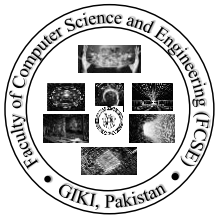
CS232L – Database Management System

INTRO TO DBMS AND POSTGRESQL

Ghulam Ishaq Khan Institute of Engineering Sciences



Faculty of Computer Science & Engineering

CS232L – Database Management system Lab

Lab 2 –Clauses and Basic Operations on PostgreSQL Data

Objective

The objective of this session is to

- PostgreSQL WHERE clause overview
- Perform the Arithmetic Operations on PostgreSQL Data using Where Clause
- Use the Logical and Comparison operators in SQL statements.
- Null Value, defining a Column alias, DISTINCT clause, IS NULL operator
- Using the update query
- Use of delete, drop and truncate.

Instructions

- Open the handout/lab manual in front of a computer in the lab during the session.
- Practice each new command by completing the examples and exercise.
- Turn-in the answers for all the exercise problems as your lab report.
- When answering problems, indicate the commands you entered and the output displayed.

PostgreSQL **WHERE** clause overview

The SELECT statement returns all rows from one or more columns in a table. To select rows that satisfy a specified condition, you use a WHERE clause.

The syntax of the PostgreSQL WHERE clause is as follows:

```
SELECT select_list
FROM table_name
WHERE condition
```

The WHERE clause appears right after the FROM clause of the SELECT statement.
The WHERE clause uses the condition to filter the rows returned from the SELECT clause.

The condition must evaluate to true, false, or unknown. It can be a boolean expression or a combination of boolean expressions using the AND and OR operators.

The query returns only rows that satisfy the condition in the WHERE clause. In other words, only rows that cause the condition evaluates to true will be included in the result set.

PostgreSQL evaluates the WHERE clause after the FROM clause and before the SELECT:

If you use column aliases in the SELECT clause, you cannot use them in the WHERE clause.

Besides the SELECT statement, you can use the WHERE clause in the UPDATE and DELETE statement to specify rows to be updated or deleted.

PostgreSQL Arithmetic Operators

You may need to modify the way in which data is displayed, perform calculation, or look at what-if scenarios. This is possible using arithmetic expressions. An arithmetic expression may contain column names, constant numeric values, and the arithmetic operators.
PostgreSQL provided many Mathematical operators for common mathematical conventions. List of arithmetic operators available in SQL are Addition (+), Subtraction (-), Multiplication (*), Division (/). You can use arithmetic operators in any clause of a SQL statement except the FROM clause. If an arithmetic expression contains more than one operator then multiplication and division are evaluated first. If operators within an expression are of same priority, then evaluation is done from left to right. You can use parentheses to force the expression within parentheses to be evaluated first.

List of Mathematical Operators

Operators	Description	Example	Output
+	Addition	5 + 8	13
-	Subtraction	6 - 9	-3
*	Multiplication	5 * 8	40
/	Division	15 / 3	5
%	Modulo (Remainder)	15 % 2	1
^	Exponentiation	4.0 ^ 2.0	16
| /	Square Root	| / 16	4
| | /	Cube Root	| | / 27	3
!	Factorial	! 6	720

PostgreSQL Arithmetic operator example

Example

```
SELECT first_name, last_name salary, 12*salary+100 from employees;
```

will give different result from

```
SELECT first_name, last_name salary, 12*(salary+100)
from employees;
```

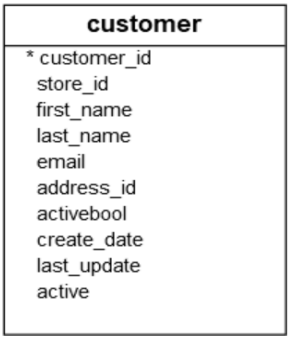
PostgreSQL Logical Operators

A **logical operator** combines the result of two component conditions to produce a single result based on them or to invert the result of a single condition. Three logical operators are available in PostgreSQL.

- AND Returns TRUE if both component conditions are TRUE
- OR Returns TRUE if either component condition is TRUE
- NOT negate the result of other operators

Using WHERE clause with the AND operator example

We will use the customer table



The following example finds customers whose first name and last name are Jamie and rice by using the AND logical operator to combine two Boolean expressions:

```
SELECT
        last_name,
        first_name
FROM
        customer
WHERE
        first_name = 'Jamie' AND
        last_name = 'Rice';
```

Output:

	last_name character varying (45)	first_name character varying (45)
1	Rice	Jamie

Using the WHERE clause with the **OR** operator example

This example finds the customers whose last name is Rodriguez or first name is Adam by using the OR operator:

```
SELECT
        first_name,
        last_name
FROM
        customer
WHERE
        last_name = 'Rodriguez' OR
        first_name = 'Adam';
```

Output:

	first_name character varying (45)	last_name character varying (45)
1	Laura	Rodriguez
2	Adam	Gooch

You can use several conditions in one WHERE clause using the AND and OR operators.

Example

```
SELECT empno,ename,job,sal
from emp
where sal>=1100 and job='CLERK';
```

Let's suppose we have an Employee Table then the query given above will return the employee number , employee name , job and salalry form employee table where the salary of the employee is greater than and equal to 1100 and job title is 'CLERK'.

PostgreSQL Comparison Operators

Comparison operators are used in conditions that compare one expression to another. The operators are

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<> or !=	Not equal

Using WHERE clause with the equal (=) operator example

The following statement uses the WHERE clause customers whose first names are Jamie:

```
SELECT
	last_name,
	first_name
FROM
	customer
WHERE
	first_name = 'Jamie';
```

Output:

	last_name character varying (45)	first_name character varying (45)
1	Rice	Jamie
2	Waugh	Jamie

Using the WHERE clause with the not equal operator (<>) example

This example finds customers whose first names are Brand or Brandy and last names are not Motley:

```
SELECT
    first_name,
    last_name
FROM
    customer
WHERE
    first_name= 'Brandy' OR first_name='Brad' AND
    last_name <> 'Motley';
```

Output:

	first_name character varying (45)	last_name character varying (45)
1	Brandy	Graves
2	Brad	Mccurdy

Other Comparison Operators:

Other comparison operators are

IN	Return true if a value matches any value in a list
BETWEEN	Return true if a value is between a range of values
LIKE	Return true if a value matches a pattern
IS NULL	Return true if a value is NULL
NOT	Negate the result of other operators

Using WHERE clause with the IN operator example

If you want to match a string with any string in a list, you can use the IN operator.

For example, the following statement returns customers whose first name is Ann, or Anne, or Annie:

```
SELECT
        first_name,
        last_name
FROM
        customer
WHERE
        first_name IN ('Ann','Anne','Annie');
```

Output:

	first_name character varying (45)	last_name character varying (45)
1	Ann	Evans
2	Anne	Powell
3	Annie	Russell

Using the WHERE clause with the LIKE operator example

To find a string that matches a specified pattern, you use the LIKE operator. The following example returns all customers whose first names start with the string Ann:

```
SELECT
        first_name,
        last_name
FROM
        customer
WHERE
        first_name LIKE 'Ann%'
```

Output:

	first_name character varying (45)	last_name character varying (45)
1	Anna	Hill
2	Ann	Evans
3	Anne	Powell
4	Annie	Russell
5	Annette	Olson

The % is called a wildcard that matches any string. The 'Ann%' pattern matches any string that starts with 'Ann'.

Using the WHERE clause with the BETWEEN operator example

The following example finds customers whose first names start with the letter A and contains 3 to 5 characters by using the BETWEEN operator.

The BETWEEN operator returns true if a value is in a range of values.

```
1  SELECT
2      first_name,
3      LENGTH(first_name) name_length
4  FROM
5      customer
6  WHERE
7      first_name LIKE 'A%' AND
8      LENGTH(first_name) BETWEEN 3 AND 5;
9
```

	character varying (45)	integer
1	Amy	3
2	Ann	3
3	Ana	3
4	Andy	4
5	Anna	4
6	Anne	4
7	Alma	4
8	Adam	4
9	Alan	4
10	Alex	4
11	Angel	5
12	Agnes	5
13	Andre	5
14	Aaron	5
15	Allan	5
16	Allen	5
17	Alice	5
18	Alvin	5
19	Anita	5
20	Amber	5
21	April	5
22	Annie	5

In this example, we used the LENGTH() function gets the number of characters of an input string.

Null value

If a row lacks the data value for a particular column, that value is said to be null, or to contain null. A null value is a value that is unavailable, unassigned, unknown, or inapplicable. A null value is not the same as zero or a space. Zero is a number and space is a character. If any column value in an arithmetic expression is null, the result is null. For example if you attempt to perform division with zero, you get an error. However if you divide a number by null, the result is a null or unknown.

PostgreSQL Column Alias

A column alias allows you to assign a column or an expression in the select list of a SELECT statement a temporary name. The column alias exists temporarily during the execution of the query.

The following illustrates the syntax of using a column alias:

```
SELECT column_name AS alias_name
FROM table_name;
```

In this syntax, the column_name is assigned an alias alias_name. The AS keyword is optional so you can omit it like this:

```
SELECT column_name alias_name
FROM table_name;
```

The following syntax illustrates how to set an alias for an expression in the SELECT clause:

```
SELECT expression AS alias_name
FROM table_name;
```

The main purpose of column aliases is to make the headings of the output of a query more meaningful.

PostgreSQL column alias examples

Assigning a column alias to a column example

If you want to retrieve first_name and last_name and rename the last_name heading, you can assign it a new name using a column alias like this:

```
SELECT
   first_name,
   last_name AS surname
FROM customer;
```

This query assigned the surname as the alias of the last_name column:

	first_name character varying (45)	surname character varying (45)
1	Jared	Ely
2	Mary	Smith
3	Patricia	Johnson
4	Linda	Williams
5	Barbara	Jones
6	Elizabeth	Brown
7	Jennifer	Davis
8	Maria	Miller
9	Susan	Wilson
10	Margaret	Moore
11	Dorothy	Taylor

Or you can make it shorter by removing the AS keyword as follows:

```
SELECT
   first_name,
   last_name surname
FROM customer;
```

Column aliases that contain spaces

If a column alias contains one or more spaces, you need to surround it with double quotes like this:

```
column_name AS "column alias"
```

PostgreSQL SELECT DISTINCT

The DISTINCT clause is used in the SELECT statement to remove duplicate rows from a result set. The DISTINCT clause keeps one row for each group of duplicates.
The DISTINCTclause can be applied to one or more columns in the select list of the SELECT statement.

The following illustrates the syntax of the DISTINCT clause:

```
SELECT
   DISTINCT column1
FROM
   table_name;
```

In this statement, the values in the column1 column are used to evaluate the duplicate.

If you specify multiple columns, the DISTINCT clause will evaluate the duplicate based on the combination of values of these columns.

```
SELECT
   DISTINCT column1, column2
FROM
   table_name;
```

PostgreSQL SELECT DISTINCT examples

Let's create a new table called distinct_demo and insert data into it for practicing the DISTINCT clause.
First, use the following CREATE TABLE statement to create the distinct_demo table that consists of three columns: id, bcolor and fcolor.

```
CREATE TABLE distinct_demo (
    id serial NOT NULL PRIMARY KEY,
    bcolor VARCHAR,
    fcolor VARCHAR
);
```

Second, insert some rows into the distinct_demo table using the following INSERT statement:

```
INSERT INTO distinct_demo (bcolor, fcolor)
VALUES
    ('red', 'red'),
    ('red', 'red'),
    ('red', NULL),
    (NULL, 'red'),
    ('red', 'green'),
    ('red', 'blue'),
    ('green', 'red'),
    ('green', 'blue'),
    ('green', 'green'),
    ('blue', 'red'),
    ('blue', 'green'),
    ('blue', 'blue');
```

Third, query the data from the distinct_demo table using the SELECT statement:

```
SELECT
    id,
    bcolor,
    fcolor
FROM
    distinct_demo ;
```

Output:

	id integer	bcolor character varying	fcolor character varying
1	1	red	red
2	2	red	red
3	3	red	[null]
4	4	[null]	red
5	5	red	green
6	6	red	blue
7	7	green	red
8	8	green	blue
9	9	green	green
10	10	blue	red
11	11	blue	green
12	12	blue	blue

PostgreSQL IS NULL operator

In the database world, NULL means missing information or not applicable. NULL is not a value, therefore, you cannot compare it with any other values like numbers or strings. The comparison of NULL with a value will always result in NULL, which means an unknown result.

In addition, NULL is not equal to NULL so the following expression returns NULL:

```
NULL = NULL
```

Assuming that you have a contacts table that stores the first name, last name, email, and phone number of contacts. At the time of recording the contact, you may not know the contact's phone number.

To deal with this, you define the phone column as a nullable column and insert NULL into the phone column when you save the contact information.

```
CREATE TABLE contacts(
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(255) NOT NULL,
    phone VARCHAR(15),
    PRIMARY KEY (id)
);
```

The following statement inserts two contacts, one has a phone number and the other does not:

```
INSERT INTO contacts(first_name, last_name, email, phone)
VALUES ('John','Doe','john.doe@example.com',NULL),
    ('Lily','Bush','lily.bush@example.com','(408-234-2764)');
```

To find the contact who does not have a phone number you may come up with the following statement:

```
SELECT
    first_name,
    last_name,
    email,
    phone
FROM
    contacts
WHERE
    phone NULL;
```

The statement returns no row. This is because the expression phone = NULL in the WHERE clause always returns false.

Even though there is a NULL in the phone column, the expression NULL = NULL returns false. This is because NULL is not equal to any value even itself.

To check whether a value is NULL or not, you use the IS NULL operator instead:

value IS NULL

The expression returns true if the value is NULL or false if it is not.

So to get the contact who does not have any phone number stored in the phone column, you use the following statement instead:

```
SELECT
    first_name,
    last_name,
    email,
    phone
FROM
    contacts
WHERE
    phone IS NULL;
```

Here is the output:

first_name character varying (50)	last_name character varying (50)	email character varying (255)	phone character varying (15)
John	Doe	john.doe@example.com	[null]

PostgreSQL UPDATE

The PostgreSQL UPDATE statement allows you to modify data in a table. The following illustrates the syntax of the UPDATE statement:

```
UPDATE table_name
SET column1 = value1,
   column2 = value2,
   ...
WHERE condition;
```

In this syntax:

- First, specify the name of the table that you want to update data after the UPDATE keyword.
- Second, specify columns and their new values after SET keyword. The columns that do not appear in the SET clause retain their original values.
- Third, determine which rows to update in the condition of the WHERE clause.

The WHERE clause is optional. If you omit the WHERE clause, the UPDATE statement will update all rows in the table.

When the UPDATE statement is executed successfully, it returns the following command tag:

UPDATE count

The count is the number of rows updated including rows whose values did not change.

PostgreSQL UPDATE examples

Let's take some examples of using the PostgreSQL UPDATE statement.

The following statements create a table called courses and insert some data into it:

```
CREATE TABLE courses(
    course_id serial primary key,
    course_name VARCHAR(255) NOT NULL,
    description VARCHAR(500),
    published_date date
);

INSERT INTO
    courses(course_name, description, published_date)
VALUES
    ('PostgreSQL for Developers','A complete PostgreSQL for Developers',
     '2020-07-13'),
    ('PostgreSQL Admininstration','A PostgreSQL Guide for DBA',NULL),
    ('PostgreSQL High Performance',NULL,NULL),
    ('PostgreSQL Bootcamp','Learn PostgreSQL via Bootcamp','2013-07-11'),
    ('Mastering PostgreSQL','Mastering PostgreSQL in 21 Days','2012-06-30');
```

The following statement returns the data from the courses table:

SELECT * FROM courses;

	course_id integer	course_name character varying (255)	description character varying (500)	published_date date
1	1	PostgreSQL for Developers	A complete PostgreSQL for Developers	2020-07-13
2	2	PostgreSQL Admininstration	A PostgreSQL Guide for DBA	[null]
3	3	PostgreSQL High Performance	[null]	[null]
4	4	PostgreSQL Bootcamp	Learn PostgreSQL via Bootcamp	2013-07-11
5	5	Mastering PostgreSQL	Mastering PostgreSQL in 21 Days	2012-06-30

PostgreSQL UPDATE – updating one row

The following statement uses the UPDATE statement to update the course with id 3. It changes the published_date from NULL to '2020-08-01'.

```
UPDATE courses
SET published_date = '2020-08-01'
WHERE course_id = 3;
```

The statement returns the following message indicating that one row has been updated:

```
UPDATE 1
```

The following statement selects the course with id 3 to verify the update:

```
SELECT *
FROM courses
WHERE course_id = 3;
```

	course_id integer	course_name character varying (255)	description character varying (500)	published_date date
1	3	PostgreSQL High Performance	[null]	2020-08-01

The DELETE Statement

The DELETE statement is used to delete rows in a table.

Syntax
`DELETE FROM table_name where column_name = some_value;`

Delete All Rows

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

Syntax:

DELETE FROM table table_name;
Or
DELETE * FROM table table_name;

PostgreSQL DROP TABLE

To drop a table from the database, you use the DROP TABLE statement as follows:

DROP TABLE [IF EXISTS] table_name ;

In this syntax:

- First, specify the name of the table that you want to drop after the DROP TABLE keywords.
- Second, use the IF EXISTS option to remove the table only if it exists.

If you remove a table that does not exist, PostgreSQL issues an error. To avoid this situation, you can use the IF EXISTS option.

In case the table that you want to remove is used in other objects such as views, triggers, functions, and stored procedures, the DROP TABLE cannot remove the table. In this case, you have two options:

- The CASCADE option allows you to remove the table and its dependent objects.
- The RESTRICT option rejects the removal if there is any object depends on the table. The RESTRICT option is the default if you don't explicitly specify it in the DROP TABLE statement.

To remove multiple tables at once, you can place a comma-separated list of tables after the DROP TABLE keywords:

DROP TABLE [IF EXISTS]
  table_name_1,
  table_name_2,
  ...
[CASCADE | RESTRICT];

Note that you need to have the roles of the superuser, schema owner, or table owner in order to drop tables.

PostgreSQL DROP TABLE examples

Let's take some examples of using the PostgreSQL DROP TABLE statement

1) Drop a table that does not exist

The following statement removes a table named `author` in the database:

```
DROP TABLE author;
```

PostgreSQL issues an error because the `author` table does not exist.

```
[Err] ERROR:  table "author" does not exist
```

To avoid the error, you can use the `IF EXISTS` option like this.

```
DROP TABLE IF EXISTS author;
```

```
NOTICE:  table "author" does not exist, skipping DROP TABLE
```

As can be seen clearly from the output, PostgreSQL issued a notice instead of an error.

2) Drop a table that has dependent objects

The following creates new tables called authors and pages:

```
CREATE TABLE authors (
        author_id INT PRIMARY KEY,
        firstname VARCHAR (50),
        lastname VARCHAR (50)
);

CREATE TABLE pages (
        page_id serial PRIMARY KEY,
        title VARCHAR (255) NOT NULL,
        contents TEXT,
        author_id INT NOT NULL,
        FOREIGN KEY (author_id)
          REFERENCES authors (author_id)
);
```

The following statement uses the `DROP TABLE` to drop the `author` table:

```
DROP TABLE IF EXISTS authors;
```

Because the constraint on the `page` table depends on the `author` table, PostgreSQL issues an error message:

```
ERROR:  cannot drop table authors because other objects depend on it
DETAIL:  constraint pages_author_id_fkey on table pages depends on table author
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
SQL state: 2BP01
```

In this case, you need to remove all dependent objects first before dropping the `author` table or use `CASCADE` option as follows:

```
DROP TABLE authors CASCADE;
```

PostgreSQL removes the `author` table as well as the constraint in the `page` table.

If the `DROP TABLE` statement removes the dependent objects of the table that is being dropped, it will issue a notice like this:

```
NOTICE:  drop cascades to constraint pages_author_id_fkey on table pages
```

3) Drop multiple tables

The following statements create two tables for the demo purposes:

PostgreSQL TRUNCATE TABLE

To remove all data from a table, you use the DELETE statement. However, when you use the DELETE statement to delete all data from a table that has a lot of data, it is not efficient. In this case, you need to use the TRUNCATE TABLE statement:

```
TRUNCATE TABLE table_name;
```

The TRUNCATE TABLE statement deletes all data from a table without scanning it. This is the reason why it is faster than the DELETE statement.

Remove all data from one table

The simplest form of the TRUNCATE TABLE statement is as follows:

```
TRUNCATE TABLE table_name;
```
Code language: SQL (Structured Query Language) (sql)

The following example uses the TRUNCATE TABLE statement to delete all data from the invoices table:

```
TRUNCATE TABLE invoices;
```

Remove all data from multiple tables

To remove all data from multiple tables at once, you separate each table by a comma (,) as follows:

```
TRUNCATE TABLE
   table_name1,
   table_name2,
   ...;
```
Code language: SQL (Structured Query Language) (sql)

For example, the following statement removes all data from invoices and customers tables:

```
TRUNCATE TABLE invoices, customers;
```

Remove all data from a table that has foreign key references

In practice, the table you want to truncate often has the foreign key references from other tables that are not listed in the TRUNCATE TABLE statement.

By default, the TRUNCATE TABLE statement does not remove any data from the table that has foreign key references.

To remove data from a table and other tables that have foreign key reference the table, you use CASCADE option in the TRUNCATE TABLE statement as follows :

```
TRUNCATE TABLE table_name
CASCADE;
```
Code language: SQL (Structured Query Language) (sql)

The following example deletes data from the invoices table and other tables that reference the invoices table via foreign key constraints:

```
TRUNCATE TABLE invoices CASCADE;
```
Code language: SQL (Structured Query Language) (sql)

The CASCADE option should be used with further consideration or you may potentially delete data from tables that you did not want.

By default, the TRUNCATE TABLE statement uses the RESTRICT option which prevents you from truncating the table that has foreign key constraint references.