

<http://www.dcs.fmph.uniba.sk/~plachetk/TEACHING/DB1>

Tomáš Plachetka

Fakulta matematiky, fyziky a informatiky,
Univerzita Komenského, Bratislava

Zima 2025–2026

SQL, Structured Query Language má zhruba 2 časti: **1.DDL, Data Definition Language** (vytváranie používateľov, vytváranie databázových schém, ...); **2.DML, Data Manipulation Language (práca so schémou)**

Základné príkazy DML:

INSERT vkladanie záznamov do tabuľky

DELETE rušenie záznamov v tabuľke

UPDATE modifikácia záznamov v tabuľke

SELECT výber (vyhľadávanie) záznamov v tabuľke

Jazyk SQL je bohatý. Spätnú kompatibilitu zabezpečuje štandardizácia: SQL-87 (ANSI), SQL-92 (SQL2), SQL-99 (SQL3), SQL-2003, SQL-2006 (ISO/IEC 9075-14:2006). Oficiálny štandard nie je voľne dostupný (dá sa kúpiť od ISO, resp. ANSI), ale 99% je k dispozícii zadarmo napr. na

<http://www.wiscorp.com/SQLStandards.html>

Príkaz SELECT

Všeobecný tvar príkazu SELECT (detaily zanedbáme):

SELECT	zoznam atribútov
FROM	zoznam relácií (kartézsky súčin, resp. join)
WHERE	selekčná podmienka (spájacie podmienky pre jeden záznam)
GROUP BY	zoznam grupovacích atribútov
HAVING	selekčná podmienka v agregovanej relácii
ORDER BY	usporiadanie výslednej relácie
...	napr. LIMIT, EXPORT atď.

Základný tvar príkazu **SELECT** (detaily zanedbáme):

SELECT	zoznam atribútov
FROM	zoznam relácií
WHERE	selekčná podmienka (spájacie podmienky)

Operačná sémantika (výpočet výsledku) SELECT... FROM... WHERE...:

1. Urobí sa **kartézsky súčin relácií za FROM**
 - Na ten kartézsky súčin sa aplikuje **selekčná podmienka za WHERE** (ktorá zachová len tie riadky, ktoré tú podmienku spĺňajú)
 - Nakoniec sa urobí **projekcia na atribúty, ktoré sú za SELECT** (ktorá zachová len tie stĺpce, ktoré sú vymenované v projekcii)

Príkaz SELECT: selekcia

produkt(Meno, Cena, Kategoria, Vyrobcu)

answer(ziarovka, C, K, V) $\leftarrow$ produkt(ziarovka, C, K, V), $C > 100$.

SELECT *

FROM produkt

WHERE Meno = 'ziarovka' AND Cena > 100;

V podmienke WHERE je možné použiť:

- mená atribútov

- porovnávacie operátory: =, <>, <, >, <=, >=

- aritmetické operátory: +, -, *, /

- operácie s reťazcami, napr. zreťazenie: ||, &

- logické spojky: NOT, AND, OR

- porovnanie regulárnych výrazov: s LIKE p

- špeciálne funkcie pre dátum a čas a ďalšie built-in funkcie

Príkaz SELECT: projekcia a premenovanie atribútov

produkt(Meno, Cena, Kategoria, Vyrobcu)

answer(**M**, **C**) $\leftarrow$ produkt(M, C, _, _), M = ziarovka, C > 100.

Projekcia (výber podmnožiny atribútov):

SELECT Meno, Cena

FROM produkt

WHERE Meno = 'ziarovka' AND Cena > 100;

Premenovanie atribútov vo výslednej relácii:

SELECT Meno **AS** Nazov, Cena **AS** Ponuka

FROM produkt

WHERE Meno = 'ziarovka' AND Cena > 100;

Príkaz SELECT: usporiadanie výsledkov

produkt(Meno, Cena, Kategoria, Vyrobcu)

```
SELECT  Meno, Cena  
FROM    produkt  
WHERE   Meno = 'ziarovka' AND Cena > 100;  
ORDER BY Kategoria DESC, Meno ASC;
```

ASC znamená rastúce usporiadanie (ascending)

DESC znamená klesajúce usporiadanie (descending)

Ak sa neuvedie ASC ani DESC, tak default je ASC

V zozname za ORDER BY môže byť viacej atribútov, pre každý nezávisle môže byť usporiadanie vzostupné alebo klesajúce
Priorita je zľava doprava.

V danom prípade sa triedi zostupne podľa hodnôt atribútu Kategoria; až keď sú niektoré hodnoty rovnaké, tak sa triedi vzostupne podľa hodnôt atribútu Meno

Príkaz SELECT: zjednotenie, prienik, rozdiel

osoba(Meno, Rodne_cislo, Telefon, Mesto)

kontrakt(Predavajuci, Kupujuci, Sklad, Vyrobok)

answer(M) $\leftarrow$ osoba(M, _, _, trnava).

answer(K) $\leftarrow$ kontrakt(_, K, zohor, _).

```
(SELECT  Meno
FROM      osoba
WHERE     Mesto = 'trnava')
```

UNION

```
(SELECT  Kupujuci AS Meno
FROM      kontrakt
WHERE     Sklad = 'zohor');
```

Podobne sa používa **INTERSECT** (prienik) a **EXCEPT** (rozdiel).
Pozor, tabuľky musia byť kompatibilné, t.j. musia mať rovnaké atribúty (inak treba atribúty premenovať tak, aby mali rovnaké mená)

Príkaz SELECT: spojenie (join)

osoba(Meno, Rodne_cislo, Telefon, Mesto)
kontrakt(Predavajuci, Kupujuci, Sklad, Vyrobok)
answer(M, S) ←
osoba(**M**, _, _, trnava),
kontrakt(_, **M**, S, vysavac).

```
SELECT  Meno, Sklad
FROM    osoba, kontrakt
WHERE   osoba.Meno = kontrakt.Kupujuci AND Mesto = 'trnava'
        AND Vyrobok = 'vysavac';
```


Príkaz SELECT: jednoznačnosť atribútov

osoba(Meno, Rodne_cislo, Telefon, Mesto)
kontrakt(Predavajuci, Kupujuci, Sklad, Vyrobok)
produkt(Meno, Cena, Kategoria, Vyrobcu)
answer(M) ←
osoba(M, _, _, _),
kontrakt(_, M, _, V),
produkt(V, _, elektronika, _).

Mená osôb, ktoré si kúpili elektroniku:

```
SELECT  osoba.Meno
FROM    osoba, kontrakt, produkt
WHERE   osoba.Meno = Kupujuci
        AND Vyrobok = produkt.Meno
        AND Kategoria = 'elektronika';
```

Príkaz SELECT: aliasy relácií a atribútov

produkt(Meno, Cena, Kategoria, Vyrobcu)

answer(M1, M2) ←
produkt(M1, _, **K**, _),
produkt(M2, _, **K**, _).

Dvojice výrobkov, ktoré majú rovnakú kategóriu:

```
SELECT  p1.Meno AS M1, p2.Meno AS M2
FROM    produkt p1, produkt p2
WHERE   p1.Kategoria = p2.Kategoria;
```

Toto je „dobrý programátorský štýl“. Aliasy relácií odporúčam používať **vždy** (aj tam, kde to nie je nutné)

Príkaz SELECT: vnútorné a vonkajšie spojenia

produkt(Meno, Cena, Kategoria, Vyrobcu)

kontrakt(Predavajuci, Kupujuci, Sklad, Vyrobovok)

Len tie výrobky, ktoré sú v oboch reláciách produkt a kontrakt:

```
SELECT Meno, Cena, Sklad
```

```
FROM produkt JOIN kontrakt ON Meno = Vyrobovok;
```

Všetky výrobky, ktoré sú v relácii produkt. Ak je niektorý výrobok zároveň v kontraktoch, doplní sa preň údaj o sklade (ak nie, doplní sa preň hodnota null):

```
SELECT Meno, Cena, Sklad
```

```
FROM produkt LEFT OUTER JOIN kontrakt ON  
Meno = Vyrobovok;
```

Príkaz SELECT: vnútorné spojenia (INNER JOIN)

Syntax spojení:

```
SELECT r.X, r.Y, r.Z  
FROM r JOIN s ON r.Y = s.Y;
```

znamená to isté čo

```
SELECT r.X, r.Y, r.Z  
FROM r INNER JOIN s ON r.Y = s.Y;
```

znamená to isté čo

```
SELECT r.X, r.Y, r.Z  
FROM r, s  
WHERE r.Y = s.Y;
```

Príkaz SELECT: vonkajšie spojenia (OUTER JOIN)

```
SELECT r.X, r.Y, s.Z  
FROM r LEFT OUTER JOIN s ON r.Y = s.Y;
```

znamená to isté čo

```
SELECT r.X, r.Y, s.Z  
FROM r, s  
WHERE r.Y = s.Y
```

UNION

```
SELECT r.X, r.Y, null AS Z  
FROM r  
WHERE r.Y NOT IN (SELECT Y FROM s);
```

Podobne sa definuje **RIGHT OUTER JOIN** a **FULL OUTER JOIN**

Príkaz SELECT: null hodnoty

SQL rozširuje všetky domény o hodnotu **null**. null predstavuje neznámu, neexistujúcu aj neurčenú hodnotu

Pozor, **SQL používa trojhodnotovú logiku**:

NOT		AND	true	false	null	OR	true	false	null
null	null	null	null	false	null	null	true	null	null

Je názorné predstaviť si $\text{true} = 1$, $\text{null} = 1/2$, $\text{false} = 0$.
Potom platí $\neg p = 1 - p$, $p \wedge q = \min(p, q)$, $p \vee q = \max(p, q)$.

Pre všetky ostatné operácie a funkcie platí: Ak niektorý z operandov resp. argumentov je null, potom aj výsledok je null

Dokonca aj $\text{null} = \text{null}$ vráti null. **Ale null IS null vráti true**

Príkaz SELECT: kontraintuitívne dotazy (null)

Príklad, na ktorom SELECT funguje (na prvý pohľad) kontraintuitívne:

A	B	C
null	6	7

```
SELECT r.C  
FROM r  
WHERE  $A < 1$  OR  $A \geq 1$ ;
```

Zdalo by sa, že tá selekčná podmienka platí vždy (ak A je typu INTEGER), takže tento SELECT by mal intuitívne vrátiť tabuľku s jedným stĺpcom a jedným riadkom, ktorý obsahuje hodnotu 7

Lenže pre $A = \text{null}$ sa výraz $A < 1$ OR $A \geq 1$ vyhodnotí ako null, nie ako true. **Tým pádom výsledkom tohto dotazu je prázdna tabuľka**

Vyskúšajte!

Príkaz SELECT: kontraintuitívne dotazy (join)

```
SELECT    r.A  
FROM      r, s, t  
WHERE     r.A = s.A OR r.A = t.A;
```

Zdá sa, že vyberá hodnoty A z r, ktoré sú v s alebo t. Skutočne?

```
SELECT    r.A  
FROM      r, s, t  
WHERE     r.A = s.A;
```

Zdá sa, že vyberá hodnoty A, ktoré sú v r a zároveň v s.
Skutočne?

Príkaz SELECT: poddotazy vracajúce reláciu

kontrakt(Predavajuci, Kupujuci, Sklad, Vyrobok)

produkt(Meno, Cena, Kategoria, Vyrobcia)

answer(V) $\leftarrow$
 produkt(P, _, _, V),
 kontrakt(_, jozef, _, P).

Výrobcovia, ktorých produkty kupuje Jozef:

```
SELECT Vyrobcia
FROM   produkt
WHERE  produkt.Meno IN
        (SELECT Vyrobok
         FROM   kontrakt
         WHERE  Kupujuci = 'jozef');
```

V takomto kontexte sa dá sa použiť aj:

s > ALL stĺpec relácie (true ak je s väčšie ako všetky hodnoty)

s > ANY stĺpec relácie (true ak je s väčšie ako niektorá hodnota)

Príkaz SELECT: poddotazy vracajúce reláciu (n-tice)

kontrakt(Predavajuci, Kupujuci, Sklad, Vyrobok)

osoba(Meno, Rodne_cislo, Telefon, Mesto)

answer(M1, T1, M2, T2) ←

osoba(M1, _, T1, _),

osoba(M2, _, T2, _),

kontrakt(M2, M1, _, _).

Kto od koho nakupuje, mená a telefónne čísla:

```
SELECT  o1.Meno as Kupujuci, o1.Telefon as Kupujuci_Tel,  
        o2.Meno as Predavajuci, o2.Telefon as Predavajuci_Tel  
FROM    osoba o1, osoba o2  
WHERE   (o1.Meno, o2.Meno) IN  
        (SELECT Kupujuci, Predavajuci  
         FROM    kontrakt);
```

Príkaz SELECT: poddotazy vracajúce práve 1 záznam

kontrakt(Predavajuci, Kupujuci, Sklad, Vyrobok)
osoba(Meno, Rodne_cislo, Telefon, Mesto)

```
SELECT kontrakt.Vyrobok
FROM kontrakt
WHERE Kupujuci =
      (SELECT Meno
       FROM osoba
       WHERE Rodne_cislo = '450626/7887');
```

Takýto poddotaz musí vrátiť práve jednu hodnotu. Inak nastane chyba (**run-time error**)

Príkaz SELECT: rozsah platnosti (scope) mien v dotaze

film(Nazov, Rok, Reziser, Dlzka)

answer(N) $\leftarrow$
film(N, R1, _, _),
film(N, R2, _, _),
not R1 = R2.

Nájdite názvy filmov, ktoré sa vyskytli viac než raz (Nazov nie je jednoznačným identifikátorom filmu, ale [Nazov, Rok] áno):

```
SELECT DISTINCT Nazov
FROM film AS f1
WHERE Rok < ANY
(SELECT Rok
 FROM film
 WHERE Nazov = f1.Nazov AND Rok <> f1.Rok);
```

Príkaz SELECT: agregáčné funkcie

produkt(Meno, Cena, Kategoria, Vytrobca)

```
SELECT  AVG(Cena)
FROM    produkt
WHERE   Vytrobca = 'matysak';
```

SQL obsahuje viacero agregáčnych funkcií:

SUM, MIN, MAX, AVG, STDEV, COUNT, ...
(sem patria tiež **EXISTS, IN, ANY, ALL**)

Všetky agregáčné funkcie sa aplikujú na jeden atribút, avšak pri COUNT na atribútoch nezáleží:

```
SELECT COUNT(*)
FROM    produkt
```

Ani pri EXISTS na atribútoch nezáleží, lebo EXISTS sa aplikuje na celú reláciu (EXISTS r vráti true ak r je neprázdna, inak false)

Príkaz SELECT: grupovanie a agregácia

produkt(Meno, Cena, Kategoria, Vyrobcu)

kontrakt(Predavajuci, Kupujuci, Sklad, Vyrobovok)

Priemerná cena kontraktovaných výrobkov po jednotlivých skladoch:

```
SELECT      kontrakt.Sklad, AVG(Cena) AS Priemer
FROM        produkt, kontrakt
WHERE       produkt.Meno = kontrakt.Vyrobovok
GROUP BY  kontrakt.Sklad
```

Za SELECT majú v tomto prípade zmysel len:

- grupovacie atribúty (atribúty v GROUP BY)
- výsledky agregáčnych funkcií

Sklad	Cena	...
bratislava	1	
bratislava	7	
detva	2	
detva	2	
detva	5	
...		



Sklad	Priemer
bratislava	4
detva	3
...	

Príkaz SELECT: HAVING

produkt(Meno, Cena, Kategoria, Vyrobcu)

kontrakt (Predavajuci, Kupujuci, Sklad, Vyrobovok)

Priemerná cena kontraktovaných výrobkov po jednotlivých skladoch, pre sklady s priemernou cenou vyššou než 3

```
SELECT      kontrakt.Sklad, AVG(Cena) AS Priemer
FROM        produkt, kontrakt
WHERE       produkt.Meno = kontrakt.Vyrobovok
GROUP BY    kontrakt.Sklad
HAVING     AVG(Cena) > 3
```

Za HAVING nasleduje **selekčná podmienka na reláciu, ktorá je výsledkom grupovania a agregácie** (teda nie na relácie za FROM)

Príkaz SELECT: odstránenie duplikátov vo výsledku

produkt(Meno, Cena, Kategória, Vytrobca)

SQL neodstraňuje duplikáty automaticky. To znamená, že ich zachováva jednotným, „prirodzeným“ spôsobom. Dôvodom je snaha o zvýšenie efektivity výpočtu (o čom sa dá polemizovať, lebo duplikátov môže byť veľa). Odstránenie duplikovaných záznamov relácie je rovnako zložité ako utriedenie relácie

Na odstránenie duplikátov slúži **DISTINCT**. Aplikuje sa buď na **výslednú reláciu** (podobne ako ORDER BY) **alebo v agregácii**

Všetci výrobcovia, pričom každý výrobca má byť vo výsledku práve raz (odstránenie duplikátov z výslednej relácie):

```
SELECT DISTINCT Vytrobca  
FROM produkt;
```


Príkaz SELECT: odstránenie duplikátov v agregácii

produkt(Meno, Cena, Kategoria, Vyrobcu)

Použitie **DISTINCT** v agregácii

Počet rôznych výrobcov v relácii produkt:

```
SELECT    count(DISTINCT Vyrobcu)
FROM      produkt;
```

Čo vyjadruje nasledujúci dotaz?

```
SELECT    Vyrobcu
FROM      produkt
GROUP BY  Vyrobcu;
```

Príkaz SELECT: zachovanie duplikátov

osoba (Meno, Rodne_cislo, Telefon, Mesto)

kontrakt (Predavajuci, Kupujuci, Sklad, Vyrobovok)

Operátory **UNION, INTERSECT a EXCEPT** pracujú **s množinami (bez duplikátov)**

Ale s použitím **ALL** sa dá vynútiť, aby zachovávali duplikáty:

```
(SELECT Meno  
FROM osoba  
WHERE Mesto = 'trnava')
```

UNION ALL

```
(SELECT Meno  
FROM osoba, kontrakt  
WHERE Kupujuci = Meno AND Sklad = 'zohor');
```

Príkaz SELECT: negácia

osoba (Meno, Telefon, Rodne_cislo, Mesto)

kontrakt (Predavajuci, Kupujuci, Sklad, Vyrobok)

$\text{answer}(M, T) \leftarrow \text{osoba}(M, T, _, _), \text{not nieco_kupil}(M).$

$\text{nieco_kupil}(M) \leftarrow \text{kontrakt}(_, M, _, _).$

Meno a telefón osôb, ktoré nič nekúpili

Negácia sa dá v SQL vyjadriť viacerými spôsobmi, napr.

NOT EXISTS, NOT IN, EXCEPT

Najvšeobecnejšie a kompilátoru najbližšie (vedie k efektívnemu plánu výpočtu) je NOT EXISTS

```
SELECT Meno, Telefon
FROM osoba
WHERE NOT EXISTS (
  SELECT *
  FROM kontrakt
  WHERE Meno = Kupujuci);
```

Príkaz SELECT: definícia relácií

osoba (Meno, Telefon, Rodne_cislo, Mesto)

kontrakt (Predavajuci, Kupujuci, Sklad, Vyrobok)

$\text{answer}(M, T) \leftarrow \text{osoba}(M, T, _, _), \text{not nieco_kupil}(M).$

$\text{nieco_kupil}(M) \leftarrow \text{kontrakt}(_, M, _, _).$

Meno a telefón osôb, ktoré nič nekúpili

Na definíciu relácií (pomenovanie medzivýsledkov v dotaze)
slúžia **WITH** a **CREATE TEMPORARY TABLE**

CREATE TEMPORARY TABLE nieco_kupil AS

SELECT Kupujuci

FROM kontrakt;

SELECT Meno, Telefon

FROM osoba

WHERE NOT EXISTS

(SELECT * FROM nieco_kupil WHERE Meno = Kupujuci);

Príkaz SELECT: definícia relácií

osoba (Meno, Rodne_cislo, Telefon, Mesto)

kontrakt (Predavajuci, Kupujuci, Sklad, Vyrobok)

$\text{answer}(M, T) \leftarrow \text{osoba}(M, T, _, _), \text{not nieco_kupil}(M).$

$\text{nieco_kupil}(M) \leftarrow \text{kontrakt}(_, M, _, _).$

Meno a telefón osôb, ktoré nič nekúpili

WITH je „čistejšie“ (priamo sa viaže k nasledujúcemu SELECT), ale nie všetky systémy implementujú WITH

WITH nieco_kupil **AS**

(SELECT Kupujuci
FROM kontrakt),

SELECT Meno, Telefon

FROM osoba

WHERE NOT EXISTS

(SELECT * FROM nieco_kupil WHERE Meno = Kupujuci);

Príkaz SELECT: rekurzia (WITH RECURSIVE)

r(Rodic, Dieta)

$\text{anc}(\text{Predok}, \text{Potomok}) \leftarrow \text{r}(\text{Predok}, \text{Potomok}).$

$\text{anc}(\text{Predok}, \text{Potomok}) \leftarrow \text{r}(\text{Rodic}, \text{Potomok}), \text{anc}(\text{Predok}, \text{Rodic}).$

? $\text{anc}(\text{Predok}, \text{Potomok}).$

Predkovia osoby

WITH RECURSIVE anc **AS**

((SELECT Rodic AS Predok, Dieta AS Potomok
FROM r)

UNION

(SELECT Predok, Dieta AS Potomok
FROM r, **anc**

WHERE Rodic = Potomok)),

SELECT Predok, Potomok
FROM anc;

Algoritmus prekladu jednoduchého Datalogového pravidla bez negovaných podcieľov do SQL:

1. V klauze **FROM** vymenuj všetky mená relácií (všetky inštancie), ktoré sú v tele pravidla. Každé relácii prirad' jednoznačné meno (alias)
2. V klauze **SELECT** vymenuj všetky atribúty, ktoré sú v hlave pravidla (konvertuj pozičnú adresáciu na mená atribútov)
3. V klauze **WHERE** vymenuj v konjunkcii väzby všetkých atribútov (selekčné podmienky)

Algoritmus prekladu ľubovoľného bezpečného Datalogového programu do SQL:

Pre každý predikát p (ktorý nie je v EDB) generuj

WITH p AS ...

/* alebo WITH RECURSIVE p AS

resp. CREATE TEMPORARY TABLE p AS */

Pre každé pravidlo definujúce p generuj **(SELECT ...)**

Ak je pravidiel definujúcich p viacej, spoj selecty generované v predošlom kroku pomocou **UNION**

Pre všetky nenegované podciele v tele pravidla použi algoritmus prekladu jednoduchých pravidiel

Pre každý negovaný podcieľ tvaru not s(...) pridaj do

WHERE klauzy

... AND NOT EXISTS (SELECT * FROM s WHERE ...)

a vymenuj vo vnútornom WHERE väzby atribútov s(...)

Príkaz INSERT

osoba (Meno, Rodne_cislo, Telefon, Mesto)

INSERT pridá nový záznam (riadok) resp. záznamy do relácie

INSERT INTO osoba **VALUES**

(‘jozef’, ‘111111/1111’, ‘02-123456’, ‘bratislava’),
(‘eva’, ‘101510/1111’, ‘045-654321’, ‘zvolen’);

Príkaz UPDATE

osoba (Meno, Rodne_cislo, Telefon, Mesto)

UPDATE zmení existujúci záznam (riadok) resp. záznamy v relácii

UPDATE osoba **SET** Telefon = '02-31777' **WHERE** Meno = 'jozef';

Vo WHERE môže byť čokoľvek, čo môže byť vo WHERE klauze pri SELECT

Príkaz DELETE

osoba (Meno, Rodne_cislo, Telefon, Mesto)

DELETE odstráni záznam (riadok) resp. záznamy z relácie

DELETE FROM osoba **WHERE** Meno = 'jozef';

Vo WHERE môže byť čokoľvek, čo môže byť vo WHERE klauze pri SELECT

SQL DDL: definícia databázovej schémy

Najskôr:

Definovať typy dát

Potom:

Definovať schému databázy (množinu tabuliek s atribútmi)

- Vytvorenie tabuliek
- Odstránenie tabuliek
- Modifikácia atribútov tabuliek

Nakoniec:

- Definovať čo najsilnejšie obmedzenia v záujme udržania konzistencie databázy
- Vytvoriť indexy pre urýchlenie behu dotazov

SQL DDL: typy dát (domains)

- Reťazce (pevnej alebo premennej dĺžky) **CHAR, VARCHAR**
- Celé čísla **INTEGER, SHORTINT**
- Čísla s pohyblivou desatinnou čiarkou **REAL, DOUBLE**
- Dátum a čas **DATE/TIME**
- ...

Domény sa používajú pri definícii tabuliek (každý atribút je z nejakej domény)

Definícia domény (alias pre typ):

```
CREATE DOMAIN d_adresa AS VARCHAR(55);
```

SQL DDL: vytvorenie relácie

```
CREATE TABLE osoba(  
    Meno                VARCHAR(30),  
    Rodne_cislo         INTEGER,  
    Vek                 SHORTINT,  
    Mesto               VARCHAR(30),  
    Pohlavie            BIT(1),  
    Datum_narodenia     DATE  
);
```

SQL DDL: odstránenie relácie

```
DROP osoba;
```

SQL DDL: modifikácia relácie

```
ALTER TABLE osoba  
    ADD Telefon CHAR(16);
```

```
ALTER TABLE osoba  
    DROP Vek;
```


Indexy slúžia na **urýchlenie výpočtu dotazov**

Uvažujme reláciu:

osoba (Meno, Rodne_cislo, Telefon, Mesto)

Index na Rodne_cislo umožňuje **oveľa rýchlejší** prístup k riadku s daným rodným číslom ako pri sekvenčnom prehľadávaní tabuľky (v indexe je uložené utriedenie riadkov tabuľky podľa daného atribútu, ako vo vyhľadávacom strome)

```
CREATE INDEX index_rc ON osoba(Rodne_cislo)
```

Môžeme vytvárať indexy aj pre viacero atribútov:

```
CREATE INDEX index_meno_rc ON  
osoba (Meno, Rodne_cislo)
```

Problém vhodnej voľby indexov súvisí s návrhom a prevádzkovaním bázy dát. **Čím viacej indexov, tým pomalšia aktualizácia**, lebo spolu s dátami sa aktualizujú indexy. Navyše, **indexy zaberajú diskový priestor**

Treba nájsť vhodný **tradeoff**:

- Časté, typické dotazy treba urýchliť
- Zároveň netreba spomaliť typické aktualizácie

Views (pohľady) sú dotazy permanentne zapamätané v databáze

VIEWS sa používajú sa ako bežné relácie (v klauze FROM), ale:

- Možno na ne vytvoriť **prístupové práva**
- Možno na ne vyrobiť **indexy**
- Umožňujú, aby rôzni užívatelia videli rôzne časti databázy
- Umožňujú aktualizáciu databázy, ktorá prebieha cez viacero relácií súčasne

VIEWS sú tiež užitočné pri štruktúrovaní zložitých dotazov

osoba (Meno, Rodne_cislo, Telefon, Mesto)
kontrakt (Predavajuci, Kupujuci, Sklad, Vyrobok)

```
CREATE VIEW      presovske_kontrakty AS  
    SELECT      Kupujuci, Predavajuci, Vyrobok, Sklad  
    FROM        osoba, kontrakt  
    WHERE       osoba.Mesto = 'presov' AND  
               osoba.Meno = kontrakt.Kupujuci
```

Neskoršie použitie VIEW:

```
SELECT  Meno, Sklad  
FROM    presovske_kontrakty, produkt  
WHERE   presovske_kontrakty.Vyrobok = produkt.Meno  
AND     produkt.Kategoria = 'textil';
```

SQL DDL: odstránenie VIEW

```
DROP VIEW presovske_kontrakty;
```

kontrakt (Predavajuci, Kupujuci, Sklad, Vyrobovok)

Aktualizácia databázy cez views je užitočná, ale treba s ňou narábať opatrne. Môže sa napríklad stať, že aktualizácia prebieha cez atribúty, ktoré view neobsahuje:

```
CREATE VIEW kontrakty_tesco AS
  SELECT  Sklad, Predavajuci, Kupujuci
  FROM    kontrakt
  WHERE   Sklad = 'tesco';
```

```
INSERT INTO kontrakty_tesco
  VALUES ('tesco', 'jozef', 'dezider');
```

vloží do relácie kontrakt riadok
(‘jozef’, ‘dezider’, ‘tesco’, **null**)

SQL DDL: aktualizácia cez VIEW

osoba (Meno, Rodne_cislo, Telefon, Mesto)
kontrakt (Predavajuci, Kupujuci, Sklad, Vyrobok)

```
CREATE VIEW poprad AS
    SELECT Predavajuci, Vyrobok, Sklad
    FROM osoba, kontrakt
    WHERE osoba.Mesto = 'poprad' AND
           osoba.Meno = kontrakt.Kupujuci;
```

Čo spôsobí nasledujúci INSERT? (Vyskúšajte!)

```
INSERT INTO poprad
    VALUES ('jozef', 'sandale', 'billa');
```

Niečo to pridá do relácií osoba a kontrakt, ale výsledok je konstraintívny