Querying Multiple Relations

- What does the following query compute?

join

```
SELECT  S.name, E.cid
FROM  students S, enrolledIn E
WHERE  S.sid=E.sid AND E.grade= 'A'
```

students

sid	name	login	age	gpa
666	Jones	jones@cs	18	3.4
688	Smith	smith@eecs	18	3.2
650	Smith	smith@math	19	3.8

enrolledIn

sid	cid	grade
831	Carnatic101	C
831	Reggae203	B
650	Topology112	A
666	History105	B

Answer:

S.name	E.cid
Smith	Topology112

SQL Query Evaluation Specification - 2

» Compute the cross-product of *relation-list in FROM clause*.

» Discard resulting tuples if they fail *qualifications in WHERE clause*.

» Delete attributes that are not in *target-list in SELECT clause*.

- *This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute the same answers.*

Expressions and Strings

"Find pairs (of ages of students and a field defined by an expression) for students whose names begin and end with B and contain at least four characters."

student(Sid,Name,Gpa,Age)
course(Cid, Title, Dept)
enrolledIn(Sid,Cid,Grade)

```
SELECT  S.name, S.age-5 AS youngerAge
FROM  students S
WHERE  S.name LIKE 'B_%_B'
```

- Illustrates use of arithmetic expressions and string pattern matching: AS is a way to name new fields in result.
- LIKE is used for string matching. '_' stands for any one character and '%' stands for 0 or more arbitrary characters.

Set Operations on Relations: ***Union***

"Find names of students who've enrolled in a CS or a Math course"

- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).
- Union-compatible: same # of cols, labeled the same way

```
SELECT  S.name
FROM enrolledIn E, students S, courses C
WHERE  S.sid=E.sid AND E.cid=C .cid
  AND (C .dept='cs'  OR C .dept='math')
```

is also:

```
SELECT  S.name
FROM enrolledIn E, students S, courses C
WHERE  S.sid=E.sid AND E.cid=C .cid
          AND C .dept='cs'
UNION
SELECT  S.name
FROM enrolledIn E, students S, courses C
WHERE  S.sid=E.sid AND E.cid=C .cid
          AND C .dept='math'
```

student(Sid,Name,Gpa,Age)
course(Cid, Title, Dept)
enrolledIn(Sid,Cid,Grade)

Set Operations on Relations: ***Difference***

"Find names of students enrolled in 101 but not 103"

- EXCEPT (like algebra operator MINUS)

```
SELECT s.name
FROM students s, enrolledin e
WHERE s.sid=e.sid AND e.cid='160:101'
AND s.name not in(
   SELECT s.name
   FROM students s, enrolledin e
   WHERE s.sid=e.sid AND e.cid='640:103'
);
```

The EXCEPT operator is not supported in mySQL, but the same effect can be obtained from the syntax given above

Set Operations on Relations: Difference

"Find names of students enrolled in 101 but not 103"

```
SELECT s.name
FROM students s, enrolledin e
WHERE s.sid=e.sid AND e.cid='160:101'
AND s.name not in(
   SELECT s.name
   FROM students s, enrolledin e
   WHERE s.sid=e.sid AND e.cid='640:103');
```

Are these equivalent to the above?

≡

```
SELECT  S.name
FROM  students S, enrolledIn E
WHERE  S.sid=E.sid AND E.cid=101
        AND NOT (E.cid=103)
```

```
SELECT  S.name
FROM  students S, enrolledIn E
WHERE  S.sid=E.sid AND E.cid=101
        AND E.cid < > 103
```

not the desired query for the English sentence at the top.

"Find sid's of students who've enrolled in a cs <u>and</u> a math course"

Key field!

```
SELECT  E.sid
FROM   enrolledIn E, courses C
WHERE  E.cid=C.cid
            AND C.dept= 'cs '
INTERSECT
SELECT  E.sid
FROM  enrolledIn E, courses C
WHERE   E.cid=C.cid
            AND C.dept= 'math '
```

- 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.)
- INTERSECT automatically removes duplicates! UNION does not (must use DISTINCT keyword in SELECT clause)

(This is less clear but possibly more efficient:

```
SELECT  E1.sid
FROM  courses C1, enrolledIn E1,
          courses C2, enrolledIn E2
WHERE  E1.sid=E2.sid
  AND (E1.cid=C1.cid  AND  C1.dept= 'cs '  )
  AND (E2.cid=C2.cid  AND C2.dept= 'math ' )
```

Nested Queries: motivation

"Find names of students who have not enrolled in course #103

student(Sid,Name,Gpa,Age)
course(Cid, Title, Dept)
enrolledIn(Sid,Cid,Grade)

Is this ok?

```
SELECT  S.name
FROM  students S, enrolledIn E
WHERE  S.sid < > E.sid and E.cid=103)
```

Nested Queries

*"Find names of students **not** enrolled in course #103"*

student(Sid,Name,Gpa,Age)
course(Cid, Title, Dept)
enrolledIn(Sid,Cid,Grade)

(the right way)

```
SELECT  S.name
FROM  students S
WHERE  S.sid NOT IN  (SELECT  E.sid
                      FROM  enrolledIn E
                      WHERE  E.cid=103)
```

- powerful feature of SQL: a WHERE clause can itself contain an SQL query!
- To understand semantics of nested queries, think of a *nested for-loop* evaluation: *For each* students *tuple, check the qualification by computing the subquery.*

Remember that in our class exercises we used '640:103' instead of just 103

Correlated Nested Queries

"Find names of students not enrolled in course #103"

(alternate version)

students(Sid,Name,Gpa,Age)
course(Cid, Title, Dept)
enrolledIn(Sid,Cid,Grade)

```
SELECT  S.name
FROM  students S
WHERE  NOT EXISTS  (SELECT  *
                    FROM  enrolledIn E
                    WHERE  E.cid=103 AND S.sid=E.sid)
```

- EXISTS is another set operator, like IN. (Both can be preceded by NOT)
- Shows why, in general, the subquery **must** be re-computed for each *students* tuple.

More on Set-Comparison Operators

student(Sid,Name,Gpa,Age)
course(Cid, Title, Dept)
enrolledIn(Sid,Cid,Grade)

- Also available: *op* ANY, *op* ALL

  [*op* SOME is same as op ANY but less ambiguous IMO)

*"Find students whose gpa is greater than that of **someone** called Horatio":*

```
SELECT *
FROM students S
WHERE S.gpa > ANY (SELECT S2.gpa
                   FROM students S2
                   WHERE S2.name='Horatio')
```