Seminar 2. SQL Queries – DML Subset

UNION: Can be used to compute the union of any two union-compatible sets of tuples (which are themselves the result of SQL queries). Duplicate rows are eliminated.

Example: *Find sid of students with grades at courses with 4 or 5 credits*

```
SELECT E.sid
FROM  Enrolled E, Courses C
WHERE E.cid=C.cid
  AND C.credits=4
UNION
SELECT E.sid
FROM  Enrolled E, Courses C
WHERE E.cid=C.cid
  AND C.credits=5
```

Alternative:

```
SELECT E.sid
FROM  Enrolled E, Courses C
WHERE E.cid=C.cid
  AND (C.credits=4 OR
       C.credits=5)
```

In this version, duplicates are not eliminated.

If we replace OR by AND in this version, what do we get?

INTERSECT: can be used to compute the intersection of any two union-compatible sets of tuples. Included in the SQL-92 standard, but some systems don't support it.

Example: *Find sid of students with grades at both a 4 credits course and a 5 credits course*

```
SELECT  E.sid
FROM  Courses C, Enrolled E
WHERE E.cid=C.cid
   AND C.credits=4
INTERSECT
SELECT  E.sid
FROM  Courses C, Enrolled E
WHERE E.cid=C.cid
```

```
AND C.credits=5
```

Alternative:

```
SELECT  E1.sid
FROM  Courses C1, Enrolled E1,
      Courses C2, Enrolled E2
WHERE E1.sid=E2.sid AND E1.cid=C1.cid AND
     E2.cid=C2.cid AND
     (C1.credits=4 AND C2.credits=5)
```

Also available: ***EXCEPT*** statement, used to obtain all the records belonging to the first set of tuples which are not part of the second set of tuples (e.g., replace UNION with EXCEPT in the UNION query above).

Nested Queries

A very powerful feature of SQL: a WHERE clause can itself contain an SQL query! (Actually, so can FROM and HAVING clauses.)

Sample: *Find names of students who're enrolled at course 'Alg1'*

```
SELECT S.name
FROM Students S
WHERE S.sid IN (SELECT  E.sid
           FROM  Enrolled E
           WHERE  E.cid='Alg1')
```

To understand semantics of nested queries, think of a nested loops evaluation: For each Students tuple, check the qualification by computing the subquery.

Sample: *Find names of students who're enrolled at course 'Alg1'*

```
SELECT S.name
FROM Students S
WHERE EXISTS (SELECT *
           FROM Enrolled E
           WHERE E.sid=S.sid
             AND E.cid='Alg1')
```

EXISTS is another set comparison operator, like IN.

The above example illustrates why, in general, subquery must be re-computed for each *Students* tuple.

Besides IN and EXISTS, we can also use NOT IN or NOT EXISTS. There are also available:

- *operator ANY* (the value is true if the condition is true for **at least one** item of the sub-query result)
- *operator ALL*(the value is true if the condition is true for **all** the items of the sub-query result)

Sample: *Find students whose age is greater than that of some student called 'Joe'*:

```
SELECT  *
FROM  Students S
WHERE  S.age > ANY (SELECT S2.age
                    FROM  Students S2
                    WHERE S2.name='Joe')
```

Rewrite INTERSECT queries using IN:

Find *sid* of students with grades at both a 4 credits course <u>and</u> a 5 credits course:

```
SELECT  E.sid
FROM  Enrolled E, Courses C
WHERE  E.cid=C.cid AND C.credits = 4
       AND E.sid IN (SELECT  E2.sid
                     FROM Enrolled E2,Courses C2
                     WHERE E2.cid=C2.cid AND
                           C2.credits=5)
```

Similarly, EXCEPT queries can be re-written using NOT IN.