SQL

Feb 17, 2025

AGENDA

- SQL
- Data Definition Language
- Data Manipulation Language
- Data Query Language

STRUCTURED QUERY LANGUAGE

- It is NOT much like other programming languages
- It is NOT PROCEDURAL
- It does not process one record at a time, rather, it is a SET processing language
- All inputs to SQL are tables
- The output from a query is a table
- Output from a query referred to as the "Answer Set"
- Some queries may produce "interim" temporary answer sets

STRUCTURED QUERY LANGUAGE

- It is a relatively simple language – brief syntax, few commands
- It is a relatively powerful language – a FEW lines of code can accomplish a LOT of work
- ANSI (American National Standards Institute) maintains a specification for "standard" SQL
- Each DBMS manufacturer follows the ANSI standard, but also adds extended features unique to their SQL

CREATE DATABASE STATEMENT

- `CREATE DATABASE <database>;`

- PostgreSQL always connects to a database. To access a different database, you must create a new connection:

- `\c <database>;`

DDL AND DML

DDL = **D**ata **D**efinition **L**anguage

- Some SQL commands are used to DEFINE or MODIFY the **structures** in the database.
  - `Create`
  - `Alter`
  - `Drop`

DML = **D**ata **M**anipulation **L**anguage

- Some SQL commands are used to MODIFY the **data** in the database
  - `Update`
  - `Insert`
  - `Delete`

DATA DEFINITION LANGUAGE

CREATE STATEMENT

```
CREATE TABLE IF NOT EXISTS <table name>(
  column DATATYPE(L) UNIQUE,
  column DATATYPE(L) NOT NULL,
  Column DATATYPE(L) CHECK <condition>,
  Column DATATYPE(L) PRIMARY KEY
);
```

CREATE RECIPES

```
CREATE TABLE IF NOT EXISTS Recipes(
RecipeId SERIAL PRIMARY KEY,
RecipeName varchar(200) NOT NULL,
PrepTimeMins int NOT NULL,
CookTimeMins int NOT NULL,
AuthorId int,
Image varchar(200),
RecipeURL text NOT NULL
);
```

CREATE USERS

```
CREATE TABLE IF NOT EXISTS Users(
UserId serial PRIMARY KEY,
UserName varchar(200) UNIQUE,
EmailId varchar(300) NOT NULL,
password varchar(200) NOT NULL,
isAdmin boolean NOT NULL
);
```

CREATE INGREDIENTS

```
CREATE TABLE IF NOT EXISTS Ingredients(
IngredientId SERIAL PRIMARY KEY,
IngredientName varchar(100) NOT NULL,
Category varchar(100)
);
```

ALTER STATEMENT

```
ALTER TABLE <table name>
ADD COLUMN column DATATYPE(L),
[CONSTRAINT <constraint name>]

ALTER TABLE <table name>
DROP CONSTRAINT <constraint name>

ALTER TABLE <table name>
DROP COLUMN IF EXISTS <column name>
```

ADDING COLUMN TO RECIPES

```
ALTER TABLE Recipes
ADD COLUMN RecipeAddedDate timestamp;

ALTER TABLE Recipes
ALTER COLUMN RecipeAddedDate
SET NOT NULL;
```

ADD FOREIGN KEY CONSTRAINT

```
ALTER TABLE Recipes
ADD CONSTRAINT fk_userId FOREIGN KEY
(authorId) REFERENCES Users(userId);


OR


ALTER TABLE Recipes
ADD CONSTRAINT fk_userId FOREIGN KEY
(authorId) REFERENCES Users;
```

DROP STATEMENT

- **DROP TABLE** `<table name>`

Example:

```
DROP TABLE Recipes;
```

In the industry, DO NOT drop a table without consent from a person of authority.

DATA MANIPULATION LANGUAGE

INSERT STATEMENT

-
  ```
  INSERT INTO <table name>
      VALUES (value, value, value, value);
  ```

(must have a value/NULL for every column in the table)

-
  ```
  INSERT INTO <table name> (column, column, column)
  VALUES (value, value, value);
  ```

(if no column/value is specified, NULL or default will be assigned)

INSERT DATA

- Single-row Insert:

```
INSERT INTO Users(username, emailid, isadmin) VALUES
('FoodVoyager', 'pqr@xyz.com', true);
```

- Multi-row Insert:

```
INSERT INTO Users(username, emailid, isadmin) VALUES
('CookingIsLife', 'abc@xyz.com', false),
('BakingGoddess', 'bkg@xyz.com',false);
```

UPDATE STATEMENT

- `UPDATE <table name>`
- `SET <column> = <value>`
- `WHERE <condition>`

NOTE: if the WHERE is omitted, ALL rows are updated.

UPDATE AUTHOR

```
UPDATE recipes
SET authorid = 3
WHERE recipeid = 2;
```

DELETE STATEMENT

- DELETE statement

```
DELETE FROM <table name>
     WHERE <condition>
```

NOTE: if the WHERE is omitted, ALL rows are deleted.

QUERYING DATA

SELECT STATEMENT

- SELECT statement can be used to return all columns of the table

```
SELECT * FROM <table A>;
```

- SELECT statement can be used to return a subset of columns from the table

```
SELECT <column1>, <column2>, <column3>, <math
expression>
FROM <table A> ;
```

SELECT STATEMENT

- Literals may be either 'Character' (in quotes) or Numeric
- Math expressions
  - Only use with columns defined as numeric data types

+	Add
-	Subtract
*	Multiply
/	Divide
**	Exponent

SELECT STATEMENT

- Rename a column in the answer set with “AS” – also known as **aliasing**
- Concatenate character columns with

  CONCAT(<column1>,column2>)
- Comment out a line or part of a line of code by prefixing it with “--” or embedding a “#”
- Limit the size of the answer set with “limit”

```
SELECT RecipeName, (preptime + cooktime) AS "Total
Duration" FROM Recipes LIMIT 5;
```

WHERE CLAUSE

```
SELECT <column1>, <column2>, <column3>, <literal>, <math
expression> AS <label>
FROM <table A>
WHERE <condition>;
```

Example:

```
SELECT * FROM Recipes
WHERE authorid = 3;
```

WHERE CLAUSE

- The WHERE clause results in a subset of ROWs to appear in the answer set
- The condition in the WHERE clause takes this format:
- **< operand >** < operator > **< operand >**
- Operands may be columns or literals or expressions
- Operator may be

=	Equals	Like
<>	Not Equals	Between
>	Greater than	In
<	Less than	

WHERE CLAUSE

- Operator may be: `In` or `Like` `In` (literal, literal, literal)
  - Like 'string' with `%` or _ as a wildcard
- Multiple conditions may be joined with Boolean operators
- `AND OR`
- Conditions may be negated with Boolean operator
- `NOT`
- Answer Set rows may be sorted with "`Order By`"
- `Order By` defaults to Ascending, can specify `DESC`

EXAMPLE

```
SELECT * FROM Recipes
WHERE RecipeName LIKE '%chicken%’;

SELECT * FROM Recipes
WHERE AuthorId IN (1,4,7,9);

SELECT * FROM Recipes
WHERE AuthorId IN (1,4,7,9) AND
RecipeName LIKE '%salmon%';
```

DISTINCT

- The answer set may contain duplicate rows
- The “`distinct`” keyword before a column removes duplicates

DISTINCT

```
SELECT AuthorId
FROM Recipes;
```

```
SELECT **DISTINCT** AuthorId
FROM Recipes;
```

HANDLING DATES

- Columns with a data type of "`TIMESTAMP`" are stored as a 4-byte binary integer representing the number of seconds since 1970-01-01 00-00-00 UTC. TIMESTAMP has a range of '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC.

- If no value is provided for the TIME portion of a DATETIME column, it defaults to 00:00.00.0000

UNDERSTANDING THE DATE

- YYYY-MM-DD and hh:mm.ss.nnn
  - YYYY is four digits from 1000 through 9999 that represent a year.
  - MM is two digits, ranging from 01 to 12, that represent a month in the specified year.
  - DD is two digits, ranging from 01 to 31 depending on the month, that represent a day of the specified month.
  - hh is two digits, ranging from 00 to 23, that represent the hour.
  - mm is two digits, ranging from 00 to 59, that represent the minute.
  - ss is two digits, ranging from 00 to 59, that represent the second.
  - nnn is zero to three digits, ranging from 0 to 999, that represent the fractional seconds.

WORKING WITH DATES

```
SELECT NOW();


SELECT * FROM Recipes
WHERE RecipeAddedDate >= '09-27-2022';
```

GROUP FUNCTIONS

- `SUM` - Provides the sum of the values in a column across many rows
- `AVG` - Provides the average of the values in a column across many rows
- `COUNT` - Provides a count of how many rows have a value in a column, counted across many rows
- `MIN` - Provides the lowest value in a column across many rows
- `MAX` - Provides the highest value in a column across many rows

GROUP FUNCTIONS

- `SUM`, `AVG` must only be used with `NUMERIC` columns
- `MIN`, `MAX` can be used with any data type
- `COUNT` can be used with any column, or with a (*) to simply count rows
- Group functions require SQL to create an interim answer set, and then process the group function against the interim answer set, delivering a final answer set that contains only the final total for the function. Always returns an integer value.
- When you combine a GROUP FUNCTION with a WHERE clause, keep in mind that the WHERE clause simply reduces the number of rows in the INTERIM answer set before the GROUP function does its calculation.

GROUP FUNCTIONS

- `SELECT` **`COUNT`**`(*) AS "Number of Recipes" FROM Recipes;`
- `SELECT` **`COUNT`**`(`**`DISTINCT`**` AuthorId) AS ”Unique Authors" FROM Recipes;`
- `SELECT` **`COUNT`**`(*) AS "Recipes from 1 author" FROM Recipes WHERE AuthorId = 3;`

GROUP FUNCTIONS

- SELECT **COUNT**(*) AS "Recipes under 50 mins" FROM Recipes WHERE (preptimemins+cooktimemins) < 50;
- SELECT **MIN**(preptimemins+cooktimemins) AS "Fastest Dish to Cook" FROM Recipes;

GROUP BY

```
SELECT AuthorId, COUNT(*) AS "Count of recipes from each author”
FROM Recipes
GROUP BY AuthorId
ORDER BY COUNT(*);
```

- Why is the following statement incorrect?

```
SELECT AuthorId, COUNT(*) AS "Count of recipes from each author”
FROM Recipes
ORDER BY COUNT(*);
```

LET’S TRY TO WRITE SOME QUERIES

- Which author has written the greatest number of recipes?

HAVING

- Is simply like a WHERE clause against the answer set when you use a GROUP BY

```
SELECT AuthorId, COUNT(*) AS ”Number of Recipes"
FROM Recipes
GROUP BY AuthorId
HAVING COUNT(*) > 2
```

SUBQUERY

- Simply: a query within a query. The answer set to an “inner” query is used as a predicate in a where clause in the “outer” query.
- The subquery must return only one column.
- If the outer query WHERE clause contains an “equals”
- condition, the subquery must return ONE row.
- If the outer query WHERE clause contains an “in” condition, the subquery may return multiple rows, presented as a list of values.
- The Subquery is embedded within parentheses
- Outer and inner queries can hit two different tables

SUBQUERY

```
SELECT AuthorId, (preptimemins+cooktimemins) AS "Total Duration"
FROM Recipes
WHERE (preptimemins+cooktimemins) > (
SELECT AVG(preptimemins+cooktimemins) FROM Recipes
);
```

- Note that with the "equals" condition, the inner query returns only one value (one row, one column)

SUBQUERY

- Note that with the "ANY" condition, the inner query returns many values (many rows, one column) as a list
- This also uses two different tables