What is SQL?

SQL (pronounced "ess-que-el") stands for Structured Query Language. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. Although most database systems use SQL, most of them also have their own additional proprietary extensions that are usually only used on their system. However, the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database. This tutorial will provide you with the instruction on the basics of each of these commands as well as allow you to put them to practice using the SQL Interpreter.

Table Basics

A relational database system contains one or more objects called tables. The data or information for the database are stored in these tables. Tables are uniquely identified by their names and are comprised of columns and rows. Columns contain the column name, data type, and any other attributes for the column. Rows contain the records or data for the columns. Here is a sample table called "weather".

city, state, high, and low are the columns. The rows contain the data for this table:

Weather			
city	**state**	**high**	**low**
Phoenix	Arizona	105	90
Tucson	Arizona	101	92
Flagstaff	Arizona	88	69
San Diego	California	77	60
Albuquerque	New Mexico	80	72

Selecting Data

The **select** statement is used to query the database and retrieve selected data that match the criteria that you specify. Here is the format of a simple select statement:

```
select "column1"
  [,"column2",etc]
  from "tablename"
  [where "condition"];
  [] = optional
```

The column names that follow the select keyword determine which columns will be returned in the results. You can select as many column names that you'd like, or you can use a "*" to select all columns.

The table name that follows the keyword **from** specifies the table that will be queried to retrieve the desired results.

The **where** clause (optional) specifies which data values or rows will be returned or displayed, based on the criteria described after the keyword **where**.

Conditional selections used in the **where** clause:

= Equal
> Greater than
< Less than
>= Greater than or equal
<= Less than or equal
<> Not equal to
LIKE *See note below

The **LIKE** pattern matching operator can also be used in the conditional selection of the where clause. Like is a very powerful operator that allows you to select only rows that are "like" what you specify. The percent sign "%" can be used as a wild card to match any possible character that might appear before or after the characters specified. For example:

```
select first, last, city
   from empinfo
   where first LIKE 'Er%';
```

This SQL statement will match any first names that start with 'Er'. **Strings must be in single quotes.**

Or you can specify,

```
select first, last
   from empinfo
   where last LIKE '%s';
```

This statement will match any last names that end in a 's'.

```
select * from empinfo
   where first = 'Eric';
```

This will only select rows where the first name equals 'Eric' exactly.

Sample Table: empinfo					
first	**last**	**id**	**age**	**city**	**state**
John	Jones	99980	45	Payson	Arizona
Mary	Jones	99982	25	Payson	Arizona
Eric	Edwards	88232	32	San Diego	California
Mary Ann	Edwards	88233	32	Phoenix	Arizona
Ginger	Howell	98002	42	Cottonwood	Arizona
Sebastian	Smith	92001	23	Gila Bend	Arizona
Gus	Gray	22322	35	Bagdad	Arizona
Mary Ann	May	32326	52	Tucson	Arizona

Erica	Williams	32327	60	Show Low	Arizona
Leroy	Brown	32380	22	Pinetop	Arizona
Elroy	Cleaver	32382	22	Globe	Arizona

Enter the following sample select statements in the SQL Interpreter Form at the bottom of this page. Before you press "submit", write down your expected results. Press "submit", and compare the results.

```
select first, last, city from empinfo;

select last, city, age from empinfo
       where age > 30;

select first, last, city, state from empinfo
       where first LIKE 'J%';

select * from empinfo;

select first, last, from empinfo
       where last LIKE '%s';

select first, last, age from empinfo
       where last LIKE '%illia%';

select * from empinfo where first = 'Eric';
```

Select statement exercises

Enter select statements to:

1. Display the first name and age for everyone that's in the table.
2. Display the first name, last name, and city for everyone that's not from Payson.
3. Display all columns for everyone that is over 40 years old.
4. Display the first and last names for everyone whose last name ends in an "ay".
5. Display all columns for everyone whose first name equals "Mary".
6. Display all columns for everyone whose first name contains "Mary".

Selecting Data Answers:

1. Display everyone's first name and their age for everyone that's in table.

```
2. select first,
3.        age
     from empinfo;
```

4. Display the first name, last name, and city for everyone that's not from Payson.

```
5. select first,
6.        last,
7.        city
8.   from empinfo
9. where city <>
     'Payson';
```

10. Display all columns for everyone that is over 40 years old.

```
11.select * from empinfo
          where age > 40;
```

12. Display the first and last names for everyone whose last name ends in an "ay".

```
13.select first, last from empinfo
          where last LIKE '%ay';
```

14. Display all columns for everyone whose first name equals "Mary".

```
15.select * from empinfo
          where first = 'Mary';
```

16. Display all columns for everyone whose first name contains "Mary".

```
17.select * from empinfo
          where first LIKE '%Mary%';
```

Creating Tables

The **create table** statement is used to create a new table. Here is the format of a simple **create table** statement:

```
create table "tablename"
("column1" "data type",
 "column2" "data type",
 "column3" "data type");
```

Format of create table if you were to use optional constraints:

```
create table "tablename"
("column1" "data type"
         [constraint],
 "column2" "data type"
         [constraint],
 "column3" "data type"
        [constraint]);
 [ ] = optional
```

Note: You may have as many columns as you'd like, and the constraints are optional.

Example:

```
create table employee
(first varchar(15),
 last varchar(20),
 age number(3),
 address varchar(30),
 city varchar(20),
 state varchar(20));
```

To create a new table, enter the keywords **create table** followed by the table name, followed by an open parenthesis, followed by the first column name, followed by the data type for that column, followed by any optional constraints, and followed by a closing parenthesis. It is important to make sure you use an open parenthesis before the beginning table, and a closing parenthesis after the end of the last column definition. Make sure you seperate each column definition with a comma. All SQL statements should end with a ";".

The table and column names must start with a letter and can be followed by letters, numbers, or underscores - not to exceed a total of 30 characters in length. Do not use any SQL reserved keywords as names for tables or column names (such as "select", "create", "insert", etc).

Data types specify what the type of data can be for that particular column. If a column called "Last_Name", is to be used to hold names, then that particular column should have a "varchar" (variable-length character) data type.

Here are the most common Data types:

`char(size)`	Fixed-length character string. Size is specified in parenthesis. Max 255 bytes.
`varchar(size)`	Variable-length character string. Max size is specified in parenthesis.
`number(size)`	Number value with a max number of column digits specified in parenthesis.
`date`	Date value
`number(size,d)`	Number value with a maximum number of digits of "size" total, with a maximum number of "d" digits to the right of the decimal.

What are constraints? When tables are created, it is common for one or more columns to have **constraints** associated with them. A constraint is basically a rule associated with a column that the data entered into that column must follow. For example, a "unique" constraint specifies that no two records can have the same value in a particular column. They must all be unique. The other two most popular constraints are "not null" which specifies that a column can't be left blank, and "primary key". A "primary key" constraint defines a unique identification of each record (or row) in a table. All of these and more will be covered in the future Advanced release of this Tutorial. Constraints can be entered in this SQL interpreter, however, they are not supported in this Intro to SQL tutorial & interpreter. They will be covered and supported in the future release of the Advanced SQL tutorial - that is, if "response" is good.

It's now time for you to design and create your own table. You will use this table throughout the rest of the tutorial. If you decide to change or redesign the table, you can either **drop** it and recreate it or you can create a completely different one. The SQL statement **drop** will be covered later.

Create Table Exercise

You have just started a new company. It is time to hire some employees. You will need to create a table that will contain the following information about your new employees: firstname, lastname, title, age, and salary. After you create the table, you should receive a small form on the screen with the appropriate column names. If you are missing any columns, you need to double check your SQL statement and recreate the table. Once it's created successfully, go to the "Insert" lesson.

IMPORTANT: When selecting a table name, it is important to select a unique name that no one else will use or guess. Your table names should have an underscore followed by your initials and the digits of your birth day and month. For example, Tom Smith, who was born on November 2nd, would name his table myemployees_ts0211 Use this convention for all of the tables you create. Your tables will remain on a shared database until you drop them, or they will be cleaned up if they aren't accessed in 4-5 days. If "support" is good, I hope to eventually extend this to at least one week. When you are finished with your table, it is important to drop your table (covered in last lesson).

Your create statement should resemble:

```
create table
  myemployees_ts0211
(firstname varchar(30),
 lastname varchar(30),
 title varchar(30),
 age number(2),
 salary number(8,2));
```

Inserting into a Table

The **insert** statement is used to insert or add a row of data into the table.

To insert records into a table, enter the key words **insert into** followed by the table name, followed by an open parenthesis, followed by a list of column names separated by commas, followed by a closing parenthesis, followed by the keyword **values**, followed by the list of values enclosed in parenthesis. The values that you enter will be held in the rows and they will match up with the column names that you specify. Strings should be enclosed in single quotes, and numbers should not.

```
insert into "tablename"
 (first_column,...last_column)
  values (first_value,...last_value);
```

In the example below, the column name `first` will match up with the value `'Luke'`, and the column name `state` will match up with the value `'Georgia'`.

Example:

```
insert into employee
  (first, last, age, address, city, state)
  values ('Luke', 'Duke', 45, '2130 Boars Nest',
          'Hazard Co', 'Georgia');
```

Note: All strings should be enclosed between **single** quotes: `'string'`

Insert statement exercises

It is time to insert data into your new employee table.

Your first three employees are the following:

Jonie Weber, Secretary, 28, 19500.00
Potsy Weber, Programmer, 32, 45300.00
Dirk Smith, Programmer II, 45, 75020.00

Enter these employees into your table first, and then insert at least 5 more of your own list of employees in the table.

After they're inserted into the table, enter select statements to:

1. Select all columns for everyone in your employee table.
2. Select all columns for everyone with a salary over 30000.
3. Select first and last names for everyone that's under 30 years old.
4. Select first name, last name, and salary for anyone with "Programmer" in their title.
5. Select all columns for everyone whose last name contains "ebe".
6. Select the first name for everyone whose first name equals "Potsy".
7. Select all columns for everyone over 80 years old.
8. Select all columns for everyone whose last name ends in "ith".

Create at least 5 of your own select statements based on specific information that you'd like to retrieve.

Your Insert statements should be similar to: (note: use your own table name that you created)

```
insert into
```

```
  myemployees_ts0211
(firstname, lastname,
 title, age, salary)
values ('Jonie', 'Weber',
        'Secretary', 28,
        19500.00);
```

1. Select all columns for everyone in your employee table.

```
2. select * from
   myemployees_ts0211
```

3. Select all columns for everyone with a salary over 30000.

```
4. select * from
5. myemployees_ts0211
   where salary > 30000
```

6. Select first and last names for everyone that's under 30 years old.

```
7. select firstname, lastname
8. from myemployees_ts0211
   where age < 30
```

9. Select first name, last name, and salary for anyone with "Programmer" in their title.

```
10.select firstname, lastname, salary
11.from myemployees_ts0211
   where title LIKE '%Programmer%'
```

12. Select all columns for everyone whose last name contains "ebe".

```
13.select * from
14.myemployees_ts0211
   where lastname LIKE '%ebe%'
```

15. Select the first name for everyone whose first name equals "Potsy".

```
16.select firstname from
17.myemployees_ts0211
   where firstname = 'Potsy'
```

18. Select all columns for everyone over 80 years old.

```
19.select * from

20.myemployees_ts0211

   where age > 80
```

21. Select all columns for everyone whose last name ends in "ith".

```
22.select * from
23.myemployees_ts0211
   where lastname LIKE '%ith'
```

Updating Records

The **update** statement is used to update or change records that match a specified criteria. This is accomplished by carefully constructing a where clause.

```
update "tablename"
set "columnname" =
    "newvalue"
 [,"nextcolumn" =
   "newvalue2"...]
where "columnname"
  OPERATOR "value"
```

```
[and|or "column"
 OPERATOR "value"];

[] = optional
```

[The above example was line wrapped for better viewing on this Web page.]

Examples:

```
update phone_book
  set area_code = 623
  where prefix = 979;

update phone_book
  set last_name = 'Smith', prefix=555, suffix=9292
  where last_name = 'Jones';

update employee
  set age = age+1
  where first_name='Mary' and last_name='Williams';
```

Update statement exercises

After each update, issue a select statement to verify your changes.

1. Jonie Weber just got married to Bob Williams. She has requested that her last name be updated to Weber-Williams.
2. Dirk Smith's birthday is today, add 1 to his age.
3. All secretaries are now called "Administrative Assistant". Update all titles accordingly.
4. Everyone that's making under 30000 are to receive a 3500 a year raise.
5. Everyone that's making over 33500 are to receive a 4500 a year raise.
6. All "Programmer II" titles are now promoted to "Programmer III".
7. All "Programmer" titles are now promoted to "Programmer II".

Create at least 5 of your own update statements and submit them

Updating Records Answers:

1. Jonie Weber just got married to Bob Williams. She has requested that her last name be updated to Weber-Williams.

```
update
 myemployees_ts0211
set lastname=
  'Weber-Williams'
where firstname=
      'Jonie'
 and lastname=
       'Weber';
```

9. Dirk Smith's birthday is today, add 1 to his age.

```
update myemployees_ts0211
  set age=age+1
  where firstname='Dirk' and lastname='Smith';
```

12. All secretaries are now called "Administrative Assistant". Update all titles accordingly.

```
update myemployees_ts0211
  set title = 'Administrative Assistant'
  where title = 'Secretary';
```

15. Everyone that's making under 30000 are to receive a 3500 a year raise.

```
update myemployees_ts0211
  set salary = salary + 3500
  where salary < 30000;
```

18. Everyone that's making over 33500 are to receive a 4500 a year raise.

```
update myemployees_ts0211
  set salary = salary + 4500
  where salary > 33500;
```

21. All "Programmer II" titles are now promoted to "Programmer III".

```
update myemployees_ts0211
  set title = 'Programmer III'
  where title = 'Programmer II'
```

24. All "Programmer" titles are now promoted to "Programmer II".

```
update myemployees_ts0211
  set title = 'Programmer II'
  where title = 'Programmer'
```

Deleting Records

The **delete** statement is used to delete records or rows from the table.

```
delete from "tablename"

where "columnname"
  OPERATOR "value"
[and|or "column"
  OPERATOR "value"];

[ ] = optional
```

[The above example was line wrapped for better viewing on this Web page.]

Examples:

```
delete from employee;
```

Note: if you leave off the where clause, **all records will be deleted!**

```
delete from employee
  where lastname = 'May';

delete from employee
  where firstname = 'Mike' or firstname = 'Eric';
```

To delete an entire record/row from a table, enter "`delete from`" followed by the table name, followed by the where `clause` which contains the conditions to delete. If you leave off the `where` clause, all records will be deleted.

Delete statement exercises

(Use the select statement to verify your deletes):

1. Jonie Weber-Williams just quit, remove her record from the table.
2. It's time for budget cuts. Remove all employees who are making over 70000 dollars.

Create at least two of your own delete statements, and then issue a command to delete all records from the table

Deleting Records Answers:

1. Jonie Weber-Williams just quit, remove her record from the table:

```
delete
   from myemployees_ts0211
   where lastname =
      'Weber-Williams';
```

5. It's time for budget cuts. Remove all employees who are making over 70000 dollars.

```
delete
   from myemployees_ts0211
   where salary >
      70000;
```

Drop a Table

The **drop table** command is used to delete a table and all rows in the table.

To delete an entire table including all of its rows, issue the **drop table** command followed by the tablename. **drop table** is different from deleting all of the records in the table. Deleting all of the records in the table leaves the table including column and constraint information. Dropping the table removes the table definition as well as all of its rows.

```
drop table "tablename"
```

Example:

```
drop table myemployees_ts0211;
```

Drop Table exercises

1. Drop your employee table.

SELECT Statement

The SELECT statement is used to query the database and retrieve selected data that match the criteria that you specify.

The SELECT statement has five main clauses to choose from, although, FROM is the only required clause. Each of the clauses have a vast selection of options, parameters, etc. The clauses will be listed below, but each of them will be covered in more detail later in the tutorial.

Here is the format of the SELECT statement:

```
SELECT [ALL | DISTINCT] column1[,column2]

FROM table1[,table2]
```

```
[WHERE "conditions"]

[GROUP BY "column-list"]

[HAVING "conditions]

[ORDER BY "column-list" [ASC | DESC] ]
```

FROM & WHERE clause quick review

Example:

```
SELECT name, age, salary

FROM employee

WHERE age > 50;
```

The above statement will select all of the values in the name, age, and salary columns from the employee table whose age is greater than 50.

Note: Remember to put a semicolon at the end of your SQL statements. The **;** indicates that your SQL statment is complete and is ready to be interpreted.

Comparison Operators

=	Equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<> or !=	Not equal to
LIKE	String comparison test

*Note about LIKE

```
Example:

SELECT name, title, dept

FROM employee


WHERE title LIKE 'Pro%';
```

The above statement will select all of the rows/values in the name, title, and dept columns from the employee table whose title starts with 'Pro'. This may return job titles including Programmer or Pro-wrestler.

ALL and **DISTINCT** are keywords used to select either ALL (default) or the "distinct" or unique records in your query results. If you would like to retrieve just the unique records in

specified columns, you can use the "DISTINCT" keyword. DISTINCT will discard the duplicate records for the columns you specified after the "SELECT" statement: For example:

```
SELECT DISTINCT age

FROM employee_info;
```

This statement will return all of the unique ages in the employee_info table.

ALL will display "all" of the specified columns including all of the duplicates. The ALL keyword is the default if nothing is specified.

items_ordered

customerid	order_date	item	quantity	price
10330	30-Jun-1999	Pogo stick	1	28.00
10101	30-Jun-1999	Raft	1	58.00
10298	01-Jul-1999	Skateboard	1	33.00
10101	01-Jul-1999	Life Vest	4	125.00
10299	06-Jul-1999	Parachute	1	1250.00
10339	27-Jul-1999	Umbrella	1	4.50
10449	13-Aug-1999	Unicycle	1	180.79
10439	14-Aug-1999	Ski Poles	2	25.50
10101	18-Aug-1999	Rain Coat	1	18.30
10449	01-Sep-1999	Snow Shoes	1	45.00
10439	18-Sep-1999	Tent	1	88.00
10298	19-Sep-1999	Lantern	2	29.00
10410	28-Oct-1999	Sleeping Bag	1	89.22
10438	01-Nov-1999	Umbrella	1	6.75
10438	02-Nov-1999	Pillow	1	8.50
10298	01-Dec-1999	Helmet	1	22.00
10449	15-Dec-1999	Bicycle	1	380.50
10449	22-Dec-1999	Canoe	1	280.00
10101	30-Dec-1999	Hoola Hoop	3	14.75
10330	01-Jan-2000	Flashlight	4	28.00
10101	02-Jan-2000	Lantern	1	16.00
10299	18-Jan-2000	Inflatable Mattress	1	38.00
10438	18-Jan-2000	Tent	1	79.99
10413	19-Jan-2000	Lawnchair	4	32.00
10410	30-Jan-2000	Unicycle	1	192.50
10315	2-Feb-2000	Compass	1	8.00
10449	29-Feb-2000	Flashlight	1	4.50
10101	08-Mar-2000	Sleeping Bag	2	88.70
10298	18-Mar-2000	Pocket Knife	1	22.38
10449	19-Mar-2000	Canoe paddle	2	40.00
10298	01-Apr-2000	Ear Muffs	1	12.50

10330	19-Apr-2000	Shovel	1	16.75

customerid	firstname	lastname	city	state
10101	John	Gray	Lynden	Washington
10298	Leroy	Brown	Pinetop	Arizona
10299	Elroy	Keller	Snoqualmie	Washington
10315	Lisa	Jones	Oshkosh	Wisconsin
10325	Ginger	Schultz	Pocatello	Idaho
10329	Kelly	Mendoza	Kailua	Hawaii
10330	Shawn	Dalton	Cannon Beach	Oregon
10338	Michael	Howell	Tillamook	Oregon
10339	Anthony	Sanchez	Winslow	Arizona
10408	Elroy	Cleaver	Globe	Arizona
10410	Mary Ann	Howell	Charleston	South Carolina
10413	Donald	Davids	Gila Bend	Arizona
10419	Linda	Sakahara	Nogales	Arizona
10429	Sarah	Graham	Greensboro	North Carolina
10438	Kevin	Smith	Durango	Colorado
10439	Conrad	Giles	Telluride	Colorado
10449	Isabela	Moore	Yuma	Arizona

Review Exercises

1. From the *items_ordered* table, select a list of all items purchased for customerid 10449. Display the customerid, item, and price for this customer.
2. Select all columns from the *items_ordered* table for whoever purchased a **Tent**.
3. Select the customerid, order_date, and item values from the items_ordered table for any items in the item column that start with the letter "S".
4. Select the distinct items in the items_ordered table. In other words, display a listing of each of the unique items from the items_ordered table.
5. Make up your own select statements and submit them.

SELECT Exercise Answers

Exercise #1

```
SELECT customerid, item, price
FROM items_ordered
WHERE customerid=10449;
```

Exercise #2

```
SELECT * FROM items_ordered
WHERE item = 'Tent';
```

Exercise #3

```
SELECT customerid, order_date, item
FROM items_ordered
WHERE item LIKE 's%';
```

Exercise #4

```
SELECT DISTINCT item
FROM items_ordered;
```

Aggregate Functions

MIN	returns the smallest value in a given column
MAX	returns the largest value in a given column
SUM	returns the sum of the numeric values in a given column
AVG	returns the average value of a given column
COUNT	returns the total number of values in a given column
COUNT(*)	returns the number of rows in a table

Aggregate functions are used to compute against a "returned column of numeric data" from your SELECT statement. They basically summarize the results of a particular column of selected data. We are covering these here since they are required by the next topic, "GROUP BY". Although they are required for the "GROUP BY" clause, these functions can be used without the "GROUP BY" clause. For example:

```
SELECT AVG(salary)

FROM employee;
```

This statement will return a single result which contains the average value of everything returned in the salary column from the *employee* table.

Another example:

```
SELECT AVG(salary)

FROM employee;

WHERE title = 'Programmer';
```

This statement will return the average salary for all employees whose title is equal to 'Programmer'

Example:

```
SELECT Count(*)

FROM employees;
```

This particular statement is slightly different from the other aggregate functions since there isn't a column supplied to the count function. This statement will return the number of rows in the employees table.

Use these tables for the exercises
items_ordered
customers

Review Exercises

1. Select the maximum price of any item ordered in the items_ordered table. Hint: Select the maximum price only.>
2. Select the average price of all of the items ordered that were purchased in the month of Dec.
3. What are the total number of rows in the items_ordered table?
4. For all of the tents that were ordered in the items_ordered table, what is the price of the lowest tent? Hint: Your query should return the price only

Aggregate Function Exercise Answers

Exercise #1

```
SELECT max(price)
FROM items_ordered;
```

Exercise #2

```
SELECT avg(price)
FROM items_ordered
WHERE order_date LIKE '%Dec%';
```

Exercise #3

```
SELECT count(*)
FROM items_ordered;
```

Exercise #4

```
SELECT min(price) FROM items_ordered WHERE item = 'Tent';
```

GROUP BY clause

The GROUP BY clause will gather all of the rows together that contain data in the specified column(s) and will allow aggregate functions to be performed on the one or more columns. This can best be explained by an example:

GROUP BY clause syntax:

```
SELECT column1,
SUM(column2)

FROM "list-of-tables"

GROUP BY "column-list";
```

Let's say you would like to retrieve a list of the highest paid salaries in each dept:

```
SELECT max(salary), dept

FROM employee

GROUP BY dept;
```

This statement will select the maximum salary for the people in each unique department. Basically, the salary for the person who makes the most in each department will be displayed. Their, salary and their department will be returned.

Multiple Grouping Columns - What if I wanted to display their lastname too?

Use these tables for the exercises
items_ordered
customers

For example, take a look at the items_ordered table. Let's say you want to group everything of quantity 1 together, everything of quantity 2 together, everything of quantity 3 together, etc. If you would like to determine what the largest cost item is for each grouped quantity (all quantity 1's, all quantity 2's, all quantity 3's, etc.), you would enter:

```
SELECT quantity, max(price)

FROM items_ordered

GROUP BY quantity;
```

Enter the statement in above, and take a look at the results to see if it returned what you were expecting. Verify that the maximum price in each Quantity *Group* is really the maximum price.

Review Exercises

1. How many people are in each unique state in the customers table? Select the state and display the number of people in each. Hint: **count** is used to count rows in a column, **sum** works on numeric data only.
2. From the items_ordered table, select the item, maximum price, and minimum price for each specific item in the table. Hint: The items will need to be broken up into separate groups.
3. How many orders did each customer make? Use the items_ordered table. Select the customerid, number of orders they made, and the sum of their orders. Click the Group By answers link below if you have any problems.

GROUP BY Exercise Answers

Exercise #1

SELECT state, count(state)
FROM customers
GROUP BY state;

Exercise #2

SELECT item, max(price), min(price)
FROM items_ordered
GROUP BY item;

Exercise #3

SELECT customerid, count(customerid), sum(price)
FROM items_ordered
GROUP BY customerid;

HAVING clause

The HAVING clause allows you to specify conditions on the rows for each group - in other words, which rows should be selected will be based on the conditions you specify. The HAVING clause should follow the GROUP BY clause if you are going to use it.

HAVING clause syntax:

```
SELECT column1,
SUM(column2)

FROM "list-of-tables"

GROUP BY "column-list"

HAVING "condition";
```

HAVING can best be described by example. Let's say you have an employee table containing the employee's name, department, salary, and age. If you would like to select the average salary for each employee in each department, you could enter:

```
SELECT dept, avg(salary)

FROM employee

GROUP BY dept;
```

But, let's say that you want to ONLY calculate & display the average if their salary is over 20000:

```
SELECT dept, avg(salary)

FROM employee

GROUP BY dept

HAVING avg(salary) > 20000;
```

Use these tables for the exercises
items_ordered
customers

Review Exercises (note: yes, they are similar to the group by exercises, but these contain the HAVING clause requirements

1. How many people are in each unique state in the customers table that have more than one person in the state? Select the state and display the number of how many people are in each if it's greater than 1.
2. From the items_ordered table, select the item, maximum price, and minimum price for each specific item in the table. Only display the results if the maximum price for one of the items is greater than 190.00.
3. How many orders did each customer make? Use the items_ordered table. Select the customerid, number of orders they made, and the sum of their orders if they purchased more than 1 item.

HAVING Exercise Answers

Exercise #1

```
SELECT state, count(state)
FROM customers
GROUP BY state
HAVING count(state) > 1;
```

Exercise #2

```
SELECT item, max(price), min(price)
FROM items_ordered
GROUP BY item
HAVING max(price) > 190.00;
```

Exercise #3

```
SELECT customerid, count(customerid), sum(price)
FROM items_ordered
GROUP BY customerid
HAVING count(customerid) > 1;
```

ORDER BY clause

ORDER BY is an optional clause which will allow you to display the results of your query in a sorted order (either ascending order or descending order) based on the columns that you specify to order by.

ORDER BY clause syntax:

```
SELECT column1, SUM(column2)

FROM "list-of-tables"

ORDER BY

"column-list" [ASC | DESC];

[ ] = optional
```

This statement will select the employee_id, dept, name, age, and salary from the employee_info table where the dept equals 'Sales' and will list the results in Ascending (default) order based on their Salary.

```
ASC = Ascending Order - default

DESC = Descending Order
```

For example:

```
SELECT employee_id, dept, name, age, salary

FROM employee_info

WHERE dept = 'Sales'
ORDER BY salary;
```

If you would like to order based on multiple columns, you must seperate the columns with commas. For example:

```
SELECT employee_id, dept, name, age, salary

FROM employee_info

WHERE dept = 'Sales'

ORDER BY salary, age DESC;
```

Use these tables for the exercises
items_ordered
customers

Review Exercises

1. Select the lastname, firstname, and city for all customers in the customers table. Display the results in Ascending Order based on the lastname.
2. Same thing as exercise #1, but display the results in Descending order.
3. Select the item and price for all of the items in the items_ordered table that the price is greater than 10.00. Display the results in Ascending order based on the price.

Exercise #1

```
SELECT lastname, firstname, city
FROM customers
ORDER BY lastname;
```

Exercise #2

```
SELECT lastname, firstname, city
FROM customers
ORDER BY lastname DESC;
```

Exercise #3

```
SELECT item, price
FROM items_ordered
WHERE price > 10.00
ORDER BY price ASC;
```

Combining conditions and Boolean Operators

The AND operator can be used to join two or more conditions in the WHERE clause. Both sides of the AND condition must be true in order for the condition to be met and for those rows to be displayed.

```
SELECT column1,
SUM(column2)

FROM "list-of-tables"

WHERE "condition1" AND
"condition2";
```

The OR operator can be used to join two or more conditions in the WHERE clause also. However, **either** side of the OR operator can be true and the condition will be met - hence, the rows will be displayed. With the OR operator, either side can be true or both sides can be true.

For example:

```
SELECT employeeid, firstname, lastname, title, salary

FROM employee_info

WHERE salary >= 50000.00 AND title = 'Programmer';
```

This statement will select the employeeid, firstname, lastname, title, and salary from the employee_info table where the salary is greater than or equal to 50000.00 AND the title is equal to 'Programmer'. Both of these conditions must be true in order for the rows to be returned in the query. If either is false, then it will not be displayed.

Although they are not required, you can use paranthesis around your conditional expressions to make it easier to read:

```
SELECT employeeid, firstname, lastname, title, salary

FROM employee_info

WHERE (salary >= 50000.00) AND (title = 'Programmer');
```

Another Example:

SELECT firstname, lastname, title, salary

FROM employee_info

WHERE (title = 'Sales') OR (title = 'Programmer');

This statement will select the firstname, lastname, title, and salary from the employee_info table where the title is either equal to 'Sales' OR the title is equal to 'Programmer'.

Use these tables for the exercises
items_ordered

customers

Review Exercises

1. Select the customerid, order_date, and item from the items_ordered table for all items unless they are 'Snow Shoes' or if they are 'Ear Muffs'. Display the rows as long as they are not either of these two items.
2. Select the item and price of all items that start with the letters 'S', 'P', or 'F'.

Combining conditions and Boolean Operators

The AND operator can be used to join two or more conditions in the WHERE clause. Both sides of the AND condition must be true in order for the condition to be met and for those rows to be displayed.

```
SELECT column1,
SUM(column2)

FROM "list-of-tables"

WHERE "condition1" AND
"condition2";
```

The OR operator can be used to join two or more conditions in the WHERE clause also. However, **either** side of the OR operator can be true and the condition will be met - hence, the rows will be displayed. With the OR operator, either side can be true or both sides can be true.

For example:

```
SELECT employeeid, firstname, lastname, title, salary

FROM employee_info

WHERE salary >= 50000.00 AND title = 'Programmer';
```

This statement will select the employeeid, firstname, lastname, title, and salary from the employee_info table where the salary is greater than or equal to 50000.00 AND the title is equal to 'Programmer'. Both of these conditions must be true in order for the rows to be returned in the query. If either is false, then it will not be displayed.

Although they are not required, you can use paranthesis around your conditional expressions to make it easier to read:

```
SELECT employeeid, firstname, lastname, title, salary

FROM employee_info

WHERE (salary >= 50000.00) AND (title = 'Programmer');
```

Another Example:

SELECT firstname, lastname, title, salary

FROM employee_info

```
WHERE (title = 'Sales') OR (title = 'Programmer');
```

This statement will select the firstname, lastname, title, and salary from the employee_info table where the title is either equal to 'Sales' OR the title is equal to 'Programmer'.

Use these tables for the exercises
items_ordered
customers

Review Exercises

1. Select the customerid, order_date, and item from the items_ordered table for all items unless they are 'Snow Shoes' or if they are 'Ear Muffs'. Display the rows as long as they are not either of these two items.
2. Select the item and price of all items that start with the letters 'S', 'P', or 'F'.

IN and BETWEEN Conditional Operators

```
SELECT col1, SUM(col2)

FROM "list-of-tables"

WHERE col3 IN
       (list-of-values);

SELECT col1, SUM(col2)

FROM "list-of-tables"

WHERE col3 BETWEEN value1
AND value2;
```

The IN conditional operator is really a set membership test operator. That is, it is used to test whether or not a value (stated before the keyword IN) is "in" the list of values provided after the keyword **IN**.

For example:

```
SELECT employeeid, lastname, salary

FROM employee_info

WHERE lastname IN ('Hernandez', 'Jones', 'Roberts', 'Ruiz');
```

This statement will select the employeeid, lastname, salary from the employee_info table where the lastname is equal to either: Hernandez, Jones, Roberts, or Ruiz. It will return the rows if it is ANY of these values.

The IN conditional operator can be rewritten by using compound conditions using the equals operator and combining it with OR - with exact same output results:

```
SELECT employeeid, lastname, salary

FROM employee_info

WHERE lastname = 'Hernandez' OR lastname = 'Jones' OR lastname = 'Roberts'
OR lastname = 'Ruiz';
```

As you can see, the IN operator is much shorter and easier to read when you are testing for more than two or three values.

You can also use **NOT IN** to exclude the rows in your list.

The BETWEEN conditional operator is used to test to see whether or not a value (stated before the keyword BETWEEN) is "between" the two values stated after the keyword BETWEEN.

For example:

```
SELECT employeeid, age, lastname, salary

FROM employee_info

WHERE age BETWEEN 30 AND 40;
```

This statement will select the employeeid, age, lastname, and salary from the employee_info table where the age is between 30 and 40 (including 30 and 40).

This statement can also be rewritten without the BETWEEN operator:

```
SELECT employeeid, age, lastname, salary

FROM employee_info

WHERE age >= 30 AND age <= 40;
```

You can also use **NOT BETWEEN** to exclude the values between your range.

Use these tables for the exercises
items_ordered
customers

Review Exercises

1. Select the date, item, and price from the items_ordered table for all of the rows that have a price value ranging from 10.00 to 80.00.
2. Select the firstname, city, and state from the customers table for all of the rows where the state value is either: Arizona, Washington, Oklahoma, Colorado, or Hawaii.

Table Joins, a must

All of the queries up until this point have been useful with the exception of one major limitation - that is, you've been selecting from only one table at a time with your SELECT statement. It is time to introduce you to one of the most beneficial features of SQL & relational database systems - the "**Join**". To put it simply, the "Join" makes relational database systems "relational".

Joins allow you to link data from two or more tables together into a single query result--from one single SELECT statement.

A "Join" can be recognized in a SQL SELECT statement if it has more than one table after the FROM keyword.

For example:

```
SELECT "list-of-columns"
FROM table1,table2
WHERE "search-condition(s)"
```

Joins can be explained easier by demonstrating what would happen if you worked with one table only, and didn't have the ability to use "joins". This single table database is also sometimes referred to as a "flat table". Let's say you have a one-table database that is used to keep track of all of your customers and what they purchase from your store:

id	first	last	address	city	state	zip	date	item	price

Everytime a new row is inserted into the table, all columns will be be updated, thus resulting in unnecessary "redundant data". For example, every time Wolfgang Schultz purchases something, the following rows will be inserted into the table:

id	first	last	address	city	state	zip	date	item	price
10982	Wolfgang	Schultz	300 N. 1st Ave	Yuma	AZ	85002	032299	snowboard	45.00
10982	Wolfgang	Schultz	300 N. 1st Ave	Yuma	AZ	85002	082899	snow shovel	35.00
10982	Wolfgang	Schultz	300 N. 1st Ave	Yuma	AZ	85002	091199	gloves	15.00
10982	Wolfgang	Schultz	300 N. 1st Ave	Yuma	AZ	85002	100999	lantern	35.00
10982	Wolfgang	Schultz	300 N. 1st Ave	Yuma	AZ	85002	022900	tent	85.00

An ideal database would have two tables:

1. One for keeping track of your customers
2. And the other to keep track of what they purchase:

"Customer_info" table:

customer_number	firstname	lastname	address	city	state	zip

"Purchases" table:

customer_number	date	item	price

Now, whenever a purchase is made from a repeating customer, the 2nd table, "Purchases" only needs to be updated! We've just eliminated useless redundant data, that is, we've just **normalized** this database!

Notice how each of the tables have a common "cusomer_number" column. This column, which contains the unique customer number will be used to **JOIN** the two tables. Using the two new tables, let's say you would like to select the customer's name, and items they've purchased. Here is an example of a join statement to accomplish this:

```
SELECT customer_info.firstname, customer_info.lastname, purchases.item

FROM customer_info, purchases

WHERE customer_info.customer_number = purchases.customer_number;
```

This particular "Join" is known as an "Inner Join" or "Equijoin". This is the most common type of "Join" that you will see or use.

Notice that each of the colums are always preceeded with the table name and a period. This isn't always required, however, it IS good practice so that you wont confuse which colums go with what tables. It is required if the name column names are the same between the two tables. I recommend preceeding all of your columns with the table names when using joins.

Note: The syntax described above will work with most Database Systems -including the one with this tutorial. However, in the event that this doesn't work with yours, please check your specific database documentation.

Although the above will probably work, here is the ANSI SQL-92 syntax specification for an Inner Join using the preceding statement above that you might want to try:

```
SELECT customer_info.firstname, customer_info.lastname, purchases.item

FROM customer_info INNER JOIN purchases

ON customer_info.customer_number = purchases.customer_number;
```

Another example:

```
SELECT employee_info.employeeid, employee_info.lastname,
employee_sales.comission

FROM employee_info, employee_sales

WHERE employee_info.employeeid = employee_sales.employeeid;
```

This statement will select the employeeid, lastname (from the employee_info table), and the comission value (from the employee_sales table) for all of the rows where the employeeid in the employee_info table matches the employeeid in the employee_sales table.

Use these tables for the exercises
items_ordered

customers

Review Exercises

1. Write a query using a join to determine which items were ordered by each of the customers in the customers table. Select the customerid, firstname, lastname, order_date, item, and price for everything each customer purchased in the items_ordered table.
2. Repeat exercise #1, however display the results sorted by state in descending order

Table Join Exercise Answers

Exercise #1

```
SELECT customers.customerid, customers.firstname, customers.lastname,
items_ordered.order_date, items_ordered.item, items_ordered.price
FROM customers, items_ordered
WHERE customers.customerid = items_ordered.customerid;
```

Exercise #2

```
SELECT customers.customerid, customers.firstname, customers.state, items_ordered.item
FROM customers, items_ordered
WHERE customers.customerid = items_ordered.customerid
ORDER BY customers.state DESC;
```