3. gyakorlat

I. Rész – Where klóz és Null értékek

1. Null értékek kezelése

SQL-ben, ha csak máshogy nem rendelkezünk, lehetnek 'NULL' értékek. Ekkor azonban az összehasonlító operátorokat (<>,=,<) és a függvényeket óvatosan kell használni. Erre megoldás az IS NOT NULL és az IS NULL használata.

```
SELECT *
FROM   patient
WHERE  name IS NULL
```

Feladatok null-ok kezeléséhez:

a) Hozzatok létre, majd töltsetek fel egy táblát úgy, hogy a következő adatokat tartalmazza, majd válasszátok ki azokat a recordokat, amelyek cost attribútuma nem 'üres'!

Treatment_id	Patient_id	Cost
T001	P1500	55
T003	P1500	
T004	P1500	57
T005	P9500	
T006	P4000	81
T007	P4000	82

b) Opcionális: válasszátok ki az összes rekordot, de úgy, hogy a Cost mezőben hiányzó értékeket 0-val helyettesítitek! (Használd az nvl() függvényt)

c) Mit ad vissza a következő lekérdezés?

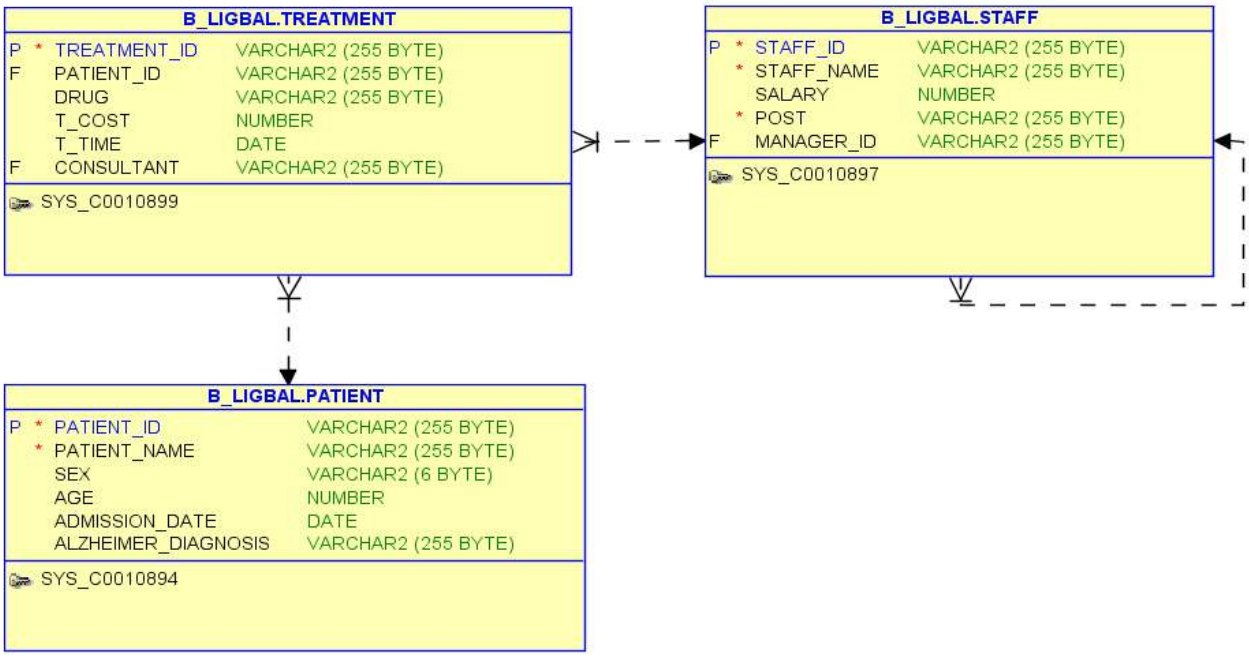
```
SELECT *
FROM   treatmentvar
WHERE  cost < 60 OR cost >= 60;
```

II. Rész – Egyszerű függvények, join

Töltsd le a tárgy wiki oldaláról a gyak3_02.sql fájlt, majd ezt a szkript fájlt futtasd le az sql developerben!

Link:

Relációs séma:



1. Egyszerű függvények

Alapvetően két típusba csoportosíthatjuk a függényeket: aggregáló vagy skaláris. Az aggregáló függvények jellemzően 1 db értékkel térnek vissza, amit az adott oszlopban lévő összes adatból számolnak ki. Az aggregáló függvényekre példa:

a) avg() – Az átlaggal tér vissza
b) count() – A sorok számával tér vissza
c) first() – Az első értékkel tér vissza
d) last() – Az utolsó értékkel tér vissza
e) max() – A legnagyobb értékkel tér vissza
f) min() – A legkisebb értékkel tér vissza
g) sum() – A sorok összegével tér vissza

A skaláris függvények ezzel szemben jellemzően egy értéket rendelnek hozzá valamilyen inputhoz. Ilyen pl:

a) to_date() – Dátum típusra konvertál egy stringet
b) to_number() – Számra konvertál egy dátumot.
c) lower() – Stringből csupa kisbetűs stringet állít elő
d) upper() – Stringből csupa nagybetűs stringet állít elő
e) nvl() – Nullokat cseréli valamilyen megadott értékre
f) round()– Számot kerekít

2. Feladatok egyszerű függvények gyakorlásához:

a) Írd ki a gyógyszerek nevét kisbetűvel!
b) Mennyi az átlag fizetés?
c) Egy hónapban a kórház mennyi pénzt költ a dolgozók fizetésére?
d) Mikor volt a legkorábbi és a legkésőbbi kezelés?
e) Hány kezelés volt összesen?
f) Hányszor adtak gyógyszert összesen?
g) Opcionális: mi a betegek vezeték - és keresztneve? Milyen függvények kellenek ehhez? (használd a google-t)

3. JOIN

Az összetett kérdéseinkre gyakran nem tudunk válaszolni 1 tábla segítségével, mert a különböző entitásokhoz tartozó információkat praktikus okokból külön táblákban tároljuk. A táblák összekapcsolását JOIN segítségével tehetjük meg, valamilyen olyan attribútum alapján, ami mindkét táblában megtalálható (ez jellemzően egy elsődleges kulcs és egy külső kulcs páros). Tehát ha az érdekel minket, hogy egy adott kezelést milyen orvos végzett el (mi az orvos neve, egyéb adatai), akkor szükségünk van a *treatment* és a *staff* táblákra. Itt a közös attribútum a kezelést végző orvos id-ja. A *treatment.consultant* a külső kulcs, és a *staff.staff_id* az elsődleges kulcs. Ekkor a join így néz ki:

```
SELECT treatment.treatment_id,
       staff.STAFF_NAME
FROM   treatment, staff -- kereszt (Descartes) szorzat
WHERE  TREATMENT.consultant = staff.staff_id;
```

4. Feladatok join gyakorlásához:

a) Melyik kezelést ki végezte és mennyibe került? Az eredményt rendezd kezelés ára alapján csökkenő sorrendbe!
b) Melyik beteget ki kezelte és mikor?
c) Kinek ki a közvetlen főnöke?

III. Rész – Egyszerű halmaz műveletek

5. Distinct

Ha arra vagyunk kíváncsiak, hogy egy oszlopban (vagy oszlopokban) milyen egymástól eltérő értékek vannak, akkor a SELECT DISTINCT konstrukciót használhatjuk.

Szintaxis:

```
SELECT DISTINCT column_name, column_name
FROM table_name;
```

Pl. A különböző nevű dolgozok listája:

```
SELECT distinct name
FROM   dolgozo
```

6. Feladatok

a. Az eddig kezelések során milyen gyógyszereket használtak?
b. Milyen lehetséges posztok és a poszttal járó fizetések vannak?

7. Union, intersect, minus

Szintén gyakori, hogy valamilyen eredményeknek az uniójára, metszetére vagy különbségére vagyunk kíváncsiak.

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

8. Feladatok

a. Ki az, aki nem végzett egyetlen kezelést sem?
b. Ki az, aki főnöke valakinek, de nem ő az igazgató?
c. Listázd ki az összes kórházban lévő ember nevét, és azt, hogy beteg-e, vagy alkalmazott!

IV. Rész – Egymásba ágyazott lekérdezések

1. VIEW

Egy lekérdezés (SELECT) eredményére tekinthetünk úgy, mint egy „táblára”. Azaz magát a query-t eltárolhatjuk egy virtuális táblaként.

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

Ekkor már a view-ból is le lehet kérdezni, úgy mintha az egy tábla volna:

```
SELECT column_name(s)
FROM view_name
WHERE condition
```

pl.

```
CREATE VIEW KivegezteAkezelest AS
SELECT
t0.treatment_id,
t1.STAFF_NAME,
t0.t_cost
FROM treatment t0
INNER JOIN staff t1
ON t1.STAFF_ID = t0.CONSULTANT;
```

2. Feladatok

a. Hozz létre egy view a következő lekérdésekhez: melyik beteget ki kezelte és mikor?
b. A nézet használatának segítségével válaszold meg, hogy mely betegeket kezelte Dr. Green 2004 októberében.

3. Lekérdezések a from, where, stb klózban

Mivel a lekérdezés eredménye egy „tábla”, ezért ezeket felhasználhatjuk más lekérdezésekben is.

4. Feladatok:

a. Melyik dolgozónak mennyivel több vagy kevesebb a fizetése, mint az átlagos fizetés?
b. Melyek azok az orvosok, aki végeztek már legalább egy kezelést?