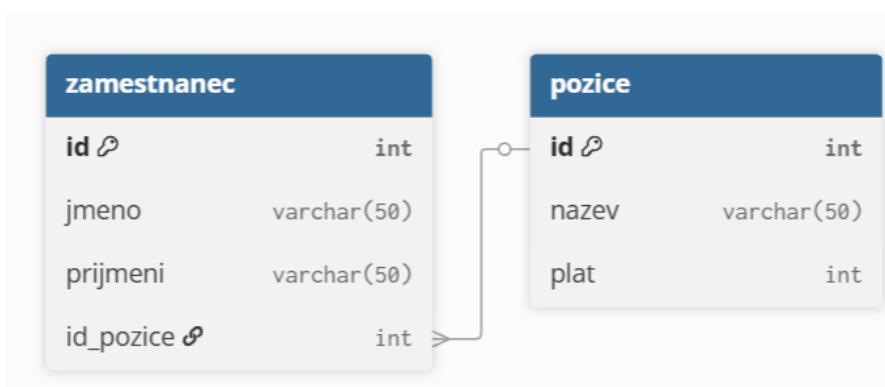


# Jazyk SQL, DDL, DML příkazy - vytvoření a úprava struktury databáze, vložení a úprava dat, příkaz SELECT včetně všech klauzulí, spojování tabulek, typy spojení, agregační funkce a jejich význam, VIEW (uložený SELECT) - vytvoření, použití a význam

Jazyk SQL, DDL, DML příkazy - vytvoření a úprava struktury databáze, vložení a úprava dat, příkaz SELECT včetně všech klauzulí, spojování tabulek, typy spojení, agregační funkce a jejich význam, VIEW (uložený SELECT) - vytvoření, použití a význam.....1

SQL - DDL, DML.....	2
SQL - Structured Query Language.....	2
DDL - Data Definition Language.....	2
DML - Data Manipulation Language.....	2
DDL - vytvoření a úprava struktury databáze.....	2
DML - vložení a úprava dat.....	3
DML - SELECT.....	3
Struktura selectu.....	3
Význam klauzulí.....	4
JOINs.....	4
Agregační funkce.....	4
GROUP BY.....	5
HAVING.....	5
Operátory.....	5
VIEW - pohled.....	7
MySQL.....	7



# SQL - DDL, DML

## SQL - Structured Query Language

- dotazovací jazyk
- interpretovaný jazyk, který je součástí db serveru
  - kontrola syntaxe, provádění příkazů

## DDL - Data Definition Language

- příkazy pro práci s objekty
  - tabulky, triggery, databáze, indexy
- **CREATE** - vytvoření objektu
- **ALTER** - změna struktury objektu
- **DROP** - smazání objektu

## DML - Data Manipulation Language

- pracuje s daty (hodnotami)
- **INSERT INTO** - vložení dat
- **UPDATE** - změna dat
- **DELETE FROM** - smazání dat
- **SELECT** - výpis dat

## DDL - vytvoření a úprava struktury databáze

### Vytvoření tabulek

```
CREATE TABLE zamestnanec (
    id      INT PRIMARY KEY auto_increment,
    jmeno   VARCHAR(50),
    prijmeni VARCHAR(50),
    id_pozice INT
) ;

CREATE TABLE pozice (
    id      INT PRIMARY KEY auto_increment,
    nazev  VARCHAR(50),
    plat   INT,
    CONSTRAINT fk_pozice FOREIGN KEY (id_pozice) REFERENCES pozice(id) ON
    DELETE SET NULL ON UPDATE CASCADE
) ;
```

### Úprava tabulek

```
ALTER TABLE zamestnanec
    ADD COLUMN email VARCHAR(100);

ALTER TABLE pozice
    modify COLUMN plat DECIMAL(10, 2);

ALTER TABLE zamestnanec
```

```
DROP COLUMN email;
```

## Mazání tabulek

```
DROP TABLE zamestnanec;
```

```
DROP TABLE pozice;
```

## DML - vložení a úprava dat

### Vložení dat

```
INSERT INTO pozice(nazev, plat)
VALUES ('Manazer', 50000), ('Obchodnik', 35000), ('Programator', 45000);
```

```
INSERT INTO zamestnanec (jmeno, prijmeni, id_pozice)
VALUES ('Jan', 'Novak', 1), ('Petr', 'Svoboda', 2), ('Eva', 'Horakova',
3), ('Lucie', 'Kralova', 3);
```

### Úprava dat

```
UPDATE pozice
SET plat = plat * 1.1
WHERE nazev = 'Programator';
```

```
UPDATE zamestnanec
SET id_pozice = 2
WHERE jmeno = 'Jan' AND prijmeni = 'Novak';
```

### Mazání dat

```
DELETE FROM zamestnanec
WHERE jmeno = 'Petr' AND prijmeni = 'Svoboda';
```

```
DELETE FROM pozice
WHERE nazev = 'Manazer';
```

## DML - SELECT

- SQL příkaz pro výběr (dotaz) dat z jedné nebo více tabulek.
- Vrací výsledky jako virtuální tabulku.

### Struktura selectu

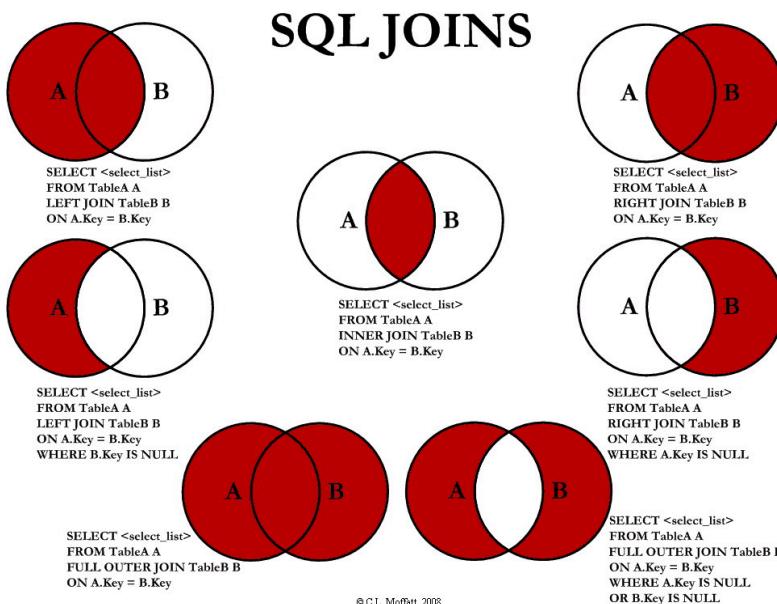
```
SELECT [sloupce | *] FROM tabulka
      [WHERE podminka]
      [GROUP BY sloupce]
      [HAVING podminka]
      [ORDER BY sloupce ASC|DESC]
      [LIMIT n]
```

## Význam klauzulí

- **SELECT** - které sloupce se zobrazí
- **FROM** - z jaké tabulky
- **WHERE** - filtrace řádků
- **GROUP BY** - seskupení podle sloupců
- **HAVING** - filtrace skupin
- **ORDER BY** - seřazení výsledků

## JOINs

- spojení tabulek
- **INNER JOIN** - spojuje jen řádky, které mají shodu v obou tabulkách
- **FULL JOIN** - všechny záznamy z obou tabulek
- **LEFT JOIN** - všechny z levé + odpovídající z pravé tabulky
- **RIGHT JOIN** - všechny z pravé + odpovídající z levé
- **CROSS JOIN** - kartézský součin, každý s každým



## Agregační funkce

- Funkce, které pracují pouze s jedním parametrem a vždy vrací pouze 1 výstupní hodnotu
- Používají se často s GROUP BY.
- Nelze je samostatně použít v klauzuli WHERE.
- **X where kniha.cena > AVG(kniha.cena)**
- ✓ **where kniha.cena > (select AVG(kniha.cena) from kniha)**
- Lze použít se selectem mezi atributy, v klauzuli having
- **COUNT(\*|atr)** - počet řádků
- **SUM(atr - číslo)** - součet hodnot
- **AVG(atr - číslo)** - průměr
- **MIN(atr - číslo, datum, text (abecedně) - A)** - minimální hodnota
- **MAX(atr - číslo, datum, text (abecedně) - Z)** - maximální hodnota
- př.:
  - počet zaměstnanců s platem > 40 000  
`SELECT Count(zam.id) AS pocet FROM zam`

```

        WHERE  plat > 40000;

○ min, max, avg plat zaměstnanců
SELECT Min(plat) AS min,
       Max(plat) AS max,
       Avg(plat) AS avg
FROM    zam;

```

## GROUP BY

- seskupený dotaz, pro každou skupinu vytvoří jednořádkový souhrn
- atributy uvedené za GROUP BY se provedou zleva doprava
- př.:
  - počet zam za každé oddělení a součet platů oddělení (seřazené podle oddělení vzestupně)
 

```

SELECT zam.oddeleni,
       Count(zam.id) AS pocet_zam,
       Sum(zam.plat) AS plat
FROM zam
GROUP BY zam.oddeleni
ORDER BY zam.oddeleni;

```

## HAVING

- omezení (podmínku) pro GROUP BY
- (bez GROUP BY nemá smysl použít)
- může obsahovat agregační funkce bez podselectu
- př.:
  - počet zam pro každé oddělení, ve kterém je více než 5 zam
 

```

SELECT zam.oddeleni,
       Count(zam.id)
FROM zam
GROUP BY zam.oddeleni
HAVING Count(zam.id) > 5;

```

## Operátory

- A. Matematické (=, <, >, <=, >=, !=(<>))
- lze použít pouze jde-li o konkrétní 1 hodnotu
  - př.: zam s platem > 40 000
 

```

SELECT * FROM zam
WHERE plat > 40000;

```
  - u podselectu pouze jsme-li si jisti, že vždy vrací jen jednu hodnotu
  - je-li v podselectu agr. funkce, vždy vrací jen 1 hodnotu
  - př.: zam s plat > prumerny plat
 

```

SELECT prijmeni
FROM zam
WHERE plat > (SELECT Avg(plat)
                FROM zam);

```

**! pokud si nejsme jisti, že vrací jen 1 hodnotu, musíme použít podmínu IN**
  - př.: výpis zam z Prahy a Brna

```
SELECT prijmeni  
FROM zam  
WHERE mesto IN ( brno, praha )
```

B. Logické (NOT, AND, OR)

C. Vyhledávání podle rozsahu BETWEEN (NOT BETWEEN)

- př.: vypište zam, jejichž plat je od 30 000 do 40 000

```
SELECT prijmeni  
FROM zam  
WHERE plat BETWEEN 30000 AND 40000;
```

- zahrnuje i krajní hodnoty

D. Vyhledávání podle množiny IN (NOT IN)

- př.: výpis všech zam s pozici manažer nebo obchodník

```
SELECT prijmeni  
FROM zam  
WHERE pozice IN( 'manazer', 'obchodnik' );
```

E. Vyhledávání podle vzoru LIKE (NOT LIKE)

- existují 2 zástupné symboly
  - % - libovolný počet libovolných znaků
  - \_ - jeden libovolný znak

- př.: všechna příjmení začínající na S

```
SELECT prijmeni  
FROM zam  
WHERE prijmeni LIKE 's%';
```

- př.: všechna příjmení začínající na S + právě 4 další znaky

```
SELECT prijmeni  
FROM zam  
WHERE prijmeni LIKE 's_____';
```

- př.: email adresa

```
SELECT prijmeni  
FROM zam  
WHERE email LIKE '%@%';
```

F. Podmínka s NULL (NOT NULL)

- př.: výpis zam, kteří ještě nemají pracovní pozici

```
SELECT prijmeni  
FROM zam  
WHERE pozice IS NULL;
```

G. ANY, ALL, EXISTS

## VIEW - pohled

- objekt uložený na serveru pod jménem
- je to virtuální tabulka
- vždy obsahuje pouze jeden select (většinou složitý = spojení více tabulek, agregační funkce, podmínky, group by,...)
  - můžou tam být podselecty, ale jen jeden hlavní select = jeden příkaz
- do view se nepoužívá order by (použije se až potom při použití view)
- spuštění view - pomocí select  
`SELECT * FROM <v_nazev>;`
- na server se ukládá předpis selectu a při spuštění se naplní virtuální tabulka aktuálními daty
- častá chyba:  
`SELECT pozice.nazev, zamestnanec.prijmeni FROM v_seznam_seznam;`  
pozice a zamestnanec už nejsou, je jen seznam - seznam.nazev, seznam.prijmeni

## MySQL

### Vytvoření

```
CREATE view v_seznam_zamestnancu
AS
SELECT z.jmeno, z.prijmeni, p.nazev AS pozice, p.plat
FROM zamestnanec z
JOIN pozice p ON z.id_pozice = p.id;
```

### Použití

```
SELECT * FROM v_seznam_zamestnancu
ORDER BY prijmeni, jmeno;
```

### Úprava

```
CREATE OR REPLACE view v_seznam_zamestnancu
AS
SELECT z.jmeno, z.prijmeni, p.nazev AS pozice, p.plat
FROM zamestnanec z
JOIN pozice p ON z.id_pozice = p.id;
```

### Mazání

```
DROP view IF EXISTS v_seznam_zamestnancu;
```