STRUCTURED QUERY LANGUAGE

SQL is a standard language for accessing and manipulating databases.

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL is an ANSI (American National Standards Institute) standard

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

SQL General Data Types

Data type	Access	SQLServer	Oracle	MySQL	PostgreSQL
boolean	Yes/No	Bit	Byte	N/A	Boolean
integer	Number (integer)	Int	Number	Int Integer	Int Integer
float	Number (single)	Float Real	Number	Float	Numeric
currency	Currency	Money	N/A	N/A	Money
string (fixed)	N/A	Char	Char	Char	Char
string (variable)	Text (<256) Memo (65k+)	Varchar	Varchar Varchar2	Varchar	Varchar
binary object	OLE Object Memo	Binary (fixed up to 8K) Varbinary (<8K) Image (<2GB)	Long Raw	Blob Text	Binary Varbinary

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

SQL CREATE TABLE Statement

- The CREATE TABLE statement is used to create a table in a database.
- Tables are organized into rows and columns; and each table must have a name.

- SQL CREATE TABLE Syntax
  - CREATE TABLE *table_name*
    (
    column_name1 data_type(*size*),
    column_name2 data_type(*size*),
    column_name3 data_type(*size*),
    ....
    );

- Example
- CREATE TABLE Persons
  (
  PersonID int,
  LastName varchar(255),
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255)
  );

SQL INSERT INTO Statement

- The INSERT INTO statement is used to insert new records in a table.

- INSERT INTO *table_name*
  VALUES (*value1*,*value2*,*value3*,...);

- INSERT INTO *table_name* (*column1*,*column2*,*column3*,...)
  VALUES (*value1*,*value2*,*value3*,...);

- Example
- INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
  VALUES ('Cardinal','Tom B. Erichsen','Skagen 21','Stavanger','4006','Norway');

SQL SELECT Statement

- The SELECT statement is used to select data from a database.
- The result is stored in a result table, called the result-set.

- SELECT * FROM *table_name*;

- SELECT *column_name*,*column_name*
  FROM *table_name*;

SQL SELECT DISTINCT Statement

- In a table, a column may contain many duplicate values; and sometimes you only want to list the different (distinct) values.
- The DISTINCT keyword can be used to return only distinct (different) values.
  - SQL SELECT DISTINCT Syntax
  - SELECT DISTINCT *column_name*,*column_name*
    FROM *table_name*;
- Example
- SELECT DISTINCT City FROM Customers;

SQL WHERE Clause

- The WHERE clause is used to extract only those records that fulfill a specified criterion.

- SQL WHERE Syntax
  - SELECT *column_name*,*column_name*
    FROM *table_name*
    WHERE *column_name operator value*;
- Example
- SELECT * FROM Customers
  WHERE Country='Mexico';

Operators in The WHERE Clause

Operator	Description
=	Equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

SQL BETWEEN Operator

- The BETWEEN operator selects values within a range. The values can be numbers, text, or dates.

- SQL BETWEEN Syntax
  - SELECT *column_name(s)*
    FROM *table_name*
    WHERE *column_name* BETWEEN *value1* AND *value2;*

- Example
- SELECT * FROM Products
  WHERE Price BETWEEN 10 AND 20;
-

  Example
- SELECT * FROM Products
  WHERE Price NOT BETWEEN 10 AND 20;

The IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.

- SQL IN Syntax
- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *column_name* IN (*value1*,*value2*,...);

- Example
- SELECT * FROM Customers
  WHERE City IN ('Paris','London');

SQL LIKE Operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

- SQL LIKE Syntax
- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *column_name* LIKE *pattern*;

- SELECT * FROM Customers
  WHERE City LIKE 's%';

SQL Wildcards

- In SQL, wildcard characters are used with the SQL LIKE operator.
- SQL wildcards are used to search for data within a table.

Wildcard	Description
%	A substitute for zero or more characters
_	A substitute for a single character

- Example
- SELECT * FROM Customers
  WHERE City LIKE 'L_n_on';

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

SQL AND & OR Operators

- The AND operator displays a record if both the first condition AND the second condition are true.
- The OR operator displays a record if either the first condition OR the second condition is true.
- Example
  - SELECT * FROM Customers
    WHERE Country='Germany'
    AND City='Berlin';
- Example
  - SELECT * FROM Customers
    WHERE Country='Germany'
    AND City='Berlin';

SQL ORDER BY Keyword

- The ORDER BY keyword is used to sort the result-set by one or more columns.
- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in a descending order, you can use the DESC keyword.
- SQL ORDER BY Syntax
  - SELECT *column_name*,*column_name*
    FROM *table_name*
    ORDER BY *column_name*,*column_name* ASC|DESC;

- Example
- SELECT * FROM Customers
  ORDER BY Country;
- Example
- SELECT * FROM Customers
  ORDER BY Country DESC;

SQL UPDATE Statement

- The UPDATE statement is used to update existing records in a table.

- SQL UPDATE Syntax
  - UPDATE *table_name*
    SET *column1=value1*,*column2=value2*,...
    WHERE *some_column=some_value*;

- Example
- UPDATE Customers
  SET ContactName='Alfred Schmidt', City='Hamburg'
  WHERE CustomerName='Alfreds Futterkiste';
-

SQL DELETE Statement

- The DELETE statement is used to delete rows in a table.

- Example
- DELETE FROM Customers
  WHERE CustomerName='Alfreds Futterkiste' AND ContactName='Maria Anders';

SQL Constraints

- SQL constraints are used to specify rules for the data in a table.
- Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

- SQL CREATE TABLE + CONSTRAINT Syntax
- CREATE TABLE *table_name*
  (
  column_name1 data_type(*size*) *constraint_name*,
  column_name2 data_type(*size*) *constraint_name*,
  column_name3 data_type(*size*) *constraint_name*,
  ....
  );

- In SQL, we have the following constraints:
- **NOT NULL** - Indicates that a column cannot store NULL value
- **UNIQUE** - Ensures that each row for a column must have a unique value
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have an unique identity which helps to find a particular record in a table more easily and quickly
- **FOREIGN KEY** - Ensure the referential integrity of the data in one table to match values in another table
- **CHECK** - Ensures that the value in a column meets a specific condition
- **DEFAULT** - Specifies a default value when specified none for this column

SQL NOT NULL Constraint

- The NOT NULL constraint enforces a column to NOT accept NULL values.

- Example
- CREATE TABLE PersonsNotNull
```
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

SQL UNIQUE Constraint

- The UNIQUE constraint uniquely identifies each record in a database table.
- The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.
- Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

SQL UNIQUE Constraint on CREATE TABLE

- The following SQL creates a UNIQUE constraint on the "P_Id" column when the "Persons" table is created:
- **SQL Server / Oracle / MS Access:**
- CREATE TABLE Persons

```
(
P_Id int NOT NULL UNIQUE,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

SQL UNIQUE Constraint on ALTER TABLE

- To create a UNIQUE constraint on the "P_Id" column when the table is already created, use the following SQL:
- **MySQL / SQL Server / Oracle / MS Access:**
- ALTER TABLE Persons
  ADD UNIQUE (P_Id)

To DROP a UNIQUE Constraint

- To drop a UNIQUE constraint, use the following SQL:
- **SQL Server / Oracle / MS Access:**
- ALTER TABLE Persons
  DROP CONSTRAINT uc_PersonID

SQL PRIMARY KEY Constraint

- The PRIMARY KEY constraint uniquely identifies each record in a database table.
- Primary keys must contain unique values.
- A primary key column cannot contain NULL values.
- Most tables should have a primary key, and each table can have only ONE primary key.

- CREATE TABLE Persons
  (
  P_Id int NOT NULL PRIMARY KEY,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255)
)

SQL PRIMARY KEY Constraint on ALTER TABLE

- ALTER TABLE Persons
  ADD PRIMARY KEY (P_Id)

To DROP a PRIMARY KEY Constraint

- ALTER TABLE Persons
  DROP CONSTRAINT pk_PersonID

Difference between Primary Key and Unique Key

Primary Key	Unique Key
Unique identifier for rows of a table	Unique identifier for rows of a table when primary key is not present
Cannot be NULL	Can be NULL
Only one primary key can be present in a table	Multiple Unique Keys can be present in a table
present in a table	present in a table
Selection using primary key creates clustered index	Selection using unique key creates non-clustered index

SQL FOREIGN KEY Constraint

- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.
-

Person Table

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	Order No	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

```
CREATE TABLE Orders
(
O_Id int NOT NULL PRIMARY KEY,
OrderNo int NOT NULL,
P_Id int FOREIGN KEY REFERENCES Persons(P_Id)
)
```

SQL CHECK Constraint

- The CHECK constraint is used to limit the value range that can be placed in a column.
- If you define a CHECK constraint on a single column it allows only certain values for this column.
- If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

- CREATE TABLE Persons
  (
  P_Id int NOT NULL CHECK (P_Id>0),
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255)
)

SQL DEFAULT Constraint

- The DEFAULT constraint is used to insert a default value into a column.

-
```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255) DEFAULT 'Sandnes'
)
```

SQL Aliases

- SQL aliases are used to give a database table, or a column in a table, a temporary name.
- Basically aliases are created to make column names more readable.

- SQL Alias Syntax for Columns
- SELECT *column_name* AS *alias_name*
  FROM *table_name;*

- SQL Alias Syntax for Tables
- SELECT *column_name(s)*
  FROM *table_name* AS *alias_name;*

- Example
- SELECT CustomerName AS Customer, ContactName AS [Contact Person]
  FROM Customers;

- Example
- SELECT o.OrderID, o.OrderDate, c.CustomerName
  FROM Customers AS c, Orders AS o
  WHERE c.CustomerName="Around the Horn" AND c.CustomerID=o.CustomerID;

SQL ALTER TABLE Statement

- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
-
```
ALTER TABLE table_name
ADD column_name datatype
```

- ALTER TABLE table_name
  DROP COLUMN column_name

- Now we want to add a column named "DateOfBirth" in the "Persons" table.
- ALTER TABLE Persons
  ADD DateOfBirth date
- Now we want to change the data type of the column named "DateOfBirth" in the "Persons" table.
- ALTER TABLE Persons
  ALTER COLUMN DateOfBirth year

- Next, we want to delete the column named "DateOfBirth" in the "Persons" table.
- ALTER TABLE Persons
  DROP COLUMN DateOfBirth

The DROP TABLE Statement

- The DROP TABLE statement is used to delete a table.
  - DROP TABLE table_name

SQL Views

- A view is a virtual table.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

- SQL CREATE VIEW Syntax
  - CREATE VIEW view_name AS
    SELECT column_name(s)
    FROM table_name
    WHERE condition
- Select * from view name

SQL Functions- Aggregate Functions

- SQL aggregate functions return a single value, calculated from values in a column.
- Useful aggregate functions:
- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- FIRST() - Returns the first value
- LAST() - Returns the last value
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum

SQL Scalar functions

- SQL scalar functions return a single value, based on the input value.
- Useful scalar functions:
- UCASE() - Converts a field to upper case
- LCASE() - Converts a field to lower case
- MID() - Extract characters from a text field
- LEN() - Returns the length of a text field
- ROUND() - Rounds a numeric field to the number of decimals specified
- NOW() - Returns the current system date and time
- FORMAT() - Formats how a field is to be displayed

- SELECT AVG(Price) AS PriceAverage FROM Products;
- SELECT COUNT(CustomerID) AS OrdersFromCustomerID7 FROM Orders
  WHERE CustomerID=7;

- SELECT MAX(Price) AS HighestPrice FROM Products;

SQL GROUP BY Statement

- The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

- SQL GROUP BY Syntax

  -
    ```
    SELECT column_name, aggregate_function(column_name)
    FROM table_name
    WHERE column_name operator value
    GROUP BY column_name;
    ```

- SELECT Shippers.ShipperName,COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders
  LEFT JOIN Shippers
  ON Orders.ShipperID=Shippers.ShipperID
  GROUP BY ShipperName;

SQL HAVING Clause

- The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

- SQL HAVING Syntax

  -
    ```
    SELECT column_name, aggregate_function(column_name)
    FROM table_name
    WHERE column_name operator value
    GROUP BY column_name
    HAVING aggregate_function(column_name) operator value;
    ```

- Example
- SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM (Orders
  INNER JOIN Employees
  ON Orders.EmployeeID=Employees.EmployeeID)
  GROUP BY LastName
  HAVING COUNT(Orders.OrderID) > 10;
-

- Example
- SELECT UCASE(CustomerName) AS Customer, City
  FROM Customers;

- Example
- SELECT ProductName, ROUND(Price,0) AS RoundedPrice
  FROM Products;

FORMAT() Function

- The FORMAT() function is used to format how a field is to be displayed.
- SQL FORMAT() Syntax
- SELECT FORMAT(column_name,format) FROM table_name;

- Example
- SELECT ProductName, Price, FORMAT(Now(),'YYYY-MM-DD') AS PerDate
  FROM Products;

SQL Date Functions

Function	Description
GETDATE()	Returns the current date and time
DATEPART()	Returns a single part of a date/time
DATEADD()	Adds or subtracts a specified time interval from a date
DATEDIFF()	Returns the time between two dates
CONVERT()	Displays date/time data in different formats

SQL Server comes with the following data types for storing a date or a date/time value in the database:
DATE - format YYYY-MM-DD
DATETIME - format: YYYY-MM-DD HH:MI:SS
SMALLDATETIME - format: YYYY-MM-DD HH:MI:SS
TIMESTAMP - format: a unique number

SQL Joins

- An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

- Example
- SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
  FROM Orders
  INNER JOIN Customers
  ON Orders.CustomerID=Customers.CustomerID;

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

Different SQL JOINs

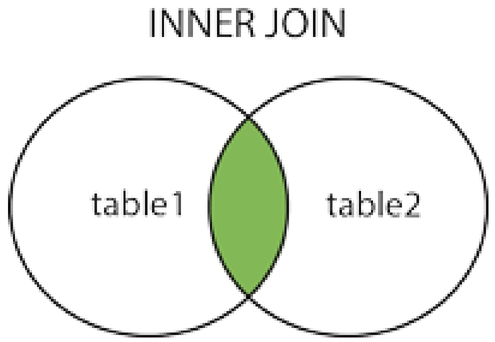
- **INNER JOIN**: Returns all rows when there is at least one match in BOTH tables
- **LEFT JOIN**: Return all rows from the left table, and the matched rows from the right table
- **RIGHT JOIN**: Return all rows from the right table, and the matched rows from the left table
- **FULL JOIN**: Return all rows when there is a match in ONE of the tables
-

SQL INNER JOIN Keyword

- The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns in both tables.

SQL INNER JOIN Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name=table2.column_name;
or:
SELECT column_name(s)
FROM table1
JOIN table2
ON table1.column_name=table2.column_name;
```



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

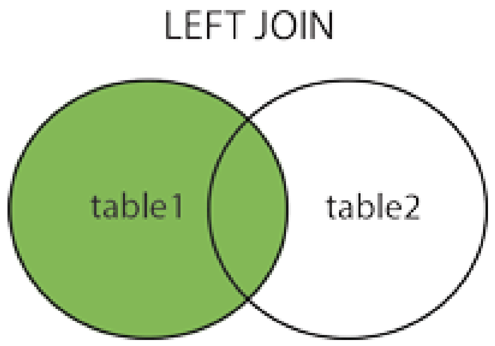
SQL INNER JOIN Example

- Example
- SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  INNER JOIN Orders
  ON Customers.CustomerID=Orders.CustomerID
  ORDER BY Customers.CustomerName;

SQL LEFT JOIN Keyword

- The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.

SQL LEFT JOIN Syntax
SELECT *column_name(s)*
FROM *table1*
LEFT JOIN *table2*
ON *table1.column_name=table2.column_name*;
or:
SELECT *column_name(s)*
FROM *table1*
LEFT OUTER JOIN *table2*
ON *table1.column_name=table2.column_name*;



SQL LEFT JOIN Example

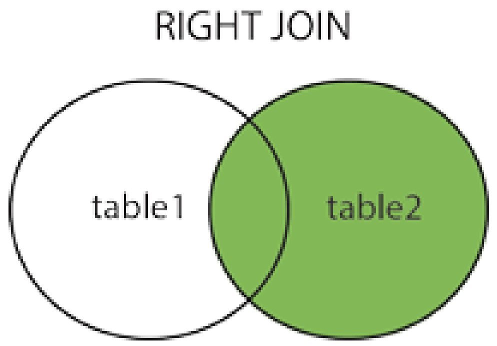
- Example
- SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  LEFT JOIN Orders
  ON Customers.CustomerID=Orders.CustomerID
  ORDER BY Customers.CustomerName;

SQL RIGHT JOIN Keyword

- The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

SQL RIGHT JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
RIGHT JOIN *table2*
ON *table1.column_name=table2.column_name*;
or:
SELECT *column_name(s)*
FROM *table1*
RIGHT OUTER JOIN *table2*
ON *table1.column_name=table2.column_name*;



SQL RIGHT JOIN Example

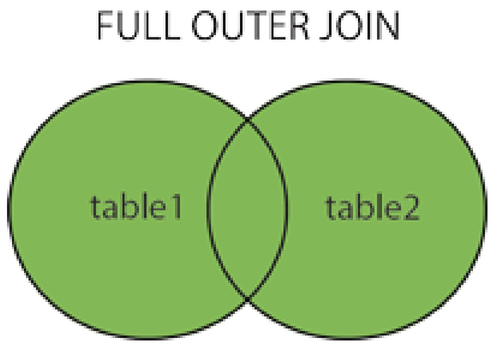
- Example
- SELECT Orders.OrderID, Employees.FirstName
  FROM Orders
  RIGHT JOIN Employees
  ON Orders.EmployeeID=Employees.EmployeeID
  ORDER BY Orders.OrderID;

SQL FULL OUTER JOIN Keyword

- The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).
- The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

SQL FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name=table2.column_name;
```



SQL FULL OUTER JOIN Example

- SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  FULL OUTER JOIN Orders
  ON Customers.CustomerID=Orders.CustomerID
  ORDER BY Customers.CustomerName;

SQL SELECT INTO Statement

- With SQL, you can copy information from one table into new table.
- The SELECT INTO statement copies data from one table and inserts it into a new table.
- SQL SELECT INTO Syntax
  We can copy all columns into the new table:

  ```
  SELECT *
  INTO newtable [IN externaldb]
  FROM table1;
  ```

  Or we can copy only the columns we want into the new table:

  ```
  SELECT column_name(s)
  INTO newtable [IN externaldb]
  FROM table1;
  ```

  The new table will be created with the column-names and types as defined in the SELECT statement. You can apply new names using the AS clause.

SQL SELECT INTO Examples

Create a backup copy of Customers:

```
SELECT *
INTO CustomersBackup2013
FROM Customers;
```

Copy only a few columns into the new table:

```
SELECT CustomerName, ContactName
INTO CustomersBackup2013
FROM Customers;
```

SQL INSERT INTO SELECT Statement

- With SQL, you can copy information from one table into another.
- The INSERT INTO SELECT statement copies data from one table and inserts it into an existing table

SQL INSERT INTO SELECT Syntax
We can copy all columns from one table to another, existing table:
INSERT INTO *table2*
SELECT * FROM *table1;*
Or we can copy only the columns we want to into another, existing table:
INSERT INTO *table2*
(column_name(s))
SELECT *column_name(s)*
FROM *table1;*

SQL INSERT INTO SELECT Examples

Example

```
INSERT INTO Customers (CustomerName, Country)
SELECT SupplierName, Country FROM Suppliers;
```

Example

```
INSERT INTO Customers (CustomerName, Country)
SELECT SupplierName, Country FROM Suppliers
WHERE Country='Germany';
```