5. RELAČNÍ DATOVÝ MO DEL

Čas ke studiu kapitoly: 4 hodiny návrh databáze, 6 hodin dotazovací jazyky

Cíl Po prostudování celé kapitoly budete umět

- definovat relaci, relační schéma a schéma relační databáze, rozumět definicím a na příkladech vysvětlit jednotlivé pojmy,
- definovat pojem funkční závislost mezi podmnožinami atributů, rozumět mu a bezpečně rozeznávat funkční závislosti v praktických úlohách,
- definovat pomocí normálních forem i vysvětlit na příkladech, co je dobře navržené relační schéma a na základě toho navrhnout optimální strukturu relační databáze,
- vyhledávat informace v relační databázi pomocí procedurálního jazyka – relační algebry i pomocí deskriptivního jazyka SQL.

Výklad

5.1. Relační schéma, relace

❑ Základní pojmy relačního datového modelu

Relační datový model popisuje databázi, její strukturu a vlastnosti i manipulaci s daty pomocí matematických pojmů, především z teorie množin a matematické logiky. Zavádí přesné definice ke všem pojmům, dosud popisovaným jen slovně. Definuje vlastnosti správně navržené databáze, popisuje chyby ve strukturách tabulek databáze. Definuje jazyky, pomocí nichž se vyhledává informace z databáze. Díky této formalizaci je možno používat přesná pravidla jak při vytváření databáze, tak při manipulaci s ní.

Základní pojmy, odpovídající nám již známým pojmům typ entity a množina entit, jsou popsány následujícími definicemi.

Definice:

Relační schéma R je výraz tvaru **R**(A, f), kde **R** je jméno schématu, A = {A_1, A_2,..., A_n} je konečná množina jmen atributů, f je zobrazení přiřazující každému jménu atributu A_i neprázdnou množinu (obor hodnot atributu), kterou nazýváme doménou atributu D_i, tedy $f(A_i) = D_i$.

Definice:

Relace R s relačním schématem **R** je konečná podmnožina kartézského součinu domén Di, příslušejících jednotlivým atributům Ai, tedy

$$R \subset D_1 \times D_2 \times ... \times D_n.$$

Číslo n nazýváme stupněm relace, o relaci R říkáme, že je typu **R.**

Relační schéma definuje nám již známý pojem typ entity (zobrazený jako název a hlavička datové tabulky), relace pak odpovídá množině entit (zobrazené jako obsah tabulky, množina jejích řádků), které jsme dosud používali v konceptuálním modelu. Tyto přesné definice jsou nutné k definování jazyka pro manipulace s databází a pro definici pravidel, které musí splňovat správně navržená databáze bez redundancí.

Již víme, že relaci je výhodné znázorňovat jako dvourozměrnou tabulku, kde každý řádek odpovídá jedné entitě, každý sloupec jednomu atributu. Jména atributů musí být v rámci relace navzájem různá. Často se tedy v relačním modelu místo pojmu relace používá pojem tabulka. Nejtěžší k pochopení je „podmnožina kartézského součinu domén" z definice relace.

Protože relace odráží část reálné skutečnosti odpovídající evidovaným datům, není prvkem relace jakýkoli prvek z D_1 x D_2 x ... x D_n. Do relace patří jen prvky vyhovující zadaným podmínkám vyjádřeným jako integritní omezení (tentokrát IO na objekty = které objekty z reality mají být do relace zahrnuty). Podívejme se na následující příklady.

Příklad:

Jméno relačního schématu: **`Učitel`**

Jména atributů: `ČU,jméno,funkce,plat`

Domény: D1 je množina řetězců tvaru Un, kde n je přirozené číslo; prvky množiny jsou osobní čísla učitelů;
D2 je množina možných jmen učitelů;
D3 množina textů označujících funkční zařazení učitelů v zaměstnání;
D4 množina nezáporných celých čísel<10000 *(IO definované jako interval)*

Zobrazení: `f(ČU)=D1, f(jméno)=D2, ...`

Relační schéma: **`Učitel`**`(ČU,jméno,funkce,plat)`

Kartézský součin domén: D1 x D2 x D3 x D4 *je množina* ***všech možných*** *čtveřic takových, že první prvek čtveřice je hodnota z* D1, …, *poslední je hodnota z* D4.

Relace:

```
Učitel={(U12,Čáp Ladislav,docent,24000),
        (U27,Holub Stanislav,asistent,23000),
        (U39,Kos Zbyněk,asistent,13000),
        (U43,Orel David,docent,24000), …}
```

přičemž do relace patří jen ty čtveřice, které odpovídají skutečným hodnotám 4 atributů odpovídajících evidovaným učitelům.

Do relace nepatří např. čtveřice `(U27,Holub Stanislav,docent,3000),` *neboť nikdo takový neexistuje, případně nepatří k našim evidovaným učitelům.*

Zobrazení relace tabulkou

Učitel

ČU	**jméno**	**fce**	**plat**
U12	Čáp Ladislav	docent	24000
U27	Holub Stanislav	asistent	23000
U39	Kos Zbyněk	asistent	13000
U43	Orel David	docent	24000
…			

♦

Příklad:

Pro schéma **Učitel** (ČU, jméno, fce, plat) *je relace* Učitel ⊂ ČU x jméno x fce x plat

Kartézský součin obsahuje všechny kombinace hodnot všech domén. Z nich jen některé patří do relace = podmnožiny kartézského součinu. Jsou to ty prvky (řádky), které odrážejí evidovanou skutečnost formulovanou integritním omezením o prvcích relace. To znamená ty, které přísluší skutečným učitelům naší evidence (Černohorský je docentem s osobním číslem U66 a platem 20000, Vondrák profesorem ... v evidenci školy).

Učitel

ČU	jméno	fce	plat
U1	Anděl Jiří	asistent	10000
U1	Anděl Jiří	asistent	20000
...			
U128	Anděl Jiří	profesor	30000
...			
U66	Černohorský Jindřich	asistent	10000
U66	Černohorský Jindřich	asistent	20000
U66	Černohorský Jindřich	asistent	30000
U66	Černohorský Jindřich	docent	20000
…			
U888	Vondrák Ivo	asistent	10000
…			
U888	Vondrák Ivo	profesor	30000

♦

Poznámka: Některé učebnice a zvláště manuály některých SŘBD používají pojem relace pro označení vazby mezi entitami, ne pro relaci (tabulku) výše definovanou. Jde o **chybný překlad** původního slova „Relationship“ odpovídajícího relační vazbě. I když bychom mohli používat místo vazba relací pojem „relace relací“, z pochopitelných důvodů tento složitý pojem zavádět nebudeme. RDM se nazývá relační podle definice relace ~ tabulky, ne podle relační vazby.

Na rozdíl od matematických relací jsou databázové relace proměnné v čase. Aktualizace databáze, která umožňuje zachytit v databázi změny nastávající v reálném světě, spočívá ve změně aktuálních relací přidáváním, rušením prvků relací nebo změnou hodnot některých atributů.

Z definice relace vyplývají tyto jejich tabulkové vlastnosti

- homogenita sloupců (hodnoty odpovídají příslušné doméně)
- každý údaj (hodnota atributu ve sloupci) je atomickou položkou,
- na pořadí řádků nezáleží (relace je množina)
- na pořadí sloupců nezáleží (pojmenované atributy také tvoří množinu)
- každý řádek tabulky je jednoznačně identifikovatelný hodnotami jednoho nebo několika atributů (primárního klíče).

Definice jednotlivých relací a jejich schémat doplníme o strukturu celé databáze.

Definice:

Schéma relační databáze je konečná množina relačních schémat {$\mathbf{R_1}(A_1,f_1)$, $\mathbf{R_2}(A_2,f_2)$, **...**, $\mathbf{R_m}(A_m,f_m)$}.

Definice:

Relační databází v daném časovém okamžiku je konečná množina relací R_1, R_2, ..., R_m, tzv. aktuálních relací, kde R_i je typu $\mathbf{R_i}$.

Formulace „v daném časovém okamžiku“ zohledňuje, že obsah databáze je proměnný v čase.

- **Realizace vazeb v relačním modelu**

Charakteristickou vlastností relačního modelu je realizace vazeb opět pomocí relací ~ tabulek. Již v konceptuálním modelu jsme některé entity nazývali vazebními, pokud reprezentovaly vazbu mezi dvěma či více entitami. Vzpomeňme příklad binární vazby mezi Firmou a Výrobkem, nazvanou VÝROBA (Firma, Výrobek) nebo n-ární vazbu ROZVRH (Třída, Předmět, Učitel, Místnost, Hodina).

Relační model obecně **realizuje vazbu mezi relacemi pomocí relace.** V ní každou entitu do vazby vstupující zastupuje její primární klíč.

V některých případech (vazby 1:1, 1:M) však není nutné použít samostatnou vazební tabulku, ale stačí doplnění tabulky na straně M o další atribut - **cizí klíč**, jak si ukážeme na příkladech.

Příklad:

Binární vazbu s kardinalitou M:N relací Učitel (ČU, jméno, fce, plat*) a* Předmět (ČP, název), *nazvanou* UČÍ (Předmět, Učitel) *realizujeme pomocí vazební relace s klíči obou relací*

Učí (ČU, ČP)

Pokud vazba má vlastní atributy, týkající se ne objektů učitel a předmět, ale vazby samotné – například počet hodin, který učí konkrétní učitel v konkrétním předmětu, pak jsou tyto atributy přidány do vazební relace:

Učí (ČU, ČP, hodin)

Když začne učitel učit nový předmět, přidá se do Učí *nový řádek, při změně učitele učícího matematiku se modifikuje v příslušném řádku* ČU*, pokud ze zruší předmět, smažou se řádky s příslušným* ČP.

♦

Příklad:

Binární vazbu 1:M relací Učitel (ČU, jmeno, fce, plat) *a* Katedra (ČK, název_kat) *zapíšeme* ČLEN_KAT (Učitel, Katedra) *a realizujeme pomocí vazební relace s atributy*

Člen_kat (ČU, ČK)

Ovšem při podrobnější analýze zjistíme, že počet řádků této relace je stejný, jako u relace Učitel – každému učiteli odpovídá jeden řádek s číslem katedry. V tomto případě je možné vazební tabulku spojit s tabulkou Učitel*, shodné sloupce* Učitel.ČU *a* Učí.ČU *sloučit, takže vznikne rozšířená tabulka*

Učitel (ČU, jméno, fce, plat, ČK)

V ní atribut ČK, který realizuje vazbu Učí, nazýváme cizím nebo vazebním klíčem.

♦

Příklad:

Vazba n-ární se realizuje opět pomocí vazební relace. Případ vazby ROZVRH (Třída, Předmět, Učitel, Místnost, Hodina) *by realizovala tabulka*

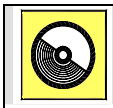
Rozvrh (ČT, ČP, ČU, ČM, Hod)

kde ČT je číslo třídy, ČP číslo předmětu, ČU číslo učitele, ČM číslo místnosti, Hod je hodina v týdnu.

♦

Celkově můžeme konstatovat, že vazba mezi relacemi se obecně vždy dá realizovat pomocí vazební tabulky. V některých případech je možné vazební tabulku sloučit s jednou z tabulek do vazby vstupujících a spojit dva totožné sloupce. Nově připojené sloupce k původní tabulce nazýváme cizí klíče. Tuto technologii s cizími klíči používáme, pokud již při datové analýze zjistíme, že některý typ vazby bude potřebný pro náš informační systém.

Velká výhoda relačního modelu je však také v tom, že i předem nepředpokládané vazby můžeme realizovat bez zásahu do původní struktury relací. Pokud se až později ukáže potřeba evidence nového, nepředpokládaného typu vazby mezi existujícími entitami, stačí definovat novou vazební relaci. Jejími atributy budou primární klíče provázaných relací, případně další atributy vazby. S novou relací se pak pracuje pomocí běžných příkazů práce s relacemi.



CD-ROM

Na CD-ROMu s tímto výukovým textem jsou animované příklady na právě probrané případy realizace vazeb v RMD. Soubory s animacemi jsou pojmenovány:

- Vazba_1k1.exe
- Vazba_1kN.exe
- Vazba_MkN.exe
- Vazba_unarni.exe
- Vazba_ternarni.exe

Shrnutí pojmů 5.1.

Relační schéma, relace. Atribut, doména. Zobrazení množiny atributů do množiny domén. Stupeň relace. Zobrazení relačního schématu a relace pomocí tabulky.

Souvislost relačního schématu a relace s typem entity a entitami.

Schéma relační databáze, relační databáze. Aktuální relace.

Realizace vazeb různých typů v RMD.

Otázky 5.1.

1. Definujte relační schéma a na vlastním příkladě vysvětlete všechny pojmy v definici použité.
2. Definujte relaci a na praktickém příkladu vysvětlete všechny pojmy v definici použité: doména, kartézský součin domén, podmnožina kartézského součinu, integritní omezení pro prvky relace.
3. Definujte schéma relační databáze a na vlastním příkladě popište jednoduché schéma databáze z praxe.
4. Definujte relační databázi a uveďte příklad některé reálné databáze z praxe.
5. Co znamená, že je databáze konzistentní?
6. Co znamená, že je databáze integritní?
7. Jakými způsoby se realizují vazby entit v relačním datovém modelu? Rozlište realizaci vazby dle počtu entit do vazby vstupujících (1, 2, více) a podle kardinality vazby.

Úlohy k řešení 5.1.

1. Projděte si znovu příklady na realizaci vazeb v této kapitole. Ke každému příkladu zobrazte formou tabulek: původní relace, vazební relaci a pokud to jde, sloučení vazební relace s jednou z původních tabulek a vznik cizího klíče.

5.2. Datová analýza schématu relační databáze

- **Datová analýza pomocí relačního datového modelu**

Vytvoření konceptuálního schématu na základě intuitivního návrhu entit, jejich atributů a vazeb mezi entitami není bezpečným způsobem, jak navrhnout správně schéma relační databáze. Ukážeme si, že i entity s malým počtem atributů je možno navrhnout špatně, s redundancemi - pokud neznáme pravidla, která nám umožní rozpoznat, co je dobře nebo špatně navržené schéma relace.

Současně se vznikem teorie relačního modelu dat vznikla také metoda návrhu relačních schémat, a to pomocí nového pojmu - funkčních závislostí.

Podívejme se na úlohu návrhu struktury databáze takto: z reálného světa máme dánu množinu atributů, které chceme rozmístit do jednotlivých schémat relací. Názvy těchto relací (názvy typů entit, názvy tabulek) ani jejich počet předem neznáme. Úkolem je navrhnout schéma databáze bez redundancí.

Příklad:

Potřebujeme malou databázi s evidencí výuky a výsledků zkoušek studentů. Představme si relaci

R (Přednáška, Učitel, Místnost, Hodina, Student, Známka).

Prvkem tohoto schématu je předmět, učitel, který ho učí v uvedené místnosti a uvedenou hodinu, student, který přednášce naslouchá a známka, kterou dostane po vykonání zkoušky z daného předmětu. Klíčem tohoto schématu je (Hodina,Student*).*

Předmět	**Učitel**	**Místnost**	**Hodina**	**Student**	**Známka**
INZ	Šarmanová	NK301	Po9	Novák	2
INZ	Fasuga	E320	Út3	Novák	2
INZ	Šarmanová	NK301	Po9	Kouba	3
INZ	Fasuga	E320	Út3	Kouba	3
OS	Černohorský	E322	Po7	Zíka	1
OS	Černohorský	E322	Po7	Tupý	2
OS	Černohorský	E322	Po7	Redl	2
OS	Černohorský	E322	Po7	Bílý	1
. . .					

Snadno si všimneme následujících nedostatků této struktury a potíží při práci s ní:

- **redundance**, pro každého studenta navštěvujícího přednášku se opakují hodnoty přednáška, učitel, místnost, hodina,
- nebezpečí vzniku **nekonzistence** při modifikacích jako důsledek redundance,
- **anomálie při vkládání** záznamů: nemůžeme vložit učitele, který právě neučí, neboť by nebyly obsazeny klíčové atributy,
- **anomálie při vypouštění** záznamů: přestane-li učitel učit, musíme vypustit prvky - řádky o tomto učiteli, tím ztratíme informaci i o jeho jménu.

Je zřejmě nutné schéma databáze změnit, atributy rozdělit do více relačních schémat, navrhnout několik typů entit a typů vazeb mezi nimi. V následujícím seznamu jsou uvedena různá relační schémata databáze, popisující stejnou realitu. Pro jednoduchost místo atributů píšeme jen jejich počáteční písmena.

Rozklad 1.
RO1 = { R1(P, U), R2(H, M, P), R3(H, U, M), R4(P, S, Z), R5(H, S, M) }
kde

R1 zaznamenává, kdo (který učitel) co (který předmět) učí,
R2 je rozvrh hodin podle předmětů – kdy a kde se předmět učí,
R3 je rozvrh hodin podle učeben,
R4 zaznamenává výsledky zkoušek studentů v jednotlivých předmětech
R5 rozvrh hodin podle studentů

Další možné rozklady si již pojmenujete sami:

RO2 = { R1(P ,U), R2(H, M, P), R3(P, S, Z), R4(H, S, P) }
RO3 = { R1(P, U), R2(H, S, P), R3(P, S, Z), R4(H, S, M) }
RO4 = { R1(H, M, P, U), R2(P, S, Z), R3(H, S, M) }
RO5 = { R1(P, U), R2(H, S, M), R3(P, S, Z), R4(H, M, P) }

Když si všechna schémata přečteme a pojmenujeme, vidíme, že všechna schémata jsou smysluplná. Otázkou tedy je, čím se vlastně od sebe liší a je-li některé z nich lepší než ostatní.

♦

- **Postup při datové analýze v RDM**

Jazykem RDM můžeme formulovat problém takto:

1. Ze **zadání** určíme popis modelované reality a potřebu evidence všech atributů sledovaných objektů {A1, A2, ...}
2. V rámci **datové analýzy** pak
   - Vytvoříme jediné **univerzální schéma** se všemi atributy
     RU (A1, A2, ...)
   - Popíšeme existující vztahy mezi atributy – funkční závislosti, tj. pro nás nové IO:
     F (f1, f2, ...)
   - Pomocí obou množin a pomocí vhodného algoritmu vytvoříme schéma databáze s dobrými vlastnostmi, bez redundancí:
     RU (A1, A2, ...) $\Rightarrow$ **RO** {**R1** (A1, ...), **R2** (Ai, ...), ... }

K tomu si však musíme nejprve vysvětlit, co jsou funkční závislosti a jak se poznají „dobré vlastnosti“ relačních schémat. Potom můžeme popsat postup, jak se k takovému výsledku dostaneme.

Zdrojem informací pro nás bude další typ integritních omezení. Mimo dosud známá IO zavedeme nový pojem - funkční závislosti (FZ). Ty nám pomohou lépe pochopit zdroj možných redundancí a definovat pravidla pro rozpoznání dobře navržených relačních schémat.

Definice:

Nechť **R**({A1, A2, ..., An}, f) je relační schéma, nechť X, Y jsou podmnožiny množiny jmen atributů {A1,A2,...,An}. Řekneme, že **Y je funkčně závislá na X**, píšeme $X \rightarrow Y$, když pro každou možnou aktuální relaci R(A1, A2, ..., An) platí, že mají-li libovolné dva prvky (= dva řádky) relace R stejné hodnoty atributů X, pak mají i stejné hodnoty atributů Y. Je-li $Y \subset X$ říkáme, že závislost $X \rightarrow Y$ je **triviální**.

Z definice plyne, že

- funkční závislosti jsou definovány mezi dvěma podmnožinami atributů v rámci jednoho schématu relace. Jde tedy o vztahy mezi atributy, nikoliv mezi entitami,
- funkční závislost je definována na základě všech možných aktuálních relací, není tedy možné soudit na funkční závislost z vlastností jediné (třeba aktuální) relace; tak můžeme poznat pouze neplatnost funkční závislosti,
- funkční závislosti jsou tvrzení o reálném světě, o významu atributů nebo vztahů mezi entitami (jako každé IO), je nutné je brát v úvahu při návrhu schématu databáze.

FZ slouží jednak při datové analýze k návrhu struktury databáze bez redundancí, jednak později u funkční analýzy ke kontrolám správnosti dat ukládaných do databáze.

Příklad:

Je dána aktuální relace dle relačního schématu z minulého příkladu. Podívejme se podrobněji na obsah tabulky:

Všimneme si vztahu mezi podmnožinami X = {místnost, hodina} a Y = {předmět}. Vždy, když 2 řádky tabulky mají stejné hodnoty atributů X, mají i stejné hodnoty atributů Y.

V tabulce to jsou tyto skupiny řádků: 1+3, 2+4, 5+6+7+8. Slovně to můžeme formulovat takto: v dané místnosti a v danou hodinu v týdnu se učí jediný předmět. Tedy existuje funkční závislost $M\,H \rightarrow$ P. *Tuto FZ známe obecně z reality.*

	P		M H			
	Předmět	**Učitel**	**Místnost**	**Hodina**	**Student**	**Známka**
1	ISZ	Šarmanová	B5	Po9	Novák	2
2	ISZ	Fasuga	D117	Út3	Novák	2
3	ISZ	Šarmanová	B5	Po9	Kouba	3
4	ISZ	Fasuga	D117	Út3	Kouba	3
5	OS	Černohorský	B2	Po7	Zíka	1
6	OS	Černohorský	B2	Po7	Tupý	2
7	OS	Černohorský	B2	Po7	Redl	2
8	OS	Černohorský	B2	Po7	Bílý	1
	. . .					

V uvažované škole platí, že každý předmět přednáší jeden učitel (= zadaná FZ: $P \rightarrow$ U*).*
Celkem v příkladě můžeme určit tuto množinu funkčních závislostí F

$$F = \{ MH \rightarrow P,\ P \rightarrow U,\ HU \rightarrow M,\ PS \rightarrow Z,\ HS \rightarrow M\}$$

Z aktuální relace by se mohlo usuzovat na platnost funkční závislosti $M \rightarrow H$*, ovšem obecně to zřejmě není pravda. Nelze tedy z jedné relace dokázat platnost funkčního vztahu.*
Naopak negativní fakta mohou být zjistitelná, protože tvoří protipříklad: není pravda $PU \rightarrow M$*, protože* TZD *se učí ve dvou posluchárnách v týdnu.*

♦

❑ Hledání funkčních závislostí

Když jsme určili ze zadání (a případnými konzultacemi se zadavatelem) všechny atributy budoucí databáze, musíme určit všechny funkční závislosti mezi nimi.

Teoreticky bychom mohli systematicky procházet všechny vztahy mezi všemi podmnožinami atributů. Při větším počtu atributů by to však trvalo velmi dlouho. Obvykle stačí rozdělit všechny atributy na podmnožiny atributů stejných entit a hledat FZ mezi nimi. Ověření platnosti FZ provádíme pomocí definice.

Příklad:

Najděte všechny FZ pro informační systém ABC soukromého zdravotnického střediska s několika lékaři. Je potřeba evidovat lékaře (osobní údaje a specializaci), pacienty (osobní údaje, pojišťovna), objednané pacienty a uskutečněné návštěvy u lékaře i lékařů u pacientů (datum a čas, diagnóza, cena pro pojišťovnu).

Entity a jejich atributy: Lekar (RC_lek, jmeno_lek, specializace), Pacient (RC_pac, jmeno_pac, adresa, pojistovna), Navsteva (RC_lek, RC_pac, datum, cas, diagnoza, cena).

Pro vyhledání FZ testujeme všechny možnosti v rámci atributů stejné entity:

RC_lek → jmeno_lek *??? ... platí, protože vždy, když je vlevo stejné* RC_lek*, je vpravo stejné jméno (rodné číslo jednoznačně určuje osobu a tedy i její jméno).*

RC_lek → specializace *??? ... obecně neplatí, protože jeden lékař zadaný svým rodným číslem může mít několik specializací; pokud by však pro naši databázi platilo (= bylo v zadání), že se u každého lékaře eviduje jen jediná hlavní specializace, pak FZ platí.*

jmeno_lek → RC_lek, specializace *??? ... neplatí, protože je-li na levé straně 2 x stejné jméno, nemusí jít o stejnou osobu a tedy stejné* RC_lek *a stejnou specializaci (jinak řečeno, 2 lékaři mohou mít stejné jméno).*

specializace → RC_lek, jmeno_lek *??? ... samozřejmě neplatí, protože je-li na levé straně stejná specializace, může být na pravé straně řada různých lékařů s různými RC i jmény.*

RC_lek, RC_pac → datum, cas *??? ... neplatí, protože tentýž pacient může navštívit téhož lékaře mnohokrát, neboli je-li na levé straně v tabulce Navsteva stejné RC_lek, RC_pac, může být na pravé straně několik různých datumů a časů.*

RC_lek, RC_pac, datum, cas → cena *??? ... platí, atributy na levé straně jednoznačně určují jednu návštěvu lékaře a ta má svou jedinou cenu.*

RC_lek, RC_pac, datum, cas → diagnoza *??? ... platí, pokud předpokládáme (= je v zadání), že při jedné návštěvě se eviduje jediná diagnóza; obecně zřejmě neplatí, protože při jedné návštěvě lékaře může být určeno několik diagnóz.*

♦

❑ Pravidla pro odvozování funkčních závislostí

Máme tedy určeny všechny atributy budoucí databáze a určeny všechny platné funkční závislosti mezi nimi. Je-li v zadání mnoho atributů, může být i mnoho funkčních závislostí. Při jejich formulování se může stát, že některé závislosti budou redundandní, že budou jiným způsobem zaznamenávat stejná integritní omezení. To by mohlo komplikovat jak návrh struktury databáze, tak kontroly dat při provozu databáze.

Existují odvozovací pravidla nazývaná Amstrongovými axiomy, která umožňují ze zadaných FZ odvozovat další FZ. Přitom všechny odvozené FZ popisují tytéž skutečnosti (IO), nepřidávají další. Pomocí těchto pravidel a pomocí speciálních algoritmů mohou profesionální analytici při návrhu databáze odstranit všechny redundance ve funkčních závislostech. Při jejich zadávání tedy není třeba hlídat, jestli jsou závislosti zadány optimálně, bez zbytečných redundancí.

❑ Pravidla pro definování dobrých vlastností relačních schémat – normální formy

Již jsme se zmínili o pravidlech, pomocí nichž rozeznáme, zda jsou relační schémata navržena bez redundancí. Pak jsou schémata i bez dalších špatných vlastností z redundance plynoucích – bez anomálií při vkládání a při rušení záznamů tabulky.

Tato pravidla jsou formulována pro relační schéma většinou pomocí funkčních závislostí platných nad atributy schématu. Nazýváme je normálními formami.

❑ První normální forma

Při návrhu relačních schémat jsme dosud používali **atributů atomických**. Pojem atomický atribut není přesně definován, myslí se jím atributy, jejichž domény jsou v jistém smyslu množiny jednoduchých hodnot, čísel, znaků, slov v nějaké abecedě ap., logicky dále nedělitelných.

Definice:

Jestliže relační schéma obsahuje pouze atomické atributy, říkáme, že je normalizovanou relací nebo že je **v první normální formě** (1NF).

Platí: relační model dat pracuje pouze s relacemi v 1NF.

Poznámka: Připomeňme si z kapitoly o konceptuálním schématu, že relaci v 1NF dostaneme z relace s neatomickými atributy buď doplněním opakovaných hodnot u vícehodnotových atributů (pak relace obsahuje redundanci), nebo dekompozicí relace na více relací.

Neatomické atributy převedeme na tabulky s atomickými atributy těmito technikami:

1. atribut opakující se n-krát: n atributů vedle sebe (pozor na formulaci dotazů) nebo nová tabulka s cizím klíčem,
2. atribut opakující se ?-krát: nová tabulka s cizím klíčem,
3. složený atribut: rozklad na několik atomických atributů, odpadne název složeného atributu.

Příklad:

Převedení neatomických atributů do 1NF. Je dána relace

Zaměstnanec (jmeno, plat:multi, dítě:multi, adresa(ulice, obec, PSČ))

1. plat se opakuje 12-krát (leden až prosinec), po úpravě odpadne atribut plat.

jmeno	plat (m1, m2,… ,m12)

⇒

jmeno	m1	m2	…	m12

2. dítě zaměstnance se opakuje proměnný počet-krát, po úpravě se děti umístí do nové tabulky s cizím klíčem.

jmeno	funkce	…	dite

⇒

jmeno	funkce	…

+

jmeno	dite

3. adresa jako skupinový atribut, po úpravě odpadne atribut adresa.

jmeno	adresa (ulice, obec, PSC)

⇒

jmeno	ulice	obec	PSC

♦

- **Druhá normální forma**

Definice:

Relační schéma je **ve druhé normální formě** (2NF), jestliže je v první normální formě a každý sekundární atribut je úplně závislý na každém klíči schématu **R.**

Jinak řečeno: každý sekundární atribut je závislý na klíči, ale není závislý na žádné části klíče. Zřejmě nesplnění 2NF může nastat jen, existuje-li klíč víceatributový – tedy má podklíče. Jednoatributový klíč podklíče nemá.

Příklad:

Je dána relace v 1NF Firma (firma, adresa, zboží, množ). *Je toto schéma ve 2NF?*
Řešení: Platí F = {firma→adresa, firma, zboží → množ}
Klíč původního schématu: {firma, zboží}
Protože platí také firma→adresa*, je* adresa závislá na podklíči
Důsledky: redundance adresy ⇒
když firma přestane dočasně vyrábět musíme vymazat (neexistuje část klíče) a tak nevíme ani to, že existuje a kde sídlí
firmu, která nevyrábí (není zadán úplný klíč), není možno zapsat
Výsledek: schéma není ve 2NF ⇒ *rozklad na 2 tabulky do 2NF: RO = {FirmaN, Výroba}*

Firma

firma	adresa	zboží	množ
AA	XAA	zb1	500
AA	XAA	zb2	200
BB	XBB	zb2	600
BB	XBB	zb3	700
…			

⇒

FirmaN

firma	adresa
AA	XAA
BB	XBB
…	

+

Výroba

firma	zboží	množ
AA	zb1	500
AA	zb2	200
BB	zb2	600
BB	zb3	700
…		

♦

Příklad:

Vezměme opět schéma Učitel Jméno, Katedra, Předmět, Hodin) *s klíčem* (Jméno, Předmět) a *závislostí* {Jméno → Katedra}. *Tato relace je v 1NF, ale není ve 2NF, protože existuje atribut Katedra, který není primární a není úplně závislý na klíči* (Jméno, Předmět). *Rozklad* RO = {R1(J,K), R2(J,P,H*)} již je ve 2NF, jak se dá snadno ověřit*.

Důsledky: redundance katedry pro stejné jméno
učitel dosud neučí → není možno ho zapsat
učitel přestane učit → musí se zrušit i jeho jméno

♦

- **Třetí normální forma**

Schémata ve 2NF mohou mít ještě typ redundance podobné předchozím, avšak z poněkud jiných příčin. Uvažme příklad:

Příklad:

Relační schéma Učitel (ČU, jméno, plat, funkce) *s jediným klíčem ČU je ve 2NF. Uvažme další, z reálného světa odpozorovanou funkční závislost Funkce → Plat (výše platu je podle předpisu určena zastávanou funkcí učitele). Opět můžeme zjistit následující potíže:*

- *redundance, plat je uváděn opakovaně pro každého učitele se stejnou funkcí,*
- *nebezpečí vzniku nekonzistence, plynoucí z redundance: že zapomeneme provést změny u všech prvků relace například při změně výše platu pro funkci;*
- *anomálie při vkládání; pokud žádný učitel není asistentem, nemůžeme ani vložit informaci o tom, jaký má asistent plat;*
- *anomálie při vypouštění; při vypuštění jediného profesora ztratíme i informaci o tom, jaký má profesor plat.*

Příčinou těchto anomálií u relací ve 2NF jsou opět jisté typy funkčních závislostí.

♦

Definice:

Relační schéma je ve **třetí normální formě (3NF)**, jestliže je ve 2NF a neexistuje závislost mezi sekundárními atributy.

Relace ve 2NF, které nejsou ve 3NF, se mohou rozložit vhodnou dekompozicí do 3NF.

Příklad:

Je schéma FirmaP (firma, město, obyvatel) *ve 3NF?*
Řešení: Platí: F = {firma → město, město → obyvatel}, odtud
Klíč: firma
Primární atributy: firma, *sekundární atributy:* město, obyvatel
Schéma je ve 2N, není ve 3NF..
Existuje závislost: město → obyvatel *tedy závislost mezi sekundárními atributy (neboli tranzitivní závislost sekundárního atributu na klíči);*
Důsledky: redundance hodnot obyvatel *pro stejné* město
zruší-li se firma, ztratíme i informaci o počtu obyvatel jejího města,
uložit počet obyvatel města bez firem nelze.
Řešením je opět rozklad na 2 tabulky, odpovídající funkčním závislostem:

FirmaP

firma	město	obyvatel
AA	Ostrava	320000
BB	Karviná	100000
CC	Ostrava	320000
..		

⇒

firma	město
AA	Ostrava
BB	Karviná
…	

+

město	obyvatel
Ostrava	320000
Karviná	100000
…	

♦

Příklad:

Schéma Učitel (ČU, jméno, plat, funkce) se zadanou F = {..., funkce → plat} je ve 3NF?

Řešení:

Klíč je ČU, funkce *i* plat *jsou sekundární atributy, schéma je ve 2NF, není ve 3NF.*

♦

Existují i další normální formy, odstraňující ještě další typy redundancí. Ty se však vyskytují zřídka a proto si je uvádět nebudeme. Za dobře navržené schéma budeme považovat již 3NF.

- **Bernsteinův algoritmus syntézy**

Otázkou nyní je, zda existují relační schémata, která by byla oproštěna od všech uvedených anomálií (jde zřejmě o 3NF).

Existují algoritmy, které libovolné relační schéma převedou na schémata v alespoň 3NF.

Závěrem si popíšeme algoritmus návrhu schématu relační databáze. Jde vlastně o dekompozici univerzálního schématu relace do 3NF.

Algoritmus syntézy vychází z dané množiny atributů a funkčních závislostí. Je dáno univerzální relační schéma **R** a množina funkčních závislostí **F**.

1. určí se klíč K univerzálního schématu a **F** se upraví tak, aby neobsahovala redundance;
2. závislosti se roztřídí do skupin tak, aby v každé skupině byly závislosti se stejnou levou stranou; atributy závislostí každé skupiny vytvoří schéma jedné relace; atributy levé strany tvoří klíč vzniklého schématu;
3. jsou-li v takto vzniklých schématech schémata s ekvivalentními klíči (evidující tutéž entitu), je vhodné je sloučit do jednoho schématu; při sloučení musíme dát pozor na to, aby schéma neobsahovalo závislosti mezi sekundárními atributy;
4. atributy, které se nevyskytují v žádné z funkčních závislostí F buď umístíme do samostatné relace, nebo je připojíme k některému ze schémat;
5. neobsahuje-li žádné schéma klíč univerzálního schématu K, připojíme k výsledku další relační schéma s množinou atributů K, tvořících klíč univerzálního schématu.

Výsledkem je databázové schéma ve 3NF se zachováním všech závislostí a zajištěním tzv. bezeztrátovosti. Ta říká, že operací spojení všech výsledných relací (viz. kapitola o relační algebře) je možno získat zpět původní univerzální schéma relace.

Příklad:

Navrhněte databázi pro evidenci projektů projekční firmy. Je potřeba evidovat jméno projektu, vedoucího projektu, zaměstnance pracující na projektu, počet plánovaných hodin na projekt pro každého zaměstnance, cenu projektu, datum zahájení, plat zaměstnance, oddělení, vedoucího oddělení a ohodnocení zaměstnance po ukončení projektu.

Platí: každý projekt má jednoznačné jméno,
každý projekt má jednoho vedoucího,
každé oddělení má jednoho vedoucího, oba vedoucí mohou být různí;
zaměstnanec pracuje na více projektech, na projektu pracuje více zaměstnanců
jména oddělení jsou jednoznačná, zaměstnanec patří jednomu oddělení

Řešení:

RU *(*nazev_pr, ved_pr, jmeno_zam, hod_zam, cena_pr, dat_zah, plat, oddel, ved_oddel, ohod_zam, **id_zam***)*

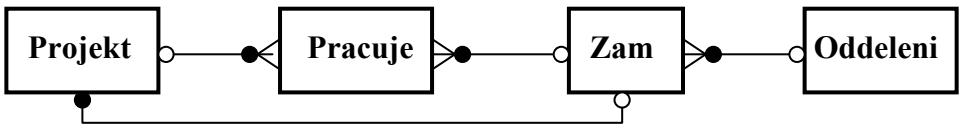
F*={* nazev_pr → ved_pr, cena_pr, dat_zah;
oddel → ved_oddel;
id_zam → jmeno_zam, oddel;
id_zam, nazev_pr → hod_zam, ohod_zam *}*

Klíčem schématu je id_zam, nazev_pr *(na těchto dvou atributech závisí všechny ostatní).*

Algoritmem syntézy dostaneme schéma R1 – R4, klíči jsou levé strany odpovídajících závislostí. Po prozkoumání atributů relací je přejmenujeme následovně:

R1 (nazev_pr, ved_pr, cena_pr, dat_zah) … Projekt
R2 (oddel, ved_oddel) … Oddeleni
R3 (id_zam, jmeno_zam, oddel) … Zam
R4 (id_zam, nazev_pr, hod_zam, ohod_zam) … Pracuje *(jde o vazební entitu mezi Zam a Projekt).*

Nyní již jen zakreslíme výsledek do ERD a vlastnosti vztahů:



Všimněme si, že tímto způsobem vyjdou „automaticky" i vazební entity. Například entita Pracuje *je vazební entitou mezi tabulkami Zam a Projekt.*

♦

Shrnutí pojmů 5.2.

Univerzální schéma databáze.

Funkční závislosti mezi množinami atributů.

Normální formy relací. 1NF, 2NF, 3NF.

Algoritmus syntézy pro návrh schématu relační databáze ve 3NF.

Otázky 5.2.

1. Co jsou normální formy relačních schémat obecně?
2. Definujte postupně 1NF, 2NF, 3NF a na každou normální formu uveďte vlastní příklady schémat, které ji splňují a které ji nesplňují.
3. Na každou normální formu uveďte příklad schématu, které ji nesplňuje a přitom splňuje formy nižší.

Úlohy k řešení 5.2.

Pro následující úlohy určete schéma univerzální relace navrhované databáze, klíč univerzálního schématu a formulujte všechny funkční závislosti. Navrhněte strukturu databáze jako množinu relací ve 3NF. Podtrhněte primární klíče relací, zakroužkujte cizí klíče.

1. Zubní lékař potřebuje evidenci svých pacientů (rodné číslo, jméno, adresu, pojišťovnu), výkonů na nich provedených (datum a druh výkonu), objednávky pacientů (datum a čas). Pro výkony existuje celostátní číselník (kód, název a cena výkonu). Pro jednoduchost

předpokládejte, že za jednu návštěvu lékaře je proveden jeden výkon, současně je ošetřován nejvýše jeden pacient.

3. Informační systém ABC soukromého zdravotnického střediska s několika lékaři. Je potřeba evidovat lékaře (osobní údaje a specializaci), pacienty (osobní údaje, pojišťovna), objednané pacienty a uskutečněné návštěvy u lékaře i lékařů u pacientů (datum a čas, diagnóza, cena pro pojišťovnu).
4. Firma potřebuje vést evidenci svých zaměstnanců (RČ, osob_číslo, jméno, adresa, fce, plat), jejich dětí (RČ, jméno, věk), seznam zakázek (dat_smlouvy, zakázka, popis, zákazník, cena_smluvní) a informace, kteří zaměstnanci pracují na kterých zakázkách (datum, kdo, na které zakázce, název_práce, cena_práce).

5.3. Relační jazyky pro vyhledávání informací

- **Rozdělení relačních dotazovacích jazyků**

Protože skutečnost, jejíž obraz zachycuje databáze, je proměnná v čase, musí být i obsah databáze proměnný. Musí tedy existovat prostředek, který umožní obsah databáze měnit. Takovým prostředkem je jazyk pro manipulaci s daty (JMD). JMD musí umožňovat základní databázové operace vyhledávání, vkládání, rušení a modifikaci prvků relace. Nejdůležitější je operace vyhledávání. Víme, že i při modifikaci a vypouštění prvků se odpovídající prvek musí nejprve vyhledat. Ostatní operace jsou z hlediska JMD jednoduché.

Jazyky pro **formulaci požadavků** na výběr dat z relační databáze (dotazovací jazyky) se dělí do dvou skupin:

1. jazyky založené na **relační algebře**, kde jsou výběrové požadavky vyjádřeny jako posloupnost speciálních operací prováděných nad daty; dotaz je tedy zadán algoritmem, jak vyhledat požadované informace;
2. jazyky založené na **predikátovém kalkulu**, které požadavky na výběr zadávají jako predikát charakterizující vybranou relaci; je úlohou překladače jazyka nalézt odpovídající algoritmus;

Oba typy jazyků jsou ekvivalentní, pokud se týče možností formulace výběrových podmínek. Každý požadavek formulovaný v relační algebře se dá vyjádřit v relačním kalkulu a naopak. Skutečně realizované jazyky se liší bohatší syntaxí a poskytují uživateli další pomocné funkce.

5.3.1. Relační algebra

Relační algebra je velmi silný dotazovací jazyk vysoké úrovně. Pracuje s celými relacemi. Operátory relační algebry se aplikují na relace - tabulky, výsledkem je opět relace - tabulka. Protože relace jsou množiny, přirozenými prostředky pro manipulaci s relacemi budou množinové operace.

I když relační algebra v této podobě není vždy implementována v jazycích SŘBD, je její zvládnutí nutnou podmínkou pro správnost manipulací s relacemi. I složitější dotazy jazyka

SQL, který je deskriptivním dotazovacím jazykem, mohou být bez zkušeností s relační algebrou problematické.

- **Základní operace relační algebry**

Jsou dány relace R a S.

Množinové operace

sjednocení relací téhož stupně $R \cup S = \{x \mid x \in R \vee x \in S\}$

průnik relací $R \cap S = \{x \mid x \in R \wedge x \in S\}$

rozdíl relací $R - S = \{x \mid x \in R \wedge x \notin S\}$

kartézský součin relace R stupně m a relace S stupně n

$$R \times S = \{rs \mid r \in R \wedge s \in S\}, \text{ kde } rs = \{r_1,...,r_m,s_1,...s_n\}$$

Další operace relační

projekce (výběr sloupců) relace R na atributy $A=\{A_{i1},...A_{im}\}$, $1 \le i_m \le n$, $i_j \ne i_k$ pro $j \ne k$

$$R[A] = \{r[A] \mid r \in R\}, \text{ kde } R[A] = (r_{i1}, \ldots, r_{im}) \text{ pro } r \in R$$

selekce (výběr řádků) z relace R podle podmínky P je

$$R(P) = \{r \mid r \in R \wedge P(r)\}$$

spojení relací R s atributy A a relace S s atributy B dle relačního operátoru

$\Theta = \{<,=,>,<=,>=,<>\}$ v atributu A_i relace R a v atributu B_j relace S je

$$R[A\Theta B]S = \{rs \mid r \in R \wedge s \in S \wedge r[A_i] \Theta s[B_j]\}$$

Θ-spojení lze definovat jako kartézský součin relací R a S následovaný selekcí s podmínkou AΘB.

Nejčastěji se používá spojení relací podle operátoru "="; v tom případě by ve výsledné relaci byly dva sloupce shodné, proto se zavádí operace

$$R\ [A*B]\ S = \{R\ [A=B]\ S\},$$

která automaticky jeden ze shodných sloupců vypouští.

přirozené spojení relací R(A) a S(B)

$$R\ [*]\ S = ((R \times S)\ [P])\ [A_1,...,A_k,C_1,..,C_{m+n-k}]$$

kde A_i jsou všechny atributy se shodnými jmény jako atributy z B a C_i jsou ostatní atributy z A i B; ze součinu RxS se vyberou ty prvky, které mají stejné hodnoty na stejně pojmenovaných atributech. Je to zvláštní případ obecného spojení. Jinak řečeno, přirozené spojení je spojení na základě rovnosti hodnot stejnojmenných atributů.

Příklad:

Jsou dány relace X, Y. Pak kartézský součin a přirozené spojení jsou:

X

A	B	C
3	S	1
5	C	2
7	T	3

Y

C	D	E
2	ff	S
2	gg	T
3	hh	C
6	jj	D

XxY

A	B	X.C	Y.C	D	E
3	S	1	2	ff	S
3	S	1	2	gg	T
3	S	1	3	hh	C
3	S	1	6	jj	D
5	C	2	2	ff	S
5	C	2	2	gg	T
5	C	2	3	hh	C
5	C	2	6	jj	D
7	T	3	2	ff	S
7	T	3	2	gg	T
7	T	3	3	hh	C
7	T	3	6	jj	D

X[*] Y

A	B	C	D	E
5	C	2	ff	S
5	C	2	gg	T
7	T	3	hh	C

♦

Příklad:

Jsou dány relace databáze VSB-TUO*:* Zaměstnanec (rodcis, jmeno, adresa, katedra), Student (rodcis, jmeno, adresa, fakulta). *Pomocí relační algebry napište*

a) seznam jmen a adres všech studentů [projekce, výběr některých sloupců celé tabulky]

Student [jmeno, adresa]

b) seznam všech údajů o studentech z fakulty elektrotechniky [selekce, výběr některých řádků splňujících podmínku]

Student (fakulta = „FEI“)

c) seznam jmen a rodných čísel studentů z fakulty elektrotechniky [selekce a pak projekce]

Student (fakulta = „FEI“) [jmeno, rodcis]

Pozor! je možno v tomto případě zaměnit pořadí selekce a projekce?

d) rodná čísla a jména studentů, zaměstnaných na této škole [= průnik obou relací, ale protože relace nemají stejnou strukturu, nejprve provedeme projekci na rodcis a jmeno *]*

Student [jmeno, rodcis] ∩ Zaměstnanec [jmeno, rodcis]

e) seznam jmen a adres studentů, kteří nepracují na této škole [od studentů odečteme zaměstnance, opět nejprve sjednotíme struktury]

Student [jmeno, adresa] - Zaměstnanec [jmeno, adresa]

f) seznam jmen a rodných čísel všech osob (studentů i zaměstnanců), majících právo se stravovat v menze [sjednocení, včetně vyloučení případných duplicit zaměstnaných studentů]

Student [jmeno, rodcis] ∪ Zaměstnanec [jmeno, rodcis]

♦

Příklad:

Mějme relaci učitel: **U** (ČU, jméno, fce, plat) *a vazbu učí* : **V** (ČU, ČP, hodin)

Máme najít čísla těch učitelů, kteří

vyučují alespoň jeden předmět:	V[ČU]
nevyučují žádný předmět:	U[ČU] - V[ČU]
vyučují předmět P2:	(V(ČP = 'P2'))[ČU]
vyučují alespoň jeden předmět, ale ne předmět P2:	V[ČU] - ((V(ČP = 'P2'))[ČU])
nevyučují předmět P2:	U[ČU] - ((V(ČP = 'P2'))[ČU])
vyučují jiný předmět, než P2:	(V(ČP <> 'P2'))[ČU]
vyučují pouze předmět P2:	((V(ČP = 'P2'))[ČU]) - ((V(ČP <> 'P2'))[ČU])

Najděte jména učitelů, kteří

vyučují alespoň jeden předmět:	(V[ČU][*]U)[jméno]

♦

Uvedená množina operací relační algebry **není minimální**, tj.existují operace, které se dají vyjádřit pomocí ostatních. Např. R ∩ S = R - (R - S). O spojení již jsme uvedli, že je realizovatelné pomocí součinu a selekce.

K relačním operacím se často pro rozšíření možností jazyka přidávají aritmetické a řetězcové operace nad hodnotami atributů nebo agregační funkce, které množinám přiřazují číslo (počet prvků množiny, součet hodnot atributu ap.). Ty se naučíme používat poprvé u jazyka SQL.

Shrnutí pojmů 5.3.1.

Relační algebra, operace s relacemi.

Množinové operace sjednocení, průnik, rozdíl, kartézský součin.

Relační operace selekce, projekce, spojení, přirozené spojení.

Databázové operace pro manipulaci s daty.

Otázky 5.3.1.

1. Jaké typy dotazovacích jazyků se používají v relačním datovém modelu?
2. Jaký vztah je mezi jazykem pro manipulaci s daty a dotazovacím jazykem?
3. Co je a k čemu je relační algebra?
4. Které operandy a které operace používá relační algebra? Definujte tyto operace.
5. Které operace RA se dají nahradit ostatními?

Úlohy k řešení 5.3.1.

1. Je dána databáze ŠKOLA se schématy

   Učitel (ČU, jméno, fce, plat)
   Předmět (ČP, název)
   Učí (ČU, ČP, hodin)

   Najděte pomocí relační algebry jména těch učitelů, kteří

   a) vyučují alespoň jeden předmět
   b) nevyučují žádný předmět
   c) vyučují předmět Překladače
   d) vyučují alespoň jeden předmět, ale ne předmět Překladače
   e) nevyučují předmět Překladače
   f) vyučují jiný předmět, než Překladače
   g) vyučují pouze předmět Překladače

2. Soukromá lékařská praxe je podporovaná relační databází se třemi relacemi se schématy:

   Lékař (číslo_licence, jménoL, specializace)
   Pacient (číslo_pacienta, jménoP, adresa, telefon, dat_narození)
   Návštěva (číslo_licence, číslo_pacienta, typ, datum, diagnóza, cena)

   kde typ znamená typ návštěvy (domů na zavolání, v ordinaci, do nemocnice ap.) Napište pomocí operací relační algebry tyto požadavky:

   a) seznam všech specializací lékařů
   b) jmenný seznam všech ortopédů
   c) jmenný seznam pacientů starších 65 let
   d) seznam licencí lékařů, které navštívila paní Marie Nová
   e) jména lékařů, kteří byli na návštěvě domů na zavolání
   f) jména a adresy pacientů, kteří byli vyšetřeni dr. Lomem dne 23.5.2003
   g) jména a adresy pacientů, kterým byla určena diagnóza HIV+
   h) jména a specializace lékařů, kteří určili diagnózu vřed na dvanácterníku
   i) jména a adresy pacientů, kteří byli vyšetřováni pouze dr. Čermákem.

3. Pomocí relační algebry napište dotazy pro 2 relace databáze VSB-TUO:

   Zaměstnanec (rodcis, jmeno, adresa, katedra),
   Student (rodcis, jmeno, adresa, fakulta).

   a) seznam jmen a adres všech studentů

   b) seznam všech údajů o studentech z fakulty elektrotechniky

   c) seznam jmen a rodných čísel studentů z fakulty elektrotechniky

   d) rodná čísla a jména studentů, zaměstnaných na této škole

   e) seznam jmen a adres studentů, kteří nepracují na této škole

   f) seznam jmen a rodných čísel všech osob, majících právo se stravovat v menze

5.3.2. Jazyk SQL

Jazyk SQL (Strucrured Query Language) byl původně navržen u firmy IBM jako dotazovací jazyk. Jeho základem je n-ticový relační kalkul, syntaxe je mírně upravena. Postupně byl rozšiřován o další skupiny příkazů, jako příkazy

- pro vytvoření a modifikace struktury tabulek (jazyk pro definici dat JDD),
- pro ukládání, modifikaci a rušení dat v databázi (jazyk pro manipulaci s daty JMD)
- a řadu dalších typů příkazů, které z něj dělají mnohem silnější prostředek, než jen dotazovací jazyk.

Probereme si z nich příkazy JDD, JMD a příkazy pro přidělování přístupových práv.

- **Příkazy jazyka pro definici dat**

Definice databáze

CREATE DATABASE Datab

Definice tabulky:

CREATE TABLE Tab (ident1 {NUMBER | CHAR | DATE }(délka)
[, ident2 ... , ident3 ...])

Příklad:

Vytvořte databázi pro IS Nemocnice s tabulkami (výsledek datové analýzy):

Pacient (rc, jmeno, dat_nar, adresa, pojistovna)
Navsteva (rc, datum, hodina, diagnoza)

Řešení:

CREATE DATABASE Nemocnice

CREATE TABLE Pacient (rc NUMBER(10), jmeno CHAR(20), dat_nar DATE,
adresa CHAR(50), pojistovna NUMBER(3));

Pacient

rc	jmeno	dat_nar	adresa	pojistovna

CREATE TABLE Navsteva (rc NUMBER(10), datum DATE, hodina NUMBER(2),
diagnoza CHAR(20));

Navsteva

rc	datum	hodina	diagnoza

♦

Definice tabulky s integritními omezeními

```
CREATE TABLE  Tab (
    ident1 {NUMBER|CHAR|DATE }(délka) PRIMARY KEY,
    ident2 … AUTO_INCREMENT,
    ident3 … NOT NULL,
    ident4 … DEFAULT i,
    ident5 … CHECK (ident5 IN (NULL, hod1, hod2, …)),
    ident6 … REFERENCES Tab2,
  ... )
```

kde **PRIMARY KEY** označuje atribut, který je primárním klíčem,
AUTO_INCREMENT atribut je automaticky číslován u každého nového záznamu,
NOT NULL atribut musí být zadán, nesmí mít hodnotu NULL,
DEFAULT i atribut má přednastavenou hodnotu **i,**
CHECK (... IN (...)) atribut může nabývat jen hodnot za zadané množiny,
REFERENCES Tab2 atribut je cizím klíčem z Tab2, realizuje vazbu do Tab2

přitom u jednoho atributu může být zadána kombinace těchto omezení.

Příklad:

```
CREATE TABLE Pojis (cis_poj NUMBER(3) PRIMARY KEY, poj CHAR(20));

CREATE TABLE Pacient (
        id_pac NUMBER(10) PRIMARY KEY AUTO_INCREMENT,
        jmeno CHAR(20) NOT NULL,
        adresa CHAR(30),
        cis_poj NUMBER(3) REFERENCES Pojis,
        datnar DATE);
```

♦

Modifikace struktury tabulky

```
ALTER TABLE tab {MODIFY (ident   dat_typ (nový rozměr)) |
                 ADD  ident   dat_typ (rozměr) |
                 DROP  ident}
```

Příklad: Sloupec pro diagnózu CHAR(20) už nestačí, je třeba přidat 10 znaků.

```
ALTER TABLE  Navsteva MODIFY (diag CHAR (30) ); ♦
```

Příklad: Do tabulky Navsteva je nutno přidat sloupce o provedeném výkonu.

```
ALTER TABLE Navsteva ADD (vykon CHAR (20) ); ♦
```

Přejmenování tabulky nebo sloupce tabulky

```
RENAME TABLE  tab_stará  TO  tab_nová

RENAME COLUMN  tab.sloupec_starý  TO  tab.sloupec_nový
```

Zrušení tabulky včetně definice

DROP TABLE tab

Vytvoření a rušení indexu:

CREATE [UNIQUE] INDEX index ON tab (seznam_klíčů)

DROP INDEX index

Klauzule UNIQUE znamená požadavek jednoznačnosti indexu

Příklad:

Potřebujeme vytvořit indexový soubor k tabulce pacientů (viz. kap. 4.) pro vyhledávání podle jména. Dále indexovat tabulku Navsteva podle rodného čísla a data návštěvy pro vyhledávání záznamů o jednom pacientovi.

CREATE UNIQUE INDEX Ind_pac ON Pacient (jmeno);

CREATE INDEX Ind_navst ON Navsteva (rc, datum);

♦

- **Příkazy jazyka pro modifikace dat**

- **Vkládání nových řádků**

se provádí příkazem INSERT. Proti použití na počátku je možno specifikovat i seznam a pořadí sloupců, do kterých se budou hodnoty ukládat a tak není nutné uvádět i hodnoty sloupců nevyplňovaných NULL (protože jejich hodnoty nejsou známy nebo budou dopočítány nebo doplněny později).

INSERT INTO tab [seznam_sloupců] VALUES (seznam_hodnot)

Příklad:

K lékaři se přihlásil nový pacient a objednal se na 4.4.2006 v 9.00 (zadávají se hodnoty pro všechny atributy)

INSERT INTO Pacient VALUES (6606065120, 'Novotný Jan', '6.6.1966', 'Opava, Květná 34', 111);

Lékař zapisuje objednanou návštěvu, zatím bez diagnózy, ta se dopíše později.

INSERT INTO Navsteva rc, datum, hodina VALUES (6606065120, '4.4.2006', 9.00)

Bez specifikace sloupců bychom museli uvést všechny hodnoty:

INSERT INTO Navsteva VALUES (6606065120, '4.4.2006', 9.00, NULL)

Pacient

rc	jmeno	dat_nar	adresa	pojistovna
6606065120	Novotný Karel	6.6.1966	Opava, Květná 34	111

Navsteva

rc	datum	hodina	diagnoza
6606065120	4.4.2006	9.00	NULL

Pomocí příkazu INSERT je možno také naplňovat řádky tabulky hodnotami z jiné tabulky tak, že místo klauzule VALUES použijeme SELECT, v němž budou zadány řádky i sloupce jiných tabulek, které se do naší tabulky mají přenést.

Protože dosud příkaz SELECT neznáme, uvedeme tuto variantu Příkazu INSERT později.

- **Modifikace hodnot** v řádcích tabulky

  UPDATE tab
  SET ident1=výraz1 [, ident2=výraz2,...] [WHERE podm]

Podmínka za WHERE definuje řádky, pro které bude změna provedena.

Příklad:

Pacient Novotný s rc=6606065120 přišel na vyšetření a je mu doplněna diagnóza.

```
UPDATE Navsteva
    SET diag = 'chřipka'
    WHERE rc = 6606065120; ♦
```

Příklad:

Všichni zaměstnanci katedry 455 dostali přidáno na platu 300 Kč.

```
UPDATE Zam
    SET plat = plat + 300
    WHERE kat = 455; ♦
```

Příklad:

Prémii ve výši 20% platu dostali všichni zaměstnanci s uvedením „ano" ve sloupci odměny.

```
UPDATE Zam
    SET plat = plat * 1.2
    WHERE odměny = „ano" ♦
```

- **Rušení řádků** tabulky

  DELETE FROM tab WHERE podm

Příklad:

Pacient Jára s rc 7707073345 ruší objednávku u lékaře na den 4.4.2006, ta se vymaže z tabulky Navsteva.

```
DELETE FROM Navsteva
    WHERE rc = 7707073345 AND datum = '4.4.2006' ♦
```

- **Výběr informací z tabulky**

Následující příkaz SELECT reprezentuje vlastní dotazovací jazyk. Jeho použitím je možno nejen vyhledávat údaje v databázi obsažené, ale i údaje odvozené, případně vhodně setříděné. Základní nejjednodušší tvar příkazu je

SELECT {seznam_sloupců | *}
FROM tab
[WHERE podm]

Použití znaku * místo seznamu sloupců znamená výpis všech sloupců tabulky.

Příkaz jazyka SQL

SELECT $A_1, A_2, \ldots, A_k$ FROM R WHERE podm

odpovídá výrazu relační algebry

$(R\ (podm)\ )\ [A_1, A_2, \ldots, A_k]$

Jednoznačnost prvků výsledné relace nezajišťuje jazyk SQL automaticky, ale musí se zadat v příkazu klauzulí UNIQUE (nebo DISTINCT)

Příklad:

Celou tabulku vytiskneme příkazem

SELECT * FROM Pacient; ♦

Příklad:

Jmenný seznam a adresy všech pacientů dostaneme příkazem

SELECT jméno, adresa FROM Pacient; ♦

Příklad:

Seznam pojišťoven, u nichž jsou přihlášeni Pacienti, dostaneme

SELECT UNIQUE pojistovna FROM Pacient; ♦

Příklad:

Seznam pacientů z pojišťovny VZP (s číslem 111)

SELECT * FROM Pacient WHERE pojistovna = 111; ♦

- **Podmínka selekce**

se zapisuje za klauzulí WHERE. Ve výběrové podmínce je možno používat:

konstanty, identifikátory sloupců,
relační operátory : **= <> < <= > >=**
logické operátory: **NOT AND OR**
další operátory : **BETWEEN dolní mez AND horní mez**
IN(seznam_prvků_množiny)
IS NULL
LIKE vzor ... pro porovnání řetězců podobně jako u hvězdičkové konvence:
% ... odpovídá skupině znaků
_ ... podtržítko zastupuje jeden znak

Příklad:

Seznam objednávek pacientů na hodiny 7.00 a 8.00 v týdnu od 20.4.2006 do 24.4.2006

```
SELECT *
        FROM Navsteva
        WHERE hodina IN (7, 8)
        AND datum BETWEEN ’20.4.2006’ AND 24.4.2006’; ♦
```

Příklad:

Seznam pacientů se jménem začínajícím písmenem K.

```
SELECT *
        FROM Pacient
        WHERE jméno LIKE ‘K%’; ♦
```

Příklad:

Seznam pacientek (zjednodušeně = jméno končí na -ová).

```
SELECT *
        FROM Pacient
        WHERE jméno LIKE ‘%ová’; ♦
```

- **Setřídění výsledku**

výsledných řádků podle třídicího klíče, ne podle pořadí uložení v souboru:

```
SELECT {seznam_sloupců | *}
    FROM tab
    [WHERE podm]
    [ORDER BY třídicí klíč [DESC]]
```

Příklad:

Abecední seznam pacientů pojišťovny 111.

```
SELECT *
        FROM Pacient
        WHERE pojistovna = 111
        ORDER BY jmeno; ♦
```

Příklad:

Vypište Zam (rc, jmeno, fce, plat*) podle platů od nejvyššího dolů, při stejném platu abecedně.*

```
SELECT *
        FROM Zam
        ORDER BY plat DESC, jmeno; ♦
```

Pokud jsou v třídicím klíči prázdné hodnoty, uvádí se tyto řádky vždy na začátku tabulky při sestupném i vzestupném třídění.

- **Spojení**

více tabulek (vazba) se provede uvedením všech tabulek za FROM a podmínka spojení se uvede jako součást výběrové podmínky za WHERE. Bez této podmínky by se provedl prostý kartézský součin vyjmenovaných tabulek. Rozlišení stejnojmenných sloupců provedeme uvedením jména tabulky před jménem sloupce a oddělené tečkou.

SELECT {seznam_sloupců | *}
FROM seznam_tabulek
[WHERE podm_spojení]

Příklad:

Jmenný seznam pacientů s adresami, objednaných na týden od 20.4.2006 do 24.4.2006 včetně jejich objednávky (nutno zrušit objednávky).

```
SELECT  jméno, adresa, datum, hodina
   FROM Pacient, Navsteva
   WHERE datum BETWEEN ’20.4.2006’ AND ’24.4.2006’  ... selekce
   AND Pacient.rc = Navsteva.rc;                    ... podmínka spojení
```

♦

Spojení pomocí JOIN (implementováno u některých SŘBD)

SELECT {seznam_sloupců | *}
FROM Tab1 JOIN Tab2 ON Tab1.iden = Tab2.iden
[WHERE podm]

Příklad:

Jmenný seznam pacientů s adresami, objednaných na týden od 20.4.2006 do 24.4.2006 včetně jejich objednávky

```
SELECT  jméno, adresa, datum, hodina
   FROM Pacient
   JOIN Navsteva  ON Pacient.rc = Navsteva.rc      ... podmínka spojení
   WHERE datum BETWEEN ’20.4.2006’ AND ’24.4.2006’ ... selekce
```

♦

- **Použití prefixů**

Pokud je název tabulky jako prefix nepohodlně dlouhý, nebo pokud potřebujeme jednu tabulku označit dvakrát pokaždé jinak (např. pro realizaci unární vazby), můžeme za klauzulí FROM každé tabulce zadat vlastní prefix.

SELECT {seznam_sloupců | *}
FROM tab1 P1, tab2 P2, . . .

Příklad:

Použití prefixu pro zjednodušení zápisu podmínky

```
SELECT  jméno, adresa, datum, hodina
   FROM Pacient P, Navsteva N
   WHERE datum BETWEEN ’20.4.06’ AND ’24.4.06’ AND P.rc = N.rc; ♦
```

Podmínka spojení může být nejen na rovnost hodnot.

Příklad:

V tabulce Zam (rc, jmeno, fce, plat, vedouci) je také jméno vedoucího katedry u každého zaměstnance. Zajímá nás seznam zaměstnanců, kteří mají vyšší plat, než jejich vedoucí.

```
SELECT X.jmeno, X.plat, Y.plat
   FROM Zam X, Zam Y                                    … X = zam, Y = vedouci
   WHERE X.vedouci = Y.jmeno AND X.plat >Y.plat;
```

Zam X

rc	jmeno	...	plat	vedouci
			1500	Novák
	Novák		3000	Novák
			3500	Novák

Zam Y

rc	jmeno	...	plat	vedouci
	Novák		3000	

♦

- **Levé a pravé spojení**

Pokud se při spojování tabulek nenajde v jedné tabulce odpovídající řádek ze druhé tabulky, ve výsledku spojení se odpovídající hodnota spojovacího klíče neobjeví. Může však nastat situace, že požadujeme ve výsledné tabulce i tento řádek (odpovídající hodnoty ze druhé tabulky zůstanou prázdné), zadáme to příznakem (+) u podmínky spojení na té straně tabulky, ze které se mají řádky doplnit všechny.

Poznámka: Konkrétní SQL mají syntaxi pro zadání levého a pravého spojení jinou.

Příklad:

V tabulce Pacient mohou být takoví, kteří v roce 2001 ještě nebyli pacienty tohoto lékaře, ale ve výsledném seznamu jejich jména požadujeme s prázdnou hodnotou data a diagnózy.

```
SELECT P.jméno, datum, diag
   FROM Pacient P, Navsteva N
   WHERE datum BETWEEN '1.1.2001' AND '31.12.2001' AND P.rc = N.rc(+)
   ORDER BY  P.rc, datumů; ♦
```

- **Výrazy a funkce, agregované funkce**

Pro vytváření výrazů používá SQL aritmetických operátorů a závorek v obvyklých významech. Místo jména sloupce můžeme použít výraz, a to v seznamu za SELECT či v podmínce za WHERE.

Sloupcové nadpisy

Pokud je výraz použit jako prvek seznamu za SELECT a chceme příslušnému sloupci na výstupu přiřadit sloupcový nadpis vlastní, zapíšeme ho po mezeře za výrazem. Pokud se nadpis skládá z více slov, uzavírá se do závorek.

Příklad:

Je dána tabulka Zam (rc, jmeno, fce, plat, proc_dan). *Vypište výplatní listinu s platem hrubým i čistým (zjednodušeně čistý plat = plat hrubý - zadané procento z platu)*

```
SELECT rc, jmeno, plat, plat - plat / 100 * proc_dan
   FROM Zam;
```

Zam

rc	jmeno	plat	plat - plat / 100 * proc_dan
	...		

nebo s přejmenovanými a pojmenovanými sloupci

```
SELECT rc, jmeno, plat (hruby plat), plat – plat / 100 * proc_dan (cisty plat)
   FROM Zam;
```

Zam

rc	jmeno	hruby plat	cisty plat
	...		

♦

- **Náhrada hodnoty NULL**

Pokud je ve výrazu proměnná, která má hodnotu NULL, má celý výraz hodnotu NULL. Někdy se takový výstup nehodí a chceme nahradit hodnotu NULL konkrétní náhradní hodnotou (např. nulou pro číslo nebo mezerou pro text). K tomu slouží funkce

NVL (prom, náhr_hodn)

Příklad:

Pokud procento daně zaměstnance ještě není vyplněno (má hodnotu NULL), při výpočtu se nahradí hodnotou 100 a tedy čistý plat bude 0.

```
SELECT jmeno, plat, plat – plat / 100 * NVL (proc_dan, 100)
      FROM Zam; ♦
```

- **Funkce nad atributy**

Dále jazyk SQL používá následující funkce

- aritmetické: **POWER(číslo,mocnitel) mocnina**
  ROUND(číslo,poč_des_míst)
  TRUNC(číslo,poč_des_míst)
  ABS(číslo)
  SIGN(číslo)
  MOD(číslo1,číslo2)
  SQRT(číslo)
  GREATEST(seznam_hodnot) maximum
  LEAST(seznam_hodnot) minimum

- řetězcové: **řetězec1 || řetězec2**
  LENGHT(řetězec)
  SUBSTR(řetězec, pozice, délka) výběr podřetězce
  INSTR(řetězec, podřetězec, pozice, pořadí)
  UPPER(řetězec)
  LOWER(řetězec)
  TO_NUMBER(řetězec) konverze text -> číslo
  TO_CHAR(řetězec [, formát]) datum či číslo -> text
  TO_DATE(řetězec [, formát]) text či číslo -> datum
  LPAD(řetězec, délka, [, znak]) doplní zleva
  RPAD(řetězec, délka, [, znak]) doplní zprava
  LTRIM(řetězec, množ_znaků) vypustí zleva
  RTRIM(řetězec, množ_znaků) vypustí zprava
  DECODE(sl, vzor1, obraz1, ..., implic)

  Decode je tzv. překlad tabulkou: obraz může být konstanta nebo jméno sloupce; může sloužit i jako přepínač, když ve sloupci nechceme homogenní hodnoty.

- datumové: číselné masky pro formátování datumových hodnot:

Příklad:

U zaměstnanců Zam *s atributem* kat = *katedra potřebujeme tisknout také kód učitelské funkce podle pravidla* asistent=1, docent = 2, profesor = 3, ostatní zaměstnanci = 4.

```
SELECT jméno, funkce, DECODE (funkce, 'asist', 1, 'doc', 2, 'prof', 3, 4) kat
    FROM Zam;
```

Jméno	funkce	kat
Adam Alois	doc	2
Beran Bedřich	topič	4
Cerman Cyril	asist	1

♦

Příklad:

U pacientů VZP požadujeme celé adresy, u ostatních jen text 'ostatní'.

```
SELECT jméno, DECODE(pojis, 111, adresa, 'ostatní' ) adresy
  FROM Pacient;
```

Jméno	adresy
Adam Alois	ostatní
Beran Bedřich	U pošty 15, Dobřichovice
Cerman Cyril	ostatní

♦

- **Funkce agregované**

V praxi se často potřebují některá data v databázi sčítat, průměrovat apod. Představíme-li si tyto operace nad tabulkou, jde buď o počty záznamů (řádků) splňujících nějakou podmínku, nebo o operace sloupcové: součet, průměr, minimum, maximum hodnoty některého sloupce.

I když nejde o operace relační (nepracují s prvky relace = řádky, ale s daty ve sloupci, tedy z každého prvku používají jen komponentu), existují ve všech databázových jazycích i v SQL funkce, které tyto výpočty provádějí. Nazýváme je souhrnně funkcemi agregovanými. Jsou to:

AVG ({[UNIQUE] sez_výr|*})	průměr ve sloupci
SUM ({[UNIQUE] sez_výr|*})	součet …
MIN ({[UNIQUE] sez_výr|*})	minimum …
MAX ({[UNIQUE] sez_výr|*})	maximum …
COUNT({[UNIQUE] sez_výr|*})	počet vyhovujících řádků

Příklad:

Určete průměrný, minimální a maximální plat zaměstnanců - profesorů.

```
SELECT AVG(plat), MIN(plat), MAX(plat)
  FROM Zam WHERE fce = 'profesor';
```
♦

Příklad:

Kolik je pacientů od pojišťovny 111 ?

```
SELECT COUNT(*) FROM Pacient WHERE pojis = 111;
```
♦

Příklad:

Od kolika pojišťoven jsou naši pacienti ?

```
SELECT COUNT (UNIQUE pojis) FROM Pacient;
```
♦

- **Grupování a podmínka pro grupy**

Někdy potřebujeme provést agregaci (určit počet, součet, průměr, …) ne pro celou tabulku nebo jednu definovanou selekci, ale současně pro více skupin řádků se stejnou hodnotou některého atributu (grupovací klíč).

Tabulku si můžeme představit uspořádanou tak, že vzniknou skupiny řádků se stejnou hodnotou grupovacího atributu. Také pro tyto skupiny můžeme provádět operace (částečné počty, součty apod.). Skupinám řádků se stejnou hodnotou grupovacího klíče říkáme grupy.

Nyní si můžeme všechny atributy tabulky rozdělit podle významu do třech skupin:

(1) atributy definující grupu, tedy ve výsledku se ke každé skupině stejných hodnot těchto atributů vstupní tabulky vytvoří jeden řádek ve výstupní tabulce

(2) atributy agregovatelné, které se za celou grupu sčítají, průměrují apod.

(3) ostatní atributy, jejich hodnoty ve výsledku nemají žádný význam.

Vytvoření skupin se provede klauzulí GROUP BY klíč

SELECT {seznam_výrazů | *}
FROM seznam_tabulek
GROUP BY seznam_sloupců

Příklad:

Chceme zjistit průměrné platy pro jednotlivé funkce zaměstnanců. Grupovat tedy budeme podle funkcí (atribut typu 1), agregovaná hodnota – zde průměr – bude plat (atribut typu 2), atributy jmeno, adresa nebo okres ve výsledku nemají význam, není možno určit celé grupě jednu hodnotu.

SELECT fce, AVG (plat)
FROM Zam
GROUP BY fce

(3) jméno	(3) adresa	(1) okres	(2) fce	(2) plat
Beneš	…	...	prof	30000
Gajdoš	…	...	prof	32000
…				
Adam		...	doc	23000
…				
Žižka	…	...	ukliz	12500

(1) fce	AVG(2) AVG(plat)
prof	
doc	
ukliz	

♦

Příklad:

*Najděte počet pacientů pro jednotlivé pojišťovny***.**

SELECT pojis, COUNT(*)
FROM Pacient
GROUP BY pojis; ♦

Příklad:

Najděte průměrný, maximální a minimální plat zaměstnanců podle funkcí

SELECT fce, AVG(plat), MIN(plat), MAX(plat)
FROM Zam

GROUP BY fce; ♦

Příklad:

Rozdělte zaměstnance fakulty podle kateder a funkcí, pro každou tuto skupinu určete jejich počet a průměrný plat.

Kat	funkce	COUNT(*)	AVG(plat)
401	prof	3	8700
401	doc	7	7800
401	asist	12	6500
402	prof	2	16500

```
SELECT kat, funkce, COUNT(*), AVG(plat)
    FROM Zam
    GROUP BY kat, funkce; ♦
```

Pokud pracujeme se skupinami a chceme formulovat podmínku pro celou skupinu, nejen pro jednotlivé řádky původních tabulek, nedává se tato podmínka za WHERE, ale za HAVING:

```
SELECT {seznam_výrazů | *}
   FROM seznam_tabulek
   [WHERE podm_pro_řádek]
   [GROUP BY seznam-klíčů
   [HAVING podm_pro_skupinu] ]
```

Příklad:

Najděte průměrný plat skupin zaměstnanců na fakultě, které mají více než 25 členů.

```
SELECT kat, funkce, AVG(plat)
    FROM Zam
    GROUP BY kat, funkce
    HAVING COUNT(*) > 25; ♦
```

Příklad:

Najděte katedry, kde jsou alespoň 2 sekretářky.

```
SELECT kat
  FROM Zam
  WHERE funkce = 'sekretářka'
  GROUP BY kat
  HAVING COUNT(*) >= 2; ♦
```

Příklad:

Najděte katedry, kde je průměrný plat asistenta větší, než 5000.

```
SELECT kat, AVG(plat)
  FROM Zam
  WHERE funkce = 'asist'
  GROUP BY kat
  HAVING AVG(plat) > 5000; ♦
```

- **Úplná syntaxe příkazu SELECT**

Shrneme-li přidávané nepovinné klauzule příkazu pro vyhledávání informací, dostaneme

```
SELECT {ALL | UNIQUE } seznam_výrazů
    FROM tab[ tabn] [, ...]
    [ WHERE podm ]
    [ GROUP BY seznam_sloupců
        [ HAVING podm ] ]
    [ ORDER BY sl [ASC | DESC] [, ...] ]
```

Klauzule ALL platí implicitně (všechny řádky i duplicitní), UNIQUE odstraní duplicitní řádky.

- **Logické kvantifikátory**

Některé dotazy v relačním kalkulu vyžadují použití logického kvantifikátoru EXISTS.

SELECT . . .
FROM . . .
[WHERE [NOT] EXISTS (SELECT ... FROM ... [...])]

kde EXISTS(SELECT ...)=TRUE, je-li množina daná výrazem v závorkách neprázdná, jinak je FALSE;

Tentýž dotaz lze formulovat pomocí IN.

Příklad:

Mějme relace Ctenar (id_ct, jmeno, adresa)
Kniha (prir, ISBN, autor, nazev)
Rezer (id_ct, ISBN, dat_od) ... aktuální rezervace

Hledáme jména čtenářů s nějakou knihu rezervovanou (= existuje záznam s jejich id_ct v Rezer). Můžeme použít klauzuli EXIST nebo IN:

```
SELECT jmeno
   FROM Ctenar
   WHERE EXIST (SELECT R.*
                        FROM Rezer R, Ctenar C
                        WHERE R.id_ct = C.id_ct)

SELECT jmeno
   FROM Ctenar
   WHERE id_ct IN (SELECT id_ct
                        FROM Rezer)  ♦
```

- **Množinové operace**

Z množinových operací dosud známe IN, NOT IN a kartézský součin. Další operace je v SQL definována takto:

Sjednocení

SELECT ... FROM ... [WHERE ... GROUP BY ... HAVING ...]
UNION
(SELECT ... FROM ... [...])
[ORDER BY ...]

Příklad:

Abecední seznam strávníků v menze školy - seznam zaměstnanců i studentů.

```
SELECT jmeno FROM Student
UNION
(SELECT jmeno FROM Zam)
ORDER BY jmeno
```

Pozor na častou chybu záměny sjednocení a spojení! Zde není možno použít:

… FROM Student, Zam ... ♦

- **Poddotazy**

V SQL je možno dotazy řetězit, pro formulaci hlavního dotazu je možno použít výsledků dotazu jiného - poddotazu. Přípustné je použít podotázky v podmínce za WHERE podle následujících pravidel:

- pokud je výsledkem poddotazu jediná hodnota (relace o 1 řádku a 1 sloupci), pak kdekoliv místo hodnoty:
  výraz rel_oper (příkaz SELECT)
- je-li výsledkem poddotazu množina hodnot (relace o 1 sloupci):
  výraz [NOT] IN (příkaz SELECT)

Poddotazy je možno použít za klauzulí WHERE i v příkazech UPDATE a DELETE.

Příklad:

Hledáme včelaře patřící do organizace jako Kovář Karel (mají stejného předsedu).

1. buď dvěma dotazy s „ručním" přenesením mezivýsledku

```
SELECT předs FROM Včelaři WHERE předs = 'Kovář Karel';
SELECT jméno, adresa FROM Včelaři WHERE předs = ' ...';
```

kde '...' je výsledek 1. dotazu.

2. nebo zřetězeně pomocí poddotazu

```
SELECT jméno, adresa
    FROM Včelaři
    WHERE předs=(SELECT předs
                 FROM Včelaři
                 WHERE jméno='Kovář Karel'); ♦
```

Příklad:

Najděte včelaře, kteří získali více medu než alespoň jeden včelař okresu Ústí (~ co má nejméně).

```
SELECT jméno, adresa
    FROM Včelaři
    WHERE medu > ANY (SELECT medu
                      FROM Včelaři
                      WHERE okres = 'Ústí'); ♦
```

Příklad:

Najděte funkce, které mají vyšší průměrný plat, než je průměrný plat docenta.

```
SELECT funkce, AVG(plat)
  FROM Fakulta
  GROUP BY kat
  HAVING AVG(plat) > (SELECT AVG(plat)
                      FROM Fakulta
                      WHERE funkce = 'doc');♦
```

Příklad:

Vypište seznam pacientů a jejich adresy z pojišťoven, u kterých má lékař více než 10 pacientů, setříděný podle pojišťoven.

```
SELECT jméno, adresa, pojis
   FROM Pacient
   WHERE pojis IN (SELECT pojis
                          FROM Pacient
                          GROUP BY pojis
                          HAVING COUNT(*) > 10)
   ORDER BY pojis, jmeno; ♦
```

- **Pohledy**

Pohled představuje **virtuální tabulku** - relaci přímo v databázi neexistující, ale definovatelnou jako výsledek některého příkazu SELECT. Může to být projekce či selekce existující tabulky či spojení několika existujících tabulek, může obsahovat i sloupce odvozené z existujících hodnot (virtuální sloupce) ap. Pohled se definuje (podobně jako skutečná tabulka) příkazem:

```
CREATE VIEW pohled AS
    SELECT {seznam_sloupců | *}
    FROM tab
    [WHERE podm]
    [GROUP BY seznam_sloupců]   [HAVING podm]
    [ORDER BY seznam_sloupců]
```

Dále se s pohledem pracuje jako se skutečnou tabulkou. Provedeme-li změny v pohledu, změní se i hodnoty v tabulce, z níž je pohled odvozen a naopak. Problémem je provedení změn ve virtuálních sloupcích, v pohledech setříděných atd., proto konkrétní implementace práci s pohledy omezují jen na některé funkce.

Příklad:

Aplikace potřebuje jméno, adresu a pojišťovnu pacientů z Ostravy.

```
CREATE VIEW Ostravaci AS
   SELECT jmeno, adresa, pojistovna
   FROM Pacient
   WHERE adresa LIKE 'Ostrava'; ♦
```

Příklad:

Jindy potřebujeme všechny údaje z obou tabulek Pacient i Navsteva.

```
CREATE VIEW Vsichni AS
   SELECT P.*, datum, hodina, diag
   FROM Pacient P, Navsteva N
   WHERE P.rc = N.rc; ♦
```

Při vytváření virtuálních sloupců zadáme jména nově vzniklých sloupců za názvem pohledu.

Příklad:

Potřebujeme tabulku ročních příjmů zaměstnanců.

```
CREATE VIEW Rocni_prijem (jmeno, mesicne, rocne) AS
   SELECT jméno, plat, plat*12
   FROM Zam; ♦
```

- **Doplnění příkazů INSERT, UPDATE , DELETE**

Pomocí příkazu INSERT je možno také naplňovat řádky tabulky hodnotami z jiné tabulky tak, že místo klauzule VALUES použijeme SELECT. Tímto příkazem zadáme řádky a sloupce nebo vypočtené hodnoty z jiných tabulek, které se mají do doplňované tabulky přenést.

```
INSERT INTO tab [seznam_sloupců]
     SELECT seznam_hodnot FROM tab2 WHERE podm ...
```

Příklad:

Do definované prázdné tabulky ostravských pacientů se opíší pacienti z celé evidence lékaře:

```
INSERT INTO Ostr_pacienti
   SELECT * FROM Pacient WHERE adresa LIKE 'Ostrava'; ♦
```

Také u příkazů UPDATE a DELETE je možno za klauzulí WHERE používat všechny funkce, operátory, konstrukty, poddotazy jako u příkazu SELECT.

Příklad:

Prémii ve výši 20% platu dostali všichni zaměstnanci uvedení v seznamu Odměny (jmeno).

```
UPDATE Zam
     SET plat = plat * 1.2
     WHERE jméno IN (SELECT jméno FROM Odměny); ♦
```

Příklad:

Zaměstnanci ze seznamu NoZam (jmeno) *dostali výpověď.*

```
DELETE FROM Zam
     WHERE jméno IN (SELECT jméno FROM NoZam); ♦
```

- **Příkazy pro sdílení a ochranu dat**

Již víme, že SQL neobsahuje jen prostředky pro práci s databází, ale i prostředky podporující správu databáze - příkazy pro přidělování a odebírání přístupových práv na různých úrovních různým uživatelům databáze.

V SŘBD, které tyto prostředky mají implementovány, platí: k tabulce, kterou uživatel vytvořil, má přístup jen on sám, pokud nezpřístupní svou tabulku jinému uživateli příkazem GRANT. Odebrat toto právo může příkazem REVOKE.

Příkaz pro přidělení přístupového práva má tvar

GRANT tabulkové_právo
ON { tab | pohled }
TO { PUBLIC | sez_jmen_uživatelů }

kde jméno uživatele je příslušný login_name, PUBLIC znamená zveřejnění tabulky všem uživatelům. Tabulková práva umožňují jemně rozlišit přesné vymezení přístupu různých uživatelů k různým datům:

SELECT [(seznam_sloupců)]	jen čtení příslušných sloupců tabulky
INSERT	vkládat nové řádky do tabulky
UPDATE [(seznam_sloupců)]	modifikovat sloupce tabulky
DELETE	rušit řádky tabulky
ALTER	modifikovat strukturu tabulky
INDEX	vytvářet indexy nad sloupci tabulky
ALL	všechna výše uvedená práva

Odebrání přístupového práva se provede příkazem

REVOKE tabulkové_právo
ON { tab | pohled }
FROM { PUBLIC | sez_jmen_uživatelů }

Některé SŘBD rozšiřují přidělování a odebírání práv i na celou databázi.

Příklad:

Pro definovanou tabulku Zam (RC, jmeno, adresa, plat, fce) *povolte záznam údajů o zaměstnanci* - RC, jmeno, adresa *perzonalistce s loginem ABC20,* plat, fce *vedoucímu katedry* DEF30*, ostatním jen prohlížení* jména, adresy, fce*, správci databáze* GHI40 *všechna práva.*

```
REVOKE ALL ON Zam FROM PUBLIC
GRANT INSERT ON Zam TO ABC20
GRANT UPDATE (RC, jmeno, adresa) ON Zam TO ABC20
GRANT SELECT (RC, jmeno, adresa) ON Zam TO ABC20
GRANT UPDATE (plat, fce) ON Zam TO DEF30
GRANT SELECT (plat, fce) ON Zam TO DEF30
GRANT ALL ON Zam TO GHI40
```

♦

❑ Další možnosti SQL

SQL není jen dotazovací jazyk, ale obsahuje i příkazy další.

Probrali jsme jeho základní příkazy. Především obsahuje všechny potřebné příkazy JDD, definuje, modifikuje a ruší databázi, tabulky, indexy, pohledy. Dále obsahuje příkazy JMD, ukládá data do databáze, modifikuje je a ruší. Dotazovací jazyk reprezentuje příkaz SELECT

se všemi svými variantami. SQL zahrnuje také příkazy sloužící správě databáze pro přidělování přístupových práv na různé úrovni různým uživatelům.

Ale to není všechno. SQL obsahuje i příkazy pro

- vytváření **modulů**,
- pro **řízení transakcí**,
- pro záznam některých **integritních omezení**,
- pro vytváření **hierarchických struktur** dat,
- pro práci se **systémovým katalogem** ap.

Shrnutí pojmů 5.3.2.

SQL a jeho význam.

Příkazy jazyka pro definici dat v SQL.

Příkazy jazyka pro manipulaci s daty v SQL, základní tvar dotazu. Funkce, agregované funkce.

Grupování, výběrové podmínky pro grupy.

Poddotazy.

Pohledy

Přidělování a odebírání přístupových práv.

Vývoj a standardy SQL, význam standardů.

Otázky 5.3.2.

1. Které skupiny příkazů jazyka SQL jsme probrali?
2. Které příkazy SQL patří do JDD a proč?
3. Které příkazy SQL patří do JMD a proč?
4. Které příkazy SQL tvoří dotazovací jazyk proč?
5. Jaká je syntaxe úplného vyhledávacího příkazu SQL?
6. Co je výsledkem příkazu SELECT?
7. Jaké typy funkcí je možno používat v SQL?
8. Co je grupování a jak se s grupami pracuje?
9. Co je poddotaz a co je jeho výsledkem?
10. Co je pohled, co je jeho výsledkem a k čemu se používá?
11. Které příkazy jazyka SQL umožňují přidělování a odnímání přístupových práv uživatelům?

Úlohy k řešení 5.3.2.

1. Je dáno schéma relační databáze pro automatizaci provozu VEŘEJNÉ KNIHOVNY :

Knihy (ISBN, autor, název, země)

Exemplář (ISBN, PŘÍR, koupeno, cena)

Čtenář (číslo, jméno, adresa)

Výpůjčky (PŘÍR, číslo, půjčeno)

Rezervace (ISBN, číslo, rezervováno)

kde položky koupeno, půjčeno, rezervováno jsou datumové, ISBN je číslo titulu knihy (bez ohledu na počet kopií v knihovně), PŘÍR je přírůstkové číslo exempláře knihy, číslo je číslo průkazu čtenáře.

a) Vypište seznam jmen a adres čtenářů.

b) Vypište seznam čtenářů setříděný abecedně podle jmen.

c) Vypište knihy nakoupené po 20.3.1992.

d) Najděte autory, na jejichž některé tituly je rezervace před 31.12.92.

e) Najděte dvojice čtenářů se stejnou adresou.

f) Najděte všechny exempláře vydané na Slovensku a jejich cenu v SK.

g) Zjistěte celkový počet čtenářů knihovny.

h) Určete počet čtenářů, kteří mají rezervovanou nějakou knihu.

i) Určete počet titulů knihovny,které byly vydány na Slovensku.

j) Jaká je celková cena slovenských knih v knihovně ?

k) Určete pro každého čtenáře počet vypůjčených knih.

l) Najděte čtenáře, kteří mají půjčeno více než 5 knih.

m) Najděte všechny tituly vydané v SR, Rusku a MR.

n) Najděte čísla čtenářů, kteří mají současně zapůjčeny nějaké knihy a rezervovány nějaké tituly do konce roku 1992.

o) Najděte jména čtenářů, kteří mají rezervovánu knihu Babička.

2. NEMOCNICE

a) Vytvořte tabulku podle schématu Pacient (id_pacient, jmeno, prijmeni, datnar, pojistovna) s následujícími integritními omezeními na datové typy: id_pacient – celočíselná nenulová hodnota – jednoznačný identifikátor – primární klíč, jmeno – jméno pacienta max 20 znaků, prijmeni – maximálně 20 znaků, pojišťovna – číslo pojištovny.

b) Přidejte k tabulce Pacient atribut povolání.

c) Změnte v tabulce Pacient rozsah atributu jmeno na 35 znaků.

d) Vložte 3 záznamy do tabulky Pacient.

e) Napište příkaz jazyka SQL pro výpis všech záznamů z tabulky Pacient

f) Napište příkaz na změnu pojišťovny na Všeobecnou (111) všem Pacientům se jménem Adam

g) Smažte tabulku Pacient.

h) Napište příkaz jazyka SQL pro výpočet počtu záznamů v tabulce Pacient

i) Vyhledejte všechny záznamy z tabulky Pacient, vypište pouze sloupce jmeno a prijmeni, setřiďte výsledek vzestupně podle atributu jmeno

j) Tentýž výpis jako v úloze i) setřiďte sestupně.

k) Vyhledejte všechny pacienty, kteří se narodili po datu 2.1.1990. Vypište pouze jméno a příjmení.

l) Vyhledejte všechny pacienty, kteří jsou narozeni mezi 2.1.1999 a 10.3.2000.

m) Vyhledejte všechny pacienty, kteří jsou narození mimo období 2.1.1999 a 10.3.2000.

n) Vyhledejte všechny pacienty, kteří mají pojišťovnu s číselným označením 111 (Všeobecná zdravotní pojištovna) nebo 201 (Vojenská zdravotní pojištovna).

o) Vypište všechna jména pacientů, jejichž jméno začíná na *a*.

p) Vypište jména všech lidí, kteří mají jméno začínající znakem ‘a‘, dál obsahuje dva libovolné znaky a končí zase znakem ‘a‘.

q) Zjistěte, zda-li je atribut titul u všech pacientů prázdný.

3. Mějme schéma Zamestnanec (id_zam, jmeno, prijmeni, datnar, funkce, plat). Vypište průměr, maximum a minimum platu zaměstnanců podle funkcí.

4. Tabulka Pacient má rozšířenou strukturu, máme další číselníky pojišťoven a států.

   Pacient ((id_pac, jmeno, prijmeni, datnar, krev_skup, rh_faktor, pojis, mesto, stat)

   Pojistovna (id_pojis, nazev_pojis)

   Stat (zkr_stat, nazev_stat)

   a) Vyhledejte všechny pacienty, kteří mají krevní skupinu 0 a k tomu rh faktor +. Vypište všechny atributy těchto pacientů.

   b) Vypište pro všechny pacienty jejich jméno a příjmení a název pojišťovny; setřiďte je podle pojišťoven a v rámci pojišťoven abecedně.

   c) Vypište pro všechny pacienty jejich jméno a příjmení, název pojišťovny a název státu; setřiďte je podle pojišťoven a v rámci pojišťoven abecedně.

   d) Vypište pacientky se jménem Anna - jejich příjmení a název pojišťovny; setřiďte je abecedně podle příjmení.

   e) Vypište pro všechny pacienty jejich jméno a příjmení a název pojišťovny; setřiďte je podle pojišťoven a v rámci pojišťoven abecedně. Vypište všechny existující pojišťovny - pokud pro některou pojišťovnu neexistují pacienti, bude jméno a příjmení pacienta prázdné.

5.3.3. Jazyk QBE

Jazyk QBE (Query By Example) byl původně vytvořen u firmy IBM v roce 1975 pro pohodlné **zadávání výběrových podmínek pro naivní uživatele**. Postupně se z něj vytvořil standard, užívaný u řady SŘBD, podobně jako jazyk SQL. Svými schopnostmi formulovat výběrové podmínky je stejně silným prostředkem, jako je jazyk SQL. V současnosti je nějaká jeho varianta součástí téměř všech SŘBD.

V tomto odstavci si uvedeme jen několik jednoduchých ukázek jeho dotazovacích možností.

Dotazy jsou vyjadřovány interaktivně pomocí příkladů (odtud název jazyka). Tabulky, z nichž se mají informace čerpat, se formou prázdných tabulkových schémat nebo formulářů zobrazují na obrazovce. Dotaz se zapíše tak, že do příslušných sloupců prázdné tabulky se vypíší ty hodnoty, které se ve sloupcích hledají.

Základní použití jazyka si ukážeme na příkladech.

Příklad:

Mějme tabulku zaměstnanců s osobním číslem a dalšími údaji

Zam(login, jméno, adresa, plat).

Po volbě tabulky se zobrazí prázdný řádek tabulky.

Zam	login	jméno	adresa	plat

nebo prázdný formulář

Zam	
osob	
jméno	
adresa	
plat	

♦

Do řádků tabulky se zapisují příkazy QBE, proměnné a výběrové podmínky. Proměnná začíná podtržítkem: _jmeno, _plat, _novak, _x, _y

Na rozdíl od SQL jazyk QBE podporuje odstranění duplicitních řádků výsledné relace. Zajištění všech řádků se naopak zajistí klauzulí ALL.

1. Výpis všech sloupců **celé tabulky** se zadá znakem P. (= print) pod jméno relace.

Příklad: *Vypište úplný seznam zaměstnanců.*

Zam	osob	jméno	adresa	plat
P.				

♦

2. Projekce (výběr sloupců) se zapíše znakem P. do příslušného sloupce, případně vyjádřením proměnné: P._Novák

Příklad: *Vypište jména a adresy všech zaměstnanců.*

Zam	osob	jméno	adresa	plat
		P.ALL._Novák	P._Opava	

_Novák *je proměnná, tedy stačí* _x, *užíváme různé proměnné v různých sloupcích*

Zam	osob	jméno	adresa	plat
		P.ALL._x	P._y	

Prostou projekci můžeme někdy zapsat i bez proměnné (záleží na implementaci)

Zam	osob	jméno	adresa	plat
		P.	P.	

♦

3. Selekce (výběr řádků) se zapíše hledanými hodnotami ve sloupcích s případným použitím relačního operátoru.

Příklad: *Vypište jména zaměstnanců bydlících v Opavě*

Zam	osob	jméno	adresa	plat
		P._x	Opava	

♦

4. Selekce a projekce současně.

Příklad: *Vypište osobní čísla a jména zaměstnanců s platem menším než 3500.*

Zam	osob	jméno	adresa	plat
	P.	P.		<3500

♦

5. Konjunkce více podmínek se píše do řádku vedle sebe.

Příklad: *Najděte zaměstnance mimo Ostravu s platem větším než 10000.*

Zam	osob	jméno	adresa	plat
		P.	<>Ostrava	>10000

♦

Složitější podmínky, jako více podmínek na jeden sloupec, disjunkce podmínek a další implementují různé SŘBD různě.

7. Disjunkce podmínek se píše do dvou a více řádků tabulky s použitím různých proměnných.

Příklad: *Najdi zaměstnance s platem pod 3000 nebo nad 30000.*

Zam	osob	jméno	adresa	plat
		P._x		<3000
		P._y		>30000

♦

8. Vzestupné **setřídění** výstupní relace se předepíše AO(n), sestupné DO(n) a v závorce zapsané číslo n znamená pořadí třídicího klíče.

Příklad: *Vypište zaměstnance dle výše platu - od nejvyššího k nejnižšímu, při stejném platu dle abecedy.*

Zam	osob	jméno	adresa	plat
	P.	P.AO(2)		P.DO(1)

♦

9. Pro dotaz vyžadující **spojení** dvou tabulek se vyvolají formuláře obou tabulek a podmínka spojení se vyjádří použitím stejné proměnné v obou tabulkách.

Příklad: *Je dána další relace* Děti (jménod, rodcis, osob). Osob *je cizí klíč ze* Zam. *Vypište adresy všech dětí (= adresy rodičů).*

Zam	osob	jméno	adresa	plat
	_x		P._z	

Děti	jménod	rodcis	osob
	P._y		_x

♦

10. Agregace používají funkcí CNT, SUM, AVG, MIN, MAX, **grupování** pomocí G.

Zam	kat	osob	jméno	adresa	plat
	P.G.				SUM._x

♦

Poznámka: Různé SŘBD mohou zobrazovat prázdný řádek tabulky i v jiném tvaru, často jako formulář.

Příklad: *Zaměstnanci z Opavy s platem nad 3000 se zadají:*

Zaměstnanci	
Osobní číslo:	
Jméno:	
Adresa:	**Opava**
Plat:	**> 3000**

♦

Dále existují možnosti formulace ještě složitějších podmínek, výpočtu agregovaných hodnot, pohledů, práce se systémovým katalogem, prostředků pro definici dat ap. Každý SŘBD má implementovánu jinou množinu prostředků a je nutné se vždy znovu s nimi a jejich zápisem seznámit.

Shrnutí pojmů 5.3.3.

Dotazy pomocí jazyka QBE.

Selekce, projekce, spojení, setřídění, agregace v QBE.

Otázky 5.3.3.

1. Co znamená QBE a jak tento jazyk vznikl?
2. Jaký je základní rozdíl mezi dotazem v SQL a QBE?
3. Jak se v jazyce QBE provede výběr informací odpovídající operacím projekce, selekce, spojení?
4. Jak se v jazyce QBE provede setřídění a agregace informací?

Úlohy k řešení 5.3.3.

1. Zapište pomocí jazyka QBE dotazy z úlohy 5.3.2., které lze zapsat pomocí probraných příkazů QBE.