

# Module 2-1

Introduction to Databases and SQL

# 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:
  - 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 Database: Example

Suppose we are interested in storing data about cars. We can model a car entity into its own table:

This table has 4 attributes: CarName, Manufacturer, NumberOfDoors, FuelEconomy



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

This table has 3 rows.



# 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).
- **char**: fixed length field containing a stream of characters.
- Various numeric data types: <https://www.postgresql.org/docs/9.3/datatype-numeric.html>
- When referring to a non-numeric “text” field (i.e. varchar or char) we must surround them in single quotes (i.e. country=**'USA'**).
- Numeric literals do not need single quotes (numberOfDoors = **4**).

# 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.

# SQL: SELECT

- The most basic SQL statement is a SELECT query, and it follows the following format:

SELECT **[column]**, **[column-n]** 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.

# SQL: 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

Note how the alias affects the column name in the output.

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

# SQL: SELECT with 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:** <=
  - **Equal:** =
  - **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 thus 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 thus 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 present: **+**, **-**, **\***, **/**, **%**

# 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 on WHERE statements

- Within the WHERE statement, various filter conditions can be combined using the AND / OR statement.
- Consider the following 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

Let's get setup!