

## Database Design Exercises (Ex No: 2)

Practitioner: Mobin Kheibary [994421017]

Supervisor: Dr. Ehsan Shoja

Solution to [Ex-14](#):

Appending **NATURAL JOIN** *section* in the **FROM** clause would not change the result because there is no logic in the query that uses attributes in *section* that are not found in *takes* relation. Also since every tuple in *takes* relation has a corresponding tuple in the *section* relation (due to foreign key constraint), there will be no change if we append the **NATURAL JOIN** *section* in the **FROM** clause.

Solution to [Ex-15](#):

```
SELECT
c.building,c.room_number,course_id,sec_id,semester,year,time_slot_id,capacity
FROM section INNER JOIN classroom c USING (building, room_number);
```

Bonus:

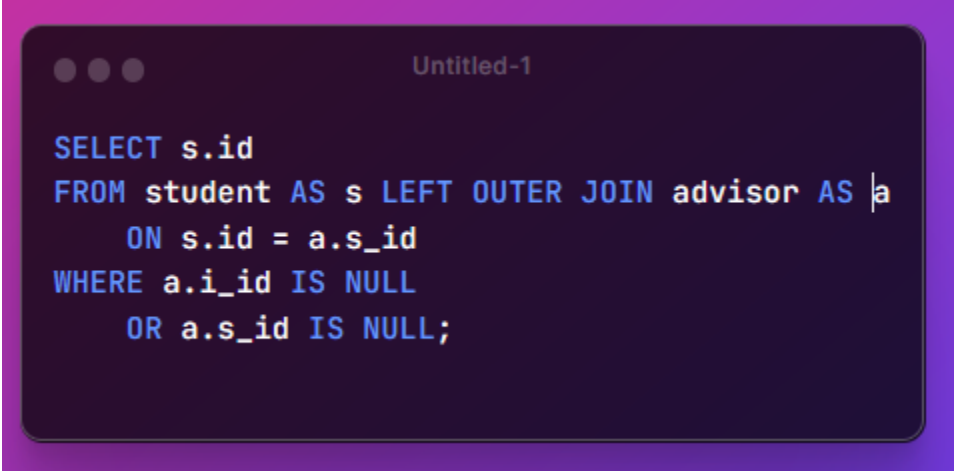
To check if the above queries give same results we can use the following query:

```
WITH q1 AS (
  -- q1 stands for query 1
  SELECT *
  FROM section NATURAL JOIN classroom
), q2 AS (
  -- q2 stands for query 2
  SELECT
  c.building,c.room_number,course_id,sec_id,semester,year,time_slot_id,capacity
  FROM section INNER JOIN classroom c USING (building, room_number)
)
-- we need to check that result of q1 is a subset of q2 and
-- result of q2 is a subset of q1
SELECT
  NOT EXISTS (
    (SELECT * FROM q1)
    EXCEPT
    (SELECT * FROM q2)
  )
  AND
  NOT EXISTS (
    (SELECT * FROM q2)
    EXCEPT
    (SELECT * FROM q1)
  )
```

On my instance of the database in PostgreSQL the above query gives 't' meaning true.

#### Solution to [Ex-17](#):

The query is going to get the ids of students that don't have an advisor. That means, those students that have ids that don't appear in the *advisor* relation or their advisor's id is set to null.



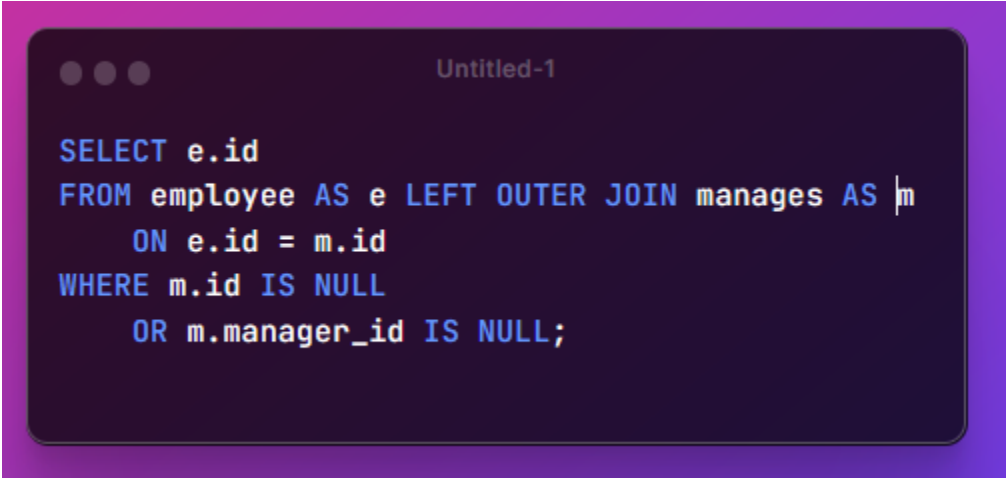
```
SELECT s.id
FROM student AS s LEFT OUTER JOIN advisor AS a
    ON s.id = a.s_id
WHERE a.i_id IS NULL
    OR a.s_id IS NULL;
```

Note that we are assuming, the primary key of *advisor* to be *s\_id*. That is each student can have at most one advisor.

#### Solution to [Ex-18](#):

Note that, the primary key of the *manages* relation is *id*. Thus each employee can have at most one manager.

Using an outer join we can write the query as:



```
SELECT e.id
FROM employee AS e LEFT OUTER JOIN manages AS m
    ON e.id = m.id
WHERE m.id IS NULL
    OR m.manager_id IS NULL;
```

Using no outer join we can write the query as:

```
SELECT id
FROM employee
WHERE id NOT IN (
    -- this gives us all of the employee ids that have a manager
    SELECT id
    FROM manages
    WHERE manager_id IS NOT NULL
);|
```

Solution to [Ex-20](#):

```
CREATE VIEW tot_credits(year,num_credits) AS (
    SELECT year, SUM(credits)
    FROM takes NATURAL JOIN course
    GROUP BY year
    ORDER BY year ASC
)
```

Solution to [Ex-21](#):

I think there is an error with the question.

There is no view in Exercise 4.18.

Solution to [Ex-22](#):

The following is taken from PostgreSQL 14.1 documentation in section 9.18.2.

```
COALESCE(value [, ...])
```

The COALESCE function returns the first of its arguments that is not null. Null is returned only if all arguments are null. |

If a query is written using coalesce function as:

```
SELECT COALESCE(  
    d1,  
    d2,  
    .  
    .  
    .  
    dn,  
) FROM r|
```

It can be rewritten as:

```
SELECT  
    CASE  
        WHEN d1 IS NOT NULL THEN d1  
        WHEN d2 IS NOT NULL THEN d2  
        .  
        .  
        .  
        WHEN dn IS NOT NULL THEN dn  
        ELSE NULL  
    END  
FROM r|
```

### Solution to [Ex-23](#):

Assume the grant is done by the user Satoshi instead of the manager role. Then if Satoshi leaves the company one day, and the DBA revokes Satoshi's authorization, all of the employees granted by Satoshi will have their authorization revoked.

If the DBA doesn't want that, he should have the manager role grant an authorization instead of the user Satoshi.

### Solution to [Ex-24](#):

I don't think so. Since user A has all authorization privileges on a relation  $r$ , B granting select on  $r$  to A doesn't bring any new privilege to user A.

I guess this depends on the internals of the Database Management System.

### Solution to [Ex-25](#):

The **references** privilege needs to be granted to the user on the relation  $r_2$ . This should not be allowed without any such authorization because, foreign-key constraints restrict deletion and update operations on the referenced relation. That is, operations such as **update** and **delete** on  $r_2$  may bring changes to  $r_1$  as well.

### Solution to [Ex-26](#):

**Integrity constraints** ensure that changes made to the database by authorized users do not result in a loss of data consistency. That is, they guard us against accidental damage to the database.

- Domain constraints
- Unique constraints
- Referential Integrity constraints

**Authorization constraints** guard against access to the database by unauthorized users.

Example:

- Authorization to read data
- Authorization to insert new data
- Authorization to delete data

*The End.*