Announcements

- Sample Midterm 1 Quiz on Canavs

SQL Basics

"**S**tructured **Q**uery **L**anguage"

- It is NOT much like other programming languages
- It is NOT PROCEDURAL
- It does not process one record at a time, rather, it is a SET processing language
- All inputs to SQL are tables
- The output from a query is a table
- Output from a query referred to as the "Answer Set"
- Some queries may produce "interim" temporary answer sets

SQL Basics

"Structured Query Language"

- It is a relatively simple language – brief syntax, few commands
- It is a relatively powerful language – a FEW lines of code can accomplish a LOT of work
- ANSI (American National Standards Institute) maintains a specification for "standard" SQL
- Each DBMS manufacturer follows the ANSI standard, but also adds extended features unique to their SQL

SQL Basics

CREATE DATABASE statement

```
CREATE DATABASE <database>;
```

PostgreSQL always connects to a particular database.

To access a different database, you must create a new connection:

```
\c <database>;
```

SQL Basics

DDL = Data Definition Language

Some SQL commands are used to DEFINE or MODIFY the structures in the database.

```
Create
Alter
Drop
```

DML = Data Manipulation Language

Some SQL commands are used to MODIFY the data in the database

```
Update
Insert
Delete
```

CREATE statement

```
CREATE TABLE IF NOT EXISTS <table name>(
   column  DATATYPE(L) UNIQUE,
   column  DATATYPE(L) NOT NULL,
   column  DATATYPE(L) CHECK <condition>,
   column  DATATYPE(L) PRIMARY KEY
);
```

CREATE football_games

```
CREATE TABLE IF NOT EXISTS football_games (
     visitor_name VARCHAR(30),
     home_score SMALLINT NOT NULL,
     visitor_score SMALLINT NOT NULL,
     game_date DATE NOT NULL,
     players INT[] NOT NULL,
     PRIMARY KEY(visitor_name, game_date)
);
```

CREATE football_players

```
CREATE TABLE IF NOT EXISTS football_players(
     id SERIAL PRIMARY KEY,
     name VARCHAR(50) NOT NULL,
     year VARCHAR(3),
     major VARCHAR(4),
     passing_yards SMALLINT,
     rushing_yards SMALLINT,
     receiving_yards SMALLINT,
     img_src VARCHAR(200)
);
```

ALTER statement

```
ALTER TABLE <table name>
   ADD COLUMN column DATATYPE(L),
   CONSTRAINT <constr name>

ALTER TABLE <table name>
   DROP CONSTRAINT <constr name>

ALTER TABLE <table name>
   DROP COLUMN IF EXISTS <column name>
```

SQL Basics

INSERT statement

```
INSERT INTO <table name>
   VALUES (value, value, value, value);
```

(must have a value or NULL for every column in the table)

```
INSERT INTO <table name> (column, column, column)
   VALUES (value, value, value);
```

(if no column/value is specified, NULL or default will be assigned)

INSERT DATA

```
INSERT INTO football_games(visitor_name, home_score, visitor_score, game_date,
players)
       VALUES('Colorado State', 45, 13, '20180831', ARRAY [1,2,3,4,5]),
       ('Nebraska', 33, 28, '20180908', ARRAY [2,3,4,5,6]),
       ('New Hampshire', 45, 14, '20180915', ARRAY [3,4,5,6,7]),
       ('UCLA', 38, 16, '20180928', ARRAY [4,5,6,7,8]),
       ('Arizona State', 28, 21, '20181006', ARRAY [5,6,7,8,9]),
       ('Southern California', 20, 31, '20181013', ARRAY [6,7,8,9,10]),
       ('Washington', 13, 27, '20181020', ARRAY [7,8,9,10,1]),
       ('Oregon State', 34, 41, '20181027', ARRAY [8,9,10,1,2]),
       ('Arizona', 34, 42, '20181102', ARRAY [9,10,1,2,3]),
       ('Washington State', 7, 31, '20181110', ARRAY [10,1,2,3,4]),
       ('Utah', 7, 30, '20181117', ARRAY [1,2,3,4,5]),
       ('California', 21, 33, '20181124', ARRAY [2,3,4,5,6]);
```

INSERT DATA

```
INSERT INTO football_players(name, year, major, passing_yards,
rushing_yards, receiving_yards)
      VALUES('Cedric Vega', 'FSH', 'ARTS', 15, 25, 33),
      ('Myron Walters', 'SPH', 'CSCI', 32, 43, 52),
      ('Javier Washington', 'JNR', 'MATH', 1, 61, 45),
      ('Wade Farmer', 'SNR', 'ARTS', 14, 55, 12),
      ('Doyle Huff', 'FSH', 'CSCI', 23, 44, 92),
      ('Melba Pope', 'SPH', 'MATH', 13, 22, 45),
      ('Erick Graves', 'JNR', 'ARTS', 45, 78, 98 ),
      ('Charles Porter', 'SNR', 'CSCI', 92, 102, 125),
      ('Rafael Boreous', 'JNR', 'MATH', 102, 111, 105),
      ('Jared Castillo', 'SNR', 'ARTS', 112, 113, 114);
```

SQL Basics

SELECT statement

```
SELECT <column1>, <column2>, <column3>,
<literal>, <math expression>
   FROM  <table A> ;
```

SQL Basics

SELECT statement

- Literals may be either `'Character'` (in quotes) or Numeric
- Math expressions

  Only use with columns defined as numeric data types

+	Add
-	Subtract
*	Multiply
/	Divide
**	Exponent

SQL Basics

SELECT statement

- Rename a column in the answer set with “AS”

```
SELECT home_score AS ‘CU Score’
```

- Concatenate character columns with

```
CONCAT(<column1>,column2>)
```

- Comment out a line or part of a line of code by prefixing it with “- -” or embedding a “#”

- Limit the size of the answer set with “limit” (MySQL)

```
SELECT passing_yards
FROM football_players LIMIT 5;
```

SQL Basics

SELECT statement

```
SELECT <column1>, <column2>, <column3>,
<literal>, <math expression> AS <label>
  FROM  <table A>
  WHERE  <condition>;
```

SQL Basics

- The WHERE clause results in a subset of ROWs to appear in the answer set
- The condition in the WHERE clause takes this format:

  < operand > < operator > < operand >

- Operands may be columns or literals or expressions
- Operator may be

=	Equals	Like
<>	Not equals	Between
>	Greater than	In
<	Less than	

SQL Basics

- Operator may be: In or Like

  In (literal, literal, literal)

  Like `'string'` with % or _ as a wildcard

- Multiple conditions may be joined with Boolean operators

  `AND, OR`

- Conditions may be negated with Boolean operator

  `NOT`

- Answer Set rows may be sorted with “Order By”
- Order By defaults to Ascending, can specify DESC

SQL Basics

`Distinct`

- The answer set may contain duplicate rows
- The "`distinct`" keyword before a column removes duplicates
- How many distinct majors are they from?

SQL Basics

```
SELECT major
   FROM football_players;
```

```
SELECT DISTINCT major
   FROM football_players
```

```
SELECT DISTINCT major, name
   FROM football_players;
```

Handling Dates in MySQL

- MySQL supports DATE, DATETIME, and TIMESTAMP data types
- Columns with a data type of "TIMESTAMP" are stored as a 4-byte binary integer representing the number of seconds since 1970-01-01 00-00-00 UTC. TIMESTAMP has a range of '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC.
- If no value is provided for the TIME portion of a DATETIME column, it defaults to 00:00.00.0000

Handling Dates in MySQL

- To make it easier for humans to deal with date/time, MySQL allows us to reference dates/times in this format:
  - YYYY-MM-DD and HH:MM.SS.nnn

- If you pass the date to MySQL as text in YYYY-MM-DD format, it will automatically convert it to the proper binary number

- If you pass the time to MySQL as text in HH:MM.SS.nnn format, it will automatically convert it to the proper binary number

YYYY-MM-DD and hh:mm.ss.nnn

- **YYYY** is **four digits** from 1000 through 9999 that represent a year.
- **MM** is **two digits**, ranging from 01 to 12, that represent a month in the specified year.
- **DD** is **two digits**, ranging from 01 to 31 depending on the month, that represent a day of the specified month.
- **hh** is **two digits**, ranging from 00 to 23, that represent the hour.
- **mm** is **two digits**, ranging from 00 to 59, that represent the minute.
- **ss** is **two digits**, ranging from 00 to 59, that represent the second.
- **nnn** is **zero to three digits**, ranging from 0 to 999, that represent the fractional seconds.

SQL Basics

```
SELECT NOW();

SELECT visitor_name, EXTRACT(Month FROM game_date) AS “Month”
    FROM football_games;

SELECT * FROM football_games WHERE game_date > CURRENT_DATE -
    INTERVAL '12 months';
```

SQL Basics

```
SELECT DATE_FORMAT(game_date,'%b %d %Y %h:%i %p’
    FROM football_games;
```

Format	Description
%a	Abbreviated weekday name (Sun-Sat)
%b	Abbreviated month name (Jan-Dec)
%c	Month, numeric (0-12)
%D	Day of month with English suffix (0th, 1st, 2nd, 3rd, □)
%d	Day of month, numeric (00-31)
%e	Day of month, numeric (0-31)
%f	Microseconds (000000-999999)
%H	Hour (00-23)
%h	Hour (01-12)
%I	Hour (01-12)
%i	Minutes, numeric (00-59)
%j	Day of year (001-366)
%k	Hour (0-23)
%l	Hour (1-12)

http://www.w3schools.com/sql/func_date_format.asp

SQL Basics

SQL provides the following GROUP FUNCTIONS

SUM - Provides the sum of the values in a column across many rows

AVG - Provides the average of the values in a column across many rows

COUNT - Provides a count of how many rows have a value in a column, counted across many rows

MIN - Provides the lowest value in a column across many rows

MAX - Provides the highest value in a column across many rows

GROUP FUNCTIONS

- SUM, AVG must only be used with NUMERIC columns
- MIN, MAX can be used with any data type
- COUNT can be used with any column, or with a (*) to simply count rows
- **Group functions require SQL to create an interim answer set**, and then process the group function against the interim answer set, delivering a final answer set that contains only the final total for the function. **Always returns an integer value**.
- When you combine a GROUP FUNCTION with a WHERE clause, keep in mind that the **WHERE clause simply reduces the number of rows in the INTERIM answer set** before the GROUP function does its calculation.

SQL Basics

```
SELECT COUNT(*) AS “Total Games Played”
      FROM football_games;

SELECT COUNT(DISTINCT major) AS “Different Majors”
      FROM football_players;

SELECT SUM(passing_yards) AS “Total Passing Yards”
   FROM football_players WHERE major = ‘ARTS’;

SELECT MIN(passing_yards)
   FROM football_players
   WHERE passing_yards > 15;

SELECT AVG(passing_yards) AS “Average Passing Yards”
   FROM football_players;
```

GROUP BY

- Group functions process against an interim answer set to return a value across many rows.
- Using a GROUP BY clause enables SQL to provide subtotals. The GROUP BY tells SQL to **perform the group function against a subset of rows in the interim answer set** and provide a total for each subset of rows.
- **When using a GROUP BY every column in the SELECT statement must either be a GROUP FUNCTION or a COLUMN that you are grouping by**.

SQL Basics

Why are the following two statements the same?

```
SELECT major, COUNT(*) AS “Total”
      FROM football_players
      GROUP BY major;

SELECT major, COUNT(major) AS “Total”
      FROM football_players
      GROUP BY major;
```

Why is the following statement incorrect?

```
SELECT name, major, COUNT(major) AS “Total”
      FROM football_players;
      GROUP BY major;
```

SQL Basics

Which major has the most passing_yards?

```
SELECT major, SUM(passing_yards) AS “Total passing yards”
   FROM football_players
   GROUP BY major;
```

In which month in 2018 did CU score the most points?

```
 SELECT EXTRACT(Month FROM game_date) AS “Month”,
   SUM(home_score) AS “Total score”
   FROM football_games
   WHERE EXTRACT(Year FROM game_date) = ‘2018'
   GROUP BY EXTRACT(Month FROM game_date)
   ORDER BY SUM(home_score) DESC
   LIMIT 1;
```

SQL Basics

HAVING

- Is simply like a WHERE clause against the answer set when you use a GROUP BY

SQL Basics

```
SELECT major, COUNT(*) AS “Count”
       FROM football_players
       GROUP BY major;
```

Which majors have more than 2 seniors on the team?

```
SELECT major, COUNT(*) AS “Total”
       FROM football_players
       GROUP BY major
       HAVING  COUNT(*) > 3;
```

SubQuery

Simply: a query within a query. The answer set to an “inner” query is used as a predicate in a where clause in the “outer” query.

- The subquery must **return only one column**.
- If the outer query WHERE clause contains an **“equals”** condition, the subquery must return **ONE row**.
- If the outer query WHERE clause contains an **“in”** condition, the subquery may return **multiple rows**, presented as a list of values.
- The Subquery is embedded within parentheses
- Outer and inner queries can hit two different tables

SQL Basics

```
SELECT name
   FROM football_players
   WHERE passing_yards = (
      SELECT MAX(passing_yards)
            FROM football_players );
```

Note that with the “equals” condition, the inner query returns only one value (one row, one column)

SQL Basics

```
SELECT name FROM football_players
   WHERE id = ANY (
      SELECT unnest(players) FROM football_games
            WHERE home_score > visitor_score )
   ORDER BY name;
```

Note that with the “ANY” condition, the inner query returns many values (many rows, one column) as a list

This also uses two different tables

SQL Basics

UPDATE statement

```
UPDATE <table name>
   SET column = <value>
   WHERE <condition>
```

NOTE: if the WHERE is omitted, ALL rows are updated.

DELETE statement

```
DELETE FROM <table name>
   WHERE <condition>
```

NOTE: if the WHERE is omitted, ALL rows are deleted.

SQL Basics

DROP statement

```
DROP TABLE <table name>
```

STOP

STOP