

# SQL.

Základy SQL

# Obsah prednášky

- Organizácia
- SQL
  - DML
  - DDL

# Organizácia

- Pridané príklady na SQL
- Link na zaevidovanie url Vášho servera
  - <https://forms.gle/trLwc6hUjuwmBVWaA>
  - V rámci termínu cvičenia prosím napíšte aj meno cvičiaceho

# Organizácia

- Bodové hodnotenie
  - 1b rozbehanie servera DB
  - 1b rozbehanie WebApp v rámci servera
  - 1b Volanie SQL do DB
  - 1b Serializácia výsledku (vytvorenie JSON)
  - **1b kvalita kódu**

# Zadanie

# SQL

- Originál vyvinutý firmou IBM s názvom **Sequel** pre **System R** v 1970
  - Sequel - Structured English Query Language
- ANSI štandard v roku 1986, ISO v roku 1987
  - Structured Query Language
- Hovorí, čo chceme získať a nie ako to chceme získať
  - Neprocedurálny jazyk
- Založený na relačnom kalkule a relačnej algebre

# SQL - história

- SQL:2019: Multi-dimensional arrays (SQL/MDA)
- SQL:2016 – JSON, Polymorphic table
- SQL:2011 – Dočasné DB, Pipelined DML
- SQL:2008 – Truncation,
- SQL:2003 – XML, Windows, Sekvencie, Auto-Gen ID
- SQL:1999 - Regex, Triggers
- <https://modern-sql.com/standard>

# Časti jazyka SQL

- **Data Definition Language (DDL)**
- **Data Manipulation Language (DML)**
- **Data Control Language (DCL)**



# Data Definition Language (DDL)


- Manažovanie databázových objektov
  - Vytváranie, modifikovanie, mazanie tabuliek/používateľov
- Napr.
  - CREATE/DROP table/database
  - ALTER table
  - TRUNCATE

# Data Manipulation Language (DML)

- Manipulácia s dátami
  - Pridávanie, modifikovanie, mazanie dát v DB
- SELECT
- INSERT
- DELETE
- UPDATE

# Data Control Language (DCL)

- Pre riadenie prístupu k údajom
- Napr. GRANT, REVOKE



Tomúto sa nebudeme  
primárne venovať

# Data Manipulation language

# Dopyt

- Angl. **Query**
- Presná žiadosť vyhľadania informácií v rámci databázy/informačného systému

# DML - základný dopyt

- Výpočet výsledku základného dopytu
  1. Vykonanie karteziánskeho súčinu z tabuliek uvedených v rámci klauzuly **FROM**
  2. Na karteziánsky súčin sa aplikujú podmienky v rámci klauzuly **WHERE**
  3. Výber stĺpcov v rámci klauzuly **SELECT**

```
SELECT stĺpce tabuľky  
FROM tabuľky  
WHERE podmienka;
```

# WHERE

- V rámci podmienky 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
- Príklad built-in funkcií v rámci PostgreSQL
  - [https://www.tutorialspoint.com/postgresql/postgresql\\_useful\\_functions.htm](https://www.tutorialspoint.com/postgresql/postgresql_useful_functions.htm)
  - <https://www.postgresql.org/docs/13/functions.html>

# SELECT - ďalšie možnosti

- **GROUP BY**
- **HAVING**
- **ORDER BY**
- **LIMIT**
- ...



# Usporiadanie výsledkov

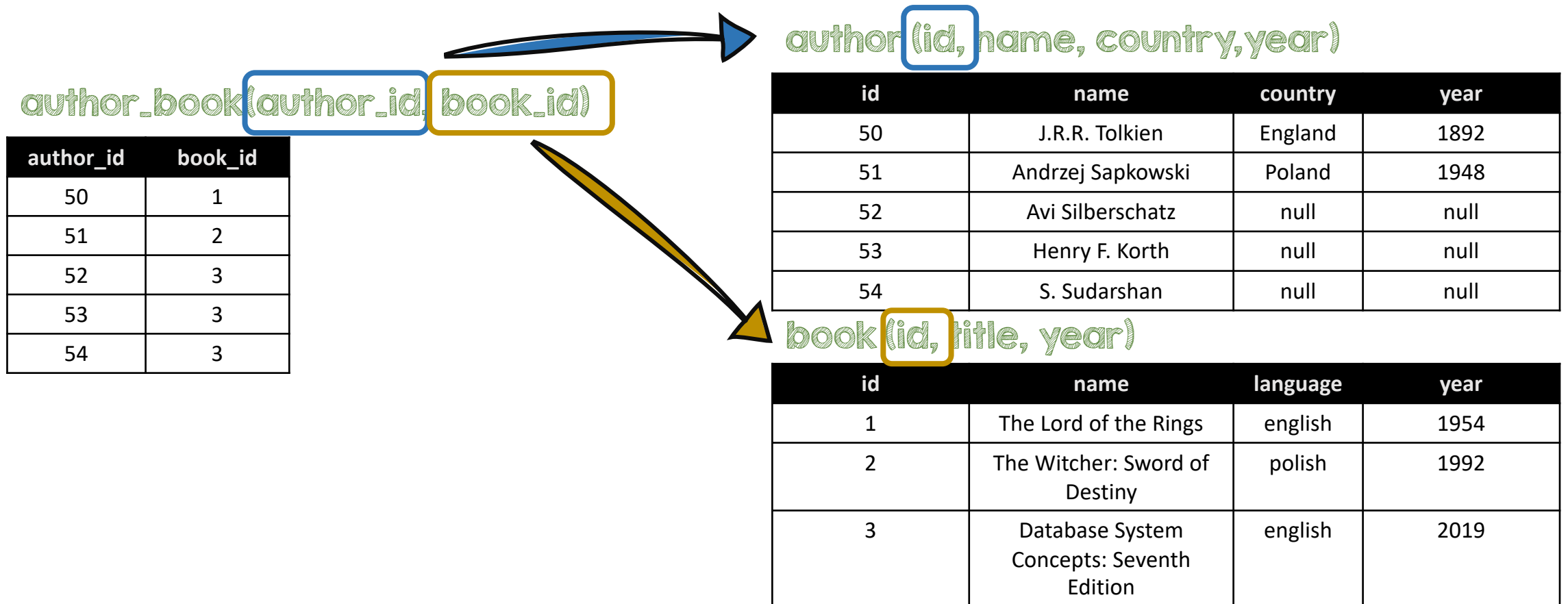
- Možnosti usporiadania
  - **ASC** – vzostupne
  - **DESC** – zostupné
- Priorita zoradenia - zľava doprava
  - Možnosti usporiadania sa môžu kombinovať

```
SELECT stĺpce tabuľky  
FROM tabuľky  
WHERE podmienka  
ORDER BY atribúty
```

# Množinové operácie

- Prienik
  - **INTERSECT**
- Zjednotenie
  - **UNION**
- Rozdiel
  - **EXCEPT**
- Odstraňujú duplikáty
  - Pre zachovanie duplikátov je potrebné použiť UNION ALL

# Opakovanie - cudzie klúče



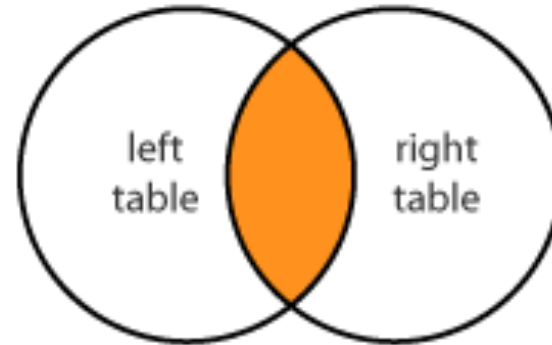
# Spájanie tabuliek

- Možnosť opätovného spojenia tabuliek
- **CROSS JOIN, NATURAL JOIN**
- Poznáme nasledujúce spojenia
  - **INNER JOIN**
  - **LEFT OUTER JOIN**
  - **RIGHT OUTER JOIN**
  - **FULL OUTER JOIN**

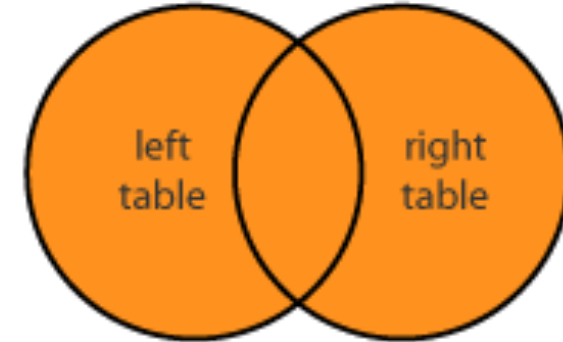
# Spájanie tabuliek - JOIN

- INNER JOIN = JOIN
- LEFT OUTER JOIN = LEFT JOIN
- RIGHT OUTER JOIN = RIGHT JOIN
- FULL OUTER JOIN = FULL JOIN

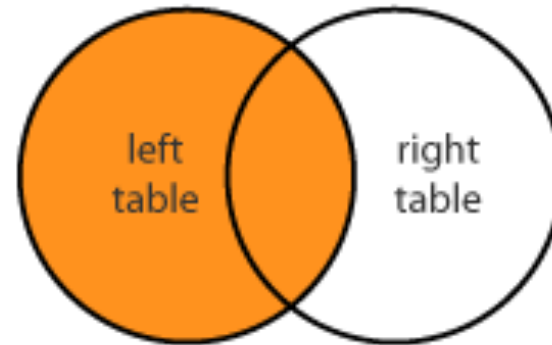
INNER JOIN



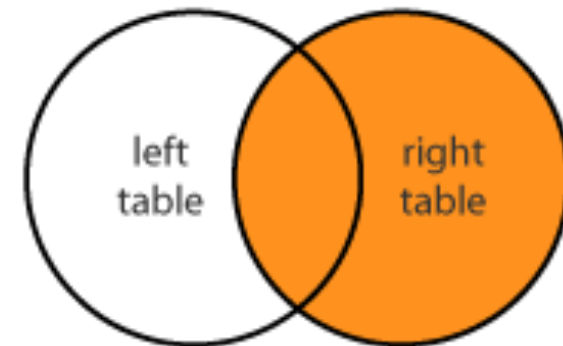
FULL JOIN



LEFT JOIN



RIGHT JOIN



Zdroj: <https://www.dofactory.com/sql/join>

# INNER JOIN

- najčastejšie používaný **JOIN**
- spojenie vznikne kombináciou záznamov na základe podmienky spojenia
  - väčšinou je táto podmienka na základe **rovnosti cudzieho a primárneho kľúča**
- **ON** vs **WHERE**
  - **ON** sa vyhodnocuje pred samotným spojením tabuliek a definuje podmienku spájania
  - **WHERE** definuje podmienku filtrovania a vyhodnocuje sa po vytvorení výsledku spojenia.

```
SELECT stĺpce tabuľky  
FROM tabuľka1 INNER JOIN tabuľka2  
ON podmienka pre spojenie
```

# INNER JOIN - príklad

- Príklad

- všetky id zákazníkov, ktorý majú objednávku väčšiu ako 100

**osoba** ( id, meno, priezvisko,....)  
**objednavka** ( id, cena, osoba\_id, .....

```
SELECT id  
FROM osoba  
INNER JOIN objednavka  
ON id = osoba_id WHERE cena > 100
```

# INNER JOIN - príklad

- Príklad

- všetky id zákazníkov, ktorý majú objednávku väčšiu ako 100

**osoba** ( id, meno, priezvisko,....)  
**objednavka** ( id, cena, osoba\_id, .....

```
SELECT id
FROM osoba
INNER JOIN objednavka
ON id = osoba_id WHERE cena > 100
```

Aký je tu problém ?



# INNER JOIN - príklad

- Príklad

- všetky id zákazníkov, ktorý majú objednávku väčšiu ako 100

**osoba** ( id, meno, priezvisko,....)  
**objednavka** ( id, cena, osoba\_id, .....

```
SELECT id
FROM osoba
INNER JOIN objednavka
ON id = osoba_id WHERE cena > 100
```

Aké ID mám na mysli ?

# INNER JOIN - príklad

- Príklad

- všetky id zákazníkov, ktorý majú objednávku väčšiu ako 100

**osoba** ( id, meno, priezvisko,....)  
**objednavka** ( id, cena, osoba\_id, .....

```
SELECT id
FROM osoba
INNER JOIN objednavka
ON id = osoba_id WHERE cena > 100
```

Nejednoznačnosť  
atribútov

# Jednoznačnosť atribútov

- potreba zabezpečiť jednoznačnosť atribútov
- Pre skrátenie zápisu a zlepšenie prehľadnosti je možné použiť **ALIAS**

```
SELECT osoba.id  
FROM osoba INNER JOIN objednavka  
ON osoba.id = objednavka.osoba_id  
WHERE objednavka.cena > 100
```

# Jednoznačnosť atribútov

- potreba zabezpečiť jednoznačnosť atribútov
- Pre skrátenie zápisu a zlepšenie prehľadnosti je možné použiť **ALIAS**

Teraz už správne

```
SELECT osoba.id  
FROM osoba INNER JOIN objednavka  
ON osoba.id = objednavka.osoba_id  
WHERE objednavka.cena > 100
```

# Aliasy

- používajú sa pre zlepšenie čitateľnosti
- V prípade, že by ste používali tu **istú tabuľku dvakrát** pre vytvorenie dvojíc, tak je potrebné použiť aliasy pre zabezpečenie jednoznačnosti

```
SELECT os.id AS identifikator  
FROM osoba AS os  
INNER JOIN objednavka AS ob  
ON os.id = ob.osoba_id  
WHERE ob.cena > 100
```

# Aliasy

- používajú sa pre zlepšenie čitateľnosti
- V prípade, že by ste používali tu **istú tabuľku dvakrát** pre vytvorenie dvojíc, tak je potrebné použiť aliasy pre zabezpečenie jednoznačnosti

id osoby sa nám vo výstupe  
zobrazí ako identifikátor

```
SELECT os.id AS identifikator  
FROM osoba AS os  
INNER JOIN objednavka AS ob  
ON os.id = ob.osoba_id  
WHERE ob.cena > 100
```

# Aliasy

- používajú sa pre zlepšenie čitateľnosti
- V prípade, že by ste používali tu **istú tabuľku dvakrát** pre vytvorenie dvojíc, tak je potrebné použiť aliasy pre zabezpečenie jednoznačnosti

```
SELECT os.id identifikator  
FROM osoba os  
INNER JOIN objednavka ob  
ON os.id = ob.osoba_id  
WHERE ob.cena > 100
```

# Aliasy

- používajú sa pre zlepšenie čitateľnosti
- V prípade, že by ste používali tu **istú tabuľku dvakrát** pre vytvorenie dvojíc, tak je potrebné použiť aliasy pre zabezpečenie jednoznačnosti

Možnosť vynechania **AS**

```
SELECT os.id identifikator  
FROM osoba os  
INNER JOIN objednavka ob  
ON os.id = ob.osoba_id  
WHERE ob.cena > 100
```



# Spájanie tabuliek - INNER vs OUTER

- **INNER JOIN**

- len tie záznamy, ktoré sú v oboch reláciách tabuľka1 a tabuľka2

- **OUTER JOIN**

- **Left, Right, Full**
  - všetky záznamy z jednej tabuľky (záleží od použitého JOINu) a ak je daný záznam aj v druhej tabuľke, tak sa doplní preňho informácia. Inak sa dáva hodnota **NULL**

# Vlastnosti spájania tabuliek

- V praxi môže poradie spojenia ovplyvniť rýchlosť vyhodnotenia
- **RIGHT** a **LEFT JOIN** **nie sú** komutatívne:  
 $(a \text{ LEFT JOIN } b) \neq (b \text{ LEFT JOIN } a)$
- typy spojenia **OUTER JOIN** **nie sú** asociatívne
  - $((a \text{ LEFT JOIN } b) \text{ LEFT JOIN } c) \neq (a \text{ LEFT JOIN } (b \text{ LEFT JOIN } c))$

# Hodnota Null

- NULL hodnota
  - **3-stupňova logika** pre vyhodnocovanie výrazov

Hodnota	Negácia hodnoty
True	False
False	True
null	null

Operácia AND	True	False	Null
null	Null	False	Null

Operácia OR	True	False	Null
null	True	Null	Null

- V prípade podmienky **null = null**, tiež dochádza k vráteniu hodnoty **null**
- Ak chceme zistiť, či je hodnota null je potrebné použiť **IS**
  - príklad: **null IS null** vráti hodnotu **true**

# Hodnota Null (2)

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

Ak **A** je **null** tak **A<5** sa vyhodnotí ako **null**

# Distinct

- Odstránenie duplikátov SQL
  - umožňuje duplikáty na rozdiel od relačnej algebry
- Možnosť aplikovania pri výsledku alebo priamo v agregácií

```
SELECT DISTINCT stĺpce tabuľky  
FROM tabuľky  
WHERE podmienka
```

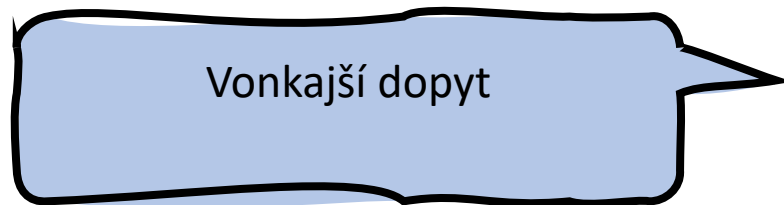
# Vnorené dopyty

- Dopyt, ktorý obsahuje ďalší dopyt
- Z pohľadu DBMS náročné optimalizovať
  - nie vždy je ich nutné použiť, možnosť využitia JOINu (častokrát efektívnejšie)
  - Veľké množstvo vnorených dopytov znižuje čitateľnosť
- Vonkajší dopyt vs vnútorný dopyt

```
SELECT stĺpce tabuľky  
FROM tabuľky  
WHERE atribút = (  
    SELECT stĺpec tabuľky  
    FROM tabuľky  
    WHERE podmienka  
)
```

# Vnorené dopyty

- Dopyt, ktorý obsahuje ďalší dopyt
- Z pohľadu DBMS náročné optimalizovať
  - nie vždy je ich nutné použiť, možnosť využitia JOINu (častokrát efektívnejšie)
  - Veľké množstvo vnorených dopytov znižuje čitateľnosť
- Vonkajší dopyt vs vnútorný dopyt

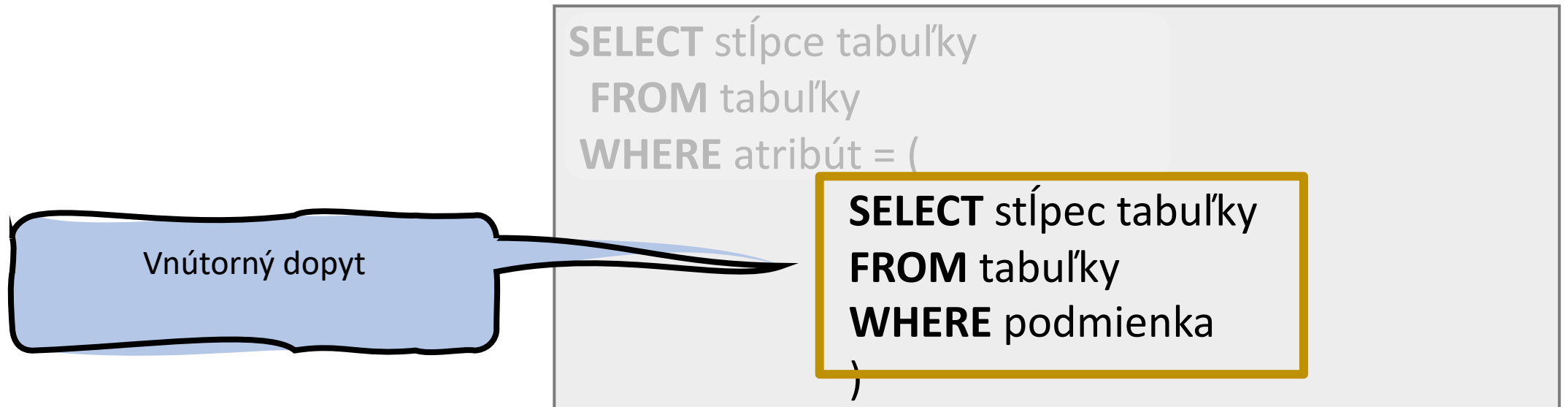


```
SELECT stĺpce tabuľky  
FROM tabuľky  
WHERE atribút = (
```

```
SELECT stĺpec tabuľky  
FROM tabuľky  
WHERE podmienka  
)
```

# Vnorené dopyty

- Dopyt, ktorý obsahuje ďalší dopyt
- Z pohľadu DBMS náročné optimalizovať
  - nie vždy je ich nutné použiť, možnosť využitia JOINu (častokrát efektívnejšie)
  - Veľké množstvo vnorených dopytov znižuje čitateľnosť
- Vonkajší dopyt vs vnútorný dopyt





# ALL, ANY, IN, EXISTS

- **ALL**

- Vráti **TRUE** ak všetky hodnoty v rámci subquery spĺnia podmienku

- **ANY**

- Vráti **TRUE** ak niektorá hodnota rámci subquery spĺnila podmienku

- **IN**

- Definovanie viacerých hodnôt v rámci **WHERE** podmienky a skrátenie **OR** podmienky

- **EXISTS**

- v prípade, že existuje v rámci Subquery nejaký záznam vráti **TRUE**

# Agregácie

- Vracajú jednu hodnotu pre zoskupenie riadkov
- Príklad agregáčnych funkcií
  - Priemer – `avg(stĺpec)` – priemerná hodnota z vybraného stĺpca
  - Minimum – `min(stĺpec)` – vráti minimálnu hodnotu zo stĺpca
  - Maximum – `max(stĺpec)` – vráti maximálnu hodnotu zo stĺpca
  - Total: `Sum(stĺpec)` – vráti sumu
  - Počet: `count` – vráti počet hodnôt

# Agregácie - COUNT

- COUNT (\*)
- COUNT (I)
- COUNT (atribút)
- COUNT ( DISTINCT atribút )

# Agregácie - GROUP BY

- Možnosť vytvorenia zoskupenia, pre ktorý sa ma vypočítať

```
SELECT stĺpce tabuľky  
FROM tabuľky  
WHERE atribút  
GROUP BY stĺpce tabuľky
```

# Agregácie - HAVING

- Za HAVING nasleduje selekčná podmienka na reláciu, ktorá je výsledkom zoskupenia a vyrátanej agregácie
  - neviaže sa táto podmienka na relácie za FROM

```
SELECT stĺpce tabuľky  
FROM tabuľky  
WHERE atribút  
GROUP BY stĺpce tabuľky  
HAVING podmienka
```

# String operácie

- Rôzne operácie pre prácu s textom
  - UPPER
  - CONCAT
  - ...

# LIKE

- slúži na vyhľadávanie prostredníctvom vzoru
  - % - 0 až N ľubovoľných znakov
  - \_ - 1 ľubovoľný znak

```
SELECT meno, priezvisko  
FROM osoba  
WHERE priezvisko LIKE '%ova' AND meno LIKE '_ana'
```

nájdenie všetkých osôb, ktorých priezvisko končí na *ova* a meno môže byť Jana, Hana prípadne nejaké iné, ktoré sa líši v prvom znaku.

# BETWEEN

- slúži na definovanie rozsahu hodnôt od-do (vrátane) alebo časových období (dátumov)
- Príklad:  
SELECT priezvisko, meno FROM osoba WHERE vek **BETWEEN 20 AND 30**;

```
SELECT priezvisko, meno  
FROM osoba  
WHERE vek BETWEEN 20 AND 30
```



# DML - INSERT

- INSERT pridá nový záznam (riadok) alebo záznamy do relácie/tabuľky

**INSERT INTO**

tabuľka

**VALUES**

(hodnota1, hodnota2, ..., hodnotaN);

**INSERT INTO**

tabuľka (názov\_atribútu1, názov\_atribútu2, .... , názov\_atribútuN)

**VALUES**

(hodnota1, hodnota2, .... , hodnotaN),

(hodnotax1, hodnotax2, .... , hodnotaxN);

# DML - INSERT

- INSERT pridá nový záznam (riadok) alebo záznamy do relácie/tabuľky

**INSERT INTO**

tabuľka

**VALUES**

(hodnota1, hodnota2, ..., hodnotaN);

Záleží na poradí

**INSERT INTO**

tabuľka (názov\_atribútu1, názov\_atribútu2, .... , názov\_atribútuN)

**VALUES**

(hodnota1, hodnota2, .... , hodnotaN),

(hodnotax1, hodnotax2, .... , hodnotaxN);

Nezáleží na poradí

# DML - UPDATE

- UPDATE zmení existujúci záznam (riadok) alebo záznamy v relácií
- počet modifikovaný hodnôt je voliteľný a tiež použitie podmienky
- v podmienke WHERE sa môže nachádzať čokoľvek, čo môže byť vo WHERE v rámci príkazu SELECT

**UPDATE** tabuľka

**SET** atr1=nová\_hodnota1, ..., atrN=nová\_hodnotaN

**WHERE** podmienka;

# DML - DELETE

- príkaz DELETE odstráni záznam (riadok) alebo záznamy v rámci relácie
- v podmienke WHERE sa môže nachádzať čokoľvek, čo môže byť vo WHERE v rámci príkazu SELECT

**DELETE FROM** tabuľka  
**WHERE** podmienka

# Data Definition Language

Definovanie databázovej schémy

# DDL - typy dát

- Môžu byť:
  - reťazce (pevné alebo premenlivej dĺžky) - CHAR, VARCHAR
  - celé čísla – INTEGER, SHORTINT
  - čísla s pohyblivou desatinnou čiarkou – REAL, DOUBLE
  - dátum a čas – DATA/TIME
  - ....

# DDL - CREATE TABLE

- Vytvorenie relácie/tabuľky
- **Obmedzenia (constraint)** - hodnoty pri vkladaní, modifikácií alebo mazaní majú ohraničenie, ktoré musia spĺňať
  - môžu byť napr. povinnosť uvedenia hodnoty, unikátna hodnota, primárny a cudzí kľúč .....

```
CREATE TABLE meno_tabuľky(  
    názov1 typ1 [obmedzenia];  
    názov2 typ2 [obmedzenia];  
    ...  
    názovN typN [obmedzenia];  
);
```

# DDL - CREATE TABLE príklad

```
CREATE TABLE osoba (  
    meno          VARCHAR(20),  
    rodne_cislo   INTEGER,  
    vek           SHORTINT  
);
```



# DDL - DROP TABLE

- odstráni tabuľku

```
DROP TABLE názov_tabuľky;
```

# DDL - modifikácia relácie

- Pridanie/vymazanie/modifikovanie atribútu/stĺpca

**ALTER TABLE** tabuľka

**ADD/DROP/MODIFY** atribút typ(v prípade ADD/MODIFY)

# Príklady na SQL

- <https://pgexercises.com>
  - postgresql
- <http://hackerrank.com>
  - MySQL, Oracle, DB2, MS SQL Server

# Zhrnutie

- DDL, DML
- SQL ponúka veľké možnosti ako realizovať jednotlivé dopyty

# Pokračovanie

- Referenčná integrita
- Ďalšie SQL