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

<table>
<thead>
<tr>
<th>Data type</th>
<th>Access</th>
<th>SQLServer</th>
<th>Oracle</th>
<th>MySQL</th>
<th>PostgreSQL</th>
<th>SQL Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>Yes/No</td>
<td>Bit</td>
<td>Byte</td>
<td>N/A</td>
<td>Boolean</td>
<td></td>
</tr>
<tr>
<td>integer</td>
<td>Number (integer)</td>
<td>Int</td>
<td>Number</td>
<td>Int</td>
<td>Int</td>
<td>Int</td>
</tr>
<tr>
<td>float</td>
<td>Number (single)</td>
<td>Float Real</td>
<td>Number</td>
<td>Float</td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>currency</td>
<td>Currency</td>
<td>Money</td>
<td>N/A</td>
<td>Money</td>
<td></td>
<td></td>
</tr>
<tr>
<td>string (fixed)</td>
<td>N/A</td>
<td>Char</td>
<td>Char</td>
<td>Char</td>
<td>Char</td>
<td>Char</td>
</tr>
<tr>
<td>string (variable)</td>
<td>Text (&lt;256) Memo (65K+)</td>
<td>Varchar</td>
<td>Varchar</td>
<td>Varchar2</td>
<td>Varchar</td>
<td>Varchar</td>
</tr>
<tr>
<td>binary object</td>
<td>OLE Object Memo</td>
<td>Binary (fixed up to 8K) Varbinary (&lt;8K) Image (&lt;2GB)</td>
<td>Long Raw</td>
<td>Blob Text</td>
<td>Binary Varbinary</td>
<td></td>
</tr>
<tr>
<td>CustomerID</td>
<td>CustomerName</td>
<td>ContactName</td>
<td>Address</td>
<td>City</td>
<td>PostalCode</td>
<td>Country</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------</td>
<td>-------------</td>
<td>----------------------</td>
<td>---------</td>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>1</td>
<td>Alfreds Futterkiste</td>
<td>Maria Anders</td>
<td>Obere Str. 57</td>
<td>Berlin</td>
<td>12209</td>
<td>Germany</td>
</tr>
<tr>
<td>2</td>
<td>Ana Trujillo Emparedados y helados</td>
<td>Ana Trujillo</td>
<td>Avda. de la Constitución 2222</td>
<td>México D.F.</td>
<td>05021</td>
<td>Mexico</td>
</tr>
<tr>
<td>3</td>
<td>Antonio Moreno Taquería</td>
<td>Antonio Moreno</td>
<td>Mataderos 2312</td>
<td>México D.F.</td>
<td>05023</td>
<td>Mexico</td>
</tr>
<tr>
<td>4</td>
<td>Around the Horn</td>
<td>Thomas Hardy</td>
<td>120 Hanover Sq.</td>
<td>London</td>
<td>WA1 1DP</td>
<td>UK</td>
</tr>
<tr>
<td>5</td>
<td>Berglunds snabbköp</td>
<td>Christina Berglund</td>
<td>Berguvsvägen 8</td>
<td>Luleå</td>
<td>S-958 22</td>
<td>Sweden</td>
</tr>
</tbody>
</table>
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 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;
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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal. <strong>Note:</strong> In some versions of SQL this operator may be written as !=</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal</td>
</tr>
<tr>
<td>BETWEEN</td>
<td>Between an inclusive range</td>
</tr>
<tr>
<td>LIKE</td>
<td>Search for a pattern</td>
</tr>
<tr>
<td>IN</td>
<td>To specify multiple possible values for a column</td>
</tr>
</tbody>
</table>
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;
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, ....);

• CREATE TABLE + CONSTRAINT Syntax
• 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.
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)
)
```
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`
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 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
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

<table>
<thead>
<tr>
<th>P_Id</th>
<th>LastName</th>
<th>FirstName</th>
<th>Address</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hansen</td>
<td>Ola</td>
<td>Timoteivn 10</td>
<td>Sandnes</td>
</tr>
<tr>
<td>2</td>
<td>Svendson</td>
<td>Tove</td>
<td>Borgvnr 23</td>
<td>Sandnes</td>
</tr>
<tr>
<td>3</td>
<td>Pettersen</td>
<td>Kari</td>
<td>Storgt 20</td>
<td>Stavanger</td>
</tr>
</tbody>
</table>

### The "Orders" table:

<table>
<thead>
<tr>
<th>O_Id</th>
<th>Order No</th>
<th>P_Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>77895</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>44678</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>22456</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>24562</td>
<td>1</td>
</tr>
</tbody>
</table>

```sql
CREATE TABLE Orders
(O_Id int NOT NULL PRIMARY KEY,
OrderNo int NOT NULL,
P_Id int FOREIGN KEY REFERENCES Persons(P_Id))
```
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'
  )
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%';
In SQL, wildcard characters are used with the SQL LIKE operator. SQL wildcards are used to search for data within a table.

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>A substitute for zero or more characters</td>
</tr>
<tr>
<td>_</td>
<td>A substitute for a single character</td>
</tr>
<tr>
<td>[charlist]</td>
<td>Sets and ranges of characters to match</td>
</tr>
<tr>
<td>[^charlist]</td>
<td>Matches only a character NOT specified within the brackets</td>
</tr>
</tbody>
</table>
• Example
• SELECT * FROM Customers
  WHERE City LIKE 'L_n_on';

• The following SQL statement selects all customers with a City starting with "b", "s", or "p":
  Example
  SELECT * FROM Customers
  WHERE City LIKE '[bsp]%';

• The following SQL statement selects all customers with a City starting with "a", "b", or "c":
  Example
  SELECT * FROM Customers
  WHERE City LIKE '[a-c]%';
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');
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;`
### 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.
  
  ```sql
  ALTER TABLE table_name
  ADD column_name datatype
  ```

- ```sql
  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;
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

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

**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
- **SMALLDURATION** - 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.

<table>
<thead>
<tr>
<th>OrderID</th>
<th>CustomerID</th>
<th>OrderDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>10308</td>
<td>2</td>
<td>1996-09-18</td>
</tr>
<tr>
<td>10309</td>
<td>37</td>
<td>1996-09-19</td>
</tr>
<tr>
<td>10310</td>
<td>77</td>
<td>1996-09-20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CustomerID</th>
<th>CustomerName</th>
<th>ContactName</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alfreds Futterkiste</td>
<td>Maria Anders</td>
<td>Germany</td>
</tr>
<tr>
<td>2</td>
<td>Ana Trujillo Emparedados y helados</td>
<td>Ana Trujillo</td>
<td>Mexico</td>
</tr>
<tr>
<td>3</td>
<td>Antonio Moreno Taquería</td>
<td>Antonio Moreno</td>
<td>Mexico</td>
</tr>
</tbody>
</table>
Example

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

<table>
<thead>
<tr>
<th>OrderID</th>
<th>CustomerName</th>
<th>OrderDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>10308</td>
<td>Ana Trujillo Emparedados y helados</td>
<td>9/18/1996</td>
</tr>
<tr>
<td>10365</td>
<td>Antonio Moreno Taquería</td>
<td>11/27/1996</td>
</tr>
<tr>
<td>10383</td>
<td>Around the Horn</td>
<td>12/16/1996</td>
</tr>
<tr>
<td>10355</td>
<td>Around the Horn</td>
<td>11/15/1996</td>
</tr>
<tr>
<td>10278</td>
<td>Berglunds snabbköp</td>
<td>8/12/1996</td>
</tr>
</tbody>
</table>
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
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;
<table>
<thead>
<tr>
<th>CustomerID</th>
<th>CustomerName</th>
<th>ContactName</th>
<th>Address</th>
<th>City</th>
<th>PostalCode</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alfreds Futterkiste</td>
<td>Maria Anders</td>
<td>Obere Str. 57</td>
<td>Berlin</td>
<td>12209</td>
<td>Germany</td>
</tr>
<tr>
<td>2</td>
<td>Ana Trujillo Emparedados y helados</td>
<td>Ana Trujillo</td>
<td>Avda. de la Constitución 2222</td>
<td>México D.F.</td>
<td>05021</td>
<td>Mexico</td>
</tr>
<tr>
<td>3</td>
<td>Antonio Moreno Taquería</td>
<td>Antonio Moreno</td>
<td>Mataderos 2312</td>
<td>México D.F.</td>
<td>05023</td>
<td>Mexico</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OrderID</th>
<th>CustomerID</th>
<th>EmployeeID</th>
<th>OrderDate</th>
<th>ShipperID</th>
</tr>
</thead>
<tbody>
<tr>
<td>10308</td>
<td>2</td>
<td>7</td>
<td>1996-09-18</td>
<td>3</td>
</tr>
<tr>
<td>10309</td>
<td>37</td>
<td>3</td>
<td>1996-09-19</td>
<td>1</td>
</tr>
<tr>
<td>10310</td>
<td>77</td>
<td>8</td>
<td>1996-09-20</td>
<td>2</td>
</tr>
</tbody>
</table>
SQL INNER JOIN Example

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

```sql
SELECT column_name(s) 
FROM table1 
LEFT JOIN table2 
ON table1.column_name = table2.column_name;
```

or:

```sql
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.
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;
```
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';;