ÚSTAV INFORMAČNÍCH STUDIÍ A KNIHOVNICTVÍ
FF UK V PRAZE

Jaromír Skřivan

Datové modely a návrhy relačních schémat

Verze 1.0

Praha

Listopad 2008

Obsah

Úvod

Databáze je pojem, se kterým se setkáváme dnes a denně. Všude kolem nás, kam jen pohlédneme, máme možnost vidět nějakou databázi. Jdeme navštívit banku, u které máme vedený svůj bankovní účet, nebo jdeme k lékaři, že nás bolí v krku. Jdeme si vybrat a zaplatit zájezd do cestovní kanceláře a nebo stojíme ve frontě u pokladny našeho oblíbeného hypermarketu.

Zorientovat se dnes ve světě databázových systémů není vůbec jednoduché. Vznik a počátek vývoje databází se datuje do první poloviny 60. let 20. století a od té doby se jak systémy, tak i standardy pro popis a práci s databázemi neustále vyvíjí, vylepšují a rozšiřují. Dnes lze vysledovat nasazení databázových systémů i tam, kde by nás to před 30 lety nenapadlo ani v tom nejdivočejším snu.

Tento učební text slouží jako pomůcka pro posluchače Vyšší odborné školy informatiky a knihovních služeb v Brně, kteří v rámci svého studia navštěvují předmět *Báze dat*. Jeho cílem je seznámit čtenáře s problematikou návrhu struktury relačních databází a použití databázových systémů.

Při zpracování tohoto učebního textu bylo předpokládáno, že čtenář má elementární znalosti práce se základními programy na PC, služeb Internetu a základy středoškolské matematiky.

Čtenář je nejprve seznámen se základními pojmy ze světa databází. Další část je věnována základům datového modelování, které je důležité při řešení úloh návrhu databázových základen. Dále pak navazují části pojednávající o relačních schématech, tabulkách, sloupcích, integritních omezeních a normálních formách. Předposlední kapitola nabízí jemný úvod do relační algebry, což je jednoduchý matematický aparát, který lze využít pro formulaci dotazů nad tabulkami v databázi. Dotazování nad daty je pak dovršeno poslední kapitolou pojednávající o standardu jazyka SQL.

Odborný výklad je doplněn řadou řešených praktických úloh, které slouží pro lepší pochopení probírané problematiky. Každá kapitola je ukončena stručným shrnutím a zadáním příkladů pro samostatnou práci posluchačů, aby si sami mohli ověřit nabyté znalosti dané kapitoly.

V Brně, dne 12. března 2006

Jaromír Skřivan

Základní pojmy

Dříve, než se pustíme do konkrétních úloh řešící návrhy databázových struktur, je potřeba se zorientovat v základních pojmech, které budeme používat. Těmi jsou především pojem databáze, databázový systém a databázový model.

Databáze

Databází rozumíme soubor (množinu, kolekci) dat, která nám slouží pro popis reálného světa nebo jeho části. Částí reálného světa máme přitom na mysli nějakou konkrétní úlohu, kterou chceme pomocí databázové evidence řešit, například:

- Střední škola se studenty a učiteli; databáze bude evidovat klasifikaci a docházku studentů
- Hypermarket se zaměstnanci; databáze bude řešit pohyb zboží na skladu
- Letiště s leteckými společnostmi a zákazníky; databáze pro odlety a přílety letadel

Je velmi důležité, aby data (údaje) v databázi byla uspořádána do struktury. Uspořádání dat do struktury umožní efektivní práci s daty. To oceníme, když budeme nějaké údaje v databázi hledat, případně když budeme chtít nějaké údaje zaktualizovat. Pěkné přirovnání bychom mohli nalézt doma s pracovním stolem. Pokud na něm všechny listiny a sešity budeme mít rozčleněné a uspořádané, jistě zde najdeme všechno mnohem dřív a snáz, než kdyby na stole byl totální nepořádek.

Struktura databáze může být následující:

- Lineární/sekvenční (textový soubor)
- Stromová (hierarchický model)
- Grafová (síťový model)
- Tabulky (relační model)

Jednotlivé modely jsou probrány později v této kapitole.

Veškerá data v databázi musí mít svůj datový typ. Datovým typem v tomto kontextu může být např.: celé číslo, textový řetězec, pravdivostní hodnota (**`true`**, **`false`**), binární data (např. obrázek), reálné číslo, datum, a další. Každá položka v databázi musí mít svůj datový typ jednoznačně určen. Podle něj databáze ví, jak má s příslušnou datovou položkou pracovat.

Příklad: rozdílný přístup v třídění čísel a textových řetězců
U číselných hodnot bude databáze provádět číselné uspořádání, zatímco u textových řetězců bude provádět uspořádání abecední (tzv. lexikografické). Že to není tentýž

způsob třídění si můžete snadno ověřit, když si zkusíte číselně i abecedně uspořádat následující trojici čísel: 28, 7 a 10. Výsledek číselného uspořádání je podle očekávání 7, 10, 28, zatímco lexikografické uspořádání dá výsledek: 10, 28, 7.

Volbu datového typu musíme velmi dobře zvážit. Na uvedeném příkladu je vidět, že pokud bychom měli například v databázi položku **`VEK_OSOBY`** a přiřadili ji datový typ textový řetězec, mohli bychom se dostat později do potíží, například při požadavku na setřídění osob podle věku. Jelikož jsme položku **`VEK_OSOBY`** definovali jako text, bude se na ni uplatňovat textové uspořádání, což by bylo z logického hlediska chybné.

Informační systém

Samotná databáze (tj. soubor dat) by nám byl celkem k ničemu, kdybychom k němu neměli žádné softwarové prostředky (programy), abychom mohli s databází nějak rozumně pracovat. Takovým programům se říká *Systémy řízení báze dat*. Ty musí především umožňovat:

- Definovat databázi
- Konstruovat databázi
- Manipulovat s databází

Vezmeme-li tedy databázi a systém pro práci s ní (SŘBD) získáváme databázový systém. Pokud k němu přidáme ještě uživatelské aplikace, které jsou propojeny s takovým databázovým systémem a stanovíme uživatelské rozhraní (např. webové rozhraní – v okně internetového prohlížeče), získáváme konečně informační systém. Informační systém je tedy složen z následujících částí:

- Databázový systém (databáze + systém řízení báze dat)
- Uživatelské aplikace
- Uživatelské rozhraní

Informační systém nám umožňuje tedy přístup k datům a získávání informací z nich.

Data versus informace

Mezi daty a informacemi je zásadní rozdíl. Daty rozumíme konkrétní hodnoty (údaje) uložené v databázi (textové řetězce, čísla, datum, hodnota A/N, a další). Informacemi rozumíme znalosti, vyplývající z hodnot uložených v databázi. Na uvedenou problematiku se můžeme podívat i z jiného úhlu. Data tvoří syntaktickou složku databáze (nebo-li strukturální), informace jsou pak považovány za složku sémantickou (nebo-li významovou). Každá položka v databázi je nějak zapsaná (má svoji strukturu – pozor, nezaměňovat se strukturou celé databáze), např. textový řetězec, číslo, datum, apod. A informace plynoucí z hodnoty položky jsou jejím významem.

Příklad: syntaktický a sémantický pohled na datovou položku
Mějme v databázi položku pod názvem **ZAPLACENO**. Tato položka je typu pravdivostní hodnota. Zde může být uložena hodnota **A** nebo **N**. Nechť v nějakém záznamu pro fakturu máme položku **ZAPLACENO = 'A'**. Když se na toto podíváme ze syntaktického pohledu, položka **ZAPLACENO** obsahuje jedno písmeno **A**. Když se zaměříme na sémantiku uvedeného zápisu, znamená to, že faktura byla zaplacená. Tedy, hodnota **A** jsou data, a tvrzení „faktura byla zaplacena" je informace, která z hodnoty **A** plyne.

Operace nad daty

Doposud jsme si popsali, jak data v databázi vypadají a že s nimi lze nějakým způsobem manipulovat. Nyní si popíšeme základní operace, které můžeme nad daty realizovat. Je jich celkem 5 (4 základní a jedna odvozená). Souboru dat se také někdy říká slovník (katalog), takže se můžete v odborné literatuře setkat s pojmy slovníkové operace, nebo také katalogové operace. Základní operace jsou následující:

- Hledání prvku (**SELECT**): chceme nalézt celý záznam prvku podle zadaných kritérií, zpravidla hledáme podle některých jeho položek (např. člověka budeme hledat podle jména a příjmení), na výstupu pak obdržíme kompletně celý záznam (a přečteme si například, jakou má nalezený člověk adresu).
- Vložení nového prvku (**INSERT**): chceme založit záznam pro nový prvek a vyplnit všechny položky nového záznamu (např. zadáme jméno, příjmení, datum narození a adresu nového člověka).
- Aktualizace existujícího prvku (**UPDATE**): nejprve chceme nalézt existující prvek podle zadaných kritérií a po té, jednu nebo více položek z jeho záznamu aktualizovat (např. podle jména a příjmení najdeme konkrétního člověka a pak změníme jeho aktuální adresu na novou).
- Smazání existujícího prvku (**DELETE**): z nějakého důvodu již nechceme nadále vést záznam nějakého prvku v databázi a chceme jej natrvalo z databáze odstranit.

Odvozená operace:

- Dotaz na existenci prvku (**EXIST**): někdy nám stačí informace, zda-li prvek v databázi existuje, či nikoliv.

Operace nad daty a textová struktura databáze je demonstrována na následující úloze.

Úloha: dětský tábor
Máme seznam účastníků dětského tábora. O každém z nich evidujeme následující údaje: jméno a příjmení, datum narození, zdravotní pojišťovnu, zda-li už má zaplacen pobyt či nikoliv, zda-li musí brát v průběhu pobytu nějaké léky a když ano, tak jaké. Navrhněte textovou databázi (tj. organizaci textového souboru) pro uvedené zadání. Slovně popište, jak by se v takové textové struktuře realizovalo:

- Přihlášení nového účastníka na tábor
- Odhlášení stávajícího účastníka z tábora
- Konkrétní účastník uhradil poplatek za pobyt
- Kolik účastníků celkem se přihlásilo na tábor?

Řešení:
Textovou databází budeme rozumět běžný textový soubor, který jsme schopni vytvořit a později upravovat v běžném textovém editoru. Pro textovou databázi nám tento přístup úplně postačí, nebudeme potřebovat žádný specializovaný databázový systém. Jednotlivé záznamy budou v textovém souboru řazeny za sebou, nebo-li budou tvořit sekvenci. Zbývá tedy vyřešit otázku, jak bude vypadat struktura textového souboru, do které budeme zanášet jednotlivé záznamy o účastnících tábora. Uvedeme zde celkem dvě možná řešení a ke každému z nich krátkou diskuzi, včetně odpovědí na výše položené otázky.

Struktura textového souboru I.
Každý záznam bude uveden na jednom řádku, jednotlivé položky (údaje o účastnících) budou od sebe odděleny tabulátorem. Začátek souboru může vypadat následovně:

```
Jan         Novák       12.10.1992  VZP   A     N
Petra       Horáková    03.04.1993  MV    N     A     riboflavin
Jiří        Dvořák      10.12.1991  VZP   A     A     zyrtec
Jana        Lukášová    15.07.1992  VZP   N     N
```

Když se na obsah souboru podíváme blíže, jsme schopni z něj vyčíst všechny potřebné údaje, které jsou evidovány. Jan Novák má zaplacen pobyt a nemusí brát žádné léky. Petra Horáková pobyt ještě zaplacený nemá (neboť v 5. sloupci má písmeno N) a musí brát léky a to riboflavin. Jiří Dvořák má již zaplaceno a musí brát zyrtec. Jana Lukášová nemá zatím zaplaceno a nemusí brát na táboře žádné léky. Vytvořili jsme tedy textový soubor o vhodné struktuře, která nám umožňuje evidovat všechno, co bylo požadováno v zadání úlohy.

Jak se zrealizuje přihlášení nového účastníka na tábor? Jednoduše. Otevřeme uvedený textový soubor v textovém editoru a přidáme nový řádek s údaji o novém účastníkovi. Jak odhlásíme účastníka z tábora? Opět otevřeme textový soubor a smažeme příslušný řádek. Jak zaevidujeme, že například Petra Horáková už zaplatila poplatek za pobyt? Otevřeme soubor, najdeme řádek, kde je Petra Horáková a v 5. sloupci, kde je

písmeno N, změníme na A. A nakonec, jak zjistíme, kolik účastníků se přihlásilo celkem na tábor? Počet účastníků se rovná počtu řádků textového souboru.

Struktura textového souboru II.

Ukázali jsme se nejjednodušší textovou strukturu, která by jistě hned napadla každého z nás při řešení této úlohy. Struktura textové databáze (neboli textového souboru) může ale vypadat i jinak. Je pochopitelné, že pokud změníme strukturu textového souboru, budou se i jednotlivé operace realizovat odlišně, než tomu bylo u předchozí struktury.

Nyní budeme uvažovat jinou textovou strukturu. Jeden záznam bude „rozprostřen“ do více řádků. Každá položka záznamu bude uvozena jménem položky a znakem rovnítka. Jednotlivé záznamy budou pak ukončeny značkou **#**, která bude zároveň sloužit jako oddělovač jednotlivých záznamů. Vezmeme-li údaje o účastnících tábora z předchozí textové struktury, dostaneme v nové textové struktuře následující výsledek:

```
Jméno=Jan
Příjmení=Novák
Datum_nar=12.10.1992
Pojišťovna=VZP
Zaplaceno=A
Musí_léky=N
#
Jméno=Petra
Příjmení=Horáková
Datum_nar=03.04.1993
Pojišťovna=MV
Zaplaceno=N
Musí_léky=A
Léky=riboflavin
#
Jméno=Jiří
Příjmení=Dvořák
Datum_nar=10.12.1991
Pojišťovna=VZP
Zaplaceno=A
Musí_léky=A
Léky=zyrtec
#
Jméno=Jana
Příjmení=Lukášová
Datum_nar=15.7.1992
Pojišťovna=VZP
Zaplaceno=N
Musí_léky=N
#
```

Jak zapracujeme přihlášení nového účastníka na tábor? Otevřeme textový soubor a podle toho, jak jsou zapsány existující záznamy, přidáme záznam nový. Ukončíme jej znakem **#**. Skutečnost, že Petra Horáková již zaplatila pobyt, zaneseme do této databáze tak, že

najdeme odpovídající „oblast“ pro Petru Horákovou (oblastí rozumíme skupinu řádků ohraničenou na jejím začátku a konci znakem **#**), kde v tomto případě budou řádky **Jméno=Petra** a **Příjmení=Horáková**. Pak v této oblasti najdeme řádek **Zaplaceno=N** a změníme na **Zaplaceno=A**. Smazání záznamu (účastník se odhlásil z tábora) budeme realizovat smazáním příslušné oblasti pro dané jméno účastníka. A konečně, kolik účastníků se přihlásilo na tábor? Počet účastníků je dán počtem znaků **#** v textovém souboru.

Nelze obecně říci, že by nějaké z uvedených řešení bylo správné nebo špatné, či lepší nebo horší. Každé z nich má své vlastnosti a určuje, jakým způsobem se bude s jednotlivými záznamy pracovat. První řešení je jednoduší a nabízí poměrně snadnou manipulaci se záznamy, ale na druhou stranu struktura záznamu je pevně daná a pokud člověk udělá chybu a vymění hodnoty mezi 5. a 6. sloupcem, může to způsobit komplikace související s pozdním nalezením chyby. Druhé řešení naopak nemá pořadí položek v záznamu striktně dané, můžeme klidně jejich pořadí proházet. O jakou položku se jedná, je totiž dáno názvem položky a znakem rovnítko. Jinými slovy, záznam pro Jana Nováka v textové struktuře II, může vypadat například takto:

```
Příjmení=Novák
Jméno=Jan
Pojišťovna=VZP
Musí_léky=N
Datum_nar=12.10.1992
Zaplaceno=A
#
```

Na způsobu, jak realizovat jednotlivé operace nad daty (vložení nového záznamu, smazání existujícího záznamu, aktualizace záznamu), tak jak jsme je uvedli, se nic nezmění.

Databázové modely

Přibližně v polovině 60. let 20. století vzniká pojem databázového modelu. Zavedli jej matematikové jako prostředek právě pro popis databázových struktur. V úvodu celé této kapitoly jsme si uvedli 4 základní struktury databází. Textovou strukturu jsme již popsali a vyřešili v předchozí úloze, nyní zbývá se podívat na ty zbývající:

- Hierarchický model – popisuje stromovou strukturu databáze
- Síťový model – popisuje grafovou strukturu databáze
- Relační model – popisuje strukturu databáze s tabulkami

Hierarchický model

Jak už sám název napovídá, hierarchický model je založen na principu modelování hierarchie mezi entitami se vztahy podřízenosti a nadřízenosti a dědičnosti. Jedná se o nejstarší z databázových modelů. Vznikl v 60. letech 20. století. Lze jej využít pro popis

všech informačních systémů, nebo jejich částí, kde lze nalézt vztahy podřízenosti a nadřízenosti. Řada informačních systémů ale tímto modelem popsat nejde. Nelze například pomocí tohoto modelu popsat všechny možné vazby (vztahy) mezi entitami. Vědci i odborníci si nedostatek tohoto modelu uvědomovali a tak nový databázový model na sebe nenechal dlouho čekat.

Příkladem informačního systému, který lze popsat pomocí hierarchického modelu, je informační systém pro rodokmen.

Síťový model

Jeho počátky se datují do 70. let 20. století. Jedná se o model, který jednotlivé entity organizuje do *grafu*. Graf je matematická struktura, která je tvořena uzly a hranami. Uzly jsou v tomto případě entity informačního systému a hrany mezi uzly jsou dokumentované vazby mezi entitami. Tento model je tedy už výrazně obecnější, než model hierarchický, neboť je schopen dokumentovat i jiné vazby, než jen hierarchické vazby (jedna entita je nadřazena druhé entitě). Nicméně stále tento model není dostatečný, neboť nedokáže popsat všechny vazby, zejména pak vazby, kde jednotlivé entity vystupují jako násobné. Proto se od tohoto modelu postupně upouští a vzniká poslední, relační model, který je hojně využíván dodnes.

Pomocí síťového modelu lze popsat některé geografické informační systémy (GIS). Jde o systémy, kde se zpracovávají geografické údaje (o územích, parcelách, rozvodů inženýrských sítí, apod.). Konkrétním příkladem může být databáze nejkratších tras mezi jednotlivými městy v České republice.

Relační model

Jedná se o poslední, dodnes velmi používaný, databázový model. Řeší nedostatky svých předchůdců. Základním stavebním prvkem je relace (tabulka). Každá entita informačního systému je popsána relací a rovněž každá vazba mezi entitami je opět popsána relací. Soubor relací pak tvoří tzv. relační schéma. Podrobněji tato problematika je probrána v samostatné kapitole věnované relačním schématům a tabulkám.

Do souvislosti se vznikem relačního modelu je dáván také vznik jazyka SQL (*Structured Query Language*), který se stal standardem pro relační databáze. Jeho první verze vznikla v roce 1986, později mírně poupravená a vyladěná verze SQL92 z roku 1992. Konkrétně tato verze je ještě dnes stále hodně používaná v některých databázových systémech. Jazyk SQL je předmětem samostatné kapitoly.

Ostatní databázové modely

Kromě výše uvedených modelů se můžeme ještě setkat s modelem objektovým a modelem objektově-relačním. Oba tyto modely jsou relativně nové, objevily se až

z příchodem *objektově orientovaného programování*, které zaznamenalo nárůst v používání během 90. let minulého století.

Základním prvkem objektového modelu je objekt. Každý objekt má své vlastnosti a metody, kterými přistupuje a manipuluje s hodnotami svých vlastností. Informační systém je pak popsán jako soubor objektů se svými vlastnostmi, metodami a jak se objekty k sobě vzájemně chovají. Tento model je robustnější, než model relační, ale na druhou stranu již poměrně velmi složitý, což ho staví poněkud do nevýhodné pozice. Zřejmě přílišná složitost tohoto modelu způsobila, že neobstál v boji proti jednoduššímu relačnímu modelu. Objektového modelu se tedy příliš neužívá.

Objektově-relační model je vzniklým kompromisem, volně řečeno, tento model rozšiřuje stávající relační model o nějaké dobré a zajímavé vlastnosti modelu objektového. Zůstává mu tedy relativní jednoduchost z relačního modelu a navíc získává „to dobré“ z modelu objektového. Standardem pro objektově-relační databáze je opět jazyk SQL, označován jako SQL2003, který vznikl z původního SQL rozšířením o některé objektové prvky, přičemž je zachována zpětná kompatibilita s původním SQL.

Databázové systémy

Poslední, co nám zbývá uvést v kapitole věnující se základním pojmům, je krátké pojednání o databázových systémech. Co je to databázový systém, již víme, nyní se podíváme jaké databázové systémy lze v současné době najít a jaké mají využití. Zjistíme, že relační databáze, které jsou hlavním předmětem tohoto studijního materiálu, jsou jen malým zlomkem toho, co všechno ve světě databázových systémů můžeme najít.

V současné době můžeme mezi komerčními i nekomerčními databázovými systémy nalézt tyto skupiny systémů:

- Relační databáze
- Objektově orientované databáze (v kombinaci s relačními databázemi)
- Deduktivní databáze
- Temporální databáze

První dvě skupiny databázových systémů jsou dány databázovým modelem, ze kterého vycházejí. Zbylé dvě skupiny jsou určeny zejména oblastí jejich využití.

Relační databáze

Relační databázové systémy vychází z relačního databázového modelu. Jde o nejrozšířenější skupinu databázových systémů, se kterou se dnes můžeme hodně setkat. Například řada informačních systémů a portálů, které jsou dostupné na Internetu, je postavena na relační databázi. Podrobněji této skupině databázových systémů se budeme věnovat v samostatné kapitole.

Do této skupiny patří například následující databázové systémy: FoxPro, Informix, MS Access, MS SQL, MySQL, Oracle, Postgress, SyBase a WinBase602.

Objektově orientované databáze

Vychází z principů objektově orientovaného programování. Základním prvkem těchto databází je objekt. Každý objekt musí mít své vlastnosti a metody pro práci s nimi. Při konstrukcích databází lze využít základních postupů v objektově orientovaném programování, kterými jsou: *zapouzdřenost*, *dědičnost* a *polymorfismus*. Dnes se většinou nejedná o čistě objektově orientované databáze, ale o kombinaci s relačními databázemi.

Příklady objektově orientovaných databázových systémů jsou: O2, CA-Ingres, ODBII, DB2 a Poet.

Deduktivní databáze

Tak jako jsou objektově orientované databáze založeny na principu objektově orientovaného programování, jsou deduktivní databáze založeny na logickém programování. Vlastní struktura databáze je velmi podobná struktuře relační databáze. Co se ale liší, je způsob získávání dat.

Základním stavebním prvkem jsou *extenze* (napevno definované vztahy mezi entitami). A dále potřebujeme *intenze*, neboli pravidla odvozování. Říkáme jim také deduktivní pravidla.

Deduktivních databází se nejvíce využívá pro vědecké účely. Jedná se o aplikace *dataminingu* (dolování dat) a umělé inteligence. Příklady databází, se kterými se zde můžeme setkat jsou: Datalog, Eclipse, LDL.

Příklad: extenze a intenze
V databázi jsou uloženy následující informace (extenze):

```
je_mladsi(Jana,Karel)
je_mladsi(Karel,Honza)
```

A dále je v databázi uloženo následující odvozovací pravidlo (intenze):

```
if je_mladsi(A,B) and je_mladsi(B,C) then je_mladsi(A,C)
```

Toto pravidlo říká: máme-li člověka A, který je mladší než člověk B, a zároveň člověk B je mladší než člověk C, pak dedukcí (odvozením) dostaneme, že člověk A je mladší než člověk C.

Položíme-li následující dotaz:

```
?je_mladsi(Jana,Honza)
```

tak se nám podaří najít takové A, B, C z původní sady extenzí, že zjistíme, že uvedený dotaz nám odpoví: „pravda“.

Temporální databáze

Jedná se o časově závislé databáze. Nasazují se tam, kde je potřeba zachycovat historii dat. Kromě historie také tam, kde je potřeba zpracovávat v jednom okamžiku jak stará, tak i nová data. Předchozí typy databází lze na temporální převést tak, že každý záznam bude opatřen přídavným časovým razítkem. Jde tedy o rozšíření předchozích typů databází o zpracování dat s ohledem na čas. S těmito systémy se setkáme zejména v bankovnictví, pojišťovnictví a podobných oblastech.

Shrnutí

- **Databáze je množina dat sloužící pro popis části reálného světa. Musí mít svou strukturu a dále každý údaj musí být určitého datového typu**
- **Informační systém je komplexní nástroj, který nám umožňuje přístup k datům uložených v databázi a nabízí možnosti, jak z těchto dat získat potřebné informace.**
- **Daty rozumíme konkrétní hodnoty položek v databázi, informace jsou pak znalosti, které lze z uložených dat interpretovat (získat)**
- **Základní operace nad daty jsou: hledání prvku, vložení prvku, aktualizace prvku, smazání prvku. Odvozenou operací je pak dotaz na existenci prvku.**
- **Databázový model je prostředek zavedený matematiky pro popis struktury databáze. Databázovými modely jsou: hierarchický, síťový, relační, objektový a objektově-relační.**
- **Databázové systémy můžeme rozdělit podle použitého modelu (relační, objektově orientované) nebo podle jejich využití (deduktivní a temporální).**

Otázky a úkoly

- Vraťte se k úloze o dětském táboře. Navrhněte další textovou strukturu, která bude řešit uvedené zadání. Diskutujte rovněž jednotlivé operace nad daty v této struktuře a zodpovězte otázku, jakým způsobem se zjistí, kolik účastníků se přihlásilo na tábor.
- Najděte alespoň jeden další příklad informačního systému, který lze popsat hierarchickým databázovým modelem.
- Najděte dva příklady informačních systémů, které lze popsat síťovým databázovým modelem.
- Najděte ze svého okolí tři existující informační systémy, které jsou vybudovány nad relační databází (tj. je použit relační model).
- Řešte následující úlohu z deduktivních databází: mějme následující extenze: `je_rodic(Jana,Karel)`, `je_rodic(Petr,Karel)`, `je_muz(Karel)`, `je_muz(Petr)`, `je_dedecek(Petr,Zuzana)`. A dále 2 odvozovací pravidla: `if je_rodic(X,Y) and je_rodic(Y,Z) and je_muz(X) then`

je_dedecek(X,Z), a **if je_rodic(X,Y) and je_dedecek(X,Z) then je_rodic(Y,Z)**. Otázka zní: **?je_rodic(Karel,Zuzana)**. Zjistěte odpověď.

Datové modelování

Datové modelování je disciplína, která si klade za cíl návrh logického datového modelu. Snaží se nalézt a zkoumá jednotlivé „objekty“ informačního systému (říkáme jim entity) a dále jaké jsou vztahy (vazby) mezi těmito entitami. U každé entity je potřeba přemýšlet o jejích vlastnostech (říkáme jim atributy), které nás zajímají. Dobře vytvořený logický datový model (značíme LDM) je potřebným podkladem pro návrh konkrétní databázové základny.

Dříve, než se pustíme do konstrukce logického datového modelu, podívejme se blíže na jednotlivé pojmy, které jsme zde zmínili.

Entita

Entitou rozumíme objekt našeho pozorování. Tím objektem může být jakákoliv věc (hmotná i nehmotná) z reálného světa, která je předmětem naší „evidence“. Jinými slovy, může jít o jakoukoliv osobu, věc, událost, informaci, kterou potřebujeme evidovat. Evidencí rozumíme potřebu evidovat (vést), např.:

- seznam nějakých osob (např.: zaměstnanců ve firmě)
- seznam věcí (např.: seznam všech vozů, které prodává místní autobazar)
- seznam událostí (např. všechny dopravní nehody na dálnici D1 za nějaký časový úsek)
- apod.

Takovým „seznamům“ dáme název, podle věci, kterou seznam eviduje. Pro výše uvedené příklady bychom mohli zavést následující entity:

- **ZAMĚSTNANEC**
- **AUTOMOBIL**
- **DOPRAVNI_NEHODA**

Objektem našeho pozorování je tedy zaměstnanec ve firmě. Potřebujeme vést evidenci (=seznam) všech takových zaměstnanců, kteří pracují ve firmě. Pro tento účel zavedeme entitu pojmenovanou **ZAMĚSTNANEC**.

Podobně to bude s vozy, které se prodávají v místním autobazaru. Objektem pozorování (tj. to, co potřebujeme evidovat) jsou ty konkrétní automobily, které jsou v nabídce k prodeji v autobazaru. Pro tyto objekty (=automobily) zavedeme entitu **AUTOMOBIL**.

Nakonec seznam všech dopravních nehod. Jde o nehmotnou, neživou věc (dopravní nehoda není ani osoba, ani hmotná věc, jde o událost). Potřebujeme vést

seznam všech takových nehod, neboli dopravní nehody jsou objektem našeho pozorování, a tedy máme pro ně entitu **DOPRAVNÍ_NEHODA**.

Poznámka
I když máme zpravidla na mysli seznam objektů, název entity udáváme v jednotném čísle. Tedy **ZAMĚSTNANEC** a ne **ZAMĚSTNANCI**, nikoliv **AUTOMOBILY**, ale **AUTOMOBIL** a pro dopravní nehody použijeme **DOPRAVNÍ_NEHODA** místo **DOPRAVNÍ_NEHODY**. Proč tomu tak je, bude vyplývat z následující kapitoly věnované instancím entit.

Instance

Nyní, když jsme si zavedli pojem entita, můžeme přejít k dalšímu pojmu. Instancí rozumíme konkrétní (jeden!) výskyt dané entity. Jinými slovy, pokud si entitu představujete jako seznam (viz předchozí kapitola), potom instance jsou konkrétní řádky v seznamu. Každý řádek popisuje jeden konkrétní objekt, který je předmětem našeho pozorování.

Konkrétní instancí entity **ZAMĚSTNANEC** může být například Jan Novák, narozen 12. dubna 1970, bydlící na Dvořákove 14 v Brně. Jinými slovy, Jan Novák je údaj na jednom z řádků seznamu, který eviduje všechny zaměstnance ve firmě. Znamená to také, že Jan Novák je opravdu zaměstnancem ve firmě. Vezmeme-li v úvahu, že Jan Novák má v tomtéž domě souseda Jiřího Dvořáka, který ale není zaměstnanec v uvažované firmě, znamená to, že Jiří Dvořák nebude instancí entity **ZAMĚSTNANEC**. To, jestli je nebo není někdo či něco instancí entity, je dáno obrazem reálného světa. Jinými slovy, na seznam zaměstnanců ve firmě si zapíšeme jen ta jména lidí, kteří ve firmě pracují. Nebudeme tam psát žádná další jména lidí, kteří ve firmě nepracují.

Podobná situace bude i s dalšími instancemi entit **AUTOMOBIL** a **DOPRAVNÍ_NEHODA**. Instancí entity **AUTOMOBIL** bude každý z automobilů, které jsou aktuálně v prodeji v místním autobazaru. Instancemi entity **AUTOMOBIL** ale nebudou všechny automobily, co jezdí po našich silnicích.

S dopravními nehodami je to jednodušší, neboť se do seznamu dostanou jen ty nehody, které se doopravdy staly. Těžko bych tam zapisovali nehodu, která se vůbec nestala. I když někdo by jistě mohl takovou potřebu mít (např. když by se snažil o pojišťovací podvod).

Atributy

Každá entita má svoje vlastnosti. Vlastnosti entity **ZAMĚSTNANEC** mohou být:

- Jméno zaměstnance
- Příjmení zaměstnance

- Datum, kdy se zaměstnanec narodil
- Adresa, na které zaměstnanec bydlí
- Funkce, kterou v naší firmě vykonává
- Plat, který dostává zaměstnanec za práci každý měsíc
- Informace, zdali má uzavřenou smlouvu na dobu neurčitou

Jednotlivým vlastnostem říkáme atributy. Podobně jako entity, jednotlivé atributy jednoslovně pojmenováváme:

- Jméno
- Příjmení
- Datum_narození
- Adresa
- Funkce
- Plat
- Doba_neurčitá

Poznámka: použití znaku „podtržítko“
Jak bylo uvedeno, pro názvy entit a jejich atributů se používají jednoslovné popisy. Nicméně, někdy se nám může hodit použít více slov, proto si v takovém případě pomůžeme znakem „_“, který nám nahradí mezeru mezi slovy. Z pohledu počítače dvě slova spojená znakem „_“ vystupují stále jako jedno slovo.

Pokud si představíme entitu jako seznam z předešlé kapitoly, potom jednotlivé atributy budou položkami seznamu, které budou na každém řádku pro každého zaměstnance vyplněny. Seznam bude obsahovat celkem 6 sloupců, každý sloupec bude určen pro jeden z výše uvedených atributů.

Typ atributu

Pokusíme-li se zamyslet nad tím, jak budou vypadat hodnoty jednotlivých atributů, zjistíme, že například jméno, příjmení, adresa a funkce budou určitě nějaké řetězce znaků české abecedy, datum_narození bude ve formátu datumu (DD.MM.RRRR) a plat bude zřejmě nějaké celé, nezáporné, číslo. Informace, zda-li má zaměstnanec uzavřenou smlouvu na dobu neurčitou, může být jednoznakový řetězec nabývající hodnoty **`A`** nebo **`N`** – v takovém případě mluvíme o *pravdivostní hodnotě*.

Můžeme tedy hovořit o tom, že každý atribut má svůj typ. Některé budou znakové (řetězec), některé budou číselné a některé budou vystupovat jako pravdivostní hodnota.

V typech atributu můžeme jít ale mnohem dál. Představte si entitu **`RODINNÝ_DOMEK`**, což by mohla být entita v informačním systému realitní kanceláře, která by popisovala všechny rodinné domky, které daná realitní kancelář nabízí k prodeji. Tato entita může mít následující atributy:

- Název, pod kterým vystupuje nabídka daného rodinného domku (např. „Rodinný dům, 4+1, po rekonstrukci, v Brně – Řečkovicích“)
- Datum, kdy byla zveřejněna nabídka k prodeji
- Celková užitná plocha rodinného domku
- Velikost parcely, na které rodinný domek je postaven
- Fotografie rodinného domku v katalogu realitní kanceláře
- Video-ukázka interiéru rodinného domku, kterou si klient může prohlédnout
- Cena za nemovitost
- Informace, zda-li jsou v rodinném domku instalovány všechny inženýrské sítě
- Popis vybavení rodinného domku (kuchyňská linka, podlahy, okna, …)
- a jiné

Název nabídky bude zřejmě nějaký znakový řetězec, datum zveřejnění bude mít formát datumu, celková užitná plocha a velikost parcely budou celá čísla. Fotografie rodinného domku bude nejspíš nějaký grafický soubor ve formátu JPEG. Video ukázka bude uložena jako soubor s příponou MPEG. Cena za nemovitost bude celé, případně reálné číslo (kdyby se hrálo na padesátníky). Informace, zda-li jsou do domku přivedeny všechny inženýrské sítě, bude pravdivostní hodnota, a nakonec popis vybavení bude opět řetězec znaků.

Z výše uvedených příkladů (**`ZAMĚSTNANEC`**, **`RODINNÝ_DOMEK`**) můžeme vysledovat následující typy atributů, se kterými budeme dále pracovat:

- řetězec znaků abecedy (**`STRING`**)
- celé číslo (**`INTEGER`**)
- reálné číslo (**`REAL`**)
- pravdivostní hodnota (**`BOOLEAN`**)
- datum (**`DATE`**)
- čas (**`TIME`**)
- obrázek (**`IMAGE`**)
- audio nahrávka (**`AUDIO`**)
- video ukázka (**`VIDEO`**)
- odkaz na konkrétní instanci jiné entity (**`LINK`**) – speciální atribut, je popsán později

Úloha: Kompletně popište entitu ZAMĚSTNANEC z této kapitoly
Tato úloha demonstruje kompletní příklad zápisu entity.

Entita: Zaměstnanec

Popis: Entitou zaměstnanec rozumíme každého člověka, který je zaměstnán v naší firmě na základě pracovní smlouvy na dobu určitou nebo neurčitou.

Atributy:

- Jméno : STRING
- Příjmení : STRING
- Datum_narození : DATE
- Adresa : STRING
- Funkce : STRING
- Plat : INTEGER
- Doba_neurčitá : BOOLEAN

Seznam atributů zapisujeme jako seznam dvojic: název atributu, dvojtečka, typ atributu.

Vazby mezi entitami

Podobně jako v reálném světě mají různé věci a objekty mezi sebou nějaký vztah, můžeme určitý vztah vysledovat také mezi entitami. Takovému vztahu budeme říkat vazba mezi entitami. Ta je určena slovním popisem, o jakou vazbu (vztah, souvislost) mezi entitami se jedná. Podívejme se na následující příklady:

Příklad: Entity OSOBA a KREDITNÍ_KARTA
Jaký vztah by mohl mezi těmito dvěma entitami být? Například: „Osoba platí v obchodech kreditní kartou". Co to znamená? Existuje konkrétní instance ze seznamu osob (tedy konkrétní osoba) a k ní je „přiřazena" konkrétní instance ze seznamu kreditních karet (určitá kreditní karta). Přiřazení je určeno sémantikou tvrzení „osoba platí kreditní kartou" – nebo-li, takové dvě instance budou k sobě přiřazeny, které odpovídají skutečnosti (obrazem reálného světa): konkrétní osoba zaplatila konkrétní kartou. Dále můžeme najít jinou instanci ze seznamu osob (jinou osobu), která bude ve vztahu s jinou instancí ze seznamu kreditních karet. Obecně lze říci, že všechny instance entity **`OSOBA`** a všechny instance entity **`KREDITNÍ_KARTA`** splňují vazbu mezi těmito entitami.

Příklad: Entity UČITEL a PŘEDMĚT
Vztah mezi těmito dvěma entitami můžeme popsat následující vazbou „Učitel učí předmět". Jistě bychom v takovém systému našli instance entity **`UČITEL`** a takové instance entity **`PŘEDMĚT`**, tak že by platilo, že určitý učitel učí daný předmět. (Tento vztah se popsat i z „opačného" směru: „předmět je vyučován (nebo přednášen) učitelem". Věta popisující danou vazbu je odlišná, ale významově stejná).

Příklad: Enity ULICE a MĚSTO
Vztah mezi entitami: „Ulice patří (je součástí) určitého města".

Příklad: Entity OSOBA a PLATNÝ_OBČANSKÝ_PRůKAZ
„Osoba vlastní platný občanský průkaz“.

Tak jako se musíme naučit detekovat konkrétní entity informačního systému při jeho návrhu, tak se musíme také zaměřit na hledání vhodných vazeb mezi těmito entitami. Tyto vazby plynou z poznání reálného světa, který modelujeme. Důležité je, že všechny vazby mezi entitami jsou symetrické: tzn., že pokud jedna entita je ve vztahu s druhou entitou, tak zároveň druhá entita je ve vztahu s první entitou.

Popisy jednotlivých vazeb mezi entitami jsou většinou založeny na významovém slovese, které určuje onen vztah mezi entitami. Takové sloveso se může vyskytnout v popisu vazby mezi entitami jakékoliv. Nicméně jsou určitá slovesa, která se vyskytují poměrně často. Jedná se o slovesa vyjadřující nadřazenost a podřazenost: „patří/je součástí“, „má/vlastní“ a slovesa určující typ: „je typu“.

Zbývá otázka, jak budeme zapisovat vazby v rámci návrhu datového modelu. Uvažme entity **OSOBA** a **KREDITNÍ_KARTA**, vazba mezi nimi bude popsána následovně:

„Určitá osoba (#OSOBA) platí v obchodě kreditní kartou (#KREDITNÍ_KARTA)./1:N“

Popis je určen jednak větou s významovým slovesem, dále pak v závorkách uvedeme speciální zápis instance dané entity: řekneme-li „určitá osoba“ myslíme tím, jednu konkrétní instanci z entity **OSOBA**, což zapíšeme do závorky jako (#OSOBA). Podobně zapíšeme k textu „kreditní kartou“, kterou máme na mysli instanci entity **KREDITNÍ_KARTA**, zapíšeme do závorky (#KREDITNÍ_KARTA). Nakonec u popisu vazby nezapomeneme vložit znak lomítka a za něj zápis o jakou násobnost vazby se jedná (viz kapitola Násobnost vazeb). Tím je zápis vazby kompletní.

Atribut typu LINK

Jedná se o speciální atribut, který budeme potřebovat pro realizaci vazeb mezi entitami. Vezměme si entitu **UČITEL** s následujícími atributy:

- číslo : INTEGER
- jméno : STRING
- příjmení : STRING
- úvazek : REAL

Tato entita bude obsahovat „seznam“ všech učitelů na vysoké škole. Obsah seznamu může být například:

ČÍSLO	JMÉNO	PŘÍJMENÍ	ÚVAZEK
1	Jan	Novák	1,0
2	Jiří	Dvořák	0,75
3	Petra	Nováčková	1,0
4	Jaroslav	Vomáčka	0,5

A dále mějme entitu **PŘEDMĚT**, která bude popisovat jednotlivé předměty, které se učí. Atributy této entity jsou:

- kód_předmětu : STRING
- název : STRING
- počet_hodin : INTEGER
- učitel : LINK

Za předpokladu, že daný předmět učí jenom jeden učitel, je tato entita v pořádku a odpovídající seznam může vypadat následovně:

KÓD_PŘEDMĚTU	NÁZEV	POČET_HODIN	UČITEL
BAD	Báze dat	2	1
NEJ	Německý jazyk	4	3
ANJ	Anglický jazyk	4	2
OBN	Občanská nauka	1	4
TEV	Tělesná výchova	2	2

Jednotlivé instance (tj. konkrétní řádky v seznamu **UČITEL**) jsme očíslovali. To je důvod, proč entita **UČITEL** má atribut číslo. Tímto můžeme libovolný řádek (instanci) jednoznačně odkazovat odjinud. Seznam předmětů na každém řádku pak má vyplněno číslo ve sloupci učitel. Tím číslem se rozumí odkaz (*link*) na řádek v seznamu entity **UČITEL**. Není zde tedy přímo napsáno jméno a příjmení učitele, ale je zde identifikace jeho „záznamu“, ve kterém jsou už jinde uvedeny jeho bližší údaje.

První řádek v seznamu entity **PŘEDMĚT** bychom mohli interpretovat následovně: Předmět s kódem **BAD** je **Báze dat**, učí se **2** hodiny týdně a učí jej **učitel s číslem 1**, což je (když se podíváme do seznamu učitelů na instanci – řádek – číslo 1) pan **Jan Novák**, který zároveň má na škole **úvazek 1,0**.

S atributy typu LINK se setkáme všude tam, kde jsou entity propojeny do vazby. Jedna z entit musí mít „očíslované“ instance (tedy mít atribut číslo) a instance druhé entity se budou odkazovat na instance entity první pomocí atributu, který bude typu LINK.

Násobnost vazeb

Jakmile určíme existující vazbu mezi entitami, kterou má smysl zmiňovat v konstrukci návrhu informačního systému, je potřeba se dále zamyslet nad násobností takové vazby. Násobnosti vazby říkáme také kardinalita vazby. Mohou nastat celkem tři různé případy kardinality vazeb. Ukážeme si je postupně na následujících příkladech. Vezměme si již

zmíněné příklady vazeb a přemýšlejme nad tím, kolik instancí na straně první entity a na straně druhé entity obecně může vstoupit do vazby:

Vazba 1:1

Entita **OSOBA** a **PLATNÝ_OBČANSKÝ_PRůKAZ**. Co můžeme říci zde o počtu instancí, které mohou obecně vstoupit do vazby mezi těmito dvěma entitami? Takový počet opět musíme zjistit z obrazu reálného světa, který je předmětem naší evidence.

Zamysleme se tedy: jedna osoba (starší 15-ti let, ale to je teď nepodstatné) vlastní vždy jeden platný občanský průkaz. (Asi je to pravda, že ano?) A obráceně, jeden konkrétní platný občanský průkaz náleží právě jedné osobě (na průkazu je vypsáno jméno, příjmení, datum narození, apod. patřící výhradně jedné osobě). V tomto případě to znamená, že do vazby mezi entitami **OSOBA** a **PLATNÝ_OBČANSKÝ_PRůKAZ** vstupuje vždy jedna instance entity **OSOBA** a jedna instance entity **PLATNÝ_OBČANSKÝ_PRůKAZ**.

Hovoříme o vazbě 1:1 (čteme „jedna ku jedné“). V reálném světě se vazby 1:1 vyskytují poměrně vzácně.

Vazba 1:N

Vezměme si entity **OSOBA** a **KREDITNÍ_KARTA**. Jaká vazba (ptáme se na násobnost) by mohla být mezi těmito entitami (vazba je dána popisem: „osoba platí kreditní kartou“)? Když uvážíme jednu konkrétní kreditní kartu, tak zcela jistě bude patřit právě jedné osobě. Je to tak? Mělo by být. A na druhou stranu se ptejme, kolik kreditní karet obecně může jedna osoba mít? Jistě mi dáte za pravdu, když řeknu, že jednu nebo i více. Dnes je už poměrně běžné, že osoba má více kreditních karet k různým bankovním účtům a nebo i jenom k jednomu účtu. Do vztahu tedy vstupuje vždy jedna osoba na straně jedné a jedna nebo více kreditních karet na straně druhé.

V takovém případě budeme mluvit o vazbě 1:N. Tento typ vazby se vyskytuje již výrazně častěji, než vazba 1:1. Nicméně, co se týče počtu výskytů, prvenství patří vazbě typu M:N. Ta je ze všech vazeb nejčastější.

Vazba M:N

Mějme entitu **UČITEL**, kterou budeme mít na mysli učitele na vysoké škole, a mějme entitu **PŘEDMĚT**, která bude zahrnovat všechny předměty, které se na vysoké škole přednáší a cvičí. Jak už jsme si řekli, vazba je dána popisem: „učitel učí předmět“ (sloveso „učí“ nám značí buď „přednáší“ nebo „cvičí“). Když se podíváme, jaká je realita na nějaké vysoké škole, zjistíme, že jeden učitel přednáší nebo cvičí zpravidla více různých předmětů (o tom, jestli to je pravda, se lze přesvědčit nahlédnutím do rozvrhu hodin). Na druhou stranu jistě může existovat nějaký předmět, který bude mít těch přednášejících nebo cvičících více, než jen jednoho.

Do vazby (vztahu) mezi entitami **UČITEL** a **PŘEDMĚT** nám může vstupovat obecně několik učitelů a zároveň několik předmětů. Doslova řekneme: „jeden učitel učí více předmětů“ (dílčí vazba 1:N) a zároveň „jeden předmět je učen více učiteli“ (druhá dílčí vazba, 1:N).

Z toho dostaneme obecný poměr M:N. Jak už bylo uvedeno, tento typ vazby je nejběžnější. Dalo by se dokonce říci, že tato vazba existuje vždy. A vazby 1:N nebo 1:1 jsou jen speciálními případy vazby M:N (tj. M=1, nebo M=N=1).

Pojmenované vazby

Doposud jsme zmiňovali vazby mezi entitami, které byly dány svým popisem a jenom říkaly, že mezi dvěmi konkrétními entitami existuje nějaký vztah. Dále již umíme určit, s jakou násobností daný vztah existuje. Někdy se nám ale může stát, že uvažovaná vazba mezi entitami je natolik důležitá, že ji potřebujeme evidovat. Evidovat ovšem neznamená nic jiného, než pro danou vazbu zavést novou entitu.

Vazbě, pro kterou zavedeme novou entitu, budeme říkat pojmenovaná vazba. Dvě původní entity, které byly „spojeny“ vazbou, budou nyní navázány (každá z nich) na nově zavedenou entitu.

Znalost existence vazby je dána obrazem reálného světa. Pokud má být vazba pojmenovaná (tj. existovat pro ni entita), opět to musí plynout z poznání o realitě. Kde tedy můžeme očekávat existenci pojmenované vazby? Obecně platí, že pokud máme dvě entity a mezi nimi vazbu, která je „dokumentována“ nějakým dokumentem (smlouva, účtenka, faktura, dodací list, atd.), můžeme pro takový dokument určit entitu. Tato entita (říkáme jí také asociativní) bude zároveň vystupovat jako pojmenovaná vazba mezi dvěma původními entitami. Uveďme si několik příkladů:

Příklad: Entity OSOBA a ZBOŽÍ

Osobou myslíme fyzickou osobu, občana našeho státu, který může nakupovat v obchodech za peníze. Zboží je pak jakýkoliv výrobek, který je nabízen na pultech prodejen. Existuje vazba „osoba kupuje zboží“. Když uvážíme situaci, že daná osoba kupuje dané zboží, může nás u této koupě zajímat mnoho věcí: ve kterém obchodě osoba zakoupila zboží, kdy (datum, čas), případně u jaké pokladny (jméno pokladníka) byl obsloužen. Tyto všechny údaje bychom chtěli evidovat pro vazbu „osoba kupuje zboží“. Všechny uvedené údaje jsou ovšem vytištěny na pokladní účtence. A takovou účtenku můžeme evidovat, nebo-li pro ní mít asociativní entitu **ÚČTENKA**.

Příklad: Entita OSOBA

Mějme entitu **OSOBA** a sledujme vztah této entity na sebe samu. Co to znamená? Budeme mít jednu instanci z entity **OSOBA** a ta může být ve vztahu s jinou instancí opět entity

OSOBA. Taková vazba může být popsána: „jedna osoba uzavírá nějakou smlouvu s jinou osobou“. Smlouva obsahuje pak údaje o jednotlivých stranách a předmět smluvního ujednání. Budeme-li chtít vazbu, že osoba uzavírá smlouvu s jinou osobou evidovat, použijeme pro to asociativní entitu **SMLOUVA**, která nám uvedenou vazbu dostatečně zdokumentuje.

Vazby mezi více entitami

Kromě vazeb mezi dvěma entitami můžeme v informačním systému vysledovat i vazby mezi více než dvěma entitami. V takovém případě uvedeme popis vazby tak, aby z něho bylo zřejmé, že do vazby vstupuje více entit. Příkladem mohou být entity **OSOBA**, **ZBOŽÍ** a **OBCHOD**. Vazbu popíšeme takto: „daná osoba kupuje konkrétní zboží v nějakém obchodě“. Zde také můžeme zavést asociativní entitu **ÚČTENKA**, která bude pojmenovanou vazbou mezi entitami **OSOBA**, **ZBOŽÍ** a **OBCHOD**.

Vybrané techniky datového modelování

Základní úlohou při návrhu datového modelu pro zadaný informační systém je nalezení entit. Ty můžeme nalézt jenom tak, že se důkladně seznámíme s celým informačním systémem a zjistíme o něm následující skutečnosti:

- Kdo bude s tímto informačním systémem (IS) pracovat?
- Kde a jak bude tento IS nasazen do provozu?
- Jaká všechna data je potřeba v tomto IS evidovat a kdo je bude evidovat?
- Jaké informace (výstupy, sestavy, …) bude IS poskytovat a komu?
- Co je cílem nasazení IS? Jaké další otázky a problémy má jeho nasazení řešit?
- Jaké konkrétní funkce a procedurální postupy mají být v IS implementovány?

To je jen základní výčet otázek, na které je potřeba se při seznamování s informačním systémem, odpovědět. Tyto odpovědi nám velmi pomohou při identifikování entit, jejich atributů a jejich vztahů (vazeb) mezi sebou.

Budeme zpravidla postupovat tak, že navrhneme první řešení datového modelu a po té toto řešení zrevidujeme. Přezkoumáme, zdali nalezené entity jsou opravdu těmi správnými entitami, zda-li je informační systém postavený na tomto souboru entit, schopen odpovědět na všechny dotazy a zpracovat všechny požadavky. Při přezkoumávání je možnost provádět některé optimalizace nebo drobné změny stávajících entit, pokud se to bude jevit jako vhodné. Těmto optimalizacím a změnám budeme říkat techniky datového modelování, z nichž některé si zmíníme.

Tyto techniky mají ještě jeden cíl a to umožnit nalezení nových entit, které jsme doposud neidentifikovali a bylo by je vhodné v návrhu mít.

Generalizace a typování

Generalizací rozumíme sloučení skupiny podobně vypadajících entit do jedné, obecnější. Uvažme následující situaci: při návrhu datového modelu jsme pro zadaný systém identifikovali mimo jiné entity **MUŽ** a **ŽENA**. Entitou **MUŽ** rozumíme všechny osoby mužského pohlaví, které budou předmětem evidence a entitou **ŽENA** rozumíme všechny osoby pohlaví ženského. Každá z těchto entit bude mít pravděpodobně atributy jméno, příjmení, datum narození, bydliště, atd. Tyto dvě entity jsou velmi podobné (mají stejnou nebo podobnou strukturu atributů). Jediné, v čem se liší, je pohlaví osoby. V takovém případě, můžeme místo těchto dvou entit zavést jen jednu entitu a tou bude **OSOBA**. Ta zatím bude mít shodné atributy, jako měly entity **MUŽ** či **ŽENA**.

Samozřejmě pokud takto převedeme dvě entity původní na novou obecnější entitu, musíme si ještě dát pozor na to, aby nedošlo ke ztrátě informace. Atributy entit zůstávají stejné, čili ke ztrátě např. jména nebo příjmení nedojde. V původním návrhu, jestli byla nějaká osoba instancí entity **MUŽ**, to znamenalo, že je pohlaví mužského. Obdobně, pokud jsme měli nějakou osobu, která byla instancí entity **ŽENA**, znamenalo to, že osoba je pohlaví ženského. Kdežto nyní, je-li nějaká osoba instancí entity **OSOBA**, o jejím pohlaví nevíme nic (jistě, můžeme usuzovat ze tvaru jména příjmení, případně rodného čísla – ale čistě z pohledu databázového, o pohlaví osoby nevíme nic). Je jasné, že v takovém případě budeme muset zavést do entity **OSOBA** ještě jeden nový atribut. Který to bude? Bude to atribut určující právě pohlaví osoby. Nebo-li atribut, určující, z jaké původní entity (**MUŽ** nebo **ŽENA**) daná instance pocházela. A zde přichází na řadu technika zvaná typování.

Typování je technika sloužící k rozlišení typu původních entit, které byly sloučeny v rámci generalizace. Založíme tedy novou entitu pro určení typu obecné (generalizované) entity. Co to znamená? Vraťme se zpět k našemu příkladu.

V našem příkladu bychom tedy měli založit novou entitu, která se bude jmenovat **TYP_OSOBY**. Instancemi této entity budou položky číselníku udávající, zdali instance entity **OSOBA** pochází původně z entity **MUŽ** nebo z entitu **ŽENA**. Jinými slovy, počet instancí entity **TYP_OSOBY** bude 2. První instance bude položka popisující, že jde o muže, druhou instancí bude položka popisující, že jde o ženu. Když se nad tím zamyslíme, tak přesně takové instance by mohla mít entita zvaná **POHLAVÍ**, čili použijeme raději tento název, než **TYP_OSOBY**.

Jak tedy bude vypadat celkový výsledek provedené generalizace entit **MUŽ** a **ŽENA**? Výsledkem bude nová entita **OSOBA**, která nahrazuje obě původní entity. Bude mít stejné atributy jako původní entity a jeden nový atribut **typ_osoby** navíc. Dále pak nová entita **POHLAVÍ**, kterou jsme popsali výše. Původní dvě entity **MUŽ** a **ŽENA** můžeme smazat. Je jasné, že mezi entitami **OSOBA** a **POHLAVÍ** musí být vazba a to násobnosti 1:N (snadno se ověří:, že každá osoba je právě jednoho pohlaví - buď je to muž nebo žena – a na druhou stranu, mužů je více osob a žen taktéž)

Příklad: generalizace entit pro dopravní prostředky
Mějme několik nalezených entit pro nějaký dopravní informační systém: **AUTOMOBIL**, **AUTOBUS**, **TRAMVAJ**, **TROLEJBUS**, **VLAK**, **LETADLO**. Řekněme, že u každé entity budeme sledovat stejné atributy. V takovém případě jsou tyto entity vhodné ke sloučení. Vznikne nová obecnější entita **DOPRAVNÍ_PROSTŘEDEK**, která bude mít atributy shodné jako původní entity a dále bude mít navíc jeden atribut **typ_prostředku**. Tento atribut bude sloužit k identifikaci, z jaké původní entity daná instance pochází. Druhá entita se bude jmenovat **TYP_PROSTŘEDKU** a její instance budou položky číselníku, určující typ dopravního prostředku. Celkem těchto položek bude 6 (protože jsme měli původně 6 entit). Jaká bude vazba mezi entitami **DOPRAVNÍ_PROSTŘEDEK** a **TYP_PROSTŘEDKU**? Opět 1:N, jak se snadno ověří.

Provedli jsme tedy generalizaci a typování, místo původních 6 entit máme v návrhu jen 2 entity. Výsledek je přehlednější.

Z výše uvedených příkladů vyplývá, že generalizace a typování jsou dvě techniky, které se praktikují souběžně. Z tohoto důvodu, jsou obě tyto techniky popsány v jedné kapitole.

Agregace

Agregace je seskupení částí do jednoho celku, tzn. že entita je součástí jiné entity. Tímto mechanismem můžeme modelovat hierarchie mezi entitami.

Poznámka:
Pozor, neplést s generalizací. Generalizací slučujeme a vytváříme novou entitu, která nahrazuje původní entity, agregací k existujícím entitám (nebo i jen k jedné existující entitě) vytvoříme novou, která bude tím „celkem“, jehož budou původní entity součástí a původní entity zůstávají.

Nejlépe si agregaci ukážeme na následujících příkladech:

Příklad: jednoduchá agregace
Mějme entity **ULICE** a **NÁMĚSTÍ**. Entitou **ULICE** rozumíme běžné ulice a entita **NÁMĚSTÍ** slouží pro evidenci náměstí. Atributy těchto entit budou různé. Pro agregaci není podmínka, že entity musí mít stejnou nebo podobnou strukturu atributů (jak tomu bylo v případě generalizace). Přesto tyto dvě entity mají něco společného. Patří do nějakého celku. A tím celkem může například být město. Zavedeme tedy entitu **MĚSTO**. Tato entita bude ve vazbě s oběma původními entitami. Vazba **ULICE** a **MĚSTO** je „ulice patří (je součástí) nějakého města“ s násobností 1:N a vazba **NÁMĚSTÍ** a **MĚSTO** je „náměstí patří (je součástí) nějakého města“ opět s násobností 1:N.

Příklad: vícenásobná agregace
Mějme entitu **STUDIJNÍ_PŘEDMĚT**, která popisuje studijní předměty na vysoké škole. Takové předměty jsou jistě součástí nějakého celku a tím může být studijní program. Zavedeme pro něj tedy entitu **STUDIJNÍ_PROGRAM**. S původní entitou **STUDIJNÍ_PŘEDMĚT** bude ve vazbě „studijní předmět je součástí nějakého studijního programu“ násobnosti 1:N. Zde dokonce můžeme jít až na vazbu M:N, pokud bychom vzali v úvahu skutečnost, že jeden konkrétní předmět může být zahrnut do více studijních programů. U entity **STUDIJNÍ_PROGRAM** zůstaňme. I pro tuto entitu můžeme identifikovat celek, do kterého bude patřit a tím je studijní obor. Pro něj vymyslíme entitu **STUDIJNÍ_OBOR**. Vazba mezi entitami **STUDIJNÍ_PROGRAM** a **STUDIJNÍ_OBOR** bude typu 1:N a popsaná „studijní program je součástí nějakého studijního oboru“. Vytvořili jsme tedy dvouúrovňovou hierarchii.

Kategorizace

Tato technika od těch předchozích se liší hlavně v tom, že se netýká primárně entit, ale pracuje s instancemi entit. Nicméně i tak má za následek úpravu datového modelu spočívající ve vytvoření nové entity k již existujícím.

Kategorizací rozumíme možné seskupování instancí entity do kategorií. Tímto způsobem objevíme novou entitu pro nalezenou kategorii. Cesta k nalezení této nové entity spočívá tedy v sledování jednotlivých instancí entity.

Příklad: entita TRASA a nalezení kategorie pro její instance
Mějme entitu **TRASA**, která bude popisovat jednotlivé vlakové trasy z města Brna do různých měst v okolí. Atributy této entity nechť jsou: číslo_trasy, město, kam trasa vede a počet_km, jak je trasa dlouhá. Nyní si musíme vybrat atribut, podle kterého můžeme jednotlivé instance entity kategorizovat. Nabízí se v podstatě jen dva: atribut **město** a **počet_km**. Jednotlivá města bychom mohli kategorizovat snad podle kraje, nebo okresu, kde leží, nicméně takový údaj nám nebude nic moc platný. Atribut **počet_km** se nabízí jako vhodnější, neboť kategorizací počtu kilometrů dostaneme určité úseky, které ale velmi dobře známe pod pojmem tarifní pásma. Zavedeme tedy novou entitu **TARIFNÍ_PÁSMO**. Ta bude popisovat rozdělení množiny (kilometrů) na jednotlivé kategorie. Každá kategorie bude pak reprezentována jedním tarifním pásmem. A v původní entitě **TRASA** přidáme pouze jeden nový atribut určující o jaké tarifní pásmo (kategorii) se jedná. Vazba mezi těmito entitami bude 1:N.

Poznámka: hledání atributu, podle kterého se bude kategorizovat
Pro realizaci kategorizace lze vzít libovolný atribut entity (resp. jeho hodnoty v jednotlivých instancích), nicméně praxe ukazuje, že kategorizace se poměrně běžně dělá z číselných atributů a to tak, že se stanoví jednotlivé intervaly (od,do) pro každou kategorii a pak každá instance, dle hodnoty sledovaného atributu (podle kterého jsme kategorizovali), bude spadat právě do jedné kategorie.

Rekurze

Rekurze je technika, která nám slouží k objevování nových entit na základě zkoumání vazeb entit na sebe samu. V případě, že při zkoumání takové vazby zjistíme potřebu vazbu pojmenovat, zavedeme novou entitu (asociativní).

Příklad: různé příklady rekurzí
Mějme entitu **OSOBA**. Zde můžeme vysledovat vazbu na sebe sama (rodič-dítě) a tím nalézt novou entitu **RODNÝ_LIST**. Dále mějme entitu **LOKALITA**. Zde můžeme najít vztah „být částí jiné lokality“, ale nemusíme nutně mít potřebu jej pojmenovávat pomocí nové entity. Posledním příkladem je entita **DOKUMENT**. Rekurzivní vztah je dán popisem „(dokument) odkazuje na (jiný dokument)“. Odtud nám může vzniknout asociativní entita **ODKAZ**.

Logický datový model

Logický datový model popisuje logickou strukturu databáze. Nezaměřujeme se přímo na konkrétní tabulky v databázi, ale sledujeme entity informačního systému a vazby mezi nimi. Logický datový model (používáme zkratku LDM) se skládá z textové a grafické části. Textová část obsahuje seznam všech nalezených entit včetně jejich popisu a dále seznam všech vazeb mezi entitami (jejich popisy). Grafickou část pak tvoří vlastní diagram ERD.

Poznámka: popis entity
Jak bylo uvedeno, je potřeba uvádět bližší popis entity. I když bude na první pohled u většiny entit zřejmé, o co se jedná, někdy tomu tak být nemusí. Příkladem může být entita **STAVBA**. Otázkou teď je, zda-li tato entita popisuje stavbu jako jednu konkrétní budovu (domek) a nebo tato entita popisuje proces, kdy se nějaký domek staví? To by měl ozřejmit popis entity. Dalším příkladem takové nejednoznačné entity je **KNIHA**. Má se na mysli kniha jako dílo nějakého autora, nebo se tím myslí jeden konkrétní exemplář díla nějakého autora? Vyřešit by to měl opět popis.

Diagram ERD

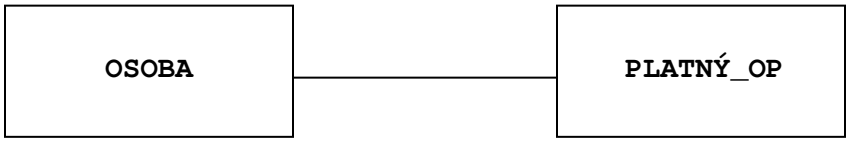
Diagram entit a relací (ERD) graficky znázorňuje všechny entity informačního systému, dále vazby mezi entitami a to včetně jejich násobnosti. Jednotlivé vazby vždy očíslujeme, abychom se pak na ně mohli odkazovat v textové části.

Základní prvky diagramu ERD

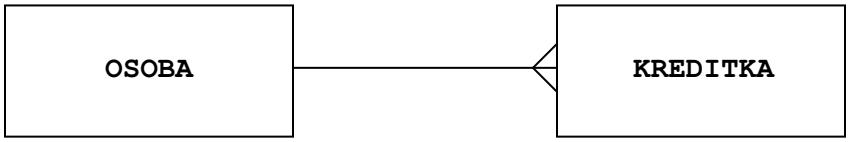
Entity zakreslujeme do obdélníku, název entity vepíšeme do něj:

Obrázek: graficky znázorněná entita ZAMĚSTNANEC

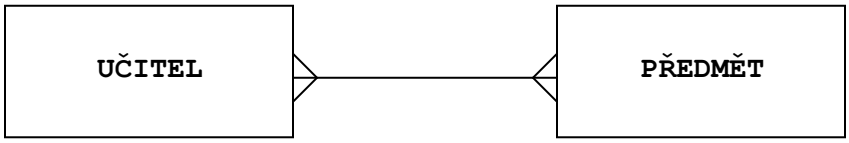
Vazby mezi entitami zachytíme tak, že entity spojíme čárou. Jsou celkem tři možnosti, jak bude spojovací čára vypadat (je to dáno násobností vazby):



Obrázek: graficky znázorněná vazba 1:1 mezi entitami

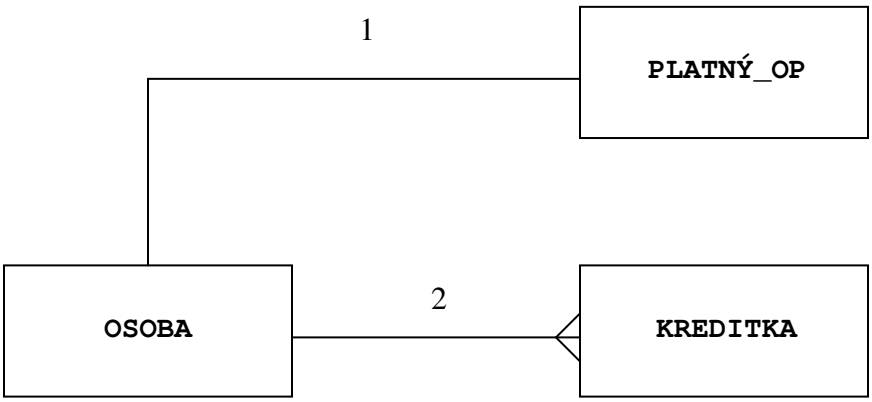


Obrázek: graficky znázorněný vztah 1:N



Obrázek: graficky znázorněná vazba M:N

Očíslování vazeb provedeme následovně:



Obrázek: ukázka očíslování vazeb mezi entitami

Úloha: školní knihovna
Navrhněte logický datový model pro jednoduchý informační systém školní knihovny. Popis fungování školní knihovny je následující: Studenti si chodí půjčovat knihy do školní knihovny. Konkrétní dílo se může v knihovně vyskytovat vícekrát. Daný exemplář nějakého díla může mít v jeden okamžik zapůjčen pouze jeden student. Každý exemplář je opatřen nálepkou s evidenčním číslem. Studenti mohou vyhledávat knihy podle názvu, autora a žánru. Každou knihu napsal minimálně jeden autor, případně více autorů. Konkrétní exemplář knihy vydalo právě jedno nakladatelství. Pokud systém podle vyhledávacích kritérií nalezne knihu a zjistí, že je volný nějaký její exemplář, umožní studentovi výpůjčku. Studenti jsou povinni danou knihu vrátit nejpozději do určeného data. Po tomto datu je studentovi vystavena upomínka. Upomínka musí obsahovat, komu je vystavena, jaký konkrétní exemplář student dluží a kdy jej měl vrátit.

Úkolem je navrhnout informační systém podporující existující fungování školní knihovny. Uvedené řešení chápejte jako ucelený návod, jak postupovat při tvorbě logického datového modelu.

Řešení:
Důkladným prozkoumáním zadání a případně pomocí aplikace některých technik datového modelování postupně identifikujeme všechny entity v systému, dále jejich atributy a vazby mezi sebou. Nakonec zobrazíme grafický ERD diagram zadaného systému. Tím bude řešení této základní úlohy kompletní.

Seznam entit
Entita: **`STUDENT`**

Popis: Entitou student rozumíme libovolnou osobu, která je zapsána jako student na dané odborné škole a která má právo si půjčovat knihy ve školní knihovně.
Atributy:

- Jméno : STRING
- Příjmení : STRING
- Datum_narození : DATE
- Ročník : INTEGER

Entita: **AUTOR**
Popis: Autorem rozumíme každou osobu, která napsala nějakou publikaci, jejíž exemplář lze vypůjčit ve školní knihovně.
Atributy:

- Jméno : STRING
- Příjmení : STRING

Entita: **ŽÁNR**
Popis: Entitou žánr máme na mysli určení, jakých žánrů mohou být knihy nabízené k půjčování ve školní knihovně (např. dětská, odborná, sci-fi, detektivní, psychologický, apod.).
Atributy:

- Název_žánru : STRING

Entita: **VYDAVATEL**
Popis: Entitou vydavatel rozumíme všechny fyzické i právnické osoby, které jsou vydavateli daných děl autorů, jejichž exempláře lze vypůjčit ve školní knihovně.
Atributy:

- Jméno : STRING
- IČO : INTEGER
- Adresa : STRING

Entita: **STAV_VÝPůJČKY**
Popis: Entita stav_výpůjčky obsahuje číselník možných stavů, ve kterých se mohou nacházet jednotlivé vypůjčené exempláře ze školní knihovny (např. „půjčeno“, „vráceno“, „upomínka“, „nevráceno z důvodu ztráty“, „nevráceno z důvodu poškození“, apod.).
Atributy:

- Název_stavu : STRING

Entita: **KNIHA**
Popis: Entitou kniha rozumíme jednotlivá díla napsaná různými autory a daného žánru.
Atributy:

- Název : STRING
- Rok_vzniku : DATE
- Autoři : LINK
- Žánr : LINK

Entita: **EXEMPLÁŘ**
Popis: Exemplářem rozumíme konkrétní jeden výtisk daného díla (knihy), který je k dispozici ve školní knihovně a je-li v daném okamžiku volný, smí se jej student půjčit.
Atributy:

- Evidenční_číslo : STRING
- Kniha : LINK
- Vydavatel : LINK

Entita: **VÝPůJČKA**
Popis: Asociativní entita výpůjčka zachycuje konkrétní výpůjčky exemplářů ze školní knihovny studenty.
Atributy:

- Datum_výpůjčky : DATE
- Student : LINK
- Exemplář : LINK

Seznam vazeb
1: Každá výpůjčka (#VÝPůJČKA) daného exempláře se nachází v nějakém stavu (#STAV_VÝPůJČKY). /1:N

2: Každý exemplář (#EXEMPLÁŘ) je vytištěn určitým vydavatelem (#VYDAVATEL). /1:N

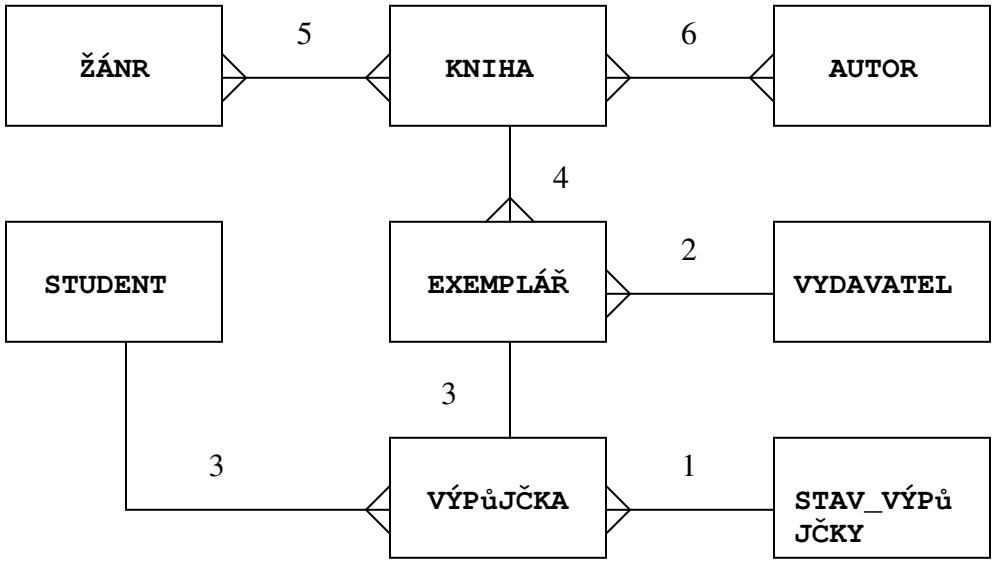
3: Student (#STUDENT) si půjčuje jednotlivé exempláře (#EXEMPLÁŘ) a záznam o půjčení je zanesen do výpůjční knihy (#VÝPůJČKA). /1:N

4: Dané dílo (#KNIHA) je vydáno v několika exemplářích (#EXEMPLÁŘ) a je k dispozici k půjčování. /1:N

5: Dané dílo (#KNIHA) je určitého žánru (#ŽÁNR). /M:N

6: Dané dílo (#KNIHA) napsalo několik autorů (#AUTOR). /M:N

Nakonec už jen zbývá nakreslit diagram ERD, který jednotlivé entity, resp. vazby mezi nimi, znázorní přehledně v grafické podobě.



Obrázek: ERD diagram řešení úlohy školní knihovna

Shrnutí

- **Základní pojmy datového modelování jsou entita, instance, vazba mezi entitami, atribut.**
- **Entitou rozumíme objekt našeho pozorování, instance je pak konkrétní výskyt entity.**
- **Atributem rozumíme vlastnost entity. Každá entita má několik atributů (vlastností). Každý atribut má svůj typ (číslo, řetězec znaků, pravdivostní hodnota, atd.).**
- **Vazba mezi entitami dokumentuje vztah mezi entitami. Vazba může být různé násobnosti. Sledujeme násobnosti vazeb 1:1, 1:N a M:N.**
- **Vazba mezi entitami může být tzv. pojmenovaná, nebo-li reprezentována novou entitou. Takové entitě říkáme asociativní.**
- **Vybranými postupy (technikami) datového modelování jsou: generalizace (náhrada podobných entit novou, obecnější) a typování, agregace (entita je součástí jiné entity), kategorizace (rozdělení instancí entity do kategorií) a rekurze (vztahy entit na sebe sama).**
- **Výstupem datové analýzy (návrhu logického datového modelu) jsou: seznam popsaných entit, seznam zdokumentovaných vazeb, včetně jejich násobnosti a grafický ERD diagram znázorňující všechny entity a vazby mezi nimi.**

Otázky a úkoly

- Vezměte si hypotetický informační systém pro řízení celého chodu Janáčkova divadla v Brně. Nalezněte alespoň 5 důležitých entit tohoto systému.
- Najděte atributy entity **POHLEDNICE** a uveďte příklad jedné její instance.
- Najděte atributy následujících entit: **ŠKOLNÍ_PŘEDMĚT**, **BYT**, **ZÁJEZD**, **NOVINOVÝ_ČLÁNEK**.
- Určete a diskutujte násobnost vazeb mezi následujícími entitami: **AUTOR_ČLÁNKU** a **NOVINOVÝ_ČLÁNEK**; **OSOBA** a **KARTÁČEK_NA_ZUBY**; **HROMADNÁ_JÍZDENKA** a **SKUPINA_OSOB**; **OSOBA** a **RODNÝ_LIST**; **ZAMĚSTNANEC** a **ZAMĚSTNAVATEL**
- Vymyslete asociativní entitu (pojmenovanou vazbu) mezi následujícími entitami: **OSOBA** a **OSOBA**; **OSOBA** a **AUTOMOBIL**; **POČÍTAČ**, **MONITOR**, **KLÁVESNICE**, **MYŠ**; **OSOBA** a **ZÁJEZD**
- Zkonstruujte kompletní logický datový model pro následující zadání IS Webové fotoalbum: Album by mělo nabízet roztřídění fotografií do jednotlivých kolekcí, chronologicky uspořádaných dle data. Každá kolekce fotografií bude mít svůj název (např. „*Dovolená ve Španělsku (2001)*“). Uživatel (návštěvník) webového alba si bude moci v jednotlivých kolekcích listovat, klikne na název a zobrazí se mu kolekce fotek. Nebudou se načítat rovnou všechny fotografie v plné velikosti, ale zobrazí se nejdříve seznam zmenšených obrázků (*thumbnailů*) a teprve po kliknutí na některých z nich se zobrazí fotografie v původní velikosti. Každá fotografie bude mít svůj popisek a datum pořízení. Ke každé fotografii bude umožněno přidávat komentáře od jednotlivých návštěvníků. Každý návštěvník může přidat více komentářů k jedné fotografii. Každou fotografii může návštěvník nejvýše jednou ohodnotit body 1 až 5 (5 výborná fotografie, 4 dobrá fotografie, 3 průměrná fotografie, 2 špatná fotografie, 1 nevhodná k vystavení). Systém by měl počítat s budoucím rozšířením bodové stupnice.

Relační databáze

V této kapitole postupně přejdeme od logického datového modelu k datovému modelu fyzickému. Předmětem našeho zájmu budou tabulky, jejich struktura a konstrukce. Dále budeme řešit realizaci vazeb mezi tabulkami a zajišťovat, aby data v relační databázi byla uložena a udržována v konzistentním stavu.

Základním stavebním prvkem relačních databází jsou *relace*. Jedná se o klíčový pojem, kterému je věnována první podkapitola.

Relace

S těmi znalostmi, co již máme, jsme schopni dát dohromady několik vágních a nepřesných definic, co je to relace. Určitě se shodneme na tom, že relace je obecně nějaký vztah. Vezměme si následující příklad relace **ZAMĚSTNANEC** (aniž bychom v této chvíli věděli, co to přesně relace je a znamená):

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	PLAT	SMLOUVA_OD
1	Jan	Novák	15.10.1975	15000	1.1.2000
2	Petr	Nový	1.4.1978	21500	12.5.1999
3	Jan	Nováček	6.9.1965	17500	7.7.1998
4	David	Vokurka	5.12.1973	28000	1.10.2002

Co můžeme z uvedeného příkladu usoudit? Relace je popsána nějakým seznamem, který má své sloupce, obsahuje pak konkrétní řádky, kde pro každého zaměstnance jsou vyplněny hodnoty sloupců. Ale to přece není nic jiného, než entita (**ZAMĚSTNANEC**). A konkrétní řádky odpovídají instancím této entity.

Dále vidíme, že celá struktura je zobrazena přehledně pomocí tabulky. Jednotlivé atributy jsou zobrazeny jako samostatné sloupce. Odtud může uvažovat následující „pseudorovnost“:

„Relace = entita = tabulka“.

Všechny tyto objekty (relace, entita, tabulka) mají totiž stejnou strukturu, jak je vidět výše a navíc, co opravdu mají společné je to, že jsou pomocí atributů definovány. Odtud tedy plyne tato rovnost a v běžné mluvě i textu můžeme tyto pojmy mezi sebou používat jako synonyma.

Nicméně v této chvíli pouze víme, co je to entita. Další dva pojmy relace a tabulka tušíme zatím jen intuitivně. Budou mít hodně společného s entitou, nicméně drobné rozdíly a odlišnosti v interpretaci tu jsou.

Kartézský součin

Dříve, než si zadefinujeme přesný pojem relace, musíme se zastavit u kartézského součinu množin. Bez pochopení operace kartézský součin není šance k porozumění pojmu relace.

Co je to tedy kartézský součin? Mějme dvě množiny, první je množina A, jejíž prvky jsou číslice 1, 2 a 3, zapíšeme: **A={1,2,3}** a druhá je množina B se dvěma prvky – písmena – x a y, zapíšeme: **B={x,y}**. Kartézský součin množin **A** a **B** je definován následovně:

$$A \times B = \{(a,b) \mid a \in A \wedge b \in B\}$$

nebo můžeme zapsat:

$$A \times B = \{(a \in A, b \in B)\}$$

Uvedený matematický zápis lze interpretovat takto: kartézský součin dvou množin **A** a **B** je množina všech dvojic, které vzniknou takovým způsobem, že postupně dosadíme na první pozici ve dvojici všechny prvky z množiny **A** a na druhou pozici ve dvojici všechny prvky **B**. Jinými slovy, „zpárujeme“ všechny prvky z množiny **A** a množiny **B** způsobem „každý s každým“. Kartézský součin našich konkrétních množin **A** a **B** vypadá následovně:

$$A \times B = \{(1,x), (1,y), (2,x), (2,y), (3,x), (3,y)\}$$

Celkový počet dvojic v kartézském součinu se bude rovnat součinu počtů prvků původních množin. Jestliže množina **A** měla 3 prvky a množina **B** 2 prvky, pak počet dvojic v kartézském součinu těchto množin bude roven 6.

Jak bude vypadat kartézský součin tří množin? Vezměme si množiny **A={1,2,3}**, **B={x,y}** a **C={I,II,III}**. Kartézský součin těchto množin bude množina všech trojic, kde na jednotlivé pozice postupně dosadíme jednotlivé prvky ze všech tří množin:

$$A \times B \times C = \{(a,b,c) \mid a \in A \wedge b \in B \wedge c \in C\}$$

Výsledek kartézského součinu těchto tří množin je následující:

$$A \times B \times C = \{$$
$$(1,x,I),\ (1,x,II),\ (1,x,III),\ (1,y,I),\ (1,y,II),\ (1,y,III),$$
$$(2,x,I),\ (2,x,II),\ (2,x,III),\ (2,y,I),\ (2,y,II),\ (2,y,III),$$
$$(3,x,I),\ (3,x,II),\ (3,x,III),\ (3,y,I),\ (3,y,II),\ (3,y,III)$$
$$\}.$$

Kolik prvků (trojic) bude celkem obsahovat kartézský součin? Celkový počet trojic je dán součinem počtů prvků jednotlivých množin, které do kartézského součinu vstupují, tedy v tomto případě je jich 18 (3×2×3).

Pokud je vám v této chvíli jasné, co je to kartézský součin množin, můžete pokračovat ve čtení textu dál, v opačném případě silně doporučuji si tuto podkapitolu projít ještě jednou.

Definice relace

Nyní se vraťme zpět k našemu příkladu relace **ZAMĚSTNANEC**. Od tohoto příkladu budeme odvíjet další poznatky a závěry, které nám pomohou se dobrat k přesné definici relace.

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	PLAT	SMLOUVA_OD
1	Jan	Novák	15.10.1975	15000	1.1.2000
2	Petr	Nový	1.4.1978	21500	12.5.1999
3	Jan	Nováček	6.9.1965	17500	7.7.1998
4	David	Vokurka	5.12.1973	28000	1.10.2002

Tato tabulka popisuje jednotlivé zaměstnance ve firmě. Každý zaměstnanec má své číslo, jméno, příjmení, datum narození, výši svého platu a datum, od kdy ve firmě pracuje. U zaměstnance popisujeme tedy celkem 6 jeho vlastností. Ovšem vlastnost zaměstnance není nic jiného, než jeho atribut. A pojem atribut již známe z kapitoly o datovém modelování. Každý atribut pro daného zaměstnance nabývá nějaké konkrétní hodnoty.

Číslo zaměstnance bude nějaké vybrané číslo z množiny přirozených čísel {1, 2, 3, 4, …}. Jméno zaměstnance bude nějaké vybrané jméno z množiny křestních českých jmen {Jan, Petr, David, Pavel, Ondřej, Karel, Jonáš, Jana, Petra, Věra, …}. Příjmení zaměstnance bude také nějaká vybraná hodnota z množiny všech možných příjmení {Novák, Nový, Dvořák, Novotný, Nejedlý, Nováková, …}. Datum narození bude určité datum z množiny všech datumů řekněme z doby od 1.1.1900 až 31.12.1987). Plat zaměstnance bude nějaké přirozené číslo, větší než 0. Datum zahájení pracovní smlouvy bude opět nějaké vybrané datum z rozmezí, např. od 1.1.2000 až po současnost.

Každý atribut tedy nabývá nějakou konkrétní hodnotu z tzv. definičního oboru (definičním oborem v našem příkladu jsou čísla, datumy, křestní jména, příjmení). V databázovém světě se takovým definičním oborům říká *doména*. Doménou atributu tedy rozumíme množinu všech hypotetických hodnot, kterých může daný atribut nabývat. Můžeme tedy zapsat:

- ČÍSLO $\in D_{číslo}$,
- JMÉNO $\in D_{jméno}$,
- PŘÍJMENÍ $\in D_{příjmení}$,
- DAT_NAROZ $\in D_{datum}$,
- PLAT $\in D_{plat}$,

- SMLOUVA_OD ∈ D_{datum}.

Znak **D** značí doménu a text uvedený jako index znaku D určuje o jakou doménu se jedná. Nyní si vzpomeňme na kartézský součin a představme si kartézský součin celkem 6 množin, kde jednotlivé množiny jsou ony domény z našeho příkladu. Každý řádek v našem příkladu **ZAMĚSTNANEC** bude reprezentován šesticí následujícího tvaru:

(číslo ∈ $D_{číslo}$, jméno ∈ $D_{jméno}$, příjmení ∈ $D_{příjmení}$, dat_naroz ∈ D_{datum}, plat ∈ D_{plat}, smlouva_od ∈ $D_{smlouva}$).

Kolik prvků by měl celkem kartézský součin těchto 6 množin (nebo-li domén)? Řekněme, že množina čísel obsahuje čísla 1 až 100 000, množina jmen obsahuje 500 křestních jmen, množina příjmení obsahuje 2000 příjmení, datumů v době od 1.1.1900 až 31.12.2006 je celkem 106 roků krát 365 dní, tj. 38 690, platů větších než 0 a maximálně 50 000 je celkem 50 000). Jestliže počet prvků kartézského součinu je dán součinem prvků množin, které do kartézského součinu vstupují, pak dostáváme celkový počet: 100 000 x 500 x 2000 x 38 690 x 50 000 x 38 690 = 74845805000000000000000000, což je přibližně 7,5 kvadriliónů možností, jak mohou jednotlivé řádky v našem příkladu **ZAMĚSTNANEC** vypadat. Je jasné, že když bude v naší firmě pracovat např. 50 zaměstnanců, znamená to, že konkrétní tabulka bude obsahovat pouze těch 50 možností ze 7,5 kvadriliónů možností. Jinými slovy, naše konkrétní relace **ZAMĚSTNANEC** je podmnožina ze 7,5 kvadriliónů možností, které jsou dané kartézským součinem jednotlivých domén atributů.

Ještě jednou, co je to relace? *Relace R nad n atributy je libovolná podmnožina kartézského součinu domén jednotlivých atributů*:

R ⊂ D1 × D2 × D3 × D4 × … × Dn

Prvkem relace je pak jedna konkrétní n-tice s konkrétními dosazenými hodnotami, nebo-li jeden řádek v tabulce. Náš příklad relace **ZAMĚSTNANEC** obsahuje celkem 4 prvky relace, nebo-li 4 řádky v tabulce.

Tabulka

Tabulka reprezentuje (zobrazuje) jednu konkrétní relaci. Tak jako každá relace má své jméno, tak i tabulka má svůj název. Podobně jako má relace své atributy, tak i tabulka má své atributy. Každý atribut odpovídá jednomu sloupci v tabulce. Jednotlivé řádky v tabulce odpovídají jednotlivým n-ticím hodnot, které danou relaci splňují.

Vezměme si náš příklad relace **ZAMĚSTNANEC**:

Tabulka: ZAMĚSTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	PLAT	SMLOUVA_OD
1	Jan	Novák	15.10.1975	15000	1.1.2000
2	Petr	Nový	1.4.1978	21500	12.5.1999
3	Jan	Nováček	6.9.1965	17500	7.7.1998
4	David	Vokurka	5.12.1973	28000	1.10.2002

Tato relace je zdokumentována tabulkou **ZAMĚSTNANEC** (jak je intuitivně hned vidět z příkladu). Uvedená tabulka obsahuje celkem 4 řádky. Například druhý řádek obsahuje šestici (2, Jan, Novák, 1.4.1978, 21500, 12.5.1999) a to znamená, že člověk Jan Novák, který se narodil 1. 4. 1978, bere plat 21500 a pracuje ve firmě od 12.5.1999, splňuje relaci **ZAMĚSTNANEC**. Tedy, určení, zda-li nějaká skupina hodnot (v n-tici) splňuje, či nesplňuje danou relaci, je defacto dáno výskytem resp. nevýskytem v příslušné tabulce, která danou relaci dokumentuje.

Každá tabulka se skládá ze sloupců. Jim odpovídají jednotlivé atributy. O atributech zde platí totéž, co v logickém datovém modelu. Každý atribut má svůj typ a má nějakou vybranou hodnotu ze své domény. Více o atributech si můžete přečíst v kapitole Atributy na straně 16. Základními operacemi nad záznamy v tabulce jsou:

- Vložení nového záznamu
- Aktualizace (změna hodnot některých sloupců) záznamu
- Smazání existujícího záznamu
- Vyhledání záznamu podle zadaných kritérií
- Dotaz na existenci záznamu

Jak je z výčtu vidět, operace nad záznamy přesně korespondují se základními operacemi nad daty, které jsme si definovali v úvodní kapitole.

Pro účely konstrukce databázových tabulek budeme uvažovat následující typy atributů:

- řetězec znaků abecedy (**VARCHAR(n)**, kde **n** značí počet znaků v řetězci, **n** ≤ 255)
- celé číslo (**INTEGER**)
- reálné číslo (**FLOAT(n)**, kde **n** značí počet desetinných míst)
- pravdivostní hodnota (**CHAR(1)**), bude nabývat hodnot **A** nebo **N**
- datum (**DATE**)
- čas (**TIMESTAMP**)
- obrázek (**BLOB**)
- dlouhé textové pole s více jak 255 znaky (**TEXT**)
- primární klíč (**PRIMARY KEY(sloupec)**)*
- cizí klíč (**FOREIGN KEY(sloupec) REFERENCES tabulka(sloupec)**)*

V uvedeném výčtu na posledních dvou řádcích jsou uvedeny speciální typy atributů (sloupců) a to tzv. klíče. Jeden z nich je primární, druhý je cizí. Oba typy si nyní

podrobně popíšeme. Pochopení významu a smyslu těchto klíčů je pro další práci s databázovými tabulkami naprosto zásadní.

Funkční závislost

Dříve, než se přesně řekneme, co je to primární klíč, musíme se na chvíli zastavit u pojmu funkční závislost. Funkční závislost si lze představit jako zjednodušená tvrzení o reálném světě. Říkáme, že něco je závislé na něčem jiném. V našem případě půjde o závislosti mezi jednotlivými atributy, nebo chcete-li, mezi sloupci. Vezměme si opět náš příklad tabulky **ZAMĚSTNANEC** rozšířenou o sloupec **FUNKCE**, který dokumentuje, jakou funkci daná osoba vykonává:

Tabulka: ZAMĚSTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	FUNKCE	PLAT	SMLOUVA_OD
1	Jan	Novák	15.10.1975	uklízeč	15000	1.1.2000
2	Petr	Nový	1.4.1978	programátor	21500	12.5.1999
3	Jan	Nováček	6.9.1965	manažer	17500	7.7.1998
4	David	Vokurka	5.12.1973	ředitel	28000	1.10.2002

Jaké funkční závislosti zde můžeme vypozorovat? Budete se mnou souhlasit, když řeknu, že hodnota atributu datumu narození závisí na osobě, které se ten údaj týká? Zcela určitě. Vezmu-li například osobu Jan Novák, tak hodnota datumu narození je 15.10.1075. Když ale uvážím osobu jinou, například David Vokurka, tak hodnota datumu narození je 5.12.1973. Co tedy znamená ona funkční závislost? To, jaká hodnota bude v datumu narození totiž závisí na tom, jaká hodnota, nebo hodnoty budou ve sloupcích **JMÉNO** a **PŘÍJMENÍ**. Takovou závislost zapíšeme **JMÉNO,PŘÍJMENÍ->DAT_NAROZ**.

Jakou další funkční závislost můžeme v této tabulce najít? Například, že plat zaměstnance závisí na funkci, kterou ve firmě vykonává (obecně to tak být nemusí, ale nechť v našem jednoduchém příkladu to tak je). Zapíšeme **FUNKCE->PLAT**. Jak takovou závislost interpretovat? Výše platu závisí na funkci, kterou člověk vykonává. Další závislost můžeme najít mezi osobou a datumem, od kdy ve firmě daná osoba pracuje. Zapíšeme tedy **JMÉNO,PŘÍJMENÍ->SMLOUVA_OD**. V této tabulce bychom mohli nalézt i další závislosti.

V uvedených příkladech lze vyčíst, že funkční závislost nemusí být vždy jen jednoduchá, ale že jedna hodnota může být závislá na více než jedné hodnotě. V praxi je toto trochu nešikovné a proto se vždy snažíme o to, abych všechny hodnoty závisely pokud možno jenom na jedné hodnotě. Pojďme se zpátky podívat na tabulku **ZAMĚSTNANEC** a vezměme si číslo osoby. Každá osoba ve firmě má přiřazeno jedinečné číslo. Odtud můžeme uvažovat závislost **ČÍSLO->JMÉNO** a **ČÍSLO->PŘÍJMENÍ**. To, že tyto dvě hodnoty jména a příjmení jsou funkčně závislé na hodnotě atributu číslo si lze jednoduše ověřit. Vezmeme-li si například osobu s číslem 1, dostaneme osobu pod jménem Jan Novák. Vezmeme-li si osobu s jiným číslem, např. 3, dostaneme osobu

jménem Jan Nováček. Tedy, to jaká hodnota bude v atributu **JMÉNO** a **PŘÍJMENÍ** závisí na tom, jaká hodnota bude v atributu **ČÍSLO**.

Funkční závislost je tranzitivní. Co to znamená? Schématicky zapsáno:
když A->B a zároveň B->C, pak z toho plyne, že A->C.

Vezmeme-li náš příklad a uvážíme následující závislosti:

- **ČÍSLO->JMÉNO,PŘÍJMENÍ**
- **JMÉNO,PŘÍJMENÍ->DAT_NAROZ**

Odtud dostaneme **ČÍSLO->DAT_NAROZ**. Zbývá ověřit, zdali opravdu hodnota **DAT_NAROZ** závisí na hodnotě **ČÍSLO**. Musíme se opět podívat na obsah tabulky **ZAMĚSTNANEC**. Vezmeme-li osobu s číslem 1, dostaneme její datum narození 15.10.1975, vezmeme-li osobu s číslem 4, dostaneme datum narození 5.12.1973. Když vezmeme jinou hodnotu atributu **ČÍSLO**, dostaneme obecně jinou hodnotu datumu narození. Tedy datum narození je závislé na čísle osoby.

Primární klíč

Primární klíč je vybraný jeden nebo více atributů, na jehož nebo jejichž hodnotách závisí všechny ostatní hodnoty atributů. Jinými slovy, na hodnotě primární klíče závisí hodnoty ostatních sloupců na jednom řádku v tabulce. Primární klíč má tedy pro každý řádek v tabulce jednoznačnou hodnotu.

Zbývá vyřešit otázku, jak takový primární klíč v tabulce nalézt. Buď takový atribut nalezneme jednoduchým způsobem, vyplývajícím z definice takového atributu (např. obsahuje-li tabulka sloupec **RODNÉ_ČÍSLO**, pak z povahy rodného čísla lze usoudit, že bude jednoznačné pro každý řádek v tabulce osob). Pokud takový atribut nebo skupinu atributů nemůžeme nalézt, pak se většinou postupuje tak, že se do tabulky přidá nový sloupec (na začátek) s názvem **ČÍSLO**, nebo **ID** (od slova identifikátor) a provedeme jednoduché „očíslování“ jednotlivých řádků. Na každý řádek zapíšeme jednoznačné číslo. Začneme například od hodnoty 1 a pokračujeme vždy zvyšováním o jedničku až k poslednímu záznamu. V našem příkladu tabulky **ZAMĚSTNANEC** již takový sloupec (s očíslováním řádků) máme. Je jím první sloupec **ČÍSLO**. Pro každý řádek je jeho hodnota jednoznačná, a aplikací funkční závislosti (včetně případné tranzitivity) ověříme, že všechny ostatní hodnoty na řádku funkčně závisí na hodnotě sloupec **ČÍSLO**. Tedy podmínka pro primární klíč je splněna a sloupec **ČÍSLO** bude v této tabulce primárním klíčem.

Při definici tabulek budeme skutečnost, že atribut **ČÍSLO** je primárním klíčem zapisovat: **PRIMARY KEY(ČÍSLO)**.

Cizí klíč

Cizí klíč je také speciálním typem atributu, jeho význam je odlišný, než je význam primárního klíče, i když s primárním klíčem poměrně hodně souvisí. Hodnota atributu, který je cizím klíčem, odkazuje na již existující hodnotu atributu, který je primárním klíčem v jiné (tzv. referencované) tabulce. Vezměme si náš příklad tabulky **ZAMĚSTNANEC** s mírně modifikovaným obsahem:

Tabulka: ZAMĚSTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	SMLOUVA_OD
1	Jan	Novák	15.10.1975	100	1.1.2000
2	Petr	Nový	1.4.1978	101	12.5.1999
3	Jan	Nováček	6.9.1965	104	7.7.1998
4	David	Vokurka	5.12.1973	102	1.10.2002

Kromě této tabulky mějme ještě novou tabulku **FUNKCE**, která bude popisovat detailněji jednotlivé funkce, resp. jaký plat pro každou funkci je stanoven:

Tabulka: FUNKCE

ID	NÁZEV	PLAT
100	uklízeč	15000
101	programátor	21500
102	ředitel	28000
104	manažer	17500

Původní tabulka **ZAMĚSTNANEC** už neobsahuje sloupce **FUNKCE** a **PLAT**, nýbrž jenom „identifikaci“ (číslo funkce), o jakou funkci se jedná. A teprve z tabulky **FUNKCE** můžeme zjistit pro každou funkci (pro její číslo), o jakou funkci se jedná (název funkce) a jaký plat bude pobírat osoba, kterou takovou funkci bude ve firmě vykonávat.

Primárním klíčem v tabulce **FUNKCE** je **ID**. Primárním klíčem v tabulce **ZAMĚSTNANEC** je sloupec **ČÍSLO**.

Cizím klíčem v tabulce **ZAMĚSTNANEC** je sloupec **ID_FUNKCE**. Hodnota tohoto sloupce na každém řádku „odkazuje“ na existující hodnotu sloupce **ID** v tabulce **FUNKCE** (tato tabulka je referencovaná).

Zaměstnanec Jan Novák vykonává funkci, jejíž **ID** je rovno 100 a z tabulky **FUNKCE** zjistíme, že funkce, jejíž **ID** je 100, je uklízeč a má plat 15000. Podobně můžeme vysledovat další informace u dalších záznamů.

Cizích klíčů může být v jedné tabulce více, mohou odkazovat do různých jiných tabulek, dokonce mohou odkazovat na „sebe sama“ tabulku.

Při definici tabulek skutečnost, že sloupec **ID_FUNKCE** je cizím klíčem, který odkazuje do tabulky **FUNKCE** na sloupec **ID**, zapisovat: **FOREIGN KEY(ID_FUNKCE) REFERENCES FUNKCE(ID)**.

Integritní omezení

Relace, respektive tabulky nám popisují strukturu dat. Definicí tabulek určíme, jak budou data strukturována a zorganizována. Kromě toho ale potřebujeme někdy také zajistit, aby se do tabulek vkládala jenom ta „správná“ data. Mechanismus, který nám to umožní, se nazývá integritní omezení.

Otázkou je, co jsou to „správná“ data. Jedná se o sloupce v tabulkách, u kterých můžeme dopředu předpokládat splnění určitých podmínek:

- Nenulová hodnota
- Jednoznačná hodnota
- Výchozí hodnota (tzv. „default“)
- Primární klíč
- Cizí klíč
- Podmínka, že hodnota je větší než …
- A jiné

Nenulová hodnota – NOT NULL

Toto integritní omezení u daného sloupce říká, že pokud bude proveden pokus o vložení záznamu do databáze s prázdnou hodnotou, databázový systém bude reagovat vygenerováním chyby a vložení takového záznamu nedovolí. Vezměme se náš původní příklad tabulky **ZAMĚSTNANEC** s následujícím obsahem:

Tabulka: ZAMĚSTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	PLAT	SMLOUVA_OD
1	Jan	Novák	15.10.1975	15000	1.1.2000
2	Petr	Nový	1.4.1978	21500	12.5.1999
3	Jan	Nováček	6.9.1965	17500	7.7.1998
4	David	Vokurka	5.12.1973	28000	1.10.2002

A uvažme, že na sloupce **JMÉNO** a **PŘÍJMENÍ** chceme aplikovat integritní omezení **NOT NULL**. Nyní bychom se pokusili vložit do této tabulky nový záznam pro nového zaměstnance, o kterém bychom měli následující údaje: osobní číslo 5, jméno nevíme, příjmení Dvořák, datum narození 3.2.1979, plat 30000 a datum zahájení pracovního poměru 1.1.2006. Snažili bychom se tedy vložit nový záznam, který vypadá následovně:

```
(5, , Dvořák, 3.2.1979, 30000, 1.1.2006)
```

Databázový systém vygeneruje chybovou hlášku, neboť došlo k porušení podmínky integritního omezení, a to takového, že **JMÉNO** musí být **NOT NULL**. Záznam se do této tabulky nevloží. Sloupec **PŘÍJMENÍ** má také integritní omezení **NOT NULL**, ale ve vkládaném záznamu je jeho hodnota vyplněna (Dvořák), tedy zde je všechno v pořádku.

Jednoznačná hodnota – UNIQUE

Integritní omezení **UNIQUE** dáváme k těm sloupcům, kde požadujeme, aby hodnota takového sloupce na každém řádku byla jedinečná. Vezmeme-li náš příklad se zaměstnanci, přidáme do tabulky nový sloupec pro rodné číslo. Na tento sloupec aplikujeme integritní omezení **UNIQUE** – databázový systém už při vkládání nebo aktualizaci záznamů zajistí, aby se nikdy nestalo, že by na dvou různých řádcích byla hodnota rodného čísla shodná.

Výchozí hodnota – DEFAULT 'hodnota'

Někdy může být výhodné, pro úsporu času při zadávání nových záznamů, nastavit výchozí hodnotu pro nějaký sloupec, která se použije v případě, že uživatel danou hodnotu nevyplní. To je vhodné například všude tam, kde mnoho řádků v tabulce bude mít většinou některé sloupce vyplněné na stejnou hodnotu. Rozšiřme si náš příklad o sloupec **TYP_UVAZKU**, který bude charakterizovat, na jaký úvazek daný zaměstnanec u nás ve firmě pracuje.

Tabulka: ZAMĚSTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	PLAT	SMLOUVA_OD	TYP_UVAZKU
1	Jan	Novák	15.10.1975	15000	1.1.2000	F
2	Petr	Nový	1.4.1978	21500	12.5.1999	F
3	Jan	Nováček	6.9.1965	17500	7.7.1998	H
4	David	Vokurka	5.12.1973	28000	1.10.2002	F

Budeme uvažovat hodnoty „**F**" – pro celý úvazek, „**H**" – pro poloviční úvazek a „**P**" – pro částečný úvazek, menší než poloviční. Běžný zaměstnanec bude u nás pracovat na plný úvazek, tedy můžeme zavést integritní omezení pro **TYP_UVAZKU** jako **DEFAULT 'F'** a pak při pokusu o vložení záznamu, kde nebude specifikována poslední položka se automaticky použije hodnota „**F**".

Po úspěšném vložení záznamu **(5, Vít, Mrkvička, 30.1.1970, 27000, 1.1.2006)** bude tabulka rozšířena o nový řádek:

Tabulka: ZAMĚSTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	PLAT	SMLOUVA_OD	TYP_UVAZKU
1	Jan	Novák	15.10.1975	15000	1.1.2000	F
2	Petr	Nový	1.4.1978	21500	12.5.1999	F
3	Jan	Nováček	6.9.1965	17500	7.7.1998	H
4	David	Vokurka	5.12.1973	28000	1.10.2002	F
5	Vít	Mrkvička	30.1.1970	27000	1.1.2006	F

Primární klíč – PRIMARY KEY(sloupec)

Primární klíč je také integritní omezením. Jeho význam jsme si již vysvětlili v dřívější kapitole. Je-li nějaký sloupec (nebo skupina sloupců) označen jako primární klíč, databázový systém při vkládání nebo aktualizaci záznamů kontroluje zda-li neustále platí,

že hodnota sloupce na daném řádku je v rámci celé tabulky jedinečná (tzn. **UNIQUE**) a že je vyplněná (tedy **NOT NULL**).

Tvrzení: primární klíč vždy existuje. Důkaz: může být zvolen jeden atribut z celé n-tice atributů, které tvoří řádek v tabulce, nebo jako m-tice atributů, kde m≤n. V nejhorším případě (pokud žádný primární klíč nenalezneme) může být primárním klíčem celá n-tice atributů, nebo-li celý řádek, který je sám sobě klíčem. Je to pravda? Ano, je. V tabulce vylučujeme duplicitní řádky. Žádný řádek se nemůže v tabulce vyskytnout dvakrát a vícekrát, tedy každý řádek v rámci tabulky je jednoznačný, každý řádek je vyplněn (alespoň nějaký z atributů má hodnotu vyplněnou) a tedy celý řádek splňuje podmínky kladené na primární klíč. Primární klíč tedy v každé tabulce existuje.

Cizí klíč – FOREIGN KEY(sloupec) REFERENCES tabulka(sloupec)

Atribut typu cizí klíč slouží k „provazování“ tabulek. Hodnota cizího klíče na daném řádku musí „ukazovat“ na existující hodnotu v referencované tabulce. Vezměme si náš příklad se dvěma tabulkami:

Tabulka: ZAMĚSTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	SMLOUVA_OD
1	Jan	Novák	15.10.1975	100	1.1.2000
2	Petr	Nový	1.4.1978	101	12.5.1999
3	Jan	Nováček	6.9.1965	104	7.7.1998
4	David	Vokurka	5.12.1973	102	1.10.2002

Tabulka: FUNKCE

ID	NÁZEV	PLAT
100	uklízeč	15000
101	programátor	21500
102	ředitel	28000
104	manažer	17500

Databázový systém i zde hlídá, aby byly odkazovány existující hodnoty. V případě, že bychom se pokusili například do tabulky **ZAMĚSTNANEC** vložit nový záznam s **ID_FUNKCE** rovno **105**, systém by vygeneroval chybovou hlášku a vložení záznamu by nedovolil. Důvodem by totiž bylo, že funkce s **ID** rovno **105** v tabulce **FUNKCE** neexistuje.

Podmínka – CHECK(podmínka)

Pomocí tohoto integritního omezení můžeme například zapsat, že hodnota sloupce, na které je toto integritní omezení aplikováno, musí být větší než **10000**. Pokud bychom tedy měli ve firmě ustanovení, že minimální plat je 10 000 Kč, potom na sloupec **PLAT** v tabulce **FUNKCE** v předchozím uvedeném příkladu aplikujeme **CHECK(PLAT>=10000)**.

Další integritní omezení

Kromě integritní omezeních, které můžeme aplikovat na konkrétní sloupce a které se vyhodnocují v kontextu každého řádku, můžeme mít také integritní omezení týkající se celé tabulky. Například:

- Ve firmě může pracovat maximálně 50 zaměstnanců (to znamená, že tabulka **ZAMĚSTNANEC** smí obsahovat maximálně 50 záznamů)
- V daném kině mohou být zároveň promítány 3 filmy (jedná se o podmínku aplikovanou přes 3 řádky v tabulce pro promítání filmů)
- Každou knihu napsalo maximálně 5 autorů (tj. existuje max. 5 odkazů na **ID** autora z tabulky **KNIHA** do tabulky **AUTOR**)

Vazby mezi tabulkami

Tak jako jsme řešili vazby mezi entitami, podobně můžeme uvažovat v kontextu tabulek. Také zde budeme řešit tři základní typy vazeb: 1:1, 1:N a M:N.

Vazba 1:1

Vezměme si entity **OSOBA** a **PLATNÝ_OBČANSKÝ_PRůKAZ**. Už z kapitoly o datovém modelování víme, že tyto entity jsou mezi sebou ve vazbě 1:1. Vytvořme si pro tyto entity tabulky, které je budou popisovat:

Tabulka: OSOBA

ID	JMÉNO	PŘÍJMENÍ	DATUM_NAROZENÍ
1	Jan	Novák	15.10.1975
2	Petr	Nový	1.4.1978
3	Jan	Nováček	6.9.1965
4	David	Vokurka	5.12.1973
5	Vít	Mrkvička	30.1.1970

Tabulka: OBČANSKÝ_PRůKAZ

ID	JMÉNO	PŘÍJMENÍ	ČÍSLO_OP	DATUM_VYDÁNÍ_OP
1	Jan	Novák	3490297	1.1.2004
2	Petr	Nový	9830393	23.4.1997
3	Jan	Nováček	7800023	12.2.2005
4	David	Vokurka	9837209	4.12.1999
5	Vít	Mrkvička	3300278	8.8.1990

Vazba 1:1 znamená, že ke každému záznamu z jedné tabulky „odpovídá“ přesně jeden záznam v tabulce druhé. Pokud je to takto, nic nám nebrání ovšem tyto dvě tabulky sloučit dohromady:

Tabulka: OSOBA_OP

ID	JMÉNO	PŘÍJMENÍ	DATUM_NAROZENÍ	ČÍSLO_OP	DATUM_VYDÁNÍ_OP
1	Jan	Novák	15.10.1975	3490297	1.1.2004
2	Petr	Nový	1.4.1978	9830393	23.4.1997
3	Jan	Nováček	6.9.1965	7800023	12.2.2005
4	David	Vokurka	5.12.1973	9837209	4.12.1999
5	Vít	Mrkvička	30.1.1970	3300278	8.8.1990

Když se podíváme na novou tabulku **OSOBA_OP**, zjistíme, že nedošlo k žádné újmě na obecnosti ani konkrétnosti. Tabulka **OSOBA_OP** má tutéž informační hodnotu, zachycuje nám všechny údaje z původních dvou tabulek. Jediné, co se změnilo, je úspora místa, neboť nám zmizely „redundantní“ sloupce (**JMÉNO**, **PŘÍJMENÍ**)

V praxi opravdu vazbu 1:1 realizujeme v tabulkách tak, že tuto vazbu „schováme“ do atributů. Zpravidla tedy vytvoříme jen jednu tabulku, která bude obsahovat atributy z obou původních, jak jsme to provedli na našem příkladě.

Vazba 1:N

Vazbu 1:N realizujeme pomocí „provázané“ dvojice primární a cizí klíč. Vezměme si náš příklad z kapitoly o datovém modelování, entity **OSOBA** a **KREDITNÍ_KARTA**. O těchto entitách již víme, že jsou mezi sebou ve vzájemné vazbě 1:N (jedna osoba může vlastnit více kreditních karet, ale ne naopak). Nechť atributy pro entitu **OSOBA** jsou: její **ID**, jméno, příjmení, datum narození a atributy entity **KREDITNÍ_KARTA** jsou číslo karty, vytištěné jméno na kartě, typ karty a její platnost. Tabulky mohou vypadat následovně:

Tabulka: OSOBA

ID	JMÉNO	PŘÍJMENÍ	DATUM_NAROZENÍ
1	Jan	Novák	15.10.1975
2	Petr	Nový	1.4.1978
3	Jan	Nováček	6.9.1965
4	David	Vokurka	5.12.1973
5	Vít	Mrkvička	30.1.1970

Tabulka: KREDITNÍ_KARTA

ČÍSLO	TISK_JMÉNO	TYP	PLATNOST_DO
23090	Jan Novák	Visa	09/07
98021	Ing. Novák	MasterCard/EuroCard	05/06
11102	Mrkvička V.	Visa Electron	10/08
45000	Vokurka D. Mgr.	Visa Gold	09/09

Máme tedy obě tabulky a nyní potřebujeme zaznamenat onu vazbu 1:N. Připomeňme si, že u vazby 1:N na „straně 1“ stojí tabulka **OSOBA** a na „straně N“ je tabulka **KREDITNÍ_KARTA**. Abychom mohli zrealizovat vazbu 1:N, musíme do tabulky stojící na „straně N“ vložit ještě jeden sloupec, který bude cizím klíčem odkazující na primární klíč do tabulky stojící na „straně 1“. V našem případě vložíme do tabulky **KREDITNÍ_KARTA** nový sloupec **ID_OS** (jako identifikace osoby, které daná kreditní karta patří), který bude odkazovat do tabulky **OSOBA** na její primární klíč, tedy sloupec **ID**.

Tabulky se zakomponovanou vazbou 1:N budou nakonec vypadat takto:

Tabulka: OSOBA

ID	JMÉNO	PŘÍJMENÍ	DATUM_NAROZENÍ
1	Jan	Novák	15.10.1975
2	Petr	Nový	1.4.1978
3	Jan	Nováček	6.9.1965
4	David	Vokurka	5.12.1973
5	Vít	Mrkvička	30.1.1970

Tabulka: KREDITNÍ_KARTA

ČÍSLO	TISK_JMÉNO	TYP	PLATNOST_DO	ID_OS
23090	Jan Novák	Visa	09/07	1
98021	Ing. Novák	MasterCard/EuroCard	05/06	1
11102	Mrkvička V.	Visa Electron	10/08	5
45000	Vokurka D. Mgr.	Visa Gold	09/09	4

Z pohledu na obsah obou tabulek zjistíme, že například osoba Jan Novák vlastní 2 kreditní karty (přesně: na prvním řádku tabulky **KREDITNÍ_KARTA** vidíme, že hodnota sloupce **ID_OS** je rovna 1. Z tabulky **OSOBA** pak zjistíme, že řádek s hodnotou **ID** rovno 1 je záznam pro Jana Nováka. Odtud tedy informace, že danou kreditní kartu vlastní Jan Novák. Podobně bychom vycházeli z druhého řádku v tabulce **KREDITNÍ_KARTA**, kde je opět **ID_OS** rovno 1, které patří osobě Jan Novák). Pánové Mrkvička a Vokurka vlastní po jedné kreditní kartě.

Vazba M:N

Jak jsme si uvedli v kapitole o datovém modelování, vazba M:N je nejběžnějším typem vazby vyskytující se mezi entitami. I v tabulkovém schématu s tímto typem vazby musíme nejvíce počítat, proto následujícím řádkům, jak konstruovat vazbu M:N mezi tabulkami, věnujte náležitou pozornost.

Vezměme si entity **KNIHA** a **AUTOR**. Entitou **KNIHA** máme na mysli konkrétní dílo daného názvu jednoho nebo více autorů. Autorem je každá osoba, která napsala alespoň jednu knihu, ať už jako výlučný autor, nebo spoluautor.

Pro tyto entity zaveďme tabulky **KNIHA** a **AUTOR**. Sloupce v těchto tabulkách budou vycházet ze základních atributů entit. Atributy knihy budou název, jazyk originálu, a kdo ji napsal(i). Atributy autora budou jeho jméno, příjmení a datum narození a také seznam jeho děl, která napsal.

Zaveďme se následující tabulky **KNIHA** a **AUTOR** s některými jejich sloupci, které jsou svou povahou jednoduché a nebudou nám činit potíže:

Tabulka: KNIHA

ID	NÁZEV	JAZYK
11	Oko	český
12	Bludiště	německý
13	Domek	český
14	Sauna a bazén	anglický

Tabulka: AUTOR

ID	JMÉNO	PŘÍJMENÍ	DATUM_NAROZENÍ
1	Jan	Novák	15.10.1975
2	Petr	Nový	1.4.1978
3	Jan	Nováček	6.9.1965
4	David	Vokurka	5.12.1973
5	Vít	Mrkvička	30.1.1970

Kdybychom stanovili, že každá kniha byla napsána právě jedním autorem (tj. ne více autory), pak by celá situace byla výrazně jednoduší, neboť by se jednalo o vazbu 1:N (jeden autor napsal více knih, ale jedna konkrétní kniha byla napsána právě jedním autorem), kterou už umíme mezi tabulkami zrealizovat. Do tabulky **KNIHA** bychom přidali sloupec **ID_AUTOR**, který by odkazoval do tabulky **AUTOR** na její primární klíč – sloupec **ID**:

Tabulka: KNIHA

ID	NÁZEV	JAZYK	ID_AUTOR
11	Oko	český	1
12	Bludiště	německý	4
13	Domek	český	4
14	Sauna a bazén	anglický	3

Tabulka: AUTOR

ID	JMÉNO	PŘÍJMENÍ	DATUM_NAROZENÍ
1	Jan	Novák	15.10.1975
2	Petr	Nový	1.4.1978
3	Jan	Nováček	6.9.1965
4	David	Vokurka	5.12.1973
5	Vít	Mrkvička	30.1.1970

Ze záznamů v obou tabulkách zjistíme, že například pan Nováček je autorem knih Bludiště a Domek č.1. Jak se ale nyní vypořádat se situací, kdy potřebujeme zajistit, abychom mohli zaznamenat také skutečnost, že například kniha Oko byla napsána jednak Janem Novákem ale také Vítem Mrkvičkou? Někoho by v tomto okamžiku mohlo napadnout, co zavést další sloupec do tabulky **KNIHA** a to **ID_AUTOR2**. Tam kde by byl druhý autor, bylo by uvedeno jeho **ID**, tam, kde by již druhý autor knihy nebyl, byla by dosazena například hodnota **0**. Výsledek by vypadal následovně:

Tabulka: KNIHA

ID	NÁZEV	JAZYK	ID_AUTOR	ID_AUTOR2
11	Oko	český	1	5
12	Bludiště	německý	4	0
13	Domek	český	4	0
14	Sauna a bazén	anglický	3	2

Tabulka: AUTOR

ID	JMÉNO	PŘÍJMENÍ	DATUM_NAROZENÍ
1	Jan	Novák	15.10.1975
2	Petr	Nový	1.4.1978
3	Jan	Nováček	6.9.1965
4	David	Vokurka	5.12.1973
5	Vít	Mrkvička	30.1.1970

Co když ale bude existovat nějaká kniha, kterou napsali 3 lidé? Mohli bychom zase rozšířit databázovou tabulku **KNIHA** o další sloupec. Nicméně tento postup, jak už asi sami vidíte, nikam nevede. Hlavním důvodem je skutečnost, že při návrhu databáze dopředu nevíme, kolik autorů může nějaká kniha mít. Musíme tedy přistoupit k jiném řešení, které je dostatečně obecné a je schopno zachytit jakoukoliv situaci. Přesněji řečeno je schopno správně zachytit vazbu M:N.

Řešením je vytvoření úplně nové pomocné „mezi-tabulky“, která se prováže s oběma původními. Tato „mezi-tabulka“ nám správně vyřeší problém vazby M:N.

Nazvěme tuto mezi-tabulku **AUTORSTVÍ**. Tato tabulka nám bude sloužit k zaznamenávání vazby M:N, tedy k tomu, kdo napsal (jaký autor) jakou knihu. Vezměme si původní tabulky **KNIHA** a **AUTOR** a rozšiřme si toto schéma o tabulku **AUTORSTVÍ**:

Tabulka: KNIHA

ID	NÁZEV	JAZYK	ID_AUTOR
11	Oko	český	1
12	Bludiště	německý	4
13	Domek	český	4
14	Sauna a bazén	anglický	3

Tabulka: AUTORSTVÍ

ID_KNIHA	ID_AUTOR
11	1
11	5
14	3
12	4
13	4

Tabulka: AUTOR

ID	JMÉNO	PŘÍJMENÍ	DATUM_NAROZENÍ
1	Jan	Novák	15.10.1975
2	Petr	Nový	1.4.1978
3	Jan	Nováček	6.9.1965
4	David	Vokurka	5.12.1973
5	Vít	Mrkvička	30.1.1970

Nyní se podrobně zaměřme na tabulku **AUTORSTVÍ**, neboť její pochopení v kontextu tohoto příkladu je velmi důležité pro pochopení konstrukce vazby M:N mezi tabulkami obecně.

Důležitým výchozím bodem jsou „očíslované“ záznamy v tabulkách **KNIHA** a **AUTOR**. Ono očíslování tvoří primární klíče. Na každém řádku v obou tabulkách jsou hodnoty těchto primárních klíčů jedinečné. Z tabulky **AUTORSTVÍ** se tedy můžeme do obou tabulek jednoznačně odkazovat na libovolné jejich řádky.

Podívejme se na první řádek v tabulce **AUTORSTVÍ**. Co nám tento řádek říká? Jednoduše řečeno, nese informaci, že knihu č. 11 napsal autor č.1. Volně přeloženo, pokud vezmeme v úvahu, že kniha č.11 je dílo s názvem Oko (vyčteme z tabulky **KNIHA**) a autor č.1 je Jan Novák (vyčteme z tabulky **AUTOR**), dostáváme interpretaci záznamu (11, 1) z tabulky **AUTORSTVÍ**: „Knihu s názvem Oko napsal Jan Novák“.

Pojďme na další řádek. Na něm je uložen záznam (11, 5). Tedy knihu č. 11 napsal autor č. 5. Kniha č. 11 je opět kniha Oko (zjistíme z tabulky **KNIHA**) a autor č. 5 je Vít Mrkvička. Dostáváme tedy interpretaci záznamu „Knihu s názvem Oko napsal Vít Mrkvička“. Z obou tvrzení nám tedy plyne, že kniha Oko byla napsána celkem 2 autory, tím prvním je Jan Novák a tím druhým je Vít Mrkvička.

Podívejme se na 4. řádek v tabulce **AUTORSTVÍ**. Na něm je uložen záznam (12, 4). Kniha č. 12 je dílo s názvem Bludiště (zjistíme z tabulky **KNIHA**), autor č. 4 je David Vokurka (zjistíme z tabulky **AUTOR**). Dostaneme tedy výsledek: „knihu Bludiště napsal David Vokurka“.

Pojďme ještě na řádek č. 5 v tabulce **AUTORSTVÍ**. Zde je uložen záznam (13, 4). Kniha č. 13 je „Domek“, autor č. 4 je opět David Vokurka. Dostaneme „Knihu Domek napsal David Vokurka“. Máme tu opět dvě tvrzení, která jsou o Davidovi Vokurkovi, odtud nám plyne, že David Vokurka napsal knihu Bludiště a knihu Domek. Tedy autor David Vokurka je autorem 2 knih.

Do tabulky **AUTORSTVÍ** můžeme postupně přidávat libovolné záznamy podle toho, kolik autorů má jedna kniha, nebo kolik knih napsal jeden autor. Tato tabulka nám dostatečně obecně umožňuje zaznamenat vazbu M:N. S přibývajícími autory nebo knihami nemusíme rozšiřovat nějaké tabulky o další sloupce (jak jsme se o to pokoušeli výše), ale stačí pouze přidat nové řádky do tabulky **AUTORSTVÍ**.

Vazba M:N je nejběžnějším typem vazby mezi tabulkami. Proto konstrukce mezi-tabulky pro realizaci této vazby budou velmi časté.

Fyzický datový model

Fyzický datový model představuje pohled na databázi o úroveň níž než tomu bylo u datového modelu logického. Zde se tedy zaměřujeme již na implementační úroveň. Fyzický datový model reprezentuje soubor tabulek včetně jejich vazeb (a včetně pomocných mezi-tabulek pro realizaci vazeb M:N). Každá tabulka musí být podrobně popsána. Popis musí obsahovat:

- Seznam všech sloupců
- Řádně vyznačený primární klíč (jeden nebo více sloupců)
- Řádně označené všechny cizí klíče (včetně referencí na jiné tabulky)
- U každého sloupce uveden jeho typ a integritní omezení

Důležité je také pořadí, ve kterém tabulky definujeme. Platí základní pravidlo, že tabulka, na kterou se odkazujeme přes nějaký cizí klíč z jiné tabulky, musí být vytvořena dříve, než tabulka, ze které se odkazujeme.

V praxi bývá také zvykem definovat všechny názvy tabulek a sloupců jako jednoslovné (pomůžeme si znakem „_“ v případě potřeby) a bez diakritiky.

Nyní nám zbývá uvést příklad, jak budeme jednotlivé tabulky popisovat. Pro lepší názornost si vezměme již existující tabulky z našich příkladů v této kapitole a uveďme se zápis jejich popisu.

Tabulky: ZAMĚSTNANEC a FUNKCE

Tabulka pro evidenci zaměstnanců ve firmě a tabulka pro jednotlivé funkce, mají následující strukturu a obsah:

Tabulka: ZAMĚSTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	SMLOUVA_OD
1	Jan	Novák	15.10.1975	100	1.1.2000
2	Petr	Nový	1.4.1978	101	12.5.1999
3	Jan	Nováček	6.9.1965	104	7.7.1998
4	David	Vokurka	5.12.1973	102	1.10.2002
5	Vít	Mrkvička	30.1.1970	107	1.1.2006

Tabulka: FUNKCE

ID	NÁZEV	PLAT
100	uklízeč	15000
101	programátor	21500
102	ředitel	28000
104	manažer	17500
107	náměstek	27000

Tabulka **ZAMĚSTNANEC** obsahuje celkem 6 sloupců. Sloupec **ČÍSLO** v této tabulce je primárním klíčem. Sloupec **ID_FUNKCE** je cizím klíčem odkazujícím do tabulky **FUNKCE** na její sloupec **ID**. Tabulka **FUNKCE** obsahuje celkem 3 sloupce. Sloupec **ID** v této tabulce je primárním klíčem.

Integritní omezení mohou být následující:

- **JMÉNO**, **PŘÍJMENÍ**, **DAT_NAROZ**, **SMLOUVA_OD** by neměly být prázdné
- **PLAT** musí být zadán a musí být vyšší než **10000**

Nyní máme shrnuty všechny poznatky o těchto dvou tabulkách a můžeme tyto tabulky popsat následujícím způsobem (uvedený zápis berte jako vzorový) – nejprve vytváříme tabulku **FUNKCE** (odkazovaná) a po té tabulku **ZAMĚSTNANEC** (odkazuje).

FUNKCE		
ID	**INTEGER**	**PRIMARY KEY(ID)**
NÁZEV	**VARCHAR(40)**	
PLAT	**INTEGER**	**CHECK(PLAT>=10000)**

ZAMĚSTNANEC		
ČÍSLO	**INTEGER**	**PRIMARY KEY(ČÍSLO)**
JMÉNO	**VARCHAR(10)**	**NOT NULL**
PŘÍJMENÍ	**VARCHAR(20)**	**NOT NULL**
DAT_NAROZ	**DATE**	**NOT NULL**
ID_FUNKCE	**INTEGER**	**FOREIGN KEY(ID_FUNKCE) REFERENCES FUNKCE(ID)**
SMLOUVA_OD	**DATE**	**NOT NULL**

Porovnání logického a fyzického datového modelu

Pro názornější přehled, jak a v čem se liší od sebe logický a fyzický datový model, je uvedena následující porovnávací tabulka:

Logický datový model	Fyzický datový model
LDM zachycuje zkoumané entity včetně jejich vazeb 1:1, 1:N a M:N	FDM zachycuje všechny tabulky včetně jejich vazeb 1:1 a 1:N, dále také všechny mezi-tabulky potřebné pro realizaci vazby M:N
U každé entity uvádíme základní atributy (vlastnosti entity)	V každé tabulce uvádíme všechny sloupce (atributy), včetně těch, které slouží k realizaci vazeb mezi tabulkami (primární a zejména pak cizí klíče)

U atributů entit nás zajímá jeho základní typ (např. číslo, řetězec, pravdivostní hodnota, apod.)	U jednotlivých sloupců nás zajímá konkrétní datový typ, včetně rozsahů a způsobu reprezentace daného typu (např. číslo – celé, reálné, kladné, …, řetězec včetně počtu znaků, pravdivostní hodnota realizována jako jednoznakový řetězec s hodnotami „A“ nebo „N“, atd.
Znázorňujeme většinou graficky a slovním popisem	Znázorňujeme většinou textově, schématicky.

Řešené úlohy

Nyní si uvedeme dvě ukázkově řešení úlohy na konstrukci vzájemně provázaných databázových tabulek.

Úloha: Vlakové trasy

Navrhněte tabulku **VLAKOVA_TRASA**, která bude popisovat trasy z Brna do dalších měst. Města budou definována v tabulce **MESTO**, každé měst kromě svého názvu bude mít u sebe uvedeno, v jakém leží tarifním pásmu. Pásma budou definována v tabulce **PASMO** (číselník). Každé pásmo kromě intervalu km (od, do) bude mít také u sebe uvedenou cenu v Kč za jízdné. Tabulka **VLAKOVA_TRASA** bude obsahovat pro každou trasu identifikaci, do které města vlak jede, číslo vlaku, který danou trasu obsluhuje a čas pravidelného odjezdu. Určete správně primární a cizí klíče tabulky, dbejte na správné provázání tabulek a stanovte IO u těch atributů, kde se to bude jevit vhodné.

Řešení:

Zkonstruujeme celkem 3 tabulky dle uvedeného zadání. Pořadí tabulek je následující: **PASMO**, **MESTO**, **VLAKOVA_TRASA** (z tabulky **VLAKOVA_TRASA** je odkazováno přes klíč do tabulky **MESTO** a z tabulky **MESTO** je odkazováno přes klíč do tabulky **PASMO** – odtud tedy pořadí vytvářených tabulek). U všech sloupců také určíme jejich datový typ a integritní omezení.

PASMO		
ID	**INTEGER**	**PRIMARY KEY(ID)**
KM_OD	**INTEGER**	**NOT NULL**
KM_DO	**INTEGER**	
JIZDNE	**FLOAT(2)**	**NOT NULL**

MESTO		
ID	INTEGER	PRIMARY KEY(ID)
NAZEV	VARCHAR(40)	NOT NULL
ID_PASMO	INTEGER	FOREIGN KEY(ID_PASMO) REFERENCES PASMO(ID)

VLAKOVA_TRASA		
ID	INTEGER	PRIMARY KEY(ID)
ID_MESTO	INTEGER	FOREIGN KEY(ID_MESTO) REFERENCES MESTO(ID)
CISLO_VLAK	VARCHAR(4)	NOT NULL
CAS_ODJEZD	TIME	NOT NULL

Úloha: Geometrické obrazce

Navrhněte tabulku pro popis geometrických obrazů v rovině (2D). Geometrickými obrazcemi rozumíme: úsečka, trojúhelník, čtverec, obdélník, pětiúhelník a šestiúhelník. Číselník geometrických obrazců bude v tabulce **TYP_OBRAZEC**, definice jednotlivých bodů v rovině v tabulce **BOD** pro každý obrazec a nakonec hlavní tabulka **OBRAZEC**, která bude popisovat konkrétní obrazce. Každý obrazec bude mít svůj název, barvu (nepovinná) a bude popsán jednotlivými body z tabulky **BOD**.

Řešení:

Zkonstruujeme čtyři tabulky dle zadání, budeme opět dbát na správné pořadí vytvářených tabulek, vyznačíme všechny primární a cizí klíče a všechna integritní omezení, tam kde budou vhodná.

TYP_OBRAZEC		
ID	INTEGER	PRIMARY KEY(ID)
NAZEV	VARCHAR(40)	NOT NULL

OBRAZEC		
ID	INTEGER	PRIMARY KEY(ID)
NAZEV	VARCHAR(80)	NOT NULL
BARVA	VARCHAR(20)	
ID_TYP	INTEGER	FOREIGN KEY(ID_TYP) REFERENCES TYP_OBRAZEC(ID)

BOD		
ID	INTEGER	PRIMARY KEY(ID)
X_SOURADNICE	INTEGER	NOT NULL
Y_SOURADNICE	INTEGER	NOT NULL
ID_OBRAZEC	INTEGER	FOREIGN KEY(ID_OBRAZEC) REFERENCES OBRAZEC(ID)

Uvedené řešení odpovídá situaci, že každý definovaný bod v rovině náleží právě jednomu obrazci. Jinými slovy, obrazce v rovině nemohou v tomto případě mít společný bod (např. že by jeden vrchol obdélníka byl shodný s jedním vrcholem trojúhelníka – tj.

obrazce obdélník a trojúhelník by se „dotýkaly“). Pokud bychom tedy vazbu mezi bodem v rovině a obrazcem chtěli z původního zadání (vazba 1:N) převést na vazbu M:N (tj. že jeden bod může být součástí více obrazců), pak musíme mezi tabulku **BOD** a **OBRAZEC** přidat jednu pomocnou mezi-tabulku a původní tabulky mírně modifikovat. Přidáme novou tabulku **BODY_OBRAZCE** a řešení bude vypadat následovně:

TYP_OBRAZEC		
ID	INTEGER	PRIMARY KEY(ID)
NAZEV	VARCHAR(40)	NOT NULL

OBRAZEC		
ID	INTEGER	PRIMARY KEY(ID)
NAZEV	VARCHAR(80)	NOT NULL
BARVA	VARCHAR(20)	
ID_TYP	INTEGER	FOREIGN KEY(ID_TYP) REFERENCES TYP_OBRAZEC(ID)

BOD		
ID	INTEGER	PRIMARY KEY(ID)
X_SOURADNICE	INTEGER	NOT NULL
Y_SOURADNICE	INTEGER	NOT NULL

BODY_OBRAZCE		
ID	INTEGER	PRIMARY KEY(ID)
ID_BOD	INTEGER	FOREIGN KEY(ID_BOD) REFERENCES BOD(ID)
ID_OBRAZEC	INTEGER	FOREIGN KEY(ID_OBRAZEC) REFERENCES OBRAZEC(ID)

Shrnutí

- **Základním stavebním prvkem relačních databází jsou relace. Relace R nad n atributy je libovolná podmnožina kartézského součinu domén atributů. Relace jsou reprezentovány tabulkami.**
- **Tabulka se skládá ze sloupců a řádků. Základními operacemi nad záznamy v tabulce jsou: vložení nového záznamu, smazání existujícího záznamu, aktualizace existujícího záznamu, vyhledání záznamu a dotaz na existenci záznamu.**
- **Funkční závislost nám říká, že hodnota jednoho sloupce (sloupců) je závislá na hodnotě jiného sloupce (sloupců). Funkční závislosti vyplývají z pozorování reálného světa.**
- **Speciálním atributem (sloupcem) v tabulce je primární klíč. Jeho hodnota je pro každý řádek jednoznačná. Primární klíč vždy existuje.**
- **Druhým speciálním atributem (sloupcem) v tabulce je cizí klíč. Hodnota tohoto klíče na každém řádku „ukazuje“ na hodnotu do jiné tabulky na hodnotu v sloupci, který je v té odkazované tabulce primárním klíčem.**
- **Integritní omezení je mechanismus, kterým databázový systém zajišťuje konzistentní data v databázi.**

- **Tabulky jsou ve vzájemných vazbách, sledujeme vazby 1:1, 1:N a M:N. Vazba M:N je realizována pomocí přidané mezi-tabulky.**

Otázky a úkoly

- Navrhněte tabulku **TRASA**. Tato tabulka popisuje trasy linkového autobusu. Každá trasa je obsluhována linkou daného čísla, někde začíná, někde končí, začíná v určitou hodinu, má specifikovaný počet cestujících, základní a zlevněnou cenu. Trasa je identifikována nějakým kódem, což je řetězec složený ze dvou písmen a 3 číslic. Určete primární klíč této tabulky. Rozhodněte o integritních omezeních jednotlivých sloupců.
- Navrhněte tabulku **ADRESA_BYT**. Tato tabulka velmi přesně popisuje adresu konkrétního bytu. Obsahuje následující položky: Ulice, číslo popisné, číslo orientační, psč, město, stát, číslo bytu, číslo podlaží. Navrhněte vhodný primární klíč této tabulky. Rozhodněte o integritních omezeních u položek, kde to bude vhodné.
- Navrhněte tabulku **LOKALITA**. Tato tabulka bude obsahovat pouze popis daného místa, kam jezdí linkový autobus. Navrhněte vhodný primární klíč této tabulky. Aktualizujte tabulku **TRASA** tak, že místo textové položky pro místo, kde trasa začíná a kde trasa končí, bude tabulka provázána se dvěma tabulkami **LOKALITA**. Správně stanovte cizí klíče a tabulky přes vazbu cizí-primární klíč dobře provažte.
- Navrhněte tabulku **ZBOZI**. V této tabulce se budou uchovávat informace o jednotlivém zboží v obchodě: Název zboží, dodavatel, původ (kde vyrobeno), základní popis (rozměry, hmotnost, apod.) Dále realizujte tabulku **SKLAD**, která bude odrážet aktuální skladové zásoby daného zboží. Pro každou položku budeme vést aktuální stav zásob (počet kusů), aktuální prodejní cenu, datum a čas posledního prodeje. Navrhněte vhodné primární klíče těchto tabulek. Správně stanovte cizí klíče a tabulky přes vazbu cizí-primární klíč dobře provažte. Rozhodněte o integritních omezeních u položek, kde to bude vhodné.
- Realizujte jednoduchý EN-CZ slovník. K tomu využijete následující tabulky: **SLOVA_CZ**, **SLOVA_EN**, **SLOVNI_DRUH** a **SLOVNIK**. Tabulka **SLOVA_CZ** bude obsahovat česká slovíčka. Každé slovíčko bude mít svůj primární klíč (jednoznačné ID) a dále u každého slovíčka bude informace o jaký slovní druh jde (to bude realizováno vazbou na tabulku **SLOVNI_DRUH**). Tabulka **SLOVA_EN** bude mít stejnou strukturu jako tabulka **SLOVA_CZ**, jenom slovíčka zde budou v angličtině. Hlavní tabulka **SLOVNIK** pak bude realizovat vlastní slovník. K jednomu slovíčku EN může být navázáno několik slovíček CZ a naopak k jednomu slovíčku CZ může být navázáno několik EN slovíček. Pozor na správné primární klíče, cizí klíče, vazby, integritní omezení.
- Realizujte obecný vícejazyčný slovník. Ve srovnání s příkladem číslo 5 je toto zadání mnohem komplikovanější. Budeme celkem potřebovat tyto tabulky: **SLOVO**, **JAZYK**, **SLOVNI_DRUH**, **SLOVNIK**. Tabulka **SLOVO** bude obsahovat konkrétní slova a u každého z nich bude informace, v jaké řečí (tj. vazba na tabulku **JAZYK**) to slovo je. Dále u každého slova bude informace o jaký slovní

druh jde (opět vazba na tabulky **SLOVNI_DRUH**). Tabulka **SLOVNIK** bude mít stejnou strukturu jako v příkladu číslo 4. Pozor na správné primární klíče, cizí klíče, vazby, integritní omezení.

- Realizujte relační schéma pro Kalendář. Tabulka **KALENDAR** bude obsahovat seznam dnů pro aktuální rok, každý den bude identifikován datumem, dále zde bude identifikace, o jaký den se jedná (Po, Út, St, ... – výběr z číselníku dnů **DEN**), dále jednoduchý údaj, zda-li se jedná o den pracovní (nebo volný). Třetí tabulka bude **POLOZKA_KALENDARE**, kde si uživatelé mohou ke každému dni zapisovat libovolné poznámky. Tabulka bude obsahovat identifikaci dne (z tabulky **KALENDAR**), čas od do a dále text poznámky, co si uživatel přeje k tomuto dni poznamenat. Určete správně primární a cizí klíče tabulky, dbejte na správné provázání tabulek a stanovte IO u těch atributů, kde se to bude jevit vhodné.

Návrh databázové základny

Po té, co jsme se seznámili se základními pojmy z relační databáze, můžeme přejít ke konstrukcím databázových základen. Databázovou základnou se myslí návrh databázové struktury (relačních schémat) pro konkrétní zadaný informační systém. Nejprve si vysvětlíme některé základní manipulace s tabulkami a na konec si uvedeme základní dva přístupu k návrhu databázové základny.

Relační schéma

Relační schéma je tvořeno souborem tabulek včetně jejich vzájemných vazeb. Vezmeme-li libovolnou podmnožinu relačního schématu, dostaneme opět relační schéma. Induktivní definice je následující:

- Jedna tabulka tvoří relační schéma.
- Vezmeme-li existující relační schéma (tj. i jenom jednu tabulku) a propojíme ho vazbou na jiné relační schéma, získáme opět relační schéma.
- Relační schéma získáme opakovanou aplikací předchozích dvou bodů.

Příklady relačních schémat:

Schéma A

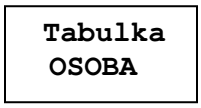


Schéma B

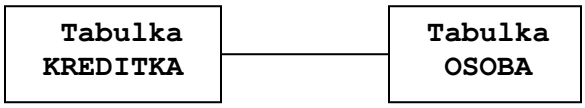
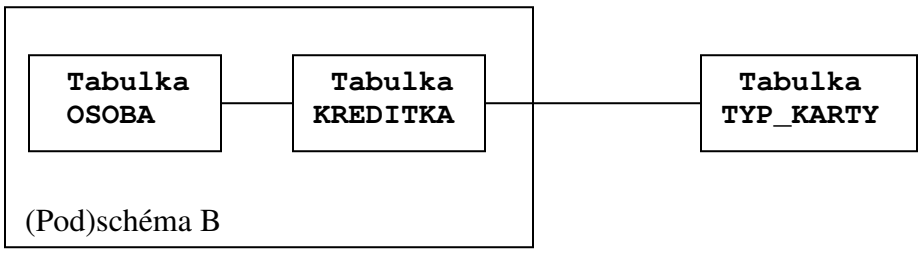


Schéma C



Syntéza relačních schémat

Syntézou relačních schémat máme na mysli slučování více tabulek do jedné. Jde o poměrně jednoduchou operaci, kterou můžeme za určitých okolností provést. Vezměme si následující tabulky: tabulku **UCITEL** s atributy **ID**, **JMENO** a **PRIJMENI** a tabulku **MZDA**

s atributy **ID_UCITEL**, **RODNE_CISLO**, **PLAT**. Ve skutečnost tabulka **MZDA** pouze rozšiřuje záznamy z tabulky **UCITEL** a skutečnost je taková, že tyto dvě tabulky jsou sobě ve vztahu 1:1. Právě pro jeden záznam v tabulce **UCITEL** existuje právě jeden záznam v tabulce **MZDA**. Analogické tvrzení můžeme říct i z „opačného směru". Právě existence vazby 1:1 mezi tabulkami nám indikuje možnost provést syntézu těchto tabulek do jedné. Výslednou tabulku můžeme nechat pojmenovanou **UCITEL** a bude mít atributy: **ID**, **JMENO**, **PRIJMENI**, **RODNE_CISLO**, **PLAT**. Zběžnou kontrolou (zkusili bychom si dosadit pár konkrétních záznamů do původních tabulek a do nové sloučené tabulky) zjistíme, že nedochází k žádné ztrátě informace. Ve skutečnosti efektivněji využíváme prostor, neboť jsme odstranili jeden redundantní sloupec **ID_UCITEL** ve druhé tabulce.

Syntézu výše zmíněných tabulek můžeme schématicky zapsat následovně:

UCITEL(ID,JMENO,PRIJMENI) ∧ MZDA(ID_UCITEL,RODNE_CISLO,PLAT)
→ UCITEL(ID,JMENO,PRIJMENI,RODNE_CISLO,PLAT)

Z důvodů, že vazba 1:1 se příliš často nevyskytuje, tak i syntézy se moc často neprovádějí. Daleko častěji se setkáme s operací opačnou k syntéze, kterou je dekompozice.

Dekompozice relačních schémat

Jak už sám název napovídá, půjde o rozdělování jedné tabulky do více (menších) tabulek. Dekompozici lze popsat na postupu rozdělení jedné tabulky na dvě tabulky:

- Určíme 2 podmnožiny atributů – jedna podmnožina bude patřit do jedné budoucí tabulky a druhá podmnožina bude patřit druhé budoucí tabulce (jinými slovy, rozdělíme sloupce tabulky, kterou chceme dekomponovat do dvou skupin – nemusí mít stejný počet sloupců)
- Vzniklé podmnožiny atributů budou definovat právě ty dvě nové tabulky.
- Nově vzniklé tabulky musíme mezi se „propojit" přes dvojici primární-cizí klíč tak, aby nedošlo ke ztrátě informace. Cizí a primární klíče doplníme do vzniklých tabulek dle potřeby.

Příklad: Dekompozice tabulky ZAMĚSTNANEC

Vezměme si náš příklad tabulky **ZAMĚSTNANEC** a zkusme provést její dekompozici na dvě menší tabulky:

Tabulka: ZAMĚSTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	FUNKCE	PLAT	SMLOUVA_OD
1	Jan	Novák	15.10.1975	uklízeč	15000	1.1.2000
2	Petr	Nový	1.4.1978	programátor	21500	12.5.1999
3	Jan	Nováček	6.9.1965	manažer	17500	7.7.1998
4	David	Vokurka	5.12.1973	ředitel	28000	1.10.2002

Tabulka **ZAMĚSTNANEC** obsahuje celkem 7 sloupců (atributů): **CISLO**, **JMENO**, **PRIJMENI**, **DAT_NAROZ**, **FUNKCE**, **PLAT**, **SMLOUVA_OD** (nebudeme již používat diakritiku).

Nyní množinu sloupců rozdělíme do dvou skupin (rozdělit ji můžeme jakkoliv, pak ji ale ovšem musíme dobře propojit, aby nedošlo ke ztrátě informace), nechť rozdělení je následující:

- 1. množina: **CISLO**, **JMENO**, **PRIJMENI**, **DAT_NAROZ**, **SMLOUVA_OD**
- 2. množina: **FUNKCE**, **PLAT**

Vzniknou tedy dvě nové tabulky, nechť ta první se jmenuje **CLOVEK**, ta druhá **POVOLANI**. Atributy (sloupce) tabulky **CLOVEK** budou sloupce z první množiny, druhá množina pak bude tvořit sloupce tabulky **POVOLANI**.

Nyní musíme zajistit, aby nedošlo ke ztrátě informace. Kdybychom už z vytvořenými tabulkami nic neprovedli, výsledek by vypadal takto:

Tabulka: CLOVEK

CISLO	JMENO	PRIJMENI	DAT_NAROZ	SMLOUVA_OD
1	Jan	Novák	15.10.1975	1.1.2000
2	Petr	Nový	1.4.1978	12.5.1999
3	Jan	Nováček	6.9.1965	7.7.1998
4	David	Vokurka	5.12.1973	1.10.2002

Tabulka: POVOLANI

FUNKCE	PLAT
uklízeč	15000
programátor	21500
manažer	17500
ředitel	28000

Jakou informační hodnotu mají obě tabulky? Určitě se nám neztratila informace, kdo se jak jmenuje, kdy se narodil a od kdy pracuje v naší firmě. Dále také vidíme, jaké platy berou určité funkce ve firmě. Co se nám ale ztratilo, je informace, kdo jakou funkci vykonává. Ve skutečnosti tím rozdělením tabulky **ZAMĚSTNANEC** jsme „roztrhli“ jednotlivé záznamy na řádcích v této tabulce. Proto nyní musíme provést „propojení“, tak aby nedošlo ke ztrátě informace.

Ono „propojení“ zrealizujeme přidáním jednoho cizího klíče do jedné tabulky a přidání primárního klíče (pokud tam ještě není) do druhé tabulky. Tím jakoby „roztržené“ řádky opět „spojíme“ dohromady. V tomto konkrétním případě přidáme do tabulky **CLOVEK** sloupec např. **CISLO_FUNKCE** a do tabulky **POVOLANI** přidáme sloupec **CISLO**, které bude primárním klíčem pro tuto tabulku (nově zavedené sloupce jsou vyznačeny kurzívou):

Tabulka: CLOVEK

CISLO	JMENO	PRIJMENI	DAT_NAROZ	SMLOUVA_OD	*CISLO_FUNKCE*
1	Jan	Novák	15.10.1975	1.1.2000	*1*
2	Petr	Nový	1.4.1978	12.5.1999	*2*
3	Jan	Nováček	6.9.1965	7.7.1998	*3*
4	David	Vokurka	5.12.1973	1.10.2002	*4*

Tabulka: POVOLANI

CISLO	FUNKCE	PLAT
1	uklízeč	15000
2	programátor	21500
3	manažer	17500
4	ředitel	28000

Provedli jsme tedy dekompozici tabulky **ZAMĚSTNANEC** na dvě nové tabulky **CLOVEK** a **POVOLANI**, schématicky můžeme zapsat následovně:

ZAMESTNANEC(CISLO,JMENO,PRIJMENI,DAT_NAROZ,FUNKCE,PLAT,SMLOUVA_OD)

$$\Downarrow$$

CLOVEK(CISLO,JMENO,PRIJMENI,DAT_NAROZ,SMLOUVA_OD,CISLO_FUNKCE) ∧
POVOLANI(CISLO,FUNKCE,PLAT)

Dekompozice tabulek může být samozřejmě i vícenásobná. Nejprve dekomponujeme jednu tabulku na dvě menší a pak libovolnou tabulku z těch dvou nových (menších) opět dále dekomponujeme.

Příklad: Vícenásobná dekompozice

Vezměme si modifikovanou tabulku **ZAMĚSTNANEC** s následujícím obsahem:

Tabulka: ZAMĚSTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	FUNKCE	PLAT	ODDELENI
1	Jan	Novák	15.10.1975	uklízeč	15000	chodba
2	Petr	Nový	1.4.1978	programátor	21500	studovna
3	Jan	Nováček	6.9.1965	manažer	17500	kancelář
4	David	Vokurka	5.12.1973	ředitel	28000	ředitelna

Vícenásobnou dekompozici tabulky **ZAMĚSTNANEC** můžeme získat tabulky **CLOVEK**, **POVOLANI** a **PRACOVISTE**:

ZAMESTNANEC(CISLO,JMENO,PRIJMENI,DAT_NAROZ,FUNKCE,PLAT,ODDELENI)

$$\Downarrow$$

CLOVEK(CISLO,JMENO,PRIJMENI,DAT_NAROZ,FUNKCE,PLAT,ID_PRACOVISTE)
∧ PRACOVISTE(ID,ODDELENI)

$$\Downarrow$$

CLOVEK(CISLO,JMENO,PRIJMENI,DAT_NAROZ,ID_FUNKCE,ID_PRACOVISTE)
∧ POVOLANI(ID,FUNKCE,PLAT)
∧ PRACOVISTE(ID,ODDELENI)

Obsah tabulek bude následující:

Tabulka: CLOVEK

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	ID_PRACOVISTE
1	Jan	Novák	15.10.1975	1	1
2	Petr	Nový	1.4.1978	2	2
3	Jan	Nováček	6.9.1965	3	3
4	David	Vokurka	5.12.1973	4	4

Tabulka: POVOLANI

ID	FUNKCE	PLAT
1	uklízeč	15000
2	programátor	21500
3	manažer	17500
4	ředitel	28000

Tabulka: PRACOVISTE

ID	ODDELENI
1	chodba
2	studovna
3	kancelář
4	ředitelna

Je jasné, že dekompozici tabulek neprovádíme jen tak z dlouhé chvíle, ale že pro ni musí být pádný důvod. Jaké důvody mohou vést k dekompozici tabulek si hned ukážeme v následující kapitole.

Normální formy

Normální formy jsou pravidla pro „dobře navržené“ tabulky. Normální formy nám definují, co musí jednotlivé tabulky splňovat, aby všechny základní operace nad daty v tabulkách se daly realizovat jednoduše s optimální časovou i prostorovou složitostí. Jinými slovy, aby vyhledávání záznamů v databázi trvalo rozumnou dobu, aby v tabulkách nebyly zbytečné redundance a aby manipulace s jednotlivými záznamy byly jednoduché a pokud možno přímé. Dobře navržené relační schéma vyžaduje, aby všechny tabulky splňovaly normální formy.

Normálních forem je několik, my si zde uvedeme tři základní:

- 1. normální forma (1NF)
- 2. normální forma (2NF)
- 3. normální forma (3NF)

Postupně si probereme jednotlivé normy, ukážeme si, k jakým situacím může dojít, pokud nejsou dodrženy.

1. normální forma

Jedná se o nejjednodušší a základní normální formu:

- Tabulka splňuje 1NF právě když všechny její atributy (sloupce) jsou atomické hodnoty.

Atomická hodnota znamená, že se jedná o hodnotu, která je dále nedělitelná. Jinými slovy, atomickou hodnotu si lze představit jako hodnotu, která nese právě jeden údaj. Vezmeme-li si náš příklad tabulky **ZAMĚSTNANEC**, pak všechny jeho hodnoty jsou atomické (pozn. jedno konkrétní datum chápeme jako jednu hodnotu, typu **DATE**):

Tabulka: ZAMĚSTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ
1	Jan	Novák	15.10.1975
2	Petr	Nový	1.4.1978
3	Jan	Nováček	6.9.1965
4	David	Vokurka	5.12.1973

Přidejme si do této tabulky nový atribut **ADRESA** pro zaevidování trvalého bydliště každého ze zaměstnanců. Nechť obsah nové tabulky **ZAMĚSTNANEC_ADRESA** vypadá následovně:

Tabulka: ZAMĚSTNANEC_ADRESA

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ADRESA
1	Jan	Novák	15.10.1975	Václavská 19, Brno, 60200
2	Petr	Nový	1.4.1978	Novákova 114, Brno, 62100
3	Jan	Nováček	6.9.1965	Hybešova 10, Brno, 60100
4	David	Vokurka	5.12.1973	Kartouzská 7, Brno, 61200

Tabulka **ZAMĚSTNANEC_ADRESA** nesplňuje 1. normální formu, neboť její atribut **ADRESA** nemá atomickou hodnotu. Hodnota adresy se skládá celkem ze čtyř údajů: ulice, číslo orientační, město a PSČ. Taková tabulka je tedy špatně navržená a musíme ji upravit tak, aby 1. normální forma byla splněna. Úpravu provedeme tak, že sloupec, jehož hodnota není atomická, rozložíme na více sloupců, kde každá hodnota již atomická bude. Následující tabulka **ZAMĚSTNANEC_ADRESA2** již splňuje 1. normální formu:

Tabulka: ZAMĚSTNANEC_ADRESA2

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ULICE	CISLO	MESTO	PSC
1	Jan	Novák	15.10.1975	Václavská	19	Brno	60200
2	Petr	Nový	1.4.1978	Novákova	114	Brno	62100
3	Jan	Nováček	6.9.1965	Hybešova	10	Brno	60100
4	David	Vokurka	5.12.1973	Kartouzská	7	Brno	61200

Chceme-li, můžeme nové sloupce pojmenovat s nějakým prefixem, ze kterého bude zřejmé, z jakého původního sloupce vychází, v našem případě například **ADRESA_ULICE**, **ADRESA_CISLO**, **ADRESA_MESTO** a **ADRESA_PSC**. V praxi se to tak běžně dělá.

2. normální forma

Tato další forma je již mírně složitější, než předchozí a zabývá se otázkou primárních klíčů a závislostí na nich.

Tabulka splňuje 2NF právě když:

- splňuje 1NF
- každý atribut, který není primárním klíčem, je na primárním klíči **úplně** závislý

První podmínkou pro splnění 2. normální formy je splnění 1. normální formy, kterou jsme si vysvětlili v předchozí části. Nyní zbývá se podívat na druhou část tvrzení, kdy je splněna 2. normální forma.

Funkční závislost byla vysvětlena v kapitole Funkční závislost na straně 40. Nyní se podívejme, co znamená úplná závislost. Uvažme následující příklad tabulky **HRA_SACH**, kde primární klíč nebude jenom jeden sloupec, nýbrž více sloupců.

Tabulka: HRA_SACH

CISLO_HRAC1	CISLO_HRAC2	DATUM	CAS	VYSLEDEK	JM_HRAC1	JM_HRAC2
10	12	12.1.2006	12:00	1:0	Jan	Pavel
11	12	14.1.2006	13:05	0:1	Karel	Pavel
12	14	14.1.2006	15:13	pat	Pavel	Petr

Primárním klíčem v tabulce **HRA_SACH** je čtveřice (**CISLO_HRAC1**, **CISLO_HRAC2**, **DATUM**, **CAS**). Těmito třemi údaji je každá šachová partie jednoznačně identifikována. Dva hráči, kteří ji hrají a časový okamžik, kdy ji začali hrát.

Jak je to s funkční závislostí? Atribut **VYSLEDEK** zcela evidentně závisí na primárním klíči (na celé čtveřici atributů **CISLO_HRAC1**, **CISLO_HRAC2** a **DATUM**, **CAS**). Ovšem sloupec **JM_HRAC1** nezávisí zcela (tj. úplně) na primárním klíči (myšleno ona čtveřice atributů), ale jenom na **CISLO_HRAC1**, který tvoří jenom část primárního klíče. Tedy sloupec **JM_HRAC1** není sám primárním klíčem a na primárním klíči je závislý jen částečně. Tato tabulka nesplňuje 2NF. Stejná situace je i se sloupcem **JM_HRAC2**, který je závislý na atributu **CISLO_HRAC2**.

Řešení, jak převést tabulku do 2NF, spočívá v dekompozici takové tabulky na nové tabulky. Výsledek dekompozice může vypadat následovně:

Tabulka: HRA_SACHY

CISLO_HRAC1	CISLO_HRAC2	DATUM	CAS	VYSLEDEK
10	12	12.1.2006	12:00	1:0
11	12	14.1.2006	13:05	0:1
12	14	14.1.2006	15:13	pat

Tabulka: HRAC1

CISLO	JM_HRAC1
10	Jan
11	Karel
12	Pavel

Tabulka: HRAC2

CISLO	JM_HRAC2
12	Pavel
14	Petr

Dekompozicí nám vznikly dvě shodné tabulky **HRAC1** a **HRAC2**, je zbytečné je mít obě, místo nich budeme mít pouze tabulku **HRAC**. V tabulce **HRA_SACHY** budou první dva sloupce cizí klíče do téže tabulky **HRAC**:

Tabulka: HRA_SACHY

CISLO_HRAC1	CISLO_HRAC2	DATUM	CAS	VYSLEDEK
10	12	12.1.2006	12:00	1:0
11	12	14.1.2006	13:05	0:1
12	14	14.1.2006	15:13	pat

Tabulka: HRAC

CISLO	JM_HRAC
10	Jan
11	Karel
12	Pavel
14	Petr

Ukážeme si, jaké potíže by nám způsobovala původní tabulka **HRA_SACH**, když bychom se snažili zavést do tabulky novou skutečnost, že například třetí hru nehrál hráč č. 12 s hráčem č. 14, ale hráč č. 10 a s hráčem č. 14. V této tabulce musíme tedy aktualizovat hodnotu na třetím řádku v sloupci **CISLO_HRAC1** na číslo 10 a dále musíme zajistit aktualizaci hodnoty sloupce **JM_HRAC1** na Jan. Oproti tomu v nové tabulce **HRA_SACHY** bychom pouze zaktualizovali hodnotu sloupce **CISLO_HRAC1** na třetím řádku.

Další potíž by například mohla nastat v okamžiku, kdy šachových partií se zúčastnil hráč Pavel (jehož číslo je 12), o kterém později zjistíme, že se ve skutečnosti jmenuje Zdeněk. Čili bude vznesen požadavek, aby se správné jméno zavedlo do databáze. V původní tabulce **HRA_SACH** budeme muset projít všechny řádky a tam kde bude **CISLO_HRAC1** rovno 12, v tomto řádku budeme muset změnit **JM_HRAC1** na Zdeněk a tam kde bude **CISLO_HRAC2** rovno 12, tak v daném řádku budeme muset změnit **JM_HRAC2** na Zdeněk. Schématicky bychom takový postup mohli zapsat následovně:

```
PROJDI všechny řádky v tabulce HRA_SACH
Pro každý řádek zjisti, zdali CISLO_HRAC1 rovno 12, pak nastav JM_HRAC1
na 'Zdeněk' a dále zjisti, zdali CISLO_HRAC2 rovno 12, pak nastav
JM_HRAC2 na 'Zdeněk'
KONEC
```

Zatímco ve druhém případě provedeme pouze změnu v tabulce **HRAC**, kde najdeme jediný řádek, kde číslo je 12 a zaktualizujeme jméno hráče, schématicky:

```
NAJDI řádek v tabulce HRAC, kde CISLO rovno 12
      pak nastav JM_HRAC na 'Zdeněk'
KONEC
```

Nemusí být čtenář zrovna počítačový odborník, aby poznal, že druhý postup je výrazně jednodušší a rychlejší.

Poznámka:
Je-li v tabulce označen jenom jeden sloupec jako primární klíč a jsou-li hodnoty ve všech sloupcích atomické, pak je 2NF splněna triviálně.

3. normální forma

Poslední normální forma pojednává o nebezpečí tzv. tranzitivní funkční závislosti, jejíž výskyt může způsobit velkou redundanci v datech.

Tabulka splňuje 3NF, právě když:

- splňuje 2NF
- žádný atribut, který není primárním klíčem, není tranzitivně závislý na žádném klíči.

Tranzitivní funkční závislost je závislost mezi atributy skrytá. Její výskyt nám opět může zkomplikovat provádění různých operací nad daty. Tranzitivní funkční závislost nám říká:

```
Když A->B a zároveň B->C pak plyne, že A->C.
```

Vezměme si následující příklad tabulky **ZAMESTNANEC** rozšířenou o sloupce pro číslo a název pracoviště:

```
Tabulka: ZAMESTNANEC
```

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	CISLO_PRAC	NAZEV_PRAC
1	Jan	Novák	15.10.1975	10	studovna
2	Petr	Nový	1.4.1978	15	vrátnice
3	Jan	Nováček	6.9.1965	10	studovna
4	David	Vokurka	5.12.1973	10	studovna
5	Vít	Mrkvička	30.1.1970	12	ředitelna

Primárním klíčem je sloupec **CISLO**. Máme tu jednu závislost, kterou je závislost **NAZEV_PRAC** na **CISLO_PRAC**. Pokud například Jan Novák změní pracoviště a již nebude pracovat na pracovišti číslo 10, ale 12, je nutné v jeho záznamu změnit také hodnotu **NAZEV_PRAC** ze studovna na ředitelna. Hodnota **NAZEV_PRAC** je tedy funkčně závislá na hodnotě **CISLO_PRAC**, neboť pokud dojde ke změně hodnoty **CISLO_PRAC**, vyvolá to změnu **NAZEV_PRAC**. A jelikož platí, že **CISLO_PRAC** je závislé na primárním klíči (tedy **CISLO->CISLO_PRAC**), a dále **CISLO_PRAC->NAZEV_PRAC**, dostaneme odtud **CISLO->NAZEV_PRAC** a máme tu tranzitivní závislost.

Problém převedení tabulky do 3NF opět spočívá v dekompozici na nové tabulky:

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	CISLO_PRAC
1	Jan	Novák	15.10.1975	10
2	Petr	Nový	1.4.1978	15
3	Jan	Nováček	6.9.1965	10
4	David	Vokurka	5.12.1973	10
5	Vít	Mrkvička	30.1.1970	12

Tabulka: PRACOVISTE

CISLO	NAZEV_PRAC
10	studovna
15	vrátnice
12	ředitelna

Problémy, které by nám tabulka nesplňující 3NF, opět spočívají v redundanci, při určitých situacích aktualizace dat bychom museli v prvním případě procházet všechny řádky a na nich případně něco aktualizovat, kdežto ve druhém případě bude zpravidla aktualizace pouze jediného řádku v jedné tabulce a změna se automaticky projeví do všech tabulek, neboť aktualizovaná hodnota je odkazovaná z jiných míst.

Shrnutí

- **Relační schéma chápeme jako soubor tabulek včetně zaznamenaných jejich vzájemných vazeb.**
- **Základními operacemi nad relačními schématy jsou syntéza (slučování více relačních schémat do jednoho) a dekompozice (rozklad relačního schématu na více relačních schémat).**
- **Normální formy jsou pravidla pro „dobře utvořené“ tabulky. Jejich dodržením je docíleno optimálního zpracování dat v tabulkách (s ohledem na časovou složitost a prostorovou složitost).**
- **Tabulka splňuje 1. normální formu, právě když všechny její atributy (sloupce) jsou atomické (= dále nedělitelné) hodnoty.**
- **Tabulka splňuje 2. normální formu, právě když splňuje 1. normální formu a dále každý atribut, který není primárním klíčem, je na primárním klíči úplně závislý.**

- **Tabulka splňuje 3. normální formu, právě když splňuje 2. normální formu a dále každý atribut, který není primárním klíčem, není tranzitivně závislý na jiném klíči.**

Otázky a úkoly

- Mějme tabulky **`ZAK(ID, JMEMO, PRIJMENI, RODNE_CISLO)`**, **`UCITEL(ID, JMENO, PRIJMENI, RODNE_CISLO)`**, **`PREDMET(ID, NAZEV, KOD, ID_UCITEL)`** a tabulku **`KLASIFIKACE(ID_PREDMET, ID_UCITEL, ZNAMKY, DATUM)`**. Tabulka **`ZAK`** obsahuje údaje o jednotlivých studentech, tabulka **`UCITEL`** eviduje všechny učitele. Tabulka **`PREDMET`** obsahuje záznamy pro jednotlivé předměty, které jsou složeny z názvu předmětu, kódu předmětu a ID učitele, který daný předmět učí. Poslední tabulkou je tabulka **`KLASIFIKACE`**, která eviduje hodnocení jednotlivých žáků v jednotlivých předmětech. Sloupec **`ID_PREDMET`** obsahuje identifikaci předmětu, **`ID_ZAK`** je identifikace žáka, kterého hodnotíme v daném předmětu, sloupec **`ZNAMKY`** obsahuje jednotlivé získané známky studentem v daném předmětu; známky jsou od sebe odděleny čárkou, např: **`1, 2, 2, 2, 1, 3`**. Poslední sloupec **`DATUM`** obsahuje datum posledního udělení známky. Zjistěte, zdali tabulka **`KLASIFIKACE`** splňuje 3NF, odpověď zdůvodněte. V případě, že není 3NF splněna, proveďte odpovídající úpravy tohoto relačního schématu tak, aby 3NF splněna byla.
- Existuje alespoň jedna tabulka, která splňuje 1NF a současně nesplňuje 2NF?
- Existuje alespoň jedna tabulka, která splňuje 3NF a zároveň nesplňuje 2NF?

Relační algebra

Doposud jsme se zabývali otázkou, jak konstruovat databázové tabulky, jak zajistit, aby byly naplněny „správnými“ daty. Touto kapitolou si uvedeme jemný úvod do problematiky získávání informací z tabulek. Popíšeme si základní pojmy, které se v této oblasti používají a jejichž pochopení nám pak pomůže snáze vstřebat standard pro relační databáze, jazyk SQL.

Relační algebra je matematický nástroj, který pracuje a manipuluje s relacemi. Vstupem do operací relační algebry jsou relace a výstupem jsou opět relace. Relace jsou v podstatě množiny (množina n-tic splňující danou relaci). Můžeme tedy říci, že do operací vstupují množiny a výstupem jsou opět množiny.

Základní prostředky pro manipulaci s relacemi budou jistě množinové operace, které známe: **kartézský součin**, **sjednocení**, **průnik** a **rozdíl**. Kromě těchto čtyř operací máme ještě tři další, které jsou ryze databázové: **projekce**, **restrikce** a **spojení**.

Uvedené operace nám slouží k formulaci dotazů. **Dotaz** je formule specifikující, jaké informace chceme z databáze získat. Výsledek dané operace je **odpovědí** na náš dotaz. Jednotlivé operace si postupně projdeme, včetně praktických příkladů. V této kapitole budeme zadání dotazů pro jejich odlišení od běžného textu zapisovat kurzívou.

Projekce

Projekce **P** nad relací **R** s množinou atributů **A** je relace **R*** s množinou atributů **B**, kde **B** je podmnožinou původní množiny atributů **A**. Jako příklad si vezměme relaci **ZAMESTNANEC** s atributy **CISLO**, **JMENO**, **PRIJMENI**, **DAT_NAROZ**, **SMLOUVA_OD**. Množina původních atributů **A = {CISLO, JMENO, PRIJMENI, DAT_NAROZ, SMLOUVA_OD}**. Nechť projekce **P** je zaměřena pouze na jména a příjmení zaměstnanců. Tedy množina **B = {JMENO, PRIJMENI}**. Výsledná relace **ZAMESTNANEC*** bude obsahovat pouze 2 atributy **JMENO** a **PRIJMENI**.

Projekci zapisujeme do hranatých závorek **[**, **]**, do kterých uvedeme jména sloupců, na které chceme projekci zrealizovat. Původní relace (tabulka) **ZAMESTNANEC** vypadala takto:

Tabulka: ZAMESTNANEC

CISLO	JMENO	PRIJMENI	DAT_NAROZ	SMLOUVA_OD
1	Jan	Novák	15.10.1975	1.1.2000
2	Petr	Nový	1.4.1978	12.5.1999
3	Jan	Nováček	6.9.1965	7.7.1998
4	David	Vokurka	5.12.1973	1.10.2002

Nová relace **ZAMESTNANEC*: ZAMESTNANEC[JMENO,PRIJMENI]** vypadá následovně:

Tabulka: ZAMESTNANEC*

JMENO	PRIJMENI
Jan	Novák
Petr	Nový
Jan	Nováček
David	Vokurka

U projekce je třeba ještě zmínit, že pokud by vyprojektováním určitých sloupců ve výsledné relaci došlo k duplicitě některých řádků, pak tato duplicita je automaticky odstraněna. Vezměme si náš příklad relace **ZAMESTNANEC** a nechť v naší firmě pracují dva lidé jménem Jan Novák, s tím rozdílem, že první se narodil 15.10.1975 a druhý 3.2.1960. Projekcí na sloupce **JMENO** a **PRIJMENI** získáme ve výsledné relaci pouze jeden záznam Jan Novák.

Příklad: projekce
Mějme relaci, která je popsána tabulkou **BYT**. Ta obsahuje sloupce:

- **CISLO** – evidenční číslo bytu
- **ADR_ULICE** – adresa: ulice, kde se byt nachází
- **ADR_CISLO** – číslo domu
- **ADR_MESTO** – město, ve kterém se byt nachází
- **ADR_PSC** – poštovní směrovací číslo
- **PODLAZI** – číslo podlaží
- **BALKON** – A/N, zda-li má byt balkon
- **SKLEP** – A/N, zda-li má byt sklep

Nechť v tabulce **BYT** je uloženo celkem 200 záznamů. To znamená, že naši relaci **BYT** splňuje celkem 200 různých bytů.

Zodpovězme nyní následující dotazy:

- *Čísla všech evidovaných bytů*: **BYT[CISLO]**
- *Seznam všech adres (ulice a město) kde jsou byty evidovány*: **BYT[ADR_ULICE,ADR_MESTO]**
- *Čísla všech podlaží, na kterých se evidované byty nachází*: **BYT[PODLAZI]**
- *Seznam všech měst, ve kterých se nachází evidované byty*: **BYT[ADR_MESTO]**

Restrikce

Tak jako nám projekce vybírá do výsledku určité sloupce tabulky, tak restrikce vybírá do výsledku určité řádky tabulky, které splňují zadanou podmínku.

Restrikce podle logické podmínky ϕ nad relací **R** je relace **R***, ve které jsou ponechány jen ty n-tice z původní relace, jejichž hodnoty splňují logickou podmínku ϕ. Logická podmínka ϕ je zadána booleovským výrazem (pomocí logických spojek **and**, **or** a **not**) složeného z atomických formulí tvaru: $\mathbf{t}_1 \theta\ \mathbf{t}_2$, kde $\theta = \{<, >, =, \neq, \leq, \geq\}$. Přičemž $\mathbf{t}_i$ je buď nějaká konstanta, nebo název existujícího atributu. Logickou podmínku restrikce zapisujeme do kulatých závorek **(**, **)**.

Výše uvedený teoretický zápis je obecným zápisem například pro jednoduché podmínky **JMENO = Jan**, **PRIJMENI = Novák**, **CISLO = 2**, **PODLAZI ≠ 1**, apod. Jednotlivé podmínky můžeme spolu kombinovat pomocí závorek a logických spojek, např: **JMENO = Jan and PRIJMENI = Novak**.

Vezměme si náš příklad relace **ZAMESTNANEC** a zkusme si na něm provést několik restrikcí. Původní relace **ZAMESTNANEC** vypadá následovně:

Tabulka: ZAMESTNANEC

CISLO	JMENO	PRIJMENI	DAT_NAROZ	PLAT	SMLOUVA_OD
1	Jan	Novák	15.10.1975	15000	1.1.2000
2	Petr	Nový	1.4.1978	21500	12.5.1999
3	Jan	Nováček	6.9.1965	17500	7.7.1998
4	David	Vokurka	5.12.1973	28000	1.10.2002

Zkusme zformulovat několik jednoduchých dotazů, na kterých je dobře vidět formulování logické podmínky pro restrikci. U každého příkladu je rovněž uvedeno, kolik záznamů z tabulky (nebo-li kolik n-tic z relace) danou restrikci splňuje – snadno se ověří.

- *Všichni zaměstnanci, kteří pracují ve firmě déle než 3 roky*:
  ZAMESTNANEC(SMLOUVA_OD <= 1.1.2003) – splňují 4 záznamy
- *Všichni zaměstnanci, kterým je více než 30 let*:
  ZAMESTNANEC(DAT_NAROZ < 31.12.1975) – splňují 2 záznamy
- *Všichni zaměstnanci, jejichž křestní jméno je Jan*:
  ZAMESTNANEC(JMENO = Jan) – splňují 2 záznamy
- *Všichni zaměstnanci, kteří mají plat vyšší než 25000 Kč*:
  ZAMESTNANEC(PLAT > 25000) – splňuje 1 záznam
- *Všichni zaměstnanci, kteří nejsou ve zkušební době a pobírají plat více jak 20000 Kč*:
  ZAMESTNANEC(SMLOUVA_OD < 1.1.2006 and PLAT > 20000) – splňují 2 záznamy
- *Všichni zaměstnanci, kteří jsou starší 30ti let a pobírají plat maximálně 18000 Kč*:
  ZAMESTNANEC(DAT_NAROZ < 31.12.1975 and PLAT >= 18000) – splňuje 1 záznam

Restrikce a projekce

Již víme, že výsledkem operace restrikce na relaci je opět relace. Nic nám nebrání v tom, abychom na výsledek operace restrikce aplikovali další operaci a to projekci. Všimněte si, že v předchozích uvedených příkladech restrikce jsme vždy z tabulky získali celé záznamy (celé n-tice). Co kdybychom ale měli dotaz: *Jména a příjmení všech zaměstnanců (další jejich údaje nás nezajímají), kteří začali pro naši firmu pracovat před rokem 2001?* Takový dotaz bychom zkonstruovali ve dvou krocích:

- Nejprve provedeme restrikci pro „*začali pracovat před rokem 2001*“ – výsledná „mezi“-relace vypadá následovně:

Tabulka: ZAMESTNANEC'

CISLO	JMENO	PRIJMENI	DAT_NAROZ	PLAT	SMLOUVA_OD
1	Jan	Novák	15.10.1975	15000	1.1.2000
2	Petr	Nový	1.4.1978	21500	12.5.1999
3	Jan	Nováček	6.9.1965	17500	7.7.1998

- A po té provedeme projekci na „*Jména a příjmení*“, s výsledkem:

Tabulka: ZAMESTNANEC*

JMENO	PRIJMENI
Jan	Novák
Petr	Nový
Jan	Nováček

Uvedený dotaz v relační algebře zapíšeme následovně:

```
ZAMESTNANEC(SMLOUVA_OD<1.1.2001)[JMENO,PRIJMENI]
```

Formulujme další jednoduché dotazy s využitím projekce a restrikce:

- *Kdy se narodil Jan Novák?*
  `ZAMESTNANEC(JMENO=Jan and PRIJMENI=Novák)[DAT_NAROZ]`
- *Jaký plat pobírá a odkdy pracuje ve firmě David Vokurka?*
  `ZAMESTNANEC(JMENO=David and PRIJMENI=Vokurka)[PLAT, SMLOUVA_OD]`
- *Kdy nastoupil do firmy pan Nováček?*
  `ZAMESTNANEC(PRIJMENI=Novacek)[SMLOUVA_OD]`

Časem zjistíte, že všechny operace relační algebry se dají mezi sebou kombinovat a aplikovat jednu na druhou – podle dotazu, který chceme pomocí relační algebry zkonstruovat. To, jak ten dotaz zkonstruujeme, bude mít vliv na výsledek.

Vezměme si naši relaci **BYT** popisující jednotlivé byty z předchozí podkapitoly. Zkonstruujme následující dotazy:

- *Seznam bytů (resp. jejich evidenční čísla), která jsou na území města Brna*:

BYT(ADR_MESTO="Brno")[CISLO]

- *Seznam bytů, které obsahují buď balkon nebo sklep*:
  BYT(BALKON=„A" or SKLEP=„A")[CISLO]
- *Seznam bytů, které nejsou v přízemí (tj. ne 1.podlaží)*:
  BYT(not PODLAZI=1)[CISLO]
- *Seznam ulic, na kterých se nachází evidované byty*:
  BYT[ADR_ULICE]
- *Seznam měst, ve kterých se nachází evidované byty*:
  BYT[ADR_MESTO]
- *Seznam bytů, které neleží v přízemí a mají buď balkon nebo sklep*:
  BYT(PODLAZI>1 and (BALKON="A" or SKLEP="A"))[CISLO]

Zbývá si vysvětlit, jak je to s pořadím operací v relační algebře. V zásadě platí, že pořadí může být libovolné, pokud zachováme podmínku, že operace předcházející musí dát na výstup takovou relaci, se kterou operace následující může pracovat. Speciálně jde o názvy atributů.

Vezměme si náš příklad relace **ZAMESTNANEC**. Nad touto relací definujme dotaz: *„Jména a příjmení všech zaměstnanců, kteří mají plat vyšší než 20000 Kč"*. Dotaz bude vypadat následovně:

```
ZAMESTNANEC(PLAT>20000)[JMENO,PRIJMENI]
```

Dotaz se vyhodnotí postupně. Nejprve se zrealizuje restrikce, která „pustí" jen ty řádky, kde **PLAT** je větší než **20000** Kč, tj. vznikne „mezi"-relace **ZAMESTNANEC'**:

Tabulka: ZAMESTNANEC'

CISLO	JMENO	PRIJMENI	DAT_NAROZ	PLAT	SMLOUVA_OD
2	Petr	Nový	1.4.1978	21500	12.5.1999
4	David	Vokurka	5.12.1973	28000	1.10.2002

Restrikce vytvořila novou relaci, která obsahuje pouze 2 z původních n-tic, které splňovaly uvedenou podmínku. Po té provedeme projekci na sloupce **JMENO** a **PRIJMENI** a výsledek bude tento:

Tabulka: ZAMESTNANEC'

JMENO	PRIJMENI
Petr	Nový
David	Vokurka

Jak by se vyhodnotil dotaz formulovaný následovně?

```
ZAMESTNANEC[JMENO,PRIJMENI](PLAT>20000)
```

Nejprve se provede projekce, vznikne „mezi"-relace **ZAMESTNANEC'** s následujícím obsahem:

Tabulka: ZAMESTNANEC'

JMENO	PRIJMENI
Jan	Novák
Petr	Nový
Jan	Nováček
David	Vokurka

A nyní na tuto „mezi"-relaci by se aplikovala uvedená restrikce **PLAT>20000**. Jenomže tato „mezi"-relace obsahuje pouze atributy **JMENO** a **PRIJMENI**, nikoliv **PLAT**. Vyhodnocení dotazu tedy skončí s chybou.

Nicméně, pokud by původní dotaz byl formulován trochu odlišně: „*Jména a příjmení a platy všech zaměstnanců, kteří pobírají plat vyšší než 20000 Kč*", pak máme dvě možnosti, jak takový dotaz zapsat:

```
ZAMESTNANEC(PLAT>20000)[JMENO,PRIJMENI,PLAT]
```

nebo

```
ZAMESTNANEC[JMENO,PRIJMENI,PLAT](PLAT>20000)
```

Oba dva dotazy vrací stejný výsledek. Ověřte si to.

Přirozené spojení

Další, třetí, databázovou operací je spojení. Spojení může být více druhů, postupně si představíme všechny, nyní se však budeme zabývat tím základním a to přirozeným spojením. Doposud probrané operace projekce a restrikce se vždy aplikovaly na jednu relaci a výsledkem byla opět jedna relace. Operace přirozeného spojení je trochu odlišná. Vstupem do přirozeného spojení jsou dvě relace a výsledkem je jedna relace. Vezměme si náš rozšířený příklad relace **ZAMESTNANEC** a **FUNKCE**:

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE
1	Jan	Novák	15.10.1975	1
2	Petr	Nový	1.4.1978	2
3	Jan	Nováček	6.9.1965	3
4	David	Vokurka	5.12.1973	4

Tabulka: FUNKCE

ID	NAZEV	PLAT
1	uklízeč	15000
2	programátor	21500
3	manažer	17500
4	ředitel	28000

Z definice uvedených tabulek víme, že sloupec **ID_FUNKCE** v tabulce **ZAMESTNANEC** je cizím klíčem do tabulky **FUNKCE** na její sloupec **ID**. Tato vazba primární-cizí klíč je

důležitá a právě skrz tuto vazbu se realizuje přirozené spojení dvou tabulek (nebo-li relací).

Přirozené spojení vezme dvě původní relace **R** nad atributy z množiny **A**, tj. **R(A)** a relace **S** nad atributy z množiny **B**, tj. **S(B)**, které do něj vstupují a vytvoří jednu (souhrnnou) relaci **R*S** a to takovým způsobem, že atributy této nové relace budou všechny atributy jak z původní množiny atributů **A** tak i z **B**, přičemž jednotlivé řádky (n-tice) budou „svázány" tak, aby si jednotlivé n-tice z původních tabulek „odpovídaly". Nebo-li budou k sobě navázány taková n-tice z první relace s takovou n-ticí z druhé relace, že hodnota příslušného primárního klíče v první relaci se rovná hodnotě příslušného cizího klíče ve druhé relaci. V našem uvedeném příkladě relací **ZAMESTNANEC** a **FUNKCE** bude výsledek přirozeného spojení vypadat následovně:

Tabulka: ZAMESTNANEC*FUNKCE

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	ID	NAZEV	PLAT
1	Jan	Novák	15.10.1975	1	1	uklízeč	15000
2	Petr	Nový	1.4.1978	2	2	programátor	21500
3	Jan	Nováček	6.9.1965	3	3	manažer	17500
4	David	Vokurka	5.12.1973	4	4	ředitel	28000

Všimněte si, že jednotlivé řádky se „spojily" tak, že **ID_FUNKCE** a **ID** jsou si rovny.

Poznámka:
Při spojení dvou tabulek se může stát, že do takového spojení vstoupí takové relace, které budou obsahovat některé stejné názvy sloupců. V takovém případě použijeme tečkovou notaci, kde před názvem sloupce uvedeme název tabulky (relace), ze které daný sloupec původně pochází.

Přirozené spojení je operací relační algebry, které nám umožňuje definovat dotazy přes více relací (tabulek). Důležitou podmínkou je ale existence vazby primární-cizí klíč mezi uvažovanými tabulkami.

Výsledkem operace přirozeného spojení je opět relace, takže můžeme na ni aplikovat další operaci relační algebry, např. projekci. Na našem příkladu dvou relací **ZAMESTNANEC** a **FUNKCE** si definujme několik dotazů:

- *Jakou funkci vykonává pan Nováček?*
  ZAMESTNANEC*FUNKCE(PRIJMENI=Nováček)[NAZEV]
- *Jaký plat pobírají lidé v naší firmě starší 30 let?*
  ZAMESTNANEC*FUNKCE(DAT_NAROZ<1.1.1976)[PLAT]
- *Seznam jmen, příjmení a funkcí, které vykonávají všichni zaměstnanci ve firmě:*
  ZAMESTNANEC*FUNKCE[JMENO,PRIJMENI,NAZEV]
- *Seznam příjmení, datumů narození a výše platů všech zaměstnanců ve firmě:*
  ZAMESTNANEC*FUNKCE[PRIJMENI,DAT_NAROZ,PLAT]

Přirozené spojení můžeme kombinovat nejen s projekcí, ale i s restrikcí:

- Seznam jmen, příjmení a funkcí všech zaměstnanců ve firmě, kteří mají plat vyšší než 20000 Kč:
  ZAMESTNANEC*FUNKCE(PLAT>20000)[JMENO,PRIJMENI,NAZEV]
- Seznam příjmení a platů všech uklízečů ve firmě:
  ZAMESTNANEC*FUNKCE(NAZEV="uklízeč")[PRIJMENI,PLAT]
- Seznam jmen, příjmení všech zaměstnanců, kterým není ještě 30 let nebo mají plat nižší než 18000 Kč:
  ZAMESTNANEC*FUNKCE(DAT_NAROZ>1.1.1976 or PLAT<18000)[JMENO, PRIJMENI]

Rozšiřme si náš příklad o další relaci (tabulku).

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	ID_PRACOVISTE
1	Jan	Novák	15.10.1975	1	11
2	Petr	Nový	1.4.1978	2	21
3	Jan	Nováček	6.9.1965	3	31
4	David	Vokurka	5.12.1973	4	41

Tabulka: POVOLANI

ID	FUNKCE	PLAT
1	uklízeč	15000
2	programátor	21500
3	manažer	17500
4	ředitel	28000

Tabulka: PRACOVISTE

ID	ODDELENI
11	chodba
21	studovna
31	kancelář
41	ředitelna

A nyní mějme následující otázku: *Na jakém pracovišti pracuje pan Vokurka?* Údaje o jednotlivých lidech máme v relaci **ZAMESTNANEC**, údaje o pracovištích máme v relaci **PRACOVISTE**. Dotaz budeme pokládat přes relace **ZAMESTNANEC** a **PRACOVISTE**:

```
ZAMESTNANEC*PRACOVISTE(PRIJMENI=Vokurka)[ODDELENI]
```

Nyní položme nový dotaz: *Na jakých pracovištích pracují manažeři?* V tomto případě musíme tuto informaci získat ze všech tří tabulek. V první, **ZAMESTNANEC**, je informace o tom, jaké **ID_FUNKCE** pracuje na jakém **ID_PRACOVISTE**, informace (zejména pak název) o funkci je uvedena v relaci **POVOLANI** a informace (zejména pak název) o jednotlivých pracovištích je v relaci **PRACOVISTE**.

Zde existují 2 možnosti, jak takový dotaz definovat:

- I. - Provedeme přirozené spojení všech zainteresovaných relací a teprve potom aplikujeme restrikci a nakonec projekci:
  (ZAMESTNANEC*POVOLANI)*PRACOVISTE(FUNKCE=manažer)[ODDELENI]
- II. - Provedeme přirozené spojení jen dvou relací ze tří, po té aplikujeme restrikci, teprve potom „dopojíme“ se zbývající třetí relací a nakonec dáme projekci:
  (ZAMESTNANEC*POVOLANI(FUNKCE=manažer))*PRACOVISTE[ODDELENI]

Postup vyhodnocení těchto dotazů se bude samozřejmě lišit, lišit se budou i jednotlivé „mezi“-relace v průběhu výpočtu, nicméně výsledek bude stejný.

Vyhodnocení dotazu I.

Nejprve zrealizujeme přirozené spojení **ZAMESTNANEC*POVOLANI**. Výsledná „mezi“-relace vypadá následovně (názvy sloupců a jejich hodnoty se již nevejdou na jeden řádek, proto jeden záznam je rozepsán v následujících příkladech přes více řádků):

Tabulka: ZAMESTNANEC*POVOLANI

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	ID_PRACOVISTE	ID	FUNKCE	PLAT
1	Jan	Novák	15.10.1975	1	11	11	uklízeč	15000
2	Petr	Nový	1.4.1978	2	21	21	programátor	21500
3	Jan	Nováček	6.9.1965	3	31	31	manažer	17500
4	David	Vokurka	5.12.1973	4	41	41	ředitel	28000

Nyní se tato „mezi“-relace opět přirození spojí s relací **PRACOVISTE**:

```
Tabulka: (ZAMESTNANEC*POVOLANI)*PRACOVISTE
ČÍSLO JMÉNO PŘÍJMENÍ    DAT_NAROZ   ID_FUNKCE   ID_PRACOVISTE
      POVOLANI.ID FUNKCE      PLAT        PRACOVISTE.ID     ODDELENI

1     Jan   Novák       15.10.1975  1           11
1           uklízeč     15000       11
chodba

2     Petr  Nový        1.4.1978    2           21
2           programátor 21500       21
studovna

3     Jan   Nováček     6.9.1965    3           31
3           manažer     17500       31
kancelář

4     David Vokurka     5.12.1973   4           41
4           ředitel     28000       41
ředitelna
```

Nyní se provede restrikce **FUNKCE=manažer**:

```
Tabulka: (ZAMESTNANEC*POVOLANI)*PRACOVISTE(FUNKCE=manažer)
ČÍSLO JMÉNO PŘÍJMENÍ    DAT_NAROZ   ID_FUNKCE   ID_PRACOVISTE
      POVOLANI.ID FUNKCE      PLAT        PRACOVISTE.ID     ODDELENI

3     Jan   Nováček     6.9.1965    3           31
      3           manažer     17500       31                kancelář
```

A nakonec projekce na sloupec **ODDELENI**:

```
Tabulka: (ZAMESTNANEC*POVOLANI)*PRACOVISTE(FUNKCE=manažer)[ODDELENI]
           ODDELENI
```

Odpovědí na dotaz I. je tedy jedno oddělení jménem **ředitelna**.

Vyhodnocení dotazu II.

Nyní si uvedeme postup vyhodnocení dotazu II. pro srovnání. Nejprve se provede přirozené spojení **ZAMESTNANEC*POVOLANI**:

Tabulka: ZAMESTNANEC*POVOLANI

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	ID_PRACOVISTE	ID
FUNKCE	PLAT					
1	Jan	Novák	15.10.1975	1	1	1
	uklízeč	15000				
2	Petr	Nový	1.4.1978	2	2	2
	programátor	21500				
3	Jan	Nováček	6.9.1965	3	3	3
	manažer	17500				
4	David	Vokurka	5.12.1973	4	4	4
	ředitel	28000				

Dále se provede restrikce **FUNKCE=manažer**:

Tabulka: ZAMESTNANEC*POVOLANI(FUNKCE=manažer)

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	ID_PRACOVISTE	ID
FUNKCE	PLAT					
3	Jan	Nováček	6.9.1965	3	31	31
	manažer	17500				

Nyní se přirozeně spojí s relací **PRACOVISTE**:

Tabulka: (ZAMESTNANEC*POVOLANI(FUNKCE=manažer))*PRACOVISTE

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	ID_PRACOVISTE	
POVOLANI.ID	FUNKCE	PLAT	PRACOVISTE.ID	ODDELENI		
3	Jan	Nováček	6.9.1965	3	31	
3	manažer	17500	31	kancelář		

A nakonec projekce na sloupec **ODDELENI**:

Tabulka: (ZAMESTNANEC*POVOLANI(FUNKCE=manažer))*PRACOVISTE[ODDELENI]

ODDELENI

Výsledek dotazu II. je tedy úplně stejný jako u dotazu I., jenom postup vedoucí k tomuto stejnému výsledku byl odlišný.

Další dotazy nad relacemi **ZAMESTNANEC**, **POVOLANI** a **PRACOVISTE**:

- *Kdo pracuje ve studovně?*
  ZAMESTNANEC*PRACOVISTE(ODDELENI=studovna)[JMENO, PRIJMENI]
- *Datumy narození těch, kteří mají plat vyšší než 20000 Kč:*
  ZAMESTNANEC*POVOLANI(PLAT>20000)[DAT_NAROZ]
- *Kdo nepracuje ve studovně?*
  ZAMESTNANEC*PRACOVISTE(ODDELENI≠studovna)[JMENO, PRIJMENI]

- *Jména a příjmení zaměstnanců, kteří pracují ve studovně nebo v kanceláři a pobírají plat nižší než 18000 Kč:*
  (ZAMESTNANEC*PRACOVISTE)*POVOLANI((ODDELENI=studovna or ODDELENI=kancelář) and PLAT<18000)) [JMENO, PRIJMENI]

Při konstrukci výsledné relace, která vznikne operací přirozeného spojení, se opět automaticky odstraňují duplicity, jak tomu je v případě projekce. Dále je potřeba si u přirozeného spojení uvědomit ještě jednu věc. Podívejme se na následující příklad:

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE
1	Jan	Novák	15.10.1975	1
2	Petr	Nový	1.4.1978	2
3	Jan	Nováček	6.9.1965	3
4	David	Vokurka	5.12.1973	4

Tabulka: FUNKCE

ID	NAZEV	PLAT
1	uklízeč	15000
2	programátor	21500
3	manažer	17500
4	ředitel	28000
5	účetní	23000
6	vrátný	10000

Co můžeme usoudit z těchto dvou tabulek? Všimněte si dvou posledních záznamů v tabulce **FUNKCE**. Je zde funkce **účetního** a **vrátného**. V tabulce **ZAMESTNANEC** ale není žádný záznam takový, kde by **ID_FUNKCE** odpovídalo **ID** funkce účetní nebo vrátný. Takový stav v databázi můžeme interpretovat takovým způsobem, že funkce účetní ani vrátný v naší firmě zatím není obsazena (i když do budoucna se s těmito funkcemi zřejmě počítá, neboť je máme zaneseny v číselníku všech funkcí). Jak bude vypadat přirozené spojení těchto dvou relací?

Tabulka: ZAMESTNANEC*FUNKCE

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	ID	NAZEV	PLAT
1	Jan	Novák	15.10.1975	1	1	uklízeč	15000
2	Petr	Nový	1.4.1978	2	2	programátor	21500
3	Jan	Nováček	6.9.1965	3	3	manažer	17500
4	David	Vokurka	5.12.1973	4	4	ředitel	28000

Jelikož v původní relaci **ZAMESTNANEC** nebyl žádná „odpovídající“ n-tice, kde by **ID_FUNKCE** bylo rovno **5** nebo **6**, vidíte, že záznam pro účetního nebo vrátného se nám vůbec ve výsledku neprojevil.

Obecné spojení

Přirozené spojení je v podstatě zvláštním případem spojení obecného. Spojovat relace (tabulky) můžeme obecně podle jakékoliv podmínky. V případě přirozeného spojení jako logická podmínka figurovala rovnost dvou atributů – každý atribut z jedné relace, jeden byl v jedné relaci primárním klíčem, druhý ve druhé cizím.

Přirozené spojení dvou relací **R(A)** a **S(B)** přes vazbu primární-cizí klíč, můžeme obecně zapsat následovně:

$R[t_1=t_2]S$, kde $t_1 \in A$, $t_2 \in B$.

Místo zápisu **$R[t_1=t_2]S$** ovšem zapisujeme **R*S**.

Podmínka, která je uvedena v hranatých závorkách mezi relacemi může být obecná, dokonce i jako booleovský výraz složený z několika podmínek a aplikovanými logickými spojkami **and**, **or** a **not**. Pravidla pro zápis podmínky jsou velmi podobná pravidlům zápisu podmínek pro restrikce.

Běžně obecný typ spojení nevyužijeme, nicméně v některých speciálních databázových úlohách (které například řeší geometrické úlohy, např. v geografických informačních systémech) se může hodit.

Ještě jedním speciálním typem spojení je kartézský součin, který by se volně dal definovat jako „každý s každým“. Kartézský součin byl probrán v kapitole pojednávající o relacích a zde v této kapitole bude zopakován později, i když jej využijeme v následujícím příkladu:

Příklad: úsečky trojúhelníka
Ukažme si příklad pro řešení jedné geometrické úlohy, kde můžeme využít obecného spojení relací.

Mějme základní relace **`USECKA_A`**, **`USECKA_B`** a **`USECKA_C`**. Jednotlivé úsečky budou reprezentovat vždy jednu stranu potencionálního trojúhelníka, který budeme moci pomocí nich konstruovat. Každá z těchto relací bude mít vždy dva atributy: **`CISLO`** a **`DELKA`**. Každou úsečku budeme tedy identifikovat jednak relací, do které patří a jednak svým jednoznačným identifikátorem (**`CISLO`**) v rámci relace. Například úsečka číslo **1** v relaci **`USECKA_A`** bude zapsána jako **`USECKA_A.CISLO=1`** (využíváme zde tečkovou notaci, neboť pracujeme se třemi relacemi, které mají shodné názvy atributů).

Nechť jednotlivé relace splňují následující n-tice:

Tabulka: USECKA_A

CISLO	DELKA
10	1
11	2
12	3

Tabulka: USECKA_B

CISLO	DELKA
13	1
14	2
15	3

Tabulka: USECKA_C

CISLO	DELKA
16	1
17	2
18	3

Mějme dotaz: *které úsečky (identifikované číslem) mohou spolu tvořit trojúhelník?*
Trojúhelník je tvořen třemi úsečkami a aby mohly tři úsečky **A**, **B** a **C** tvořit trojúhelník, musí mezi nimi platit trojúhelníková nerovnost (všechny tři podmínky):

- **A+B>C**
- **B+C>A**
- **A+C>B**

Je jasné, že musíme spojit všechny tři relace podle podmínky trojúhelníkové nerovnosti. Jelikož operace spojení je binární (vstupují do ní dvě relace a výstupem je jedna relace) nemůžeme provést spojení všech tří tabulek zaráz a do hranatých závorek vypsat všechny potřebné podmínky pro trojúhelníkovou nerovnost. Pomůžeme si tedy kartézským součinem, pomocí kterého spojíme první dvě relace metodou „každý s každým“:

USECKA_A × USECKA_B

„Mezi“-relace **USECKA_A × USECKA_B** bude obsahovat následující (řetězec **USECKA** je nahrazen třemi tečkami)

Tabulka: USECKA_A×USECKA_B

…_A.CISLO	…_A.DELKA	…_B.CISLO	…_B.DELKA
10	1	13	1
10	1	14	2
10	1	15	3
11	2	13	1
11	2	14	2
11	2	15	3
12	3	13	1
12	3	14	2
12	3	15	3

Na tuto „mezi“-relaci aplikujeme obecné spojení s třetí relací **USECKA_C** na základě podmínky pro trojúhelníkovou nerovnost:

```
(USECKA_A×USECKA_B)[(USECKA_A.DELKA+USECKA_B.DELKA>USECKA_C.DELKA
)and(USECKA_B.DELKA+USECKA_C.DELKA>USECKA_A.DELKA)and
(USECKA_A.DELKA+USECKA_C.DELKA>USECKA_B.DELKA)]USECKA_C
```

Další mezi-výsledek bude vypadat následovně:

…

…_A.CISLO	…_A.DELKA	…_B.CISLO	…_B.DELKA	…_C.CISLO	…_C.DELKA
10	1	13	1	16	1
10	1	14	2	17	2
10	1	15	3	18	3
11	2	13	1	17	2
11	2	14	2	16	1
11	2	14	2	17	2
11	2	14	2	18	3
11	2	15	3	17	2
11	2	15	3	18	3
12	3	13	1	18	3
12	3	14	2	18	3
12	3	14	2	17	2
12	3	15	3	17	2
12	3	15	3	18	3

Nyní zbývá už jenom projekce na čísla úseček:

```
(USECKA_A×USECKA_B)[(USECKA_A.DELKA+USECKA_B.DELKA>USECKA_C.DELKA
)and(USECKA_B.DELKA+USECKA_C.DELKA>USECKA_A.DELKA)and
(USECKA_A.DELKA+USECKA_C.DELKA>USECKA_B.DELKA)]USECKA_C
[USECKA_A.CISLO, USECKA_B.CISLO, USECKA_C.CISLO]
```

…

USECKA_A.CISLO	USECKA_B.CISLO	USECKA_C.CISLO
10	13	16
10	14	17
10	15	18
11	13	17
11	14	16
11	14	17
11	14	18
11	15	17
11	15	18
12	13	18
12	14	18
12	14	17
12	15	17
12	15	18

Z hypotetických 27 možností (3×3×3) jsem nakonec získali relaci, ve které jsou uvedeny jen ty možné kombinace úseček **A**, **B**, **C**, které jsou schopny vytvořit trojúhelník.

Shrnutí: formulace dotazu „*které úsečky mohou spolu tvořit trojúhelník*“ v relační algebře vypadá následovně:

```
(USECKA_A×USECKA_B)[(USECKA_A.DELKA+USECKA_B.DELKA>USECKA_C.DELKA
)and(USECKA_B.DELKA+USECKA_C.DELKA>USECKA_A.DELKA)and
(USECKA_A.DELKA+USECKA_C.DELKA>USECKA_B.DELKA)]USECKA_C
[USECKA_A.CISLO, USECKA_B.CISLO, USECKA_C.CISLO]
```

Kartézský součin

Kartézský součin jsme již probrali v kapitole věnující se relacím. Uváděli jsme si kartézský součin množin, kde jednotlivé prvky byly jednoduché (např. množina **A** měla prvky **1, 2, 3**; a množina **B** měla prvky **x, y**). Kartézský součin byly dvojice utvořené z prvků z množiny **A** a z prvků množiny **B**. Jelikož prvky v množinách **A** a **B** byly jednoduché, kartézský součin obsahoval dvojice (1+1).

V případě relací je situace velmi podobná. Do operace kartézský součin budou vstupovat opět dvě relace. Jediný rozdíl oproti jednoduchým množinám bude ten, že již relace samy o sobě obsahují jako prvky n-tice. Řekněme že relace R nad množinou atributů A obsahuje trojice (tj. 3 atributy) a relace S nad množinou atributů B obsahuje dvojice (2 atributy). Kartézský součin bude v tomto případě obsahovat pětice (3+2).

V řeči tabulek si lze kartézský součin představit jako „sloučení“ dvou tabulek. Výsledná tabulka bude obsahovat záznamy z obou původních tabulek, které vzniknou tak, že zkombinujeme jednotlivé záznamy z původních tabulek metodou „každý s každým“. Je jasné, že pokud jedna tabulka obsahuje 10 záznamů a druhá 5 záznamů, výsledná tabulka bude mít celkem 50 záznamů (10x5 kombinací).

V předchozí podkapitole v příkladu jsme měli kartézský součin dvou relací **USECKA_A** a **USECKA_B**. Každá z těchto relací byla popsána 2 atributy, a každou relaci splňovaly 3 dvojice. Kartézský součin bude tedy obsahovat čtveřice (2+2) a kartézský součin bude splňovat celkem 9 takových čtveřic (3x3).

Tabulka: USECKA_A

CISLO	DELKA
10	1
11	2
12	3

Tabulka: USECKA_B

CISLO	DELKA
13	1
14	2
15	3

Tabulka: USECKA_A×USECKA_B

USECKA_A.CISLO	USECKA_A.DELKA	USECKA_B.CISLO	USECKA_B.DELKA
10	1	13	1
10	1	14	2
10	1	15	3
11	2	13	1
11	2	14	2
11	2	15	3
12	3	13	1
12	3	14	2
12	3	15	3

Kartézský součin dvou relací **R** a **S** se zapisuje jako **R×S**.

Přirozené spojení dvou relací bylo definováno jako spojení dvou relací přes rovnost dvojic klíčů (primární v jedné z relací, cizí v té druhé). Přirozené spojení můžeme pomocí kartézského součinu a operace restrikce zapsat následovně:

R×S(R.CIZI_KLIC=S.PRIMARNI_KLIC)

Nyní si provedeme na příkladu důkaz, že tomu tak opravdu je. Jako příklad si vezměme naše záznamy o zaměstnancích a jejich funkcích, které vykonávají. Dvě původní relace **ZAMESTNANEC** a **FUNKCE** vypadají takto:

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE
1	Jan	Novák	15.10.1975	1
2	Petr	Nový	1.4.1978	2
3	Jan	Nováček	6.9.1965	3
4	David	Vokurka	5.12.1973	4

Tabulka: FUNKCE

ID	NAZEV	PLAT
1	uklízeč	15000
2	programátor	21500
3	manažer	17500
4	ředitel	28000

Jako relace **R** nám bude vystupovat relace **ZAMESTNANEC**, jako **S** pak relace **FUNKCE**. Cizím klíčem v relaci **ZAMESTNANEC** je **ID_FUNKCE**, primárním klíčem v relaci **FUNKCE** je atribut **ID**. Chceme tedy ukázat, že:

ZAMESTNANEC*FUNKCE

je totéž, co

ZAMESTNANEC×FUNKCE(ID_FUNKCE=ID)

Z kapitoly o přirozeném spojení již víme, že výsledkem **ZAMESTNANEC*FUNKCE** je následující relace:

Tabulka: ZAMESTNANEC*FUNKCE

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	ID	NAZEV	PLAT
1	Jan	Novák	15.10.1975	1	1	uklízeč	15000
2	Petr	Nový	1.4.1978	2	2	programátor	21500
3	Jan	Nováček	6.9.1965	3	3	manažer	17500
4	David	Vokurka	5.12.1973	4	4	ředitel	28000

Zbývá tedy ukázat výsledek operací **ZAMESTNANEC×FUNKCE(ID_FUNKCE=ID)**. Nejprve provedeme kartézský součin **ZAMESTNANEC×FUNKCE**. Výsledek této operace bude vypadat následovně:

Tabulka: ZAMESTNANEC×FUNKCE

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	ID	NAZEV	PLAT
1	Jan	Novák	15.10.1975	1	1	uklízeč	15000
1	Jan	Novák	15.10.1975	1	2	programátor	21500
1	Jan	Novák	15.10.1975	1	3	manažer	17500
1	Jan	Novák	15.10.1975	1	4	ředitel	28000
2	Petr	Nový	1.4.1978	2	1	uklízeč	15000
2	Petr	Nový	1.4.1978	2	2	programátor	21500
2	Petr	Nový	1.4.1978	2	3	manažer	17500
2	Petr	Nový	1.4.1978	2	4	ředitel	28000
3	Jan	Nováček	6.9.1965	3	1	uklízeč	15000
3	Jan	Nováček	6.9.1965	3	2	programátor	21500
3	Jan	Nováček	6.9.1965	3	3	manažer	17500
3	Jan	Nováček	6.9.1965	3	4	ředitel	28000
4	David	Vokurka	5.12.1973	4	1	uklízeč	15000
4	David	Vokurka	5.12.1973	4	2	programátor	21500
4	David	Vokurka	5.12.1973	4	3	manažer	17500
4	David	Vokurka	5.12.1973	4	4	ředitel	28000

Nyní zaměříme pozornost na sloupce **ID_FUNKCE** a **ID**. Všimněte si, že na některých řádcích jsou shodné hodnoty **ID_FUNKCE** a **ID** a na jiných jsou hodnoty různé. Provedeme restrikci podle podmínky **ID_FUNKCE=ID**. Výsledek bude následující:

Tabulka: ZAMESTNANEC×FUNKCE(ID_FUNKCE=ID)

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	ID	NAZEV	PLAT
1	Jan	Novák	15.10.1975	1	1	uklízeč	15000
2	Petr	Nový	1.4.1978	2	2	programátor	21500
3	Jan	Nováček	6.9.1965	3	3	manažer	17500
4	David	Vokurka	5.12.1973	4	4	ředitel	28000

Podíváme-li se na jednotlivé záznamy v tomto výsledku, tak zjistíme, že totéž nám dává přirozené spojení **ZAMESTNANEC*FUNKCE**.

Sjednocení, průnik, rozdíl

Má smysl uvažovat tyto tři množinové operace nad relacemi, neboť relace lze v jistém smyslu chápat jako množiny. Kromě klasických množin zde musíme uvážit jistá omezení oproti klasickým množinám. Ne vždy má sjednocení, průnik nebo rozdíl smysl nad jakýmikoliv relacemi, rovněž musíme klást určité podmínky, jaké relace mohou do těchto operací vstupovat.

Jednou z hlavních omezujících podmínek pro tyto množinové operace je tzv. „kompatibilita“ relací (tabulek). To znamená, že do množinové operace mohou vstupovat jen ty relace, které jsou kompatibilní. Kompatibilitou se rozumí:

- Shodný počet sloupců (atributů)
- Shodné typy dat v jednotlivých sloupcích

Lidově řečeno, nemůžeme míchat „hrušky s jabkama“.

Sjednocení

Do sjednocení vstupují dvě kompatibilní relace, výsledkem je jedna relace, která vznikne sjednocením původních relací. Značíme $\mathbf{R \cup S}$.

Vezměme si náš příklad relací **ZAMESTNANEC** a **FUNKCE** a mějme následující dotaz: kteří lidé ve firmě (zajímají nás jejich jména a příjmení) pracují jako programátor nebo jako analytik? Takový dotaz se dá zapsat v relační algebře s využitím sjednocení.

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE
1	Jan	Novák	15.10.1975	1
2	Petr	Nový	1.4.1978	2
3	Jan	Nováček	6.9.1965	3
4	David	Vokurka	5.12.1973	4

Tabulka: FUNKCE

ID	NAZEV	PLAT
1	uklízeč	15000
2	programátor	21500
3	manažer	17500
4	ředitel	28000

Nejprve zkonstruujeme „pod“-dotaz, kde se zeptáme na všechny programátory a po té druhý „pod“-dotaz, kde totéž provedeme na analytiky. Nakonec provedeme projekci na **JMENO** a **PRIJMENI**. Výsledkem bude pak sjednocení lidí, kteří pracují buď jako programátor nebo jako analytik:

```
(ZAMESTNANEC*FUNKCE(NAZEV=programátor)∪
ZAMESTNANEC*FUNKCE(NAZEV=analytik))[JMENO,PRIJMENI]
```

Samozřejmě dotaz „kteří lidé ve firmě pracují jako programátoři nebo jako analytici“ můžeme zapsat i bez použití operace sjednocení. Bude nám stačit přirozené spojení, restrikce a projekce:

```
ZAMESTNANEC*FUNKCE(NAZEV=programátor and NAZEV=analytik)[JMENO,
PRIJMENI]
```

Na tomto příkladu je vidět, že většinou existuje více způsobů jak daný dotaz zapsat. Každý způsob se bude zpracovávat pravděpodobně odlišně, nicméně výsledek musí vracet stejný. Jaký způsob zvolit je pak úlohou optimalizace dotazů pro konkrétní databázový systém, tím se zde zabývat nebudeme.

Průnik

Do průniku vstupují dvě kompatibilní relace, výsledkem je pak jedna relace, která vznikne průnikem dvou původních relací. Zapisujeme $\mathbf{R \cap S}$.

Vezměme si příklad o autorství knih, který jsme použili v kapitole věnované vazbě M:N v tabulkách:

Tabulka: KNIHA

ID	NÁZEV	JAZYK	ID_AUTOR
11	Oko	český	1
12	Bludiště	německý	4
13	Domek	český	4
14	Sauna a bazén	anglický	3

Tabulka: AUTORSTVÍ

ID_KNIHA	ID_AUTOR
11	1
11	5
14	3
12	4
13	4

Tabulka: AUTOR

ID	JMÉNO	PŘÍJMENÍ	DATUM_NAROZENÍ
1	Jan	Novák	15.10.1975
2	Petr	Nový	1.4.1978
3	Jan	Nováček	6.9.1965
4	David	Vokurka	5.12.1973
5	Vít	Mrkvička	30.1.1970

Uvažme následující dotaz: *Jaké knihy (zajímají nás názvy) napsali autoři Jan Novák a Vít Mrkvička jako spoluautoři?* Pokud bychom neuvažovali průnik a zaměřili se pouze na použití přirozeného spojení, restrikce a projekce, mohl by nás napadnout následující výraz v relační algebře:

```
((KNIHA*AUTORSTVI)*AUTOR)((JMENO=Jan and PRIJMENI=Novák) and
(JMENO=Vít and PRIJMENI=Mrkvička))[NAZEV]
```

Na první pohled by se mohlo zdát, že tento dotaz přesně řeší zadání, ale není tomu tak. Potíž je v tom, že restrikce se vždy vyhodnocuje v daném okamžiku v rámci jednoho řádku. A těžko může nějaký autor v tabulce **AUTOR** mít hodnotu sloupce **JMENO Jan** a zároveň **Vít**. To by bylo nesmyslné. Proto zde musíme přistoupit k formulaci dotazu pomocí operace průnik, jinou možnost nemáme. Dotaz bude vypadat:

```
(((KNIHA*AUTORSTVI)*AUTOR)(JMENO=Jan and PRIJMENI=Novák)∩
((KNIHA*AUTORSTVI)*AUTOR)(JMENO=Vít and PRIJMENI=´Mrkvička))
[NAZEV]
```

Projekci můžeme provádět buď až po skončení množinové operace (sjednocení, průnik, rozdíl) a nebo v každé „pod“-relaci vstupující do množinové operace. Výše uvedený dotaz s průnikem můžeme zapsat také takto:

```
((KNIHA*AUTORSTVI)*AUTOR(JMENO=Jan and PRIJMENI=Novák)[NAZEV])∩
((KNIHA*AUTORSTVI)*AUTOR(JMENO=Vít and PRIJMENI=Mrkvička)[NAZEV])
```

Když se podíváme na obsah původních tabulek, zjistíme, že odpovědí na tento dotaz je kniha **Oko**.

Rozdíl

Poslední zmiňovanou množinovou operací je rozdíl. Vstupem jsou opět dvě kompatibilní relace, výstupem pak relace, která vznikne rozdílem dvou původních relací. Zapisujeme **R-S**.

Pro nejjednodušší pochopení operace rozdílu si vezměme náš příklad o autorství knih.

Tabulka: KNIHA

ID	NÁZEV	JAZYK	ID_AUTOR
11	Oko	český	1
12	Bludiště	německý	4
13	Domek	český	4
14	Sauna a bazén	anglický	3

Tabulka: AUTORSTVÍ

ID_KNIHA	ID_AUTOR
11	1
11	5
14	3
12	4
12	3
13	4

Tabulka: AUTOR

ID	JMÉNO	PŘÍJMENÍ	DATUM_NAROZENÍ
1	Jan	Novák	15.10.1975
2	Petr	Nový	1.4.1978
3	Jan	Nováček	6.9.1965
4	David	Vokurka	5.12.1973
5	Vít	Mrkvička	30.1.1970

A formulujme následující dotaz: *Jaké knihy napsal David Vokurka bez toho, aniž by mu pomáhal Jan Nováček jako spoluautor?* Jinými slovy, hledáme takové knihy, které napsal **David Vokurka** a zároveň je nenapsal **Jan Nováček**. Použijeme operaci rozdíl:

```
((KNIHA*AUTORSTVI)*AUTOR(JMENO=David and PRIJMENI=Vokurka))-
((KNIHA*AUTORSTVI)*AUTOR(JMENO=Jan and PRIJMENI=Nováček))[NAZEV]
```

Opět zde máme možnost zapsat uvedený výraz i jiným způsobem:

```
((KNIHA*AUTORSTVI)*AUTOR(JMENO=David and PRIJMENI=Vokurka)
 [NAZEV])-
((KNIHA*AUTORSTVI)*AUTOR(JMENO=Jan and PRIJMENI=Nováček)[NAZEV])
```

Shrnutí

- **Relační algebra je matematický nástroj pro formulování dotazů nad relacemi (tabulkami)**
- **Základními operacemi, ryze databázovými, jsou projekce, restrikce, spojení.**
- **Dalšími operacemi, množinovými, jsou kartézský součin, sjednocení, průnik, rozdíl.**
- **Vstupem do operací je jedna nebo dvě relace, výstupem je opět relace. To umožňuje skládat operace za sebe, na výsledek jedné operace, aplikovat operaci další.**
- **V případě množinových operací jakými jsou sjednocení, průnik a rozdíl je potřeba zajistit tzv. kompatibilitu relací. Kompatibilitou se rozumí shodný počet sloupců (atributů) a shodný typ dat v jednotlivých sloupcích (atributech).**

Otázky a úkoly

- Nad relacemi konstruujte dotazy v relační algebře:

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	ID_PRACOVISTE
1	Jan	Novák	15.10.1975	1	11
2	Petr	Nový	1.4.1978	2	21
3	Jan	Nováček	6.9.1965	3	31
4	David	Vokurka	5.12.1973	4	41

Tabulka: POVOLANI

ID	FUNKCE	PLAT
1	uklízeč	15000
2	programátor	21500
3	manažer	17500
4	ředitel	28000

Tabulka: PRACOVISTE

ID	ODDELENI
11	chodba
21	studovna
31	kancelář
41	ředitelna

- *Jakou funkci vykonává David Vokurka?*
- *Jaký plat pobírá programátor?*
- *Všichni zaměstnanci (jméno, příjmení) starší 20ti let*
- *Kteří zaměstnanci (jméno, příjmení) pracují na chodbě?*
- *Na kterých pracovištích (název) pracují programátoři?*

- Nad relacemi konstruujte dotazy v relační algebře:

Tabulka: KNIHA

ID	NÁZEV	JAZYK
11	Oko	český
12	Bludiště	německý
13	Domek	český
14	Sauna a bazén	anglický

Tabulka: AUTORSTVÍ

ID_KNIHA	ID_AUTOR
11	1
11	5
14	3
12	4
13	4

Tabulka: AUTOR

ID	JMÉNO	PŘÍJMENÍ	DATUM_NAROZENÍ
1	Jan	Novák	15.10.1975
2	Petr	Nový	1.4.1978
3	Jan	Nováček	6.9.1965
4	David	Vokurka	5.12.1973
5	Vít	Mrkvička	30.1.1970

- *Jaké knihy (název) napsal Vít Mrkvička?*
- *Kdo napsal (jméno, příjmení) knihu „Bludiště“?*
- *V jakém jazyce byla napsána kniha „Sauna a bazén“?*
- *Jaké knihy (název) napsali Petr Nový a Jan Nováček jako spoluautoři?*
- *Kdy se narodil autor/autoři knihy „Domek“?*
- *V jakých jazycích napsal knihy Jan Novák?*
- *Jaké knihy (název) nenapsali Vít Mrkvička a David Vokurka jako spoluautoři?*

- Nad relacemi konstruujte dotazy v relační algebře:

Tabulka: BYT

CISLO	ADR_ULICE	ADR_CISLO	ADR_MESTO	ADR_PSC	PODLAZI	BALKON	SKLEP
1	Novákova	123	Brno	61500	2	A	N
2	Jirkova	2	Praha	11400	1	A	A
3	Tulíkova	900	Zlín	57890	7	N	N
4	Prackova	7	Znojmo	52109	3	A	A
5	Holíkova	1	Brno	62100	2	N	A
6	U Boba	100	Zlín	57880	6	N	N
7	Krtkova	88	Praha	11100	-1	N	N

- *Ve kterém městě (název) leží ulice Tulíkova?*
- *Které byty (číslo) leží nejvýše ve 2. podlaží?*
- *Jaká je přesná adresa bytu č. 5?*
- *Seznam bytů (čísla) s balkonem.*
- *Seznam bytů (čísla) buď s balkonem a nebo se sklepem (ale ne dohromady).*

Jazyk SQL

Jazyk SQL je jazykem relačních databází. Na jazyk SQL lze pohlížet jako na nástroj, který je integrován v SŘBD a dále jako na jazyk, pomocí kterého lze konstruovat dotazy. Počátky jazyka SQL spadají do 70. let 20. století, kdy vznikla jeho první verze pod názvem **SEQUEL**. Jednalo se o dotazovací jazyk – jednotlivé dotazy byly konstruovány pomocí tzv. strukturované angličtiny. Jazyk SEQUEL byl tedy předchůdcem jazyka SQL, jehož první verze se datuje do roku 1986. Během několika let se projevily drobné nedostatky, které byly opraveny a vznikla nová verze jazyka SQL v roce 1992, někdy označovaná jako **SQL92**. Tato verze byla standardizovaná ANSI a ISO. Tato verze je v podstatě dodnes využívána, její drobná vylepšení můžeme najít pod označeními SQL:1999, resp. SQL:2003. Zkratka SQL znamená **Structured Query Language**.

Jazyk SQL patří do skupiny tzv. deklarativních (neprocedurálních) programovacích jazyků. V praxi to znamená, že pomocí jazyka SQL popisujeme **CO** je zadáním úlohy a nezabýváme se tím, **JAK** bude úloha vyřešena. Například zkonstruujeme dotaz, který chceme, aby nám databázový systém odpověděl, ale už nás vůbec nezajímá, jakým algoritmem databázový systém projde jednotlivé tabulky a spočítá výsledek.

Jazyk SQL má poměrně široké využití. Prvním kritériem může být rozdělení jazyka podle toho, kdo jej bude používat. Každý uživatel využije jazyk SQL na jiné úrovni. S jazykem SQL se v podstatě mohou setkat následující skupiny uživatelů:

- Administrátoři a návrháři databázových systémů
- Programátoři uživatelských aplikací běžících nad relačním databázovým systémem
- Koncoví uživatelé, kteří si pomocí aplikací mohou konstruovat např. své dotazy nad databází

Jazyk SQL se skládá z několika částí:

- Jazyk pro definici dat (**DDL – Data Definition Language**) – tato část jazyka SQL umožňuje vytvářet, odstraňovat a modifikovat tabulky v relačních schématech
- Jazyk pro manipulaci dat (**DML – Data Manipulation Language**) – dovoluje uživateli/programátorovi formulovat dotazy a dále realizovat vkládání, mazání a aktualizaci záznamů v jednotlivých tabulkách
- Způsob ukládání tabulek je definován v jazyce **SDL – Storage Definition Language**
- Pro práci s virtuálními tabulkami (tzv. view) je určena část jazyka nazvaná **VDL – View Definition Language**
- Dynamický SQL – jazyk SQL je vnořen do hostitelského programovacího jazyka, kde je možno konstruovat SQL dotazy dynamicky za běhu programu

- Další části jazyka SQL jsou: definice přístupových práv, definice integritních omezení, nebo řízení databázových transakcí (např. **DCL – Data Control Language**)

V rámci této kapitoly se budeme zabývat převážně částmi jazyka DDL a DML. Předmětem této kapitoly budou následující témata:

- Datové typy sloupců
- Integritní omezení, primární a cizí klíč
- Vytváření, mazání a modifikace tabulek v relačních schématech
- Práce se záznamy – vložení nového, aktualizace a mazání existujících záznamů
- Konstrukce dotazů nad daty
  - Jednoduché dotazy
  - Skládání dotazů
  - Agregační funkce

Datové typy sloupců

Dříve než se pustíme do tvorby tabulek pomocí jazyka SQL, musíme si uvést přehled datových typů. Již víme, že každý atribut (sloupec) musí být nějakého datového typu. Z kapitoly o datovém modelování máme základní přehled, o jaké typy se jedná, nyní si je uvedeme přesněji, podle toho, jaké typy nám nabízí jazyk SQL.

- Numerické typy:
  - **`INTEGER`**: $-2^{31}-1$ až 2^{31}
  - **`SMALLINT`**: -32768 až 32767
  - **`NUMERIC(p,q)`** : p – počet platných číslic, q – počet desetinných míst
  - **`FLOAT(n)`** : reálné číslo s plovoucí desetinnou čárkou, n – počet des. míst
  - **`REAL`**: reálné číslo (rozsah dán implementací příslušného DBS)
- Znakové řetězce:
  - **`CHAR(n)`** : n-znakový řetězec (např. CHAR(1))
  - **`VARCHAR(n)`** : n-znakový řetězec (n max. 255)
  - **`CLOB(n)`** : dlouhé n-znakové řetězce (* v implementaci MySQL)
- Bitové řetězce:
  - **`BIT(n)`**: n udává počet bitů
- Temporální data (přesný formát závisí na konkrétní implementaci DBS):
  - **`DATE`**
  - **`TIME`**
  - **`TIMESTAMP`**

Uvedené datové typy nabízí většina databázových systémů. Můžeme se ale setkat s tím, že některé databázové systémy nebudou všechny tyto typy podporovat, pak je třeba se podívat do dokumentace konkrétního databázového systému, jaké datové typy nabízí.

Integritní omezení

Přehled a vysvětlení jednotlivých integritních omezeních jsme si vysvětlili už dříve. Klíčová slova, která jsme pro integritní omezení u sloupců tabulek používali, jsou většinou přesně ta, která můžeme nalézt v jazyce SQL. Pro naše účely budeme tedy používat následující integritní omezení:

- **`PRIMARY KEY(sloupec)`** – primární klíč
- **`FOREIGN KEY(sloupec) REFERENCES tabulka(sloupec)`** – cizí klíč
- **`NOT NULL`** – nenulová hodnota
- **`IS NULL`** – nulová hodnota
- **`UNIQUE`** – jednoznačná hodnota
- **`DEFAULT 'hodnota'`** – výchozí hodnota
- **`CHECK(podmínka)`** – ověření platnosti podmínky

Integritní omezení lze vzájemně kombinovat, pro jeden sloupec jich lze uvést více.

Tabulky

Základní operace s definicemi tabulek jsou:

- Vytvoření tabulky (**`CREATE TABLE`**)
- Aktualizace sloupců tabulky (**`ALTER TABLE`**)
- Smazání tabulky (**`DROP TABLE`**)

Všechny tyto operace se řídí několika pravidly:

- Je potřeba brát ohled na pořadí vytváření tabulek. Tabulka, která obsahuje primární klíč, jehož hodnota je odkazována z jiné tabulky pomocí cizího klíče, musí být vytvořena dříve, než tabulka, ze které se odkazujeme
- Podobné to je i při modifikaci tabulek, případně při mazání tabulek. Nesmí se stát, že by v nějaké tabulce zůstal cizí klíč, který by odkazoval na neexistující tabulku (třeba tím, že odkazovaná tabulka byla smazána)
- Sloupce v tabulce musí mít (v rámci tabulky) jednoznačné názvy
- Každý sloupec musí mít definovaný datový typ
- U každého sloupce můžeme uvést integritní omezení (dobrovolné)

Uvedené operace nad definicemi tabulek spadají do části jazyka DDL.

Vytvoření tabulky

Tabulku vytvoříme v jazyce SQL pomocí příkazu **`CREATE TABLE`**. Jeho syntaxe je následující:

```
CREATE TABLE název_tabulky (
jméno_sloupce_1  typ_sloupce_1 [ integritní_omezení ],
jméno_sloupec_2  typ_sloupce_2 [ integritní_omezení ],
...
PRIMARY KEY (sloupce...),
FOREIGN KEY (sloupec) REFERENCES jiná_již_existující_tabulka (sloupec),
...
);
```

Příklad: vytvořte tabulku ZAMESTNANEC pomocí jazyka SQL
Nechť tabulka **ZAMESTNANEC** vypadá následovně (sloupec **CISLO** je jejím primárním klíčem):

Tabulka: ZAMESTNANEC

CISLO	JMENO	PRIJMENI	DAT_NAROZ	SMLOUVA_OD
1	Jan	Novák	15.10.1975	1.1.2000
2	Petr	Nový	1.4.1978	12.5.1999
3	Jan	Nováček	6.9.1965	7.7.1998
4	David	Vokurka	5.12.1973	1.10.2002

SQL:

```
CREATE TABLE ZAMESTNANEC (
CISLO       INTEGER,
JMENO       VARCHAR(5),
PRIJMENI    VARCHAR(10),
DAT_NAROZ   DATE,
SMLOUVA_OD  DATE,
PRIMARY KEY(CISLO));
```

Jelikož můžeme u všech sloupců požadovat, aby hodnota byla vždy vyplněna, měli bychom k definici každého sloupce přidat ještě integritní omezení **NOT NULL**. Výsledný SQL příkaz pro vytvoření tabulky **ZAMESTNANEC** by vypadal tedy takto:

```
CREATE TABLE ZAMESTNANEC (
CISLO       INTEGER,
JMENO       VARCHAR(5)  NOT NULL,
PRIJMENI    VARCHAR(10) NOT NULL,
DAT_NAROZ   DATE        NOT NULL,
SMLOUVA_OD  DATE        NOT NULL,
PRIMARY KEY(CISLO));
```

U sloupce **CISLO** nemusíme integritní omezení **NOT NULL** zadávat, neboť je definován jako primární klíč a už z této definice plyne, že nesmí mít prázdnou hodnotu.

Příklad: vytvořte tabulky ZAMESTNANEC a FUNKCE pomocí jazyka SQL
Nechť tabulky **ZAMESTNANEC** a **FUNKCE** jsou následující (sloupec **CISLO** v tabulce **ZAMESTNANEC** je primárním klíčem, sloupec **ID_FUNKCE** v tabulce **ZAMESTNANEC** je

cizím klíčem do tabulky **FUNKCE** na její primární klíč **ID**). Nechť dále platí, že výše platu musí být alespoň 10000 Kč.

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE
1	Jan	Novák	15.10.1975	1
2	Petr	Nový	1.4.1978	2
3	Jan	Nováček	6.9.1965	3
4	David	Vokurka	5.12.1973	4

Tabulka: FUNKCE

ID	NAZEV	PLAT
1	uklízeč	15000
2	programátor	21500
3	manažer	17500
4	ředitel	28000
5	účetní	23000
6	vrátný	10000

SQL: z důvodu, že tabulka **FUNKCE** je odkazovaná (odkazuje na ni cizí klíč z tabulky **ZAMESTNANEC**), musíme tabulku **FUNKCE** vytvořit jako první.

```
CREATE TABLE FUNKCE (
ID          INTEGER,
NAZEV       VARCHAR(10) NOT NULL,
PLAT        INTEGER     CHECK(PLAT>=10000),
PRIMARY KEY(ID));

CREATE TABLE ZAMESTNANEC (
CISLO       INTEGER,
JMENO       VARCHAR(5)  NOT NULL,
PRIJMENI    VARCHAR(10) NOT NULL,
DAT_NAROZ   DATE        NOT NULL,
ID_FUNKCE   INTEGER,
PRIMARY KEY(CISLO),
FOREIGN KEY(ID_FUNKCE) REFERENCES FUNKCE(ID));
```

Příklad: vytvořte tabulku HRA_SACH s násobným primárním klíčem v jazyce SQL
Nechť tabulka vypadá následovně (primárním klíčem pro každý záznam v této tabulce je čtveřice sloupců **CISLO_HRAC1**, **CISLO_HRAC2**, **DATUM** a **CAS**):

Tabulka: HRA_SACH

CISLO_HRAC1	CISLO_HRAC2	DATUM	CAS	VYSLEDEK	JM_HRAC1	JM_HRAC2
10	12	12.1.2006	12:00	1:0	Jan	Pavel
11	12	14.1.2006	13:05	0:1	Karel	Pavel
12	14	14.1.2006	15:13	pat	Pavel	Petr

SQL:

```
CREATE TABLE HRA_SACH (
CISLO_HRAC1 INTEGER,
CISLO_HRAC2 INTEGER,
DATUM       DATE,
CAS         TIME,
VYSLEDEK    VARCHAR(5),
JM_HRAC1    VARCHAR(10) NOT NULL,
JM_HRAC2    VARCHAR(10) NOT NULL,
PRIMARY KEY(CISLO_HRAC1,CISLO_HRAC2,DATUM,CAS));
```

Aktualizace tabulky

Aktualizací tabulky rozumíme změnu definice sloupců tabulky, nikoliv aktualizaci záznamů v tabulce (ta se realizuje příkazem **UPDATE**). Příkaz pro aktualizaci definice tabulky je **ALTER TABLE**. Jeho syntaxe je následující:

```
ALTER TABLE název_tabulky (
<OPERACE> sloupec [typ] [integritní omezení]
);
```

kde **<OPERACE>** může být:

- **ADD COLUMN** – přidání nového sloupce do tabulky, např. **ADD COLUMN RODNE_CISLO VARCHAR(11) UNIQUE**
- **DROP COLUMN** – smazání existujícího sloupce v tabulce, např. **DROP COLUMN SMLOUVA_OD**
- **ADD CONSTRAINT** – přidání integritního omezení k existujícímu sloupci, např. **ADD CONSTRAINT VYSLEDEK NOT NULL**
- **DROP CONSTRAINT** – zrušení integritního omezení u existujícího sloupce, např. **DROP CONSTRAINT JM_HRAC1**

Příklad: do tabulky ZAMESTNANEC přidejte nový sloupec NAROK_STRAVENKY s výchozí hodnotou „A“
Vezměme naši původní tabulku ZAMESTNANEC:

Tabulka: ZAMESTNANEC

CISLO	JMENO	PRIJMENI	DAT_NAROZ	SMLOUVA_OD
1	Jan	Novák	15.10.1975	1.1.2000
2	Petr	Nový	1.4.1978	12.5.1999
3	Jan	Nováček	6.9.1965	7.7.1998
4	David	Vokurka	5.12.1973	1.10.2002

SQL:

```
ALTER TABLE ZAMESTNANEC (
ADD COLUMN NAROK_STRAVENKY    DEFAULT 'A');
```

Po provedení tohoto příkazu bude tabulka **ZAMESTNANEC** vypadat následovně:

```
Tabulka: ZAMESTNANEC
```

CISLO	JMENO	PRIJMENI	DAT_NAROZ	SMLOUVA_OD	NAROK_STRAVENKY
1	Jan	Novák	15.10.1975	1.1.2000	
2	Petr	Nový	1.4.1978	12.5.1999	
3	Jan	Nováček	6.9.1965	7.7.1998	
4	David	Vokurka	5.12.1973	1.10.2002	

To, jestli bude ve sloupci **NAROK_STRAVENKY** automaticky předvyplněna hodnota **A** závisí na konkrétní implementaci databázového systému. Pro doplnění hodnoty do sloupce **NAROK_STRAVENKY** je potřeba použít příkaz pro aktualizaci záznamů **UPDATE**, který probereme později.

Příklad: v tabulce FUNKCE změňte integritní omezení u sloupce PLAT, tak aby minimální výše platu byla 12000 Kč:
Máme-li změnit integritní omezení u některého sloupce, musíme původní integritní omezení zrušit a nové definovat:

```
ALTER TABLE FUNKCE (
DROP CONSTRAINT PLAT,
ADD CONSTRAINT PLAT CHECK(PLAT>=12000));
```

Odstranění tabulky

Odstranění tabulky a smazání celého obsahu tabulky není totéž a často se stává, že dochází k záměně těchto operací. Odstraněním tabulky (pomocí příkazu **DROP TABLE**) se má na mysli odstranění definice tabulky z databáze, jinými slovy, tabulka přestane existovat. Smazání tabulky (pomocí příkazu **DELETE**) znamená smazání všech jejich záznamů, tj. tabulka bude nadále v databázi existovat a bude prázdná (0 záznamů).

Syntaxe příkazu **DROP TABLE** je velmi jednoduchá a vypadá následovně:

```
DROP TABLE název_tabulky
```

Příklad: zrušte tabulku HRA_SACH z databáze

SQL:

```
DROP TABLE HRA_SACH;
```

Příklad: vezměme relační schéma tabulek ZAMESTNANEC a FUNKCE a pokusme se zrušit (odstranit) tabulku FUNKCE
Naše dvě tabulky vypadají takto:

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE
1	Jan	Novák	15.10.1975	1
2	Petr	Nový	1.4.1978	2
3	Jan	Nováček	6.9.1965	3
4	David	Vokurka	5.12.1973	4

Tabulka: FUNKCE

ID	NAZEV	PLAT
1	uklízeč	15000
2	programátor	21500
3	manažer	17500
4	ředitel	28000
5	účetní	23000
6	vrátný	10000

Podle výše uvedeného, použijeme příkaz pro odstranění tabulky:

```
DROP TABLE FUNKCE;
```

Tento příkaz ovšem skončí s chybou a neprovede se. Důvodem je skutečnost, že po odstranění tabulky **FUNKCE** by zůstal neplatný odkaz z tabulky **ZAMESTNANEC** (sloupec **ID_FUNKCE** by ukazoval na sloupec **ID** neexistující tabulky **FUNKCE**). Jak tedy odstranit tabulku **FUNKCE**? Máme dvě možnosti:

- Buď před tabulkou **FUNKCE** odstraníme také tabulku **ZAMESTNANEC**
- Nebo před odstraněním tabulky **FUNKCE** odstraníme sloupec **ID_FUNKCE** z tabulky **ZAMESTNANEC** a tabulku **ZAMESTNANEC** jinak zachováme

První možnost můžeme realizovat následovně:

```
DROP TABLE ZAMESTNANEC;
DROP TABLE FUNKCE;
```

Nebo můžeme použít jeden příkaz:

```
DROP TABLE FUNKCE CASCADE;
```

Tím, že jsme do příkazu uvedli klíčové slovo **CASCADE**, docílili jsme toho, že před odstraněním tabulky **FUNKCE** budou automaticky (kaskádovitě) odstraněny také všechny tabulky, které na tabulku **FUNKCE** odkazují. Kdyby na tabulku **ZAMESTNANEC** odkazovala například ještě tabulka **CLOVEK**, pak by byla v případě tohoto příkazu odstraněna také tabulka **CLOVEK**.

S klíčovým slovem **CASCADE** se musí nakládat velmi opatrně. Při zajímavějším provázání a propojení tabulek mezi sebou, se může stát, že jediným příkazem pro

odstranění jediné tabulky se kaskádovitě odstraní všechny tabulky v databázi a vám zůstanou jen oči pro pláč.

Druhá možnost spočívá v použití dvou následujících příkazů:

```
ALTER TABLE ZAMESTNANEC (
DROP COLUMN ID_FUNKCE);

DROP TABLE FUNKCE;
```

Záznamy

Jakákoliv manipulace se záznamy spadá do části jazyka DML. V našem kontextu budeme za manipulaci se záznamy považovat následující:

- Vložení nového záznamu (**INSERT**)
- Aktualizace existujícího záznamu (**UPDATE**)
- Smazání existujícího záznamu (**DELETE**)

Uvedené operace se opět řídí pravidly:

- Musí se brát v úvahu integritní omezení a typ dat jednotlivých sloupců
- Pokud záznam obsahuje cizí klíč odkazující na jiný záznam v jiné tabulce, musí takový odkazovaný záznam již existovat

Vkládání záznamů

Vložením nového záznamu do tabulky rozumíme přidání nového řádku do tabulky. Příkaz pro vložení v jazyce SQL je příkaz **INSERT**, jehož syntaxe je následující:

```
INSERT INTO název_tabulky (seznam sloupců)
VALUES (seznam hodnot vkládaného záznamu č. 1)[,
       (seznam hodnot vkládaného záznamu č. 2),
       ...];
```

Příklad: do tabulky ZAMESTNANEC vložte nový záznam (6, David, Dvořák, 14.3.1980, 3.3.2006)
Nechť tabulka **ZAMESTNANEC** vypadá následovně:

Tabulka: ZAMESTNANEC

CISLO	JMENO	PRIJMENI	DAT_NAROZ	SMLOUVA_OD
1	Jan	Novák	15.10.1975	1.1.2000
2	Petr	Nový	1.4.1978	12.5.1999
3	Jan	Nováček	6.9.1965	7.7.1998
4	David	Vokurka	5.12.1973	1.10.2002

SQL:

```
INSERT INTO ZAMESTNANEC (CISLO,JMENO,PRIJMENI,DAT_NAROZ,SMLOUVA_OD)
VALUES (6, 'David', 'Dvořák',14.3.1980,3.3.2006)
```

Řetězcové hodnoty (u sloupců definovaných jako **CHAR** nebo **VARCHAR**) je potřeba zadávat do apostrofů.

Pokud zadáváme nový záznam, který definuje hodnoty všech sloupců a to přesně v pořadí, v jakém sloupce jsou, pak se nemusí seznam sloupců v příkazu **INSERT** uvádět:

```
INSERT INTO ZAMESTNANEC
VALUES (6, 'David', 'Dvořák',14.3.1980,3.3.2006)
```

Ale obecně se doporučuje vždy seznam sloupců uvádět, neboť v budoucnosti se může změnit struktura databáze (změní se některé sloupce v tabulce) a příkaz **INSERT** by přestal fungovat, neboť by jednotlivé zadané hodnoty neodpovídaly správným sloupcům.

Při vkládání záznamu se kontrolují všechna integritní omezení a pokud je nějaké porušení, vložení záznamu skončí s chybou a neprovede se. Konkrétní reakce na chyby tohoto typu jsou záležitosti implementace daného databázového systému.

Příklad: přidání nových záznamů do tabulek ZAMESTNANEC a FUNKCE

Vezměme si náš příklad tabulek **ZAMESTNANEC** a **FUNKCE** a řekněme, že potřebujeme do databáze zavést novou skutečnost: do firmy jsme přijali úplně nového člověka (Janu Novákovou, narozenou 3.10.1970 na úplně novou pozici personalisty s platem 30000 Kč). Nechť stávající tabulky **ZAMESTNANEC** a **FUNKCE** vypadají následovně:

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE
1	Jan	Novák	15.10.1975	1
2	Petr	Nový	1.4.1978	2
3	Jan	Nováček	6.9.1965	3
4	David	Vokurka	5.12.1973	4

Tabulka: FUNKCE

ID	NAZEV	PLAT
1	uklízeč	15000
2	programátor	21500
3	manažer	17500
4	ředitel	28000
5	účetní	23000
6	vrátný	10000

Jelikož tabulka **FUNKCE** je odkazovaná, musíme nejprve vložit záznam do této tabulky – sem zavedeme novou pracovní pozici „personalista“ s platem 30000 Kč. Po té vložíme do tabulky **ZAMESTNANEC** nový záznam pro výše zmíněnou Janu Novákovou:

```
INSERT INTO FUNKCE (ID,NAZEV,PLAT)
VALUES (7,'personalista',30000);

INSERT INTO ZAMESTNANEC (CISLO,JMENO,PRIJMENI,DAT_NAROZ,ID_FUNKCE)
VALUES (8,'Jana','Nováková',3.10.1970,7);
```

Uvedené příkazy **INSERT** budou mít vliv na výsledek:

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE
1	Jan	Novák	15.10.1975	1
2	Petr	Nový	1.4.1978	2
3	Jan	Nováček	6.9.1965	3
4	David	Vokurka	5.12.1973	4
8	Jana	Nováková	3.10.1970	7

Tabulka: FUNKCE

ID	NAZEV	PLAT
1	uklízeč	15000
2	programátor	21500
3	manažer	17500
4	ředitel	28000
5	účetní	23000
6	vrátný	10000
7	personalista	30000

Aktualizace záznamů

Aktualizací existujícího záznamu nebo skupiny záznamů máme na mysli změnu hodnoty v konkrétním sloupci nebo sloupcích. Jednotlivé záznamy, kterých se aktualizace týká, jsou určeny podmínkou restrikce, jejíž význam je stejný jako u relační algebry.

Při provádění aktualizace se opět musí brát v úvahu integritní omezení. Pokud by aktualizací nějaké hodnoty ve sloupci mělo dojít k porušení integritního omezení, databázový systém nahlásí chybu a aktualizaci neumožní provést.

Aktualizace záznamů se realizuje příkazem **UPDATE**. Jeho syntaxe je následující:

```
UPDATE název_tabulky
SET sloupec=hodnota[, další_sloupec=hodnota, ...]
[WHERE restrikce];
```

Určení tabulky, ve které se bude provádět aktualizace je dána v **název_tabulky** za klíčovým slovem **UPDATE**. Konkrétní sloupce, jejichž hodnoty se mají nastavit jsou specifikovány v části **SET**. Jednotlivé výrazy se skládají z názvu sloupce, znaku rovnítka a dále z hodnoty. Tou hodnotou může být:

- Konkrétní číslo, řetězec, atd. (dle typu sloupce)

- Matematický výraz složený z běžných aritmetických operátorů a konstant
- Matematický výraz složený z běžných aritmetických operátorů a názvů jiných sloupců, jejichž hodnoty se pro výpočet použijí

Pokud budeme chtít, aby se příkaz aktualizace realizoval na každém řádku v tabulce, potom nebudeme v příkazu **UPDATE** uvádět část **WHERE**. Pokud ale budeme chtít specifikovat jenom některé řádky, na kterých se má aktualizace provést, musíme uvést příslušné restrikce v části **WHERE**. Restrikce je logická podmínka, tak jak ji známe z kapitoly o relační algebře. Může se také jednat o podmínku kombinovanou (s použitím logických spojek and, or, případně not).

Použití příkazu **UPDATE** si ukážeme názorně na následujících příkladech. Mějme tabulky **ZAMESTNANEC** a **FUNKCE** s následujícím počátečním obsahem:

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE
1	Jan	Novák	15.10.1975	1
2	Petr	Nový	1.4.1978	2
3	Jan	Nováček	6.9.1965	3
4	David	Vokurka	5.12.1973	4
8	Jana	Nováková	3.10.1970	7

Tabulka: FUNKCE

ID	NAZEV	PLAT
1	uklízeč	15000
2	programátor	21500
3	manažer	17500
4	ředitel	28000
5	účetní	23000
6	vrátný	10000
7	personalista	30000

Pokusme se nyní provést aktualizace v tabulkách tak, aby nám vyřešili následující situace, které ve firmě nastaly:

- *Jan Novák povýšil, z uklízeče se stal vrátným*
```
UPDATE ZAMESTNANEC
SET ID_FUNKCE=6
WHERE JMENO='Jan' PRIJMENI='Novák';
```
- *Všem účetním se zvyšuje plat o 1000 Kč*
```
UPDATE FUNKCE
SET PLAT=PLAT+1000
WHERE NAZEV='účetní';
```
- *Uklízečům se snižuje plat o 10%*
```
UPDATE FUNKCE
SET PLAT=0,9*PLAT
WHERE NAZEV=`uklízeč`;
```
- *Ředitelovi se plat zdvojnásobuje*
```
UPDATE FUNKCE
```

```
SET PLAT=2*PLAT
WHERE NAZEV=`ředitel`;
```

- *Všem ve firmě se zvyšuje plat o 300 Kč*

```
UPDATE FUNKCE
SET PLAT=PLAT+300;
```

Po provedení všech uvedených příkazů **UPDATE** bude obsah tabulek **ZAMESTNANEC** a **FUNKCE** vypadat následovně:

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE
1	Jan	Novák	15.10.1975	6
2	Petr	Nový	1.4.1978	2
3	Jan	Nováček	6.9.1965	3
4	David	Vokurka	5.12.1973	4
8	Jana	Nováková	3.10.1970	7

Tabulka: FUNKCE

ID	NAZEV	PLAT
1	uklízeč	13800
2	programátor	21800
3	manažer	17800
4	ředitel	56300
5	účetní	24300
6	vrátný	10300
7	personalista	30300

Smazání záznamů

Poslední ze základních operací pro manipulaci se záznamy je smazání záznamu. K realizaci smazání nám slouží příkaz **DELETE**, jehož syntaxe je ve srovnání s příkazy **INSERT** nebo **UPDATE** nejjednodušší, zároveň ale nechtěné provedení příkazu **DELETE** může napáchat hodně škod, neboť může dojít k trvalé ztrátě dat.

Syntaxe příkazu **DELETE** je poměrně jednoduchá:

```
DELETE FROM název_tabulky
[WHERE restrikce]
```

Hodnota **název_tabulky** opět specifikuje, ve které tabulce se mají mazat záznamy. Podmínka restrikce definuje, které záznamy mají být smazány. Smazány budou právě ty záznamy, které přesně splňují uvedenou podmínku (nebo podmínky).

Při mazání záznamů musíme brát v úvahu provázanost tabulek mezi sebou a vyvarovat se situaci, že bychom smazali nějaký záznam, který obsahuje klíč, na jehož hodnotu se odkazuje z jiné tabulky. Dobrý databázový systém by v případě pokusu o smazání „referencovaného“ záznamu měl vypsat chybovou hlášku a smazání daného záznamu nedovolit.

Vezměme si náš příklad tabulek **ZAMESTNANEC** a **FUNKCE** a zrealizujme několik příkladů na použití příkazu **DELETE** – pokusme se zaznamenat následující situace do databáze (a diskutujme, zdali se smazání provede úspěšně)

- *Paní Jana Nováková přestala být zaměstnankyní naší firmy*:
  ```
  DELETE FROM ZAMESTNANEC
  WHERE JMENO="Jana" AND PRIJMENI="Nováková";
  ```
  - provede se úspěšně
- *V naší firmě jsme zrušili funkci uklízeče*:
  ```
  DELETE FROM FUNKCE WHERE NAZEV="uklízeč";
  ```
  - provede se úspěšně
- *V naší firmě rušíme všechny funkce, které mají plat vyšší než 25000 Kč*:
  ```
  DELETE FROM FUNKCE WHERE PLAT>25000;
  ```
  - podmínce vyhovují 2 záznamy, ovšem záznam č. 4 (ředitel) se nesmaže, neboť na něj odkazujeme z tabulky **ZAMESTNANEC** (konkrétně *David Vokurka* plní funkci *ředitele*)

Po provedení výše uvedených příkazů **DELETE** obsah tabulek **ZAMESTNANEC** a **FUNKCE** je následující:

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE
1	Jan	Novák	15.10.1975	6
2	Petr	Nový	1.4.1978	2
3	Jan	Nováček	6.9.1965	3
4	David	Vokurka	5.12.1973	4

Tabulka: FUNKCE

ID	NAZEV	PLAT
2	programátor	21800
3	manažer	17800
4	ředitel	56300
5	účetní	24300
6	vrátný	10300

Dotazy

V poslední podkapitole věnované jazyku SQL se budeme věnovat podrobněji tvorbě dotazů. Pozornější čtenář zjistí, že tvorba dotazů v jazyce SQL a tvorba dotazů v relační algebře mají k sobě hodně blízko a pokud čtenář pochopil principy tvorby dotazů v relační algebře, pak i tvorba dotazů v jazyce SQL by mu neměla činit velké problémy.

Příkazem pro položení dotazu nad databází je příkaz **SELECT**. Jeho syntaxe je ze všech příkazů jazyka SQL nejsložitější. Na druhou stranu ale tento příkaz je ze všech nejmocnější a skýtá velké možnosti jeho použití. V této kapitole probereme několik základních rysů příkazu **SELECT** v následujících bodech:

- Jednoduché dotazy nad jednou tabulkou
- Spojování tabulek, dotazy nad více tabulkami
- Skládání dotazů (jejich sjednocení, průnik, rozdíl)
- Dotazy s agregačními funkcemi

Jednoduché dotazy

Nejjednodušší verze syntaxe příkazu **SELECT** vypadá následovně:

```
SELECT jména_sloupců
FROM jména_tabulek
WHERE podmínka
ORDER BY podmínka_řazení
```

- **jména_sloupců** – zde uvedeme názvy těch sloupců, které nás ve výsledku zajímají – uvedeme jednotlivé názvy, mezi sebou oddělené čárkou; z pohledu relační algebry tyto jména sloupců určují projekci
- **jména_tabulek** – v případě jednoduchých dotazů nad jednou tabulkou zde bude uveden název jedné tabulky, v opačném případě (při dotazu nad více tabulkami) zde bude uveden seznam názvů tabulek, vzájemně oddělených čárkou
- **podmínka** – podmínka restrikce, tak jak ji známe z relační algebry; může zde být jedna jednoduchá podmínky, nebo celý booleovský výraz složený z několika podmínek a operátorů **AND**, **OR**, **NOT** a jiných.
- **podmínka_řazení** – seznam názvů sloupců, podle kterých se má seřadit výstup (jako odpověď) na dotaz

Části **SELECT** a **FROM** musí být uvedena vždy, části **WHERE** a **ORDER BY** nejsou povinné. Vezměme si naše příklady tabulek z kapitoly věnované relační algebře a pokusme se zkonstruovat postupně všechny dotazy, které jsme již zkonstruovali v relační algebře, v jazyce SQL:

Tabulka: ZAMESTNANEC

CISLO	JMENO	PRIJMENI	DAT_NAROZ	SMLOUVA_OD
1	Jan	Novák	15.10.1975	1.1.2000
2	Petr	Nový	1.4.1978	12.5.1999
3	Jan	Nováček	6.9.1965	7.7.1998
4	David	Vokurka	5.12.1973	1.10.2002

- *Seznam všech zaměstnanců (jména a příjmení) ve firmě*:
- **SELECT JMENO, PRIJMENI**
- **FROM ZAMESTNANEC;**

Výsledek:

```
JMENO PRIJMENI
Jan   Novák
Petr  Nový
Jan   Nováček
David Vokurka
```

- *Abecedně seřazený seznam všech zaměstnanců ve firmě*:
  ```
  SELECT JMENO, PRIJMENI
  FROM ZAMESTNANEC
  ORDER BY PRIJMENI, JMENO;
  ```

Výsledek:

```
JMENO PRIJMENI
Jan   Nováček
Jan   Novák
Petr  Nový
David Vokurka
```

Chceme-li vypsat všechny sloupce, nemusíme je všechny vyjmenovávat v části **SELECT**, ale můžeme místo nich použít znak hvězdičky:

- *Údaje o všech zaměstnancích ve firmě, setříděné dle datumu narození*:
  ```
  SELECT * FROM ZAMESTNANEC
  ORDER BY DAT_NAROZ;
  ```

Výsledek:

```
CISLO JMENO PRIJMENI    DAT_NAROZ   SMLOUVA_OD
3     Jan   Nováček     6.9.1965    7.7.1998
4     David Vokurka     5.12.1973   1.10.2002
1     Jan   Novák       15.10.1975  1.1.2000
2     Petr  Nový        1.4.1978    12.5.1999
```

Podmínka restrikce

Již víme, že podmínka restrikce se skládá z jedné nebo více atomických formulí tvaru $\mathbf{t}_1 \theta \mathbf{t}_2$, jak tomu bylo v případě relační algebry. Ve srovnání s relační algebrou máme však v jazyce SQL více možností pro zápis různých podmínek:

- Relační operátory: **A<hodnota**, **A>hodnota**, **A<=hodnota**, **A>=hodnota**, **A=hodnota** (rovná se), **A<>hodnota** (nerovná se)
- **A IS NULL** (hodnota sloupce **A** je prázdná/nulová), **A IS NOT NULL** (hodnota sloupce **A** není prázdná/nulová)
- **A LIKE 'regulární výraz'** (regulární výraz obsahuje normální text, s tím, že část textu lze nahradit speciálním znakem **%**, jedno písmeno lze nahradit znakem _, např. tedy **PRIJMENI LIKE 'N%'** – tuto podmínku splňují všichni,

jejichž příjmení začíná písmenem **N**. Znaky **%** a podtržítka _ lze vzájemně v regulárním výrazu kombinovat, např: **JMENO LIKE 'L_O%D'**.

Vezměme si náš příklad tabulky **ZAMESTNANEC** a formulujme následující dotazy:

Tabulka: ZAMESTNANEC

CISLO	JMENO	PRIJMENI	DAT_NAROZ	PLAT	SMLOUVA_OD
1	Jan	Novák	15.10.1975	15000	1.1.2000
2	Petr	Nový	1.4.1978	21500	12.5.1999
3	Jan	Nováček	6.9.1965	17500	7.7.1998
4	David	Vokurka	5.12.1973	28000	1.10.2002

- *Jména a příjmení všech zaměstnanců, kteří pracují ve firmě déle než 3 roky*:

```
SELECT JMENO, PRIJMENI
FROM ZAMESTNANEC
WHERE SMLOUVA_OD<=1.1.2003;
```

Výsledek:

JMENO	PRIJMENI
Jan	Novák
Petr	Nový
Jan	Nováček
David	Vokurka

- *Jména a příjmení všech zaměstnanců, kterým je více než 30 let*:

```
SELECT JMENO, PRIJMENI
FROM ZAMESTNANEC
WHERE DAT_NAROZ < 31.12.1975;
```

Výsledek:

JMENO	PRIJMENI
Jan	Novák
Jan	Nováček
David	Vokurka

- *Příjmení všech zaměstnanců, kteří se křestním jménem jmenují Jan*:

```
SELECT PRIJMENI
FROM ZAMESTNANEC
WHERE JAN='Jan';
```

Výsledek:

JMENO	PRIJMENI
Jan	Novák
Jan	Nováček

- *Jména a příjmení všech zaměstnanců, kteří nejsou ve zkušební době a pobírají plat vyšší jak 20000 Kč*:

```
SELECT JMENO, PRIJMENI
FROM ZAMESTNANEC
WHERE SMLOUVA_OD<1.1.2006 AND PLAT>20000;
```

Výsledek:

```
JMENO PRIJMENI
=============
Petr  Nový
David Vokurka
```

- *Kdy se narodil Jan Novák?*

```
SELECT DAT_NAROZ
FROM ZAMESTNANEC
WHERE JMENO=`Jan` AND PRIJMENI='Novák';
```

Výsledek:

```
DAT_NAROZ
=========
15.10.1975
```

- *Jaký plat pobírá a od kdy pracuje ve firmě David Vokurka?*

```
SELECT PLAT, SMLOUVA_OD
FROM ZAMESTNANEC
WHERE JMENO='David' AND PRIJMENI='Vokurka';
```

Výsledek:

```
PLAT  SMLOUVA_OD
================
28000 1.10.2002
```

Dotazy nad více tabulkami

Chceme-li konstruovat dotazy nad více tabulkami, musíme pomocí jazyka SQL realizovat jejich přirození spojení, které již známe z relační algebry. Přirozené spojení dvou a více tabulek se pomocí SQL realizuje velmi snadno a to v zásadě ve dvou krocích:

- Do části **FROM** uvedeme seznam všech tabulek, které vstupují do dotazu a které budeme „spojovat“.
- V části **WHERE** specifikujeme podmínku, podle které se má zrealizovat spojení. Zpravidla, pokud budeme mít např. 2 tabulky **R** a **S**, a bude možno je spojit přes dvojici primární (v tabulce **S**) a cizí klíč (v tabulce **R**), potom specifikujeme rovnost. **R.primární_klíč = S.cizí_klíč**. Pokud by nám při spojení dvou a více tabulek došlo na nejednoznačné názvy sloupců (různé tabulky budou mít stejně pojmenované některé sloupce), potom využijeme tečkové notace a před každý název sloupce uvedeme název tabulky, ke které náleží.

- Dojde-li k nejednoznačnosti názvů sloupců, potom i v části **SELECT** uvádíme názvy sloupců v tečkové notaci spolu s názvy tabulek, odkud sloupce pochází.

Vezměme si příklad tabulek **ZAMESTNANEC** a **FUNKCE** a zkonstruujme dotaz, na kterém budeme demonstrovat přirozené spojení těchto dvou tabulek:

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE
1	Jan	Novák	15.10.1975	1
2	Petr	Nový	1.4.1978	2
3	Jan	Nováček	6.9.1965	3
4	David	Vokurka	5.12.1973	4

Tabulka: FUNKCE

ID	NAZEV	PLAT
1	uklízeč	15000
2	programátor	21500
3	manažer	17500
4	ředitel	28000

Dotaz:

```
SELECT * FROM ZAMESTNANEC, FUNKCE
WHERE ID_FUNKCE=ID;
```

Kde **ID_FUNKCE** je název sloupce – cizího klíče z původní tabulky **ZAMESTNANEC** a název **ID** je primární klíč v tabulce **FUNKCE**. V praxi ovšem bývá zvykem, i když názvy sloupců v tabulkách spolu nekolidují, vždy uvádět název tabulky, ze které sloupec pochází, tedy:

```
SELECT * FROM ZAMESTNANEC, FUNKCE
WHERE ZAMESTNANEC.ID_FUNKCE=FUNKCE.ID;
```

Výsledek tohoto dotazu vypadá následovně:

CISLO	JMENO	PRIJMENI	DAT_NAROZ	ID_FUNKCE	ID	NAZEV	PLAT
1	Jan	Novák	15.10.1975	1	1	uklízeč	15000
2	Petr	Nový	1.4.1978	2	2	programátor	21500
3	Jan	Nováček	6.9.1965	3	3	manažer	17500
4	David	Vokurka	5.12.1973	4	4	ředitel	28000

Kdybychom nyní chtěli dotaz: *kdo pracuje (jméno, příjmení) ve firmě jako programátor*, dotaz by vypadal následovně:

```
SELECT JMENO, PRIJMENI
FROM ZAMESTNANEC, FUNKCE
WHERE ZAMESTNANEC.ID_FUNKCE=FUNKCE.ID
AND FUNKCE.NAZEV='programátor';
```

Výsledek:

```
JMENO PRIJMENI
Petr  Nový
```

Formulujme další dotazy nad tabulkami **ZAMESTNANEC** a **FUNKCE**:

- *Jakou funkci vykonává pan Nováček?*
```
SELECT NAZEV
FROM ZAMESTNANEC, FUNKCE
WHERE ZAMESTNANEC.ID_FUNKCE=FUNKCE.ID
AND PRIJMENI='Nováček';
```
- *Jaký plat pobírají v naši firmě lidé starší 30 let?*
```
SELECT PLAT
FROM ZAMESTNANEC, FUNKCE
WHERE ZAMESTNANEC.ID_FUNKCE=FUNKCE.ID
AND DAT_NAROZ<1.1.1976;
```
- *Seznam jmen, příjmení a funkcí, které vykonávají všichni zaměstnanci ve firmě?*
```
SELECT JMENO, PRIJMENI, NAZEV
FROM ZAMESTNANEC, FUNKCE
WHERE ZAMESTNANEC.ID_FUNKCE=FUNKCE.ID;
```
- *Seznam příjmení, datumů narození a výše platů všech zaměstnanců ve firmě:*
```
SELECT PRIJMENI, DAT_NAROZ, PLAT
FROM ZAMESTNANEC, FUNKCE
WHERE ZAMESTNANEC.ID_FUNKCE=FUNKCE.ID;
```
- *Seznam jmen, příjmení a funkcí všech zaměstnanců, kteří mají plat vyšší než 20000Kč:*
```
SELECT JMENO, PRIJMENI, NAZEV
FROM ZAMESTNANEC, FUNKCE
WHERE ZAMESTNANEC.ID_FUNKCE=FUNKCE.ID
AND PLAT>20000;
```
- *Seznam příjmení a platů všech uklízečů ve firmě:*
```
SELECT PRIJMENI, PLAT
FROM ZAMESTNANEC, FUNKCE
WHERE ZAMESTNANEC.ID_FUNKCE=FUNKCE.ID
AND NAZEV=`uklízeč`;
```
- *Seznam jmen a příjmení všech zaměstnanců, kterým není ještě 30 let nebo mají plat nižší než 18000 Kč:*
```
SELECT JMENO, PRIJMENI
FROM ZAMESTNANEC, FUNKCE
WHERE ZAMESTNANEC.ID_FUNKCE=FUNKCE.ID
AND (DAT_NAROZ>1.1.1976 OR PLAT<18000);
```

Rozšiřme naše tabulkové schéma o další tabulku pro pracoviště:

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	ID_PRACOVISTE
1	Jan	Novák	15.10.1975	1	11
2	Petr	Nový	1.4.1978	2	21
3	Jan	Nováček	6.9.1965	3	31
4	David	Vokurka	5.12.1973	4	41

Tabulka: POVOLANI

ID	FUNKCE	PLAT
1	uklízeč	15000
2	programátor	21500
3	manažer	17500
4	ředitel	28000

Tabulka: PRACOVISTE

ID	ODDELENI
11	chodba
21	studovna
31	kancelář
41	ředitelna

Formulujme dotaz: *Na jakých pracovištích pracují manažeři?*

```
SELECT ODDELENI
FROM ZAMESTNANEC, POVOLANI, PRACOVISTE
WHERE ZAMESTNANEC.ID_FUNKCE=POVOLANI.ID
AND ZAMESTNANEC.ID_PRACOVISTE=PRACOVISTE.ID
AND FUNKCE='manažer';
```

Modifikátory

Modifikátory si lze představit jako speciální přepínače, kterými můžeme nějakým způsobem upřesnit výsledek dotazu. Modifikátory se zapisují v první části příkazu **SELECT** následovně:

```
SELECT [MODIFIKATOR] jména_sloupců
```

V relační algebře, byla vždy výsledná relace, které vznikla aplikací nějaké operace relační algebry na relaci, zbavena duplicit. To znamená, že ve výsledku se nám nikdy neobjevily dva stejné řádky. V případě jazyka SQL to ale neplatí a pokud chceme vynutit, aby veškeré duplicity byly ve výsledku odstraněny, musíme zadat modifikátor **DISTINCT**. Například:

```
SELECT DISTINCT JMENO
FROM ZAMESTNANEC;
```

Dalším často používaným modifikátorem je **TOP**. Někdy dotaz vrátí spoustu řádků, ale nás zajímá třeba jen prvních deset záznamů. Pomocí modifikátoru **TOP** a čísla **N** určíme,

kolik **N** prvních záznamů se má do výsledku zahrnout. Tento modifikátor má většinou smysl použít spolu se tříděním výsledku (použití **ORDER BY**). Příklad: *3 služebně nejstarší zaměstnanci ve firmě (tj. ti, kteří pracují pro naši firmu nejdéle)*:

```
SELECT TOP 3 JMENO, PRIJMENI
FROM ZAMESTNANEC
ORDER BY SMLOUVA_OD;
```

Skládání dotazů

Podobně, jako tomu bylo v relační algebře, i zde máme možnost konstruovat dotazy s využitím základních množinových konstrukcí, kterými jsou sjednocení, průnik a rozdíl. Pokud tedy zajistíme, že nám 2 dotazy vrací kompatibilní výsledek (=vrací stejné sloupce), pak můžeme výsledky těchto dotazů sjednocovat, pronikat a odečítat. Jednotlivé množinové operace realizujeme pomocí následujících příkazů:

- Sjednocení: **UNION**
- Průnik: **INTERSECT**
- Rozdíl: **EXCEPT**

Syntaxe je následující:

```
SELECT …
FROM ….
WHERE …..
[UNION|INTERSECT|EXCEPT]
SELECT …
FROM …
WHERE …
```

Každý z uvedených dotazů se vyhodnotí zvlášť a po té se na výsledky aplikuje příslušná množinová operace a výsledek jde na výstup. Ukažme si následující příklad na relačním schématu tabulek **KNIHA**, **AUTOR** a **AUTORSTVI**:

Tabulka: KNIHA

ID	NÁZEV	JAZYK
11	Oko	český
12	Bludiště	německý
13	Domek	český
14	Sauna a bazén	anglický

Tabulka: AUTORSTVÍ

ID_KNIHA	ID_AUTOR
11	1
11	5
14	3
12	4
13	4

Tabulka: AUTOR

ID	JMÉNO	PŘÍJMENÍ	DATUM_NAROZENÍ
1	Jan	Novák	15.10.1975
2	Petr	Nový	1.4.1978
3	Jan	Nováček	6.9.1965
4	David	Vokurka	5.12.1973
5	Vít	Mrkvička	30.1.1970

A mějme následující dotaz: *Jaké knihy (zajímají nás názvy) napsali autoři Jan Novák a Vít Mrkvička jako spoluautoři?* Nejprve definujme první dotaz: *jaké knihy napsal Jan Novák*:

```
SELECT NAZEV
FROM KNIHA, AUTORSTVI, AUTOR
WHERE KNIHA.ID=AUTORSTVI.ID_KNIHA
AND AUTORSTVI.ID_AUTOR=AUTOR.ID
AND JMENO='Jan' AND PRIJMENI='Novák';
```

Druhý dotaz, *jaké knihy napsal Vít Mrkvička*, bude vypadat velmi podobně:

```
SELECT NAZEV
FROM KNIHA, AUTORSTVI, AUTOR
WHERE KNIHA.ID=AUTORSTVI.ID_KNIHA
AND AUTORSTVI.ID_AUTOR=AUTOR.ID
AND JMENO='Vít' AND PRIJMENI='Mrkvička';
```

A výsledný původní dotaz s využitím průniku je následující:

```
SELECT NAZEV
FROM KNIHA, AUTORSTVI, AUTOR
WHERE KNIHA.ID=AUTORSTVI.ID_KNIHA
AND AUTORSTVI.ID_AUTOR=AUTOR.ID
AND JMENO='Jan' AND PRIJMENI='Novák'
INTERSECT
SELECT NAZEV
FROM KNIHA, AUTORSTVI, AUTOR
WHERE KNIHA.ID=AUTORSTVI.ID_KNIHA
AND AUTORSTVI.ID_AUTOR=AUTOR.ID
AND JMENO='Vít' AND PRIJMENI='Mrkvička';
```

Dotazy s agregací

Poslední částí, kterou se budeme v rámci jazyka SQL zabývat a která navíc nemá svůj „protějšek“ v relační algebře, jsou dotazy s použitím agregačních funkcí. Někdy kromě konkrétních údajů z databáze potřebujeme získat údaje souhrnné, které budou použitelné například pro statistické zpracování. Právě pomocí agregace jsme schopni z databáze získat následující typy údajů:

- Počty záznamů vyhovující dané podmínce (Např. *kolik zaměstnanců bere plat více jak 20000 Kč?*)
- Aritmetický průměr hodnot sloupců z dané skupiny záznamů nebo celé tabulky
- Součet hodnot sloupců z dané skupiny záznamů nebo celé tabulky
- Minimální hodnota sloupce z dané skupiny záznamů nebo celé tabulky
- Maximální hodnota sloupce z dané skupiny záznamů nebo celé tabulky

Abychom mohli využívat agregační funkce, musíme si rozšířit stávající syntaxi příkazu **SELECT**:

```
SELECT jména_sloupců/agregační_funkce(sloupec)
FROM jména_tabulek
WHERE podmínka
GROUP BY výraz_pro_seskupení
ORDER BY podmínka_řazení
```

Seskupení

Než budeme pokračovat dále, je potřeba se zastavit a vysvětlit si, co to je seskupení. Seskupení řádků se provádí podle hodnot vybraných sloupců. Můžeme seskupovat (agregovat) podle jednoho sloupce, nebo obecně podle libovolných sloupců. Seskupení spočívá v tom, že když se při vyhodnocení dotazu objeví dva řádky, ve který je hodnota sloupce, podle kterého seskupujeme, shodná, databázový systém bude na tyto dva řádky pohlížet jako na jeden řádek a na takovou skupinu řádků můžeme aplikovat agregační funkci. Vezměme si náš příklad tabulek **KNIHA**, **AUTORSTVI** a **AUTOR** a zaměřme se na tabulku **AUTORSTVI**, v níž provedeme seskupování podle sloupce **ID_KNIHA**. Nechť tabulka **AUTORSTVI** má následující obsah:

Tabulka: AUTORSTVÍ

ID_KNIHA	ID_AUTOR
11	1
11	5
14	3
12	4
13	4
12	1
11	3

Jak bude vypadat seskupení podle **ID_KNIHA**? Všimněte si, že jedinečné hodnoty v tomto sloupci jsou celkem čtyři: **11**, **12**, **13** a **14**. Přičemž **11** se opakuje 3x, **12** se

opakuje 2x, **13** je jednou a **14** je také jednou. Jako hodnota sloupce **ID_AUTOR** bude poslední použitá v rámci skupiny záznamů se stejným **ID_KNIHA**. Pro seskupování dle hodnoty určitého sloupce se používá klauzule **GROUP BY**:

```
SELECT * FROM AUTORSTVI
GROUP BY ID_KNIHA;
```

nám dá tento výsledek:

ID_KNIHA	ID_AUTOR
11	3
14	3
12	1
13	4

COUNT(*) – počet výskytů

Jenom seskupování záznamů samo o sobě by nám bylo celkem k ničemu. Seskupíme-li tabulku **AUTORSTVI** podle **ID_KNIHA**, pak je vhodné na zbývající sloupce (v našem případě **ID_AUTOR**) aplikovat nějakou funkci, má-li to nějaký smysl. Zde bychom se mohli například zeptat, kolik autorů napsalo každou knihu. Pokud tedy chceme získat údaj o počtu daných záznamů v rámci skupiny agregovaných záznamů, použijeme funkci **COUNT(název_sloupce)**.

```
SELECT ID_KNIHA, COUNT(ID_AUTOR)
FROM AUTORSTVI
GROUP BY ID_KNIHA;
```

Výsledek podle očekávání:

ID_KNIHA	COUNT(ID_AUTOR)
11	3
14	1
12	2
13	1

V původní tabulce byla hodnota v **ID_KNIHA** **11** celkem třikrát, hodnota **14** jednou, **12** dvakrát a **13** jednou. Odtud tedy hodnoty ve výsledku v sloupci nazvaném **COUNT(ID_AUTOR)**. Pokud bychom chtěli ve výsledku nemít jen **ID** knihy, ale její celý název, použijeme přirozené spojení tabulek a celý SQL dotaz bude vypadat následovně:

```
SELECT NAZEV, COUNT(ID_AUTOR)
FROM KNIHA, AUTORSTVI
WHERE KNIHA.ID = AUTORSTVI.ID_KNIHA
GROUP BY ID_KNIHA;
```

Výsledek dotazu:

NAZEV	COUNT(ID_AUTOR)
Oko	3
Sauna a bazén	1
Bludiště	2
Domek	1

Formulujme dotazy nad tabulkou **BYT** s využitím agregační funkce **COUNT** (všimněte si, že u některých dotazů se nepoužívá klauzule **GROUP BY** – není totiž potřeba):

Tabulka: BYT

CISLO	ADR_ULICE	ADR_CISLO	ADR_MESTO	ADR_PSC	PODLAZI	BALKON	SKLEP
1	Novákova	123	Brno	61500	2	A	N
2	Jirkova	2	Praha	11400	1	A	A
3	Tulíkova	900	Zlín	57890	7	N	N
4	Prackova	7	Znojmo	52109	3	A	A
5	Holíkova	1	Brno	62100	2	N	A
6	U Boba	100	Zlín	57880	6	N	N
7	Krtkova	88	Praha	11100	-1	N	N

- *Kolik bytů je na území města Brna?*
  ```
  SELECT COUNT(*) FROM BYT WHERE ADR_MESTO="Brno";
  ```
- *Kolik bytů má balkon?*
  ```
  SELECT COUN(T(*) FROM BYT WHERE BALKON="A";
  ```
- *Kolik bytů leží alespoň ve 2. podlaží?*
  ```
  SELECT COUNT(*) FROM BYT WHERE PODLAZI>=2;
  ```
- *Kolik bytů je v každém městě?*
  ```
  SELECT ADR_MESTO, COUNT(*)
  FROM BYT
  GROUP BY ADR_MESTO;
  ```
- *Kolik bytů v prvním patře je v každém městě?*
  ```
  SELECT ADR_MESTO, COUNT(*)
  FROM BYT
  WHERE PODLAZI=1
  GROUP BY ADR_MESTO;
  ```

Formulujme další dotazy nad relačním schématem **KNIHA**, **AUTOR**, **AUTORSTVI** s využitím agregační funkce **COUNT**:

Tabulka: KNIHA

ID	NÁZEV	JAZYK
11	Oko	český
12	Bludiště	německý
13	Domek	český
14	Sauna a bazén	anglický

Tabulka: AUTORSTVÍ

ID_KNIHA	ID_AUTOR
11	1
11	5
14	3
12	4
13	4

Tabulka: AUTOR

ID	JMÉNO	PŘÍJMENÍ	DATUM_NAROZENÍ
1	Jan	Novák	15.10.1975
2	Petr	Nový	1.4.1978
3	Jan	Nováček	6.9.1965
4	David	Vokurka	5.12.1973
5	Vít	Mrkvička	30.1.1970

- *Kolik českých knih bylo napsáno?*
  ```
  SELECT COUNT(*) FROM KNIHA WHERE JAZYK="český";
  ```
- *Kolik autorů napsalo knihu Oko?*
  ```
  SELECT COUNT(*)
  FROM KNIHA, AUTORSTVI
  WHERE KNIHA.ID=AUTORSTVI.ID_KNIHA
  AND NAZEV="Oko";
  ```
- *Kolik knih napsal Jan Novák?*
  ```
  SELECT COUNT(*)
  FROM AUTOR, AUTORSTVI
  WHERE AUTOR.ID=AUTORSTVI.ID_AUTOR
  AND JMENO="Jan" AND PRIJMENI="Novák";
  ```
- *Kolik knih napsali jednotliví autoři?*
  ```
  SELECT JMENO, PRIJMENI, COUNT(ID_KNIHA)
  FROM KNIHA, AUTOR, AUTORSTVI
  WHERE KNIHA.ID=AUTORSTVI.ID_KNIHA
    AND AUTORSTVI.ID_AUTOR=AUTOR.ID
  GROUP BY JMENO, PRIJMENI;
  ```

SUM(sloupec) – součet, AVG(sloupec) – průměr

Funkce **SUM** vypočítá součet hodnot určeného sloupce v rámci agregované skupiny záznamů, funkce **AVG** vypočítá aritmetický průměr. Argumentem funkcí **SUM** a **AVG** může být název konkrétního sloupce, nebo také libovolný matematický výraz.

Tabulka: BYT

CISLO	ADR_ULICE	ADR_CISLO	ADR_MESTO	ADR_PSC	PODLAZI	BALKON	SKLEP
1	Novákova	123	Brno	61500	2	A	N
2	Jirkova	2	Praha	11400	1	A	A
3	Tulíkova	900	Zlín	57890	7	N	N
4	Prackova	7	Znojmo	52109	3	A	A
5	Holíkova	1	Brno	62100	2	N	A
6	U Boba	100	Zlín	57880	6	N	N
7	Krtkova	88	Praha	11100	−1	N	N

- *Jaké je průměrné podlaží bytů ve všech městech?*
  ```
  SELECT ADR_MESTO, AVG(PODLAZI)
  FROM BYT
  GROUP BY ADR_MESTO;
  ```
- *Jaký je součet čísel (ADR_CISLO) v jednotlivých ulicích města Brna?*
  ```
  SELECT ADR_ULICE, ADR_MESTO, SUM(ADR_CISLO)
  FROM BYT
  WHERE ADR_MESTO="Brno"
  GROUP BY ADR_ULICE;
  ```

Mějme následující tabulkové schéma pro evidenci deskových her (tabulka **DESKOVA_HRA**), jejich hráčů (tabulka **HRAC**) a výsledků těchto her (tabulka **ZAPASY**):

Tabulka: HRAC

ID	JMENO	PRIJMENI	DAT_NAROZ
1	Jan	Novák	15.10.1975
2	Petr	Nový	1.4.1978
3	Jan	Nováček	6.9.1965
4	David	Vokurka	5.12.1973
5	Vít	Mrkvička	30.1.1970

Tabulka: DESKOVA_HRA

ID	NAZEV
20	Osadníci z Katanu
21	Carcasonne
22	Citadela
23	Alhambra

Tabulka: ZAPASY

ID	ID_HRA	ID_HRAC1	ID_HRAC2	BODY_HRAC1	BODY_HRAC2
101	20	1	3	10	7
102	20	2	3	10	8
103	23	4	1	87	47
104	21	2	3	156	137
105	22	3	1	28	24
106	21	3	4	212	197
107	21	1	5	131	97

Formulujme dotazy nad uvedeným relačním schématem a s využitím agregačních funkcí **SUM** a **AVG**:

- *Jaký je průměrný bodový zisk hráčů ve hře Carcasonne?*
```
SELECT AVG((BODY_HRAC1+BODY_HRAC2)/2)
FROM ZAPASY, HRA
WHERE ZAPASY.ID_HRA=HRA.ID
AND NAZEV="Carcasonne";
```
- *Kolik bodů celkem uhráli všichni hráči hry Osadníci z Katanu?*
```
SELECT SUM(BODY_HRAC1+BODY_HRAC2)
FROM ZAPASY, HRA
WHERE ZAPASY.ID_HRA=HRA.ID
AND NAZEV="Osadníci z Katanu";
```
- *Jaký je průměrný bodový zisk hráčů v jednotlivých odehraných hrách?*
```
SELECT NAZEV, AVG((BODY_HRAC1+BODY_HRAC2)/2)
FROM ZAPASY, HRA
WHERE ZAPASY.ID_HRA=HRA.ID
GROUP BY ID_HRA;
```

MIN(sloupec) – minimální hodnota, MAX(sloupec) – maximální hodnota

Poslední dvě agregační funkce **MIN** a **MAX** vrací minimální, resp. maximální hodnotu konkrétního sloupce. Použití je velmi podobné jako u funkcí **SUM** a **AVG**.

Tabulka: BYT

CISLO	ADR_ULICE	ADR_CISLO	ADR_MESTO	ADR_PSC	PODLAZI	BALKON	SKLEP
1	Novákova	123	Brno	61500	2	A	N
2	Jirkova	2	Praha	11400	1	A	A
3	Tulíkova	900	Zlín	57890	7	N	N
4	Prackova	7	Znojmo	52109	3	A	A
5	Holíkova	1	Brno	62100	2	N	A
6	U Boba	100	Zlín	57880	6	N	N
7	Krtkova	88	Praha	11100	-1	N	N

- *Jaká jsou minimální a maximální podlaží bytů s balkony v jednotlivých městech?*
```
SELECT ADR_MESTO, MIN(PODLAZI), MAX(PODLAZI)
FROM BYT
WHERE BALKON="A"
GROUP BY ADR_MESTO;
```

Shrnutí

- **Jazyk SQL je standardem pro relační databázové systémy, většina relačních databázových systémů jej dodržují, v menší míře pak doplňují o vlastní další rozšíření.**
- **Příkazem pro vytvoření tabulky je CREATE TABLE, u každého sloupce je potřeba definovat jeho datový typ a případně integritní omezení.**
- **Tabulka, která je odkazovaná z jiné tabulky, musí být vytvořena dříve, než tabulka odkazující.**
- **Základními příkazy pro manipulaci se záznamy jsou INSERT (vkládání), UPDATE (aktualizace) a DELETE (mazání).**

- **Základním a nejsložitějším příkazem pro formulaci dotazů nad daty je SELECT.**
- **Kromě základních operací jako jsou projekce, restrikce, přirozené spojení a množinové operace (které jsou známy z relační algebry) máme k dispozici také agregační funkce.**
- **Agregačními funkcemi jsou COUNT (počet výskytů), SUM (součet hodnot), AVG (průměr hodnot), MIN (nejnižší hodnota) a MAX (nejvyšší hodnota).**

Otázky a úkoly

- Nad uvedenými tabulkami **ZAMESTNANEC**, **POVOLANI** a **PRACOVISTE** realizujte dotazy:

Tabulka: ZAMESTNANEC

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAROZ	ID_FUNKCE	ID_PRACOVISTE
1	Jan	Novák	15.10.1975	1	11
2	Petr	Nový	1.4.1978	2	21
3	Jan	Nováček	6.9.1965	3	31
4	David	Vokurka	5.12.1973	4	41

Tabulka: POVOLANI

ID	FUNKCE	PLAT
1	uklízeč	15000
2	programátor	21500
3	manažer	17500
4	ředitel	28000

Tabulka: PRACOVISTE

ID	ODDELENI
11	chodba
21	studovna
31	kancelář
41	ředitelna

- *Jakou funkci vykonává David Vokurka?*
- *Jaký plat pobírá programátor?*
- *Všichni zaměstnanci (jméno, příjmení) starší 20ti let*
- *Kteří zaměstnanci (jméno, příjmení) pracují na chodbě?*
- *Na kterých pracovištích (název) pracují programátoři?*

- Nad uvedenými tabulkami **KNIHA**, **AUTOR** a **AUTORSTVI** realizujte dotazy

Tabulka: KNIHA

ID	NÁZEV	JAZYK
11	Oko	český
12	Bludiště	německý
13	Domek	český
14	Sauna a bazén	anglický

Tabulka: AUTORSTVÍ

ID_KNIHA	ID_AUTOR
11	1
11	5
14	3
12	4
13	4

Tabulka: AUTOR

ID	JMÉNO	PŘÍJMENÍ	DATUM_NAROZENÍ
1	Jan	Novák	15.10.1975
2	Petr	Nový	1.4.1978
3	Jan	Nováček	6.9.1965
4	David	Vokurka	5.12.1973
5	Vít	Mrkvička	30.1.1970

- *Jaké knihy (název) napsal Vít Mrkvička?*
- *Kdo napsal (jméno, příjmení) knihu „Bludiště“?*
- *V jakém jazyce byla napsána kniha „Sauna a bazén“?*
- *Jaké knihy (název) napsali Petr Nový a Jan Nováček jako spoluautoři?*
- *Kdy se narodil autor/autoři knihy „Domek“?*
- *V jakých jazycích napsal knihy Jan Novák?*
- *Jaké knihy (název) nenapsali Vít Mrkvička a David Vokurka jako spoluautoři?*

- Nad uvedenými tabulkami **SPORTOVEC**, **DISCIPLINA** a **VYKON** realizujte dotazy

Tabulka: SPORTOVEC

ID	JMENO	PRIJMENI	DAT_NAROZ
1	Jan	Novák	15.10.1975
2	Petr	Nový	1.4.1978
3	Jan	Nováček	6.9.1965
4	David	Vokurka	5.12.1973
5	Vít	Mrkvička	30.1.1970

Tabulka: DISCIPLINA

ID	NAZEV
10	Běh na 12min
11	Skok do dálky
12	Hod oštěpem
13	Hod kriketem

Tabulka: VYKON

ID	ID_DISCIPLINA	ID_CLOVEK	VZDALENOST
201	10	1	2450
202	10	3	2660
203	11	1	2,05
204	11	4	1,98
205	11	2	2,01
206	11	1	2,08
207	12	5	97
208	13	4	63
209	13	1	47
210	13	2	69

- *Jaký byl nejlepší skok do dálky Jan Nováka?*
- *Kolik dohromady hodili kriketem všichni soutěžící?*
- *Jaká je výsledková listina (JMENO, PRIJMENI, VZDALENOST) všech sportovců v běhu na 12 minut?*

Rejstřík

Literatura

1. Pokorný, J. *Dotazovací jazyky*. Skripta UK, Karolinum, Praha. 2002. 255s.
2. Skřivan, J. *Databáze a SQL*. Zoner s.r.o, Brno. 2000. http://www.interval.cz
3. Šešera, L. a kol. *Datové modelování v příkladech*. Grada, Praha. 2001. 152s. ISBN 80-247-0049-2.
4. Šimůnek, M. *SQL – kompletní kapesní průvodce*. Grada, Praha. 1999. 248s. ISBN 80-7169-692-7.