* 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 |

* Format of using SELECT
* 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 |

* Display the first name and age for everyone that's in the table.
* **SELECT first, last from empinfo;**
* Display the first name, last name, and city for everyone that's not from Payson.
* **SELECT first, last, city from empinfo where city <> ‘Payson’;**
* Display all columns for everyone that is over 40 years old.
* **SELECT \* from empinfo where age > 40;**
* Display the first and last names for everyone whose last name ends in an "ay".
* **SELECT first, last from empinfo where last LIKE ‘%ay’;**
* Display all columns for everyone whose first name equals "Mary".
* **SELECT \* from empinfo where first = 'Mary';**
* Display all columns for everyone whose first name contains "Mary".
* **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).
* CREATE TABLE myemployees\_anna0820
* (
* firstname varchar(20),
* lastname varchar(20),
* title varchar(20),
* age number(3),
* salary number(7)
* );
* **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.
* INSERT INTO myemployees\_anna0820
* (
* firstname,
* lastname,
* title,
* age,
* salary)
* values
* ('Jonie', 'Weber', 'Secretary', 28,19500.00 );
* INSERT INTO myemployees\_anna0820
* (
* firstname,
* lastname,
* title,
* age,
* salary)
* values
* ('Dirk',
* 'Smith',
* 'Programmer II',
* 45,
* 75020.00 );
* INSERT INTO myemployees\_anna0820
* (
* firstname,
* lastname,
* title,
* age,
* salary)
* values
* (‘Potsy’,
* ' Weber ',
* 'Programmer',
* 32,
* 45300.00 );
* INSERT INTO myemployees\_anna0820
* (
* firstname,
* lastname,
* title,
* age,
* salary)
* values
* ('Anna',
* 'Mathew',
* 'Programmer II',
* 27,
* 70000.00 );
* INSERT INTO myemployees\_anna0820
* (
* firstname,
* lastname,
* title,
* age,
* salary)
* values
* ('Mathew',
* 'K',
* 'Programmer II',
* 31,
* 150000.00 );
* After they're inserted into the table, enter select statements to:
* Select all columns for everyone in your employee table.
* Select all columns for everyone with a salary over 30000.
* Select first and last names for everyone that's under 30 years old.
* Select first name, last name, and salary for anyone with "Programmer" in their title.
* Select all columns for everyone whose last name contains "ebe".
* Select the first name for everyone whose first name equals "Potsy".
* Select all columns for everyone over 80 years old.
* 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.