CS W4111.001
Introduction to Databases
Spring 2017

Computer Science Department
Columbia University

JOINS

```
SELECT [DISTINCT] target_list
FROM table_name
    [INNER | {LEFT |RIGHT | FULL } {OUTER}] JOIN table_name
ON qualification_list
WHERE ...
```

INNER is default
Difference in how to deal with NULL values

PostgreSQL documentation:
Inner/Natural Join

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

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

SELECT s.sid, s.name, r.bid
FROM Sailors s NATURAL JOIN Reserves r

Natural Join means equijoin for each pair of attributes with same name

Sailor names and their reserved boat ids

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eugene</td>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>Luis</td>
<td>2</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Ken</td>
<td>8</td>
<td>27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>102</td>
<td>9/12</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>9/13</td>
</tr>
</tbody>
</table>

Result

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eugene</td>
<td>102</td>
</tr>
<tr>
<td>2</td>
<td>Luis</td>
<td>102</td>
</tr>
</tbody>
</table>
Sailor names and their reserved boat ids

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

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eugene</td>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>Luis</td>
<td>2</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Ken</td>
<td>8</td>
<td>27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>102</td>
<td>9/12</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>9/13</td>
</tr>
</tbody>
</table>

Result

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eugene</td>
<td>102</td>
</tr>
<tr>
<td>2</td>
<td>Luis</td>
<td>102</td>
</tr>
</tbody>
</table>

Notice: No result for Ken!
Left Outer Join

Returns all matched rows and all unmatched rows from table on left of join clause
(at least one row for each row in left table)

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

All sailors & bid for boat in their reservations bid set to NULL if no reservation
Right Outer Join

Same as LEFT OUTER JOIN, but guarantees result for rows in table on right side of JOIN

```
SELECT s.sid, s.name, r.bid
FROM Reserves r RIGHT OUTER JOIN Sailors s
ON s.sid = r.sid
```

Full Outer Join

Returns all matched or unmatched rows from both sides of JOIN

```
SELECT s.sid, s.name, r.bid
FROM Sailors s FULL OUTER JOIN Reserves r
ON s.sid = r.sid
```
Full Outer Join

```
SELECT s.sid, s.name, r.bid
FROM Sailors s
FULL OUTER JOIN Reserves r
ON s.sid = r.sid
```

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eugene</td>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>Luis</td>
<td>2</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Ken</td>
<td>8</td>
<td>27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>102</td>
<td>9/12</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>9/13</td>
</tr>
<tr>
<td>4</td>
<td>109</td>
<td>9/20</td>
</tr>
</tbody>
</table>

```
Result

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eugene</td>
<td>102</td>
</tr>
<tr>
<td>2</td>
<td>Luis</td>
<td>102</td>
</tr>
<tr>
<td>3</td>
<td>Ken</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>109</td>
</tr>
</tbody>
</table>
```

Why is sid NULL?

ORDER BY, LIMIT

```
SELECT [DISTINCT] target-list
FROM relation-list
WHERE qualification
GROUP BY grouping-list
HAVING group-qualification
ORDER BY order-list
LIMIT limit-expr [OFFSET offset-expr]
```

FROM | WHERE | GROUP BY

HAVING | SELECT | DISTINCT

ORDER BY | LIMIT
ORDER BY

SELECT S.name, S.rating, S.age
FROM Sailors S
ORDER BY S.rating ASC,
        S.age DESC

List of order-list expressions dictates ordering precedence
Sorted in ascending by rating
If ties, sorted high-to-low on age

ORDER BY

SELECT S.name, (S.rating/2)::int, S.age
FROM Sailors S
ORDER BY (S.rating/2)::int ASC,
         S.age DESC

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eugene</td>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>Luis</td>
<td>2</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Ken</td>
<td>8</td>
<td>27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>int4</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Luis</td>
<td>1</td>
<td>39</td>
</tr>
<tr>
<td>Ken</td>
<td>4</td>
<td>27</td>
</tr>
<tr>
<td>Eugene</td>
<td>4</td>
<td>22</td>
</tr>
</tbody>
</table>
ORDER BY

```sql
SELECT S.name, (S.rating/2)::int, S.age
FROM Sailors S
ORDER BY (S.rating/2)::int ASC, S.age ASC
```

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eugene</td>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>Luis</td>
<td>2</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Ken</td>
<td>8</td>
<td>27</td>
</tr>
</tbody>
</table>

Result

<table>
<thead>
<tr>
<th>name</th>
<th>int4</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Luis</td>
<td>1</td>
<td>39</td>
</tr>
<tr>
<td>Eugene</td>
<td>4</td>
<td>22</td>
</tr>
<tr>
<td>Ken</td>
<td>4</td>
<td>27</td>
</tr>
</tbody>
</table>

LIMIT

```sql
SELECT S.name, (S.rating/2)::int, S.age
FROM Sailors S
ORDER BY (S.rating/2)::int ASC, S.age DESC
LIMIT 2
```

Only the first 2 results

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eugene</td>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>Luis</td>
<td>2</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Ken</td>
<td>8</td>
<td>27</td>
</tr>
</tbody>
</table>

Result

<table>
<thead>
<tr>
<th>name</th>
<th>int4</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Luis</td>
<td>1</td>
<td>39</td>
</tr>
<tr>
<td>Ken</td>
<td>4</td>
<td>27</td>
</tr>
</tbody>
</table>
LIMIT

SELECT S.name, (S.rating/2)::int, S.age
FROM Sailors S
ORDER BY (S.rating/2)::int ASC, S.age DESC
LIMIT 2 OFFSET 1

Only the first 2 results

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eugene</td>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>Luis</td>
<td>2</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Ken</td>
<td>8</td>
<td>27</td>
</tr>
</tbody>
</table>

Result

<table>
<thead>
<tr>
<th>name</th>
<th>int4</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ken</td>
<td>4</td>
<td>27</td>
</tr>
<tr>
<td>Eugene</td>
<td>4</td>
<td>22</td>
</tr>
</tbody>
</table>

LIMIT

SELECT S.name, (S.rating/2)::int, S.age
FROM Sailors S
ORDER BY (S.rating/2)::int ASC, S.age DESC
LIMIT (SELECT count(*) / 2 FROM Sailors AS S2)

Can have expressions instead of constants

Result

<table>
<thead>
<tr>
<th>name</th>
<th>int4</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Luis</td>
<td>1</td>
<td>39</td>
</tr>
</tbody>
</table>
User Defined Functions (UDFs)

Custom functions that can be called in database
Many languages: SQL, python, C, perl, etc.

CREATE FUNCTION function_name(p1 type, p2 type, ...) 
RETURNS type

AS $$
-- logic

$$ LANGUAGE language_name;
User Defined Functions (UDFs)

Custom functions that can be called in database
Many languages: SQL, python, C, perl, etc.

```
CREATE FUNCTION function_name(p1 type, p2 type, ...) 
RETURNS type 
AS $$
-- logic
$$
LANGUAGE language_name;
```

A simple UDF (language: SQL)

```
CREATE FUNCTION mult1(v int) RETURNS int 
AS $$
SELECT v * 10;
$$ LANGUAGE SQL;
```

```
CREATE FUNCTION function_name(p1 type, p2 type, ...) 
RETURNS type 
AS $$
-- logic
$$
LANGUAGE language_name;
```
A simple UDF (language: SQL)

CREATE FUNCTION mult1(v int) RETURNS int AS $$
SELECT v * 10;
$$ LANGUAGE SQL;

SELECT mult1(S.age) FROM sailors AS S

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eugene</td>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>Luis</td>
<td>2</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Ken</td>
<td>8</td>
<td>27</td>
</tr>
</tbody>
</table>

Process a Record (language: SQL)

CREATE FUNCTION mult2(x sailors) RETURNS int AS $$
SELECT (x.sid + x.age) / x.rating;
$$ LANGUAGE SQL;

SELECT mult2("") FROM sailors AS S

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eugene</td>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>Luis</td>
<td>2</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Ken</td>
<td>8</td>
<td>27</td>
</tr>
</tbody>
</table>

http://www.postgresql.org/docs/9.1/static/xfunc-sql.html
CREATE FUNCTION proc(v int) RETURNS int 
AS $$
DECLARE  
-- define variables
BEGIN  
-- PL/SQL code
END;
$$ LANGUAGE plpgsql;

http://www.postgresql.org/docs/9.4/static/plpgsql.html
Procedural Code (language: plpython2u)

CREATE FUNCTION proc(v int) RETURNS int
AS $$
import random
return random.randint(0, 100) * v
$$ LANGUAGE plpython2u;

Very powerful – can do anything so must be careful
run in a python interpreter with no security protection
plpy module provides database access
plpy.execute("select 1")

http://www.postgresql.org/docs/9.4/static/plpython.html

Procedural Code (language: plpython2u)

CREATE FUNCTION proc(word text) RETURNS text
AS $$
import requests
resp = requests.get('http://google.com/search?q=%s' % v)
return resp.content.decode('unicode-escape')
$$ LANGUAGE plpython2u;

Very powerful – can do anything so must be careful
run in a python interpreter with no security protection
plpy module provides database access
plpy.execute("select 1")

http://www.postgresql.org/docs/9.4/static/plpython.html
WITH

WITH RedBoats(bid, count) AS
    (SELECT B.bid, count(*)
    FROM Boats B, Reserves R
    WHERE R.bid = B.bid AND B.color = 'red'
    GROUP BY B.bid)

SELECT name, count
FROM Boats AS B, RedBoats AS RB
WHERE B.bid = RB.bid AND count < 2

Names of unpopular boats

WITH

WITH RedBoats(bid, count) AS
    (SELECT B.bid, count(*)
    FROM Boats B, Reserves R
    WHERE R.bid = B.bid AND B.color = 'red'
    GROUP BY B.bid)

SELECT name, count
FROM Boats AS B, RedBoats AS RB
WHERE B.bid = RB.bid AND count < 2

WITH tablename(attr1, ...) AS (select_query)
    [,tablename(attr1, ...) AS (select_query)]
main_select_query
Views

CREATE VIEW view_name
AS select_statement

“tables” defined as query results rather than inserted base data
Makes development simpler
Used for security
References to view_name replaced with select_statement
Similar to WITH, lasts longer than one query

Names of popular boats

CREATE VIEW boat_counts
AS SELECT bid, count(*)
FROM Reserves R
GROUP BY bid
HAVING count(*) > 10

Used like a normal table

SELECT bname
FROM boat_counts bc, Boats B
WHERE bc.bid = B.bid

Names of popular boats
Rewritten expanded query
CREATE TABLE

CREATE TABLE <table_name> AS
<SELECT STATEMENT>

Guess the schema:

CREATE TABLE used_boats1 AS
SELECT r.bid
FROM Sailors s,
Reservations r
WHERE s.sid = r.sid

used_boats1(bid int)

CREATE TABLE used_boats2 AS
SELECT r.bid as foo
FROM Sailors s,
Reservations r
WHERE s.sid = r.sid

used_boats2(foo int)

How is this different than views?
What if we insert a new record into Reservations?