– Part 4

Data Manipulation Language (Advanced SQL Queries)

Advanced SQL Queries – Set Operations

- SQL incorporates set operations: `UNION` (set union) and `INTERSECT` (set intersection), and `EXCEPT` (set difference / minus).
- Set operations result in return of a relation of tuples (no duplicates).
- Set operations apply to relations that have the same attribute types appearing in the same order, e.g., list all students who have either a gmail or hotmail email account.

```
(SELECT * FROM Student WHERE Email like '%@gmail.com')
 UNION
(SELECT * FROM Student WHERE Email like '%@hotmail.com');
```

- For example, the following query will not work

```
(SELECT StudentID, Name FROM Student)
 UNION
(SELECT Email FROM Student);
```

Advanced SQL Queries – Join Operations

- When we want to retrieve data from *more than one relations*, we often need to use **join** operation.
- Consider the following queries, which both need a join operation between two relations:
  - List the names of all courses which have been enrolled by at least one student.
  - List all students, and their enrolled courses if any.

STUDENT			
StudentID	Name	DoB	Email

COURSE		
No	Cname	Unit

ENROL				
StudentID	CourseNo	Semester	Status	EnrolDate

Advanced SQL Queries – Inner Join

- **Inner Join**: tuples a[illegible] in the result only if there is at least one matching in both rela[illegible]
- For the query "list the [illegible] all courses which have been enrolled by at least one student", we use:

```
SELECT DISTINCT c.Cname
  FROM Course c INNER JOIN Enrol e ON c.No=e.CourseNo;
```


Course		
No	Cname	Unit
COMP2400	Relational Databases	6
COMP3600	Advanced Database Concepts	6

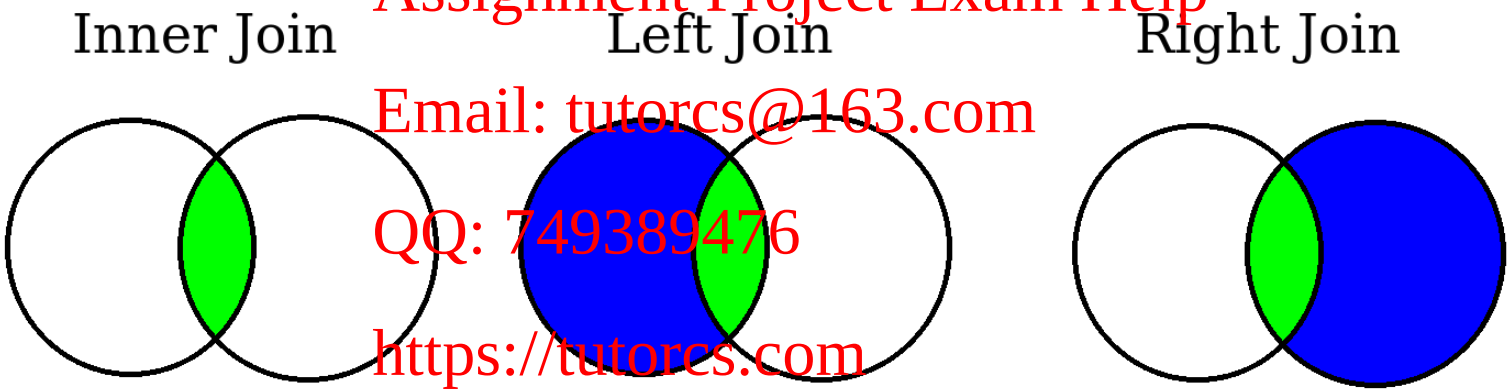
Enrol				
StudentID	CourseNo	Semester	Status	EnrolDate
456	COMP1130	2016 S1	active	25/02/2016
458	COMP1130	2016 S1	active	25/02/2016
456	COMP2400	2016 S2	active	09/03/2016

- Result:

Cname
Relational Databases

Advanced SQL Queries – Outer Join

- **Outer Join** includes **Left Join** and **Right Join**.
- **Left/Right Join**: all tuples of the left/right table are included in the result, even if there are no matches in the relations.



Advanced SQL Queries – Outer Join

- **Left Join**: A left join ows of the left table regardless of whether there is a row that m e right table.

ENROL1

	CourseNo	Semester
456	COMP1130	2016 S1
457	COMP1130	2016 S1
456	COMP2400	2016 S2

STUDENT

StudentID	Name	DoB	Email
456	Tom	25/01/1988	tom@gmail.com
458	Peter	20/02/1991	peter@hotmail.com


```
SELECT *
  FROM STUDENT s LEFT JOIN ENROL1 e
                        ON s.StudentID=e.StudentID;
```


StudentID	Name	DoB	Email	StudentID	CourseNo	Semester
456	Tom	25/01/1988	tom@gmail.com	456	COMP1130	2016 S1
456	Tom	25/01/1988	tom@gmail.com	456	COMP2400	2016 S2
458	Peter	20/02/1991	peter@hotmail.com	null	null	null

Advanced SQL Queries – Outer Join

- **Right Join**: A right j… all rows of the right table regardless of whether there is a ro… hes on the left table.

ENROL1		
StudentID	CourseNo	Semester
456	COMP1130	2016 S1
457	COMP1130	2016 S1
456	COMP2400	2016 S2

STUDENT			
StudentID	Name	DoB	Email
456	Tom	25/01/1988	tom@gmail.com
458	Peter	20/02/1991	peter@hotmail.com


```
SELECT *
 FROM STUDENT s RIGHT JOIN ENROL1 e
                      ON s.StudentID=e.StudentID;
```


StudentID	Name	DoB	Email	StudentID	CourseNo	Semester
456	Tom	25/01/1988	tom@gmail.com	456	COMP1130	2016 S1
null	null	null	null	457	COMP1130	2016 S1
456	Tom	25/01/1988	tom@gmail.com	456	COMP2400	2016 S2

Advanced SQL Queries – Outer Join

- For the query "list all students and their enrolled courses if any", we can use either of the following statements:

```
SELECT s.*, e.CourseNo, e.Semester
  FROM STUDENT s LEFT JOIN ENROL1 e
                 ON s.StudentID=e.StudentID;

SELECT s.*, e.CourseNo, e.Semester
  FROM ENROL1 e RIGHT JOIN STUDENT s
                 ON e.StudentID=s.StudentID;
```


- If we have 1000 tuples in STUDENT, then the query result should contain at least 1000 tuples (one tuple in STUDENT may occur multiple times) with the following attributes:

StudentID	Name	DoB	Email	CourseNo	Semester
...	...	...	...	...	...

Advanced SQL Queries – Natural Join

- **Motivation**: An inne s all the data of the two tables for , with duplication
- ```
  SELECT *
    FROM STUDENT s INNER JOIN ENROL1 e
                                On s.StudentID=e.StudentID;
  ```


ENROL1

StudentID	CourseNo	Semester
456	COMP1130	2016 S1
457	COMP1130	2016 S1
456	COMP2400	2016 S2

STUDENT

StudentID	Name	DoB	Email
456	Tom	25/01/1988	tom@gmail.com
458	Peter	20/02/1991	peter@hotmail.com

- Result:

StudentID	Name	DoB	Email	StudentID	CourseNo	Semester
456	Tom	25/01/1988	tom@gmail.com	456	COMP1130	2016 S1
456	Tom	25/01/1988	tom@gmail.com	456	COMP2400	2016 S2
  ```

Advanced SQL Queries – Natural Join

- **Natural Join**: A natu[illegible]ins all the data of the two tables for only the matched rows, witho[illegible]on

```
SELECT *
  FROM STUDENT NATURAL JOIN ENROL1 e;
```


ENROL1

StudentID	CourseNo	Semester
456	COMP1130	2016 S1
457	COMP1130	2016 S1
456	COMP2400	2016 S2

STUDENT

StudentID	Name	DoB	Email
456	Tom	25/01/1988	tom@gmail.com
458	Peter	20/02/1991	peter@hotmail.com

- Result:

StudentID	Name	DoB	Email	CourseNo	Semester
456	Tom	25/01/1988	tom@gmail.com	COMP1130	2016 S1
456	Tom	25/01/1988	tom@gmail.com	COMP2400	2016 S2

Advanced SQL Queries – Natural Join

- **Natural Join**: One k join, in which two relations are joined implicitly by compari utes of the same names in both relations.
- For the query "list all ho have enrolled and their courses", use:

```
SELECT * FROM STUDENT NATURAL JOIN ENROL;
```


ENROL

StudentID	CourseNo	Semester	Status	EnrolDate
456	COMP1130	2016 S1	active	25/02/2016
457	COMP1130	2016 S1	active	25/02/2016

STUDENT

StudentID	Name	DoB	Email
456	Tom	25/01/1988	tom@gmail.com
458	Peter	20/02/1991	peter@hotmail.com

- Result:(STUDENT.StudentID=ENROL.StudentID is used in the query)

StudentID	Name	DoB	Email	CourseNo	Semester	Status	EnrolDate
456	Tom	25/01/1988	tom@gmail.com	COMP1130	2016 S1	active	25/02/2016

Advanced SQL Queries – Subqueries

- **Subqueries** are just [illegible]t are used where a relation is required.
- Subqueries can be sp[illegible]hin the `FROM`-clause (usually in conjunction with aliases and renaming) to create *inline view* (exist only for the query)
- Subqueries can also be specified within the `WHERE`-clause, e.g.,
  - `IN` *subquery* tests if tuple occurs in the result of the subquery
  - `EXISTS` *subquery* tests whether the subquery results in non-empty relation
  - using `ALL`, `SOME` or `ANY` before a subquery makes subqueries usable in comparison formulae
  - in all these cases the condition involving the subquery can be negated using a preceding `NOT`

Subqueries – In

- Recall that, for the q students who have enrolled and their courses", we have:

```
SELECT *
  FROM STUDENT NATURAL JOIN ENROL;
```


- Now if we want to query: "list all students who have enrolled in a course *that has less than 10 students enrolled* and the CourseNo of these courses", we have


```
SELECT s.*,e1.CourseNo
  FROM STUDENT s NATURAL JOIN ENROL e1
 WHERE e1.CourseNo IN
          (SELECT e2.CourseNo
             FROM ENROL e2
         GROUP BY e2.CourseNo
           HAVING COUNT(*)<10);
```

Subqueries – Exists

- For the query: "list all students who have enrolled in at least one course", we have

```
SELECT s.*
  FROM STUDENT s
 WHERE EXISTS (SELECT *
                 FROM ENROL e
                WHERE s.StudentID=e.StudentID);
```


- For the query: "list all students who have *not* enrolled in any course", we have

```
SELECT s.*
  FROM STUDENT s
 WHERE NOT EXISTS (SELECT *
                     FROM ENROL e
                    WHERE s.StudentID=e.StudentID);
```

Subqueries – More Complicated

- For the query: "list th hat have the largest number of students enrolled in Semester e have

```
SELECT e.CourseNo
  FROM (SELECT e1.CourseNo, COUNT(*) AS NoOfStudents
          FROM ENROL e1
         WHERE e1.Semester = '2016 S2'
         GROUP BY e1.CourseNo) e
 WHERE e.NoOfStudents =
           (SELECT MAX(e2.NoOfStudents)
              FROM (SELECT e1.CourseNo, COUNT(*) AS NoOfStudents
                      FROM ENROL e1
                     WHERE e1.Semester = '2016 S2'
                     GROUP BY e1.CourseNo) e2);
```

Subqueries – More Complicated

- For the query: "list all courses that have more students enrolled than at least one other course in Semester 2 2016", we have

```
SELECT e.CourseNo
  FROM (SELECT e1.CourseNo, COUNT(*) AS NoOfStudents
          FROM Enrol e1
         WHERE e1.Semester = '2016 S2'
         GROUP BY e1.CourseNo) e
 WHERE e.NoOfStudents > ANY
           (SELECT e2.NoOfStudents
              FROM (SELECT e1.CourseNo, COUNT(*) AS NoOfStudents
                      FROM Enrol e1
                     WHERE e1.Semester = '2016 S2'
                     GROUP BY e1.CourseNo) e2);
```

Views in SQL

- A view in SQL is a vi[illegible]hat is derived from other tables in the same database or previous[illegible]views.
- How to Create Views?
  - Suppose we already have tables STUDENT(StudentID, Name, DoB, Email) and ENROL(StudentID, CourseNo, Semester, Status, EnrolDate). Then we can create a view ENROL1 as follows:


```
CREATE VIEW ENROL1
AS SELECT s.StudentID, s.Name, e.CourseNo, e.EnrolDate
FROM STUDENT s, ENROL e
WHERE s.StudentID=e.StudentID;
```