14. Databáze - relace a referenční integrita

Obsah

- Základní pojmy (relace, klíč, cizí klíč, unikátní klíč)
- Vnitřní a vnější spojení
- Referenční integrita
- Podmínky (check constraint)
- Transakce
- ER diagram
- Efektivita a příklady

Základní pojmy

Relace

V relačních databázích je **relace** matematický koncept reprezentovaný jako tabulka s řádky a sloupci. Každá relace (tabulka) představuje entitu nebo vztah v databázi.

Vlastnosti relace: - Každý řádek (záznam) představuje jednu instanci entity - Každý sloupec (atribut) představuje vlastnost entity - Každý řádek musí být unikátní (díky primárnímu klíči) - Pořadí řádků a sloupců není významné - Všechny hodnoty v daném sloupci musí být stejného datového typu

Klíče

Primární klíč (Primary Key) **Primární klíč** je atribut nebo kombinace atributů, které jednoznačně identifikují každý záznam v tabulce.

Vlastnosti primárního klíče: - Musí být unikátní v rámci tabulky - Nesmí obsahovat hodnotu NULL - Musí být stabilní (neměnný v čase) - Většinou by měl být jednoduchý a co nejkratší (efektivita)

Příklad definice tabulky s primárním klíčem:

```
CREATE TABLE zamestnanci (
    id_zamestnance INT PRIMARY KEY,
    jmeno VARCHAR(50),
    prijmeni VARCHAR(50),
    email VARCHAR(100)
);
```

Cizí klíč (Foreign Key) **Cizí klíč** je atribut nebo skupina atributů v jedné tabulce, které odkazují na primární klíč v jiné tabulce. Vytváří vztah mezi tabulkami.

Vlastnosti cizího klíče: - Odkazuje na primární klíč jiné tabulky - Může obsahovat hodnotu NULL (pokud není specifikováno jinak) - Zajišťuje referenční integritu

Příklad definice tabulky s cizím klíčem:

```
CREATE TABLE objednavky (
    id_objednavky INT PRIMARY KEY,
    id_zakaznika INT,
    datum DATE,
    celkova_cena DECIMAL(10, 2),
    FOREIGN KEY (id_zakaznika) REFERENCES zakaznici(id_zakaznika)
);
```

Unikátní klíč (Unique Key) **Unikátní klíč** je podobný primárnímu klíči, ale může obsahovat hodnoty NULL a v tabulce může být více unikátních klíčů.

Vlastnosti unikátního klíče: - Zajišťuje, že všechny hodnoty v daném sloupci nebo kombinaci sloupců jsou unikátní - Na rozdíl od primárního klíče může obsahovat NULL (ale pouze jednou) - Může jich být více v jedné tabulce

Příklad definice tabulky s unikátním klíčem:

```sql
CREATE TABLE zamestnanci (
    id_zamestnance INT PRIMARY KEY,
    jmeno VARCHAR(50),
    prijmeni VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    rodne_cislo VARCHAR(11) UNIQUE
);
```

Kandidátní klíč (Candidate Key) **Kandidátní klíč** je atribut nebo minimální kombinace atributů, které by mohly sloužit jako primární klíč (jsou unikátní a nemají NULL hodnoty). Z kandidátních klíčů se vybírá primární klíč.

Složený klíč (Composite Key) **Složený klíč** je klíč tvořený dvěma nebo více atributy.

```sql
CREATE TABLE objednavka_polozky (
    id_objednavky INT,
    id_produktu INT,
    mnozstvi INT,
    cena_za_kus DECIMAL(10, 2),
    PRIMARY KEY (id_objednavky, id_produktu),
    FOREIGN KEY (id_objednavky) REFERENCES objednavky(id_objednavky),
    FOREIGN KEY (id_produktu) REFERENCES produkty(id_produktu)
);
```

Vnitřní a vnější spojení

Spojení (JOIN) slouží k propojení dat z více tabulek na základě určitého vztahu. Existuje několik typů spojení.

Vnitřní spojení (INNER JOIN)

Vnitřní spojení vrací pouze řádky, kde existuje shoda mezi sloupci v obou tabulkách.

```sql
SELECT z.jmeno, z.prijmeni, o.nazev AS oddeleni
FROM zamestnanci z
INNER JOIN oddeleni o ON z.id_oddeleni = o.id_oddeleni;
```

Graficky:

```
Tabulka A    |    Tabulka B
-----------------------------
    A        |        B

  Výsledek
```

Vnější spojení (OUTER JOIN)

Vnější spojení vracejí všechny záznamy z jedné nebo obou tabulek, i když neexistuje shoda.

Levé vnější spojení (LEFT OUTER JOIN) Vrací všechny řádky z levé tabulky a odpovídající řádky z pravé tabulky. Pokud neexistuje shoda, obsahují sloupce z pravé tabulky hodnotu NULL.

```sql
SELECT z.jmeno, z.prijmeni, o.nazev AS oddeleni
FROM zamestnanci z
LEFT JOIN oddeleni o ON z.id_oddeleni = o.id_oddeleni;
```

Graficky:

```
Tabulka A    |    Tabulka B
-----------------------------
    A        |        B

  Výsledek (všechny z A)
```

Pravé vnější spojení (RIGHT OUTER JOIN) Vrací všechny řádky z pravé tabulky a odpovídající řádky z levé tabulky. Pokud neexistuje shoda, obsahují sloupce z levé tabulky hodnotu NULL.

```sql
SELECT z.jmeno, z.prijmeni, o.nazev AS oddeleni
FROM zamestnanci z
RIGHT JOIN oddeleni o ON z.id_oddeleni = o.id_oddeleni;
```

Graficky:

```
Tabulka A    |    Tabulka B
-----------------------------
    A        |        B

             Výsledek (všechny z B)
```

Úplné vnější spojení (FULL OUTER JOIN) Vrací všechny řádky z obou tabulek. Pokud neexistuje shoda, obsahují odpovídající sloupce hodnotu NULL.

```sql
SELECT z.jmeno, z.prijmeni, o.nazev AS oddeleni
FROM zamestnanci z
FULL OUTER JOIN oddeleni o ON z.id_oddeleni = o.id_oddeleni;
```

Graficky:

```
Tabulka A    |    Tabulka B
-----------------------------
    A        |        B

  Výsledek (všechny z A i B)
```

Cross Join (kartézský součin)

Vrací všechny možné kombinace řádků z obou tabulek.

```sql
SELECT z.jmeno, z.prijmeni, o.nazev AS oddeleni
FROM zamestnanci z
CROSS JOIN oddeleni o;
```

Self Join (spojení tabulky se sebou samotnou)

Používá se, když chceme spojit tabulku samu se sebou.

```sql
-- Příklad: Najít všechny páry zaměstnanců, kteří pracují ve stejném oddělení
SELECT z1.jmeno AS zamestnanec1, z2.jmeno AS zamestnanec2, o.nazev AS oddeleni
FROM zamestnanci z1
JOIN zamestnanci z2 ON z1.id_oddeleni = z2.id_oddeleni AND z1.id_zamestnance < z2.id_zamestnance
JOIN oddeleni o ON z1.id_oddeleni = o.id_oddeleni;
```

Referenční integrita

Referenční integrita je soubor pravidel, která zajišťují, že vztahy mezi tabulkami zůstávají konzistentní. Hlavní myšlenkou je, že pokud v jedné tabulce existuje odkaz na řádek v jiné tabulce, musí tento řádek skutečně existovat.

Omezení referenční integrity

Při definování cizího klíče můžeme specifikovat, co se má stát, když se změní nebo odstraní primární klíč, na který odkazuje.

ON DELETE Definuje akci, která se provede, když je odstraněn řádek, na který odkazuje cizí klíč:

1. **CASCADE** - automaticky odstraní odpovídající řádky v tabulce s cizím klíčem
2. **SET NULL** - nastaví hodnotu cizího klíče na NULL
3. **SET DEFAULT** - nastaví hodnotu cizího klíče na výchozí hodnotu
4. **RESTRICT** - zabrání odstranění řádku, pokud na něj odkazuje cizí klíč
5. **NO ACTION** - podobné jako RESTRICT, ale kontrola se provádí po všech ostatních omezeních

ON UPDATE Definuje akci, která se provede, když je aktualizován primární klíč, na který odkazuje cizí klíč. Akce jsou stejné jako u ON DELETE.

Příklad definice referenční integrity:

```sql
CREATE TABLE objednavky (
    id_objednavky INT PRIMARY KEY,
    id_zakaznika INT,
    datum DATE,
    celkova_cena DECIMAL(10, 2),
    FOREIGN KEY (id_zakaznika)
    REFERENCES zakaznici(id_zakaznika)
    ON DELETE RESTRICT
    ON UPDATE CASCADE
);
```

Důsledky porušení referenční integrity

Porušení referenční integrity může vést k: - Osiřelým záznamům (orphaned records) - Nekonzistenci dat - Chybám v aplikaci - Ztrátě důvěryhodnosti dat

Podmínky (check constraint)

Check constraint je omezení, které definuje, jaké hodnoty jsou přijatelné pro daný sloupec nebo skupinu sloupců. Slouží k zajištění integrity dat na úrovni tabulky.

Vytvoření check constraint

```sql
-- Při vytváření tabulky
CREATE TABLE zamestnanci (
    id_zamestnance INT PRIMARY KEY,
    jmeno VARCHAR(50) NOT NULL,
    prijmeni VARCHAR(50) NOT NULL,
    vek INT CHECK (vek >= 18),
    plat DECIMAL(10, 2) CHECK (plat > 0)
);

-- Přidání k existující tabulce
ALTER TABLE zamestnanci
ADD CONSTRAINT check_vek CHECK (vek >= 18);
```

Příklady check constraint

```sql
-- Omezení pro jednotlivý sloupec
CHECK (cena > 0)
```

```sql
-- Složitější podmínka
CHECK (datum_nastupu < datum_ukonceni OR datum_ukonceni IS NULL)

-- Kombinace sloupců
CHECK (datum_nastupu <= CURRENT_DATE AND plat BETWEEN 15000 AND 150000)

-- Kontrola formátu (například PSČ)
CHECK (psc SIMILAR TO '[0-9]{5}')
```

Výhody check constraint

- Zajišťuje integritu dat
- Umisťuje validační logiku přímo do databáze
- Centralizuje pravidla pro všechny aplikace přistupující k databázi
- Zlepšuje výkon (místo provádění kontrol v aplikaci)

Transakce

Transakce je logická jednotka práce, která se buď provede celá, nebo vůbec. Zajišťuje konzistenci dat i v případě chyb, souběžných přístupů nebo selhání systému.

Vlastnosti transakcí (ACID)

1. **Atomicita (Atomicity)** - transakce se buď provede celá, nebo vůbec
2. **Konzistence (Consistency)** - transakce převádí databázi z jednoho konzistentního stavu do jiného
3. **Izolace (Isolation)** - transakce jsou vzájemně izolované a neovlivňují se
4. **Trvalost (Durability)** - po potvrzení (commit) jsou změny trvalé

Základní příkazy pro práci s transakcemi

```sql
-- Začátek transakce
BEGIN TRANSACTION;

-- Potvrzení transakce
COMMIT;

-- Zrušení transakce
ROLLBACK;

-- Vytvoření bodu pro částečný rollback
SAVEPOINT nazev_bodu;

-- Rollback k bodu
ROLLBACK TO nazev_bodu;
```

Příklad použití transakce

```sql
-- Převod peněz z jednoho účtu na druhý
BEGIN TRANSACTION;

UPDATE ucty SET zustatek = zustatek - 1000 WHERE id_uctu = 1;
UPDATE ucty SET zustatek = zustatek + 1000 WHERE id_uctu = 2;

-- Kontrola, zda zůstatek neklesl pod 0
IF EXISTS (SELECT 1 FROM ucty WHERE zustatek < 0) THEN
    ROLLBACK;
ELSE
```

```
    COMMIT;
END IF;
```

Úrovně izolace transakcí

Úroveň izolace definuje, jak transakce "vidí" změny provedené jinými transakcemi.

1. **READ UNCOMMITTED** - transakce vidí i nepotvrzené změny jiných transakcí (nejnižší izolace)
2. **READ COMMITTED** - transakce vidí pouze potvrzené změny jiných transakcí
3. **REPEATABLE READ** - zajišťuje, že opakované čtení stejných dat vrátí stejné výsledky
4. **SERIALIZABLE** - nejvyšší úroveň izolace, transakce probíhají, jako by byly provedeny jedna po druhé

```
-- Nastavení úrovně izolace
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

Problémy při souběžném přístupu

1. **Dirty read** - čtení nepotvrzených dat
2. **Non-repeatable read** - opakované čtení vrací různé výsledky
3. **Phantom read** - opakovaný dotaz vrací nové řádky
4. **Lost update** - ztráta aktualizace při současném zápisu

ER diagram

ER diagram (Entity-Relationship Diagram) je grafická reprezentace entit a jejich vztahů v databázi. Pomáhá vizualizovat a navrhnout databázovou strukturu.

Základní prvky ER diagramu

Entity Entity reprezentují objekty nebo koncepty, které chceme ukládat v databázi. V diagramu jsou zobrazeny jako obdélníky.

Příklady: Zákazník, Produkt, Objednávka, Zaměstnanec

Atributy Atributy jsou vlastnosti entit. V diagramu jsou zobrazeny jako elipsy nebo jako seznam uvnitř entity.

Příklady pro entitu Zákazník: id, jméno, příjmení, email, telefon

Vztahy Vztahy popisují, jak spolu entity souvisejí. V diagramu jsou zobrazeny jako kosočtverce nebo linky mezi entitami.

Příklady: Zákazník "vytváří" Objednávku, Produkt "patří do" Kategorie

Kardinalita vztahů Kardinalita určuje, kolik instancí jedné entity může být spojeno s jednou instancí druhé entity.

1. **Jedna k jedné (1:1)** - jedna instance entity A je spojena s právě jednou instancí entity B Příklad: Zaměstnanec "má přiděleno" jedno Parkovací místo
2. **Jedna k mnoha (1:N)** - jedna instance entity A je spojena s více instancemi entity B Příklad: Oddělení "zaměstnává" více Zaměstnanců
3. **Mnoho k mnoha (M:N)** - více instancí entity A je spojeno s více instancemi entity B Příklad: Student "navštěvuje" více Kurzů a Kurz "je navštěvován" více Studenty

Příklad ER diagramu

Představme si jednoduchý e-shop s entitami Zákazník, Objednávka, Produkt a Kategorie:

```
[Zákazník] 1 ---- N [Objednávka]
                      |
                      | N
                      |
[Kategorie] 1 ---- N [Produkt] M ---- N [Objednávka]
```

Převod ER diagramu na schéma databáze

ER diagram je abstraktním návrhem, který je třeba převést na konkrétní databázové schéma.

1. Entity se stávají tabulkami

```sql
CREATE TABLE zakaznici (
    id_zakaznika INT PRIMARY KEY,
    jmeno VARCHAR(50),
    prijmeni VARCHAR(50),
    email VARCHAR(100)
);

CREATE TABLE produkty (
    id_produktu INT PRIMARY KEY,
    nazev VARCHAR(100),
    cena DECIMAL(10, 2),
    id_kategorie INT
);

CREATE TABLE kategorie (
    id_kategorie INT PRIMARY KEY,
    nazev VARCHAR(50)
);

CREATE TABLE objednavky (
    id_objednavky INT PRIMARY KEY,
    datum DATE,
    id_zakaznika INT
);
```

2. Vztahy se převádějí podle kardinality

- **1:1** - cizí klíč může být v kterékoli z tabulek
- **1:N** - cizí klíč je v tabulce na straně "N"
- **M:N** - vytvoří se vazební tabulka s cizími klíči na obě entity

```sql
-- Vztah 1:N mezi Zákazník a Objednávka
ALTER TABLE objednavky
ADD FOREIGN KEY (id_zakaznika) REFERENCES zakaznici(id_zakaznika);

-- Vztah 1:N mezi Kategorie a Produkt
ALTER TABLE produkty
ADD FOREIGN KEY (id_kategorie) REFERENCES kategorie(id_kategorie);

-- Vztah M:N mezi Objednávka a Produkt (vazební tabulka)
CREATE TABLE objednavka_polozky (
    id_objednavky INT,
    id_produktu INT,
    mnozstvi INT,
```

```
    cena DECIMAL(10, 2),
    PRIMARY KEY (id_objednavky, id_produktu),
    FOREIGN KEY (id_objednavky) REFERENCES objednavky(id_objednavky),
    FOREIGN KEY (id_produktu) REFERENCES produkty(id_produktu)
);
```

Efektivita a příklady

Efektivní návrh databáze

1. Normalizace Normalizace je proces organizace dat v databázi tak, aby se minimalizovala redundance a závislosti. Existuje několik normálních forem (1NF, 2NF, 3NF, BCNF, 4NF, 5NF), přičemž běžně se používá 3NF.

Výhody normalizace: - Minimalizace redundance dat - Lepší integrita dat - Menší velikost databáze - Jednodušší aktualizace

Nevýhody normalizace: - Složitější dotazy (více spojení) - Potenciálně horší výkon pro čtení

2. Indexy Správné používání indexů může výrazně zlepšit výkon dotazů.

Zásady pro indexy: - Indexujte sloupce používané v podmínkách WHERE, JOIN a ORDER BY - Neindexujte příliš (každý index zpomaluje operace INSERT, UPDATE, DELETE) - Zvažte složené indexy pro často používané kombinace sloupců - Pravidelně analyzujte využití indexů

3. Efektivní spojení

- Používejte správný typ spojení pro daný účel
- Minimalizujte počet spojení v dotazech
- Zvažte denormalizaci pro analytické dotazy

4. Správně nastavená referenční integrita

- Zvolte vhodné akce pro ON DELETE a ON UPDATE
- Používejte cizí klíče pro zajištění integrity dat
- Zvažte dopady na výkon při kaskádových operacích

Příklady efektivního použití relací a integrity

Příklad 1: Systém objednávek s referenční integritou

```
-- Vytvoření tabulek
CREATE TABLE zakaznici (
    id_zakaznika SERIAL PRIMARY KEY,
    jmeno VARCHAR(50) NOT NULL,
    prijmeni VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    telefon VARCHAR(20),
    CHECK (LENGTH(telefon) >= 9)
);

CREATE TABLE objednavky (
    id_objednavky SERIAL PRIMARY KEY,
    id_zakaznika INT NOT NULL,
    datum_objednavky DATE DEFAULT CURRENT_DATE,
    stav VARCHAR(20) DEFAULT 'nova',
    FOREIGN KEY (id_zakaznika)
    REFERENCES zakaznici(id_zakaznika)
    ON DELETE RESTRICT,
    CHECK (stav IN ('nova', 'zpracovana', 'odeslana', 'dorucena', 'stornovana'))
```

```
);

CREATE TABLE produkty (
    id_produktu SERIAL PRIMARY KEY,
    nazev VARCHAR(100) NOT NULL,
    cena DECIMAL(10, 2) NOT NULL,
    skladem INT NOT NULL DEFAULT 0,
    CHECK (cena > 0),
    CHECK (skladem >= 0)
);

CREATE TABLE objednavka_polozky (
    id_objednavky INT,
    id_produktu INT,
    mnozstvi INT NOT NULL,
    cena_za_kus DECIMAL(10, 2) NOT NULL,
    PRIMARY KEY (id_objednavky, id_produktu),
    FOREIGN KEY (id_objednavky)
    REFERENCES objednavky(id_objednavky)
    ON DELETE CASCADE,
    FOREIGN KEY (id_produktu)
    REFERENCES produkty(id_produktu)
    ON DELETE RESTRICT,
    CHECK (mnozstvi > 0),
    CHECK (cena_za_kus > 0)
);

-- Vytvoření indexů
CREATE INDEX idx_objednavky_zakaznik ON objednavky(id_zakaznika);
CREATE INDEX idx_objednavky_datum ON objednavky(datum_objednavky);
CREATE INDEX idx_produkty_nazev ON produkty(nazev);
```

Příklad 2: Transakce pro vytvoření objednávky

```
-- Transakce pro vytvoření objednávky s kontrolou dostupnosti produktů
BEGIN TRANSACTION;

-- Vytvoření objednávky
INSERT INTO objednavky (id_zakaznika, datum_objednavky, stav)
VALUES (1, CURRENT_DATE, 'nova')
RETURNING id_objednavky INTO @id_objednavky;

-- Přidání položek objednávky
INSERT INTO objednavka_polozky (id_objednavky, id_produktu, mnozstvi, cena_za_kus)
VALUES (@id_objednavky, 1, 2, (SELECT cena FROM produkty WHERE id_produktu = 1));

-- Kontrola dostupnosti produktů
IF EXISTS (
    SELECT 1 FROM objednavka_polozky op
    JOIN produkty p ON op.id_produktu = p.id_produktu
    WHERE op.id_objednavky = @id_objednavky
    AND op.mnozstvi > p.skladem
) THEN
    ROLLBACK;
    RAISE EXCEPTION 'Nedostatek zboží na skladě';
ELSE
    -- Aktualizace skladu
    UPDATE produkty p
```

```
    SET skladem = p.skladem - op.mnozstvi
    FROM objednavka_polozky op
    WHERE p.id_produktu = op.id_produktu
    AND op.id_objednavky = @id_objednavky;

    COMMIT;
END IF;
```

Příklad 3: Složitější dotazy se spojením

```
-- Souhrn objednávek zákazníků s použitím různých typů spojení
SELECT
    z.jmeno || ' ' || z.prijmeni AS zakaznik,
    COUNT(o.id_objednavky) AS pocet_objednavek,
    COALESCE(SUM(op.mnozstvi * op.cena_za_kus), 0) AS celkova_utrata
FROM zakaznici z
LEFT JOIN objednavky o ON z.id_zakaznika = o.id_zakaznika
LEFT JOIN objednavka_polozky op ON o.id_objednavky = op.id_objednavky
GROUP BY z.id_zakaznika, z.jmeno, z.prijmeni
ORDER BY celkova_utrata DESC;

-- Přehled nejprodávanějších produktů podle kategorií
SELECT
    k.nazev AS kategorie,
    p.nazev AS produkt,
    SUM(op.mnozstvi) AS prodano_celkem,
    SUM(op.mnozstvi * op.cena_za_kus) AS trzby
FROM kategorie k
JOIN produkty p ON k.id_kategorie = p.id_kategorie
LEFT JOIN objednavka_polozky op ON p.id_produktu = op.id_produktu
GROUP BY k.id_kategorie, k.nazev, p.id_produktu, p.nazev
HAVING SUM(op.mnozstvi) > 0
ORDER BY kategorie, prodano_celkem DESC;
```

Příklad 4: Příklad se Self Join

```
-- Organizační struktura zaměstnanců pomocí self join
CREATE TABLE zamestnanci (
    id_zamestnance SERIAL PRIMARY KEY,
    jmeno VARCHAR(50) NOT NULL,
    prijmeni VARCHAR(50) NOT NULL,
    pozice VARCHAR(50),
    id_nadrizeneho INT,
    FOREIGN KEY (id_nadrizeneho) REFERENCES zamestnanci(id_zamestnance)
);

-- Dotaz pro zobrazení zaměstnanců a jejich nadřízených
SELECT
    z.jmeno || ' ' || z.prijmeni AS zamestnanec,
    z.pozice,
    n.jmeno || ' ' || n.prijmeni AS nadrizeny,
    n.pozice AS pozice_nadrizeneho
FROM zamestnanci z
LEFT JOIN zamestnanci n ON z.id_nadrizeneho = n.id_zamestnance
ORDER BY n.id_zamestnance NULLS FIRST, z.id_zamestnance;
```

Shrnutí

- **Relace** jsou tabulky, které reprezentují entity nebo vztahy v databázi.
- **Primární klíč** jednoznačně identifikuje každý záznam v tabulce.
- **Cizí klíč** vytváří vztah mezi tabulkami a zajišťuje referenční integritu.
- **Unikátní klíč** zajišťuje, že hodnoty v daném sloupci jsou unikátní.
- **Vnitřní spojení** vrací pouze záznamy, kde existuje shoda mezi tabulkami.
- **Vnější spojení** vrací všechny záznamy z jedné nebo obou tabulek.
- **Referenční integrita** zajišťuje konzistenci vztahů mezi tabulkami.
- **Check constraint** definuje, jaké hodnoty jsou přijatelné pro daný sloupec.
- **Transakce** jsou logické jednotky práce, které se buď provedou celé, nebo vůbec.
- **ER diagram** je grafická reprezentace entit a jejich vztahů v databázi.