# SQL 92

Dotazovací jazyk pro relační SŘBD.

**`SQL = DDL + DML + TCL + DCL`**

- **DDL** = *Data Definition Language*
    - `CREATE/ALTER/DROP TABLE/VIEW/INDEX/...`
- **DML** = *Data Manipulation Language*
    - `SELECT, INSERT, UPDATE, DELETE`
- **TCL** = *Transaction Control Language*
    - `COMMIT, ROLLBACK, SAVEPOINT`
- **DCL** = *Data Control Language*
    - `GRANT, REVOKE`

# Výběr záznamů - příkaz SELECT

- základní stavební kámen SQL
- příkaz `SELECT` či jeho část je součástí mnoha příkazů SQL

Zjednodušená syntaktická konstrukce příkazu :

```
SELECT  [DISTINCT] {*|[sloupec_vyraz [[AS] alias] [,...]}
  FROM  tabulka [alias] [,...]
[WHERE  podmínka]
[GROUP  BY seznam_neagregovaných_sloupců_výrazů [HAVING podmínka]]
[ORDER  BY seznam_řazených_sloupců_výrazů]
;
```

## Dotazy nad jednou tabulkou - projekce

```
-- toto je komentář platný do konce řádky
-- vybrat celý obsah tabulky
SELECT * FROM cd;

-- hodnoty vybraných sloupců tabulky
SELECT pisen, d_min, d_sec FROM cd;

-- hodnoty vybraného sloupce – mohou se opakovat
SELECT interpret FROM cd;

-- hodnoty vybraného sloupce – každá jen jednou
SELECT DISTINCT interpret FROM cd;

-- výpočet nových hodnot ze sloupců tabulky
SELECT pisen, d_min * 60 + d_sec FROM cd;

-- to samé, přidělení aliasu vypočtenému výrazu
SELECT pisen, d_min * 60 + d_sec AS stopaz FROM cd;
```

## Dotazy nad jednou tabulkou - selekce (a projekce)

```
-- vybrat písně hrající déle než 5 minut
SELECT pisen, d_min, d_sec FROM cd WHERE d_min >= 5;
```

V podmínce **WHERE** je možné použít následující relační operátory a logické spojky:

### Relační operátory

- rovnost: `=`
- menší než: `<`
- větší než: `>`
- menší rovno: `<=`
- většíší rovno: `>=`
- nerovno: `<>`, případně i `!=`

### Logické spojky

- a zároveň: `AND`
- nebo: `OR`
- negace: `NOT`

```
-- vybrat písně, hrající 5 až 7 minut
SELECT pisen, d_min, d_sec
  FROM cd
 WHERE d_min >= 5 AND d_min <= 7;

-- to samé, použit operátor BETWEEN
SELECT pisen, d_min, d_sec
  FROM cd
 WHERE d_min BETWEEN 5 AND 7;

-- negace podmínky s operátorem BETWEEN je legrace
SELECT pisen, d_min, d_sec
  FROM cd
 WHERE d_min NOT BETWEEN 5 AND 7;

-- vybrat písně oblíbených interpretů – množinový operátor IN
SELECT interpret, pisen
  FROM cd
 WHERE interpret IN ('The Beatles', 'The Cranberies');

-- vybrat písně neoblíbených interpretů (operátor NOT IN)
SELECT interpret, pisen
  FROM cd
 WHERE interpret NOT IN ('The Beatles', 'The Cranberies');

-- vybrat písně interpretů začínající The – operátor LIKE
SELECT interpret, pisen
  FROM cd
 WHERE interpret LIKE 'The %';

-- negace operátoru LIKE
SELECT interpret, pisen
  FROM cd
 WHERE interpret NOT LIKE 'The %';
```

Ve vyhledávacím řetězci lze použít tyto dva speciální znaky:

- **%** (procento) - odpovídá libovolnému počtu libovolných znaků

- _ (podtržítko) - odpovídá právě jednomu libovolnému znaku

```
-- použití tzv. escape znaku pro hledání 50%
SELECT pisen, poznamka
  FROM cd
 WHERE poznamka LIKE '%50#%%' ESCAPE '#';

-- vybrat písně, které nemají vyplněnu poznámku – IS NULL
SELECT pisen, poznamka
  FROM cd
 WHERE poznamka IS NULL;

-- vybrat písně, které mají poznámku vyplněnu – IS NOT NULL
SELECT pisen, poznamka
  FROM cd
 WHERE poznamka IS NOT NULL;
```

## Dotazy nad jednou tabulkou - řazení

Výsledná data, která získáme zadaným dotazem, nejsou nijak uspořádána. Ve skutečnosti SŘBD tato data vypisuje chronologicky, tj. data jsou seřazena podle času, kdy byla do databáze vložena. Toto kritérium nám rozhodně nestačí a občas požadujeme vypisovaná data seřadit podle našeho kritéria. K tomuto účelu slouží v příkazu **SELECT** klauzule **ORDER BY**.

```
SELECT interpret, pisen
  FROM cd
 ORDER BY interpret ASC;

SELECT interpret, pisen
  FROM cd
 ORDER BY interpret DESC;
```

Směr řazení hodnot udává klíčové slovo :

- **ASC** vzestupné řazení (ASCending order); výchozí
- **DESC** sestupné řazení (DESCending order)

```
-- seřazení interpretů proti abecedě,
-- abecedně seřazeny písně každému interpretovi
SELECT interpret, pisen
  FROM cd
 ORDER BY interpret DESC, pisen ASC;

-- řazení podle výrazu využívá jeho alias
SELECT pisen, d_min * 60 + d_sec AS stopaz
  FROM cd
 ORDER BY stopaz DESC;

-- to samé, použito číselné označení sloupce
SELECT pisen, d_min * 60 + d_sec
  FROM cd
 ORDER BY 2 DESC;
```

## Dotazy nad jednou tabulkou - agregační funkce

```
-- funkce COUNT spočítá všechny (i neúplné) záznamy tabulky
SELECT COUNT(*) FROM cd;

-- počet záznamů, kde je vyplněna položka pisen
```

```
SELECT COUNT(pisen) FROM cd;

-- počet záznamů, kde je vyplněna položka interpret
SELECT COUNT(interpret) FROM cd;

-- počet záznamů, kde je vyplněna položka poznamka – bude jiný
SELECT COUNT(poznamka) FROM cd;

-- počet různých interpretů v tabulce
SELECT COUNT(DISTINCT interpret) FROM cd;

-- ostatní agregační funkce
SELECT MIN(d_min), MAX(d_min), SUM(d_min), AVG(d_min) FROM cd;
```

**Poznámka:**

- agregační funkce nelze použít v klauzuli WHERE.
- agregační funkce nelze vnořovat do sebe.

```
-- každému interpretovi (tvoří skupinu atributů) spočítám,
-- kolik písní mám a jak dlouho budou všechny hrát,
-- celé to abecedně seřadím
SELECT interpret,
       COUNT(stopa) AS pocet_stop,
       SUM(d_min * 60 + d_sec) AS delka
  FROM cd
 GROUP BY interpret
 ORDER BY interpret;
```

**Poznámka:**

Aby dotaz byl korektní, musí klauzule GROUP BY obsahovat všechny neagregované sloupce, jejichž hodnotu chceme z dotazu znát.

```
-- ten samý dotaz s podmínkou (HAVING), že chci jen ty
-- interprety, kteří nabízí max. 12 písní a budou hrát
-- alespoň 5 minut (300 sekund)
-- v podmínce HAVING nelze používat aliasy a výrazy zde musí
-- být jen agregované
SELECT interpret,
       COUNT(stopa) AS pocet_stop,
       SUM(d_min * 60 + d_sec) AS delka
  FROM cd
 GROUP BY interpret
HAVING SUM(d_min * 60 + d_sec) > 300 AND COUNT(stopa) < 12
 ORDER BY interpret;
```

**Poznámka:**

- v klauzuli HAVING nejdou použít nově definované názvy sloupců.
- v klauzuli HAVING lze použít jen agregační funkce nebo konstanty.

## Dotazy nad jednou tabulkou - vnořený dotaz

```
-- hledám nejdéle hrající písně
-- vnořený dotaz vrací právě jednu hodnotu
SELECT pisen, d_min, d_sec
  FROM cd
 WHERE d_min * 60 + d_sec = (SELECT MAX(d_min * 60 + d_sec) FROM cd);

-- to samé, použit množinový operátor ALL
```

```
-- vnořený dotaz vrací množinu hodnot
SELECT pisen, d_min, d_sec
  FROM cd
 WHERE d_min * 60 + d_sec >= ALL(SELECT d_min * 60 + d_sec FROM cd);

-- hledám všechny písně, které nejsou nejdéle hrající
-- použit množinový operátor ANY/SOME
-- vnořený dotaz vrací množinu hodnot
SELECT pisen, d_min, d_sec
  FROM cd
 WHERE d_min * 60 + d_sec < ANY(SELECT d_min * 60 + d_sec FROM cd);
```

**Poznámka:**

- Ve vnořeném dotazu nemá smysl používat klauzuli ORDER BY.
- Vnořený dotaz je vždy uzavřen mezi závorkami.
- Vnořený dotaz je ve výrazu vždy na pravé straně.
- Vnořený dotaz musí (až na pár výjimek) vracet vždy pouze jednu hodnotu.

## Dotazy nad více tabulkami - přirozené spojení (natural join, inner join, equijoin)

- možné kombinace zápisu přirozeného spojení v SŘBD Oracle
- spojení tabulek TITULY a PISEN
- dostaneme stejný výsledek, jako je uložen v tabulce CD

```
-- bez aliasů, plně kvalifikované názvy atributů
SELECT tituly.id, tituly.titul, pisne.pisen, pisne.d_min, pisne.d_sec
  FROM tituly, pisne
 WHERE tituly.id = pisne.id;

-- podmínka přirozeného spojení
SELECT t.id, t.interpret, t.titul,
       p.stopa, p.pisen, p.d_min, p.d_sec, p.poznamka
  FROM tituly t, pisne p
 WHERE t.id = p.id;

-- operátor INNER JOIN s podmínkou ON
SELECT t.id, t.interpret, t.titul,
       p.stopa, p.pisen, p.d_min, p.d_sec, p.poznamka
  FROM tituly t INNER JOIN pisne p ON t.id = p.id;

-- operátor JOIN s podmínkou ON
SELECT t.id, t.interpret, t.titul,
       p.stopa, p.pisen, p.d_min, p.d_sec, p.poznamka
  FROM tituly t JOIN pisne p ON t.id = p.id;

-- operátor INNER JOIN s konstrukcí USING
SELECT id, t.interpret, t.titul,
       p.stopa, p.pisen, p.d_min, p.d_sec, p.poznamka
  FROM tituly t INNER JOIN pisne p USING (id);

-- operátor JOIN s konstrukcí USING
SELECT id, t.interpret, t.titul,
       p.stopa, p.pisen, p.d_min, p.d_sec, p.poznamka
  FROM tituly t JOIN pisne p USING (id);

-- operátor NATURAL JOIN
SELECT id, t.interpret, t.titul,
       p.stopa, p.pisen, p.d_min, p.d_sec, p.poznamka
  FROM tituly t NATURAL JOIN pisne p;

-- operátor NATURAL INNER JOIN
```

```
SELECT id, t.interpret, t.titul,
       p.stopa, p.pisen, p.d_min, p.d_sec, p.poznamka
  FROM tituly t NATURAL INNER JOIN pisne p;
```

## Dotazy nad více tabulkami - levé spojení (left outer join)

- možné kombinace zápisu levého spojení v SŘBD Oracle
- ke každému záznamu z tabulky PREDMETY hledám odpovídající data z tabulky ROZVRH

```
-- operátor LEFT OUTER JOIN
SELECT p.katedra, p.zkratka,
       r.semestr, r.den, r.h_od, r.h_do, r.poznamka
  FROM predmety p LEFT OUTER JOIN rozvrh r
    ON p.id = r.id_predm;

-- specifická syntaxe Oracle
SELECT p.katedra, p.zkratka,
       r.semestr, r.den, r.h_od, r.h_do, r.poznamka
  FROM predmety p, rozvrh r
 WHERE p.id = r.id_predm(+);
```

## Dotazy nad více tabulkami - pravé spojení (right outer join)

- možné kombinace zápisu levého spojení v SŘBD Oracle
- ke každému záznamu z tabulky ROZVRH hledám odpovídající data z tabulky PREDMETY

```
-- operátor RIGHT OUTER JOIN
SELECT p.katedra, p.zkratka,
       r.semestr, r.den, r.h_od, r.h_do, r.poznamka
  FROM predmety p RIGHT OUTER JOIN rozvrh r
    ON p.id = r.id_predm;

-- specifická syntaxe Oracle
SELECT p.katedra, p.zkratka,
       r.semestr, r.den, r.h_od, r.h_do, r.poznamka
  FROM predmety p, rozvrh r
 WHERE r.id_predm = p.id(+);
```

## Dotazy nad více tabulkami - plné spojení (full outer join)

```
-- levé a pravé spojení provedeno současně
-- operátor FULL OUTER JOIN
SELECT p.katedra, p.zkratka,
       r.semestr, r.den, r.h_od, r.h_do, r.poznamka
  FROM predmety p FULL OUTER JOIN rozvrh r
    ON p.id = r.id_predm;
```

## Dotazy nad více tabulkami - polospojení (semijoin)

- hledám jen rozvrhované předměty
- možné kombinace zápisu polospojení v SŘBD Oracle
- vybírám data z tabulky PREDMETY na základě existence odpovídajících dat (podmínka polospojení) v tabulce ROZVRH

```
-- operátor EXISTS s poddotazem
SELECT p.zkratka, p.nazev
  FROM predmety p
 WHERE EXISTS ( SELECT *
                  FROM rozvrh r
                 WHERE p.id = r.id_predm);

-- operátor IN s poddotazem
SELECT p.zkratka, p.nazev
  FROM predmety p
 WHERE p.id IN ( SELECT r.id_predm
                   FROM rozvrh r
                  WHERE r.id_predm IS NOT NULL);

-- lze realizovat přirozeným spojením s DISTINCT
SELECT DISTINCT p.zkratka, p.nazev
  FROM predmety p, rozvrh r
 WHERE p.id = r.id_predm;
```

## Dotazy nad více tabulkami - antijoin

- hledám jen nerozvrhované předměty
- možné kombinace zápisu antijoinu v SŘBD Oracle
- vybírám data z tabulky PREDMETY na základě neexistence odpovídajících dat v tabulce ROZVRH

```
-- operátor NOT EXISTS s poddotazem
SELECT p.zkratka, p.nazev
  FROM predmety p
 WHERE NOT EXISTS ( SELECT *
                      FROM rozvrh r
                     WHERE p.id = r.id_predm);

-- operátor NOT IN s poddotazem
SELECT p.zkratka, p.nazev
  FROM predmety p
 WHERE p.id NOT IN ( SELECT r.id_predm
                       FROM rozvrh r
                      WHERE r.id_predm IS NOT NULL);
```

## Dotazy nad více tabulkami - selfjoin

- specifický typ přirozeného spojení
- spojuje jednu a tu samou tabulku – nutné aliasy

```
-- podmínka přirozeného spojení
SELECT zamestnanec.prijmeni, zamestnanec.jmeno,
       vedouci.prijmeni, vedouci.jmeno
  FROM osoby zamestnanec, osoby vedouci
 WHERE zamestnanec.id_nad = vedouci.id_osoby;

-- operátor INNER JOIN s podmínkou ON
SELECT zamestnanec.prijmeni, zamestnanec.jmeno,
       vedouci.prijmeni, vedouci.jmeno
  FROM osoby zamestnanec INNER JOIN osoby vedouci
    ON zamestnanec.id_nad = vedouci.id_osoby;
```

## Dotazy nad více tabulkami - kartézský součin

- každému předmětu nabízím všechny rozvrhové časy
- možné kombinace zápisu kartézského součinu v SŘBD Oracle

```
-- operátor CROSS JOIN – vyžaduje disjunktní množiny atributù
SELECT p.katedra, p.zkratka, p.nazev,
       r.semestr, r.den, r.h_od, r.h_do, r.poznamka
  FROM predmety p CROSS JOIN rozvrh r;

-- bez operátoru či podmínky
SELECT p.katedra, p.zkratka, p.nazev,
       r.semestr, r.den, r.h_od, r.h_do, r.poznamka
  FROM predmety p, rozvrh r;
```

## Dotazy z více dotazů - množinové operace

```
-- nad kompatibilními dotazy lze použít množinové operátory
-- operátor UNION – sjednocení, odstraňuje duplicitní data
-- komutativní operace, lze použít nad větším počtem dotazů
-- výsledkem je FULL OUTER JOIN
SELECT p.katedra, p.zkratka,
       r.semestr, r.den, r.h_od, r.h_do, r.poznamka
  FROM predmety p LEFT OUTER JOIN rozvrh r
    ON p.id = r.id_predm
UNION
SELECT p.katedra, p.zkratka,
       r.semestr, r.den, r.h_od, r.h_do, r.poznamka
  FROM predmety p RIGHT OUTER JOIN rozvrh r
    ON p.id = r.id_predm;

-- operátor UNION ALL – sjednocení, ponechá duplicitní data
-- komutativní operace, lze použít nad větším počtem dotazů
SELECT p.katedra, p.zkratka,
       r.semestr, r.den, r.h_od, r.h_do, r.poznamka
  FROM predmety p LEFT OUTER JOIN rozvrh r
    ON p.id = r.id_predm
UNION ALL
SELECT p.katedra, p.zkratka,
       r.semestr, r.den, r.h_od, r.h_do, r.poznamka
  FROM predmety p RIGHT OUTER JOIN rozvrh r
    ON p.id = r.id_predm;

-- operátor INTERSECT – průnik, odstraňuje duplicitní data
-- komutativní operace, lze použít nad větším počtem dotazů
-- výsledkem je INNER JOIN, tj. přirozené spojení
SELECT p.katedra, p.zkratka,
       r.semestr, r.den, r.h_od, r.h_do, r.poznamka
  FROM predmety p LEFT OUTER JOIN rozvrh r
    ON p.id = r.id_predm
INTERSECT
SELECT p.katedra, p.zkratka,
       r.semestr, r.den, r.h_od, r.h_do, r.poznamka
  FROM predmety p RIGHT OUTER JOIN rozvrh r
    ON p.id = r.id_predm;

-- operátor MINUS – rozdíl
-- záleží na pořadí dotazů, lze aplikovat jen nad dvěma dotazy
-- výsledek je srovnatelný s ANTIJOINem
SELECT p.katedra, p.zkratka,
       r.semestr, r.den, r.h_od, r.h_do, r.poznamka
  FROM predmety p LEFT OUTER JOIN rozvrh r
    ON p.id = r.id_predm
```

```
MINUS
SELECT p.katedra, p.zkratka,
       r.semestr, r.den, r.h_od, r.h_do, r.poznamka
  FROM predmety p RIGHT OUTER JOIN rozvrh r
    ON p.id = r.id_predm;
```

# Data definition language

## Vytvoření tabulky

- každý atribut má svůj datový typ
- každé SŘBD nabízí svoji množinu datových typů
- ve všech příkladech použiji syntaxi a datové typy ze SŘBD Oracle

### Vybrané datové typy SŘBD Oracle

| | |
|---|---|
| NUMBER | libovolné číslo |
| NUMBER(p) | celé číslo, p - počet cifer čísla v rozsahu 1 až 38 |
| NUMBER(p, s) | číslo, s - přesnost čísla v rozsahu -84 až 127 (záporná na desítky, kladná na desetiny) |
| VARCHAR2(n) | textový řetězec dlouhý maximálně n bytů, n - počet bytů v rozsahu 1 až 4000 |
| VARCHAR2(n CHAR) | textový řetězec dlouhý maximálně n znaků, délka řetězce nesmí překročit velikost 4000 bytů |
| CHAR(n) | textový řetězec dlouhý právě n bytů, n - počet bytů v rozsahu 1 až 2000 |
| DATE | datum a čas v rozsahu 1.1.4712 př.n.l. až 31.12.9999 n.l. |
| BLOB | velký binární objekt velikosti až 4 GB |

```
CREATE TABLE osoby (
  os_cislo   NUMBER(5)    NOT NULL,
  rc         VARCHAR2(30) NOT NULL,
  jmeno      VARCHAR2(30) NOT NULL,
  prijmeni   VARCHAR2(30) NOT NULL,
  dat_naroz  DATE,
  pohlavi    CHAR(1)      NOT NULL,
  telefon    VARCHAR2(50),
  plat       NUMBER(5),
  cislo_prac NUMBER(5)    NOT NULL
);

CREATE TABLE pracoviste (
  cislo_prac NUMBER(5)    NOT NULL,
  nazev      VARCHAR2(30) NOT NULL
);

CREATE TABLE predmety (
  zkratka   VARCHAR2(5),
  katedra   VARCHAR2(3),
  nazev     VARCHAR2(30) NOT NULL,
  kredity   NUMBER(2)    NOT NULL
);
```

## Změna tabulky: přidání a odebrání sloupce

```
ALTER TABLE pracoviste ADD ( adresa VARCHAR2(80) );

ALTER TABLE pracoviste DROP ( adresa );
```

## Entitní integritní omezení

```
-- primární klíč pro tabulku OSOBY
ALTER TABLE osoby ADD CONSTRAINT pk_osoby PRIMARY KEY (os_cislo);

-- primární klíč pro tabulku OSOBY
ALTER TABLE pracoviste ADD CONSTRAINT pk_pracoviste PRIMARY KEY (cislo_prac);

-- primární klíč pro tabulku PREDMETY
ALTER TABLE predmety ADD CONSTRAINT pk_predmety PRIMARY KEY (zkratka, katedra);

-- nastavení unikátnosti hodnot atributu RC v tabulce OSOBY
ALTER TABLE osoby ADD CONSTRAINT unikatni_rc UNIQUE (rc);

-- nastaveni unikatnosti hodnot atributu NAZEV v tabulce PRACOVISTE
ALTER TABLE pracoviste ADD CONSTRAINT unikatni_nazev UNIQUE (nazev);
```

## Doménové integritní omezení

```
-- hlídání velikosti platu osoby
ALTER TABLE osoby ADD CONSTRAINT kontrola_platu CHECK ( plat > 5000 AND plat < 20000 );
```

## Referenční integritní omezení

```
-- restriktivni omezeni, plati jako vychozi
ALTER TABLE osoby
ADD CONSTRAINT fk_prac FOREIGN KEY ( cislo_prac )
REFERENCES pracoviste ( cislo_prac );

-- pracovnici ruseneho pracoviste budou mit cislo pracoviste NULL
ALTER TABLE osoby
ADD CONSTRAINT fk_prac FOREIGN KEY ( cislo_prac )
REFERENCES pracoviste ( cislo_prac ) ON DELETE SET NULL;

-- kaskadni mazani
-- krome ruseneho pracoviste budou smazani i vsichni jeho pracovnici
ALTER TABLE osoby
ADD CONSTRAINT fk_prac FOREIGN KEY ( cislo_prac )
REFERENCES pracoviste ( cislo_prac ) ON DELETE CASCADE;
```

## Zrušení integritního omezení

```
-- zruseni primarniho klice
ALTER TABLE osoby DROP CONSTRAINT pk_osoby;

-- zruseni domenoveho integritniho omezeni
ALTER TABLE osoby DROP CONSTRAINT kontrola_platu;

-- zruseni ciziho klice
ALTER TABLE osoby DROP CONSTRAINT fk_prac;
```

## Vyprázdnění tabulky

- vyprázdní obsah tabulky, její definice zůstane zachována
- rychlejší operace než příkazy jazyka DML
- bez omezení, nevratná operace

```
TRUNCATE TABLE osoby;
```

## Zrušení tabulky

```
-- smazani obsahu tabulky i jeji definice
DROP TABLE predmety;

-- ke smazani nedojde, brani tomu referencni integrita (krome CASCADE)
DROP TABLE osoby;

-- smazani tabulky vcetne vsech referencni integritnich omezeni, ktere by tomu branily
DROP TABLE osoby CASCADE CONSTRAINTS;

-- plati pouze v SRBD Oracle
-- tabulka vcetne obsahu bude skutecne smazana, jinak je presunuta do odpadkoveho kose
DROP TABLE pracoviste PURGE;
```

## Vytvoření databázové pohledu

- virtuální relace (fyzicky neexistuje), okamžitě promítá změnu dat do odpovědi
- v dotazu lze kromě tabulek používat i pohledy
- v dotazu NELZE použít řazení, tj. konstrukci ORDER BY

```
-- pojmenovani sloupcu pohledu vyuziva nazvy tabulky
CREATE VIEW cd_beatles AS
SELECT *
  FROM cd
 WHERE interpret LIKE '%Beatles%';

-- vlastni pojmenovani sloupcu pohledu
CREATE VIEW cd_prehled ( skladatel, dilo ) AS
SELECT DISTINCT interpret, titul
  FROM cd;

-- pohledy byvaji zpravidla spojovany se slozitejsimy dotazy
CREATE VIEW cd_stat ( skladatel, dilo, pocet_stop, delka ) AS
SELECT a.interpret,
       a.titul,
           count(b.stopa),
           sum(b.d_min)
  FROM tituly a, pisne b
 WHERE a.id = b.id
 GROUP BY  a.interpret, a.titul;
```

## Zrušení databázového pohledu

Zrušení databázového pohledu nemá žádný vliv na uložená data v databázi.

```
DROP VIEW cd_beatles;

DROP VIEW cd_prehled;

DROP VIEW cd_stat;
```

### Vytvoření indexu

logicky seřadí záznamy tabulky podle daného atributu (množiny atributů)

- unikátní index (hodnoty se neopakují) - generují se automaticky
  - primární klíč (PRIMARY KEY)
  - unikátní atribut či množina atributů (UNIQUE)
- běžný index (hodnoty se mohou opakovat) - definuje programátor sám
  - cizí klíč (FOREIGN KEY)
  - libovolný atribut (množina atributů)

```
-- unikatni index pro primarni klic
-- obvykle ma shodny nazev s integritnim omezenim
CREATE UNIQUE INDEX pk_osoby ON osoby (os_cislo);

-- bezny index (pro cizi klic)
-- nazev se muze shodovat s integritnim omezenim
CREATE INDEX fk_prac ON osoby (cislo_prac);
```

### Zrušení indexu

- specifickým příkazem
- zrušení tabulky také zruší všechny její indexy

```
-- zruseni unikatniho indexu
DROP INDEX pk_osoby;

-- zruseni bezneho indexu
DROP INDEX fk_prac;
```

# Data manipulation language

Před manipulací s daty v databázi se vytvoříme kopie těchto tabulek:

```
DROP TABLE rozvrh;
DROP TABLE predmety;
DROP TABLE mistnosti;

CREATE TABLE predmety  AS SELECT * FROM predmety;
CREATE TABLE rozvrh    AS SELECT * FROM rozvrh;
CREATE TABLE mistnosti AS SELECT * FROM mistnosti;

ALTER TABLE predmety ADD CONSTRAINT pk_predmety PRIMARY KEY (id);
```

### Vkládání dat do databáze

Jedním příkazem INSERT lze vložit jeden záznam do jedné tabulky:

```
-- vkladani celeho zaznamu vcetne nevyplnenych polozek (NULL)
-- vyzaduje znalost poradi atributu v definici tabulky
INSERT  INTO predmety
VALUES ( 99, 'DS', 'KIV', 'Distribuovane systemy', NULL, NULL, NULL );

-- skonci chybou - porusena integrita primarniho klice
INSERT  INTO predmety
VALUES ( 99, 'PP', 'KIV', 'Pocitace a programovani', NULL, NULL, NULL );

-- vkladani jen vybranych polozek noveho zaznamu
-- jejich poradi lze definovat dle uvazeni
INSERT  INTO predmety ( nazev, katedra, zkratka, kredity, id )
VALUES ( 'Pocitace a programovani', 'KIV', 'PP', 5, 98 );
```

Výsledek dotazu (obvykle více záznamů) lze uložit do tabulky jedním příkazem INSERT:

```
INSERT INTO rozvrh (id_predm, semestr, poznamka)
SELECT id, 'LS', 'Garant: '||garant
  FROM predmety
 WHERE katedra = 'KIV';
```

## Aktualizace dat v databázi

Aktualizací dat se rozumí změna údajů v existujících záznamech v tabulce. Vždy dochází k přepsání původní hodnoty, případně nastavení hodnoty místo prázdné položky.

```
-- aktualizace vsech zaznamu v tabulce
UPDATE predmety
   SET kredity = 20;

-- aktualizace jen vybranych zaznamu
UPDATE predmety
   SET kredity = 20
 WHERE katedra = 'KIV'
   AND zkratka = 'KP';

-- vymazani hodnoty polozky ve vybranem zaznamu
UPDATE predmety
   SET pocet_studentu = NULL
 WHERE katedra = 'KIV'
   AND zkratka = 'PC';

-- aktualizace nekolika polozek v zaznamu najednou
-- tak, kde neni nastaven pocet studentu, zustava hodnota NULL
UPDATE predmety
   SET kredity = 3,
       pocet_studentu = pocet_studentu + 10;
```

## Smazání/odstranění dat z databáze

- mažou se vždy celé záznamy
- mazání jednotlivých položek se koná příkazem UPDATE
- úspěch smazání dat v tabulce je závislé na nastavení referenčních integritních omezení

```
-- vymazani celeho obsahu tabulky
-- to same vykona (jinak) prikaz TRUNCATE
DELETE
```

```
  FROM rozvrh;

-- smazani jen vybranych zaznamu
DELETE
  FROM rozvrh
 WHERE poznamka LIKE 'Seminar%';
```

# Transaction control language

## Transakční zpracování dat

- všechny příkazy jazyka DML běží v tzv. transakční režimu
- transakce začíná prvním příkazem jazyka DML
- všechny tabulky a pohledy použité v transakci jsou výlučně zamykány
- každá transakce končí:
    - jejím potvrzením (spácháním)
    - jejím zrušením (odvoláním, odrolováním)

### Spáchání transakce

```
COMMIT;
```

Poznámka:

Transakci také spáchá u většiny SŘBD jakýkoliv příkaz jazyků DDL a DCL!

### Odrolování transakce

```
ROLLBACK;
```

### Bod uložení v transakci

- v transakci lze definovat tzv. body uložení
- slouží k odrolování jen části transakce, právě danému k bodu uložení

```
-- zahajeni transakce
DELETE FROM predmety WHERE zkratka = 'HB';
SELECT zkratka FROM predmety;

-- nastaveni bodu lozeni
SAVEPOINT prvni;

-- dalsi prikaz DELETE v transakci
DELETE FROM predmety WHERE zkratka LIKE 'P%';
SELECT zkratka FROM predmety;

-- odvolani vsech operaci k danemu bodu ulozeni
ROLLBACK TO SAVEPOINT prvni;
SELECT zkratka FROM predmety;
```

**Nastavení režimu transakce**

```
-- transakce dovoluje pouze cist data, ne zapis
SET TRANSACTION READ ONLY;

-- vychozi nastaveni
-- transakce dovoluje cist i zapisovat data
SET TRANSACTION READ WRITE;
```

## Explicitní zamykání tabulek a záznamů

Každou tabulku lze zamknout

- výlučním (exklusivním) zámkem
- sdíleným zámkem

```
-- uzamceni cele tabulky vylucnim zamkem
-- zadna jina transakce nema do teto tabulky pristup
LOCK TABLE rozvrh IN EXCUSIVE MODE;

-- uzamceni cele tabulky sdilenym zamkem
-- ostatni transakce mohou z dane tabulky pouze cist
LOCK TABLE rozvrh IN SHARE MODE;
```

Některé SŘBD (zpravidla v nejvyšší edici) umožní zamykání jen vybraných záznamů

```
-- uzamknuti jen rozvrhovych akci konanych v ZS
SELECT *
  FROM rozvrh
 WHERE semestr = 'ZS'
   FOR UPDATE;
```

## Odemknutí zamčených tabulek a záznamů

Odemknutí vybrané tabulky nelze, všechny uzamčené objekty v transakci se odemykají ukončením transakce.

```
-- spachani transakce a take odemceni vsech tabulek uzamcenych v transakci
COMMIT;

-- zruseni transakce a take odemceni vsech tabulek uzamcenych v transakci
ROLLBACK;
```

# Data control language

- řízení přístupu k datům (jiného uživatele)
- základní oprávnění:
    - SELECT - čtení z dané tabulky
    - INSERT - vkládání nových dat do tabulky
    - UPDATE - aktualizace stávajících dat v tabulce
    - DELETE - mazání stávajících dat v tabulce

## Přidělení oprávnění

```
-- prideleni prava SELECT nad tabulkou PREDMETY uzivateli soused
GRANT SELECT ON predmety TO soused;

-- prideleni vice prav najednou
GRANT INSERT, UPDATE, DELETE ON predmety TO soused;

-- prideleni opravneni vsem uzivatelum (PUBLIC)
GRANT SELECT ON predmety TO PUBLIC;

-- prideleni prava SELECT nad tabulkou PREDMETY uzivateli soused
-- navic uzivatel soused ziskal moznost toto opravnini sirit dale
GRANT SELECT ON predmety TO soused WITH ADMIN OPTION;
```

## Přístup k datům, ke kterým jsme získali přístup

- uvádí se vždy plně kvalifikované jméno tabulky
- předpokládáme, že jsme získali přístup k objektům uživatele ZIMA

```
SELECT * FROM zima.predmety;
```

## Odebrání oprávnění

- lze odebrat to oprávnění, které bylo přiděleno
- lze odebrat oprávnění tomu, komu bylo přiděleno

```
-- odebrani prava SELECT nad tabulkou PREDMETY uzivateli soused
-- odebira tez moznost pridelene pravo sirit dale, pokud bylo prideleno
REVOKE SELECT ON rozvrh FROM soused;

-- odebrani vice pridelenych prav najednou
REVOKE INSERT, UPDATE, DELETE ON rozvrh FROM soused;

-- odebrani opravneni vsem uzivatelum
REVOKE SELECT ON predmety FROM PUBLIC;
```

## Uživatelské role

Uživatelská práva/oprávnění můžeme sdružovat do rolí a uživatelům přidělovat celé role.

### Definice role

```
-- role pro cteni dat z tabulek
CREATE ROLE role_cteni;

-- prideleni dilcich opravneni roli
GRANT SELECT ON predmety TO role_cteni;
GRANT SELECT ON rozvrh TO role_cteni;
GRANT SELECT ON mistnosti TO role_cteni;
```

```
-- role pr zapis dat do tabulek
CREATE ROLE role_zapis;

-- prideleni dilcich opravneni roli
GRANT INSERT, UPDATE, DELETE ON predmety TO role_zapis;
GRANT INSERT, UPDATE, DELETE ON rozvrh TO role_zapis;
GRANT INSERT, UPDATE, DELETE ON mistnosti TO role_zapis;
```

## Přidělení role

```
-- prideleni role uzivateli
GRANT role_zapis TO soused;

-- prideleni role vsem uzivatelum
-- uzivatele mohou tuto roli sirit dal i ji modifikovat (pripadne smazat)
GRANT role_cteni TO PUBLIC WITH ADMIN OPTION;
```

## Odebrání role

```
-- odebrani role uzivateli
REVOKE role_zapis FROM soused;

-- odebrani role vsem uzivatelum
-- odebrani moznosti sirit, menit pridelenou roli, pokud to bylo prideleno
REVOKE role_cteni FROM PUBLIC;
```