Data Definition Statements (DDL)

Notes by Mannan Ul Haq (22L-7556)

Create and Drop Statements:

Create Database:

In SQL Server, `CREATE DATABASE` is like building a new house. You give it a name, like "testDB," and it creates a place where you can store your data.

Syntax:

```
CREATE DATABASE database_name;
```

Explanation:

- `CREATE DATABASE`: This tells SQL Server that you want to make a new database.
- `database_name`: Here, you give your database a name, like "testDB" or anything you want to call it.

Example:

```
CREATE DATABASE UniversityDB;
```

Drop Database:

In SQL Server, `DROP DATABASE` is like tearing down a building. You specify the name of the database you want to remove, and SQL Server deletes it entirely.

Syntax:

```
DROP DATABASE database_name;
```

Explanation:

- `DROP DATABASE`: This tells SQL Server that you want to delete a database.
- `database_name`: Here, you specify the name of the database you want to remove.

Example:

```
DROP DATABASE UniversityDB;
```

Create Table:

In SQL Server, `CREATE TABLE` is like setting up a table for a spreadsheet. You define the structure of your table, including the names and types of each column.

Syntax:

```
CREATE TABLE table_name (
    column1 datatype1,
    column2 datatype2,
    ...
);
```

Explanation:

- `CREATE TABLE`: This tells SQL that you want to make a new table.
- `table_name`: Here, you give your table a name, like "Students".
- `(column1 datatype1, column2 datatype2, ...)`: Inside the parentheses, you list the columns of your table along with their data types. Each column has a name (like "Name" or "Age") and a data type (like "VARCHAR" for text or "INT" for numbers).

Example:

```
CREATE TABLE Students (
    StudentID INT,
    Name VARCHAR(50),
    Age INT,
    GPA FLOAT
);
```

Below is a table listing some important data types commonly used in SQL databases:

Data Type	Description	Example
INT	Integer (whole number)	123, -45, 0
VARCHAR(n)	Variable-length string with maximum length of n	'Hello', '12345', 'SQL'
CHAR(n)	Fixed-length string with exactly length of n	'John ', 'ABCD'
FLOAT	Floating-point number	3.14, -0.001, 123.456
DATE	Date (year, month, day)	'2024-01-23', '1999-12-31'
TIME	Time (hour, minute, second, fraction)	'12:30:45.678', '09:00:00'
BOOLEAN	Boolean value (true/false)	TRUE, FALSE

Drop Table:

In SQL Server, `DROP TABLE` is like removing a table from your database. It completely deletes the table and all its data.

Syntax:

```
DROP TABLE table_name;
```

Explanation:

- `DROP TABLE`: This tells SQL that you want to delete a table.
- `table_name`: Here, you specify the name of the table you want to remove.

Example:

```
DROP TABLE Students;
```

Truncate Table:

In SQL Server, `TRUNCATE TABLE` is like clearing a table without deleting its structure. It removes all rows from the table, but the table itself still exists.

Syntax:

```
TRUNCATE TABLE table_name;
```

Explanation:

- `TRUNCATE TABLE`: This tells SQL to remove all rows from a table.
- `table_name`: Here, you specify the name of the table you want to clear.

Example:

```
TRUNCATE TABLE Students;
```

Alter Table Statements:

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

Add Column:

In SQL Server, `ALTER TABLE ADD COLUMN` is like adding a new column to an existing table. It allows you to expand the structure of your table by adding new fields.

Syntax:

```
ALTER TABLE table_name
ADD column_name datatype;
```

Explanation:

- `ALTER TABLE`: This tells SQL that you want to change the structure of a table.

- `table_name`: Here, you specify the name of the table you want to modify.
- `ADD COLUMN`: This specifies that you want to add a new column to the table.
- `column_name`: Here, you specify the name of the new column.
- `datatype`: This specifies the data type of the new column.

Example:

```
ALTER TABLE Students
ADD Email VARCHAR(100);
```

Drop Column:

In SQL Server, `ALTER TABLE DROP COLUMN` is like removing a column from an existing table. It allows you to modify the structure of your table by removing unwanted fields.

Syntax:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Explanation:

- `ALTER TABLE`: This tells SQL that you want to change the structure of a table.
- `table_name`: Here, you specify the name of the table you want to modify.
- `DROP COLUMN`: This specifies that you want to remove a column from the table.
- `column_name`: Here, you specify the name of the column you want to remove.

Example:

```
ALTER TABLE Students
DROP COLUMN Email;
```

Rename Column:

In SQL Server, we use `sp_rename` to rename a column in an existing table. Here's how you do it:

Syntax:

```
EXEC sp_rename 'table_name.old_column_name', 'new_column_name', 'COLUMN';
```

Explanation:

- `EXEC sp_rename`: Executes the stored procedure `sp_rename`, which is used to rename database objects.
- `'table_name.old_column_name'`: Specifies the current name of the column you want to rename. This should be in the format 'table_name.column_name'.

- `'new_column_name'`: Specifies the new name you want to assign to the column.
- `'COLUMN'`: Indicates that you're renaming a column.

Example:

```
EXEC sp_rename 'Students.Student_Name', 'Full_Name', 'COLUMN';
```

Modify Column Datatype:

In SQL Server, you use `ALTER TABLE ALTER COLUMN` to change the data type of a column in an existing table. Here's how you do it:

Syntax:

```
ALTER TABLE table_name
ALTER COLUMN column_name new_datatype;
```

Explanation:

- `ALTER TABLE`: Indicates that you're modifying the structure of a table.
- `table_name`: Specifies the name of the table you want to modify.
- `ALTER COLUMN`: Specifies that you're altering the data type of a column.
- `column_name`: Specifies the name of the column whose data type you want to change.
- `new_datatype`: Specifies the new data type you want to assign to the column.

Example:

```
ALTER TABLE Students
ALTER COLUMN Age INT;
```

Constraints Statements:

SQL constraints are used to specify rules for the data in a table.

The following constraints are commonly used in SQL Server:

Not Null:

On Create Table:

- When creating a new table, you can use the `NOT NULL` constraint to specify that a column must have a value and cannot be left empty.
- **Syntax:**

```
CREATE TABLE TableName (
    Column1 DataType NOT NULL,
    Column2 DataType NOT NULL,
    ...
);
```

- **Example:**

```
CREATE TABLE Students (
    StudentID INT NOT NULL,
    Name VARCHAR(50) NOT NULL,
    Age INT
);
```

On Alter Table:

- You can also add the `NOT NULL` constraint to an existing attribute using the `ALTER TABLE` statement.
- **Syntax:**

```
ALTER TABLE TableName
ALTER COLUMN ColumnName DataType NOT NULL;
```

- **Example:**

```
ALTER TABLE Students
ALTER COLUMN Age INT NOT NULL;
```

Unique:

On Create Table:

- When creating a new table, the `UNIQUE` constraint ensures that all values in a column are different.
- **Syntax:**

```
CREATE TABLE TableName (
    Column1 DataType UNIQUE,
    Column2 DataType,
    ...
);
```

- **Example:**

```
CREATE TABLE Employees (
    EmployeeID INT NOT NULL UNIQUE,
```

```
    Name VARCHAR(50),
    Email VARCHAR(100) UNIQUE
);
```

For Multiple Columns:

- You can apply the `UNIQUE` constraint to multiple columns **at one time** to ensure that combinations of values across those columns are unique.
- **Syntax:**

```
CREATE TABLE TableName (
    Column1 DataType,
    Column2 DataType,
    ...
    CONSTRAINT ConstraintName UNIQUE (Column1, Column2, ...)
);
```

- **Example:**

```
CREATE TABLE Orders (
    OrderID INT,
    ProductID INT,
    CONSTRAINT UniqueOrderProduct UNIQUE (OrderID, ProductID)
);
```

On Alter Table:

- You can also add the `UNIQUE` constraint to an existing table using the `ALTER TABLE` statement.
- **Syntax:**

```
ALTER TABLE TableName
ADD CONSTRAINT ConstraintName UNIQUE (ColumnName);
```

- **Example:**

```
ALTER TABLE Employees
ADD CONSTRAINT UniqueEmail UNIQUE (Email);
```

Drop a UNIQUE Constraint:

- If you need to remove a `UNIQUE` constraint from a column or combination of columns, you can use the `DROP CONSTRAINT` clause.
- **Syntax:**

```
ALTER TABLE TableName
DROP CONSTRAINT ConstraintName;
```

- **Example:**

```
ALTER TABLE Orders
DROP CONSTRAINT UniqueOrderProduct;
```

Primary Key:

On Create Table:

- When creating a new table, the `PRIMARY KEY` constraint is used to uniquely identify each row in the table.
- **Syntax:**

```
CREATE TABLE TableName (
    Column1 DataType PRIMARY KEY,
    Column2 DataType,
    ...
);
```

- **Example:**

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(50),
    Age INT
);
```

For Multiple Columns:

- You can apply the `PRIMARY KEY` constraint to multiple columns to create a composite primary key, which uniquely identifies each row based on the combination of values in those columns.
- **Syntax:**

```
CREATE TABLE TableName (
    Column1 DataType,
    Column2 DataType,
    ...
    CONSTRAINT ConstraintName PRIMARY KEY (Column1, Column2, ...)
);
```

- **Example:**

```
CREATE TABLE Orders (
    OrderID INT,
    ProductID INT,
    CONSTRAINT PK_Orders PRIMARY KEY (OrderID, ProductID)
);
```

On Alter Table:

- You can also add the `PRIMARY KEY` constraint to an existing table using the `ALTER TABLE` statement.
- **Syntax:**

```
ALTER TABLE TableName
ADD CONSTRAINT ConstraintName PRIMARY KEY (ColumnName);
```

- **Example:**

```
ALTER TABLE Students
ADD CONSTRAINT PK_Students PRIMARY KEY (StudentID);
```

Drop a PRIMARY KEY Constraint:

- To remove a `PRIMARY KEY` constraint from a table, you can use the `ALTER TABLE` statement with the `DROP CONSTRAINT` clause.
- **Syntax:**

```
ALTER TABLE TableName
DROP CONSTRAINT ConstraintName;
```

- **Example:**

```
ALTER TABLE Orders
DROP CONSTRAINT PK_Orders;
```

Foreign Key:

On Create Table:

- When creating a new table, the `FOREIGN KEY` constraint establishes a link between two tables by enforcing referential integrity.
- **Syntax:**

```
CREATE TABLE TableName (
    Column1 DataType,
    Column2 DataType,
```

```
    ...
    column_name datatype REFERENCES ReferencedTable(ReferencedColumn)
);
```

- **Example:**

```
CREATE TABLE OrderDetails (
    OrderID INT,
    ProductID INT,
    OrderNumber int NOT NULL,
    PersonID int REFERENCES Persons(PersonID)
);
```

For Multiple Columns:

- You can apply the `FOREIGN KEY` constraint to multiple columns to create a composite foreign key, which establishes a link between two tables based on the combination of values in those columns.
- **Syntax:**

```
CREATE TABLE TableName (
    Column1 DataType,
    Column2 DataType,
    ...
    CONSTRAINT ConstraintName FOREIGN KEY (Column1, Column2, ...) REFERENC
ES ReferencedTable(ReferencedColumn1, ReferencedColumn2, ...)
);
```

- **Example:**

```
CREATE TABLE OrderDetails (
    OrderID INT,
    ProductID INT,
    CONSTRAINT OrderDetails_FK FOREIGN KEY (OrderID, ProductID) REFERENCES
Orders(OrderID, ProductID)
);
```

On Alter Table:

- You can also add the `FOREIGN KEY` to an existing table using the `ALTER TABLE` statement.
- **Syntax:**

```
ALTER TABLE TableName
ADD CONSTRAINT ConstraintName FOREIGN KEY (ColumnName) REFERENCES Referenc
edTable(ReferencedColumn);
```

- **Example:**

```
ALTER TABLE OrderDetails
ADD CONSTRAINT OrderDetails_FK FOREIGN KEY (ProductID) REFERENCES Products
(ProductID);
```

Check:

On Create Table:

- When creating a new table, the `CHECK` constraint ensures that values in a column satisfy a specific condition.
- **Syntax:**

```
CREATE TABLE TableName (
    Column1 DataType CHECK (Condition),
    Column2 DataType,
    ...
);
```

- **Example:**

```
CREATE TABLE Employees (
    EmployeeID INT,
    Age INT CHECK (Age >= 18),
    Department VARCHAR(50)
);
```

For Multiple Columns:

- You can apply the `CHECK` constraint to multiple columns by combining conditions for each column.
- Syntax:

```
CREATE TABLE TableName (
    Column1 DataType,
    Column2 DataType,
    ...
    CONSTRAINT ConstraintName CHECK (Condition1, Condition2, ...)
);
```

- Example:

```
CREATE TABLE Employees (
    EmployeeID INT,
    Age INT,
```

```
    Salary FLOAT,
    CONSTRAINT CHK_Employee CHECK (Age >= 18 AND Salary >= 0)
);
```

On Alter Table:

- You can also add the `CHECK` constraint to an existing table using the `ALTER TABLE` statement.
- **Syntax:**

```
ALTER TABLE TableName
ADD CONSTRAINT ConstraintName CHECK (Condition);
```

- **Example:**

```
ALTER TABLE Employees
ADD CONSTRAINT CHK_EmployeeSalary CHECK (Salary >= 0);
```

Drop a CHECK Constraint:

- To remove a `CHECK` constraint from a table, you can use the `ALTER TABLE` statement with the `DROP CONSTRAINT` clause.
- **Syntax:**

```
ALTER TABLE TableName
DROP CONSTRAINT ConstraintName;
```

- **Example:**

```
ALTER TABLE Employees
DROP CONSTRAINT CHK_Employee;
```

Default:

On Create Table:

- When creating a new table, the `DEFAULT` constraint sets a default value for a column if no value is specified during insertion.
- **Syntax:**

```
CREATE TABLE TableName (
    Column1 DataType DEFAULT DefaultValue,
    Column2 DataType DEFAULT DefaultValue,
    ...
);
```

- **Example:**

```
CREATE TABLE Employees (
    EmployeeID INT,
    Name VARCHAR(50),
    Department VARCHAR(50) DEFAULT 'Unknown'
);
```

On Alter Table:

- You can also add the `DEFAULT` constraint to an existing table using the `ALTER TABLE` statement.
- **Syntax:**

```
ALTER TABLE TableName
ADD CONSTRAINT ConstraintName DEFAULT DefaultValue FOR ColumnName;
```

- **Example:**

```
ALTER TABLE Employees
ADD CONSTRAINT DF_Employees_Department DEFAULT 'Unknown' FOR Department;
```

Drop a DEFAULT Constraint:

- To remove a `DEFAULT` constraint from a column, you can use the `ALTER TABLE` statement with the `DROP CONSTRAINT` clause.
- **Syntax:**

```
ALTER TABLE TableName
DROP CONSTRAINT ConstraintName;
```

- **Example:**

```
ALTER TABLE Employees
DROP CONSTRAINT DF_Employees_Department;
```

Add Auto Increment Field:

In SQL Server, the equivalent of an auto-increment field is known as an **"identity"** column. Here's a simple explanation of how to create an identity column:

On Create Table:

- When creating a new table, you can define a column as an identity column to automatically generate sequential numeric values for each new row inserted into the table.
- **Syntax:**

```
CREATE TABLE TableName (
    ColumnName DataType IDENTITY(SeedValue, IncrementValue),
    ...
);
```

- **Example:**

```
CREATE TABLE Students (
    StudentID INT IDENTITY(1,1) PRIMARY KEY,
    Name VARCHAR(50),
    Age INT
);
```

On Alter Table (Add Identity Column):

- You can also add an identity column to an existing table using the `ALTER TABLE` statement.
- **Syntax:**

```
ALTER TABLE TableName
ADD ColumnName DataType IDENTITY(SeedValue, IncrementValue);
```

- **Example:**

```
ALTER TABLE Students
ADD StudentID INT IDENTITY(1,1) PRIMARY KEY;
```

On Alter Table (Modify Existing Column to Identity):

- You cannot directly modify an existing column to become an identity column. Instead, you may need to create a new identity column, copy data, and drop the old column if necessary.