13. Databáze - selekce, projekce a agregace

Obsah

Základní pojmy

Selekce

Selekce je operace, která slouží k **výběru řádků (záznamů)** z databázové tabulky podle určitých podmínek.

Princip: - Vybírá podmnožinu řádků, které splňují zadanou podmínku - Používá klauzuli `WHERE` v SQL - Podmínky mohou zahrnovat porovnávání, logické operátory, vnořené dotazy atd.

Příklad:

```sql
-- Vybere všechny zaměstnance s platem vyšším než 50000
SELECT * FROM zamestnanci WHERE plat > 50000;

-- Vybere zaměstnance z IT oddělení, kteří jsou starší než 30 let
SELECT * FROM zamestnanci WHERE oddeleni = 'IT' AND vek > 30;
```

Projekce

Projekce je operace, která slouží k **výběru sloupců (atributů)** z databázové tabulky.

Princip: - Omezuje počet sloupců ve výsledku dotazu - Definuje se seznamem sloupců za klíčovým slovem `SELECT` - Umožňuje transformaci a přejmenování sloupců

Příklad:

```sql
-- Vybere pouze jméno a příjmení z tabulky zaměstnanců
SELECT jmeno, prijmeni FROM zamestnanci;

-- Vybere jméno, příjmení a vypočtený roční plat s přejmenováním
SELECT jmeno, prijmeni, plat * 12 AS rocni_plat FROM zamestnanci;
```

Důležité: Selekce a projekce se často používají společně v jednom dotazu.

Agregace

Agregace je operace, která slouží k **výpočtu souhrnných hodnot** z množiny záznamů.

Princip: - Produkuje výsledek z více řádků - Používá agregační funkce v SQL - Často se kombinuje se seskupováním (GROUP BY)

Příklad:

```sql
-- Zjistí průměrný plat v celé firmě
SELECT AVG(plat) AS prumerny_plat FROM zamestnanci;

-- Zjistí maximální plat v každém oddělení
SELECT oddeleni, MAX(plat) AS nejvyssi_plat
FROM zamestnanci
GROUP BY oddeleni;
```

Řazení (ORDER BY)

Řazení umožňuje uspořádat výsledky dotazu podle hodnot v jednom nebo více sloupcích.

Syntaxe:

```
SELECT sloupce FROM tabulka [WHERE podmínka] ORDER BY sloupec1 [ASC|DESC], sloupec2 [ASC|DESC], ...;
```

Vlastnosti: - `ASC` (ascending) - vzestupně (výchozí, pokud není specifikováno) - `DESC` (descending) - sestupně - Můžeme řadit podle více sloupců (řazení se aplikuje zleva doprava) - Můžeme řadit podle výrazů nebo výsledků funkcí

Příklady:

```
-- Řazení podle jednoho sloupce vzestupně
SELECT jmeno, prijmeni, plat FROM zamestnanci ORDER BY plat;

-- Řazení podle jednoho sloupce sestupně
SELECT jmeno, prijmeni, plat FROM zamestnanci ORDER BY plat DESC;

-- Řazení podle více sloupců
SELECT jmeno, prijmeni, oddeleni, plat FROM zamestnanci
ORDER BY oddeleni ASC, plat DESC;

-- Řazení podle pořadí sloupců ve výsledku (číselný index)
SELECT jmeno, prijmeni, plat FROM zamestnanci ORDER BY 3 DESC;

-- Řazení podle výpočtu
SELECT jmeno, prijmeni, plat FROM zamestnanci ORDER BY plat * 0.9;
```

Indexy a jejich využití

Index je datová struktura, která zrychluje vyhledávání a přístup k datům v databázi.

Princip fungování indexů

- Index je podobný rejstříku v knize
- Umožňuje rychlé vyhledávání hodnot bez nutnosti procházet celou tabulku
- Typicky implementován jako vyvážený strom (B-tree, B+ tree) nebo hash tabulka
- Databázový systém automaticky udržuje indexy aktuální

Typy indexů

1. Primární klíč (PRIMARY KEY)

- Vždy unikátní
- Automaticky indexován
- Používá se k jednoznačné identifikaci řádků

2. Unikátní index (UNIQUE)

- Zajišťuje unikátní hodnoty v sloupci nebo kombinaci sloupců
- Zabraňuje duplicitám

3. Běžný index (INDEX)

- Urychluje vyhledávání, ale dovoluje duplicitní hodnoty
- Vhodný pro sloupce často používané v podmínkách WHERE

4. Složený index (COMPOSITE INDEX)

- Index vytvořený na více sloupcích
- Efektivní pro dotazy zahrnující všechny nebo první sloupce indexu

Vytvoření a odstranění indexu

```sql
-- Vytvoření běžného indexu
CREATE INDEX idx_prijmeni ON zamestnanci(prijmeni);

-- Vytvoření unikátního indexu
CREATE UNIQUE INDEX idx_email ON zamestnanci(email);

-- Vytvoření složeného indexu
CREATE INDEX idx_jmeno_prijmeni ON zamestnanci(jmeno, prijmeni);

-- Odstranění indexu
DROP INDEX idx_prijmeni ON zamestnanci;
```

Výhody a nevýhody indexů

Výhody: - Dramaticky zrychlují vyhledávání a řazení - Optimalizují složitost z O(n) na O(log n) nebo O(1) - Zlepšují výkon JOIN operací a podmínek WHERE - Pomáhají při vynucení unikátnosti (UNIQUE)

Nevýhody: - Zabírají paměťový prostor - Zpomalují operace INSERT, UPDATE a DELETE (nutnost aktualizace indexů) - Příliš mnoho indexů může způsobit zpomalení

Seskupování (GROUP BY)

Seskupování umožňuje agregovat data podle hodnot v jednom nebo více sloupcích.

Syntaxe:

```sql
SELECT sloupec1, sloupec2, ..., agregační_funkce(sloupec)
FROM tabulka
[WHERE podmínka]
GROUP BY sloupec1, sloupec2, ...
[HAVING podmínka_pro_skupiny];
```

Princip: - Rozdělí řádky do skupin podle hodnot ve specifikovaných sloupcích - Umožňuje aplikovat agregační funkce na každou skupinu - `HAVING` umožňuje filtrovat výsledky na základě agregovaných hodnot

Rozdíl mezi WHERE a HAVING: - `WHERE` filtruje řádky **před** seskupením - `HAVING` filtruje skupiny **po** seskupení, na základě výsledků agregací

Příklady:

```sql
-- Zjištění počtu zaměstnanců v každém oddělení
SELECT oddeleni, COUNT(*) AS pocet_zamestnancu
FROM zamestnanci
GROUP BY oddeleni;

-- Průměrné platy v odděleních, která mají více než 5 zaměstnanců
SELECT oddeleni, AVG(plat) AS prumerny_plat, COUNT(*) AS pocet
FROM zamestnanci
GROUP BY oddeleni
HAVING COUNT(*) > 5;

-- Seskupení podle více sloupců
SELECT oddeleni, pozice, AVG(plat) AS prumerny_plat
```

```
FROM zamestnanci
GROUP BY oddeleni, pozice;
```

Základní agregační funkce

Agregační funkce provádějí výpočty na množině hodnot a vrací jednu hodnotu.

1. COUNT()

Počítá počet řádků nebo neprázdných hodnot.

```
-- Počet všech řádků v tabulce
SELECT COUNT(*) FROM zamestnanci;

-- Počet neprázdných hodnot ve sloupci
SELECT COUNT(email) FROM zamestnanci;

-- Počet unikátních hodnot
SELECT COUNT(DISTINCT oddeleni) FROM zamestnanci;
```

2. SUM()

Počítá součet hodnot.

```
-- Celkový součet platů
SELECT SUM(plat) FROM zamestnanci;

-- Součet platů podle oddělení
SELECT oddeleni, SUM(plat) AS celkove_platy
FROM zamestnanci
GROUP BY oddeleni;
```

3. AVG()

Počítá aritmetický průměr hodnot.

```
-- Průměrný plat
SELECT AVG(plat) FROM zamestnanci;

-- Průměrný plat podle oddělení
SELECT oddeleni, AVG(plat) AS prumerny_plat
FROM zamestnanci
GROUP BY oddeleni;
```

4. MIN() a MAX()

Vrací minimální a maximální hodnotu.

```
-- Minimální a maximální plat
SELECT MIN(plat) AS nejnizsi_plat, MAX(plat) AS nejvyssi_plat
FROM zamestnanci;

-- Minimální a maximální plat podle oddělení
SELECT oddeleni, MIN(plat) AS nejnizsi_plat, MAX(plat) AS nejvyssi_plat
FROM zamestnanci
GROUP BY oddeleni;
```

5. Další užitečné agregační funkce (specifické pro různé DBMS)

PostgreSQL, Oracle, SQL Server, MySQL

```sql
-- Směrodatná odchylka
SELECT STDDEV(plat) FROM zamestnanci;

-- Rozptyl
SELECT VARIANCE(plat) FROM zamestnanci;
```

PostgreSQL, SQL Server

```sql
-- Textová agregace (spojování řetězců)
SELECT oddeleni, STRING_AGG(jmeno, ', ') AS zamestnanci
FROM zamestnanci
GROUP BY oddeleni;
```

MySQL

```sql
-- Textová agregace (spojování řetězců)
SELECT oddeleni, GROUP_CONCAT(jmeno) AS zamestnanci
FROM zamestnanci
GROUP BY oddeleni;
```

Vhodné použití

Selekce

- **Kdy použít:** Když potřebujeme omezit výsledky dotazu na podmnožinu řádků
- **Příklady použití:**
  - Filtrování dat podle určitých kritérií (např. datum, kategorie, stav)
  - Vyhledávání záznamů odpovídajících specifickým podmínkám
  - Odstranění nerelevantních dat z výsledků

Projekce

- **Kdy použít:** Když potřebujeme pouze specifické sloupce místo všech dat
- **Příklady použití:**
  - Omezení objemu přenášených dat mezi databází a aplikací
  - Zobrazení pouze relevantních informací uživateli
  - Transformace dat (např. výpočty, formátování)

Agregace a seskupování

- **Kdy použít:** Když potřebujeme souhrnné informace místo jednotlivých řádků
- **Příklady použití:**
  - Výpočet statistik (počty, průměry, součty, minima, maxima)
  - Vytváření přehledů a reportů
  - Analýza dat podle kategorií nebo časových období
  - Business Intelligence a datové sklady

Indexy

- **Kdy použít:** Pro optimalizaci často používaných dotazů na velkých tabulkách
- **Příklady použití:**
  - Sloupce používané v podmínkách WHERE
  - Sloupce používané pro JOIN operace
  - Sloupce používané pro řazení nebo seskupování
  - Sloupce s vysokou selektivitou (mnoho unikátních hodnot)

Efektivita

Optimalizace selekce

- Použití indexů na sloupcích v podmínkách WHERE
- Používání vhodných datových typů
- Specifikace konkrétních sloupců místo `SELECT *`
- Vhodné formulování podmínek (indexovatelné podmínky)

Optimalizace agregací a seskupování

- Použití indexů na sloupcích v GROUP BY
- Filtrování dat před seskupením (WHERE místo HAVING, kde je to možné)
- Omezení počtu skupin (příliš mnoho skupin může způsobit výkonnostní problémy)

Optimalizace řazení

- Vytvoření indexů na sloupcích používaných v ORDER BY
- Omezení množství dat před řazením
- Vyhýbání se řazení podle výpočtů nebo funkcí (pokud možno)

Index best practices

- Indexujte pouze sloupce, které skutečně potřebujete
- Pravidelně analyzujte využití indexů
- Zvažte složené indexy pro dotazy s více podmínkami
- Při velkém množství operací INSERT/UPDATE/DELETE zvažte dočasné vypnutí indexů

Obecné tipy pro výkon

- Omezení množství dat před zpracováním (selekce před projekcí)
- Používání pohledů (VIEW) pro často používané komplexní dotazy
- Denormalizace schématu pro analytické dotazy v některých případech
- Využití funkce EXPLAIN pro analýzu plánu vykonávání dotazu

Příklady

Selekce a projekce

Příklad 1: Získání informací o zaměstnancích z IT oddělení s platem nad 50000

```sql
-- Selekce s projekcí
SELECT jmeno, prijmeni, plat
FROM zamestnanci
WHERE oddeleni = 'IT' AND plat > 50000;
```

Příklad 2: Výběr produktů s cenou mezi 100 a 500, které jsou skladem

```sql
SELECT nazev, cena, pocet_skladem
FROM produkty
WHERE cena BETWEEN 100 AND 500
  AND pocet_skladem > 0
ORDER BY cena;
```

Agregace a seskupování

Příklad 3: Analýza objednávek podle měsíců a let

```sql
SELECT
    YEAR(datum_objednavky) AS rok,
    MONTH(datum_objednavky) AS mesic,
```

```
    COUNT(*) AS pocet_objednavek,
    SUM(cena_celkem) AS trzby,
    AVG(cena_celkem) AS prumerna_objednavka
FROM objednavky
GROUP BY YEAR(datum_objednavky), MONTH(datum_objednavky)
ORDER BY rok, mesic;
```

Příklad 4: Zjištění nejprodávanějších produktů podle kategorií

```
SELECT
    k.nazev AS kategorie,
    p.nazev AS produkt,
    SUM(op.mnozstvi) AS celkem_prodano
FROM produkty p
JOIN objednavka_polozky op ON p.id = op.produkt_id
JOIN kategorie k ON p.kategorie_id = k.id
GROUP BY k.nazev, p.nazev
HAVING SUM(op.mnozstvi) > 10
ORDER BY kategorie, celkem_prodano DESC;
```

Složitější příklady s indexy

Příklad 5: Vytvoření indexů pro optimalizaci často používaných dotazů

```
-- Vytvoření indexu pro vyhledávání zaměstnanců podle oddělení a platu
CREATE INDEX idx_zamestnanci_oddeleni_plat ON zamestnanci(oddeleni, plat);

-- Dotaz, který nyní bude rychlejší
SELECT jmeno, prijmeni, plat
FROM zamestnanci
WHERE oddeleni = 'IT' AND plat > 50000;
```

Příklad 6: Porovnání plánů vykonávání s indexem a bez něj

```
-- Analýza plánu dotazu bez indexu
EXPLAIN SELECT * FROM produkty WHERE kod = 'ABC123';

-- Vytvoření indexu
CREATE INDEX idx_produkty_kod ON produkty(kod);

-- Analýza plánu dotazu s indexem
EXPLAIN SELECT * FROM produkty WHERE kod = 'ABC123';
```

Subselect a složené dotazy

Příklad 7: Nalezení zaměstnanců s nadprůměrným platem v jejich oddělení

```
SELECT z.jmeno, z.prijmeni, z.oddeleni, z.plat
FROM zamestnanci z
WHERE z.plat > (
    SELECT AVG(plat)
    FROM zamestnanci
    WHERE oddeleni = z.oddeleni
)
ORDER BY z.oddeleni, z.plat DESC;
```

Příklad 8: Sumarizace prodejů za poslední měsíc s porovnáním s předchozím měsícem

```
SELECT
    p.kategorie_id,
    k.nazev AS kategorie,
    SUM(CASE WHEN o.datum_objednavky >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
```

```
            THEN op.mnozstvi * op.cena
            ELSE 0
        END) AS prodeje_tento_mesic,
    SUM(CASE WHEN o.datum_objednavky BETWEEN DATE_SUB(CURDATE(), INTERVAL 2 MONTH)
                                         AND DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
            THEN op.mnozstvi * op.cena
            ELSE 0
        END) AS prodeje_minuly_mesic
FROM produkty p
JOIN kategorie k ON p.kategorie_id = k.id
JOIN objednavka_polozky op ON p.id = op.produkt_id
JOIN objednavky o ON op.objednavka_id = o.id
WHERE o.datum_objednavky >= DATE_SUB(CURDATE(), INTERVAL 2 MONTH)
GROUP BY p.kategorie_id, k.nazev
ORDER BY prodeje_tento_mesic DESC;
```

Shrnutí

- **Selekce** (WHERE) umožňuje vybírat řádky podle podmínek
- **Projekce** (SELECT sloupce) umožňuje vybírat pouze potřebné sloupce
- **Agregace** (COUNT, SUM, AVG, MIN, MAX) umožňuje provádět výpočty nad množinou dat
- **Řazení** (ORDER BY) umožňuje uspořádat výsledky podle určitých kritérií
- **Seskupování** (GROUP BY) umožňuje agregovat data podle kategorií
- **Indexy** dramaticky zlepšují výkon dotazů za cenu dodatečného úložného prostoru
- Pro efektivní dotazy je klíčové správně kombinovat všechny tyto koncepty