Tato skripta jsou určena studentům oboru Aplikovaná informatika a všem odborníkům z informatické praxe, kdož potřebují získat základní znalosti z problematiky databází.

Skripta jsou rozdělena do tří částí:

První část teoreticky pojednává o různých *databázových modelech*: o souborovém přístupu, o síťových, hierarchických, relačních, objektově-relačních, objektových a multi-dimenzionálních databázích.

Druhá část se intenzívně a zevrubně zabývá problematikou *navrhování databází*.

Třetí část podává *přehled jazyka SQL*, text prvních dvou částí se občas odkazuje do tohoto přehledu.

Výklad je doprovázen množstvím názorných příkladů.

OBSAH

ÚVOD **5**
DATA A INFORMAČNÍ SYSTÉMY 6
1. ČÁST - STRUKTURY DAT **8**
LOGICKÁ A FYZICKÁ STRUKTURA DAT 9
LOGICKÉ STRUKTURY DAT 10
Síťový databázový model *12*
Hierarchický databázový model *13*
Ostatní databázové modely *14*
RELAČNÍ DATABÁZE 15
OBJEKTOVĚ-RELAČNÍ DATABÁZE 19
OBJEKTOVÉ DATABÁZE 21
DATOVÉ SKLADY 23
Analýza a návrh pro datové sklady *25*
Architektura datového skladu *26*
2. ČÁST - NAVRHOVÁNÍ DATOVÉ ZÁKLADNY **27**
KONCEPTUÁLNÍ MODEL DATOVÉ ZÁKLADNY 28
ER modelování *29*
Entity 30
Vztah 30
Role 33
Atribut 33
Klíč 34
Identifikační závislost 35
ISA hierarchie 35
Další integritní omezení 36
Příklady ER modelů *36*
Northwind 37
Knihovna 38
Diskusní klub 39
IS soukromé VŠ 40
Boxeři 41
KONCEPTUÁLNÍ SCHÉMA → RELAČNÍ SCHÉMA 43
Rozmyslete *46*
NORMALIZACE DATABÁZOVÉHO SCHÉMATU 47
Normální formy relačního schématu *48*
1.normální forma 48
2.normální forma 48
3.normální forma 49
Boyce-Coddova normální forma 49
4.normální forma 50
Další normální formy… 50
„Postupem shora dolů vytvoříme normalizované tabulky.“ *50*
Dvoutvářné entity 51
Návrh zdola nahoru *51*
Jenže 51

Doporučený postup *51*
Denormalizace *52*
Příklady *52*
Northwind 52
IS soukromé VŠ 52
PŘÍKLADY RELAČNÍCH DATABÁZOVÝCH SCHÉMAT 54
Knihovna *54*
Diskusní klub *55*
IS soukromé VŠ *55*
Boxeři *56*
FYZICKÉ SCHÉMA – INDEXY 57
Indexy *57*
B-stromy 57
Hašované indexy 58
Další možnosti organizace souborů… 58
Vyhledávání podle pole, jež není indexováno *58*
Další ovlivnění fyzických charakteristik *58*
Co indexovat? *58*
3. ČÁST - JAZYK SQL 59
DOTAZOVACÍ JAZYK SQL, JEDNODUCHÉ DOTAZY 60
Dotazy s vnějším spojením (SQL99) *66*
DOTAZOVACÍ JAZYK SQL, SESKUPOVACÍ DOTAZY 67
Výběr skupin v seskupovacím dotazu *72*
DOTAZOVACÍ JAZYK SQL, PODDOTAZY 73
Vektorový subselect *73*
Skalární subselect *73*
Subselecty jako tabulkové výrazy *74*
SQL SELECT PŘEHLED SYNTAXE 75
Poznámka o standardu SQL *76*
OPTIMALIZACE DOTAZŮ 77
PŘEHLED OSTATNÍHO JAZYKA SQL 79
Jazyk definice dat *79*
Základní datové typy 79
Definice domény 79
Definice tabulky 79
Změna definice tabulky 79
Zrušení tabulky 79
Definice indexu 79
Zrušení indexu 79
Vytvoření pohledu 80
Zrušení pohledu 80
Odvozené tabulky 80
Temporární tabulky 80
Konstrukty pro vyšší úrovně organizace dat 80
Integritní omezení *80*
Na 1 pole 80
Na řádek tabulky 80
Mezi tabulkami 81
Definice tabulek pro příklad Northwind *81*

Jazyk manipulace dat ... *82*
Přidávání záznamů ... 82
Aktualizace záznamů ... 82
Mazání záznamů ... 82
Triggery ... *82*
Uložené procedury ... *83*
Tvorba aplikací přistupujících k databázi ... *83*
Jak zadat databázové příkazy v aplikaci? ... 84
Kurzory ... *84*
Dynamické SQL ... 85
Transakce a zámky ... *87*
Vlastnosti ACID ... 88
Ochrana dat ... *89*
Práva a pohledy ... 89
LITERATURA ... 90

ÚVOD

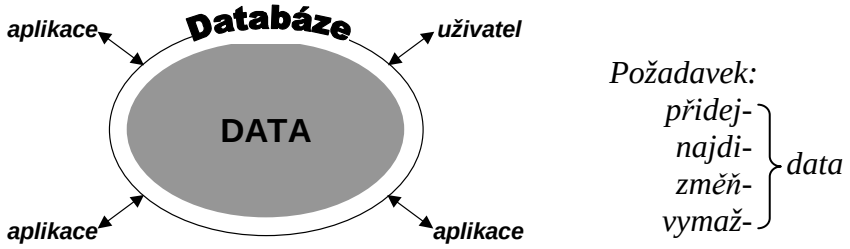
DATA A INFORMAČNÍ SYSTÉMY

Informační systémy mají základ v datech. Organizační nebo pracovní procesy, resp. lidé nebo automatická zařízení v nich, připojené k informačního systému ukládají data, a to ve vhodně předem navržených a vytvořených strukturách. Z uložených dat jsou na základě požadavků, vzniklých popřípadě v jiných částech organizace, některá data vybírána, a prezentována lidem či předávána dalším automatickým systémům. Tedy funkce informačního systému napojené na organizační procesy ukládají data proto, aby byla dostupná *uživatelům k naplnění jejich informačních potřeb* nebo *automatickým funkcím k řízení organizačních procesů*.

Vysvětleme vztah mezi daty a informacemi. Data jsou exaktní údaje o zjištěných hodnotách veličin, které vyjadřují *atributy (vlastnosti) reálných objektů nebo dějů.* Informace jsou smysluplné *interpretace dat* a vztahů mezi nimi. Informace
- odstraňují nejistotu
- ovlivňují rozhodování
- závisí na znalostech, zkušenostech, světovém názoru příjemce
- mají užitnou hodnotu a tomu odpovídající cenu, mohou být zbožím.

Informace jsou strategickým zdrojem pro fungování všech systémů. *Informační systémy* jsou podpůrné systémy v organizacích, jejichž účelem je *informovat předáváním dat.*

Nejběžnější architektury informačních systémů jsou založeny na tzv. databázovém přístupu. Ten spočívá v tom, že data jsou spravována *databázovou vrstvou oddělující aplikační programy či uživatele od fyzického ukládání dat* (fyzické ukládání dat jinak běžně vykonávají operační systémy počítačů na základě příkazů zadávaných programy). Požadavky na data vycházející od aplikací nebo od uživatelů jsou vyřizovány touto databázovou vrstvou:



Základními požadavky na data, jež aplikace nebo uživatelé zadávají, jsou: přidat nová data, nalézt určitá data podle požadavku, změnit či vymazat určitá data.

Z inženýrského hlediska bývá architektura informačního systému rozdělena do tří vrstev: prezentační, aplikační a databázové. Prezentační a aplikační vrstvu vytvářejí aplikační programátoři, eventuelně i odděleně od sebe; uživatelé přicházejí do styku pouze s prezentační vrstvou. Databázová vrstva je běžně provozována programy nazývanými systémy řízení báze dat (SŘBD, anglicky DBMS – database management system), jež bývají spravovány administrátory databáze. Strukturu dat v databázi navrhují designéři na základě modelů navržených analytiky.

V závislosti na rozsáhlosti informačního systému mohou oddělené vrstvy (prezentační, aplikační, databázová) probíhat na různých výpočetních jednotkách (počítačích), ale také může být provádění dvou vrstev, či dokonce všech tří vrstev, realizováno na jediném počítači. Stejně tak některé role tvůrců informačního systému mohou eventuelně splývat v jedné osobě.

Co dělá systém řízení báze dat? Spravuje datovou základnu – tzv. databázi, jež typicky obsahuje **velké množství persistentních** (tj. trvale dostupných)**, spolehlivých a sdílených dat**. Spolehlivost

znamená, že data v databázi uložená odpovídají očekávání uživatelů o nich, tj. že jsou „správně“. Spolehlivost je zajišťována jak organizačními opatřeními lidí tak ochranou a kontrolou dat prováděnou systémem řízení báze dat. – Sdílení dat znamená, že tatáž data využívají různí lidé a programy.

SŘBD zajišťují:

- *sdílení dat, paralelní přístup*
  Aplikace i uživatelé mohou k datům v databázi přistupovat současně a vzájemně data sdílet.
- *nezávislost aplikací na fyzickém uložení dat*; centrální správu dat
  Fyzické uložení dat řídí a spravuje SŘBD, aplikace jsou od fyzického ukládání dat odděleny – o fyzickém ukládání více pojednává 2. ČÁST, kapitola FYZICKÉ SCHÉMA – INDEXY.
- *jazyk přístupu k datům*, centrální popis dat
  Své požadavky na data sdělují aplikace a uživatelé ve formálním jazyce SŘBD, tento jazyk umožňuje pojmenovávat části datových struktur. Jména mohou odpovídat jménům reálných objektů, navíc SŘBD umožňuje doplnit tato jména o popisy, jež vysvětlují jejich význam (sémantiku). *Je žádoucí, aby struktury dat pokud možno odrážely strukturu reality.*
  – o jazycích přístupu k datům pojednává 3. ČÁST.
- *ochranu* před neoprávněným přístupem a poruchami
  SŘBD selektivně umožňuje či brání přístupu k datům podle určených přístupových práv; SŘBD zajišťuje persistenci dat i v případech poruch hardware, operačního systému, komunikačních kanálů (pomocí zálohování, žurnálů, rozvrhů akcí).
- kontrolu konzistence dat
  SŘBD dodržuje *strukturu dat* definovanou správcem databáze.
  SŘBD umožňuje definovat tzv. *integritní omezení*, tj. pravidla, jež SŘBD kontroluje při manipulacích s daty (pravidlo může být např. „Nelze uložit záznam o objednávce provedené zákazníkem, o němž neexistuje v databázi záznam.“).
  – o integritních omezeních více pojednává 3.ČÁST, kapitola PŘEHLED OSTATNÍHO JAZYKA SQL, Integritní omezení.
  SŘBD umožňuje definovat logické celky manipulací s daty, tzv. *transakce*, jež mají být provedeny celé, nebo v případě poruchy zabraňující dokončení transakce má SŘBD vrátit stav databáze před počátek provedení první z manipulací transakce
  – o transakcích více pojednává 3.ČÁST, kapitola PŘEHLED OSTATNÍHO JAZYKA SQL, Transakce a zámky.
- velké objemy dat
  Všechny požadavky má SŘBD zvládat s tak velkým objemem dat, jaký potřebujeme.

Výše zmíněné vlastnosti SŘBD jsou maximalistické a do jisté míry ideální, konkrétní SŘBD dodávané různými dodavateli je naplňují v různé míře. Silné SŘBD usilují o naplnění těchto požadavků v míře co nejvyšší.

Pro konkrétní informační systém však lze použít i slabší produkty, např. tzv. klientské databáze, či dokonce plniče souborů, zajišťující minimum z vyjmenovaných požadavků (tj. oddělení aplikací od dat a abstraktní jazyk přístupu k datům).

V některých aplikačních prostředích dokonce nemusí být data ani přímo ukládána na perzistentní paměťová média, ale mohou být neustále dostupná v operační paměti řízené správou dat. Objemy dat takto spravovaných nemohou být velké. Takovéto databáze bývají občas archivovány na discích, ochrana před poruchami je tak zajištěna pouze pro již archivovaná data.

1. ČÁST -
STRUKTURY DAT

LOGICKÁ A FYZICKÁ STRUKTURA DAT

Data v databázích jsou typicky organizována do záznamů. Záznam je tvořen několika údaji, je logicky strukturován do tzv. **atributů**. Například

KřestníJméno : Jan
Příjmení : Novák
Ulice : Sluneční
ČísloDomu : 19
Obec : Prašivice
PSČ : 543 21
DatumNarození : 12.7.1958

popisuje strukturu záznamu, ve které jsou jednotlivé údaje hodnotami atributů pojmenovaných pořadě: „KřestníJméno“, „Příjmení“, „Ulice“, „ČísloDomu“, „Obec“, „PSČ“, „DatumNarození“. **Typ tohoto záznamu** je 7mice: (KřestníJméno, Příjmení, Ulice, ČísloDomu, Obec, PSČ, DatumNarození). Databáze typicky obsahuje velké množství záznamů určitého typu, databáze také většinou zahrnuje více různých typů záznamů.

Atributy mají jméno (např. „KřestníJméno“, „ČísloDomu“) a předem určený obor hodnot – tzv. **doménu**. Doména bývá v praxi často specifikována jako určitý datový typ (např. „text délky 15 znaků“, nebo „celé číslo“).

Ve starší terminologii se záznamům také říká věty. Terminologie:

Atribut ≈ jméno s odpovídající množinou hodnot, tzv. doménou
Typ logického záznamu (typ vět) ≈ seznam atributů
Logický záznam (věta) ≈ seznam hodnot atributů

Kolekce záznamů stejného typu se v různých databázích nazývá různě, v těchto skriptech ji nazvěme „databázový soubor“ (pozor – nepleťte se soubory operačního systému počítače, databázový soubor může a nemusí odpovídat nějakému souboru operačního systému).

Databázové soubory jsou ukládány na vnější paměťová média, logickému záznamu odpovídá **fyzický záznam**. Fyzický záznam může kromě hodnot atributů obsahovat i jiné znaky: oddělovače, ukazatele k jiným záznamům, příznaky zrušení záznamu, údaj o délce záznamu…

Fyzické záznamy musí být organizovány do fyzických stránek vnější paměti – **bloků**.

Blok ≈ jednotka přenosu mezi vnější pamětí a operační pamětí

Do jednoho bloku se může vejít více fyzických záznamů, nebo pouze jediný záznam, nebo může nastat případ, že k uložení fyzického záznamu je třeba více bloků vnější paměti. Otázkami fyzické struktury dat se více zabývá 2.ČÁST, kapitola FYZICKÉ SCHÉMA – INDEXY. Tato skripta jsou však převážně zaměřena na problematiku logických struktur dat v databázích.

Kromě logických záznamů, jež slouží jako záznamy údajů o objektech reálného světa, databáze nějakým způsobem uchovává i některé vztahy mezi záznamy, tj. zaznamenává i některé vztahy mezi objekty reálného světa. Způsoby a metody zaznamenávání vztahů závisí na druhu databáze, tzv. **databázovém modelu**. O databázových modelech pojednává následující kapitola.

LOGICKÉ STRUKTURY DAT

Tato kapitola dává přehled o datových strukturách používaných v aplikačních programech, a o tom, jaké druhy struktur dat umožňují ukládat SŘBD – o takzvaných *databázových modelech*.

Datová struktura je účelné uspořádání dat v paměti počítače; datové struktury používané aplikačními programy jsou:

Základní typy dat:
Celočíselná proměnná
Reálná proměnná
Logická (tzv. booleova, boolská) proměnná
Textový řetězec

Základní konstrukce:
Pole (vektor, matice, n-rozměrný prostor)
Strukturní skupina

Skládáním základních konstrukcí lze vytvářet složité datové struktury, jimiž modelujeme složité objekty, eventuelně vztahy v reálném světě. Některé speciální často používané konstrukce mají svá jména, jejich popis následuje:

Speciální konstrukce:

Záznam (věta)
je speciální případ strukturní skupiny, vyznačující se tím, že 1)zpravidla odpovídá záznamu údajů o nějaké jedné entitě, 2)účastní se vstupních a výstupních operací jako celek. Často je přidána konstrukce klíče: skupiny atributů, podle níž je záznam vyhledáván. Záznam je základní logickou strukturou většiny databázových programů.

Soubor
je množina výskytů struktury „záznam“, často odpovídající nějaké množině entit stejného typu v určité problémové oblasti. Má počáteční návěští, za nímž následuje posloupnost vět, a je ukončen koncovým návěštím. Většinou se užívá pro ukládání ve vnější paměti.

Seznam
se využívá pro vytvoření soupisu eventuelně se opakujících hodnot, jejichž počet není předem znám. Jeho prvky jsou strukturní skupiny složené z hodnot a jednoho či dvou ukazatelů – ukazatele na následující prvek seznamu nebo ukazatele na předchozí prvek. Vedlejším efektem tohoto provázání je uspořádání seznamu. Seznam může být doplněn samostatným ukazatelem na počátek event. konec seznamu. (Pozn.: soupis hodnot bez uspořádání se někdy nazývá multimnožina.)

Fronta FIFO
(first in first out) je speciální případ seznamu, specifického se svým užitím: hlavními operacemi jsou připojení na konec fronty a odpojení ze začátku fronty – tj. zpracování prvku, který je na řadě.

Zásobník (LIFO)
(last in first out) je jiný speciální případ seznamu. Hlavními operacemi jsou přidání do zásobníku – připojení na začátek seznamu (nahoru na hromadu), a odebrání ze zásobníku – odpojení ze začátku seznamu (odebrání shora hromady).

Tabulka
má řádky a sloupce, lze si ji představit jako jednorozměrné pole výskytů strukturních skupin

stejného typu, představujících její řádky. Sloupce pak vzniknou abstrakcí z jednotlivých složek struktury jejích řádků, jsou tedy jednorozměrnými poli hodnot toho kterého typu. Od dvourozměrného pole se tabulka odlišuje tím, že prvky v různých sloupcích mohou být různého typu.

Strom

je dynamickou strukturou tzv. uzlů – strukturních skupin tvořených hodnotou a několika ukazateli, každý z nichž ukazuje na následníky neboli potomky uzlu. Předchůdcem uzlu je uzel, jehož je tento následníkem. Rozlišujeme kořen stromu, vnitřní uzly a listy. Ve stromu má každý uzel kromě kořene právě jediného předchůdce, listy nemají následníky. Ve stromu vede z kořene ke každému listu jediná cesta.

Speciálním případem stromu jsou vyvážené stromy, v nichž všechny cesty od kořene k listům mají stejnou délku. Využívá se hlavně v databázových aplikacích pro podpůrné struktury zvané indexy.

Síť

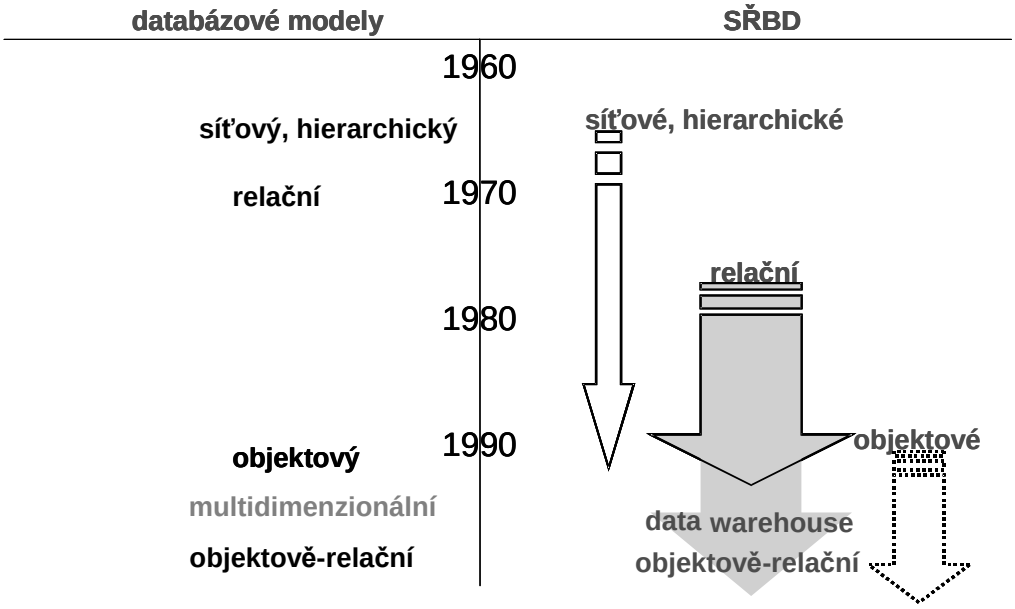
je vzácně používanou strukturou odlišující se od stromu tím, že uzly mohou mít více předchůdců a graf následnosti může tvořit cykly.

Vztah

je rozličně realizovaná struktura zaznamenávající vztah entit jednoho typu k entitám jiného typu, například „fakulta (entita prvního typu) vychovává studenta (entita druhého typu)“. Může být realizována jako tabulka o dvou sloupcích, z nichž první hodnota v každém řádku ukazuje na (identifikuje) první entitu (o níž je např. někde existující záznam) a druhá hodnota ukazuje na druhou entitu v daném vztahu; realizace může být ovšem zcela jiná.

Rozlišují se vztahy 1:1, 1:n, n:m.

Systémy řízení báze dat umožňují zaznamenávat pouze některé z logických struktur, podle těchto možností je rozdělujeme na *síťové*, *hierarchické*, *relační*, *multi-dimenzionální*, *objektově relační* a *objektové*. Množině typů logických struktur, jež databáze může zaznamenávat, se říká **databázový model**. Následující obrázek ukazuje historický přehled databázových modelů a existence SŘDB realizujících tyto modely:



Všem uvedeným modelům historicky předcházel „lineární“ model – to je tzv. *souborový přístup* k ukládání dat. Ten *je předchůdcem databázového přístupu* popsaného v úvodní kapitole. Souborový přístup spočívá v tom, že aplikace ukládají svá data do souborů „přímo“, tj. za pomoci služeb operačního systému, záznamy v jednom souboru jsou přístupovány sekvenčně. Variacemi jsou tzv. indexsekvenční a indexované organizace dat, o nich se můžete dočíst v [Pok97] nebo v [GCCC]. Souborový přístup se dodnes používá, neboť nevyžaduje použití SŘBD.

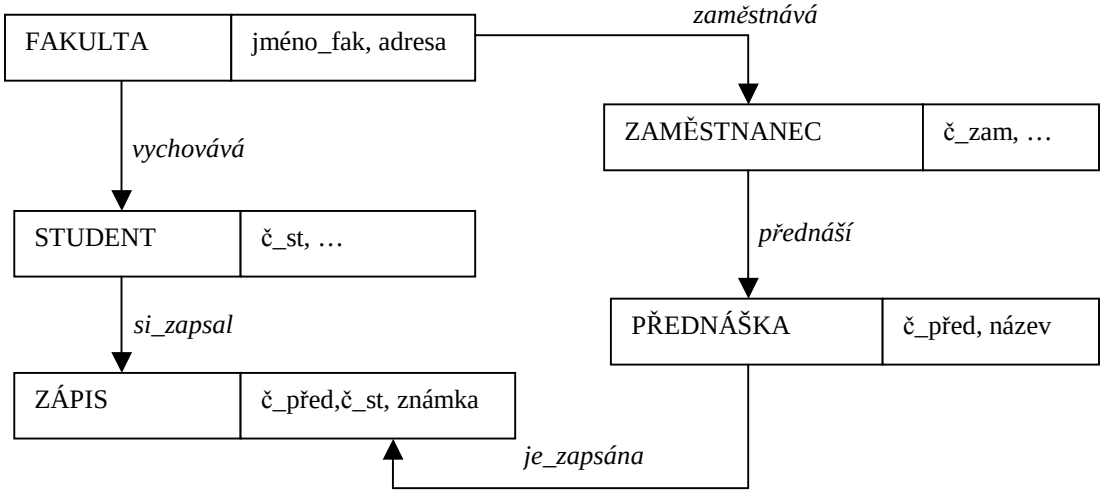
Síťový, hierarchický, relační i objektově-relační model umožňují definovat typy záznamů, jež budou ukládány, a pak je způsobem specifickým pro konkrétní SŘBD ukládat. Objektový model namísto toho umožňuje definovat typy tzv. objektů, což jsou z datového hlediska vývojoví následníci logické struktury „strukturní skupina“. Odlišností objektového modelu je to, že kromě logických datových struktur (tzv. statické stránky modelu) umožňuje zahrnout i typové chování těchto struktur (tzv. dynamickou stránku modelu).

Multi-dimenzionální databázový model je založen na zcela jiném principu, v němž základními stavebními kameny nejsou záznamy (více naleznete v kapitole DATOVÉ SKLADY).

Všechny databázové modely umožňují pro konkrétní databanku definovat tzv. *schéma databáze*, tj. abstraktní model dat v databázi, popisující a pojmenovávající použité typy logických struktur a případná integritní omezení.

Síťový databázový model

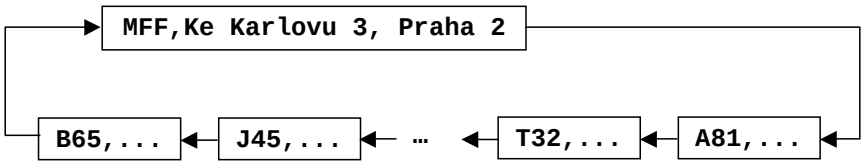
Umožňuje kromě typů záznamů definovat také typy vztahů mezi záznamy: konkrétní typ vztahu má vždy jméno, jež charakterizuje reálný význam vztahu, dále má určen typ záznamů, jež mohou být vlastníky ve vztahu, a typ záznamů, jež mohou být podřízenými členy v tomto vztahu. Modelovaný typ vztahu tedy musí být 1:n . Vztahy n:m je třeba uměle rozkládat vytvořením tzv. „průnikového“ typu záznamu. Příklad konkrétního schématu databáze znázorňuje následující obrázek:



Průnikovým typem záznamu je „Zápis“, realizujícím zaznamenání vztahu n:m mezi studenty a přednáškami.

Síťový SŘBD způsobem sobě vlastním ukládá jednotlivé výskyty typu vztahu do tzv. sady záznamů, v každé z nichž je jeden záznam vlastnický a několik záznamů členských. Typicky jsou sady záznamů realizovány jako kruhové seznamy, eventuelně obohacené o ukazatele od každého člena k vlastníku. Jestliže je záznam vlastníkem nebo členem ve vztazích různých typů, je k němu

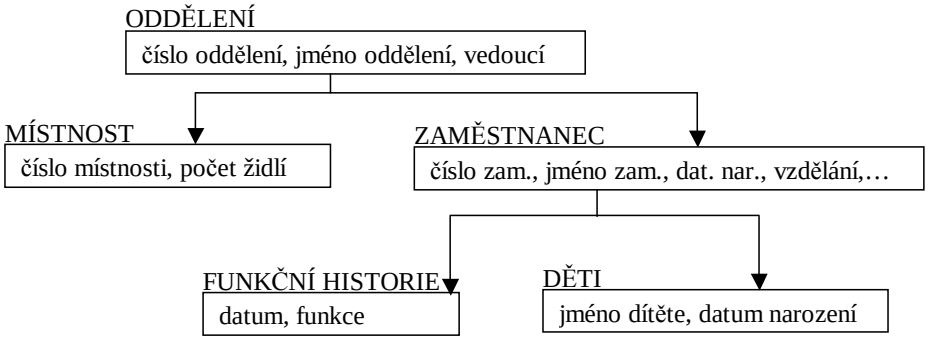
připojeno příslušně více ukazatelů realizujících odpovídající strukturu. Kruhový seznam realizující vztah „vychovává“ znázorňuje následující obrázek:



Síťový databázový model má standard skupiny CODASYL, příkladem konkrétního SŘBD byl systém IDMS.

Hierarchický databázový model

Umožňuje definovat hierarchickou strukturu záznamů, odpovídající logické struktuře stromu, v jehož uzlech jsou záznamy. Schéma databáze popisuje typ stromů, jež může databáze ukládat – viz příklad:



Konkrétní struktura dat pak může vypadat takto:

```
03  PROGRAMOVÁNÍ  Novák Jiří
      324    4
      322    10
      321    5
      1321   Albl Antonín     25.1.1953   VŠ
        1.1.1983  programátor
        Jiří 28.4.1981
        Eva  6.2.1980
        Jan  5.1.1978
      1239   Kusý Jan  31.12.1938   SŠ
        1.8.1983  programátor
      1234   Novák Jiří 27.5.1950   VŠ
        1.1.1984  vedoucí
        15.5.1983 programátor
        1.7.1982  operátor
        Eva  12.8.1982
```

Hierarchická databáze typicky obsahuje duplicity kvůli vztahům m:n :

```
1321  Albl Antonín 25.1.1953  VŠ
       AKZ-43-63 Opel Astra 1800 benzin
       ZKY-66-99 VW   Golf  1600 benzin
1234  Novák Jiří   27.5.1950  VŠ
       ZKY-66-99 VW   Golf  1600 benzin
       AKY-25-41 Ford Mondeo2400 diesel
```

zaměstnanci, a jaká řídí auta …

duplicitní údaje o vozu ZKY-66-99

Hierarchický databázový model nemá standard, příklady SŘBD jsou IMS a MUMPS.

Předností i slabinou síťových a hierarchických databází jsou přístupové cesty k datům. V prvním zde uvedeném příkladu bude požadavek „vypiš všechny přednášky a známky ze zkoušek studenta Filosofické fakulty, Jánského Vladimíra“ snadno formulován a rychle vyplněn. Ale požadavek „z kterých předmětů bylo nejvíce jedniček, bez ohledu na fakulty“ bude zpracováván velmi obtížně a dlouho. Podobně v druhém příkladu požadavek „s kterými auty jezdí zaměstnanci z různých oddělení“ bude patřit k náročnějším. – Kvůli požadavkům na vytváření pohledů na data, jež nesledují cesty struktur ukládání dat v databázi, vznikl v sedmdesátých letech 20. století nový databázový model – relační. O něm vykládá další kapitola.

Dokud však jedinou vnější pamětí bylo medium se sekvenčním přístupem – magnetická páska, pak hierarchické uspořádání dat či kruhové seznamy byly velmi efektivní.

Více o síťových a hierarchických databázích se můžete dočíst ve [GCCC] nebo [PH92].

Ostatní databázové modely

O ostatních, historicky mladších modelech je v těchto skriptech pojednáno podrobněji v dalších kapitolách. V současnosti nejvíce používaným databázovým modelem je relační model, a ten je v těchto skriptech popsán nejpodrobněji. Následující kapitola popisuje základní model relačních databází, ve 2. části skript je popsána problematika navrhování relačních databází, 3.část podává v hlavních rysech přehled jazyka SQL, jenž je standardem jazyka relačních databází.

Mladší databázové modely: objektově-relační, objektový a multi-dimenzionální, jsou v hrubých rysech popsány v dalších kapitolách této 1.části.

RELAČNÍ DATABÁZE

Relační model dat vymyslel v roce 1970 E.F.Codd [Codd]. Úspěšných implementací se však dočkal až s nástupem vnějších pamětí s přímým přístupem – magnetických disků.

„Relační“ znamená, že model vychází z matematického pojmu relace.

Matematická relace je podmnožina kartézského součinu nějakých množin, jejími prvky jsou tedy n-tice prvků těchto množin po řadě ze „správné“ množiny. Relací (což je anglický výraz pro vztah) se nazývá proto, že n-tice interpretujeme jako případy skutečnosti, že prvky n-tice jsou v námi sledovaném vztahu. Například relace A, jejímiž prvky jsou všechny dvojice (x,y), kde x je student a y předmět navštěvovaný tímto studentem, je podmnožinou kartézského součinu množin X – všech studentů a Y – všech předmětů, a je formální reprezentací vztahu „navštěvuje“ mezi studenty a předměty.

Matematické relace lze skládat: když jako B označíme relaci mezi předměty Y a učiteli Z reprezentující vztah „je vyučován“ (jejími prvky jsou tedy dvojice (y,z) kde y je z Y a z je z Z), pak složením relací A a B vznikne relace C mezi studenty X a učiteli Z reprezentující vztah „je učen“. Přesněji: prvky relace C jsou dvojice (x,z), kde x je student – patří do X, a z je učitel – patří do Z, x je učen učitelem z , tj. navštěvuje předmět vyučovaný tímto učitelem.

Podobně složením vztahu „je otcem“ a „je rodičem“ získáme vztah „je dědem“.

S matematickými relacemi lze provádět ještě další operace:

běžné množinové sjednocení, průnik a množinový rozdíl (například sjednocení relací reprezentujících vztah „navštěvuje“ a vztah „má uznán“ reprezentuje vztah „navštěvuje nebo má uznán“; průnik relací „navštěvuje“ a „navštěvoval dříve“ představuje vztah „navštěvuje a navštěvoval dříve“ – tedy „opakuje“ ; rozdíl relací „navštěvuje“ a „navštěvoval dříve“ představuje relaci „navštěvuje a nenavštěvoval dříve“, tedy „navštěvuje poprvé“)

operace relační algebry – projekce (například projekce relace vyjadřující vztah „muž x a žena y jsou rodiči dítěte z“ do složek x a z je relace reprezentující vztah „je otcem“)

další operace relační algebry – selekce neboli restrikce podmínkou (například restrikce relace reprezentující vztah „navštěvuje“ daná podmínkou „předmět je ‘Úvod do databází’ “ je relace tvořená dvojicemi, v nichž druhý prvek je vždy ‘Úvod do databází’ a první prvek je student, který tento předmět navštěvuje)[1].

E. F. Codd spojil pojetí matematických relací s databázovou představou organizace dat a vytvořil nový databázový model. Dnes je tento relační databázový model běžně rozuměn tak, že *data jsou organizována do tabulek* (jež sice zobrazují nějaké matematické relace, ale z hlediska užití v databázích toto nemá běžně žádný význam). Jedna tabulka většinou uchovává záznamy o jednom typu objektů reálného světa, její řádky jsou jednotlivé záznamy (viz předchozí kapitola a výklad o logické struktuře „tabulka“), sloupce jsou tvořeny jednotlivými atributy záznamů – v relačně-databázové terminologii se atributům často říká *pole* tabulky. Schéma tabulky se zapisuje v této syntaxi:

$R(A_1{:}D_1 , A_2{:}D_2 , \dots , A_n{:}D_n)$

kde R je jméno tabulky, $A_1, A_2, \dots , A_n$ jsou jména jednotlivých polí tabulky, $D_1 , D_2 , \dots , D_n$ jejich domény. Například:

ZÁKAZNÍCI (KódZákazníka : text(5) , Firma : text(40) , KontaktníOsoba : text(30), Funkce : text(30), Adresa : text(60), Město : text(15), Region : text(40), PSČ : text(10), Země : text(15), Telefon : text(24), Fax : text(24))

[1] Má smysl zvláště tehdy, když následně za touto restrikcí provedeme projekci do první složky, čímž získáme množinu studentů, kteří navštěvují předmět ‘Úvod do databází’.

je schématem tabulky uchovávající záznamy o jednotlivých zákaznících. Zde jsou jako domény zvoleny datové typy, což je v praxi běžné. Kapitola OBJEKTOVĚ-RELAČNÍ DATABÁZE lépe vysvětluje vztah mezi doménou a datovým typem. V našem příkladu mohly být zvoleny domény také takto:

ZÁKAZNÍCI (KódZákazníka : KódyZákazníků , Firma : JménaFirem , KontaktníOsoba : PříjmeníAJména,
Funkce : JménaFunkcí, Adresa : JménaUlicAČíslaDomů, Město : JménaMěst,
Region : JménaRegionů, PSČ : PoštovníSměrovacíČísla, Země : JménaZemí,
Telefon : TelefonníČísla, Fax : TelefonníČísla)

a domény zde pojmenované mohly být specifikovány zvlášť. Například doména KódyZákazníků mohla být specifikována jako texty délky 5 sestávající pouze ze znaků abecedy a pouze z velkých písmen. Doména JménaFunkcí mohla být specifikována výčtem přípustných hodnot… Všimněte si, že pole Telefon a Fax zde měly stejnou doménu.

Tabulka s daty prvních 16ti záznamů, tak jak by je databáze v určitý okamžik mohla obsahovat, je zobrazena na následující stránce.

Paradigma relačního modelu spočívá v tom, že data jsou vyhledávána event. zapisována jako hodnoty do buněk tabulky, s obsahem buněk nakládáme jako s nedělitelným celkem. Nelze tedy do buňky vložit logickou strukturu seznamu, abychom pak z tohoto seznamu vybírali jednotlivé prvky[2], do buňky nelze vložit logickou strukturu „strukturní skupina“ a vyhledávat či měnit složky takové struktury[3]. *Pole v relačním modelu jsou z hlediska logické struktury databáze atomické.*

Schéma databáze sestává ze schémat jednotlivých tabulek, a eventuelně dalších informací (o nich bude výklad v dalších kapitolách těchto skript). Schéma je možno zobrazit i graficky, na str. 18 je zobrazen příklad jednoho databázového schématu[4], s vynecháním domén. Čáry spojující některá pole představují další informace o schématu databáze. V tuto chvíli je chápejme jako informace o tom, že spojená pole spolu souvisí – „v reálném světě znamenají totéž“. Symboly 1 a ∞ představují další informace: například u polí ČísloZaměstnance v tabulkách Objednávky a Zaměstnanci tyto symboly říkají „pro jednoho zaměstnace lze nalézt popřípadě více objednávek, v nichž je číslo tohoto zaměstannce, ale pro danou objednávku lze nalézt jediného zaměstance se stejnou hodnotou Číslozaměstnance“.

Jak je v příkladu vidět, nejsou všechna pole tabulek na první pohled atomická: např. pole Adresa nebo KontaktníOsoba. Rozhodnutí o této podobě schématu, tento „návrh“, znamená, že nemáme v úmyslu samostatně vyhledávat či ukládat číslo domu nebo příjmení kontaktních osob. Všechny manipulace s poli Adresa a KontaktníOsoba budou prováděny s celky „ulice a číslo domu“ a „jméno a příjmení“.

Vysvětleme některé obtížněji pochopitelné části vzorového schématu: Tabulka Rozpis_objednávek uchovává data o tom, jaké výrobky byly v jaké objednávce objednány, v jakém množství, ceně a s jakou slevou (sleva je číslo mezi 0 a 1, tedy např. 15% je zapsáno jako 0,15). Údaje o příjemci jsou v objednávce proto, že jím nemusí být zákazník. Pole Nadřízený v tabulce Zaměstnanci obsahuje ČísloZaměstnance nadřízeného zaměstnance.

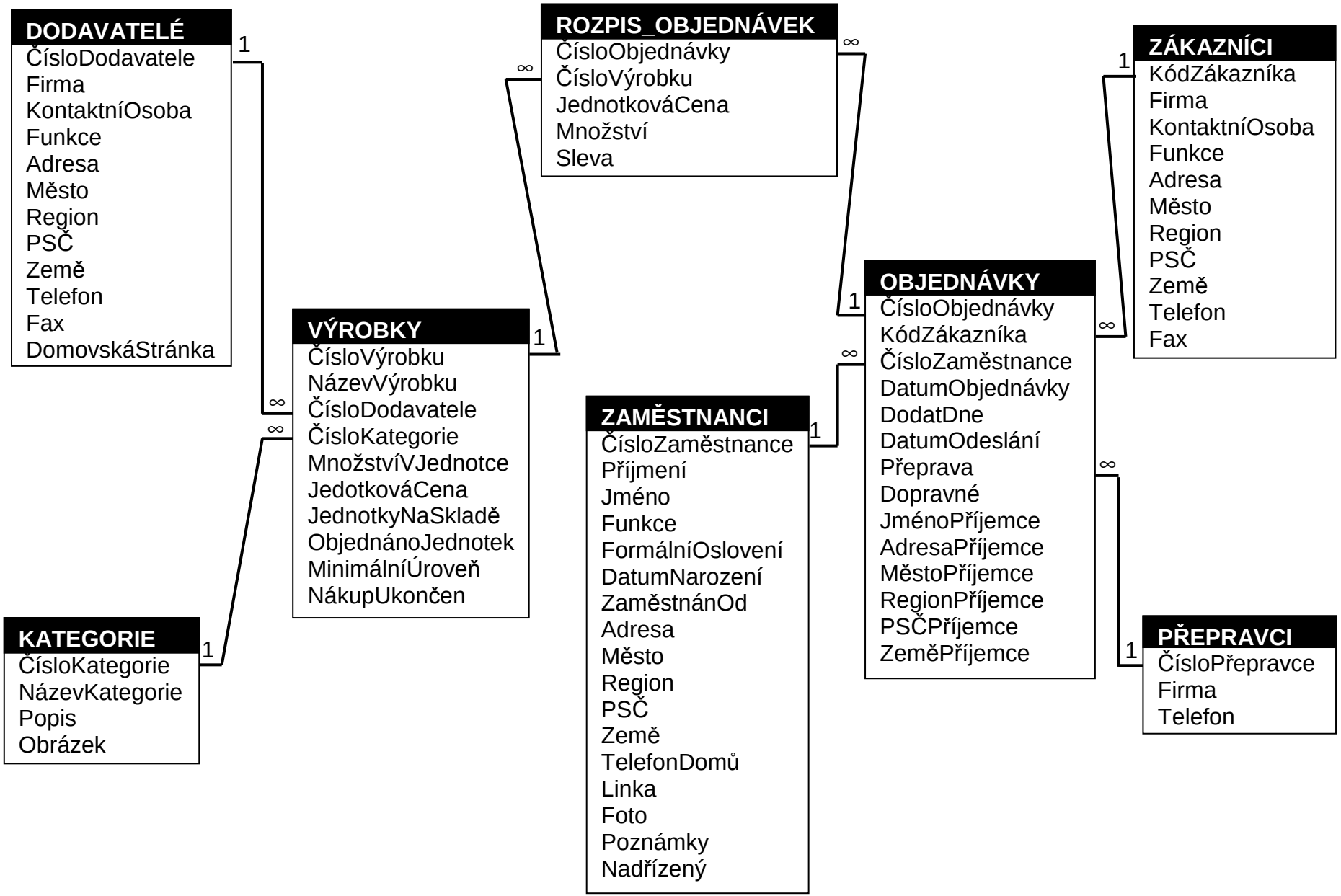
---

[2] tzn. že atributy záznamů nemohou být *vícehodnotové*, o vícehodnotových atributech viz str.33

[3] tzn. že atributy záznamů nemohou být *složené*, o složených atributech viz str. 33

[4] Příklad je převzat z ukázkové databáze Northwind firmy Microsoft dodávané k SŘBD MS Access.

Kód zákazníka	Firma	Kontaktní osoba	Funkce	Adresa	Město	Region	PSČ	Země	Telefon	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Obchodní zástupce	Obere Str. 57	Berlín		12209	Španělsko	030-0074321	030-0076545
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Majitel	Avda. de la Constitución 2222	Mexico D.F.		05021	Mexiko	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taquería	Antonio Moreno	Majitel	Mataderos 2312	Mexico D.F.		05023	Mexiko	(5) 555-3932	
AROUT	Around the Horn	Thomas Hardy	Obchodní zástupce	120 Hanover Sq.	Londýn		WA1 1DP	Velká Británie	(71) 555-7788	(71) 555-6750
BERGS	Berglunds snabbköp	Christina Berglund	Vedoucí nákupu	Berguvsvägen 8	Lulea		S-958 22	Švédsko	0921-12 34 65	0921-12 34 67
BLAUS	Blauer See Delikatessen	Hanna Moos	Obchodní zástupce	Forsterstr. 57	Mannheim		68306	Německo	0621-08460	0621-08924
BLONP	Blondel pere et fils	Frédérique Citeaux	Vedoucí marketingu	24, place Kléber	Štrasburg		67000	Francie	88.60.15.31	88.60.15.32
BOLID	Bólido Comidas preparadas	Martín Sommer	Majitel	C/ Araquil, 67	Madrid		28023	Španělsko	(91) 555 22 82	(91) 555 91 99
BONAP	Bon app'	Laurence Lebihan	Majitel	12, rue des Bouchers	Marseille		13008	Francie	91.24.45.40	91.24.45.41
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Hlavní účetní	23 Tsawassen Blvd.	Tsawassen	BC	T2F 8M4	Kanada	(604) 555-4729	(604) 555-3745
BSBEV	B's Beverages	Victoria Ashworth	Obchodní zástupce	Fauntleroy Circus	Londýn		EC2 5NT	Velká Británie	(71) 555-1212	
CACTU	Cactus Comidas para llevar	Patricio Simpson	Prodejce	Cerrito 333	Buenos Aires		1010	Argentina	(1) 135-5555	(1) 135-4892
CENTC	Centro comercial Moctezuma	Francisco Chang	Vedoucí marketingu	Sierras de Granada 9993	Mexico D.F.		05022	Mexiko	(5) 555-3392	(5) 555-7293
COMMI	Comércio Mineiro	Pedro Afonso	Obchodní zástupce	Av. dos Lusíadas, 23	São Paulo	SP	05432-043	Brazílie	(11) 555-7647	
CONSH	Consolidated Holdings	Elizabeth Brown	Obchodní zástupce	Berkeley Gardens 12 Brewery	Londýn		WX1 6LT	Velká Británie	(71) 555-2282	(71) 555-9199
DRACD	Drachenblut Delikatessen	Sven Ottlieb	Vedoucí nákupu	Walserweg 21	Cáchy		52066	Německo	0241-039123	0241-059428


DODAVATELÉ
ČísloDodavatele
Firma
KontaktníOsoba
Funkce
Adresa
Město
Region
PSČ
Země
Telefon
Fax
DomovskáStránka
VÝROBKY
ČísloVýrobku
NázevVýrobku
ČísloDodavatele
ČísloKategorie
MnožstvíVJednotce
JedotkováCena
JednotkyNaSkladě
ObjednánoJednotek
MinimálníÚroveň
NákupUkončen
KATEGORIE
ČísloKategorie
NázevKategorie
Popis
Obrázek
ROZPIS_OBJEDNÁVEK
ČísloObjednávky
ČísloVýrobku
JednotkováCena
Množství
Sleva
ZAMĚSTNANCI
ČísloZaměstnance
Příjmení
Jméno
Funkce
FormálníOslovení
DatumNarození
ZaměstnánOd
Adresa
Město
Region
PSČ
Země
TelefonDomů
Linka
Foto
Poznámky
Nadřízený
OBJEDNÁVKY
ČísloObjednávky
KódZákazníka
ČísloZaměstnance
DatumObjednávky
DodatDne
DatumOdeslání
Přeprava
Dopravné
JménoPříjemce
AdresaPříjemce
MěstoPříjemce
RegionPříjemce
PSČPříjemce
ZeměPříjemce
ZÁKAZNÍCI
KódZákazníka
Firma
KontaktníOsoba
Funkce
Adresa
Město
Region
PSČ
Země
Telefon
Fax
PŘEPRAVCI
ČísloPřepravce
Firma
Telefon
1
∞

OBJEKTOVĚ-RELAČNÍ DATABÁZE

Před čtením této kapitoly je doporučitelné prostudovat 2. a 3. části skript. Mnoho odkazů a poznámek vám bude jasnější.

Vývoj standardního jazyka pro relační databáze, SQL, se stále více odchyluje od původního jednoduchého relačního databázového modelu. Ve 3.části skript v kapitole PŘEHLED OSTATNÍHO JAZYKA SQL jsou zachyceny rysy jazyka týkající se *chování dat*: uložené procedury a triggery jsou hlavními nástroji pro definici chování dat v rámci objektově-relační databáze. Řízení určitého chování dat je jedním z charakteristických rysů objektově-relačních SŘBD. Další rysy jsou zmíněny dále v této kapitole.

V celkovém souhrnu tyto rysy charakterizují to, co nazýváme *objektově-relačními* databázemi.

Jedním z významných rysů objektově-relačních databází je rozšíření nabídky datových typů – základní datové typy jsou vyjmenovány v kapitole PŘEHLED OSTATNÍHO JAZYKA SQL, objevují se *nové datové typy* jako jsou BLOB – velké binární datové objekty [5], dlouhé texty, časová razítka, časové intervaly, logické odkazy na soubory operačního systému nebo na jiné objekty (např. URL), a jiné. Tyto nabídky je možno dále rozšířit vlastními, tzv. *uživatelsky definovanými typy*.

K SŘBD předních dodavatelů lze přikoupit *doplňující balíky* již hotových definovaných dalších datových a objektových typů.

Každý datový typ se vyznačuje nejen množinou přípustných hodnot a specifickým způsobem kódování, ale také množinou přípustných *operací a funkcí* nad ním. Tedy nové datové typy jsou vybaveny i novými možnostmi manipulace s nimi v rámci jazyka SQL. Uživatelsky definované typy lze vybavit uživatelsky definovanými funkcemi.

Původní idea domén z relačního databázového modelu, jež byla v praxi redukována na volbu datového typu, je rozšířením nabídky typů poněkud přiblížena. Původní ideál však stále zůstává nenaplněn, možnosti definice datových typů jsou omezené.

V intencích předchozí poznámky lze v současné době v SQL definovat pouze tzv. *odlišující typy*, jež jsou založeny na nějakém datovém typu podporovaném SŘBD a od něj se odlišují pouze jménem a předpokládaným použitím, významem.

Dále je možno definovat *strukturované typy* pro ukládání složených atributů.

V definici pole lze využít konstrukce pro *vícehodnotové atributy*, současná verze standardu SQL popisuje konstrukci ARRAY, plánovány jsou i LIST, SET, MULTISET.

Na následující stránce najdete příklady SQL kódu s objektově relačními prvky.

[5] Mohou jimi být obrázky, audio- či video-záznamy, technická data…, ale to vše bez možnosti odpovídajících manipulací s nimi jazykovými prostředky SŘBD. Z hlediska správy dat jsou pouze dlouhými bitovými řetězci.

Příklad 1.
Lze definovat strukturovaný typ:

```
CREATE TYPE adresa  AS
  UliceAČíslo CHAR(40),
  Město CHAR(30),
  PSČ CHAR(5);
```

Tento typ lze použít při vytvoření tabulky či definici řádkového typu (viz dále):

```
CREATE TABLE Osoba (
RČ CHAR(10) PRIMARY KEY,
Jméno CHAR(20),
Příjmení CHAR(30),
Adresa adresa,
DatumNarození DATE,
Děti SET(REF(Osoba)),
REF(RČ),
METHOD Věk(p Osoba) RETURNS INTEGER;
```

Pole Děti obsahuje množinu hodnot odkazů na osoby (viz typ odkazu dále). REF(RČ) specifikuje, že odkaz na řádky tabulky Osoba bude založen na rodném čísle osoby. (Jinou možností by bylo nechat odkazy generovat systémem, tedy vytvořit jakési objektové identifikátory.) Metody mají v definici tabulky pouze hlavičku, tělo je definováno jinde jako uživatelsky definovaná funkce.

Dalším rysem objektově-relačních databází je *pohled na řádky tabulek jako na objekty*. Je možno definovat typ řádku (tzv. *řádkový typ*), podtypy i metody. Tabulku je možno vytvořit podle předem definovaného řádkového typu. Je možno vytvořit pod-tabulku nějaké tabulky, jež eventuelně může mít další specifická pole či metody.

Řádky tabulek vytvořených podle definovaného řádkového typu mají OID (object identifier).

Pro zaznamenávání vztahů lze v objektově-relačních databázích použít nový konstrukt, a to *typ odkazu*. Odkaz vždy ukazuje na nějaký řádek nějaké tabulky. Rozdíl oproti použití cizích klíčů spočívá v tom, že odkaz nemusí mít významovou hodnotu, může být generován systémem. Pole, jež obsahuje odkaz, může být použito v SQL výrazech v konstrukci *cest*, viz následující příklad:

Příklad 2.
Pokud máme definovánu tabulku Účty s polem Vlastník typu odkaz do tabulky Klient, pak můžeme zadat takovýto příkaz

```
SELECT U.Vlastník → Jméno
FROM Účty U
WHERE U.Vlastník → Adresa.Město="Pardubice" AND U.Zůstatek>100000;
```

OBJEKTOVÉ DATABÁZE

Od 90.let se objevují objektově orientované SŘBD. Od objektově-relačních se liší tím, že vznikají jako zcela nové produkty, bez relačně-databázové historie. Jejich možnosti vytváření logických datových struktur jsou mnohem bohatší než je nabídka objektově-relačních struktur. Tyto logické struktury jsou nahlíženy a manipulovány jako objekty (z objektově orientovaného přístupu), tedy zapouzdřují implementační charakteristiky a k manipulaci s nimi jsou k dispozici metody společné pro objekty stejného typu – pro instance téže třídy. Chování dat se stává integrální součástí databáze. Typy či třídy mohou tvořit hierarchii[6], v níž se charakteristiky a chování dědí. Zděděné chování může být změněno (overloading), tj. pod stejným jménem se může skrývat předefinovaná metoda. Objekty jsou nahlíženy jako autonomní, požadavky či oznámení mezi objekty jsou zasílány jako zprávy.

Objekty v OO SŘBD mohou tedy být *složité* , čímž výborně slouží v aplikačních prostředích, v nichž jsou složité objekty přirozenou částí světa zájmu. OO SŘBD nabízejí *efektivní manipulaci* se složitými objekty.

Standardizační skupina ODMG vytváří *Standard objektových databází*, jehož součástí je objektový model OMG, jazyk definice objektů, objektový dotazovací jazyk a specifikace návaznosti na objektově orientované programovací jazyky. Bohužel přední dodavatelé OO SŘBD měli již v době před uvedením standardu svá proprietární řešení, takže standard je naplňován spíše v ideové rovině nežli v syntaktické podobě jazyků příslušných databází.

V současné době OO SŘBD nedosahují výkonnosti silných relačních či objektově-relačních databázových serverů, používají se hlavně v oblastech, kde možnost pracovat se složitými objekty má prioritní důležitost. Například aplikace podporující inženýrský návrh, aplikace pro správu dokumentů, geografické informační systémy, aplikace s časovou dimenzí, aplikace pracující s verzemi, aplikace pracující s multimedii, telekomunikace.

Uveďme nakonec několik příkladů dotazů v jazyce OQL (Object Query Language), bez podrobného vysvětlování syntaxe:

Příklad 3.
Vytvořte seznam jmen pracovníků, jmen jejich dětí s věkem, názvem školy a adresou školy, pro pracovníky s platem nižším než 15 000,-Kč.

```
SELECT STRUCT(Pracovník: Z.Jméno, Dítě: D.Jméno, ŠkolaDítěte: D.Navštěvuje.Název,
AdresaškolyDítěte: D.Navštěvuje.Adresa)
FROM Z IN Zaměstnanec, D IN Z.Děti
WHERE Z.Plat<15000
```

Výsledkem předchozího dotazu je multi-množina struktur.

[6] V některých systémech je povoleno i vícenásobné dědění, tedy struktura tvořená vztahy nadtyp-podtyp netvoří hierarchii, ale síť.

Vytvořte seznamy dětí pracovníků s malým, středním a vysokým platem.

```
SELECT D FROM Z IN Zaměstnanec, D IN Z.Děti
GROUP BY malý: Z.Plat<10000,
         střední: Z.Plat<2000 AND Z.Plat>=10000,
         vysoký: Z.Plat>=20000
```

Výsledkem předchozího dotazu jsou tři strukturní skupiny, každá složená z identifikace skupiny („malý“ nebo „střední“ nebo „vysoký“) a z multi-množiny objektů dětí (mezi zaměstnanci mohou být oba rodiče…).

Vytvořte seznam měst, v nichž bydlí zaměstnanci, bez opakování.

```
SELECT DISTINCT Z.Adresa.Město FROM Z IN Zaměstnanec
```

Výsledkem předchozího dotazu je množina.

Vytvořte seznamy dětí pracovníků v každém oddělení, každý seznam seřazený podle věku dětí sestupně.

```
SELECT  STRUCT(NázevOddělení: O.Název,
               SeznamDětí:
                (SELECT D FROM Z IN O.Zaměstnanci, D IN Z.Děti ORDER BY D.Věk DESC) )
FROM O IN Oddělení
```

Předchozí příklady naznačují, že jazyk objektových databází je silný, ale rozhodně není „uživatelsky přívětivý“, není určen pro ad hoc dotazování pokročilými uživateli. Všechny informační služby musí být vytvářeny programátorským týmem.

Výhodou objektových databází je to, že jak aplikace tak databáze mohou být navrhovány a vytvářeny s objektově orientovaným přístupem.

DATOVÉ SKLADY

Datové sklady (Data Warehouse, DWH) používají jiný databázový model, a to *multi-dimenzionální*. V multi-dimenzionální databázi jsou data uspořádána do vícerozměrných prostorů, analogických křížovým tabulkám, o nichž se zmiňuje **Příklad 27.** na str.66. V případě pouhých dvou dimenzí je možno vytvářet pohledy na data podobné obrázku na následující stránce. Pokud je dimenzí víc (v našem příkladu by jimi mohly být dimenze zákazníků se souhrny podle zemí, nebo dimenze zaměstnanců). Při uživatelském prohlížení dat je možno zobrazit jen dvě ze všech dimenzí.

Fyzickou realizaci může zajišťovat multi-dimenzionální SŘBD, nebo relační SŘBD poskytující pohledy na data v podobě tzv. *datových kostek* – ve standardu SQL je operátor CUBE, jejž je možno používat v SELECT dotazu. Jeho použití ukazuje následující příklad:

Příklad 4.
SQL kód pro vytvoření datové kostky odpovídající tabulce na následující stránce (s omezením na kategorii Maso/Drůbež a rok 1997, abychom omezili velikost tabulky odpovědi):

```
SELECT YEAR(O.DatumObjednávky) AS Rok, QUARTER(O.DatumObjednávky) AS Čtvrtletí,
NázevVýrobku,SUM(Mezisoučet) AS Tržba
FROM ((Objednávky O INNER JOIN Ceny_v_rozpisu_objednávek C ON
O.ČísloObjednávky=C.čísloObjednávky) INNER JOIN Výrobky V ON C.ČísloVýrobku=V.ČísloVýrobku)
INNER JOIN Kategorie K ON V.ČísloKategorie=K.ČísloKategorie
WHERE YEAR(O.DatumObjednávky)=1997 AND QUARTER(O.DatumObjednávky) IN (3,4) AND
NázevKategorie="Maso/Drůbež"
GROUP BY CUBE (YEAR(O.DatumObjednávky, QUARTER(O.DatumObjednávky), NázevVýrobku)
ORDER BY 1,2,3
```

Výsledek je potom následující (v kódu bylo možno v klauzuli GROUP BY CUBE přidat ještě NázevKategorie, potom by tabulka odpovědi měla dvojnásobné množství řádků, podobně jako to způsobil sloupec Rok, ačkoli je v našem případě rok pouze jediný):

Rok	Čtvrtletí	NázevVýrobku	Tržba
1997	3	Alice Mutton	120900,00
1997	3	Mishi Kobe Niku	90937,50
1997	3	Pâté chinois	21600,00
1997	3	Perth Pasties	37515,00
1997	3	Thüringer Rostbratwurst	222822,00
1997	3	Tourtiere	22238,25
1997	3	-	516012,80
1997	4	Alice Mutton	152197,50
1997	4	Mishi Kobe Niku	14550,00
1997	4	Pâté chinois	50400,00
1997	4	Perth Pasties	100737,00
1997	4	Thüringer Rostbratwurst	361157,30
1997	4	Tourtiere	20953,12
1997	4	-	699994,90
1997	-	Alice Mutton	273097,50
1997	-	Mishi Kobe Niku	105487,50
1997	-	Pâté chinois	72000,00
1997	-	Perth Pasties	138252,00
1997	-	Thüringer Rostbratwurst	583979,30
1997	-	Tourtiere	43191,37
1997	-	-	1216008,00

-	3	Alice Mutton	120900,00
-	3	Mishi Kobe Niku	90937,50
-	3	Pâté chinois	21600,00
-	3	Perth Pasties	37515,00
-	3	Thüringer Rostbratwurst	222822,00
-	3	Tourtiere	22238,25
-	3	-	516012,80
-	4	Alice Mutton	152197,50
-	4	Mishi Kobe Niku	14550,00
-	4	Pâté chinois	50400,00
-	4	Perth Pasties	100737,00
-	4	Thüringer Rostbratwurst	361157,3,00
-	4	Tourtiere	20953,12
-	4	-	699994,90
-	-	Alice Mutton	273097,50
-	-	Mishi Kobe Niku	105487,50
-	-	Pâté chinois	72000,00
-	-	Perth Pasties	138252,00
-	-	Thüringer Rostbratwurst	583979,30
-	-	Tourtiere	43191,37
-	-	-	1216008,00

		1997		Celkem z 1997	1998		Celkem z 1998	Celkový součet
NázevKategorie	NázevVýrobku	3	4		1	2		
Cukrovinky		506920,43	541517,39	1048437,82	1094644,9	280703,2	1375348,1	2423785,92
Koření		332875,93	404369,68	737245,61	545128,37	274324,38	819452,75	1556698,36
Maso / Drůbež	Alice Mutton	120900	152197,5	273097,5	124020	79267,5	203287,5	476385
	Mishi Kobe Niku	90937,5	14550	105487,5	7275		7275	112762,5
	Pâté chinois	21600	50400	72000	35400	106110	141510	213510
	Perth Pasties	37515	100737	138252	43542	76670	120212	258464
	Thüringer Rostbratwurst	222822	361157,32	583979,32	551174,97	290906,5	842081,47	1426060,79
	Tourtiere	22238,25	20953,12	43191,37	11817,56	4656,25	16473,81	59665,18
Celkem z Maso / Drůbež		516012,75	699994,94	2024377,69	1216007,69	557610,25	1330839,78	1330839,78
Mléčné výrobky		765854,99	892658,12	1658513,11	1070779,37	882700,5	1953479,87	3611992,98
Mořské produkty		640313,88	516889,86	1157203,74	859897,56	262884,81	1122782,37	2279986,11
Nápoje		456958,74	524351,25	981309,99	2141603,12	601631,25	2743234,37	3724544,36
Obilné výrobky		381113,12	351517,5	732630,62	485205,93	248915,13	734121,06	1466751,68
Plodiny	Chai	29362,5	53212,5	82575	98550	58837,5	157387,5	239962,5
	Longlife Tofu		12812,5	12812,5	5000	5000	10000	22812,5
	Manjimup Dried Apples	71550	314025	385575	116600	160656,25	277256,25	662831,25
	Rössle Sauerkraut	47025	116280	163305	68400	106362	174762	338067
	Tofu	32956,87	18745,31	51702,18		9282,56	9282,56	60984,74
	Uncle Bob's Organic Dried Pears	67500	95662,5	163162,5	203250	104400	307650	470812,5
Celkem z Plodiny		248394,37	610737,81	859132,18	491800	444538,31	936338,31	1795470,49
Celkový součet		3848444,21	4542036,55	8390480,76	7462288,78	3553307,83	11015596,61	19406077,37

Pohled na tržby za výrobky v jednotlivých kategoriích rozdělené podle času, v němž je vybráno období posledních dvou čtvrtletí roku 1997 a první dvě čtvrtletí roku 1998. Tržby jsou zpodrobněny pro výrobky kategorií Maso/Drůbež a Plodiny.

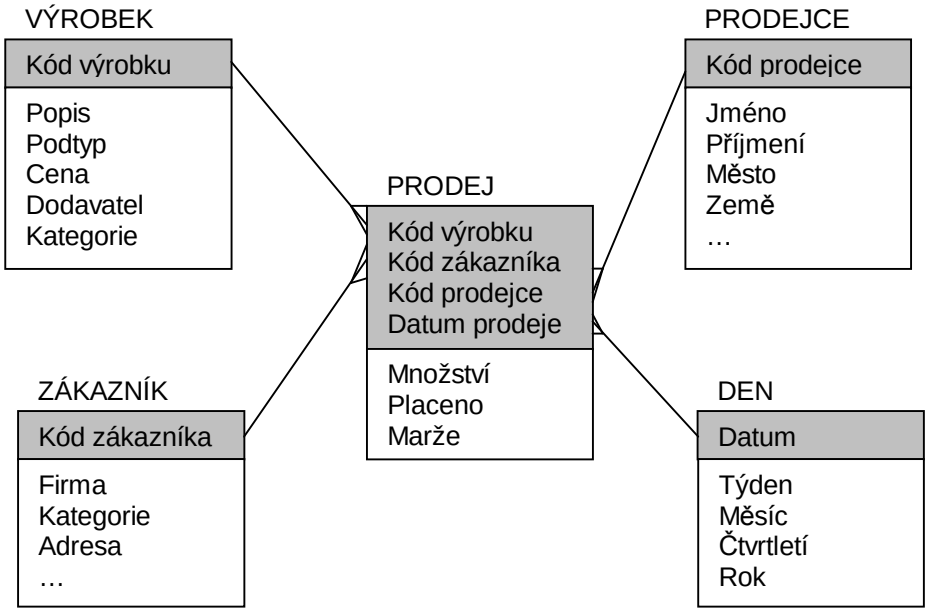
Standard SQL99 obsahuje další jazykové konstrukty pro práci s multi-dimenzionální reprezentací dat z relačních databázových tabulkek.

„Multi-dimenzionální“ fyzická organizace dat je pro účely podobných analytických pohledů na data paměťově úspornější a časově efektivnější, než realizace prostřednictvím tabulek relačního SŘBD. Multi-dimenzionální SŘBD používají vlastní jazyk definice a manipulace dat, podobný jazyku SQL.

Analýza a návrh pro datové sklady

Účelem návrhu pro datový sklad je nalézt metrické funkce, jež budou v multi-dimenzionálních polích ukládány (cena, tržba, množství apod.) za účelem sledování trendů a počítání souhrnů – těmto funkcím se říká *fakty*[7];
a vymezit *dimenz*e, podle nichž budou fakty sledovány. Zároveň je třeba zjistit případné vztahy mezi dimenzemi, či jejich hierarchické členění. Například mezi „Země“ (v níž sídlí zákazník) a „Zákazník“ je vztah zpodrobnění/sbalení (drill down/roll up). Podobně mezi „Rok“, „Čtvrtletí“, „Měsíc“, „Den“. Dimenzí se nazývá vždy *celá taková hierarchická linie*.

Konceptuální návrh se podobá ER diagramům (viz kapitolu KONCEPTUÁLNÍ MODEL DATOVÉ ZÁKLADNY), jen je specifický omezením celkového modelu: modelují se typy dimenzí a typy faktů, a celé schéma sestává z množiny hvězdových schémat[8], viz následující obrázek.

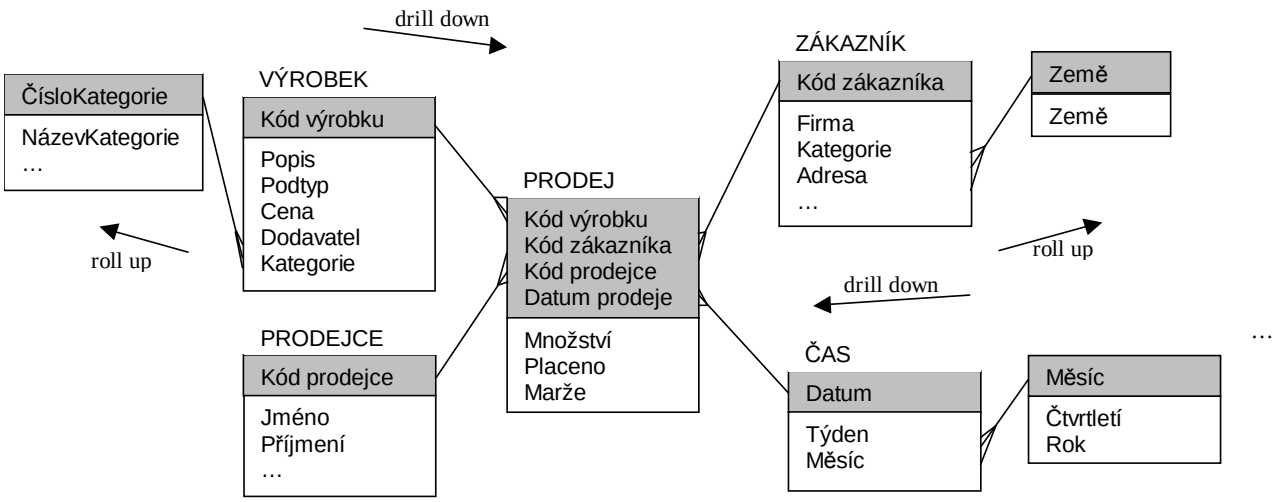


Pro názornost jsou v tomto schématu v obdélníku představujícím typ faktu „Prodej“ zakresleny i cizí klíče, jde tedy spíše o logický databázový model. Ve hvězdovém schématu tvoří typ faktů střed hvězdy, k němuž jsou připojeny jednotlivé dimenze. Povšimněte si, že návrh je nenormalizovaný. Hierarchické členění dimenzí lze znázornit podobně jako na následujícím obrázku:

---

[7] Fakta mohou mít i „myšlenkovou“ povahu, která spočívá v tom, že zaznamenáváme prognózy či předpovědi budoucího vývoje. DWH tedy pak může poskytnou odpověď i na otázky typu: „Jaký byl předpovídaný objem prodeje pro jednotlivá čtvrtletí roku 1999, a jaký byl pak skutečný stav?“

[8] Používají se i jiné varianty, tzv. schémata sněhové vločky, v nichž jsou souhrny faktů připojeny i k jednotlivým úrovním hierarchie dimenzí.



Při návrhu je vhodné, aby nebyly příliš pravidlem závislosti mezi jednotlivými dimenzemi, tj. např. aby prodejci nebyly zaměřeni na určitý omezený sortiment výrobků.

Zvláštní případ nastává, když návrh záznamů faktů neobsahuje žádné číselné ukazatele, jako Množství, Peníze apod. V takovém případě je samotným faktem to, že entity v příslušných dimenzích vstoupily do sledovaného vztahu.

Analýza a návrh se dělá v celé organizaci, pro niž se datový sklad vytváří, protože přínosy plynoucí ze zavedení a provozování DWH rapidně rostou s jeho centralizací – model DWH je jednotný pro celou organizaci. Proto je třeba odhalit synonymii mezi dimenzemi používanými různými organizačními jednotkami a eventuelní souměřitelnost typů faktů (totéž vyjádřeno jinými jednotkami…).

Architektura datového skladu

Často jsou data do datového skladu „pumpována“ z transakčních procesních systémů (TPS). Potom jsou za tímto účelem definovány tzv. *datové pumpy*, jež mají stanoven rozvrh a algoritmus provádění *snímků* z TPS do pomocného datového prostoru, z něhož jsou potom *přidávány* do DWH.

Zdůrazněme, že data jsou do DWH pouze přidávána, a to ještě ve vymezených časových úsecích, tedy DWH je většinu času v klidu co se týče změn dat. V DWH jsou data historická, nejsou zpětně měněna. Fyzické struktury DWH jsou navrženy tak, aby v co nejefektivnější míře poskytovaly možnost tvořit analytické pohledy na data, technologie těchto pohledů se nazývá OLAP[9] (On Line Analytical Processing).

Pro specifické účely některých organizačních jednotek jsou někdy vytvářena tzv. *datová tržiště* (Data Marty), jež vznikají výběrem, eventuelně agregací dat z DWH. Strukturou jsou to také multi-dimenzionální databáze, a proto je projekt DWH někdy veden opačným směrem, integrací meších datových tržišť do centrálního datového skladu. Ovšem tento postup je méně efektivní, důvodem k tomuto postupu může být počáteční omezenost finančních zdrojů pro projekt, konečné náklady jsou však takto vyšší.

---

[9] Zpracování dat v TPS se někdy nazývá OLTP (On Line Transaction Processing).

2. ČÁST -
NAVRHOVÁNÍ DATOVÉ ZÁKLADNY

KONCEPTUÁLNÍ MODEL DATOVÉ ZÁKLADNY

Projekt informačního systému má životní cyklus, který začíná zadáním a sběrem požadavků, pokračuje analýzou využívající interview se zástupci uživatelů, dotazníků uživatelům a pozorování v organizaci, pro kterou vytváříme informační systém. Po analýze následuje fáze návrhu neboli designu, ve které integrujeme jednotlivé požadavky a odstraňujeme nekonzistence, vytváříme jednotný slovník a konečně vytváříme model informačního systému. Po dokončení návrhu následuje implementace spočívající v instalování hardware a software, založení databází, programování … Implementovaný informační systém je následně provozován a udržován, i tato fáze patří k životnímu cyklu projektu, podpora při provozu je běžným požadavkem pro dodávku informačního systému. Pokud se během doby užívání informačního systému objeví nové požadavky, můžou být dodány doplňky nebo provedeny úpravy. Pokud požadavky překračují možnosti stávajícího systému, nastává nový životní cyklus, nový projekt. Tato problematika je důkladněji probírána v samostatném předmětu zabývajícím se projektováním informačních systémů, v tomto úvodu zde je podán pouze stručný přehled.

Tato kapitola se týká fáze analýzy v životním cyklu projektu IS, a to ještě pouze její datové části. Profesí, jíž je třeba k provádění analýzy, je „analytik“. Ten má mít schopnosti komunikace s uživateli a zadavateli, schopnost abstraktního logicko-analytického myšlení, znalost technik modelování. Dále je výhodou obecná zkušenost v analýze IS, a pokud má znalost konkrétní problematiky, je tím jeho práce ještě více usnadněna. Analytik produkuje dokumenty analýzy – dokumenty datové analýzy budou charakterizovány v textu této kapitoly.

Dokumenty analýzy zahrnují přehled požadavků a zjištěných skutečností o jevech týkajících se budoucího informačního systému, a srozumitelný a jednoznačný popis modelu. Tento popis slouží k

- ověření správnosti, potvrzení zadavateli
- podkladu pro implementaci/programování

Navíc je tato dokumentace východiskem pro případné doplňující úpravy vzniklé z požadavků během provozu IS.

Model IS může být členěn do různých částí, v moderní objektově orientované metodice to mohou být např.: model jednání (use case), model spolupráce, objektový model, funkční model, dynamické modely – scénáře, model objektové interakce, mapa událostí, model chování objektu, model událostí (viz např. [Drb1] a [Drb2]). V tradičních metodikách šlo o členění do funkční (diagram datových toků, specifikace procesů/funkcí, stavové diagramy) a datové části (ER diagram, slovník dat). (O tradičním pojetí viz např. [Tiet].) Zhruba lze říci, že model IS zahrnuje

- logiku faktů
- logiku procesů
- „business pravidla“

Model dat, jímž se tato kapitola zabývá, je specifický a práce na něm je relativně osamostatněna, protože

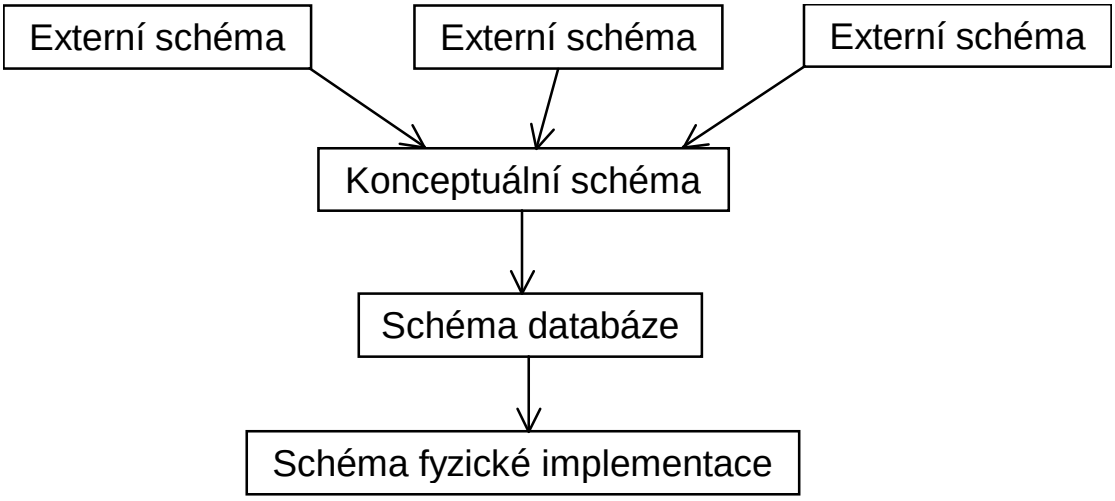
- data mají být společným zdrojem a centrálně řízena
- data mají být spolehlivá
- datová základna má být flexibilní k novým požadavkům (což bylo zkušeností rozpoznáno jako požadavek na to, aby data byla co nejvěrnějším odrazem reality)
- datová základna má umožňovat zvyšování užitné hodnoty dat, jež obsahuje (tzv. koncepce paměti organizace)

Dopady nevhodného návrhu datové základny bývají osudnější, než u ostatních částí.

Analytik nejprve vytváří tzv. konceptuální model[10] datové základny, jenž zároveň říká, jak data v databázi strukturovat, i jakou mají sémantiku. Konceptuální model je implementačně nezávislý popis té části reálného světa, která se týká budovaného IS, je to společný základ pro

- chápání objektů aplikace uživateli, projektanty, správcem databáze
- integraci uživatelských pohledů
- vytvoření vstupu pro návrh implementace

Podle ANSI/SPARC z roku 1975 má návrh sestávat z nezávislých vrstev podle následujícího obrázku, praktické zkušenosti ukazují, že nejvhodnější je postup „shora dolů“ : Externí schémata neboli uživatelské pohledy mají být integrovány do konceptuálního schématu, jež je dále mapováno do schématu databáze. Schéma databáze je pak následně mapováno do fyzického schématu.



Mapování konceptuálního schématu do schématu databáze popisuje kapitola KONCEPTUÁLNÍ SCHÉMA → RELAČNÍ SCHÉMA, převod (logického) schématu databáze do fyzického obvykle provádí SŘBD, některé volby týkající se fyzického schématu může činit správce databáze.

Zdůrazněme: dokumentace k IS má obsahovat **konceptuální model**, jenž

- vylučuje specifika implementace (*zásada konceptualizace*)
- obsahuje vše podstatné (*zásada 100%*)
- je mentálně zvládnutelný pro všechny (*zásada srozumitelnosti a přehlednosti*)

Nejrozšířenějším nástrojem pro zápis konceptuálního modelu datové základny jsou tzv. ER diagramy, vycházející z myšlenek Petera Chena z roku 1976 [Chen]. Do dnešní doby vzniklo několik modifikací základního značení, tyto modifikace se částečně liší i množinou základních konstruktů, z nichž lze model sestavit. I moderní značení objektově orientovaného jazyka UML pro konceptuální modely je v části odpovídající datovému modelu jistým klonem ER diagramů.

ER modelování

ER modelování spočívá ve využití základních konstruktů jazyka pro tvorbu diagramů a v metodice tvorby těchto diagramů. Základní myšlenkou je, že databáze uchovává fakta o entitách a o vztazích mezi entitami.

---

[10] „Konceptuální“ je odvozeno od slova koncept = pojem.

V závěru této kapitoly je uvedeno několik příkladů ER modelů, na nichž si lze objasnit abstraktní výklad. Základní konstrukty ER diagramů jsou následující:
entity, vztahy
atributy (atomické, skupinové, vícehodnotové)
integritní omezení:
–kardinalita vztahu 1:N, M:N, 1:1, jemnější vyjádření (min, max)
–členství ve vztahu povinné, nepovinné
–identifikační vztahy
–klíčové atributy
ISA hierarchie

Entity

Entita je něco co pojmenováváme. V ER modelech modelujeme *entitní typy*, obvykle zobrazované jako pojmenované obdélníky. Volíme je tak, aby
–výskyty entit daného typu vykazovaly *podstatnou podobnost*
–výskyty entit daného typu měly *podobné role* v modelovaném systému

Entitními typy mohou být například „Učitel“, „Student“, „Vyučovaný předmět“, „Učebna“. Jednotlivými entitami (výskyty entitního typu) jsou pak například: učitel Filip Zavoral, student Jan Novák, vyučovaný předmět Úvod do databází, učebna P15 (tj. jméno typu a jméno výskytu).

Výskyty entit musí být identifikovatelné na základě konstrukčních částí užitých v modelu (na základě jejich atributů nebo vztahů s jinými entitami).

Z hlediska užití konceptuálního modelu musí být jasné, co entitním typem myslíme, tj. musí být jasně vymezeno, co do něho patří.

Do dokumentace patří:

- pojmenování entitního typu
- popis entitního typu
- eventuelně pravidla členství v entitním typu, lze-li je formulovat
- eventuelně příklady výskytů k lepšímu pochopení – příklady nejen běžné, ale i výjimečné, ukazující na to, co všechno pod tento typ zahrnujeme

Vztah

Vztah mezi entitami lze popsat větou, ve které vztah vyjádříme *přísudkovou částí*. Například
„Zákazník podává Objednávku.“
„Zaměstnanec Jan Novák má nadřízeného Zaměstnance Petra Műllera.“
„Výrobek je dodáván Dodavatelem.“

Vztah v ER modelu má vždy pevně určeny ty entitní typy, které v něm hrají jednotlivé role: „Zákazník“ hraje roli toho, kdo podává, „Objednávka“ hraje roli toho, co je podáno. „Výrobek“ hraje roli toho, co je dodáváno, „Dodavatel“ hraje roli toho, kdo dodává. V případě druhého z výše uvedených vztahů jde o tzv. *involutorní* vztah, což znamená, že obě role (nadřízený, podřízený) hrají entity typu „Zaměstnanec“.

Do ER diagramů bývá volitelně možno zapsat i pojmenování rolí ve vztahu (například „nadřízený“ a „podřízený“).

Vztahy mohou být binární (vztah dvou entit), ternární (vztah tří entit – např. „Student skládá zkoušku z Předmětu u Učitele“) i více-ární.

> Ternární a více-ární vztahy lze vždy rozložit do binárních vytvořením tzv. vazební entity, ne vždy je však toto řešení nejvhodnější. Dá se však říci, že potřeba použít v modelu skutečně více-ární vztah je spíše výjimečná.

Pro pojmenování vztahu v modelu použijeme slovesný tvar, u binárních vztahů bývá vhodné použít tvar pro čtení „tam“ i tvar pro čtení „zpět“: například „Učitel učí Předmět“ i „Předmět je učen Učitelem“ [11].

> Pro pojmenování vztahu většinou používáme pozitivní formulaci, protože chceme, aby IS zaznamenával to co je, a ne to co není. Ve výjimečných případech, kdy pravděpodobnost toho, že něco je, je vysoká, a my chceme zaznamenávat ty výjimečné případy, kdy to není, můžeme použít negativní formulaci: například „Učitel nesmí zkoušet Studenta“ (protože to například děkan zakázal). V mnoha případech lze ovšem i pro takovéto vztahy nalézt pozitivní formulaci: například „Učitel má zakázáno učit Studenta“ – pozitivně nastal fakt zákazu.

Někdy je užitečné si uvědomit, že modelujeme *typ* vztahu, jehož výskyty si lze představit jako řádky tabulky:

„má nadřízeného“	**Podřízený**	**Nadřízený**
	Davolio, Nancy	Callahan, Laura
	Leverling, Janet	Fuller, Andrew
	Peacock, Margaret	Fuller, Andrew
	Buchanan, Steven	Fuller, Andrew
	Suyama, Michael	Buchanan, Steven
	King, Robert	Buchanan, Steven
	Callahan, Laura	Fuller, Andrew
	Dodsworth, Anne	Buchanan, Steven

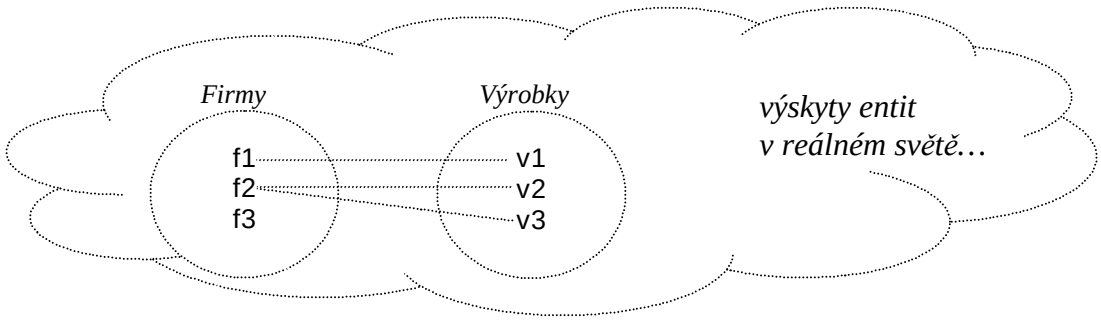
U vztahu zjišťujeme a do diagramu zakreslujeme tzv. *kardinalitu vztahu*. Kardinalita vztahu říká, kolik výskytů entit jednoho typu může být v daném vztahu s jedinou entitou druhého typu – říkáme, že vztah je buď 1:1 nebo 1:n nebo n:1 nebo n:m. Kardinalitu vztahu lze dobře ověřovat na tabulce s příklady podobné tabulce uvedené nad tímto odstavcem (v našem příkladu vidíme, že každý podřízený je v ní pouze jednou). Pro zakreslení kardinality vztahu v diagramu je několik ustálených značení – viz ukázky na obrázku na následující straně.

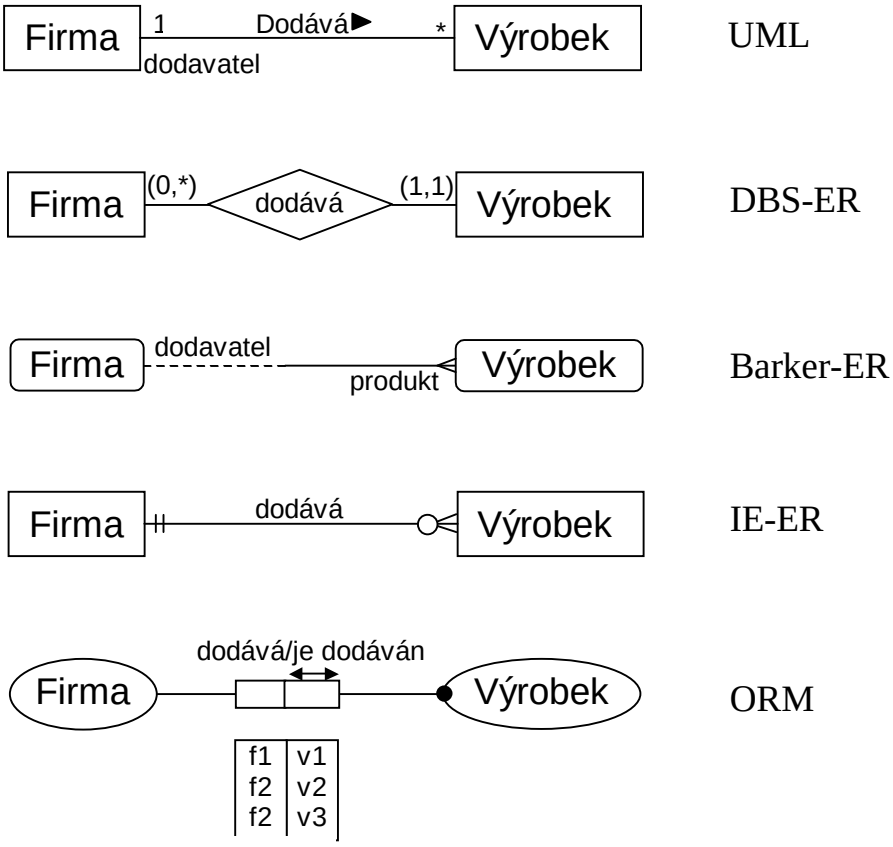
Dále zjišťujeme tzv. *povinnost členství ve vztahu*. Ta říká, zda všechny výskyty entitního typu, jenž je určen pro danou roli v tomto vztahu, musí do tohoto vztahu skutečně vstupovat – například ne všichni učitelé musí v konkrétním semestru učit nějaký předmět, někteří mohou mít tzv. sabbatický semestr. Ne všichni studenti musí v daném semestru navštěvovat nějaký předmět, student může studium přerušit. I pro povinnost členství ve vztahu je několik ustálených značení. Obrázek na následující straně podává přehled nejdůležitějších značení. Doplňující informací může být, kolik entit daného typu obyčejně v daném vztahu je. To lze například zaznamenat do tabulky připojené k modelu. Příklad takové tabulky je na dalším obrázku na následující straně.

---

[11] Některé školy učí pojmenovávat vztahy jmenným tvarem odpovídajícím příslušnému slovesnému tvaru, v tomto příkladu by pojmenování vztahu bylo „Výuka“. Pro pochopení uživateli je tento způsob zápisu do diagramu méně vhodný než slovesný tvar, který lze přirozeně číst jako věty.

Někdy do konceptuálního modelu zahrnujeme i informaci o tom, zda je vztah *přenositelný*. Například „Učitel učí Předmět“ může být přenositelný vztah, když se učitelé ve výuce daného předmětu střídají semestr od semestru. Pro přenositelnost nebývá ustálené značení.





Různá značení kardinality a povinnosti členství ve vztahu

	0	1	N	
OSOBA	95%	0,01%	4,99%	ROMÁN
ROMÁN	0%	99,99%	0,01	OSOBA

Tabulka % výskytů ve vztahu: 95% osob nenapsalo žádný román, 0,01% napsalo 1 román, 4,99% napsalo více románů; neexistuje román bez autora, 99,99% románů má jediného autora, 0,01% románů má více autorů.

Mezi vztahy, jež chceme mít v informačním systému zachycené, určíme vztahy základní, z nichž lze ostatní vztahy v modelu odvodit. Ostatní vztahy jsou *odvozené*. Například „Učitel učí Studenta“ je odvozeno z „Učitel učí Předmět“ a „Student navštěvuje Předmět“. Ne všechny odvozené vztahy je nutno do modelu zahrnovat, spíše jde o výjimku – důvodem k zakreslení odvozeného vztahu do modelu může být to, že ho uživatelé specielně pojmenovávají a častokrát je používán. Odvozené vztahy v modelu nějakým zřetelným způsobem označíme, a k modelu připojíme definici odvození.

Do dokumentace patří:

- pojmenování vztahu, popis vztahu
- entitní typy hrající role
- kardinalita vztahu
- povinnosti členství ve vztahu
- eventuelně tabulka nebo diagram příkladů pro vysvětlení kardinality a povinnosti
- popis odvození odvozených vztahů
- eventuelně statistika členství ve vztahu
- eventuelně informace o přenositelnosti vztahu

Role

Do diagramů můžeme eventuelně zakreslovat i role, pokud je to v konkrétním případě vhodné. Viz role „dodavatel“, „produkt“ z obrázku na předchozí straně.

Atribut

Atribut je modelovaná *vlastnost entit nebo vztahů* zahrnutých do modelu. Lze si ho představit jako funkci z entitního (nebo vztahového) typu do množiny hodnot z určité domény. Daný entitní nebo vztahový typ má v modelu vyjmenovány atributy, jež se o výskytech tohoto typu budou zaznamenávat. Atribut může být buď

- *povinný* – každý výskyt daného typu musí mít tuto vlastnost zaznamenanou v IS
- *volitelný* – některé výskyty daného typu tuto vlastnost mít zaznamenanou nemusí; buď z důvodu, že pro ně nepřipadá v úvahu (výskyt tuto vlastnost nemá), nebo proto, že je její hodnota neznámá, nebo nakonec z důvodu, že její hodnotu zaznamenat nechceme protože ji například chceme utajit

Model můžeme doplnit o informaci, zda se hodnoty daného atributu pro konkrétní výskyty mohou měnit, či jsou stálé, *neměnné*.

Atribut, jenž může nabývat množinu hodnot z dané domény, se nazývá *vícehodnotový*. – Například u entitního typu „SmluvníPartner“ můžeme mít atribut „JménoKontaktníOsoby“, jenž pro konkrétní výskyty typu, tj. konkrétní partnerské firmy, nabývá i více hodnot, tj. zaznamenáváme eventuelně více jmen kontaktních osob u jednotlivé partnerské firmy .

Atribut, jenž má vnitřní významovou strukturu, se nazývá *složený* neboli *strukturovaný* (někdy též *skupinový*) – například u osob chceme zaznamenávat adresu ve struktuře: ulice a číslo, město, PSČ, stát. Můžeme tedy do modelu zakreslit atribut „Adresa“, jenž bude složen z „UliceAČíslo“, „Město“, „Stát“, „PSČ“. Takto to uděláme proto, že pojem „Adresa“ je běžně používaný, a chtěli bychom ho používat i v komunikaci s budoucím IS; stejně tak ovšem chceme hovořit o „Městě“, „Státu“ – budeme se chtít ptát jak na adresu jako celek, tak v jiných případech na jednotlivé složky

adresy. Navíc můžeme někde v modelu definovat, z čeho je složen atribut „Adresa“, a pak ho použít u různých entitních typů, například u zaměstnanců i u smluvních partnerů.

Atributy, jež nejsou vícehodnotové ani složené, jsou *atomické*. Na hodnotu atomického atributu se v IS budeme ptát vždy pouze jako na celek.

Některé vlastnosti entit se dají odvodit z vlastností jiných entit, jež jsou s nimi ve vztahu, pak takové vlastnosti do modelu nezahrnujeme. Například jméno dodavatelské firmy nezahrneme do modelu jako atribut entitního typu „Výrobek“, pokud je v modelu zahrnut i vztah „dodává“ a atribut „Firma“ entitního typu „Dodavatel“. Heslovitě tuto zásadu lze vyjádřit: *každý atribut je v modelu pouze jedenkrát*. Tím se myslí: každá vlastnost zjistitelná pro nějakou entitu je v modelu zakreslena nejvýše jedenkrát.

> Pokud se v konceptuálním modelu vyskytují dva atributy stejného jména (musí být u různých entitních nebo vztahových typů), znamenají tedy každý něco jiného. Například atributy „Adresa“ u zákazníků a u dodavatelů, nebo běžně používané atributy „Jméno“.[12]

Někdy ovšem máme dobrý důvod do modelu zahrnout i atribut, jejž lze z modelu odvodit, takový atribut se nazývá *odvozený*. Dobrým důvodem může být to, že uživatelé takovýto atribut pojmenovávají a často používají. Například věk lze odvodit z data narození, do modelu pravděpodobně zahrneme „DatumNarození“, a můžeme zakreslit „Věk“ jako odvozený atribut. Odvozené atributy *specielně označíme* a k modelu přidáme informaci o tom, jakým způsobem je odvozován.

Do dokumentace patří:

- jméno atributu
- popis atributu
- eventuelně specifikace domény atributu, eventuelně příklady hodnot
- povinnost/volitelnost
- eventuelně stálost/proměnlivost
- popis odvození odvozených atributů

Klíč

Klíč je *minimální* skupina atributů jednoznačně identifikující entity daného typu. V některých případech může pro daný entitní typ existovat více alternativních klíčů. Jednoduché klíče obvykle v diagramu podtrháváme, v notaci UML se používá označení (PK) za jménem atributu. U složených nebo alternativních klíčů je třeba jasně označit, zda máme na mysli složený klíč nebo několik alternativních klíčů. Obvykle se to řeší tak, že podtržením či značkou (PK) vyznačíme všechny složky tzv. primárního klíče, a pro eventuelní alternativní klíče použijeme jiné označení. Nebo je možno tyto informace uvést v komentáři.

Primární klíč je klíč, který jsme z několika alternativních klíčů zvolili za hlavní.

---

[12] Při pojmenovávání je vhodné dodržovat tuto zásadu: stejně pojmenovávejme jedině ty atributy, jejichž hodnoty má smysl porovnávat. V modelech, které umožňují pojmenovávat i domény atributů, se tato zásada přesouvá na jména domén.

Identifikační závislost

Některé entitní typy nelze jednoznačně identifikovat na základě jejich atributů (nemají v konceptuálním modelu klíč), pak k identifikaci výskytů je zapotřebí ještě vztahů, v nichž hrají roli. Například entitní typ „Dítě“, jehož výskyty budou děti zaměstnanců, může mít pouze atributy „Jméno“, „DatumNarození“ a „Pohlaví“. Na základě těchto údajů nemusí být možné dítě identifikovat, k jednoznačné identifikaci dítěte bude zapotřebí ještě vědět, který zaměstnanec je rodičem tohoto dítěte.

V případě identifikačních závislostí se často začínající analytici dopouštějí chyby v tom, že mezi atributy identifikačně závislé entity zahrnou i klíče entit, na nichž jsou tyto identifikačně závislé. V našem příkladu by bylo chybou mezi atributy entity „Dítě“ zahrnout i atribut „KódZaměstance“, kde „KódZaměstance“ identifikuje rodiče dítěte . Fakt o rodičovství by pak byl v modelu zahrnut dvakrát, jednou jako vztah „je rodičem“ a podruhé jako atribut entity „Dítě“.

Identifikačně závislé entity jsou *vždy existenčně závislé* na entitách, na nichž jsou závislé identifikačně. – Nemůžeme mít záznam o dítěti, které není potomkem některého ze zaměstnanců. „Existenční závislost“ znamená povinnost členství ve vztahu – dítě musí mít rodiče.

ISA hierarchie

V ER digramu můžeme vytvořit dvojici entitních typů, z nichž jeden je nadtypem druhého. Důvodem k tomu může být

- generalizace – několik entitních typů má určité společné rysy, a ty chceme zachytit v obecnějším typu

  Například společné vlastnosti studentů a učitelů (jméno, příjmení, adresa, datum narození…) chceme zachytit v nadtypu „Osoba“.

- specializace – některé výskyty entit určitého typu mají další, specifické charakteristiky, jež chceme v modelu podchytit

  Například někteří zaměstnanci absolvovali školení řidičů, a proto smějí vozit kolegy na služební cestu osobním autem. O nich chceme podchycovat i fakta o posledním školení, o uskutečněných jízdách, a o tom koho vezli. Vytvoříme proto podtyp typu „Zaměstnanec“, který nazveme „Řidič referent“.

- klasifikace – každou entitu určitého typu musíme zatřídit do některé z podskupin

  Například každý učitel je buď lektor, odborný asistent, docent nebo profesor. (Takovýto důvod je však zřídkakdy dostatečný k tomu, abychom vytvářeli samostatné typy. V tomto případě by to byly typy „Lektor“ , „Odborný asistent“, „Docent“, „Profesor“.)

Dvojice typů, z nichž jeden je nadtypem druhého, je v ER diagramu spojena vztahem „ISA“. Název pochází s anglického „is a“ – například „Učitel je Osoba“ zní v angličtině „Teacher <u>is a</u> Person“. Vztah ISA bývá obvykle značen šipkou spojující oba entitní typy, směřující od podtypu k nadtypu.

Při tvorbě nadtypů za účelem generalizace je třeba pamatovat na účel konceptuálního modelu: je společným základem, jejž mají chápat uživatelé, analytici i programátoři. Nelze tedy na konceptuální úrovni extrémně generalizovat a vytvářet umělé entitní typy, jimž uživatelé nerozumí.

Někdy zvažujeme vytvoření podtypu z těch výskytů nějakého entitního typu, jež vstupují do uvažovaného vztahu, ve kterém obecně entity tohoto typu nemají povinné členství. Například ve vztahu „Firma zaměstnává Osobu“ některé osoby do něj skutečně vstupují, jsou tedy zaměstnanci, jiné nikoli. Při modelování dané oblasti zájmu můžeme chtít vytvořit podtyp „Zaměstnanec“. Toto je na zvážení vlastně u každého vztahu s nepovinným členstvím, množství podtypů takto vzniklých by mohlo být značné. Oprávněnost volby vytvořit podtyp by měla být založena ještě na dalších specifických rysech uvažovaného podtypu. I v tomto případě platí, že všem objektům modelu mají uživatelé rozumět, tedy i uvažovanému podtypu.

Příznakem naznačujícím, že vytváříme umělý typ, jemuž nebude rozuměno, může být to, že máme potíže nově vzniklý typ pojmenovat.

Některý podtyp je možno zcela přesně definovat na základě atributů nadtypu nebo vztahů, do nichž nadtyp vstupuje. Podtyp je pak vlastně odvozeným typem. V takovém případě má být do konceptuálního modelu připojeno pravidlo odvození.

Do dokumentace patří:

- pravidlo odvození u odvozených typů

Další integritní omezení

Nástroje pro tvorbu konceptuálních modelů (CASE nástroje) různě podporují možnost zahrnout do modelu všelijaká integritní omezení. Například „jestliže Zaměstnanec je vedoucím Oddělení, pak tento Zaměstnanec je zaměstnán v tomto Oddělení“ je pravidlo říkající, že vztah „je vedoucím“ je podmnožinou vztahu „je zaměstnán“. V některých značeních existují specifické grafické symboly pro zahrnutí takovýchto omezení. Vždy zbývají ovšem integritní omezení, pro něž grafické symboly nemáme.

Například v IS pro knihovnu můžeme chtít podchytit „business pravidlo“, že čtenář si smí rezervovat knihu pouze tehdy, když nemá v knihovně žádný dluh a u žádné ze svých aktuálních výpůjček nepřekročil výpůjční lhůtu.

Ke konceptuálnímu modelu připojujeme takovéto další informace ve formě textových poznámek, i při nich pamatujeme na zásadu srozumitelnosti. Není až tak nutné, aby pravidla byla vyjádřena formálně, jako aby byla pochopitelná a jednoznačně vyjádřena.

Můžeme váhat, zda nějaké integritní omezení do konceptuálního modelu zahrneme, když si myslíme, že v informačním systému nebude implementováno. V takovém případě je vhodné řídit se zásadou, že znalost tohoto „business pravidla“ je cenou informací, a do nějaké dokumentace je zahrnout s tím, že může být implementováno později. Nebo může jít o informaci o tom, že toto pravidlo dodržují sami uživatelé IS, a eventuelně ho využít v logice procesů.

Do dokumentace patří:

- integritní omezení, jež mají být implementována
- eventuelně integritní omezení, jež nemíníme implementovat

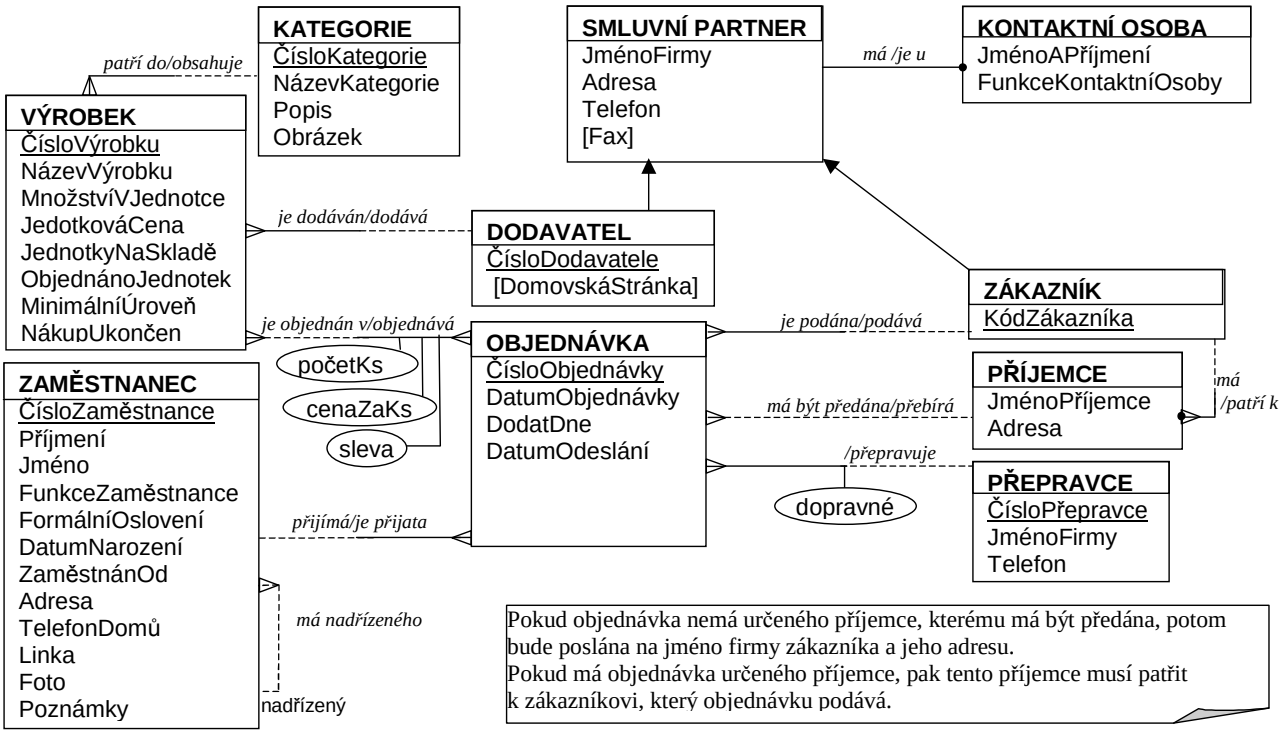
Příklady ER modelů

Následují příklady ER modelů, doplněné o komentáře. Jako značení pro vztahy, kardinalitu a povinnosti členství zde použijeme notaci z Barker-ER. Atributy vztahu budeme zapisovat jako

oválky připojené k typu vztahu, a atributy entity do obdélníku pro daný entitní typ. Ostatní značení bude okomentováno.

Northwind

Vzorová databáze z příkladu na str. 18 může mít konceptuální schéma:



Komentáře ke značení: ISA vztah je značen šipkou ➔, nepovinné atributy jsou uzavřeny do hranatých závorek [ ]. Čtení vztahů po směru hodinových ručiček okolo čáry používá část před lomítkem, čtení proti směru hodinových ručiček používá část za lomítkem. Identifikační vztah je označen znakem ● u spojnice identifikačně závislého entitního typu (u „Příjemce“ a „Kontaktní Osoba“) (značení převzato z IDEFIX). Domény atributů jsou vynechány. Atribut „Adresa“ má být složený, což by bylo lze zapsat jako Adresa(UliceAČíslo, Město, Země, [Region], PSČ).
S doménami by byly atributy zapisovány takto

DodatDne : Datum

Adresa (UliceAČíslo : JménaUlicAČíslaDomů, Město : JménaMěst , Země : JménaZemí , [Region : JménaRegionů] , PSČ : PoštovníSměrovacíČísla)

Domény atributů není třeba zakreslovat do diagramu, eventuelně můžou být zaznamenány v jiné části dokumentace. Pokud je model velmi rozsáhlý, může být rozdělen do více diagramů, v nichž se některé entitní typy opakují. Potom mohou být atributy entit rozptýleny do různých diagramů, vždy tam, kde se týkají modelované problematiky.

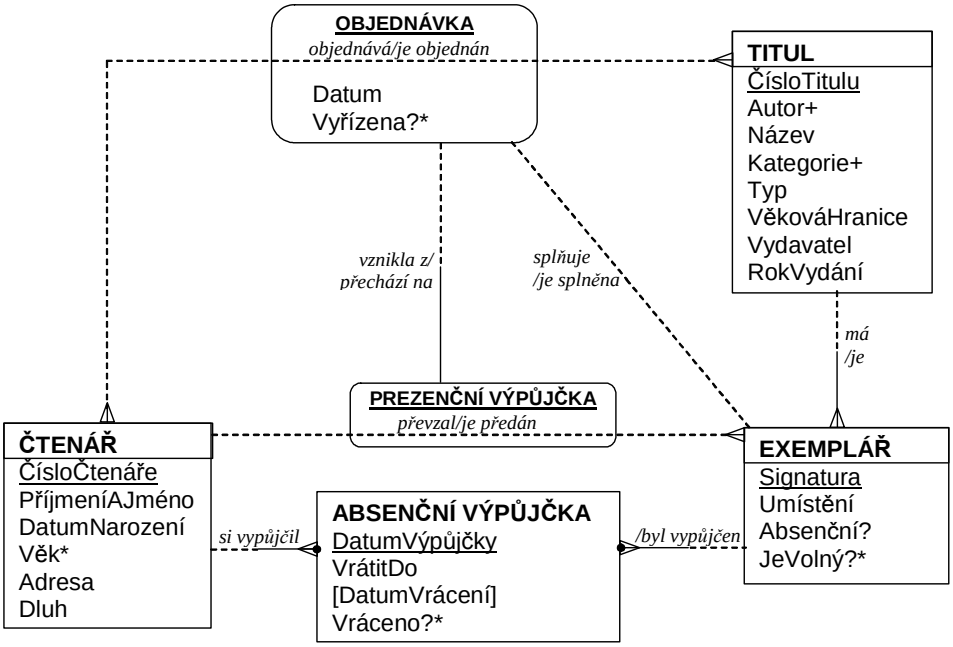
Volba typů „Příjemce“, Smluvní partner“ a „Kontaktní osoba“ nebyla nutná, vyšla z názoru tvůrce modelu. Jevu, že pro danou oblast zájmu může analytik zkonstruovat model různými způsoby, teoretici říkají *sémantický relativismus*. Nicméně, některé modely jsou pro danou oblast zájmu vhodnější a jiné méně. Model má zobrazovat svět zájmu tak, jak ho vidí uživatelé.

Volba entitního typu „Kontaktní osoba“ umožňuje bezproblémově změnit informační systém tak, abychom mohli mít záznamy o více kontaktních osobách pro jednotlivé partnerské firmy. Stačilo by změnit vztah 1:1 na 1:n. Konceptuální model by si zachoval téměř stejnou podobu.

Jak pochopíme později, při zmapování do databázového schématu v podobě uvedené na str.18 jsou zanedbány některé konstrukce konceptuálního modelu zde uvedeného.

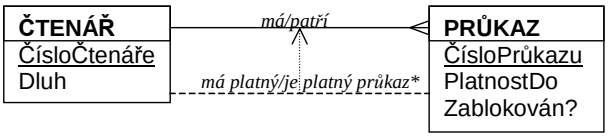
Knihovna

Konceptuální schéma pro IS městské knihovny (vynechána problematika rezervací), 1.diagram:



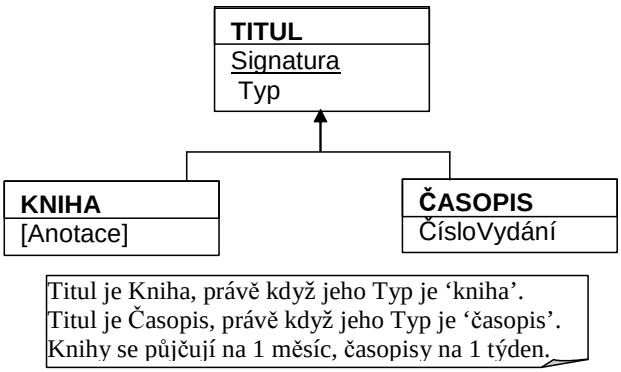
Autory zapisujeme ve formě „příjmení, jméno“.
Věk čtenáře se vypočítává z aktuálního data a datumu narození.
Titul smí být objednán čtenářem nebo čtenářem vypůjčen jeho exemplář pouze tehdy, když věk čtenáře není menší než věková hranice titulu.
Objednávka může být splněna jedině exemplářem objednaného titulu.
Vyřízená objednávka přechází do prezenční výpůjčky, když si čtenář vyzvedne exemplář, který objednávku splňuje. Po vrácení prezenčně zapůjčeného exempláře nebo jeho absenčním vypůjčení je smazána objednávka, ze které prezenční výpůjčka vznikla.
Exemplář je volný, když nesplňuje nějakou objednávku ani nebyl vypůjčen a.dosud nevrácen.
Objednávky jsou vyřizovány tentýž den.
Nevyzvednuté objednávky starší 2 dnů jsou zrušeny (smazány).
Jestliže je absenční výpůjčka je vrácena, pak musí mít vyplněno datum vrácení.
Dluh čtenáře se aktualizuje kromě jiného také podle zpozdného plynoucího z pozdě vrácených absenčních výpůjček.

Znakem + jsou označeny vícehodnotové atributy, znakem * jsou označeny odvozené atributy. „Objednávka“, „Prezenční výpůjčka“ jsou vazební entitní typy. – 2. diagram:



Čtenáři patří nanejvýš jeden nezablokovaný průkaz, a ten je jeho platným průkazem.
Jestliže má čtenář nenulový dluh, je jeho platný průkaz zablokován. Jestliže má čtenář nevrácenou absenční výpůjčku s překročenou lhůtou vrácení, je jeho platný průkaz zablokován.
Pokud čtenář nahlásí ztrátu průkazu, je tento zablokován.
Pokud je překročena platnost průkazu, je zablokován.

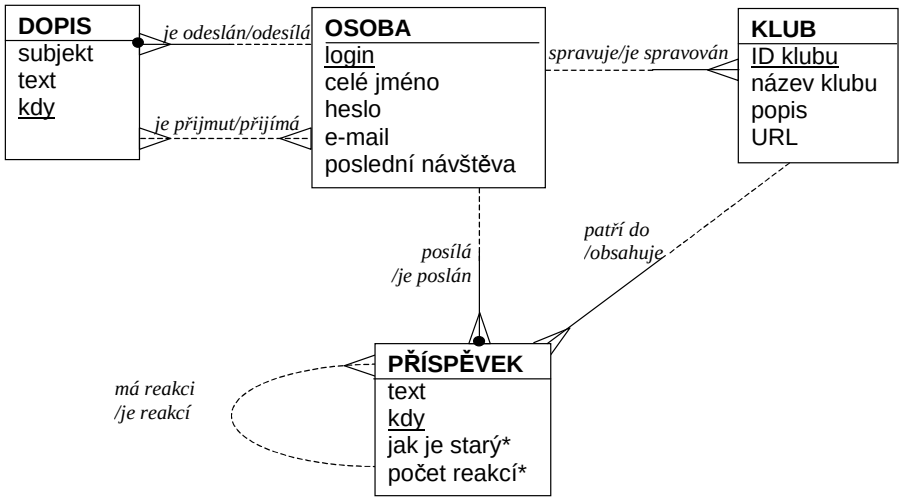
Znakem * je označen odvozený vztah. Tečkovaná šipka značí inkluzi mezi vztahy. (Všimněte si, že atribut „Zablokován?“ není zcela odvozen v rámci tohoto IS!) – 3.diagram:



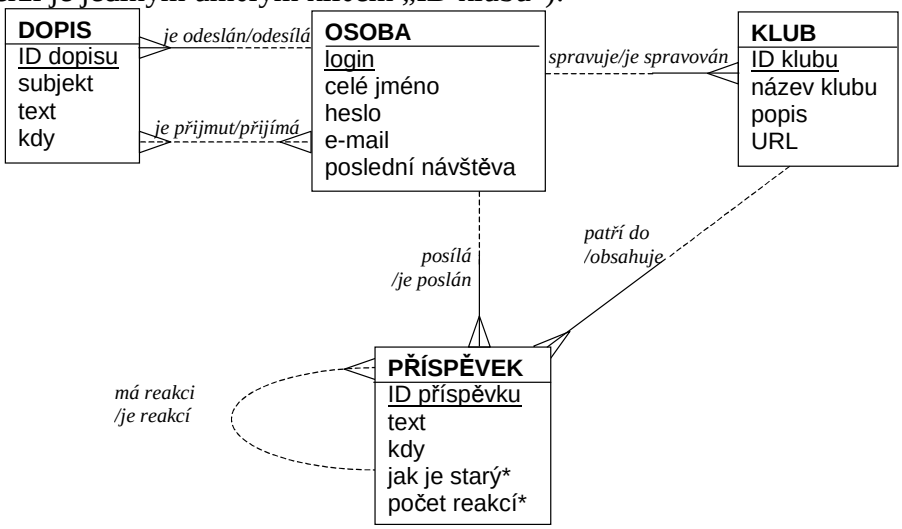
Podtypy „Kniha“ a „Časopis“ jsou v tomto případě odvozené.

Diskusní klub

Konceptuální schéma pro IS internetového diskusního klubu:



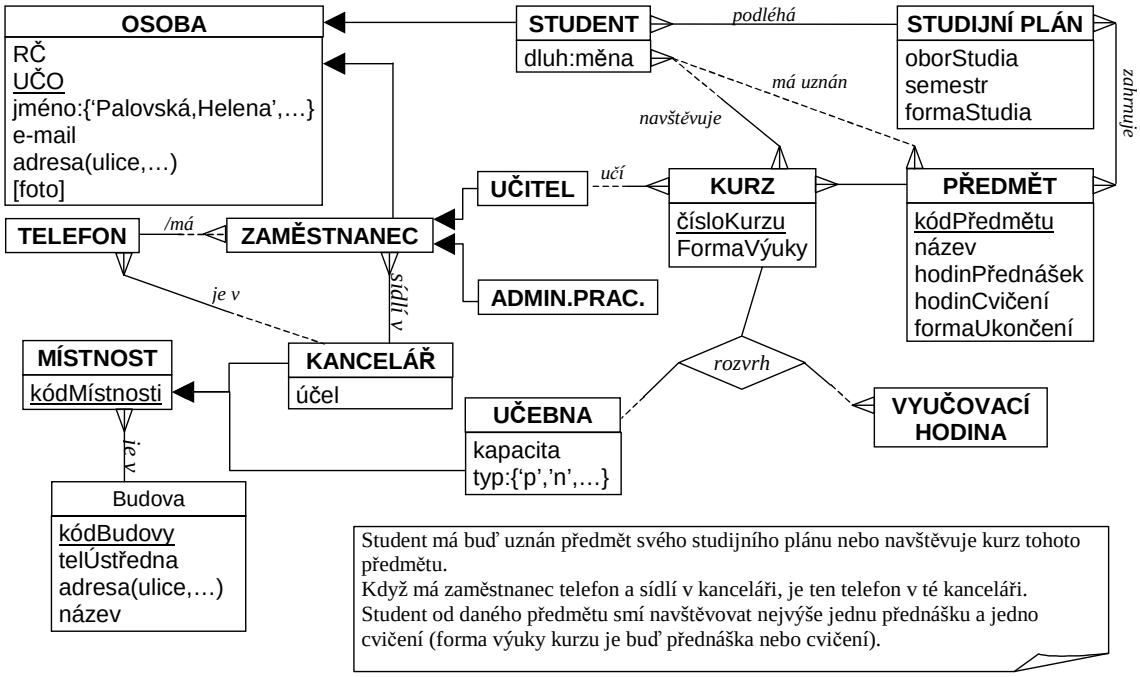
Tato verze obsahuje identifikačně závislé entitní typy „Dopis“ a „Příspěvek“. Složená identifikační schémata těchto entit by mohla být na překážku, následující verze obsahuje další umělé klíče (v předchozí verzi je jediným umělým klíčem „ID klubu“):



Počet reakcí na příspěvek a stáří příspěvku jsou odvozené atributy, ale zatímco počet reakcí bude aktualizován s novou reakcí na příspěvek, stáří příspěvku bude vypočítáváno na požádání. Tyto otázky ovšem již nejsou konceptuální, týkají se implementace.

IS soukromé VŠ

Konceptuální schéma pro část informačního systému soukromé vysoké školy. Tato část nezahrnuje problematiku přihlašování, zkoušek a zápočtů.



Ternární vztah (zde rozvrh) je značen kosočtvercem s pojmenováním uvnitř, připojeným k entitním typům hrajícím role v tomto vztahu.

Samostatné entitní typy „Telefon“ a „Vyučovací hodina“ mohou budit rozpaky, když de facto jde o hodnotové typy (Telefon vyjádříme telefonním číslem, vyučovací hodinu kódem např. „Po3“, „St4“). Zde jsou použity namísto atributů entity proto, aby byla lépe zachycena sémantika integritního omezení týkajícího se telefonů, a kardinality vztahu „rozvrh“.

Kardinalita ternárního typu vztahu ve zde použité notaci se chápe takto: muří noha znamená, že dvojice ostatních dvou entit ve vztahu neurčuje jednoznačně tuto entitu, připojení bez muří nohy značí, že dvojice ostatních dvou entit ve vztahu tuto entitu určují jednoznačně. Konkrétně v našem příkladu: Daný kurz se v dané učebně může učit vícekrát, v dané učebně v danou vyučovací hodinu se může učit nejvýše jeden kurz, kurz se v danou vyučovací hodinu může učit nanejvýše v jediné učebně.

Ternární typ vztahu lze obvykle těžko pojmenovat slovesným tvarem, zde by mohl být pojmenován „se učí v a v“ namísto „Rozvrh“.

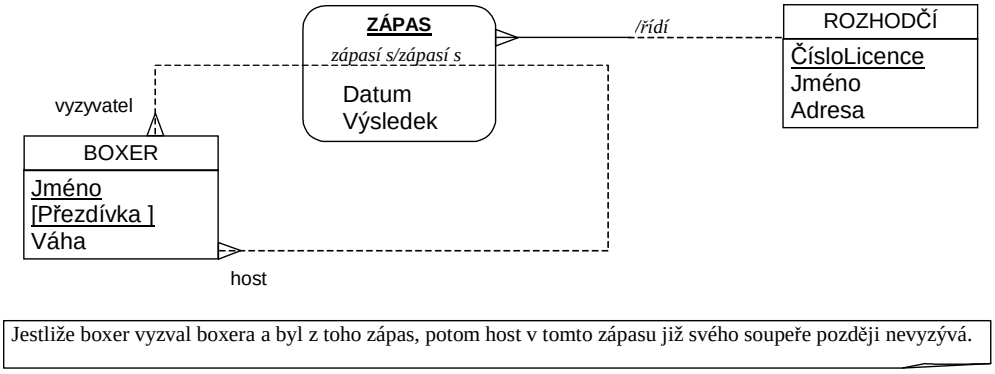
Alternativou k použití ternárního vztahu by bylo vytvořit vazební entitní typ „Rozvrh“ s jediným atributem „VyučovacíHodina“ a se dvěma alternativními klíči („VyučovacíHodina“, identifikační vztah k „Učebně“) a („VyučovacíHodina“, identifikační vztah ke „Kurzu“). Volba s ternárním vztahem je ale přirozenější.

Formy studia, formy ukončení a formy výuky mají domény dány vyjmenováním hodnot, podobně jako je tomu u typu učebny.

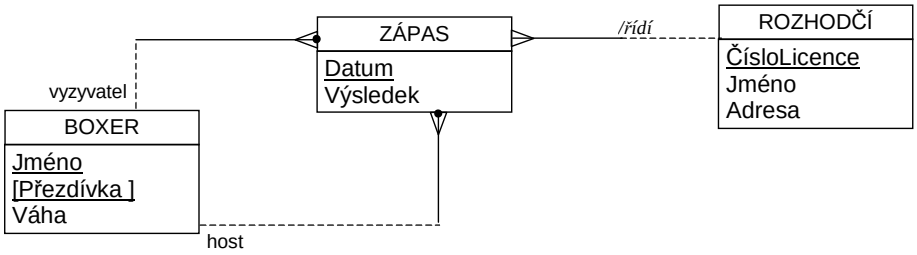
ISA vztahy u osob a jejich podtypů netvoří *členění*, student může být zároveň zaměstnancem, učitel může být zároveň administrativním pracovníkem. ISA vztahy u místností členění tvoří, místnost je buď kancelář nebo učebna, nikoli obojí. Značení se liší připojením šipek, členění má společné zakončení šipky.

Boxeři

Konceptuální schéma pro databázi o zápasech boxerů.

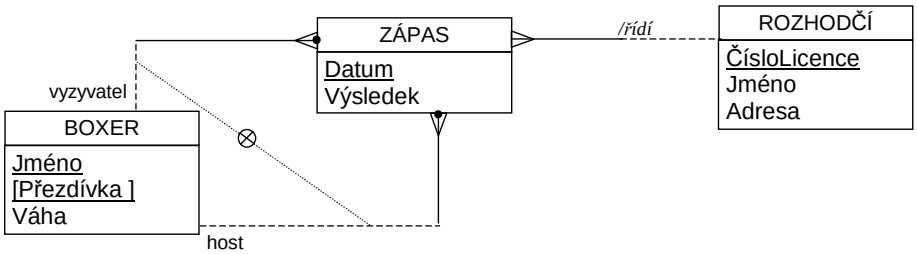


Tato varianta je založena na předpokladu, že dva boxeři spolu zápasí nejvýše jednou. Pokud by tomu tak nebylo, není předchozí model správný, a je nutno modelovat takto:



Zde zápas má identifikační závislost na boxeru-vyzyvateli i boxeru-hostu.

Mohli bychom přidat omezení, že boxer nezápasí sám se sebou (značení převzato z ORM):



V obou případech je výsledek zápasu atribut složený z výsledku vyzyvatele a výsledku hosta – Výsledek(Vyzyvatele,Hosta).

Identifikační schéma boxerů je třeba vysvětlit. Pokud má boxer přezdívku, je tato jedinečná a identifikuje ho. Pokud přezdívku nemá, identifikuje ho jméno. (Takovéto složité identifikační

schéma se v lidských systémech občas objevuje.) V našem případě je asi nejvhodnější vytvořit pro účely databáze umělý identifikační klíč pro boxery.

Následující dvě kapitoly pojednávají o tvorbě relačních databázových schémat, po nich jsou uvedena vzorová mapování našich příkladů do relačních databázových schémat.

KONCEPTUÁLNÍ SCHÉMA → RELAČNÍ SCHÉMA

Mapování konceptuálního modelu do relačního databázového schématu bývá víceméně automatizováno CASE nástroji, nicméně jde o ne zcela rutinní záležitost. V některých otázkách musí návrhář učinit vlastní rozhodnutí. Tato kapitola popisuje jistý postup, jímž se lze při mapování řídit, a ukazuje místa, v nichž je třeba se rozhodnout.[13]

Krok 0
Rozhodněte o všech odvozených atributech a vztazích, zda mají být ukládány. (Odvození se mapují do integritních omezení.)

V příkladu Knihovna budou atributy Vráceno? u výpůjčky a JeVolný u exempláře vždy ukládány a aktualizovány při změně hodnot, které tyto atributy definují. Atribut Věk u čtenáře ukládán nebude, bude vypočítáván při manipulacích, jež ho budou používat.
Odvozený vztah „Čtenář má platný Průkaz“ bude ukládán – mapován jako vztah 1:1. Jak, o tom viz dále.

Krok 1
Rozhodněte o mapování typů v ISA vztazích. Defaultní je

absorpce všech podtypů entit do jejich vrchního nadtypu (atributy podtypů budou tvořit volitelná pole)

Nebo zvažte jinou volbu z následujících dvou možností:

separace fakty specifické pro podtyp jsou mapovány do samostatné tabulky

rozklad každý podtyp bude tvořit samostatnou tabulku, nadtyp je tak rozdělen do několika tabulek – to je vhodné při úplném disjunktním členění nadtypu do podtypů

V příkladu Notrhwind mohou být typy Dodavatel a Zákazník mapovány do jediné tabulky SmluvníPartner, jež bude mít volitelné pole DomovskáStránka. Pak by ale bylo třeba vytvořit společné identifikační schéma pro dodavatele i zákazníky. Stálo by za zvážení i vytvoření samostatného pole určujícího, zda jde o zákazníka nebo dodavatele. Ve vzorové databázi byl namísto toho zvolen rozklad. Důvodem bylo nejspíš to, že společný pohled na dodavatele i zákazníky nebude téměř nikdy potřeba.
Separaci uvidíme při mapování příkladu IS soukromé VŠ.

Krok 2
U entitních typů rozhodněte o způsobu identifikace – o klíčích. Zvažte nahrazení libovolného složeného identifikačního schématu použitím umělého či zřetězeného klíče, či volbou default hodnot (a změnou nepovinný atribut → povinný).

U osob v příkladu IS soukromé VŠ přicházely do úvahy dva klíče, rodné číslo nebo uživatelské číslo UČO. Již v konceptuálním schématu bylo rozhodnuto, že bude používána druhá z možností.
U studijního plánu ve stejném příkladu připadá do úvahy složený klíč (oborStudia, semestr, formaStudia), toto však bude nahrazeno zřetězeným klíčem.

---

[13] Následující kapitola ukáže, že pro účely provozu IS může být relační schéma ještě následně přizpůsobeno.

Velmi zvláštní způsob identifikace boxerů bude nahrazen umělým klíčem.
Dalším příkladem je v praxi běžný způsob identifikace místností v organizaci, jež má více budov, založený na sémantickém klíči vzniklým zřetězením kódu budovy, v níž je místnost, čísla patra, a čísla místnosti na patře.
V biologické klasifikaci se může použít název druhu, a pokud je druh použit, ještě je pak volitelný název poddruhu. Takovéto identifikační schéma je založeno na volitelném atributu NázevPoddruhu. Namísto toho je možno zvolit default hodnotu pro NázevPoddruhu místo prázdné hodnoty, a identifikační klíč vytvořit zřetězením NázevDruhu+NázevPoddruhu.

Jak uvidíme v kapitole FYZICKÉ SCHÉMA – INDEXY, pro účely urychlení dotazů je vždy možno zvolit náhradu přirozeného identifikátoru umělým, jenž bude optimalizován co do délky.

Krok 3
Každý entitní typ mapujte do samostatné tabulky (kromě výjimek popsaných v kroku 1 a následujícím kroku 4), atomické atributy mapujte do jejích polí.

Krok 4
Rozhodněte o mapování 1:1 vztahů. Obecně se mapují do samostatné tabulky, rozhodování většinou ovlivní povinné/nepovinné členství ve vztahu. Pokud jeden z entitních typů má povinné členství, pak je možno tento vztah mapovat jako cizí klíč tabulky mapující tento entitní typ.

Cizí klíč v tabulce je pole obsahující hodnotu primárního klíče jiné tabulky.

Cizí klíč může být pojmenován podle role, již v daném vztahu hraje příslušná entita.

V příkladu Knihovna bude informace o exempláři, který splňuje objednávku, mapována jako cizí klíč v tabulce objednávek - bude jím pole obsahující ČísloExempláře.

Jestliže mají povinné členství oba z entitních typů, máme na výběr ještě další možnost, a to spojení obou tabulek do jedné. Rozhodnutí o této volbě bude ovlivněno také přenositelností vztahu.

Ve vzorové databázi mapující příklad Northwind byly entitní typy KontaktníOsoba a Dodavatel nebo Zákazník mapovány do spojené tabulky. Tato volba ovšem nedává volný prostor pro změnu vztahu „má/je u“ na 1:n.

Obecně jsou tedy na výběr čtyři možnosti:
- samostatná tabulka mapující 1:1 vztah, která má dvě pole, každé z nich s cizím klíčem tabulky mapující jeden z entitních typů v tomto vztahu
- cizí klíč v tabulce mapující první entitní typ
- cizí klíč v tabulce mapující druhý entitní typ
- spojení obou tabulek dohromady

V příkladu knihovna odvozený vztah „má platný/je platný průkaz“ má nepovinné členství obou entit. Bylo by možno ho mapovat do samostatné tabulky. Poněvadž ovšem naprostá většina čtenářů platný průkaz má, bude namísto toho mapován jako cizí klíč do tabulky čtenářů.

Krok 5
Mapujte každý vztah n:m a každý ternární nebo více-ární vztah do samostatné tabulky. Klíč takové tabulky bude složen z klíčů entitních typů vstupujících do tohoto vztahu. Případné atributy vztahu tvoří další pole.

Ve vzorové databázi Northwind je vztah „je objednán v/objednává“ mapován do tabulky Rozpis_objednávek, jež má pole ČísloObjednávky, ČísloVýrobku, Množství, JednotkováCena, Sleva.
V příkladu Diskusní klub bude vztah „je přijmut/přijímá“ mapován do samostatné tabulky s poli IDdopisu, Login.
V IS soukromé VŠ budou do samostatných tabulek mapovány vztahy „navštěvuje“ a „má uznán“, stejně jako ternární vztah „rozvrh“.
V příkladu Boxeři v první variantě bude vztah „zápasí s“ mapován do tabulky Zápas, jež bude mít pole Vyzyvatel, Host, Rozhodčí, Datum, VýsledekVyzyvatele, VýsledekHosta.

Krok 6
Vztahy 1:n mapujte do cizích klíčů. Všechny atributy vztahu jsou připojeny do tabulky mapující stranu n.

Pokud je ve vztahu 1:n nepovinné členství na straně n, je možno zvážit mapování do samostatné tabulky, podobně jako v předchozím kroku.

V příkladu Northwind jsou vztahy objednávky se zákazníkem, přepravcem a dodavatelem mapovány do polí tabulky Objednávky postupně KódZákazníka, Přeprava, ČísloZaměstanace. Atribut vztahu „/přepravuje“ je připojen k tabulce objednávek. Vztah s příjemcem je mapován jinak, důvody budou vysvětleny v následující kapitole.
Vztah „převzal/je předán“ v příkladu Knihovna se v první fázi mapuje do samostatné tabulky, ve druhé fázi jsou tabulky mapující objednávku a prezenční výpůjčku spojeny do jedné (naprostá většina objednávek vyústí v prezenční výpůjčku).
Vztah „je v“ mezi místností a budovou v příkladu IS soukromé VŠ je v první fázi mapován do cizího klíče v tabulce učeben a v tabulce kanceláří, následně je ale vyloučen, protože KódMístnosti je sémantický klíč složený z kódu budovy, patra a čísla na patře.

Všechny ostatní vztahy 1:n ve všech příkladech jsou mapovány do cizích klíčů.

Krok 7
Vícehodnotové atributy mapujte jako vztah k 1:n umělému entitnímu typu.

V příkladu Knihovna jsou atributy Autor a Kategorie mapovány do samostatných tabulek.

Krok 8
U složených atributů rozhodněte o rozdělení na jednotlivé složky, konkatenaci, nebo vytvoření umělého entitního typu.

Tributy Adresa ve všech příkladech budou rozloženy do svých složek.
Atribut Výsledek v příkladu Boxeři bude rozložen do svých složek.
Na všechny konkatenací vytvořené klíče ze složených klíčů se lze dívat také jako na složené atributy. V takovýchto případek lze zvážit mapování zřetězeného klíče do jednoho pole, a ještě (redundantně) mapování všech nebo některých složek do dalších samostatných polí.

Vytvoření umělého entitního typu v žádném z našich příkladů nebylo vhodné. Byl by jím např. typ Adresa, mapovaný do samostatné tabulky, pak adresy osob nebo firem by byly mapovány do pole obsahujícího cizí (a umělý) klíč entity adresa. Takováto volba by snad byla vhodná v databázi výzvědné služby, kde agenti jsou umísťováni na příhodné z dostupných tajných adres. – Ve všech takových případech však lze smysluplně entitu Adresa do modelu zahrnout.

Krok 9
Transformujte všechna integritní omezení a pravidla odvození. Použije se jazyk SŘBD i programovací jazyky 3.generace. Viz 3.ČÁST, kapitolu PŘEHLED OSTATNÍHO JAZYKA SQL, podkapitoly Integritní omezení a Triggery.

Rozmyslete

Rozvažte důsledky mapování *vícehodnotových atributů do pevně zvoleného počtu polí* stejného typu!

V příkladu Knihovna by v tabulce titulů mohly být vytvořeny např. 3 pole pro autory, z nichž 2 by u většiny knih byly prázdné.

Výhody – Při vyhledávání podle autorů není třeba dělat spojení tabulek.

Nevýhody – Při vyhledávání podle autorů je třeba prohledávat všechna tři pole. Při výskytu knihy, která má více než 3 autory, by vznikl vážný problém.

Rozvažte variantu *více záznamů pro jednu entitu s opakováním některých polí*. (Lépe bude vysvětleno v následující kapitole.)

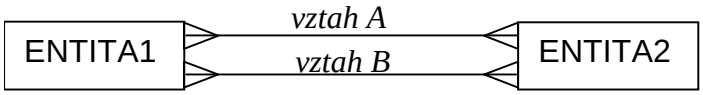
Analogické úvahy lze dělat i při mapování vztahu n:m.

Například v Northwind by namísto tabulky Rozpis_objednávek mohlo být do tabulky Objednávky přidáno 10krát trojice polí (Množství, JednotkováCena, Sleva). Problémy vzniklé v tomto případě by byly ještě výraznější.

Zvažte mapování nepovinných atributů do samostatné tabulky – jde o jakousi analogii separace. V takových případech proti sobě hovoří argumenty zbytečných prázdných hodnot v záznamech, a zbytečného provádění spojení tabulek v dotazech.

Jak uvidíme při mapování příkladu Boxeři, v případě srovnatelných rolí ve vztahu může připadat do úvahy ještě jiné mapování. Je to případ z následující obecné množiny příkladů:

V případě více typově stejných (kardinalita, event. členství) vztahů mezi stejnými entitními typy, kteréžto vztahy se liší pouze svým významem, lze zvážit jejich mapování do jediné tabulky s přidáním pole nesoucího informaci o vztahu, jenž je mapován. Schéma



může být mapováno do:	nebo do:
A(IDentity1, IDentity2) B(IDentity1, IDentity2)	C(IDentity1, IDentity2, TypVztahu : {‘A’, ‘B’})

Smysluplnost druhé volby je podmíněna *srovnatelností rolí* Entity1 v obou vztazích i srovnatelností rolí Entity2 v těchto vztazích. Příklady pro vztahy n:m jsou vzácné. Pro případ 1:n je to příklad Boxeři.

NORMALIZACE DATABÁZOVÉHO SCHÉMATU

Předchozí kapitola popisovala tvorbu databázového schématu „shora dolů“. V historii relačního databázového modelování byla detailně rozpracována ještě jedna metoda, kterou můžeme nazvat „zdola nahoru“. Spočívala ve sběru všech typů údajů jež mají být ukládány – polí zatím nespecifikovaných tabulek – a zjištění všech závislostí mezi nimi. Obecně šlo o závislosti symbolicky zapisované

$$\{A_i, A_j, \dots, A_k\} \rightarrow \{A_l, A_m, \dots, A_n\}$$

Tento zápis znamená, že na základě znalosti hodnot polí A_i, A_j, … ,A_k lze v modelovaném světě zájmu jednoznačně určit hodnoty polí A_l, A_m, … ,A_n. V běžném databázovém schématu platí taková závislost mezi klíči tabulek a ostatními hodnotami. Pokud je ve schématu ještě jiná netriviální[14] závislost než závislost na klíčích, obsahují tabulky redundance.

Například pokud budeme mít ve schématu tabulku, v níž budou záznamy o jednotlivých výrobkách, a tyto záznamy budou obsahovat i všechny údaje o dodavatelích výrobků, pak se budou údaje o dodavateli v databázi opakovat, pokud nějaký dodavatel dodává více výrobků:

Číslo dodavatele	Firma	Kontaktní osoba	…	Číslo výrobku	Název výrobku	…
1	Exotic Liquids	Charlotte Cooper		2	Chang	
2	New Orleans Cajun Delights	Shelley Burke		4	Chef Anton's Cajun Seasoning	
2	New Orleans Cajun Delights	Shelley Burke		5	Chef Anton's Gumbo Mix	
2	New Orleans Cajun Delights	Shelley Burke		65	Louisiana Fiery Hot Pepper Sauce	
2	New Orleans Cajun Delights	Shelley Burke		66	Louisiana Hot Spiced Okra	
3	Grandma Kelly's Homestead	Regina Murphy		6	Grandma's Boysenberry Spread	
3	Grandma Kelly's Homestead	Regina Murphy		7	Uncle Bob's Organic Dried Pears	
3	Grandma Kelly's Homestead	Regina Murphy		8	Northwoods Cranberry Sauce	
4	Tokyo Traders	Yoshi Nagase		9	Mishi Kobe Niku	
4	Tokyo Traders	Yoshi Nagase		10	Ikura	
4	Tokyo Traders	Yoshi Nagase		74	Longlife Tofu	
5	Cooperativa de Quesos 'Las Cabras'	Antonio del Valle Saavedra		11	Queso Cabrales	
5	Cooperativa de Quesos 'Las Cabras'	Antonio del Valle Saavedra		12	Queso Manchego La Pastora	
6	Mayumi's	Mayumi Ohno		13	Konbu	
6	Mayumi's	Mayumi Ohno		14	Tofu	
6	Mayumi's	Mayumi Ohno		15	Genen Shouyu	
…						

[14] Triviálními závislostmi myslíme takové, které se rozumí samo sebou bez ohledu na kontext, např. $A_5 \rightarrow A_5$ apod.

Klíčem této tabulky je ČísloVýrobku, ale nalezneme v ní závislosti na neklíčovém poli ČísloDodavele:

ČísloDodavatele → Firma , ČísloDodavatele → KontaktníOsoba , ….

Opakování údajů o dodavatelích je principielním důsledkem existence těchto závislostí.

Kromě zbytečně použitého místa má tato tabulka vady, jimž říkáme *aktualizační anomálie*. Aktualizační anomálie spočívají ve dvou jevech. První z nich se týká případů, kdy chceme změnit nějaké údaje o některém dodavateli, například kontaktní osobu. V takovém případě je nutno tyto údaje změnit v záznamech všech výrobků tohoto dodavatele. Druhá vada se týká případů, kdy v IS případně mažeme záznamy o výrobkách, když už tyto výrobky neprodáváme. Pokud bychom takto smazali záznamy o všech výrobkách některého dodavatele, ztratíme tím veškeré údaje o tomto dodavateli.

Ještě další závada takto navržené tabulky vyplývá z toho, že záznamy jsou zbytečně dlouhé, a případné sekvenční prohledávání (tzv. skenování) tabulky bude trvat příliš dlouho, a v případě vyhledávání údajů o dodavatelích je zase počet záznamů, které je třeba prohledat, zbytečně vysoký.

To vše se dalo předvídat z pouhého faktu existence netriviální funkční závislosti, která není závislostí na klíči tabulky. Odstranění závad pak spočívá v rozdělení tabulky do více tabulek, v našem případě do tabulek Výrobky a Dodavatelé, tak jak je to v naší vzorové databázi.

Metoda, která upravuje schéma databáze tak, aby v něm nebyly nežádoucí závislosti, se nazývá *normalizace databáze*. Existují i formálně popsané stupně míry normalizace, nazývané *normální formy*. Někteří zadavatelé projektů IS výslovně požadují, aby navržené tabulky databáze byly v určité normální formě. Proto zde normální formy stručně popišme:

Normální formy relačního schématu

1.normální forma

Tato normální forma se vymyká předloze ostatních normálních forem, nejde v ní o odstraňování nežádoucích závislostí mezi poli. 1.normální forma je pouze výslovným zopakováním požadavku relačního databázového modelu, že atributy jsou atomické. Při normalizaci databáze se v tomto kroku myslí hlavně to, aby pole tabulek nebyla sémanticky dělitelná, jako např. textový řetězec „W. Churchilla 4, Praha 3“, který sice můžeme uložit do jediného pole tabulky, ale nejspíš tím zanedbáváme to, že později budeme vyhledávat jeho jednotlivé významové složky. Po úpravě do první normální formy bude tabulka obsahovat pole pro ulici a číslo domu a pole pro obec (čísla domů asi nikdo samostatně vyhledávat nebude).

2.normální forma

Žádný neklíčový atribut není funkčně závislý na pouhé části klíče. Tím se myslí, že v případě složeného klíče tabulky nejsou některá z ostatních polí tabulky funkčně závislá pouze na některé složce tohoto složeného klíče. Například kdybychom v tabulce Rozpis_objednávek, která má složený klíč (ČísloObjednávky,ČísloVýrobku), měli ještě pole nesoucí údaje o dodavateli výrobku a kategorii výrobku, byla by tato pole závislá na poli ČísloVýrobku, a na poli ČísloObjednávky by nezávisela. Tedy symbolicky klíč tabulky je (ČísloObjednávky,ČísloVýrobku), ale platí ČísloVýrobku → {ČísloDodavatele , ČísloKategorie}.

Číslo objednávky	Číslo výrobku	ČísloDodavatele	ČísloKategorie	...
10248	72	20	5	
10248	42	14	4	
10248	11	5	4	
10249	51	6	7	
10709	51	24	7	

Šipky na obrázku znázorňují funkční závislosti, pole tvořící klíč jsou zvýrazněna. Tabulka na tomto obrázku není v 2.NF.

Normalizace v tomto kroku znamená v našem konkrétním případě oddělení polí ČísloDodavatele a ČísloKategorie do jiné tabulky, v níž bude klíčem pole ČísloVýrobku (takovou tabulkou je ve vzorové databázi tabulka Výrobky).

3.normální forma

Není žádná funkční závislost mimo klíč. Tím se myslí, že v tabulce není funkční závislost nějakých polí na jiných polích, jež nejsou součástí žádného klíče. Například kdybychom do tabulky objednávek přidali pole se jménem zákaznické firmy a se jménem kontaktní osoby v zákaznické firmě, byla by tato pole funkčně závislá na poli KódZákazníka, jež není součástí klíče (klíčem tabulky je ČísloObjednávky). Tabulka na obrázku není v 3.NF:

ČísloObjednávky	KódZákazníka	Firma	KontaktníOsoba	...
10248	WOLZA	Wolski Zajazd	Zbyszek Piestrzeniewicz	
10249	TOMSP	Toms Spezialitäten	Karin Josephs	
10250	HANAR	Hanari Carnes	Mario Pontes	
10251	VICTE	Victuailles en stock	Mary Saveley	
10334	VICTE	Victuailles en stock	Mary Saveley	
10252	SUPRD	Suprēmes délices	Pascale Cartrain	
10458	SUPRD	Suprēmes délices	Pascale Cartrain	
11038	SUPRD	Suprēmes délices	Pascale Cartrain	

Normalizace v tomto kroku znamená opět rozdělní této hypotetické tabulky, a oddělením polí Firma a KontaktníOsoba do nějaké tabulky, jejímž klíčem je KódZákazníka (ve vzorové databázi je jí tabulka Zákazníci).

Boyce-Coddova normální forma

V tabulkách není žádná netriviální funkční závislost. Tím se myslí, že jediné netriviální funkční závislosti v tabulkách jsou závislosti na celém klíči. Například pokud bychom měli tabulku ADRESÁŘ s poli Město, Ulice, PSČ , jež má dva alternativní klíče (Město, Ulice) a (Ulice, PSČ), pak by nebyla ve BCNF, protože v ní platí závislost PSČ → Město. PSČ je ale pouhou částí klíče, tabulku by bylo možno rozdělit do tabulek s poli (PSČ, Město) a (Ulice, PSČ) (klíče těchto tabulek jsou podtrženy). Ale pravděpodobně se vám tato možnost rozdělení ADRESÁŘe do dvou tabulek bude zdát nevhodné, protože při používání databáze bychom tyto dvě tabulky vždy spojovali. Takovéto rozdělení ADRESÁŘe se zdá být zhoršením.

Nejčastěji požadovanou normální formou je 3. normální forma.

4.normální forma

Tato normální forma se týká odstranění přebytečných tzv. multizávislostí. Vysvětleme si tento pojem na příkladu Knihovna. V něm mají Tituly dva vícehodnotové atributy Autor a Kategorie. Představme si, že bychom vytvořili tabulku s poli (Signatura, Autor, Kategorie), jež by obsahovala pro každý titul všechny možné kombinace autorů a kategorií tohoto titulu. Tedy např. pro titul „Acorna the Unicorn Girl“ se signaturou „AA 38014“, s autorkami „MAC CAFFREY, Anne“ a „BALL, Margaret“ a kategoriemi „sci-fi“, „pro mládež“, „anglicky“ by tabulka obsahovala řádky

AA 38014	MAC CAFFREY, Anne	sci-fi
AA 38014	MAC CAFFREY, Anne	pro mládež
AA 38014	MAC CAFFREY, Anne	anglicky
AA 38014	BALL, Margaret	sci-fi
AA 38014	BALL, Margaret	pro mládež
AA 38014	BALL, Margaret	anglicky

Chápeme, že tato tabulka obsahuje redundance. Lépe je vytvořit tabulky

AA 38014	MAC CAFFREY, Anne
AA 38014	BALL, Margaret
…	

AA 38014	sci-fi
AA 38014	pro mládež
AA 38014	anglicky
…	

Takto to také radí metodika mapování vícehodnotových atributů z konceptuálního modelu do relačního schématu.

Ve zmíněné hypotetické tabulce multizávisí pole Autor na poli Signatura a také pole Kategorie mutlizávisí na poli Signatura. Tomu nejlépe porozumíme tak, že ve skutečnosti jsou Autor a Kategorie vícehodnotové atributy jednoznačně závislé na identifikujícím atributu Signatura.

Další normální formy…

Je možno definovat ještě další normální formy, o nich se ale v těchto skriptech nebudeme zmiňovat. Pro začínající analytiky je tento přehled dostačující.

„Postupem shora dolů vytvoříme normalizované tabulky.“

Toto je široce sdílená *pověra*, jež je pravdou jedině tehdy, když je konceptuální schéma skutečně dobře navrženo. Kvalita konceptuálního návrhu závisí do vysoké míry na zkušenosti analytika a na jeho schopnosti získat informace o podstatě modelovaného světa zájmu. Uveďme jeden příklad běžné pasti, jež na analytiky čeká:

Dvoutvářné entity

Dvojici entitních typů „Titul“ a „Exemplář“ v příkladu Knihovna bychom bez znalosti problematiky knihoven mohli nahradit entitním typem „Kniha“, jenž by měl atributy (ČárovýKód, Signatura, Autor+, Název, Kategorie+, Typ, VěkováHranice, Vydavatel, RokVydání, Umístění, Absenční?, JeVolný?). Mapováním do relačního schématu bychom tak získali nenormalizovanou databázi, nebyla by v 3.NF.

„Kniha“ je příklad tzv. *dvoutvářné entity*, jejíž dvě tváře odhalíme až po zjištění:

- Rozdílných cílů lidí používajících IS co se týče tohoto typu entity

  Čtenář vyhledává knihu podle autora a názvu. Knihovník chce vědět, kde kniha je, či je-li volná.

- Nesmiřitelných rozdílů v popisech či objemech

  Čtenář na otázku, kolik je knih daného autora s daným názvem, řekne, že pouze jediná! Knihovník na stejnou otázku odpoví, že počet se různí podle toho, kolik jich máme.

Návrh zdola nahoru

Postupem zdola nahoru bychom předchozí chybu neudělali, protože ČárovýKód → Signatura, Signatura → {Název, Typ, VěkováHranice, Vydavatel, RokVydání}, Autor a Kategorie multizávisí na Signatura…

Jenže

postup zdola nahoru je založen na zjištění všech závislostí a multizávislostí, jež v analyzovaném světě zájmu platí mezi navrženými poli. To je neúnosné pro praktickou realizaci, protože potenciální množství závislostí, na něž je třeba se zadavatelů zeptat, je ohromné. Například pokud by databáze měla obsahovat pouze 10 polí, možných závislostí je 59049. Pokud bychom uvažovali pouze závislosti na jednom či dvojici polí (klíče jsou většinou nanejvýš dvousložkové), je možností 405. Bez doplňujících informací bychom tedy museli dát uživatelům 405 otázek, jež by je obtěžovaly svým velkým množstvím a jednotvárností, a povětšinou svou hloupostí, neboť byste to ve většině případů „přeci měli vědět“. (Počet otázek bychom sice mohli zmenšit vhodnou strategií vyptávání s využitím předchozích odpovědí, ale to je náročné, protože nesmíme dlouho zdržovat vlastním přemýšlením.) Jistě je možno vytipovat pole, jež jsou pravděpodobně klíčová, když se zeptáme na příklady hodnot v polích a zjistíme, že hodnotami jsou jakési kódy. Ne vždy je však klíčové pole kódovaným identifikátorem, lidé používají i přirozené sémantické identifikátory.

Dalším nedostatkem metody zdola nahoru jsou nicneříkající jména tabulek, jež takto vzniknou. Přitom velmi pravděpodobně jsou to odrazy nějakých entitních typů běžně pojmenovávaných v dané oblasti zájmu.

Doporučený postup

Doporučený postup je tento: podle svých nejlepších schopností vytvořte konceptuální model pro daný svět zájmu, a tento mapujte podle návodu v kapitole KONCEPTUÁLNÍ SCHÉMA → RELAČNÍ SCHÉMA do tabulek. Protože však nejste dokonalí, zkontrolujte výsledné databázové schéma, zda v něm nejsou nežádoucí závislosti. Počet případných závislostí je takto podstatně nižší než při postupu zdola nahoru.

Velkou pomocí je porozumění problematice. Proto jsou také v praxi často využíváni lidé, kteří znají věcnou problematiku, a zároveň znají nebo alespoň chápou techniky modelování.

Proto je také velkou pomocí zkušenost analytika, jenž ví, jak si lidé své záležitosti běžně organizují a jaké mají zvyky v pojmenovávání věcí či vztahů.

Denormalizace

Normalizace databáze vyústí v návrh množství tabulek, jež budeme následně při provozu databáze spojovat na základě požadavků na různé výstupy systému. Spojení – join – tabulek je časově a zdrojově náročná operace. Můžeme zvažovat, zda výhody normalizace nejsou převáženy zatížením databázového stroje. Pak se cílevědomě můžeme rozhodnout, že některé tabulky spojíme do jedné, protože dotazy požadující jejich spojení jsou tak časté, že je to pro provoz databáze významné. (Pokud bychom pro nějaké sporadické účely potřebovali i pohledy na některou z původních tabulek, můžeme je definovat jako VIEW.) Takovéto úpravě návrhu databázového schématu říkáme denormalizace.

Denormalizace vychází ze zvažování provozních záležitostí databáze. Proti denormalizaci hovoří vše co bylo v této kapitole uvedeno v úvodu, tedy

- zbytečně zabrané místo redundantními daty
- aktualizační anomálie při aktualizaci záznamů a mazání záznamů
- skenování tabulek s dlouhými záznamy a velkým počtem záznamů trvá dlouho

Pro denormalizaci hovoří často prováděná spojení tabulek.

Příklady

Ve všech příkladech je zanedbána závislost PSČ → Město. V přísném smyslu jsou tedy všechny tabulky s adresami denormalizované. Jak jsou nakonec databázová schémata pro naše příklady vytvořena viz následující kapitolu[15].

Northwind

Podle konceptuálního schématu pro vzorovou databázi Northwind bychom vytvořili tabulku Příjemce s umělým klíčem např. IDpříjemce a v tabulce Objednávky by bylo pole s cizím klíčem obsahujícím IDpříjemce příjemce této objednávky. Toto pole by bylo prázdné v těch případech, kdy příjemcem je firma zákazníka. Místo toho je provedena denormalizace levým polospojením těchto dvou tabulek, a do prázdných polí takto vzniklých v řádcích, ve kterých příjemcem je firma zákazníka, jsou zopakovány hodnoty z polí Firma a složek adresy zákazníka. IDpříjemce se takto stává zbytečným, a je vypuštěn.

IS soukromé VŠ

V příkladu IS soukromé VŠ je vytvořen umělý klíč pro studijní plán KódStudPlánu, a je provedena denormalizace spojením tabulek Student a Studijní_plán, a podruhé ještě spojením tabulky pro vztah „zahrnuje“ a tabulky Studijní_plán. (Studijní_plán jakoby se před denormalizací zdvojil.)

---

[15] Podoby tabulek vznikly v diskusích na cvičeních z předmětu Úvod do databází na VŠFS v zimním semestru 2002/2003.

Toto je jeden z možných pohledů na to, jak jsou tabulky pro tento příklad nakonec navrženy. Jiný pohled je ten, že je vytvořen složený sémantický klíč KódStudPlánu, a do tabulek Student a Zahrnuje jsou přidána z tohoto sémantického klíče odvozená pole OborStudia, Semestr, FormaStudia. Číselník[16] studijních plánů není ve schématu zahrnut, protože je odvoditelný z možných hodnot pro obor, semestr a formu studia.

[16] Číselník je tabulka kódů a jejich vysvětlení.

PŘÍKLADY RELAČNÍCH DATABÁZOVÝCH SCHÉMAT

Ve všech příkladech jsou až na výjimky vynechány specifikace domén. Domény s vyjmenovanými hodnotami jsou uvedeny ve složených závorkách. Primární klíče tabulek jsou podtrženy. Alternativní klíč je značen UNIQUE . Šipky ⟶ znázorňují referenční integritu, o ní viz 3.ČÁST, kapitolu PŘEHLED OSTATNÍHO JAZYKA SQL, podkapitolu Integritní omezení. Některá další integritní omezení z konceptuálních modelů jsou diskutována také ve zmíněné kapitole.

Knihovna

Titul(ČísloTitulu, Název, Typ:{‘kniha‘,‘časopis‘}, Vydavatel, RokVydání, VěkováHranice:{0,15,18}, [Anotace], [ČísloVydání])

Má(ČísloTitulu, ČísloAutora)

PatříDo(ČísloTitulu, Kategorie)

Exemplář(Signatura, ČísloTitulu, JeAbsenční:Ano/Ne, Umístění, JeVolný:Ano/Ne)

Průkaz(ČísloPrůkazu,ČísloČtenáře, PlatnostDo, JeZablokován:Ano/Ne)

Čtenář(ČísloČtenáře, ČísloPrůkazu UNIQUE, PříjmeníAJméno, UliceAČíslo, Město, PSČ, DatumNarozením, Dluh)

Objednávka(ČísloČtenáře, ČísloTitulu, DatumObjednání, [Signatura], Převzato:Ano/Ne)

Kategorie(ČísloKategorie, NázevKategorie UNIQUE)

Autor(ČísloAutora, PříjmeníAJméno)

Výpůjčka(Signatura, ČísloČtenáře, DatumVypůjčení : timestamp, VrátitDo, [DatumVrácení])

Číselník Kategorie a Autor jsou přidány kvůli efektivnosti, důvody budou objasněny v kapitole FYZICKÉ SCHÉMA – INDEXY. V tabulce Autor by mohla být ještě další pole s informacemi o autorovi.

Objednávka je vyřízena, když je do jejího pole Signatura vložena signatura exempláře, jenž objednávku splňuje.

Věk čtenáře se vypočítává pouze při interakcích čtenáře s IS.

Diskusní klub

Osoba (Login, CeléJméno, Heslo, Email, PosledníNávštěva)

Klub(IDklubu , Administrátor, NázevKlubu, Popis:memo, URL)

Dopis(IDdopisu , Odesílatel, [Subjekt] , TextDopisu:memo, DatumOdeslání:timestamp)

Příspěvek(IDpříspěvku, TextPříspěvku:memo , KdyVložen:timestamp , PočetReakcí, [ReakcíNa])

Příjemci(IDdpoisu, Login)

Počet reakcí příspěvku by mohl být aktualizován triggerem, o triggerech viz kap. PŘEHLED OSTATNÍHO JAZYKA SQL, Triggery .

IS soukromé VŠ

Osoba(UČO, RČ, Jméno, e-mail, UliceAČíslo, Město, PSČ, [Foto]:obrázek)

Student(UČO, Dluh:měna, KódStudPlánu, KódOboru, Semestr, KódFormy)

Učitel(UČO, KódKanceláře, [ČísloTel])

Admin(UČO, KódKanceláře, [ČísloTel])

Obor(KódOboru, NázevOboru)

FormyStudia(KódFormy, NázevFormy)

Budova(KódBudovy, NázevBudovy, TelÚstředna, UliceAČíslo, Město, PSČ)

Kancelář(KódKanceláře, KódBudovy, Účel)

Telefon(ČísloTel, KódKanceláře)

Předmět(KódPředmětu, NázevPředmětu, HodinPřednášek, HodinCvičení, FormaZakončení:{‘z‘,‘zk‘,‘kz‘,‘szk‘})

Zahrnuje(KódPředmětu, KódStudPlánu, KódOboru, Semestr, KódFormy)

Kurz(KódKurzu, KódPředmětu, FormaVýuky:{‘p’,’c’}, Učitel)

MáUznán(UČO, KódPředmětu)

Navštěvuje(UČO, KódKurzu)

Rozvrh(KódKurzu, KódUčebny, VyučovacíHodina)

Učebna(KódUčebny, Kapacita, Typ)

Při tomto mapování byla zvolena separace typů Osoba, Student, Učitel, Administrativní_pracovník. Typ učebny bude mít vyjmenované hodnoty.

Boxeři

První varianta:

Boxer(IDboxera, CeléJméno, [Přezdívka], Váha)

Rozhodčí(ČísloLicence, PříjmeníAJméno, UliceAČíslo, Město, PSČ)

Zápas(Vyzyvatel, Host, Datum, VýsledekVyzyvatele, VýsledekHosta, Rozhodčí)

Zde bylo opomenuto integritní omezení týkající se antisymetrie a antireflexivity vztahu „zápasí s“. To lze napravit takto (Boxer a Rozhodčí zůstávají beze změny):

Zápas(IDboxera1, IDBoxera2, Vyzyvatel : {1,2}, Výsledek1, Výsledek2, Datum, Rozhodčí)

s omezením IDboxera1<IDBoxera2.

Ve druhé variantě, ve které spolu boxeři mohou zápasit opakovaně, je změna pouze v klíči:

Zápas(IDboxera1, IDBoxera2, Datum, Vyzyvatel : {1,2}, Výsledek1, Výsledek2, Rozhodčí))

Obě varianty lze ovšem mapovat ještě takto:

Zápas(IDzápasu, Datum, Rozhodčí)

VýsledekZápasu(IDzápasu, IDboxera, Vyzyvatel : Ano/Ne, Výsledek)

Výhoda takovéhoto mapování je ve snadných výpočtech statistik výkonů boxerů. Aby relační schéma opravdu odpovídalo konceptuálnímu modelu, je nutno ještě přidat integritní omezení o tom, že do tabulky VýsledekZápasu musí být zapsány výsledky obou soupeřů,
a UNIQUE pro dvojici (IDzápasu, Vyzyvatel).

FYZICKÉ SCHÉMA – INDEXY

Data jsou ukládána na médiích s přímím přístupem (disky) nebo na mediích se sekvenčním přístupem (pásky). Pásky obvykle slouží k zálohování databáze. Data jsou zpracovávána v operační paměti. Jednotkou přenosu mezi vnější pamětí a operační pamětí je blok. Bloky jsou operativně ukládány ve vyrovnávací paměti – bufferu, SŘBD má vyhrazeno několik stránek v operační paměti pro kopie dat z vnější paměti, s nimiž probíhá zpracování. Správu přenosu mezi vnější pamětí a buffery v běžných aplikacích dělá operační systém, v databázovém zpracování však *modul řízení vyrovnávacích pamětí* a *modul řízení souborů* jsou součástí SŘBD. Databáze tedy v tomto směru nevyužívá služeb operačního systému, používá vlastní algoritmy.

Modul řízení souborů využívá tzv. *schéma organizace souboru*, jež sestává z volby způsobu uložení dat, způsobu přístupu k datům a způsobu vyhledávání dat, kteréžto jsou vzájemně podmíněny. O různých druzích schémat organizace souborů se lze dočíst v [Pok97], částečně také v [GCCC] a v [TLM]. Na základě definice schématu databáze v jazyce definice dat si SŘBD vytváří schéma organizace souboru sám, moderní silné SŘBD mohou toto schéma upravovat na základě provozních charakteristik databáze. Databázový administrátor, neboli správce databáze, může schéma organizace souboru ovlivňovat pouze velmi omezeně. Typická možnost administrátorovy volby je vymezení adresního prostoru pro databázi a zadání příkazů pro vytvoření indexů.

Indexy

Index je podpůrná struktura k zefektivnění vyhledávání záznamů datového souboru, tzv. *primárního souboru*, podle nějakého atributu nebo kombinace atributů. Tento atribut nebo tuto kombinaci atributů nazýváme *indexní klíč*. Nejdostupnější možností, jak ovlivnit fyzické schéma databáze, je volba indexních klíčů a definice indexů podle nich. Nejčastěji používaná logická struktura indexů v SŘBD je struktura tzv. B-stromů.

B-stromy

B-stromy jsou logickou strukturou stromů, v nichž *uzly jsou jednotlivé fyzické stránky* uložené (jako souvislé celky) na vnějších paměťových médiích. Vnitřní struktura stránek tvořících uzly se liší u kořene nebo vnitřních uzlů a u listů stromu.

Všechny uzly B-stromu obsahují sekvenci indexních záznamů. Indexní záznamy sestávají z konkrétní hodnoty indexního klíče a ukazatele. Ukazatelé v listech ukazují na stránku primárního souboru, kde se nalézá záznam s příslušnou hodnotou indexního klíče. Vnitřní uzly a kořen stromu jsou řídící uzly B-stromu, ukazatelé v nich ukazují na potomka řídícího uzlu, v němž má prohledávání pokračovat (tj. má se načíst příslušná stránka indexu). Sekvence indexních záznamů v každém uzlu B-stromu je uspořádaná podle hodnot indexního klíče.

Při aktualizacích primárního souboru se *dynamicky mění struktura B-stromu*. Při přidávání nových záznamů do primárního souboru se přidávají indexní záznamy do uzlů B-stromu. Pokud je překročena kapacita stránky uzlu, jsou vytvářeny uzly nové, do nichž se přesunou některé indexní záznamy z jiných uzlů. Jindy dochází ke slévání uzlů, aby byl uvolněn nevyužitý prostor. Toto vše se děje vždy tak, že cesty od kořene k listům B-stromu jsou všechny stejně dlouhé, a stránky v uzlech jsou využity alespoň do poloviny.

Poučení pro správu databáze je následující: délka indexního klíče určuje, kolik indexních záznamů se vejde do jedné stránky indexu, a to zase ovlivňuje nezbytný počet úrovní B-stromu, což je nutný

počet přístupů na disk při prohledávání B-stromu. Samozřejmě se tím ovlivňuje také celkový počet potřebných fyzických stránek pro index.

Při aktualizacích primárního souboru, při kterých se mění hodnoty indexních klíčů, je potřeba aktualizovat i index, což znamená zdržení. Při vytváření indexu musíme tedy zvážit to, zda zisky plynoucí z urychlení vyhledávání nejsou převáženy ztrátami plynoucími ze zpomalení aktualizací.

Údržba indexů znamená **režijní náklady**.

Hašované indexy

Některé SŘBD alternativně využívají schéma organizace primárního souboru nebo indexu založené na tzv. *hašování* (z anglického hash). To spočívá v tom, že záznam je umístěn do fyzické stránky, jejíž adresa se vypočítá z hodnoty indexního klíče. Přitom je vyhodnocováno využití cílových stránek a eventuelně je upravován algoritmus výpočtu haše. Toto schéma organizace souboru se vyznačuje velmi rychlým vyhledáváním za cenu vyšších nákladů na potřebný adresní prostor pro databázi.

Další možnosti organizace souborů…

Pokud jsou záznamy v primárním souboru sekvenčně uspořádány podle klíče, je vyhledávání rychlejší.

V části věnované síťovým databázím jsme viděli, že organizace datového souboru může být založena také na zřetězení pomocí směrníků. Různá schémata organizace mohou být vzájemně kombinována. Některé pomocné struktury pro vyhledávání mohou být trvale přítomny přímo v operační paměti.

Vyhledávání podle pole, jež není indexováno

V takovém případě je časová i paměťová náročnost prohledávání úměrná délce a počtu záznamů.

Délka záznamů ovlivňuje počet nutných přístupů na disk při skenování tabulky!

Další ovlivnění fyzických charakteristik

Data, podle nichž nebudeme vyhledávat, je možno komprimovat. Volba kódování ovlivňuje uspořádání při prohledávání. Proměnná délka polí většinou negativně ovlivňuje prohledávání.

Co indexovat?

Primární klíče a cizí klíče v tabulkách jsou běžně indexovány. Vytipování dalších polí, podle nichž se často prohledává, může vyústit v tvorbu dalších indexů.

Některé **dotazy** lze realizovat přímo na indexech.

(Další otázky související s vyhodnocováním dotazů jsou předmětem kapitoly OPTIMALIZACE DOTAZŮ.)

Příklad 5.
V příkladu Knihovna byly zvoleny samostatné číselníky autorů a kategorií kvůli délce indexů pro cizí klíč v tabulkách Má a PatříDo.

3. ČÁST -
JAZYK SQL

DOTAZOVACÍ JAZYK SQL, jednoduché dotazy

SQL je mezinárodní standard jazyka přístupu k datům poskytovaného relačními SŘBD. V této a následujících kapitolách je podán výklad jeho nejznámější a nejpoužívanější části, dotazovacího jazyka[17]. Dotazovací jazyk je jazyk, jímž lze vznášet požadavky na poskytnutí hodnot dat uložených v databázi. Dotazovací jazyk v relačních databázích umožňuje formulovat jen takové dotazy, na něž odpovědí je nějaká tabulka hodnot – SŘBD vytvoří tabulku odpovědi a tuto pošle dotazujícímu.

Syntaxe *jednoduchých dotazů* v SQL vypadá následovně (klíčová slova SQL jsou v těchto skriptech psána tučně):

SELECT (seznam polí) **FROM** (seznam tabulek) **WHERE** (podmínky)

nebo

SELECT (seznam polí) **FROM** (seznam tabulek) **WHERE** (podmínky)
UNION
SELECT …

kde (seznam polí) je čárkami oddělený seznam polí, jež má obsahovat tabulka odpovědi, (seznam tabulek) je seznam tabulek databázového schématu, z nichž má být tabulka odpovědi sestavena, (podmínky) udávají podmínky výběru záznamů, které mají být vybrány k vytvoření odpovědi. Verze standardu SQL 99 má samostatnou klauzuli pro spojení:

SELECT (seznam polí) **FROM** (spojení tabulek) **WHERE** (podmínky) …

Následují příklady dotazů nad vzorovou databází ze str. 18, vždy též „přeložené“ do relační algebry – jazyka matematických relací. Z něj zde použijeme pouze *projekci*, *restrikci*, a *spojení* neboli *join*, zapisované následovně

projekce	R [A_i, A_j, … ,A_k]	A_i, A_j, … ,A_k jsou pole, do kterých se má projekce provést
restrikce neboli selekce	R (podmínka)	
join neboli spojení	R (podmínka spojení) S	

Operace join je analogií skládání matematických relací, spočívající v tom, že spojujeme řádky tabulky R se řádky tabulky S podle „podmínky spojení“. Příklady, jež následují, význam jednotlivých operací dostatečně objasní. V příkladech se používá schéma ze str. 18.

U každého příkladu je ještě uvedena podoba v poslední verzi standardu – SQL 99. Z konstrukcí SQL 99 jsou použity pouze ty nejběžnější, protože účelem této kapitoly je do jazyka SQL čtenáře pouze uvést.

Příklad 6.
Vypiš jméno firmy a jméno kontaktní osoby každého zákazníka!

SELECT Firma, KontaktníOsoba
FROM Zákazníci

} dotaz SQL, verze SQL99 se v tomto případě neliší

[17] Měně známý je též poměrně rozšířený dotazovací jazyk QBE (Query By Example).

Zákazníci [Firma, KontaktníOsoba] } výraz relační algebry, zde je použita pouze projekce

Seznam za klíčovým slovem SELECT v SQL znamená výběr sloupců výsledku. Výsledná tabulka je:

Firma	Kontaktní osoba
Frankenversand	Peter Franken
France restauration	Carine Schmitt
Franchi S.p.A.	Paolo Accorti
Furia Bacalhau e Frutos do Mar	Lino Rodriguez
Galería del gastrónomo	Eduardo Saavedra
…	…

všichni zákazníci

Příklad 7.
Vypiš všechny objednávky, ve kterých je částka za dopravné větší než 20 000,- !

SELECT * **FROM** Objednávky **WHERE** Dopravné>20000

Symbol * za slovem SELECT znamená, že se mají vybrat všechna pole. Výraz relační algebry:

Objednávky (Dopravné>20000)

Zde je použita restrikce. Znamená výběr řádků – výsledná tabulka:

pouze některé objednávky

Číslo objednávky	Zákazník	Zaměstnanec	Datum objednávky	…	Dopravné	…
10372	Queen Cozinha	Buchanan, Steven	04.12.1996	…	22 269,50	…
10540	QUICK-Stop	Leverling, Janet	19.05.1997	…	25 191,00	…
10691	QUICK-Stop	Fuller, Andrew	03.10.1997	…	20 251,25	…
11030	Save-a-lot Markets	King, Robert	17.04.1998	…	20 768,75	…

všechny sloupce z tabulky Objednávky

Verze SQL99 se ani v tomto případě neliší.

Příklad 8.
Vypiš vše o objednávkách a zákaznících, kteří je podali! (Objednávku podal zákazník, jehož KódZákazníka je shodný s hodnotou KódZákazníka v objednávce.)

SELECT * **FROM** Objednávky, Zákazníci
WHERE Objednávky. KódZákazníka=Zákazníci. KódZákazníka

Verze v SQL99:
SELECT * **FROM** Objednávky **INNER JOIN** Zákazníci **ON**
(Objednávky. KódZákazníka=Zákazníci. KódZákazníka)

Výraz relační algebry, je použito spojení neboli join:

Zákazníci(Zákazníci.KódZákazníka=Objednávky.KódZákazníka)Objednávky

Vysvětlíme operaci spojení neboli join. V relační algebře je používána operace *kartézský součin*, jež znamená vytvoření relace spojováním řádků relací systémem každý s každým:

Kartézský součin – běžně se nepotřebuje:

Zákazníci × Objednávky … má 11+14 sloupců, řádky jsou tvořeny systémem každý s každým, tedy v jednom výsledném řádku je spolu i to, co „normálně“ k sobě nepatří

Pak spojení neboli join z příkladu 3 je totéž, jako provedení kartézského součinu a následné restrikce:

(Zákazníci × Objednávky)(Zákazníci.KódZákazníka=Objednávky.KódZákazníka)

Příklad 9.
Vyberte objednávky zákazníka „Alfreds Futterkiste“ !

```
SELECT Objednávky.*
FROM Objednávky,Zákazníci
WHERE Zákazníci.KódZákazníka=Objednávky.KódZákazníka
AND Zákazníci.Firma=“Alfreds Futterkiste“
```

nebo

```
SELECT A.*
FROM Objednávky  A , Zákazníci B
WHERE A.KódZákazníka=B.KódZákazníka
AND B.Firma=“Alfreds Futterkiste“
```

V SQL je možno pro zjednodušení zápisu použít tzv. aliasů…

```
SELECT Objednávky.*
FROM Objednávky INNER JOIN Zákazníci
ON Zákazníci.KódZákazníka=Objednávky.KódZákazníka
WHERE Zákazníci.Firma=“ Alfreds Futterkiste“
```

nebo

```
SELECT A.*
FROM Objednávky A INNER JOIN Zákazníci B
ON A.KódZákazníka=B.KódZákazníka
WHERE B.Firma=“Alfreds Futterkiste“
```

SQL99

Výraz relační algebry:

(Zákazníci(Zákazníci.KódZákazníka=
Objednávky.KódZákazníka)Objednávky)(Zákazníci.Firma=“Alfreds Futterkiste“)[Objednávky.*]

nebo

A = Objednávky
B = Zákazníci
(A(A.KódZákazníka=B.KódZákazníka)B)(B.Firma=“Alfreds Futterkiste“) [A.*]

spojení *restrikce* *projekce*

Příklad 10.
Ve kterých dnech prodával zaměstnanec č.3 zákazníkovi „Alfreds Futterkiste“?

```
SELECT A.DatumObjednávky
FROM Objednávky  A , Zákazníci B
WHERE A.KódZákazníka=B.KódZákazníka
AND A.ČísloZaměstnance=3
AND B.Firma=“Alfreds Futterkiste“
```

SQL99:

```
SELECT A.DatumObjednávky
FROM Objednávky A INNER JOIN Zákazníci B
ON A.KódZákazníka=B.KódZákazníka
WHERE A.Číslo zaměstnance=3
AND B.Firma=“Alfreds Futterkiste“
```

Relační algebra:

```
A = Objednávky
B = Zákazníci
(A(A.KódZákazníka=B.KódZákazníka)B)(A.Číslo zaměstnance=3 AND B.Firma=“Alfreds Futterkiste“)
[DatumObjednávky]
```

Příklad 11.
Ve kterých zemích bydlí zaměstnanci, kteří prodávají zákazníkům z USA?

```
SELECT DISTINCT C.Země
FROM Zákazníci A, Objednávky B, Zaměstnanci C
WHERE A.KódZákazníka = B.KódZákazníka
AND B.ČísloZaměstnance = C.ČísloZaměstnance
AND A.Země="USA"
```

Klíčové slovo DISTINCT za SELECT v SQL znamená, že výsledná tabulka odpovědi nemá obsahovat duplicitní řádky.

SQL99

```
SELECT DISTINCT B.Země
FROM Zákazníci A INNER JOIN (Zaměstnanci B INNER JOIN Objednávky C ON B.ČísloZaměstnance =
C.ČísloZaměstnance) ON A.KódZákazníka = C.KódZákazníka
WHERE A.Země="USA"
```

Relační algebra:

```
A=Zákazníci, B=Objednávky, C=Zaměstnanci
((A (A.KódZákazníka=B.KódZákazníka)B)( B.ČísloZaměstnance=
C.ČísloZaměstnance)C)(A.Země="USA")[C.Země]
```

Ve SQL lze vypočítat souhrn pole přes celou tabulku:

Příklad 12.
Jaká je nejvyšší cena výrobku v kategorii Plodiny?

```
SELECT MAX(B.JednotkováCena)
FROM Kategorie A, Výrobky B
WHERE A.ČísloKategorie =B.ČísloKategorie
AND A.NázevKategorie=“Plodiny“
```

Možné souhrny jsou maximum, minimum, součet, počet hodnot, průměr (klíčová slova MAX, MIN, SUM, COUT, AVG).

Jednoduché dotazy mají ještě klauzuli pro uspořádání řádků tabulky, syntaxe:

SELECT … **FROM** … **WHERE** … **ORDER BY** …

Příklad 13.
Vypište všechny názvy výrobků a ceny výrobků z kategorie Plodiny! Uspořádejte je podle velikosti sestupně!

```
SELECT B.NázevVýrobku, B.JednotkováCena
FROM Kategorie A, Výrobky B
WHERE A.ČísloKategorie =B.ČísloKategorie
AND A.NázevKategorie="Plodiny"
ORDER BY B.JednotkováCena DESC
```

nebo

```
SELECT B.NázevVýrobku, B.JednotkováCena
FROM Kategorie A, Výrobky B
WHERE A.ČísloKategorie =B.ČísloKategorie
AND A.NázevKategorie="Plodiny"
ORDER BY 2 DESC
```

syntaxe za ORDER BY, kde pole, dle kterého se má uspořádávat, udáváme podle jeho pořadí za slovem SELECT

V SQL 99 lze při uspořádání použít i požadavek na vypsání určitého počtu prvních řádků:

Příklad 14.
Jaký je nejdražší výrobek kategorie Plodiny ?

SQL99 (v předchozích verzích SQL by byla formulace daleko obtížnější!):

```
SELECT TOP 1 B.NázevVýrobku, B.JednotkováCena
FROM Kategorie A INNER JOIN Výrobky B ON A.ČísloKategorie =B.ČísloKategorie
WHERE A.NázevKategorie="Plodiny "
ORDER BY B.JednotkováCena DESC
```

Vypiš tři nejdražší výrobky v kategorii Plodiny!

SQL99:

```
SELECT TOP 3 B.NázevVýrobku, B.JednotkováCena
FROM Kategorie A INNER JOIN Výrobky B ON A.ČísloKategorie =B.ČísloKategorie
WHERE A.NázevKategorie="Plodiny "
ORDER BY B.JednotkováCena DESC
```

V SQL lze v dotazech za klíčovým slovem SELECT použít i vypočítaná pole, tedy taková, která nejsou ve zdrojových tabulkách:

Příklad 15.
Vypočítejte peníze v rozpisu objednávek!

```
SELECT ČísloObjednávky,ČísloVýrobku,
 JednotkováCena*Množství*(1-Sleva) AS Mezisoučet
FROM Rozpis_objednávek
```

pojmenování vypočítaného pole

Často používané dotazy lze „uložit“ do databáze ve formě tzv. *pohledů*. Pohledy jsou pojmenované a uložené definice dotazů do schématu databáze:

jméno pohledu

```
DEFINE VIEW Ceny_v_rozpisu_objednávek AS
SELECT ČísloObjednávky,ČísloVýrobku,
JednotkováCena*Množství*(1-Sleva) AS VýslednáCena
FROM Rozpis_objednávek
```

definice pohledu

Definovaný pohled lze použít jako virtuální tabulku v dalších dotazech:

Příklad 16.
Vypište datum a výslednou cenu jednotlivých nákupů výrobku „Aniseed syrup“!

```
SELECT A.DatumObjednávky, B.VýseldnáCena
FROM Objednávky A INNER JOIN (Ceny_v_rozpisu_objednávek B INNER JOIN Výrobky C ON
B.ČísloVýrobku=C. ČísloVýrobku) ON A.ČísloObjednávky=B.ČísloObjednávky
WHERE C.NázevVýrobku=“Aniseed syrob“
ORDER BY 1
```

Příklad 17.
Jaký byl sumární nákup za výrobek „Aniseed syrup“ v roce 1997?

```
SELECT SUM( B.VýslednáCena)
FROM Objednávky A INNER JOIN (Ceny_v_rozpisu_objednávek B INNER JOIN Výrobky C ON
B.ČísloVýrobku=C. ČísloVýrobku) ON A.ČísloObjednávky=B.ČísloObjednávky
WHERE C.NázevVýrobku=“Aniseed syrup“
AND A.DatumObjednávky BETWEEN #1/1/1997# AND #1/31/1997#
```

Co takhle spočítat sumární nákup každého výrobku? Na to jednoduché dotazy nestačí, je potřeba použít tzv. seskupovací dotaz. O nich viz další kapitolu DOTAZOVACÍ JAZYK SQL, seskupovací dotazy.

Uspořádání jakéhokoli dotazu může mít více klíčů uspořádání:

Příklad 18.
Vypište název kategorie, název výrobku a jednotkovou cenu výrobků, uspořádejte je podle kategorií a v rámci kategorie sestupně podle ceny!

```
SELECT A.NázevKategorie, B.NázevVýrobku, B.JednotkováCena
FROM Kategorie A INNER JOIN Výrobky B ON A.ČísloKategorie=B.ČísloKategorie
ORDER BY 1 , 3 DESC
```

pro uživatelsky přívětivé prohlížení odpovědi je ovšem vhodnější uspořádávat podle polí zleva doprava:

```
SELECT A.NázevKategorie, B.JednotkováCena, B.NázevVýrobku
FROM Kategorie A INNER JOIN Výrobky B ON A.ČísloKategorie=B.ČísloKategorie
ORDER BY 1 , 2 DESC
```

Klíčovým slovem UNION lze spojit dva dotazy, které mají stejný seznam polí (stejná jména a stejné domény). Takovýto UNION-dotaz lze navíc uspořádat „přes všechny propojené dotazy“.

Příklad 19.
Vypište zemi, město, jméno firmy, kontaktní osobu a telefon zákazníků a také dodavatelů, uspořádejte je podle zemí a měst!

```
SELECT Země, Město, KontaktníOsoba, Telefon
FROM Zákazníci
UNION
SELECT Země, Město, KontaktníOsoba, Telefon
FROM Dodavatelé
ORDER BY 1 , 2
```

Dotazy s vnějším spojením (SQL99)

Levé vnější (polo-)spojení LEFT JOIN obsahuje všechny záznamy z první (levé) ze dvou tabulek, i když ve druhé (pravé) tabulce neodpovídají záznamům žádné hodnoty. Pravé vnější (polo-)spojení RIGHT JOIN zahrnuje všechny záznamy z druhé (pravé) ze dvou tabulek, dokonce i pokud neexistují žádné hodnoty odpovídající záznamům v první (levé) tabulce.

Operaci LEFT JOIN můžete například použít pro tabulky Oddělení (levá) a Zaměstnanci (pravá) k výběru všech oddělení včetně těch, do kterých nejsou zařazeni žádní zaměstnanci:

```
SELECT * FROM Oddělení O LEFT JOIN Zaměstnanci Z ON O.ČísloOddělení=Z.ČísloOddělení
```

Chcete-li vybrat všechny zaměstnance včetně těch, kteří nejsou zařazeni do žádného oddělení, použijte operaci RIGHT JOIN:

```
SELECT * FROM Oddělení O RIGHT JOIN Zaměstnanci Z ON O.ČísloOddělení=Z.ČísloOddělení
```

Příklad 20.
Vypište zákazníky, kteří od 1.4.1998 nic nekoupili! (Použije se tzv. subselect – o něm více viz kapitolu DOTAZOVACÍ JAZYK SQL, poddotazy)

```
SELECT A.Firma FROM Zákazníci A LEFT JOIN (SELECT KódZákazníka FROM Objednávky WHERE
DatumObjednávky>#4/1/1998#) B ON A.KódZákazníka=B.KódZákazníka
WHERE B.KódZákazníka IS NULL
```

Cvičení (formulujte SQL pro následující dotazy):
Jaké je formální oslovení obchodního ředitele?
Kteří dodavatelé dodávají cukrovinky?
Které kontaktní osoby jsou majiteli zákaznické firmy?
Kteří zákazníci nakupují výrobek „Sir Rodney’s Marmelade“?
Kteří dodavatelé dodávají zboží dražší než 1 000,-?
Jaký je popis kategorie „Obilné výrobky“?
Kteří přepravci účtovali přepravné vyšší než 20 000,-?
Které výrobky od r.1997 zdražily (porovnání cen v tabulkách Rozpis_objednávek a Výrobky)?

DOTAZOVACÍ JAZYK SQL, seskupovací dotazy

Jednoduchým dotazům můžeme nahlížet jako *jednoduché pohledy*. Seskupovací dotaz – *seskupený pohled* – vznikne seskupením a agregací z jednoduchého pohledu.

Příklad 21. Uvažme ceny v rozpisu objednávek z ledna roku 1997. SQL dotaz ve verzi 99 může vypadat takto:

```
SELECT A.ČísloObjednávky,B.ČísloVýrobku, B.JednotkováCena*B.Množství*(1-B.Sleva) AS VýslednáCena
FROM Objednávky A INNER JOIN Rozpis_objednávek B ON A.ČísloObjednávky=B. ČísloObjednávky
WHERE A.DatumObjednávky Between #1/1/1997# And #1/31/1997#
```

Vznikne pohled:

ČísloObjednávky	ČísloVýrobku	VýslednáCena
10400	29	51 975,00
10400	35	12 600,00
10400	49	12 000,00
10401	30	9 315,00
10401	56	53 200,00
10401	65	8 400,00
10401	71	25 800,00
…	…	…

V pohledu přicházívají do úvahy různá *rozdělení do skupin* – např.:

Číslo objednávky	ČísloVýrobku	VýslednáCena	
10400	35	12 600,00	ČísloObjednávky 10400
10400	49	12 000,00	
10400	29	51 975,00	
10401	30	9 315,00	ČísloObjednávky 10401
10401	56	53 200,00	
10401	65	8 400,00	
10401	71	25 800,00	
10402	23	10 800,00	ČísloObjednávky 10402
10402	63	57 037,50	
…	…	…	…

podle čísel objednávek

nebo

Číslo objednávky	ČísloVýrobku	VýslednáCena	
10413	1	8 640,00 Kč	ČísloVýrobku 1
10406	1	3 600,00 Kč	
10418	2	22 800,00 Kč	ČísloVýrobku 2
10405	3	10 000,00 Kč	ČísloVýrobku 3
10420	9	34 920,00 Kč	ČísloVýrobku 9
10407	11	12 600,00 Kč	ČísloVýrobku 11
10420	13	216,00 Kč	ČísloVýrobku 13
10409	14	5 580,00 Kč	ČísloVýrobku 14
10412	14	8 370,00 Kč	
…	…	…	…

podle čísel výrobků

Sloupce, podle kterých neprobíhá rozdělení do skupin, mohou být v *seskupeném pohledu* agregovány. Zde jsou vybrány agregace „počet řádků“ a „součet pro peníze“:

ČísloObjednávky	Počet	Součet
10400	3	76 575,00
10401	4	96 715,00
10402	2	67 837,50
…	…	…

seskupeno podle

agregováno

nebo

Počet	ČísloVýrobku	Součet
2	1	12 240,00
1	2	22 800,00
1	3	10 000,00
1	9	34 920,00
1	11	12 600,00
1	13	216,00
3	14	30 225,00
…	…	…

seskupeno podle ČíslaVýrobku, ostatní agregováno

SQL kód pro první agregovaný pohled je (změny vůči jednoduchému pohledu jsou zvýrazněny):

```
SELECT A.ČísloObjednávky, COUNT(*) AS Počet,
SUM(B.JednotkováCena*B.Množství*(1-B.Sleva)) AS Součet
FROM Objednávky A INNER JOIN Rozpis_objednávek B ON A.ČísloObjednávky=B. ČísloObjednávky
WHERE A.DatumObjednávky Between #1/1/1997# And #1/31/1997#
GROUP BY A.ČísloObjednávky
```

SQL kód pro druhý agregovaný pohled je:

```
SELECT COUNT(*) AS Počet, B.ČísloVýrobku,
SUM(B.JednotkováCena*B.Množství*(1-B.Sleva)) AS Součet
FROM Objednávky A INNER JOIN Rozpis_objednávek B ON A.ČísloObjednávky=B. ČísloObjednávky
WHERE A.DatumObjednávky Between #1/1/1997# And #1/31/1997#
GROUP BY B.ČísloVýrobku
```

Poznámka

Ve všech příkladech v této kapitole lze syntaxi SQL99 používající tabulkové výrazy sestavené pomocí INNER JOIN nahradit starší syntaxí se seznamem tabulek a s podmínkami spojení napsanými za WHERE.

Příklad 22.

Ve vzorové databázi můžeme vytvořit pohled, který pro každou objednávku vypočítává celkovou částku k platbě za objednané zboží[18].

```
SELECT ČísloObjednávky, SUM(JednotkováCena* Množství*(1- Sleva)) AS Mezisoučet
FROM Rozpis_objednávek
GROUP BY ČísloObjednávky
```

Můžeme i definovat pohled „Mezisoučty objednávek“ na základě tohoto dotazu:

```
DEFINE VIEW Mezisoučty_objednávek AS
SELECT ČísloObjednávky, SUM(JednotkováCena* Množství*(1- Sleva)) AS Mezisoučet
FROM Rozpis_objednávek
GROUP BY ČísloObjednávky
```

[18] Ve vzorové databázi Northwind v SŘBD MS Access je analogický dotaz častokrát používaný jako pohled. V tomto dotazu je ale při výpočtu částky pro „Mezisoučet“ použita ještě konverze datových typů, aby byly zaručena přesnost výpočtů s penězi. To proto, že výraz JednotkováCena* Množství*(1- Sleva) je typu DOUBLE PRECISSION, což není typ zachovávající přesnost. Více o datových typech viz kapitolu PŘEHLED OSTATNÍHO JAZYKA SQL.

Příklad 23.
Pro každého zákazníka spočítejte nákup za rok 1997! (Výsledek lze spočítat ze základních tabulek nebo lze použít pohled „Mezisoučty objednávek“.)

```
SELECT A.KódZákazníka, A.Firma, SUM(JednotkováCena*Množství*(1-Sleva)) AS Nákup
FROM (Zákazníci  A INNER JOIN Objednávky B ON A.KódZákazníka = B.KódZákazníka) INNER JOIN
Rozpis_objednávek C ON B.ČísloObjednávky = C.ČísloObjednávky
WHERE Objednávky.DatumObjednávky BETWEEN #1.1.1997# AND #31.12.1997#
GROUP BY A.KódZákazníka
```

Nebo s použitím pohledu „Mezisoučty objednávek“

```
SELECT A.KódZákazníka, A.Firma, SUM(C.Mezisoučet) AS Nákup
FROM (Zákazníci  A INNER JOIN Objednávky B ON A.KódZákazníka = B.KódZákazníka) INNER JOIN
Mezisoučty_objednávek C ON B.ČísloObjednávky = C.ČísloObjednávky
WHERE Objednávky.DatumObjednávky BETWEEN #1.1.1997# AND #31.12.1997#
GROUP BY A.KódZákazníka
```

Příklad 24.
Spočítejte pro každý výrobek tržbu v USA! (Využijeme pohled „Ceny v rozpisu objednávek“.)

```
SELECT C.ČísloVýrobku, C.NázevVýrobku, SUM(B.VýslednáCena) AS Tržba_v_USA
FROM (Objednávky A INNER JOIN Ceny_v_rozpisu_objednávek B ON A.ČísloObjednávky =
B.ČísloObjednávky ) INNER JOIN Výrobky C ON B.ČísloVýrobku = C.ČísloVýrobku
WHERE A.ZeměPříjemce="USA"
GROUP BY C.ČísloVýrobku
```

Seskupovat lze i podle více polí:
Příklad 25.
Spočítejte pro každý výrobek tržbu v jednotlivých zemích!

```
SELECT C.ČísloVýrobku, C.NázevVýrobku, A.ZeměPříjemce AS Země, SUM(B.VýslednáCena) AS Tržba
FROM (Objednávky A INNER JOIN Ceny_v_rozpisu_objednávek B ON A.ČísloObjednávky =
B.ČísloObjednávky ) INNER JOIN Výrobky C ON B.ČísloVýrobku = C.ČísloVýrobku
GROUP BY C.ČísloVýrobku, A.ZeměPříjemce
```

Vytvoření odpovědi na takový dotaz si můžeme představit následovně. Nejprve si představme odpovídající jednoduchý pohled rozdělený do skupin podle výrobků a zemí:

Číslo výrobku	Název výrobku	ZeměPříjemce	VýslednáCena	
1	Chai	Belgie	4 500,00 Kč	
1	Chai	Brazílie	9 450,00 Kč	
1	Chai	Brazílie	8 550,00 Kč	
1	Chai	Brazílie	3 600,00 Kč	
1	Chai	Finsko	900,00 Kč	
1	Chai	Finsko	4 050,00 Kč	
1	Chai	Finsko	3 060,00 Kč	
1	Chai	…	…	další řádky s výrobkem Chai
2	Chang	Brazílie	3 800,00 Kč	
2	Chang	Brazílie	2 256,25 Kč	
2	Chang	Brazílie	4 560,00 Kč	
2	Chang	Francie	7 125,00 Kč	
2	Chang	Francie	6 768,75 Kč	
2	Chang	…	…	další řádky s výrobkem Chang
3	Aniseed Syrup	…	…	řádky s výrobkem Aniseed Syrup
…	…	…	…	řádky s dalšími výrobky

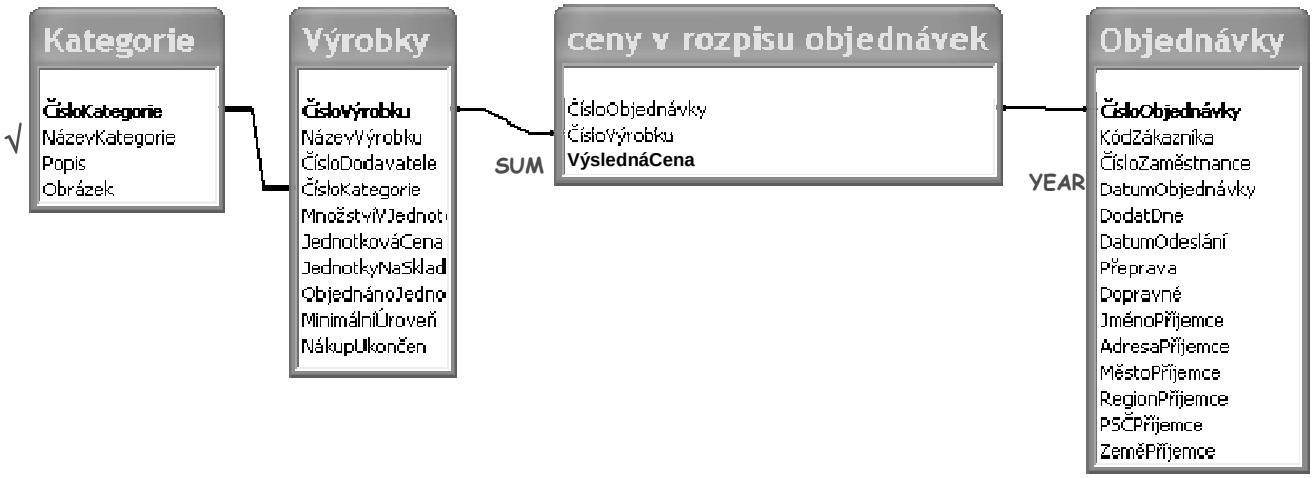
V každé skupině vzniklé předchozím rozdělením se provede agregace. Výsledek:

Číslo výrobku	Název výrobku	Země	Tržba
1	Chai	Belgie	4 500,00 Kč
1	Chai	Brazílie	21 600,00 Kč
1	Chai	…	…
2	Chang	Brazílie	20 116,25 Kč
2	Chang	Francie	13 893,75 Kč
2	Chang	…	…
3	Aniseed Syrup	…	…
…	…	…	…

Příklad 26.
Pro každou kategorii spočítejte tržbu za jednotlivé roky! (Využije se standardní funkce YEAR z verze standardu SQL99. Ne každý SŘBD tuto funkci podporuje.)

SELECT A.NázevKategorie, **YEAR(**D.DatumObjednávky**) AS** Rok, **SUM(**C.VýslednáCena) **AS** Tržba
FROM (Kategorie A **INNER JOIN** (Výrobky B **INNER JOIN** Ceny_v_rozpisu_objednávek C **ON** B.ČísloVýrobku = C.ČísloVýrobku) **ON** A.ČísloKategorie = B.ČísloKategorie) **INNER JO**IN Objednávky D **ON** C.ČísloObjednávky = D.ČísloObjednávky
GROUP BY A.ČísloKategorie, **YEAR(**D.DatumObjednávky**)**

„Schéma“ tohoto dotazu si lze představit takto:



Příklad 27.
Kolik utržil v jednotlivých kategoriích a jednotlivých letech zaměstnanec Andrew Fuller?

SELECT B.NázevKategorie, **YEAR(**E.DatumObjednávky**) AS** Rok, **SUM(**D.VýslednáCena**) AS** Tržba
FROM Zaměstnanci A **INNER JOIN** (((Kategorie B **INNER JOIN** Výrobky C **ON** B.ČísloKategorie = C.ČísloKategorie) **INNER JOIN** Ceny_v_rozpisu objednávek D **ON** C.ČísloVýrobku = D.ČísloVýrobku) **INNER JOIN** Objednávky E **ON** D.ČísloObjednávky = E.ČísloObjednávky) **ON** Zaměstnanci.ČísloZaměstnance = Objednávky.ČísloZaměstnance
WHERE A.Příjmení="Fuller" AND A.Jméno="Andrew"
GROUP BY B.NázevKategorie, **YEAR(**E.DatumObjednávky**)**

Většina klientských aplikací pro tvorbu dotazů do relačních databází umožňuje zobrazit odpovědi na dotazy se dvěma hledisky seskupování jako „křížovou tabulku“. Tedy namísto

Název kategorie	Rok	Tržba
Cukrovinky	1996	70 500,00
Cukrovinky	1997	292 653,25
Cukrovinky	1998	173 238,87
Koření	1996	90 796,00
Koření	1997	150 919,25
Koření	1998	129 551,50
Maso / Drůbež	1996	176 010,00
Maso / Drůbež	1997	299 464,75
Maso / Drůbež	1998	271 365,25
Mléčné výrobky	1996	52 800,00
Mléčné výrobky	1997	311 001,25
Mléčné výrobky	1998	231 512,50
Mořské produkty	1996	29 076,00
Mořské produkty	1997	192 692,50
Mořské produkty	1998	171 920,75
Nápoje	1996	53 760,00
Nápoje	1997	208 127,50
Nápoje	1998	706 518,75
Obilné výrobky	1996	38 497,50
Obilné výrobky	1997	170 970,00
Obilné výrobky	1998	69 856,25
Plodiny	1996	32 487,00
Plodiny	1997	135 275,00
Plodiny	1998	104 450,00

zobrazit odpověď jako:

Název kategorie	1996	1997	1998
Cukrovinky	70 500,00	292 653,25	173 238,87
Koření	90 796,00	150 919,25	129 551,50
Maso / Drůbež	176 010,00	299 464,75	271 365,25
Mléčné výrobky	52 800,00	311 001,25	231 512,50
Mořské produkty	29 076,00	192 692,50	171 920,75
Nápoje	53 760,00	208 127,50	706 518,75
Obilné výrobky	38 497,50	170 970,00	69 856,25
Plodiny	32 487,00	135 275,00	104 450,00

V datových skladech je možno vytvářet útvary analogické křížovým tabulkám nazývané datové kostky. Jsou vícedimenzionálním zobecněním křížových tabulek, navíc jsou doplněny agregovanými útvary, jež v případě dvoudimenzionální tabulky jsou řádkem a sloupcem součtů:

Název kategorie	1996	1997	1998	Celkový součet
Cukrovinky	70500	292653,25	173238,87	536392,12
Koření	90796	150919,25	129551,5	371266,75
Maso / Drůbež	176010	299464,75	271365,25	746840
Mléčné výrobky	52800	311001,25	231512,5	595313,75
Mořské produkty	29076	192692,5	171920,75	393689,25
Nápoje	53760	208127,5	706518,75	968406,25
Obilné výrobky	38497,5	170970	69856,25	279323,75
Plodiny	32487	135275	104450	272212
Celkový součet	543926,5	1761103,5	1858413,87	4163443,87

Příklad 28.
Pro každého dodavatele spočítejte počet druhů výrobků, které dodává, průměrnou, nejnižší a nejvyšší cenu!

```
SELECT A.Firma, COUNT(*), AVG(B.JednotkováCena), MIN(B.JednotkováCena), MAX(B.JednotkováCena)
FROM Dodavatelé A , Výrobky B
WHERE A.ČísloDodavatele=B.ČísloDodavatele
GROUP BY B.ČísloDodavatele
```

V SQL99 je možno vytvářet i výrazy z agregací:

Příklad 29.
Pro každého přepravce spočítejte celkovou cenu přepravovaného zboží a celkovou sumu za dopravné, a podíl těchto hodnot, za rok 1997!

```
SELECT A.Firma, SUM(C.Mezisoučet) AS Za_zboží, SUM(B.Dopravné) AS Za_dopravu,
SUM(B.Dopravné)/SUM(C.Mezisoučet) AS Podíl
FROM (Přepravci A INNER JOIN Objednávky B ON A.ČísloPřepravce = B.Přeprava) INNER JOIN
Mezisoučty_objednávek C ON B.ČísloObjednávky = C.ČísloObjednávky
WHERE YEAR(B.DatumObjednávky)=1997
GROUP BY A.ČísloPřepravce
```

(V tomto dotazu je použití pohledu „Mezisoučty objednávek“ podstatné. Použití cen v rozpisu objednávek místo toho by způsobilo započítání každého dopravného za objednávku tolikrát, kolik je v objednávce druhů výrobků!)

Příklad 30.
Spočítejte počet různých výrobků nakupovaných jednotlivými zákazníky![19]

```
SELECT A.Firma, COUNT DISTINCT(C.ČísloVýrobku) AS Počet_druhů
FROM (Zákazníci A INNER JOIN Objednávky B ON A.KódZákazníka = B. KódZákazníka) INNER JOIN
Rozpis_Objednávek C ON B.ČísloObjednávky=C. ČísloObjednávky
GROUP BY A. KódZákazníka
```

Cvičení:
Spočítejte průměrnou cenu výrobků v jednotlivých kategoriích!
Spočítejte počet objednávek jednotlivých zaměstnanců!
Pro každého zákazníka spočítejte celkovou výši plateb za zboží a za dopravu!
Spočítejte tržbu za výrobky jednotlivých dodavatelů v jednotlivých kategoriích!
Spočítejte pro každý výrobek celkovou tržbu!

Výběr skupin v seskupovacím dotazu

Seskupovací dotazy lze doplnit požadavkem na výběr skupin, jež mají být do odpovědi zařazeny.

Příklad 31.
Vypište zákazníky, kteří v roce 1997 celkem nakoupili více než za 500 000,-!

```
SELECT A.KódZákazníka, A.Firma
FROM (Zákazníci A INNER JOIN Objednávky B ON Zákazníci.KódZákazníka = Objednávky.KódZákazníka)
INNER JOIN Mezisoučty_objednávek C ON Objednávky.ČísloObjednávky = C.ČísloObjednávky
WHERE YEAR(B.DatumObjednávky)=1997
GROUP BY A.KódZákazníka
HAVING SUM(C.Mezisoučet)>500000
```

Jak je vidět, v tomto typu dotazů nebývá nutné agregovanou hodnotu, podle které se skupina vybírá, zobrazovat v odpovědi. Jsou možné i varianty s uspořádáním:

V předchozím příkladu seřaďte zákazníky sestupně podle celkového nákupu v roce 1997, tento zobrazte!

```
SELECT A.KódZákazníka, A.Firma, SUM(C.Mezisoučet) AS Nákup
FROM (Zákazníci A INNER JOIN Objednávky B ON Zákazníci.KódZákazníka = Objednávky.KódZákazníka)
INNER JOIN Mezisoučty_objednávek C ON Objednávky.ČísloObjednávky = C.ČísloObjednávky
WHERE YEAR(B.DatumObjednávky)=1997
GROUP BY A.KódZákazníka
HAVING SUM(C.Mezisoučet)>500000
ORDER BY 3 DESC
```

[19] Pokud k procvičování používáte MS Access, pak toto cvičení nelze provést – MS Access nepodporuje COUNT DISTINCT. (Ke stejnému výsledku se v MS Access můžete dobrat ve dvou krocích, nejprve výběrem různých čísel výrobků a pak jejich spočítáním.)

DOTAZOVACÍ JAZYK SQL, poddotazy

Poddotazy neboli subselecty se v SQL dotazu mohou vyskytovat na několika místech.

Vektorový subselect

Je používaný za klauzulí WHERE. Použitelný pro porovnání IN nebo EXISTS.

Příklad 32.
Kteří zákazníci v roce 1997 nakupovali výrobky, za něž byla v tomto roce celková tržba menší než 40 000,-?

SELECT DISTINCT A.Firma **FROM** Zákazníci A, Objednávky B, Rozpis_objednávek C **WHERE** A.KódZákazníka=B.KódZákazníka **AND** B.ČísloObjednávky=C.ČísloObjednávky **AND** B.DatumObjednávky **BETWEEN** #1/1/1997# **AND** #12/31/1997#
AND C.ČísloVýrobku **IN (SELECT** D.ČísloVýrobku **FROM** Rozpis_objednávek D, Objednávky E **WHERE** D.ČísloObjednávky=E.ČísloObjednávky **AND** E.DatumObjednávky **BETWEEN** #1/1/1997# **AND** #12/31/1997# **GROUP BY** D.ČísloVýrobku **HAVING SUM(**D.JednotkováCena*D.Množství*(1-D.Sleva))<40000**)**

Příklad 33.
Kteří zákazníci někdy nakupovali do dvou dnů po sobě?

SELECT DISTINCT A.KódZákazníka FROM Objednávky A WHERE EXISTS (SELECT * FROM Objednávky B WHERE B.DatumObjednávky<A.DatumObjednávky AND B.DatumObjednávky>A.DatumObjednávky-2 AND A.KódZákazníka=B.KóZákazníka)

Lze použít i porovnání s ALL:

Příklad 34.
Které mléčné výrobky jsou dražší než jakýkoli obilný výrobek?

SELECT A.NázevVýrobku **FROM** Výrobky A, Kategorie B **WHERE** A.ČísloKategorie=B.ČísloKategorie **AND** B.NázevKategorie="Mléčné výrobky" **AND** A.JednotkováCena>**ALL(SELECT** C.JednotkováCena **FROM** Výrobky C, Kategorie D **WHERE** C.ČísloKategorie=D.ČísloKategorie **AND** B.NázevKategorie="Obilné výrobky"**)**

Mohli bychom tento dotaz přeformulovat i pomocí agregační funkce MAX:

SELECT A.NázevVýrobku **FROM** Výrobky A, Kategorie B **WHERE** A.ČísloKategorie=B.ČísloKategorie **AND** B.NázevKategorie="Mléčné výrobky" **AND** A.JednotkováCena> **(SELECT MAX(**C.JednotkováCena**) FROM** Výrobky C, Kategorie D **WHERE** C.ČísloKategorie=D.ČísloKategorie **AND** B.NázevKategorie="Obilné výrobky"**)**

ale pokud by snad náhodou neexistoval žádný obilný výrobek, pak tato varianta dotazu dává oproti očekávání prázdnou odpověď, tedy žádný mléčný výrobek. Varianta s ALL tedy spíše odpovídá přirozenému chápání smyslu dotazu.

Skalární subselect

Ve verzích SQL předcházejících SQL99 může být skalární subselect použit za klauzulí WHERE.

Příklad 35.
Vyber výrobky s podprůměrnou cenou!

```
SELECT * FROM Výrobky A WHERE A.JednotkováCena>(SELECT AVG(B.JednotkováCena) FROM
Výrobky B)
```

Ve verzi SQL99 lze použít skalární subselect v seznamu polí za SELECT:

Příklad 36.
V každé zemi nalezněte zákazníka s nejvyšším celkovým nákupem.

```
SELECT A.Země, (SELECT TOP 1 B.KódZákazníka FROM Zákazníci B INNER JOIN (Objednávky C
INNER JOIN Mezisoučty_objednávek D ON C.ČísloObjednávky = D.ČísloObjednávky) ON B.KódZákazníka
= C.KódZákazníka WHERE B.Země=A.Země GROUP BY C.KódZákazníka ORDER BY SUM(D.Mezisoučet)
DESC)
FROM Zákazníci A
```

Subselecty jako tabulkové výrazy

Standard SQL99 umožňuje použít subselect na místě tabulky za FROM. Jedním příkladem je **Příklad 20.** na str.66. Jiný příklad může být efektivnější varianta řešení pro **Příklad 36.**:

```
SELECT A.Země, (SELECT TOP 1 B.Firma FROM Zákazníci B INNER JOIN (Objednávky C INNER JOIN
Mezisoučty_objednávek D ON C.ČísloObjednávky = D.ČísloObjednávky) ON B.KódZákazníka =
C.KódZákazníka WHERE B.Země=A.Země GROUP BY C.KódZákazníka ORDER BY SUM(D.Mezisoučet)
DESC)
FROM (SELECT DISTINCT E.Země FROM Zákazníci E) A
```

Subselect jako tabulkový výraz lze vždy nahradit pomocí temporární tabulky (viz kapitola PŘEHLED OSTATNÍHO JAZYKA SQL, podkapitola Jazyk definice dat) nebo pohledu. Provedení se subselectem je však rychlejší.

Následující dotaz je efektivnější variantou řešení pro **Příklad 32.**

```
SELECT DISTINCT A.Firma
FROM ((Zákazníci A INNER JOIN Objednávky B ON A.KódZákazníka=B.KódZákazníka) INNER JOIN
Rozpis_objednávek C ON B.ČísloObjednávky=C.ČísloObjednávky)
INNER JOIN
(SELECT D.ČísloVýrobku FROM Rozpis_objednávek D, Objednávky E WHERE
D.ČísloObjednávky=E.ČísloObjednávky AND E.DatumObjednávky BETWEEN #1/1/1997# AND
#12/31/1997# GROUP BY D.ČísloVýrobku HAVING SUM(D.JednotkováCena*D.Množství*(1-
D.Sleva))<40000)
F ON C.ČísloVýrobku=F. ČísloVýrobku
WHERE B.DatumObjednávky BETWEEN #1/1/1997# AND #12/31/1997#
```

Tento příklad bude ještě řešen v kapitole OPTIMALIZACE DOTAZŮ.

SQL SELECT přehled syntaxe

Jednoduché dotazy:

SELECT [*predikat*] { * | *tabulka*.* | [*tabulka*.]*pole1* [AS *alias1*] [, [*tabulka*.]*pole2* [AS *alias2*] [, ...]]}
FROM *tabulkovyvyraz* [, ...]
[WHERE...]
[ORDER BY...]

predikat	Jeden z následujících predikátů: ALL, DISTINCT, nebo TOP a číslo udávající počet řádků odpovědi. Predikáty slouží k omezení počtu vybraných záznamů. Pokud není zadán žádný, platí standardně ALL.
*	Určuje, že ze zadané tabulky nebo tabulek budou vybrána všechna pole.
tabulka	Název tabulky obsahující pole, ze kterých jsou vybírány záznamy. Pokud se název pole v klauzuli FROM vyskytuje ve více než jedné tabulce, umístěte před něj název tabulky a operátor tečka (.).
pole1, pole2	Názvy polí obsahující data, která chcete načíst. Pokud zadáte více než jedno pole, budou načtena v pořadí, v jakém byla zadána. Mohou být i výrazy v polích tabulek z tabulkového výrazu.
alias1, alias2	Názvy, které budou použity jako záhlaví sloupců namísto původních názvů sloupců v tabulce.
tabulkovyvyraz	Výraz určující jednu nebo více tabulek, ze kterých budou načtena data. Může se jednat o název jedné tabulky, název pohledu nebo (pouze ve verzi SQL99) výraz složený z tabulkových výrazů pomocí operací INNER JOIN, LEFT JOIN nebo RIGHT JOIN. Ve verzi SQL99 může v roli tabulky vystupovat i sub-select.

Pokud zadáte v dotazu více než jednu tabulku a neuvedete klauzuli WHERE ani JOIN, vytvoří dotaz kartézský součin tabulek.

Dotazy s GROUP BY – seskupovací:

SELECT [*predikat*] *seznampoli*
FROM *tabulkovyvyraz* [, ...]]
[WHERE *kriteria*]
[GROUP BY *seznampoliskupiny*
[HAVING *kriteriaskupiny*]]
[ORDER BY...]

seznampoli	Název pole nebo polí, která mají být vybrána, a dále všechny aliasy názvů polí, agregační funkce SQL (AVG, COUNT, COUNT DISTINCT, MIN, MAX, SUM), výběrové predikáty (ALL, DISTINCT, nebo TOP) a výrazy z předchozího.
kriteria	Podmínka výběru. Pokud příkaz obsahuje klauzuli WHERE, seskupí databázový stroj hodnoty až po provedení výběru záznamů podle podmínek uvedených v klauzuli WHERE.
seznampoliskupiny	Názvy polí použitých k seskupení záznamů. Pořadí názvů polí uvedených v *seznamupolískupiny* určuje úroveň skupin od nejvyšší do nejnižší.
kriteriaskupiny	Výraz určující, které seskupené záznamy mají být zobrazeny

Dotazy s UNION:

Operace UNION vytvoří sjednocovací dotaz, který slouží ke slučování výsledků dvou nebo více nezávislých dotazů nebo tabulek:

[TABLE] *dotaz1* UNION [ALL] [TABLE] *dotaz2* [UNION [ALL] [TABLE] *dotazn* [...]] [ORDER BY...]

dotaz1-n	Příkaz SELECT, název pohledu nebo název tabulky, kterému předchází klíčové slovo TABLE.

Při operaci UNION se standardně nevrací duplicitní záznamy. Vrácení všech záznamů lze zajistit pomocí predikátu ALL. Je nutné, aby všechny dotazy v operaci UNION vracely stejný počet polí, odpovídající pole stejného typu. Aliasy používejte pouze v prvním příkazu SELECT, jinde jsou ignorovány. Klauzuli GROUP BY nebo HAVING lze použít k seskupení vrácených dat v každém argumentu *dotaz1-n*. V klauzuli ORDER BY odkazujte na pole stejnými názvy jako v prvním příkazu SELECT nebo pořadím polí.

Poznámka o standardu SQL

Syntaxe se u jednotlivých relačních SŘBD v detailech liší. Také podpora jednotlivých rysů jazyka SQL nemusí být vždy samozřejmá, dokonce i když dodavatel SŘBD tvrdí o svém produktu, že tento podporuje určitou verzi standardu SQL. Nutné je vždy konzultovat dokumentaci.

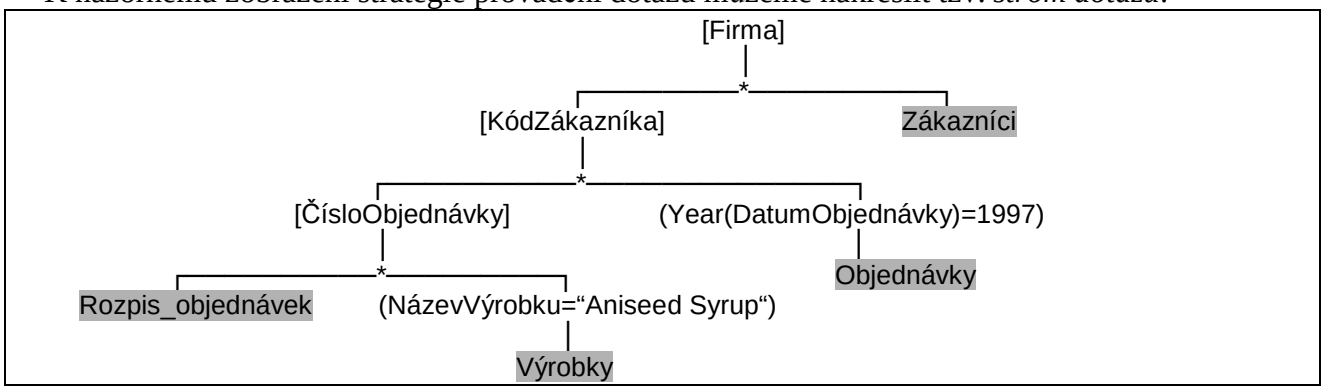
Z toho vyplývá, že přenositelnost SQL kódu mezi různými SŘBD není zaručena. Tento nedostatek do jisté míry odstraňují „překladače z jednoho SQL do druhého“ (např. v prostředí MS Windows nazývané ODBC ovladače). Jejich spolehlivost však není stoprocentní.

OPTIMALIZACE DOTAZŮ

Relační SŘBD běžně provádějí optimalizaci SQL dotazů, jež spočívá ve volbě pořadí mezivýsledků a volbě algoritmu pro operaci spojení. Výběr algoritmu pro spojení závisí na počtu řádků tabulek, délce řádků, velikosti domén atributů, existenci indexu, selektivitě podmínek…. Toto nemůže programátor databázové aplikace ani databázový administrátor ovlivnit přímo, nanejvýš jinou formulací dotazu, u níž experimentálně zjistí rychlejší provádění. K volbě pořadí mezivýsledků se používají heuristiky, jež povětšinou dávají dobrý výsledek (základní strategií je omezit velikosti mezivýsledků operacemi selekce a projekce):

1. selekce neboli restrikce provést co nejdříve, event. kaskády selekcí
2. projekce co nejdříve; vyzkoušej zda některé nejsou zbytečné
3. selekci na kartézském součinu transformuj na spojení, je-li to možné
4. selekce a projekce po spojení aplikuj vždy v okamžiku vytvoření nového spojeného řádku
5. uprav pořadí mezivýsledků tak, aby selekce produkující menší výsledek byly volány dříve
6. jestliže se nějaký podvýraz v dotazu opakuje a jeho výsledek není příliš velký, spočítej ho jednou do pomocné tabulky

Příklad 37.
Kteří zákazníci objednali v roce 1997 výrobek Aniseed Syrup?
– K názornému zobrazení strategie provádění dotazu můžeme nakreslit tzv. *strom dotazu*:



Strom se čte od listů, na nichž jsou tabulky, cestou vzhůru postupně provádíme operace, které potkáváme.

Optimalizaci má SŘBD provádět „v pozadí“, tak že se o ni při tvorbě aplikací nemusíme starat. Můžeme ovšem dojít k názoru, že provádění některého dotazu je pomalejší, než by mohlo být. Potom můžeme provést optimalizaci ručně, tvorbou a použitím mezivýsledků.

Příklad 38.
Kteří zákazníci v roce 1997 vytvořili více než 5% naší celkové roční tržby?

```
CREATE TEMPORARY TABLE Pom1 SELECT O.KódZákazníka, SUM(JednotkováCena*Množství*(1-
Sleva) AS tržba
FROM Objednávky O INNER JOIN Rozpis_objednávek R ON O.ČísloObjednávky = R.ČísloObjednávky
WHERE Year(O.DatumObjednávky)=1997
GROUP BY O.KódZákazníka;

(SELECT sum(tržba) FROM Pom1) uložíme do proměnné pom2

SELECT Z.Firma
FROM Zákazníci Z INNER JOIN Pom1 ON Z.KódZákazníka = pom1.KódZákazníka
WHERE pom1.tržba>0.05*pom2;

DROP TABLE Pom1;
```

Příklad 39.
Na str.74 byl řešen příklad „Kteří zákazníci v roce 1997 nakupovali výrobky, za něž byla v tomto roce celková tržba menší než 40 000,-?“. Zde se jeho řešení pokusíme optimalizovat.

```
CREATE TEMPORARY TABLE pom SELECT ČísloVýrobku, JednotkováCena*Množství*(1-Sleva) AS
Platba, KódZákazníka
FROM Objednávky O INNER JOIN Rozpis_objednávek R ON O.ČísloObjednávky = R.ČísloObjednávky
WHERE Year(O.DatumObjednávky)=1997;

SELECT Z.Firma FROM
(SELECT DISTINCT pom.KódZákazníka FROM
(SELECT ČísloVýrobku FROM pom GROUP BY ČísloVýrobku HAVING SUM(Platba)<40000) A INNER
JOIN pom ON A.ČísloVýrobku=pom.ČísloVýrobku) ) B INNER JOIN Zákazníci Z ON
B.KódZákazníka=Z.KódZákazníka);

DROP TABLE pom;
```

Tabulka pom slouží k tomu, aby se nemuselo spojení tabulek Objednávky a Rozpis_objednávek dělat dvakrát. Druhý příkaz je dotaz se dvěma vnořenými subselecty, z nichž jeden obsahuje klíčové slovo DISTINCT. Pokud by verze SQL, kterou máte k dispozici, toto neumožňovala, nahradili bychom druhý a třetí příkaz následujícími čtyřmi:

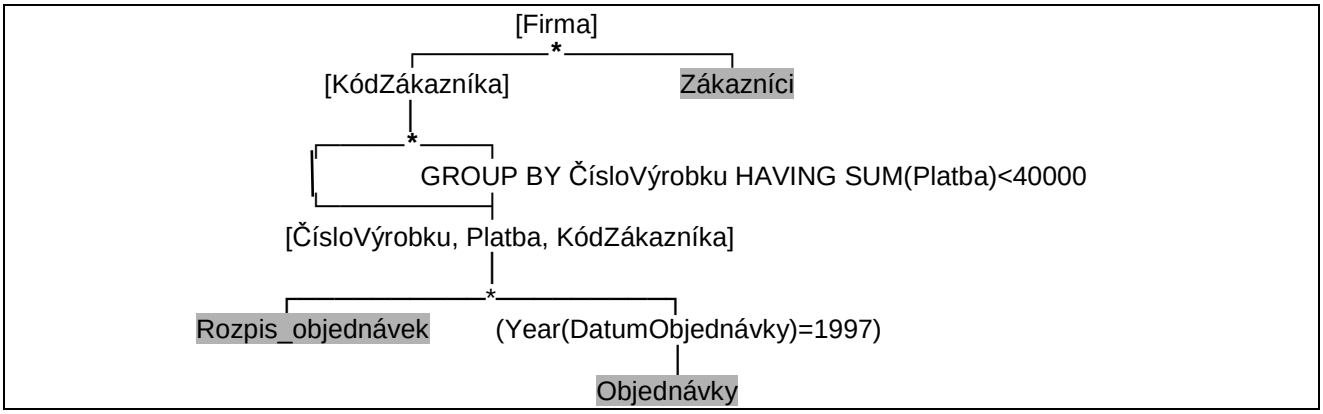
```
CREATE TEMPORARY TABLE pom1 SELECT DISTINCT pom.KódZákazníka FROM
(SELECT ČísloVýrobku FROM pom GROUP BY ČísloVýrobku HAVING SUM(Platba)<40000) A INNER
JOIN pom ON A.ČísloVýrobku=pom.ČísloVýrobku);

DROP TABLE pom;

SELECT Z.Firma FROM pom1 INNER JOIN Zákazníci Z ON pom1.KódZákazníka=Z.KódZákazníka;

DROP TABLE pom1;
```

Analogií stromu dotazu je síť:



PŘEHLED OSTATNÍHO JAZYKA SQL

V této kapitole je podán pouze základní přehled jazyka s hrubým náznakem syntaxe. Konkrétní syntaxe a možnosti příkazů se liší u jednotlivých dodavatelů, čtěte v dokumentaci k SŘBD, který používáte. – Výklad v této kapitole je doplněn příklady.

Jazyk definice dat

Základní datové typy

Jsou textové řetězce pevné délky (doplňují se mezerami), textové řetězce variabilní délky, přesná čísla s pevnou desetinou čárkou, nepřesná čísla s plovoucí desetinnou čárkou, datum (event. datum a čas), logické hodnoty.

SŘBD často podporují ještě další datové typy, např. typ vyjmenovaných hodnot a další.

Ve všech datových typech je možno vložit prázdnou hodnotu Null.

Definice domény[20]

CREATE DOMAIN *jméno_domény definice_domény*;
(například
CREATE DOMAIN Znamka INT CHECK VALUE >=1 AND VALUE <=4)

Definice tabulky

CREATE TABLE *jméno_tabulky* (*seznam_polí_a_jejich_typů_s_eventuelními_omezeními*);
(Lze použít i definované domény pro typ pole.)

Změna definice tabulky

ALTER TABLE …;
(přidání pole, zrušení pole, změna pole, přidání indexu, přidání omezení)

Zrušení tabulky

DROP TABLE *jméno_tabulky*;

Definice indexu

CREATE [UNIQUIE] INDEX *jméno_indexu* ON *jméno_tabulky* (*pole* [ASC|DESC],…);

Index lze definovat i v rámci definice tabulky.

Zrušení indexu

DROP INDEX *jméno_indexu*;

---

[20] Tento příkaz není většinou SŘBD podporován. Variantou je CREATE TYPE, o něm viz kapitolu OBJEKTOVĚ-RELAČNÍ DATABÁZE.

Vytvoření pohledu

CREATE VIEW *jméno_pohledu* AS SELECT…;

Zrušení pohledu

DROP VIEW *jméno_pohledu*;

Odvozené tabulky

CREATE TABLE *jméno_tabulky* AS SELECT…;

Temporární tabulky

CREATE [GLOBAL] TEMPORARY TABLE *jméno_tabulky* …;

Konstrukty pro vyšší úrovně organizace dat

CREATE DATABASE …; DROP DATABASE …; CREATE SCHEMA …; DROP SCHEMA …;

Integritní omezení

Mohou být pojmenovaná (definice začíná CONSTRAINT jméno_omezení) či nepojmenovaná. Pojmenovaná omezení lze vypínat a zapínat.

Na 1 pole

NOT NULL
UNIQUE
CHECK (*podmínka_na_pole*)
DEFAULT *hodnota*
PRIMARY KEY
REFERENCES *jméno_tabulky* [(*jméno_pole*)] [ON DELETE …][ON UPDATE…]

Na řádek tabulky

UNIQUE (*seznam_polí*)
PRIMARY KEY (*seznam_polí*)
CHECK (*podmínka_v_polích_tabulky*)
FOREIGN KEY (*seznam_polí*) REFERENCES *jméno_tabulky* [(*seznam_polí*)] [ON DELETE …][ON UPDATE…]

Poslední možnost uvedená u omezení na 1 pole i na řádek se nazývá *referenční integrita*. Ta značí, že SŘBD nedovolí do polí s cizím klíčem vložit hodnotu, jež není hodnotou v nějakém řádku v rodičovské tabulce (uvedené za REFERENCES) v poli uvedeného jména. Je-li jméno pole v rodičovské tabulce vynecháno, myslí se jím klíč této tabulky. Volby ON DELETE a ON UPDATE specifikují, co se má dělat v případě smazání řádku v rodičovské tabulce, na nějž se odkazují nějaké řádky podřízené tabulky, a co se má dělat při změně hodnoty v řídícím poli v nařízené tabulce. Například definice tabulky Dopis v příkladu Diskusní kluby bude (v jazyce MySQL), vysvětlení následuje:

CREATE TABLE Dopis (IDdopisu INT AUTO_INCREMENT PRIMARY KEY, Odesílatel INT REFERENCES Osoba (Login) ON DELETE SET NULL ON UPDATE CASCADE, Subjekt CHAR(30), INDEX(Subject), TextDopisu TEXT NOT NULL, DatumOdeslání TIMESTAMP NOT NULL)

Vysvětlení: hodnota vkládaná do pole Odesílatel, pokud není Null, musí být k nalezení v nějakém řádku tabulky Osoby v poli Login. Pokud by byl smazán řádek v tabulce Osoby, pak ve všech

řádcích tabulky Dopis, ve kterých hodnota pole Osoba byla hodnotou pole Login ve smazaném řádku, se hodnota pole Osoba změní na Null. Pokud by se v tabulce Osoba v nějakém řádku změnila hodnota pole Login, změní se kaskádovitě i v příslušných řádcích tabulky Dopis.

Další možnosti pro ON DELETE jsou:
RESTRICT (to je default, není nutno uvádět – znamená to zákaz mazání řádků, které mají nějaké podřízené řádky, tedy zákaz mazání Osob, pokud by měly nějaké dopisy)
SET NULL
SET DEFAULT (vloží se defaultní hodnota definovaná pro cizí klíč)
CASCADE (odstraní se i podřízené řádky, tedy když smažeme osobu, smazaly by se i její dopisy)
NO ACTION (nic se neudělá)

Další možnosti pro ON UPDATE jsou stejné jako u ON DELETE, ačkoli některé SŘBD umožňují pouze některé z nich.

Mezi tabulkami

Integritní omezení mezi tabulkami lze realizovat pomocí triggerů, viz dále v této kapitole.

Definice tabulek pro příklad Northwind

Kód je v jazyce MySQL:

```
CREATE TABLE Zakaznici (KodZakaznika CHAR(5) PRIMARY KEY,Firma CHAR(40) NOT
NULL,INDEX(Firma),KontaktniOsoba CHAR(30),Funkce CHAR(30),Adresa CHAR(60),Mesto
CHAR(15),INDEX(Mesto),Region CHAR(15),INDEX(Region),PSC CHAR(10),INDEX(PSC),Zeme
CHAR(15),Telefon CHAR(24),Fax CHAR(24));

CREATE TABLE Zamestnanci (CisloZamestnance INT UNSIGNED PRIMARY KEY, Prijmeni CHAR(20)
NOT NULL,INDEX(Prijmeni),Jmeno CHAR(10) NOT NULL,Funkce CHAR(30),FormalniOsloveni
CHAR(25),DatumNarozeni DATE, ZamestnanOd DATE, Adresa CHAR(60),Mesto CHAR(15),Region
CHAR(15),PSC CHAR(10),INDEX(PSC),Zeme CHAR(15),TelefonDomu CHAR(24),Linka
CHAR(4),Poznamky TEXT,Nadrizeny INT UNSIGNED);

CREATE TABLE Kategorie (CisloKategorie INT UNSIGNED PRIMARY KEY,NazevKategorie
CHAR(15),INDEX(NazevKategorie),Popis TEXT);

CREATE TABLE Dodavatele (CisloDodavatele INT UNSIGNED PRIMARY KEY,Firma CHAR(40) NOT
NULL,INDEX (Firma),KontaktniOsoba CHAR(30),Funkce CHAR(30),Adresa CHAR(60),Mesto
CHAR(15),Region CHAR(15),PSC CHAR(10),INDEX (PSC),Zeme CHAR(15),Telefon CHAR(24),Fax
CHAR(24));

CREATE TABLE Vyrobky (CisloVyrobku INT UNSIGNED PRIMARY KEY,NazevVyrobku CHAR(40) NOT
NULL,INDEX(NazevVyrobku),CisloDodavatele INT REFERENCES
Dodavatele,INDEX(CisloDodavatele),CisloKategorie INT UNSIGNED REFERENCES
Kategorie,INDEX(CisloKategorie),MnozstviVJednotce CHAR(20),JednotkovaCena DEC(17,2)
UNSIGNED,JednotkyNaSklade SMALLINT UNSIGNED ,ObjednanoJednotek SMALLINT
UNSIGNED,MinimalniUroven SMALLINT UNSIGNED,NakupUkoncen BIT DEFAULT 0);

CREATE TABLE Prepravci (CisloPrepravce INT UNSIGNED PRIMARY KEY, Firma CHAR(40) NOT
NULL,Telefon CHAR(24));

CREATE TABLE Objednavky (CisloObjednavky INT UNSIGNED PRIMARY KEY,KodZakaznika CHAR(5)
REFERENCES Zakaznici ON DELETE SET NULL ON UPDATE
CASCADE,INDEX(KodZakaznika),CisloZamestnance INT REFERENCES
```

Zamestnanci,INDEX(CisloZamestnance),DatumObjednavky DATE,INDEX(DatumObjednavky),DodatDne DATE,DatumOdeslani DATE,INDEX(DatumOdeslani),Preprava INT REFERENCES Prepravci,Dopravne DEC(17,2),JmenoPrijemce CHAR(40),AdresaPrijemce CHAR(60),MestoPrijemce CHAR(15),RegionPrijemce CHAR(15),PSCPrijemce CHAR(10),INDEX(PSCPrijemce),ZemePrijemce CHAR(15));

CREATE TABLE RozpisObjednavek (CisloObjednavky INT UNSIGNED NOT NULL REFERENCES Objednavky ON DELETE CASCADE,CisloVyrobku INT UNSIGNED NOT NULL REFERENCES Vyrobky,JednotkovaCena DEC(17,2) UNSIGNED,Mnozstvi INT UNSIGNED,Sleva FLOAT UNSIGNED, CHECK Sleva>= 0 AND Sleva<=1,PRIMARY KEY (CisloObjednavky,CisloVyrobku));

Jazyk manipulace dat

Přidávání záznamů

INSERT INTO *jméno_tabulky* [(*jméno_pole*, …)] VALUES (*hodnota*, …);

INSERT INTO *jméno_tabulky* [(*jméno_pole*, …)] SELECT …;

– Výsledek SELECTu se připojí na konec tabulky.

Aktualizace záznamů

UPDATE *jméno_tabulky* SET *pole=hodnota*,… WHERE *podmínka;*

Mazání záznamů

DELETE FROM *jméno_tabulky* WHERE *podmínka*;

Triggery

Trigger je svázán vždy s jednou tabulkou, a s jednou z operací INSERT, DELETE, UPDATE buď na vybrané sloupce nebo na celé řádky. Dále je svázán buď s BEFORE nebo AFTER situací aktualizace.

Může mít navíc spouštěcí podmínku.

Podstatou triggeru je akce, která se provede při spouštějící události. Akce může být spouštěna pro každý řádek (FOR EACH ROW), který splňuje spouštěcí podmínku, nebo jednou pro celý příkaz spouštějící trigger (FOR EACH STATEMENT -implicitní).

CREATE TRIGGER *jméno_triggeru*
{BEFORE|AFTER} {DELETE|INSERT|UPDATE [OF *pole1*[,*pole2*…]]} ON *tabulka*
[FOR EACH ROW [WHEN (*podmínka*)]]
SQL_blok;

SQL_blok může být volání uložené procedury, o uložených procedurách viz dále.

Příklad triggeru pro databázi Northwind v jazyce ORACLE (používá specifický kód pro identifikaci nové a staré hodnoty):

CREATE TRIGGER objednej AFTER UPDATE OF JednotkyNaSklade ON Vyrobky
REFERENCING NEW AS Nový
WHEN Nový.JednotkyNaSklade<Nový.MinimalniUroven+Novy.ObjednanoJednotek
FOR EACH ROW
CALL Objednej(Nový.CisloVyrobku, Nový.MinimalniUroven+Novy.ObjednanoJednotek-Nový.JednotkyNaSklade, Novy.ObjednanoJednotek);

Integritní omezení s telefony v příkladu IS soukromé VŠ lze realizovat:

```
CREATE TRIGGER telefony1
BEFORE UPDATE OF ČísloTel ON Učitel
REFERENCING OLD AS Původní
                    NEW AS Nový
WHEN Nový.ČísloTel NOT IN
                              (SELECT T.ČísloTel FROM Kancelář K INNER JOIN Telefon T
                                 ON K.ČísloKanceláře=T.ČísloKanceláře
                                WHERE K.KódKanceláře=Nový.KódKanceláře)
FOR EACH ROW
CALL To_nejde("Takový telefon není v dané kanceláři!");
```

Při vkládání nového učitele by šlo o obdobný trigger pro akci událost INSERT.

Kromě jiného použití lze pomocí triggerů aktualizovat uložené odvozené datové objekty.
V SŘBD bývá omezeno povolené množství triggerů na jednu tabulku.

Uložené procedury

Některé SŘBD umožňují ukládat nejen data v předem definovaných logických strukturách, ale i procedury. Tyto procedury lze psát v SQL (poslední verze standardu SQL jej doplnila na výpočetně úplný jazyk, tedy obsahuje příkazy pro řízení běhu programu, přiřazovací příkazy…) nebo v nějakém programovacím jazyce, jejž SŘBD za tímto účelem podporuje. SŘBD tzv. *uložené procedury* ukládá v kompilované podobě do přiděleného adresního prostoru, spravuje uložené procedury stejně jako data. Výhodou takovýchto uložených procedur je to, že jsou spouštěny na stejném počítači, na kterém jsou i data, a tedy jsou prováděny velmi rychle. Navíc v nich může být implementována logika dat či manipulací s daty, kterou mohou sdílet všechny aplikace.

CREATE PROCEDURE *jméno_procedury* ([IN | OUT | INOUT *jméno_parametru typ_parametru*],…)
deklarace_a_definice
tělo_procedury

Tělo procedury může být napsáno v SQL nebo být specifikací externí procedury napsané v jiném programovacím jazyce.

Variantou jsou definované funkce, jejich syntaxe je zhruba takováto:

CREATE FUNCTION *jméno_funkce* (*jméno_argumentu typ_argumentu*,…)
RETURNS *typ_návratové_hodnoty*
deklarace_a_definice
tělo_funkce

Tělo funkce obsahuje příkaz RETURN následovaný návratovou hodnotu. Výhodou definovaných funkcí je, že je lze použít v SQL příkazech na místě výrazů, například v SELECT v klauzuli WHERE.

Tvorba aplikací přistupujících k databázi

Pokud má aplikace komunikovat s databází a zadávat jí příkazy, musí být ošetřeny dvě event. tři záležitosti. První z nich je připojení k databázi. Standard jazyka SQL popisuje příkazy pro připojení a odpojení, konkrétní SŘBD se v syntaxi těchto příkazů příliš neodlišují. Pro připojení je třeba předat základní informace: identifikace databázového serveru, ke kterému chcete aplikaci připojit, uživatelské jméno a heslo.

Druhou záležitostí je předání příkazů SŘBD.

Třetí záležitost se týká specifické problematiky SQL SELECTu, jenž navrací tabulku hodnot. Běžné programovací jazyky ovšem nemají prostředky pro práci s tabulkami, a navíc často předem nevíme, kolik řádků bude návratová tabulka obsahovat. Touto záležitostí se zabývá odstavec nazvaný Kurzory dále. Někdy můžeme použít „mezipřekladiště“, realizované na klientském počítači pomocí k tomuto účelu vytvořeného rozhraní.

Jak zadat databázové příkazy v aplikaci?

Možnosti jsou zhruba dvě. První, starší, je použití tzv. embedded SQL v kódu vaší aplikace, a před kompilací vaší aplikace překladačem vašeho programovacího jazyka použití předkompilátoru, jenž SQL příkazy ve vašem programovém kód přeloží. K tomu účelu musí být SQL příkazy ve vaší aplikaci specielně vyznačeny, pro daný druh předkompilátoru je toto vyznačení specifické (většinou určitý prefix SQL příkazů). V embedded SQL musíte zajistit prostředky, jimiž vám SŘBD bude sdělovat systémové informace doprovázející vaše příkazy (úspěšnost, identifikace chyb…).

Druhá možnost je použít nějaké hotové rozhraní a ovladač pro příslušný SŘBD, potom SQL příkazy zadáváte v syntaxi specifikované v rozhraní. I v takovém případě dostáváte informace o průběhu exekuce vašich příkazů databází.

Specifickou třetí skupinu tvoří programovací jazyky, jež jsou pro spolupráci s databází přímo navrženy, a rozhraní pro komunikaci s databází je v nich zabudováno. Příkladem jsou např. PHP či ASP, nebo jazyky 4.generace dodávané s některými SŘBD.

Kurzory

Pokud databázi zadáte příkaz SELECT, výsledkem je tabulka hodnot, množina záznamů. Tuto tabulku či množinu záznamů běžnými prostředky programovacích jazyků nelze obvykle zpracovat, programy mohou v jeden okamžik přijmout pouze jeden záznam.

K tomuto účelu je v SQL podchycena konstrukce tzv. *kurzorů*, datových objektů sloužících k dočasnému uchování výsledku dotazu. Tímto kurzorem se pak lze pohybovat záznam za záznamem a předávat je proměnným či objektům aplikačního programu. Některé kurzory lze procházet pouze od prvního záznamu vpřed, v jiných se lze vracet či specifikovat kolikátý záznam právě chcete – záleží na možnostech, jež daný SŘBD pro kurzory poskytuje.

SQL příkazy pro práci s kurzory jsou

DECLARE *jméno_kurzoru* CURSOR FOR SELECT…;	Toto je neexekutivní příkaz, tím SŘBD pouze předáváte kód SELECTu a jméno kurzoru, jehož prostřednictvím budete požadovat výstedek SELECTu.
OPEN *jméno_kurzoru*;	Tímto zadáváte příkaz pro vytvoření odpovědi na váš SELECT; SŘBD ji nemusí vytvářet celou najednou, může řádky vytvářet dynamicky na základě vašich požadavků. V takovém případě je třeba zvážit, jestli chcete, aby vám předávané hodnoty eventuelně odrážely změny, jež se mezitím v databázi se zdrojovými daty udály.

Tuto rozhodnutí činíte již v DECLARE … CURSOR příkazu v jeho volitelných klauzulích.

FETCH FROM jméno_kurzoru INTO specifikace_proměnných;

Tímto načtete hodnoty dalšího záznamu do proměnných hostitelského jazyka. Variantami jsou přepínače NEXT, PRIOR, LAST, FIRST, ABSOLUTE *pořadí*, RELATIVE *počet_záznamů*…

CLOSE CURSOR jméno_kurzoru;

Tímto se objekt kurzoru uzavře. V některých prostředích musíte ještě explicitně zadat příkaz pro dealokování paměti pro tento kurzor, v takových případech je možno kurzor opět otevřít, pokud ještě nebyl dealokován.

Dynamické SQL

Zvláštní případ tvoří situace, kdy SQL příkaz, který chceme zadat SŘBD v aplikačním programu, neznáme v době psaní programu (není znám v době kompilace), ale až za běhu programu.
V takovém případě je potřeba pro SQL kód příkazu zřídit proměnnou odpovídajícího typu; a překlad obsahu této proměnné musí být proveden za běhu aplikace. U některých SŘBD je nutno explicitně o tento překlad požádat příkazem
PREPARE *jméno_proměnné_s_SQL_kódem*
pak zjistit úspěšnost překladu, a následně přikázat provedení příkazem
EXECUTE *jméno_proměnné_s_SQL_kódem*
V jiných systémech se překlad děje automaticky bez výslovného zásahu programátora. V každém případě je dynamické SQL pomalejší než statické, tj. kompilované zároveň s aplikací.

Příklad 40.
Toto je příklad ukazující použití kurzoru v uložené proceduře.
Z tabulek Objednávky a Rozpis objednávek vytvořte tabulku „Ceny“ zobrazující historii cen výrobků. Tabulka Ceny bude mít tyto sloupce:

Ceny
ČísloVýrobku
JednotkováCena
Od
Do

Jako mezivýsledek si připravíme kurzor:

ČísloVýrobku	DatumObjednávky	JednotkováCena
↓ řazení- 1.klíč	↓ řazení- 2.klíč	

Tento kurzor projdeme odshora dolů a tak získáme řádky tabulky Ceny.

(Dialekt MS SQL Serveru:)

```
CREATE PROCEDURE VytvorCeny AS
IF EXISTS (SELECT * FROM SYSTABLES WHERE TBName='Ceny')    ---toto je pseudokód, konkrétní
                                                              zjištění existence tabulky je
                                                              v jednotlivých SŘBD
                                                              specifické
        BEGIN
```

```
                PRINT ('Tabulka cen už existuje!')
                RETURN –6                                   ---návrat s určením chybového kódu
        END
DECLARE @cv1 INT, @cv2 INT, @d1 DATE, @d2 DATE, @c1 MONEY, @c2 MONEY
DECLARE @do DATE
DECLARE c CURSOR FOR
        SELECT ČísloVýrobku, DatumObjednávky,JednotkováCena
        FROM Objednávky A, RozpisObjednávek B
        WHERE A.ČísloObjednávky=B.ČísloObjednávky
        ORDER BY 1,2
CREATE TEMPORARY TABLE Ceny (ČísloVýrobku INT,JednotkováCena MONEY, Od DATE, Do DATE)
OPEN c
FETCH c INTO @cv1, @d1, @ce1
SET @do=@d1
FETCH c INTO @cv2, @d2, @ce2
WHILE  (@@FETCH_STATUS=0)
        BEGIN
                IF (@ce1<>@ce2)OR(@cv1<>@cv2) BEGIN
                        INSERT INTO Ceny VALUES (@cv1, @ce1, @d1, @do)
                        SET @cv1=@cv2
                        SET @d1=@d2
                        SET @ce1=@ce2
                END
                SET @do=@d2
                FETCH c INTO @cv2, @d2, @ce2
                CONTINUE
        END
INSERT INTO Ceny VALUES (@cv1, @ce1, @d1, @do)
CLOSE c
DEALOCATE c
RETURN 0
GO
```

Příklad 41.

Toto je příklad řešící alternativu k předchozímu zadání.
Persistentní tabulku Ceny budeme vytvářet postupně, změny budeme zaznamenávat vždy, když se změní JednotkováCena v tabulce Výrobky.

Inicializace tabulky Ceny:

```
CREATE TABLE Ceny (ČísloVýrobku INT,JednotkováCena MONEY, Od DATE, Do DATE);
INSERT INTO Ceny SELECT ČísloVýrobku, JednotkováCena,GETDATE(),Null FROM Výrobky;
```

Aktualizace cen:
(Dialekt SQL ORACLE:)

```
CREATE TRIGGER AktualizujCeny
        AFTER UPDATE OF JednotkováCena ON Výrobky
        REFERENCING NEW AS Nový OLD AS Starý
        FOR EACH ROW
        WHEN Nový.JednotkováCena<>Starý.JednotkováCena
        BEGIN
                UPDATE Ceny SET Do=GETDATE() -1 WHERE
                        Ceny.ČísloVýrobku= Starý.ČísloVýrobku
                        AND Ceny.JednotkováCena= Starý.JednotkováCena
                        AND Ceny.Do IS NULL;
                INSERT INTO Ceny VALUES (Starý.ČísloVýrobku, Nový.JednotkováCena,
                        GETDATE(), NULL);
        END;
```

Transakce a zámky

Při konkurenčním zpracování stejných dat různými aplikacemi mohou nastat kolize, jimž je nutno předcházet. Ochrana před nežádoucími zásahy do zpracovávaných dat je založena na zámcích. Ve starším způsobu zpracování byly zámky přikazovány explicitně, většina SŘBD nadále podporuje tuto možnost různými variantami příkazu LOCK. Zamknout lze v principu celé tabulky, či jejich části (řádky, množiny řádků…), zámek může být pro čtení či pro zápis. Explicitně přikázaný zámek je buď možno opět explicitně uvolnit příkazem UNLOCK, nebo zámek implicitně uvolňuje SŘBD za pro něj specifických podmínek.

Zamykání však není třeba dělat explicitně, ke stejné ochraně před konkurencí slouží konstrukce *transakcí*. Transakce jsou navíc jednotkou obnovy databáze při haváriích.

Tedy, v aplikačním kódu můžete explicitně oznámit začátek transakce, pak zadávat příkazy k manipulaci s daty, eventuelně zjišťovat zda nedošlo k nějaké chybě, a pokud ano, zadat příkaz k navrácení stavu databáze před počátek transakce, po úspěšném dokončení všech manipulací, jež potřebujete v rámci transakce provést, transakci ukončit. Obvyklé příkazy jsou

BEGIN TRANSACTION	oznámení začátku transakce
ROLLBACK	příkaz k navrácení stavu databáze před začátek transakce
COMMIT	potvrzení transakce
END TRANSACTION	ukončení transakce

SŘBD tyto příkazy či jejich varianty podporují v různé míře, někdy jsou počátek a konec transakce rozpoznávány automaticky.

SŘBD také vytváří implicitní transakce, jsou jimi např. všechny SQL příkazy zadávané z příkazového řádku.

Na počátku transakce uzamkne SŘBD všechny datové zdroje, jež bude transakce zpracovávat, a to buď pouze pro čtení, nebo i pro zápis. Zdroje uzamčené pro čtení mohou číst i jiné aplikace připojené k databázi, zdroje uzamčené pro zápis jsou dostupné výlučně aplikaci, jež je uzamkla. Druh zámku volí SŘBD podle typu manipulačních příkazů v transakci. Po úspěšném ukončení transakce jsou všechny její zámky uvolněny.

Transakce jsou zároveň jednotkou obnovy databáze: všechny změny dat jsou prováděny do záložních datových objektů, při UPDATE příkazech jsou zaznamenávány původní i nové hodnoty. Technologie provádění tohoto zabezpečení jsou proprietární SŘBD, bývají zajišťovány různými žurnály, i několika různými žurnály pro různé účely. Teprve po potvrzení transakce jsou změny skutečně zapisovány do databáze, a teprve potom jsou uvolněny všechny zámky transakce a ta je ukončena. Pokud někdy mezitím dojde k přerušení vinou havárie, zaviněním jiných transakcí[21] či příkazem ROLLBACK, je stav databáze navrácen (po eventuelním znovunastartování systému) do

[21] Zavinění může být i explicitním zadáním příkazu k uzamčení dat. Pak může dojít k tzv. uváznutí (deadlock), kdy několik transakcí v kruhu čeká každá na následující. SŘBD běžně taková uváznutí detekují, a pak musí některé z transakcí násilně ukončit a vrátit (ROLLBACK). Pokud jde o automatické zámky, SŘBD běžně používají tzv. bezpečné rozvrhy, které uváznutí znemožňují.

původního stavu před započetím nepotvrzených transakcí, a potvrzené neuložené transakce jsou zrekonstruovány[22].

Pro uložení dosavadních manipulací transakce ale nikoli její ukončení (tj. neuvolnění zámků) je možno eventuelně použít varianty příkazu SAVEPOINT. Potom je následně možno zadat příkaz ROLLBACK TO SAVEPOINT.

Vlastnosti ACID

Ideálním teoretickým požadavkem je, aby transakce měly tzv. vlastnosti ACID – atomicity (transakce je provedena buď celá nebo nic), consistency (před transakcí a po transakci je databáze v konzistentním stavu), isolation (v průběhu transakce ostatní aplikace nevidí změny s daty, jež transakce provádí), durability (změny s daty provedené potvrzenou transakci jsou trvalé). Zejména vlastnost isolace, tj. nezávislosti transakcí, je relativizována. V mnoha SŘBD je možno nastavovat různé úrovně transakcí, obvyklý příkaz s běžnými přepínači tomuto odpovídající je

SET TRANSACTION ISOLATION LEVEL [READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE]

v němž READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE znamenají právě různé úrovně izolace, od nejnižší po nejvyšší. Jde o to, zda jiné transakce mohou číst nepotvrzená data nebo provádět neopakovatelná čtení či získat data, jež jiná probíhající transakce smaže. Úroveň SERIALIZABLE je nejvyšší úrovní, ve které jsou transakce skutečně izolovány. Proč volit jiné stupně izolace? Důvodem je rychlost zpracování – transakce zpomalují zpracování dat.

V některých případech je kontrola transakcí skutečně na obtíž, je to tehdy, když najednou vkládáme do databáze obrovská množství dat. SŘBD běžně pro tyto případy poskytují prostředky pro urychlení spočívající ve zmírnění protokolování transakcí.

Příklad 42.
V příkladu Northwind by mohlo konkurenční objednávání výrobků způsobit, že by několika zákazníkům byla potvrzena objednávka výrobků, jež jsou na skladě, přičemž by nebylo možno uspokojit všechny (na tom, zda je výrobek na skladě či nikoli, může záviset lhůta dodání sjednávaná při objednávce). Řešením je transakční zpracování objednávek, kdy změny hodnot pole NaSkladě v tabulce Výrobky a přidání řádků do tabulek Rozpis_objednávek a Objednávky by byly prováděny v celcích transakcí.

V příkladě Knihovna půjde o vyřizování objednávek Titulů.

V příkladě Diskusní kluby jde o konkurenční aktualizaci pole PočetReakcí v tabulce Příspěvek.

Příklad IS soukromé VŠ ilustruje stav IS po zápisu studentů do kurzů. Pokud by byl tento příklad modifikován tak, aby zachycoval i průběh zápisů, pak by přicházela do úvahy konkurence studentů při zápisu na jednotlivé kurzy (podle rozvrhu má kurz určitou kapacitu danou učebnou, ve které se má vyučovat).

[22] Vidíme, že i zápis změněných dat potvrzené transakce musí používat svůj žurnál.

Explicitně definované transakce je vhodné omezovat na nejnutnější minimum, aby nedocházelo ke zbytečně dlouhodobým uzamčením dat. Extrémně nevhodné je vytvářet transakce, v jejichž průběhu se čeká na vstup dat od uživatelů.

Ochrana dat

Tato sekce pojednává o ochraně před neoprávněným přístupem. Ochrana před poruchami je vysvětlena v sekci Transakce a zámky.

Ochrana před neoprávněným přístupem je založena na systému uživatelských jmen a hesel, jimž jsou přiřazena přístupová práva k jednotlivým objektům databáze. Práva jsou standardně: právo číst, měnit, přidávat a mazat. SŘBD mohou rozlišovat ještě další práva. Standardním SQL příkazem je

GRANT *právo* TO *uživatel* ON *zdroj* [WITH GRANT OPTION]

právo pro případ UPDATE může být specifikováno na vyjmenovaná pole
[WITH GRANT OPTION] znamená právo přidělit některá ze svých práv jiným uživatelům

zdroj je nejčastěji tabulka nebo pohled. Tvůrce tabulky k ní má běžně všechna přístupová práva.

SŘBD většinou poskytují pohodlnější prostředky pro správu databáze, jimiž lze práva přidělovat. Bývá možné definovat i skupiny uživatelů a jim přidělit práva, a pak jednotlivé uživatele zařazovat do skupin.

Práva a pohledy

SŘBD mohou poskytovat selektivní přístupová práva i k jiným objektům, než jsou tabulky, např. sloupce, nebo naopak celé databáze. Pokud jde o menší struktury, než je tabulka, je zde možnost přidělovat parciální práva tak, že definujeme pohled na určitou část tabulky, a k němu přidělíme přístupová práva. Takto lze přidělit práva pouze k určitým řádkům či sloupcům.

Příklad 43.
V příkladu IS soukromé VŠ mohou mít učitelé přístup pouze k polím UČO, Jméno a e-mail v tabulce Osoba v řádcích, jež jsou studenty:

```
DEFINE VIEW Studenti_veřejné AS SELECT  UČO, Jméno, email FROM Osoba WHERE UČO IN (SELECT
A.UČO FROM Student);

GRANT SELECT TO učitel ON Studenti_veřejné;
```

přičemž učitel bude skupina uživatelů. Do tabulky rozvrh bude mít všechna práva pouze rozvrhová komise:

```
GRANT ALL PRIVILEGES TO rozvrhkom ON Rozvrh WITH GRANT OPTION;
```

Rozvrhová komise bude moci delegovat svá práva jiným uživatelům (např. pro případ zastoupení v nemoci). Přístupová práva lze i odebrat:

REVOKE INSERT, UPDATE, DELETE TO *zástupce* ON Rozvrh;

Tímto příkazem zástupce rozvrhové komise přišel o práva měnit rozvrh, nadále ho však může číst.

Literatura

[Codd] Codd E.F.: A Relational Model for Large Shared Data Banks, CACM, 13,6,1997

[Coop] Cooper R.: Object Databases: An ODMG Approach, International Thompson Computer Press, 1997

[Drb1] Drbal P. a spolupracovníci: Objektově orientované metodiky a technologie, 1.díl , části modely a analýza, skripta VŠE Praha, 1997

[Drb2] Drbal P. a spolupracovníci: Objektově orientované metodiky a technologie, 2.díl , části design, dokumentace a porovnání, skripta VŠE Praha, 1997

[GCCC] Gregor J., Chlapek D., Chudějová E., Chvalovský, V.: Datová základna, skripta Vysoké školy ekonomické Praha, 1993

[Chen] Chen, P. P. P.: The Entity-Relationship Model: Toward a Unified View of Data, ACM TODS, 1,1,1976, pp.9-36

[Chud] Chudějová E.: SQL, Strukturovaný dotazovací jazyk a jeho implementace v databázovém sysytému PROGRESS, skripta Vyskoké školy ekonomické, Praha, 1993

[PH92] Pokorný J., Halaška I.: Databázové systémy, Vybrané kapitoly a cvičení, ČVUT, 1992

[PH97] Pokorný J., Halaška I.: Databázové systémy, Cvičení (Jazyk SQL a systém ORACLE), skripta ČVUT, Fakulta elektrotechnická, Praha 1997

[Pok94] Pokorný J.: Dotazovací jazyky, Science, Veletiny, 1994

[Pok97] Pokorný J.: Základy implementace souborů a databází, skripta Univerzita Karlova, Karolinum Praha, 1997

[Pok98] Pokorný J.: Databázová abeceda, Science, Veletiny, 1998

[Tiet] Tietze P.: Strukturní analýza, Úvod do projektu řízení, Grada 1992

[TLM] Telnarová Z., Lukasová A., Matula P.: Úvod do databází, skripta Ostravské university, Přírodovědecká fakulta, Ostrava 1999