**Jászberényi Katolikus Óvoda, Általános Iskola Liska József Középiskola és Kollégium Tagintézménye**

**Készítette: Nagy Ferenc, Szakmai informatika tanár**

**14. tétel**

**Adatbázis- és szoftverfejlesztés – Adattípusok (2.1.2.)**

**Adatbázis kezelés – SQL utasítások**

Definiálja a következő kifejezéseket: adatbázis, tábla, mező, rekord, kulcs, normalizálás. Ismertesse az Ön által tanult adatbázis kezelő rendszer adattípusait!

Ismertesse az Ön által tanult adatbázis-kezelő rendszer alapvető utasításait, lekérdezés típusait, táblák közötti kapcsolatok létrehozását.

***Szempontok a tartalom értékeléséhez***

• Adatbázis, mező, rekord, kulcs, normalizálás ...

• Numerikus adattípusok, dátum, idő, karakterlánc...

• Lekérdezések: *SELECT, FROM, WHERE, BETWEEN...AND, ORDER BY, GROUP BY, LIKE, LIMIT…*

• Lekérdezés típusok: *INSERT INTO… VALUES..., DELETE, UPDATE ...SET…*

• Kapcsolatok létrehozása: *INNER JOIN...ON, LEFT OUTER JOIN, RIGHT OUTER JOIN*

• *COUNT(), MIN(), MAX(), SUM()...*

Alapfogalmak:

* **Az információ nem azonos az adattal**, hanem annak valamilyen jelentése. Az adat az információval ellentétben objektív. Az adatbázis-kezelés arra alkalmas, hogy tényeket tároljon adatbázisokban és azokat olyan formában mutassa meg, hogy abból információhoz jussunk. **Az adat a számítástechnikában tehát csak egy jelsorozat, amely a feldolgozás során válik információvá.**
* Az **adatállomány** olyan összefüggő adathalmaz, amelyben minden olyan adat megtalálható, amire egy bizonyos cél megvalósítása érdekében szükség lehet.
* **Az adat értelmezhető (észlelhető, érzékelhető, felfogható, és megérthető) ismeret.**
* **Az információ új ismeretté értelmezett adat.**
* Tulajdonság **amivel leírjuk a bennünket érdeklő jelenséget, tulajdonságnak nevezzük.** Tulajdonság-értéknek nevezzük egy tulajdonság konkrét értékét. A tulajdonság-értékhalmaz az egy adott időpontban előforduló összes érték.
* **Elsődleges kulcs**: az egyed azon tulajdonságát, amely **minden egyed-előfordulásra eltérő értéket vesz fel**, az egyed azonosítójának, más néven elsődleges kulcsának nevezzük.
* **Az egyedek között lévő összefüggéseket kapcsolatnak hívjuk.** A gyakorlatban az, hogy az egyedek között milyen kapcsolatot hozunk létre nem magától értetődő és igen nehéz feladat. Ez jelenti az adatbázis tervezésének lényegét és nehézségét.
* **Adattábla, tábla, adatállomány**: megegyezik az egyed fogalmával. A rendszer az adatokat táblázatos formában kezeli, és táblázatokban kell gondolkodnunk.
* **Mező, oszlop**: ez a tulajdonságnak felel meg. Az adott tulajdonság nevét **mezőnévnek**, a tulajdonság-előfordulást pedig **mező-értéknek** nevezzük. A mezőt **oszlopnak** is szokták nevezni.
* **Rekord, sor**: megegyezik az **egyed-előfordulással**. Rekordnak nevezzük a tábla sorába bevitt értékeket. Ezek kizárólag csak egy egyedre vonatkoznak.
* **Elemi adatok**: a tábla celláiban szereplő értékek.
* **Egyed**: amelynek az adatait tároljuk az adatbázisban. Egyednek tekinthetünk például egy személyt.
* **Attribútum**: tulajdonság, az egyed valamely jellemzője. Egy személy jellemzője lehet, például a magassága.
* **Egyedtípus**: az egyedre vonatkozó tulajdonságok összessége.
* **Egyed-előfordulás**: az egyedre vonatkozó konkrét tulajdonságok. Például: Opel Astra, 10 éves 1400 ccm-es, kék színű.
* **Normalizálás**: A tervezés során olyan adatstruktúrákat alakítsunk ki, amelyek segítik a hatékony adatkezelést. Fontos hogy egy-egy táblába csak a valóban logikailag összetartozó adatok kerüljenek, és hogy minél kevesebb ismétlődés legyen az adatok között.  
  A relációs modellben külön eljárást fejlesztettek ki arra vonatkozóan, hogy az adatok megfelelő strukturálását, a redundancia csökkentését elősegítsék.  
  Ez a módszer a normalizálás.
* Azt mondjuk, hogy egy *reláció* **első normálformában** (1NF) van, ha a relációban minden érték elemi, vagyis a reláció nem tartalmaz adatcsoportot.
* **Második normál forma:** Egy relációt **második normálformájú**nak (2NF) szokás nevezni, ha első normálformájú, és egyetlen másodlagos attribútuma sem függ egyetlen kulcsának valódi részhalmazától.  
  Ha tehát a reláció kulcsa egyetlen mező (vagyis attribútum), a reláció automatikusan 2NF-ben is van. Ha a kulcs összetett, akkor a kulcs semmilyen része nem képes a reláció nem-kulcs mezőinek értékét egyértelműen meghatározni.
* **Harmadik normál forma:**Egy R reláció **harmadik normálformá**ban van, ha  
  második normálformájú és egyetlen másodlagos (nem-kulcs) attribútuma sem függ tranzitíven a kulcstól
* **Tranzitív függés:** Egy A – C funkcionális függést tranzitív függésnek nevezünk, ha létezik olyan B attribútum halmaz, hogy A - B továbbá B - C.
* **Funkcionális függés:** Adatok között akkor áll fenn funkcionális függőség, ha egy vagy több adat konkrét értékéből más adatok egyértelműen következnek.

Kulcstípusok (összesítve):

* **Egyszerű kulcs:** a kulcs egyetlen attribútumból áll.
* **Összetett kulcs:** a kulcsot kettő vagy több oszlop kombinációja alkotja, előfordulhat az is, hogy a összes oszlop szerepel a kulcsban.
* **Minimális kulcs:** ha összetett kulcs esetén bármely attribútumot elhagyjuk a kulcsból, és az így megmaradt oszlopok kombinációja már nem rendelkezik kulcs tulajdonsággal, akkor az összetett kulcsot minimálisnak nevezzük. Az egyszerű kulcs mindig minimális.
* **Elsődleges kulcs** (primary key): az a kulcs, melyet kiválasztunk a kulcsjelöltek közül, és kulcsként használunk. A ki nem választott kulcsjelölteket alternatív kulcsnak nevezzük. Az elsődleges kulcsnak nem lehet NULL az értéke.
* **Idegen kulcs** (foreign key): olyan attribútum vagy attribútum halmaz egy adott relációban, amelyik egy másik relációban elsődleges kulcsként szerepel. Az idegen kulcsot tartalmazó relációt hivatkozó relációnak, a másikat, melyben ez a kulcs elsődleges, hivatkozott relációnak nevezzük.

Adatmodell típusok:

* **Hierarchikus adatmodell**: az adatokat egy hierarchikus szerkezetben tárolja, amely egy **fához hasonlítható.** A fa mindegyik csomópontja egy rekordtípusnak felel meg. **Az adatok között un. szülő-gyermek kapcsolat van.** Minden adatnak tetszőleges számú leszármazottja lehet, de csak egy őse. Előnye, hogy a hierarchikus szerkezet egyszerűen leírható, könnyen elkészíthető. Napjainkban ez az adatbázis már **elavult**.
* **Hálós adatmodell: a hierarchikus modell továbbfejlesztése.** Ebben a modellben az adatok között tetszőleges kapcsolatrendszer alakítható ki. Egy adatnak több őse is lehet. **Hátránya: bonyolult kapcsolatok és nagy tároló igény. Nagy-gépes környezetben fordul elő.**
* **Relációs adatmodell:** a különböző jellegű, de mégis kapcsolatba hozható adathalmazokat önálló táblákban tároljuk, amely táblák között egy azonos adatot tartalmazó mező tartja a kapcsolatot. **Jelenleg ez a legelterjedtebb adatmodell.** Ezt az adatmodellt támogatják a legismertebb adatbázis-kezelő programok is, mint a **dBase, Clipper, FoxPro, Access, Oracle, MSSQL, MySQL**.
* **Objektumorientált adatmodell:** valósághű adatbázist lehet tervezni és készíteni **objektumorientált technikával.** Ez a modell ma még kevésbé elterjedt.

Adattípusok:

* Karakter(CHAR, **VARCHAR**, TEXT, TINYTEXT, MEDIUMTEXT, LONGTEXT)
* Egyéb karakter típusok ( BINARY, VARBINARY, ENUM, SET stb…)
* Szám – egész (**INT**, TINYINT, SMALLINT, MEDIUMINT, BIGINT)
* Szám – valós (DECIMAL, FLOAT, **DOUBLE,** REAL)
* Egyéb szám típusok (**BIT,** BOOLEAN, SERIAL)
* Idő(**DATE, TIME**, DATETIME, TIMESTAMP,YEAR)
* Térbeli (GEOMETRY, POINT, POLYGON, LINESTRING, MULTIPOINT …)stb

A **SELECT ÉS A FROM**: elsőként kell megadni, **SELECT** mögé azokat a mezőket adjuk meg, amiket szeretnénk megjeleníteni, **FROM** mögé pedig azokat a táblákat soroljuk fel, amikből megtörténik az elem kiválasztása.

***SELECT mezőnév1, mezőnév2 FROM táblanév1, táblanév2;***

Ha a **SELECT** után **csillagot** írunk (**\***) akkor az összes mezőt kiíratjuk a lekérdezéssel az adott táblákból.

**DISTINCT** parancs esetén, ha egy elem többször fordulna elő a lekérdezésben, akkor is csak egyszer jelenik meg! Mindig közvetlenül a SELECT után kell írni az alábbi minta szerint:

*SELECT* ***DISTINCT*** *mezőnév1, mezőnév2 FROM táblanév1, táblanév2;*

**LIMIT** segítségével az adott lekérdezésben megjelenő elemek számát adhatjuk meg. Ezt pedig a lekérdezés végére szokás megadni. Pl.: 10 darab elem kiíratása, ami megfelel a lekérdezésnek.

*SELECT mezőnév1, mezőnév2 FROM táblanév1, táblanév2* ***LIMIT 10;***

**ORDER BY** paranccsal rendezhetjük a lekérdezés elemeit, mégpedig az alapján rendezünk, amit az **ORDER BY** után megadunk. Ha fordított sorrendet szeretnénk a **DESC** kifejezést írjuk a mezők után.

*SELECT mezőnév1, mezőnév2 FROM táblanév1, táblanév2* ***ORDER BY*** *mezőnév1, mezőnév2* ***DESC;***

**WHERE** után feltétel megadással is szoktuk szűrni a lekérdezéseket.

*SELECT mezőnév1, mezőnév2 FROM táblanév1, táblanév2* ***WHERE mezőnév=”feltétel”;***

*feltétel megadáskor a számokat csak simán leírjuk, szöveg megadásakor idézőjelet használunk!*

Valamint használhatjuk az **OR (vagy), AND (és) illetve a NOT (nem)** logikai operátorokat, is mint más nyelvek esetében, a feltételek összekapcsolására, annak megadásakor!

*SELECT mezőnév1, mezőnév2 FROM táblanév1, táblanév2* ***WHERE*** *mezőnév1=”feltétel1”* ***AND*** *mezőnév2=”feltétel2”* ***;***

*Feltételben még használhatod a LIKE kifejezést is, ilyenkor elég, ha a feltétel egy része egyezik!*

**AS** kifejezéssel egy alternatív nevet is megadhatunk egy adott mezőnek ha szeretnénk:

*SELECT* ***mezőnév1 AS aliasnév1, mezőnév2 AS aliasnév2*** *FROM táblanév1, táblanév2;*

Ilyenkor a lekérdezésben ezen a néven is hivatkozhatunk rá, valamint a végén ezen a néven jelenik meg a mezőnév, a lekérdezés adatai „fejléceként”.

**GROUP BY** opciót használva, egy adott adat alapján csoportosíthatjuk a többit. (Összesítéskor jön jól)

*SELECT mezőnév1, mezőnév2 FROM táblanév1, táblanév2* ***GROUP BY*** *mezőnév1;*

***FONTOS! GROUP BY*** *parancs esetén, ha feltételt használunk azt a* ***HAVING*** *kifejezés után adjuk meg*

*SELECT mezőnév1, mezőnév2 FROM táblanév1, táblanév2* ***GROUP BY*** *mezőnév1* ***HAVING*** *feltétel;*

Táblák összekapcsolása szintén mindig van, 2 féle képpen lehet megvalósítani:

1. Vagy az **INNER JOIN** paranccsal, melyet a **FROM**után kell közvetlenül megadni pl.:

**INNER JOIN** *táblanév ON táblanév2.mezőnév2 =táblanév1.mezőnév1;*

A lényeg az egyenlőség jel jobb és baloldalán ugyan azoknak az értékeknek kell lennie, ezzel összekapcsolhatjuk a két tábla adatait. *(általában a táblában Primary Key(PK), elsődleges kulcs)*

1. Illetve nem kell összekapcsolni a táblákat, ha minden mezőnév megadásakor elé írjuk mely táblából szeretnénk azt megadni! ***táblanév.mezőnév***

Végül, de nem utolsó sorban mindig van egy **függvény** használata, melyet a **SELECT**-**nél** szoktunk legtöbbször megadni, **de máshol is lehet(pl.: feltétel megadáskor)**, ilyenkor az adott mezőnévben lévő adatokkal végzünk el egy műveletet és annak eredményével dolgozhatunk.

SUM(mezőnév) -> az adott mezőnévben lévő elemek összegét kaphatjuk meg

AVG(mezőnév) -> az adott mezőnévben lévő elemek átlagát adja eredményül

MIN(mezőnév) -> az adott mezőnévben lévő elemek közüli legkisebb értéket kapjuk vissza

MAX(mezőnév) -> az adott mezőnévben lévő elemek közüli legnagyobb értéket kapjuk vissza

COUNT(mezőnév)-> az adott mezőnévben lévő elemek számát adja meg

Igen gyakran szükségünk lehet egy már meglévő adatbázis táblázatának bővítésére, ilyenkor az INSERT INTO parancsot használjuk, mi a nevében is benne van ilyenkor illesztünk be a táblázatba egy adott rekordot, és adunk neki értéket.

**INSERT INTO `**táblázat\_neve` (`mező01\_neve`, `mező02\_neve`, `mező\_03\_neve`) **VALUES** (mező01\_értéke, mező02\_értéke, mező03\_értéke);

Például: INSERT INTO `tanulo` (`nev`, `eletkor`) VALUES („Kis József”, 17);

Megjegyzések: a táblázatot belül az értékadás a **VALUES** utasítás után történik.   
Az értékeket a nekik megfelelő formátumban kell megadni, például szöveg esetén idézőjelek „” között!

INSERT Új sor(ok) felvitele táblába, nézetbe

UPDATE Táblában meglévő sor(ok) módosítása

DELETE Táblában meglévő sor(ok) törlése