CS186 Discussion Section Week 5

SQL

Outline for Today

- Quick SQL Intro
- Except, Union, Intersect
- Nested Subqueries
- Joins
    - Inner, Left, Right, Full
- Aggregation

What is it?

- Structured query language
- AKA the most useful thing you will learn out of this course
- Used to communicate with databases
- ANSI standard for relational databases

Basic SQL

```
SELECT   target-list
FROM      relation-list
WHERE    qualification
```

- target-list: list of attributes in each relation (or *)
- relation-list: list of relations, usually a table
- qualification: set of select clauses

Tables

```
CREATE TABLE Students
(sid text PRIMARY KEY,
name text,
gpa integer)

CREATE TABLE Courses
(cid text PRIMARY KEY,
name text,)
```

```
CREATE TABLE Enrollments
(sid text,
cid text,
FOREIGN KEY(sid) REFERENCES students (sid),
FOREIGN KEY(cid) REFERENCES courses (cid),
PRIMARY KEY(sid, cid))
```

Basic SQL Examples

- SELECT name FROM Students WHERE gpa = 3;
- SELECT * FROM Students;
- SELECT DISTINCT name FROM Students;
- SELECT name, cid

  FROM Students, Enrollments

  WHERE sid = sid;

Basic SQL Examples

- SELECT name FROM Students WHERE gpa = 3;
- SELECT * FROM Students;
- SELECT DISTINCT name FROM Students;
- SELECT name, cid
  FROM Students, Enrollments
  WHERE Students.sid = Enrollments.sid;

Basic SQL Examples

- SELECT name FROM Students WHERE gpa = 3;
- SELECT * FROM Students;
- SELECT DISTINCT name FROM Students;
- SELECT name, cid
  FROM Students S, Enrollments E
  WHERE S.sid = E.sid;

Except, Union, Intersect

```
SELECT cid FROM Courses
EXCEPT
SELECT DISTINCT cid
   FROM Students S, Enrollments E
   WHERE S.sid = E.sid;
```

“Select all courses with no one enrolled.”

Notice that this is equivalent to subtraction.

Except, Union, Intersect

```
SELECT cid FROM Courses
EXCEPT
SELECT cid
  FROM Students S, Enrollments E
  WHERE S.sid = E.sid;
```

“Select all courses with no one enrolled.”

Notice that this is equivalent to subtraction.

Except, Union, Intersect

```
SELECT cid FROM Courses
UNION
SELECT cid
   FROM Students S, Enrollments E
   WHERE S.sid = E.sid;
```

“Select all courses.”

UNION removes duplicates, UNION ALL does not.

Except, Union, Intersect

```
SELECT cid FROM Courses
INTERSECT
SELECT cid
   FROM Students S, Enrollments E
   WHERE S.sid = E.sid;
```

“Select all courses with someone enrolled.”

INTERSECT removes duplicates, INTERSECT ALL does not.

Nested Subqueries

- IN operator: check if attribute is in subrelation

-
  ```
  SELECT name
   FROM Students
   WHERE sid IN (SELECT sid FROM
   Enrollments);
  ```
- “Select students enrolled in some class.”

Nested Subqueries

- EXISTS operator: check if subrelation is empty
- SELECT name

  FROM Students

  WHERE EXISTS (SELECT sid FROM
                Enrollments);
- Wait, what?
  - EXISTS doesn't seem useful here…
  - Result: Correlated subqueries!
- This returns names of all students if the inner query finds one or more sid. If inner query does not return any sids, full query returns nothing.

Correlated Subqueries

- Inner query depends on outer query.
-
  ```
  SELECT name
  FROM Students S
  WHERE EXISTS (SELECT sid FROM Enrollments
                WHERE sid = S.sid)
  ```

- Same query as before!
- Any Caveats?
  - Much slower. Inner query must run once per outer tuple.

Other Subquery Operations

- UNIQUE: is every result unique?

-
  ```
  SELECT name
  FROM Students S
  WHERE UNIQUE (SELECT sid FROM Enrollments
                WHERE sid = S.sid)
  ```
- “Select students enrolled in exactly 1 class (since sid is a key of students).”

Other Subquery Operations

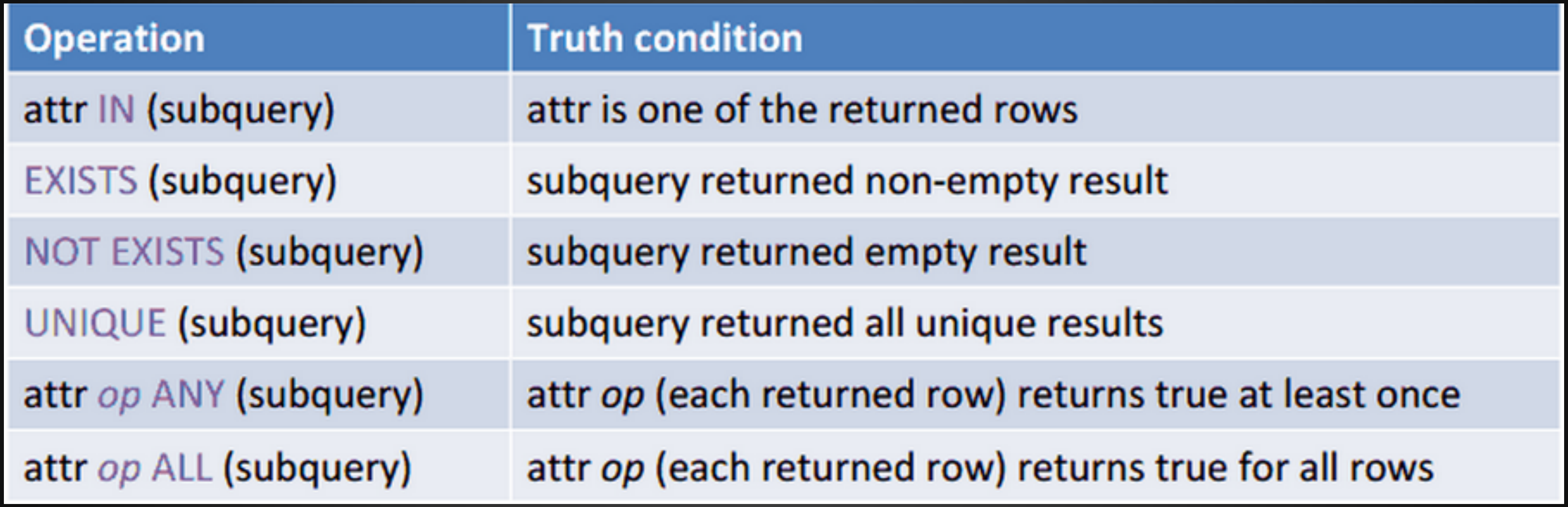
- `> ANY, > ALL`
-
  ```
  SELECT name
  FROM Students S
  WHERE gpa >= ALL
      (SELECT gpa FROM Students)
  ```
- “Select students with the highest GPA.”
- For the WHERE condition to be met, gpa >= every single gpa returned by subquery
- In this case, subquery is not correlated.

Other Subquery Operations

- > ANY, > ALL

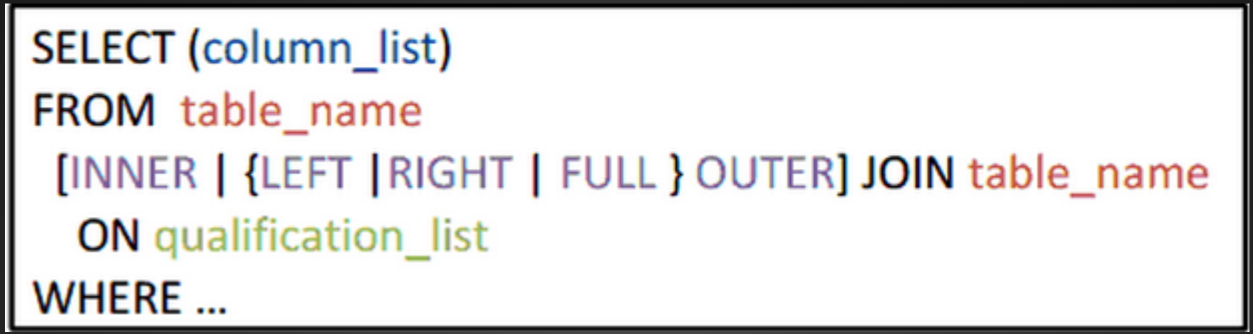
-
  ```
  SELECT name
  FROM Students S
  WHERE gpa = ANY
      (SELECT gpa FROM Students)
  ```
- “Select all students!”
- For the WHERE condition to be met, gpa = any gpa returned by subquery
- Again, subquery is not correlated.

Summary of Operations



Operation	Truth condition
attr IN (subquery)	attr is one of the returned rows
EXISTS (subquery)	subquery returned non-empty result
NOT EXISTS (subquery)	subquery returned empty result
UNIQUE (subquery)	subquery returned all unique results
attr *op* ANY (subquery)	attr *op* (each returned row) returns true at least once
attr *op* ALL (subquery)	attr *op* (each returned row) returns true for all rows

Special Joins



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

- SELECT * FROM Students S
  INNER JOIN Enrollments E
  ON S.sid = E.sid;
- Not really different than before:
- SELECT * FROM Students S, Enrollments E
  WHERE S.sid = E.sid;

Outer Joins (LEFT)

- Outer joins add in unmatched rows.
- SELECT sname, cid FROM Students S
  LEFT OUTER JOIN Enrollments E
  ON S.sid = E.sid;
- Example output
- Rows of left relation that couldn't be matched are still here with a NULL right hand side.

sname	cid
Dan	160
Evan	186
Lu	186
Victor	186
Liwen	NULL
Mike	NULL

Outer Joins (RIGHT)

- Outer joins add in unmatched rows.
- SELECT sname, cid FROM Students S
  RIGHT OUTER JOIN Enrollments E
  ON S.sid = E.sid;
- Example output
- Rows of right relation that couldn’t be matched are still here with a NULL left hand side.

sname	cid
Dan	160
Evan	186
Lu	186
Victor	186
NULL	188
NULL	189

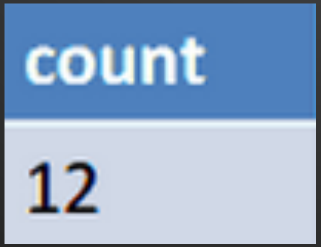
Outer Joins (FULL)

- Outer joins add in unmatched rows.
- SELECT sname, cid FROM Students S
  FULL OUTER JOIN Enrollments E
  ON S.sid = E.sid;
- Example output
- Any unmatched rows are still there, with the other side NULL

sname	cid
Dan	160
Evan	186
Lu	186
Victor	186
NULL	188
NULL	189
Liwen	NULL
Mike	NULL

Aggregation

```
SELECT COUNT(*) FROM Students;
SELECT AVG(gpa) FROM Students;
SELECT COUNT(DISTINCT name) FROM Students;
SELECT DISTINCT COUNT(name) FROM Students;
```

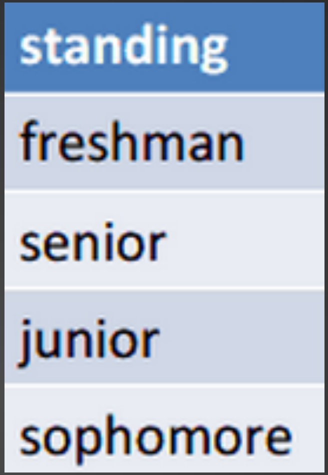


count
12

GROUP BY

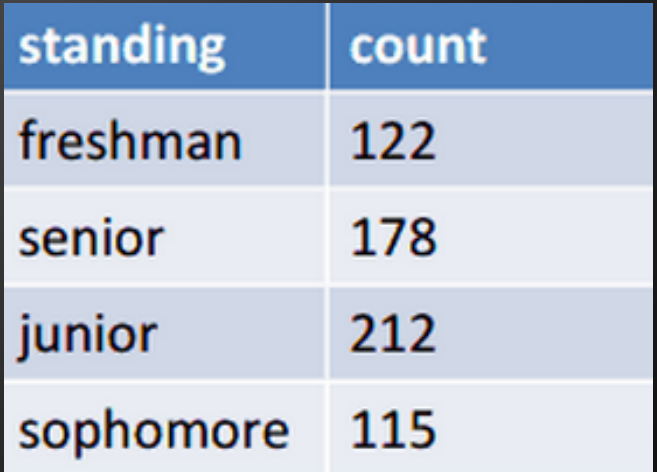
- GROUP BY aggregates rows together.

- SELECT standing
  FROM Students
  GROUP BY standing;



standing
freshman
senior
junior
sophomore

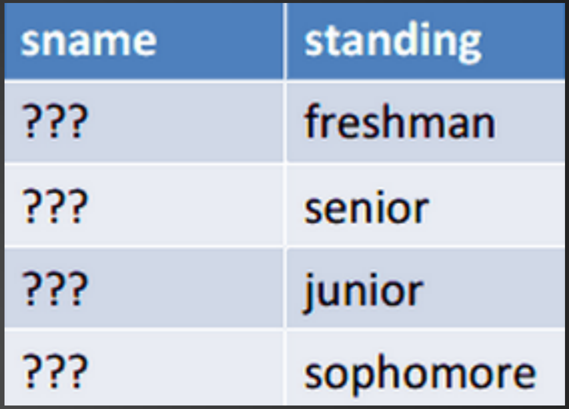
- SELECT standing, COUNT(*)
  FROM Students
  GROUP BY standing;



standing	count
freshman	122
senior	178
junior	212
sophomore	115

GROUP BY

- SELECT sname, standing
  FROM Students
  GROUP BY standing;

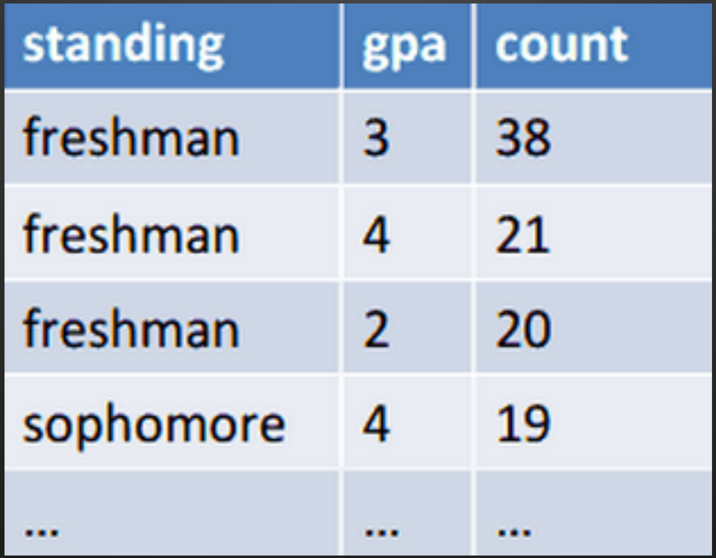


sname	standing
???	freshman
???	senior
???	junior
???	sophomore

- Illegal! Can’t find one ‘sname’ for whole table.
- Rule:
  - SELECT fields must either be inside the GROUP BY or aggregates.

GROUP BY

- How about grouping by multiple fields?
- SELECT standing, gpa, COUNT(*)
  FROM Students
  GROUP BY standing, gpa;

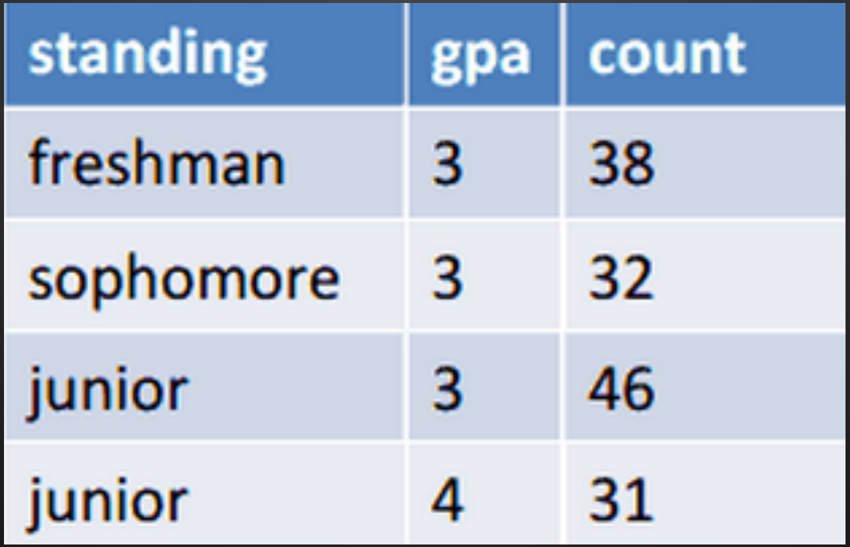


standing	gpa	count
freshman	3	38
freshman	4	21
freshman	2	20
sophomore	4	19
...	...	...

HAVING

- HAVING: Modifies which groups are returned.
- SELECT standing, gpa, COUNT(*)

  ```
  FROM Students
  GROUP BY standing, gpa
  HAVING COUNT(*) > 30;
  ```



standing	gpa	count
freshman	3	38
sophomore	3	32
junior	3	46
junior	4	31

HAVING vs. WHERE

- HAVING: condition on aggregations
  - usually after group by
- WHERE: condition on individual rows
  - usually before group by
- SELECT standing, gpa, COUNT(*)

  FROM Students

  WHERE sname STARTS_WITH ‘A’

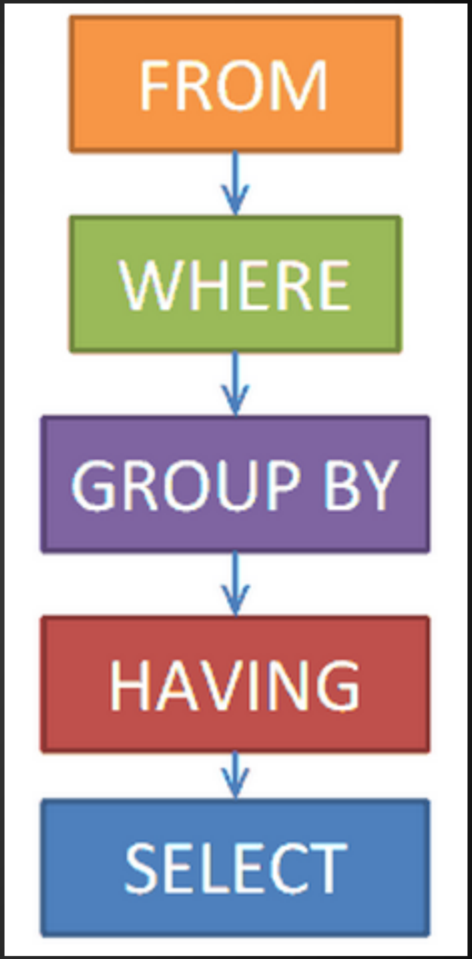
  GROUP BY standing, gpa

  HAVING COUNT(*) > 30;

Logical Order of a Query

- SELECT standing, gpa, COUNT(*)
  FROM Students
  WHERE sname STARTS_WITH ‘A’
  GROUP BY standing, gpa
  HAVING COUNT(*) > 3;

- Where does aggregation happen?
  During GROUP BY!


FROM
WHERE
GROUP BY
HAVING
SELECT