# Spatial Database Management
# GEP 664 / GEP 380
## Class #2: Database Fundamentals and SQL DML

Frank Donnelly

Dept of EEGS, Lehman College CUNY

Spring 2019

## Today's Topics

Relational database fundamentals

SQL DML - SELECT

SQL DML - JOINS

Next Class

## A Database

"A collection of related data, organized to allow a computer to efficiently answer questions about that data. A database management system (DBMS) is the software used to store, manage, and retrieve the data in a database."

- Encyclopedia of Geographic Information Science, 2008

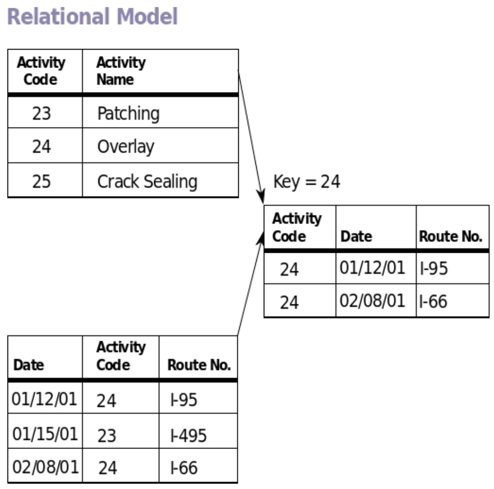
## Database Features

Image source: `https://www-03.ibm.com/ibm/history/ibm100/us/en/icons/system360/impacts`

- Computationally efficient
- Data independence
- Data integrity
- Self-describing

## The Relational Model

Originally proposed by Edgar Codd in 1970



Relational Model

| Activity Code | Activity Name |
|---|---|
| 23 | Patching |
| 24 | Overlay |
| 25 | Crack Sealing |

Key = 24

| Activity Code | Date | Route No. |
|---|---|---|
| 24 | 01/12/01 | I-95 |
| 24 | 02/08/01 | I-66 |

| Date | Activity Code | Route No. |
|---|---|---|
| 01/12/01 | 24 | I-95 |
| 01/15/01 | 23 | I-495 |
| 02/08/01 | 24 | I-66 |

Image source: https://en.wikipedia.org/wiki/Relational_model

## Tables

The central component of the relational database is the table

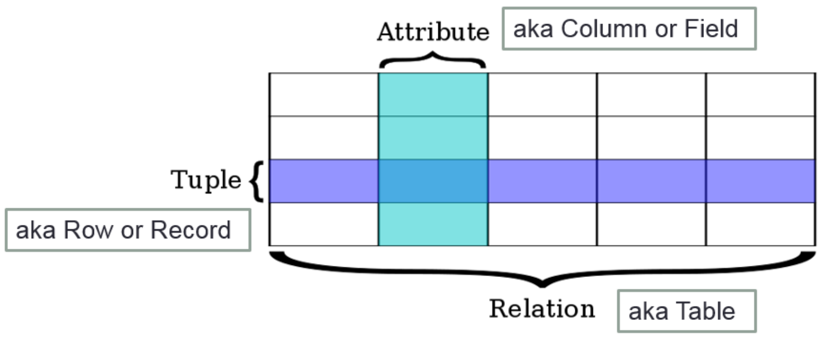


Image source: https://en.wikipedia.org/wiki/Relational_database

## Relational Database Principles

Codd had 12 rules; the following are fundamental principles

1. Attributes drawn from a domain
2. Order is irrelevant
3. Records must be distinct
4. Data items should be indivisible

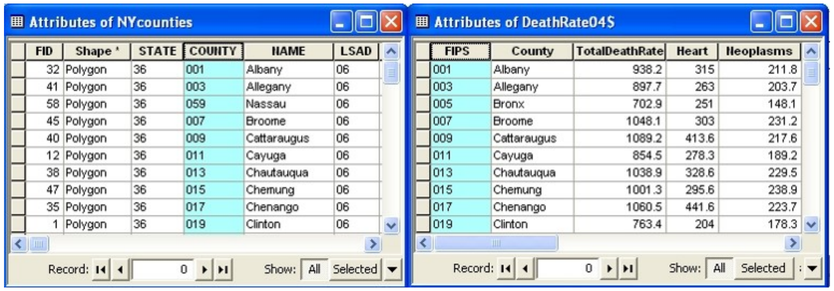
## Domains and Types

Values for specific attributes are drawn from an allowable set called a domain. Attributes are assigned data types, which limits the allowable values and operations that can be performed.

- Variable characters / Text (string)
- Integers (whole numbers)
- Reals / Floats (decimal numbers)
- Time and Date

## Keys and Joins

This structure helps to insure the integrity of the data and makes it possible to relate values in one table to values in another using an attribute they hold in common: a unique ID code called a primary key.



Attributes of NYcounties

| FID | Shape * | STATE | COUNTY | NAME | LSAD |
|---|---|---|---|---|---|
| 32 | Polygon | 36 | 001 | Albany | 06 |
| 41 | Polygon | 36 | 003 | Allegany | 06 |
| 58 | Polygon | 36 | 059 | Nassau | 06 |
| 45 | Polygon | 36 | 007 | Broome | 06 |
| 40 | Polygon | 36 | 009 | Cattaraugus | 06 |
| 12 | Polygon | 36 | 011 | Cayuga | 06 |
| 38 | Polygon | 36 | 013 | Chautauqua | 06 |
| 47 | Polygon | 36 | 015 | Chemung | 06 |
| 35 | Polygon | 36 | 017 | Chenango | 06 |
| 1 | Polygon | 36 | 019 | Clinton | 06 |

Attributes of DeathRate04$

| FIPS | County | TotalDeathRate | Heart | Neoplasms |
|---|---|---|---|---|
| 001 | Albany | 938.2 | 315 | 211.8 |
| 003 | Allegany | 897.7 | 263 | 203.7 |
| 005 | Bronx | 702.9 | 251 | 148.1 |
| 007 | Broome | 1048.1 | 303 | 231.2 |
| 009 | Cattaraugus | 1089.2 | 413.6 | 217.6 |
| 011 | Cayuga | 854.5 | 278.3 | 189.2 |
| 013 | Chautauqua | 1038.9 | 328.6 | 229.5 |
| 015 | Chemung | 1001.3 | 295.6 | 238.9 |
| 017 | Chenango | 1060.5 | 441.6 | 223.7 |
| 019 | Clinton | 763.4 | 204 | 178.3 |

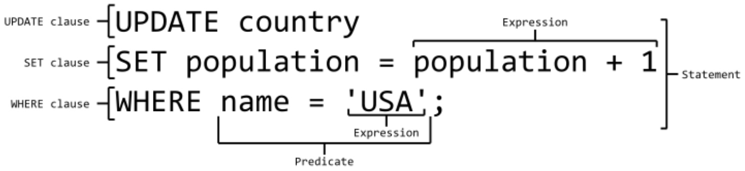
## SQL

SQL is the language for creating and manipulating relational databases; originally based on relational algebra, it uses declarative commands in English.

```
SELECT county_name, pop AS population
FROM countypop
WHERE state='NY' AND pop > 50000
ORDER BY pop;
```

## SQL Statement Components

SQL is an international standard, first formalized in SQL-86. Major revisions in SQL-92 and SQL:1999. Last revision SQL:2016.



```
UPDATE country
SET population = population + 1
WHERE name = 'USA';
```

Image source: `https://en.wikipedia.org/wiki/SQL`

## Subsets of the SQL Language

- Data Manipulation Language
  - SELECT... FROM... WHERE
  - INSERT... INTO... VALUES
  - DELETE FROM... WHERE
  - UPDATE... SET... WHERE
- Data Definition Language
  - CREATE...
  - DROP...
  - ALTER...
  - RENAME...
- Data Control Language
  - GRANT...
  - REVOKE...

## Today's Topics

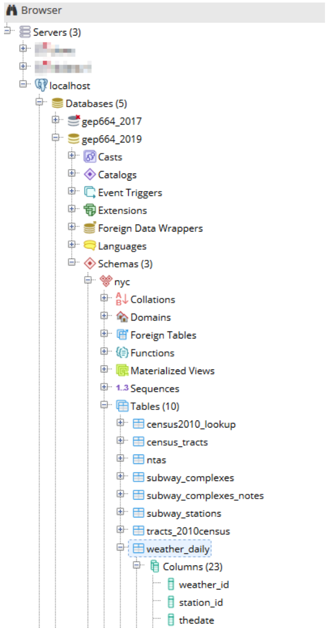
Relational database fundamentals

SQL DML - SELECT

SQL DML - JOINS

Next Class

## Database Objects



```
Browser
Servers (3)
    [redacted]
    [redacted]
    localhost
        Databases (5)
            gep664_2017
            gep664_2019
                Casts
                Catalogs
                Event Triggers
                Extensions
                Foreign Data Wrappers
                Languages
                Schemas (3)
                    nyc
                        Collations
                        Domains
                        Foreign Tables
                        Functions
                        Materialized Views
                        Sequences
                        Tables (10)
                            census2010_lookup
                            census_tracts
                            ntas
                            subway_complexes
                            subway_complexes_notes
                            subway_stations
                            tracts_2010census
                            weather_daily
                                Columns (23)
                                    weather_id
                                    station_id
                                    thedate
```

Database
- Schema
  - Table (or object)
    - Attribute / Column

## Sample Data - NYC Weather

The gep664 database has three schemas: nyc, nys, and public. We'll use weather data tables in the nyc schema as examples:

nyc.weather_daily : observations made at varying intervals (between 10 and 60 minutes) for stations in the NYC metro area from 2016 to 2017

nyc.weather_reptype : code descriptions for classifying the type of geophysical surface observations

nyc.weather_stations : location details for the ten stations in the NYC metro area that are included in the weather_daily table

Source: NOAA's Local Climatological Database (LCD)

## Selection and Projection

Selection (subset of rows)

```
SELECT *
FROM nyc.weather_stations
WHERE elevation > 50;
```

Projection (subset of columns)

```
SELECT station_id, station_name, elevation
FROM nyc.weather_stations;
```

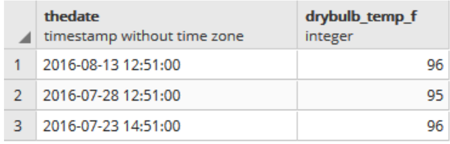
Beware of selecting or projecting everything for large datasets

```
SELECT *
FROM nyc.weather_daily
LIMIT 100;
```

## Multiple Criteria

Boolean logic, numbers and strings, sorting

```
SELECT thedate, drybulb_temp_f
FROM nyc.weather_daily
WHERE drybulb_temp_f > 94
AND station_id = 'WBAN:94728'
ORDER BY thedate DESC;
```



| | thedate | drybulb_temp_f |
|---|---|---|
| | timestamp without time zone | integer |
| 1 | 2016-08-13 12:51:00 | 96 |
| 2 | 2016-07-28 12:51:00 | 95 |
| 3 | 2016-07-23 14:51:00 | 96 |

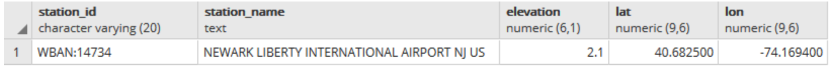
```
SELECT thedate, station_id, drybulb_temp_f
FROM nyc.weather_daily
WHERE station_id IN ('WBAN:14732', 'WBAN:94789')
AND year != 2016;
```

## Pattern Matching

Only use this when searching for text embedded in values

Find stations that begin with NEW

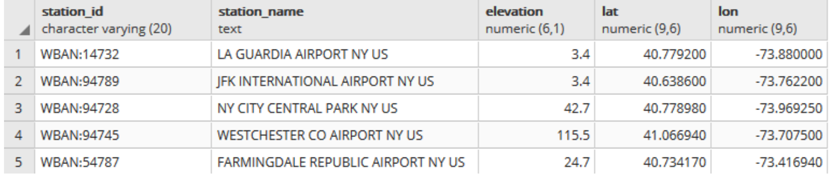
```
SELECT *
FROM nyc.weather_stations
WHERE station_name LIKE 'NEW%';
```



| | station_id | station_name | elevation | lat | lon |
|---|---|---|---|---|---|
| | character varying (20) | text | numeric (6,1) | numeric (9,6) | numeric (9,6) |
| 1 | WBAN:14734 | NEWARK LIBERTY INTERNATIONAL AIRPORT NJ US | 2.1 | 40.682500 | -74.169400 |

Find stations where NY is embedded in the value

```
SELECT *
FROM nyc.weather_stations
WHERE station_name LIKE '%NY%';
```



| | station_id | station_name | elevation | lat | lon |
|---|---|---|---|---|---|
| | character varying (20) | text | numeric (6,1) | numeric (9,6) | numeric (9,6) |
| 1 | WBAN:14732 | LA GUARDIA AIRPORT NY US | 3.4 | 40.779200 | -73.880000 |
| 2 | WBAN:94789 | JFK INTERNATIONAL AIRPORT NY US | 3.4 | 40.638600 | -73.762200 |
| 3 | WBAN:94728 | NY CITY CENTRAL PARK NY US | 42.7 | 40.778980 | -73.969250 |
| 4 | WBAN:94745 | WESTCHESTER CO AIRPORT NY US | 115.5 | 41.066940 | -73.707500 |
| 5 | WBAN:54787 | FARMINGDALE REPUBLIC AIRPORT NY US | 24.7 | 40.734170 | -73.416940 |

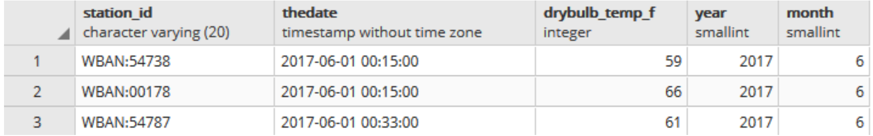
## Timestamps

Year and month stored in their own field

```
SELECT station_id, thedate, drybulb_temp_f
FROM nyc.weather_daily
WHERE year=2017 AND month BETWEEN 6 AND 8;
```

Pull data out of timestamp field

```
SELECT station_id, thedate, drybulb_temp_f
FROM nyc.weather_daily
WHERE EXTRACT (YEAR FROM thedate)=2017
AND EXTRACT (MONTH FROM thedate) BETWEEN 6 AND
  8;
```

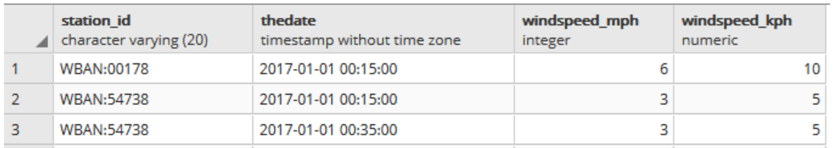


| | station_id | thedate | drybulb_temp_f | year | month |
|---|---|---|---|---|---|
| | character varying (20) | timestamp without time zone | integer | smallint | smallint |
| 1 | WBAN:54738 | 2017-06-01 00:15:00 | 59 | 2017 | 6 |
| 2 | WBAN:00178 | 2017-06-01 00:15:00 | 66 | 2017 | 6 |
| 3 | WBAN:54787 | 2017-06-01 00:33:00 | 61 | 2017 | 6 |

## Calculated Fields

Used for summarizing data across columns. Use AS to provide an alias for the new column, symbols to do the math (+ - * /)

```
SELECT station_id, thedate, windspeed_mph,
ROUND(windspeed_mph * 1.609344) AS windspeed_kph
FROM nyc.weather_daily
WHERE year=2017 AND month=1
ORDER BY thedate;
```

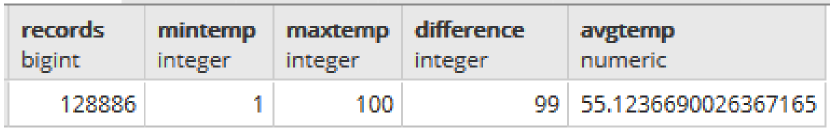


| | station_id | thedate | windspeed_mph | windspeed_kph |
|---|---|---|---|---|
| | character varying (20) | timestamp without time zone | integer | numeric |
| 1 | WBAN:00178 | 2017-01-01 00:15:00 | 6 | 10 |
| 2 | WBAN:54738 | 2017-01-01 00:15:00 | 3 | 5 |
| 3 | WBAN:54738 | 2017-01-01 00:35:00 | 3 | 5 |

## Group Functions

Use to summarize data across rows. If your statement only contains group functions, you summarize the entire table:

```
SELECT COUNT(weather_id) AS records,
MIN(drybulb_temp_f) AS mintemp,
MAX(drybulb_temp_f) AS maxtemp,
MAX(drybulb_temp_f) − MIN(drybulb_temp_f) AS difference,
AVG(drybulb_temp_f) AS avgtemp
FROM nyc.weather_daily
WHERE year=2017;
```

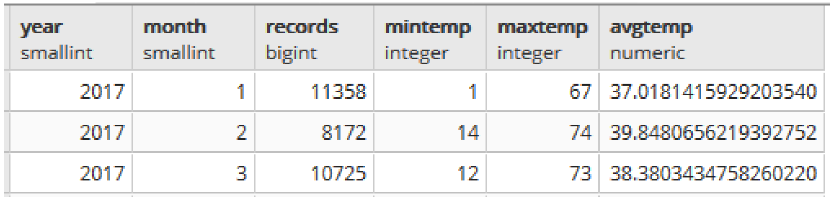


| records<br>bigint | mintemp<br>integer | maxtemp<br>integer | difference<br>integer | avgtemp<br>numeric |
|---|---|---|---|---|
| 128886 | 1 | 100 | 99 | 55.1236690026367165 |

## Group By

Use to summarize data by specific attributes across rows

```
SELECT year, month,
COUNT(weather_id) AS records,
MIN(drybulb_temp_f) AS mintemp,
MAX(drybulb_temp_f) AS maxtemp,
AVG(drybulb_temp_f) AS avgtemp
FROM nyc.weather_daily
WHERE year=2017
GROUP BY year, month
ORDER BY month;
```

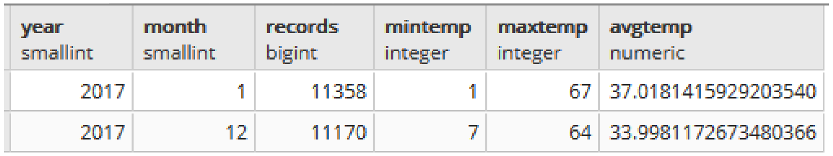


| year<br>smallint | month<br>smallint | records<br>bigint | mintemp<br>integer | maxtemp<br>integer | avgtemp<br>numeric |
|---|---|---|---|---|---|
| 2017 | 1 | 11358 | 1 | 67 | 37.0181415929203540 |
| 2017 | 2 | 8172 | 14 | 74 | 39.8480656219392752 |
| 2017 | 3 | 10725 | 12 | 73 | 38.3803434758260220 |

## Having

Use to specify criteria for grouped values as opposed to individual row values

```
SELECT year, month,
COUNT(weather_id) AS records,
MIN(drybulb_temp_f) AS mintemp,
MAX(drybulb_temp_f) AS maxtemp,
AVG(drybulb_temp_f) AS avgtemp
FROM nyc.weather_daily
WHERE year=2017
GROUP BY year, month
HAVING MAX(drybulb_temp_f) < 70
ORDER BY month;
```



| year<br>smallint | month<br>smallint | records<br>bigint | mintemp<br>integer | maxtemp<br>integer | avgtemp<br>numeric |
|---|---|---|---|---|---|
| 2017 | 1 | 11358 | 1 | 67 | 37.0181415929203540 |
| 2017 | 12 | 11170 | 7 | 64 | 33.9981172673480366 |

## Distinct Values, Null Values

Grab distinct instances of a value

```
SELECT DISTINCT station_id
FROM nyc.weather_daily;
```

Nulls represent missing values or the absence of data. Capture with IS NULL or IS NOT NULL

```
SELECT station_id, thedate, drybulb_temp_f
FROM nyc.weather_daily
WHERE drybulb_temp_f IS NULL
AND year=2017;
```

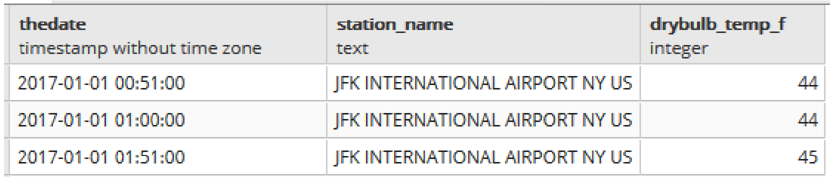
## Today's Topics

- Relational database fundamentals
- SQL DML - SELECT
- SQL DML - JOINS
- Next Class

## Joins

Tie tables together using unique ID columns, use table alias

```
SELECT d.thedate, s.station_name, d.drybulb_temp_f
FROM nyc.weather_stations s, nyc.weather_daily d
WHERE s.station_id=d.station_id
AND s.station_id = 'WBAN:94789'
AND d.year=2017
ORDER BY thedate;
```

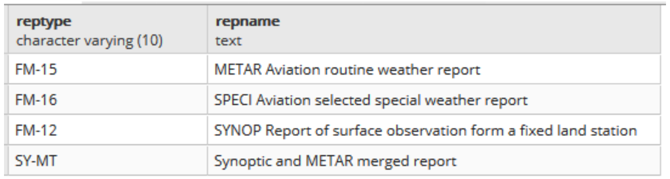


| thedate<br>timestamp without time zone | station_name<br>text | drybulb_temp_f<br>integer |
|---|---|---|
| 2017-01-01 00:51:00 | JFK INTERNATIONAL AIRPORT NY US | 44 |
| 2017-01-01 01:00:00 | JFK INTERNATIONAL AIRPORT NY US | 44 |
| 2017-01-01 01:51:00 | JFK INTERNATIONAL AIRPORT NY US | 45 |

## Inner Joins

Two syntaxes, same result - keep matching records from tables

```
SELECT DISTINCT d.reptype, rep.repname
FROM nyc.weather_reptype rep, nyc.weather_daily d
WHERE d.reptype=rep.reptype;
```



| reptype<br>character varying (10) | repname<br>text |
|---|---|
| FM-15 | METAR Aviation routine weather report |
| FM-16 | SPECI Aviation selected special weather report |
| FM-12 | SYNOP Report of surface observation form a fixed land station |
| SY-MT | Synoptic and METAR merged report |

```
SELECT DISTINCT d.reptype, rep.repname
FROM nyc.weather_reptype rep
INNER JOIN nyc.weather_daily d
ON (d.reptype=rep.reptype);
```

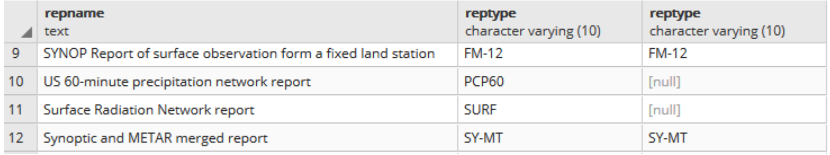
Note the DISTINCT command is being used here as an example - it is not an implicit part of the Join statements.

Leave it out, and you'll return reptype and repname for every record.

## Left Outer Join

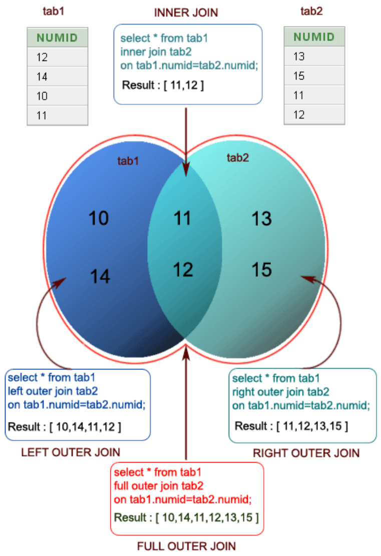
Keep all records from table on the left (in the FROM clause) regardless of whether there are matching records in the other

```
SELECT DISTINCT rep.repname, rep.reptype, d.reptype
FROM nyc.weather_reptype rep
LEFT OUTER JOIN nyc.weather_daily d
ON  (d.reptype=rep.reptype);
```



|  | repname<br>text | reptype<br>character varying (10) | reptype<br>character varying (10) |
|---|---|---|---|
| 9 | SYNOP Report of surface observation form a fixed land station | FM-12 | FM-12 |
| 10 | US 60-minute precipitation network report | PCP60 | [null] |
| 11 | Surface Radiation Network report | SURF | [null] |
| 12 | Synoptic and METAR merged report | SY-MT | SY-MT |

## Types of Joins



tab1

| NUMID |
|---|
| 12 |
| 14 |
| 10 |
| 11 |

tab2

| NUMID |
|---|
| 13 |
| 15 |
| 11 |
| 12 |

```
select * from tab1
inner join tab2
on tab1.numid=tab2.numid;

Result : [ 11,12 ]
```

```
select * from tab1
left outer join tab2
on tab1.numid=tab2.numid;

Result : [ 10,14,11,12 ]
```

```
select * from tab1
right outer join tab2
on tab1.numid=tab2.numid;

Result : [ 11,12,13,15 ]
```

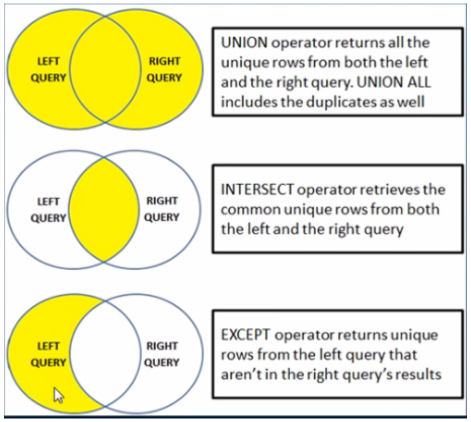
```
select * from tab1
full outer join tab2
on tab1.numid=tab2.numid;

Result : [ 10,14,11,12,13,15 ]
```

Image source: http://www.w3resource.com/sql/joins/sql-joins.php

## Combining Results by Rows

These operations require that the structure of the two tables are compatible: same number of columns and same data types.



UNION operator returns all the unique rows from both the left and the right query. UNION ALL includes the duplicates as well

INTERSECT operator retrieves the common unique rows from both the left and the right query

EXCEPT operator returns unique rows from the left query that aren't in the right query's results

Image source:
http://developer.e-power.com.kh/difference-between-union-intersect-except-in-sql-server/

## Example of Combining Results

Insert UNION, INTERSECT, or EXCEPT

```
(SELECT station_id, station_name
FROM nyc.weather_stations)
UNION
(SELECT station_id, station_name
FROM phl.weather_stations);
```

## Subqueries

Query within a query. Inner query passes a value, row, or table to outer query. Example for passing one value:

```
SELECT station_id, thedate, drybulb_temp_f
FROM nyc.weather_daily
WHERE year=2017 and month=1 and station_id =
  (SELECT station_id
  FROM nyc.weather_stations
  WHERE station_name LIKE 'JFK%');
```

## Subqueries with Many Results

Requires a key word: IN, ANY, ALL, EXISTS, NOT EXISTS

```
SELECT station_id, thedate, drybulb_temp_f
FROM nyc.weather_daily
WHERE year=2017 AND month=1 AND station_id IN
(SELECT station_id
FROM nyc.weather_stations
WHERE elevation > 50);
```

Subqueries can only return values from the outer query in the final result.

## Join vs. Subquery

If you can accomplish a query with a join, do that instead of using a subquery. This statement returns the same result as the one on the previous slide':

```
SELECT d.station_id, d.thedate, d.drybulb_temp_f
FROM nyc.weather_stations s, nyc.weather_daily d
WHERE d.year = 2017 AND d.month=1 AND s.elevation > 50
AND s.station_id=d.station_id;
```

## Today's Topics

Relational database fundamentals

SQL DML - SELECT

SQL DML - JOINS

Next Class

## Due Next Class

The following are due at the beginning of our next class:

Assignment #2

Posted on the course website (under Assignments)
*No class next week, assignment due Feb 12 by 9:30pm*

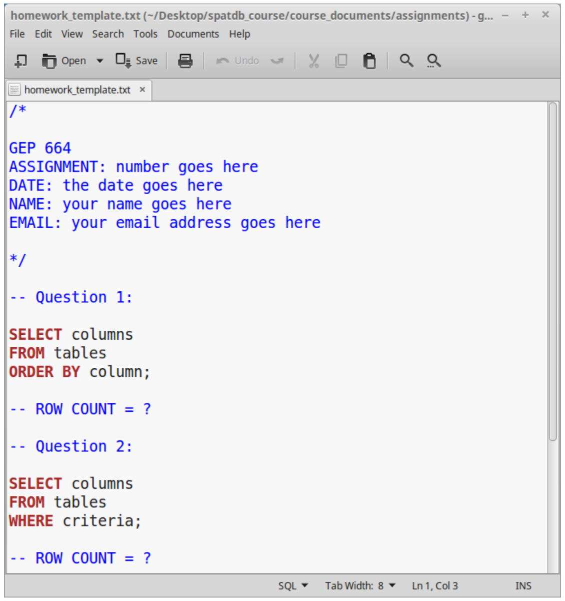
Readings for Class #3

Listed in the syllabus, in the *Practical SQL* book
Note: There is overlap in course content and readings for classes 2 & 3

Homework Guidelines

Instructions and homework template are posted on the course website (Under Readings and Docs)

## Homework Template


homework_template.txt (~/Desktop/spatdb_course/course_documents/assignments) - g...
File Edit View Search Tools Documents Help
Open
Save
Undo
homework_template.txt ×
SQL
Tab Width: 8
Ln 1, Col 3
INS

```
/*

GEP 664
ASSIGNMENT: number goes here
DATE: the date goes here
NAME: your name goes here
EMAIL: your email address goes here

*/

-- Question 1:

SELECT columns
FROM tables
ORDER BY column;

-- ROW COUNT = ?

-- Question 2:

SELECT columns
FROM tables
WHERE criteria;

-- ROW COUNT = ?
```

## Sample Data - NYS Labor Force Data

The nys schema in the gep664 database has NY State labor force data by county from the Census. Select each table and view the properties to see a full description

nys.metadata : description of columns in each table

nys.metros : list of all counties that are part of metropolitan areas that fall within NY State

nys.popworkers : number of people that live in each county (population) and that work in each county (workers)

nys.resworkers : resident workers in each county; describes whether residents work where they live, or work elsewhere