More on Set-Comparison Operators

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

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

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

Aggregate Operators

Find the name of those students with a maximum gpa

- *Significant extension of FOL.*

```
COUNT (*)
COUNT (A)
SUM ( A)
AVG ( A)
MAX (A)
MIN (A)
```

single column

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

```
SELECT  AVG (S.age)
FROM  students S
WHERE  S.gpa=10
```

```
SELECT  S.name
FROM  students S
WHERE  S.gpa= (SELECT  MAX(S2.gpa)
                FROM  students S2)
```

GROUP BY

"Find the age of the youngest student for each gpa level'

- In general, we don't know how many gpa levels exist, and what the rating values for these levels are!
- (Even if we did know that gpa values go from 1 to 10, we would have to write 10 (!) queries that look like this :

For $i = 1, 2, \ldots, 10$:

```
SELECT  MIN (S.age)
FROM  students S
WHERE  S.gpa = i
```

- Instead, use GROUP BY:

```
SELECT  MIN (S.age)
FROM  students S
GROUP BY  S.gpa
```

"For each school number, find the age of the youngest student"

```
SELECT  S.school#,  MIN (S.age)
FROM  students S
GROUP BY  S.school#
```

sid	name	school#	age
22	dustin	7	45
31	lubber	8	55
71	zorba	10	16
64	horatio	7	35
29	brutus	1	33
58	rusty	10	35

school#	age
1	33
7	45
7	35
8	55
10	16
10	35

school#	age
1	33
7	35
8	55
10	16

Answer

GROUP BY and HAVING

- So GROUP BY creates in some sense sub-tables, after the WHERE filters out some rows

```
SELECT  MIN (S.age)
FROM  students S
WHERE ...
GROUP BY  S.school#
```

- What if we want to eliminate some of the sub-tables, in turn?
  - » e.g., don't want to see min-ages if there is only one person of that school?

HAVING-clause is evaluated for each group/sub-table.

```
SELECT  MIN (S.age)
FROM  students S
GROUP BY  S.school#
HAVING count(*)>=2
```

"For each school number, find the age and name of the youngest student"

```
SELECT  S.school#, S.name,
        MIN (S.age)
FROM  students S
GROUP BY  S.school#
```

- Is the above meaningful?

Which name would you choose for group 7: dustin or horatio?

sid	name	school#	age
22	dustin	7	45
31	lubber	8	55
71	zorba	10	16
64	horatio	7	35
29	brutus	1	33
58	rusty	10	35

school#	age
1	33
7	45
7	35
8	55
10	16
10	35

school#	age
1	33
7	35
8	55
10	16

Answer

Full syntax of queries With GROUP BY and HAVING

```
SELECT      [DISTINCT] target-list
FROM        relation-list
WHERE       qualification
GROUP BY    grouping-list
HAVING      group-qualification
```

- The *target-list* can contain only (i) permissible attribute names and (ii) terms with aggregate ops (e.g., MIN (*S.age*)).
  - The only permissible attribute names are the ones in the *grouping-list*. **(Intuitively, each answer tuple corresponds to a *group*, and these attributes must have a single value per group. In SQL99, primary keys also allowed if relation list is singleton.)**
  - *group-qualification* can also only be applied to *permissible attribute* names or terms with aggregate operation.

"Find name and age of the oldest student(s)"

- The first query is illegal (What if you said SUM(S.age)?)
- In SQL, if *some* aggregate is used in SELECT then *all columns* appearing in SELECT must be aggregates. (or GROUPed BY- see next)
- If you really wanted to see every name paired with the largest age, use this

```
SELECT  S.name, MAX (S.age)
FROM  students S
```

Nested query in FROM !!

```
SELECT  S.name, Temp.maxage
FROM  students S,
      (SELECT  MAX (S2.age) AS maxage
      FROM  students S2) AS Temp
--WHERE true
```

BUT this is not what we wanted to do...

"Find name and age of the oldest student(s)"

```
SELECT name,
              age
FROM    students
WHERE age=(select max(age)
                    from students);
```

Notice that it is possible not to use alias in the names of the tables or the names of the attributes (fields) as long as there is no ambiguity.

We can use this kind of subquery when the value returned by it is unique. Otherwise we could use IN or NOT IN instead of =

"Find the gpa of the youngest student with age >= 18, for each age group with at least 2 such students"

```
SELECT  S.gpa,  MIN (S.age)
FROM  students S
WHERE  S.age >= 18
GROUP BY  S.gpa
HAVING  COUNT (*) > 1
```

sid	name	gpa	age
22	dustin	7	45
31	lubber	8	55
71	zorba	10	16
64	horatio	7	35
29	brutus	1	33
58	rusty	10	35

- Only S.gpa and S.age are mentioned in the SELECT, GROUP BY or HAVING clauses; other attributes `*unnecessary*'.
- 2nd column of result is unnamed. (Use AS to name it.)

gpa	age
1	33
7	45
7	35
8	55
10	35

rating	
7	35

Answer Relation

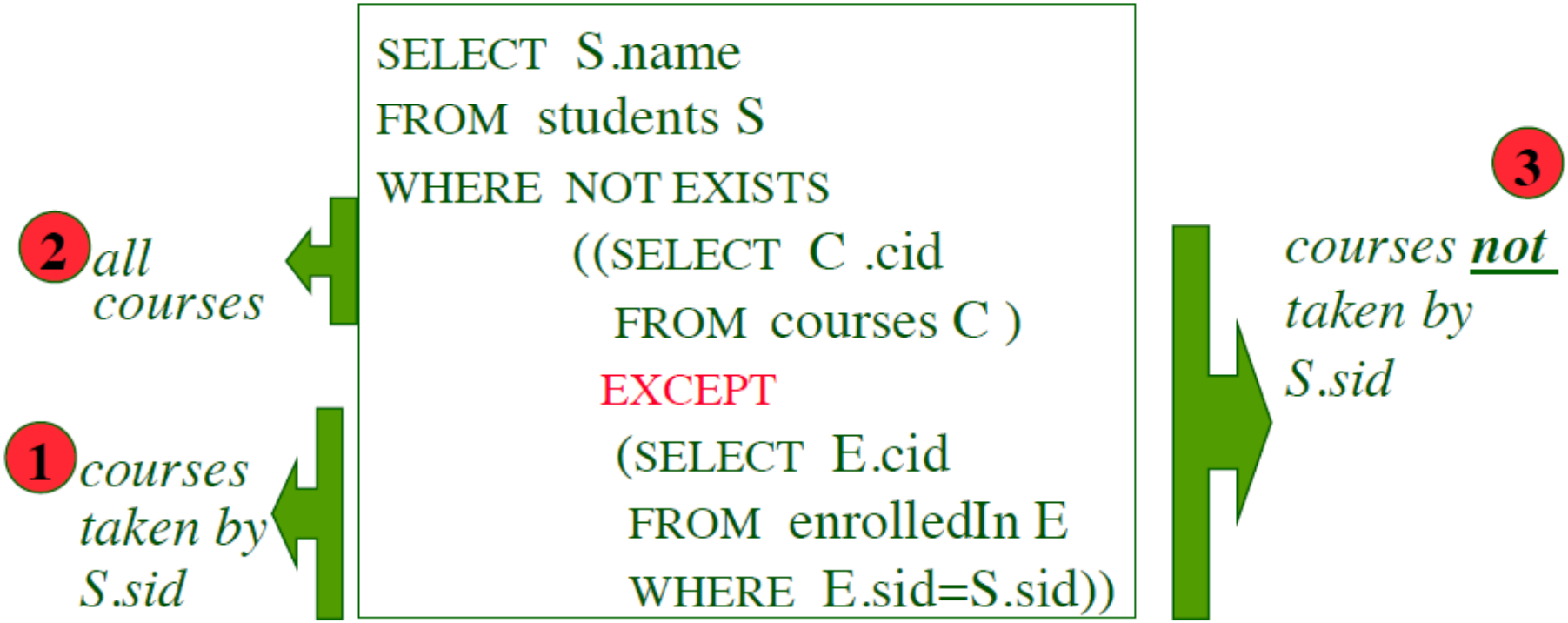
summary:Conceptual Evaluation of full SQL

- The cross-product of *relation-list* is computed; tuples that fail *qualification* are discarded; `*unnecessary*' fields (ones not in *grouping/target-list* or arguments of aggregate functions in *target/group-qual list*) are deleted; the remaining tuples are partitioned into groups by the value of attributes in *grouping-list*.
- The *group-qualification* is then applied to eliminate some groups. (Expressions in *group-qualification* must have a *single value per group*; determined statically: each attribute in *group-qualification* that is not an argument of an aggregate op appears in *grouping-list*.)
- One answer tuple is generated per qualifying group.

Expressing universal quantifiers in SQL

"Find students who've enrolled in all courses."

= *"Find students who **don't** have any courses that they **haven't** enrolled*



- *What does the query compute if you remove the "WHERE E.sid=S.sid" clause?*

Efficiency & Clarity of Queries

- These may conflict.
- In the following example there is no debate:

Rewriting EXCEPT aueries

"students who took cs but not math courses"

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

Much less clear **and** less efficient because of correlated nested query.

```
SELECT  E.sid
FROM  enrolledIn E, courses C
WHERE E.cid=C .cid AND C .dept= 'cs '
        AND NOT EXISTS(
                SELECT  E2.sid
                FROM  courses C 2, enrolledIn E2
                WHERE  E2.cid=C 2.cid
                 AND C 2.dept= 'math '
                 AND E.sid=E2.sid )
```

Duplicates in SQL queries

SQL, unlike the pure relational model, allows duplicate rows in a query answer (not in stored relations). This is for efficiency reasons – removing duplicates requires sorting the entire answer set, which may be enormous.

SQL allows putting the keyword DISTINCT after SELECT attribute name in order to remove duplicate tuples.

"Find students who've enrolled in at least one course"

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

```
SELECT  E.sid
FROM  enrolledIn E
[ WHERE  true ]
```

- Would writing SELECT DISTINCT E.sid make a difference?
- What if we wanted to get student names?

```
SELECT  S.name
FROM  enrolledIn E, student S
        WHERE  S.sid = E.sid
```

- Would writing SELECT DISTINCT S.name make a difference?

Aggregate operators have optional DISTINCT

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

single column

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

```
SELECT  AVG (S.age)
FROM  students S
WHERE  S.gpa=10
```

```
SELECT  COUNT (DISTINCT S.gpa)
FROM  students S
WHERE S.name= 'Bob'
```

```
SELECT  AVG ( DISTINCT S.age)
FROM  students S
WHERE  S.gpa=10
```

* *"Find average age of students enrolled in courses in which they got an A"* *(optional)*

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

```
SELECT  avg(S.age)
FROM  students S, enrolledIn E
WHERE  S.sid = E.sid AND  E.grade='A'
```

```
SELECT  avg(S.age)
FROM  students S
WHERE  S.sid IN (SELECT  E.sid
                 FROM  enrolledIn E
                 WHERE  E.grade='A')
```

The second query gives each student once. The first would repeat the student every time they got an 'A'. If you now wanted to find the average age of students who got an 'A', you are in trouble with the first query (putting DISTINCT is bad because likely there are different students with same age yet these would be collapsed).

*Uniqueness conditions with set operations (optional)

- Unlike standard SELECT queries, which use bag/ multiset semantics, INTERSECT/ EXCEPT/UNION use set semantics, eliminating duplicates from answers, as the operations are applied. (To counter this, say UNION ALL
- Why? Efficiency dictates language semantics :-(
  - » To eliminate duplicates in SELECT..., would have to do a lot of work
  - » Intersection/difference are implemented by sort+merge, which makes duplicate elimination cheap

Using nested queries in FROM clause

"Find average age of students enrolled in course in which they got an A":

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

```
SELECT  avg(S.age)
FROM  students S
WHERE   S.sid IN (SELECT  E.sid
                  FROM  enrolledIn E
                  WHERE  E.grade='A')
```

```
SELECT  avg(S.age)
FROM  students S, (SELECT  E.sid  FROM  enrolledIn E
                   WHERE E.grade='A') AS A_enrolled
WHERE   S.sid IN A_enrolled
```

JOIN operations in FROM clause

"Find names of students enrolled in 103"

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

```
SELECT  R.name
FROM  (students   JOIN enrolledIn  USING(sid)) AS R
WHERE  R.cid=103
```

```
SELECT  S.name
FROM  students S JOIN enrolledIn E ON S.sid = E.sid
WHERE  E.cid=103
```

?mySQL requires naming in FROM?

```
SELECT  S.name
FROM  students S JOIN enrolledIn E
        ON (S.sid = E.sid AND E.cid=103)
```

JOIN operations in FROM clause (2)

"Find names of students who did not fail in 103"

```
SELECT  S.name
FROM  students S, enrolledIn E
WHERE  S.sid=E.sid AND E.cid=103 AND
E.grade<> 'F';
```

But this query omits students who did not even take 103! (Maybe these are wanted!)

```
SELECT  S.name
FROM  students S LEFT OUTER JOIN enrolledIn E ON
S.sid = E.sid AND E.grade<>'F'
WHERE  E.cid=103
```

GROUP BY issues

"For each 'cs' course, find the number of enrolments in that course"

```
SELECT  C.cid,  COUNT (*)
FROM  courses C, enrolledIn E,
WHERE  C.dept='cs' AND E.cid=C.cid
GROUP BY  C.cid
```

- *What happens to courses not taken by anyone?*
- *What do you get if we remove* C.dept='cs' *from the* WHERE *clause and add a clause*

  HAVING C.dept='cs'

  - (Answer: even though in each group C.dept will be the same because C.cid is a key, this is illegal because *dept* is not GROUPed BY. So add *C.dept* to GROUP BY)

* *"Find the age and gpa of the youngest adult student for each gpa with at least 3 students (of any age) having it."*

```
SELECT  S.gpa,  MIN (S.age)
FROM  students S
WHERE S.age > 18
GROUP BY  S.gpa
HAVING  2  <  (SELECT  COUNT (*)
                FROM  students S2
                WHERE  S.gpa=S2.gpa)
```

- Shows HAVING clause can also contain a subquery.
- Note: this is not the same as HAVING clause being replaced by

  HAVING 2 < COUNT(*)