7. Relačná integrita

Relačná integrita je časťou relačného modelu a je v súčasnej dobe jednou z najprepracovávanejších oblastí relačných databázových systémov. Je veľmi dôležité, aby :

- Hodnoty uložené v DB reprezentovali realitu modelovanú v danom systéme
- Hodnoty dát uložené v DB boli správne a mali zmysel

7.1 Klasifikácia integritných obmedzení

Samotná relačná integrita je potom reprezentovaná nasledovnou klasifikáciou integritných obmedzení:

1. Doménová integrita D-Integrita
2. Integrita stĺpcov C-integrita
3. Integrita entít E-integrita
4. Referenčná integrita R-integrita
5. Užívateľská integrita U-integrita

Problematika relačnej integrity je jednou z častí RDBS, ktorá sa rozvíjala v posledných rokoch veľmi dynamicky a v každom relačnom databázovom systéme je nutné zabezpečiť, aby podporoval aspoň:

- **Integritu entít**, ktorá je založená na pojme **primárneho kľúča**.

Referenčnú integritu, ktorá je založená na pojme **cudzieho** (spojovacieho) a **primárneho kľúča**.

Referenčná integrita je podporovaná niektorými implementáciami SQL a je súčasťou štandardu SQL2. Súvisí to priamo so sémantikou konceptuálnych modelov a transformáciami konceptuálnych schém do databázových.

7.2 Spracovanie IO v SRBD

Pretože je nutné správne spracovať IO, preto v relačných systémoch sú definované Coddom [Codd90] dve fázy kontroly spracovania IO:

- stĺpcová (TC)
- transakčná (viactabuľková) (TT)

V stĺpcovej fáze kontrola integrity nie je nikdy vykonávaná SRBD neskôr ako na konci spracovania akejkoľvek požiadavky relačného typu. Zvyčajne je požiadavka súčasťou aplikačného programu, alebo interaktívne zadaná užívateľom.

Pri viactabuľkovom spracovaní sa kontrola IO vykonáva SRBD na konci transakcie, ktorej súčasťou je daná požiadavka.

Informácie o všetkých typoch IO musia byť súčasťou katalógov, v ktorých SRBD vyhľadáva. SRBD pre spracovanie IO obsahuje nasledovný algoritmus:

1. Na začiatku spracovanie požiadavky SRBD zistí, ktoré IO a akého typu sa viažu na danú požiadavku.
2. SRBD pre danú požiadavku určí stĺpcové IO, t. j. také, ktoré sa týkajú len jednej tabuľky.
3. Pred ukončením spracovania požiadavky SRBD zistí, či uvedená požiadavka vyhovuje definovaným stĺpcovým IO.
4. SRBD spracuje IO, ktoré sa týka viacerých tabuliek.
5. SRBD skontroluje ešte pre potvrdením zmien či požiadavka vyhovuje IO.

7.3 Primárny kľúč

Definícia - Kandidát PK (KPK)
Je taká množina atribútov, ktorá spĺňa podmienku:

- ***Unikátnosti*** – neexistuje v relácii dve a viac n-tíc, ktoré majú rovnaké hodnoty atribútov tvoriacich KPK.
- ***Minimálnosti*** *(Neredukovatelnosť)* - žiadna podmnožina atribútov KPK nespĺňa podmienku unikátnosti.

♦

Definícia - Primárny kľúč (PK)
Je práve jedna množina atribútov tvoriaca jeden z kandidátov primárneho kľúča, ktorá spĺňa podmienky:

- **Unikátnosti**
- **Minimálnosti** (Neredukovatelnosť)

♦

 Označenie
V relačnej schéme primárny kľúč označujeme pomocou znaku ***#****.*

Každá schéma relácie musí mať definovaný práve jeden primárny kľúč.

Entity, ktoré majú málo atribútov a nie sú schopné vytvoriť primárny kľúč sa nazývajú **slabé entity** (podriadená entita je obvykle slabou entitou, a potom sa primárny kľúč slabej entity definuje ako primárny kľúč silnej entity + diskriminant, ktorý umožní rozlíšiť slabé entity). Entity, ktoré majú dostatok atribútov sa nazývajú **silné entity.** Tento problém je však nutné riešiť pri tvorbe konceptuálneho modelu. (viď kapitolu 4) .

Definícia - Alternatívny kľúč (AK)
Je taká množina atribútov, ktorá tvorí kandidáta primárneho kľúča, ale nie je primárnym kľúčom.

♦

Definícia - Super kľúč (SUPERKEY)
Superkľúč je množina atribútov relácie R, ktorá obsahuje kandidáta primárneho kľúča.

♦

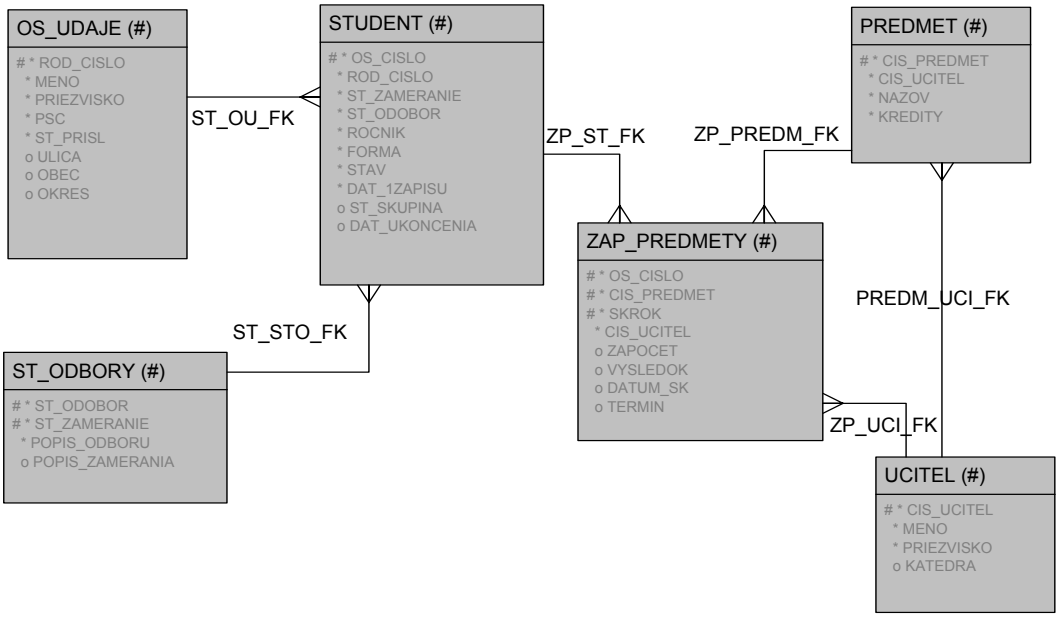
Poznámka
Superkľúč je množina atribútov, ktorá spĺňa podmienku jednoznačnosti, ale nemusí spĺňať podmienku minimálnosti.

7.4 Cudzí kľúč (Foreign key)

Definícia - Cudzí kľúč (Foreign key)
Cudzí kľúč FK je množina atribútov z relácie R taká, že existuje relácia R´ s kandidátom primárneho kľúča KPK, ktorého množina atribútov je identická s atribútmi tvoriacimi FK.
♦

Poznámka
V staršej literatúre sa stretávame s definíciou, ktorá sa viaže len na existenciu PK v inej relácii (nie KPK).

Príklad 7.6



V tomto príklade vidíme, že primárne a cudzie kľúče jednotlivých relácií sú nasledovné:

Relácia	Primárny kľúč	Cudzie kľúče
Os_udaje	{rod_cislo}	
Student	{os_cislo}	{rod_cislo} {st_odbor, st_zameranie}
Zap_predmety	{os_cislo, cis_predmet,skrok}	{os_cislo} {cis_predmet} {cis_ucitel}
Predmet	{cis_predmet}	{cis_ucitel}
Ucitel	{cis_ucitel}	
St_odbory	{st_odbor, st_zameranie}	

Pre cudzí kľúč platí [Date96]:

- FK môže byť množinou atribútov (teda môže byť kompozitný).
- Každá hodnota FK, ktorá sa objaví v relácii sa musí vyskytovať ako hodnota PK v inej relácii (opačne to neplatí).
- FK je kompozitný len ak PK je kompozitný.
- Každý z atribútov tvoriaci FK musí mať definovanú rovnakú doménu ako príslušný atribút KPK.
- Nie je nutné, aby FK bol KPK alebo jeho súčasťou. Vo všeobecnosti každý atribút relácie alebo množina atribútov relácie môžu byť definované ako FK.
- FK je odkazom na riadok relácie, kde hodnota KPK je totožná s hodnotou FK.
- Môže byť definovaný referenčný diagram R_{i-1}<-R_i->R_{r+1}

  (napr.: Student<-Zap_predmety->Predmet), kde tento diagram vyjadruje väzby medzi reláciami definované pomocou FK. Referenčný diagram predstavuje orientovaný graf, v ktorom existuje orientovaná hrana predstavujúca prepojenie v smere FK->KPK, medzi dvomi reláciami, ktoré sú vrcholmi grafu.
- môže byť definovaná referenčná cesta R_n->R_{n-1}->..->R_2->R_1, kde referenčná cesta predstavuje postupnosť relácií, medzi ktorými existuje väzba definovaná pomocou FK. Referenčná cesta predstavuje orientovanú cestu v grafe, kde orientácia hrany je v zmysle FK ->KPK.
- Relácie R1 s KPK a relácia R2 s FK môžu byť totožné (rekurzívny vzťah – kapitola 4.4.5.8).
- V relácii sa môže vyskytnúť referenčný cyklus. Napr.: R_n->R_{n-1}->..->R_2->R_1->R_n. Referenčný cyklus predstavuje cyklus v orientovanom grafe (v referenčnom diagrame).
- FK je vyjadrením vzťahu medzi dvojicou n-tíc a reprezentuje spojenie medzi reláciami.

7.5 Doménová integrita

Doménová integrita reprezentuje množinu integritných obmedzení, ktoré zdieľajú všetky hodnoty atribútov priradených k tejto doméne.

Doménové integritné obmedzenia sú:

- Typ dát
- Množina prípustných hodnôt
- Usporiadatelnosť – t. j., či je možné pre porovnanie hodnôt domény použiť relačný operátor >, >=, =< alebo < .

Príklad 7.7

- *Použitie s podporou SQL*

Definovanie domény pre osobné číslo v rozsahu dátového typu integer

```
CREATE DOMAIN DOM_OC
    AS INTEGER;
```

Definovanie domény pre osobné číslo v rozsahu dátového typu smallint s obmedzením na interval <1,5000>

```
CREATE DOMAIN DOM_OC
    AS SMALLINT,
    CHECK DOM_OC>=1 AND DOM_OC<=5000;
```

Definovanie domény pre osobné číslo v rozsahu dátového typu smallint s obmedzením na interval <1,3000> s podmienkou možnosti usporiadania

```
CREATE DOMAIN DOM_OC
    AS SMALLINT,
    CHECK DOM_OC>=1 AND DOM_OC<=3000,
    ORDERABILITY
```

Definovanie domény pre dátum ukončenia v rozsahu dátového typu DATE s obmedzením na interval od 1.1.1990 do 31.12.2500

```
CREATE DOMAIN DOM_ROK
    AS DATE
    CHECK VALUE >= 1.1.1990 AND VALUE <= 31.12.2500;
```

Definovanie domény pre stav štúdia v rozsahu definovanom vymenovanou množinou s prednastavenou hodnotou stav=S

```
CREATE DOMAIN DOM_STAV
    AS CHAR(1)
    DEFAULT  "S"
    CHECK VALUE IN ("S","K", "P", "E");
```

Definovanie domény pre stav štúdia v rozsahu definovanom množinou, ktorá je obsahom relácie CIS_STAVY s prednastavenou hodnotou stav=S

```
CREATE DOMAIN STAV
    AS CHAR(1)
    DEFAULT  "S"
    CHECK VALUE IN (SELECT znak FROM CIS_STAVY);
```

- *Použitie pri definícii tabuľky:*

```
CREATE TABLE T(
    OC DOM_OC,
    DATUM DOM_ROK,
    PLAT SMALLINT,
    ....
    );
```

Vo väčšine RDBS je podpora doménovej integrity slabá alebo žiadna. Napriek tomu je možné tento problém riešiť. Najvhodnejším spôsobom môže byť vytvorenie špecializovaných relácií reprezentujúcich doménu (číselníky) a pri operáciách INSERT a UPDATE kontrolovať správnosť použitých hodnôt.

Príklad 7.8 – Použitie definície domény ako tabuľky:
V tomto príklade v tabuľke CISELNIK_STAVY uchovávame množinu prípustných hodnôt atribútu stav relácie Student, ktorú môžeme kontrolovať pri vykonávaní operácií nad databázou.

```
CREATE TABLE CISELNIK_STAVY(
  STAV CHAR(1) NOT NULL UNIQUE
  );
```

- ***Použitie v operácii UPDATE***

V príklade povolíme vykonanie operácie UPDATE len v tom prípade, ak hodnota premennej prem_stav sa nachádza v tabuľke Ciselnik_stavy.

```
BEGIN WORK
  SELECT COUNT(*) pocet FROM CISELNIK_STAVY
        WHERE STAV = prem_stav
  IF pocet > 0THEN
        UPDATE student SET stav=prem_stav
          WHERE OC=prem_oc
  END IF
COMMIT;
```

- ***Príklad pre INSERT***

V príklade povolíme vykonanie operácie INSERT len v tom prípade, ak hodnota premennej prem_stav sa nachádza v tabuľke Ciselnik_stavy.

```
BEGIN WORK
  SELECT COUNT(*) pocet FROM CISELNIK_STAVY
        WHERE STAV = prem_stav
  IF pocet > 0THEN
        INSERT INTO student (oc, stav)
            VALUES (prem_oc, prem_stav)
  END IF
COMMIT;
```

Príklad 7.9 s riešením výnimočných stavov
V príklade operácia UPDATE bude vykonaná len v tom prípade, ak hodnota premennej prem_stav sa nachádza v množine definovanej doménou pre atribút stav.

```
BEGIN WORK
  WHENEVER ERROR CONTINUE
    UPDATE student SET stav=prem_stav
      WHERE OC=prem_oc
  WHENEVER ERROR STOP
  IF status < 0 THEN ROLLBACK
              ELSE COMMIT
  END IF
```

7.6 Stĺpcová integrita

Stĺpcové IO sú:

1. Dodatočné IO pre rozsah hodnôt, ktoré sú podmnožinou príslušnej domény
2. NULL alebo NOT NULL
3. DISTINCT alebo DUPLICATE

Príklad 7.10 – Použitie pri definícii tabuľky:

nepovolenie vkladania nedefinovaných hodnôt

```
CREATE TABLE T(
 OC SMALLINT NOT NULL,
 DATUM  DATE,
 PLAT SMALLINT ,
 ....
);
```

nepovolenie duplicít hodnôt

```
CREATE TABLE T(
 OC SMALLINT UNIQUE,
 DATUM  DATE,
 PLAT SMALLINT ,
 ....
);
```

nepovolenie vkladania nedefinovaných hodnôt a duplicít

```
CREATE TABLE T(
 OC SMALLINT NOT NULL UNIQUE,
 DATUM  DATE,
 PLAT SMALLINT ,
 ....
);
```

implicitne definované meno IO

```
CREATE TABLE T(
 OC SMALLINT NOT NULL UNIQUE,
 DATUM  DATE,
 PLAT SMALLINT NOT NULL,
 ....
 CHECK PLAT >=5000);
```

s pomenovaným IO

```
CREATE TABLE T(
 OC SMALLINT NOT NULL UNIQUE,
 DATUM  DATE,
 PLAT SMALLINT NOT NULL,
 ....
 CONSTRAINT IO_PLAT CHECK PLAT >=5000);
```

s dodatočným obmedzením hodnôt atribútu s definovanou doménou

```
CREATE TABLE T(
 OC DOM_OC NOT NULL UNIQUE,
 DATUM  DATE,
 PLAT SMALLINT NOT NULL,
 ....
 CHECK OC >=5000);
```

7.7 Integrita entít

Definícia - Integrita entít
Atribút, ktorý je súčasťou PK, nesmie nadobúdať nedefinované hodnoty (atribút alebo skupina atribútov je definovaná ako NOT NULL).

♦

Príklad 7.11 -SQL

jednoduchý atribút primárneho kľúča

```
CREATE TABLE  T1(…
ID SMALLINT NOT NULL UNIQUE ,
A1 SMALLINT
…
);
```

alebo s definíciou jednoznačného indexu

```
CREATE TABLE  T1(
…
ID1 SMALLINT NOT NULL,
A1 SMALLINT
…
);

CREATE UNIQUE INDEX uix1 ON T1(ID1);
```

kompozitný primárny kľúč

```
CREATE TABLE  T1(
…
ID1 SMALLINT NOT NULL,
ID2 SMALLINT NOT NULL,
A1 SMALLINT
…
);

CREATE UNIQUE INDEX uix1 ON T1(ID1, ID2);
```

- *SQL 92*

```
CREATE TABLE  T1(
…
ID CHAR(1) PRIMARY KEY ,
A1 SMALLINT
…
);

CREATE TABLE  T1(
ID CHAR(1),
A1 SMALLINT,
PRIMARY KEY (ID),
…
);
```

pre kompozitný PK

```
CREATE TABLE  T1(
ID1 CHAR(1),
ID2 SMALLINT,
A1 SMALLINT,
PRIMARY KEY (ID1,ID2),
…
);
```

pre pomenované IO pre kompozitný PK

```
CREATE TABLE  T1(
ID1 CHAR(1),
ID2 SMALLINT,
A1 SMALLINT,
CONSTRAINT  pk_t1 PRIMARY KEY (ID1,ID2),
…
);
```

- *Príklad 7.13* - ***Import dát bez ochrany***

```
INSERT INTO Student
   SELECT  * FROM vst_data;
```

- *Príklad 7.14* - ***Import dát s eliminovaním duplicít na vstupe***

```
INSERT INTO Student
   SELECT  DISTINCT *  FROM vst_data;
```

V prípade potreby vypísania dát, ktoré sú vo vstupnej množine duplicitné, je možné použiť nasledovný select:

```
SELECT zoznam_stlpcov, count(*) FROM vst_data
 GROUP BY zoznam_stlpcov
 HAVING count(*)>1;
```

- *Príklad 7.15* - ***Import dát s kontrolou duplicít v cieľovej množine***

```
INSERT INTO Student
   SELECT  DISTINCT *  FROM vst_data
     WHERE vst_data.os_cislo NOT IN
       (SELECT oc FROM student);
```

V prípade, že by sme chceli poznať dáta kde je porušená integrita entít, môžeme pred operáciou insert použiť nasledovný select:

```
SELECT  *  FROM vst_data
 WHERE vst_data.os_cislo IN
   (SELECT oc FROM student);
```

- *Príklad 7.16* - ***Import dát so zabránením vstupu nedefinovaných hodnôt***

```
INSERT INTO Student
   SELECT  * FROM vst_data
       WHERE vst_data.os_cislo IS NOT NULL
           AND vst_data.os_cislo NOT IN
             (SELECT oc FROM student);
```

V prípade, že by sme chceli zistiť porušenie integrity entít, môžeme pred operáciou insert použiť nasledovný select:

```
SELECT  *  FROM vst_data
   WHERE vst_data.os_cislo  IS NULL
     OR vst_data.os_cislo IN
       (SELECT oc FROM student);
```

7.8 Referenčná integrita

Definícia – Pravidlo referenčnej integrity
Cudzí kľúč môže nadobúdať hodnotu primárneho kľúča referencovanej relácie, alebo nedefinovanú hodnotu.

♦

Príklad 7.18
Ak máme dve relácie T1 a T2, potom referenčnú integritu týchto relácií môžeme pomocou príkazov SQL zabezpečiť deklaratívnym spôsobom nasledovne:

```
CREATE TABLE  T1(
    ID_T1 SMALLINT NOT NULL,
    ...
    PRIMARY KEY (ID_T1)
);

CREATE TABLE  T2(
    ID_T2 SMALLINT NOT NULL,
    FK_T2 SMALLINT,
…
PRIMARY KEY (ID_T2),
FOREIGN KEY (FK_T2) REFERENCES T1(ID_T1)
);
```

V prípade, že verzia jazyka SQL nedovoľuje použiť klauzulu PRIMARY KEY a definovanie cudzieho kľúča pomocou klauzuly FOREIGN KEY je nutné, aby programátori tieto väzby poznali a ošetrili pri deštruktívnych operáciách nad databázou.

7.8.1 Dôsledky referenčnej integrity

Z hľadiska zabezpečenia konzistencie databázy je nutné zvážiť, ktoré operácie (typu DELETE, UPDATE) budú odmietnuté, alebo akceptované. Existujú dve základné otázky:

- Čo sa môže stať, ak sa pokúsime vymazať riadok relácie, na ktorý existuje odkaz (cudzí kľúč v inej relácii s hodnotou primárneho kľúča rušeného riadku)?

  Napr. pokúsime sa zmazať predmet, ktorý majú zapísaný nejakí študenti. Odpoveďou by mohli byť pre tento príklad tri možnosti:

  - povoliť túto operáciu. V prípade, že ju povolíme, je nutné zabezpečiť kaskádovité zrušenie všetkých riadkov, ktoré sa odkazujú na rušený riadok základnej tabuľky
  - odmietnuť túto operáciu
  - okrem toho môže nastať situácia, že chceme povoliť zrušenie riadku v hlavnej relácii, ale zároveň zachovať všetky riadky v podriadenej relácii (referencujúcej relácii). Aby bolo dodržané pravidlo referenčnej integrity je nutné zmeniť hodnotu FK na nedefinovanú hodnotu (NULL)

- Čo sa môže stať keď sa pokúsime zmeniť hodnotu primárneho kľúča, na ktorý existuje referencia v podriadenej tabuľke?

  Vo všeobecnosti opätovne existujú tri možnosti:

  - odmietnuť vykonanie takejto operácie
  - povoliť operáciu s kaskádovitou zmenou hodnôt cudzích kľúčov
  - povoliť s nastavením hodnoty cudzieho kľúča na nedefinovanú hodnotu

V jazyku SQL je možné pre operáciu DELETE použiť pre voľbu režimu práce jednu z možností:

- RESTRICTED
- CASCADE
- NULLIFIED

a pre operáciu UPDATE použiť pre voľbu režimu práce tiež jednu z možností:

- RESTRICTED
- CASCADE
- NULLIFIED

Poznámka: ***Default***
Niektoré verzie jazyka SQL, vrátane SQL92 umožňujú použiť voľbu DEFAULT, čo znamená, že namiesto vkladania nedefinovanej hodnoty do cudzieho kľúča je jeho hodnota nastavená na prednastavenú hodnotu (DEFAULT).

Príklad 7.19

```
CREATE TABLE  T1(
…
PRIMARY KEY (ID),
FOREIGN  KEY (ID_T2) REFERENCES T2 ON DELETE CASCADE
ON UPDATE RESTRICTED,
….  );
```

Príklad 7.20 - Pomenované IO

```
CREATE TABLE  T1(
…
CONSTRAINT PKT1 PRIMARY KEY (ID),
CONSTRAINT FKT1 FOREIGN  KEY (ID_T2) REFERENCES T2 ON DELETE
CASCADE
ON UPDATE RESTRICTED,
….  );
```

Príklad 7.21 – Procedurálny prístup

Zmena osobného čísla študenta

Majme zjednodušenú relačnú schému
Student(#os_cislo, meno, priezvisko)
Zap_predmety(#os_cislo,#cis_predmet,#skrok, vysledok)

Zmeníme os_cislo=100 na os_cislo=1000 s predpokladom, že os_cislo=1000 neexistuje v DB.

KASKÁDA Variant 1

```
UPDATE Student SET os_cislo=1000
   WHERE os_cislo=100;

UPDATE zap_predmety  SET os_cislo=1000
   WHERE os_cislo=100;
```

KASKÁDA Variant 2 (Dekompozícia príkazu UPDATE)

```
SELECT * FROM student
   WHERE os_cislo=100
   INTO TEMP xxx;

UPDATE xxx SET os_cislo=1000;

DELETE FROM student WHERE os_cislo=100;

INSERT INTO student SELECT * FROM xxx;
DROP TABLE xxx;

SELECT * FROM zap_predmety
   WHERE os_cislo=100
   INTO TEMP xxx;

UPDATE xxx SET os_cislo=1000;

DELETE FROM zap_predmety WHERE os_cislo=100;

INSERT INTO zap_predmety SELECT * FROM xxx;

DROP TABLE xxx;
```

KASKÁDA Variant 3 (Dekompozícia príkazu UPDATE 2)

```
SELECT * FROM student
   WHERE os_cislo=100
   INTO TEMP xxx;

DELETE FROM student WHERE os_cislo=100;

INSERT INTO student  SELECT (1000,meno, priezvisko)
                          FROM xxx;

DROP TABLE xxx;

SELECT * FROM zap_predmety
   WHERE os_cislo=100
   INTO TEMP xxx;

DELETE FROM zap_predmety WHERE os_cislo=100;
INSERT INTO zap_predmety
SELECT (1000,cis_predmet,skrok,vysledok)
   FROM xxx;

DROP TABLE xxx;
```

KASKÁDA Variant 4 (Predpokladáme existenciu oc=1000)

```
SELECT * FROM student
   WHERE os_cislo=100
   INTO TEMP xxx;

SELECT * FROM student
   WHERE os_cislo=1000
   INTO TEMP aaa;

UPDATE xxx SET os_cislo=1000
   WHERE os_cislo=100
     AND 0=(SELECT count(*) FROM student
              WHERE os_cislo=1000);

DELETE FROM student WHERE os_cislo=100;

INSERT INTO student  SELECT * FROM xxx;

DROP TABLE xxx;

UPDATE zap_predmety SET os_cislo=1000
   WHERE os_cislo=100
     AND 0=(SELECT count(*) FROM aaa)
DROP TABLE aaa;
```

KASKÁDA Variant 5 (Predpokladáme existenciu oc=1000)

```
SELECT * FROM student
   WHERE os_cislo=100
   INTO TEMP xxx;

SELECT count(*) počet FROM student
   WHERE os_cislo=1000
   INTO TEMP aaa;

UPDATE xxx SET os_cislo=1000
   WHERE os_cislo=100
     AND 0=(SELECT pocet FROM aaa);

DELETE FROM student WHERE os_cislo=100;

INSERT INTO student  SELECT * FROM xxx;

DROP TABLE xxx;

UPDATE zap_predmety SET os_cislo=1000
   WHERE os_cislo=100
     AND 0=(SELECT pocet FROM aaa);

DROP TABLE aaa;
```

REŠTRIKCIA

```
SELECT * FROM student
   WHERE os_cislo=100
   INTO TEMP xxx;


SELECT count(*) počet_st FROM student
   WHERE os_cislo=1000
   INTO TEMP aaa;
```

```
SELECT count(*) počet_zp FROM zap_predmety
   WHERE os_cislo=1000
   INTO TEMP bbb;


UPDATE xxx SET OC=1000
   WHERE 0=(SELECT počet_st FROM aaa)
     AND  0=(SELECT počet_zp FROM bbb);

DELETE FROM student WHERE OC=100;

INSERT INTO student  SELECT * FROM xxx;

DROP TABLE xxx;
```

Príklad 7.22 –Procedurálny prístup

Zmena osobného čísla učiteľa

Majme nasledovné relácie:

Predmet(#cis_predmet, nazov,cis_ucitel)

Ucitel (#cis_ucitel,meno, priezvisko)

Zmeňte osobné číslo učiteľa z hodnoty 100 na hodnotu 1000.

NULOVANIE Variant 1

```
SELECT * FROM ucitel
   WHERE cis_ucitel=100
   INTO TEMP xxx;

UPDATE xxx SET cis_ucitel=1000;

DELETE FROM ucitel WHERE cis_ucitel=100;

INSERT INTO ucitel SELECT * FROM xxx;

DROP TABLE xxx;

UPDATE Predmet SET cis_ucitel=NULL WHERE cis_ucitel=100;
```

NULOVANIE Variant 2 (Ak cis_ucitel=1000 môže existovať)

```
SELECT count(*) pocet  FROM ucitel
   WHERE cis_ucitel=100
   INTO TEMP xxx;

UPDATE ucitel SET cis_ucitel=1000
   WHERE cis_ucitel=100
     AND 0=(SELECT počet FROM xxx);

UPDATE Predmet SET cis_ucitel=NULL
   WHERE cis_ucitel=100
   AND 0=(SELECT počet FROM aaa);

DROP TABLE xxx;
```

7.9 Užívateľská integrita

Užívateľská integrita umožní zadefinovať integritné obmedzenia, ktoré podporujú logiku aplikácie. Tento druh integritných obmedzení je možno zabezpečiť deklaratívnym, ako aj procedurálnym spôsobom. Deklaratívny spôsob znamená, že pri definícii tabuľky zadáme kontrolu hodnôt niektorých atribútov, ktoré rozširujú možnosti doménovej a stĺpcovej integrity. Častejšie je potrebné zabezpečiť užívateľskú integritu procedurálnym spôsobom, pretože sa častokrát týka viacerých relácií a tým pádom nie je možné zabezpečiť deklaratívny spôsob. Príkladom užívateľskej integrity môže byť kontrola hodnôt dátumu nástupu do zamestnania tak, že dátum by nemal byť väčší ako systémový dátum, ďalej kontrola veku študenta alebo zamestnanca, alebo kontrola opätovného zapísania predmetu študentovi a pod.

Príklad 7.23 – Deklaratívny prístup

```
CREATE TABLE  T1(
ID SMALLINT,
DAT_NAR DATE,
DAT_NASTUPU,

…
PRIMARY KEY (ID),
CHECK DAT_NAR < DAT_NASTUPU
CHECK DAT_NASTUPU< TODAY
CHECK DAT_NASTUPU > DAT_NAR+15 YEAR

….  );
```

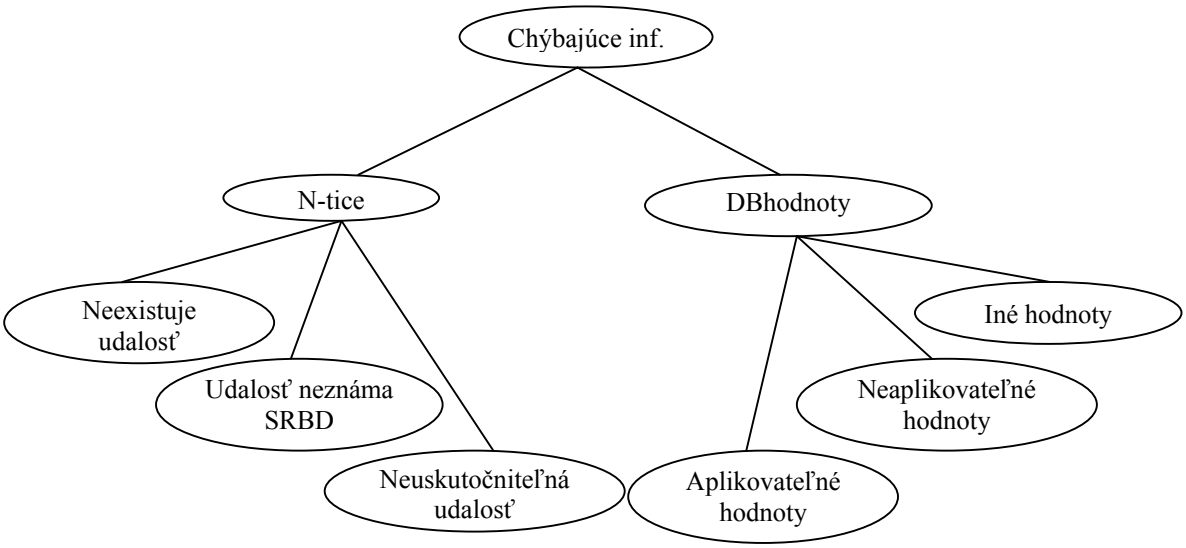
Niektoré verzie jazyka SQL, vrátane SQL92, umožňujú definovať externé integritné obmedzenie (ASSERTION), ktoré zabezpečí kontrolu vkladaných dát aj nad viacerými tabuľkami, alebo z pohľadu použitia agregačných funkcií a pod..

Príklad 7.24 – Externé integritné obmedzenie

```
CREATE ASSERTION pocet_zap_predm
CHECK   (2>=
        (SELECT COUNT(*)  FROM zap_predmety
            GROUP BY oc,cp
            WHERE vysledok IS NULL
        )
    );
```

7.10 Chýbajúce informácie

7.10.1 Klasifikácia chýbajúcich informácií:



Obr. 7.10.1Chýbajúce informácie

Samotné chýbajúce informácie je možné klasifikovať do dvoch kategórií:

1. Chýbajúce n-tice – ktoré reprezentujú chyby, pre ktoré sa do databázy nedostal celý riadok tabuľky, a to z dôvodu, že:
    - Udalosť nenastala.
    - Nastala udalosť, ktorú SRBD nebol schopný spracovať.
    - Nastala udalosť, ktorú nespracoval aplikačný systém.
2. Nedefinované hodnoty – ktoré neboli do databázy zadané, pretože boli neznáme, zabudnuté, alebo nie je povolené tieto hodnoty zadávať. Sem patria aj chybne zadané hodnoty, ktoré poukazujú na nesprávnosť vstupných dát.
   Tento druh nedefinovaných hodnôt môže byť:
    - Aplikovateľná nedefinovaná hodnota.
    - Neaplikovateľná nedefinovaná hodnota.
    - Iná hodnota.

Implementácia nedefinovaných hodnôt môže byť:

- Prázdny znakový reťazec
- Znakový reťazec z určených znakov
- Nula
- Definované číslo
- Bitový reťazec
- Príznak pre každý atribút

Pri nedefinovaných hodnotách sa stretávame s dvomi základnými druhmi, a to tzv. aplikovateľné hodnoty alebo neaplikovateľné hodnoty. Aplikovateľná hodnota NULL je takou hodnotou, ktorú môžeme kedykoľvek nahradiť jednou z hodnôt, ktorá je prvkom príslušnej domény. Neaplikovateľná hodnota NULL je takou hodnotou, pre ktorú v žiadnom prípade nie je možné dosadiť inú hodnotu.

Pri samotnom spracovaní dát sú najdôležitejšie problémy identifikované pri aritmetických operáciách, agregačných funkciách a pri aplikácii príkazov relačnej algebry (výber, spojenie, delenie). V prípade, že projektant informačného systému alebo programátor pracujúci s príslušnou databázou nerešpektuje výskyt NULL hodnôt dochádza k znehodnoteniu výsledku. Preto je potrebné pri vyššie uvedených operáciách upraviť riešenie tak, aby dávalo správne výsledky aj v prípade nedefinovateľných hodnôt.

7.10.2 Trojhodnotová logika

Trojhodnotovú logiku je možné pochopiť z tabuliek 7.10.1 až 7.10.3, ktoré poskytujú pravdivostné hodnoty pre prípad, že v logických operáciách sa objavia nedefinované hodnoty.

Označovanie :

T – TRUE (pravdivá hodnota)

F – FALSE (nepravdivá hodnota)

N - NULL- (nedefinovaná hodnota)

P – predikát

7.10.2.1 Operácia Negácia

Tab. 7.10.1 Negácia

P	NOT P
T	F
F	T
N	N

7.10.2.2 Operácia Logický súčet

Tab. 7.10.2 Logický súčet

OR	T	N	F
T	T	T	T
N	T	N	N
F	T	N	F

7.10.2.3 Operácia Logický súčin

Tab. 7.10.3 Logický súčin

AND	T	N	F
T	T	N	F
N	N	N	F
F	F	F	F

7.10.2.4 Aritmetické operácie

Ak niektorá z hodnôt, ktoré sú súčasťou aritmetického výrazu je nedefinovaná hodnota je potrebné si uvedomiť, ako táto hodnota ovplyvňuje výsledok operácie. Nech X reprezentuje platnú hodnotu časti výrazu a N reprezentuje aplikovateľnú chýbajúcu informáciu a nech operátor ? je jeden z nasledovných aritmetických operátorov:

+ (súčet)

- (rozdiel)

* (súčin)

/ (podiel)

Potom:

- Pre výsledok operácie platí:
  X?X=hodnota výrazu
  X?N=N
  N?X=N
  N?N=N
- Pre prácu s reťazcami (konkatenácia) je výsledok operácie nasledovný

Nech X, Y reprezentujú platnú hodnotu reťazca a N reprezentuje aplikovateľnú chýbajúcu informáciu a nech operátor ? je operátor konkatenácie, potom platí:

X?Y= XY
X?N=N
N?X=N
N?N=N

Príklad

Majme nasledovnú tabuľku pre reláciu OSOBA s atribútmi relačnej schémy (rod_číslo, meno, priezvisko, psc, obec).

Tab. 7.10.4 Relácia OSOBA

rod_cislo	meno	priezvisko	psc	obec
725406/8437	Erika	Grondzakova	N	N
720417/7123	Ivan	Janosik	95701	Banovce nad Bebravou
715428/8438	Stanislava	Csanyiova	02061	Lednicke Rovne
705418/5699	Blanka	Bacinova	75111	Radslavice
760404/8507	Vladimir	Labdavsky	01001	Zilina
740424/6696	Csaba	Molnar	N	Hronovce
765410/6757	Ivana	Rajtarova	95135	Velke Zaluzie
760427/7967	Slavomir	Kopinec	01701	Povazska Bystrica
760406/7669	Vladimir	Mednansky	N	Sebedrazie
760412/8125	Vladimir	David	01001	Zilina

Majme nasledovný príkaz SQL, ktorý vyjadruje operáciu výber z relačnej algebry nad reláciu OSOBA.

```
SELECT * FROM Osoba WHERE <výberová podmienka>
```

A ak výberová podmienka x je napríklad : psc="01001"

Ak definujeme požiadavku nasledovne [CAOT92] :

```
KNOWN TO BE x          musí byť x
POSSIBLY x             asi x
KNOWN NOT TO BE x      nesmie byť x
NOT (KNOWN TO BE x)    nesmie byť x
UNKNOWN                nedefinované x
NOT UNKNOWN nie        nedefinované x
```

potom ak vyjadríme pre rôzne interpretáciu požiadavky reprezentovanej SQL príkazom dostávame nasledovné výstupy:

Test	**Výsledok**
KNOWN TO BE x	David, Labdavsky
POSSIBLY x	David, Labdavsky, Grondzakova, Molnar, Mednansky
KNOWN NOT TO BE x	Janosik, Csanyiova, Bacinova, Rajtarova, Kopinec
NOT (KNOWN TO BE x)	Janosik, Csanyiova, Bacinova, Rajtarova, Kopinec, Grondzakova, Molnar, Mednansky
UNKNOWN	Grondzakova, Molnar, Mednansky
NOT UNKNOWN	Janosik, Csanyiova, Bacinova, Rajtarova, Kopinec, David, Labdavsky

Zodpovedajúce podmienky sú definované v nasledovnej tabuľke:

KNOWN TO BE x	psc="01001"
POSSIBLY x	psc="01001" OR psc IS NULL
KNOWN NOT TO BE x	psc <> "01001"
NOT (KNOWN TO BE x)	psc <> "01001"
UNKNOWN	psc IS NULL
NOT UNKNOWN	psc IS NOT NULL

Ak využijeme rozšírenie jazyka SQL od normy SQL92 o možnosť vyjadrenia výberovej podmienky v tvare:

```
<výberová podmienka >::=
```

```
<boolovský výraz> IS [NOT] {TRUE | FALSE | UNKNOWN}
```

```
potom môžeme písať v podmienke SQL príkazu:
KNOWN TO BE x          psc="01001"
POSSIBLY x             psc="01001" IS NOT FALSE
KNOWN NOT TO BE x        psc="01001" IS FALSE
NOT (KNOWN TO BE x)      psc="01001" IS NOT TRUE
UNKNOWN                  psc="01001" IS UNKNOWN
NOT UNKNOWN              psc="01001" IS NOT UNKNOWN
```

7.10.3 Štvorhodnotová logika

Označovanie :

T – TRUE (pravdivá hodnota)

F – FALSE (nepravdivá hodnota)

A - NULL- (nedefinovaná aplikovateľná hodnota)

I - NULL- (nedefinovaná neaplikovateľná hodnota)

P - predikát

7.10.3.1 Operácia Negácia

Tab. 7.10.5 Negácia

P	***NOT P***
T	F
A	A
I	I
F	T

7.10.3.2 Operácia Logický súčet

Tab. 7.10.6 Logický súčet

OR	T	A	I	**F**
T	T	T	T	T
A	T	A	A	A
I	T	A	I	F
F	T	A	F	F

7.10.3.3 Operácia Logický súčin

Tab. 7.10.7 Logický súčin

AND	T	A	I	**F**
T	T	A	I	F
A	A	A	I	F
I	I	I	I	F
F	F	F	F	F

7.10.3.4 Aritmetické operácie

Nech X reprezentuje platnú hodnotu časti výrazu, A reprezentuje aplikovateľnú chýbajúcu informáciu, I reprezentuje neaplikovateľnú chýbajúcu informáciu a nech operátor ? je jeden z nasledovných aritmetických operátorov:

+ (súčet)

- (rozdiel)

* (súčin)

/ (podiel)

Potom:

X?X=hodnota výrazu
X?A=A
A?X=A
A?A=A
A?I=I
I?A=I
I?I=I
X?I=I
I?X=I

7.10.3.5 Práca s reťazcami (Konkatenácia)

Nech X, Y reprezentujú platnú hodnotu reťazca a A, I reprezentujú aplikovateľnú resp. neaplikovateľnú chýbajúcu informáciu a nech operátor ? je operátor konkatenácie, potom platí:

X?Y= XY
X?A=A
A?X=A
A?A=A
A?I=I
I?A=I
I?I=I
X?I=I
I?X=I