

Databázové systémy

Integritní omezení

Vilém Vychodil

KMI/DATA1, Přednáška 7

Databázové systémy

Přednáška 7: Přehled

1 Relační algebra:

- výrazy relační algebry,
- formalizace relačního dotazování.

2 Integritní omezení:

- obecný pohled, vztah k relační algebře,
- vztah k relačnímu přiřazení.

3 Důležité typy integritních omezení:

- omezení hodnot atributů,
- referenční integrita a cizí klíče,
- nadklíče a klíče.

4 Funkční závislosti:

- syntax a sémantika funkčních závislostí,
- funkční závislosti jako integritní omezení.

Motivace pro integritní omezení

připomeňme (PŘEDNÁŠKA 3):

instance databáze, angl.: *database instance*

Instance databáze je konečná množina relačních proměnných, jejich aktuálních hodnot a integritních omezení (zatím nepotřebujeme).

dotaz, angl.: *query*

Dotaz je částečná rekurzivní funkce z množiny všech instancí databáze do množiny všech relací (nad relačními schématy).

úloha integritních omezení:

Zajistit, aby bylo možné provádět pouze ta relační přiřazení (modifikace relačních proměnných), která zaručují, že databáze bude v konzistentním stavu – všechna integritní omezení budou splněna.

poznámka: užité pojmy nejprve formalizujeme, ...

Relační algebra

zatím jsme ukázali:

- formalizaci pojmu relace (a souvisejících pojmů)
- množinu relačních operací (některé odvoditelné z ostatních)
- korespondence v jazycích Tutorial D (dobrá) a SQL (slabší)

nyní formalizujeme:

- *skalární a relační výrazy* relační algebry
- dva aspekty:
 - *syntaxe výrazů* $\mathcal{E}, \mathcal{F}, \mathcal{G}, \dots$
(definujeme jejich přípustné tvary)
 - *sémantika výrazů* $\mathcal{E}^{\mathcal{D}}, \mathcal{F}^{\mathcal{D}}, \mathcal{G}^{\mathcal{D}}, \dots$
(definujeme jejich interpretaci v instancích databáze)
- omezení úvah: pro naše účely zvolíme pouze fragment relačních operací

Definice (Syntaxe výrazů relační algebry)

Relační výrazy (RV) relační algebry (RA) jsou:

- 1 $\{\emptyset\}$ je RV typu \emptyset ,
- 2 pokud je R relační typ, pak je \emptyset_R RV typu R ,
- 3 pokud $y \in D_y$, pak je $[y:d]$ RV typu $\{y\}$,
- 4 pokud je r relační proměnná typu R , pak je r RV typu R ,
- 5 pokud jsou \mathcal{E} a \mathcal{F} RV typu R , pak jsou i $(\mathcal{E} \cup \mathcal{F})$ a $(\mathcal{E} \setminus \mathcal{F})$ RV typu R ,
- 6 pokud je \mathcal{E} RV typu R a $S \subseteq R$, pak je $\pi_S(\mathcal{E})$ RV typu S ,
- 7 pokud je \mathcal{E} RV typu R , $y \in R$ a $z \in R \cup D_y$, pak je $\sigma_{y=z}(\mathcal{E})$ RV typu R ,
- 8 pokud jsou \mathcal{E} a \mathcal{F} RV typů R a S , pak je $(\mathcal{E} \bowtie \mathcal{F})$ RV typu $R \cup S$,
- 9 pokud je \mathcal{E} RV typu R a $f: R \rightarrow Y$ je injektivní, pak je $\rho_f(\mathcal{E})$ RV typu $f(R)$.

Skalární výrazy (SV) relační algebry jsou:

- 1 pokud jsou \mathcal{E} a \mathcal{F} RV téhož typu, pak je $\mathcal{E} \subseteq \mathcal{F}$ SV,
- 2 pokud jsou \mathcal{E} a \mathcal{F} RV téhož typu, pak je $\mathcal{E} = \mathcal{F}$ SV.

Definice (Sémantika relačních a skalárních výrazů RA)

Nechť \mathcal{D} je instance databáze. Pokud je \mathcal{E} va tvaru

- 1 $\{\emptyset\}$, pak $\mathcal{E}^{\mathcal{D}}$ je relace nad \emptyset obsahující \emptyset (TABLE_DEE)
- 2 \emptyset_R , pak $\mathcal{E}^{\mathcal{D}}$ je prázdná relace typu R ,
- 3 $[y:d]$, pak $\mathcal{E}^{\mathcal{D}} = \{\{\langle y, d \rangle\}\}$,
- 4 r , pak $\mathcal{E}^{\mathcal{D}}$ je hodnota r v instanci \mathcal{D} ,
- 5 $\mathcal{E} \cup \mathcal{F}$, pak $\mathcal{E}^{\mathcal{D}} = \mathcal{E}^{\mathcal{D}} \cup \mathcal{F}^{\mathcal{D}}$ (analogicky pro \setminus)
- 6 $\pi_S(\mathcal{E})$, pak $\mathcal{E}^{\mathcal{D}} = \pi_S(\mathcal{E}^{\mathcal{D}})$,
- 7 $\sigma_{y=z}(\mathcal{E})$, pak $\mathcal{E}^{\mathcal{D}} = \sigma_{y=z}(\mathcal{E}^{\mathcal{D}})$
- 8 $\mathcal{E} \bowtie \mathcal{F}$, pak $\mathcal{E}^{\mathcal{D}} = \mathcal{E}^{\mathcal{D}} \bowtie \mathcal{F}^{\mathcal{D}}$,
- 9 $\rho_f(\mathcal{E})$, pak $\mathcal{E}^{\mathcal{D}} = \rho_f(\mathcal{E}^{\mathcal{D}})$.

Pro SV relační algebry říkáme, že

- 1 $\mathcal{E} \subseteq \mathcal{F}$ je pravdivý v \mathcal{D} pokud $\mathcal{E}^{\mathcal{D}} \subseteq \mathcal{F}^{\mathcal{D}}$,
- 2 $\mathcal{E} = \mathcal{F}$ je pravdivý v \mathcal{D} pokud $\mathcal{E}^{\mathcal{D}} = \mathcal{F}^{\mathcal{D}}$.

Integritní omezení

integritní omezení, angl.: *integrity constraints*

Množina integritních omezení relační databáze je konečná množina skalárních výrazů typu „pravdivostní hodnota“. Instance databáze splňuje danou množinu integritních omezení, pokud jsou všechny dané skalární výrazy *pravdivé*.

dva základní tvary výrazů RA definujících integritní omezení

- 1 $\mathcal{E} = \emptyset_R$, kde \mathcal{E} je výraz RA typu R ;
- 2 $\mathcal{E} \subseteq \mathcal{F}$, kde \mathcal{E} a \mathcal{F} jsou výrazy RA stejného typu.

poznámky:

- $\mathcal{E} = \emptyset_R$ interpretujeme: „žádná hodnota nesmí splňovat \mathcal{R} “
- $\mathcal{E} \subseteq \mathcal{F}$ interpretujeme: „každá hodnota splňující \mathcal{E} splňuje i \mathcal{F} “
- vzájemná převoditelnost ($\mathcal{E} = \emptyset_R$ p. k. $\mathcal{E} \subseteq \emptyset_R$; $\mathcal{E} \subseteq \mathcal{F}$ p. k. jako $\mathcal{E} \setminus \mathcal{F} = \emptyset_R$)
důsledek adjunkce \setminus a \cup (PŘEDNÁŠKA 3)

Poznámky k účelu integritních omezení

původ pojmu:

- jedna z komponent relačního modelu dat (od počátku)



Codd, E. F.: A relational model of data for large shared data banks
Communications of the ACM 13: 6 (1970)

integritní omezení × efektivita:

- integritní omezení = *prostředek zaručení konzistence dat*
- součást logického modelu dat
- neplést s prostředky efektivity: indexy a pod. (záležitost fyzické vrstvy)

jak formulovat:

- obecné pravidlo – čím více, tím lépe
- je možné rozlišit: integritní omezení na straně serveru × uživatele (klienta)
- na straně uživatele (klienta) se považuje za nedostatečné

Obecná integritní omezení v Tutorial D a SQL

Tutorial D:

```
CONSTRAINT <jméno> <skalární-vyraz>;
```

```
DROP CONSTRAINT <jméno>;
```

SQL:

```
ALTER TABLE <jméno-tabulky> ADD  
    CONSTRAINT <jméno-omezení> <definice-omezení>;
```

```
ALTER TABLE <jméno-tabulky> DROP  
    CONSTRAINT <jméno-omezení>;
```

```
CREATE TABLE <jméno-tabulky> (  
    <definice-atributu> CONSTRAINT <jméno-omezení1> <definice-omezení1>,  
    ⋮  
    CONSTRAINT <jméno-omezenín> <definice-omezenín>);
```

Příklad (Tutorial D: Obecná integritní omezení)

```
VAR rt BASE
  RELATION {foo INT, bar INT, baz CHAR}
  KEY {foo};

CONSTRAINT rt_nnega
  IS_EMPTY (rt WHERE (foo < 0));

CONSTRAINT rt_fubar
  IS_EMPTY (rt WHERE (foo > bar));

INSERT rt RELATION {TUPLÉ {foo 10, bar 20, baz "aaa"}};
INSERT rt RELATION {TUPLÉ {foo -1, bar 20, baz "bbb"}}; /* fail */
INSERT rt RELATION {TUPLÉ {foo 30, bar 20, baz "ccc"}}; /* fail */

DROP CONSTRAINT rt_nnega;
DROP CONSTRAINT rt_fubar;

DROP VAR rt;
```

Příklad (SQL: Obecná integritní omezení)

```
CREATE TABLE tab (  
    foo NUMERIC NOT NULL PRIMARY KEY,  
    bar NUMERIC NOT NULL,  
    baz VARCHAR NOT NULL);  
  
ALTER TABLE tab ADD  
    CONSTRAINT tab_nnega CHECK (foo < 0);  
  
ALTER TABLE tab ADD  
    CONSTRAINT tab_fubar CHECK (foo < bar);  
  
INSERT INTO tab VALUES (10, 20, 'aaa');  
INSERT INTO tab VALUES (-1, 20, 'bbb'); /* fail */  
INSERT INTO tab VALUES (30, 20, 'ccc'); /* fail */  
  
ALTER TABLE tab DROP CONSTRAINT tab_nnega;  
ALTER TABLE tab DROP CONSTRAINT tab_fubar;
```

Omezení přípustných hodnot atributů jako integritní omezení

motivace:

Chceme vyjádřit, že pouze některé hodnoty daných typů jsou přípustnými hodnotami atributů relací (např. nezáporné hodnoty označující kvantitu).

Tutorial D:

```
TYPE <jméno> POSSREP {... CONSTRAINT <skalární-výraz>;
```

```
CONSTRAINT <jméno>  
  IS_EMPTY (<relační-výraz> WHERE NOT <podmínka>);
```

SQL (jako v obecném tvaru pomocí klauzule **CHECK**):

```
... CHECK (<podmínka>)
```

```
ALTER TABLE <jméno-tabulky> ADD  
  CONSTRAINT <jméno-omezení> CHECK (<podmínka>); ...
```

Příklad (Tutorial D: Omezení přípustných hodnot)

```
/* type definition, not an integrity constraint per se */
TYPE IPv4_Addr POSSREP {
  a INTEGER, b INTEGER, c INTEGER, d INTEGER
  CONSTRAINT AND {a >= 0, b >= 0, c >= 0, d >= 0,
                  a <= 255, b <= 255, c <= 255, d <= 255}};

TYPE Cnt POSSREP {n INTEGER CONSTRAINT n >= 0};

Cnt (10)  $\models$  Cnt (10)
Cnt (-10) /* fails */

/* making constraint on integer values */
CONSTRAINT foo_in_rt_is_non_negative
  IS_EMPTY (rt WHERE foo < 0);

/* equivalently */
CONSTRAINT foo_in_rt_is_non_negative
  rt = (rt WHERE foo >= 0);
```

Příklad (SQL: Omezení přípustných hodnot)

```
CREATE TABLE tab (  
    foo NUMERIC NOT NULL PRIMARY KEY CHECK (foo >= 0),  
    bar NUMERIC NOT NULL,  
    CHECK (foo <= bar),  
    baz VARCHAR NOT NULL);
```

```
CREATE TABLE tab (  
    foo NUMERIC NOT NULL PRIMARY KEY CONSTRAINT c1 CHECK (foo >= 0),  
    bar NUMERIC NOT NULL,  
    CONSTRAINT c2 CHECK (foo <= bar),  
    baz VARCHAR NOT NULL);
```

poznámky k použití CHECK:

- na úrovni *atributu* může odkazovat pouze na hodnotu atributu (v každé n -tici)
- na úrovni *tabulky* může operovat s hodnotami všech atributů (každé z n -tic)
- nepodporuje vnořené dotazy (PostgreSQL, !!)

Opakování: Klíče (PŘEDNÁŠKA 2)

klíč, angl.: *key*

Uvažujme relační proměnnou X typu $R = \{y_1, \dots, y_n\}$. Množina klíčů proměnné X je libovolná neprázdná množina $\{K_1, \dots, K_n\}$ jejíž prvky jsou podmnožiny R a splňují podmínku, že $K_i \not\subseteq K_j$ pro každé $i \neq j$.

relační přiřazení, angl.: *relational assignment*

Mějme relační proměnnou X typu R a necht' $\{K_1, \dots, K_n\}$ je množina klíčů proměnné X . Pak relaci \mathcal{D} typu R lze **přiřadit jako hodnotu** proměnné X pokud je splněna následující podmínka: Pro každé $i = 1, \dots, n$ a libovolné $r_1, r_2 \in \mathcal{D}$ platí:

pokud $r_1(y) = r_2(y)$ pro každý $y \in K_i$, pak $r_1 = r_2$.

V opačném případě říkáme, že \mathcal{D} porušuje integritní omezení dané některým klíčem relační proměnné X .

Klíče jako integritní omezení

předpoklad:

- relační proměnná \mathbb{r} typu R
- množina atributů $K \subseteq R$

integritní omezení vyjadřující, že K je klíč \mathbb{r} (lépe: *nadklíč*, PŘEDNÁŠKA 8):
pro $R = \{y_1, \dots, y_k, \dots, y_n\}$ a $K = \{y_1, \dots, y_k\}$ uvažujeme

$$\sigma_{y_1=y'_1 \wedge \dots \wedge y_k=y'_k \wedge \neg(y_{k+1}=y'_{k+1} \wedge \dots \wedge y_n=y'_n)} (\mathbb{r} \bowtie \rho_{y'_1 \leftarrow y_1, \dots, y'_n \leftarrow y_n}(\mathbb{r})) = \emptyset_R.$$

předchozí výraz je pravdivý v \mathcal{D} pokud pro každé dvě n -tice $r, r' \in \mathbb{r}^{\mathcal{D}}$ platí:

$$\text{pokud } r(K) = r'(K), \text{ pak } r = r'.$$

SQL (jako v obecném tvaru pomocí klauzule **UNIQUE**):

\dots **UNIQUE** ($\langle \text{atribut}_1 \rangle, \langle \text{atribut}_2 \rangle, \dots$)

Příklad (Tutorial D: Integritní omezení pomocí klíčů)

```
VAR rt BASE
```

```
RELATION {foo INT, bar INT, baz INT, qux INT}
```

```
KEY {qux};
```

```
/* define constraint: {foo, bar} is a key */
```

```
CONSTRAINT fubar_key
```

```
IS_EMPTY ((rt TIMES (rt RENAME {PREFIX "" AS "n"}))
```

```
WHERE AND {foo = nfoo, bar = nbar} AND
```

```
NOT AND {baz = nbaz, qux = nqux});
```

```
/* alternatively, using WRAP */
```

```
CONSTRAINT fubar_key
```

```
IS_EMPTY (((rt WRAP ({baz, qux} AS a)) WRAP ({foo, bar} AS c))
```

```
JOIN
```

```
((rt WRAP ({baz, qux} AS b)) WRAP ({foo, bar} AS c)))
```

```
WHERE NOT (a = b));
```

Příklad (SQL: Integritní omezení pomocí klíčů)

```
ALTER TABLE tab ADD
    CONSTRAINT fubar_key UNIQUE (foo, bar);

CREATE OR REPLACE FUNCTION fubar_trg () RETURNS TRIGGER AS $$
BEGIN
    IF (0 < (SELECT count (*) FROM tab
            WHERE foo = NEW.foo AND bar = NEW.bar AND
            NOT (baz = NEW.baz AND qux = NEW.qux))) THEN
        RAISE EXCEPTION 'fubar_key_constraint_violated!';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER fubar_key BEFORE INSERT OR UPDATE ON tab
    FOR EACH ROW EXECUTE PROCEDURE fubar_trg ();
```

Referenční integritní omezení

motivace:

Hodnoty atributů jedné relace se musí nacházet jako hodnoty (jiných) atributů jiné relace.

referenční integritní omezení, angl.: *referential integrity constraint*

Mějme relační proměnné \mathfrak{r} typu R a \mathfrak{s} typu S . Referenční integritní omezení je výraz ve tvaru $\rho_f(\pi_{R'}(\mathfrak{r})) \subseteq \pi_{S'}(\mathfrak{s})$, kde $R' \subseteq R$ a $S' \subseteq S$.

omezení $\rho_f(\pi_{R'}(\mathfrak{r})) \subseteq \pi_{S'}(\mathfrak{s})$ je *splněno* v \mathcal{D} pokud pro každou $r \in \mathfrak{r}^{\mathcal{D}}$ platí, že existuje $s \in \mathfrak{s}^{\mathcal{D}}$ tak, že $r(f(y)) = s(y)$ pro každý atribut $y \in S'$

Příklad (Referenční integritní omezení)

\mathfrak{s} (studenti), \mathfrak{p} (předměty), \mathfrak{z} (student má zapsaný předmět)

$$\rho_{\text{ID} \leftarrow \text{STUDENT_ID}}(\pi_{\text{STUDENT_ID}}(\mathfrak{z})) \subseteq \pi_{\text{ID}}(\mathfrak{s})$$

$$\rho_{\text{ID} \leftarrow \text{COURSE_ID}}(\pi_{\text{COURSE_ID}}(\mathfrak{z})) \subseteq \pi_{\text{ID}}(\mathfrak{p})$$

Příklad (Tutorial D: Referenční integritní omezení)

VAR student BASE

RELATION {id StudentId, name Name, major Major} KEY {id};

VAR course BASE

RELATION {id CourseId, name Name, ver CourseVersion} KEY {id};

VAR enrolled BASE

RELATION {student_id StudentId, course_id CourseId, year Year}
KEY {student_id, course_id, year};

/ referential integrity constraints */*

CONSTRAINT enrolled_student_fkey

((enrolled {student_id}) RENAME {student_id AS id}) <=
student {id};

CONSTRAINT enrolled_course_fkey

((enrolled {course_id}) RENAME {course_id AS id}) <= course {id};

Referenční integritní omezení v SQL

SQL (částečně) implementuje pomocí cizích klíčů: pro $\rho_f(\pi_{R'}(r)) \subseteq \pi_{S'}(s)$ je

- R' **cizí klíč** relační proměnné r , který se odkazuje na atributy S' proměnné s
- nutný předpoklad: S' musí být klíč v s (**PRIMARY KEY** nebo **UNIQUE**, !!)

jednoatributový cizí klíč (sloupcové omezení):

... **REFERENCES** $\langle jméno-tabulky \rangle$ ($\langle atribut \rangle$)

víceatributový cizí klíč (omezení v rámci celé tabulky):

FOREIGN KEY ($\langle r-atribut_1 \rangle, \langle r-atribut_2 \rangle, \dots$)

REFERENCES $\langle jméno-tabulky \rangle$ ($\langle s-atribut_1 \rangle, \langle s-atribut_2 \rangle, \dots$)

poznámka:

- jako ostatní omezení lze použít s **CONSTRAINT** a **ALTER TABLE**

Příklad (SQL: Použití REFERENCES)

```
CREATE TABLE student (  
  id NUMERIC NOT NULL PRIMARY KEY,  
  name VARCHAR NOT NULL,  
  major VARCHAR NOT NULL);
```

```
CREATE TABLE course (  
  id NUMERIC NOT NULL PRIMARY KEY,  
  name VARCHAR NOT NULL,  
  ver NUMERIC NOT NULL);
```

/ table with two foreign keys */*

```
CREATE TABLE enrolled (  
  student_id NUMERIC NOT NULL REFERENCES student (id),  
  course_id NUMERIC NOT NULL REFERENCES course (id),  
  year NUMERIC NOT NULL,  
  PRIMARY KEY (student_id, course_id, year));
```

Příklad (SQL: Použití FOREIGN KEY a REFERENCES)

```
CREATE TABLE student (  
  id NUMERIC NOT NULL PRIMARY KEY,  
  name VARCHAR NOT NULL,  
  major VARCHAR NOT NULL);
```

```
CREATE TABLE course (  
  name VARCHAR NOT NULL,  
  ver NUMERIC NOT NULL,  
  PRIMARY KEY (name, ver));
```

```
CREATE TABLE enrolled (  
  student_id NUMERIC NOT NULL REFERENCES student (id),  
  c_name VARCHAR NOT NULL,  
  c_ver NUMERIC NOT NULL,  
  FOREIGN KEY (c_name, c_ver) REFERENCES course (name, ver),  
  year NUMERIC NOT NULL,  
  PRIMARY KEY (student_id, c_name, c_ver, year));
```

Příklad (SQL: Modifikace cizích klíčů pomocí ALTER TABLE)

```
CREATE TABLE enrolled (  
    student_id NUMERIC NOT NULL,  
    c_name VARCHAR NOT NULL,  
    c_ver NUMERIC NOT NULL,  
    year NUMERIC NOT NULL,  
    PRIMARY KEY (student_id, c_name, c_ver, year));  
  
/* foreign key referencing student ids */  
ALTER TABLE enrolled ADD  
    CONSTRAINT student_id_fkey  
    FOREIGN KEY (student_id) REFERENCES student (id);  
  
/* foreign key referencing course names and versions */  
ALTER TABLE enrolled ADD  
    CONSTRAINT course_fkey  
    FOREIGN KEY (c_name, c_ver) REFERENCES course (name, ver);
```


Chování cizích klíčů v SQL

předpoklad: je dáno $\rho_{y \leftarrow x}(\pi_{\{x\}}(\mathbb{r})) \subseteq \pi_{\{y\}}(\mathbb{s})$

modifikace proměnné \mathbb{r} , které končí **chybou**:

- vložení hodnoty x do \mathbb{r} , která se nenachází mezi hodnotami y z \mathbb{s}
příklad: vložení výsledku zkoušky pro ID, které nepatří žádnému studentovi
- jako v předchozím případě, pro **UPDATE** místo **DELETE**
příklad: snaha modifikovat ID na hodnotu, která nepatří žádnému studentovi

modifikace proměnné \mathbb{s} , kdy je možné **specifikovat chování**:

- pokus smazat z \mathbb{s} n -tici s , kde $s(y)$ se stále používá v \mathbb{r}
příklad: smazání studenta, který má stále záznamy o zkoušce
- jako v předchozím případě, pro **UPDATE** místo **DELETE**
příklad: modifikace ID studenta, který má záznamy o zkoušce

Možnosti reakcí na porušení omezení v SQL

předpoklad:

- je dáno $\rho_{y \leftarrow x}(\pi_{\{x\}}(\mathbb{r})) \subseteq \pi_{\{y\}}(\mathbb{s})$
- snažíme se provést modifikaci \mathbb{s} porušující toto omezení

možné způsoby ošetření:

- 1 **NO ACTION: implicitní chování** – okamžité zastavení, nahlášení chyby
- 2 **RESTRICT: nahlášení chyby** – bez možnosti odložení kontroly (viz dále)
- 3 **CASCADE: kaskádování** – nedojde k chybě, ale změna se propaguje do tabulek, ve kterých je hodnota přítomna jako cizí klíč:
 - při **DELETE** se smažou odpovídající n -tice
 - při **UPDATE** se adekvátně změní hodnoty
- 4 **SET NULL: smazání hodnoty z n -tic** – hodnoty v tabulkách s cizím klíčem budou nedefinované (nebezpečné, PŘEDNÁŠKA 5)
- 5 **SET DEFAULT: nastavení na implicitní hodnotu** – pokud má atribut danu implicitní hodnotu pomocí **DEFAULT**

Příklad (SQL: Reakce na porušení integritních omezení)

```
CREATE TABLE enrolled (  
  student_id NUMERIC NOT NULL  
    REFERENCES student (id) ON UPDATE CASCADE ON DELETE CASCADE,  
  course_id NUMERIC NOT NULL  
    REFERENCES course (id) ON DELETE RESTRICT ON UPDATE CASCADE,  
  year NUMERIC NOT NULL,  
  PRIMARY KEY (student_id, course_id, year));  
  
/* the following removes tuples from enrolled referencing 666 */  
DELETE FROM student WHERE id = 666;  
  
/* the following updates tuples in enrolled */  
UPDATE student SET id = 777 WHERE id = 666;  
  
DROP TABLE student;           /* fail */  
DROP TABLE student RESTRICT; /* fail */  
DROP TABLE student CASCADE;  /* removes constraint, not values */
```

Pozdržené vyhodnocování integritních omezení v SQL

pro **UNIQUE**, **PRIMARY KEY** a **REFERENCES** je možné omezení deklarovat jako:

- ❶ **NOT DEFERRABLE** (implicitní hodnota) – platnost integritního omezení je testována *bezprostředně* po provedení modifikující operace
- ❷ **DEFERRABLE** s uvedením následující specifikace:
 - **INITIALLY IMMEDIATE** (implicitní hodnota) – platnost integritního omezení je možné pozdržet v rámci jedné transakce, počáteční nastavení omezení ale je provádět kontrolu okamžitě
 - **INITIALLY DEFERRED** – platnost omezení je testována až na konci transakce

modifikace režimu pozdržení v rámci transakce:

```
SET CONSTRAINTS ALL DEFERRED;  
SET CONSTRAINTS ALL IMMEDIATE;  
SET CONSTRAINTS ⟨jméno1⟩, ⟨jméno2⟩, ... DEFERRED;  
SET CONSTRAINTS ⟨jméno1⟩, ⟨jméno2⟩, ... IMMEDIATE;
```

Příklad (SQL: Problém záměny hodnot primárního klíče bez pozdržení)

```
CREATE TABLE foo (  
  x NUMERIC NOT NULL PRIMARY KEY,  
  y VARCHAR NOT NULL);  
  
/* primary key violation and transaction rollback */  
BEGIN;  
  UPDATE foo SET x = 10 WHERE y = 'Blangis'; /* fails */  
  UPDATE foo SET x = 20 WHERE y = 'Abbe';  
ROLLBACK;  
  
/* workaround (or, using deletes and inserts) */  
BEGIN;  
  UPDATE foo SET x = 666 WHERE y = 'Abbe';  
  UPDATE foo SET x = 10 WHERE y = 'Blangis';  
  UPDATE foo SET x = 20 WHERE y = 'Abbe';  
COMMIT;
```

Příklad (SQL: Záměna hodnot primárního klíče s pozdržením)

```
CREATE TABLE foo (  
  x NUMERIC NOT NULL PRIMARY KEY DEFERRABLE,  
  y VARCHAR NOT NULL);  
  
/* primary key violation and transaction rollback */  
BEGIN;  
  UPDATE foo SET x = 10 WHERE y = 'Blangis'; /* fails */  
  UPDATE foo SET x = 20 WHERE y = 'Abbe';  
ROLLBACK;  
  
/* making all initially immediate constraints deferred */  
BEGIN;  
  SET CONSTRAINTS ALL DEFERRED;  
  UPDATE foo SET x = 10 WHERE y = 'Blangis';  
  UPDATE foo SET x = 20 WHERE y = 'Abbe';  
COMMIT; /* success */
```

Příklad (SQL: Rozdíl mezi NO ACTION a RESTRICT)

```
CREATE TABLE foo (  
  x NUMERIC NOT NULL PRIMARY KEY);  
  
/* foreign key with NO ACTION */  
CREATE TABLE bar (  
  x NUMERIC NOT NULL PRIMARY KEY  
  REFERENCES foo (x) ON DELETE NO ACTION DEFERRABLE);  
  
/* foreign key with RESTRICT */  
CREATE TABLE baz (  
  x NUMERIC NOT NULL PRIMARY KEY  
  REFERENCES foo (x) ON DELETE RESTRICT DEFERRABLE);  
  
INSERT INTO foo VALUES (10), (20), (30);  
INSERT INTO bar VALUES (10), (20);  
INSERT INTO baz VALUES (10);
```

Příklad (SQL: Příklad užití NO ACTION a RESTRICT)

```
INSERT INTO foo VALUES (10), (20), (30);  
INSERT INTO bar VALUES (10), (20);  
INSERT INTO baz VALUES (10);
```

```
BEGIN;  
  SET CONSTRAINTS ALL DEFERRED;  
  DELETE FROM foo WHERE x = 20;  
  INSERT INTO foo VALUES (20);  
COMMIT; /* succes */
```

```
/* RESTRICT in table "baz" cannot be deferred */
```

```
BEGIN;  
  SET CONSTRAINTS ALL DEFERRED;  
  DELETE FROM foo WHERE x = 10; /* fail */  
  INSERT INTO foo VALUES (10);  
ROLLBACK;
```


Funkční závislosti

motivace:

Formule $A \Rightarrow B$ ve tvaru implikace mezi konjunkcemi atributů vyjadřující podmínky ve tvaru když-pak – pokud jsou si hodnoty rovny na attributech z A , musí si být rovny na attributech z B , např. $\{\text{time}, \text{room}\} \Rightarrow \{\text{course}\}$.

související problémy:

- modely a sémantické vyplývání
- algoritmy pro rozhodování, jestli formule plyne z množiny formulí
- axiomatizace, automatické dokazovací systémy
- grafické dokazovací metody a normalizace důkazů
- minimální báze závislostí v databázových instancích
- vztah k pojmům nadklíč, klíč,...
- vztahy k fragmentům výrokových a predikátových logik
-

Syntax a sémantika funkčních závislostí

Definice (funkční závislost, angl.: *functional dependency*)

Nechť R je relační schéma. Pak **funkční závislost** nad schématem R je formule ve tvaru $A \Rightarrow B$, kde $A, B \subseteq R$.

Definice (pravdivost funkční závislosti v datech)

Nechť R je relační schéma a \mathcal{D} je relace nad schématem R . Pak funkční závislost $A \Rightarrow B$ nad schématem R je **pravdivá** v \mathcal{D} , což označujeme $\mathcal{D} \models A \Rightarrow B$, pokud pro každé n -tice $r_1, r_2 \in \mathcal{D}$ platí:

pokud $r_1(A) = r_2(A)$, pak $r_1(B) = r_2(B)$.

V opačném případě říkáme, že $A \Rightarrow B$ neplatí v \mathcal{D} a píšeme $\mathcal{D} \not\models A \Rightarrow B$.

slovně: $\mathcal{D} \models A \Rightarrow B$ právě tehdy, když pro každé dvě n -tice r_1 a r_2 z \mathcal{D} platí, že pokud jsou si rovny na všech natributech z množiny A , pak jsou si rovny i na všech atributech z množiny B .

Příklad (Funkční závislosti, které jsou/nejsou pravdivé v \mathcal{D})

mějme relaci $\mathcal{D} = \{r_1, r_2, r_3, r_4\}$ nad schématem $R = \{\text{FOO}, \text{BAR}, \text{BAZ}, \text{QUX}\}$:

FOO	BAR	BAZ	QUX
10	22	a	222
10	33	b	333
10	22	a	444
20	33	a	555

$$r_1 = \{\langle \text{FOO}, 10 \rangle, \langle \text{BAR}, 22 \rangle, \langle \text{BAZ}, a \rangle, \langle \text{QUX}, 222 \rangle\}$$

$$r_2 = \{\langle \text{FOO}, 10 \rangle, \langle \text{BAR}, 33 \rangle, \langle \text{BAZ}, b \rangle, \langle \text{QUX}, 333 \rangle\}$$

$$r_3 = \{\langle \text{FOO}, 10 \rangle, \langle \text{BAR}, 22 \rangle, \langle \text{BAZ}, a \rangle, \langle \text{QUX}, 444 \rangle\}$$

$$r_4 = \{\langle \text{FOO}, 20 \rangle, \langle \text{BAR}, 33 \rangle, \langle \text{BAZ}, a \rangle, \langle \text{QUX}, 555 \rangle\}$$

$$\mathcal{D} \not\models \{\text{BAZ}\} \Rightarrow \{\text{BAR}\} \quad (\text{kvůli } r_1 \text{ a } r_4)$$

$$\mathcal{D} \not\models \{\text{FOO}, \text{BAZ}\} \Rightarrow \{\text{QUX}\} \quad (\text{kvůli } r_1 \text{ a } r_3)$$

$$\mathcal{D} \not\models \{\text{BAR}\} \Rightarrow \{\text{BAZ}\} \quad (\text{kvůli } r_2 \text{ a } r_4)$$

$$\mathcal{D} \models \{\text{FOO}, \text{BAR}\} \Rightarrow \{\text{BAZ}\}$$

$$\mathcal{D} \models \{\text{QUX}\} \Rightarrow S \quad (\text{pro jakékoliv } S \text{ je triviálně splněná})$$

$$\mathcal{D} \models \{\text{FOO}, \text{BAR}\} \Rightarrow \{\text{FOO}\}$$

Funkční závislosti jako integritní omezení

motivace:

Analogicky jako (nad)klíče lze chápat funkční závislosti jako integritní omezení omezující přípustné hodnoty relačních proměnných.

integritní omezení: $A \Rightarrow B$ musí být pravdivá v hodnotě \mathfrak{r} nad $R \supseteq A \cup B$:
pro $A = \{y_1, \dots, y_k\}$, $B = \{z_1, \dots, z_l\}$ uvažujeme

$$\sigma_{y_1=y'_1 \wedge \dots \wedge y_k=y'_k \wedge \neg(z_1=z'_1 \wedge \dots \wedge z_l=z'_l)} \left(\pi_{A \cup B}(\mathfrak{r}) \bowtie \right. \\ \left. \rho_{y'_1 \leftarrow y_1, \dots, y'_k \leftarrow y_k, z'_1 \leftarrow z_1, \dots, z'_l \leftarrow z_l}(\pi_{A \cup B}(\mathfrak{r})) \right) = \emptyset_{A \cup B},$$

předchozí výraz je pravdivý v \mathcal{D} pokud pro každé dvě n -tice $r_1, r_2 \in \mathfrak{r}^{\mathcal{D}}$ platí:

pokud $r_1(A) = r_2(A)$, pak $r_1(B) = r_2(B)$

Příklad (Tutorial D: Funkční závislosti jako integritní omezení)

```
VAR rt BASE
  RELATION {foo INT, bar INT, baz INT, qux INT}
  KEY {qux};

/* define constraint: {foo, bar} => {baz} */
CONSTRAINT fubar_fd
  IS_EMPTY ((rt TIMES (rt RENAME {PREFIX "" AS "n"}))
    WHERE AND {foo = nfoo, bar = nbar} AND
      NOT (baz = nbaz));

/* define constraint: {foo} => {bar, baz} */
CONSTRAINT barbaz_fd
  IS_EMPTY ((rt TIMES (rt RENAME {PREFIX "" AS "n"}))
    WHERE AND (foo = nfoo) AND
      NOT AND {bar = nbar, baz = nbaz});
```

Příklad (SQL: Funkční závislosti jako integritní omezení)

```
CREATE OR REPLACE FUNCTION fubar_fd_trg () RETURNS TRIGGER AS $$
BEGIN
    IF (0 < (SELECT count (*) FROM tab
            WHERE foo = NEW.foo AND bar = NEW.bar AND
            NOT (baz = NEW.baz))) THEN
        RAISE EXCEPTION 'FD_{foo,bar}=>_{baz}violated!';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER fubar_fd BEFORE INSERT OR UPDATE ON tab
FOR EACH ROW EXECUTE PROCEDURE fubar_fd_trg ();
```

Triviální funkční závislosti

$A \Rightarrow B$ se nazývá **triviální**, pokud $\mathcal{D} \models A \Rightarrow B$ pro každou \mathcal{D} nad $R \supseteq A \cup B$

Věta (Charakterizace triviálních funkčních závislostí)

$A \Rightarrow B$ je triviální právě tehdy, když $B \subseteq A$.

Důkaz.

Pokud $B \not\subseteq A$, pak existuje $y \in B$ tak, že $y \notin A$. Uvažujme $\mathcal{D} = \{r_1, r_2\}$ nad $A \cup B$ tak, že $r_1(z) = 1$ pro každé $z \in A \cup B$ a

$$r_2(z) = \begin{cases} 1, & \text{pokud } z \in A, \\ 0, & \text{jinak.} \end{cases}$$

Pak zřejmě $\mathcal{D} \not\models A \Rightarrow B$, protože $r_1(A) = r_2(A)$, ale $r_1(y) \neq r_2(y)$ a $y \in B$.

Obráceně předpokládejme, že $B \subseteq A$. Pokud pro $r_1, r_2 \in \mathcal{D}$ platí, že $r_1(A) = r_2(A)$, tím spíš $r_1(B) = r_2(B)$ a tedy $\mathcal{D} \models A \Rightarrow B$. □

Věta (Vztah k pojmu funkce)

$\mathcal{D} \models A \Rightarrow B$ právě tehdy, když je $f = \{\langle r(A), r(B) \rangle \mid r \in \mathcal{D}\}$ zobrazení.

Důkaz.

Předpokládejme, že platí $\mathcal{D} \models A \Rightarrow B$ a uvažujme $\langle s, s_1 \rangle \in f$, $\langle s, s_2 \rangle \in f$. Stačí ukázat, že $s_1 = s_2$. Dle definice f existují $r_1, r_2 \in \mathcal{D}$ tak, že $r_1(A) = s = r_2(A)$. Odtud ale ihned užitím $\mathcal{D} \models A \Rightarrow B$ pro r_1 a r_2 dostáváme, že $r_1(B) = r_2(B)$, což znamená $s_1 = r_1(B) = r_2(B) = s_2$.

Obráceně, předpokládejme, že f je zobrazení a vezměme libovolné $r_1, r_2 \in \mathcal{D}$ takové, že $r_1(A) = r_2(A)$. Dle definice pravdivosti $A \Rightarrow B$ v \mathcal{D} stačí ukázat, že $r_1(B) = r_2(B)$. Dle definice f , $\langle r_1(A), r_1(B) \rangle \in f$ a $\langle r_2(A), r_2(B) \rangle \in f$. Jelikož je f zobrazení a $r_1(A) = r_2(A)$, musí platit $r_1(B) = r_2(B)$. □

poznámka:

- $\mathcal{D} \models A \Rightarrow B$ p. k. f z věty je zobrazení z $\pi_A(\mathcal{D})$ do $\pi_B(\mathcal{D})$

Přednáška 7: Závěr

pojmy k zapamatování:

- relační algebra, relační a skalární výrazy
- integritní omezení, konzistence, referenční integrita, cizí klíče
- pozdržené vyhodnocení integritních omezení
- funkční závislosti, jejich syntaxe a sémantika, vztah k pojmu zobrazení

použité zdroje:



Date C. J.: *Database in Depth: Relational Theory for Practitioners*
O'Reilly Media 2005, ISBN 978-0596100124



Date C. J., Darwen H.: *Databases, Types and the Relational Model*
Addison Wesley 2006, ISBN 978-0321399427



Maier D: *Theory of Relational Databases*
Computer Science Press 1983, ISBN 978-0914894421