Table Name: Persons

_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Street 25	Hyderabad
2	John	Tove	Street 10	Hyderabad
3	Pettersen	Kari	Street 32	Bangalore

Table Name: Orders

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

- Note: SQL is not case sensitive

SQL DML and DDL

DML:

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database

DDL:

- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database
- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

SQL CREATE DATABASE Statement

The CREATE DATABASE statement is used to create a database.

Syntax:

CREATE DATABASE database_name

SQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a table in a database

Syntax:

CREATE TABLE table_name

(

column_name1 data_type,

column_name2 data_type,

column_name3 data_type,

....

)

SQL INSERT INTO Statement

The INSERT INTO statement is used to insert a new row in a table

Syntax:

INSERT INTO table_name VALUES (value1, value2, value3,...)

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

SQL UPDATE Statement

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

Syntax:

UPDATE table_name SET column1=value, column2=value2,...

WHERE some_column=some_value

SQL DELETE Statement

The DELETE statement is used to delete rows in a table

DELETE FROM table_name WHERE some_column=some_value

Note: If you omit the WHERE clause, all records will be deleted

SQL ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table

Syntax: To Add a column

ALTER TABLE table_name

ADD column_name datatype

Syntax: To Delete a column

ALTER TABLE table_name

DROP COLUMN column_name

Syntax: To Change the Data Type

ALTER TABLE table_name

ALTER COLUMN column_name datatype

Example:

ALTER TABLE Persons

ADD DateOfBirth date

Drop Column

ALTER TABLE Persons

DROP COLUMN DateOfBirth

SQL SELECT Syntax

```
SELECT column_name(s)
FROM table_name
or
SELECT * FROM table_name
```

Example: SELECT LastName,FirstName FROM Persons

SQL SELECT DISTINCT Statement

In a table, some of the columns may contain duplicate values. This is not a problem, however, sometimes you will want to list only the different (distinct) values in a table

Syntax:

SELECT DISTINCT column_name(s)

FROM table_name

SQL WHERE Clause

The where clause is used to filter the records

Syntax:

SELECT column_name(s)

FROM table_name

WHERE column_name operator value

Exercise: select only the persons living in the city "Hyderabad" from the table above

Ans: SELECT * FROM Persons WHERE City='Hyderabad'

Quotes around Text Fields

SQL uses single quotes around text values (most database systems will also accept double quotes Although, numeric values should not be enclosed in quotes

Example:

SELECT * FROM Persons WHERE FirstName='Tove'

SELECT * FROM Persons WHERE Year=1965

Operators Allowed in the WHERE Clause

With the WHERE clause, the following operators can be used:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	If you know the exact value you want to return for at least one of the columns

SQL AND & OR Operators

The AND & OR operators are used to filter records based on more than one condition

Example:

select only the persons with the first name equal to "Tove" AND the last name equal to "John":

SELECT * FROM Persons WHERE FirstName='Tove' AND LastName='John'

Example for Combination of AND and OR

SELECT * FROM Persons WHERE LastName='John' AND (FirstName='Tove' OR FirstName='Ola')

SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set by a specified column

The ORDER BY keyword sort the records in ascending order by default

Syntax:

SELECT column_name(s) FROM table_name ORDER BY column_name(s) ASC|DESC

Delete All Rows

DELETE FROM table_name

or

DELETE * FROM table_name

Note: Be very careful when deleting records. You cannot undo this statement!

SQL TOP Clause

The TOP clause is used to specify the number of records to return

Syntax:

SELECT TOP number|percent column_name(s)

FROM table_name

SQL Wildcards

SQL wildcards can substitute for one or more characters when searching for data in a database.

SQL wildcards must be used with the SQL LIKE operator.

With SQL, the following wildcards can be used:

Wildcard	Description
%	A substitute for zero or more characters
_	A substitute for exactly one character
[charlist]	Any single character in charlist
[^charlist] Or [!charlist]	Any single character not in charlist

SELECT * FROM Persons WHERE FirstName LIKE '_la'

SELECT * FROM Persons WHERE LastName LIKE '[bsp]%'

SQL LIKE Operator

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

SELECT column_name(s)

FROM table_name

WHERE column_name LIKE pattern

SELECT * FROM Persons

WHERE City LIKE 's%'

SELECT * FROM Persons

WHERE City LIKE '%tav%'

SELECT * FROM Persons

WHERE City NOT LIKE '%tav%'

SQL IN Operator

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

Syntax:

SELECT column_name(s) FROM table_name WHERE column_name IN (value1,value2,...)

Example: SELECT * FROM Persons WHERE LastName IN ('Hansen','Pettersen')

SQL BETWEEN Operator

The BETWEEN operator selects a range of data between two values. The values can be numbers, text, or dates

Syntax:

SELECT column_name(s)

FROM table_name

WHERE column_name

BETWEEN value1 AND value2

SELECT * FROM Persons

WHERE LastName

BETWEEN 'Hansen' AND 'Pettersen'

SELECT * FROM Persons

WHERE LastName

NOT BETWEEN 'Hansen' AND 'Pettersen'

SQL Alias

You can give a table or a column another name by using an alias. This can be a good thing to do if you have very long or complex table names or column names.

An alias name could be anything, but usually it is short.

Syntax For Tables:

SELECT column_name(s)

FROM table_name

AS alias_name

Syntax For Columns:

SELECT column_name AS alias_name

FROM table_name

Example:

SELECT po.OrderID, p.LastName, p.FirstName

FROM Persons AS p,

Product_Orders AS po

WHERE p.LastName='Hansen' AND p.FirstName='Ola'

SQL UNION Operator

The SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

Notice that each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types. Also, the columns in each SELECT statement must be in the same order.

Syntax:

SELECT column_name(s) FROM table_name1

UNION

SELECT column_name(s) FROM table_name2

Note: The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL.

SELECT column_name(s) FROM table_name1

UNION ALL

SELECT column_name(s) FROM table_name2

PS: The column names in the result-set of a UNION are always equal to the column names in the first SELECT statement in the UNION.

SQL UNION Example

"Employees_Norway":

E_ID	E_Name
01	Hansen, Ola
02	John, Tove
03	John, Stephen

04	Pettersen, Kari

"Employees_USA":

E_ID	E_Name
01	Turner, Sally
02	Kent, Clark
03	John, Stephen
04	Scott, Stephen

Now we want to list **all the different** employees in Norway and USA.

We use the following SELECT statement:

```
SELECT E_Name FROM Employees_Norway
UNION
SELECT E_Name FROM Employees_USA
```

The result-set will look like this:

E_Name
Hansen, Ola
John, Tove
John, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
Scott, Stephen

Note: This command cannot be used to list all employees in Norway and USA. In the example above we have two employees with equal names, and only one of them will be listed. The UNION command selects only distinct values.

SQL UNION ALL Example

Now we want to list **all** employees in Norway and USA:

```
SELECT E_Name FROM Employees_Norway
UNION ALL
SELECT E_Name FROM Employees_USA
```

Result

E_Name
Hansen, Ola
John, Tove
John, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
John, Stephen
Scott, Stephen

SQL SELECT INTO Statement

The SELECT INTO statement selects data from one table and inserts it into a different table.

The SELECT INTO statement is most often used to create backup copies of tables.

Syntax:

```
SELECT *
INTO new_table_name [IN externaldatabase]
FROM old_tablename
```

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

```
SELECT column_name(s)
INTO new_table_name [IN externaldatabase]
FROM old_tablename
```

SQL SELECT INTO Example

Make a Backup Copy - Now we want to make an exact copy of the data in our "Persons" table.

We use the following SQL statement:

```
SELECT *
INTO Persons_Backup
FROM Persons
```

We can also use the IN clause to copy the table into another database:

```
SELECT *
INTO Persons_Backup IN 'Backup.mdb'
FROM Persons
```

We can also copy only a few fields into the new table:

```
SELECT LastName,FirstName
INTO Persons_Backup
FROM Persons
```

SQL SELECT INTO - With a WHERE Clause

We can also add a WHERE clause.

The following SQL statement creates a "Persons_Backup" table with only the persons who lives in the city "Hyderabad":

```
SELECT LastName,Firstname
INTO Persons_Backup
FROM Persons
WHERE City='Hyderabad'
```

SQL Constraints

Constraints are used to limit the type of data that can go into a table.

Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).

We will focus on the following constraints:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

SQL NOT NULL Constraint

By default, a table column can hold NULL values. The NOT NULL constraint enforces a column to NOT accept NULL values.

The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field

Example:

```
CREATE TABLE Persons
(
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.

CREATE TABLE Persons

(

P_Id int NOT NULL **UNIQUE**,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

Constraint on multiple columns

CREATE TABLE Persons

(

P_Id int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName)

)

SQL UNIQUE Constraint on ALTER TABLE

ALTER TABLE Persons

ADD UNIQUE (P_Id)

ALTER TABLE Persons

ADD CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName)

To DROP a UNIQUE Constraint

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.

Each table 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)

)

CREATE TABLE Persons

(

P_Id int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)

)

SQL PRIMARY KEY Constraint on ALTER TABLE

ALTER TABLE Persons

ADD PRIMARY KEY (P_Id)

ALTER TABLE Persons

ADD CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)

To DROP a PRIMARY KEY Constraint

```
ALTER TABLE Persons
DROP PRIMARY KEY
ALTER TABLE Persons
DROP CONSTRAINT pk_PersonID
```

SQL FOREIGN KEY Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

Let's illustrate the foreign key with an example. Look at the following two tables:

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Street 25	Hyderabad
2	John	Tove	Street 10	Hyderabad
3	Pettersen	Kari	Street 32	Bangalore

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Note that the "P_Id" column in the "Orders" table points to the "P_Id" column in the "Persons" table.

The "P_Id" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "P_Id" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

The FOREIGN KEY constraint also prevents that invalid data form being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

Example:

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

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

SQL FOREIGN KEY Constraint on ALTER TABLE

```
ALTER TABLE Orders
ADD FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
```

```
ALTER TABLE Orders
ADD CONSTRAINT fk_PerOrders
FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
```

To DROP a FOREIGN KEY Constraint

```
ALTER TABLE Orders
DROP CONSTRAINT fk_PerOrders
```

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

SQL CHECK Constraint on CREATE TABLE

The following SQL creates a CHECK constraint on the "P_Id" column when the "Persons" table is created. The CHECK constraint specifies that the column "P_Id" must only include integers greater than 0.

```
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 CHECK Constraint on ALTER TABLE

ALTER TABLE Persons

ADD CHECK (P_Id>0)

SQL DEFAULT Constraint

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

The default value will be added to all new records, if no other value is specified

CREATE TABLE Persons

(

P_Id int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255) **DEFAULT** 'Hyderabad'

)

The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE():

```
CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
OrderDate date DEFAULT GETDATE()
)
```

SQL AGGREGATE FUNCTIONS

SQL Aggregate functions return a single value, using values in a table column.

Sales Table:

OrderID	OrderDate	OrderPrice	OrderQuantity	CustomerName
1	12/22/2005	160	2	Smith
2	08/10/2005	190	2	Johnson
3	07/13/2005	500	5	Baldwin
4	07/15/2005	420	2	Smith
5	12/22/2005	1000	4	Wood
6	10/2/2005	820	4	Smith

The SQL COUNT Function:

Returns the number of rows in a table satisfying the criteria specified in the WHERE clause.

SELECT COUNT(*) FROM SALES

WHERE CustomerName='Smith'

How can we get the number of unique customers that have ordered from our store? We need to use the DISTINCT keyword along with the COUNT function to accomplish that:

SELECT COUNT(DISTINCT CustomerName) FROM Sales

The SQL SUM Function:

Used to select the sum of values from numeric column.

SELECT SUM(OrderPrice) FROM Sales

The SQL AVG Function:

Retrieves the average value for a numeric column.

SELECT AVG(OrderQuantity) FROM Sales

You can use AVG function with the WHERE clause, thus restricting the data you operate on

SELECT AVG(OrderQuantity) FROM Sales WHERE OrderPrice > 200

The SQL MIN Function:

Selects the smallest number from a numeric column.

SELECT MIN(OrderPrice) FROM Sales

The SQL MAX Function: Retrieves the maximum numeric value from a numeric column.

SELECT MAX(OrderPrice) FROM Sales