INFS1200 — Week 9 Notes

Table of contents

Applied Class 8: DQL Nested Queries and Views

Practice for 2026-04-13-nested-queries-views-and-generative-ai[1]. Same BestTechLtd schema as week8-tutorial-applied-class-7-multiple-relation-queries[2]:

```
Employee               [EmployeeID, FirstName, LastName, DOB, PasswordHash, PasswordSalt]
AdministrativeEmployee [EmployeeID, Level, Type]
Role                    [RoleID, Name, Description]
RoleGranting            [EmployeeID, RoleID, AdministrationID, Timestamp]
Permission              [WebsiteURI, RoleID, GrantType, Description]

AdministrativeEmployee.EmployeeID references Employee.EmployeeID
RoleGranting.EmployeeID references Employee.EmployeeID
RoleGranting.RoleID references Role.RoleID
RoleGranting.AdministrationID references AdministrativeEmployee.EmployeeID
Permission.RoleID references Role.RoleID
```

[1] courses/infs1200/2026-04-13-nested-queries-views-and-generative-ai.pdf
[2] courses/infs1200/week8-tutorial-applied-class-7-multiple-relation-queries.pdf

Nested queries

Question 1

Return the Date of Birth, first and last name of the youngest employee(s).

> **i** Working

```
SELECT E.DOB, E.FirstName, E.LastName
FROM   Employee E
WHERE  DOB >= (SELECT MAX(DOB) FROM Employee E2);
```

`>=` (not `=`) against the max, so **ties** for youngest are all returned, not just one arbitrary row.

Question 2

*Return all information about all employees that have **not** been granted a role from an Administrator with type `'ProductEngineer'`. Restriction: must use a subquery.*

> **i** Working

```
SELECT E.*
FROM   Employee E
WHERE  EmployeeID NOT IN (
    SELECT DISTINCT E2.EmployeeID
    FROM   Employee E2
    JOIN   RoleGranting RG ON RG.EmployeeID = E2.EmployeeID
    JOIN   AdministrativeEmployee AE ON AE.EmployeeID = RG.AdministrationID
    WHERE  AE.Type = 'ProductEngineer'
);
```

Question 3

Find the number of employees who share a surname with another employee.

Working

```
SELECT COUNT(*) AS employees_with_shared_surname
FROM   Employee E1
WHERE  LastName IS NOT NULL
  AND  EXISTS (
      SELECT * FROM Employee E2
      WHERE  E1.LastName = E2.LastName AND E1.EmployeeID != E2.EmployeeID
  );
```

A correlated `EXISTS` check — for each `E1`, is there some *other* employee `E2` with the same last name?

Question 4

Return the `EmployeeId` of the Administrative Employee(s) that have granted at least as many roles to employees as Mehdi Rahman. Assume only one Mehdi Rahman exists, and include Mehdi Rahman in the result.

Working

```
SELECT     RG.AdministrationID
FROM       AdministrativeEmployee AE
JOIN       RoleGranting RG ON AE.EmployeeID = RG.AdministrationID
GROUP BY   RG.AdministrationID
HAVING     COUNT(DISTINCT RG.EmployeeID, RG.RoleID) >= (
    SELECT   COUNT(DISTINCT RG.EmployeeID, RG.RoleID)
    FROM     RoleGranting RG
    JOIN     Employee E ON RG.AdministrationID = E.EmployeeID
    WHERE    E.FirstName = 'Mehdi' AND E.LastName = 'Rahman'
    GROUP BY RG.AdministrationID
);
```

Question 5

Find the first name and last name of all employees that have access to at least all the website URIs that Employee `E0007` has access to.

Working

```
SELECT E.FirstName, E.LastName
FROM   Employee E
WHERE  NOT EXISTS (
    SELECT P.WebsiteURI
    FROM   Employee E2
    JOIN   RoleGranting RG1 ON RG1.EmployeeID = E2.EmployeeID
    JOIN   Permission P ON P.RoleID = RG1.RoleID
    WHERE  E2.EmployeeID = 'E0007'
      AND  P.WebsiteURI NOT IN (
          SELECT P2.WebsiteURI
          FROM   RoleGranting RG2
          JOIN   Permission P2 ON P2.RoleID = RG2.RoleID
          WHERE  RG2.EmployeeID = E.EmployeeID
      )
);
```

This is a **relational division via double negation** — "there is no website E0007 can access that E cannot" — see 2026-04-13-nested-queries-views-and-generative-ai[3] for the general pattern.

Views

Question 6

*Return the EmployeeID, FirstName, LastName and RoleId of all employees who were the **first** to receive a given RoleID.*

[3] courses/infs1200/2026-04-13-nested-queries-views-and-generative-ai.pdf

Working

```
DROP VIEW IF EXISTS EarliestRoleGranting;

CREATE VIEW EarliestRoleGranting AS
SELECT   RoleID, MIN(TimeStamp) AS EarliestTimeStamp
FROM     RoleGranting
GROUP BY RoleID;

SELECT E.EmployeeID, E.FirstName, E.LastName, RG.RoleID
FROM   RoleGranting RG
JOIN   EarliestRoleGranting ERG
       ON ERG.RoleID = RG.RoleID AND ERG.EarliestTimeStamp = RG.TimeStamp
JOIN   Employee E ON E.EmployeeID = RG.EmployeeID;
```

Question 7

Return the EmployeeID of the Administrator(s) that have granted the most permissions of any other Administrator. Restriction: must use one or more views.

Working

```
DROP VIEW IF EXISTS AdminPermissionsCount;

CREATE VIEW AdminPermissionsCount AS
SELECT      AE.EmployeeID,
            COUNT(WebsiteURI) AS NumOfPermissionsGranted
FROM        AdministrativeEmployee AE
LEFT JOIN   RoleGranting RG ON RG.AdministrationID = AE.EmployeeID
LEFT JOIN   Permission P ON P.RoleID = RG.RoleID
GROUP BY    AE.EmployeeID;

SELECT *
FROM   AdminPermissionsCount
WHERE  NumOfPermissionsGranted >= (
    SELECT MAX(APC2.NumOfPermissionsGranted)
    FROM   AdminPermissionsCount APC2
);
```