**SQLcourse.com** - <http://www.sqlcourse.com/>

**Welcome to SQLCourse.com!**

This unique introductory SQL tutorial not only provides easy-to-understand SQL instructions, but it allows you to practice what you learn using the on-line SQL interpreter. You will receive immediate results after submitting your SQL commands. You will be able to create your own unique tables as well as perform selects, inserts, updates, deletes, and drops on your tables. This SQL tutorial currently supports a subset of ANSI SQL. The basics of each SQL command will be covered in this introductory tutorial. Unless otherwise stated, the interpreter will support everything covered in this course.

If you're already familar with the basics of SQL, you can still use this as a refresher, and practice some SQL statements.

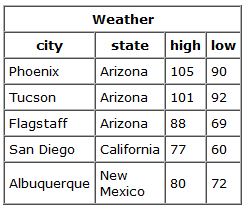
# What is SQL?

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:



**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

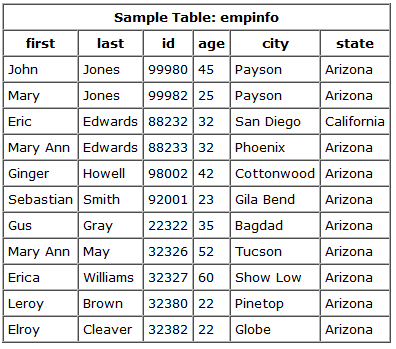
where last LIKE '%s';

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.

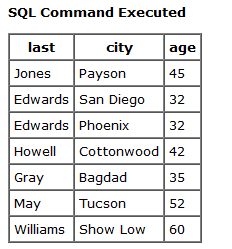


Enter the following sample select statements in the SQL Interpreter Form at the bottom of this page. Before you press "submit", write down your expected results. Press "submit", and compare the results.

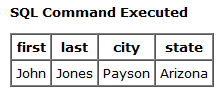
select first, last, city from empinfo;



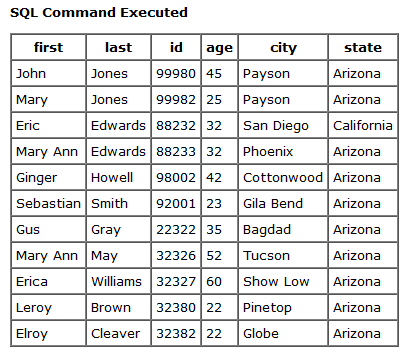
select last, city, age from empinfo where age > 30;



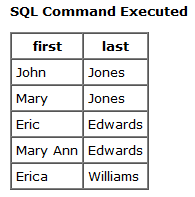
select first, last, city, state from empinfo where first LIKE 'J%';



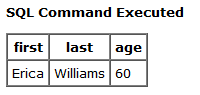
select \* from empinfo;



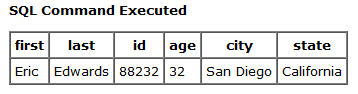
select first, last, from empinfo where last LIKE '%s';



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



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



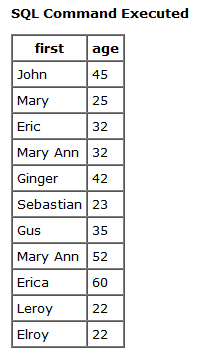
### **Select statement exercises**

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".

### **Solutions to Select statement exercises**

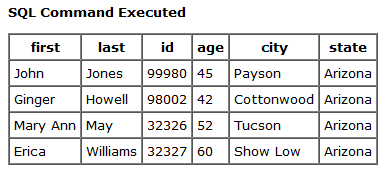
1. select first, age from empinfo;



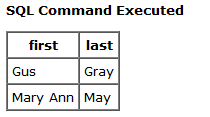
1. select first, last, city from empinfo where city <> 'Payson';



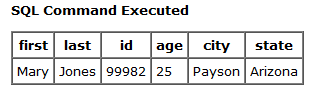
1. select \* from empinfo where age > 40;



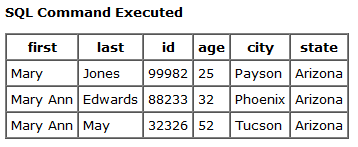
1. select first, last from empinfo where last LIKE '%ay';



1. select \* from empinfo where first = 'Mary';



1. 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),

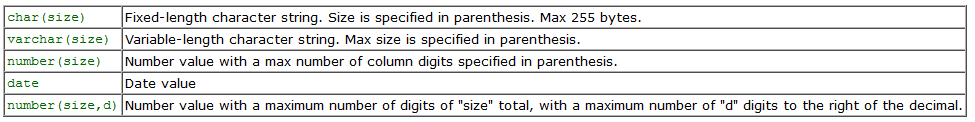
state varchar(20));

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:



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).

### **Solution :**

create table myemployees\_aroy2604

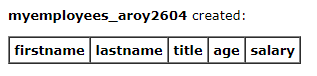
(firstname varchar(30),

lastname varchar(30),

title varchar(50),

age number(3),

salary number(15,2));



# 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.

# Solutions to Insert Statement exercises

Data insertion step :

insert into myemployees\_aroy2604

(firstname, lastname, title, age, salary)

values ('Jonie', 'Weber', 'Secretary', 28, 19500.00);

insert into myemployees\_aroy2604

(firstname, lastname, title, age, salary)

values ('Potsy', 'Weber', 'Programmer', 32, 45300.00);

insert into myemployees\_aroy2604

(firstname, lastname, title, age, salary)

values ('Dirk', 'Smith', 'Programmer II', 45, 75020.00);

insert into myemployees\_aroy2604

(firstname, lastname, title, age, salary)

values ('John', 'Parker', 'Data Analyst I', 35, 69500.00);

insert into myemployees\_aroy2604

(firstname, lastname, title, age, salary)

values ('Ann', 'Roberts', 'Programmer III', 37, 80500.00);

insert into myemployees\_aroy2604

(firstname, lastname, title, age, salary)

values ('Gus', 'Powell', 'Data Analyst II', 29, 100200.00);

insert into myemployees\_aroy2604

(firstname, lastname, title, age, salary)

values ('Eric', 'Nightingale', 'Accountant', 40, 85000.00);

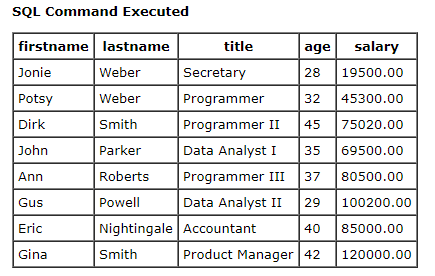
insert into myemployees\_aroy2604

(firstname, lastname, title, age, salary)

values ('Gina', 'Smith', 'Product Manager', 42, 120000.00);

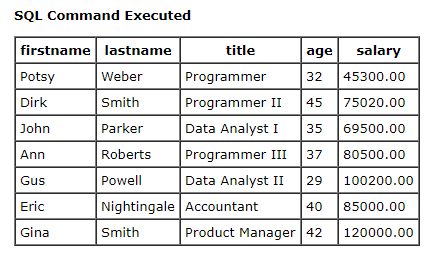
1. Select all columns for everyone in your employee table.

select \* from myemployees\_aroy2604;



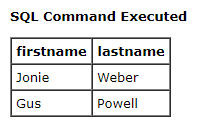
1. Select all columns for everyone with a salary over 30000.

select \* from myemployees\_aroy2604 where salary > 30000;



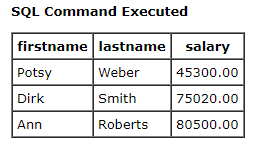
1. Select first and last names for everyone that's under 30 years old.

select firstname, lastname from myemployees\_aroy2604 where age < 30;



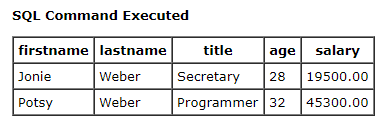
1. Select first name, last name, and salary for anyone with "Programmer" in their title.

select firstname, lastname, salary from myemployees\_aroy2604 where title LIKE '%Programmer%';



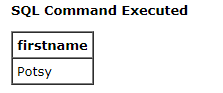
1. Select all columns for everyone whose last name contains "ebe".

select \* from myemployees\_aroy2604 where lastname LIKE '%ebe%';



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

select firstname from myemployees\_aroy2604 where firstname = 'Potsy';



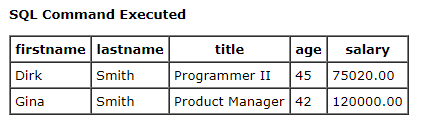
1. Select all columns for everyone over 80 years old.

select \* from myemployees\_aroy2604 where age > 80;



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

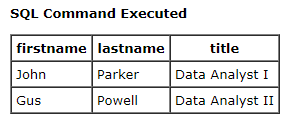
select \* from myemployees\_aroy2604 where lastname LIKE '%ith';



# Questions and Solutions to own select statements

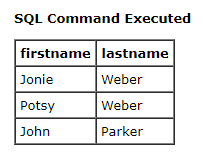
1. Select first name, last name, title for everyone whose title contains ‘Data’.

select firstname, lastname, title from myemployees\_aroy2604 where title LIKE '%Data%';



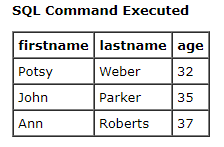
1. Select first name, last name for everyone whose last name ends in ‘er’.

select firstname, lastname from myemployees\_aroy2604 where lastname LIKE '%er';



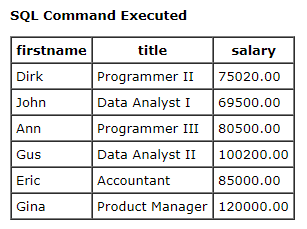
1. Select first name, last name, age for everyone whose age is between 30 to 40 yrs old.

select firstname, lastname, age from myemployees\_aroy2604 where age >= 30 and age < 40;



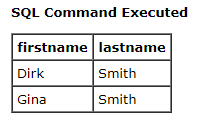
1. Select first name, title, salary for everyone whose salary >= 50000.

select firstname, title, salary from myemployees\_aroy2604 where salary >= 50000;



1. Select first name, last name for everyone whose last name is ‘Smith’.

select firstname, lastname from myemployees\_aroy2604 where lastname = 'Smith';



# 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.

### **Solutions to Update statement exercises**

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

update myemployees\_aroy2604

set lastname = 'Weber-Williams' where firstname = 'Jonie';



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

update myemployees\_aroy2604

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



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

update myemployees\_aroy2604

set title = 'Administrative Assistant' where title = 'Secretary';



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

update myemployees\_aroy2604

set salary = salary + 3500 where salary < 30000;



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

update myemployees\_aroy2604

set salary = salary + 4500 where salary > 33500;



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

update myemployees\_aroy2604

set title = 'Programmer III' where title = 'Programmer II';



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

update myemployees\_aroy2604

set title = 'Programmer II' where title = 'Programmer';



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

update myemployees\_aroy2604

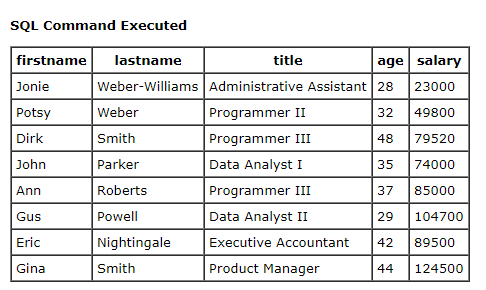
set title = 'Executive Accountant' where title = 'Accountant';

update myemployees\_aroy2604

set age = age + 2 where age >= 40;

\*\* Can’t create more UPDATE statements – poor server connectivity

After performing all updates our table looks like,



**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.

### **Solutions to Delete statement exercises**

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

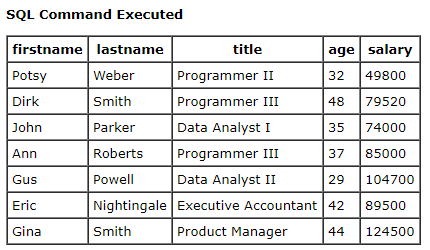
delete from myemployees\_aroy2604

where firstname = 'Jonie' and lastname = 'Weber-Williams';



Verification step :

select \* from myemployees\_aroy2604;



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

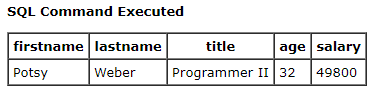
delete from myemployees\_aroy2604

where salary > 70000;



Verification step:

select \* from myemployees\_aroy2604;



# 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.

### **Solution to Drop Table exercises**

drop table myemployees\_aroy2604;



# Welcome to SQLCourse2.com! - [http://www.sqlcourse2.com](http://www.sqlcourse2.com/)

This unique SQL Tutorial is the "sequel" to the highly successful [SQLCourse.com](http://sqlcourse.com/) site and will provide you with more advanced easy-to-follow SQL Instruction and the ability to practice what you learn on-line with immediate feedback! You will receive immediate results on a web page after submitting your SQL Commands.

This continuation course will provide you with critical need-to-know advanced features and clauses of the SELECT statement that weren't supported in the previous SQLCourse.com site. Everything you learn here will be ANSI SQL compliant and should work with most SQL databases such as Oracle, SQL Server, mySQL, MS Access, Informix, Sybase, or any other ANSI SQL compliant database.

If you're already familar with the basics of SQL, you can still use this as a refresher, and practice some SQL statements.

# Start Here - Intro

This Intermediate/Advanced SQL Tutorial will cover the SELECT statement in great detail. The SELECT statement is the core of SQL, and it is likely that the vast majority of your SQL commands will be SELECT statements. Due to the enormous amount of options available for the SELECT statement, this entire tutorial has been dedicated to it.

When constructing SQL Queries (with the SELECT statement), it is very useful to know all of the possible options and the best or more efficient way to do things. This Tutorial will help provide you with those skills.

Although it is recommended that you go through SQLCourse.com (the original site) or at least the SELECT statement on SQLCourse.com, it isn't required. YOu can start with this site, and then proceed to SQLCourse.com when you are finished. SQLCourse.com covers four other SQL Commands and allows you to practice those as well.

Note: This Intermediate/Advanced Tutorial will allow you to practice all of the advanced features covered here on this site. Not all of these commands are available at the original [SQLCourse.com](http://www.sqlcourse.com/) site. **The four other commands covered at**[**SQLCourse.com**](http://www.sqlcourse.com/)**are not supported with this Interpreter on this site**.

**You can submit all of your SQL commands in the Text submission box at the bottom of each of the main pages.**

# 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](http://www.sqlcourse2.com/from_where_review.html)

 Example:

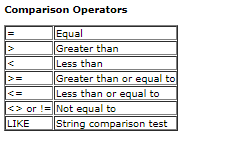
SELECT name, age, salary

FROM employee

WHERE age > 40;

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.



\*[Note about LIKE](http://www.sqlcourse2.com/like.html)

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.

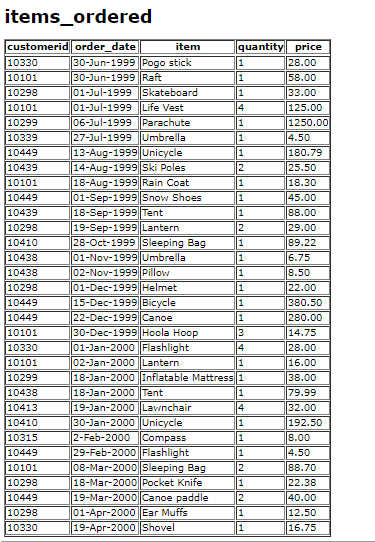
ALL will display "all" of the specified columns including all of the duplicates. The ALL keyword is the default if nothing is specified.

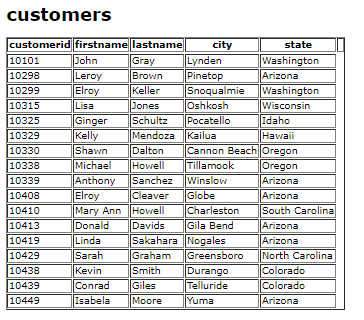
**Note:** The following two tables will be used throughout this course. It is recommended to have them open in another window or print them out.



<http://www.sqlcourse2.com/items_ordered.html>

<http://www.sqlcourse2.com/customers.html>

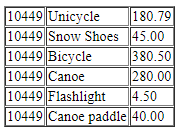




### **Review Exercises**

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.

select customerid, item, price from items\_ordered where customerid = 10449;



1. Select all columns from the items\_ordered table for whoever purchased a **Tent**.

select \* from items\_ordered where item = 'Tent';



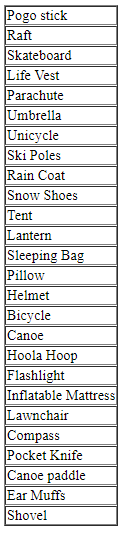
1. Select the customerid, order\_date, and item values from the items\_ordered table for any items in the item column that start with the letter "S".

select customerid, order\_date, item from items\_ordered where item LIKE 'S%';

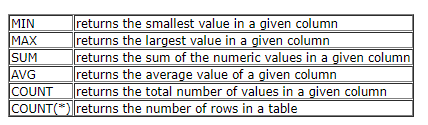


1. Select the distinct items in the items\_ordered table. In other words, display a listing of each of the unique items from the items\_ordered table.

select DISTINCT item from items\_ordered;



# Aggregate Functions



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:

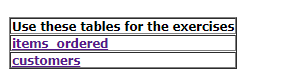
SELECT AVG(salary)  
  
  
FROM employee  
  
WHERE title = 'Programmer';

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

Example:

SELECT Count(\*)  
  
FROM employee;

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.

’

<http://www.sqlcourse2.com/items_ordered.html>

<http://www.sqlcourse2.com/customers.html>

**Review Exercises**

1. Select the maximum price of any item ordered in the items\_ordered table. Hint: Select the maximum price only.

select MAX(price) from items\_ordered;



1. Select the average price of all of the items ordered that were purchased in the month of Dec.

select AVG(price) from items\_ordered where

order\_date LIKE '%Dec%';



1. What are the total number of rows in the items\_ordered table?

select COUNT(\*) from items\_ordered;



1. 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.

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:

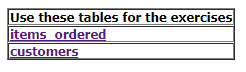
SELECT column1,   
SUM(column2)  
   
FROM "list-of-tables"  
   
GROUP BY "column-list";

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](http://www.sqlcourse2.com/groupby-1.html) - What if I wanted to display their lastname too?



<http://www.sqlcourse2.com/items_ordered.html>

<http://www.sqlcourse2.com/customers.html>

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:

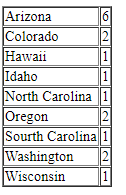
SELECT quantity, max(price)  
   
FROM items\_ordered  
   
GROUP BY quantity;

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**

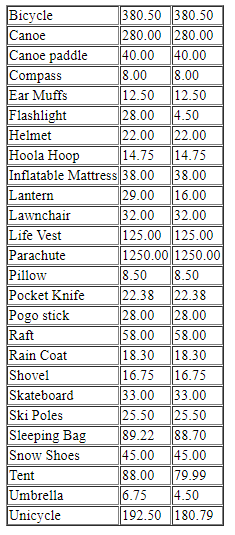
1. How many people are in each unique state in the customers table? Select the state and display the number of people in each. Hint: **count** is used to count rows in a column, **sum** works on numeric data only.

select state, COUNT(\*) from customers GROUP BY state;



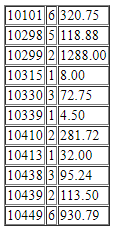
1. From the items\_ordered table, select the item, maximum price, and minimum price for each specific item in the table. Hint: The items will need to be broken up into separate groups.

select item, MAX(price), MIN(price) from items\_ordered GROUP BY item;



1. 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.

select customerid, COUNT(quantity), 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;

<http://www.sqlcourse2.com/items_ordered.html>

<http://www.sqlcourse2.com/customers.html>

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

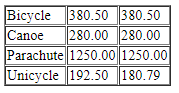
1. How many people are in each unique state in the customers table that have more than one person in the state? Select the state and display the number of how many people are in each if it's greater than 1.

select state, COUNT(\*) from customers GROUP BY state HAVING COUNT(\*) > 1;



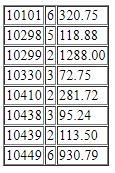
1. From the items\_ordered table, select the item, maximum price, and minimum price for each specific item in the table. Only display the results if the maximum price for one of the items is greater than 190.00.

select item, MAX(price), MIN(price) from items\_ordered GROUP BY item HAVING MAX(price) > 190.00;



1. 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.

select customerid, COUNT(quantity), SUM(price) from items\_ordered GROUP BY customerid HAVING COUNT(quantity) > 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;

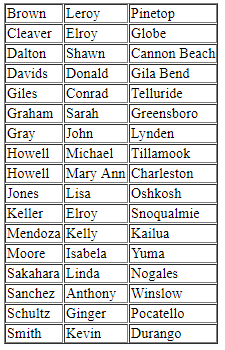
<http://www.sqlcourse2.com/items_ordered.html>

<http://www.sqlcourse2.com/customers.html>

**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.

select lastname, firstname, city from customers ORDER BY lastname;



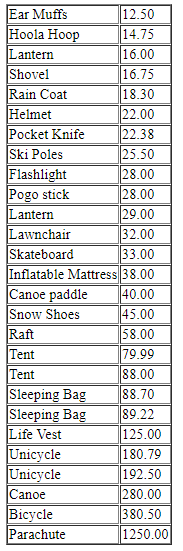
1. Same thing as exercise #1, but display the results in Descending order.

select lastname, firstname, city from customers ORDER BY lastname DESC;



1. 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.

select item, price from items\_ordered where price > 10.00 ORDER BY price;



# Combining Conditions & 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 >= 45000.00 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 >= 45000.00) 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'.

<http://www.sqlcourse2.com/items_ordered.html>

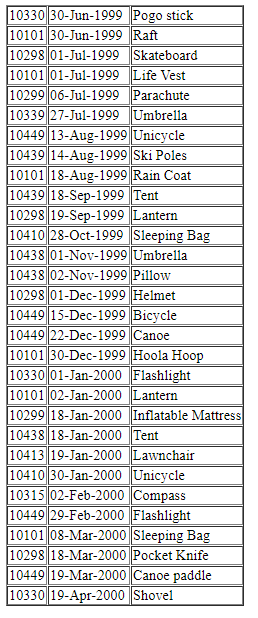
<http://www.sqlcourse2.com/customers.html>

**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.

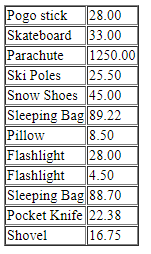
select customerid, order\_date, item from items\_ordered where item <> 'Snow Shoes' AND

item <> 'Ear Muffs';



1. Select the item and price of all items that start with the letters 'S', 'P', or 'F'.

select item, price from items\_ordered where (item LIKE 'S%') OR (item LIKE 'P%') OR (item LIKE 'F%');



# IN & BETWEEN

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.

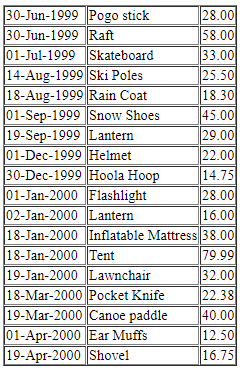
<http://www.sqlcourse2.com/items_ordered.html>

<http://www.sqlcourse2.com/customers.html>

**Review Exercises**

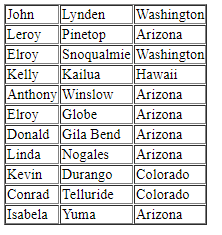
1. Select the date, item, and price from the items\_ordered table for all of the rows that have a price value ranging from 10.00 to 80.00.

SELECT order\_date, item, price FROM items\_ordered WHERE price BETWEEN 10.00 AND 80.00;



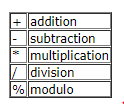
1. Select the firstname, city, and state from the customers table for all of the rows where the state value is either: Arizona, Washington, Oklahoma, Colorado, or Hawaii.

SELECT firstname, city, state FROM customers WHERE state IN ('Arizona', 'Washington', 'Oklahoma', 'Colorado', 'Hawaii');

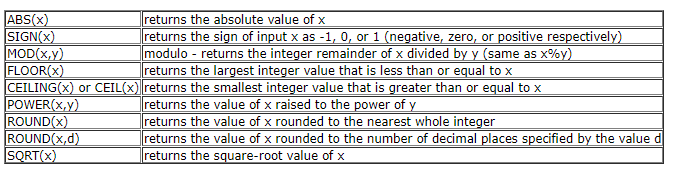


# Mathematical Functions

Standard ANSI SQL-92 supports the following first four basic arithmetic operators:



The modulo operator determines the integer remainder of the division. This operator is not ANSI SQL supported, however, most databases support it. The following are some more useful mathematical functions to be aware of since you might need them. These functions are not standard in the ANSI SQL-92 specs, therefore they may or may not be available on the specific RDBMS that you are using. However, they were available on several major database systems that I tested. They WILL work on this tutorial.



For example:

SELECT round(salary), firstname  
  
FROM employee\_info

This statement will select the salary rounded to the nearest whole value and the firstname from the employee\_info table.

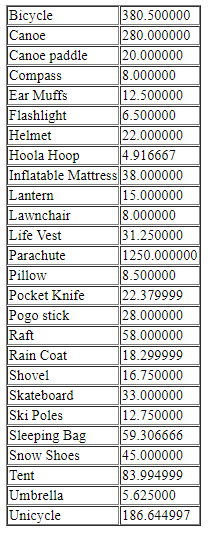
<http://www.sqlcourse2.com/items_ordered.html>

<http://www.sqlcourse2.com/customers.html>

**Review Exercises**

1. Select the item and per unit price for each item in the items\_ordered table. Hint: Divide the price by the quantity.

SELECT item, SUM(price)/SUM(quantity) AS per\_unit\_price FROM items\_ordered GROUP BY item;



# 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.

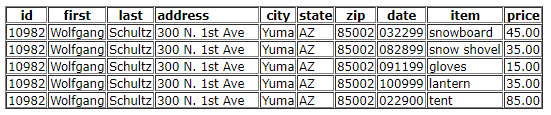
For example:

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:



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:



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:



"Purchases" table:



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**](http://www.sqlcourse2.com/normalization.html) 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.

<http://www.sqlcourse2.com/items_ordered.html>

<http://www.sqlcourse2.com/customers.html>

**Review Exercises**

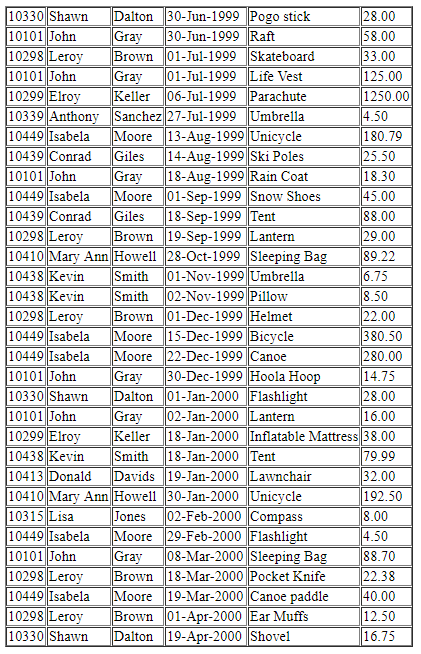
1. Write a query using a join to determine which items were ordered by each of the customers in the customers table. Select the customerid, firstname, lastname, order\_date, item, and price for everything each customer purchased in the items\_ordered table.

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;



1. Repeat exercise #1, however display the results sorted by state in descending order.

SELECT customers.customerid, customers.firstname, customers.lastname,customers.state,

items\_ordered.order\_date, items\_ordered.item, items\_ordered.price FROM

customers, items\_ordered

WHERE customers.customerid = items\_ordered.customerid ORDER BY customers.state DESC;

