NOT NULL CONSTRAINT

The NOT NULL constraint ensures that a column cannot contain NULL values, meaning it enforces that every row must have a value for that column.

```
-- Creating a table with the NOT NULL
constraint

CREATE TABLE Employees (
    EmployeeID INT NOT NULL,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Age INT
);
```

In this example, we've created a table called Employees. We've applied the NOT NULL constraint to the EmployeeID, FirstName, and LastName columns. This means that when inserting data into this table, you must provide values for these columns. The Age column does not have a NOT NULL constraint, so it can have NULL values.

```
-- Adding a NOT NULL constraint to an existing column using ALTER command

ALTER TABLE Employees MODIFY Age INT NOT NULL;);
```

In this example, we're altering the Employees table. We're modifying the Age column to add the NOT NULL constraint. After executing this command, the Age column will not accept NULL values, and any attempt to insert a row without specifying an Age value will result in an error.

UNIQUE CONSTRAINT

The UNIQUE constraint ensures that values in a column are unique across all rows in a table. It prevents duplicate values from being inserted into the specified column.

```
-- Creating a table with the UNIQUE constraint

CREATE TABLE Students (
    StudentID INT UNIQUE,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);
```

In this example, we've created a table called Students and applied the UNIQUE constraint to the StudentID column. This means that every value in the StudentID column must be unique. You cannot insert two rows with the same StudentID.

```
-- Adding a UNIQUE constraint to an existing column using ALTER command

ALTER TABLE Students ADD CONSTRAINT UC_StudentID UNIQUE (StudentID);
```

In this example, we're altering the Students table and adding a UNIQUE constraint named UC_StudentID to the StudentID column.
After executing this command, the StudentID column will enforce uniqueness, ensuring that no two rows have the same StudentID.

PRIMARY KEY CONSTRAINT

The PRIMARY KEY constraint is a powerful constraint in SQL.
It's used to identify each row uniquely in a table.

```
-- Creating a table with the PRIMARY KEY constraint

CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);
```

In this example, we've created a table called Students and applied the PRIMARY KEY constraint to the StudentID column. This makes StudentID both unique and non-null, ensuring that each student has a unique identifier.

```
-- Adding a PRIMARY KEY constraint to an existing column using ALTER command

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

In this example, we're altering the Students table and adding a PRIMARY KEY constraint named PK_StudentID to the StudentID column.

Why Not Null and Unique Are Not Required When Primary Key Is Applied?

By using PRIMARY KEY, you're automatically ensuring that the column will contain unique, non-null values, making the NOT NULL and UNIQUE constraints redundant when applied to the same column. It simplifies the schema while guaranteeing data integrity.

DEFAULT CONSTRAINT

The DEFAULT constraint allows you to specify a default value for a column. When a new row is inserted into the table and no value is provided for that column, the default value is used.

```
-- Creating a table with DEFAULT constraint

CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Department VARCHAR(50) DEFAULT
‘GENERAL’
);
```

In this example, we've created an Employees table with a DEFAULT constraint on the Department column. If you insert a new employee without specifying a department, it will default to 'General'.

```
-- Adding a DEFAULT constraint to an existing column using ALTER command

ALTER TABLE employees MODIFY DEPARTMENT VARCHAR(50) DEFAULT ('GENERAL');
```

In this example, we're altering the Employees table to set a default value of 'General' for the Department column.

CHECK CONSTRAINT

The CHECK constraint allows you to define a condition that values in a column must satisfy. If a value doesn't meet the specified condition, the CHECK constraint prevents the row from being inserted or updated.

```
-- Creating a table with CHECK constraint

CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100),
    Price DECIMAL,
    Quantity INT,
    CHECK (Price >= 0 AND Quantity >= 0)
);
```

In this example, we've created a Products table with a CHECK constraint. It ensures that both the Price and Quantity columns have values greater than or equal to zero.

```
-- Adding a CHECK constraint to an existing table using ALTER command

ALTER TABLE Products ADD CONSTRAINT CHK_Price_Quantity CHECK (Price >= 0 AND Quantity >= 0);
```

This command adds a CHECK constraint named CHK_Price_Quantity to the Products table, enforcing the condition that both Price and Quantity must be non-negative.

FOREIGN KEY CONSTRAINT

The FOREIGN KEY constraint is used to establish a relationship between two tables in a database. It ensures that values in one table's column match the values in another table's primary key column. This enforces referential integrity, maintaining the consistency and accuracy of related data.

```
-- Creating the Customers table with a primary key

CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(50),
    Email VARCHAR(100)
);

-- Creating the Orders table with a foreign key reference

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    OrderDate DATE,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

In this example, we first create a Customers table with a primary key on the CustomerID column. Then, we create an Orders table and use the FOREIGN KEY constraint to link the CustomerID column in the Orders table to the CustomerID column in the Customers table. This ensures that every order's CustomerID value corresponds to a valid customer in the Customers table.

```
-- Adding a FOREIGN KEY constraint to an existing table

ALTER TABLE OrderItems ADD
CONSTRAINT FK_ProductID FOREIGN KEY
(ProductID) REFERENCES
Products(ProductID);
```

In this example, we have an existing OrderItems table, and we want to create a relationship between the ProductID column in the OrderItems table and the ProductID column in the Products table. The ALTER TABLE command adds the FOREIGN KEY constraint FK_ProductID to enforce referential integrity.