## Architektúra

## SQL

## struktúrált adatlekérő nyelv; különböző szabványai léteznek

**Server oldal**

## DBMS -> database management system (adatbázis kezelő rendszer)

## RDBMS – relational database management system

**MariaDB**

több, külön kezelhető, szeparált adatbázis séma (adatbázis) elhelyezésére

A MariaDB majdhogynem = a MySQL-el, jogi csatározások.

**ACID elvek** - > mit követelünk meg egy DBMS-től?

## *Atomicitás*

## csak akkor tekintjük sikeresnek a tranzakciót, ha minden részművelet is végbement

## *Konzisztencia*

## egy kimaradás vagy szoftveres hiba miatti félbe maradt adatbeszúrás okán nem kerülhet sérült vagy valamilyen rossz adat a rendszerbe; az adatbázis nem válthat olvashatatlanná

## *Izoláció*

## tranzakciók izolációja: az egy időben zajló tranzakciók ugyanolyan állapothoz vezessenek, amit egyébként a sorban végrehajtott tranzakciók érnének el

## *Durability*

## Tartósan kell tárolni a tranzakciók és a változások eredményét, nem csak RAM-ban, hanem a sikeres tranzakciókat le kell tárolni máshol is, ahol hosszútávon is elérhetőek lesznek

**Kliens oldal**

**HeidiSQL**

grafikus kliens, ami az adatbázissal kommunikál

üzeneteket/kéréseket küld SQL nyelven a DBMS-hez és válaszokat fogad

A server és kliens oldali programot is telepíteni kell; nem feltétlen kell egy helyen legyenek. Localhost esetén az adatbázisok háttérben fileban is tárolva vannak a Windows valamilyen mappájában.

## SQL nyelv részei

## DML - Adat manipuláló

## DDL - Adat definiáló (adatokat létrehozó, adatstruktúrákat meghatározó; != manipulációval)

## DCL – jogokat biztosítja az adatokhoz (Revoke = elvesz)

## TCL – tranzakció kezelést segíti; amikor összefűzünk tranzakciókat egy nagy csomagba és azokat egyszerre hajtjuk végre (pl. lekérünk, módosítunk, frissítünk töröljük egyszerre – összevárjuk ezt a sok utasítást és egy lépésben hajtjuk végre -> ha valamiért nem sikerült, akkor a művelet visszavonható (pl. ha nem sikerült a beszúrás, nem hajtom végre a törlést)

## 

## A különböző kapcsolatok típusai

## Az adatbázisokban az adatokat táblákban vagy táblázatokban tároljuk

## a táblák általában nem önállóan léteznek, vannak közöttük kapcsolatok

## fejléc, oszlopok mezők vagy cellák, benne az adatok; sorok = rekordok

## egy az egyhez kapcsolat

## egy a többhöz kapcsolat

## több a többhöz kapcsolat

## Jelölések

## dupla - - kötelező, hogy a kapcsolt táblának legyen ebből

## kör – nem kötelező, hogy a kapcsolt táblának legyen ebből

## háromláb - a kapcsolt táblának lehet több ebből

## kör + háromláb

## kör + vonal – nem kötelező, de ha van, akkor csak egy lehet?

## Adattípusok, egyéb

## INT, BIGINT, CHAR, VARCHAR, FLOAT, BOOLEAN, BLOB(byteok), TIMESTAMP, DATE

## külön lekérdezés lapok

## F9 - futtatás

## CTRL + F9 – kijelölt futtatása

## F5 – Frissítés!

## CTRL+ - oszlop kihúzás

## foglalt nevek miatt inkább `` használata

## neveknél kis betű és alulvonás használata

## ; használata a parancs végén; értékeknél ’’

## 4) ””soha!

## PARANCSOK, LEKÉRDEZÉSEK

## AUTO\_INCREMENT

## fontossága túlértékelt, régi cucc

## ma már nem egy számot pörgetnek, hanem egy globális azonosítót - > nagy projekten térben és időben szét van pakolva, kifejezetten nem jó, nem tartalmaz elég infót (UUID unique identifier, egyedi szekvencia generátorokkal csinálják)

## törlésnél megszakad a számozás; visszaállítás ALTER TABLE-el

## PRIMARY KEY

## egyedileg azonosít minden sort ezen oszlopadat alapján

## FOREIGN KEY – csak olyan lehet, ami a saját táblájában primary key?

## táblák között kell kialakítani olyan vizsgálható relációt, amit a beszúráskor már lehet vizsgálni, hogy valid-e, beszúrható-e

## > kulcsok lehetnek összetettek is, primarynél van jelentősége, idegen összetett kulcsokra nem jellemző, hogy szükség volna

## JOIN

## egyik táblához hozzáválogatja a másik tábla sorait az adott illesztés oszlopadatai mentén

## *fajtái:* INNER JOIN, LEFT JOIN, RIGHT JOIN (és FULL OUTER JOIN)

## LEFT – összes első táblás sor benne lesz, és hozzá ha van, a második táblás adatok még

## INNER JOIN a RIGHT és a LEFT metszete; nem lesz hiányzó adatos sor

## Beágyazott lekérdezések

## egy query közepén egy beágyazott query

## általánosságban a WHERE feltételben használt subquery

## a reláció jobb oldalán lévő kifejezést zárójelbe!

## Ami az értéket adja nekem az összehasonlításhoz, azt a lekérdezést ágyazom be

## UNION - Táblák egyesítése

## kettő vagy több selectet kombinál össze

## *megkötések:*

## selecteknek azonos számú oszlopot és azonos típusú adatot kell tartalmazniuk, és a sorrendnek is egyeznie kell

## Kapcsoló tábla

## a több többhöz kapcsolatot kiváltjuk egy kapcsolótáblával

## ncd id / nc id / c id

## Constraintek

## minden, amit az oszlopokhoz írunk, a típusdefinicíó után, az kényszer (pl NOT NULL? UNIQUE, AUTO\_INCREMENT)

## PARANCSOK, LEKÉRDEZÉSEK

*Létrehozás*

## CREATE DATABASE `hambi`;

## USE CUSTOMERS;

## CREATE TABLE `products` (

## `id` int,

## `name varchar` (255),

## `description` varchar(255),

## `price` int

## );

*SELECT*

## SELECT \* FROM table\_name;

## SELECT name, city FROM customers;

## SELECT DISTINCT country FROM customers;

## SELECT DISTINCT country AS ’Ország’ FROM customers;

*WHERE - Selectek feltételek alapján*

## SELECT \* FROM customers WHERE country = ’Hungary’;

## *WHERE operátorai*

## <>(nem egyenlő), < és >, between 2 and 4, in(2,4,5)

*WHERE feltételek összekacsolása*

## SELECT \* FROM customers WHERE country = ’Hungary’ AND city = ’Budapest’;

## SELECT \* FROM customers WHERE country = ’Hungary’ OR city = ’Budapest’;

## SELECT \* FROM customers WHERE NOT country = ’Germany’;

## SELECT \* FROM customers WHERE country = ’Hungary’ AND city = Berlin OR city = Budapest’;

## SELECT \* FROM customers WHERE country = ’Hungary’ AND (city = Berlin; OR city = Budapest’);

*ORDER BY - Találatok sorba rendezése*

## SELECT \* FROM customers ORDER BY country;

## SELECT \* FROM customers ORDER BY country, name;

## SELECT \* FROM customers ORDER BY country DESC, name;

## SELECT \* FROM customers ORDER BY country ASC, name DESC;

*LIMIT – csak az első x rekord!*

…

## ORDER BY.. DESC LIMIT 10;

*MIN, MAX COUNT, AVG, SUM*

**SELECT** MIN**(price) AS SmallestPrice FROM products;**

**SELECT** MAX**(price) AS LargestPrice FROM products;**

**SELECT COUNT(id)FROM products WHERE category\_Id = 2;**

**SELECT COUNT(\*) FROM `orszagok` WHERE `terulet` > 50000 AND `terulet` <** **150000;**

**SELECT AVG(price) FROM products;**

**SELECT SUM(price) FROM products WHERE category\_id=1;**

*GROUP BY - Találatok csoportosítása - A GROUP BY mindig az ORDER BY előtt van!*

**SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country;**

*LIKE - Keresés hasonlóság alapján - ’\_egy karakter ÉS ’%’ bármi*

**SELECT \* FROM customers WHERE name LIKE ’a%’;**

**SELECT \* FROM Customers WHERE CustomerName LIKE '\_r%';**

*Beágyazott lekérdezések – selectben select*

**SELECT `orszag` FROM `orszagok`**

**WHERE `terulet` < (SELECT `terulet` FROM `orszagok` WHERE `orszag` = 'MAGYARORSZÁG');**

**SELECT `orszag`, `terulet` FROM `orszagok` WHERE `terulet` = (SELECT MAX(`terulet`) FROM `orszagok`);**

**SELECT `orszag`, `terulet` FROM `orszagok` WHERE `terulet` = (SELECT MIN(`terulet`) FROM `orszagok`);**

*INSERT – Új rekordok beszúrása az adattáblákba*

**INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)**

**VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');**

**INSERT INTO Customers VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');**

**INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)**

**VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway'),**

**('King', 'Koen Jensen', 'Oslo 21', 'Oslo', '7000', 'Norway'),**

**('Cool', 'Viking Kirksen', 'Berlin 44', 'Bergen', '2001', 'Norway');**

*ADD – meglévő tábla oszloppal való bővítése*

**ALTER TABLE `befiz` ADD `ugyfel\_azon` INT(10) UNSIGNED NOT NULL;**

*UPDATE - Az egyes rekordok frissítése; legyen mindig WHERE!*

**UPDATE Customers SET ContactName = 'Alfred Schmidt', City = 'Frankfurt' WHERE CustomerID = 1;**

*DELETE - Rekordok törlése; legyen mindig WHERE!*

**DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste'**

*JOIN - Táblák összekötése - egyszerre több táblából is szeretnénk lekérni adatot, egy lekérdezéssel*

*inner/left/right, full outer join*

**SELECT `orders`.id, `customers`.name, `orders`.order\_date FROM `orders` INNER JOIN `customers` ON `orders`.customer\_id = `customers`.id;**

*Join megvalósítása összerendelő tábla segítségével több a többhöz kapcsolat esetén*

**SELECT \* FROM products p JOIN product\_to\_category ptc ON ptc.product\_id = p.Productid JOIN categories c ON c.id = ptc.category\_id ORDER by c.id DESC;**

**SELECT citizens.citizen\_name, citizens.age, cities.city FROM citizens JOIN cities ON citizens.zip=cities.zip WHERE citizens.age >100;**

*Egy tábla csatolása saját magához – egyezések keresése vagy párokba rendezés*

**SELECT A.name AS CustomerName1, B.name AS CustomerName2, A.city FROM customers A, customers B WHERE A.id <> B.id AND A.city = B.city;**

*Megszorítások – Constraints; bizonyos hosszúságú vagy formátumú, ne ismétlődjön, növekedjen automatikusan, ne legyen null*

**ALTER TABLE customers ADD CONSTRAINT unique\_name\_pairs UNIQUE (`name`, `contact`);**

**ALTER TABLE `customers` MODIFY COLUMN `id` int NOT NULL AUTO\_INCREMENT;**

*Beszúrás után autoincrement helyreállítás:*

**ALTER TABLE `customers` AUTO\_INCREMENT = 6;**

*Index hozzárendelése (gyorsabb kereséspl. név és kontakt alapján, de plusz memória)*

**CREATE INDEX name\_contact ON customers (NAME, contact);**

## ON DELETE CASCADE: töri a hozzátartozókat is (rekordhoz tartozó összes másik rekordot az idegenek táblákból a foreign kulcs mentén)

## ON UPTADE CASCADE – végigviszi a változtatást az összes kacsolódó táblában is

## SET DEFAULT – megmarad az adat, ráállítják egy default azonosítóra

## nem törlünk, hanem inkább archiválunk vagy set default egy törlésre előkészített azonosítóra

## ON DELETE SET NUL – ami foreign keyből jön, arra törléskor(pl tábla drop) NULL-t állít be

*Tábla kreálás auto incrementtel és keys-el:*

CREATE TABLE `log` (

id INT NOT NULL AUTO\_INCREMENT,

message VARCHAR(500) NOT NULL,

product\_id INT NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY (product\_id) REFERENCES products (id)

);

*UNIQUE:*

CREATE TABLE Persons (

    ID int NOT NULL UNIQUE,

    ID email NOT NULL UNIQUE,

    LastName varchar(255) NOT NULL,

    FirstName varchar(255),

    Age int

);

*UNION - kettő vagy több SELECT utasítás eredménykészletét kombinálja; a MariadBD nem ismeri a full outer joint, a helyett is jó az union*

**SELECT City, Country FROM Customers WHERE Country='Germany' UNION**

**SELECT City, Country FROM Suppliers WHERE Country='Germany' ORDER BY City;**

**SELECT 'Customer' As Type, ContactName, City, Country**

**FROM Customers UNION SELECT 'Supplier', ContactName, City, Country FROM Suppliers;**

**SELECT \* FROM `pizza` WHERE `pizza\_id` < 2000 UNION**

**SELECT \* FROM `pizza` WHERE `pizza\_name` LIKE '%m%';**

*Having - Csoportfeltételek létrehozása; ha már csoportosítottam az oszlopokat group by-al, utána már alapba nem tudok szűrőt megadni; ezt lehet megoldani HAVING-el*

**SELECT Sz\_Helye AS 'Születési Hely', SUM(fiz) AS 'Össz fizetés', AVG(fiz) AS 'Átlag fizetés'**

**FROM employees GROUP BY Sz\_Helye HAVING AVG(fiz) >= 120000;**