The SQL CREATE DATABASE Statement

The `CREATE DATABASE` statement is used to create a new SQL database.

Syntax

```
CREATE DATABASE databasename;
```

The SQL DROP DATABASE Statement

The `DROP DATABASE` statement is used to drop an existing SQL database.

Syntax

```
DROP DATABASE databasename;
```

The SQL BACKUP DATABASE Statement

The `BACKUP DATABASE` statement is used in SQL Server to create a full back up of an existing SQL database.

Syntax

```
BACKUP DATABASE databasename

TO DISK = 'filepath';
```

Example

Get your own SQL Server

```
BACKUP DATABASE testDB

TO DISK = 'D:\backups\testDB.bak';
```

Tip: Always back up the database to a different drive than the actual database. Then, if you get a disk crash, you will not lose your backup file along with the database.

BACKUP WITH DIFFERENTIAL Example

The following SQL statement creates a differential back up of the database "testDB":

Example

```
BACKUP DATABASE testDB

TO DISK = 'D:\backups\testDB.bak'

WITH DIFFERENTIAL;
```

The SQL CREATE TABLE Statement

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)
);
```

The SQL DROP TABLE Statement

The `DROP TABLE` statement is used to drop an existing table in a database.

Syntax

```
DROP TABLE table_name;
```

Note: Be careful before dropping a table. Deleting a table will result in loss of complete information stored in the table!

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;
```

SQL 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.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name

ADD column_name datatype;
```

The following SQL adds an "Email" column to the "Customers" table:

Example

Get your own SQL Server

```
ALTER TABLE Customers

ADD Email varchar(255);
```

ALTER TABLE - DROP COLUMN

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name

DROP COLUMN column_name;
```

The following SQL deletes the "Email" column from the "Customers" table:

Example

```
ALTER TABLE Customers

DROP COLUMN Email;
```

SQL Create Constraints

Constraints can be specified when the table is created with the `CREATE TABLE` statement, or after the table is created with the `ALTER TABLE` statement.

Syntax

```
CREATE TABLE table_name (

  column1 datatype constraint,

  column2 datatype constraint,

  column3 datatype constraint,
    ....

);
```

SQL Constraints

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

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- `NOT NULL` - Ensures that a column cannot have a NULL value
- `UNIQUE` - Ensures that all values in a column are different
- `PRIMARY KEY` - A combination of a `NOT NULL` and `UNIQUE`. Uniquely identifies each row in a table
- `FOREIGN KEY` - Prevents actions that would destroy links between tables
- `CHECK` - Ensures that the values in a column satisfies a specific condition
- `DEFAULT` - Sets a default value for a column if no value is specified
- `CREATE INDEX` - Used to create and retrieve data from the database very quickly

SQL NOT NULL Constraint

By default, a column can hold NULL values.

The `NOT NULL` constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

SQL NOT NULL on CREATE TABLE

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

Example

Get your own SQL Server

```
CREATE TABLE Persons (

  ID int NOT NULL,

    LastName varchar(255) NOT NULL,

  FirstName varchar(255) NOT NULL,

  Age int

);
```

SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a `UNIQUE` constraint on the "ID" column when the "Persons" table is created:

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (

  ID int NOT NULL UNIQUE
```

```
    LastName varchar(255) NOT NULL,

    FirstName varchar(255),

    Age int

);
```

SQL PRIMARY KEY Constraint

The `PRIMARY KEY` constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

SQL PRIMARY KEY on CREATE TABLE

The following SQL creates a `PRIMARY KEY` on the "ID" column when the "Persons" table is created:

MySQL:

```
CREATE TABLE Persons (

    ID int NOT NULL,

    LastName varchar(255) NOT NULL,

    FirstName varchar(255),

    Age int,
    PRIMARY KEY (ID)

);
```

SQL FOREIGN KEY Constraint

The `FOREIGN KEY` constraint is used to prevent actions that would destroy links between tables.

A `FOREIGN KEY` is a field (or collection of fields) in one table, that refers to the `PRIMARY KEY` in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

SQL 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 column it will allow only certain values for this column.

If you define a `CHECK` constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SQL CHECK on CREATE TABLE

The following SQL creates a `CHECK` constraint on the "Age" column when the "Persons" table is created. The `CHECK` constraint ensures that the age of a person must be 18, or older:

MySQL:

```
CREATE TABLE Persons (

  ID int NOT NULL,

  LastName varchar(255) NOT NULL,

  FirstName varchar(255),

  Age int,

    CHECK (Age>=18)

);
```

SQL DEFAULT Constraint

The `DEFAULT` constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

SQL DEFAULT on CREATE TABLE

The following SQL sets a `DEFAULT` value for the "City" column when the "Persons" table is created:

My SQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

City varchar(255) DEFAULT 'Sandnes'
```

SQL CREATE INDEX Statement

The `CREATE INDEX` statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

```
CREATE INDEX index_name

ON table_name (column1, column2, ...);
```

CREATE UNIQUE INDEX Syntax

Creates a unique index on a table. Duplicate values are not allowed:

```
CREATE UNIQUE INDEX index_name

ON table_name (column1, column2, ...);
```

Note: The syntax for creating indexes varies among different databases. Therefore: Check the syntax for creating indexes in your database.

CREATE INDEX Example

The SQL statement below creates an index named "idx_lastname" on the "LastName" column in the "Persons" table:

```
CREATE INDEX idx_lastname

ON Persons (LastName);
```

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

```
CREATE INDEX idx_pname

ON Persons (LastName, FirstName);
```

DROP INDEX Statement

The `DROP INDEX` statement is used to delete an index in a table.

MS Access:

```
DROP INDEX index_name ON table_name;
```

SQL Server:

```
DROP INDEX table_name.index_name;
```

DB2/Oracle:

```
DROP INDEX index_name;
```

MySQL:

```
ALTER TABLE table_name

DROP INDEX index_name;
```

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.

Syntax for MySQL

The following SQL statement defines the "Personid" column to be an auto-increment primary key field in the "Persons" table:

```
CREATE TABLE Persons (

    Personid int NOT NULL AUTO_INCREMENT,

  LastName varchar(255) NOT NULL,

  FirstName varchar(255),
  Age int,

    PRIMARY KEY (Personid)
```

MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:

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

To insert a new record into the "Persons" table, we will NOT have to specify a value for the "Personid" column (a unique value will be added automatically):

```
INSERT INTO Persons (FirstName,LastName)

VALUES ('Lars','Monsen');
```

SQL Date Data Types

MySQL comes with the following data types for storing a date or a date/time value in the database:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY

SQL Injection

SQL injection is a code injection technique that might destroy your database.

SQL injection is one of the most common web hacking techniques.

SQL injection is the placement of malicious code in SQL statements, via web page input.

SQL in Web Pages

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.

Look at the following example which creates a `SELECT` statement by adding a variable (txtUserId) to a select string. The variable is fetched from user input (getRequestString):

Example

Get your own SQL Server

```
txtUserId = getRequestString("UserId");


txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

Key Concepts in Data Modeling:

1. **Entities and Attributes**:
   - **Entities** represent objects or concepts in the real world (e.g., customers, products, transactions).

    - **Attributes** are details or properties of those entities (e.g., customer name, product price, transaction date).
2. **Relationships**:
    - Defines how entities are related to each other. For example, a customer might place multiple orders, so there is a **one-to-many relationship** between customers and orders.
3. **Primary and Foreign Keys**:
    - **Primary Key**: A unique identifier for each record in a table (e.g., CustomerID).
    - **Foreign Key**: A field that links one table to another, establishing relationships between entities.
4. **Normalization**:
    - The process of organizing data to reduce redundancy and improve data integrity. This typically involves splitting large tables into smaller, related tables.
5. **Data Types**:
    - Defining the data types for each attribute (e.g., `integer`, `text`, `date`) ensures consistency and avoids data entry errors.
6. **Schema Design**:
    - A schema defines the structure of the database. It can be visualized through **ER Diagrams** (Entity-Relationship Diagrams) which graphically represent entities, attributes, and relationships.