CS186 Discussion 2

(SQL)

Matthew Deng

Single-Table SQL

Single-Table SQL

```
SELECT [DISTINCT] <column expression list>
FROM <single table>
[WHERE <predicate>]
[GROUP BY <column list>
 [HAVING <predicate>]]
[ORDER BY <column list>];
```

SELECT FROM

```
SELECT [DISTINCT] <column expression list>
FROM <single table>
[WHERE <predicate>]
[GROUP BY <column list>
 [HAVING <predicate>]]
[ORDER BY <column list>];
```

- Retrieve entries from the table in the FROM clause
- Output columns in the SELECT clause

WHERE

```
SELECT [DISTINCT] <column expression list>
FROM <single table>
[WHERE <predicate>]
[GROUP BY <column list>
 [HAVING <predicate>]]
[ORDER BY <column list>];
```

- Keep only the tuples that satisfy the predicate in the WHERE clause

DISTINCT

```
SELECT [DISTINCT] <column expression list>
FROM <single table>
[WHERE <predicate>]
[GROUP BY <column list>
 [HAVING <predicate>]]
[ORDER BY <column list>];
```

- Remove duplicate tuples before outputting

GROUP BY

```
SELECT [DISTINCT] <column expression list>
FROM <single table>
[WHERE <predicate>]
[GROUP BY <column list>
 [HAVING <predicate>]]
[ORDER BY <column list>];
```

- Partition table into groups in GROUP BY predicate
- Produces aggregate result for each group
- Aggregates: AVG, SUM, COUNT, MAX, MIN

HAVING

```
SELECT [DISTINCT] <column expression list>
FROM <single table>
[WHERE <predicate>]
[GROUP BY <column list>
 [HAVING <predicate>]]
[ORDER BY <column list>];
```

- Keep only the tuples that satisfy the predicate in the HAVING clause
- Can be used on aggregates or GROUP BY columns
- Can ONLY be used with GROUP BY

ORDER BY

```
SELECT [DISTINCT] <column expression list>
FROM <single table>
[WHERE <predicate>]
[GROUP BY <column list>
 [HAVING <predicate>]]
[ORDER BY <column list>];
```

- Sorts results by columns from left to right
  - Students.age, Students.name
- ASC for ascending [default], DESC for descending
- Must refer to columns in output

Other Clauses

- LIKE
  - Compare similar values with wildcard operators
  - In WHERE clause
  - ‘_’: exactly one character
  - ‘%’: any number of characters
- LIMIT
  - At the end of query
  - Number of results to return

Worksheet #1-2

Single-Table SQL Exercises

1. Find the total number of “Techno” albums released each year.

Single-Table SQL Exercises

1. Find the total number of “Techno” albums released each year.

```
SELECT year_released, COUNT(*)
FROM Albums
WHERE genre = ‘Techno’
GROUP BY year_released;
```

Single-Table SQL Exercises

2. Find the genre and the number of albums released per genre; don’t include genres that have a count of less than 10.

Single-Table SQL Exercises

2. Find the genre and the number of albums released per genre; don't include genres that have a count of less than 10.

```
SELECT genre, COUNT(*)
FROM Albums
GROUP BY genre
HAVING COUNT(*) >= 10;
```

Multi-Table SQL

Single-Table SQL

```
SELECT [DISTINCT] <column expression list>
FROM <single table>
[WHERE <predicate>]
[GROUP BY <column list>
 [HAVING <predicate>]]
[ORDER BY <column list>];
```

Multi-Table SQL

```
SELECT [DISTINCT] <column expression list>
FROM <multiple tables>
[WHERE <predicate>]
[GROUP BY <column list>
 [HAVING <predicate>]]
[ORDER BY <column list>];
```

Inner Joins

- Multiple tables in the FROM clause
- Join predicate in the WHERE clause
  - Boats.owner_id = Sailors.sid

- FROM <table 1>, <table 2> WHERE <predicate>

- FROM <table 1> INNER JOIN <table 2> ON <predicate>

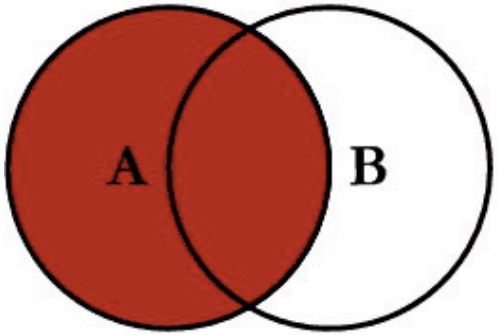
- FROM <table 1> NATURAL JOIN <table 2>

Outer Joins

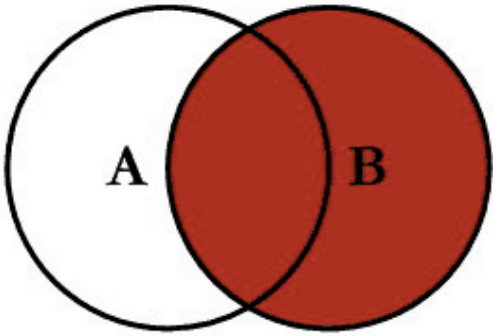
- FROM <table 1> {LEFT | RIGHT | FULL} OUTER JOIN <table 2> ON <predicate>

- Returns all matching rows (like inner join), PLUS all unmatched rows from the table on the {LEFT | RIGHT | BOTH}
  - Use nulls to fill in the rest

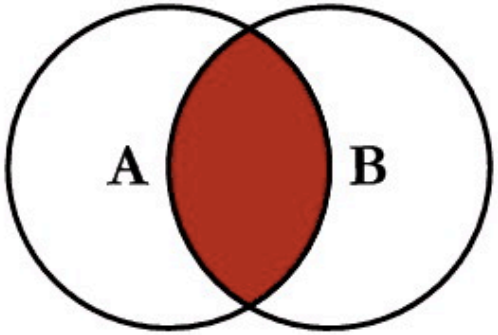
SQL JOINS



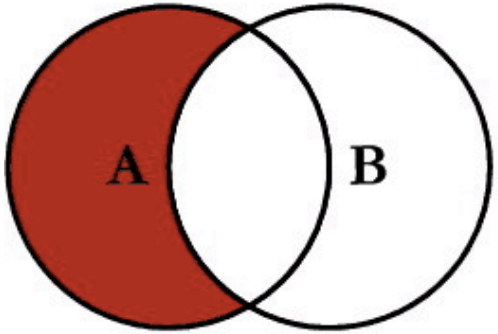

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



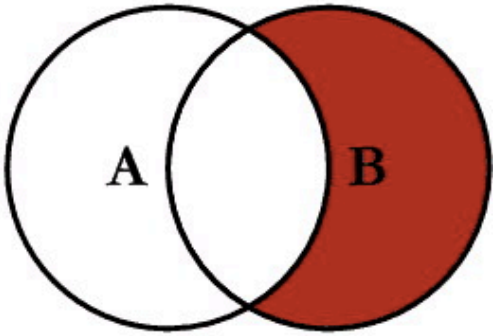

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



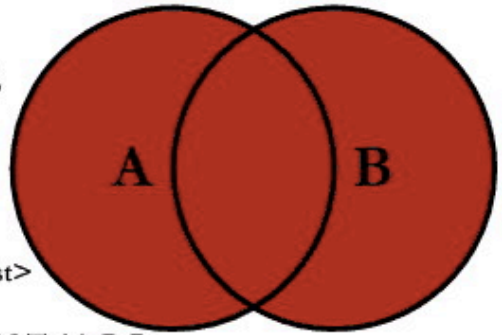

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



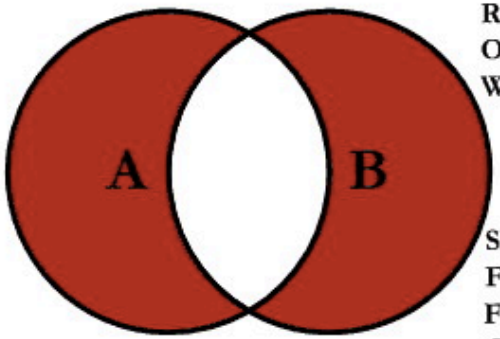

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Named Queries

- On the Fly
- Views
  - CREATE VIEW view_name AS select_statement
- Subqueries in FROM
  - SELECT * FROM (select_statement) AS view_name
- WITH
  - WITH view_name AS (select_statement)

Worksheet #3-5

Multi-Table SQL Exercises

3. For each song, its name, the name of its album, and the name of its artist.

Multi-Table SQL Exercises

3. For each song, its name, the name of its album, and the name of its artist.

```
SELECT Songs.song_name,
       Albums.album_name,
       Artists.artist_name
FROM Artists, Albums, Songs
WHERE Artists.artist_id = Albums.artist_id
AND Songs.album_id = Albums.album_id;
```

Multi-Table SQL Exercises

3. For each song, its name, the name of its album, and the name of its artist.

```
SELECT song_name,
       album_name,
       artist_name
FROM Artists
   NATURAL JOIN Albums
   NATURAL JOIN Songs
```

Multi-Table SQL Exercises

4. Find singers, with no duplicates, who released both “Techno” and “Pop” albums.

Multi-Table SQL Exercises

4. Find singers, with no duplicates, who released both “Techno” and “Pop” albums.

```
SELECT DISTINCT Artists.artist_name
FROM Artists, Albums A1, Albums A2
WHERE Artists.artist_id = A1.artist_id
  AND A1.artist_id = A2.artist_id
  AND A1.genre = ‘Techno’
  AND A2.genre = ‘Pop’;
```

Multi-Table SQL Exercises

5. Find all album id’s and names for every artist active since 2000 or later. If an artist does not have any albums, you should still include the artist's information in your output. Use LEFT OUTER JOIN, RIGHT OUTER JOIN, or INNER JOIN.

Multi-Table SQL Exercises

5. Find all album id's and names for every artist active since 2000 or later. If an artist does not have any albums, you should still include the artist's information in your output. Use LEFT OUTER JOIN, RIGHT OUTER JOIN, or INNER JOIN.

```
SELECT Ar.artist_id, Ar.artist_name,
          Al.album_id, Al.album_name
FROM Artists Ar
LEFT OUTER JOIN Albums Al
ON Ar.artist_id=Al.artist_id
WHERE Ar.first_year_active >= 2000;
```

Worksheet #6, 7

Multi-Table SQL Exercises

6. The six degrees of separation is a theory that says that everyone is connected to each other by a chain of people that is 6 or less steps long. Given the undirected friend graph below, where each link is stored once, find all pairs of Users who are exactly six steps away from each other.

```
CREATE TABLE Friends(
   fromID integer,
   toID integer,
   since date,
   PRIMARY KEY (fromID, toID),
   FOREIGN KEY (fromID) REFERENCES Users,
   FOREIGN KEY (toID) REFERENCES Users,
   CHECK (fromID < toID));
```

Multi-Table SQL Exercises

6. Create a view that goes both directions:

```
CREATE VIEW BothFriends AS
SELECT * FROM Friends
UNION ALL
SELECT F.toID AS fromID,
       F.fromID AS toID,
       F.since FROM Friends F;
```

Multi-Table SQL Exercises

6.

```
SELECT F1.fromID, F5.toID
FROM BothFriends F1,
     BothFriends F2,
     BothFriends F3,
     BothFriends F4,
     BothFriends F5
WHERE F1.toID = F2.fromID
   AND F2.toID = F3.fromID
   AND F3.toID = F4.fromID
   AND F4.toID = F5.fromID;
```

Multi-Table SQL Exercises

7. Given the following schema, find the median value:

```
T (c integer)
```

Multi-Table SQL Exercises

7. Given the following schema, find the median value:

```
T (c integer)
```

```
SELECT c AS median FROM T
WHERE
   (SELECT COUNT(*) from T AS T1
    WHERE T1.c <= T.c)
   =
   (SELECT COUNT(*) from T AS T2
    WHERE T2.c >= T.c);
```

Multi-Table SQL Exercises

7. Given the following schema, find the median value:

```
T (c integer)
```

```
SELECT x.c as median
FROM T x, T y
GROUP BY x.c
HAVING
SUM(CASE WHEN y.c <= x.c THEN 1 ELSE 0 END)
>= (COUNT(*)+1)/2 -- ceiling(N/2)
AND
SUM(CASE WHEN y.c >= x.c THEN 1 ELSE 0 END)
>= (COUNT(*)/2)+1 -- floor(N/2) +1
```

Multi-Table SQL Exercises

7. Given the following schema, find the median value:

```
T (c integer)
```

```
CREATE VIEW twocounters AS
(SELECT c,
ROW_NUMBER() OVER (ORDER BY c ASC) AS RowAsc,
ROW_NUMBER() OVER (ORDER BY c DESC) AS RowDesc
FROM T );

SELECT MIN(c) FROM twocounters
WHERE RowAsc IN (RowDesc, RowDesc - 1, RowDesc + 1);
```