**a**

**Advance Database Systems**

**PostgreSQL Tutorial**

**UBUNTU SERVER 20.04 64 bits**

**PGADMIN 4**

**By: Ronald M. Baclayon**

In this PostgreSQL Tutorial I will show you how to use POSTGRESQL using PGADMIN 4.

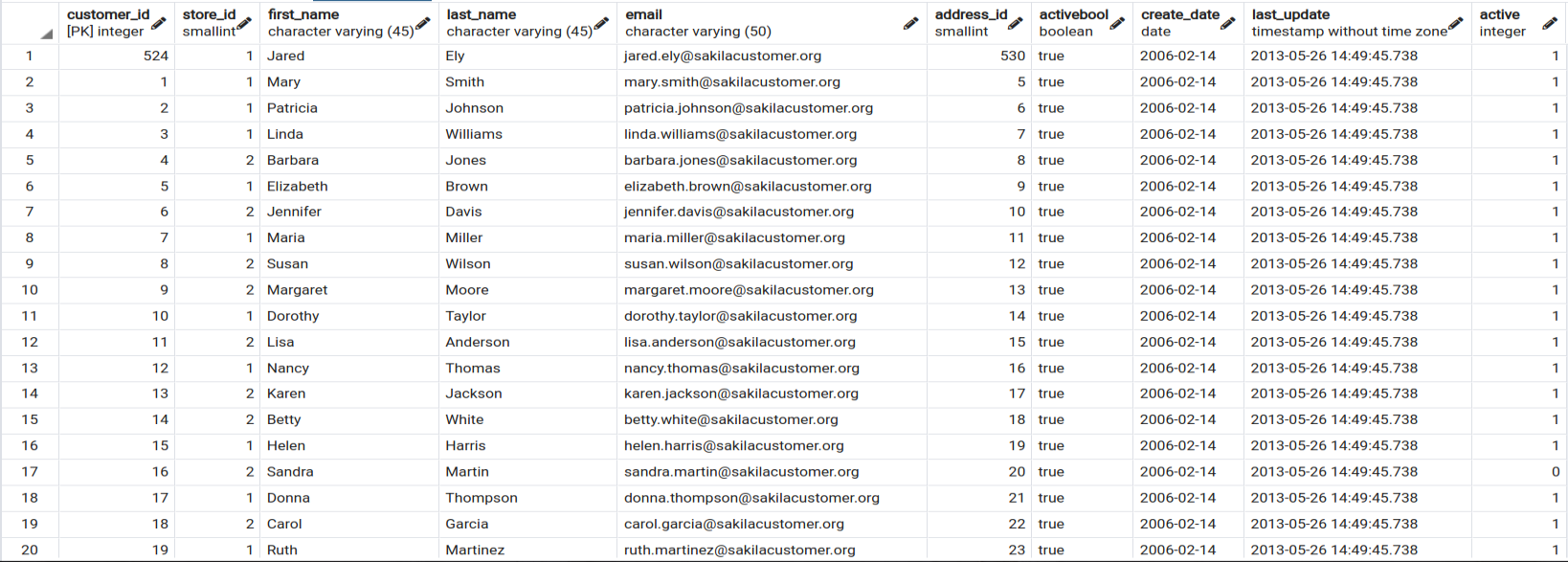
You can download PGADMIN 4 here:

[**https://www.pgadmin.org/download/**](https://www.pgadmin.org/download/)

**Section 1**

**Querying Data**

**SAMPLE TABLE: customer**



*Section 1: Querying Data*

|  |  |
| --- | --- |
| **Select** – show you how to query data from a single table. | |
| **SYNTAX** | **SELECT**  column\_name  **FROM**  table\_name; |
| You can directly specify what table you wanted to view using the syntax.  SELECT  *first\_name*,  *last\_name*  FROM  customer; |  |
| If you wanted to view all columns inside the table. Just use the syntax below.  SELECT  *\**  FROM  customer;  As you can see, I’m using asterisk \* instead of a specific column name. |  |

*Section 1: Querying Data*

|  |  |
| --- | --- |
| **Column aliases –** learn how to assign temporary names to columns or expressions in a query. | |
| **SYNTAX** | **SELECT**  column\_name **AS** alias\_name  **FROM**  table\_name; |
| If you want to change the column name of your output you can use the syntax below.  SELECT  first\_name AS customer\_name  FROM  customer; | |
| ***WITHOUT*** COLUMN ALIAS | ***WITH*** COLUMN ALIAS |
| You can also combine 2 columns into 1 column. *first\_name || ' ' || last\_name AS "full name"*  Note: you can add space or additional characters between single quote ‘ ’  *||* is use as concatenating operator  If your new column name has spaces, you can instead put it inside a double quote “new column name”.  SELECT  first\_name || ’, ’ || last\_name  AS “customer full name”  FROM  Customer; |  |

*Section 1: Querying Data*

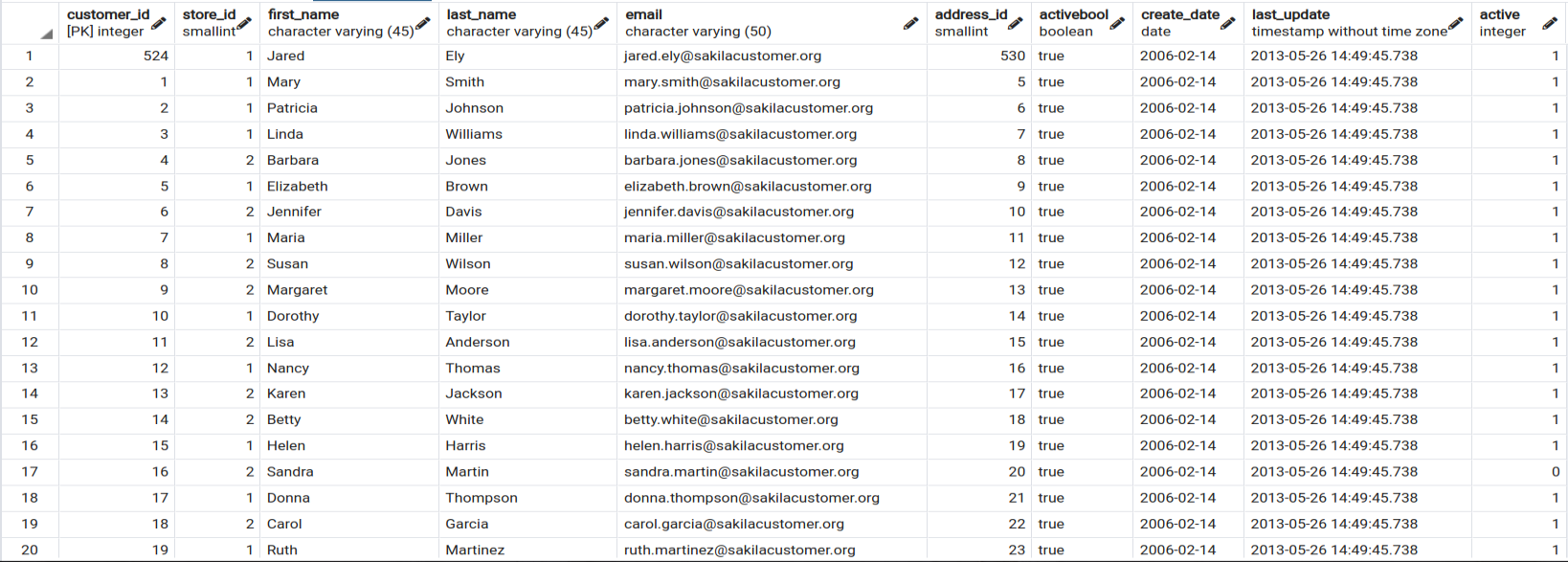
|  |  |
| --- | --- |
| **Order By** – guide you on how to sort the result set returned from a query. | |
| **SYNTAX** | **SELECT**  column\_name  **FROM**  table\_name  **ORDER BY**  column\_name **ASC/DSC;** |
| You can use “Order By” to sort a column either in ascending or descending order.  ASC – ascending (default)  DSC – descending  If you will not specify if you want to sort the column by ASC or DSC, by default automatically it will be sorted by ASC.  SELECT  last\_name || ‘, ’ || first\_name  AS “full name”  FROM  customer  ORDER BY  “full name” DESC; | |
| **SORT BY ASC** | **SORT BY DESC** |

*Section 1: Querying Data*

|  |  |
| --- | --- |
| **Select Distinct –** provide you a clause that removes duplicate rows in the result set. | |
| **SYNTAX** | **SELECT**  **DISTINCT** column\_name  **FROM**  table\_name; |
| You can use distinct to remove duplicates in a column.  **SELECT**  **DISTINCT** first\_name  **FROM**  customer  **ORDER** **BY**  first\_name **DESC**; | You can use DISTINCT ON if you have multiple columns output.  **SELECT**  **DISTINCT** **ON** (first\_name) first\_name, last\_name  **FROM**  Customer  **ORDER** **BY**  first\_name DESC, last\_name DESC; |
| **WITHOUT DISTINCT** | **WITH DISTINCT** |
| **WITHOUT ON (2 COLUMNS )** | **WITH ON (2 COLUMNS)** |

**Section 2**

**Filtering Data**

**SAMPLE TABLE: customer**

*Section 2: Filtering Data*

|  |  |
| --- | --- |
| **Where –** filter rows based on a specified condition. | |
| **SYNTAX** | **SELECT**  **Column\_Name**  **FROM**  **Table\_Name**  **WHERE**  **Condition;** |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Operator | Description |  | Operator | Description | | = | Equal |  | AND | Logical operator AND | | > | Greater than |  | OR | Logical operator OR | | < | Less than |  | IN | Return true if a value matches any value in a list | | >= | Greater than or equal |  | BETWEEN | Return true if a value is between a range of values | | <= | Less than or equal |  | LIKE | Return true if a value matches a pattern | | <> or != | Not equal |  | IS NULL | Return true if a value is NULL | |  |  |  | NOT | Negate the result of other operators | | |
| **SELECT**  **last\_name,**  **first\_name**  **FROM**  **customer**  **WHERE**  **last\_name LIKE ‘An%’;**  On this syntax, we specify that we only wanted to view customers with last\_name starting with AN. |  |

*Section 2: Filtering Data*

|  |  |
| --- | --- |
| **Limit** – get a subset of rows generated by a query. | |
| **SYNTAX** | **SELECT**  **Column\_Name**  **FROM**  **Table\_Name**  **LIMIT amount;** |
| **Limit is used to filter number of results. For example you only wanted to view up to 10 result per query.** |  |

*Section 2: Filtering Data*

|  |  |
| --- | --- |
| **Fetch–** limit the number of rows returned by a query. | |
| **SYNTAX** | **SELECT**  **Column\_Name**  **FROM**  **Table\_Name**  **FETCH FIRST amount ROW ONLY;** |
| **FETCH is same as LIMIT, it limits the number of result.**  **OFFSET 5 ROWS**  **– means that you wanted to skip the first 5 results**  **– so instead of having 1 to 10 as a result we have instead 6 to 15.** |  |

*Section 2: Filtering Data*

|  |  |
| --- | --- |
| **In** – select data that matches any value in a list of values. | |
| **SYNTAX** | **SELECT**  **Column\_Name**  **FROM**  **Table\_Name**  **WHERE**  **Column\_name IN (value, value, ….);** |
| **IN is used to provide specific results base on the given list, instead of using OR.** |  |

*Section 2: Filtering Data*

|  |  |
| --- | --- |
| **Between –** select data that is a range of values. | |
| **SYNTAX** | **SELECT**  **Column\_Name**  **FROM**  **Table\_Name**  **WHERE**  **Column\_name BETWEEN value AND value;** |
| **BETWEEN is used to provide specific results base on given range.** |  |

*Section 2: Filtering Data*

|  |  |
| --- | --- |
| **Like – filter data based on pattern matching.** | |
| **SYNTAX** | **SELECT**  **Column\_Name**  **FROM**  **Table\_Name**  **WHERE**  **Column\_name BETWEEN value AND value;** |
| On this syntax, we specify that we only wanted to view customers with last\_name starting with AN. |  |

*Section 2: Filtering Data*

|  |  |
| --- | --- |
| **Like – filter data based on pattern matching.** | |
| **SYNTAX** | **SELECT**  **Column\_Name**  **FROM**  **Table\_Name**  **WHERE**  **Column\_name IS NULL;** |
| The query results empty rows because there is no row that contains null value. It means all customer information has been supplied with data. |  |

**Section 3**

**Joining Multiple Tables**

|  |  |
| --- | --- |
| **TABLE NAME: film\_actor** | **TABLE NAME: actor** |

*Section 3: Joining Multiple Tables*

|  |  |
| --- | --- |
| **Inner Join –** select rows from one table that has the corresponding rows in other tables. | |
| **SYNTAX** | **SELECT**  **Column\_Name *(Note: Primary Key)***  **Column\_Name**  **FROM**  **Table\_Name\_1**  **INNER JOIN**  **Table\_Name\_2**  **ON Table\_Name\_1.Primary\_Key = Table\_Name2.Foreign\_Key;** |
|  | |

*Section 3: Joining Multiple Tables*

|  |  |
| --- | --- |
| **Left Join –** select rows from one table that may or may not have the corresponding rows in other tables. | |
| **SYNTAX** | **SELECT**  **Column\_Name *(Note: Primary Key)***  **Column\_Name**  **FROM**  **Table\_Name\_1**  **LEFT JOIN**  **Table\_Name\_2**  **ON Table\_Name\_1.Primary\_Key = Table\_Name2.Foreign\_Key;** |
|  | |

*Section 3: Joining Multiple Tables*

|  |  |
| --- | --- |
| **Full Outer Join – use the full join to find a row in a table that does not have a matching row in another table** | |
| **SYNTAX** | **SELECT**  **Column\_Name *(Note: Primary Key)***  **Column\_Name**  **FROM**  **Table\_Name\_1**  **FULL OUTER JOIN**  **Table\_Name\_2**  **ON Table\_Name\_1.Primary\_Key = Table\_Name2.Foreign\_Key;** |
|  | |

*Section 3: Joining Multiple Tables*

|  |  |
| --- | --- |
| **Cross Join** – produce a Cartesian product of the rows in two or more tables. | |
| **SYNTAX** | **SELECT**  **Column\_Name *(Note: Primary Key)***  **Column\_Name**  **FROM**  **Table\_Name\_1**  **CROSS JOIN Table\_Name\_2;** |
|  | |