DBMS

24246

[EXPERIMENT - 1]

AIM

TO STUDY BASIC DATA TYPES AND THEIR USES
TO STUDY VARIOUS [DDL] COMMANDS

THEORY

[DATA TYPES]

— [SQL] STRUCTURED QUERY LANGUAGE SUPPORTS VARIOUS DATA TYPES THAT DEFINE DIFFERENT KIND OF VALUES A FIELD CAN STORE IN A DATABASE TABLE

— [NUMERIC]

① INT — USED TO STORE INTEGER VALUES
② FLOAT, DOUBLE — USED TO STORE DECIMALS

— [CHARACTER]

① CHAR (n) - STRING OF FIXED LENGTH "n"
② VARCHAR (n) - STRING OF VARIABLE LENTH WITH MAXIMUM BEING "n"
③ LONG - STRING OF VARIABLE LENGTH
④ RAW - BINARY STRING OF VARIABLE LENGTH

— [DATE AND TYPE]

① DATE - STORES DATE VALUES IN [YYYY-MM-DD]
② TIME - STORES TIME VALUES IN [HH-MI-SS]
③ TIMESTAMP - STORES BOTH DATE & TIME

- [BOOLEAN]

① BOOLEAN - USED TO STORE TRUE OR FALSE

[BINARY LARGE OBJECT]

① TINY BLOB - LENGTH : 255 BYTES
② BLOB - LENGTH : 65,535 BYTES
③ MEDIUMBLOB - LENGTH : 16,777,215 BYTES
④ LARGE BLOB - LENGTH : 4,294,967,295 CHARS

Eg

```
CREATE TABLE EMP (

    id INT,
    f_name VARCHAR (50),
    l_name VARCHAR (50),
    birthdate DATE,
    salary DECIMAL (10,2),
    is_executive BOOLEAN,
    profile_img LARGEBLOB

);
```

NXT Pg →

THEORY [DDL COMMANDS]

- THESE COMMANDS ARE USED CREATE, DELETE, MODIFY OUR DATABASE STRUCTURES

(1) CREATE - USED TO CREATE DATABASE TABLES

SYNTAX

```
CREATE TABLE <table_name> (
    colA    datatype,
    colB    datatype,
    ..............
);
```

Eg.

```
CREATE TABLE student (
    id    INT,
    name  VARCHAR (50),
    age   INT
);
```

NXT Pg →

② ALTER – MODIFY AN EXISTING TABLE

SYNTAX

ALTER TABLE <table_name>
ADD col_name datatype ;

Eg

ALTER TABLE student
ADD gender boolean ;

③ DROP - DROPS A TABLE COMPLETELY

SYNTAX

DROP TABLE <table_name> ;

Eg:

DROP TABLE student;

④ TRUNCATE - REMOVES ALL ROWS FROM TABLE

SYNTAX

TRUNCATE TABLE <table_name>;

Eg. TRUNCATE TABLE student;

PBMS

242466

[EXPERIMENT - 2]

AIM — TO STUDY DML COMMANDS

THEORY — [DML COMMANDS]

— DML COMMANDS ARE USED TO RETRIEVE, INSERT, UPDATE AND DELETE DATA AND ROWS WITHIN A RELATIONAL DATABASE. THIS HELPS IN MANAGING AND MANIPULATING DATA STORED IN A DATABASE

① SELECT - RETRIEVES DATA FROM A TABLE

SYNTAX

— SELECT colA, colB, ...
FROM table_name ;

colA, colB — SPECIFY COLUMNS TO RETRIEVE
table_name — SPECIF TABLE NAME

Eg

SELECT name, job
FROM emp;

NXT Pg
→

(2) INSERT - INSERTS DATA INTO A TABLE

SYNTAX

```
INSERT INTO <table_name>
( colA, colB ..... )
 VALUES
( valA, valB ..... );
```

table_name - SPECIFY TABLE TO INSERT DATA
colA, colB - SPECIFY TABLE COLUMNS
valA, valB - VALUES TO INSERT

Eg.

```
INSERT INTO emp (

    id, name, dept_no, job,
    hire_date , salary, manager , commission

)
  VALUES (

    101, "Jon Doe", 1, "Hire MANAGER",
    '2024-09-01', 80000,
    "Micheal Scott", 5000

);
```

② UPDATE - MODIFIES EXISTING ROWS IN A TABLE

SYNTAX

```
UPDATE   table_name
SET      colA = valA,
         colB = valB
         ..........
WHERE    CONDITION;
```

table_name - SPECIFY TABLE TO UPDATE
colA, colB - SPECIFY COLUMN TO UPDATE
valA, valB - VALUE TO ADD
CONDITION - SPECIFY CONDITION, ROWS THAT MEET CONDITION WILL UPDATE

Eg

```
UPDATE   emp
SET      name = "Jon Doe",
         commission = 111
WHERE
         id = 106;
```

③ DELETE - REMOVE ROW FROM TABLE

SYNTAX - DELETE FROM table_n WHERE ?

Eg. - DELETE FROM emp WHERE id = 104;

242466

[EXPERIMENT -3]

AIM — TO STUDY [DCL] COMMANDS

THEORY — [DCL COMMANDS]

IN SQL, DATA CONTROL LANGUAGE [DCL] COMMANDS ARE USED TO MANAGE ACCESS AND CONTROL PRIVELEGES ON THE DATABASE.

① GRANT

— USED TO GIVE USERS ACCESS TO THE DATABASE OR SPECIFIC OBJECTS LIKE TABLES, VIEWS.

— YOU CAN CONTROL WHO CAN INSERT, UPDATE OR DELETE DATA.

SYNTAX

```
GRANT    priv_name
ON       obj_name
TO       user_name;
```

priv_name — TYPE OF PRIVILEGDE
obj_name — DATABASE OBJECT
user_name — USER TO GRANT

NXT Pg

Eg

```
GRANT   SELECT, INSERT
ON      emp
TO      john;
```

- THE QUERY ABOVE ALLOWS USER "jon" TO RETRIEVE AND INSERT DATA

② REVOKE

- USED TO REMOVE OR TAKE BACK PRIVILEDGES THAT WERE GRANTED TO A USER.

- THIS ENSURES USER CANT DO THOSE OPERATIONS ON THOSE OBJECTS.

SYNTAX

```
REVOKE    priv_name
ON        obj_name
FROM      user_name;
```

priv_name – TYPE OF OPERATION
obj_name – DATABASE OBJECT
user_name – REVOKE FROM USER

Eg.

```
REVOKE    INSERT
ON        emp
FROM      'john';
```

- THIS ENSURES USER "john" CANT INSERT DATA INTO "emp" TABLE.

(3) ROLE

- USED TO CREATE A ROLE / DROP A ROLE

SYNTAX

```
CREATE   ROLE
role_name    , role_pass;

DROP     ROLE
role_name;
```

Eg.

```
CREATE   ROLE    manager_role;

DROP     ROLE    manager_role;
```

DBMS

242466

[EXPERIMENT - 4]

AIM TO STUDY CONSTRAINTS

THEORY [CONSTRAINTS]

- CONSTRAINTS IN SQL ARE RULES APPLIED TO TABLE COLUMNS TO ENSURE THAT DATA ADHERES TO SPECIFIC CONDITIONS.

- THESE HELP IN MAINTAINING INTEGRITY AND ACCURACY OF THE DATABASE.

(1) PRIMARY KEY

- ENSURES THAT EACH ROW IN A TABLE IS UNIQUELY IDENTIFIABLE. A PRIMARY KEY IS COLUMN THAT CANNOT CONTAIN NULL VALUES AND MUST HAVE UNIQUE VALUES.

SYN

```
CREATE TABLE table_name (
colA datatype PRIMARY KEY
.....
);
```

Eg.

```
CREATE TABLE emp (
    id INT PRIMARY KEY,
    name VARCHAR(50)
);
```

(2) FOREIGN KEY

- ENSURES REFRENTIAL INTEGRITY BETWEEN TWO TABLES. A FOREIGN KEY IN ONE TABLE POINTS TO A PRIMARY KEY IN ANOTHER TABLE, ENFORCING A RELATIONSHIP BETWEEN THE TABLES.

SYN

```
CREATE TABLE table_name (

    colA datatype,
    FOREIGN KEY colB REFRENCES
    other_table (other_col)

);
```

NXT Pg →

Eg.

```
CREATE TABLE dept (
    dept_id INT PRIMARY KEY,
    dept_name VARCHAR (50)
);

CREATE TABLE emp (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR (50),
    dept_id INT
    FOREIGN KEY (dept_id) REFRENCES
        dept (dept_id)
);
```

(3) UNIQUE

- ENSURES THAT ALL VALUES IN A COLUMN ARE UNIQUE ACROSS THE TABLE. UNLIKE THE PRIMARY KEY, IT ALLOWS [NULL] VALUES, BUT ONLY ONE NULL PER COLUMN

NXT Pg →

SYN

```
CREATE TABLE table_name (
    colA datatype UNIQUE
);
```

Eg.

```
CREATE TABLE users (
    id INT PRIMARY KEY,
    email VARCHAR (255) UNIQUE
);
```

(4) NOT NULL

- ENSURES THAT A COLUMN CANNOT CONTAIN NULL VALUES. THIS CONSTRAINT CAN BE USED TO ENFORCE THAT THE CERTAIN COLUMN MUST HAVE A VALUE

SYN

```
CREATE TABLE table_name (
    colA datatype NOT NULL
);
```

Eg.

```
CREATE TABLE products (
    id INT PRIMARY KEY,
    name VARCHAR (50) NOT NULL
);
```

(5) CHECK

— ENSURES THAT ALL VALUES IN A COLUMN SATISFY A SPECIFIC CONDITION. THIS IS USED TO ENFORCE DOMAIN INTEGRITY

SYN

```
CREATE TABLE table_name (
    colA datatype CHECK (CONDITION)
);
```

Eg.

```
CREATE TABLE EMPLOYEES (
    id INT PRIMARY KEY
    sal DECIMAL (10,2) CHECK (sal > 0)
);
```

⑥ DEFAULT

- PROVIDES A DEFAULT VALUE FOR A COLUMN IF NO VALUE IS GIVEN DURING INERT OPERATION

SYN

```
CREATE TABLE table_name (

    colA datatype DEFAULT def_val

);
```

Eg.

```
CREATE TABLE orders (

    id INT PRIMARY KEY,
    order_date DATE DEFAULT CURRENT_DATE

);
```

DBMS

24246

[EXPERIMENT - 5]

AIM — TO STUDY AGGREGATE FUNCTIONS

THEORY — [AGGREGATE FUNCTIONS]

— THESE FUNCTIONS OPERATE ON A SET OF VALUES AND RETURN A SINGLE VALUE. THESE ARE COMMONLY USED TO IN [SQL] QUERIES TO SUMMARIZE OR AGGREGATE DATA FROM A TABLE.

① COUNT - RETURNS NUMBER OF ROWS

SYN — SELECT COUNT(col_name) FROM table_name;

Eg — SELECT COUNT(*) FROM emp;

② SUM - RETURNS SUM OF NUMERIC VALUES

SYN — SELECT SUM (col_name) FROM table_name;

Eg. — SELECT SUM ~~(*)~~ (salary) FROM emp;

③ AVG - RETURNS AVERAGE OF NUMERIC VALU

SYN — SELECT AVG (col_name) FROM table_name;

Eg. — SELECT AVG (salary) FROM emp;

21

(4) MIN - RETURNS SMALLEST VALUE IN A SET

SYN SELECT MIN(colname) FROM table-name;

Eg. SELECT MIN (salary) FROM emp;

(5) MAX - RETURNS BIGGEST VALUE IN A SET

SYN SELECT MAX (col_name) FROM table_name;

Eg. SELECT MAX (salary) FROM emp;

(6) GROUP_CONCAT - RETURNS A CONCATED STRING OF VALUES FROM A GROUP

SYN SELECT GROUP_CONCAT(col_name SEPERATOR "sep") FROM table_name;

Eg.

Eg. SELECT GROUP_CONCAT(name SEPERATION ",") FROM emp;

242466

EXPERIMENT - 6

AIM TO STUDY GROUP BY, HAVING AND WHERE CLAUSES

THEORY

CLAUSES

① GROUP BY

— GROUPS ROWS THAT HAVE SAME VALUES IN SPECIFIED COLUMNS INTO SUMMARY ROWS, LIKE FINDING SUM OR AVERAGE OF A GROUP. IT IS COMMONLY USED WITH AGGREGATE FUNCTIONS

SYN

```
SELECT   colA, colB   agg_func(colC)
FROM     table_name
WHERE    CONDITION
GROUP BY   colA, colB;
```

Eg.

```
SELECT   dept_id , SUM(salary)
FROM     emp
GROUP    BY   deptid ;
```

— SPECIFY COLUMNS TO GROUP BY AND USE AGGREGATE FUNCTIONS TO FURTHER QUERY

NXT Pg →

(2) HAVING

- USED TO FILTER RECORDS AFTER THE "GROUP BY" CLAUSE HAS BEEN APPLIED. WORKS SIMILARLY TO "WHERE" CLAUSE BUT IS USED FOR GROUPED DATA.

SYN

```
SELECT colA, agg-func (colB)
FROM table_name
WHERE condition
GROUP BY colA
HAVING agg-func(colB) condition ;
```

Eg.

```
SELECT dept_id, SUM (salary)
FROM emp
GROUP BY deptid
HAVING SUM (salary) > 150000 ;
```

- THE ABOVE QURIES FILTERS THE DEPARTMENTS WHERE TOTAL SALARY IS GREATER THAN 1,50,000

- FIRST IT GROUPS THE ROWS BY "dept_id" THE HAVE CLAUSE FILTERS THE RESULT.

NXT Pg →

(3) ORDER BY

— THIS CLAUSE IS USED TO SORT THE RESULT SET IN EITHER ASENDING OR DESCENDING ORDER. ITS COMMONLY USED TO SORT DATA BASED ON ONE OR MORE COLUMNS.

SYN

```
SELECT  colA , colB
FROM    table_name
WHERE   condition
ORDER   BY  colA [ASC | DESC];
```

Eg.

```
SELECT  name, salary
FROM    emp
ORDER   BY  salary  DESC;
```

— THE ABOVE QUARY SORTS THE RECORDS BY THEIR SALARY IN A DESCENDING ORDER