Information Technology Institute

PostgreSQL

Day02

Prepared by:

Noha Shehab
Teaching Assistant
Information Technology Institute (ITI)

Table of contents

Column data type

- Other data types that can be used with PostgreSQL
  - Geometric Types (point, line, circle)
  - Network Address Types (IP, Mac)
  - XML Types
  - JSON Types
  - Binary Data Types
  - **Composite data type**

Composite data type

- **it is essentially just a list of field names and their data types**

```
CREATE TYPE employee_info AS (

name text,

National_id integer);
```

```
iti=# create type gender as enum ('male', 'female');
CREATE TYPE
iti=# create type employee_info as ( name text, national_id integer, emp_gender gender);
CREATE TYPE
iti=#
```

Composite data type

- **Use the created type within your table**

```
iti=# create table manager (mgr_id serial primary key, mgr_info employee_info, dept_no integer);
CREATE TABLE
iti=# select * from manager;
 mgr_id | mgr_info | dept_no
--------+----------+---------
(0 rows)
```

- Then insert a new record, use the **Row keyword** to insert the data from the **employee_info type**

```
iti=# insert into manager values (1,row('Mostafa', 76786687, 'male'), 5);
INSERT 0 1
iti=# select * from manager;
 mgr_id |         mgr_info          | dept_no
--------+---------------------------+---------
      1 | (Mostafa,76786687,male) |       5
(1 row)
```

Composite data type

- Select the data from the table based on the name of the manager....

```
iti=# select * from manager where (mgr_info).name = 'Noha';
 mgr_id |        mgr_info        | dept_no
--------+------------------------+---------
      3 | (Noha,76786687,female) |       4
(1 row)
```

- You can update the mgr_info or part of it using the update keyword.

```
iti=# update manager set mgr_info.national_id = 77777 where (mgr_info).name ='Mostafa';
UPDATE 1
iti=# select * from manager;
 mgr_id |       mgr_info       | dept_no
--------+----------------------+---------
      2 | (Ali,666666,male)    |       3
      3 | (Noha,666666,female) |       4
      1 | (Mostafa,77777,male) |       5
(3 rows)
```

(DML)Data manipulation language

- Insert
- Update
- Delete
- Select

Insert

- **Basic syntax of INSERT INTO statement is as follows:**

```
INSERT INTO TABLE_NAME [(column1, column2, column3,...columnN)]

VALUES (value1, value2, value3,...valueN);
```

```
iti=# insert into manager (mgr_id, mgr_info, dept_no) values (2,row('Ali', 76786687, 'male'), 3);
INSERT 0 1
iti=#
```

Update

- **The basic syntax of UPDATE query with WHERE clause is as follows:**

UPDATE table_name

SET column1 = value1, column2 = value2...., columnN =valueN

[**WHERE** condition];

```
iti=# update manager set mgr_info.national_id = 77777 where (mgr_info).name ='Ali';
UPDATE 1
```

```
iti=# update manager set mgr_info.national_id = 77777, dept_no= 10 where (mgr_info).name ='Ali';
UPDATE 1
iti=# select * from manager;
 mgr_id |        mgr_info         | dept_no
--------+-------------------------+---------
      3 | (Mostafa,76786687,male) |       3
      2 | (Ali,77777,male)        |      10
(2 rows)
```

Delete

- The basic syntax of DELETE query with WHERE clause is as follow

```
Delete from table_name

where [conditions]
```

```
iti=# select * from manager;
 mgr_id |       mgr_info       | dept_no
--------+----------------------+---------
      2 | (Ali,666666,male)    |       3
      3 | (Noha,666666,female) |       4
      1 | (Mostafa,77777,male) |       5
(3 rows)


iti=# delete from manager where mgr_id = 2;
DELETE 1
iti=# select * from manager;
 mgr_id |       mgr_info       | dept_no
--------+----------------------+---------
      3 | (Noha,666666,female) |       4
      1 | (Mostafa,77777,male) |       5
(2 rows)
```

Truncate

- TRUNCATE TABLE command is used to **delete** complete data from an existing table.
- It has the same effect as an DELETE on each table, but since it **does not actually scan the tables, it is faster**

TRUNCATE TABLE table_name;

```
iti=# truncate table manager;
TRUNCATE TABLE
iti=# select * from manager;
 mgr_id | mgr_info | dept_no
--------+----------+---------
(0 rows)
```

Drop

- To delete the complete table

  Drop TABLE [if exists] table_name;

- If exists to avoid receiving exception if the table doesn't exist.

```
iti=# create table staff (mgr_id serial primary key, mgr_info employee_info, dept_no integer);
CREATE TABLE
iti=# drop table if exists staff;
DROP TABLE
```

Select

- PostgreSQL SELECT statement is used to fetch the data from a database table
- The basic syntax of SELECT statement is as follows:
    - **SELECT column1, column2, columnN FROM table_name;**

- If you want to fetch all the fields available in the table then you can use the following syntax:
    - **SELECT * FROM table_name;**

Select

The complete syntax of select as following:

```
SELECT column1, column2
FROM table1
[ WHERE conditions ]
[ GROUP BY column ]
[ HAVING conditions ]
[ ORDER BY column ]
[LIMIT no of rows]
```

Where clause

- WHERE clause is used to specify a condition while fetching the data from single table or joining with multiple tables.

- Example

```
SELECT *
FROM manger
WHERE mgr_id != 1 AND dept_no >= 4;
```

Where clause and operators

- Operators are used to specify conditions in a PostgreSQL statement and to serve as conjunctions for multiple conditions in a statement.
- **It can be classified into:**
  - Arithmetic operators
  - Comparison operators
  - Logical operators
  - Like operator
  - Other operators. jkhjkkh@jhjjhg.fff

Arithmetic operators

Operator	Description	Example	Result
+	addition	2 + 3	5
-	subtraction	2 - 3	-1
*	multiplication	2 * 3	6
/	division (integer division truncates the result)	4 / 2	2
%	modulo (remainder)	5 % 4	1
^	exponentiation (associates left to right)	2.0 ^ 3.0	8
|/	square root	|/ 25.0	5
||/	cube root	||/ 27.0	3
!	factorial	5 !	120
!!	factorial (prefix operator)	!! 5	120

Comparison operators

Operator	Description
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
=	equal
<> or !=	not equal

Logical Operators

Operator	Description
And	The AND operator allows the existence of multiple conditions in a PostgresSQL statement's WHERE clause.
Or	The OR operator is used to combine multiple conditions in a PostgresSQL statement's WHERE clause.
Not	The NOT operator reverses the meaning of the logical operator with which it is used. Eg. NOT EXISTS, NOT BETWEEN, NOT IN etc. This is negate operator.

Like Operator

Operator	Description
WHERE SALARY LIKE '200%'	Finds any values that start with 200
WHERE SALARY LIKE '%2'	Finds any values that end with 2.
WHERE SALARY LIKE '_2%3'	Finds any values that have a 2 in the second position and end with a 3

Other operators

- SELECT * FROM COMPANY WHERE AGE **IS NULL**;
- SELECT * FROM COMPANY WHERE AGE **NOT IS NULL**;
- SELECT * FROM COMPANY WHERE AGE **IS NOT NULL**;
- SELECT * FROM COMPANY WHERE **AGE IN (25, 27);**
  integer
- SELECT * FROM COMPANY WHERE AGE **NOT IN (25, 27);**
- SELECT * FROM COMPANY WHERE **AGE BETWEEN 25 AND 27**;
- SELECT * FROM Orders WHERE OrderDate **BETWEEN #07/04/1996# AND #07/09/1996#;**

Expressions

- The expression is a combination of one or more values with previous operators.
- Examples:
  - SELECT * FROM COMPANY WHERE SALARY = 10000;
  - SELECT * FROM COMPANY **WHERE (SALARY + 6) = 1005**;
  - SELECT * FROM COMPANY **WHERE (SALARY * Age) >= 1005**;.

Aggregation Functions

- Compute a single result from a set of input values Such as: **COUNT()**, **MAX()**, **MIN()**, **AVG()**, **SUM()**.

- Examples
  - SELECT MAX(salary) FROM COMPANY;
  - SELECT MIN(salary) FROM COMPANY;
  - SELECT AVG(SALARY) FROM COMPANY;
  - SELECT SUM(salary) FROM company;
  - SELECT COUNT(salary) FROM COMPANY;

GROUP BY

- The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.
- This is done to eliminate redundancy in the output and/or compute aggregates that apply to these groups:

```
SELECT NAME, SUM(SALARY)

FROM COMPANY

GROUP BY dept;
```

HAVING

- The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

```
SELECT NAME

FROM COMPANY

GROUP BY name

HAVING count(name) < 2;
```

Order by

- **Order by** is used to sort the data in ascending or descending order, based on one or more columns:

```
SELECT *

FROM COMPANY

ORDER BY AGE ASC; [default] or DESC
```

Limit

- LIMIT is used to limit the data amount returned by the SELECT statement, OFFSET rows are skipped before starting to count the LIMIT rows that are returned.

  - **SELECT** * FROM COMPANY **LIMIT 4**;
  - **SELECT** * FROM COMPANY **LIMIT 3 OFFSET 2**;

SELECT DISTINCT

- SELECT DISTINCT is used to eliminate all the duplicate records and fetching only unique records:
  - SELECT **DISTINCT** name FROM COMPANY;
  - SELECT **DISTINCT name, age** FROM COMPANY;

Case expression

- The CASE expression is a generic conditional expression, similar to if/else statements in other programming languages

  CASE WHEN condition **THEN** result

  [WHEN condition THEN result]

  [ELSE result]

  END

- **SELECT column1,column2,[CASE Expression] FROM table**

Case expression

```
iti=# select * from numbers;
 num
-----
   1
   2
   3
   4
(4 rows)
```

Example:

```
SELECT num,
        CASE
                WHEN num =1 THEN 'one'
                WHEN num =2 THEN 'two'
                ELSE 'other'
        END
FROM numbers;
```

```
iti=# select num , case when num=1 then 'one' when num =2 then 'two' else 'other' End from numbers;
 num | case
-----+-------
   1 | one
   2 | two
   3 | other
   4 | other
(4 rows)
```

Union

- UNION clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.
- Each UNION query must have the same number of columns

```
SELECT EMP_ID, NAME, DEPT FROM COMPANY

UNION

SELECT EMP_ID, NAME, DEPT FROM COMPANY1
```

CONSTRAINTS

- Constraints are the rules enforced on data columns on table.
- These are used to prevent invalid data from being entered into the database. exists.
- Examples
    - Primary key
    - Not null
    - Unique
    - Foreign key
    - Check

Not null

- By default, a column can hold NULL values.
- If you do not want a column to have a NULL value, then you need to define such constraint on this column specifying that NULL is now not allowed for that column

```
CREATE TABLE Employees(

ID INT NOT NULL,

NAME TEXT NOT NULL,

AGE INT NOT NULL,

SALARY INT);
```

Unique

- UNIQUE Constraint prevents two records from having identical values in a particular column.

```
CREATE TABLE Employee(
ID INT PRIMARY KEY,      // unique and not null
NAME TEXT,
AGE INT,
phone Int UNIQUE,
SALARY float DEFAULT 50000.00);
```

Foreign key

- A foreign key constraint specifies that the values in a column (or a group of columns) must match the values appearing in some row of another table.
- We say this maintains the **referential integrity** between two related tables.

```
CREATE TABLE departments(
id INT PRIMARY KEY NOT NULL,
dept_name CHAR(50) NOT NULL,
mgr_id INT references employees(id)  on delete cascade );
```

Check

- The **CHECK Constraint** enables a condition to check the value being entered into a record.
- If the condition evaluates to false, the record violates the constraint and isn't entered into the table.

```
CREATE TABLE Employees(

ID INT NOT NULL Primary key,

NAME TEXT NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR(50),

SALARY REAL CHECK(SALARY > 999));
```

Data Definition Language (DDL)

- Create
- Drop
- Alter

Alter

Alter is used to change the definition of a table as follows

ALTER TABLE table_name action [, ...]

Examples:

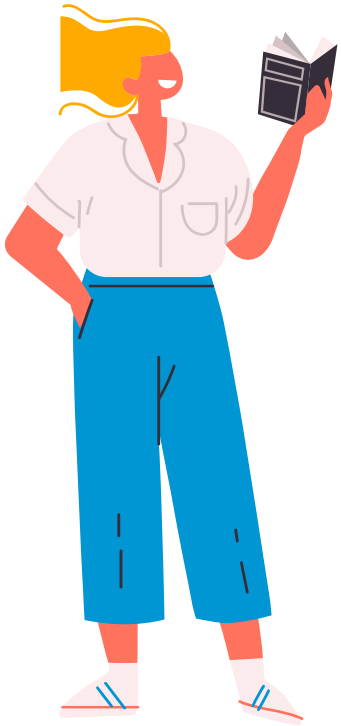
- ALTER TABLE table_name **RENAME TO** new_name
- ALTER TABLE table_name **ADD column** data_type
- ALTER TABLE table_name **DROP column_name**
- ALTER TABLE table_name ALTER column **SET DATA TYPE data_type**
- ALTER TABLE table_name **RENAME COLUMN column TO new_column**
- **ALTER TABLE** table_name
  ADD CONSTRAINT fk_name **FOREIGN KEY** (col1)
  REFERENCES foreign_table (foreign_field)
  ON DELETE CASCADE;

Sub queries

- A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
- Subqueries that return more than one row can only be used with multiple value operators, such as IN, NOT IN operator

```
SELECT * FROM COMPANY
WHERE ID IN (SELECT ID
FROM COMPANY_bkp WHERE SALARY > 45000);

INSERT INTO COMPANY_BKP (SELECT * FROM COMPANY);
```



Lab time

Lab 02

1. Add gender column for the student table[Enum]. It holds two value (male or female).
2. Add birth date column for the student table.
3. Delete the name column and replace it with two columns first name and last name.
4. Delete the address and email column and replace it with contact info (Address, email) as object/Composite Data type.
5. Change any Serial Datatype at your tables to smallInt
6. Add/Alter foreign key constrains in Your Tables.
7. Insert new data in all Tables.
8. Display all students' information.

Lab 02

9. Display male students only.
10. Display the number of female students.
11. Display the students who are born before 1992-10-01.
12. Display male students who are born before 1991-10-01.
13. Display subjects and their max score sorted by max score.
14. Display the subject with highest max score
15. Display students' names that begin with A.
16. Display the number of students' their name is "Mohammed"
17. Display the number of males and females.

Lab 02

18. Display the repeated first names and their counts if higher than 2.
19. Display the all Students and track name that belong to it
20. (Bouns) Display students' names, their score and subject name.

Break time

Thanks

Noha
nshehab@iti.gov.eg