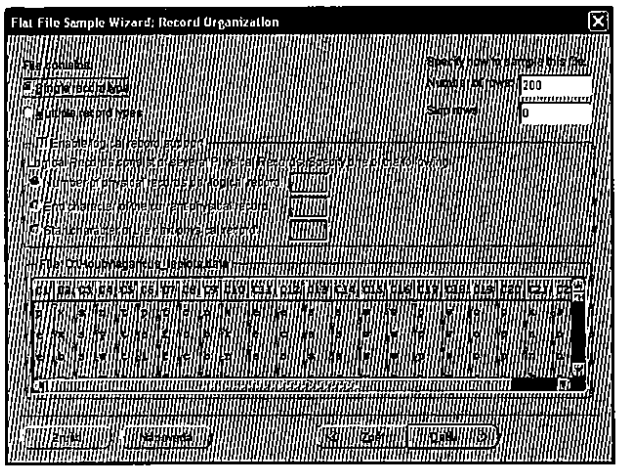
Obr. 10.14 OWB – průvodce importem metadat, souhrnný dialog

Soubor, který jsme použili jako zdroj údajů, je sice, co se týká názvů, zakódovaný, ale jinak je to přímo vzorová ukázka flat-souboru, kde jsou jednotlivé atributy (budoucí sloupce) odděleny čárkami a jednotlivé záznamy začínají vždy na nových řádcích. Proto v následujícím dialogu už nemusíme nic nastavovat. Průvodce všechno učinil za nás.



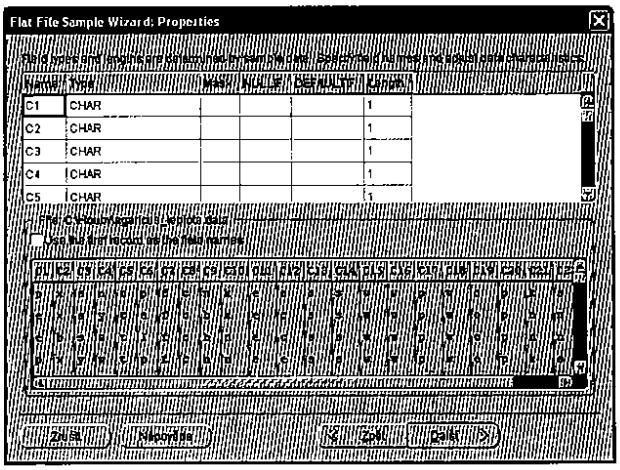
Obr. 10.15 OWB – průvodce importem flat-souboru, nastavení parametrů

Pokračujeme nastavením parametrů organizace jednotlivých záznamů. Jelikož v našem případě platí, že každý řádek flat-souboru bude tvořit jeden záznam v cílové tabulce databáze, ponecháme nastavenou volbu Single record type.



Obr. 10.16 OWB – průvodce importem flat-souboru, organizace záznamů

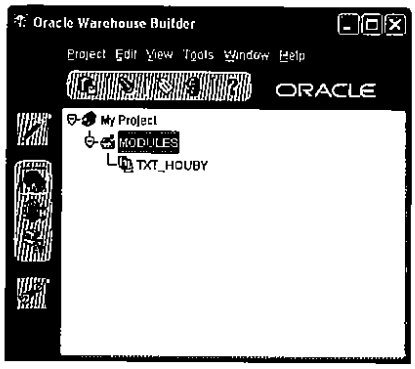
Už v předchozích dialozích průvodce jsme viděli vyznačenou tabulku, která vznikne transformací flat-souboru. V následujícím dialogu můžeme ještě nastavit názvy sloupců a jejich datové typy. V našem případě celkem vyhovuje implicitní volba, kdy budou sloupce pojmenované C1 až C23 a budou datového typu CHAR s rozsahem jeden znak.



Obr. 10.17 OWB – průvodce importem flat-souboru, organizace záznamů

Následuje souhrnný dialog, ve kterém uvidíme metadata pro import flat-souboru v přehledné stromové struktuře. Po nadefinování metadat se červený kroužek v dialogu průvodce importem metadat změní na zelený.

První krok, tedy nadefinování parametrů a metadat pro zdroj údajů, máme úspěšně za sebou. Tento zdroj údajů se objeví i ve složce MODULES v pracovním okně aplikace OWB Client.

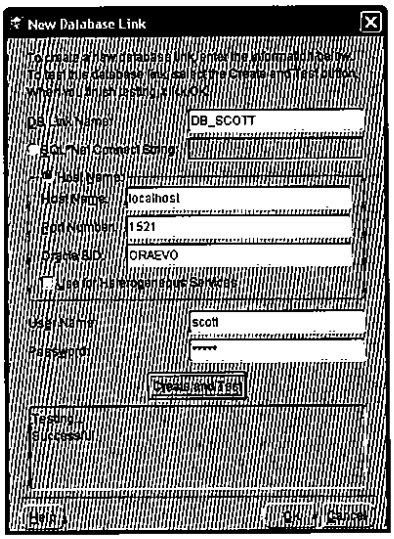


Obr. 10.18 OWB – složka zdroje údajů

Nadefinování modulu pro cílovou databázi

Znovu spustíme průvodce vytvořením nového modulu. V prvním dialogu průvodce znovu pojmenujeme nový modul (v našem případě DB_HOUBY) a určíme ho za cílový (Warehouse Target). V následujícím dialogu budou všechny parametry pro cílovou databázi nastavené „napevno". Dialog pro nastavení připojovacích parametrů přeskočíme, jelikož údaje i metadata půjdou do depozitáře (anglicky repository) OWB.

Ve finálním dialogu průvodce se, podobně jako to bylo při definování zdrojového modulu, zobrazí souhrnné informace o zdroji údajů. V tomto dialogu také zaškrtneme volbu Proceed to the Import Metadata Wizard, která spustí průvodce importem vstupních metadat. Tentokrát budeme údaje ukládat do databáze. Pro jednoduchost je uložíme do schématu cvičného klienta SCOTT, který má implicitní heslo TIGER.

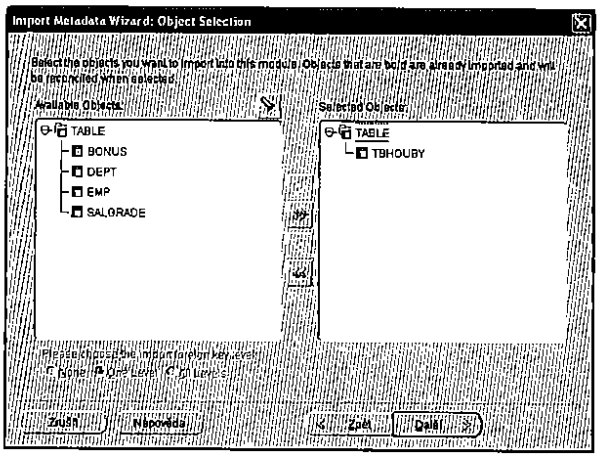


Obr. 10.19 OWB – nastavení parametrů pro připojení se k cílové databázi

Po nastavení parametrů pro připojení k databázi můžeme toto připojení pomocí tlačítka *Create and Test* vyzkoušet. V následujících dialozích průvodce pro výběr typů objektů vybereme, že budeme chtít používat jen tabulky. V dialogu pro výběr konkrétních objektů (v našem případě se tento výběr logicky zúžil jen na databázové tabulky) vidíme, že byla vytvořena metadata pro novou databázovou tabulku TBHOUBY.

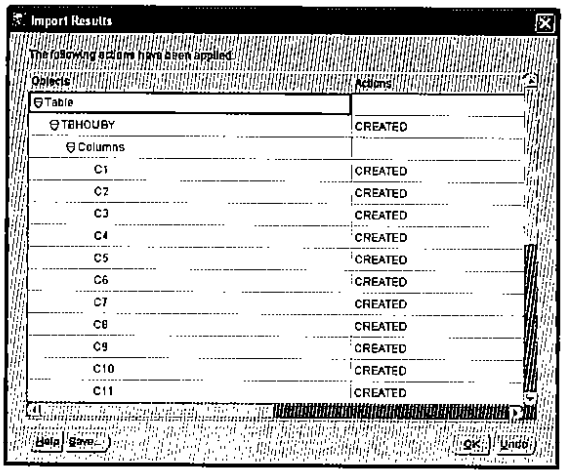
O tom, že se jedná skutečně zatím jen o metadata, se přesvědčíme tak, že si pomocí klientské konzolové aplikace necháme vypsat informace o této tabulce. Zjistíme, že zatím fyzicky neexistuje.

```
desc TBHOUBY;

ERROR:
ORA-04043: objekt TBHOUBY neexistuje
```



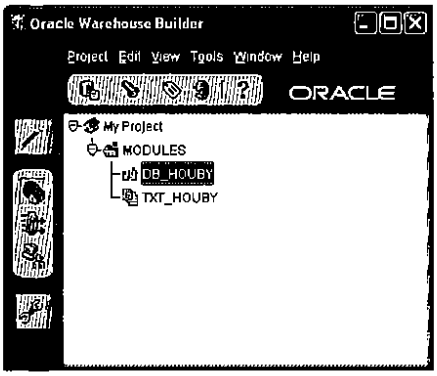
Obr. 10.20 OWB – výběr cílové tabulky

Potvrdí nám to i následující dialog průvodce, kde je naplánována akce vytvoření nové databázové tabulky TBHOUBY. Po klepnutí na tlačítko Dokončit proběhne import metadat.



Obr. 10.21 OWB – výsledky importu metadat

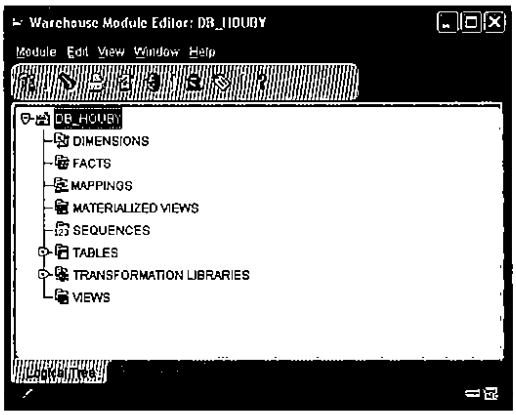
Po úspěšném absolvování importu metadat máme vytvořenu i destinaci (zatím jen na úrovni metadat) pro transformované údaje.



Obr. 10.22 OWB – složka zdroje údajů a jejich destinace

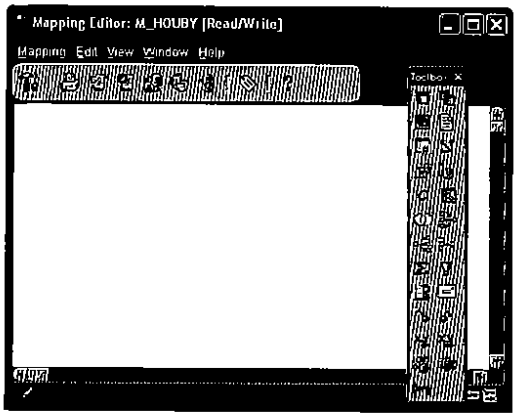
Definování mapování a transformace

V kontextové nabídce cílové složky údajů vybereme volbu Editor, čímž se zobrazí dialog nástroje Warehouse Module Editor. V něm aktivujeme kontextovou nabídku pro položku Mappings.



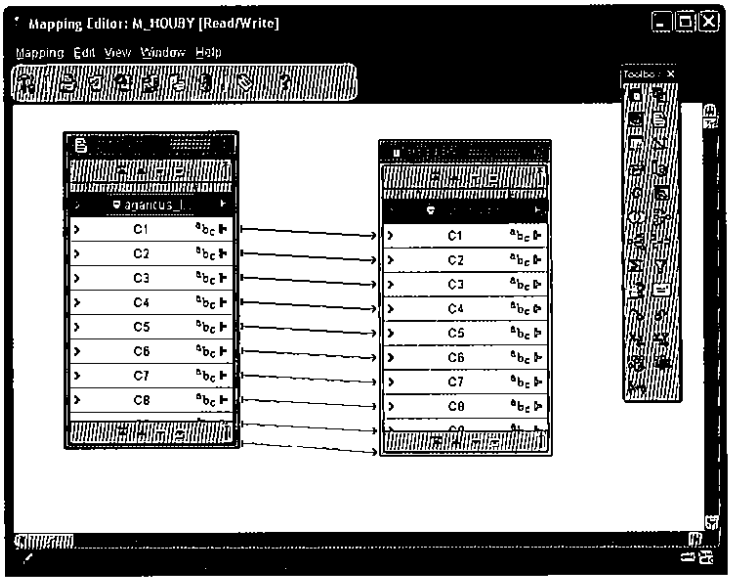
Obr. 10.23 OWB – složka zdroje údajů a jejich destinace

Následně aktivujeme volbu Create Mapping. Ani tentokrát nebude porušeno pravidlo, že při všech úkonech spojených s transformací nás provázel nějaký průvodce. Tentokrát to bude průvodce mapováním (Mapping Wizard). V prvním a s výjimkou potvrzovacího dialogu i posledním dialogu průvodce náš mapovací modul nějak pojmenujeme, například v našem případě M_HOUBY. Kromě pojmenování mapovacího modulu nám průvodce jeho vytvořením aktivuje okno nástroje Mapping editor.



Obr. 10.24 OWB – Mapping Editor

Pomocí nástroje Mapping editor můžeme v přehledné formě grafického návrhu vytvořit modul pro mapování údajů ze zdrojového flat-souboru do cílové databázové tabulky. Na plochu aplikace tedy jednoduše z okna Toolboxu přesuneme ikonky zdrojového textového souboru a databázové tabulky. Následně definujeme mapování jednoduchým natažením čáry ze záhlaví zobrazené tabulky, která reprezentuje strukturu zdrojového flat-souboru do záhlaví tabulky reprezentující strukturu cílové databázové tabulky.



Obr. 10.25 OWB – Mapping Editor

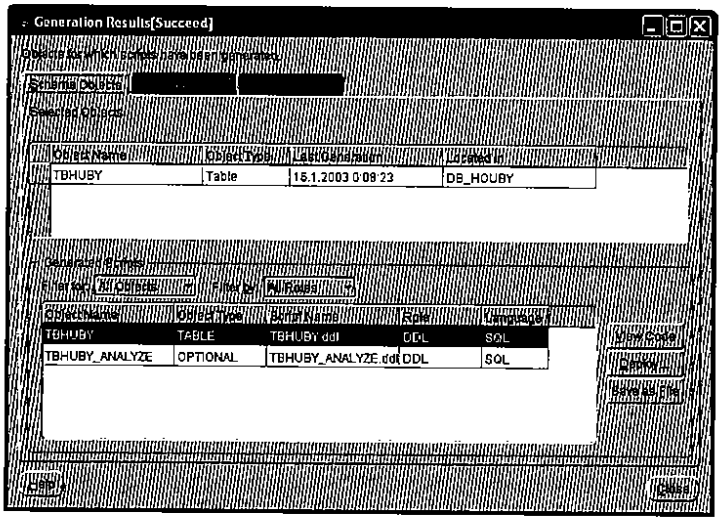
Po klepnutí na tlačítko Generate bude vygenerován předpis pro mapování a transformaci, v našem případě:

```
OPTIONS ( DIRECT=TRUE,PARALLEL=FALSE, ERRORS=50, BINDSIZE=50000, ROWS=200, READSIZE=65536)
LOAD DATA
  CHARACTERSET EE8MSWIN1250
  INFILE 'C:\Houby\agaricus_lepiota.data'
  READBUFFERS 4
  INTO TABLE "TBHUBY"
  APPEND
  REENABLE DISABLED_CONSTRAINTS

  FIELDS
    TERMINATED BY ','
    OPTIONALLY ENCLOSED BY '"'

  (
     "C1" POSITION (1) CHAR ,
     "C2" CHAR ,
     "C3" CHAR ,
   ...
   ...
```

```
    "C22" CHAR ,
    "C23" CHAR
)
```

Je to vlastně skript pro nástroj SQL * Loader.

V okně nástroje Warehouse module označíme cílový modul, v našem případě DB_HOUBY, a aktivujeme volbu Generate Create scripts.



Obr. 10.26 OWB – výsledek generování skriptů pro cílové objekty

Skripty si můžeme nechat vypsat.

```
/*************************************************************************
-- Product                    : Oracle Warehouse Builder
-- Generator Version          : 9.0.3.33.0
-- Created Date               : Thu Jan 16 00:08:21 CET 2003
-- Modified Date              : Thu Jan 16 00:08:21 CET 2003
-- Created By                 : owb
-- Modified By                : owb
-- Generated Object Type      : TABLE
-- Generated Object Name      : TBHUBY
-- Comments                   :
-- Copyright(c) 1999-2002 Oracle Corporation.
*************************************************************************/
```

```
WHENEVER SQLERROR EXIT FAILURE;

CREATE TABLE "TBHUBY"
(
  "C1" CHAR(1),
  "C2" CHAR(1),
  "C3" CHAR(1),
...
...
  "C22" CHAR(1),
  "C23" CHAR(1)
)
TABLESPACE "USERS"
  PARALLEL
  LOGGING
;
```

Praktický příklad analýzy OLAP pro Oracle 9i Release2

Cvičné schéma SH (Sales History)

Naši praktickou ukázku analýzy OLAP pro Oracle 9i budeme realizovat na cvičných údajích ze schématu **SH** (Sales History). Toto schéma se nainstaluje v procesu instalace při vytváření databáze. Jedná se o databázi obchodní firmy, která obchoduje převážně s oděvy. Nad strukturou a údaji ve schématu SH jsou vytvořeny i dvě „cvičné" krychle COST_CUBE a SALES_CUBE. Krychle jsou dostupné prostřednictvím aplikace Enterprise Manager Console, složka OLAP ->CUBES.

Abychom mohli s údaji ve cvičném schématu SH pracovat, potřebujeme znát přihlašovací údaje. Na produkčním počítači samozřejmě požádáme o přístup příslušného administrátora, na vývojářském počítači si můžeme tyto přihlašovací údaje nastavit sami. Příkazem v konzolové aplikaci SQL*Plus se připojíme jako uživatel SYS typu SYSDBA:

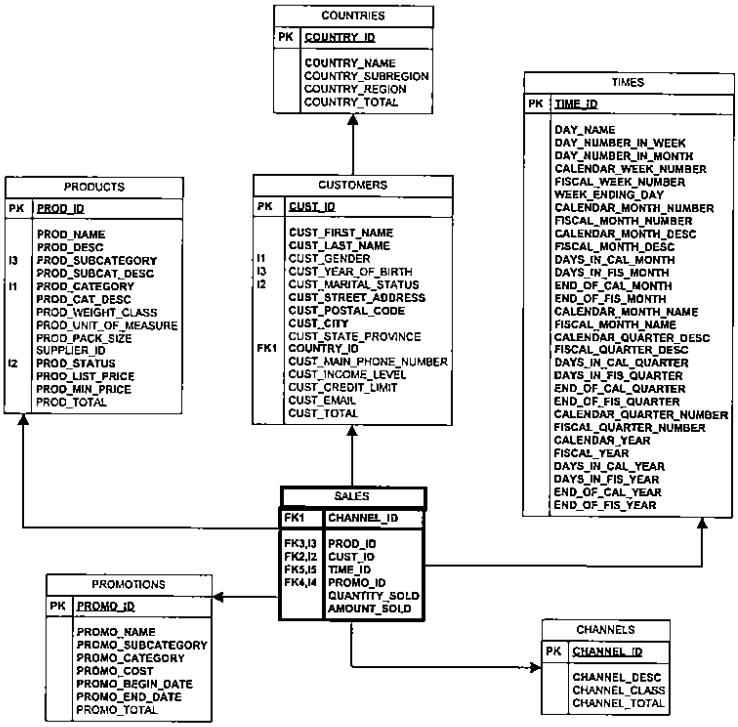
```
SQL> CONNECT / AS SYSDBA
```

Implicitní heslo uživatele SYS v Oracle 9i a předchozích verzích bylo change_on_install. Ve verzi Oracle 9i Release 2 musíme toto heslo změnit už při instalaci. Ale vraťme se k nastavení přístupových údajů schématu SH. Použijeme sekvenci příkazů:

```
ALTER USER SH ACCOUNT UNLOCK;
ALTER USER SH IDENTIFIED BY TIGER;
```

(V našem případě jsme nastavili heslo TIGER, populární implicitní heslo cvičného klienta SCOTT.) Po této změně nastavení se už můžeme přihlásit ke schématu SH s heslem TIGER.

Schéma obsahuje 15 tabulek a materializovaných pohledů. Pro naší ukázku stačí znát strukturu tabulky faktů **SALES** a tabulky dimenzí **TIMES, PRODUCTS, CUSTOMERS, COUNTRIES** a **PROMOTIONS**.



Obr. 10.27 Diagram schématu Sales History

Tabulky faktů a dimenzí si představíme podrobněji (*poznámka: u platformy Oracle zjistíme návrhovou strukturu libovolné tabulky příkazem DESC název_tabulky, například DESC sales;*).

Tabulka faktů

Tabulka faktů **SALES** obsahuje tyto sloupce:

Název	Typ
PROD_ID	NUMBER(6)
CUST_ID	NUMBER
TIME_ID	DATE
CHANNEL_ID	CHAR(1)
PROMO_ID	NUMBER(6)
QUANTITY_SOLD	NUMBER(3)
AMOUNT_SOLD	NUMBER(10,2)

Tabulka SH.SALES je poměrně velká, protože obsahuje více než milion záznamů (přesně 1 016 271). Jsou v ní údaje o obchodování fiktivní firmy za tři roky (1998, 1999, 2000). Pro zajímavost, tabulka SH.SALES je rozdělená na části (oddíly) po jednotlivých kvartálech. Tři roky tedy představují 12 oddílů. Pokud se chceme podívat na ukázku údajů, potřebujeme dotaz SQL, který vrátí rozumné množství údajů. Náš výběr musíme omezit na krátké časové období, například jeden den. Proto použijeme například dotaz

```
SELECT * FROM sh.sales WHERE time_id='24.12.99';.
```

```
   PROD_ID    CUST_ID TIME_ID  C   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- -------- - ---------- ------------- -----------
      7975      25970 24.12.99 C       9999            13         182
      2655      37280 24.12.99 T       9999             2         390
      2415     180430 24.12.99 P       9999             8         552
      3250     136490 24.12.99 I       9999            12        1500
      6455       2630 24.12.99 I       9999             4         152
      8880      16480 24.12.99 S       9999             3         117
     31635       6910 24.12.99 T       9999             2          20
...
     44635      45940 02.01.98 T       9999             2          34

1 261 řádek vybráno.
```

Schválně jsme vybrali datum Vánoc, v domnění, že tam bude málo údajů. Pokud položíme ten samý dotaz i pro jiné, pracovní dny zjistíme, že i v tyto dny je v databázové tabulce SH.SALES přibližně stejné množství záznamů. Takže jsme pravděpodobně odhalili, že se jedná o umělé vygenerovanou databázi. Ale nevadí, snad byla poupravená na základě skutečných údajů.

Tabulky dimenzí

Bude nás zajímat především časová dimenze vytvořená na základě údajů z tabulky SH.TIMES, dále produktová dimenze vytvořená nad tabulkou SH.PRODUCT, regionální dimenze vytvořená nad tabulkami SH.CUSTOMERS a SH.COUNTRIES, dimenze prodej-

ního kanálu vytvořená na základě tabulky SH.CHANNELS a dimenze inzertních kanálů vytvořená na základě tabulky SH.PROMOTIONS

Časová dimenze

Jako první budeme zkoumat nejdůležitější – časovou – dimenzi. Tabulka SH.TIMES, která obsahuje údaje, které slouží jako podklad pro vytváření časových dimenzí, má poměrně složitou, značně denormalizovanou strukturu, která umožňuje vytvářet dimenze jednak podle kalendářních zvyklostí a jednak podle fiskálních období příslušné firmy.

Název	Typ
TIME_ID	DATE
DAY_NAME	VARCHAR2(9)
DAY_NUMBER_IN_WEEK	NUMBER(1)
DAY_NUMBER_IN_MONTH	NUMBER(2)
CALENDAR_WEEK_NUMBER	NUMBER(2)
FISCAL_WEEK_NUMBER	NUMBER(2)
WEEK_ENDING_DAY	DATE
CALENDAR_MONTH_NUMBER	NUMBER(2)
FISCAL_MONTH_NUMBER	NUMBER(2)
CALENDAR_MONTH_DESC	VARCHAR2(8)
FISCAL_MONTH_DESC	VARCHAR2(8)
DAYS_IN_CAL_MONTH	NUMBER
DAYS_IN_FIS_MONTH	NUMBER
END_OF_CAL_MONTH	DATE
END_OF_FIS_MONTH	DATE
CALENDAR_MONTH_NAME	VARCHAR2(9)
FISCAL_MONTH_NAME	VARCHAR2(9)
CALENDAR_QUARTER_DESC	CHAR(7)
FISCAL_QUARTER_DESC	CHAR(7)
DAYS_IN_CAL_QUARTER	NUMBER
DAYS_IN_FIS_QUARTER	NUMBER
END_OF_CAL_QUARTER	DATE
END_OF_FIS_QUARTER	DATE
CALENDAR_QUARTER_NUMBER	NUMBER(1)
FISCAL_QUARTER_NUMBER	NUMBER(1)
CALENDAR_YEAR	NUMBER(4)
FISCAL_YEAR	NUMBER(4)
DAYS_IN_CAL_YEAR	NUMBER
DAYS_IN_FIS_YEAR	NUMBER
END_OF_CAL_YEAR	DATE
END_OF_FIS_YEAR	DATE

Tabulka logicky obsahuje 1 461 řádků, tedy jeden řádek pro každý den za sledované období tří let. V tomto případě je zas problém s velkým množstvím sloupců, proto do našeho výpisu pomocí dotazu SQL zahrneme jen některé.

```
SELECT TIME_ID, DAY_NAME, DAY_NUMBER_IN_WEEK, DAY_NUMBER_IN_MONTH, WEEK_ENDING_DAY
    FROM sh.times;

TIME_ID  DAY_NAME  DAY_NUMBER_IN_WEEK DAY_NUMBER_IN_MONTH WEEK_END
-------- --------- ------------------ ------------------- --------
21.01.98 Wednesday                  3                  21 25.01.98
22.01.98 Thursday                   4                  22 25.01.98
23.01.98 Friday                     5                  23 25.01.98
24.01.98 Saturday                   6                  24 25.01.98
25.01.98 Sunday                     7                  25 25.01.98
26.01.98 Monday                     1                  26 01.02.98
27.01.98 Tuesday                    2                  27 01.02.98
28.01.98 Wednesday                  3                  28 01.02.98
29.01.98 Thursday                   4                  29 01.02.98
...
...
02.05.01 Wednesday                  3                   2 06.05.01
03.05.01 Thursday                   4                   3 06.05.01
04.05.01 Friday                     5                   4 06.05.01
31.12.01 Monday                     1                  31 06.01.02

1 461 řádek vybráno.
```

Tabulka SH.TIMES by mohla sloužit jako učebnicový odstrašující příklad typické nenormalizované tabulky, ale na druhou stranu umožňuje velmi efektivní drilování a vysoký dotazovací výkon v databázi OLAP.

Typickým příkladem hierarchie časové dimenze vytvořené nad touto tabulkou by mohlo být schéma

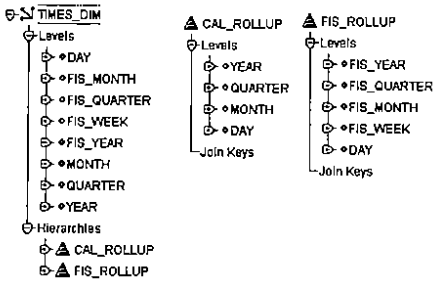
- YEAR — sloupec CALENDAR_YEAR
  - QUARTER — sloupec CALENDAR_QUARTER_DESC
    - MONTH — sloupec CALENDAR_MONTH_DESC
      - DAY — sloupec TIME_ID

Pokud ve výpisu chceme nechat jen sloupce pro vytvářenou časovou dimenzi, použijeme příkaz jazyka SQL:

```
SELECT CALENDAR_YEAR, CALENDAR_QUARTER_DESC, CALENDAR_MONTH_DESC, TIME_ID FROM sh.times;

CALENDAR_YEAR CALENDA CALENDAR TIME_ID
------------- ------- -------- --------
         1998 1998-Q1 1998-01  21.01.98
         1998 1998-Q1 1998-01  22.01.98
         1998 1998-Q1 1998-01  23.01.98
         1998 1998-Q1 1998-01  24.01.98
         1998 1998-Q1 1998-01  25.01.98
         1998 1998-Q1 1998-01  26.01.98
         1998 1998-Q1 1998-01  27.01.98
```

```
1998 1998-Q1 1998-01  28.01.98
1998 1998-Q1 1998-01  29.01.98
1998 1998-Q1 1998-01  30.01.98
```

Cvičná krychle SALES_CUBE má hierarchii časové dimenze TIMES_DIM navrženu takto:



Obr. 10.28 Dimenze TIMES_DIM cvičné krychle SALES_CUBE

Přitom využívá dvě hierarchie časové dimenze. Jednak kalendářní a jednak fiskální hierarchii.

Produktová dimenze

Produktová dimenze je ve schématu SH vytvořena nad údaji z tabulky **SH.PRODUCTS**, která má strukturu:

Name	Type
PROD_ID	NUMBER(6)
PROD_NAME	VARCHAR2(50)
PROD_DESC	VARCHAR2(4000)
PROD_SUBCATEGORY	VARCHAR2(50)
PROD_SUBCAT_DESC	VARCHAR2(2000)
PROD_CATEGORY	VARCHAR2(50)
PROD_CAT_DESC	VARCHAR2(2000)
PROD_WEIGHT_CLASS	NUMBER(2)
PROD_UNIT_OF_MEASURE	VARCHAR2(20)
PROD_PACK_SIZE	VARCHAR2(30)
SUPPLIER_ID	NUMBER(6)
PROD_STATUS	VARCHAR2(20)
PROD_LIST_PRICE	NUMBER(8,2)
PROD_MIN_PRICE	NUMBER(8,2)
PROD_TOTAL	VARCHAR2(13)

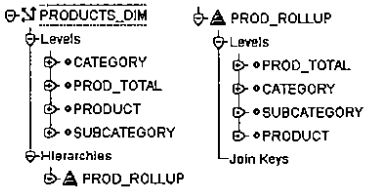
Tato tabulka obsahuje zaokrouhleně 10 000 řádků. I v tomto případě musíme ukázkový výpis přizpůsobit, jelikož tabulka obsahuje poměrně hodně sloupců, ale to by nebylo z hlediska výpisu nejhorší. Některé sloupce, například sloupec PROD_DESC pro popis produktu, mohou mít šířku až 4 000 znaků. Pro ilustrační výpis jsme vybrali jen některé položky, ale bylo potřeba upravit šířku některých sloupců ve výpisu. Nic neříkající sloupec PROD_ID jsme ve výpisu nahradili sloupcem PROD_NAME.

```
SET linesize 120;
COLUMN prod_category  format a10
COLUMN prod_subcategory  format a30
COLUMN prod_name  format a35
SELECT prod_category,
       prod_subcategory,
       prod_name
FROM sh.products WHERE prod_id<10;

PROD_CATEG PROD_SUBCATEGORY               PROD_NAME
---------- ------------------------------ -----------------------------------
Women      Trousers - Women               Gurfield& Murks Pleated Trousers
Women      Trousers - Women               Gurfield& Murks Pleated Trousers
Girls      Trousers And Jeans - Girls     Coin Pocket Twill Cargo Trousers
Boys       Shirts - Boys                  And 2 Crosscourt Tee Kids
Boys       Shirts - Boys                  And 2 Crosscourt Tee Kids
Boys       Shirts - Boys                  And 2 Crosscourt Tee Kids
Men        Shorts - Men                   Kahala Pleated Chino Short
Boys       Shirts - Boys                  And 2 Crosscourt Tee Kids
Women      Shoes - Women                  Vunelli Bb-956

9 řádek vybráno.
```

Možným příkladem hierarchie produktové dimenze vytvořené nad tabulkou PRODUCTS by mohlo být schéma

- KATEGORIE — sloupec PROD_CATEGORY
- • PODKATEGORIE — sloupec PROD_SUBCATEGORY
- • • PRODUKT — sloupec PROD_NAME

Cvičná krychle SALES_CUBE má navrženu hierarchii produktové dimenze PRODUCTS_DIM takto:



Obr. 10.29 Dimenze PRODUCTS_DIM cvičné krychle SALES_CUBE

Geografická dimenze (podle bydliště zákazníků)

Tabulka **SH.CUSTOMERS** umožní vytvářet dimenze podle územní lokalizace zákazníka. Její struktura je v tabulce:

Název	Typ
CUST_ID	NUMBER
CUST_FIRST_NAME	VARCHAR2(20)
CUST_LAST_NAME	VARCHAR2(40)
CUST_GENDER	CHAR(1)
CUST_YEAR_OF_BIRTH	NUMBER(4)
CUST_MARITAL_STATUS	VARCHAR2(20)
CUST_STREET_ADDRESS	VARCHAR2(40)
CUST_POSTAL_CODE	VARCHAR2(10)
CUST_CITY	VARCHAR2(30)
CUST_STATE_PROVINCE	VARCHAR2(40)
COUNTRY_ID	CHAR(2)
CUST_MAIN_PHONE_NUMBER	VARCHAR2(25)
CUST_INCOME_LEVEL	VARCHAR2(30)
CUST_CREDIT_LIMIT	NUMBER
CUST_EMAIL	VARCHAR2(30)
CUST_TOTAL	VARCHAR2(14)

Tabulka obsahuje údaje o 50 000 zákaznících. Ukázku výpisu získáme například příkazem

```
SELECT CUST_ID, CUST_FIRST_NAME, CUST_LAST_NAME, CUST_CITY
   FROM SH.CUSTOMERS WHERE CUST_ID<100;


   CUST_ID CUST_FIRST_NAME  CUST_LAST_NAME   CUST_CITY
---------- ---------------- ---------------- ---------------
        10 Abigail          Kessel           Downham Market
        20 Abner            Everett          Dolores
        30 Abraham          Odenwald         Dolores
        40 Absolom          Sampson          Lowndesville
        50 Ada              Nenninger        Gif-sur-Yvette
        60 Adel             Rhodes           Downham Market
        70 Angie            Riffken          Springhill
        80 Angela           Wiley            Salamanca
        90 Anand            Hanes            Springhill
```

Pokud chceme ve výpisu sloučit jméno a příjmení zákazníka, použijeme příkaz

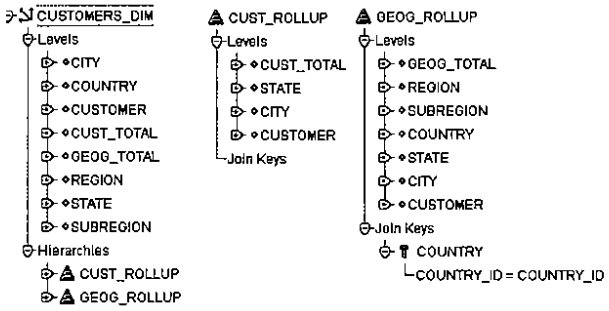
```
SET linesize 120;
COLUMN cust_state_province  format a25
SELECT cust_state_province,
       cust_city,
```

```
      cust_last_name ||' '|| cust_first_name AS CUST_NAME
FROM sh.customers WHERE cust_id<100;

CUST_STATE_PROVINCE   CUST_CITY         CUST_NAME
--------------------- ----------------- ----------------------------
England - Norfolk     Downham Market    Kessel Abigail
CO                    Dolores           Everett Abner
CO                    Dolores           Odenwalld Abraham
SC                    Lowndesville      Sampson Absolom
Ile-de-France         Gif-sur-Yvette    Nenninger Ada
England - Norfolk     Downham Market    Rhodes Adel
MN                    Springhill        Riffken Angie
Salamanca             Salamanca         Wiley Angela
MN                    Springhill        Hanes Anand...
```

Možným příkladem hierarchie zákaznické dimenze je například

```
CUSTOMERS
  • STATE              sloupec CUST_STATE_PROVINCE
  • • CITY             sloupec CUST_CITY
  • • • CUSTOMER       sloupec CUST_ID
```

Cvičná krychle SALES_CUBE má navrženu hierarchii zákaznické dimenze CUSTOMERS_DIM takto:



Obr. 10.30 Dimenze CUSTOMERS_DIM cvičné krychle SALES_CUBE

Pro regionální členění je důležitá i tabulka SH.COUNTRIES. Její struktura a obsah nebude pro znalce zeměpisu žádným překvapením.

Název	Typ
COUNTRY_ID	CHAR(2)
COUNTRY_NAME	VARCHAR2(40)
COUNTRY_SUBREGION	VARCHAR2(30)
COUNTRY_REGION	VARCHAR2(20)

```
SELECT * FROM sh.countries;

CO COUNTRY_NAME              COUNTRY_SUBREGION  COUNTRY_REGION
-- ------------------------- ------------------ --------------
US United States of America  Northern America   Americas
DE Germany                   Western Europe     Europe
UK United Kingdom            Western Europe     Europe
NL The Netherlands           Western Europe     Europe
IE Ireland                   Western Europe     Europe
DK Denmark                   Western Europe     Europe
FR France                    Western Europe     Europe
ES Spain                     Western Europe     Europe
TR Turkey                    Western Europe     Europe
PL Poland                    Eastern Europe     Europe
BR Brazil                    Southern America   Americas
AR Argentina                 Southern America   Americas
MY Malaysia                  Asia               Asia
JP Japan                     Asia               Asia
IN India                     Asia               Asia
AU Australia                 Australia          Oceania
NZ New Zealand               Australia          Oceania
ZA South Africa              Africa             Africa
SA Saudi Arabia              Middle East        Middle East
```

Dimenze prodejních kanálů

Samozřejmě nás bude zajímat i efektivita jednotlivých prodejních kanálů. Dimenze prodejních kanálů je ve schématu SH vytvořena nad údaji z tabulky **SH.CHANNELS**, která má tuto strukturu a obsah:

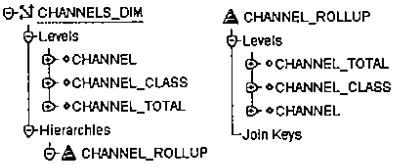
Název	Typ
CHANNEL_ID	NOT NULL CHAR(1)
CHANNEL_DESC	NOT NULL VARCHAR2(20)
CHANNEL_CLASS	VARCHAR2(20)

```
SELECT * FROM sh.channels;

C CHANNEL_DESC         CHANNEL_CLASS
- -------------------- --------------------
S Direct Sales         Direct
T Tele Sales           Direct
C Catalog              Indirect
I Internet             Indirect
P Partners             Others
```

Možným příkladem hierarchie dimenze prodejních kanálů je

```
CHANNELS
  • TOTAL              sloupec CHANNEL_TOTAL
  • • DRUH KANALU      sloupec CHANNEL_CLASS
  • • • KANAL          sloupec CHANNEL
```

Cvičná krychle SALES_CUBE má navrženu hierarchii zákaznické dimenze CHANNELS_DIM takto:



Obr. 10.31 Dimenze CHANNELS_DIM cvičné krychle SALES_CUBE

Dimenze inzertních kanálů

Kromě efektivity prodejních kanálů jsou důležité i inzertní kanály, proto jsou v databázi i údaje o tom, na základě jakého podnětu byla ta která obchodní transakce uskutečněna. Dimenze inzertních kanálů je ve schématu SH vytvořena nad údaji z tabulky **SH.PROMOTIONS**, která má tuto strukturu:

Název	Typ
PROMO_ID	NUMBER(6)
PROMO_NAME	VARCHAR2(20)
PROMO_SUBCATEGORY	VARCHAR2(30)
PROMO_CATEGORY	VARCHAR2(30)
PROMO_COST	NUMBER(10,2)
PROMO_BEGIN_DATE	DATE
PROMO_END_DATE	DATE

Obsahuje 501 záznamů, proto náš výpis zkrátíme.

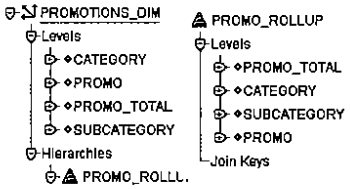
```
SELECT * FROM sh.promotions;

PROMO_ID PROMO_NAME           PROMO_SUBCATEGORY       PROMO_CATEGORY  PROMO_COST PROMO_BE PROMO_EN
---------- -------------------- ----------------------- --------------- ---------- -------- --------
         1 promotion name# 1    downtown billboard      post             77200 15.09.98 15.11.98
         2 promotion name# 2    hospital flyer          flyer            47100 13.04.99 13.07.99
         3 promotion name# 3    coupon news             newspaper        23900 25.08.00 25.09.00
         4 promotion name# 4    manufacture rebate news newspaper         8700 18.11.98 18.01.99
         5 promotion name# 5    TV program sponsorship  TV               5700 17.03.99 17.06.99
         6 promotion name# 6    ad news                 newspaper        76800 08.05.00 08.06.00
```

```
     7 promotion name# 7   online discount         internet          36400 30.06.98 30.08.98
     8 promotion name# 8   manufacture rebate news newspaper         65600 26.10.99 26.01.00
     9 promotion name# 9   billboard              post              39500 11.12.00 11.01.01
    10 promotion name# 10  coupon news             newspaper         22400 16.07.98 16.09.98
...
   497 promotion name# 497 promotion movie        TV                23600 19.03.99 19.06.99
   498 promotion name# 498 coupon news            newspaper         25900 21.05.00 21.06.00
   499 promotion name# 499 TV program sponsorship TV                62200 03.06.98 03.08.98
   500 promotion name# 500 TV commercial          TV                99900 05.10.99 05.01.00
  9999 NO PROMOTION        NO RPOMOTION          NO PROMOTION          0 01.01.99 01.01.99
```

Možným příkladem hierarchie dimenze inzertních kanálů je například

```
PROMOTIONS
• TOTAL                 sloupec PROMO_TOTAL
• • KATEGORIE           sloupec CATEGORY
• • • PODKATEGORIE      sloupec SUBCATEGORY
• • • • INZERCE         sloupec PROMO
```

Cvičná krychle SALES_CUBE má navrženou hierarchii zákaznické dimenze PROMOTIONS_DIM takto:



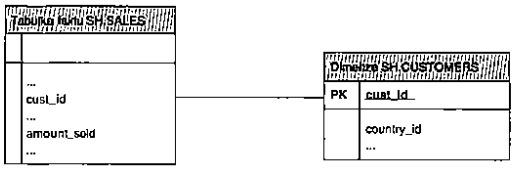
Obr. 10.32 Dimenze PROMOTIONS_DIM cvičné krychle SALES_CUBE

Analýza v jazyce SQL – klauzule CUBE a ROLLUP

Po seznámení se se strukturou tabulek v cvičném schématu SH a popisu schémat a struktur tabulek faktů a dimenzí můžeme přejít k jednoduchým analýzám pomocí klauzulí CUBE a ROLLUP. Pomocí klauzule CUBE, která rozšiřuje možnosti příkazu SELECT, získáme multidimenzionální přehled všech možných kombinací podle vybraných dimenzí. Syntaktický předpis je zdánlivě velmi jednoduchý,

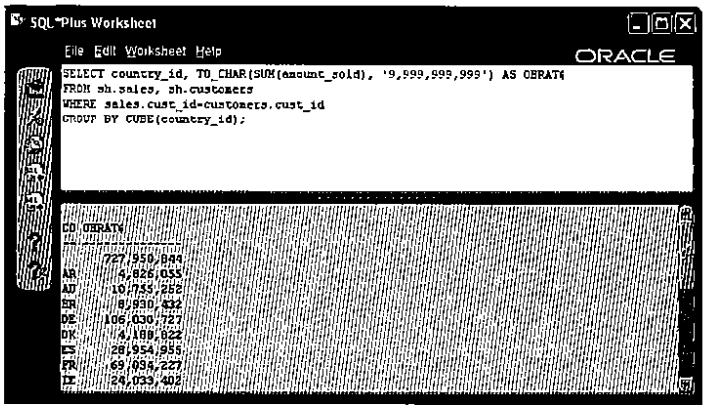
```
SELECT ... GROUP BY CUBE( seznam_seskupených_sloupců )
```

ale nad vytvořením správně fungujícího příkazu, který vrátí námi požadované údaje, musíme trochu zapřemýšlet a uvidíme, že desetiřádkový dotaz SQL nebude ničím výjimečným. Nejjednodušším příkladem použití klauzule CUBE, který se nám nad databází SALES HISTORY podaří vytvořit, bude asi přehled obratu podle jedné dimenze – krajiny. Jako tabulku faktů použijeme tabulku **SH.SALES** a jako tabulku pro jedinou dimenzi použijeme tabulku **SH.CUSTOMERS**.



Obr. 10.33 Nejjednodušší schéma star (hvězda) s tabulkou faktů (SH.SALES) a jednou tabulkou dimenzí (SH.CUSTOMERS)

Příkazy v jazyce SQL můžeme po přihlášení zadávat v některé konzolové aplikaci, například SQL*Plus Worksheet.

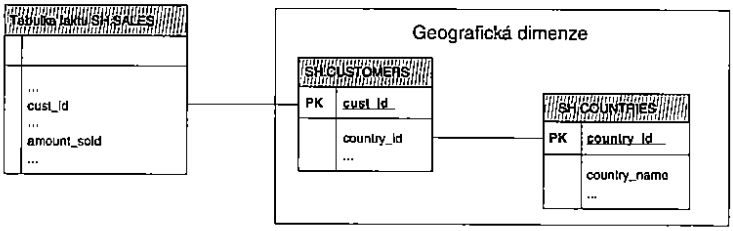


Obr. 10.34 Zadávání příkazů SQL pomocí konzolové aplikace

```
SELECT country_id, TO_CHAR(SUM(amount_sold), '9,999,999,999') AS OBRAT$
FROM sh.sales, sh.customers
WHERE sales.cust_id=customers.cust_id
GROUP BY CUBE(country_id);


CO OBRAT$
-- --------------
      727,950,844
AR      4,826,055
AU     10,755,252
BR      8,930,432
DE    106,030,727
DK      4,188,822
```

```
ES      28,954,955
FR      69,034,227
IE      24,033,402
IN      10,772,702
JP       8,303,362
MY       7,990,384
NL     112,500,261
NZ       2,812,685
PL       8,144,037
SA       2,379,861
TR       1,610,852
UK     103,991,934
US     209,606,696
ZA       3,084,197
```

Abychom získali přehledný výpis toho nejdůležitějšího, co nás zajímá, tedy souhrnů, který bude srozumitelný pro lidi z finančního světa, použijeme pro výpis funkci SUM zkonvertovanou do řetězcového formátu `TO_CHAR(SUM(amount_sold), '9,999,999,999')`. Tento výpis má stále ještě jednu chybičku na kráse, dvoupísmenný kód země bychom mohli nahradit jejím názvem, proto do hry zapojíme i tabulku SH.COUNTRIES a místo sloupce country_id z tabulky SH.SUSTOMERS použijeme sloupec country_name z tabulky SH.COUNTRIES. Tímto krokem jsme se dostali k nejjednoduššímu hvězdicovému schématu, protože geografickou dimenzi máme vytvořenou pomocí dvou relačně svázaných tabulek SH.CUSTOMERS a SH.COUNTRIES. Tato relační vazba je definována podmínkou `customers.country_id = countries.country_id`.



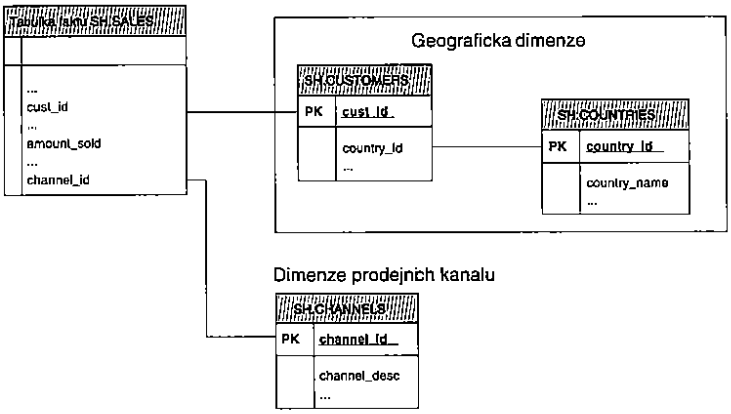
Obr. 10.35 Nejjednodušší schéma snowflake (vločka) s tabulkou faktů (SH.SALES) a jednou dimenzí tvořenou tabulkami SH.CUSTOMERS a SH.COUNTRIES

Další vylepšení, které můžeme aplikovat, je setřídění řádků výpisu podle obratu, a to sestupně.

```
SELECT country_name, TO_CHAR(SUM(amount_sold), '9,999,999,999') AS OBRAT$
FROM sh.sales, sh.customers, sh.countries
WHERE sales.cust_id=customers.cust_id AND customers.country_id = countries.country_id
GROUP BY CUBE(country_name)
ORDER BY OBRAT$ DESC;
```

```
COUNTRY_NAME                   OBRATS
------------------------------ -------------
                                727,950,844
United States of America        209,606,696
The Netherlands                 112,500,261
Germany                         106,030,727
United Kingdom                  103,991,934
France                           69,034,227
Spain                            28,954,955
Ireland                          24,033,402
India                            10,772,702
Australia                        10,755,252
Brazil                            8,930,432
Japan                             8,303,362
Poland                            8,144,037
Malaysia                          7,990,384
Argentina                         4,826,055
Denmark                           4,188,822
South Africa                      3,084,197
New Zealand                       2,812,685
Saudi Arabia                      2,379,861
Turkey                            1,610,852
```

Všimněme si součtu v prvním řádku. Jedná se o souhrn, tedy celkovou sumu ze všech zemí. Do naší jednorozměrné krychle můžeme přidat další dimenzi, například prodejní kanál, tedy tabulku SH.CHANNELS.



Obr. 10.36 Schéma snowflake se dvěma dimenzemi

Vazby mezi tabulkou faktů a tabulkami dimenzí budou definovány podmínkami

```
...
WHERE sales.cust_id    =customers.cust_id  AND customers.country_id=countries.country_id
  AND sales.channel_id =channels.channel_id
...
```

Příkaz jazyka SQL pro výpis této „krychle" se dvěma dimenzemi bude vypadat takto:

```
SELECT country_name, channel_desc, TO_CHAR(SUM(amount_sold), '9,999,999,999') AS OBRAT$
FROM sh.sales, sh.customers, sh.countries, sh.channels
WHERE sales.cust_id =customers.cust_id
      AND customers.country_id = countries.country_id
      AND sales.channel_id = channels.channel_id
GROUP BY CUBE(country_name, channel_desc);
```

```
COUNTRY_NAME                CHANNEL_DESC    OBRAT$
--------------------------- --------------- -------------
                                            727,950,844
                            Catalog          99,059,355
                            Internet        199,485,838
                            Partners         83,538,698
                            Tele Sales       33,037,423
                            Direct Sales    312,829,530
India                                        10,772,702
India                       Catalog           1,445,541
India                       Internet          2,949,274
India                       Partners          1,221,765
India                       Tele Sales          470,161
India                       Direct Sales      4,685,961
Japan                                         8,303,362
Japan                       Catalog           1,154,903
Japan                       Internet          2,307,437
Japan                       Partners            968,034
Japan                       Tele Sales          408,452
Japan                       Direct Sales      3,464,537
...
...
United States of America                    209,606,696
United States of America    Catalog          28,756,542
United States of America    Internet         57,382,416
United States of America    Partners         24,190,869
United States of America    Tele Sales        9,577,205
United States of America    Direct Sales     89,699,665
```

Takto vygenerovaná tabulka má už 120 řádků, proto jsme v našem výpisu ponechali jen některé. Abychom nemuseli zkracovat výpis, bude nás zajímat prodej jen v USA a Japonsku, proto přidáme do našeho příkazu omezující podmínku `AND customers.country_id IN ('JP', 'US')`.

```
SELECT country_name, channel_desc, TO_CHAR(SUM(amount_sold), '9,999,999,999') AS OBRAT$
FROM sh.sales, sh.customers, sh.countries, sh.channels
WHERE sales.cust_id =customers.cust_id
      AND customers.country_id = countries.country_id
      AND sales.channel_id = channels.channel_id
      AND customers.country_id IN ('JP', 'US')
GROUP BY CUBE(country_name, channel_desc);
```

```
OUNTRY_NAME               CHANNEL_DESC    OBRAT$
------------------------- --------------- -------------
                                          217,910,058
                          Catalog          29,911,445
                          Internet         59,689,853
                          Partners         25,158,903
                          Tele Sales        9,985,656
                          Direct Sales     93,164,201
Japan                                       8,303,362
Japan                     Catalog           1,154,903
Japan                     Internet          2,307,437
Japan                     Partners            968,034
Japan                     Tele Sales          408,452
Japan                     Direct Sales      3,464,537
United States of America                  209,606,696
United States of America  Catalog          28,756,542
United States of America  Internet         57,382,416
United States of America  Partners         24,190,869
United States of America  Tele Sales        9,577,205
United States of America  Direct Sales     89,699,665
```

Pro orientaci v takovém „strojovém" výpisu je potřeba mít trochu představivosti. Pokud tento výpis zkonvertujeme do podoby tabulky, bude to mnohem srozumitelnější a už není ani potřeba vysvětlovat, proč má náš vygenerovaný výpis právě 18 řádků. Jde o jednoduchou matici hodnot 6 řádků (5 hodnot možností prodejního kanálu a řádek souhrnu) u sloupce (dvě země, USA a Japonsko, a sloupec souhrnu), takže 6 x 3 = 18.

[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	28,756,542	1,154,903	29,911,445
[illegible]	57,382,416	2,307,437	59,689,853
[illegible]	24,190,869	968,034	25,158,903
[illegible]	9,577,205	408,452	9,985,656
[illegible]	89,699,665	3,464,537	93,164,201
[illegible]	209,606,696	8,303,362	217,910,058

Pokud máme dvě dimenze `country_name` a `channel_desc`, můžeme je mezi sebou zaměnit a pak dostaneme ty samé údaje, jen v jiném tvaru.

```
SELECT channel_desc, country_name, TO_CHAR(SUM(amount_sold), '9,999,999,999') AS OBRAT$
FROM sh.sales, sh.customers, sh.countries, sh.channels
WHERE sales.cust_id =customers.cust_id
      AND customers.country_id = countries.country_id
      AND sales.channel_id = channels.channel_id
      AND customers.country_id IN ('JP', 'US')
GROUP BY CUBE(channel_desc, country_name);
```

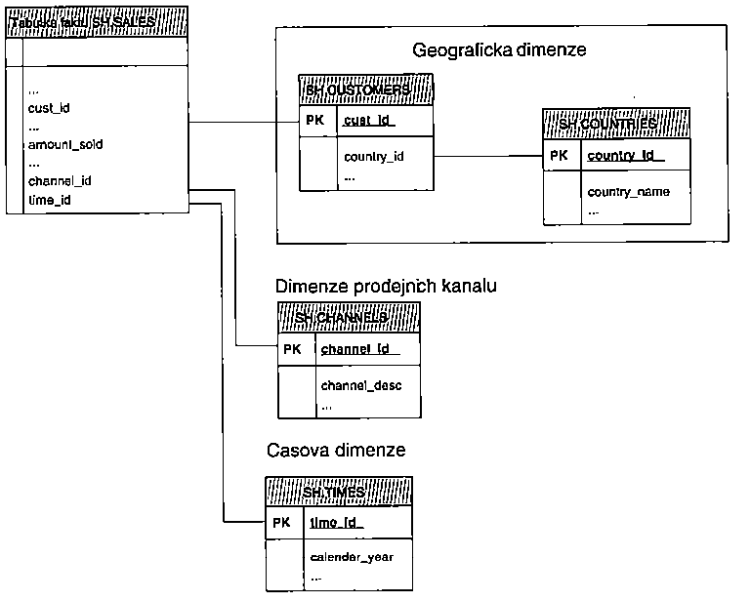
```
CHANNEL_DESC    COUNTRY_NAME                OBRAT$
--------------- --------------------------- -------------
                                            217,910,058
                Japan                         8,303,362
                United States of America    209,606,696
Catalog                                      29,911,445
Catalog         Japan                         1,154,903
Catalog         United States of America     28,756,542
Internet                                     59,689,853
Internet        Japan                         2,307,437
Internet        United States of America     57,382,416
Partners                                     25,158,903
Partners        Japan                           968,034
Partners        United States of America     24,190,869
Tele Sales                                    9,985,656
Tele Sales      Japan                           408,452
Tele Sales      United States of America      9,577,205
Direct Sales                                 93,164,201
Direct Sales    Japan                         3,464,537
Direct Sales    United States of America     89,699,665
```

Při tomto inkrementálním způsobu vytváření dotazu jazyka SQL vidíme, jak nám jeho délka nenápadně narůstá. Nyní nastal ten správný čas zavést do naší „krychle" třetí dimenzi – čas.

Budeme zkoumat obrat za tři roky, ale protože by tímto způsobem získaná tabulka měla až 72 řádků, omezíme se jen na dva prodejní kanály, na přímý a internetový prodej (podmínka `channels.channel_desc IN ('Direct Sales', 'Internet')`).

```
SELECT country_name, channel_desc, calendar_year,
       TO_CHAR(SUM(amount_sold), '9,999,999,999') AS OBRAT$
FROM sh.sales, sh.customers, sh.countries, sh.channels, sh.times
WHERE sales.time_id=times.time_id
      AND sales.cust_id =customers.cust_id
      AND customers.country_id = countries.country_id
      AND sales.channel_id = channels.channel_id
      AND customers.country_id IN ('JP', 'US')
     AND channels.channel_desc IN ('Direct Sales', 'Internet')
GROUP BY CUBE(country_name, channel_desc, calendar_year);
```

```
COUNTRY_NAME              CHANNEL_DESC  CALENDAR_YEAR OBRAT$
------------------------- ------------- ------------- -------------
                                                      152,854,054
                                                2000    56,353,187
                                                1998    44,255,360
                                                1999    52,245,506
                          Internet                      59,689,853
                          Internet              2000    21,957,589
                          Internet              1998    17,084,632
                          Internet              1999    20,647,632
                          Direct Sales                  93,164,201
                          Direct Sales          2000    34,395,599
                          Direct Sales          1998    27,170,728
                          Direct Sales          1999    31,597,874
Japan                                                     5,771,974
Japan                                           2000      2,082,689
Japan                                           1998      1,769,272
Japan                                           1999      1,920,013
Japan                     Internet                        2,307,437
Japan                     Internet              2000        834,632
Japan                     Internet              1998        716,541
```



Obr. 10.37 Schéma snowflake se třemi dimenzemi

```
Japan                          Internet              1999        756,264
Japan                          Direct Sales                    3,464,537
Japan                          Direct Sales          2000      1,248,056
Japan                          Direct Sales          1998      1,052,732
Japan                          Direct Sales          1999      1,163,748
United States of America                                     147,082,080
United States of America                             2000     54,270,499
United States of America                             1998     42,486,088
United States of America                             1999     50,325,494
United States of America       Internet                      57,382,416
United States of America       Internet              2000     21,122,956
United States of America       Internet              1998     16,368,092
United States of America       Internet              1999     19,891,368
United States of America       Direct Sales                  89,699,665
United States of America       Direct Sales          2000     33,147,542
United States of America       Direct Sales          1998     26,117,996
United States of America       Direct Sales          1999     30,434,126
```

Náš příkaz jazyka SQL utěšeně narostl, ale poskytuje nám přesně ty údaje, které požadujeme. Kdybychom měli rekapitulovat vazbu mezi tabulkou faktů a tabulkami dimenzí, tato vazba je definovaná následovně (zvýrazněné jsou cizí klíče v tabulce faktů):

```
...
WHERE sales.time_id  =times.time_id
  AND sales.cust_id  =customers.cust_id AND customers.country_id=countries.country_id
  AND sales.channel_id =channels.channel_id
...
```

Dimenze krychle můžeme pochopitelně i v tomto případě libovolně měnit.

Všimněme si, že některé řádky ve výpisu (označili jsme je tučným písmem) představují souhrnné informace, například za rok, pro zemi nebo prodejní kanál. Pro obchodní prezentace je někdy potřeba zobrazit údaje v trochu zhuštěném a přehlednějším formátu, tedy zobrazit jen tyto souhrnné informace. Pomocí funkce **GROUPING** můžeme vytvořit „masky" jednotlivých dimenzí.

```
SELECT country_name, channel_desc, calendar_year,
       TO_CHAR(SUM(amount_sold), '9,999,999,999') AS OBRAT$,
GROUPING (channel_desc) AS C,
GROUPING (calendar_year) AS R,
GROUPING (country_name) AS K
FROM sh.sales, sh.customers, sh.countries, sh.channels, sh.times
WHERE sales.time_id=times.time_id
      AND sales.cust_id =customers.cust_id
      AND customers.country_id = countries.country_id
      AND sales.channel_id = channels.channel_id
      AND customers.country_id IN ('JP', 'US')
     AND channels.channel_desc IN ('Direct Sales', 'Internet')
GROUP BY CUBE(country_name, channel_desc, calendar_year);
```

Získáme stejný výpis jako u předchozí „krychle" doplněný o masky dimenzí **R, C a K** (ROK, KANAL_PRODEJE a KRAJINA). Ukážeme jen jeho část. Tučně uvedené řádky mají v masce dimenzí alespoň jednu hodnotu 1.

COUNTRY_NAME	CHANNEL_DESC	CALENDAR_YEAR	OBRAT$	C	R	K
			152,854,054	1	1	1
		2000	56,353,187	1	0	1
		1998	44,255,360	1	0	1
		1999	52,245,506	1	0	1
	Internet		59,689,853	0	1	1
	Internet	2000	21,957,589	0	0	1
	Internet	1998	17,084,632	0	0	1
	Internet	1999	20,647,632	0	0	1
	Direct Sales		93,164,201	0	1	1
	Direct Sales	2000	34,395,599	0	0	1
	Direct Sales	1998	27,170,728	0	0	1
	Direct Sales	1999	31,597,874	0	0	1
Japan			5,771,974	1	1	0
Japan		2000	2,082,689	1	0	0
Japan		1998	1,769,272	1	0	0
Japan		1999	1,920,013	1	0	0
Japan	Internet		2,307,437	0	1	0
...						

Například řádky, které mají masky CRK(0,0,1), obsahují údaje o prodeji prostřednictvím Internetu a přímého prodeje v daném roce pro všechny země.

COUNTRY_NAME	CHANNEL_DESC	CALENDAR_YEAR	OBRAT$	C	R	K
....						
	Internet	2000	21,957,589	0	0	1
....						
	Direct Sales	1998	27,170,728	0	0	1
....						

Pokud si nechámc pomocí klauzule **HAVING** vypsat jen sloupce, kde maska CRK (KANAL_PRODEJE, ROK a KRAJINA) obsahuje alespoň jednu jedničku, získáme mnohem přehlednější „řidší krychli", která obsahuje jen požadované souhrnné údaje (masky sloupců jsme z důvodu přehlednosti výpisu vynechali).

```
SELECT country_name, channel_desc, calendar_year,
       TO_CHAR(SUM(amount_sold), '9,999,999,999') AS OBRAT$,
GROUPING (channel_desc) AS C,
GROUPING (calendar_year) AS R,
GROUPING (country_name) AS K
FROM sh.sales, sh.customers, sh.countries, sh.channels, sh.times
WHERE sales.time_id=times.time_id
      AND sales.cust_id =customers.cust_id
      AND customers.country_id = countries.country_id
      AND sales.channel_id = channels.channel_id
```

```
      AND customers.country_id IN ('JP', 'US')
     AND channels.channel_desc IN ('Direct Sales', 'Internet')
GROUP BY CUBE(country_name, channel_desc, calendar_year)
HAVING (GROUPING (country_name)=1) OR
       (GROUPING (channel_desc)=1) OR
       (GROUPING (calendar_year)=1);
```

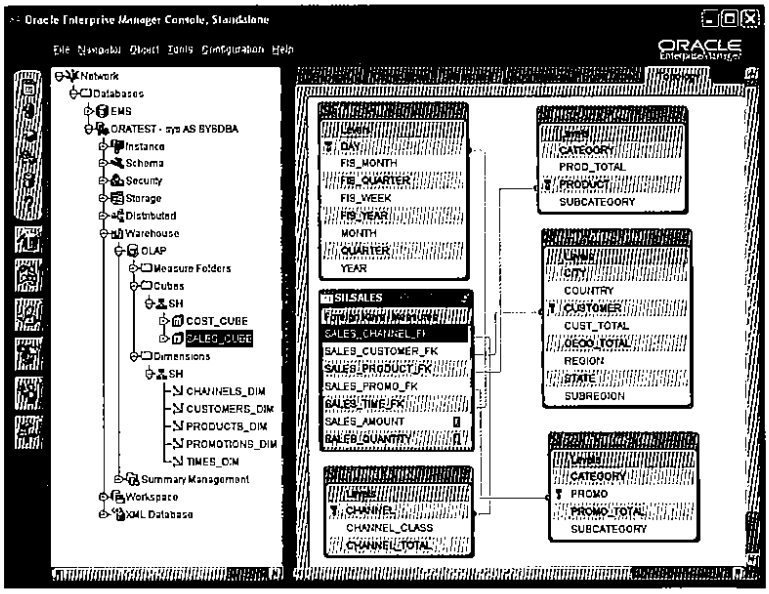
COUNTRY_NAME	CHANNEL_DESC	CALENDAR_YEAR	OBRAT$
	Internet	2000	21,957,589
	Internet	1998	17,084,632
	Internet	1999	20,647,632
	Direct Sales	2000	34,395,599
	Direct Sales	1998	27,170,728
	Direct Sales	1999	31,597,874
Japan		2000	2,082,689
United States of America		2000	54,270,499
		2000	56,353,187
Japan		1998	1,769,272
United States of America		1998	42,486,088
		1998	44,255,360
Japan		1999	1,920,013
United States of America		1999	50,325,494
		1999	52,245,506
	Internet		59,689,853
	Direct Sales		93,164,201
Japan			5,771,974
United States of America			147,082,080
			152,854,054
Japan	Internet		2,307,437
United States of America	Internet		57,382,416
Japan	Direct Sales		3,464,537
United States of America	Direct Sales		89,699,665

Pokud to potřebujeme, můžeme analyzovat údaje jen za určité časové období, například za rok 2000, nebo jen pro konkrétní prodejní kanál, případně pro konkrétní zemi.

Problematiku vytváření jednoduchých multidimenzionálních krychlí pomocí příkazů jazyka SQL zadávaných například pomocí konzolové aplikace SQL*Plus Worksheet máme zvládnutou. V reálných aplikacích s gigabajtovými až terabajtovými databázemi a datovými sklady používáme k vytváření krychlí specializované nástroje, které vypočtenou krychli uloží do samostatné databáze a umožňují všechny operace s ní, například výměnu dimenzí, masky a podobně, provádět pomocí komfortního uživatelského rozhraní. Databáze Oracle 9i má už podporu OLAP včetně serveru OLAP zabudovanou v sobě a tato podpora se při standardní instalaci nainstaluje společně s databází. (*Licenční podmínky a poplatky za používání jednotlivých komponent v této publikaci neprobíráme.*)

Přístup k objektům OLAP prostřednictvím aplikace Oracle Enterprise Manager

Aplikaci Oracle Enterprise Manager (OEMC) jsme už použili pro seznámení se s dimenzemi cvičné krychle SALES_CUBE. V pravém okně jsou v jednotlivých záložkách podrobné údaje o objektech v těchto složkách, tedy měrných jednotkách, krychlích a dimenzích. Ve složce OLAP jsou dvě cvičné krychle COST_CUBE a SALES_CUBE. Když se podíváme na topologii cvičné krychle SALES_CUBE, vidíme, že je typickým příkladem hvězdicového schématu, i když mezi tabulkami dimenzí nejsou žádné relační vztahy. OEMC nám umožní vytváření dimenzí a krychlí pomocí průvodce vytvořením dimenze (Dimension Wizard) a průvodce vytvořením krychle (Cube Wizard).



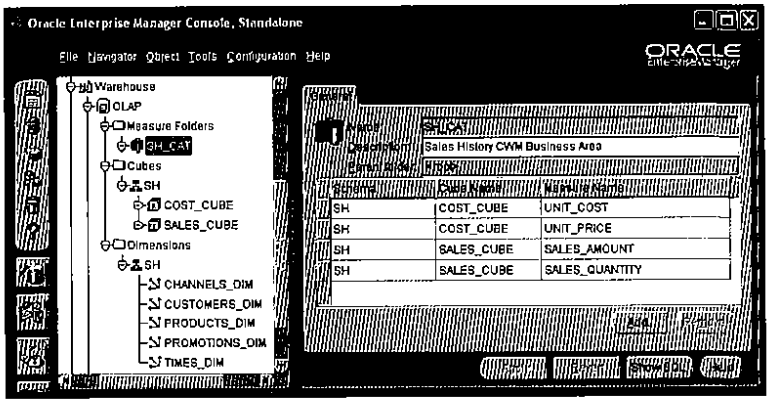
Obr. 10.38 Oracle Enterprise Manager – složka Warehouse | OLAP

V kořenové složce Warehouse | OLAP příslušné databáze jsou hierarchicky uspořádané podsložky

- Measure Folders,
- Cubes,
- Dimensions.

Measure Folders

Složka Measure Folders obsahuje údaje o měrných jednotkách obchodování. Pro cvičnou krychli COST_CUBE jsou definovány dvě měrné jednotky UNIT_COST a UNIT_PRICE. Pro druhou cvičnou krychli SALES_CUBE jsou definovány měrné jednotky SALES_AMOUNT a SALES_QUANTITY.



Obr. 10.39 Složka Measure Folders

Cubes

Složka obsahuje symboly všech krychlí OLAP, které se nachází v analytické databázi. Implicitně se po nainstalování v této složce nacházejí krychle COST_CUBE a SALES_CUBE.

Dimensions

Složka obsahuje ikony všech dimenzí v analytické databázi.

Průvodce vytvořením dimenze (Dimension Wizard)

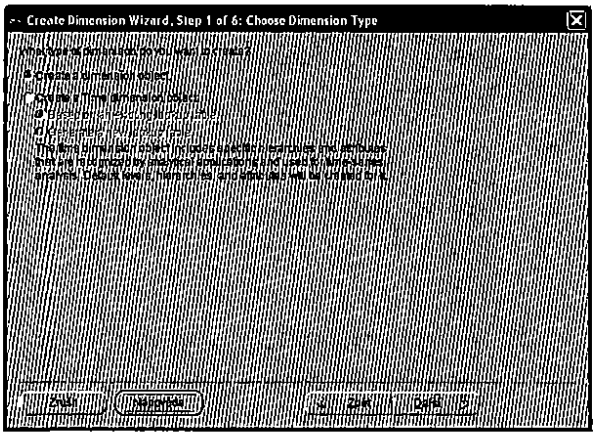
Tento průvodce se skládá ze šesti po sobě logicky uspořádaných kroků. Aktivuje se z aplikace Oracle Enterprise Manager Console spuštěné z kontextové nabídky. V pracovním okně EMC klepneme levým tlačítkem myši na složku *Dimensions* a aktivujeme volbu *Create Using Wizard*.

Předvedeme postup při vytváření „klasické" dimenze, v našem případě nad tabulkou SH.PRODUCTS, a časové dimenze nad tabulkou SH.TIMES. Hierarchii produktové dimenze jsme navrhli třístupňovou.

- KATEGORIE sloupec PROD_CATEGORY
  - SUBKATEGORIE sloupec PROD_SUBCATEGORY
    - PRODUKT sloupec PROD_NAME

Dimension Wizard krok 1 – výběr typu dimenze

V prvním kroku je potřeba zvolit typ dimenze, ať už se bude jednat o časovou, případně „klasickou" dimenzi. Naše dimenze bude klasická, proto označíme volbu *Create a dimension object*.



Obr. 10.40 Dimension Wizard krok 1 – výběr typu dimenze

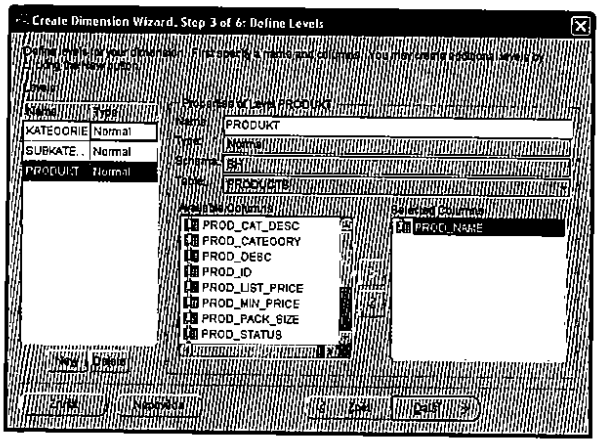
Dimension Wizard krok 2 – pojmenování a výběr schématu dimenze

V druhém kroku se zobrazí jednoduchý dialog, ve kterém zadáme jen dva parametry, a sice název dimenze, v našem případě PRODUKTOVA_DIM, a uživatelské schéma. Slovo schéma nemá v tomto případě nic společného se schématem star nebo schématem typu snowflake. Pod pojmem schéma se v tomto případě v Oracle myslí uživatelské schéma, tedy společný přístup pro uživatele k objektům. V našem případě jsme sice byli přihlášeni jako správce systému SYS, ale pracujeme nad objekty ve schématu SH (SALES HISTORY), takže i naše schéma pro dimenzi bude SH.

Dimension Wizard krok 3 – definování úrovní dimenze

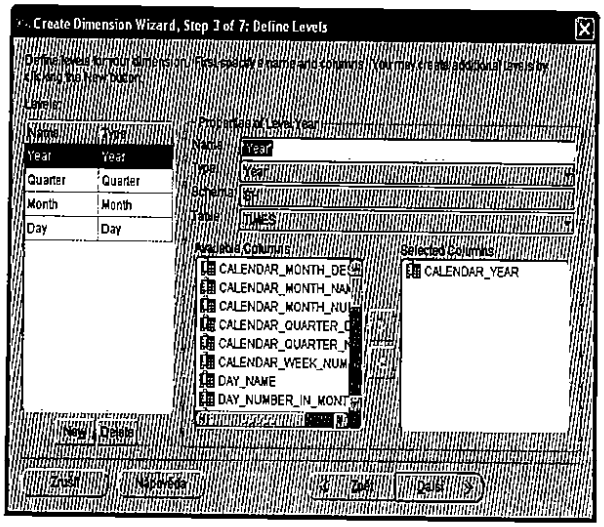
Po zadání názvu dimenze specifikujeme v tradičním přesouvacím dialogu pomocí přesouvání názvů sloupců tabulky SH.PRODUCT jednotlivé úrovně dimenze. Novou úroveň vytvoříme pokaždé tlačítkem **New**.

Po návrhu všech tří úrovní produktové dimenze přejdeme pomocí tlačítka Další do **4. kroku** průvodce, kde v případě potřeby definujeme atributy úrovní.



Obr. 10.41 Dimension Wizard krok 3 – definování úrovní dimenze

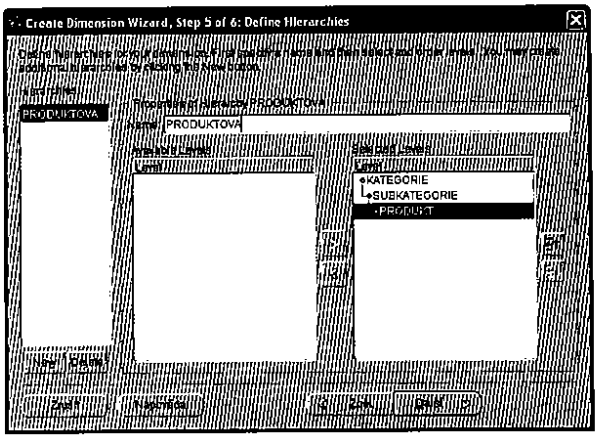
Pokud bychom v prvním kroku definovali, že chceme vytvořit časovou dimenzi, nabídl by nám průvodce ve třetím kroku typický návrh úrovní časové dimenze (ROK, KVARTAL, MESIC, DEN).



Obr. 10.42 Dimension Wizard krok 3 – definování úrovní časové dimenze

Dimension Wizard krok 5 – definování hierarchie úrovní dimenze

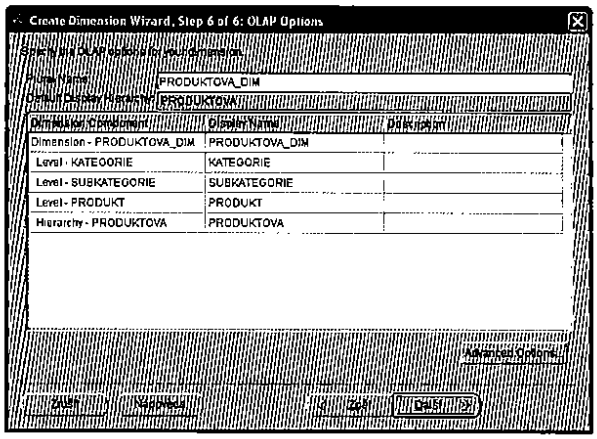
Následuje stanovení hierarchie jednotlivých úrovní navrhované dimenze. Z dostupných úrovní v levém okně dialogu přesuneme jednotlivé úrovně ve stanoveném pořadí do pravého okna. V pravém okně máme zobrazenou hierarchickou strukturu úrovní. Pomocí tlačítek vedle pravého okna je možno „pohybovat" jednotlivými úrovněmi směrem nahoru nebo dolů.



Obr. 10.43 Dimension Wizard krok 5 – definování hierarchie úrovní dimenze

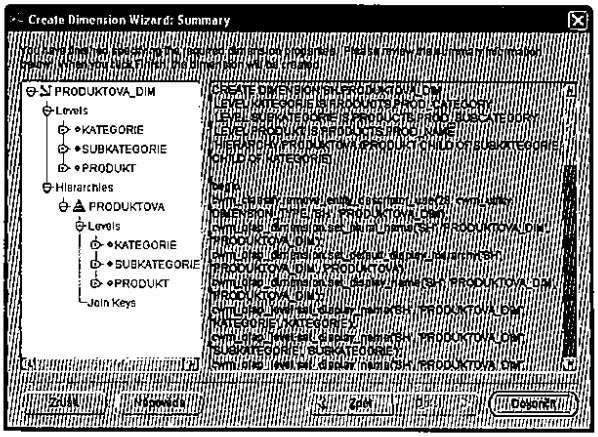
Dimension Wizard krok 6 – definování hierarchie úrovní dimenze

Pokud jsme zadali český název dimenze, dočkáme se v posledním šestém kroku transformace tohoto názvu do množného čísla podle pravidel anglického jazyka, tedy například z názvu dimenze PRODUKTY vytvoří slovo PRODUKTYS. Naštěstí se to dá v editačním okně změnit. Na náš název PRODUKTOVA_DIM si generátor množného čísla raději ani netroufl.



Obr. 10.44 Dimension Wizard krok 6 – zobrazení parametrů navržené dimenze

Následuje dialog s rekapitulací parametrů námi navržené dimenze.



Obr. 10.45 Dimension Wizard – rekapitulace

Podle složitosti vygenerovaného příkazu CREATE DIMENSION... asi nejlépe oceníme práci, kterou za nás průvodce při vytváření dimenze provedl (a to je ještě náš výpis ve snaze šetřit naše lesy zkrácený na třetinu).

```
CREATE DIMENSION SH.PRODUKTOVA_DIM
 LEVEL KATEGORIE IS PRODUCTS.PROD_CATEGORY
 LEVEL SUBKATEGORIE IS PRODUCTS.PROD_SUBCATEGORY
 LEVEL PRODUKT IS PRODUCTS.PROD_NAME
 HIERARCHY PRODUKTOVA (PRODUKT CHILD OF SUBKATEGORIE CHILD OF KATEGORIE)

begin
cwm_classify.remove_entity_descriptor_use(28, cwm_utility.DIMENSION_TYPE, 'SH', 'PRODUKTOVA_DIM');
cwm_olap_dimension.set_plural_name('SH', 'PRODUKTOVA_DIM', 'PRODUKTOVA_DIM');
cwm_olap_dimension.set_default_display_hierarchy('SH', 'PRODUKTOVA_DIM', 'PRODUKTOVA');
cwm_olap_dimension.set_display_name('SH', 'PRODUKTOVA_DIM', 'PRODUKTOVA_DIM');
cwm_olap_level.set_display_name('SH', 'PRODUKTOVA_DIM', 'KATEGORIE', 'KATEGORIE');
cwm_olap_level.set_display_name('SH', 'PRODUKTOVA_DIM', 'SUBKATEGORIE', 'SUBKATEGORIE');
cwm_olap_level.set_display_name('SH', 'PRODUKTOVA_DIM', 'PRODUKT', 'PRODUKT');
cwm_olap_hierarchy.set_display_name('SH', 'PRODUKTOVA_DIM', 'PRODUKTOVA', 'PRODUKTOVA');
cwm_olap_dim_attribute.create_dimension_attribute('SH', 'PRODUKTOVA_DIM', 'Long_Description',
...
...
commit;
end;
```

Když se podíváme na torzo našeho výpisu, asi nás napadne, že věnovat se návrhu dimenzí pomocí příkazů jazyka SQL CREATE DIMENSION by bylo velmi pracné a neefektivní. Při posuzování složitosti vygenerovaného příkazu si ale musíme uvědomit, že takto jej vygeneroval stroj. Jak uvidíme za chvíli, použití příkazu CREATE DIMENSION je vcelku jednoduché a přehledné, takže někdy můžeme dát klidně přednost tomuto postupu. Podívejme se na syntaktický předpis:

```
CREATE DIMENSION [schema .] dimension level_clause [level_clause]...
{ hierarchy_clause | attribute_clause } [ hierarchy_clause | attribute_clause ]...;
```

a jednoduchý příklad vytvoření produktové dimenze:

```
CREATE DIMENSION produktova_dimenze
        LEVEL product           IS (products.prod_id)
        LEVEL subcategory       IS (products.prod_subcategory)
        LEVEL category          IS (products.prod_category)
  HIERARCHY prod_rollup
  (
      product         CHILD OF
      subcategory     CHILD OF
      category
 );
```

Pokud chceme definovat i atributy dimenzí, rozšíříme náš příkaz například takto:

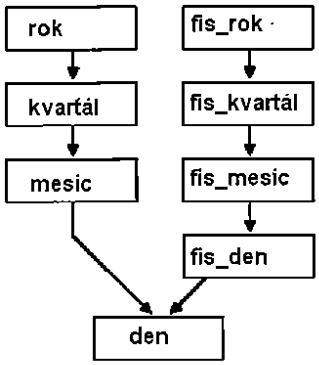
```
CREATE DIMENSION produktova_dimenze
        LEVEL product           IS (products.prod_id)
        LEVEL subcategory       IS (products.prod_subcategory)
        LEVEL category          IS (products.prod_category)
  HIERARCHY prod_rollup
  (
      product         CHILD OF
      subcategory     CHILD OF
      category
 )
ATTRIBUTE product DETERMINES
        (products.prod_name, products.prod_desc,
         prod_weight_class, prod_unit_of_measure,
         prod_pack_size,prod_status, prod_list_price, prod_min_price)
ATTRIBUTE subcategory DETERMINES (prod_subcategory, prod_subcat_desc)
ATTRIBUTE category DETERMINES (prod_category, prod_cat_desc);
```

Příkaz pro vytvoření zákaznické (regionální) dimenze, která je vytvořena nad dvěma relačně svázanými tabulkami (`JOIN KEY (customers.country_id) REFERENCES country`), by byl

```
CREATE DIMENSION customers_dim
   LEVEL customer   IS (customers.cust_id)
   LEVEL city       IS (customers.cust_city)
   LEVEL state      IS (customers.cust_state_province)
   LEVEL country    IS (countries.country_id)
   LEVEL subregion  IS (countries.country_subregion)
   LEVEL region     IS (countries.country_region)
   HIERARCHY geog_rollup (
      customer      CHILD OF
      city          CHILD OF
      state         CHILD OF
      country       CHILD OF
      subregion     CHILD OF
      region
   JOIN KEY (customers.country_id) REFERENCES country
   )
   ATTRIBUTE customer DETERMINES
   (cust_first_name, cust_last_name, cust_gender,
    cust_marital_status, cust_year_of_birth,
    cust_income_level, cust_credit_limit)
   ATTRIBUTE country DETERMINES (countries.country_name)
;
```

Vytvoření časové dimenze

A nakonec si ukážeme příklad příkazu pro vytvoření časové dimenze se dvěma hierarchickými úrovněmi – kalendářní a fiskální.

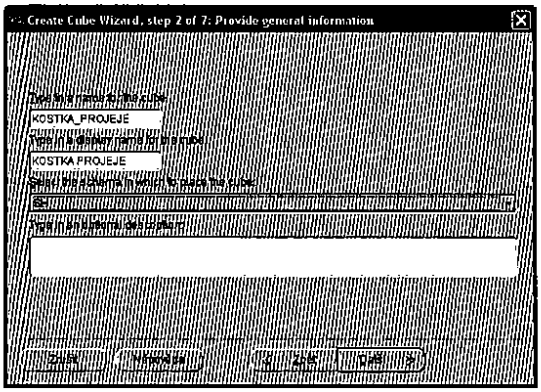


Obr. 10.46 Dvě hierarchie časové dimenze

```
CREATE DIMENSION times_dim
   LEVEL day          IS TIMES.TIME_ID
   LEVEL month        IS TIMES.CALENDAR_MONTH_DESC
   LEVEL quarter      IS TIMES.CALENDAR_QUARTER_DESC
   LEVEL year         IS TIMES.CALENDAR_YEAR
   LEVEL fis_week     IS TIMES.WEEK_ENDING_DAY
   LEVEL fis_month    IS TIMES.FISCAL_MONTH_DESC
   LEVEL fis_quarter IS TIMES.FISCAL_QUARTER_DESC
   LEVEL fis_year     IS TIMES.FISCAL_YEAR
   HIERARCHY cal_rollup    (
             day      CHILD OF
             month    CHILD OF
             quarter CHILD OF
             year
   )
   HIERARCHY fis_rollup    (
             day          CHILD OF
             fis_week     CHILD OF
             fis_month    CHILD OF
             fis_quarter CHILD OF
             fis_year
   ) ATTRIBUTE...
```

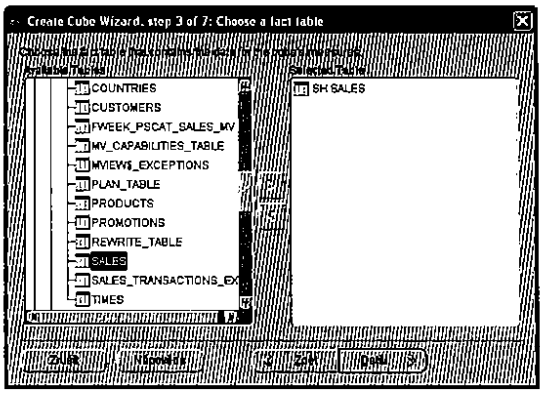
Průvodce vytvořením krychle (Cube Wizard)

Po vytvoření jednotlivých dimenzí můžeme přejít k vytvoření krychle. Průvodce vytvořením krychle se spouští také v aplikaci Oracle Enterprise Manager Console výběrem volby *Create Using Wizard* z kontextové nabídky složky *Cubes* a skládá se ze sedmi kroků. Po tradičním uvítání ve druhém kroku zadáme název krychle a vybereme schéma. V našem případě jsme zvolili název KRYCHLE_PRODEJE ve schématu SH.



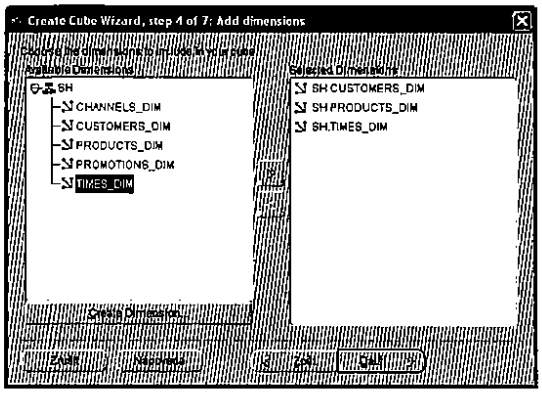
Obr. 10.47 Cube Wizard krok 2 – základní informace o vytvářené krychli

V následujících dvou krocích vybereme tabulku faktů (tabulka SH.SALES):



Obr. 10.48 Cube Wizard krok 3 – výběr tabulky faktů

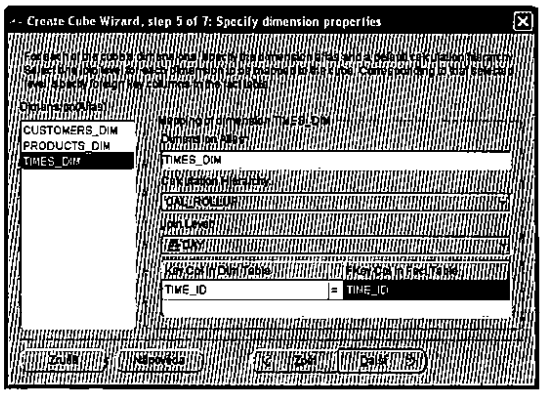
a dimenze. V našem případě jsme vybrali časovou, produktovou a zákaznickou (regionální) dimenzi.



Obr. 10.49 Cube Wizard krok 4 – výběr dimenzí

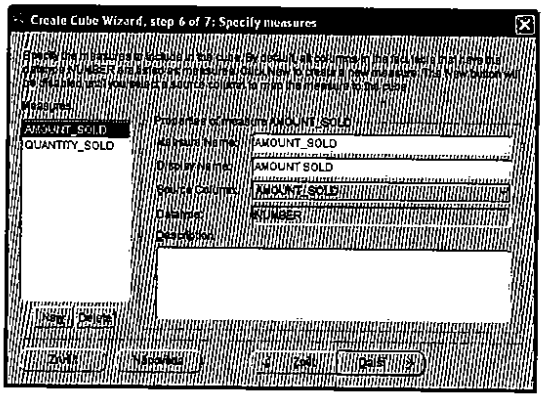
V pátém kroku můžeme upřesnit vlastnosti použitých dimenzí a určit jejich vztah k tabulce faktů. Pokračovat můžeme až po zadání cizích klíčů pro všechny dimenze. V našem případě definujeme cizí klíče do tabulky faktů.

- TIME_ID pro časovou dimenzi.
- CUST_ID pro zákaznickou (regionální) dimenzi.
- PROD_ID pro produktovou dimenzi.



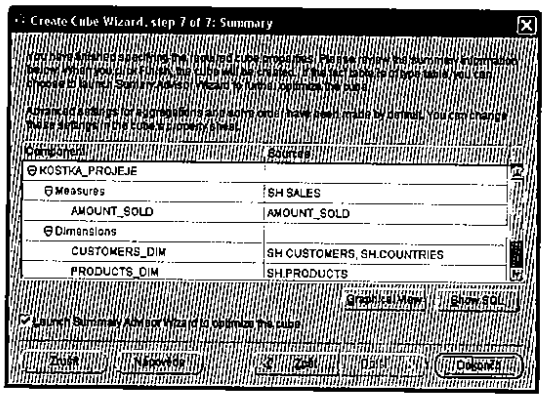
Obr. 10.50 Cube Wizard krok 5 – vlastnosti dimenzí

V šestém kroku zadáme měrné jednotky (sloupce z tabulky faktů, jejichž agregované souhrny budou umístěné v navrhované krychli OLAP). Průvodce nám nabídne všechny sloupce numerických datových typů, které jsme dosud nepoužili jako cizí klíče do tabulek dimenzí. V našem případě bylo potřeba vymazat sloupec PROMO_ID a zůstaly nám sloupce AMOUNT_SOLD a QUANTITY_SOLD.

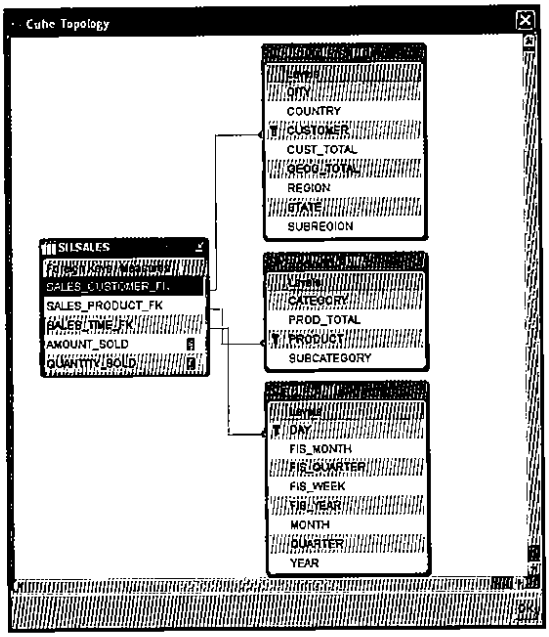


Obr. 10.51 Cube Wizard krok 6 – měrné jednotky z tabulky faktů

V posledním kroku – souhrnném dialogu – si můžeme nechat zobrazit topologii námi navržené krychle.



Obr. 10.52 Cube Wizard krok 7 – souhrnný dialog

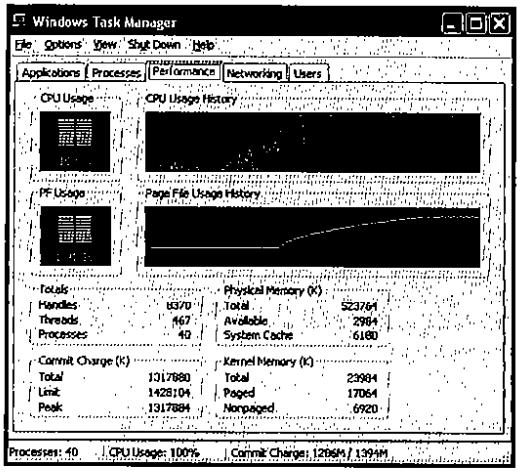


Obr. 10.53 Cube Wizard – topologie navržené krychle

Nebo také vygenerovaný příkaz CREATE CUBE pro vytvoření krychle. Příkaz samozřejmě nezapře, že byl vygenerován strojem, má totiž asi dvě strany.

```
declare CUSTOMERS_DIM number;
            PRODUCTS_DIM number;
           TIMES_DIM number;
            tmp number;
begin
CWM_OLAP_CUBE.Create_Cube('SH', KRYCHLE _PRODEJE', ' KRYCHLE PRODEJE', '');
CUSTOMERS_DIM := CWM_OLAP_CUBE.Add_Dimension('SH', ' KRYCHLE _PRODEJE', 'SH',
'CUSTOMERS_DIM', 'CUSTOMERS_DIM');
CWM_OLAP_CUBE.Set_Default_Calc_Hierarchy('SH', ' KRYCHLE _PRODEJE', 'CUST_ROLLUP',
'SH', 'CUSTOMERS_DIM', 'CUSTOMERS_DIM');
...
...
commit;
end;
```

Všimněme si ještě v souhrnném dialogu (sedmý krok průvodce) zaškrtávací položky Launch Summary Advisor Wizard for optimize the Cube. Do této akce se můžeme pustit jen v případě, že máme dostatek paměti. Pojem dostatek paměti v tomto případě znamená, že gigabajt je málo. V našem případě požadavky na paměť výrazně překročily 3 GB. Na obrázku (stav 1,91 GB) je jen začátek optimalizace...



Obr. 10.54 Spotřeba paměti při aktivování průvodce Summary Advisor Wizard

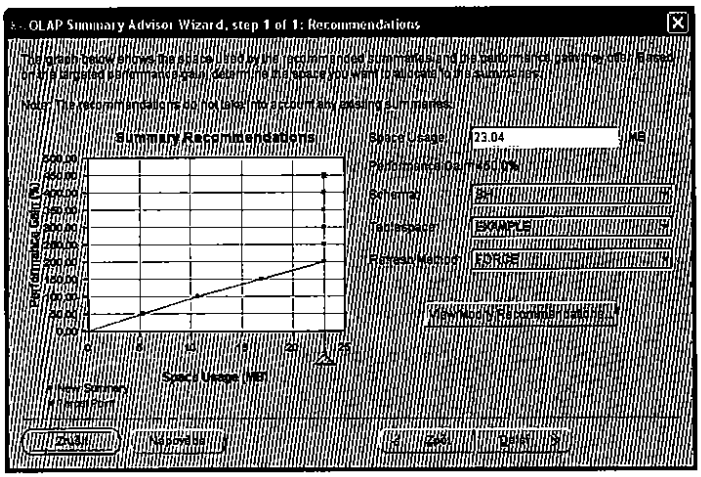
Takže optimalizace se na průměrně vybavené vývojářské pracovní stanici dělat skutečně nedá.

Pokud si to chceme vyzkoušet, můžeme například vytvořit novou menší tabulku faktů SH.SALES1 se stejnou strukturou, jako má tabulka SH.SALES.

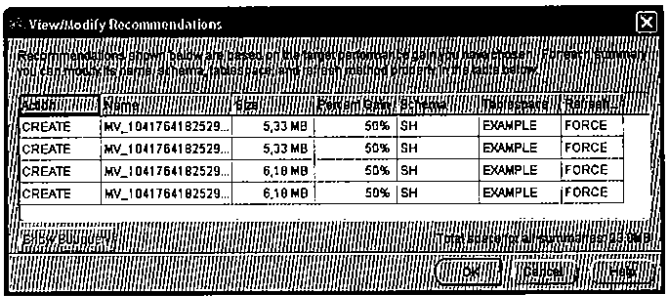
Do tabulky přeneseme údaje za kratší časové období, třeba za jeden měsíc příkazem

```
INSERT INTO sh.sales1
      SELECT * from sh.sales WHERE time_id > TO_DATE('01.11.2000', 'dd.mm.YYYY');

39312 řádek vytvořeno.
```

Nad takto vytvořenou tabulkou faktů vytvoříme novou krychli.

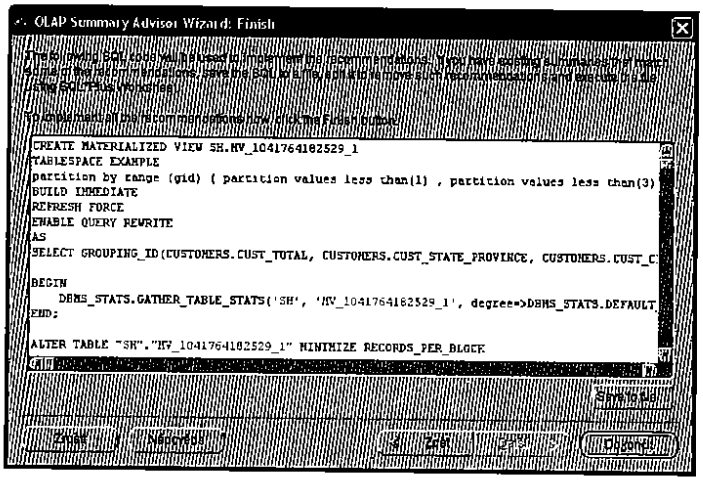


Obr. 10.55 Summary Advisor Wizard – doporučení

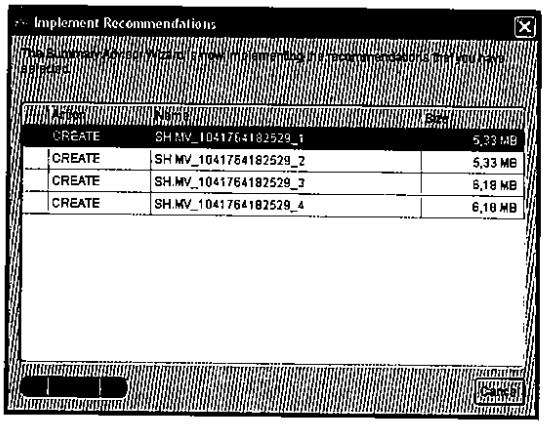


Obr. 10.56 Summary Advisor Wizard – detaily doporučení

Výsledkem práce optimalizátoru je skript jazyka SQL pro vytvoření materializovaných pohledů a bitmapových indexů.

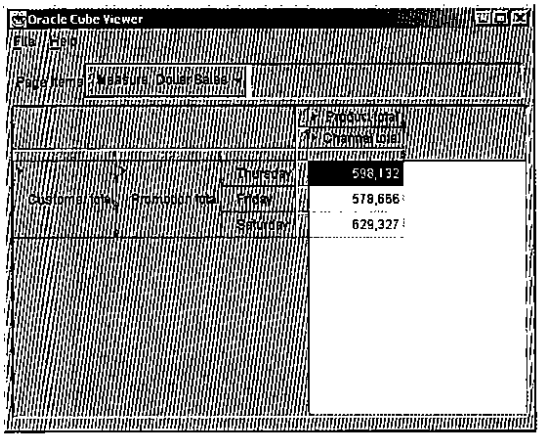


Obr. 10.57 Summary Advisor Wizard – příkazy jazyka SQL pro vytvoření optimalizačních struktur

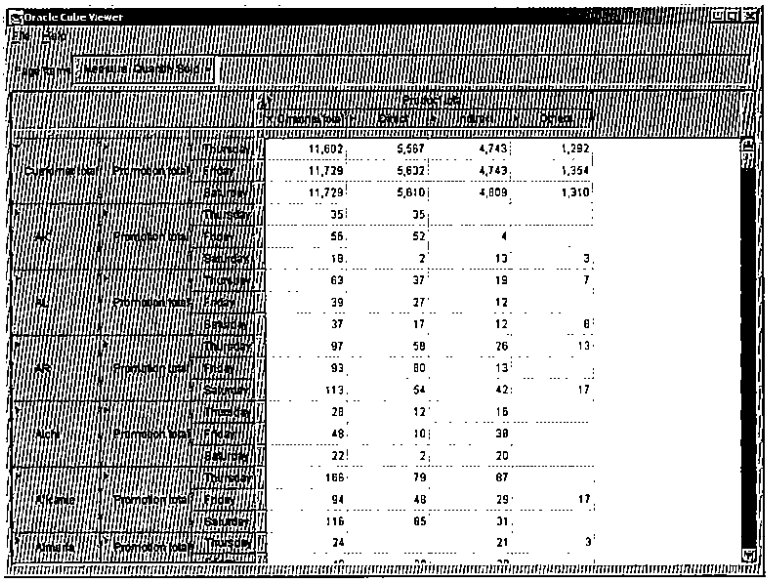


Obr. 10.58 Summary Advisor Wizard – realizace doporučených úkonů

Po spuštění serveru OLAP (jeho konfigurace pro Windows není příliš jednoduchá záležitost, v tomto případě doporučujeme postupovat důsledně podle dokumentace) můžeme s vytvořenou krychlí pracovat, zavrtávat se do jednotlivých dimenzí, stránkovat podle jednotlivých dimenzí a podobně.



Obr. 10.59 Oracle Cube Viewer



Obr. 10.60 Oracle Cube Viewer – rozvinutí úrovní dimenzí

V kapitole věnované databázi Oracle jsme si ukázali příklady analýzy OLAP přímo v databázi Oracle 9i Release 2. Takový je trend firmy, který prezentují její nejvyšší představitelé. V oblasti drahých softwarových technologií business intelligence se nezavádějí nové technologie takovým tempem jako teenageři mění grafické karty ve svých herních PC. Proto se určitě budeme ještě několik let setkávat se staršími, ale osvědčenými produkty společnosti Oracle z této oblasti – softwarovými nástroji Oracle Expres a Oracle Discoverer.

Příloha 1

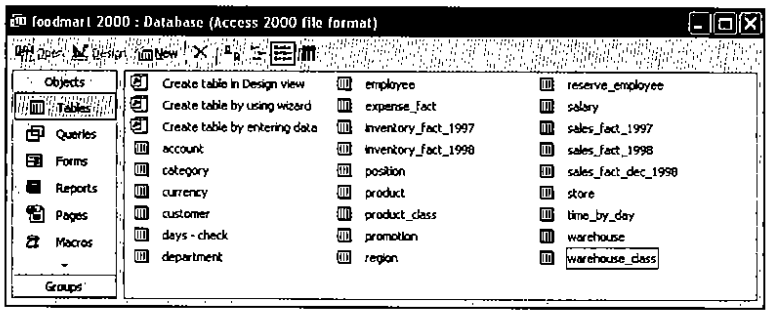
Cvičná databáze FoodMart 2000

Cvičná databáze **FoodMart** je dodávaná s SQL Serverem 2000 (nainstaluje se při instalaci analytických služeb). FoodMart je databáze velkého potravinářského obchodního řetězce, který má své obchody v USA, Mexiku a Kanadě. Je potřeba rozlišovat transakční databázi OLTP a analytickou databázi OLAP. Abychom mohli tuto cvičnou databázi s úspěchem použít, měli bychom ji důkladně poznat. V reálné praxi toto vývojáři nehrozí, v procesu vytváření a ladění ETL poznáme strukturu databáze nebo datového skladu mnohem detailněji, než bychom si možná přáli ☺.

Transakční databáze OLTP FoodMart

Fyzicky se transakční databáze OLTP FoodMart 2000 nainstaluje jako souborová databáze programu Microsoft Access, konkrétně soubor *FoodMart 2000.mdb* v adresáři

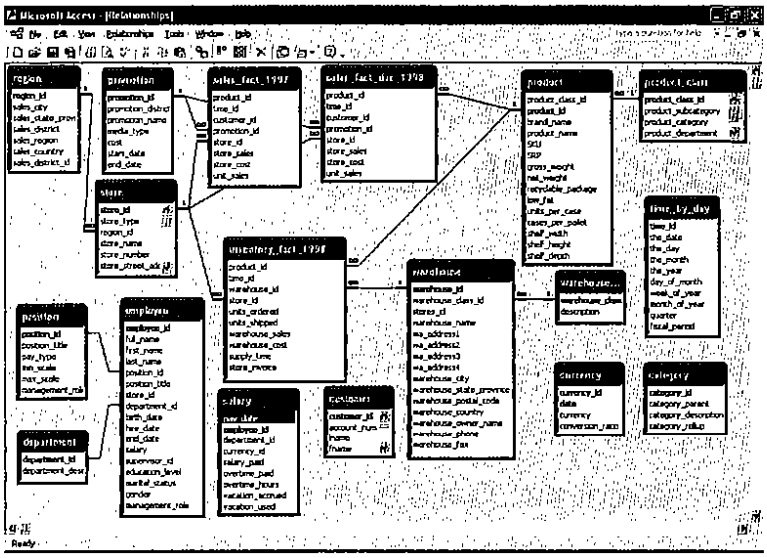
C:\Program Files\Microsoft Analysis Services\Samples.



Obr. P.1 Databáze OLTP FoodMart 2000.mdb

V předchozí verzi Analytických služeb dodávané s Microsoft SQL Serverem 7.0 byla databáze FoodMart uložená v souboru *FoodMart.mdb*.

Diagram databáze FoodMart 2000 můžeme vykreslit buď pomocí programu Microsoft Access, nebo pomocí programu Microsoft Visio 2002.



Obr. P.2 Diagram databáze OLTP FoodMart 2000.mdb zobrazený v programu MS Access

Tabulka account

Struktura tabulky.

Sloupec	Typ
account_id	Long Integer
account_parent	Long Integer
account_description	Text
account_type	Text
account_rollup	Text
Custom Members	Text

Tabulka implicitně obsahuje 11 záznamů.

Příklad uložených údajů (kompletní výpis obsahu tabulky):

```
account_id  account_parent account_description        account_type account_rollup Custom Members
----------- -------------- ---------------------------- ------------ ------------- ---------------
1000        NULL           Assets                       Asset        -             NULL
2000        NULL           Liabilities                  Liability    -             NULL
3000        5000           Net Sales                    Income       +             NULL
3100        3000           Gross Sales                  Income       +     LookUpCube("[Sales]",
                                                                            "(Measures.[Store Sales],
                                                                          "+time.currentmember.UniqueName+",
                                                                        "Store.currentmember.UniqueName+")")
3200        3000           Cost of Goods Sold           Income       -             NULL
4000        5000           Total Expense                Expense      -             NULL
4100        4000           General & Administration     Expense      +             NULL
4200        4000           Information Systems          Expense      +             NULL
4300        4000           Marketing                    Expense      +             NULL
4400        4000           Lease                        Expense      +             NULL
5000        NULL           Net Income                   Income       +             NULL
```

Všimněme si příkazu

```
LookUpCube("[Sales]","(Measures.[Store Sales],"+time.currentmember.UniqueName+","+
Store.currentmember.UniqueName+")")
```

v sloupci `Custom Members`.

Tabulka category

Struktura tabulky.

Sloupec	Typ
category_id	Text
category_parent	Text
category_description	Text
category_rollup	Text

Tabulka implicitně obsahuje 4 záznamy.

Příklad uložených údajů (kompletní výpis obsahu tabulky):

```
category_id     category_parent category_description           category_rollup
--------------- --------------- ------------------------------ ---------------
ACTUAL          NULL            Current Year's Actuals         NULL
ADJUSTMENT      NULL            Adjustment for Budget input    NULL
BUDGET          NULL            Current Year's Budget          NULL
FORECAST        NULL            Forecast
```

Tabulka currency

Struktura tabulky.

Sloupec	Typ
currency_id	Long Integer
date	Date/Time
currency	Text
conversion_ratio	Currency

Tabulka implicitně obsahuje 72 záznamů.

Příklad uložených údajů (zkrácено):

```
currency_id date        currency        conversion_ratio
----------- ----------- --------------- --------------------
1           1997-01-01  USD             1.0000
1           1997-02-01  USD             1.0000
...
2           1997-01-01  Mexican Peso    .1000
2           1997-02-01  Mexican Peso    .1000
...
3           1997-08-01  Canadian Dollar .6900
3           1997-09-01  Canadian Dollar .6900
...
```

Tabulka customer

Struktura tabulky.

Sloupec	Typ
customer_id	Long Integer
account_num	Double
lname	Text
fname	Text
mi	Text
address1	Text
address2	Text
address3	Text
address4	Text
city	Text
state_province	Text
postal_code	Text
country	Text

Sloupec	Typ
customer_region_id	Long Integer
phone1	Text
phone2	Text
birthdate	Date/Time
marital_status	Text
yearly_income	Text
gender	Text
total_children	Integer
num_children_at_home	Integer
education	Text
date_accnt_opened	Date/Time
member_card	Text
occupation	Text
houseowner	Text
num_cars_owned	Integer

Tabulka implicitně obsahuje 10 281 záznamů.

Příklad uložených údajů:

customer : Table

customer_id	account_num	lname	fname	mi	address1	city	state_province	postal_code	country
1	87460024688	Nowmer	Sheri	A	2433 Bailey Road	Oaxaca	Oaxaca	15057	Mexico
2	87470686299	Whelply	Derrick	I	2219 Dawing Avenue	Sooke	BC	17172	Canada
3	87475757600	Derry	Jeanne		7640 First Ave	Issaquah	WA	73960	USA
4	87900482201	Spence	Michael	J	337 Tosca Way	Burnaby	BC	74674	Canada
5	87514054179	Gutierrez	Maya		8658 Via Neruda	Novato	CA	67366	USA
6	87517782449	Damstra	Robert	F	1619 Stillman Court	Lynnwood	WA	90792	USA
7	87521172900	Kanagaki	Rebecca		2860 D Mt. Hood Circle	Tlaxiaco	Oaxaca	13343	Mexico
8	87539744377	Brunner	Kim	H	8064 Brodia Court	San Andres	DF	12942	Mexico
9	87544797658	Blumberg	Brenda	C	7560 Trees Drive	Richmond	BC	17256	Canada
10	87568712234	Stanz	Darren	M	1019 Kenwal Rd	Lake Oswego	OR	82017	USA
11	87572821378	Murrain	Jonathan	V	5423 Camby Rd	La Mesa	CA	39590	USA

Record: 1 of 10281

Obr. P.3 Příklad údajů uložených v tabulce Customer

Tabulka days – check

Struktura tabulky.

Sloupec	Typ
day	Long Integer
week_day	Text

Tabulka implicitně obsahuje 7 záznamů.

Příklad uložených údajů (kompletní výpis obsahu tabulky).

```
day         week_day
----------- ------------
1           Sunday
2           Monday
5           Thursday
4           Wednesday
3           Tuesday
6           Friday
7           Saturday
```

Tabulka department

Struktura tabulky.

Sloupec	Typ
department_id	Long Integer
department_description	Text

Tabulka implicitně obsahuje 12 záznamů.

Příklad uložených údajů (kompletní výpis obsahu tabulky):

```
department_id department_description
------------- ------------------------
1             HQ General Management
2             HQ Information Systems
3             HQ Marketing
4             HQ Human Resources
5             HQ Finance and Accounting
11            Store Management
14            Store Information Systems
15            Store Permanent Checkers
16            Store Temporary Checkers
17            Store Permanent Stockers
18            Store Temporary Stockers
19            Store Permanent Butchers
```

Tabulky employee a reserve_employee

Struktura tabulky (tabulka reserve_employee neobsahuje sloupec `management_role`).

Sloupec	Typ
employee_id	Long Integer
full_name	Text
first_name	Text
last_name	Text

Sloupec	Typ
position_id	Long Integer
position_title	Text
store_id	Long Integer
department_id	Long Integer
birth_date	Date/Time
hire_date	Date/Time
end_date	Date/Time
salary	Currency
supervisor_id	Long Integer
education_level	Text
marital_status	Text
gender	Text
management_role	Text

Tabulka employee implicitně obsahuje 1 155 záznamů.
Tabulka reserve_employee implicitně obsahuje 143 záznamů.

Příklad uložených údajů:

employee : Table

employee_id	full_name	first_name	last_name	position	position_title	store_id	departm	birth_date	salary	education_level
1	Sheri Nowmer	Sheri	Nowmer	1	President	0	1	25.8.1961	$80 000,00	Graduate Degree
2	Derrick Whelply	Derrick	Whelply	2	VP Country Ma	0	1	3.7.1915	$40 000,00	Graduate Degree
4	Michael Spence	Michael	Spence	2	VP Country Ma	0	1	20.6.1969	$40 000,00	Graduate Degree
5	Maya Gutierrez	Maya	Gutierrez	2	VP Country Ma	0	1	10.5.1951	$35 000,00	Bachelors Degree
6	Roberta Damstra	Roberta	Damstra	3	VP Information	0	2	8.10.1942	$25 000,00	Bachelors Degree
7	Rebecca Kanagaki	Rebecca	Kanagaki	4	VP Human Res	0	3	27.3.1949	$15 000,00	Bachelors Degree
8	Kim Brunner	Kim	Brunner	11	Store Manager	9	11	10.8.1922	$10 000,00	Bachelors Degree
9	Brenda Blumberg	Brenda	Blumberg	11	Store Manager	21	11	23.6.1979	$17 000,00	Graduate Degree
10	Darren Stanz	Darren	Stanz	5	VP Finance	0	5	26.8.1949	$50 000,00	Partial College
11	Jonathan Murrain	Jonathan	Murrain	11	Store Manager	1	11	20.6.1967	$15 000,00	Graduate Degree

Record: 1 of 1155

Obr. P.4 Příklad údajů uložených v tabulce Employee

Tabulka expense_fact

Struktura tabulky.

Sloupec	Typ
store_id	Long Integer
account_id	Long Integer
exp_date	Date/Time
time_id	Long Integer
category_id	Text
currency_id	Long Integer
amount	Currency

Tabulka implicitně obsahuje 2 400 záznamů.

Příklad uložených údajů:

```
store_id  account_id  exp_date    time_id   category_id  currency_id amount
--------- ----------- ----------- --------- ------------ ----------- -------
0         4100        1997-01-01  367       ACTUAL       1           942.0000
0         4100        1997-02-01  398       ACTUAL       1           942.0000
0         4100        1997-03-01  426       ACTUAL       1           942.0000
0         4100        1997-04-01  457       ACTUAL       1           942.0000
0         4100        1997-05-01  487       ACTUAL       1           942.0000
...
```

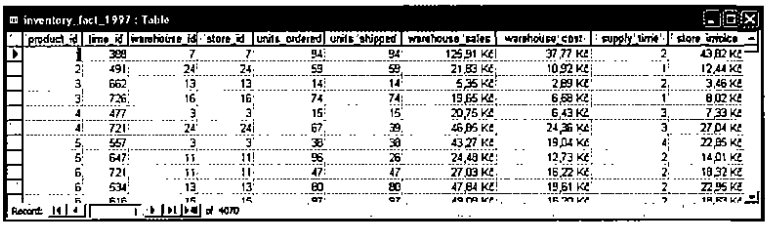
Tabulky inventory_fact_1997, inventory_fact_1998

Struktura tabulky.

Sloupec	Typ
product_id	Long Integer
time_id	Long Integer
warehouse_id	Long Integer
store_id	Long Integer
units_ordered	Double
units_shipped	Long Integer
warehouse_sales	Currency
warehouse_cost	Currency
supply_time	Integer
store_invoice	Currency

Tabulka inventory_fact_1997 implicitně obsahuje 4 070 záznamů.
Tabulka inventory_fact_1998 implicitně obsahuje 7 282 záznamů.

Příklad uložených údajů:



Obr. P.5 Příklad údajů uložených v tabulkách inventory_fact_199x

Tabulka position

Struktura tabulky.

Sloupec	Typ
position_id	Long Integer
position_title	Text
pay_type	Text
min_scale	Currency
max_scale	Currency
management_role	Text

Tabulka implicitně obsahuje 18 záznamů.

Příklad uložených údajů (kompletní výpis):

position_id	position_title	pay_type	min_scale	max_scale	management_role
1	President	Monthly	25000.0000	85000.0000	Senior Management
6	HQ Information Systems	Monthly	6700.0000	10000.0000	Middle Management
7	HQ Marketing	Monthly	5000.0000	8500.0000	Middle Management
8	HQ Human Resources	Monthly	5000.0000	6700.0000	Middle Management
9	HQ Finance and Accounting	Monthly	5000.0000	6700.0000	Middle Management
11	Store Manager	Monthly	8500.0000	17000.0000	Store Management
14	Store Information Systems	Monthly	6700.0000	10000.0000	Store Full Time Staff
15	Store Permanent Checker	Monthly	5000.0000	9000.0000	Store Full Time Staff
16	Store Temporary Checker	Hourly	20.0000	40.0000	Store Full Time Staff
17	Store Permanent Stocker	Monthly	3400.0000	6700.0000	Store Full Time Staff
18	Store Temporary Stocker	Hourly	20.0000	40.0000	Store Temp Staff
19	Store Permanent Butcher	Monthly	6700.0000	10000.0000	Store Full Time Staff
2	VP Country Manager	Monthly	20000.0000	40000.0000	Senior Management
3	VP Information Systems	Monthly	12500.0000	25000.0000	Senior Management
4	VP Human Resources	Monthly	10000.0000	20000.0000	Senior Management
5	VP Finance	Monthly	17000.0000	35000.0000	Senior Management
12	Store Assistant Manager	Monthly	6700.0000	12500.0000	Store Management
13	Store Shift Supervisor	Monthly	6700.0000	8500.0000	Store Management

Tabulka product

Struktura tabulky.

Sloupec	Typ
product_class_id	Long Integer
product_id	Long Integer
brand_name	Text
product_name	Text

Sloupec	Typ
SKU	Double
SRP	Currency
gross_weight	Single
net_weight	Single
recyclable_package	Yes/No
low_fat	Yes/No
units_per_case	Integer
cases_per_pallet	Integer
shelf_width	Single
shelf_height	Single
shelf_depth	Single

Tabulka implicitně obsahuje 1 560 záznamů.

Příklad uložených údajů:

product : Table

product_class_id	product_id	brand_name	product_name	SKU	SRP	gross_weight	net_weight	rec
30	1	Washington	Washington Berry Juice	90748583674	$2,65	8,39	6,39	
52	2	Washington	Washington Mango Drink	96516502499	$0,74	7,42	4,42	
52	3	Washington	Washington Strawberry Drink	58427771925	$0,83	13,1	11,1	
19	4	Washington	Washington Cream Soda	64412155747	$3,64	10,6	9,6	
19	5	Washington	Washington Diet Soda	95561191439	$2,19	6,66	4,66	
19	6	Washington	Washington Cola	29804842796	$1,15	15,8	13,8	
19	7	Washington	Washington Diet Cola	20191444754	$2,61	18	17	
30	8	Washington	Washington Orange Juice	69770532250	$2,59	8,97	6,97	
30	9	Washington	Washington Cranberry Juice	48395100474	$2,42	7,14	5,13	

Record: 3 of 1560

Obr. P.6 Příklad údajů uložených v tabulce Product

Tabulka product_class

Struktura tabulky.

Sloupec	Typ
product_class_id	Long Integer
product_subcategory	Text
product_category	Text
product_department	Text
product_family	Text

Tabulka implicitně obsahuje 110 záznamů.

Příklad uložených údajů:

```
product_class_id product_subcategory  product_category  product_department  product_family
---------------- -------------------- ----------------- ------------------- --------------
1                Nuts                 Specialty         Produce             Food
2                Shellfish            Seafood           Seafood             Food
3                Canned Fruit         Fruit             Canned Products     Food
4                Spices               Baking Goods      Baking Goods        Food
5                Pasta                Starchy Foods     Starchy Foods       Food
6                Yogurt               Dairy             Dairy               Food
7                Coffee               Dry Goods         Baking Goods        Drink
...
```

Tabulka promotion

Struktura tabulky.

Sloupec	Typ
promotion_id	Long Integer
promotion_district_id	Long Integer
promotion_name	Text
media_type	Text
cost	Double
start_date	Date/Time
end_date	Date/Time

Tabulka implicitně obsahuje 1 864 záznamů.

Příklad uložených údajů (zkráceno):

```
promotion_id promot promotion_name       media_type            cost     start_date end_date
------------ ------ -------------------- --------------------- -------- ---------- ----------
0            0      No Promotion         No Media              0.0      NULL       NULL
1            110    High Roller Savings  Product Attachment    14435.0  1996-01-03 1996-01-06
2            110    Green Light Special  Product Attachment    8907.0   1996-01-18 1996-01-20
3            110    Wallet Savers        Radio                 12512.0  1996-02-02 1996-02-05
4            110    Weekend Markdown     In-Store Coupon       11256.0  1996-02-13 1996-02-15
5            110    Bag Stuffers         Sunday Paper,Radio    12275.0  1996-02-28 1996-03-01
6            110    Save-It Sale         Daily Paper           9472.0   1996-03-14 1996-03-16
7            110    Fantastic Discounts  Sunday Paper,Radio,TV 14278.0  1996-03-29 1996-04-02
...
```

Tabulka region

Struktura tabulky.

Sloupec	Typ
region_id	Long Integer
sales_city	Text
sales_state_province	Text
sales_district	Text
sales_region	Text
sales_country	Text
sales_district_id	Long Integer

Tabulka implicitně obsahuje 110 záznamů.

Příklad uložených údajů:

region : Table

region_id	sales_city	sales_state_pro	sales_district	sales_region	sales_country	sales_district_id
0	None	None	No District	No Region	No Country	0
1	San Francisco	CA	San Francisco	Central West	USA	123
2	Mexico City	DF	Mexico City	Mexico Central	Mexico	118
3	Los Angeles	CA	Los Angeles	South West	USA	116
4	Guadalajara	Jalisco	Guadalajara	Mexico West	Mexico	114
5	Vancouver	BC	Vancouver	Canada West	Canada	128
6	Victoria	BC	Victoria	Canada West	Canada	129
7	San Diego	CA	San Diego	South West	USA	133

Record: 14 1 of 110

Obr. P.7 Příklad údajů uložených v tabulce Region

Tabulka salary

Struktura tabulky.

Sloupec	Typ
pay_date	Date/Time
employee_id	Long Integer
department_id	Long Integer
currency_id	Long Integer
salary_paid	Currency
overtime_paid	Currency
overtime_hours	Single
vacation_accrued	Single
vacation_used	Single

Tabulka implicitně obsahuje 21 252 záznamů.

Příklad uložených údajů (zkráceno):

pay_date	emplo	departm	curr	salary_paid	overtime	overtime_hours	vacation_acc	vacation_used
1997-01-01	1	1	1	72.0000	.0000	0.0	1.0	0.0
1997-01-01	2	1	1	36.0000	.0000	0.0	1.0	0.0
1997-01-01	6	2	1	22.5000	.0000	0.0	1.0	0.0
1997-01-01	7	3	1	13.5000	.0000	0.0	1.0	0.0
1997-01-01	10	5	1	45.0000	.0000	0.0	1.0	0.0
1997-01-01	20	1	1	27.0000	.0000	0.0	1.0	0.0
1997-01-01	21	1	1	31.5000	.0000	0.0	1.0	0.0

Tabulky sales_fact_1997, sales_fact_1998, sales_fact_dec_1998

Struktura tabulky.

Sloupec	Datový typ
product_id	Long Integer
time_id	Long Integer
customer_id	Long Integer
promotion_id	Long Integer
store_id	Long Integer
store_sales	Currency
store_cost	Currency
unit_sales	Double

Tabulka sales_fact_1997 implicitně obsahuje 86 837 záznamů.

Tabulka sales_fact_1998 implicitně obsahuje 164 558 záznamů.

Tabulka sales_fact_dec_1998 implicitně obsahuje 18 325 záznamů.

Příklad uložených údajů:

product_id	time_id	customer_id	promotion_id	store_id	store_sales	store_cost	unit_sales
337	371	6280	0	2	1.5000	.5100	2.0
1512	371	6280	0	2	1.6200	.6318	3.0
963	371	4018	0	2	2.4000	.7200	1.0
181	371	4018	0	2	2.7900	1.0323	3.0
1383	371	4018	0	2	5.1800	2.1756	2.0
1306	371	4018	0	2	7.4100	2.7417	3.0
1196	371	1418	0	2	5.8400	1.9856	2.0
...							

Tabulka store

Struktura tabulky.

Sloupec	Typ
store_id	Long Integer
store_type	Text
region_id	Long Integer
store_name	Text
store_number	Double
store_street_address	Text
store_city	Text
store_state	Text
store_postal_code	Text
store_country	Text
store_manager	Text
store_phone	Text
store_fax	Text
first_opened_date	Date/Time
last_remodel_date	Date/Time
lease_sqft	Double
store_sqft	Double
grocery_sqft	Double
frozen_sqft	Double
meat_sqft	Double
coffee_bar	Yes/No
video_store	Yes/No
salad_bar	Yes/No
prepared_food	Yes/No
florist	Yes/No

Tabulka implicitně obsahuje 25 záznamů.

Příklad uložených údajů:

store : table

store_id	store_type	region_id	store_nam	store_number	store_street_address	store_city	store_state	store_postal_	store_coun
0	HeadQuarters	0	HQ	0	1 Alameda Way	Alameda	CA	55555	USA
1	Supermarket	26	Store 1	1	2853 Bailey Rd	Acapulco	Guerrero	55555	Mexico
2	Small Grocery	76	Store 2	2	5203 Catanzaro Way	Bellingham	WA	55555	USA
3	Supermarket	76	Store 3	3	1501 Ramsey Circle	Bremerton	WA	55555	USA
4	Gourmet Supermarket	27	Store 4	4	433 St George Dr	Camacho	Zacatecas	55555	Mexico
5	Small Grocery	4	Store 5	5	1250 Coggins Drive	Guadalajara	Jalisco	55555	Mexico
6	Gourmet Supermarket	47	Store 6	6	5495 Mitchell Canyon	Beverly Hills	CA	55555	USA

Record: 1 of 25

Obr. P.8 Příklad údajů uložených v tabulce Store

Tabulka time_by_day

Struktura tabulky.

Sloupec	Typ
time_id	Long Integer
the_date	Date/Time
the_day	Text
the_month	Text
the_year	Integer
day_of_month	Integer
week_of_year	Double
month_of_year	Integer
quarter	Text
fiscal_period	Text

Tabulka implicitně obsahuje 730 záznamů.

Příklad uložených údajů:

```
time_id  the_date    the_day    the_month  the_year day_of week_of month_of quarter fiscal_period
-------- ---------- ---------- ---------- -------- ------ ------- -------- ------- -------------
738      1998-01-07 Wednesday  January    1998     7      4.0     1        Q1      NULL
739      1998-01-08 Thursday   January    1998     8      4.0     1        Q1      NULL
740      1998-01-09 Friday     January    1998     9      4.0     1        Q1      NULL
741      1998-01-10 Saturday   January    1998     10     4.0     1        Q1      NULL
742      1998-01-11 Sunday     January    1998     11     5.0     1        Q1      NULL
743      1998-01-12 Monday     January    1998     12     5.0     1        Q1      NULL
744      1998-01-13 Tuesday    January    1998     13     5.0     1        Q1      NULL
...
```

Tabulka warehouse

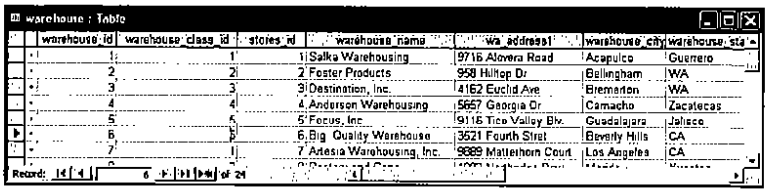
Struktura tabulky.

Sloupec	Typ
warehouse_id	Long Integer
warehouse_class_id	Long Integer
stores_id	Long Integer
warehouse_name	Text
wa_address1	Text
wa_address2	Text
wa_address3	Text
wa_address4	Text
warehouse_city	Text

Sloupec	Typ
warehouse_state_province	Text
warehouse_postal_code	Text
warehouse_country	Text
warehouse_owner_name	Text
warehouse_phone	Text
warehouse_fax	Text

Tabulka implicitně obsahuje 24 záznamů.

Příklad uložených údajů:



warehouse_id	warehouse_class_id	stores_id	warehouse_name	wa_address1	warehouse_city	warehouse_sta
1	1	1	Salka Warehousing	9716 Alovera Road	Acapulco	Guerrero
2	2	2	Foster Products	958 Hilltop Dr	Bellingham	WA
3	3	3	Destination, Inc.	4162 Euclid Ave	Bremerton	WA
4	4	4	Anderson Warehousing	5657 Georgia Dr	Camacho	Zacatecas
5	5	5	Focus, Inc.	9116 Tico Valley Blv.	Guadalajara	Jalisco
6	6	6	Big Quality Warehouse	3521 Fourth Strst	Beverly Hills	CA
7	1	7	Artesia Warehousing, Inc.	9889 Matterhorn Court	Los Angeles	CA

Obr. P.9 Příklad údajů uložených v tabulce Warehouse

Tabulka warehouse_class

Struktura tabulky.

Sloupec	Typ
warehouse_class_id	Long Integer
description	Text

Tabulka implicitně obsahuje 6 záznamů.

Příklad uložených údajů:

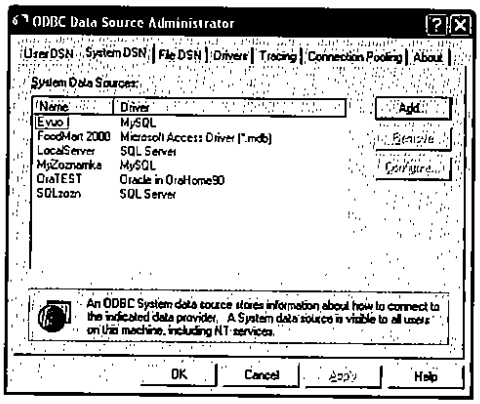
```
warehouse_class_id description
------------------ --------------
1                  Small Independent
2                  Medium Independent
3                  Large Independent
4                  Small Owned
5                  Medium Owned
6                  Large Owned
```

Příloha 2

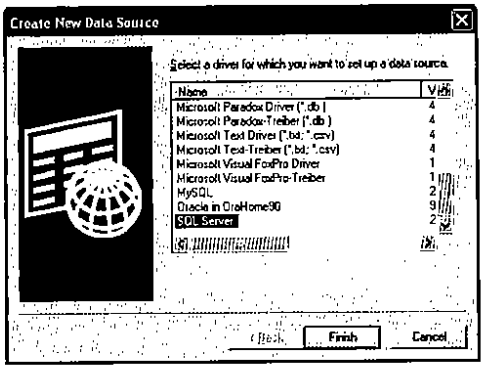
Připojení se ke zdroji údajů přes ODBC

V kapitole věnované analýze OLAP probíráme různé možnosti připojení se ke zdroji údajů. Jedním z nejčastěji používaných rozhraní je **ODBC** (Open Database Connectivity), což je otevřené databázové rozhraní, které definuje způsob přístupu aplikací systému Windows k údajům v databázích. Ukážeme si příklad konfigurace připojení pro databázi pod správou SQL Serveru. Pro mnohé je to samozřejmost, proto jsme to zahrnuli jako přílohu ke kapitole, ale každý jednou začínal...

Pomocí nástroje **ODBC Data Source Administrator**. (*Nabídka Windows Control Panels, ve Windows 2000 a Windows XP je ve složce Administrative Tools.*), tlačítka Add..., pomocí kterého vytvoříme nové připojení typu **System DSN** s názvem například **dbSupermarket**. V dialogu **Create new datasource** zvolíme možnost **SQL Server**.

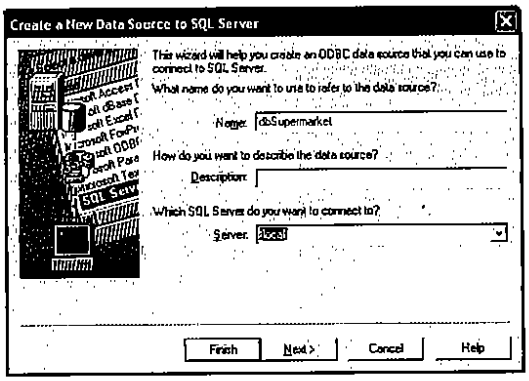


Obr. P.10 ODBC Data Source Administrator



Obr. P.11 ODBC Create New Datasource

V dalším dialogu nastavíme potřebné parametry pro připojení k naší databázi **dbSupermarket**.

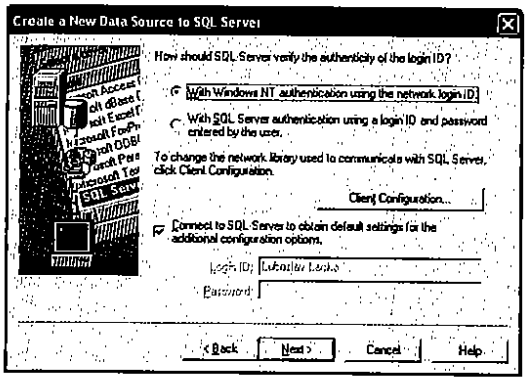


Obr. P.12 ODBC Create a New Datasource to SQL Server

Name: ***dbSupermarket***

Server: Název serveru. Pokud se jedná o lokální server, například na notebooku, nastavíme ***local****.*

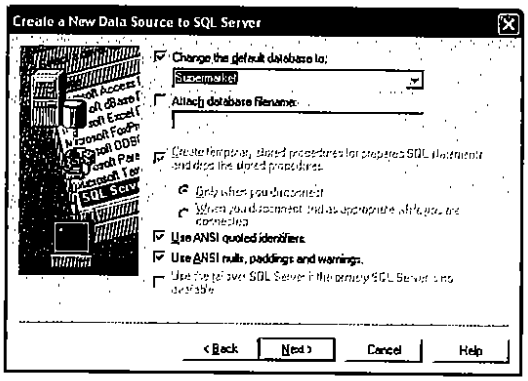
Následuje nastavení parametrů pro připojení k vybranému SQL Serveru a ověřování klienta.



Obr. P.13 ODBC Create a New Datasource to SQL Server II

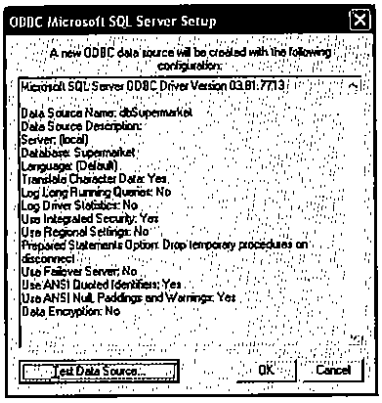
Ověřování: With Windows NT authentication...

V dalším dialogu nastavíme název databáze, ke které se chceme připojit, v našem případě to bude Supermarket.



Obr. P.14 ODBC Create a New Datasource to SQL Server III

Můžeme nastavit jazyk, ve kterém nám bude server oznamovat chybová hlášení, a nakonec otestujeme připojení.



Obr. P.15 ODBCMicrosoft SQL Server Setup

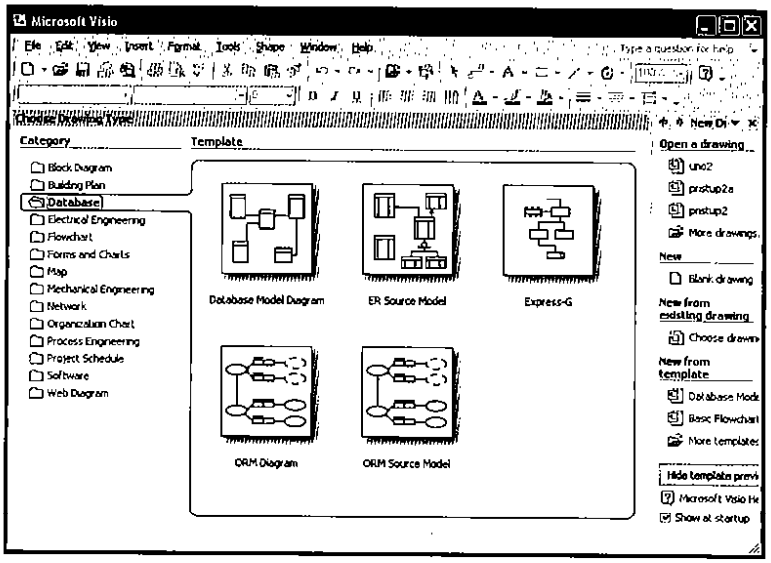
Optimistické oznámení svědčí o úspěšném připojení.

```
Microsoft SQL Server ODBC Driver Version 03.81.7713
Running connectivity tests...
Attempting connection
Connection established
Verifying option settings
Disconnecting from server

TESTS COMPLETED SUCCESSFULLY!
```

Příloha 3

Vytvoření relačního diagramu pomocí programu Microsoft Visio 2002

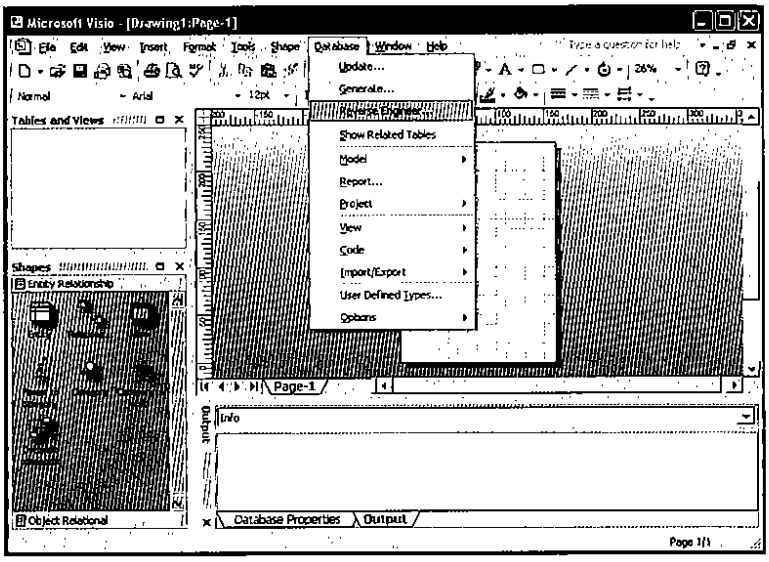
Při návrhu datových skladů a hlavně v etapě ETL je často potřeba analyzovat strukturu existující databáze. Pokud dokumentace ohledně návrhových struktur jednotlivých objektů v databázi není k dispozici nebo není úplná, můžeme si pomoci použitím vhodné aplikace pro správu příslušného databázového serveru nebo použít program Microsoft Visio 2002. Po jeho spuštění si vybereme z nabízených možností typ diagramu, který budeme vytvářet, tedy v našem případě Database Model Diagram. Není třeba se bát, předchozí věta byla trochu zavádějící, sami nebudeme vytvářet téměř nic, vše za nás provede Visio 2002.



Obr. P.16 Visio 2002 – úvodní dialog, výběr typu diagramu

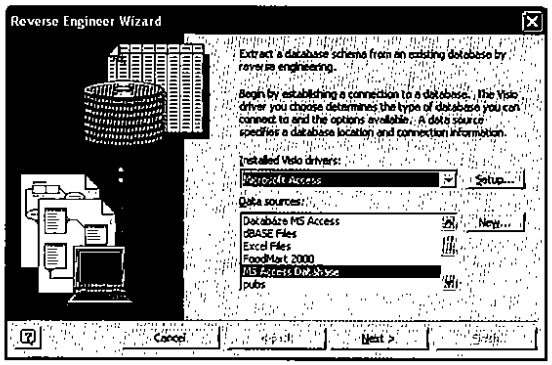
V úvodním dialogu aplikace Visio 2002 zvolíme typ diagramu. V levém okně „Category" vybereme složku „Database". V pravém okně „Template" vybereme typ „Database Model Diagram".

Jak jsme naznačili v předchozím odstavci, v následující fázi se nepustíme do „manuální" tvorby diagramu, proto si symbolů pro tabulky, relace a pohledy v pravé části programu momentálně nebudeme všímat, ale použijeme položku nabídky: ***Database - Reverse Engineer***.



Obr. P.17 Visio 2002 – výběr funkce Reverse Engineer

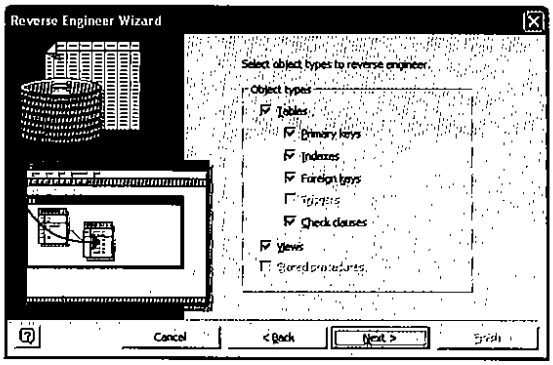
Zobrazí se úvodní dialog průvodce Reverse Engineer Wizard. Pomocí něj vybereme zdroj údajů, který je zaregistrován přes rozhraní ODBC nebo databázi programu MS Access.



Obr. P.18 Visio 2002 – úvodní dialog Reverse Engineer Wizard

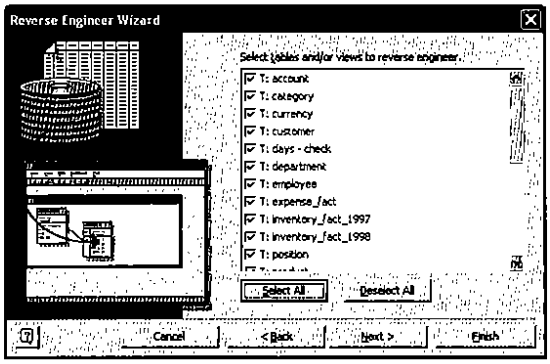
V našem případě se připojíme k souborové databázi *FoodMart 2000.mdb* v adresáři *C:\Program Files\Microsoft Analysis Services\Samples*. Protože se jedná o cvičnou databázi, která není chráněna heslem, můžeme údaje pro přístupové jméno a heslo v následujícím dialogu nechat nevyplněné.

V následujícím dialogu vybereme druhy objektů v databázi, které budou předmětem reverzní analýzy. Pod pojmem objekty rozumíme databázové tabulky, pohledy a uložené procedury. K databázovým tabulkám bývají přidružené další objekty, například primární a cizí klíče, indexy, spouště a omezení.



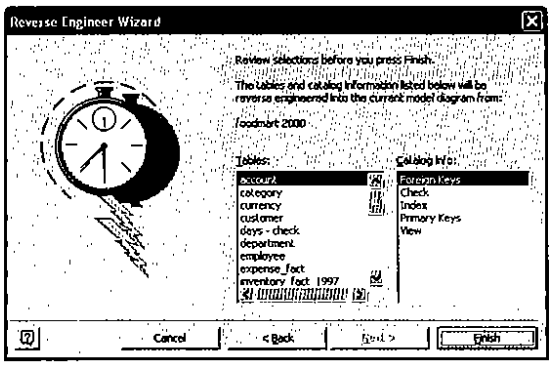
Obr. P.19 Visio 2002 – výběr typů objektů pro reverzní analýzu

Po označení druhů objektů, které budou předmětem našeho konkrétního zájmu, pokračujeme výběrem konkrétních objektů (v našem případě tabulek).



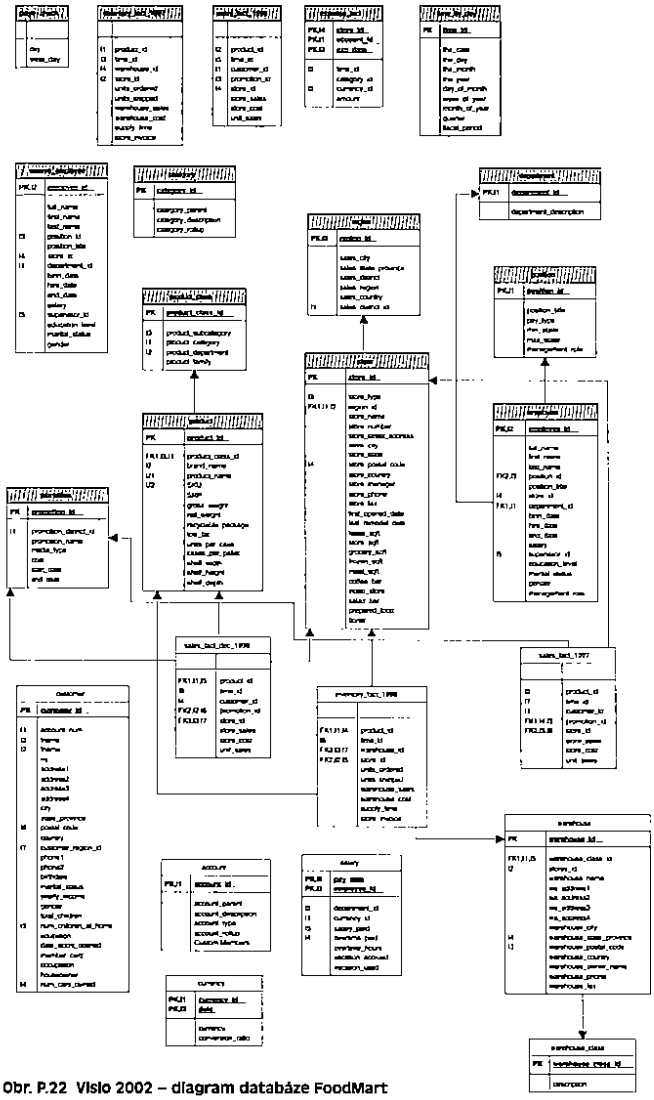
Obr. P.20 Visio 2002 – výběr konkrétních objektů pro reverzní analýzu

Po stisku tlačítka **Next** postoupíme do finále. V posledním obrázku průvodce Reverse Engineer Wizard se zobrazí seznam vybraných objektů.



Obr. P.21 Visio 2002 – tvorba diagramu

Po stisku tlačítka Finish se automaticky vygeneruje databázový diagram. Ve starší verzi Visio 2000 jsme kostru diagramu vytvořili ručně klepnutím na čtvercový symbol vedle názvu tabulky a jeho přesunutím na kreslicí plochu. Tento postup opakujeme pro všechny tabulky, které jsou předmětem našeho zájmu. Nakonec uspořádáme tabulky na ploše podle



Obr. P.22 Visio 2002 – diagram databáze FoodMart

našich představ, nadefinujeme případné další vazby a podobně. Po prvotním zobrazení grafu pro účely vytvoření analytické databáze se můžeme začít zamýšlet nad návrhem schématu dimenzí, výběrem tabulek faktů a podobně.

Produkt Microsoft Visio 2002 poskytne podstatně více funkcí, v našem případě jsme vytyčenou úlohu zobrazit strukturu tabulek splnili, takže případný další zájem databázového vývojáře o tento produkt uspokojí studium produktové dokumentace.

Příloha CD

Instalace 120denní verze Microsoft 2QL Serveru 2000

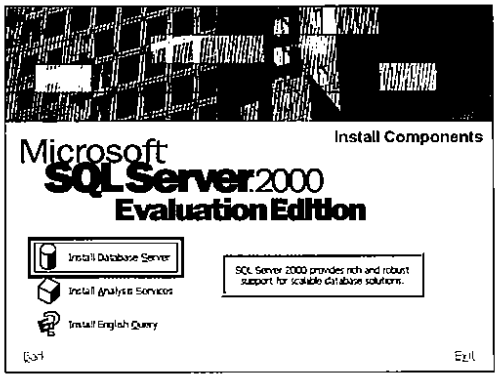
Instalace na operační systém Windows 2000 nebo Windows XP Professional je poměrně bezproblémová, stačí postupovat podle pokynů instalačního programu a ve většině případů akceptovat implicitní nastavení.

Požadavky na hardwarové vybavení počítače

Procesor: Kompatibilní s procesory Intel, 166 MHz a vyšší.

Paměť RAM: 128 MB a více.

Operační systém: Microsoft Windows NT Workstation 4.0 nebo Windows NT Server 4.0 nebo Windows NT Server 4.0 Enterprise Edition, všechny s nainstalovaným rozšířením (service pack) SP5 nebo vyšším, nebo Microsoft Windows 2000 Professional, Server, Windows 2000 Advanced Server, případně Windows 2000 Datacenter Server (verze Trial totiž na rozdíl od komerční verze Enterprise funguje také na počítačích se systémy Windows NT Workstation a Windows 2000 Professional).



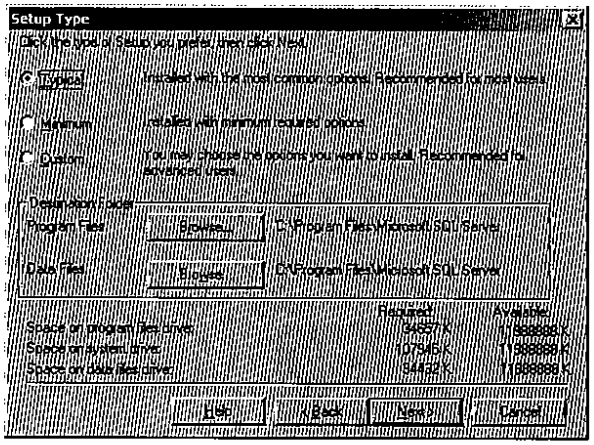
Obr. P.23 Instalace Serveru SQL

Pokud nemáme k dispozici firemní server, zvolíme instalaci na lokální počítač, na kterém budeme pracovat s databází. Vybereme nejjednodušší způsob instalace: ***Create a new instance of SQL Server, or install Client Tools*** *(Vytvořit novou instanci Serveru SQL nebo nainsta-*

lovat klientské nástroje.). Po odsouhlasení licenčního ujednání následuje další dialog, ve kterém zvolíme předdefinovanou možnost ***Server And Clients Tools*** *(Server a klientské nástroje).* Pokud nejsme zkušenými správci, vybereme si v dalším dialogu typickou instalaci. Zároveň se dozvíme přibližné nároky Serveru SQL na diskovou kapacitu, které jsou vzhledem k dostupné kapacitě dnešních komerčních disků zanedbatelné. V našem případě se jednalo o položky:

Program files drive	34 MB
System drive	107 MB
Data files drive	34 MB

Takže dohromady o něco méně než 200 MB.

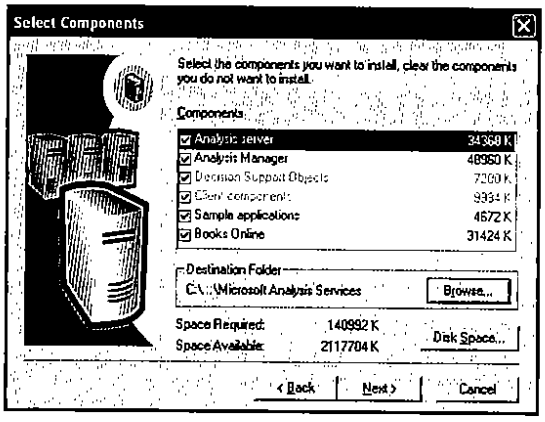


Obr. P.24 Typy instalací Serveru v

V okně Services Accounts zvolíme možnost ***Domain User Account***, zadáme přihlašovací heslo do systému Windows 2000. Ověřování nastavíme nejlépe na Windows NT mode, v případě speciálních požadavků na ***Mixed mode***, kdy se použije ověřování operačního systému Windows NT nebo Windows 2000 zároveň s přihlašovacím heslem Serveru SQL. Zadáme heslo, které budeme používat. Tento údaj je zároveň poslední, který musíme při instalaci MS SQL Serveru 2000 zadat.

Instalace analytických služeb

Po úspěšné instalaci databázového serveru SQL Server 2000 je pro účely této publikace potřeba nainstalovat i analytické služby. Znovu spustíme instalaci serveru, ale tentokrát vybereme možnost „Install Analysis Services". Instalátor provede vše za nás. V dialogu Select Components případně můžeme změnit parametry, ale implicitně se nainstaluje vše potřebné.



Obr. P.25 Instalace komponentů analytických služeb

Instalace servisního balíčku Service Pack 3

Jak vyplývá z čísla typového označení SQL Serveru 2000, jedná se o produkt ověřený téměř tříletým provozem. Za tuto dobu připravila firma Microsoft mnoho různých záplat a vylepšení, proto je potřeba nainstalovat z instalačního CD i Service Pack 3.

Seznam použité a doporučené literatury

Literatura

[1] Chon S. Chua, Green, R.: Data Warehouse Method – studijní literatura firmy ORACLE, Oracle University 1999

[2] Matiaško, K.; Vnuk, L.; Ševčíková, K.: Dátové sklady ako informačný zdroj pre podporu rozhodovania, Žilinská univerzita

[3] Rud, Olivia Parr: Data mining, Computer Press 2001

[4] Seidman, C.: Data mining with SQL Server 2000, Microsoft Press 2001

[5] Vieira, R.: SQL Server 2000, Programujeme profesionálne, Computer Press 2001

[6] Humphries, M.: Data warehousing, Návrh a implementace, Computer Press 2001

[7] Bain, T.: Professional SQL Server 2000 Data Warehousing with analysis service, Wrox 2001

[8] Lacko, L.: Oracle Správa, programování a použití databázového systému, Computer Press Brno 2002

[9] Lacko, L.: Web a databáze, Computer Press Brno 2001

[10] Lacko, L.: Analytické možnosti SQL Servera, sborník Microsoft Praha 2002

[11] Lacko, L.: Vyššia škola databázová – data mining, PC SPACE – 8/2002, strany 6–10

[12] Lacko, L.: Budovanie dátových skladov, PC REVUE – seriál 10, 11, 12/2000, 1/2001

[13] Lacko, L.: Databázy pre každého, PC SPACE – 8/2001

Cvičná databáze pro Data Mining byla získána ze zdrojů:

Blake, C. L. & Merz, C. J. (1998). UCI Repository of machine learning databases http://www.ics.uci.edu/~mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science.

Department of Information and Computer Science
University of California, Irvine CA 92697-3425

Materiály z konference Microsoft TechEd 2002 Barcelona

Webové stránky společností

www.microsoft.com
www.oracle.com
www.peyo-home.sk

Rejstřík

N

O

P

Q

R

S

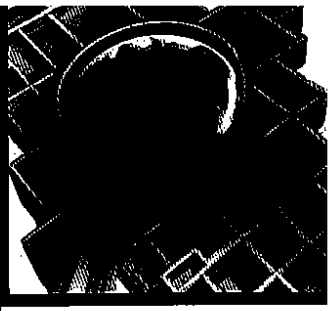
T

U

V

W

X



Databáze

Datové sklady analýza OLAP a dolování dat

s příklady v SQL Serveru a Oracle

Intenzivní nasazení výpočetní techniky do různých odvětví lidské činnosti s sebou přináší shromažďování velkého množství různorodých údajů: z technologických zařízení, prodeje zboží, firemní administrativy apod. Moderní databázové servery umožňují s takovým množstvím dat nejen bezpečně a rychle pracovat, ale i získávat z těchto dat skutečné informace a z nich pak vyvozovat poznatky, které již lze efektivně využívat v procesu rozhodování např. ve výrobě, obchodu a marketingu či výzkumu. Moderní databázové servery obsahují rozsáhlou podporu pro budování datových skladů (data warehousing), analýzu OLAP (On-line Analytical Processing) a dolování dat (data mining).

Předkládaná publikace, jedinečná svou srozumitelností a uceleností, poskytuje solidní vstup do těchto pokročilejších databázových technologií všem, kdo již dnes s databázemi pracují, ale nedokáží plně využívat bohatství informací, jež jim dosud evidovaná data skrývají. Jedná se především o marketingové pracovníky, manažery a zejména datové analytiky; své kapitoly v knize najdou také tvůrci a správce databází a programátoři kancelářských i webových aplikací. S knihou mj. zvládnete:

- Základní pojmy a principy, nezbytné statistické minimum
- Metody vytváření datového skladu, jeho naplňování a provoz
- Nástroje pro administraci a práci s analytickými službami
- OLAP-analýzu dat uložených v datovém skladu
- Přístup k datům v databázích OLAP z klientských aplikací
- Ukázky business intelligence v programech Microsoft Office
- Jazyk MDX pro přístup k multidimenzionálním datům
- Snadný vývoj aplikací pomocí Pivot Table Services, ADOMD, Office Web Components, stránek HTML/ASP, Microsoft XML for Analysis SDK nebo Visual Basicu .NET
- Data mining: teoretický základ, přípravu dat, výběr algoritmu a modelu, fázi učení, analýzu a predikci budoucích případů

Teoretické objasnění principů je postupně prokládáno podrobně popsanými postupy a bohatě ilustrovanými příklady – od jednoduché analýzy rodinného rozpočtu či databáze hub přes záznamy telefonní ústředny, stanovení úvěrového rizika či analýzy nákupního chování zákazníků až po ukázku řešení dvouterabajtového datového skladu obchodního řetězce.

Platformou pro většinu příkladů je SQL Server 2000, samostatná kapitola je věnována Oracle.

Přiložené CD obsahuje Microsoft SQL Server 2000 (120denní verzi doplněnou o Service Pack 3), nástroje Easy Olap firmy Peyo, příklady a cvičné databáze ke knize.

O autorovi:

Lubomír Lacko (*1965) je odborníkem na databázové technologie a programování s více než desetiletou praxí ve vývoji databázových aplikací. Pracuje ve Vojenském technickém ústavu v Liptovském Mikuláši a působí jako školitel a konzultant. Ve slovenských časopisech PC Revue a PC Space publikuje články a seriály především o programování, databázích a data miningu. Je autorem několika odborných příruček pro společnost Microsoft a knih nakladatelství Computer Press *Web a databáze, Oracle – správa, programování a použití databázového systému a SQL – hotové řešení.*

Zařazení publikace:

Computer Press,a.s.
nám.28.dubna 48, 635 00 Brno

Objednávejte na:
www.knihy.cpress.cz
distribuce@cpress.cz

Bezplatná telefonní linka:
800 555 513

ISBN 80-7226-969-0
Prodejní kód: K0788

Doporučená cena:
469 Kč
699 Sk
CD v ceně