Module-4

Set Operators:
Set operators are used to combine the results of two or more SELECT statements into a single result set. The most common set operators are UNION, INTERSECT, and MINUS.

Union, intersect & Minus Clause:

UNION
The UNION operator combines the results of two or more SELECT statements into a single result set, removing any duplicate rows.
This can be useful for tasks such as finding all customers who have placed an order in the last month or identifying the products that have been ordered by more than one customer.

```
-- Syntax:
SELECT * FROM table1
UNION
SELECT * FROM table2;

-- Example:
SELECT * FROM customers
UNION
SELECT * FROM employees;
```

This query will return a list of all customers and employees, removing any duplicate rows.

INTERSECT
The INTERSECT operator returns the rows that are present in all of the SELECT statements.
This can be useful for tasks such as finding all people who are both customers and employees or finding all products that are in stock in both warehouses.

```
-- Syntax:
SELECT * FROM table1
INTERSECT
SELECT * FROM table2;

-- Example:
SELECT * FROM customers
INTERSECT
SELECT * FROM employees;
```

This query will return a list of all people who are both customers and employees.

MINUS
The MINUS operator returns the rows that are present in the first SELECT statement but not in the second SELECT statement.
This can be useful for tasks such as finding all customers who are not employees or finding all products that are in stock in the first warehouse but not in the second warehouse.

```
-- Syntax:
SELECT * FROM table1
MINUS
```

SELECT * FROM table2;

DDL commands (CREATE, drop, alter, rename, truncate):

DDL commands (Data Definition Language) are used to create, modify, or delete database objects, such as tables, views, and indexes. DDL commands are typically executed by database administrators, but they can also be used by application developers to create and manage the database objects needed for their applications.

CREATE

The CREATE command is used to create new database objects. For example, the following statement creates a new table called customers:

```
CREATE TABLE customers (
  customer_id INT NOT NULL AUTO_INCREMENT,
  customer_name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  PRIMARY KEY (customer_id)
);
```

DROP

The DROP command is used to delete existing database objects. For example, the following statement drops the customers table:

```
DROP TABLE customers;
```

ALTER

The ALTER command is used to modify existing database objects. For example, the following statement adds a new column called phone_number to the customers table:

```
ALTER TABLE customers
ADD COLUMN phone_number VARCHAR(255);
```

RENAME

The RENAME command is used to rename existing database objects. For example, the following statement renames the customers table to customers_old:

```
RENAME TABLE customers TO customers_old;
```

TRUNCATE

The TRUNCATE command is used to remove all data from a table without deleting the table itself. For example, the following statement truncates the customers table:

```
TRUNCATE TABLE customers;
```

Data Types:

Data types are used to define the type of data that can be stored in a column in a database table. Data types are important because they help to ensure that data is stored and processed correctly.

There are many different data types available in , each with its own purpose. Some of the most common data types include:

- **Numeric data types:** These data types are used to store numeric data, such as integers, decimals, and floating-point numbers. Examples of numeric data types include INTEGER, DECIMAL, and FLOAT.

- **Character data types:** These data types are used to store character data, such as strings and text. Examples of character data types include CHAR, VARCHAR, and TEXT.
- **Date and time data types:** These data types are used to store date and time data. Examples of date and time data types include DATE, TIME, and DATETIME.
- **Binary data types:** These data types are used to store binary data, such as images and video files. Examples of binary data types include BLOB and BINARY.
- **Specialized data types:** These data types are used to store specialized data, such as spatial data and object-relational data. Examples of specialized data types include GEOMETRY and XML.

Examples of Data Types

Here are some examples of data types and how they can be used:

- **INTEGER:** The INTEGER data type is used to store integer data, such as whole numbers. For example, the following column definition uses the INTEGER data type to store the customer ID:

```
CREATE TABLE customers (
  customer_id INTEGER NOT NULL AUTO_INCREMENT,
  customer_name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  PRIMARY KEY (customer_id)
);
```

- **VARCHAR:** The VARCHAR data type is used to store variable-length character data, such as strings and text. For example, the following column definition uses the VARCHAR data type to store the customer name:

```
CREATE TABLE customers (
  customer_id INTEGER NOT NULL AUTO_INCREMENT,
  customer_name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  PRIMARY KEY (customer_id)
);
```

- **DATE:** The DATE data type is used to store date and time data. For example, the following column definition uses the DATE data type to store the customer's birth date:

```
CREATE TABLE customers (
  customer_id INTEGER NOT NULL AUTO_INCREMENT,
  customer_name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  birth_date DATE,
  PRIMARY KEY (customer_id)
);
```

- **BLOB:** The BLOB data type is used to store binary data, such as images and video files. For example, the following column definition uses the BLOB data type to store the customer's profile picture:

```
CREATE TABLE customers (
  customer_id INTEGER NOT NULL AUTO_INCREMENT,
  customer_name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  birth_date DATE,
  profile_picture BLOB,
```

```
  PRIMARY KEY (customer_id)
);
```

TABLE Command:
The TABLE command in is used to create a new table in a database.

Syntax:

```
CREATE TABLE table_name (
  column_name1 data_type1,
  column_name2 data_type2,
  ...
);
```

For example, the following statement creates a new table called customers:

```
CREATE TABLE customers (
  customer_id INT NOT NULL AUTO_INCREMENT,
  customer_name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  PRIMARY KEY (customer_id)
);
```

This table will have three columns: customer_id, customer_name, and email. The customer_id column will be the primary key for the table.
The TABLE command can also be used to create a table with constraints. For example, the following statement creates a new table called orders with a foreign key constraint on the customer_id column:

```
CREATE TABLE orders (
  order_id INT NOT NULL AUTO_INCREMENT,
  customer_id INT NOT NULL,
  order_date DATE NOT NULL,
  PRIMARY KEY (order_id),
  FOREIGN KEY (customer_id) REFERENCES customers (customer_id)
);
```

This table will have three columns: order_id, customer_id, and order_date. The order_id column will be the primary key for the table. The customer_id column will be a foreign key that references the customer_id column in the customers table.

The TABLE command is a powerful tool for creating new tables in a database. By using the TABLE command, you can create tables with any number of columns and any data type. You can also create tables with constraints to ensure that the data is stored and processed correctly.

Constraints:
Constraints in are used to define rules that must be satisfied by the data in a table. Constraints can be used to ensure that the data is valid, consistent, and accurate.

There are many different types of constraints available in , each with its own purpose. Some of the most common types of constraints include:

- **Not null constraints:** These constraints prevent null values from being inserted into a column.
- **Unique constraints:** These constraints ensure that each value in a column is unique.
- **Primary key constraints:** These constraints identify a unique column or set of columns in a table that can be used to uniquely identify each row in the table.

- **Foreign key constraints:** These constraints ensure that the values in a column match the values in a corresponding column in another table.
- **Check constraints:** These constraints define custom rules that the data in a column must satisfy.

Constraints can be defined when a table is created or added to an existing table. Constraints can also be dropped from tables.

Examples of Constraints

Here are some examples of constraints:

```
CREATE TABLE customers (
  customer_id INT NOT NULL AUTO_INCREMENT,
  customer_name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  PRIMARY KEY (customer_id)
);

ALTER TABLE orders
ADD CONSTRAINT fk_orders_customers
FOREIGN KEY (customer_id) REFERENCES customers (customer_id);

ALTER TABLE products
ADD CONSTRAINT ck_products_price
CHECK (price > 0);
```

The first example defines a not null constraint on the customer_name and email columns in the customers table. This constraint ensures that null values cannot be inserted into those columns.
The second example defines a foreign key constraint on the customer_id column in the orders table. This constraint ensures that the values in the customer_id column of the orders table match the values in the customer_id column of the customers table.
The third example defines a check constraint on the price column in the products table. This constraint ensures that the values in the price column are greater than 0.

Benefits of Constraints

Constraints offer a number of benefits, including:

- **Improved data integrity:** Constraints help to ensure that the data in a database is valid, consistent, and accurate. This can help to prevent errors and improve the reliability of applications.
- **Increased performance:** Constraints can help to improve the performance of database queries by reducing the amount of data that needs to be processed.
- **Simplified application development:** Constraints can help to simplify the development of database applications by providing a way to enforce business rules in the database itself.

Conclusion

Constraints are a powerful tool for improving the quality and reliability of data in a database. By using constraints, you can ensure that the data in your database is valid, consistent, accurate, and performs well.

Primary key:
A primary key in a database table is a column or set of columns that uniquely identifies each row in the table. Primary keys are important because they help to ensure that the data in the table is accurate and consistent.

Primary keys must be unique, meaning that no two rows in the table can have the same primary key value. Primary keys must also be not null, meaning that they cannot contain null values.

Primary keys are often used to create relationships between tables. For example, the customer_id column in the orders table could be a foreign key that references the customer_id column in the customers table. This relationship allows you to easily find all of the orders placed by a particular customer.

Benefits of Primary Keys

To create a primary key on a column in a table, you can use the following statement:

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name PRIMARY KEY (column_name);
```

For example, the following statement creates a primary key on the customer_id column in the customers table:

```
ALTER TABLE customers
ADD CONSTRAINT pk_customers PRIMARY KEY (customer_id);
```

If you are creating a new table, you can also define the primary key when you create the table. To do this, you can use the following statement:

```
CREATE TABLE table_name (
  column_name1 data_type1,
  column_name2 data_type2,
  ...,
  PRIMARY KEY (column_name)
);
```

For example, the following statement creates a new table called products with a primary key on the product_id column:

```
CREATE TABLE products (
  product_id INT NOT NULL AUTO_INCREMENT,
  product_name VARCHAR(255) NOT NULL,
  product_price DECIMAL(10,2) NOT NULL,
  PRIMARY KEY (product_id)
);
```

Primary keys offer a number of benefits, including:

- **Improved data integrity:** Primary keys help to ensure that the data in a database is valid and consistent. For example, primary keys can prevent duplicate rows from being inserted into a table and can ensure that foreign key constraints are satisfied.
- **Increased performance:** Primary keys can help to improve the performance of database queries by reducing the amount of data that needs to be processed. For example, when you query a table using the primary key, the database can quickly find the row that you are looking for.
- **Simplified application development:** Primary keys can help to simplify the development of database applications by providing a way to uniquely identify rows in a table. For example, primary keys can be used to create relationships between tables and to generate unique identifiers for new rows.

Foreign key:

A foreign key in a relational database is a column or set of columns in one table that references the primary key of another table. Foreign keys are used to create relationships between tables, which allows you to link data together and perform complex queries.

For example, the customer_id column in the orders table could be a foreign key that references the customer_id column in the customers table. This relationship allows you to easily find all of the orders

placed by a particular customer.
Foreign key constraints can be enforced to ensure that the data in the two tables is consistent. For example, the foreign key constraint on the customer_id column in the orders table would prevent you from inserting an order for a customer who does not exist in the customers table.
Foreign keys are a powerful tool for improving the quality and reliability of data in a database. By using foreign keys, you can ensure that the data in your database is linked together correctly and that it is consistent.

Benefits of Foreign Keys

Foreign keys offer a number of benefits, including:

- **Improved data integrity:** Foreign keys help to ensure that the data in a database is valid and consistent. For example, foreign keys can prevent duplicate rows from being inserted into a table and can ensure that foreign key constraints are satisfied.
- **Increased performance:** Foreign keys can help to improve the performance of database queries by reducing the amount of data that needs to be processed. For example, when you query a table using a foreign key, the database can quickly find the related row in the other table.
- **Simplified application development:** Foreign keys can help to simplify the development of database applications by providing a way to link data together in the database itself. For example, foreign keys can be used to create relationships between tables and to enforce business rules.

Check, not null, unique:
CHECK:

A CHECK constraint in is used to define a custom rule that the data in a column must satisfy. CHECK constraints can be used to validate data, ensure that it is within a certain range, or prevent certain values from being inserted.
For example, the following statement creates a CHECK constraint on the price column in the products table to ensure that the values in the column are greater than 0:

```
ALTER TABLE products
ADD CONSTRAINT ck_products_price
CHECK (price > 0);
```

Once this constraint is created, you will not be able to insert a row into the products table with a negative value in the price column.

NOT NULL:

A NOT NULL constraint in prevents null values from being inserted into a column. NOT NULL constraints are useful for ensuring that all rows in a table have a value for a particular column.
For example, the following statement creates a NOT NULL constraint on the customer_name column in the customers table:

```
ALTER TABLE customers
ADD CONSTRAINT ck_customers_customer_name
NOT NULL;
```

Once this constraint is created, you will not be able to insert a row into the customers table with a null value in the customer_name column.

UNIQUE

A UNIQUE constraint in ensures that each value in a column is unique. UNIQUE constraints are useful for preventing duplicate rows from being inserted into a table.

For example, the following statement creates a UNIQUE constraint on the email column in the customers table:

```
ALTER TABLE customers
ADD CONSTRAINT ck_customers_email
UNIQUE;
```

Once this constraint is created, you will not be able to insert a row into the customers table with the same email address as another row in the table.

Benefits of CHECK, NOT NULL, and UNIQUE constraints

CHECK, NOT NULL, and UNIQUE constraints offer a number of benefits, including:

- **Improved data integrity:** These constraints help to ensure that the data in a database is valid and consistent. For example, CHECK constraints can prevent invalid data from being inserted into a column, NOT NULL constraints can prevent rows from being inserted without a value for a particular column, and UNIQUE constraints can prevent duplicate rows from being inserted into a table.
- **Increased performance:** These constraints can help to improve the performance of database queries by reducing the amount of data that needs to be processed. For example, when you query a table using a UNIQUE constraint, the database can quickly find the row that you are looking for.
- **Simplified application development:** These constraints can help to simplify the development of database applications by providing a way to enforce business rules in the database itself. For example, CHECK constraints can be used to validate data before it is inserted into a table, and NOT NULL and UNIQUE constraints can be used to ensure that the data in a table is consistent.

Conclusion

CHECK, NOT NULL, and UNIQUE constraints are powerful tools for improving the quality and reliability of data in a database. By using these constraints, you can ensure that the data in your database is valid, consistent, and accurate.

Setting constraints using Column level & table Level Constraints:

Both column-level and table-level constraints can be used to define rules for the data in a database table. Column-level constraints apply to individual columns, while table-level constraints apply to the entire table.

To set a column-level constraint, specify the constraint after the column definition in the CREATE TABLE statement. For example, the following statement creates a customers table with a NOT NULL constraint on the customer_name column:

```
CREATE TABLE customers (
  customer_id INT NOT NULL AUTO_INCREMENT,
  customer_name VARCHAR(255) NOT NULL,
  PRIMARY KEY (customer_id)
);
```

To set a table-level constraint, specify the constraint after the column definitions in the CREATE TABLE statement. For example, the following statement creates a products table with a UNIQUE constraint on the product_name column:

```
CREATE TABLE products (
  product_id INT NOT NULL AUTO_INCREMENT,
  product_name VARCHAR(255) UNIQUE,
  PRIMARY KEY (product_id)
);
```

Table-level constraints can also be used to specify relationships between tables. For example, the following statement creates a orders table with a FOREIGN KEY constraint on the customer_id column that references the customer_id column in the customers table:

```
CREATE TABLE orders (
  order_id INT NOT NULL AUTO_INCREMENT,
  customer_id INT NOT NULL,
  order_date DATE NOT NULL,
  PRIMARY KEY (order_id),
  FOREIGN KEY (customer_id) REFERENCES customers (customer_id)
);
```

This constraint ensures that every order in the orders table is associated with a valid customer in the customers table.

Benefits of using column-level and table-level constraints

Using column-level and table-level constraints offers a number of benefits, including:

- **Improved data integrity:** Constraints help to ensure that the data in a database is valid, consistent, and accurate. For example, constraints can prevent invalid data from being inserted into a table, prevent duplicate rows from being inserted into a table, and ensure that relationships between tables are maintained.
- **Increased performance:** Constraints can help to improve the performance of database queries by reducing the amount of data that needs to be processed. For example, when you query a table using a constraint, the database can quickly find the rows that you are looking for.
- **Simplified application development:** Constraints can help to simplify the development of database applications by providing a way to enforce business rules in the database itself. For example, constraints can be used to validate data before it is inserted into a table and to ensure that the data in a table is consistent with business rules.

Conclusion

Column-level and table-level constraints are powerful tools for improving the quality and reliability of data in a database. By using constraints, you can ensure that the data in your database is valid, consistent, accurate, and performs well.

defining different constraints on the table:

There are many different types of constraints that can be defined on a table in . Some of the most common types of constraints include:

- **NOT NULL:** This constraint prevents null values from being inserted into a column.
- **UNIQUE:** This constraint ensures that each value in a column is unique.
- **PRIMARY KEY:** This constraint identifies a unique column or set of columns in a table that can be used to uniquely identify each row in the table.
- **FOREIGN KEY:** This constraint ensures that the values in a column match the values in a corresponding column in another table.
- **CHECK:** This constraint defines a custom rule that the data in a column must satisfy.

Examples of different constraints on a table

The following examples show how to define different constraints on a table:

```
CREATE TABLE customers (
  customer_id INT NOT NULL AUTO_INCREMENT,
  customer_name VARCHAR(255) NOT NULL,
  email VARCHAR(255) UNIQUE,
  PRIMARY KEY (customer_id)
);
```

This table has the following constraints:

- The customer_id column is a primary key, which means that each value in the column must be unique and non-null.
- The customer_name column is not null, which means that it cannot contain null values.
- The email column is unique, which means that each value in the column must be unique.

```
CREATE TABLE orders (
  order_id INT NOT NULL AUTO_INCREMENT,
  customer_id INT NOT NULL,
  order_date DATE NOT NULL,
  PRIMARY KEY (order_id),
  FOREIGN KEY (customer_id) REFERENCES customers (customer_id)
);
```

This table has the following constraints:

- The order_id column is a primary key, which means that each value in the column must be unique and non-null.
- The customer_id column is not null, which means that it cannot contain null values.
- The customer_id column has a foreign key constraint that references the customer_id column in the customers table. This constraint ensures that every order in the orders table is associated with a valid customer in the customers table.

```
CREATE TABLE products (
  product_id INT NOT NULL AUTO_INCREMENT,
  product_name VARCHAR(255) NOT NULL,
  product_price DECIMAL(10,2) NOT NULL,
  PRIMARY KEY (product_id),
  CHECK (product_price > 0)
);
```

This table has the following constraints:

- The product_id column is a primary key, which means that each value in the column must be unique and non-null.
- The product_name column is not null, which means that it cannot contain null values.
- The product_price column is not null, which means that it cannot contain null values.
- The product_price column also has a check constraint that ensures that the value in the column is greater than 0.

Benefits of defining constraints on a table

Defining constraints on a table offers a number of benefits, including:

- **Improved data integrity:** Constraints help to ensure that the data in a database is valid, consistent, and accurate. For example, constraints can prevent invalid data from being inserted into a table, prevent duplicate rows from being inserted into a table, and ensure that relationships between tables are maintained.
- **Increased performance:** Constraints can help to improve the performance of database queries by

reducing the amount of data that needs to be processed. For example, when you query a table using a constraint, the database can quickly find the rows that you are looking for.

- **Simplified application development:** Constraints can help to simplify the development of database applications by providing a way to enforce business rules in the database itself. For example, constraints can be used to validate data before it is inserted into a table and to ensure that the data in a table is consistent with business rules.

Conclusion

Defining constraints on tables is an important part of database design. By using constraints, you can improve the quality, reliability, and performance of your database applications.

Defining Integrity Constraints in the ALTER:

To define integrity constraints in the ALTER TABLE statement, you use the ADD CONSTRAINT clause. The following syntax shows how to add a constraint to a table:

ALTER TABLE table_name
ADD CONSTRAINT constraint_name constraint_type;
The constraint_name is the name of the constraint. The constraint_type is the type of constraint that you want to add.

Constraint type	Description
`NOT NULL`	Prevents null values from being inserted into a column.
`UNIQUE`	Ensures that each value in a column is unique.
`PRIMARY KEY`	Identifies a unique column or set of columns in a table that can be used to uniquely identify each row in the table.
`FOREIGN KEY`	Ensures that the values in a column match the values in a corresponding column in another table.
`CHECK`	Defines a custom rule that the data in a column must satisfy.

Examples of defining integrity constraints in the ALTER TABLE statement

The following examples show how to define different integrity constraints in the ALTER TABLE statement:

ALTER TABLE customers
ADD CONSTRAINT ck_customers_customer_name
NOT NULL;
This statement adds a NOT NULL constraint to the customer_name column in the customers table. This constraint ensures that the customer_name column cannot contain null values.

ALTER TABLE orders
ADD CONSTRAINT ck_orders_customer_id
FOREIGN KEY (customer_id) REFERENCES customers (customer_id);
This statement adds a FOREIGN KEY constraint to the customer_id column in the orders table. This constraint ensures that every order in the orders table is associated with a valid customer in the customers table.

ALTER TABLE products
ADD CONSTRAINT ck_products_product_price

CHECK (product_price > 0);

This statement adds a CHECK constraint to the product_price column in the products table. This constraint ensures that the values in the product_price column are greater than 0.

Benefits of defining integrity constraints in the ALTER TABLE statement

Defining integrity constraints in the ALTER TABLE statement offers a number of benefits, including:

- **Improved data integrity:** Constraints help to ensure that the data in a database is valid, consistent, and accurate. For example, constraints can prevent invalid data from being inserted into a table, prevent duplicate rows from being inserted into a table, and ensure that relationships between tables are maintained.
- **Increased performance:** Constraints can help to improve the performance of database queries by reducing the amount of data that needs to be processed. For example, when you query a table using a constraint, the database can quickly find the rows that you are looking for.
- **Simplified application development:** Constraints can help to simplify the development of database applications by providing a way to enforce business rules in the database itself. For example, constraints can be used to validate data before it is inserted into a table and to ensure that the data in a table is consistent with business rules.

Conclusion

Defining integrity constraints in the ALTER TABLE statement is a powerful way to improve the quality and reliability of data in a database. By using constraints, you can ensure that the data in your database is valid, consistent, accurate, and performs well.