SQL Notes

What is SQL

* **Structured Query Language (SQL)** - standard computer language for relational database management and data manipulation
  + used to query, insert, update and modify data
* used to communicate with databases
* statements are made up of descriptive words and are easy to learn

SQL is a **non-procedural language**:

* cannot write complete applications
* simple, but powerful

SQL used for 3 things:

1. Read/retrieve data
2. Write data – add data to a table
3. Update data – insert new data

DBMS – Database Management System

**Database (DBA) Administrator**

* Manages/governs entire database
* Gives permissions to users
* Determines access to data
* Manages and creates tables
* Uses SQL to query and retrieve data

Data Scientist

* End user of a database
* Uses SQL to query and retrieve data

Data scientists may:

* Retrieve data
* Create their own table or test environment
* Combine multiple sources together
* Writes complex queries for analysis

We’re using SQLite in this course

Databases and Tables

* Database
  + A container (usually a file or set of files) to store organized data; a set of related information
* Tables
  + A structured list of data or a specific type
* Columns
  + Single field in a table: all tables are made up of one or more columns
* Row
  + A record in a table

Data Modeling – organizes and structures information into multiple, related tables

* Can represent a business process or show relationships between them

Models for prediction vs a DATA MODEL; which represents the organization of tables in a database

NoSQL – Not Only SQL

* Mechanism for storage and retrieval of unstructured data modeled by means other than tabular relations in relational databases

Relational vs Transactional Databases

Relational – allows for easy querying and data manipulation in an easy, logical and intuitive way

Transactional – operational database – insurance claims or hospital records

* Transactional data can be extracted and moved to relational models for querying and data modeling

Data Model Building Blocks

**Entity** – person, place, thing, or even (distinguishable, unique, and distinct)

**Attribute** – a characteristic of an entity

**Relationship** – describes association among entities

One – to – many (customer with many invoices)

Many – to many (students have multiple classes or class has multiple students)

One – to – one (manager of a store, because only one in each)

ER Diagrams – composed of entity types and specifies relationships that can exist between instances of those entity types

* Show relationships
* Business process
* Represented visually
* Show links (primary keys)

Primary Keys and Foreign Keys

* Primary Keys – column (or set of columns) whose values uniquely identify every row in a table
* Foreign Keys – one or more columns that can be used together to identify a single row in another table

ER Diagram Notation

Chen Notation – 1:M, M:M, 1:1

Crow’s Foot Notation – train tracks represent 1, crow’s foot represents many

UML Class Diagram Notation – 1.1 represents 1, 1.\* represents many

Retrieving data from a SELECT statement

SELECT prod\_name (column)

FROM Products ;

* Retrieves prod\_name column and each item within

SELECT prod\_name, prod\_id, prod\_price

FROM Products;

* Retrieves all three columns from the data

SELECT \*

FROM Products;

* Retrieves every single column

LIMIT number of records

* Allows you to see a specific number of items from a column

Creating Tables

CREATE TABLE Shoes

(

Id char(10) PRIMARY KEY,

Brand char(10) NOT NULL,

Type char(250) NOT NULL,

Color char(250) NOT NULL,

Price decimal(8,2) NOT NULL,

Desc varchar(750) NULL

) ;

CHAR, DECIMAL, AND VARCHAR ARE THE DATA TYPES AND THE AMOUNT ALLOWED

Querying data

SELECT \*

FROM player

* selects all columns in the table “player”

SELECT

player\_name as name, birthday

FROM player

* selects the two columns in the table and relabels them as “name” and “birthday”
* if want to use “Full Name”, you must include quotes because of the space between

SELECT \*

FROM player

where weight>180 and height>190

where weight>180 or height>190

where weight between 180 and 190

where player\_name= ‘Aaron Galindo’

where player\_name like ‘Aaron Galindo’

* same as equals sign

where player\_name like ‘Aaron%’

* any data that starts with Aaron

where player\_name like ‘%Aaron’

* any data that ends with Aaron

where player\_name like ‘%Aaron%’

* any data that contains Aaron, even if a last name like “Aarons”

where player\_name like ‘A%n’

* any names that start with “A” and ends in “n”

where player\_name like ‘T\_m%’

* any names that start with “T” followed by another letter and then “m”
* Tim, Tamir, Temitope, Tamas

where player\_name in (‘Cristiano Ronaldo’, ‘Lionel Messi’)

* Pulls entries for both names
* In: multiple statements more simply but returns exact matches. Can’t use %

where home\_player\_1 is null

where home\_player\_1 is not null

SELECT

\*

FROM

player

order by weight desc

How to join tables

SELECT

player\_attributes.player\_api\_id,

player.player\_name,

player\_attributes.date,

player\_attributes.overall\_rating

FROM

player\_attributes

inner join player on player\_attributes.player\_api\_id = player.player\_api\_id

can make more succinct

SELECT

a.player\_api\_id,

b.player\_name,

a.date,

sum(a.overall\_rating)

FROM

player\_attributes a

inner join player b on a.player\_api\_id = b.player\_api\_id

to help group it

SELECT

a.player\_api\_id,

b.player\_name,

sum(a.overall\_rating) as rating

FROM

player\_attributes a

inner join player b on a.player\_api\_id = b.player\_api\_id

group by a.player\_api\_id,

b.player\_name

order by rating desc

* don’t include a.date, because we’re trying to sum them together, not get every entry

changing the sum function to avg so that we’ll include Ronaldo and Messi and add a separate column for the count of the ratings

SELECT

a.player\_api\_id,

b.player\_name,

avg(a.overall\_rating) as rating,

count(a.overall\_rating) as rating

FROM

player\_attributes a

inner join player b on a.player\_api\_id = b.player\_api\_id

group by a.player\_api\_id,

b.player\_name

order by rating desc

using having feature to find specific ranges of ratings

SELECT

a.player\_api\_id,

b.player\_name,

avg(a.overall\_rating) as rating,

count(a.overall\_rating) as rating

FROM

player\_attributes a

inner join player b on a.player\_api\_id = b.player\_api\_id

group by a.player\_api\_id,

b.player\_name

having rating>85

order by rating desc

* having must be after group by function and before order by function