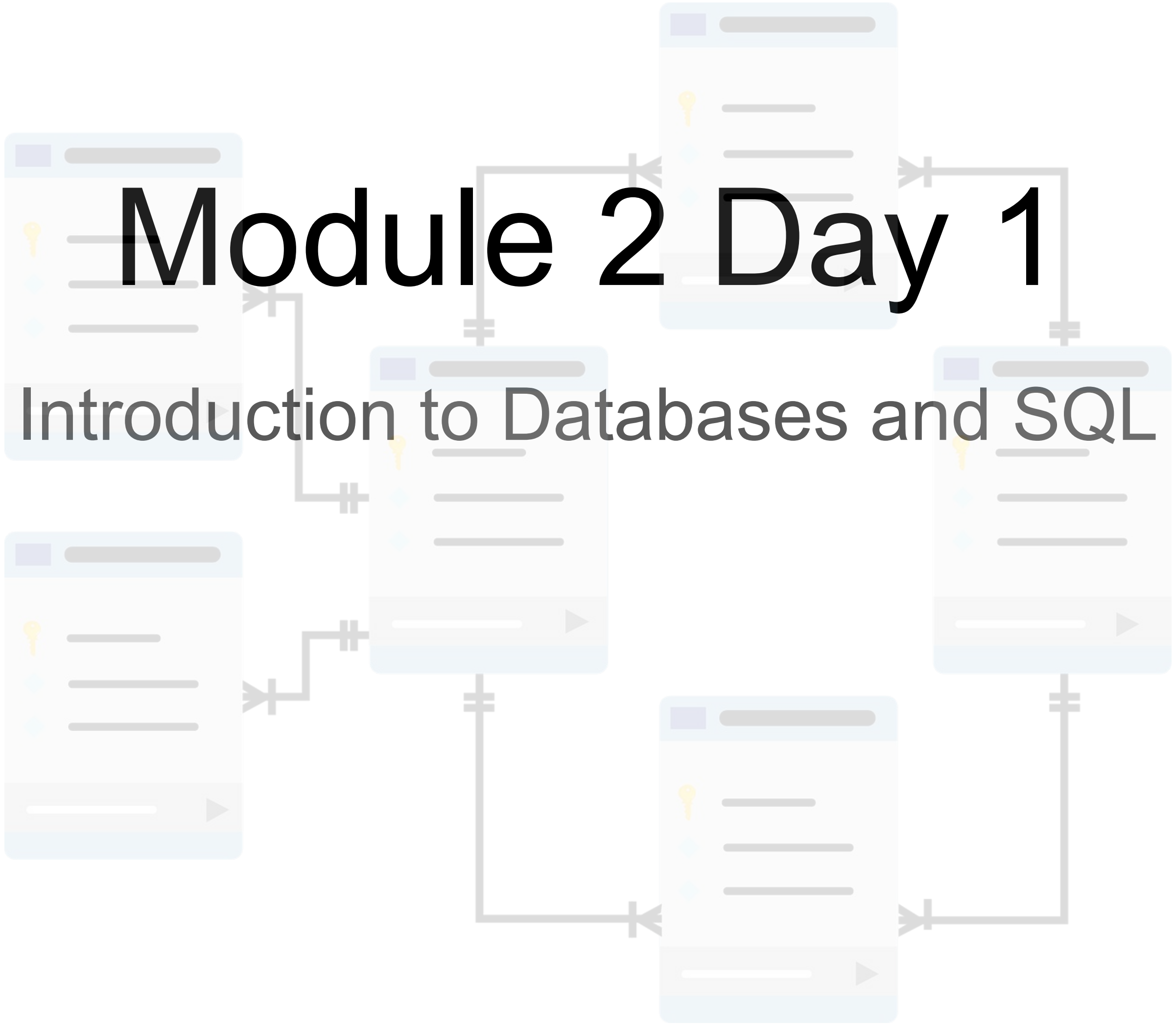
Module 2 Day 1

Introduction to Databases and SQL



Module 2 Day 1 Intro to SQL

Can you … ?

- … name examples of **RDBMS** products and explain their benefits
- … explain the concept of **ATTRIBUTES** and **ENTITIES** as they relate to Tables within an RDBMS?
- … construct basic **SELECT** statements
- … limit results using a **WHERE** clause
- … explain and demonstrate the purpose and use of **ALIASES**

Let’s get setup!

Databases

- A database is an electronically stored organized collection of data.
- A **relational database** is one in which the data is organized around columns and tables and relationships between tables are defined using row elements:
    - A table is designed to store an **entity**, a data representation of a real world object.
    - Each row of a table represents one instance of the entity.
    - The columns represent attributes the entity might have.
- Relational databases are a class of databases. The suite of tools used to host and manage them are referred to as Relational Database Management Systems (RDBMS) Some examples of popular RDBMS include:
    - Oracle, MySQL, MS SQL Server, PostgreSQL, and DB2

Talking to a Relational Database: SQL

- **SQL** is an acronym for **Structured Query Language**
- SQL is the language used to interact with relational database management systems.
- The exact implementation of SQL varies slightly depending on the database system involved, i.e. there will be minor differences in the language between PostgreSQL and MS SQL Server.
- This class will be using PostgreSQL.

Relational Database: Attribute Data Types

There is a large variety of data types in Postgresql, to name a few:

- **varchar**: holds text containing letters and numbers (somewhat like a String in Java) of *varying length* up to a maximum length.
- **char**: *fixed length* field containing a stream of characters.
- Actually, there are quite a few, let's take a look: PostgreSQL Data Type Reference

- When assigning values to a "text" field (i.e. varchar or char), we must surround that data with single quotes (i.e. country=**'USA'**).
- Numeric literals do not use quotes (numberOfDoors = **4**).

Relational Databases: The Table

Suppose we are interested in storing data about cars. We can model car ***entities*** using a table dedicated to storing the attributes that define a single car:

This table has 4 Columns. Columns store attributes:
CarName, Manufacturer, NumberOfDoors, FuelEconomy

This table has 3 rows. Each row represents an Entity.

CarName	Manufacturer	NumberOfDoors	FuelEconomy
Explorer	Ford	4	23
C-Class	Mercedes Benz	4	28
Jeep Wrangler	Fiat Chrysler	2	20

Using SQL to Talk to the DB: SELECT Statement

- The most basic SQL statement is SELECT. It is used to retrieve rows of data and it follows the following format:

SELECT **[column]**, **[column-n] AS aliasname** FROM **[table]**;

- **[column]** and **[column-n]** are stand ins for the attributes or columns that you want returned from your query.

- **[table]** refers to the name of the table you are querying.

- You can create column Aliases using the “**AS**” keyword followed by the alias.

Using SQL to Talk to the DB: SELECT Example

Let’s take the Vehicle table we just saw as an example:

- We could write the following SELECT statement:

  SELECT CarName, NumberOfDoors AS doors FROM Vehicle;

  The output of this would be:

CarName	doors
Explorer	4
C-Class	4
Jeep Wrangler	2

The alias becomes the name assigned to the column in the query output.

- Instead of listing specific columns we could also use the wildcard * to indicate that all columns should be returned: *SELECT * FROM Vehicle;*

Using SQL to Talk to the DB: The WHERE clause

- We can include a WHERE clause in our select statements to limit the data returned by specifying a condition.
- The WHERE statement relies on comparison operators.
  - **Greater Than**: **>**
  - **Greater Than or Equal To**: **>=**
  - **Less Than**: **<**
  - **Less Than or Equal To**: **<=**
  - **Equa**l: **=**
  - **Not Equal To**: **<>**
- There is a special comparison operator called **LIKE** which is often used in conjunction with a wildcard (**%**) operator.

SQL: SELECT with WHERE clause Example 1

Let’s take the Vehicle table we just saw as an example:

- We could write the following SELECT statement:

  SELECT * FROM Vehicle WHERE Manufacturer = ‘Ford’;

- Only 1 row matches this criteria, and the results of the query will be:

CarName	Manufacturer	NumberOfDoors	FuelEconomy
Explorer	Ford	4	23

SQL: SELECT with WHERE clause Example 2

Here is an example of the WHERE clause using the LIKE / Wildcard.

- We could write the following SELECT statement:

  SELECT * FROM Vehicle WHERE CarName like ‘Ex%’;

- Only 1 row matches this criteria, and the results of the query will be:

CarName	Manufacturer	NumberOfDoors	FuelEconomy
Explorer	Ford	4	23

Derived Columns with Math Operations

A custom field containing math operations can be included in the SELECT.

- The basic math operators are available: **+**, **-**, *****, **/**, **%**
- **Enforce your desired order of operations using ( )**
  SQL will honor basic PEMDAS OoO, but Parenthesis make your intent clear and explicit to the compiler and humans
- **Derived columns require an Alias or the compiler will add one for you.**
  Relying on the default/auto Alias is discouraged as it is unpredictable and fails to describe the attribute when the column is returned.

Derived Columns Example

- Consider the following example:

SELECT CarName, FuelEconomy * 0.425144 AS kpl FROM Vehicle;

CarName	kpl
Explorer	9.778312
C-Class	9.778312
Jeep Wrangler	8.50288

SQL: AND / OR in WHERE statements

- Within the WHERE statement, various filter conditions can be combined using the AND / OR statement.
- Using the following Select statement as an example:
  SELECT * FROM Vehicle WHERE Manufacturer = 'Ford' OR NumberOfDoors = 4;

  Two rows are returned:

CarName	Manufacturer	NumberOfDoors	FuelEconomy
Explorer	Ford	4	23
C-Class	Mercedes Benz	4	28