Module - II

SQL - Structured Query Language

Lecture - 2

Data Definition Language (DDL)

SQL - DDL

- Introduction
- SQL Data Types
- Create Table Construct
- Integrity Constraints in Create Table

Data Definition Language (DDL)

The SQL Data-Definition Language (DDL) allows the specification of information about relations, including:

- The schema for each relation.
- The type of values associated with each attribute.
- The Integrity constraints
- The set of indices to be maintained for each relation.
- Security and authorization information for each relation.
- The physical storage structure of each relation on disk.

Data Definition Language (DDL)

Data Definition Language (DDL) commands:

1. **CREATE:** This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
2. **DROP:** This command is used to delete objects from the database.
3. **ALTER:** This is used to alter the structure of the database.
4. **TRUNCATE:** This is used to remove all records from a table, including all spaces allocated for the records are removed.
5. **COMMENT:** This is used to add comments to the data dictionary.
6. **RENAME:** This is used to rename an object existing in the database.

Domain Types / Data Types in SQL

1. **`char(n)`**. Fixed length character string, with user-specified length `n`.
2. **`varchar(n)`**. Variable length character strings, with user-specified maximum length `n`.
3. **`int`**. Integer (a finite subset of the integers that is machine-dependent).
4. **`smallint`**. Small integer (a machine-dependent subset of the integer domain type).

Domain Types / Data Types in SQL

5. **`numeric(p,d)`**. Fixed point number, with user-specified precision of `p` digits, with `d` digits to the right of decimal point. (ex., `numeric(3,1)`, allows `44.5` to be stores exactly, but not `444.5` or `0.32`)

6. **`real, double precision`**. Floating point and double-precision floating point numbers, with machine-dependent precision.

7. **`float(n)`**. Floating point number, with user-specified precision of at least `n` digits.

Create Table Construct

An SQL relation is defined using the create table command:

create table r (A_1 D_1, A_2 D_2, ..., A_n D_n,
(integrity-constraint$_1$),
...,
(integrity-constraint$_k$))

- r is the name of the relation
- each A_i is an attribute name in the schema of relation r
- D_i is the data type of values in the domain of attribute A_i

Create Table Construct

Example:

```
create table instructor (
        ID              char(5),
        name            varchar(20),
        dept_name       varchar(20),
        salary          numeric(8,2))
```

Integrity Constraints in Create Table

- Types of integrity constraints:
  - **primary key** (A_1, ..., A_n)
  - **foreign key** (A_m, ..., A_n) **references** r
  - **not null**
- SQL prevents any update to the database that violates an integrity constraint.

Integrity Constraints in Create Table

Example-1:

```
create table instructor (
   ID           char(5),
   name         varchar(20) not null,
   dept_name    varchar(20),
   salary       numeric(8,2),

    primary key (ID),
    foreign key (dept_name) references department);
```

Integrity Constraints in Create Table

Example-2:

```
create table student (
        ID              varchar(5),
        name            varchar(20) not null,
        dept_name       varchar(20),
        tot_cred        numeric(3,0),
        primary key (ID),
        foreign key (dept_name) references department);
```

Integrity Constraints in Create Table

Example-3:

```
create table takes (
        ID                  varchar(5),
        course_id           varchar(8),
        sec_id              varchar(8),
        semester            varchar(6),
        year                numeric(4,0),
        grade               varchar(2),
primary key (ID, course_id, sec_id, semester, year),
foreign key (ID) references  student,
foreign key (course_id, sec_id, semester, year) references
section);
```

Integrity Constraints in Create Table

Example-4:

```
create table course (
        course_id varchar(8),
        title     varchar(50),
        dept_name varchar(20),
        credits   numeric(2,0),
    primary key (course_id),
    foreign key (dept_name) references department);
```

Modifying the Structure of Tables

- **Alter**

  Adding New Columns

  - **alter table** *r* **add** *A D*
    - where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A*.
    - All exiting tuples in the relation are assigned *null* as the value for the new attribute.
  - **alter table** *r* **drop** *A*
    - where *A* is the name of an attribute of relation *r*
    - Dropping of attributes not supported by many databases.

Modifying the Structure of Tables

- **Alter - Adding New Columns**

Example: *Enter a new filed city in the table BRANCH_MSTR*

```
ALTER TABLE BRANCH_MSTR ADD(
        city varchar(8));
```

Modifying the Structure of Tables

- **Alter**

  Dropping a Column from a Table

  - **alter table** *r* **drop column** *A*
    - where *A* is the name of an attribute of relation *r*
    - Dropping of attributes not supported by many databases.

Modifying the Structure of Tables

- **Alter**

Dropping a Column from a Table

Example: *Drop the column city from the table BRANCH_MSTR*

```
ALTER TABLE BRANCH_MSTR DROP COLUMN city;
```

Modifying the Structure of Tables

- **Alter**

Modifying Existing Columns

Syntax:

```
ALTER TABLE <Table_Name> MODIFY (<Column_Name>
<New_datatype>(<New_size>));
```

Modifying the Structure of Tables

- **Alter**

Modifying Existing Columns

Example: *Alter the BRANCH_MSTR table to allow the NAME filed to hold maximum of 50 characters*

```
ALTER TABLE BRANCH_MSTR MODIFY (NAME varchar(50));
```

Modifying the Structure of Tables

- **Renaming Tables**

Syntax:

```
RENAME <Table_Nmae> TO <New_Tablename>;
```

Modifying the Structure of Tables

- **Renaming Tables**

Example: *Change the name of BRANCH_MASTER table to BRANCHES table*

```
RENAME BRANCH_MASTER TO BRANCHES;
```

Modifying the Structure of Tables

- **Truncating Tables**
  - Truncate table empties a table completely, this is equivalent to a DELETE statement that deletes all rows, but there are practical differences under some circumstances.

Modifying the Structure of Tables

- **Truncating Tables**

Truncate table differs from DELETE in the following ways:

- Truncate operations drop and re-create the table, which is must faster than deleting rows one bye one.
- Truncate operations are not transaction-safe (i.e.. An error will occur if an active transaction or an active table lock exists)
- The number of deleted rows are not returned.

Modifying the Structure of Tables

- **Truncating Tables**

Syntax:

```
TRUNCATE TABLE <Table_Name>;
```

Modifying the Structure of Tables

- **Truncating Tables**

Example: *Truncate the table BRANCH_MASTER*

```
TRUNCATE TABLE BRANCH_MASTER;
```

Modifying the Structure of Tables

- **Destroying Tables**
  - Sometimes tables within a particular database become obsolete and need to be discarded. In such situation using DROP TABLE statement with the table name can destroy a specific table.

Modifying the Structure of Tables

- **Destroying Tables**

Syntax:

```
DROP TABLE <Table_Name>;
```

Note:

If a table is dropped all records held within it are lost and cannot be recovered.

Modifying the Structure of Tables

- **Destroying Tables**

Example: *Remove table BRANCH_MASTER along with the data held*

```
DROP TABLE BRANCH_MASTER;
```

Modifying the Structure of Tables

- **Comment**
  - Comments in SQL are similar to comments in other programming languages such as Java, C++, Python, etc.
  - They are primarily used to define a code section for easier understanding.
  - Comments can be a single line, multi-line, or even inline types.

Modifying the Structure of Tables

- **Comment**
  - Single line comment

```
-- this is a single line comment
SELECT * FROM instructor;
```

Modifying the Structure of Tables

- **Comment**
  - Multi-line comment

```
/* this is a multi line comment
SELECT * FROM instructor;*/
SELECT * FROM BRANCH_MASTER;
```

Modifying the Structure of Tables

- **Comment**
  - Inline comment

```
SELECT   CID   FROM   instructor   /*WHERE
CID=101*/;
```