**Data Definition Language (DDL)** is used to create and modify the structure of objects in a database using predefined commands and a specific syntax.

DDL commands are a set of SQL statements used to define, modify, and manage the structure of database objects. These commands allow you to create, alter, and drop database objects such as tables, indexes, views, schemas, and more. DDL commands are essential for defining the logical structure of a database and managing its organization. Here are some common DDL commands:

* CREATE: Used to create new database objects like tables, indexes, views, and schemas.
* ALTER: Used to modify the structure of existing database objects, such as adding, modifying, or dropping columns in a table, or renaming existing objects.
* DROP: Used to delete existing database objects, such as tables, indexes, views, or schemas.
* TRUNCATE: Used to remove all records from a table, but keep the table structure intact. Unlike DELETE, TRUNCATE does not log individual row deletions, making it faster and less resource-intensive for large tables.
* RENAME: Used to rename existing database objects, such as tables, columns, or constraints.
* COMMENT: Used to add comments to database objects, providing descriptive information about the object's purpose or usage.
* GRANT: Used to grant specific privileges or permissions on database objects to users or roles.
* REVOKE: Used to revoke previously granted privileges or permissions from users or roles.
* CREATE INDEX: Used to create indexes on columns in tables, which improve query performance by allowing faster data retrieval.
* DROP INDEX: Used to remove indexes from tables.

DDL commands are crucial for database administrators and developers to manage the structure and access control of a database efficiently. It's important to use them carefully, as they directly affect the database schema and its integrity.

**CREATE TABLE**

Typical syntax for the CREATE TABLE statement:

CREATE TABLE table\_name (

column1 datatype [constraint],

column2 datatype [constraint],

...

[table\_constraint]

);

Here's what each part of the syntax represents:

* CREATE TABLE: This is the SQL keyword used to create a new table.
* table\_name: The name of the table you want to create. This should be unique within the database.
* column1, column2, ...: The list of columns in the table, each specified with a name and a datatype. You can define multiple columns, separated by commas.
* datatype: The data type of each column, such as VARCHAR, INT, DECIMAL, DATE, etc. This specifies the kind of data that can be stored in each column.
* [constraint]: Optional constraints that can be applied to each column to enforce rules or restrictions. Constraints include PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, DEFAULT, CHECK, etc.
* [table\_constraint]: Optional constraints that apply to the entire table, such as PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, etc.

**About constraints:**

In SQL Server, constraints can be applied at different levels: attribute level (column level), table level, and even database level. Each level of constraint offers different advantages and considerations. Here's an overview of each:

1. Attribute Level Constraint (Column Level Constraint):

Definition: Attribute level constraints are applied to individual columns within a table.

Advantages:

* Constraints are directly associated with specific columns, providing clarity and granularity.
* Allows for fine-grained control over each column's data integrity rules.

Syntax: Attribute level constraints are specified within the column definition when creating or altering a table. Here's an example of defining a NOT NULL constraint at the attribute level:

CREATE TABLE TableName (

Column1 INT NOT NULL,

Column2 VARCHAR(50) UNIQUE,

Column3 DECIMAL(10, 2) CHECK (Column3 >= 0) );

2. Table Level Constraint:

Definition: Table level constraints are applied to the entire table, involving one or more columns.

Advantages:

* Constraints are defined separately from individual columns, allowing for easier management of multiple constraints.
* Useful when a constraint involves multiple columns or when the constraint applies to the table as a whole.

Syntax: Table level constraints are defined after specifying all the columns within the table definition or using the ALTER TABLE statement. Here's an example of defining a PRIMARY KEY constraint at the table level:

CREATE TABLE TableName (

Column1 INT,

Column2 VARCHAR(50),

Column3 DECIMAL(10, 2),

CONSTRAINT PK\_TableName PRIMARY KEY (Column1, Column2) );

3. Database Level Constraint:

Definition: Database level constraints apply to multiple tables or the entire database.

Advantages:

* Constraints are defined at the database level, making them applicable to all tables within the database.
* Useful for enforcing global data integrity rules or constraints that span multiple tables.

Syntax: Database level constraints are typically defined using DDL (Data Definition Language) statements that apply to the entire database. For example, CHECK constraints can be applied to multiple tables using the CREATE TABLE or ALTER TABLE statements.

Each level of constraint offers flexibility and control over data integrity within the database schema. The choice of constraint level depends on the specific requirements of your application and the complexity of the data integrity rules you need to enforce.

Example:

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100) NOT NULL,

Location VARCHAR(100)

);

In this "Departments" table:

We have columns for DepartmentID, DepartmentName, and Location.

DepartmentID is defined as the primary key column.

DepartmentName is specified as VARCHAR with a maximum length of 100 characters and is marked as NOT NULL to ensure that every department has a name.

Location is specified as VARCHAR with a maximum length of 100 characters.

Example:

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT,

Salary DECIMAL(10, 2),

HireDate DATE,

CONSTRAINT FK\_Department FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

);

In this example:

We create a table named "Employees" with columns for EmployeeID, FirstName, LastName, DepartmentID, Salary, and HireDate.

EmployeeID is defined as the primary key column.

DepartmentID is a foreign key column referencing the DepartmentID column in the "Departments" table.

We specify data types for each column (INT, VARCHAR, DECIMAL, DATE).

Optional constraints like PRIMARY KEY and FOREIGN KEY are applied to enforce data integrity.

Here, FK\_Department is a constraint name used for the foreign key constraint on the DepartmentID column in the Employees table. This constraint defines a foreign key relationship between the DepartmentID column in the Employees table and the DepartmentID column in the Departments table. It ensures that every value in the DepartmentID column of the Employees table must exist in the DepartmentID column of the Departments table. The name FK\_Department is simply a label for this foreign key constraint. It can be any valid identifier name and is used to identify the constraint within the database. Naming conventions for constraints can vary, but it's common to use a prefix like FK\_ to indicate a foreign key constraint followed by a descriptive name that reflects the relationship being enforced.

Some exercises:

1: Create a table named Employees with the following columns:

EmployeeID as an integer primary key.

FirstName and LastName as varchar with a maximum length of 50 characters, both should not allow NULL values.

DepartmentID as an integer, allowing NULL values.

Salary as a decimal number with precision 10 and scale 2, ensuring that it's non-negative.

Also, ensure that the DepartmentID column references the DepartmentID column in the Departments table.

2: Create a table named Products with the following columns:

ProductID as an integer primary key.

ProductName as a varchar with a maximum length of 100 characters.

UnitPrice as a decimal number with precision 10 and scale 2, ensuring that it's non-negative.

UnitsInStock as an integer, ensuring that it's non-negative.

3: Create a table named OrderDetails with the following columns:

OrderID and ProductID together form a composite primary key (integer).

Quantity as an integer.

Ensure that OrderID references the OrderID column in the Orders table, and ProductID references the ProductID column in the Products table.

4: Create a table named Customers with the following columns:

CustomerID as an integer primary key.

FirstName and LastName as varchar with a maximum length of 50 characters, both should not allow NULL values.

Email as varchar with a maximum length of 100 characters, ensuring that it's unique.

Gender as a character with length 1, with a default value of 'U' (for unknown), and allowing only 'M' (for male) or 'F' (for female) as valid values.

A1: Creating a table with attribute-level constraints

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

DepartmentID INT,

Salary DECIMAL(10, 2) CHECK (Salary >= 0),

CONSTRAINT FK\_Department FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID) );

A2: Creating a table with table-level constraints:

CREATE TABLE Products (

ProductID INT,

ProductName VARCHAR(100),

UnitPrice DECIMAL(10, 2),

UnitsInStock INT,

CONSTRAINT PK\_Products PRIMARY KEY (ProductID),

CONSTRAINT CHK\_UnitPrice CHECK (UnitPrice >= 0),

CONSTRAINT CHK\_UnitsInStock CHECK (UnitsInStock >= 0) );

A3: Creating a table with a composite primary key and foreign key constraint:

CREATE TABLE OrderDetails (

OrderID INT,

ProductID INT,

Quantity INT,

CONSTRAINT PK\_OrderDetails PRIMARY KEY (OrderID, ProductID),

CONSTRAINT FK\_OrderDetails\_Order FOREIGN KEY (OrderID) REFERENCES Orders(OrderID), CONSTRAINT FK\_OrderDetails\_Product FOREIGN KEY (ProductID) REFERENCES Products(ProductID) );

A4: Creating a table with default values and unique constraint:

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) UNIQUE,

Gender CHAR(1) DEFAULT 'U' CHECK (Gender IN ('M', 'F', 'U')) );

**DROP**

The DROP command is used to delete existing database objects such as tables, indexes, views, procedures, functions, and triggers. Here's the basic syntax for the DROP command:

DROP OBJECT\_TYPE [IF EXISTS] object\_name;

Where:

* OBJECT\_TYPE is the type of object to be dropped (e.g., TABLE, INDEX, VIEW, PROCEDURE, FUNCTION, TRIGGER).
* IF EXISTS is an optional clause that prevents an error from occurring if the object doesn't exist. This clause is available in SQL Server 2016 and later versions.
* object\_name is the name of the object to be dropped.

Example to Drop a Table:

DROP TABLE IF EXISTS TableName;

Some exercises:  
1: Write a SQL command to drop the table named "Employees".

DROP TABLE Employees;

2: What SQL statement would you use to drop the table "Products" if it exists?

DROP TABLE IF EXISTS Products;

**ALTER**

The ALTER command is used to modify existing database objects, such as tables, views, procedures, functions, and indexes. Here's the basic syntax for the ALTER command:

ALTER OBJECT\_TYPE object\_name {

ADD | ALTER | DROP | MODIFY

<object\_definition>

};

Where:

* OBJECT\_TYPE is the type of object to be altered (e.g., TABLE, INDEX, VIEW, PROCEDURE, FUNCTION).
* object\_name is the name of the object to be altered.
* ADD, ALTER, DROP, or MODIFY are the keywords used to specify the action to be performed.
* <object\_definition> represents the new definition of the object being altered.

Alter Table - Add Column:

ALTER TABLE TableName

ADD ColumnName datatype;

Alter Table - Alter Column:

ALTER TABLE TableName

ALTER COLUMN ColumnName new\_datatype;

Alter Table - Drop Column:

ALTER TABLE TableName

DROP COLUMN ColumnName;

Some exercises:

1: How would you add a new column named "Email" to the table "Customers" with a maximum length of 100 characters?

ALTER TABLE Customers

ADD Email VARCHAR(100);

2: Suppose you need to modify the data type of the column "PhoneNumber" in the table "Contacts" to support longer numbers. How would you accomplish this?

ALTER TABLE Contacts

ALTER COLUMN PhoneNumber VARCHAR(20);

**TRUNCATE**

The TRUNCATE TABLE statement is used to quickly delete all rows from a table. Unlike the DELETE statement, which removes rows one by one and logs each deletion, TRUNCATE TABLE removes all rows from a table in a single operation and does not log individual row deletions. This makes TRUNCATE TABLE much faster and less resource-intensive, especially for large tables.

Syntax:

TRUNCATE TABLE table\_name;

Where table\_name is the name of the table from which you want to delete all rows.

It's important to note the following considerations when using TRUNCATE TABLE:

* TRUNCATE TABLE removes all rows from the table, but the table structure, indexes, constraints, and statistics remain unchanged.
* TRUNCATE TABLE is a DDL (Data Definition Language) statement and typically requires fewer locks and resources compared to the DELETE statement.
* TRUNCATE TABLE is not allowed on tables that are referenced by a foreign key constraint from another table, unless the foreign key constraint is disabled or removed.
* TRUNCATE TABLE cannot be rolled back. Once it is executed, the operation is irreversible.

Example:

TRUNCATE TABLE MyTable;

This statement deletes all rows from the "MyTable" table, resetting any identity columns to their seed value, but keeps the table structure intact. Use TRUNCATE TABLE when you need to quickly remove all data from a table without logging individual row deletions.

Some exercises:

1: How would you delete all rows from the table "Employees" using the TRUNCATE TABLE command?

TRUNCATE TABLE Employees;

2: What happens to identity columns when you use the TRUNCATE TABLE command?

TRUNCATE TABLE resets any identity columns in the table to their seed value. Identity values are reseeded to the original starting value specified for the column.

3: Can you use the TRUNCATE TABLE command to delete specific rows based on a condition?

No, TRUNCATE TABLE deletes all rows from a table. It does not support filtering based on conditions like the DELETE statement.

4: How does TRUNCATE TABLE differ from the DELETE statement?

TRUNCATE TABLE is a faster operation than DELETE as it removes all rows from a table in a single operation, without logging each deletion. However, TRUNCATE TABLE cannot be rolled back, and it doesn't activate triggers.

**SQL Server data types**

Data types define the type of data that can be stored in each column of a table. SQL Server provides a wide range of data types to accommodate different types of data and storage requirements. Here are some commonly used data types in SQL Server:

Numeric Data Types:

* int: Integer data type that stores whole numbers in the range of -2^31 (-2,147,483,648) to 2^31-1 (2,147,483,647).
* bigint: Large integer data type that stores whole numbers in the range of -2^63 (-9,223,372,036,854,775,808) to 2^63-1 (9,223,372,036,854,775,807).
* smallint: Small integer data type that stores whole numbers in the range of -32,768 to 32,767.
* tinyint: Tiny integer data type that stores whole numbers in the range of 0 to 255.
* decimal(p, s): Fixed precision and scale numeric data type that stores exact numeric values with a specified precision p (total number of digits) and scale s (number of digits to the right of the decimal point).

Character String Data Types:

* char(n): Fixed-length character string data type that stores up to n characters.
* varchar(n): Variable-length character string data type that stores up to n characters.
* nvarchar(n): Variable-length Unicode character string data type that stores up to n characters.

n for SQL Server 2019 (all editions):

CHAR(n): 8,000 bytes (1 character = 1 byte)

VARCHAR(n): 8,000 bytes

NVARCHAR(n): 4,000 characters (8,000 bytes)

* text: Variable-length character string data type for storing large text data (deprecated, use varchar(max) instead).
* nvarchar(max): Variable-length Unicode character string data type for storing large text data.

Binary Data Types:

* binary(n): Fixed-length binary data type that stores up to n bytes.
* varbinary(n): Variable-length binary data type that stores up to n bytes.

The range of n for the binary(n) data type is from 1 to 8,000. For example:

binary(1) can store a maximum of 1 byte of binary data.

* image: Variable-length binary data type for storing large binary data (deprecated, use varbinary(max) instead).
* varbinary(max): Variable-length binary data type for storing large binary data.

Date and Time Data Types:

* datetime: Date and time data type that stores date and time values from January 1, 1753, to December 31, 9999, with an accuracy of 0.0033 seconds (equivalent to 3.33 milliseconds).
* datetime2: Date and time data type that stores date and time values from January 1, 0001, to December 31, 9999, with an accuracy of 100 nanoseconds.
* date: Date data type that stores date values only.
* time: Time data type that stores time values only.
* smalldatetime: Date and time data type that stores date and time values from January 1, 1900, to June 6, 2079, with an accuracy of 1 minute.

**Other Data Types:**

* bit: Boolean data type that stores either 0 or 1.
* uniqueidentifier: Unique identifier data type that stores a globally unique identifier (GUID).
* xml: XML data type for storing XML documents.
* sql\_variant: Variable data type that can store values of various SQL Server data types.

In addition to the basic data types, SQL Server provides several advanced data types to handle specific types of data or scenarios. These advanced data types offer more functionality or storage capabilities beyond the standard data types. Here are some of the advanced data types in SQL Server:

Geometry and Geography:

* geometry: Represents data in a Euclidean (flat) coordinate system.
* geography: Represents data in a round-earth coordinate system (ellipsoidal).

These data types are used to store spatial data, such as points, lines, and polygons, and support operations like spatial analysis and calculations.

HierarchyID:

hierarchyid: Represents hierarchical data in a table. It stores a position in a hierarchy tree.

This data type is useful for managing hierarchical data structures like organizational charts, folder structures, or category hierarchies.

JSON:

json: Stores JSON (JavaScript Object Notation) data in a structured format.

SQL Server introduced native support for JSON data in version 2016, allowing you to store, retrieve, and query JSON data directly within the database.

Full-Text Search:

varchar(max) with Full-Text Index: Full-Text Indexing enables efficient searching of text data stored in varchar(max) columns.

Full-Text Search allows you to perform advanced searches on textual data, including linguistic analysis, stemming, and proximity searches.

Temporal Data:

datetime2 with System-Versioning: System-Versioned Temporal Tables allow you to track historical changes to data automatically.

Temporal tables keep track of data changes over time, allowing you to query data as it existed at different points in time.

Table-Valued Parameters (TVP):

Table-Valued Parameters: Allows passing a table variable as a parameter to stored procedures or functions.

TVPs are useful for passing multiple rows of data to a stored procedure or function, enabling bulk operations and reducing round trips to the database server.

Filestream:

varbinary(max) with Filestream: Stores large binary data (e.g., documents, images) directly in the file system.

Filestream allows you to efficiently manage and access large binary data while benefiting from SQL Server's transactional consistency and backup capabilities.

These advanced data types provide additional functionality and capabilities beyond the basic data types, enabling more sophisticated data storage, retrieval, and manipulation in SQL Server.

**SQL Server constraints**

In SQL Server, constraints are rules that enforce data integrity and maintain consistency within a database. Constraints define limits or conditions on the data that can be stored in a table, ensuring that only valid data is inserted, updated, or deleted. SQL Server supports various types of constraints to enforce different rules and requirements. Here are some common constraints used in SQL Server:

**Primary Key Constraint:**

Ensures that each row in a table is uniquely identified by a column or combination of columns.

Primary key columns must contain unique values and cannot contain NULL values.

Syntax:

ALTER TABLE table\_name

ADD CONSTRAINT pk\_constraint\_name PRIMARY KEY (column1, column2, ...);

**Foreign Key Constraint:**

Enforces referential integrity by ensuring that values in a column or combination of columns match values in a related table's primary key or unique key.

Foreign key columns must match values in the referenced table's primary key or unique key columns, or be NULL.

Syntax:

ALTER TABLE child\_table

ADD CONSTRAINT fk\_constraint\_name FOREIGN KEY (child\_column)

REFERENCES parent\_table (parent\_column);

**Unique Constraint:**

Ensures that values in a column or combination of columns are unique across the table.

Unlike primary key constraints, unique constraints allow NULL values, but only one NULL value is allowed per column.

Syntax:

ALTER TABLE table\_name

ADD CONSTRAINT unique\_constraint\_name UNIQUE (column1, column2, ...);

**Check Constraint:**

Enforces that values in a column meet a specific condition or set of conditions.

Check constraints are used to validate the data integrity based on specified conditions.

Syntax:

ALTER TABLE table\_name

ADD CONSTRAINT check\_constraint\_name CHECK (condition);

**Default Constraint:**

Specifies a default value for a column when no value is explicitly provided during an INSERT operation.

Default constraints are automatically applied to new rows if no value is provided for the column.

Syntax:

ALTER TABLE table\_name

ADD CONSTRAINT default\_constraint\_name DEFAULT default\_value FOR column\_name;

**NOT NULL constraint:**

NOT NULL constraint in SQL Server ensures that a column cannot contain NULL values. When you define a column with the NOT NULL constraint, it means that every row in the table must have a value for that column, and NULL is not allowed. This constraint enforces data integrity by ensuring that essential data is always provided and cannot be omitted.

Here's how you can apply the NOT NULL constraint when creating or altering a table in SQL Server:

When Creating a Table:

CREATE TABLE table\_name (

column1 datatype NOT NULL,

column2 datatype, ...

);

Example:

CREATE TABLE Employees (

EmployeeID INT NOT NULL,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

DepartmentID INT

);

When Altering an Existing Table:

ALTER TABLE table\_name

ALTER COLUMN column\_name datatype NOT NULL;

Example:

ALTER TABLE Employees

ALTER COLUMN FirstName VARCHAR(50) NOT NULL;

By specifying the NOT NULL constraint, you ensure that the specified column cannot contain NULL values. If you attempt to insert or update a row without providing a value for a NOT NULL column, SQL Server will generate an error, indicating that the constraint has been violated.

Using the NOT NULL constraint is important for columns that must always have a value, such as primary key columns, foreign key columns, or columns that are essential for data integrity and business logic. It helps maintain consistency and accuracy in your database by preventing the insertion of incomplete or missing data.

The NOT NULL constraint in SQL Server ensures that a column cannot contain NULL values. When you define a column with the NOT NULL constraint, it means that every row in the table must have a value for that column, and NULL is not allowed. This constraint enforces data integrity by ensuring that essential data is always provided and cannot be omitted. The NOT NULL constraint in SQL Server ensures that a column cannot contain NULL values. When you define a column with the NOT NULL constraint, it means that every row in the table must have a value for that column, and NULL is not allowed. This constraint enforces data integrity by ensuring that essential data is always provided and cannot be omitted. Top of Form

Top of Form