SQL in Easy Steps

What is SQL?

- Databases allow collections of data to be stored in an organized manner. SQL commands are known as "queries" and utilize special keywords that can be used both to add data to a database, or extract details of data contained within a database.

Forms of SQL query

- How to execute SQL queries using a variety of popular software:
    - **Microsoft Access**
        - The SQL View allows you to enter SQL queries to be executed when you click !Run button
    - **Microsoft SQL Server**
        - The SQL Server DBMS products from Microsoft are popular. SQL queries can be executed from the SQL Server Management Studio
    - **Microsoft Visual Studio**
        - Visual Studio can be used to create computer programs that make queries against a database via an ODBC Data Source.
    - **Oracle**
        - The Oracle DBMS is popular and widely used in commerce.
    - **IBM DB2**
        - Powerful multi-platform database system.
    - **MySQL**
        - The world's most popular open-source database server
        - www.mysql.com/downloads/mysql

ODBC driver

- Usually, third-party client applications can only connect to the MySQL server if an appropriate Open DataBase Connectivity (ODBC) driver is installed on the system.
    - www.mysql.com/downloads/connector/odbc
    - Start > Control Panel > Administrative Tools > Data Sources (ODBC)
    - Choose the System DSN tab, click on the Add button, then Select the MySQL driver from the list and click finish
    - The default name for a local server is "localhost" and port 3306 is the default port used by the MySQL database server

Using Microsoft Query tool

- Can be used to make SQL queries to a database.
    - Launch Excel then select From Other Sources > From Microsoft Query on Excel's Data menu
    - When the Choose Data Source dialog appears select the MySQL Databases item then click OK
    - Add Table dialogs that appear to pen the Microsoft Query Window, then select File > Execute Query to open the Execute Query dialog
    - Type Show Databases into the SQL statement field of the Execute SQL dialog
    - Click the Execute button

Introducing databases

- Databases are simply convenient storage containers that store data in a structured manner.
    - At the mysql> prompt type exit, quit or \p then hit Return to close the MySQL monitor
- **Query that can be used to reveal all existing databases**
    - Show databases;
        - Terminate each SQL query with a semi-colon in the MYSQL monitor
        - It is conventional to use uppercase characters for all SQL keywords
- **Creating a database**
    - Query to create a brand new database
        - Create database *database-name*;
        - To first check to see if a database already exists:
            - Create database if not exists *database-name*;
                - Use only lowercase characters for all database names to avoid any confusion of case-sensitivity
- **Deleting a database**
    - Drop database *database-name*;
    - Drop database if exists *database-name*;

Running SQL scripts

- A number of SQL queries can be created as an SQL script which can then be run by MySQL. The script is simply a plain text file with a ".sql" file extension.
- Comments can usefully be added to SQL scripts to explain the purpose of particular queries.
- Multi-line C-style comments begin with "/*" and end with "*/"

Exploring database tables

- Tell MySQL which database to use with query:
  - Use *database-name*;
- View a list of all the tables it contains:
  - Show tables;
- To examine any table format its column specifications
  - Explain *table-name*;

Creating a table

- A new table can be created in a database
  - Create table *table-name*;
- To ensure that a table of the specified name does not already exist:
  - Create table if not exists *table-name*;
- This query must be followed by parentheses defining the name of each column and the type of data that it may contain. Each column definition is separated from the next by a comma

- **#list all databases**
  - Show databases;
- **Use the "my_database"**
  - Use *my_database*;
- **Create a table called "fruit" with 3 columns**
  - Create table if not exists *fruit*
  - (
  - Id Int,
  - Name Text,
  - Color text
  -);
- **Show that the table has been created**
  - Show tables;
- **Confirm the "fruit" table format**
  - Explain *fruit*;

Deleting a table

- A table can be deleted from a database using this query:
  - Drop table *table-name*;
- To ensure that a table of the specified name does not already exist:
  - Drop table if exists *table-name*;

- The deletion of both tables can be accomplished with a single drop table query:
    - Drop table if exists *dogs*, *fruit*;
    - The deletion is permanent and there is no "undo" facility

Table field modifiers

- Modifier keywords can optionally be used to further control column content:

Modifier	Purpose
Not Null	Insists that each record must include data
Unique	Insists that records may not duplicate any entry
Auto_Increment	Automatically generate a number that is one more than the previous value in that column
Default	Specifies a default value to be used
Primary Key	Specifies the column to be used as the primary key for that table

- **Example:**
    - "ID" column automatically numbers each row and the "qty" column will contain 1 unless another value is inserted. All other columns must contain data values
- **Use the "my_database"**
    - Use *my_database*;
- **Create a table called "products" with 5 columns**
    - Create table if not exists *products*
    - (
    - Id Int unique Auto_Increment,
    - Code int Not Null,
    - Name Varchar(25) Not Null,
    - Qty Int Default 1,
    - Price decimal (6,2) Not Null
- **Confirm the "Products" table format**
    - Explain *products*;
- **Delete this sample table**
    - Drop table *products*;

Setting the primary key

- A "constraint" that is applied to a column to uniquely identify each row of that database table.

    - Each field in that column must have a value
    - Each value in that column must be unique
- Primary key can be set elsewhere in the Create Table query by starting the name of the column in parentheses after the Primary Key keywords
- **Example:**
    - Use *my_database*;
    - Create table if not exists *cups*
        - (
        - Id Int auto_increment primary key,
        - Cup_pattern varchar(25)
        -);
    - Create table if not exists *saucers*
        - (
        - Id int auto_increment,
        - Scr_pattern varchar (25), Primary key (id)
        -);
    - Explain *cups*; explain *saucers*;
    - Drop table *cups*, *sacuers*;

Altering a table

- The format of an existing database table can be changed with an Alter Table query.
- **Add a complete new column**
    - Alter table *table-name*
    - Add column name *data-type optional-modifiers*;
- **Add a primary key**
    - Alter table *table-name*
    - Add primary key (*column-name*);
- **Change the name of an existing column**
    - Alter table *table-name*
    - Change *old-column-name new-column-name*
    - Date-type *optional-modifiers*;
- **Permanently delete an entire column**
    - Alter table *table-name*
    - Drop column *column-name;*
- **Example:**
    - Use *my_database*;
    - Create table if not exists *dishes*
        - (

        - Id int not null,
        - Pattern varchar(25) not null,
        - Price decimal (6,2)
        -);
    - Explain *dishes*;
    - Alter table *dishes*
        - Add primary key (id)
        - Add column code int unique not null,
        - Change *pattern* *dish_pattern* varchar(25) not null,
        - Drop column *price*;
    - Explain *dishes*;
    - Drop table *dishes*;

Inserting complete rows

- Data can be inserted into an existing database table with an Insert Into query
    - Insert into table-name values (value, value);
    - Insert just one row into the database table. A data value must be specified for each column in the corresponding order.
    - All data contained in a table can be displayed with a Select query using the * wildcard and From keyword:
        - Select * from *table-name*;
- **Example:**
    - Use *my_database*;
    - Create table *prints*
        - (
        - Id int not null,
        - Code varchar(8) not null,
        - Name varchar(20) not null,
        - Primary key(id)
        -);
    - Insert into *prints* values
        - (
        - 1, “624/1636”, “Lower Manhattan”
        -);
    - Insert into *prints* values
        - (
        - 2, “624/1904”, “Hill Town”
        -);

    - Insert into *prints* values
        - (
        - 3, “624/1681”, “Roscoff Trawlers”
        -);
    - Select * From *prints*;
    - Drop table *prints*;

Including a columns list

- The Insert Into can be improved by adding a “columns list” to explicitly specify the table column
    - Insert into *table-name* (column, column)
    - Values (value, value);
- **Example:**
    - Use *my_database*;
    - Create table *towels*
        - (
        - Code varchar(8) not null primary key,
        - Name varchar(20) not null,
        - Color varchar(20) default “White”
        -);
    - Insert 3 records into the towels table
        - Insert into *towels* (code, name, color)
            - Values (“821/7355”, “Dolphin”, “Blue”);
        - Insert into towels (color, code, name)
            - Values (“Lilac”, “830/1921”, “Daisy”);
        - Insert into towels (code, name)
            - Values (“830/2078”, “Starburst”);
    - Select * From *towels*;

Inserting selected data

- Data can be inserted into a table from another table with an Insert Select query
    - Insert into *destination-table* (column, column)
    - Select * From *source-table*;
- **Example:**
    - Use *my_database*;
    - Create table *bath_towels*
        - (
            - Code varchar(8) not null primary key,

                - Name varchar (20)   not null,
                - Color   varchar (20)   default “white”
                -);
        - Insert into *bath_towels* (code, name, color)
            - Values (“821/9735”, “Harvest”, “Beige”);
        - Insert into *bath_towels* (code, name, color)
            - Values (“821/9628”, “Wine”, “Maroon”);
        - Show *tables*;
        - Select * from *bath_towels*;
        - Select * from *towels*;
        - Insert into *bath_towels* (code, name, color)
        - Select * from *towels*;
        - Select * from *bath_towels;*

Updating data

- All the data contained within a column of a database table can be changed with an update query
    - Update *table-name* set *column-name* = value;
- Query will update every field in the specified column with the single specified value
- **Example:**
    - Use *my_databse*;
    - Select * from *bath_towels*;
    - Update *bath_towels* set *color* = “White”;
    - Select * From *bath_towels*;

Changing specific data

- The row can be identified by adding the Where keyword to the Update query to match a value in a specified column
    - Update table-name set column-name = value
    - Where column-name = value;
- **Example:**
    - Use *my_database*;
    - Select * From *bath_towels;*
    - Update specific field in the “color” column
        - Update *bath_towels*
            - Set *color* = “Beige” where *name* = “Harvest”;
        - Update *bath_towels*
            - Set *color* = “Blue” where *name* = “Dolphin”;

        - Update *bath_towels*
            - Set *color* = “Lilac” where *name* = “Daisy”;
        - Update *bath_towels*
            - Set *name* = “Tempest”, *color* = “Maroon”
            - Where *code* = “821/9628”;
    - Select * from *bath_towels*;
- The final update query more correctly identifies the row by its primary key value in the “code” column

Deleting Data

- Rows can be removed from a database table with a Delete query
    - Delete from *table-name;*
- Specific row can be removed from a table by adding Where
    - Delete from *table-name* where *column* = value;
- **Example:**
    - Use *my_database*;
    - Select * From *bath_towels;*
    - Delete two specific rows
        - Delete from *bath_towels* where *code* = “821/9735”;
        - Delete from *bath_towels* where *code* = “821/7355”;
    - Select * From *bath_towels*
    - Delete all remaining rows
        - Delete from *bath_towels*;
    - Select * From *bath_towels*;
    - Drop table *towels*;
    - Drop table *bath_towels;*

Retrieving data from tables

- Retrieve only data from the particular column
    - Select * From *table-name*;
    - Select *column-name* From *table-name;*
- **Example:**
    - Use *my_database*;
    - Create table if not exists *microwaves*
        - (
        - Id int auto_increment primary key,
        - Maker varchar (20) not null,
        - Model varchar (20) not null,

        - Power Int not null
        -);
    - Insert data into the “microwaves” table
        - Insert into *microwaves (maker, model, power)*
            - Values (“Sharp”, “R254SL”, 800);
    - Show all data
        - Select * From *microwaves*;
        - Select maker from *microwaves*;
        - Select model from *microwaves*;
- A select query can return the data from multiple columns by including the required column names as a comma-delimited list
    - Select *column*, *column, column* From *table-name*;
- **Example:**
    - Show all data
        - Select * From *microwaves*;
    - Show all data in “ID” and “maker” columns
        - Select id, *maker* From *microwaves*;
- A Select query can retrieve a specific row from a database table if it includes the Where keyword
    - Select * From *table-name* Where *column* = value;
- **Example:**
    - Show all data in row 2
        - Select * From *microwaves* Where id=2;
- Database tables can be created dynamically by combining a Create Table query with a Select query
    - Create a table called “800w_microwaves” and copy all 800w microwave data from “microwaves”
        - Create table if not exists *800w_microwaves*
        - Select * From *microwaves* Where *power* = 800;
    - Show tables;
    - Select * From *800w_microwaves*;
    - Select * From *microwaves*;
    - Drop table if exists *800w_microwaves*;
- Specific fields of a table can be copied into specific fields of another table using an Insert Into query
    - Insert Into *table-name* (column, column)
    - Select *column*, *column* Where *column* = value;
- **Example:**

- Create table *sharp_ovens*
  - (
  - Id int auto_increment primary key,
  - Model varchar (20) not null,
  - Power int not null,
  - Grill varchar(3) default "No"
  -);
- Insert data into the "*sharp_ovens*" table
  - Insert into *sharp_ovens* (model, power, grill)
    - Values ("R654",800,"Yes")
- Copy specific fields from "*microwaves*" to "*Sharp_ovens*"
  - Insert into *sharp_ovens* (model, power)
  - Select *model*, *power* From microwaves
  - Where *maker* = "Sharp";
- Show all data
  - Select * From *sharp_ovens*;
- Delete tables
  - Drop table *microwaves*;
  - Drop table *sharp_ovens*;

Sorting retrieved data

- The data returned by a Select query may not always appear in the same order as the rows of the table – especially following updates that that table. To explicitly sort into a specified order using Order By:
  - Use *my_database*;
  - Create a table called "critters"
    - Create table *critters*
      - (
      - Id int primary key,
      - Name varchar (20) not null
      -);
  - Insert records in the table
    - Insert into *critters* (id, name), values (3, "Beaver");
    - Insert into *critters* (id, name), values (1, "Duck");
  - Show all data in numerically ordered format
    - Select * From *critters* Order By *id*;
  - Show the name column in critters alphabetically
    - Select *name* From *critters* Order By *name*;

- Order By clause can sort retrieved data by multiple columns
    - Select *id*, *name* from *Critters*
    - Order By *name*, *critters*;
- Order By clause can optionally refer to a column of retrieved data by its position
    - Select *id*, *name* From *Critters* Order By *2*
- By default Select queries are automatically sorted in ascending or descending order
    - Select * From *Critters* order By *Name* ASC;
    - Select * From *Critters* order By *Id* DESC;

Simple data filtering

- Most commonly recognized comparison operators:

Operator:	Description:
=	Equality
!= or <>	Inequality
<	Less than
<=	Less than or equal
>	More than
>=	More than or equal
Between min And max	Within the range min to max
IS NULL	Is a NULL value
IS NOT NULL	Is not a NULL value

- Comparing a single value
- **Example:**
    - Use *my_database*;
    - Create a table called "*clock radios*"
        - (
        - Code char(8) primary key,
        - Make varchar (25) not null,
        - Model varchar (25) not null,
        - Price decimal (4,2) not null
        -);
    - Insert records into the table
        - Insert into *clock_radios* (code, make, model, price)
            - Values ("512", "Alba", "C2108", 6.75);
            - Values ("280", "Hitachi", "KC30", 8.99);
            - Values ("350", "Sony", "C253", 19.99);
            - Values ("350", "Sony", "C250", 14.99);
    - Show records if price is below 7.99
        - Select * From *clock_radios* Where *price* < 7.99

    - Show all records where the price is between 6 and 10
        - Select * From *clock_radios*
        - Where *price* between 6 And 10;
    - Show records where make is between Alba and Sony
        - Select * From *clock_radios*
        - Where *make* between “A” and “S”;
    - Show all records where make is not “Sony”
        - Select * From *clock_radios* where *make* != “Sony”
    - Show all records where there is no make
        - Select * from *clock_radios* where make is null
- A where clause in a select query can make multiple comparisons using the And logical operator
    - Select data from table-name
    - Where *column* = *value* And *column* = *value*;
    - Show all records where the make is “Sony” and the price is above 15
        - Select * From *clock_radios*
        - Where *make* = “Sony” AND *price* > 15;
- A where clause in a select query can make multiple comparisons using OR logical operator
    - Show all records where the make is not “Sony” or the price is below 15
        - Select * from *clock_radios*
        - Where *make* != “Sony” or *price* < 15
- A where clause in a select query can specify a list of alternative value for comparison with column data using the IN keyword
    - Show records where make is “Sony” or “Hitachi” and the model is not “KC30” or “C2108”
        - Select * From *clock_radios*
        - Where *make* in (“Sony”, “Hitachi”)
        - And *model* not in (“KC30”, “C2108”);
- A where clause in a select query can contain any number of comparison tests using the AND and OR keywords
    - Show records where make is “Sony” or “Hitachi” and price is 14.99
        - Select * From *clock_radios*
        - Where (*Make* = “Sony” or *make* = “Hitachi”)
        - And *price* = 14.99;
- The Like keyword offers to make comparisons against text strings without requiring an exact complete match
    - Select data from table-name

- Where column Like search-pattern;
- Show records where the model is “C250” series
  - Select * From *clock_radios* where *model* like “%C25%”;
    - % wildcard represents zero, one or more characters in a pattern
    - “_” is used for just a single character in a search pattern (i.e., So_y or S__y

- REGEXP = regular expression used in search pattern to denote their significance

Expression	Matches
“A”	Containing a single character
“[ABC]”	Containing one of characters
“[A-K]”	Containing characters from A-K
“[0-5]”	Range of digits like above
“^M”	Any string starting with letter
“H$”	Any string ending with letter

  - Records where the name contains W
    - Where *name* regexp “W”;
  - Records where the name contains W or N
    - Where *name* regexp “[WN]”;
  - Records where the name beings with B
    - Where *name* regexp “^B”;
  - Records where the name ends with H
    - Where *name* regexp “H$”
  - Records where the name begins with B or C
    - Where *name* regexp “^[BC]”;
- Put not keyword before the Regexp to return all strings that don’t match the expression

Generating calculated fields

Concatenating fields

- A calculated field is created using data stored within several columns of a database. It’s useful to present a range of data in a formatted manner
- DBMS favor concatenation using the “+” operator:
  - Calculated field = column 1+ separator + column 2
  - Concat_WS(Separator, column1, column2)
- **Example:**
  - Use *my_database;*
  - Create table if not exist *hotels*

        - (
        - Name varchar(25) primary key,
        - Street varchar(25),
        - City varchar(25),
        - State varchar(25),
        - Zip Int
        -);
    - Insert a record into the “hotels” table
        - Insert into *hotels* (*name, street, city, state, zip*)
            - Values (“Las Vegas Hilton”, 3000, “Las Vegas”, “Nevada”, 89109);
        - Retrieve 2 concatenated calculated fields
            - Select concat (*name, “,”,state*) from *hotels*;
            - Select Concat_WS*(“,\n”, name, street, city, state, zip*)
            - From *hotels*;
                - \n adds a comma and a newline separator between each item of data

Trimming padded spaces

- All leading and trailing spaces can be removed from a string with the Trim keyword
    - Trim (column)
- Most DBMS also support LTRIM and RTIM keywords which trim spaces from the left and right of the string.
    - Select concat (trim(*name*), “,”,rtrim(*state*)) from *hotels*;
    - From *hotels* where *state* = “Nevada”

Adopting aliases

- Assign a meaningful heading to label the calculated field instead
    - Select Concat_WS*(“,”, name, street, city, state, zip*)
    - As *Crazy_Party*
    - From *hotels*;

Doing arithmetic

- Perhaps the most useful aspect of calculated field is to perform arithmetical operations on existing data and present the result

+	**Addition**
-	**Subtraction**
*****	**Multiplication**
/	**Division**

  - Generate calculated fields
    - Select *name*, *street* as jangso, *zip* as doshi,
    - *Jangso* + *doshi* as ultimate_place

Manipulating data

- SQL function is a keyword that performs a particular pre-ordained operation on a specified piece of data
- Common functions operations

Operation	SQL
Return part of a string	Substring()
Convert a data type	Convert()
Return a rounded-up number	Ceiling()
Return the current date	Curdate()

- **Example:**
  - Use my_database
  - Create a table called *party*
    - Create table if not exists *party*
      - (
      - Id int auto_increment primary key,
      - Dept char(10),
      - Name char(25)
      -);
  - Insert 3 records
    - Insert into *party* (dept, name)
      - Values (“accounts”, “Graham Miller”);
      - Values (“Sales”, “Gary Miller”);
      - Values (“production”, “Graham Wallace”);
  - Get 3 letter sub strings from the dept column
    - Select substring (dept, 1,3) from *party*;
  - Get name and length
    - Select upper (name), lower (name), length (name)

        - From party where *dept* = “Production”;
    - String data which sound similar can be returned using the soundex() function
        - Select soundex(name), name from party
        - Where soudex(name)-soundex*(“Gary Miller”*);

Numeric functions

- **Example:**
    - Get some square roots
        - Select sqrt (144), sqrt (125), round(sqrt(125));
    - Get Pi and round it up and down
        - Select PI(), Ceiling (PI()), Floor(PI());
    - Get some random numbers
        - Select rand(), rand()l
    - Get some random numbers in the range 1-1000
        - Select ceiling(rand()*100), ceiling (rand()*100);

Date and time functions

- Date, time or datetime data types.
    - Get the current full date object, current date & time
        - Select now(), curdate(), curtime();
    - Use *my_database;*
    - Create table if not exists *labor_day*
        - (
        - Id int auto_increment primary key,
        - Date datetime not null
        -);
    - Insert 1 record into the labor_day table
        - Insert into *labor_day* (date)
        - Values (2014-09-05 12:45:30”);
    - Get the name of the day
        - Select dayname (date) from *labor_day*;
    - Get the day, month and year date components
        - Select dayofmonth(date), monthname(date), year(date) from *labor_day;*
    - Get the hour, minute and second time components
        - Select hour(date), minute (date), second(date)
        - From *labor_day;*
        - Use the Between and And keywords to match a range of dates

- Get the version number and current user
  - Select version (), user();
- Get the thread identity
  - Show processlist;
- Create a new user with full privileges
  - Grant all privileges on *.* to *monty@localhost*
  - Identified by "*monty-pwd*" with grant option;
- Confirm privileges for the new user
  - Show grants for *monty@localhost;*
- Drop table if exists *labor_day*

Finding summary values

- The avg() aggregate function returns a summarized average value of all the values within the column specified as its argument
- The max() aggregate function returns the single highest value
- The min() aggregate function returns the single lowest value
  - Use *my_database*;
  - Create table if not exists *multimeters*
    - (
    - Id int auto_increment primary key,
    - Model char(10 not null,
    - Price decimal(4,2) not null
    -);
  - Insert records
    - Insert into *multimeters* (model, price)
      - Values ("Standard", 11.75);
      - Values ("Super", 19.75);
      - Values ("Super", 23.00);
      - Values ("Deluxe", 24.99);
  - Get the average price
    - Select Max(*price*) As Max_price, Min(*Price*) As min_price
    - From *multimeters*;
- To return the total number of rows
  - Count(*)
- To return the number of non-empty rows in a particular column the column's name must be specified
  - Count(column-name)
  - Count the total number of rows

        - Select count(*) as total_number_or_rows
        - From *multimeters*;
    - Count the total number of rows with model#s
        - Select count(*model*)
        - From *multimeters*;
    - Get total cost
        - Select sum (*price*) as total_price
        - From *multimeters* where *model* = “Super”;
- Using the Distinct keyword ensured the count() function will only count rows with unique values in the specified column
    - Get the number of unique rows
        - Select count(distinct *model*) As Count
        - From *multimeters*;
    - Get all the unique values
        - Select distinct model from *multimeters*;
- Data can be grouped by adding a Group By clause to the end of a Select statement
    - Get the number of items for each type
        - Select *model*, count(*) as num_items
        - From *multimeters* group by *model*;
    - Get the number of items for each type
        - Select *price*, count (*) as prices
        - From *multimeters* group by *price*;
- A Having clause is very similar to a Where clause – except that Where filters rows and Having filters groups.
    - Get the number of items for each model where the price exceeds 15 and when there is more than 1 item for that model
        - Select *model*, count(*) As num_items_model
        - From *multimeters*
        - Where *price* >= 15
        - Group by *models*
        - Having count (*) >1;

- It is important that multiple clauses in a Select statement appear in the correct order:

Clause	Specifies
Select	Columns or Expressions to return
From	Table to retrieve data from
Where	Row-level filter
Group By	Column to group around
Having	Group-level filter
Order By	Return sort order

- Get the ID # and number of number of items ordered where the model is not Super and the number of items ordered is fewer than 3 sorted by ID#
  - Select *ID*, Count(*) As Num_Items
  - From *multimeters* where *model* != “Super”
  - Group by *ID* having count (*)<3 order by *ID*;
  - Drop table if exists *multimeters*;

Making complex queries

- Sub-queries are useful to retrieve data from a table specifying what an outer Select statement should return from another table.
- **Example:**
  - Create table if not exists customers
    - (
    - Acc_num int primary key,
    - Name char(20) not null
    -);
  - Insert 2 records
    - Insert into customers (acc_num, name)
    - Values (123, “T.Smith”);
    - Insert into customers (acc_num, name)
    - Values (124, “P.Jones”);
  - Create table if note exists orders
    - (
    - Ord_num int primary key ,
    - Acc_num int not null
    -);
  - Insert 2 records
    - Insert into orders (ord_num, acc_num) values (3, 123);
    - Insert into orders (ord_num, acc_num) values (4, 124);
  - Retrieve the name of the customer placing order 4

        - Select ord_num, customers.acc_num , name
        - From customers, orders
        - Where customer.acc_num = orders.acc_num
        - And orders.ord_num = 4;
            - Use the dot syntax to explicitly identify table columns wherever there is possible ambiguity

Sub-query calculated fields

- Can be used to generate a calculated field that returns values from a table to an outer Select statement
    - Get the number of orders per customer
        - Select *name*, *customers.acc_num*,
        - Count(*) As number_of_orders
        - From *customers*, *orders*
        - Where *customer.acc_num* = *orders.acc_num*
        - Group by name order by *customers.acc_num*;

Combining queries

- Multiple Select queries can be made to combine their returns into a single result set using the Union keyword
    - Display all data as a single result
        - Select * from *customers*
        - Union
        - Select * from *orders*
- By default Union automatically excludes any duplicate rows from the data returned by the Select statement.
    - To display all data (including duplicates)
        - Select * from *customers*
        - Union All
        - Select * from *orders*;

Sorting combined results

- The data returned from multiple tables using the Union keyword can be sorted into a specified order by adding a single Order By clause.
        - Select * from *customers*
        - Union All
        - Select * from *orders*
        - Order By *orders.acc_num;*

Joining database tables

- The ability to dynamically join together multiple tables with a single Select query is one of SQL's most powerful features.
- The data can be returned from multiple joined tables by stating the table names as a comma-separated list after the From keyword in a Select statement
    - Create table if not exists *games*
        - (
        - Id varchar(10) primary key,
        - Vendor int not null,
        - Name char(20) not null,
        - Price decimal(6,2) not null
        -);
    - Insert records
        - Insert into *games* (id, vendor, name, price)
        - Values (371, 1, "Scrabble", 14.50)
        - Insert into *games* (id, vendor, name, price)
        - Values (373, 2, "Jenga", 6.99)
    - Create table if not exists vendors
        - (
        - Id int primary key,
        - Name char(20) not null,
        - Location char(20) not null
        -);
    - Insert records
        - Insert into vendors (id, name, location)
        - Values (1, "Matte", "El Segundo");
        - Insert into vendors (id, name, location)
        - Values (2, "Hasbro", "El Segundo");
    - Display game code, name, price and vendor name for each game in the two joined tables
        - Select
            - *Games.id* as product_code,
            - *Games.name* as game,
            - *Vendors.name* as vendor,
            - *Games.price* as price
        - From
            - *Games*, *vendors*

        - Where
            - *Vendors.id* = games.vendor;

Joining multiples tables

- There is theoretically no limit to the number of tables that can be joined by a single Select statement
    - Create table if not exists *items*
        - (
        - Id int primary key,
        - Vendor int not null,
        - Name char (20) not null,
        - Price decimal(6,2) not null
        -);
    - Insert records
        - Insert into *items* (id, vendor, name, price)
        - Values (1, 2, “Elephants”, 149.99);
        - Insert into *items* (id, vendor, name, price)
        - Values (2, 2, “Reindeer”, 123.99);

    - Display Id, price, name, vendor and location of id# 2
    - Select
        - Games.id
        - Games. Price
        - Items.name
        - Items.price
        - Vendors.location
    - From
        - *Items*, *games*, *vendors*
    - Where
        - *Items.id* = games.id
    - And
        - *Games.id* = vendors.id
    - And
        - *Vendors.id* = 2;

Creating self joins

- A self join enables a Select query to make more than one reference to the same table. In the Select statement this table is given two alias names of “S1” and “S2”.

- Consider these to be two virtual tables containing identical data in identical columns.
    - Display the all vendor names in the same location as “El Segundo”
        - Select
            - *V1*.name as Vendor_Name, *V2*.name as Duplicate_Name
        - From
            - *Vendors* as V1, *Vendors* as V2
        - Where
            - V1.*location* = V2.*location*
        - And
            - V2.*location* = “*El Segundo*”

Creating natural joins

- A natural join is simply a technique to eliminate a column that contains duplicate data
    - Display each vendor number, name and price
        - Select
            - *V.*; R.Price*
        - From
            - *Vendors* as V, *Records* as R
        - Where
            - V.id = r.id;

Creating outer joins

- Return data from rows that have no matched relationship. For instance, to include products with zero orders in a list of products and ordered quantities. This can be achieved by adding the Outer Join keywords to a From clause, preceded by the Left or Right keyword.
- When Left Outer Join is specified, all the rows in the table specified to the left of this statement are joined to the table on its right. Vice versa for right.
    - Select
        - V.name
        - R.id
        - R.price
    - From
        - *Vendors* as V left outer join *Records* as R
    - On
        - V.id = R.id;