Project

on

Postgresql

Neorays Software Solution

Employee\_Project

-Dinesh Pranav N

|  |  |
| --- | --- |
| CONTENT  Introduction  1.Data Types  1.1 Numeric Type  1.2Character Type  1.3Date/Time Type  1.4Array Type  1.5Json  2.Database  2.1creating Database  2.2drop Database  2.3select Database  2.4connect Database  3.Table Creation  4.Schema  5.Insert Query  6.Select Query  7.Operator  7.1AND,OR  8.Clauses  8.1Where  8.2Like  8.3Limit  8.4Groupby  8.5Orderby  8.6Having  8.9Update  8.10Delete  9.Distinct Keyword | 10.Constraints  10.1Not Null  10.2Unique  10.3Primary Key  10.4Foreign key  10.5Check  11.Joins  11.1Cross Join  11.2Inner Join  11.3Left and Right Join  11.4Full Join  12.Indexes,Alter command, Views  13.Subqueries |

INTRODUCTION

**To DBMS:**

**Database:** Database is a repository, which stores data.

**DBMS:** DBMS is a Computer-Application which handles database,

It provides Manipulation of Data in Database, and Ensures data security (Unauthorized access is restricted).

There are many types of DBMS, prominently used is **RDBMS** and **ORDBMS**.

RDBMS:

RDBMS is referred as relational database management system. It is an implementation of DBMS.

* It stores the data in Tabular format.
* It uses primary key and foreign key to relate the tables each other.
* RDBMS supports normalization.
* Crud Operations held using SQL in Editor, SQL is Structured Query Language which allows you to write query for crud operations for manipulation of data.

**ORDBMS**:

ORDBMS is object relational database management system.

* ORDBMS has same features as RDBMS with even more enhancement.
* ORDBMS is rarely used in java Applications.
* Developers tend to use RDBMS and ORM external framework like hibernate, ibatis, etc...
* Today almost all databases like Oracle, Mysql, Db2etc.are ORDBMS, but we use only RDBMS.

**To PostgreSQL:**

PostgreSQL is an Object-relational database management system (ORDBMS) with an emphasis on extensibility and standards compliance.

Its primary functions are to store data securely and return that data in response to requests from other software applications.

It can handle workloads ranging from small single-machine applications to large internet-facing applications with many concurrent users; on macOS server, postgresql is the default database.

PostgreSQL is developed by the PostgreSQL Global Development Group, a diverse group of many companies and individual contributors. It is free and open-source, released under the terms of the PostgreSQL License.

**DATA TYPES**

**Numeric Types**

|  |  |  |  |
| --- | --- | --- | --- |
| smallint | 2 bytes | small-range integer | -32768 to +32767 |
| integer | 4 bytes | typical choice for integer | -2147483648 to +2147483647 |
| double precision | 8 bytes | variable-precision, inexact | 15 decimal digits precision |
| serial | 4 bytes | auto incrementing integer | 1 to 2147483647 |

**Character Types**

**Varchar (n) variable**-length with limit

**Character (n), char (n) fixed**-length, blank padded

**Text** variable unlimited length

**Date/Time Types**

Date ()

Time ()

Timestamp ()

Time () with time zone

Boolean ()

**JSON Type**

The *json* data type can be used to store JSON (JavaScript Object Notation) data. Such data can also be stored as *text*, but the *json* data type has the advantage of checking that each stored value is a valid JSON value. There are also related support functions available

**Array Type**

PostgreSQL gives the opportunity to define a column of a table as a variable length multidimensional array. Arrays of any built-in or user-defined base type, enum type, or composite type can be created.

**Declaration of Arrays**

Array type can be declared as

CREATE TABLE savings (

name text,

Saving\_per\_quarter integer [],

Scheme text [][]

);

**Inserting values**

INSERT INTO savings values (‘aaa’,’{1400, 2000, 2000}’,’ {{“food”,”property”,”money”}}’);

**Accessing Arrays**

SELECT name FROM savings WHERE savings\_per\_quater [1]>savings\_per\_quater[2];

**Searching Arrays**

SELECT \* FROM savings where savings\_per\_quater [1]=10000

**CREATIND DATABASE**

Syntax

CREATE DATABASE database name

**COMMAND “\l”**

You can check the list of databases using **\l** COMMAND.

**DROP DATABASE**

DROP DATABASE [IF EXISTS] name

**SELECT Database**

Command used to connect with database is “**\c**”

POSTGRES=#\C database name

**CREATE Table**

Syntax

CREATE TABLE table name (

Column1 data type,

Column2 data type,

Column3 data type,

.....

Column data type,

PRIMARY KEY (one or more columns)

);

**Schema**

A **schema** is a named collection of tables. A schema can also contain views, indexes, sequences, data types, operators, and functions.

Syntax

CREATE SCHEMA name;

Syntax to Create Table in Schema

CREATE TABLE myschema.mytable (

..);

**INSERT Query**

INSERT INTO TABLE\_NAME (column1, column2, column3, column)

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

**SELECT Query**

Syntax

SELECT column1, column2, column FROM table name;

**Operators**

* Arithmetic operators
* Comparison operators
* Logical operators
* Bitwise operators

|  |  |  |
| --- | --- | --- |
| + | Addition - Adds values on either side of the operator |  |
|  | Subtraction - Subtracts right hand operand from left hand operand |  |
| \* | Multiplication - Multiplies values on either side of the operator |  |
| / | Division - Divides left hand operand by right hand operand |  |
| % | Modulus - Divides left hand operand by right hand operand and returns remainder |  |
| ^ | Exponentiation - This gives the exponent value of the right hand operand |  |
| |/ | Square root |  |
| ||/ | Cube root |  |
| ! | Factorial |  |

**The AND Operator**

Syntax

SELECT column1, column2, columnN

FROM table name

WHERE [condition1] AND [condition2]...AND [conditionN];

**The OR Operator**

Syntax

SELECT column1, column2, columnN

FROM table name

WHERE [condition1] OR [condition2]...OR [conditionN]

**Clauses**

**WHERE clause:**

Syntax

SELECT column1, column2, columnN

FROM table name

WHERE [search condition]

**LIKE Clause:**

There are two wildcards used in conjunction with the LIKE operator −

* The percent sign (%)
* The underscore (\_)

Syntax

SELECT FROM table name

WHERE column LIKE 'XXXX%'

Or

SELECT FROM table name

WHERE column LIKE '\_XXXX'

**LIMIT Clause**

Syntax

SELECT column1, column2, columnN

FROM table name

LIMIT [no of rows]

The following is the syntax of LIMIT clause when it is used along with OFFSET clause −

SELECT column1, column2, columnN

FROM table name

LIMIT [no of rows] OFFSET [row num]

LIMIT and OFFSET allow you to retrieve just a portion of the rows that are generated by the rest of the query.

**ORDER BY Clause**

**ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns.

Syntax

SELECT column-list

FROM table name

[WHERE condition]

[ORDER BY column1, column2, .. columnN] [ASC | DESC];

**GROUP BY**

The PostgreSQL **GROUP BY** clause is used in collaboration with the SELECT statement to group together those rows in a table that have identical data.

Syntax

SELECT column-list

FROM table name

WHERE [conditions]

GROUP BY column1, column2....columnN

ORDER BY column1, column2....columnN

**HAVING Clause**

The HAVING clause allows us to pick out particular rows where the function's result meets some condition.

HAVING clause places conditions on groups created by the GROUP BY clause.

Syntax

SELECT

FROM

WHERE

GROUP BY

HAVING

ORDER BY

**UPDATE Query**

**UPDATE** Query is used to modify the existing records in a table. You can use WHERE clause with UPDATE query to update the selected rows. Otherwise, all the rows would be updated.

Syntax

UPDATE table name

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

WHERE [condition];

**DELETE Query**

**DELETE** Query is used to delete the existing records from a table. You can use WHERE clause with DELETE query to delete the selected rows. Otherwise, all the records would be deleted.

Syntax

DELETE FROM table name

WHERE [condition];

**DISTINCT Keyword**

**DISTINCT** keyword is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.

Syntax

SELECT DISTINCT column1, column2 ...columnN

FROM table name

WHERE [condition]

**CONSTRAINTS**

Constraints are the rules enforced on data columns on table. These are used to prevent invalid data from being entered into the database. This ensures the accuracy and reliability of the data in the database.

The following are commonly used constraints available in PostgreSQL.

* **NOT NULL Constraint** − Ensures that a column cannot have NULL value.
* **UNIQUE Constraint** − Ensures that all values in a column are different.
* **PRIMARY Key** − uniquely identifies each row/record in a database table.
* **FOREIGN Key** − Constrains data based on columns in other tables.
* **CHECK Constraint** − The CHECK constraint ensures that all values in a column satisfy certain conditions.

**NOT NULL Constraint**

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.

Syntax

CREATE TABLE COMPANY1 (

ID INT PRIMARY KEY NOT NULL,

NAME TEXT NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (50),

SALARY REAL

);

**UNIQUE Constraint**

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

Syntax

CREATE TABLE COMPANY1 (

ID INT PRIMARY KEY UNIQUE,

NAME TEXT UNIQUE,

AGE IN NOT NULL,

ADDRESS CHAR (50),

SALARY REAL

);

**PRIMARY KEY Constraint**

The PRIMARY KEY constraint uniquely identifies each record in a database table. There can be more UNIQUE columns, but only one primary key in a table. Primary keys are important when designing the database tables. Primary keys are unique ids.

A primary key is a field in a table, which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values.

A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a **composite key**.

Example

CREATE TABLE COMPANY4 (

ID INT PRIMARY KEY NOT NULL,

NAME TEXT NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (50),

SALARY REAL

);

**FOREIGN KEY Constraint**

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. They are called foreign keys because the constraints are foreign; that is, outside the table. Foreign keys are sometimes called a referencing key.

Example

/\* PRIMARY KEY \*/

CREATE TABLE COMPANY6 (

ID INT PRIMARY KEY NOT NULL,

NAME TEXT NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (50),

SALARY REAL);

/\* FOREIGN KEY \*/

CREATE TABLE DEPARTMENT1 (

ID INT PRIMARY KEY NOT NULL,

DEPT CHAR (50) NOT NULL,

EMP\_ID INT references COMPANY6 (ID)

);

**CHECK Constraint**

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 is not entered into the table.

Example

CREATE TABLE COMPANY5 (

ID INT PRIMARY KEY NOT NULL,

NAME TEXT NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (50),

SALARY REAL CHECK (SALARY > 0)

);

**JOINS**

**Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Join Types in PostgreSQL are −

* The CROSS JOIN
* The INNER JOIN
* The LEFT OUTER JOIN
* The RIGHT OUTER JOIN
* The FULL OUTER JOIN

**The CROSS JOIN**

A CROSS JOIN matches every row of the first table with every row of the second table. If the input tables have x and y columns, respectively, the resulting table will have x+y columns. Because CROSS JOINs have the potential to generate extremely large tables, care must be taken to use them only when appropriate.

Example

SELECT EMP\_ID, NAME, DEPT FROM COMPANY CROSS JOIN DEPARTMENT

**The INNER JOIN**

An INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows, which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of table1 and table2 are combined into a result row.

Example

SELECT EMP\_ID, NAME, DEPT FROM COMPANY INNER JOIN DEPARTMENT ON COMPANY.ID=DEPARTMENT.EMP\_ID

**The LEFT OUTER JOIN**

In case of LEFT OUTER JOIN, an inner join is performed first. Then, for each row in table T1 that does not satisfy the join condition with any row in table T2, a joined row is added with null values in columns of T2. Thus, the joined table always has at least one row for each row in T1.

Example

SELECT EMP\_ID, NAME, DEPT FROM COMPANY LEFT OUTER JOIN DEPARTMENT ON COMPANY.ID=DEPARTMENT.EMP\_ID

**The RIGHT OUTER JOIN**

First, an inner join is performed. Then, for each row in table T2 that does not satisfy the join condition with any row in table T1, a joined row is added with null values in columns of T1. This

Is the converse of a left join; the result table will always have a row for each row in T2.

Example

SELECT EMP\_ID, NAME, DEPT FROM COMPANY RIGHT OUTER JOIN DEPARTMENT ONCOMPANY.ID=DEPARTMENT.EMP\_ID

**The FULL OUTER JOIN**

First, an inner join is performed. Then, for each row in table T1 that does not satisfy the join condition with any row in table T2, a joined row is added with null values in columns of T2. In addition, for each row of T2 that does not satisfy the join condition with any row in T1, a joined row with null values in the columns of T1 is added.

Example

SELECT EMP\_ID, NAME, DEPT FROM COMPANY FULL OUTER JOIN DEPARTMENT ONCOMPANY.ID=DEPARTMENT.EMP\_ID

**INDEXES**

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.

Syntax

CREATE INDEX name ON table name;

**ALTER TABLE Command**

**ALTER TABLE** command is used to add, delete or modify columns in an existing table.

You would also use ALTER TABLE command to add and drop various constraints on an existing table.

Syntax

The basic syntax of **ALTER TABLE** to add a new column in an existing table is as follows −

ALTER TABLE table name ADD column\_name data type;

The basic syntax of ALTER TABLE to **DROP COLUMN** in an existing table is as follows −

ALTER TABLE table name DROP COLUMN column\_name;

The basic syntax of ALTER TABLE to change the **DATA TYPE** of a column in a table is as follows −

ALTER TABLE table\_name ALTER COLUMN column\_name TYPE data type;

The basic syntax of ALTER TABLE to add a **NOT NULL** constraint to a column in a table is as follows −

ALTER TABLE table\_name MODIFY column\_name data type NOT NULL;

The basic syntax of ALTER TABLE to **ADD UNIQUE CONSTRAINT** to a table is as follows −

ALTER TABLE table\_name

ADD CONSTRAINT MyUniqueConstraint UNIQUE (column1, column2...);

The basic syntax of ALTER TABLE to **ADD CHECK CONSTRAINT** to a table is as follows −

ALTER TABLE table\_name

ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);

The basic syntax of ALTER TABLE to **ADD PRIMARY KEY** constraint to a table is as follows −

ALTER TABLE table\_name

ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);

The basic syntax of ALTER TABLE to **DROP CONSTRAINT** from a table is as follows −

ALTER TABLE table\_name

DROP CONSTRAINT MyUniqueConstraint;

**VIEWS**

A view can contain all rows of a table or selected rows from one or more tables. A view can be created from one or many tables, which depends on the written PostgreSQL query to create a view.

Views, which are kind of virtual tables, allow users to do the following −

* Structure data in a way that users or classes of users find natural or intuitive.
* Restrict access to the data such that a user can only see limited data instead of complete table.
* Summarize data from various tables, which can be used to generate reports.

Creating Views

The basic CREATE VIEW syntax is as follows −

CREATE [TEMP | TEMPORARY] VIEW view\_name AS

SELECT column1, column2.....

FROM table\_name

WHERE [condition];

Dropping Views

DROP VIEW view\_name

**Sub queries**

Sub queries with the SELECT Statement

Sub queries are most frequently used with the SELECT statement. The basic syntax is as follows −

SELECT column\_name [, column\_name]

FROM table1 [, table2]

WHERE column\_name OPERATOR

(SELECT column\_name [, column\_name]

FROM table1 [, table2]

[WHERE])

Sub queries with the INSERT Statement

The basic syntax is as follows −

INSERT INTO table\_name [(column1 [, column2])]

SELECT [\*|column1 [, column2] ]

FROM table1 [, table2]

[WHERE VALUE OPERATOR]

Sub queries with the UPDATE Statement

The basic syntax is as follows −

UPDATE table

SET column name = new value

[WHERE OPERATOR [VALUE]

(SELECT COLUMN\_NAME

FROM TABLE\_NAME)

[WHERE)]

Sub queries with the DELETE Statement

DELETE FROM TABLE\_NAME

[WHERE OPERATOR [VALUE]

(SELECT COLUMN\_NAME

FROM TABLE\_NAME)

[WHERE)]