Introduction to Structured Query Language

DBMS LAB

Department of Computer Science & Engineering
National Institute of Technology Calicut.

August 27, 2021

Structured Query Language

- Structured Query Language aimed to store, retrieve, and manipulate data in a Relational Database.
- In the early 1970s, IBM sequel language was developed as a part of the system R project at the IBM SanJose Research Laboratory.

Data Definition Language (DDL)

DDL specifies the information about the relation regarding,

- Schema of the relation
- Domain values associated with the attribute
- Integrity Constraints
- Security and Authentication

Domain types in SQL

- ***char*(n)** : fixed-length character string, with user-specified length n.
- ***varchar*(n)** : variable-length character string, with user-specified maximum length n.
- ***int/integer***: an integer (length is machine-dependent).
- ***smallint***: a small integer (length is machine-dependent).
- ***numeric*(p, d)**: a fixed-point number with user-specified precision, consists of p digits (plus a sign), and d of p digits are to the right of the decimal point.
- ***real*** : floating-point or double-precision floating-point numbers, with machine-dependent precision.
- ***float*(n)**: floating-point, with user-specified precision of at least n digits.
- ***date***: a calendar date containing four-digit year, month, and day of the month.
- ***time***: the time of the day in hours, minutes, and seconds.

Create Table Construct

The relation is defined using create table command.
create table relation$_A(A_1\ D_1, A_2\ D_2, A_3\ D_3, ...A_n\ D_n, (Integrity\_Constraint_1), ..$
(Integrity_Constraint$_n$));

relation$_A$ *is the name of the relation.*
A$_i$ *is the attribute names in the relation.*
D$_i$ *is the data type of the attribute.*

Example :
create table student (ID char(5), name varchar(20), age int, address varchar(30),dep varchar(20));

Integrity Constraints

- Not null
- Primary key
- foreign key
- Unique

Example:
create table student (ID char(5), name varchar(20) NOT NULL, age int, address varchar(30), dep varchar(20), PRIMARY KEY(ID),FOREIGN KEY(dep) REFERENCES Department);

DDL Commands

- **DROP**
  - delete objects from the database.
  - ***drop table*** name;
  - example : ***drop table*** student;
- **ALTER TABLE**
  - used to alter the structure (add, delete, or modify columns).
  - **ALTER TABLE** name **ADD** column-name datatype;
    Example : **ALTER TABLE** student **ADD** email char(30);
  - **ALTER TABLE** name **DROP** COLUMN column-name;
    Example : **ALTER TABLE** student **DROP** COLUMN email;
  - **ALTER TABLE** name **MODIFY** COLUMN column-name datatype;
    Example : **ALTER TABLE** student **MODIFY** COLUMN email varchar(30);

DDL Commands

- **TRUNCATE**
  used to remove all records from a table, including all spaces allocated for the records are removed.
  TRUNCATE TABLE tablename;
  Example:**TRUNCATE TABLE** student;
- **RENAME**
  used to rename an object existing in the database.
  RENAME oldtable-name **To** newtable-name ;
  Example: **RENAME** student **To** class;

Data Manipulation Language

- Used to retrieve and manipulate data in a relational database.

- **Insert**

  - Used for inserting a data into a table.

    - **Method 1** Specifying the column and values.
      INSERT INTO table_name (column1, column2, column3, ...) VALUES (value1, value2, value3, ...);
      Example:
      INSERT INTO student (ID, name, age, address, dep); VALUES ('CS12', 'Tom', '21', 'Sreenagar', 'comp sci');
    - **Method 2** Adding values for all the columns of the table
      INSERT INTO table_name VALUES (value1, value2, value3, ...);
      Example:
      INSERT INTO student VALUES ('CS12', 'Tom', '21', 'Sreenagar', 'comp sci');

Data Manipulation Language

- **delete**
  - Used to delete existing records in a table.
  - DELETE FROM table_name WHERE condition;

  Example:

  DELETE FROM student WHERE name=' Tom ';

  DELETE FROM student; - delete all record.
- **Update**
  - Used to modify the existing records in a table.
  - UPDATE table_name SET column1 = value1, column2 = value2,... WHERE condition;

  Example:

  UPDATE student SET age = '22', dep = 'phy' WHERE ID = ' CS12 ';

Basic Query Structure

Basic query structure is like

select $a_1, a_2, a_3....a_n$ **from** $r_1, r_2, ...r_m$ **where** *Predicate*

a_i *represents desired attributes in the result.*

r_i *represent relations.*

Examples:

select ID, name **from** student

select distinct name **from** student - eliminate duplication

select * **from** student - select all attribute

select ID, name, mark/5 **from** student - support arithmetic operation(+, -, *, and /) on attribute

select ID, name, mark/5 **as** avg_mark **from** student

where clause

- specifies the condition that the result must satisfy.

Examples :

- Find all the students from "Compsci" department.

  select name from student **where** dep = 'Compsci'

- Find all the students from "Compsci" department having mark grater than or equal to 300.

  select name from student **where** dep = 'Compsci' **and** marks >= 300

- **between** comparison operator

  - Find the name of all students with a mark between 200 and 400

  select name from student **where** marks **between** 200 **and** 400

String Operation

- String matching Operator for comparison.
  - The percent(%) character matches any sub-string.
    select name **from** student **where** name **like** '%sac%'
  - underscore('_') matches any sub-string

Aggregate Functions

- **avg( )** - average of values
  select avg(age**) from** student;
- **sum( )** - sum of values.
  select sum(marks**) from** student;
- **min( )** - minimum value
  select min(marks**) from** student;
- **max( )** - maximum value
  select max(marks**) from** student;
- **count( )** - number of values.
  select count(distinct name**) from** student; - number of distinct names
  select count(*) from student; - number tuples in the relation.

Aggregate Functions

- **order by** - arranging values in the ascending or descending order. By default it will be in ascending order.
  select ID, name, mark **from** student **order by** mark **desc**;
- **group by** - grouping tuples.
  select dep, **avg(**marks **) as** avg_mark **from** student **group by** dep;
- **having** - predicate in having clause are applied after forming group.
  select dep, **avg(**marks**) as** avg_mark **from** student **group by** dep **having avg(**marks **)** > 70;

Thank You...