Database Administrator

Coordinates all the activities of the database system

has a good understanding of the enterprise’s information resources and needs.

Database administrator's duties include:

Storage structure and access method definition

Schema and physical organization modification

Granting users authority to access the database

Backing up data

Monitoring performance and responding to changes

• Database tuning

Terminologies

Field: Every table is broken up into smaller entities called fields.

- I.e.: Name, age, Address and salary

Record or row:

1. XYX 32 Delhi 2000

A horizontal entity

Null value: Appears to be blank

Constraints in SQL

- NOT NULL Constraint − Ensures that a column cannot have a NULL value.
- DEFAULT Constraint − Provides a default value for a column when none is specified.
- UNIQUE Constraint − Ensures that all the values in a column are different.
- PRIMARY Key − Uniquely identifies each row/record in a database table.
- FOREIGN Key − Uniquely identifies a row/record in any another database table.
- CHECK Constraint − The CHECK constraint ensures that all values in a column satisfy certain conditions.
- INDEX − Used to create and retrieve data from the database very quickly.

Data Integrity

- **Entity Integrity –** There are no duplicate rows in a table.
- **Domain Integrity –** Enforces valid entries for a given column by restricting the type, the format, or the range of values.
- **Referential integrity –** Rows cannot be deleted, which are used by other records.
- **User-Defined Integrity –** Enforces some specific business rules that do not fall into entity, domain or referential integrity.

Structured Query Language

Creation of Tables

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

- Syntax

CREATE TABLE *table_name* (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);

- Example

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

Drop table

- Syntax

  DROP TABLE *table_name*;

The following SQL statement drops the existing table "Shippers":

- DROP TABLE Shippers;

- SQL TRUNCATE TABLE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

- **Syntax**

TRUNCATE TABLE *table_name*;

INSERT INTO

In specific column

- Syntax

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

- Syntax

For all values

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

SELECT

- Syntax

SELECT *column1*, *column2, ...*
FROM *table_name*;

SELECT CustomerName, City FROM Customers;

- Syntax

SELECT * FROM *table_name*;

SELECT * FROM Customers;

WHERE Clause

- The WHERE clause is used to filter records.
- The WHERE clause is used to extract only those records that fulfill a specified condition.
- Syntax
- SELECT *column1*, *column2, ...*
  FROM *table_name*
  WHERE *condition*;
- Example

```
1   SELECT * FROM Customers
    WHERE Country='Mexico';
2   SELECT * FROM Customers
    WHERE CustomerID=1;
```

AND, OR and NOT Operators

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

- The AND operator displays a record if all the conditions separated by AND is TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

UPDATE

The UPDATE statement is used to modify the existing records in a table.

- SYNTAX

UPDATE *table_name*
SET *column1* = *value1*, *column2* = *value2*, ...
WHERE *condition*;

EXAMPLE :

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!

TRIM

The TRIM() function removes leading and trailing spaces from a string.

- Syntax

TRIM(*string*)

DISTINCT

The SELECT DISTINCT statement is used to return only distinct (different) values.

- Syntax

SELECT DISTINCT *column1*, *column2, ...*
FROM *table_name*;

Example:

ORDER BY

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in **ascending** order by default.

- Syntax

SELECT *column1*, *column2, ...*
FROM *table_name*
ORDER BY *column1, column2, ...* ASC|DESC;

- Example 1

SELECT * FROM Customers
ORDER BY Country;

- Example 2

SELECT * FROM Customers
ORDER BY Country DESC;

DELETE

The DELETE statement is used to delete existing records in a table.

- Syntax

DELETE FROM *table_name*
WHERE *condition*;

NOTE:

- The WHERE clause specifies which record(s) that should be deleted.
- If you omit the WHERE clause, all records in the table will be deleted!

NULL Values

A field with a NULL value is a field with no value.

- IS NULL Syntax

SELECT *column_names*
FROM *table_name*
WHERE *column_name* IS NULL;

- IS NOT NULL Syntax

SELECT *column_names*
FROM *table_name*
WHERE *column_name* IS NOT NULL;

SQL TOP, LIMIT or ROWNUM Clause

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

SYNTAX:

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

Example 1:

```
SELECT TOP 3 * FROM Customers;
```

Example 2:

```
SELECT * FROM Customers
LIMIT 3;
```

Example 3:

```
SELECT TOP 50 PERCENT * FROM Customers;
```

Example 4:

```
SELECT TOP 3 * FROM Customers
WHERE Country='Germany';
```

The SQL LIKE Operator

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

There are two wildcards used in conjunction with the LIKE operator:

- % - The percent sign represents zero, one, or multiple characters
- _ - The underscore represents a single character

LIKE Syntax

- SELECT *column1, column2, ...*
  FROM *table_name*
  WHERE *columnN* LIKE *pattern*;

Contd.

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends 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 starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.
- The IN operator is a shorthand for multiple OR conditions.

Syntax

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

SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* IN (*SELECT STATEMENT*);

Examples:

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK’);
```

```
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK’);
```

```
SELECT * FROM Customers
WHERE Country IN (SELECT Country FROM Suppliers);
```

Order by

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

Syntax

- SELECT *column1*, *column2, ...*
  FROM *table_name*
  ORDER BY *column1, column2, ...* ASC|DESC;

Example:

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

BETWEEN Operator

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.

- 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;

SQL Aliases

- SQL aliases are used to give a table, or a column in a table, a temporary name.
- Aliases are often used to make column names more readable.
- An alias only exists for the duration of the query.
- Syntax

SELECT *column_name* AS *alias_name*
FROM *table_name;*

Example:

```
SELECT CustomerName, Address + ', ' + PostalCode + ' ' +
City + ', ' + Country AS Address
FROM Customers;
```

Advantages:

- There are more than one table involved in a query
- Functions are used in the query
- Column names are big or not very readable
- Two or more columns are combined together

HAVING Clause

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

SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*
GROUP BY *column_name(s)*
HAVING *condition*
ORDER BY *column_name(s);*

- **Example**

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

ALTER TABLE Statement

- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
- The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ADD Column

ALTER TABLE *table_name*
ADD *column_name datatype*;

DROP COLUMN

ALTER TABLE *table_name*
DROP COLUMN *column_name*;

ALTER/MODIFY COLUMN

ALTER TABLE *table_name*
ALTER COLUMN *column_name datatype*;

Primary key

- The PRIMARY KEY constraint uniquely identifies each record in a database table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only one primary key, which may consist of single or multiple fields.

EXAMPLE

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

Contd…

Example

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

Example:

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

Contd…

Example

```
ALTER TABLE Persons
ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastNam
e);
```

DROP a PRIMARY KEY Constraint

Example

```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

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.

Example

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

Example

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes'
)
);
```

AUTO INCREMENT Field

- Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.
- Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

Example

```
CREATE TABLE Persons (
    ID int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

Example

```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

ANY and ALL Operators

- The ANY and ALL operators are used with a WHERE or HAVING clause.
- The ANY operator returns true if any of the subquery values meet the condition.
- The ALL operator returns true if all of the subquery values meet the condition.

ANY

```
SELECT column_name(s)
FROM table_name
WHERE column_name
operator ANY
(SELECT column_name FRO
M table_name WHERE cond
ition);
```

ALL

```
SELECT column_name(s)
FROM table_name
WHERE column_name
operator ALL
(SELECT column_name FRO
M table_name WHERE cond
ition);
```

CREATE VIEW

- a view is a virtual table based on the result-set of an SQL statement.

- Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

FOREIGN KEY Constraint

- A FOREIGN KEY is a key used to link two tables together.
- A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
- The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Example

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```