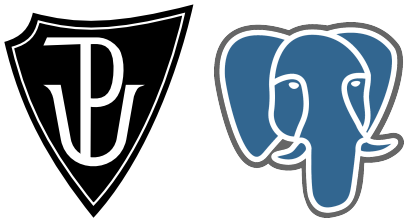
PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

PŘEPISY PŘEDNÁŠEK

Databázové systémy 1

KMI/DATA1



5. ledna 2013

Martin Rotter

Abstrakt

Tento dokument obsahuje přepisy přednášek, které vedl doc. RNDr Vilém Vychodil, Ph.D.. Uvedená práce (dílo) podléhá licenci Attribution-NonCommercial-NoDerivs 3.0 Unported, a to včetně vložených souborů, to neplatí pro loga jiných společností, jako je například logo PostgreSQL.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

1 Přehled databázových systémů a jejich historie

Databázový systém (anglicky Database Management System (DBMS)) je soustava komplexního aplikačního vybavení, které je podloženo určitým teoretickým základem. Jeden bez druhého nemůže existovat (resp. může, což má ale za následek formální selhání DBMS jako takového), a tak je nutné znát obě strany pomyslné databázové barikády. Databázový systém tedy tvoří:

1. Aplikační software, který je obvykle používán jako rozhraní pro přístup k databázi samotné.

2. Teorie, která formálně podkládá návrh databáze, organizaci dat a obsahuje algoritmickou stránku věci.

Mějme na paměti, že obě částí databázových systémů se rozvíjely postupně a mnohdy metodou pokus - omyl. Obecně platí, že pokud selže teoretický základ, tak již ani sebelepší frontend nic nezmůže. V databázovém systému se objeví formální rozpor a jedinou cestou vpřed je začít znovu.

Hlavním úkolem DBMS je poskytovat *perzistentní uložení dat*, dále poskytnout svým uživatelům konzistentní rozhraní a případně nabídnout *transakční zpracování dat*. Nutnto podotknout, že poslední bod nemusí být takovou samozřejmostí, jak by se mohlo zdát.

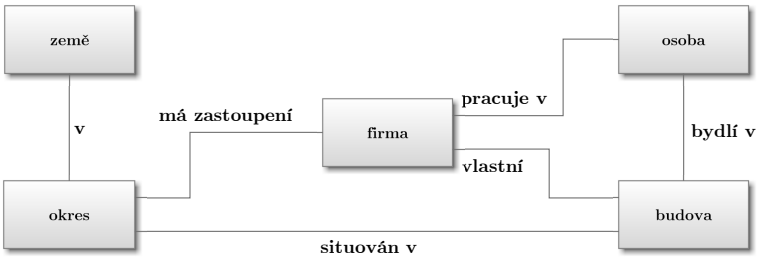
1.1 Historie databázových systémů

Potřeba organizace dat je stará jako lidstvo samo. Již staří Egypťané si vedli podrobné záznamy o výběru daní, stavbách chrámů a jiných činnostech. Zde můžeme hovořit o databázích založených na souborech.

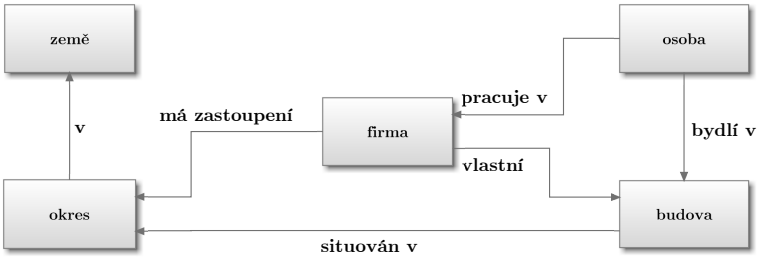
1.1.1 Databáze založené na souborech

Souborem může být například papyrus, hliněná destička nebo (lépe) papír. Na papíře mohou být napsány v řádcích nějaké záznamy. Například seznam dlužníků nějakého podnikatele. *Vytvoření* takového seznamu je vskutku *lehké*. Představme si, že dlužník uprostřed seznamu splatil svůj dluh a bude ze seznamu vyškrtnut. Místo něj v seznamu vznikne prázdné místo. Zbytek seznamu se následně musí zkonsolidovat (přepsat na nový papír), aby vypadal konzistentně. Odtud plyne určitá *těžkopádnost* vyplývající z definice souboru a z určité *nízkoúrovňovosti* práce s ním. Přitom souborem může být myšlen i soubor na disku počítače.

Představme si navíc, že daný podnikatel si vede další soubor se seznamem, kde si u každého dlužníka značí jeho adresu, aby jej mohl v případě nutnosti navštívit. Jméno dlužníka tedy uchovává hned na dvou seznamech, přitom je přirozeně jasné, že stačí dlužníka evidovat jednou a poté se na něj „odkazovat.“ *Redundace dat* je tedy zřejmá.



Obrázek 1: Datový model s obousměrnými odkazy



Obrázek 2: Datový model s jednosměrnými odkazy

1.1.2 Databáze založené na síťovém modelu

Průkopníkem této oblasti byl Charles Bachman. Aktuálně se toto paradigma považuje za dávno překonané, nicméně pro časy budoucí poskytlo několik dobrých poznatků. Typicky „síťový“ databázový systém je tvoří mj. i:

1. *Záznamy*, které mají určitý *typ*. Ten deklaruje, jaké *atributy* daný záznam obsahuje. Atributy mohou nabývat různých hodnot.

2. *Odkazy*, které reprezentují *vztahy* mezi jednotlivými záznamy.

Obecně se ví, že databázové systémy tohoto typu mohou poskytovat velmi rychlé prostředky pro získávání jednoduchých dat, ale na druhou stranu na komplexnější data se dotazuje hůře. Tvorba dat není rovněž jednoduchá, protože práce s odkazy požaduje vyšší míru sofistikace.

1.1.3 Databáze založené na hierarchickém modelu

Jedná se o speciální typ síťového modelu, kde bylo úmyslem tento model zjednodušit. Výsledkem bylo vytvoření hierarchie v záznamech a odkazech ve formě

stromu[1].

1.1.4 Databáze založené na objektovém modelu

V objektovém modelu jsou hlavní prvky, jenž reprezentují data, *objekty*, tedy serializované entity, které lze interpretovat pomocí nějakého vyššího programovacího jazyka. Prvním jazykem podporujícím takovéto pojetí byl Common Lisp.

1.1.5 Databáze založené na relačním modelu

Jedná se o aktuálně používaný model, který přináší určité výhody. Primárním nosičem informace v tomto modelu jsou *relace* (poněkud amatérsky je můžeme nazývat také tabulkami). Základy tohoto modelu položil Edgar Codd v roce 1970.

1.1.6 Nerelační databáze

Vyvíjeny v 90. letech 20. století. Někdy chybně značovaný jako „No SQL“. Rychlé pro jednoduché dotazy. Patří sem např. BerkeleyDB, MongoDB.

2 Relační model dat

Model má 3 základní komponenty:

1. *Struktury*, které uchovávají data a reprezentují výsledky dotazů.
2. *Integritní omezení*, která popisují vztahy mezi daty, které musí být splněny. Integritní omezení jsou v podstatě formule. Tato omezení slouží k popisu toho, jak mají data vypadat, slouži k odstranění jejich chyb.
3. *Manipulativní formalismy*, které říkají jak z uložených dat získat jiná data určitými operacemi.

Relační model je konkrétním modelem dat, avšak modely jako takové lze rozdělit do dvou skupin, tedy:

1. Abstraktní modely dat, které poskytují vyčerpávající logické definice datových struktur nebo operací s daty. Ty dohromady tvoří abstraktní formalismus. Tedy jakýsi abstraktní stroj, se kterým může uživatel operovat.
2. Konkrétní implementace modelů, pracující s persistentními daty. Od uživatele databáze se očekává dobrá znalost modelu dat, detailní znalost fyzické implementace není nutná.

[1]Strom je jednou ze základních grafových struktur. Jedná se o (neorientovaný) graf bez kružnic. Více napoví [1, str. 91 – 96].

Tabulka 1: Záhlaví tabulky

ID	JMÉNO	ADRESA
15	Vyacheslav Drsoň	Peklo, 666
⋮	⋮	⋮

3 Struktura relačních dat

Základním kamenem dat jsou tzv. „datové tabulky“, které obsahují záhlaví a další data.

1. Záhlaví (viz. tabulka 1) je množina atributů a jejich typů. U tabulky se seznamem zaměstnanců může být atributem například jméno zaměstnance atp.

2. Vlastní data (neboli tělo) tabulky by u takovéto tabulky představovala data samotných zaměstnanců.

3.1 Požadavky na tabulky

1. Řádky by neměly mít žádné pořadí. Tabulka by měla být chápána jako množina řádků. A v množině na pořadí nezáleží.

2. To samé platí pro sloupce.

3. Z definice množiny také vyplývá, že tabulka by neměla obsahovat identické řádky. Tento požadavek není v mnoha DBMS splněn.

4. V každém poli uvnitř tabulky je právě jedna hodnota daného typu.(SQL porušuje nedefinovanou hodnotou)

5. Všechny atributy jsou regulární.(Nejsou přítomné žádné skryté atributy.)

Tyto požadavky formuloval C. J. Date. Pokud tabulka splňuje body 1 až 5, pak je v 1. normální formě.

Zavedli jsme si několik pojmů jako *atribut* a tak dále. Formalizujme je nyní přesněji:

Atribut popisuje účel daného sloupce v tabulce, je to jméno sloupce.

Typ je jméno(označení) datového typu. Např.: INTEGER

Doména je množinou všech možných hodnot daného typu. Tedy například nějaký pomyslný typ `BOOL` by mohl mít doménu ve tvaru:

$$\mathrm{dom}(BOOL) = \{true, false\}$$

3.2 Relační schéma

Již dříve jsme si uvedli pojem *záhlaví tabulky*, avšak ten byl poněkud vágní. Relační schéma je jeho analogií s přesnější definicí.

Definice 3.1 (Relační schéma)**:** Mějme spočetnou množinu atributů Y a množinu typů T Následně platí, relační schéma R je množina definovaná jako:

$$R = \{\langle y, t\rangle \mid y \in Y, t \in T\}$$

Proveďme dohodu, že typ atributu y budeme zapisovat také jako typ(y) a analogicky doménu pro každý typ budeme značit jako dom(y).

3.3 Obecné kartézské součiny

Mějme následující systém množin:

$$\{A_i \mid i \in I\}$$

kde I je tzv. indexní (a libovolná) množina a i je index z této množiny. Kartézský součin $A_i (i \in I)$ se značí $\prod_{i\in I} A_i$ a je definován jako množina všech zobrazení

$$f : I \to \bigcup_{i\in I} A_i \text{ , kde } f(i) \in A_i \text{ pro } \forall i \in I$$

.

Prvky $\prod_{i\in I} A_i$ jsou zobrazení, nezáleží v nich na pořadí indexů.

Příklad 3.1 (Kartézský součin)**:** Vypočítejme kartézský součin množin A a B, tedy $A \times B$ se zadáním:

$$A = \{a, b, c\}, \quad B = \{10, 10\}, \quad I = \{1, 2\}, \quad A_1 = A, \quad A_2 = B$$

Následně řešení:

$$\prod_{i\in\{1,2\}} A_i = \{\langle a, 10\rangle, \langle b, 10\rangle . \langle a, 20\rangle, \langle b, 20\rangle\}$$ [2]

Příklad 3.2 (Kartézský součin s tečkovou indexní množinou)**:** Mějme $I = \{\bullet\}$ a $A_\bullet$. Pak platí, že $\prod_{i\in I} A_\bullet = A_\bullet$ s funkcí $f : \{\bullet\} \to A_\bullet$.

Příklad 3.3 (Kartézský součin žádné množiny)**:** Nechť $I = \varnothing$. Pak $\prod_{i\in I} A_i = \{\varnothing\}$ s funkcí $f : \varnothing \to \varnothing$.

[2] Zmíněnou rovnost nutno brát mírně s rezervou, protože formálně výsledkem kartézského součinu je množina zobrazení, nikoliv uspořádaných množin.

3.4 Relace nad relačním schématem

Definice 3.2 (Relace nad relačním schématem)**:** Mějme relační schéma $R \subseteq Y$, pak relace nad R je libovolná konečná podmnožina na kartézském součinu domén atributů z R, tedy

$$\mathcal{D} \subseteq \prod_{y \in R} \mathrm{dom}(y)$$

Relace v takovémto pojetí přibližně odpovídá vágnějšmu pojmu *tabulka*. Součin lze pochopitelně také rozepsat. Obsahuje-li relační schéma R celkem n atributů, pak platí, že:

$$\mathcal{D} \subseteq \mathrm{dom}(A_1) \times \cdots \times \mathrm{dom}(A_n)$$

Pozorný čtenář určitě pozoruje, že každá relace (tabulka) může mít určitý maximáln možný počet řádků. Matematicky:

$$|\mathcal{D}| = |dom(A_1)| \times \cdots \times |dom(A_n)|$$

Relaci lze také označit jako dvojici $\langle \mathcal{D}, R \rangle$. Tato dvojice takřak kompletně popisuje tabulku.

Každý prvek relace (tabulky) se nazývá n-tice. To je analogické vzhledem k matematickému pojetí relaci jako takových. Matematické relace též obsahují n-tice.

3.4.1 Speciální případy relací

1. Prázdná relace (tabulka) nad R, tedy $\langle \varnothing, R \rangle$.
2. Tabulka „dum“, tedy $\langle \varnothing, \varnothing \rangle$.
3. Tabulka „dee“, tedy $\langle \{\varnothing\}, \varnothing \rangle$.

4 Tutorial D

Programovací jazyk Tutorial D formuje opozici vůči mnohem známějšímu a v praxi používanějšímu Structured Query Language (SQL). Tutorial D má jedinou funkční a (víceménně) použitelnou implementaci známou jako Rel. Frontend Rel je naprogram v jazyce Java a jeho prostředí vypadá bohužel hrůzostrašně. Rel je silně staticky typovaný, nemá konverze typů. Obsahuje typy:

Skalární , které jsou opozitem hodnotových typů ze známějších jazyků a patří sem například `BOOLEAN`, `INTEGER` nebo `STRING` a také uživatelsky definované typy.

N-ticové , které jsou dané jmény atributů a jejich typy.

Relační , které jsou dány atributy a jejich typy. Hodnoty představují relace, t.j. množiny n-tic.

4.1 Hodnoty vs. proměnné

Hodnoty - jednotlivé konstanty, relace a n-tice. Nemohou se měnit - „Jako ve Scheme“

Proměnné - ukazatele na reprezentace hodnot. Mají přiřazený typ, mohou měnit svůj odkaz.

4.2 Výrazy versus příkazy

Výrazem v Tutorial D (resp. v Rel) je například prostá logická rovnost hodnot. Výraz může mít nějaký výsledek. Příkaz se od výrazu liší tak, že končí středníkem a obvykle obsahuje volání nějaké funkce.

Zdrojový kód 1: Výraz a příkaz (Tutorial D)

```
/* výraz */
666 = 6661215
/* příkaz */
WRITELN(10 = 20);
```

4.3 N-tice jako hodnoty

V Tutorial D lze přímočaře vytváře řádky relací a považovat je za hodnoty, viz. kód 2.

Zdrojový kód 2: Základy n-tic (Tutorial D)

```
/* prázdná n-tice */
TUPLE {}
/* n-tice včetně dat */
TUPLE {id 666, name "Vilík", lab 5077}
/* rovnost n-tic */
WRITELN(TUPLE {id 666, name "Vilík", lab 5077} = TUPLE {id 007, name
    "James␣Bond", lab 1111});
/* vnořené n-tice */
TUPLE {id 666, TUPLE {jmeno "Vilík" prijmeni "Devil"}, lab 5077}
```

Z n-tic lze pochopitelně získávat hodnoty nebo n-tice sjednocovat, zúžovat a tak podobně.

Zdrojový kód 3: Operace s n-ticemi (Tutorial D)

```
/* vypíše hodnoty atributů id a lab (zúžení n-tice) */
TUPLE {id 666, TUPLE {jmeno "Vilík" prijmeni "Devil"}, lab 5077} {id
    , lab}
/* vypíše vše kromě lab (zúžení n-tice) */
TUPLE {id 666, TUPLE {jmeno "Vilík" prijmeni "Devil"}, lab 5077} {
    ALL BUT lab}

/* sjednocení n-tic */
TUPLE {id 666, jmeno "Vilík"} UNION TUPLE {jmeno "Vilík", lab 5077}

/* přejmenování atributů n-tice */
```

```
10 TUPLE {id 666, jmeno "Vilík"} RENAME (id AS cislo, jmeno AS name)
```

Sjednocení n-tic bere dvě n-tice, které se shodují na nějakém atributu a ve výsledku se objeví konkatenace n-tic s jedním výskytem společného atributu. Kompozice n-tic funguje obdobně jako sjednocení, avšak ve výsledku se společné atributy vůbec neobjeví.

Zdrojový kód 4: Další operace s n-ticemi (Tutorial D)

```
1 /* rozšíření n-tice */
2 EXTEND TUPLE {id 666, jmeno "Vilík"} ADD (5077 AS lab)
3 /* aktualizace hodnot v n-tici */
4 UPDATE TUPLE {id 666, jmeno "Vilík"} (jmeno := "God")
```

4.4 Relace v Tutorial D

Tutorial D (potažmo Rel) samozřejmě podpruje relace.

Zdrojový kód 5: Operace s relacemi (Tutorial D)

```
1 /* relace s několika n-ticemi*/
2 RELATION {
3     TUPLE {id 666, jmeno "Vilík"},
4     TUPLE {id 007, jmeno "Bond"}
5 }
6 /* vytvoření relace jakožto typu */
7 VAR seznam_lidi BASE RELATION {id INTEGER, jmeno STRING} KEY {id};
```

5 Modifikace dat

Již známe pojmy jako relace nebo relační proměnná. Modifikací dat v databázi se myslí přiřazení nové hodnoty k nějaké relační proměnné. V SQL se k tomuto používají příkazy `INSERT`, `UPDATE` nebo `DELETE`. V Tutorial D k obdobným operacím slouží operátor „:=“, tak jak jej známe například z jazyka Pascal.

Zdrojový kód 6: Modifikace relace (SQL)

```
1 UPDATE  zakaznici
2 SET     plat = 0
3 WHERE   (plat > 50000);
```

Zdrojový kód 7: Modifikace relace (Tutorial D)

```
1 INSERT zakaznici RELATION {TUPLE {id 666, jmeno "Vilík", }};
```

K získávání dat z databáze slouží tzv. *relační dotazování*. Formátování dotazů pak definuje *dotazovací jazyk*. Ten říká, jak se budou dotazy vyhodnocovat. Tyto jazyky jsou obvykle deklarativní. Známe například SQL a Tutorial D.

Relační algebra specifikuje množinu operací s relacemmi a dotazy se skládají z postupné aplikace těchto operací. Dotazy se formulují pomocí termů a vyhodnocování dotazů odpovídá vyhodnocování termů v algebře. Vyhodnocování může být přímočaré, avšak u složitějších dotazů rovněž netriviální.

Relační kalkuly (výpočty nad relacemi) existují hned v několika variantách:

- doménový kalkul
- n-ticový kalkul

Dotaz je formule predikátové logiky, ve které relační symboly označují relační proměnné a vyhodnocování dotazu je ohodnocování formulí v dané predikátové struktuře (ta představuje instanci databáze), ve které jsou relační symboly interpretovány n-árními relacemi.

Doménový relační kalkul má zhruba stejnou sílu jako relační algebra.

6 Relační algebra

Relační algebra poskytuje formální podklad pro množinové relační operace. Ty jsou analogií ke standardním operacím na množinách.

1. *Průnik*, který obsahuje pouze společné n-tice dvou relací. Mějme tedy relace $\mathcal{D}_1$ a $\mathcal{D}_2$ nad relačním schématem $R \subseteq Y$. Následně průnik definujeme jako:

$$\mathcal{D}_1 \cap \mathcal{D}_2 = \left\{ r \in \prod_{y \in R} dim(y) \mid r \in \mathcal{D}_1 \text{ a zároveň } r \in \mathcal{D}_2 \right\}$$

2. *Sjednoceni*, které obsahuje ty n-tice, které se vyskytnou alespoň v jedné ze vstupních relací. Mějme tedy relace $\mathcal{D}_1$ a $\mathcal{D}_2$ nad relačním schématem $R \subseteq Y$. Následně sjednocení definujeme jako:

$$\mathcal{D}_1 \cup \mathcal{D}_2 = \left\{ r \in \prod_{y \in R} dim(y) \mid r \in \mathcal{D}_1 \text{ nebo } r \in \mathcal{D}_2 \right\}$$

3. *Rozdíl*, který obsahuje n-tice, které se nacházejí v první relaci, avšak nenacházejí se relaci druhé. Mějme tedy relace $\mathcal{D}_1$ a $\mathcal{D}_2$ nad relačním schématem $R \subseteq Y$. Následně rozdíl definujeme jako:

$$\mathcal{D}_1 \setminus \mathcal{D}_2 = \left\{ r \in \prod_{y \in R} dim(y) \mid r \in \mathcal{D}_1 \text{ a zároveň } r \notin \mathcal{D}_2 \right\}$$

Tyto operace lze provádět také v SQL či v Tutorial D.

Zdrojový kód 8: Relační operace (Tutorial D)

```
1 rexpr1 INTERSECT /* nebo UNION či MINUS */ rexpr2
```

Zdrojový kód 9: Relační operace (SQL)

```sql
SELECT * FROM table1
    INTERSECT /* nebo UNION či EXCEPT */
SELECT * FROM table2;
```

Množinové operace v SQL používají implicitně volbu `DISTINCT`, takže duplicitní n-tice jsou ignorovány. Naproti tomu u operací typu `SELECT` se jako výchozí používá `ALL`.

6.1 Efektivita množinových operací

Efektivitá závisí na implementaci fyzické vrstvy databázového systému. Obecně platí, že pokud lze na množině potřebných n-tic zavést lineární uspořádání, tak lze množinové operace provádět pomocí slévání.

Je třeba aby dané uspořádání bylo navíc totální. Tedy každé dvě n-tice musejí být porovnatelné. Relace uspořádání musí být tranzitivní, symetrická a reflexivní. Z požadavku totality musí být uspořádání také úplné.

Pokud máme na každé doméně zavedeno toto uspořádání $\leq_y$ a zavedeme uspořádání také pro dané relační schéma R, tedy $\leq_R$, pak nám všechna uspořádání

$$\leq_y (y \in R) \text{ a } \leq_R$$

indukují uspořádání na n-ticích $r_1 < r_2$ právě tehdy, když existuje atribut $y \in R$ tak, že

$$r_1(z) = r_2(z) \text{ pro každé } z <_R y \text{ a navíc } r_1(y) <_y r_2(y)$$

V SQL se efektivita zajišťuje použitím tzv. indexu. Index dokáže provést ono totální uspořádání.

Zdrojový kód 10: Tvorba indexu (SQL)

```sql
CREATE INDEX muj_index ON moje_tabulka (sloupec_1, sloupec_2);
```

6.2 Další pojmy

Je třeba vysvětlit několik dalších pojmů:

Nadklíč (superkey) Mějme relaci $\mathcal{D}$ nad relačním schématem R, která obsahuje několik n-tic. $K \subseteq R$ se nazývá *nadklíč*, pokud pro každé n-tice r_1,$r_2 \in \mathcal{D}$ platí: Pokud jsou si r_1 a r_2 rovny na všech atributech z K, pak si jsou rovny na všech atributech z R. Triviálním nadklíčem je $K = R$.

Klíč (key) Mějme libovolný nadklíč a relaci $\mathcal{D}$. Takový nadklíč může obsahovat „nadbytečné“ prvky resp. atributy. Jako *klíč* K označíme nadklíč takový, že odstraněním jakéhokoliv dalšího atributu z klíče K by vedlo k tomu, že K již nebude nadklíč. Tedy:

1. Pro každé různé n-tice musí platit, že nemohou mít shodné hodnoty na (všech) atributech z klíče.
2. Klíč je minimálním nadklíčem.

6.3 Relační operace

6.3.1 Projekce

Mějme relaci $\mathcal{D}$ na schématu T. Projekce z $\mathcal{D}$ přes $R \subseteq T$ je relace $\pi_R(\mathcal{D})$, definovaná jako:

$$\pi_R(\mathcal{D}) = \left\{ r \in \prod_{y \in R} dom(y) \mid \text{existuje } s \in \prod_{y \in T \setminus R} dom(y) \text{ tak, že } rs \in \mathcal{D} \right\}$$

kde rs znázorňuje (určitý druh) zřetězení n-tic, kde část r je nad schématem R a s je nad schématem $T \setminus R$.

V Tutorial D lze n-tice řetězit prostřednictvým klíčového slova `COMPOSE`.

Zdrojový kód 11: Projekce (SQL)

```sql
/* část id, jmeno, plat je projekcí */
SELECT DISTINCT id, jmeno, plat FROM table_1;
```

Zdrojový kód 12: Projekce (Tutorial D)

```
/* část {id} je projekcí */
TUPLE {id 666, jmeno "Vilík"} {id}
/* vše kromě id */
TUPLE {id 666, jmeno "Vilík"} {ALL BUT id}
```

Mějme na vědomí, že bez `DISTINCT` se ze zřejmých důvodů nejedná o relační operaci.

6.3.1.1 Aspekty fyzické vrstvy

Projekce v SQL přes `DISTINCT` je rychlá operace. V případě užití `DISTINCT` se už o rychlou operaci jednat nemusí. To ale neplatí v případě, že $R \subseteq T$ obsahuje klíč. Projekcí umí zrychlit:

1. Hashování, kde se potencíální n-tice, které se objeví ve výsledku, hashují.
2. Použití totálního uspořádání n-tic.

Věta 6.1: Pokud má-me $S \subseteq R \subseteq T$, pak

$$\pi_S(\pi_R(\mathcal{D})) = \pi_S(\mathcal{D})$$
$$\pi_T(\mathcal{D}) = \mathcal{D}$$
$$\pi_R(\mathcal{D}) = \begin{cases} \varnothing & \text{pokud } \mathcal{D} = \varnothing \\ \{\varnothing\} & \text{pokud } \mathcal{D} \neq \varnothing \end{cases}$$

6.3.2 Restrikce (selekce)

Mějme danou relaci $\mathcal{D}$ a formulí popisující podmínku $\varphi : t_1 \approx t_2$. t_1 je atribut a t_2 je hodnota z domény atributu. Následně restrikce:

$$\sigma_\varphi(\mathcal{D}) = \{t \in \mathcal{D} \mid t \text{ splňuje } \varphi\}$$

Zdrojový kód 13: Restrikce (SQL)

```sql
/* část id = 666 je restrikcí */
SELECT * FROM table_1 WHERE id = 666;
```

Zdrojový kód 14: Restrikce (Tutorial D)

```
/* část id = 666 je restrikcí */
relexpr WHERE id = 666;
```

Věta 6.2 (O komutativitě restrikcí)**:** Platí, že:

$$\sigma_{\varphi}(\sigma_{\psi}(\mathcal{D})) = \sigma_{\psi}(\sigma_{\varphi}(\mathcal{D})) = \sigma_{\varphi\&\psi}(\mathcal{D})$$

6.3.2.1 Záměna pořadí projekce a restrikce

Prohlašme, že $\pi_R(\sigma_{\varphi}(\mathcal{D})) \backsim \sigma_{\varphi}(\pi_R(\mathcal{D}))$. Pokud platí, že φ závisí na některém atributu z $T \setminus R$, pak pravá strana nedává smysl.

Předchozí poznatek má použití, a sice duální operaci k projekci, tedy přidání atributů.

Zdrojový kód 15: Přidání atributů (SQL)

```sql
SELECT table_1.*, expr AS additional_column FROM table_1;
```

Zdrojový kód 16: Přidání atributů (Tutorial D)

```
EXTEND relexpr ADD (expr AS additional_column)
```

Zdrojový kód 17: Kombinace restrikce a duality projekce (SQL)

```sql
SELECT   money * 2 AS doubled_money, money, id
FROM     employees
WHERE    money * 2 > 5000;
```

Zdrojový kód 18: Kombinace restrikce a duality projekce (Tutorial D)

```
((EXTEND relexpr ADD (money * 2 AS doubled_money))
WHERE doubled_money > 5000) {doubled_money, money, id}
```

6.3.3 Sumarizace (agregace)

Zdrojový kód 19: Počet n-tic v relaci (SQL)

```sql
SELECT COUNT(*) FROM my_table;
```

Výsledkem obdobného dotazu v SQL bude vždy relace. Například v tomto případě relace o jedné jednoprvkové n-tici, která obsahuje počet řádků původní relace.

Zdrojový kód 20: Počet n-tic v relaci (Tutorial D)

```
COUNT(relexpr)
```

Naopak v Tutorial D nebude výsledkem relace, avšak skalární hodnota. Tedy jednoduše číslo. Chování jazyka SQL lze v Tutorial D imitovat.

Zdrojový kód 21: Imitace chování SQL (Tutorial D)

```
SUMMARIZE relexpr ADD (COUNT() AS count_of_tuples)
```

Zdrojový kód 22: Pokročilá sumarizace (Tutorial D)

```
SUMMARIZE relexpr1
PER (relexpr2)
ADD (summary AS name)

/* místo PER lze použít také BY */
/* platí, že BY {seznam atributů} odpovídá PER (relexpr1 {seznam
    atributů}) */
SUMMARIZE relexpr1
BY (seznam atributů)
ADD (summary AS name)
```

Navíc by mělo platit, že `relexpr1` $\subseteq$ `relexpr2`. Význam zdrojového kódu 22 je takový, že jdeme přes všechny n-tice z `relexpr2` a počítáme sumarizaci ze všech n-tic z `relexpr1`, které se shodují na atributech z relačního schématu z `relexpr2`.

6.3.4 Seskupování

Uplatňuje se v modelech, které umožňují atributy s doménami, které jsou množiny relací.

6.3.4.1 Metoda UNGROUP

Zdrojový kód 23: Seskupování UNGROUP (Tutorial D)

```
/* atributy jsou typu relace */
relexpr UNGROUP (atribut, dalsi_atribut)
```

Výsledkem je relace nad schématem, které vznikne tak, že místo atributů ze seznamu budeme mít atributty vnořených tabulek a výsledek obsahuje n-tice tak, že pro každou n-tici je ve výsledku (obecně) několik n-tic, jejichž hodnoty jsou brány z tabulek ve výchozí n-tici metodou každý s každým.

6.3.4.2 Metoda GROUP

Zdrojový kód 24: Seskupování GROUP (Tutorial D)

```
/* atributy jsou typu relace */
relexpr GROUP (atribut, dalsi_atribut)
```

Na základě relace a seznamu atributů množin atributů, které se mají seskupit pod danými novými jmény se vytvoří relace, ve které jsou zbylé atributy výchozí relace a nové atributy, které nabývají hodnot, v nichž jsou maximální relace obsahující n.tice hodnot výchozí tabulky tak, že individuální hodnoty výsledných n-tic jsou v relaci se vzniklými relacemi právě tehdy, když zřetězení libovolných individuálních n-tic se zbylými individuálními hodnotami z každé výsledné n-tice se nachází ve výchozí tabulce.

6.3.5 Přejmenování

Z relace $\mathcal{D}$ nad relačním schématem R se vytvoří nová relace

$$\rho_{\,y'_1 \Leftarrow y_1, \ldots, y'_n \Leftarrow y_n}(\mathcal{D})$$

nad schématem R, ve kterém byly atributy $y_1, \ldots, y_n$ přejmenovány na atributy $y'_1, \ldots, y'_n$, ale za předpokladu, že každé dva atributy y_i, y'_i pro $1 \leq i \leq n$ mají stejný typ.

Zdrojový kód 25: Přejmenování (Tutorial D)

```
relexpr RENAME (old AS new, old_2 AS new_2))
```

Zdrojový kód 26: Přejmenování (SQL)

```
SELECT old AS new, old_2 AS new_2 FROM table_1;
```

6.3.6 Přirozené spojení

Definice 6.1 (Přirozené spojení)**:** Mějme relace $\mathcal{D}_1$ nad schématem $R \cup S$ a $\mathcal{D}_2$ nad schématem $S \cup T$ tak, že R, S, T jsou po dvou disjunktní a S je množina všech společných atributů. Pak je přirozené spojení $\mathcal{D}_1, \mathcal{D}_2$ relace nad schématem $R \cup S \cup T$ dáno předpisem

$$\mathcal{D}_1 \bowtie \mathcal{D}_2 = \{rst \mid rs \in \mathcal{D}_1, st \in \mathcal{D}_2\}$$

Zdrojový kód 27: Sjednocení n-tic (Tutorial D)

```
/* výsledkem je TUPLE {x 10, y 20, z 30} */
TUPLE {x 10, y 20} UNION TUPLE {y 20, z 30}
```

Příklad 6.1 (Přirozené spojení)**:** Přírozeným spojením získáme tabulku, která má společné záhlaví, avšak obsahuje pouze n-tice, které se shodují na společných

Tabulka 2: Přirozené spojení tabulek

(a) První operand

w	x	y
1	1	2
2	2	2
3	3	2
4	3	4
5	5	5

(b) Druhý operand

x	y	z
1	1	2
2	2	1
3	2	4
4	4	5
5	5	5

(c) Výsledná tabulka

w	x	y	z
2	2	2	1
3	3	2	4
5	5	5	5

atributech daných vstupních tabulek resp. relací. Názorně na tabulkách ve skupině tabulek 2.

Libovolné n-tice r nad schématem R a s nad schématem S nazveme *spojitelné*, pokud projekce $r(R \cap S) = s(R \cap S)$. Tedy pokud mají stejné hodnoty ve společných atributech. Přirozené spojení odpovídá množině všech spojitelných n-tic z nějakých vstupních relací $\mathcal{D}_1, \mathcal{D}_2$.

Zdrojový kód 28: Přirozené spojení (Tutorial D)

```
relexpr1 JOIN relexpr2
```

Zdrojový kód 29: Přirozené spojení (SQL)

```
SELECT * from table_1 NATURAL JOIN table_2;

/* případně */
SELECT  table_1.*, table_2.*
FROM    table_1, table_2
WHERE   table_1.x = table_2.x AND table_1.y = table_2.y;
```

Definice 6.2 (Visící n-tice)**:** Pokud uvažujeme spojení $\mathcal{D}_1 \bowtie \mathcal{D}_1$, pak se $rs \in \mathcal{D}_1$ nazývá *visící* n-tice, pokud $rs \notin \pi_{R \cup S}(\mathcal{D}_1 \bowtie \mathcal{D}_2)$, tedy právě, když rs není spojitelné se žádnou n-ticí z $\mathcal{D}_2$.

Definice 6.3 (Úplné přirozené spojení)**:** Pokud relace $\mathcal{D}_1, \mathcal{D}_2$ nemají vzhledem ke svému přirozenému spojení žádné visící n-tice, pak lze toto přirozené spojení označit jako *úplné*.

6.3.6.1 Speciální případy přirozeného spojení

Předpokládejme, že $\mathcal{D}_1$ je relace nad schématem $R \cup S$ a $\mathcal{D}_2$ je relace nad schématem $S \cup T$. Schémata R, S, T jsou pod dvou disjunktní.

Průnik Jedná se o speciální případ pro $R = \varnothing$ a $T = \varnothing$, pak obě relace $\mathcal{D}_1$ a $\mathcal{D}_2$

jsou situovány nad schématem S. Následně spojení:

$$\begin{aligned}\mathcal{D}_1 \bowtie \mathcal{D}_2 &= \{rst \mid rs \in \mathcal{D}_1 \text{ a } st \in \mathcal{D}_2\} \\ &= \{\varnothing s\varnothing \mid \varnothing \in \mathcal{D}_1 \text{ a } s\varnothing \in \mathcal{D}_2\} \\ &= \{s \mid s \in \mathcal{D}_1 \text{ a } s \in \mathcal{D}_2\} \\ &= \mathcal{D}_1 \cap \mathcal{D}_2\end{aligned}$$

Kartézský součin Jedná se o spojení, kde společné schéma (seznam atributů) $S = \varnothing$. Následně:

$$\begin{aligned}\mathcal{D}_1 \bowtie \mathcal{D}_2 &= \{rst \mid rs \in \mathcal{D}_1 \text{ a } st \in \mathcal{D}_2\} \\ &= \{rt \mid r \in \mathcal{D}_1 \text{ a } t \in \mathcal{D}_2\}\end{aligned}$$

Tento případ spojení značíme $\boxtimes$. Velikost výsledku tohoto spojení, tedy $|\mathcal{D}_1 \boxtimes \mathcal{D}_2| = |\mathcal{D}_1| \cdot |\mathcal{D}_2|$.

$\boxtimes$ tedy označuje množinu všech spojení tohoto typu. V Tutorial D nemá tato operace žádnou analogii. Dá se udělat pomocí JOIN a přejmenování atributů. V SQL se používá klíčové slovo `CROSS JOIN`.

Polospojení Značí se $\ltimes$. Tedy $\mathcal{D}_1 \ltimes \mathcal{D}_2 = \pi_{R \cup S}(\mathcal{D}_1 \bowtie \mathcal{D}_2)$. Výsledek obsahuje takové n-tice $rs \in \mathcal{D}_1 \ltimes \mathcal{D}_2$, pro které platí, že jsou spojitelné s alespoň jednou n-ticí z $\mathcal{D}_2$.

Zdrojový kód 30: Polospojení (SQL)

```sql
SELECT DISTINCT table_1.* FROM table_1 NATURAL JOIN table_2
```

Zdrojový kód 31: Polospojení (Tutorial D)

```
relexpr1 MATCHING relexpr2
```

Pokud je $\begin{cases} T = \varnothing & \text{pak } \mathcal{D}_1 \ltimes \mathcal{D}_2 = \mathcal{D}_1 \bowtie \mathcal{D}_2 \\ R = \varnothing & \text{pak } \mathcal{D}_2 \ltimes \mathcal{D}_1 = \mathcal{D}_1 \bowtie \mathcal{D}_2 \end{cases}$

Restrikce (na rovnost) $\sigma_{y \approx d}(\mathcal{D}) = \mathcal{D} \bowtie \{\{\langle y, d \rangle\}\}$, kde $\langle y, d \rangle$ je n-tice nad schématem $R = \{y\}$ a $f(y) = d$.

Zdrojový kód 32: Restrikce na rovnost (Tutorial D)

```
relexpr1 WHERE y = d
/* je ekvivalentní s */
relexpr1 JOIN RELATION {TUPLE {y d}}
```

Kompozice binárních relací Klasické kompozice z pohledu matematických relací se definuje jako:

$$R_1 \circ R_2 = \{\langle a, b \rangle \mid \exists c \text{ tak, že } \langle a, c \rangle \in R_1 \text{ a zároveň } \langle c, b \rangle \in R_2\}$$

Naproti tomu kompozice s přihlédnutím k relacím nad určitým relačním schématem se definuje jako:

$$\mathcal{D}_1 \circledcirc \mathcal{D}_2 = \pi_{T \cup R}(\mathcal{D}_1 \bowtie \mathcal{D}_2)$$

kde relační schémata pro $\mathcal{D}_1, \mathcal{D}_2$ jsou $\{x, y\}$ resp. $\{y, z\}$.

6.3.6.2 Vlastnosti přirozeného spojení

Věta 6.3 (Komutativita, asociativita, idempotence, neutralita a anihilace přirozeného spojení)**:** Platí, že:

$$\begin{aligned}
\mathcal{D}_1 \bowtie \mathcal{D}_2 &= \mathcal{D}_2 \bowtie \mathcal{D}_1 && \text{(Komutativita)} \\
\mathcal{D}_1 \bowtie (\mathcal{D}_2 \bowtie \mathcal{D}_3) &= (\mathcal{D}_1 \bowtie \mathcal{D}_2) \bowtie \mathcal{D}_3 && \text{(Asociativita)} \\
\mathcal{D} \bowtie \mathcal{D} &= \mathcal{D} && \text{(Idempotence)} \\
\mathcal{D} \bowtie \{\varnothing\} &= \mathcal{D} && \text{(Neutralita vůči množině obs. } \varnothing) \\
\mathcal{D} \bowtie \varnothing &= \varnothing && \text{(Anihilace vůči prázdné množině)}
\end{aligned}$$

Věta 6.4 (Další vlastnosti spojení)**:** Mějme relace $\mathcal{D}_i$ na schématech $R_j (j = 1, \ldots, n)$ pak platí:

1. $\pi_{R_j}(\bowtie_{i=1}^{n} \mathcal{D}_i) \subseteq \mathcal{D}_j, \quad \forall j = 1, \ldots, n$
2. $\pi_{R_j}(\bowtie_{i=1}^{n} \mathcal{D}_i) = \mathcal{D}_j, \quad \forall j = 1, \ldots, n$ právě, když lze relace $\mathcal{D}_1, \ldots, \mathcal{D}_n$ úplně spojit.
3. $\pi_{R_j}(\bowtie_{k=1}^{n} \pi_{R_j}(\bowtie_{i=1}^{n} \mathcal{D}_i)) = \pi_{R_j}(\bowtie_{i=1}^{n} \mathcal{D}_i), \quad j = 1, \ldots, n$

Mějme $\mathcal{D}$ na schématu $R_1 \cup \cdots \cup R_n$. Následně platí:

4. $\mathcal{D} \subseteq \bowtie_{i=1}^{n} \pi_{R_i}(\mathcal{D})$
5. $\bowtie_{i=1}^{n} \pi_{R_i}(\bowtie_{j=1}^{n} \pi_{R_j}(\mathcal{D})) = \bowtie_{i=1}^{n} \pi_{R_i}(\mathcal{D})$

Definice 6.4 (Bezztrátová dekompozice)**:** Relace $\mathcal{D}$ na schématu R má bezztrátovou dekompozici, pokud

$$\mathcal{D} = \bowtie_{i=1}^{n} \pi_{R_i}(\mathcal{D}) \text{ pro } R_1 \cup \cdots \cup R_n = R$$

Poznámka 6.1 (O bezztrátové dekompozici)**:** Relační schéma R má vždy bezztrátovou dekompozici vzhledem k R.

6.3.7 Spojení na rovnost a Théta-spojení

Spojuje data z tabulek (relací) na základě specifikovaného vztahu, který musí splňovat hodnoty atributů.

Mějme dáno $\mathcal{D}_1, \mathcal{D}_2$ a obecnou *Theta*-podmínku, jenž je dána obecně nějakou formulí. Pak výsledkem bude podmnožina

$$\mathcal{D}_1 \bowtie_{\Theta} \mathcal{D}_2 = \sigma_{\Theta}(\mathcal{D}_1 \boxtimes \mathcal{D}_2)$$

kde σ_Θ je restrikcí na základě podmínky Θ.

Spojení na rovnost je speciálním případem pro Θ-spojení pro Θ ve tvaru:

$$x_1 = y_2 \wedge \cdots \wedge x_n = y_n$$

6.3.7.1 Vztah mezi přirozeným spojením a spojením na rovnost

Spojení na rovnost lze vyjádřit pomocí přirozeného spojení a restrikcí následovně:

$$\mathcal{D}_1 \bowtie_{x_1=y_1 \wedge \cdots \wedge x_n=y_n} \mathcal{D}_2 = \sigma_{x_1=y_1}(\sigma_{x_2=y_2}(\ldots(\sigma_{x_n=y_n}(\mathcal{D}_1 \bowtie \mathcal{D}_2))\ldots))$$

Přirozené spojení lze vyjádřit pomocí restrikcí, přejmenování a projekcí. Mějme $\mathcal{D}_1$ na $R \cup S$ a $\mathcal{D}_2$ na $S \cup T$. R, S, T jsou po dvou disjunktní pro $S = \{y_1, \ldots, y_n\}$. Bereme $\mathcal{D}_2$ a přejmenujeme atributy. Tedy:

$$\pi_{R \cup S \cup T}(\rho_{y'_1 \Leftarrow y_1, \ldots, y'_n \Leftarrow y_n}(\mathcal{D}_2) \bowtie_{y_1=y'_1 \wedge \cdots \wedge y_n=y'_n} \mathcal{D}_1) = \mathcal{D}_1 \bowtie \mathcal{D}_2$$

Zdrojový kód 33: Vyjádření přirozeného spojení pomocí dalších operací (Tutorial D)

```
(relexpr1 JOIN (relexpr2 RENAME (...))) WHERE ...
```

Zdrojový kód 34: Vyjádření přirozeného spojení pomocí dalších operací (SQL)

```
SELECT * FROM table_1, table_2 WHERE table_1.* = table_2.*;
```

6.3.8 Relační dělení

Dělení je protípólem projekce a značí se „$\cong$“. Bývá odpovědí na dotazu typu: „Najdi osobu, která obsolvovala každý kurz.“ Určitá n-tice r se nachází v tabulce $r \in \pi_R(\mathcal{D})$ právě, když existuje s tak, že $rs \in \mathcal{D}$. Dělení lze interpretovat jako odpověď na univerzální dotazy.

Uvažujme relace:

1. $\mathcal{D}_1$ na schématu R, které říkejme *dělenec*.
2. $\mathcal{D}_2$ na schématu S, které říkejme *dělitel*.
3. $\mathcal{D}_3$ na schématu T, které říkejme *prostředník*.

Výsledkem dělení $\mathcal{D}_1 \div_{\mathcal{D}_3} \mathcal{D}_2$, tedy tzv. podílem je relace nad schématem $R \cap T$. Matematicky:

$$\begin{aligned}\mathcal{D}_1 \div_{\mathcal{D}_3} \mathcal{D}_2 = \{&r(R \cap T) \mid r \in \mathcal{D}_1 \text{ tak, že } \forall s \in \mathcal{D}_2 \text{ takové, že } r(R \cap S \cap T) \\ &= s(R \cap S \cap T) \text{ platí, že } (rs)((R \cup S) \cap T) \in \pi_{(R \cup S) \cap T}(\mathcal{D}_3)\}\end{aligned}$$

Pokud $R \cap S = \varnothing$, pak platí, že:

$$\begin{aligned}\mathcal{D}_1 \div_{\mathcal{D}_3} \mathcal{D}_2 = \{&r(R \cap T \mid r \in \mathcal{D}_1 \text{ tak, že pro } \forall s \in \mathcal{D}_2 \text{ platí, že} \\ &(rs)\,((R \cup S) \cap T) \in \pi_{(R \cup S) \cap T}(\mathcal{D}_3\}\end{aligned}$$

Tabulka 3: Logické funkce

(a) Logický součin

$\wedge$	0	N	1
0	0	0	0
N	0	N	N
1	0	N	1

(b) Logická implikace

$\rightarrow$	0	N	1
0	1	1	1
N	N	N	1
1	0	N	1

Dělení trošku jinak: Pro $T = R \cup S$

$$\mathcal{D}_1 \div_{\mathcal{D}_3} \mathcal{D}_2 = \{r \mid r \in \mathcal{D}_1 \text{ tak, že pro } \forall s \in \mathcal{D}_2 \text{ platí, že } rs \in \mathcal{D}_3\}$$

Nebo také: (Vhodné pro vyjádření v SQL)

$$\mathcal{D}_1 \div_{\mathcal{D}_3} \mathcal{D}_2 = \pi_{R \cap T}(\mathcal{D}_1) \setminus \pi_{R \cap T}\left((\pi_{R \cap T}(\mathcal{D}_1) \bowtie \pi_{S \cap T}(\mathcal{D}_2)) \setminus \pi_{(R \cup S) \cap T}(\mathcal{D}_3)\right)$$

Zdrojový kód 35: Relační dělení (Tutorial D)

```
1 relexpr1 DIVIDE BY relexpr2 PER relexpr3;
```

6.3.9 Vnitřní a vnější spojení

Doposud probrané spojení bylo v podstatě vnitřní spojení (klíčové slovo `INNER JOIN` v SQL). Abychom mohli uvažovat vnější spojení, je třeba uvažovat koncept *chybějící* hodnoty, to jest porušit první normální formu. V SQL by se tedy místo `NATURAL (INNER)JOIN` použilo `FULL OUTER JOIN`, `LEFT OUTER JOIN` nebo `RIGHT OUTER JOIN`.

Zdrojový kód 36: Vnější spojení (SQL)

```
1 SELECT * FROM table_1 FULL OUTER JOIN ON table_1.x < table_2.x
```

Při levém vnějším spojení se do výsledku přidávají i visící n-tice z první tabulky.

7 Neznámé hodnoty

Drtivá většina „moderních“ DBMS podporuje tzv. tříhodnotovou logiku, kde vstupuje do popředí, kromě *pravdy* a *nepravdy*, také hodnota *NULL*, která říká, že daná hodnota je nedefinovaná nebo neznámá, což se bude jevit každému myslícímu člověku jako absolutní nesmysl. Těmto hodnotám je dobré se vyhnout, protože činí logické operace (potažmo funkce) nepřehlednými, jak ukazuje tabulka 3.

Příklad 7.1 (Datův příklad k NULLovým hodnotám)**:** Mějme vstupní data (tabulka 4) a nad těmito daty proveďme dotaz:

Zdrojový kód 37: Dotaz pracující s NULL hodnotami (SQL)

```
1 SELECT  x, y
```

Tabulka 4: Příklad relace s NULL hodnotou

(a) Tabulka n1

x	yn
45	London

(b) Tabulka n2

y	zn
33	-

```
2 FROM    n1, n2
3 WHERE   yn <> zn OR zn <> "Paris";
```

Logicky bychom očekávali jako výsledek operace n-tici $\langle 45, 33 \rangle$, avšak v SQL tomu tak není, protože dané porovnání v dotazu nevrací nepravdu nebo pravdu, ale hodnotu NULL.

> "You can never trust the answer you get from database with nulls."
>
> C. J. Date

8 Integritní omezení

Jedná se o jednu z komponent Coddova relačního modelu. Účel integritních omezení je ten, že je v databázi třeba udrže určou efektivitu operací a hlavně uchovat data v konzistentním stavu z hlediska jejich smyslu.

8.1 Výrazy relační algebry

Relační algebra (RA) obsahuje atomické výrazy, tedy relační proměnné a konstantní relace (TABLE_DEE, TABLE_DUM). Dále jsou obsaženy také složené výrazy, které jsou ustaveny rekurzivně, tedy:

1. Pokud jsou R_1 a R_2 relační výrazy.
2. Pak je i $R_1 \bowtie R_2$ relační výraz.
3. A taktéž i $R_1 \cap R_2$ je relační výraz.
4. Podobně s dalšími operacemi.

8.2 Omezení z pohledu relační algebry

1. Pro výraz R definujeme omezení tvaru $R = \varnothing$, tedy nesmí existovat n-tice, které by jej splňovala.
2. Pro výrazy R, S definujeme omezení tvaru $R \subseteq S$, tedy každá n-tice z R musí splňovat S.

Pro 1 a 2 existuje vzájemná převoditelnost:

$$\text{„}\Rightarrow\text{“}, R = \varnothing \text{ lze chápat jako } R \subseteq \varnothing$$
$$\text{„}\Leftarrow\text{“}, R \subseteq S \text{ lze chápat jako } R \setminus S = \varnothing$$

Tabulka 5: Cizí klíče v relacích

(a) Pracovnící

id	jméno	příjmení	laboratoř
01	Vilík	Devil	666
02	James	Bond	007
03	Kryštof	Kolumbus	1492
04	Jiří	Dikobraz	1111

(b) Vlastníci aut

id	auto
04	Trabant
02	Porsche 911
01	Nimbus 3000
04	Autobus

8.3 Referenční integritní omezení

Výchozí myšlenkou je, že pokud máme systém tabulek, tak hodnoty v určitých sloupcích, které popisují vlastnost určité entity, se musí shodovat s hodnotami ve sloupcích (popisujících stejnou vlastnost) pro stejnou entitu v ostatních tabulkách v tomto systému tabulek.

Tuto vlastnost nazýváme *cizí klíč*.

Příklad 8.1 (Cizí klíče v relacích)**:** Představme si dvojici tabulek (systém tabulek 5), která popisuje vlastnictví aut jednotlivými akademickými pracovníky. Dle tabulky vidíme, že číslo každého vlastníka auta v tabulce 5b odpovídá osobě z tabulky 5a.

Zároveň neexistuje takový majitel auta, který by neměl v první tabulce záznam. Auto zkrátka nemůže být v tomto idealizovaném systému bez majitele. Tento vztah lze zapsat matematicky. Pojmenujme první tabulku $\mathcal{D}_1$ a druhou $\mathcal{D}_2$, pak lze napsat následující tvrzení:

$$\rho_{id} \leftarrow pracovnik(id(\pi_{\{id\}}(\mathcal{D}_2))) \subseteq \pi_{\{id\}}(\mathcal{D}_1)$$

V SQL lze modelovat cizí klíče pomocí klíčového slova `FOREIGN KEY`.

8.3.1 Chování při použití cizího klíče

Konkrétní tvrzení z příkladu 8.1 lze pochopitelně zobecnit:

$$\rho_y \leftarrow z(\pi_x(R_1)) \subseteq \pi_y(R_2)$$

1. Nastane chyba, pokud se pokusíme do R_1 vložit n-tici, jejíž hodnota pro atribut x se nenachází v žádné n-tici z R_2 v atributu y.

2. Pokus o aktualizaci cizího klíče, kde mohou nastav v zásadě dvě možné situace:

   (a) Pokus smazat záznam z R_2, jehož hodnota pro y se nachází v R_1. Tedy v analogii k předchozímu příkladu 8.1 bychom se snažili smazat pracovníka, který ale vlastní nějaké auto.

   (b) Pokus změnit záznam z R_2 tak, že nová hodnota pro y se nenachází v R_1. Tedy v analogii k předchozímu příkladu 8.1 bychom se snažili změnit vlastníka nějakého auta na pracovníka, který neexistuje.

Tabulka 6: Tranzitivní uzávěr tabulky

(a) Výchozí tabulka

podmiňující př.	podmíněný př.
PP1	PP2
VT	PP2
PP2	PP3

(b) Tranizitivně uzavřená tabulka

podmiňující př.	podmíněný př.
PP1	PP2
PP1	PP3
PP2	PP3
VT	PP2
VT	PP3

DBMS může určitým způsobem reagovat na předchozí situace:

1. Databázový systém nemusí vykonat nic, jen nahlásí chybu. Tohle chování je vlastní systémům pracujícím s SQL.
2. Může být použito *kaskádování*, kde se provedená změna propaguje do ostatních tabulek, ve kterých je změněná hodnota referována jako cizí klíč.
3. Databázový systém může rovněž nastavit výskyt cizího klíče buď na NULL nebo na výchozí hodnotu (viz. klíčove slovo `DEFAULT` v SQL.

9 Tranzitivní uzávěr relace

Definice 9.1 (Tranzitivní uzávěr relace (v matematickém smyslu))**:** Mějme $R \subseteq A \times A$. Pak tranzitivní uzávěr $R^\infty \subseteq A \times A$ je definován jako:

$$R^\infty = \bigcup_{n=1}^{\infty} R^n = \bigcup_{n=1}^{\infty} \underbrace{R \circ \cdots \circ R}_{\text{n-krát}}$$

Pokud je A je konečná, pak existuje index m tak, že $R^\infty = \bigcup_{n=1}^{m} R^n$. Je-li navíc R reflexivní, pak $R^\infty = R^m$.

Nyní aplikujme tranzitivní uzávěr na databáze.

Příklad 9.1 (Tranzitivní uzávěr tabulky)**:** Mějme tabulku (tabulka 6a) podmiňujících a podmíněných předmětů se dvěma sloupci. Výsledkem tranzitivního uzávěru je úplná informace o podmiňujích atributech. Tabulky zobrazují podmiňující a podmíněné předměty. Například vidíme, že pro splnění předmětu PP3 je třeba mít přímo splněný předmět PP2, avšak ke splnění předmětu PP2 je třeba mít splněn VT i PP1. Tyto informace jdou z tabulky číst relativně obtížně a pro gigantické tabulky se nepřímé čtení těchto vztahů stává problémové.

Pokud je $\mathcal{D}$ relace nad schématem $\{x, y\}$, která je protějškem $R \subseteq A \times A$, pak

$$\mathcal{D} = \{\{\langle x, a\rangle, \langle x, b\rangle\} \mid \langle a, b\rangle \in R\}$$

a vyjídříme protějšek R^∞ jako

$$\mathcal{D}^\infty = \bigcup_{n=1}^{m} \pi_{\{x,y\}} \left(\bowtie_{i=1}^{n} \rho_{x'_{i,n} \leftarrow x, y'_{i,n} \leftarrow y}(\mathcal{D}) \right)$$

kde

$$\begin{array}{ll} x'_{1,n} = x & \text{pro každé } n \\ y'_{m,n} = y & \text{pro každé } n \\ x'_{i,n} = y'_{i-2,n} & \text{pro každé } 2 \leq i \leq n \end{array}$$

Tranzitivní uzávěr je definovatelný, ale definice závisí na m.

Zdrojový kód 38: Tranzitivní uzávěr (Tutorial D)

```
TCLOSE relexpr
```

Zdrojový kód 39: Kostra rekurzivního dotazu (SQL)

```
WITH RECURSIVE jmeno(sloupce) AS (
    /* nerekurzivní výraz
    SELECT  *
    FROM    foo
    UNION   DISTINCT
    /* rekurzivní výraz */
    SELECT  *
    FROM    bar;
)
SELECT  *
FROM    jmeno_sloupce;
```

Zdrojový kód 40: Tranzitivní uzávěr (SQL)

```
WITH RECURSIVE tr(x, y) AS (
    SELECT  *
    FROM    r
    UNION   DISTINCT
    SELECT  x, y tr.y FROM r, tr
    WHERE   r.y = tr.x
)
SELECT * FROM tr;
```

Vyhodnocení dotazu z kódu 39 probíhá následovně:

1. Vyhodnotí se nerekurzivní dotaz a jeho výsledky se uloží do dočasných tabulek RESULT a WORK.
2. Dokud je WORK neprázdná, tak se opakuje následující:
   (a) Vyhodnotí se rekurzivní dotaz, v němž je self reference nahrazena tabulkou WORK. Výsledek si označme R_1.
   (b) Pokud je použit `UNION DISTINCT`, pak se z R_1 ostraní duplicity a vše, co už je v RESULT.

(c) Obsah R_1 se přidá do RESULT.

(d) WORK se nastaví na R_1.

10 Transakční zpracování dat

V dnešních DBMS je třeba, aby k databázi mohlo přistupovat několik uživatelů (aplikací) současně.

Příklad 10.1 (Souběžný přístup dvou uživatelů k databázi)**:** Představme si tabulku vytvořenou příkazem `CREATE TABLE table_1 (x INTEGER NOT NULL PRIMARY KEY)`. Do tabulky se vloží hodnota 100. Následně se s touto hodnotou snaží pracovat dva uživatelé:

Zdrojový kód 41: Úprava hodnoty v tabulce dvěma uživateli (uživatel 1)

```
1 UPDATE table_1 SET x = 100;
2 /* nyní se nic neděje */
3 SELECT * FROM table_1 WHERE x = 100;
```

Zdrojový kód 42: Úprava hodnoty v tabulce dvěma uživateli (uživatel 2)

```
1 /* nyní se nic neděje */
2 UPDATE table_1 SET x = 20;
3 /* nyní se nic neděje */
```

Vidíme, že v prvním časovém okamžiku uživatel 1 upravil hodnotu x na 100 a uživatel 2 nedělal nic. V druhém časovém okamžiku naopak uživatel 1 nic nedělá a uživatel 2 mění hodnotu x na 20. Ve třetím časovém okamžiku uživatel 1 pokládá dotaz na zobrazení všech záznamů z tabulky, kde $x = 100$, tedy hodnotě, na kterou předtím x nastavil.

V předchozí situaci může nastat několik problémů. Vidíme, že uživatelé pracují víceméně ve stejný čas se stejným atributem z tabulky. Databázový systém by se tedy mohl zachovat tak, že některému uživateli práci zakáže. Co si myslíte, že vrátí finální dotaz uživatele 1? Dostane uživatel 1 zpět své x s hodnotou 100? Nebo se u něj projeví změna, kterou provedl uživatel 2? Jak se DBMS zachová?

Předpokládejme, že uživatel 1 potřebuje provést oba své dotazy nebo ani jeden z nich. Pokud by provedl jen jeden, tak to pro něj může znamenat ještě horší stav, než když neprovede žádný. To, co uživatel 1 potřebuje, je tzv. *transakce*. Transace je posloupnost operací nad databází, která se (podle okolností) provede kompletně celá nebo se neprovede vůbec, tedy změny se do databáze perzistentně neuloží.

Začátek transakce se v SQL označuje klíčovým slovem `BEGIN TRANSACTION`. Po aktuálním provedené všech dotazů lze transakci buďto potvrdit klíčovým slovem `COMMIT` nebo vrátit změny provedené v transakci zpět pomocí `ROLLBACK`.

10.1 Sériové provedení DB operací

V tomto přístupu jsou operace prováděny jedna po druhé. Přístup více uživatelů nehraje žádnou zásadní roli.

10.2 Serializovatelné provedení DB operací

Operace mohou být prováděný souběžně (paralelně), ale tak, aby výsledky odpovídaly tomu, jako by operace běžely sériově.

Transakci lze označit jako read-only pomocí klíčového slova `SET TRANSACTION READ ONLY` či jako read-write pomocí `SET TRANSACTION READ WRITE`.

10.3 Atributy transakcí

Transakcím lze nastavovat různé chování v kritických situacích pomocí atributu:

READ UNCOMMITTED Transakci je umožněno číst špinavá data, tedy data změněná jinou probíhající transakcí.

READ COMMITTED Transakci je umožněno číst pouze čistá data, tedy data, která byla upravena jinou dokončenou transakcí.

REPEATED READ Zaručuje, že po sobě jdoucí provedení stejného dotazu vrací stejný výsledek, i když mezitím jiná transakce provede commit.

SERIALIZABLE Serializovatelné provedené operace, viz kapitolka 10.2. Toto chování je u většiny DBMS obvykle nastaveno jako implicitní.

10.4 Body uložení

Body uložení (anglicky „save points“) jsou prostředkem, jak lze uvnitř transakce tuto transakci rozdělit na podtransakci. Poté lze po dokončení celé transakce vracet změny k tomu bodu uložení. Toto je možné například v PostgreSQL.

11 Funkční závislosti

Uvažujem fixní relační schéma $R \subseteq \mathcal{T}$.

Definice 11.1 (Funkční závislost)**:** Funkční závislost nad schématem R je formule ve tvaru $A \Rightarrow B$, kde $A, B \subseteq R$. Říkáme, že: „B funkčně závisí na A.“ $A \Rightarrow B$ právě, když pro libovolné dvě n-tice t_1, t_1 každého přípustného stavu určité relace (například relace $\mathcal{D}$ nad schématem R platí, že je-li a_1 resp. b_1 (tedy hodnota atributu A resp. B) v n-tici t_1 a a_2 resp. b_2 (opět hodnota atributu A resp. B) v n-tici t_2 a $a_1 = a_2$, pak i $b_1 = b_2$.

Tedy každé dvě závislé n-tice jsou si rovny i na atributech z B, pokud jsou si rovny na atributech z A. Pokud máme ntice

$$t_1(A), t_2(A), \text{ což jsou n-tice z } A \text{ a } t_{1,2} \in \prod_{y \in R} dom(y), A \subseteq R$$

Tabulka 7: Tabulky k příkladu 11.2

(a) Tabulka $\mathcal{D}_1$

w	x	y	z
a	b	1	5
a	c	0	0
d	d	1	5

(b) Tabulka $\mathcal{D}_2$

w	x	y	z
a	b	1	2
b	a	15	7
a	b	1	48

(c) Tabulka $\mathcal{D}_3$

w	x	y	z
a	b	7	8
a	a	48	8
a	b	8	8

pak, když

$$t_1(A) = t_2(A), \text{ pak } t_1(B) = t_2(B), \forall t_{1,2} \in \mathcal{D}$$

a v takovýchto případech můžeme říkaž, že „funkční závislost je pravdivá.“

Příklad 11.1 (Funkční závislosti)**:** Mějme relační schéma $R = \{w, v, x, y, z\}$. Pak funkčními závislostmi nad tímto schématem jsou mj.:

$$\begin{aligned} \{x\} &\Rightarrow \{y\} \\ \{w, x\} &\Rightarrow \{z, y\} \\ \varnothing &\Rightarrow \{\dots\} \\ \{\dots\} &\Rightarrow \varnothing \\ &\vdots \Rightarrow \vdots \end{aligned}$$

Příklad 11.2 (Vybrané funkční závislosti)**:** Mějme závislost:

$$\begin{aligned} \mathcal{D}_1 &\vDash \{w, y\} \Rightarrow \{y\} \\ \mathcal{D}_2 &\vDash \{w, y\} \Rightarrow \{y\} \\ \mathcal{D}_3 &\nvDash \{w, y\} \Rightarrow \{y\} \end{aligned}$$

Zdrojové tabulky $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$ zobrazuje systém tabulek 7.

Věta 11.1 (Pravdivá tvrzení k funkčním závislostem)**:** Platí následující tvrzení:

1. Pokud $A \subseteq B$, pak z $t_1(B) = t_2(A)$ plyne, že $t_1(A) = t_2(B)$.
2. $t_1(\varnothing) = t_1(\varnothing), \forall t_1, t_2$
3. Pokud $B \subseteq A$, pak $\mathcal{D} \vDash A \Rightarrow B, \forall \mathcal{D}$.
4. Pokud $\mathcal{D} \vDash A \Rightarrow B$, pak $\mathcal{D} \vDash A \Rightarrow B \cap C$.
5. Pokud $\mathcal{D} \vDash A \Rightarrow B$, pak $\mathcal{D} \vDash A \cup C \Rightarrow B$.
6. Speciálně platí, že cokoliv $\Rightarrow \varnothing$.

11.1 Funkční závislosti versus (nad)klíče

Věta 11.2: Mějme relační schéma R a nad ním relaci $\mathcal{D}$. Pak $K \subseteq R$ je nadklíč právě, když $\mathcal{D} \vDash K \Rightarrow R$.

Důkaz 11.1: Pokud $t(K) = t'(K)$ pak $t = t'$ (tedy $t(R) = t'(R)$), pak je K nadklíč. K je klíč pro $\mathcal{D}$ právě, když $\mathcal{D} \vDash K \Rightarrow R$ a pro $\forall K' \subset K$ platí, že $\mathcal{D} \vDash K' \Rightarrow R$. □

Věta 11.3 (Existence výrazu relační algebry pro funkční závislost)**:** Pro každou $A \Rightarrow B$ existuje výraz relační algebry S tak, že $\mathcal{D} \vDash A \Rightarrow B$ právě, když integritní omezení $S = \varnothing$ je splněno v $\mathcal{D}$.

Důkaz 11.2 (Existence výrazu relační algebry pro funkční závislost)**:** Předpokládejme, že $A \Rightarrow B, A$ a $B \subseteq R$ a dále, že

$$A = \{y_1, \ldots, y_n\}$$
$$B = \{z_1, \ldots, z_k\}$$

Následně

$$\underbrace{\pi_{A \cup B}(\mathcal{D}) \bowtie \rho_{y'_1 \leftarrow y_1, \ldots, y'_n \leftarrow y_n, z'_1 \leftarrow z_1, \ldots, z'_k \leftarrow z_k} \left(\pi_{A \cup B}(\mathcal{D})\right)}_{\mathcal{D}_2}$$
$$\sigma_{y_1 = y'_1 \wedge \cdots \wedge y_n = y'_n \neg (z_1 = z'_1 \wedge \cdots \wedge z_k = z'_k)} \left(\mathcal{D}_2\right)$$

□

Definice 11.2 (Teorie a model teorie)**:** Mějme relační schéma R. Každá množina funkčních závislostí nad R se nazývá *teorie*. $\mathcal{D}$ je model teorie T, pokud $\mathcal{D} \vDash A \Rightarrow B, \forall A \Rightarrow B \in T$. Závislost $A \Rightarrow B$ sémanticky plyne z T, pokud $A \Rightarrow B$ je podnožinou každého modelu T.

Třídu všech modelů nad T značíme $\mathrm{Mod}(T)$. Systém relací (tedy množinu určitých tabulek) značíme $\mathfrak{M}$. $\mathfrak{M} \vDash A \Rightarrow B$, pokud $\mathcal{D} \vDash A \Rightarrow B, \forall \mathcal{D} \in \mathfrak{M}$.

Dále zjevně $\mathrm{Mod}(T) \vDash A \Rightarrow B$ a také $T \vDash A \Rightarrow B$.

Věta 11.4 (Další fakta o funkčních závislostech)**:** Platí, že:

1. Pokud $T_1 \subseteq T_2$, pak $\mathrm{Mod}\,(T_2) \subseteq \mathrm{Mod}\,(T_1)$.
2. $\mathrm{Mod}(\varnothing)$ označuje všechny relace nad R.
3. $\mathrm{Mod}(T)$ obsahuje vždy prázdnou relaci a všechny jednoprvkové relace.

Věta 11.5 (O podmnožinách teorie)**:** Pokud $T_1 \subseteq T_2$ a $T_1 \vDash A \Rightarrow B$, pak i $T_2 \vDash A \Rightarrow B$. $T \vDash A \Rightarrow B, \forall A \Rightarrow B \in T$.

Položme si fundamentální otázku, a sice zda $T \vDash A \Rightarrow B$.

12 Kanonické relace

Výchozí idea je taková, že místo jedné relace $\mathcal{D}$ uvažujeme několik relací

$$\mathcal{D}_1, \mathcal{D}_2, \ldots$$

přičemž všechny tyto dílčí relace jsou dvouprvkové, tedy obsahují 2 n-tice.

Tabulka 8: Tabulka $\mathcal{D}_M$ pro příklad 12.1

v	w	x	y	z
p	p	p	p	p
q	p	q	q	p

12.1 Funkční závislosti

Kanonická relace nad R je taková, že pro $M \subseteq R$ definujeme $\mathcal{D}_M$ jako relaci nad R. Pro každý atribut $y \in R$ uvažujeme doménu $\mathcal{D}_y \subseteq \{p, q\}$, kde p, q jsou dva fixní prvky. Dále uvažujeme n-tici t_1 tak, že:

$$t_1(y) = p, \forall y \in R$$
$$t_1(y) = \begin{cases} p & \text{pokud } y \in M \\ q & \text{pokud } y \notin M \end{cases}$$
$$\mathcal{D}_M = \{t_1, t_2\}$$

Příklad 12.1 (Funkční závislosti v kanonické relaci)**:** Mějme:

$$R = \{x, y, w, v, z\}$$
$$M = \{w, z\}$$

Výslednou relaci popisuje tabulka 8. Takových relací existuje pro každé R celkem $2^{|R|}$.

Věta 12.1 (Pravdivost závislosti v kanonické relaci)**:** V kanonické relaci je pravdivá $\mathcal{D}_M \vDash A \Rightarrow B$ právě, když $A \subseteq M \implies B \subseteq M$.

Důkaz 12.1 (Pravdivost závislosti v kanonické relaci)**:** Jediná netriviální dvojice n-tic z $\mathcal{D}_M$ je nějaká t_1, t_2. Pak $t_1(A) = t_2(A)$ právě, když $A \subseteq M$. Situace platí analogicky také pro B. □

Věta 12.2 (Existence systému kan. relací pro jinou relaci)**:** Pro každou relaci $\mathcal{D}$ nad schématem R existuje systém kanonických relací $\mathfrak{S}$ nad R tak, že pro $\forall A \Rightarrow B$ plati, že $\mathcal{D} \vDash A \Rightarrow B$ právě, když $\mathfrak{S} \vDash A \Rightarrow B$.

Důkaz 12.2 (Existence systému kan. relací pro jinou relaci)**:** Platí, že:

$$\mathfrak{S} = \left\{\mathcal{D}_{M_{t_1,t_2}} \mid M_{t_1,t_2} \in R \text{ taková, že } y \in M_{t_1,t_2} \Leftrightarrow t_1(y) = t_2(y) \text{ a } t_1, t_2 \in \mathcal{D}\right\}$$

Dále je řešení (prý) jasné. □

Tabulka 9: Vstupní tabulka pro příklad 12.2

x	y	z
10	30	6
10	40	7
20	50	6

Tabulka 10: Výsledek příkladu 12.2

(a) Tabulka pro M_{t_1,t_2}

x	y	z
p	p	p
p	q	q

(b) Tabulka pro M_{t_2,t_3}

x	y	z
p	p	p
q	q	q

(c) Tabulka pro M_{t_1,t_3}

x	y	z
p	p	p
q	q	p

Příklad 12.2 (Existence systému kan. relací pro jinou relaci)**:** Mějme tabulku 9 a nechť navíc:

$$M_{t_1,t_2} = \{x\}$$
$$M_{t_2,t_3} = \varnothing$$
$$M_{t_1,t_3} = \{z\}$$

Výsledky jsou obsaženy v tabulce 10.

Věta 12.3: Platí, že:

$$T \vDash A \Rightarrow B \Leftrightarrow \mathrm{Mod}_c(T) \vDash A \Rightarrow B$$

kde

$$\mathrm{Mod}_c = \{\mathcal{D}_M \mid \mathcal{D}_M \in \mathrm{Mod}(T)\}$$

Důkaz 12.3: Důkaz je triviální, vzhledem k tomu, že $\mathrm{Mod}_c(T) \subseteq \mathrm{Mod}(T)$. Proto pak $T \vDash A \Rightarrow B$, tedy $\mathrm{Mod}(T) \vDash A \Rightarrow B$ a tedy také $\mathrm{Mod}_c(T)$.

Pro opačný směr předpokládejme, že $\mathrm{Mod}_c(T) \vDash A \Rightarrow B$. Vezmeme libovolnou $\mathcal{D} \in \mathrm{Mod}(T)$. Dle předchozí věty existuje $\mathcal{S}$ tak, že $\mathcal{D} \vDash E \Rightarrow F$ právě, když $\mathcal{S} \vDash E \Rightarrow F$ pro každé $E \Rightarrow F$. Jelikož $\mathcal{S} \subseteq \mathrm{Mod}_c(T)$, tak dostáváme, že $\mathcal{D} \vDash A \Rightarrow B$. □

Definice 12.1 (Uzávěrový systém)**:** Platí, že:

$$\mathfrak{M}_T = \{M \subseteq R \mid \mathcal{D}_M \in \mathrm{Mod}_c(T)\}$$
$$\mathfrak{M}_T = 2^R$$

Věta 12.4 (Uzávěrový systém)**:** Tvrdíme, že $\mathfrak{M}_T$ je skutečně uzávěrovým systémem.

Důkaz 12.4 (Uzávěrový systém)**:** Vezmeme libovolné $M_i \in \mathfrak{M}_T, i \in I$ a ověříme, že $\bigcap_{i\in I} M_i \in \mathfrak{M}_T$. Dle definice uzávěrového systému (12.1) stačí ověřit, že:

$$\mathcal{D}_{\bigcap_{i\in I} M_i} \in \mathrm{Mod}_c(T)$$

pro libovolnou $A \Rightarrow B \in T$ takovou, že $A \subseteq \bigcap_{i\in T} M_i$, pak $A \subseteq \bigcap_{i\in T} M_i$.

Avšak každá $\mathcal{D}_M \in \mathrm{Mod}_c(T), (i \in I)$, to jest $B \subseteq M_i, (i \in I)$, tedy $B \subseteq \bigcap_{i\in I} M_i$. Tedy $\mathcal{D}_{\bigcap_{iinI} M_i} \vDash A \Rightarrow B$ □

Definice 12.2 (Uzávěrový operátor)**:** Pro všechna $M \subseteq R$ jsme tedy schopni definovat *uzávěrový operátor* příslušný k určité $\mathfrak{M}_T$. Konkrétně:

$$[M]_T = \bigcap \{N \in \mathfrak{M}_T \mid M \subseteq N\}$$

$[\ldots]_T$ značí uzávěrový operátor. To znamená, že pro libovolnou $M \in R$ je $\mathcal{D}_{[M]_T} \in \mathrm{Mod}_c(T)$.

Věta 12.5: Následující tvrzení jsou ekvivalentní:

$$T \vDash A \Rightarrow B \tag{1}$$
$$\mathcal{D}_{[A]_T} \vDash A \Rightarrow B \tag{2}$$
$$B \subseteq [A]_T \tag{3}$$

Důkaz 12.5: Budeme dokazovat implikace mezi tvrzeními z věty 12.5:

1. 1 $\Longrightarrow$ 2
2. 2 $\Longrightarrow$ 3
3. 1 $\Longrightarrow$ 3

1 má řešení, protože $\mathcal{D}_{[A]_T} \in \mathrm{Mod}(T)$. 2 má řešení, protože $A \subseteq A_{[T]}$ (extenzitivita). 3 má také řešení, avšak narozdíl od předešlých není triviální.

Předpokládejme, že $B \subseteq [A]_T$, následně stačí ověřit, že $A \Rightarrow B$ v každém $\mathcal{D}_M \in \mathrm{Mod}_c(T)$. Pokud $A \subseteq M$, pak z monotonie uzávěrových operátorů plyne:

$$B \subseteq [A]_T \subseteq [M]_T = M$$

Předchozí je možné, protože $\mathcal{D}_M$ je kanonický model. □

13 Boyce – Codova normální forma

Boyce–Codova normální forma (BCNF) je jednou z normálních forem používaných při normalizaci databáze. BCNF byla vytvořena v roce 1974 Raymondem Boycem a Edgarem Coddem, aby odstranila některé anomálie, které se vyskytují v relacích, které jsou ve 3. normální formě (13.1).

Poznámka 13.1 (Třetí normální forma)**:** 3. normální forma (3NF) je soubor doporučení (metodika) pro návrh datové struktury databáze. 3NF obsahuje podmožinu. Je to 2. normální forma (2NF) a 1. normální forma (1NF). 3NF následující pravidla:

1. Eliminuj duplicitní sloupce v jednotlivých relacích.
2. Pro každou skupinu dat s jasně vymezeným významem vytvoř zvláštní relaci, každý záznam opatři unikátním primárním klíčem.
3. Obsahem jednotlivých sloupců relace by měla být jednoduchá, dále nedělitelná informace.
4. Podmnožinu dat se shodnou hodnotou pro určitý sloupec relace převeď do samostatné relace a spoj s původní relací cizím klíčem.
5. Odstraň z relace sloupce, které jsou přímo závislé na jiné skupině sloupců relace než pouze na primárním klíči.

Definice 13.1 (Boyce–Coddova normální forma)**:** Relace R je v BCNF tehdy a jen tehdy, když pro každou netriviální funkční závislost $X \Rightarrow Y$, kde X, Y jsou množiny atributů a zároveň $Y \not\subseteq X$, platí, že X je nadmnožinou nějakého klíče, nebo X je klíčem relace R.

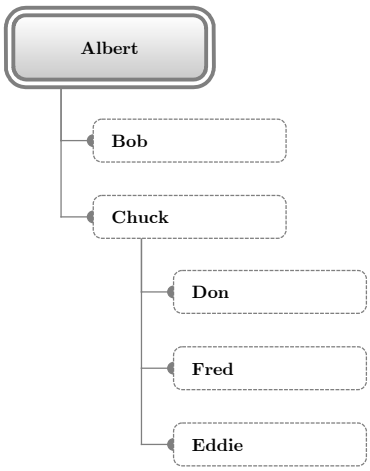
Definice 13.2 (Alternativní pojetí BCNF)**:** Relační schéma R vzhledem k teorii T je v BCNF, pokud platí, že každá netriviální $A \Rightarrow B$, která $T \vDash A \Rightarrow B$, obsahuje klíč v A. Jinak řečeno, pokud A je nadklíč.

Příklad 13.1 (Úvaha i nešikovné konstrukci relace)**:** Uvažujme tabulku 5, konkrétně její část 5b. Vidíme, že každé *id* představuje jednoho pracovníka. Tabulka jako taková představuje vztah *vlastnit auto*, kde každý pracovník může vlastnit i několik aut.

Takto konstruovaná tabulka, která využívá cizí klíče k modelování obdobných situací, je víceméně ideálním řešením. Naivní začátečník by mohl prohlásit, že použití cizích klíčů je složité a zbytečně robustní pro jeho danou aplikaci a sloupec *id* by nahradil přímo celým jménem pracovníka. Tak by měl k dispozici konkrétní údaje o pracovníkovi pouhým dotazem nad touto tabulkou. Prosté *id* by mu přece nic neřeklo a musel by konstruovat složitější dotaz, aby jméno pracovníka (vlastníka auta) vůbec získal.

Jenže, co když se některá pracovnice vdá a její jméno se změní. Pak bude muset tento šikovný začátečník aktualizovat všechny záznamy, které obsahovaly staré jméno a bude jej muset nahradit jménem novým. To se v přístupu s cizím kličem nestane, protože bude potřeba aktualizovat právě jeden záznam v tabulce 5a.

Věta 13.1 (O triviální funkční závislosti)**:** Funkční závislost $A \Rightarrow B$ je triviální právě, když $B \subseteq A$.



Obrázek 3: Ukázkový strom

Tabulka 11: Hranová reprezentace stromu

id	rodič
Albert	Albert (nebo NULL)
Bob	Albert
Chuck	Albert
Don	Chuck
Fred	Chuck
Eddie	Chuck

14 Reprezentace hierarchických struktur

Databáze může modelovat chování známých datových hierarchických struktur.

14.1 Reprezentace stromů

Stromy lze reprezentova hned několika způsoby.

14.1.1 Hranová reprezentace

V této reprezentaci je třeba jen jedna tabulka, které obsahuje dva sloupce. Jeden sloupec popisuje data, které jsou nesena daným uzlem, a druhý sloupec určuje rodiče daného uzlu. Příklad stromu popisuje tabulka 11. Strom lze vidět na obrázku 3.

Vidíme, že pokud jsou hodnoty v obou sloupcích shodné nebo je hodnota ve sloupci *rodič* rovna hodnotě NULL, pak je daný uzel kořenem stromu. Již letmým pohledem vidíme, že tabulka obsahuje relativně minimum dat a nedá se z ní vyčíst například, zda Chuck je levým nebo pravým potomkem Alberta. Implementace vyhledávacího stromu je tedy prakticky nemožná.

Tabulka 12: Cestová reprezentace stromu

id	cesta
Albert	/Albert
Bob	/Albert/Bob
Chuck	/Albert/Chuck
Don	/Albert/Chuck/Don
Fred	/Albert/Chuck/Fred
Eddie	/Albert/Chuck/Eddie

Tabulka 13: Reprezentace stromu vnořenými množinami

id	lt	rt
Albert	1	12
Bob	2	3
Chuck	4	11
Don	5	6
Fred	7	8
Eddie	9	10

14.1.2 Cestová reprezentace

V cestové reprezentaci je pro každý uzel popsána jeho úplná cesta napříč celým stromem. V tomto případě je nejlepší pohled na tabulku 12, která způsob reprezentace dokonale vystihuje.

Na procházení takto reprezentovaného stromu se hodí operátor `LIKE` z SQL.

14.1.3 Reprezentace vnořenými množinami

Speciální reprezentace, která se již neřadí mezi naivní, nýbrž pokročilé. Využívá v podstatě kombinace preorder a postorder průchodů stromem, kdy se každý uzel ohodnocuje dvojicí čísel, které popisují jeho „pořadí“ ve stromu. Více napoví tabulka 13.

Bystré oko čtenáře vidí, jak se strom tímto způsobem „ohodnocuje“. Hodnoty atributů *lt* a *rt* odhalují vlastnosti a umístění jednotlivých uzlů ve stromů.

Příklad 14.1 (Dotazy na části množinově reprezentovaného stromu)**:** Vidíme, že kořen má zvláštní hodnotu ve sloupci *lt*.

Zdrojový kód 43: Dotaz na kořen stromu v množinové reprezentaci

```
1 SELECT * FROM tree WHERE lt = 1;
```

Také listy mají poněkud unikátní vztah mezi oběma hodnotami.

Zdrojový kód 44: Dotaz na listy stromu v množinové reprezentaci

```
1 SELECT * FROM tree WHERE lt = rt - 1;
```

Velkou zbraní této reprezentace je určení podstromu určitého uzlu.

Zdrojový kód 45: Dotaz na podstrom stromu v množinové reprezentaci

```
SELECT  tree_1.id AS from_node, tree_2.id AS to_node
FROM    tree AS tree_1, tree AS tree_2
WHERE   tree_2.lt BETWEEN tree_1.lt AND tree_1.rt;
```

Seznam zkratek

1NF 1. normální forma

2NF 2. normální forma

3NF 3. normální forma

BCNF Boyce–Codova normální forma

DBMS Database Management System

RA Relační algebra

SQL Structured Query Language

Seznam teorémů

Literatura

[1] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. 1st ed. Cambridge, MA: The MIT Press, June 18, 1990. ISBN: 0-262-03141-8.

[2] Carlos Coronel, Steven Morris, and Peter Rob. *Database Systems: Design, Implementation, and Management*. 9th ed. Boston, MA: Course Technology, Nov. 23, 2009. ISBN: 0-538-74884-2.

[3] Ramez A. Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. 6th ed. Boston, MA: Addison-Wesley, Apr. 9, 2010. ISBN: 0-136-08620-9.