Seminar 1. SQL Queries – DML Subset

SELECT statement

A very simple example of using SELECT statement is:

```
SELECT  *
FROM  Students S
WHERE  S.age = 21
```

which returns all 21 year old students form ***Students*** table:

1234	John	j@cs.ro	21	331
1236	Anne	a@cs.ro	21	332

To find just names and email addresses, we should replace the first line with:

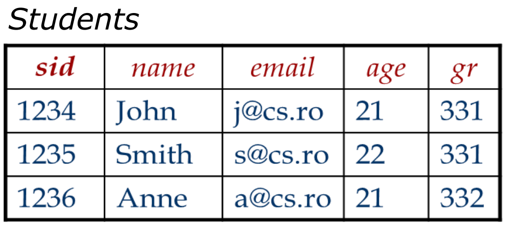
```
SELECT S.name, S.email
```

John	j@cs.ro
Anne	a@cs.ro

What does the following query compute?

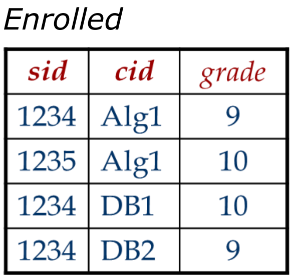
```
SELECT S.name, E.cid
FROM   Students S, Enrolled E
WHERE  S.sid=E.sid AND E.grade=10
```

Given the following instances of *Students* and *Enrolled*



Students

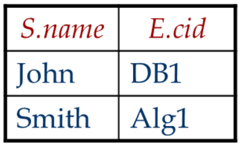
sid	*name*	*email*	*age*	*gr*
1234	John	j@cs.ro	21	331
1235	Smith	s@cs.ro	22	331
1236	Anne	a@cs.ro	21	332



Enrolled

sid	***cid***	*grade*
1234	Alg1	9
1235	Alg1	10
1234	DB1	10
1234	DB2	9

We get:



S.name	*E.cid*
John	DB1
Smith	Alg1

Semantics of a query:

A *conceptual evaluation method* for the previous query:

1. FROM clause: Compute *cross-product* of Students and Enrolled (12 tuples)

2. WHERE clause: Check conditions, discard tuples that fail (4 tuples meet the 1st condition, 6 tuples meet the 2nd one; the `S.sid=E.sid AND E.grade=10` condition is met by 2 tuples)

3. SELECT clause: Delete unwanted fields (only S.name and E.cid remain in the result set)

Remember, this is *conceptual*. Actual evaluation will be *much* more efficient, but must produce the same answers.

A very simple SQL query looks like:

```
SELECT [DISTINCT]target-list
FROM relation-list
WHERE qualification
```

where:

- ***relation-list*** is a list of relation names (possibly with a range-variable after each name);
- ***target-list*** is a list of attributes of relations in relation-list;
- ***qualification*** contains logical expressions having comparisons (Attr op const or Attr1 op Attr2, where op is one of <, >, =, ≤, ≥, ≠) combined with logical operators AND, OR and NOT;
- ***DISTINCT*** is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are not eliminated!

The semantics of an SQL query is defined in terms of the following conceptual evaluation strategy:

- Compute the cross-product of relation-list;
- Discard resulting tuples if they fail qualifications;
- Delete attributes that are not in target-list;
- If DISTINCT is specified, eliminate duplicate rows.

Range variables are really needed only if the same relation appears twice in the FROM clause.

It is good style, however, to always use range variables.

So, we can write the same query in two distinct ways:

```
SELECT S.name, E.cid
FROM   Students S, Enrolled E
WHERE  S.sid=E.sid AND E.grade=10
```

or

```
SELECT name, cid
FROM   Students, Enrolled
WHERE  Students.sid=Enrolled.sid
       AND grade=10
```

Find students with at least one grade:

```
SELECT S.sid
FROM   Students S, Enrolled E
WHERE  S.sid=E.sid
```

Would adding DISTINCT to this query make a difference?

What is the effect of replacing *S.sid* by *S.name* in the SELECT clause? Would adding DISTINCT to this variant of the query make a difference?

The following query illustrates the use of arithmetic expressions and string pattern matching: *Find triples (of ages of students and two fields defined by expressions) for students whose names begin and end with B and contain at least three characters.*

```
SELECT S.age, age1=S.age-5, 2*S.age AS age2
FROM   Students S
WHERE  S.name LIKE 'B_%B'
```

Observations:

- Note that AS and = are two ways to name fields in result.
- LIKE operator is used for string matching.
- `_' stands for any one character
- `%' stands for 0 or more arbitrary characters.

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:

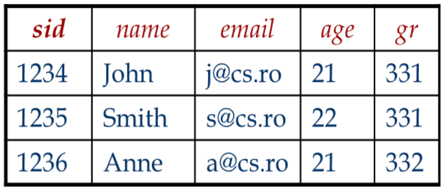
Find *sid* of students with grades at both a 4 credits course and a 5 credits course:

```
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.

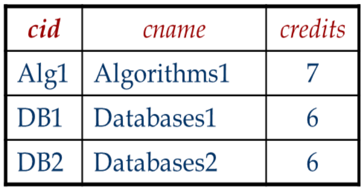
Join Queries

Students



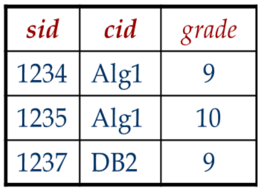
sid	*name*	*email*	*age*	*gr*
1234	John	j@cs.ro	21	331
1235	Smith	s@cs.ro	22	331
1236	Anne	a@cs.ro	21	332

Courses



cid	*cname*	*credits*
Alg1	Algorithms1	7
DB1	Databases1	6
DB2	Databases2	6

Enrolled



sid	**cid**	*grade*
1234	Alg1	9
1235	Alg1	10
1237	DB2	9

<table>
<tr><th>Join variant</th><th>Sample query</th><th>Result</th></tr>
<tr><td><b>INNER JOIN</b></td><td>SELECT S.name, C.cname
FROM Students S
INNER JOIN Enrolled E ON S.sid = E.sid
INNER JOIN Courses C ON E.cid = C.cid</td><td><table><tr><th><i>name</i></th><th><i>cname</i></th></tr><tr><td>John</td><td>Algorithms1</td></tr><tr><td>Smith</td><td>Algorithms1</td></tr></table></td></tr>
<tr><td><b>LEFT OUTER JOIN</b>
(e.g., find students who never enrolled to a course)</td><td>SELECT S.name, C.cname
FROM Students S
LEFT OUTER JOIN Enrolled E
ON S.sid = E.sid
LEFT OUTER JOIN Courses C
ON E.cid = C.cid</td><td><table><tr><th><i>name</i></th><th><i>cname</i></th></tr><tr><td>John</td><td>Algorithms1</td></tr><tr><td>Smith</td><td>Algorithms1</td></tr><tr><td>Anne</td><td><b>NULL</b></td></tr></table></td></tr>
<tr><td><b>RIGHT OUTER JOIN</b>
(e.g., find all grades given by mistake to non-existing students)</td><td>SELECT S.name, C.cname
FROM Students S
RIGHT OUTER JOIN Enrolled E
ON S.sid = E.sid
INNER JOIN Courses C
ON E.cid = C.cid</td><td><table><tr><th><i>name</i></th><th><i>cname</i></th></tr><tr><td>John</td><td>Algorithms1</td></tr><tr><td>Smith</td><td>Algorithms1</td></tr><tr><td><b>NULL</b></td><td>Databases2</td></tr></table></td></tr>
<tr><td><b>FULL OUTER JOIN</b>
(LEFT + RIGHT OUTER JOIN)</td><td>SELECT S.name, C.cname
FROM Students S
FULL OUTER JOIN Enrolled E
ON S.sid = E.sid
FULL OUTER JOIN Courses C
ON E.cid = C.cid</td><td><table><tr><th><i>name</i></th><th><i>cname</i></th></tr><tr><td>John</td><td>Algorithms1</td></tr><tr><td>Smith</td><td>Algorithms1</td></tr><tr><td><b>NULL</b></td><td>Databases2</td></tr><tr><td><b>NULL</b></td><td>Databases1</td></tr><tr><td>Anne</td><td><b>NULL</b></td></tr></table></td></tr>
</table>

NULL values

Field values in a tuple are sometimes *unknown* (e.g., a rating has not been assigned) or *inapplicable*. SQL provides a special value <u>*null*</u> for such situations.

The presence of *null* complicates many issues. E.g.:

- Special operators are needed to check if a value is/is not *null*.
- Is *rating>8* true or false when *rating* is equal to *null*? What about AND, OR and NOT connectives?

Solution: we need a <u>3-valued logic</u> (**true**, **false** and ***unknown***). Meaning of constructs must be defined carefully (e.g., WHERE clause eliminates rows that don't evaluate to true.). New operators (in particular *outer joins*) are possible/needed.

Aggregate Operators

Most used aggregate operators are (A is a table field name):

```
-   COUNT (*)
-   COUNT ( [DISTINCT] A)
-   SUM ( [DISTINCT] A)
-   AVG ( [DISTINCT] A)
-   MAX (A)
-   MIN (A)
```

Sample: *Get the total number of students*

```
SELECT  COUNT (*)
FROM  Students S
```

Sample: *Get age average of group 311*

```
SELECT  AVG (S.age)
FROM  Students S
WHERE  S.gr=311
```

Sample: *Find how many groups have assigned at least one student named Bob*

```
SELECT COUNT (DISTINCT S.gr)
FROM  Students S
WHERE S.name=‘Bob’
```

Sample: *Find the names of oldest students*

```
SELECT S.name
FROM Students S
WHERE S.age = ANY
      (SELECT MAX(S2.age)
       FROM  Students S2)
```