FTDS // DATABASE & SQL (Basic)

Hacktiv8 DS Curriculum Team

Phase 0 Learning Materials

Hacktiv8 DS
Curriculum
Team

Contents

Objectives

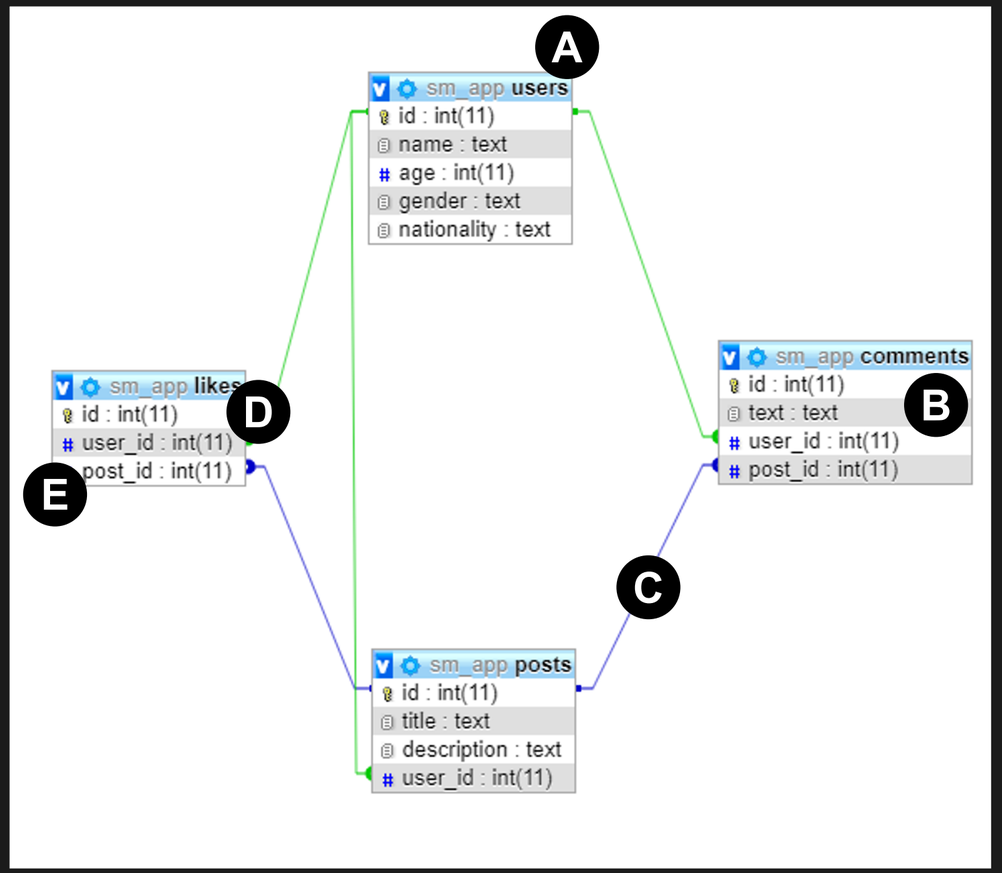
- Basic understanding of database scheme
- Basic understanding of SQL syntax

Data: Collection of information which can be numerics, images, sound, sentences, etc.

Database: Group of data that stored in a structured and possesses connections Interdata.

Database Scheme

A. **Table/Entity** which contains columns and rows/tuples (Ex. **Table users** in sm_app database)
B. **Columns/Attributes/Fields** (Ex. Column **id** in **comments** table)
C. **Relation line** (Ex. Line that connect **id** in **posts** to **post_id** in **comments**)
D. **Primary Key:** Identity key of each tuple in a table which must be unique
E. **Foreign Key:** A key that connect to a primary key of other tables



One-to-One

“A book written by an author”

One-to-Many

“A book written by many authors”

Many-to-Many

“Many book written by many authors”

In the database scheme, there are three relation types

Entity Set Symbol

Relationship Set Symbol

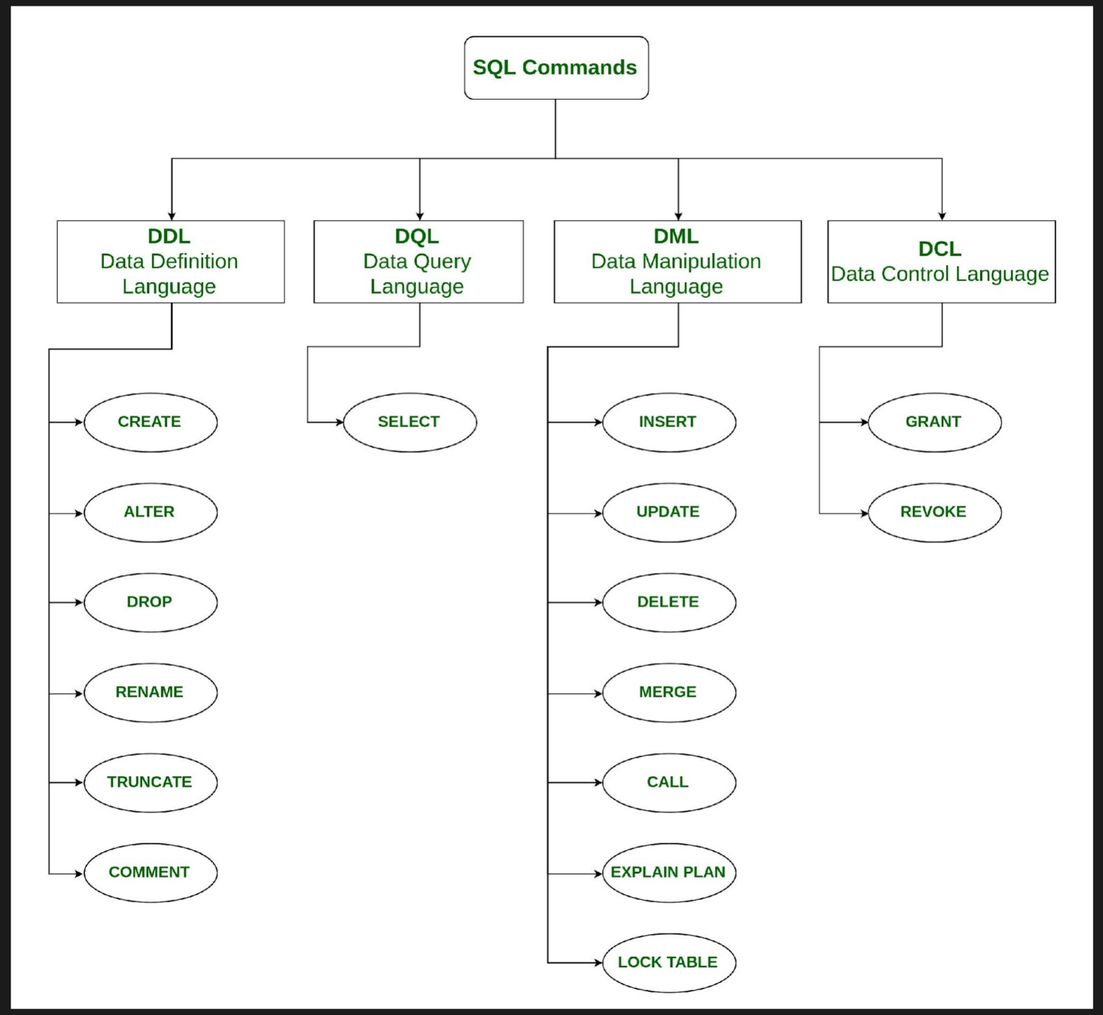
Crows Foot Notations

Popular Relational Database Management Systems (RDMS)

PostgreSQL

Pros	Cons	When to Use	When not to use
• File based • Standards-aware • Great for developing even testing	• No User management • Lack of possibility to tinker with for additional performance	• Embedded Applications • Disk access replacement • Testing	• Multi-user applications • Applications requiring high write volumes
• Easy to work with • Feature rich • Secure • Scalable and powerful • Speedy	• Known limitations • Reliability issues • Stagnated Development	• Distributed operations • High security Web-services and Web-applications • Custom solutions	• SQL compliences • Cocurrency • Lack of features
• An open source SQL standard compliant RDMS • Strong community • Strong third-party support • Extensible • Objective	• Performance • Popularity • Hosting	• Data integrity complex, custom procedures • Integration • Complex Designs	• Speed • Simple to sets up • Replication

SQL Basic Syntax



CREATE

To define a new table and the attributes. In this command, you can define the primary and/or foreign key as well.

```
CREATE TABLE TABLENAME(
    COLUMN1 datatype,
    COLUMN2 datatype,
    ...
    );
```

Also you can define a new database using CREATE.

```
CREATE DATABASE DB_NAME;
```

Example

```
CREATE TABLE teachers (
    id INT PRIMARY KEY AUTO_INCREMENT,
    first_name varchar(25),
    last_name varchar(50),
    school varchar(50),
    hire_date date,
    salary numeric
    );
```

DROP

Using DROP to delete a table

```
DROP TABLE TABLENAME;
```

Example

```
DROP TABLE teachers;
```

INSERT

Using INSERT to input values into the columns.

```
INSERT INTO Table_Name (colomn1, colomn2,...)
VALUES (value1, value2,...);
```

Example

```
INSERT INTO teachers (id, first_name, last_name, school, hire_date, salary)
    VALUES (1,'Janet', 'Smith', 'F.D. Roosevelt HS', '2011-10-30', 36200),
           (2,'Lee', 'Reynolds', 'F.D. Roosevelt HS', '1993-05-22', 65000),
           (3,'Samuel', 'Cole', 'Myers Middle School', '2005-08-01', 43500),
           (4,'Samantha', 'Bush', 'Myers Middle School', '2011-10-30', 36200),
           (5,'Betty', 'Diaz', 'Myers Middle School', '2005-08-30', 43500),
           (6,'Kathleen', 'Roush', 'F.D. Roosevelt HS', '2010-10-22', 38500);
```

UPDATE

UPDATE used for changing certain values in certain conditions.

```
UPDATE Table_Name SET column1=value1
WHERE <condition>;
```

Example

```
UPDATE teachers
    SET salary=56000
    WHERE teachers.first_name='Samuel';
```

DELETE

Remove values in a table using DELETE

To remove all values in a table

```
DELETE FROM Table_Name;
```

To remove values on certain conditions

```
DELETE FROM Table_Name
WHERE <conditions>
```

Example

```
DELETE FROM teachers
    WHERE id=6;
```

SELECT

Retreiving data or table from the database, we can use SELECT command. The basic syntax:

```
SELECT * FROM Table_Name;
```

```
SELECT column1,column2,…
FROM Table_Name;
```

Example

```
SELECT first_name,salary
FROM teachers;
```

NOTE: Using the asterisk wildcard is helpful for discovering the entire contents of a table. But often it's more practical to limit the columns the query retrieves, especially with large databases. You can do this by naming columns, separated by commas, right after the SELECT keyword.

Sometimes you want to retrieve the columns but you want to change the name of the prior table. You can use alias to rename the columns.

Example

```
SELECT first_name as “First Name”,
salary as “Yearly Stipend”
FROM teachers;
```

In a table, it’s not unusual for a column to contain rows with duplicate values. To understand the range of values in a column, we can use the DISTINCT keyword as part of a query that eliminates duplicates and shows only unique values. Use the DISTINCT keyword immediately after SELECT.

Example

```
SELECT DISTINCT School FROM teachers
```

also works on more than one column at a time. If we add a column, the query returns each unique pair of values.

Example

```
SELECT DISTINCT School,Salary FROM teachers
```

In SQL, we can order the results of a query using a clause containing the keywords ORDER BY followed by the name of the column or columns to sort. Applying this clause doesn't change the original table, only the result of the query.

Example

```
SELECT first_name, last_name, salary
FROM teachers
ORDER BY salary DESC;
```

In SQL, we can order the results of a query using a clause containing the keywords ORDER BY followed by the name of the column or columns to sort. Applying this clause doesn't change the original table, only the result of the query.

By default, ORDER BY sorts values in ascending order, but in order to sort in descending order by adding the DESC keyword.

Example

```
SELECT first_name, last_name, salary
FROM teachers
ORDER BY salary DESC;
```

Sometimes, you'll want to limit the rows a query returns to only those in which one or more columns meet certain criteria. This can be done by using WHERE keyword. The WHERE keyword allows you to find rows that match a specific value, a range of values, or multiple values based on criteria supplied via an operator. You also can exclude rows based on criteria.

Example

```
SELECT last_name, school, hire_date
FROM teachers
WHERE school = 'Myers Middle School';
```

Operator	Function	Example
=	Equal to	WHERE school = 'Baker Middle'
<> or !=	Not equal to	WHERE school <> 'Baker Middle'
>	Greater than	WHERE salary > 20000
<	Less than	WHERE salary < 60500
>=	Greater than or equal to	WHERE salary >= 20000
<=	Less than or qual to	WHERE salary <= 60500
BETWEEN	Within a range	WHERE salary BETWEEN 20000 AND 40000
IN	Match one of a set value	WHERE last_name IN ('Bush','Roush')
LIKE	Match a pattern (case sensitive)	WHERE first_name LIKE 'Sam%'
ILIKE	Match a pattern (incase sensitive)	WHERE first_name ILIKE 'sam%'
NOT	Negates a condition	WHERE first_name NOT ILIKE 'sam%'

Table beside provides a summary of the most commonly used comparison operators. Depending on your database system, many more might be available.

Comparison operators become even more useful when we combine them. To do this, we connect them using keywords AND and OR along with, if needed, parentheses.

Example

```
SELECT *
FROM teachers
WHERE school = 'Myers Middle School'
AND salary < 40000;
```

```
SELECT *
FROM teachers
WHERE last_name = 'Cole'
OR last_name = 'Bush';
```

```
SELECT *
FROM teachers
WHERE school = 'F.D. Roosevelt HS'
AND (salary < 38000 OR salary > 40000);
```

Hacktiv8 DS
Curriculum
Team

External References

Colab Link

Visit Here