Kapitola 6: Omezení integrity

- Omezení domény
- Referenční integrita
- Aserce
- Spouštěče (Triggers)
- Funkční závislosti

Omezení domény

- Omezení integrity zabraňují poškození databáze; zajišťují, že autorizované zásahy do databáze nezpůsobí ztrátu konzistence dat.
- Omezení domény je základní formou omezení integrity.
- Omezení integrity testují hodnoty vkládané do databáze a kontroluje dotazy, aby bylo zajištěno, že porovnávání dávají smysl.
- Klauzule **check** dovoluje v SQL-92 omezit domény:
  - Použijte klauzuli **check** k zajištění, že doména *hodinová-mzda* je větší než specifikovaná hodnota

    create domain *hodinová-mzda* **numeric(5,2)**
    constraint *value-test* **check** (*value* >= 4.00)
  - Doména *hodinová-mzda* je deklarována jako dekadické číslo s pěti číslicemi, z nichž 2 jsou za desetinnou čárkou
  - Doména má omezení, které zaručuje, že *hodinová-mzda* je větší než nebo rovno 4.00
  - Klauzule **constraint** *value-test* je volitelná; užitečná pro indikaci, které omezení bylo při aktualizaci porušeno.

Referenční integrita

- Zajišťuje, že hodnota, která se objeví v jedné relaci pro danou množinu atributů, se také objeví v určitých množinách atributů v jiné relaci.
  - Příklad: jestliže „Praha-Spořilov“ je jméno pobočky, které se objeví v jedné z n-tic v relaci *účet*, pak zde existuje n-tice v relaci *pobočka* pro pobočku „Praha-Spořilov“.
- Formální definice
  - Nechť r_1 (R_1) a r_2 (R_2) jsou relace s primárními klíči K_1 a K_2.
  - Podmnožina α z R_2 je *cizí klíč* odkazující na K_1 v relaci r_1, jestliže pro každé t_2 v r_2 musí být n-tice v taková, že $t_1[K_1] = t_2[\alpha]$.
  - Omezení referenční integrity: $\Pi_\alpha (r_2) \subseteq \Pi_{K1} (r_1)$

Referenční integrita v E-R modelu

- Uvažujme množinu vztahů R mezi množinami entit E_1 a E_2. Relační schéma pro R zahrnuje primární klíče K_1 z E_1 a K_2 z E_2. Pak K_1 a K_2 jsou v R cizí klíče pro relační schémata E_1 a E_2.
- Slabé množiny entit jsou též zdroje pro omezení referenční integrity. Proto musí relační schéma pro každou slabou množinu entit zahrnovat primární klíč množiny vztahů, na které závisí, tj. nadřazené entitní množiny

Modifikace databáze

- Tyto testy musí být provedeny, aby se zachovala následující mez referenční integrity:

$$\Pi_{\alpha}(r_2) \subseteq \Pi_K(r_1)$$

- **Vkládání.** Je-li n-tice t_2 vkládána do r_2, systém musí zajistit, že v r_1 je n-tice t_1 taková, že $t_1[K_1] = t_2[\alpha]$. Tj.

$$t_2[\alpha] \in \Pi_K(r_1)$$

- **Mazání.** Je-li n-tice t_1 mazána z r_1, systém musí spočítat množinu n-tic v r_2, která odkazuje na t_1:

$$\sigma_{\alpha = t1[K]}(r_2)$$

  Jestliže tato množina není prázdná, příkaz mazání skončí s chybou nebo se n-tice, které odkazují na t_1, musí samy smazat (rekurzivní mazání je možné).
- **Aktualizace.** Jsou dvě možnosti:
  - Je-li n-tice t_2 aktualizován v relaci r_2 a aktualizace modifikuje hodnoty pro cizí klíč α, pak je proveden podobný test jako při operaci vkládání. Nechť t_2‘ značí novou hodnotu n-tice t_2. Systém musí zajistit, že

$$t_2'[\alpha] \in \Pi_K(r_1)$$

  - Je-li n-tice t_1 aktualizována v r_1 a aktualizace modifikuje hodnoty primárního klíče (K), pak je prováděn podobný test jako při operaci mazání. Systém musí spočítat

$$\sigma_{\alpha = t1[K]}(r_2)$$

    s použitím staré hodnoty t_1. Není-li tato množina prázdná, aktualizace skončí s chybou nebo je změna kaskádovitě provedena na těchto n-ticích nebo mohou být tyto n-tice v relaci smazány.

Referenční integrita v SQL

- Primární a kandidátní klíče a cizí klíče můžou být specifikovány jako část SQL příkazu **create table**:
  - Klauzule **primary key** v příkazu **create table** zahrnuje seznam atributů, které tvoří primární klíč.
  - Klauzule **unique key** v příkazu **create table** zahrnuje seznam atributů, které tvoří kandidátní klíč.
  - Klauzule **foreign key** v příkazu **create table** zahrnuje seznam atributů, které tvoří cizí klíč a jméno relace odkazované cizím klíčem.

Příklad referenční integrity v SQL

create table *zákazník*
(*zákazník-jméno* char(20) **not null**,
zákazník-ulice char(30),
zákazník-město char(30),
primary key (*zákazník-jméno*))

create table *pobočka*
(*pobočka-jméno* char(15) **not null**,
pobočka-město char(30),
aktiva integer,
primary key (*pobočka-jméno*))

create table *účet*
(*pobočka-jméno* char(15),
číslo-účtu char(10) **not null**,
zůstatek integer,
primary key (*číslo-účtu*),
foreign key (*pobočka-jméno*) **references** *pobočka*)

create table *vkladatel*
(*zákazník-jméno* char(20) **not null**,
číslo-účtu char(10) **not null**,
primary key (*zákazník-jméno, číslo-účtu*),
foreign key (*číslo-účtu*) **references** *účet*,
foreign key (*zákazník-jméno*) **references** *zákazník*)

Kaskádové akce v SQL

create table *účet*
…
foreign key () **references** *pobočka*
on delete cascade
on update cascade,
…)

- Díky klauzuli **on delete cascade**, vyvolá-li mazání n-tice v relaci *pobočka* porušení referenční integrity, mazání „přeteče“ do relace *účet* a smaže n-tice, které odkazují na n-tice v relaci *pobočka*, které jsou mazány.
- Kaskádové aktualizace jsou obdobné.
- Je-li mezi více relacemi sada závislostí cizích klíčů (každý se specifikací **on delete cascade**), mazání nebo aktualizace se může rozšířit do celé této sady.
- Jestliže kaskádová aktualizace či mazání vyvolá porušení integritních omezení, které nemůže zachytit dřívější kaskádová operace, systém zruší transakci. Výsledek: všechny změny způsobené transakcí a jejími kaskádovými akcemi jsou zrušeny.

Tvrzení (Assertion)

- Tvrzení (aserce, prosazování) je predikát vyjadřující podmínku, kterou musí databáze vždy splňovat.
- V SQL-92 má tvar
  create assertion <jméno-tvrzení> **check** <predikát>
- Když je tvrzení vytvořeno, systém ji otestuje, je-li platná.

Příklad

- Suma všech částek půjček pro každou pobočku musí být menší než suma všech zůstatků účtů v pobočce.

```
create assertion mez-sumy check
    (not exists (select * from pobočka
       where (select sum (částka) from půjčka
         where půjčka.pobočka-jméno = pobočka.pobočka-jméno)
             >= (select sum (částka) from účet
         where půjčka.pobočka-jméno = pobočka.pobočka-jméno)))
```

- Každá půjčka má alespoň jednoho dlužníka, který udržuje účet s minimálním zůstatkem $1000.00.

```
create assertion mez-zůstatku check
    (not exists (select * from půjčka
    where not exists ( select *
       from dlužník, vkladatel, účet
       where půjčka.půjčka-číslo = dlužník.půjčka-číslo
          and dlužník.zákazník-jméno = vkladatel.zákazník-jméno
          and vkladatel.účet-číslo = účet.účet-číslo
          and účet.zůstatek >= 1000)))
```

Spouště (Triggers)

- *Spouštěč (Trigger,* spoušť) je příkaz, který systém automaticky spouští jako vedlejší efekt modifikace databáze.
- Pro aktivaci tohoto mechanismu musíme:
  – Specifikovat podmínky, za jakých je spouštěč prováděn.
  – Specifikovat akce, které se budou dít při jeho spuštění.
- Standard SQL-92 nezahrnuje spouště, ale mnoho implementací je podporuje.

Příklad triggeru

- Předpokládejme, že místo povolení záporných zůstatků účtů banka zachází se saldem (přečerpáním) takto:
  – nastaví zůstatek účtu na nulu
  – vytvoří půjčku s částkou salda
  – dá této půjčce číslo tohoto účtu
- Podmínka spuštění této spouště je aktualizace relace *účet*, která vrátí zápornou hodnotu *zůstatek*.

```
define trigger saldo on update of účet as T
      (if new T.zůstatek < 0
      then (insert into půjčka values
              (T.pobočka-jméno, T.číslo-účtu, - new T.zůstatek)
            insert into půjčovatel
              (select zákazník-jméno, číslo-účtu
              from vkladatel
              where T.číslo-účtu = vkladatel.číslo-účtu)
            update účet as S
            set S.zůstatek = 0
            where S.číslo-účtu = T.číslo-účtu)
       )
```

Klíčové slovo **new** před *T.zůstatek* značí, že by měla být použita hodnota *T.zůstatek* po aktualizaci; je-li vynecháno, je použita hodnota před aktualizací

Funkční závislosti

- Omezení na množině povolených (legálních) relací.
- Vyžaduje, že hodnota jisté množiny atributů jednoznačně určuje hodnotu jiné množiny atributů.
- Funkční závislost je zobecnění představy *klíče*.
- Nechť *R* je relační schéma

$$\alpha \subseteq R,\ \beta \subseteq R$$

- Funkční závislost $\alpha \rightarrow \beta$

je platná pro schéma *R* právě tehdy, když pro jakoukoli povolenou relaci *r*(*R*), jakékoli dvě n-tice t_1 a t_2 z *r* jsou stejné na atributech α, souhlasí též na atributech β. Tj.

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- *K* je superklíč pro relační schéma *R* právě tehdy, když $K \rightarrow R$
- *K* je kandidátní klíč *R* právě tehdy, když
  – $K \rightarrow R$, a
  – pro žádné $\alpha \subset K$, $\alpha \rightarrow R$
- Funkční závislosti umožňují vyjádřit omezení, které nelze vyjádřit pomocí superklíčů. Představme si schéma:

  info-půjčka = (*pobočka-jméno, půjčka-číslo, zákazník-jméno, částka*).

Očekáváme, že se bude platit tato množina funkčních závislostí:

půjčka-číslo $\rightarrow$ *částka*

půjčka-číslo $\rightarrow$ *pobočka-jméno*

ale nechceme splňovat následující:

půjčka-číslo $\rightarrow$ *zákazník-jméno*

Použití funkčních závislostí

- Funkční závislosti používáme k:
  - testování relací, jsou-li povolené na dané množině funkčních závislostí. Je-li relace *r* povolená na množině *F* funkčních závislostí, říkáme, že *r splňuje F*.
  - definování omezení na množině povolených relací; říkáme, že *F je platná na R*, když všechny povolené relace na *R* splňují množinu *F*.
- Poznámka: Některá instance relačního schématu (tj. relace) může splňovat funkční závislosti, i když tyto závislosti nejsou platné pro všechny povolené instance. Např. specifická instance *info-půjčka* může (náhodou) vyhovovat *půjčka-číslo* $\rightarrow$ *zákazník-jméno*

Uzávěr množiny funkčních závislostí

- Pro danou množinu funkčních závislostí *F* existují další funkční závislosti, které *F* logicky implikuje (tzv. *uzávěr* množiny *F*).
- *Uzávěr F* značíme F^+.
- Všechny F^+ můžeme najít pomocí Armstrongových axiomů:
  - je-li $\beta \subseteq \alpha$, pak $\alpha \rightarrow \beta$ (**reflexivita**)
  - je-li $\alpha \rightarrow \beta$, pak $\gamma\alpha \rightarrow \gamma\beta$ (**rozšíření**)
  - je-li $\alpha \rightarrow \beta$ a $\beta \rightarrow \gamma$, pak $\alpha \rightarrow \gamma$ (**tranzitivita**)

  Tyto axiomy tvoří minimální a úplnou množinu axiomů.
- Výpočet F^+ můžeme zjednodušit použitím následujících doplňkových pravidel.
  - je-li $\alpha \rightarrow \beta$ a $\alpha \rightarrow \gamma$, pak $\alpha \rightarrow \beta\gamma$ (**sjednocení**)
  - je-li $\alpha \rightarrow \beta\gamma$, pak $\alpha \rightarrow \beta$ a $\alpha \rightarrow \gamma$ (**rozklad**)
  - je-li $\alpha \rightarrow \beta$ a $\gamma\beta \rightarrow \delta$, pak $\alpha\gamma \rightarrow \delta$ (**pseudotranzitivita**)

  Tato pravidla jsou odvozena z Armstrongových axiomů.

Příklad

- *R = (A, B, C, G, H, I)*
- *F = {A* $\rightarrow$ *B*
  A $\rightarrow$ *C*
  CG $\rightarrow$ *H*
  CG $\rightarrow$ *I*
  B $\rightarrow$ *H}*
- některé prvky F^+:
  - *A* $\rightarrow$ *H*
  - *AG* $\rightarrow$ *I*
  - *CG* $\rightarrow$ *HI*

Uzávěr množiny atributů

- *Uzávěr* atributu α pod *F* (značíme α^+) definujeme jako množinu atributů, které jsou funkčními závislostmi *F* určeny z α:

$$\alpha \rightarrow \beta \text{ je z } F^+ \Leftrightarrow \beta \subseteq \alpha^+$$

- Algoritmus pro výpočet α^+

```
result := α;
while (byla změna v result) do
    for each β → γ in F do begin
        if β ⊆ result then result := result ∪ γ;
    end
```

Příklad

- *R = (A, B, C, G, H, I)*
- *F = {A → B*
  A → C
  CG → H
  CG → I
  B → H}
- (AG^+)
  - 1. krok: *result = AG*
  - 2. krok: *result = ABCG* ($A \rightarrow C$ a $A \subseteq AGB$)
  - 3. krok: *result = ABCGH* ($CG \rightarrow H$ a $CG \subseteq AGBC$)
  - 4. krok: *result = ABCGHI* ($CG \rightarrow I$ a $CG \subseteq AGBCH$)
- Je *AG* kandidátní klíč?
  - 1. $AG \rightarrow R$
  - 2. je $A^+ \rightarrow R$?
  - 3. je $G^+ \rightarrow R$?

Kanonický obal (Canonical Cover)

- Uvažujme množinu funkčních závislostí *F* a funkční závislost $\alpha \rightarrow \beta$ z *F*.
  - Atribut *A* je nadbytečný v α, jestliže $A \in \alpha$ a *F* logicky implikuje
    $$(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$$
  - Atribut *A* je nadbytečný v β, jestliže $A \in \beta$ a množina funkčních vztahů
    $$(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$$ logicky implikuje *F*.
- Kanonický obal F_C pro *F* je množina závislostí taková, že *F* logicky implikuje všechny závislosti v F_C a F_C logicky implikuje všechny závislosti ve *F* a navíc
  - žádná funkční závislost v F_C neobsahuje nadbytečný atribut
  - každá levá strana funkční závislosti z F_C je jedinečná

- Spočítejte kanonický obal pro *F*:
  repeat
    Použijeme pravidlo sjednocení pro nahrazení všech závislostí
      $\alpha_1 \rightarrow \beta_1$ a $\alpha_1 \rightarrow \beta_2$ z *F* jednou závislostí $\alpha_1 \rightarrow \beta_1 \beta_2$
    Najdeme funkční závislost $\alpha \rightarrow \beta$ s nadbytečným atributem buď
      v α nebo v β
    Je-li tento atribut nalezen, smažeme ho z $\alpha \rightarrow \beta$
  until *F* se nezměnilo

Příklad

- *R = (A, B, C)*
- *F = {*$A \rightarrow BC$
  $B \rightarrow C$
  $B \rightarrow C$
  $A \rightarrow B$
  $AB \rightarrow C$*}*
- Zkombinujeme $A \rightarrow BC$ a $A \rightarrow B$ do závislosti $A \rightarrow BC$.
- *A* je nadbytečné v $AB \rightarrow C$, protože $B \rightarrow C$ logicky implikuje $AB \rightarrow C$.
- *C* je nadbytečné v $A \rightarrow BC$, je-li $A \rightarrow BC$ logicky implikováno z $A \rightarrow B$ a $B \rightarrow C$.
- Kanonický obal je:

$$A \rightarrow B$$
$$B \rightarrow C$$