CS460: Intro to Database Systems

Class 9: SQL, The Query Language – Part II

Instructor: Manos Athanassoulis

https://bu-disc.github.io/CS460/

Recap: Basic SQL Query

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

relation-list : a list of relations

target-list : a list of attributes of tables in *relation-list*

qualification : comparisons using AND, OR and NOT

comparisons are: <attr> <*op*> <const> or <attr1> <*op*> <attr2>, where *op* is:

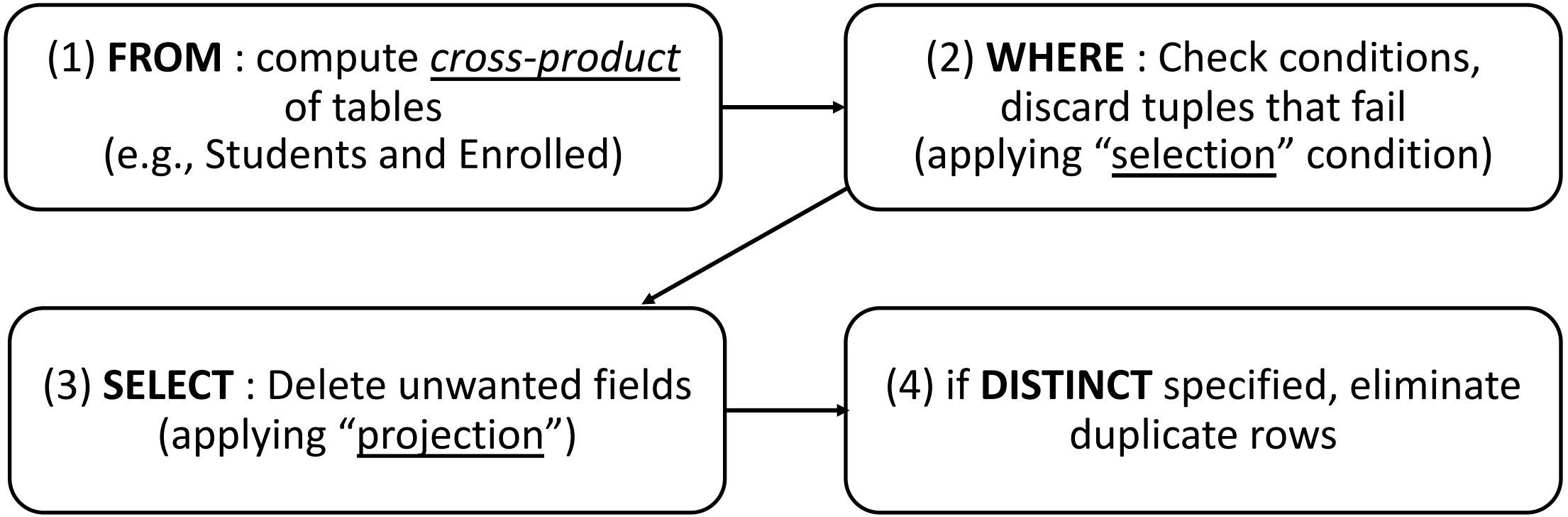
$$<, >, =, \leq, \geq, \neq$$

DISTINCT: *optional,* removes duplicates

By default SQL SELECT does *not* eliminate duplicates! ("multiset")

Recap: Query Semantics

Conceptually, a SQL query can be computed:



probably the least efficient way to compute a query!

Query Optimization finds the *same answer* more efficiently

Recap: Range Variables

```
SELECT sname
FROM Sailors,Reserves
WHERE Sailors.sid=Reserves.sid AND bid=103
```

can be rewritten using range variables as:

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND bid=103
```

Can use Range Variables – do not need though. Why?

Recap: Expressions

Use AS to provide column names

```
SELECT S.age, S.age-5 AS age1, 2*S.age AS age2
FROM  Sailors S
WHERE  S.sname = 'dustin'
```

Can also have expressions in WHERE clause:

```
SELECT  S1.sname AS name1, S2.sname AS name2
FROM  Sailors S1, Sailors S2
WHERE  2*S1.rating = S2.rating - 1
```

Recap: String operations

SQL also supports some string operations

"LIKE" is used for string matching.

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

'_' stands for any one character

'%' stands for 0 or more arbitrary characters

>, < string comparison is supported by most systems

Recap: Nested Queries

WHERE clause can itself contain an SQL query!

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid IN (SELECT R.sid
                 FROM  Reserves R
                 WHERE  R.bid=103)
```

Recap: Nested Queries with Correlation

Subquery must be recomputed for each Sailors tuple.

Think of subquery as a function call that runs a query!

```
SELECT  S.sname
FROM  Sailors S
WHERE EXISTS (SELECT  *
              FROM  Reserves R
              WHERE R.bid=103 AND S.sid=R.sid)
```

Recap: Set Operations

```
SELECT R.sid
FROM   Boats B, Reserves R
WHERE  R.bid=B.bid
       AND B.color='red'
UNION
SELECT R.sid
FROM   Boats B, Reserves R
WHERE  R.bid=B.bid
       AND B.color='green'
```

```
SELECT S.sid
FROM Sailors S, Boats B,
     Reserves R
WHERE S.sid=R.sid
      AND R.bid=B.bid
      AND B.color='red'
INTERSECT
SELECT S.sid
FROM Sailors S, Boats B,
     Reserves R
WHERE S.sid=R.sid
     AND R.bid=B.bid
     AND B.color='green'
```

Recap: ANY and ALL Set-Comparison Operators

Find sailors with rating greater than the rating of at least one sailor called ‘Horatio’:

```
SELECT  *
FROM  Sailors S
WHERE  S.rating > ANY (SELECT  S2.rating
                       FROM  Sailors S2
                      WHERE S2.sname=‘Horatio’)
```

Find sailors with rating greater than the rating of all 20-year old sailors:

```
SELECT  *
FROM  Sailors S
WHERE  S.rating > ALL (SELECT  S2.rating
                       FROM  Sailors S2
                      WHERE S2.age = 20)
```

Recap: Set-Difference using NOT IN

Find all sailors who have not reserved a red boat

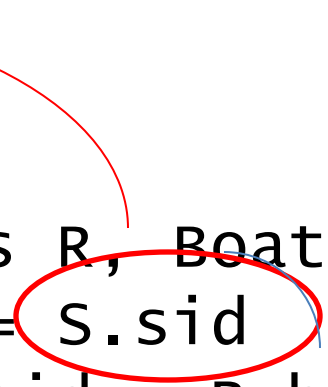
```
SELECT S.sid
FROM   Sailors S
WHERE  S.sid NOT IN
           (SELECT R.sid
            FROM Reserves R, Boats B
            WHERE R.bid = B.bid
                  AND B.color = ‘red’)
```

Nested – NO correlation!

Recap: Set-Difference using NOT EXISTS

Find all sailors who have not reserved a red boat

```
SELECT S.sid
FROM   Sailors S
WHERE  NOT EXISTS
           (SELECT *
            FROM Reserves R, Boats B
            WHERE R.sid = S.sid
                  AND R.bid = B.bid
                  AND B.color = 'red')
```



Nested – correlation!

Aggregate Operators

Significant extension of relational algebra.

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

single column

```
SELECT  COUNT (*)
FROM  Sailors S

SELECT  AVG (S.age)
FROM  Sailors S
WHERE  S.rating=10

SELECT  COUNT (DISTINCT S.rating)
FROM  Sailors S
WHERE S.sname='Bob'
```

Find name and age of the oldest sailor(s)

The first query is incorrect!

```
SELECT  S.sname, MAX (S.age)
FROM  Sailors S
```

Third query equivalent to second query

allowed in SQL/92 standard, but not supported in some systems.

```
SELECT  S.sname, S.age
FROM  Sailors S
WHERE  S.age =
      (SELECT  MAX (S2.age)
       FROM    Sailors S2)
```

```
SELECT  S.sname, S.age
FROM  Sailors S
WHERE  (SELECT  MAX (S2.age)
        FROM    Sailors S2)
               = S.age
```

ARGMAX?

The Sailor with the highest rating

What about ties for highest?

```
SELECT *
FROM   Sailors S
WHERE  S.rating >= ALL
   (SELECT  S2.rating
    FROM    Sailors S2)
```

```
SELECT *
FROM   Sailors S
WHERE  S.rating =
(SELECT  MAX(S2.rating)
   FROM  Sailors S2)
```

```
SELECT *
FROM   Sailors S
ORDER BY rating DESC
LIMIT 1;
```

Division in SQL

Find sailors who have reserved all boats.

Sailors S for which …

```
SELECT S.sname
FROM   Sailors S
WHERE  NOT EXISTS  (SELECT  B.bid
                    FROM  Boats B
                    WHERE  NOT EXISTS (SELECT  R.bid
                                       FROM  Reserves R
                                       WHERE  R.bid=B.bid
                                       AND R.sid=S.sid))
```

there is no boat B without …

a Reserves tuple showing S reserved B

SQL DDL

Recap: SQL DDL

```
CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid) REFERENCES Students )
```

SQL DDL – General Constraints

```
CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid) REFERENCES Students,
   CHECK grade LIKE ‘A’ OR grade LIKE ‘B’
         OR grade LIKE ‘C’ OR grade LIKE ‘D’)
```

SQL DDL – General Constraints

```
CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid) REFERENCES Students,
   CONSTRAINT checkGrade
   CHECK (grade LIKE 'A' OR grade LIKE 'B'
          OR grade LIKE 'C' OR grade LIKE 'D')
```

SQL DDL – General Constraints

```
CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid) REFERENCES Students,
   CONSTRAINT checkNumber
   CHECK ( (SELECT COUNT (sid) FROM Students)
                +
           (SELECT COUNT DISTINCT (cid) FROM Enrolled)
                < 1000 )
```

JOINS

Joins

```
SELECT (column_list)
FROM  table_name
 [INNER | NATURAL | {LEFT | RIGHT | FULL} | {OUTER}]
JOIN table_name
    ON qualification_list
WHERE …
```

INNER is default

SELECT sname FROM sailors S JOIN reserves R ON S.sid=R.sid;

SELECT sname FROM sailors S NATURAL JOIN reserves R

WHERE R.bid = 102;

Inner Joins

```
SELECT s.sid, s.sname, r.bid
  FROM Sailors s, Reserves r
 WHERE s.sid = r.sid
```

```
SELECT s.sid, s.sname, r.bid
 FROM Sailors s INNER JOIN Reserves r
              ON s.sid = r.sid
```

Both are equivalent!

Left Outer Join

Returns all matched rows, plus all unmatched rows from the table on the **left** of the join clause

(use nulls in fields of non-matching tuples)

```
SELECT s.sid, s.sname, r.bid
  FROM Sailors s LEFT OUTER JOIN
       Reserves r
        ON s.sid = r.sid;
```

Returns all sailors & bid for boat in any of their reservations

Note: no match for s.sid? r.sid `IS NULL`!

```
SELECT s.sid, s.sname, r.bid
  FROM Sailors s LEFT OUTER JOIN Reserves r
                 ON s.sid = r.sid;
```

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
95	Bob	3	63.5

sid	bid	day
22	101	10/10/96
95	103	11/12/96

s.sid	s.name	r.bid
22	Dustin	101
95	Bob	103
31	Lubber	

NULL

Right Outer Join

Returns all matched rows, plus all unmatched rows from the table on the **right** of the join clause

(use nulls in fields of non-matching tuples)

```
SELECT s.sid, b.bid, b.bname
  FROM Reserves r RIGHT OUTER JOIN
       Boats b
        ON r.bid = b.bid;
```

Returns all boats & information on which ones are reserved

Note: no match for b.bid? r.bid `IS NULL`!

Full Outer Join

Full Outer Join returns all (matched or unmatched) rows from the tables on both sides of the join clause

```
SELECT r.sid, b.bid, b.bname
  FROM Reserves2 r FULL OUTER JOIN
       Boats2 b
        ON r.bid = b.bid;
```

Returns all boats & all information on reservations

No match for r.bid?

- b.bid IS NULL AND b.bname is NULL

No match for b.bid?

- r.sid is NULL

GROUP BY AND HAVING

GROUP BY and HAVING

So far, we've applied aggregate operators to all (qualifying) tuples.

Sometimes, we want to apply them to each of several *groups* of tuples.

Consider: *Find the age of the youngest sailor for each rating level.*

In general, we don't know how many rating levels exist, and what the rating values for these levels are!

Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this (!):

For i = 1, 2, ... , 10:

```
SELECT MIN (S.age)
FROM   Sailors S
WHERE  S.rating = i
```

Queries With GROUP BY and HAVING

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

Group rows by columns in *grouping-list*

Every column from *target-list* mast appear in the *grouping-list*

HAVING restricts through an *aggregate* which group-rows are part of the result

Conceptual Evaluation

(1) Cross-product of *relation-list*

(2) Select only tuples that follow the where clause *qualification*)

(3) Partition rows by the value of attributes in *grouping-list*

(4) Select only groups that follow the *group-qualification*

Attributes in *target-list* must also be in *grouping-list*.

(5) One answer tuple is generated per qualifying group, showing *target-list*

Expressions in *group-qualification* must have a *single value per group*! That is, attributes in *group-qualification* must be part of an aggregate op / must appear in the *grouping-list*.

Find the age of the youngest sailor with age ≥ 18, for each rating with at least 2 such sailors

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

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
71	zorba	10	16.0
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0

2

rating	age
1	33.0
7	45.0
7	35.0
8	55.5
10	35.0

3

rating	m-age	count
1	33.0	1
7	35.0	2
8	55.0	1
10	35.0	1

4

rating	
7	35.0

Find sailors who have reserved all boats.

Can you do this using Group By and Having?

```
SELECT    S.name
FROM      Sailors S, Reserves R
WHERE     S.sid = R.sid
GROUP BY  S.name, S.sid
HAVING    COUNT(DISTINCT R.bid) =
          (Select COUNT (*) FROM Boats)
```

Note: must have both sid and name in the GROUP BY clause. Why?

(1) Attributes in *target-list* must also be in *grouping-list*.
(2) Expressions in *group-qualification* must have a *single value per group*!

```
SELECT   S.name, S.sid
FROM     Sailors S, reserves R
WHERE    S.sid = R.sid
GROUP BY S.name, S.sid
HAVING   COUNT(DISTINCT R.bid) =
         (Select COUNT (*) FROM Boats)
```

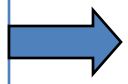
s.name	s.sid	r.sid	r.bid
Dustin	22	22	101
Lubber	31	22	101
Bob	95	22	101
Dustin	22	95	102
Lubber	31	95	102
Bob	95	95	102

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Count (*) from boats = 4

s.name	s.sid	bcount
Dustin	22	1
Bob	95	1

Apply having clause to groups



s.name	s.sid

Sorting the Results of a Query

ORDER BY *column* [ASC | DESC] [, ...]

```
SELECT   S.rating, S.sname, S.age
FROM     Sailors S, Boats B, Reserves R
WHERE    S.sid=R.sid AND R.bid=B.bid
         AND B.color='red'
ORDER BY S.rating, S.sname;
```

Extra reporting power obtained by combining with aggregation.

```
SELECT   S.sid, COUNT (*) AS redrescnt
FROM     Sailors S, Boats B, Reserves R
WHERE    S.sid=R.sid AND R.bid=B.bid
         AND B.color='red'
GROUP BY S.sid
ORDER BY redrescnt DESC;
```

Summary: The SQL Query

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