**[01]. Data Definition Language (DDL)**

1. **Create Statement:**

*Syntax: create table r (A1 D1, A2 D2……………….An Dn)*

R= Relation/Table

A=Attribute/Column name

D= Domain/Data type

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: create table account*

*(*

*customer\_name varchar(15)*

*customer\_city varchar(15)*

*balance numeric(12,2)*

*)*

1. Select execute query option or press F5.
2. To show the result write the sql command given below:

Select \* from account

1. **Drop Statement:**

*Syntax: drop table r*

R= Relation/Table

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: drop table account*

1. Select execute query option or press F5.
2. To show the result write the sql command given below:

Select \* from account

1. **Alter add Statement:**

*Syntax: alter table r add A D*

R= Relation/Table

A=Attribute/Column name

D= Domain/Data type

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: alter table account add balance numeric(12,2)*

1. Select execute query option or press F5.
2. To show the result write the sql command given below:

Select \* from account

1. **Alter add Statement:**

*Syntax: alter table r drop A*

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: alter table account drop balance*

1. Select execute query option or press F5.
2. To show the result write the sql command given below:

Select \* from account

**[02]. Data Manipulation language (DML):**

1. **Select Statement:**

*Syntax: select A1, A2 , ….… An*

*From r*

*Where p*

R= Relation/Table

A= Attribute/Column name

P= Predicate

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: Select customer\_name*

*From account*

*Where balance >=12,000*

1. Select execute query option or press F5.
2. To show the result write the sql command given below:

Select \* from account

1. **Insert Statement:**

*Syntax: insert into r*

*Values (attribute values)*

R= Relation/Table

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: Insert into account*

*Values (‘G’, ’H’, 15000)*

1. Select execute query option or press F5.
2. To show the result write the sql command given below:

Select\*from account

1. **Delete Statement:**

*Syntax: delete from r*

*where p*

P= Predicate

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: delete from account*

*where balance <1000*

1. Select execute query option or press F5.
2. To show the result write the sql command given below:

Select\*from account

1. **Update Statement:**

*Syntax: update r*

*set (changed attribute values)*

*where p*

P= Predicate

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: update account*

*set balabce=balance \* 1.75*

*where balance <100000*

1. Select execute query option or press F5.
2. To show the result write the sql command given below:

Select\*from account

**[03]. SQL Set Operation:**

1. **Union All Operation:**

Gives all the values from two tables with duplicate value.

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: (Select customer\_name from Account)*

*Union All*

*(Select customer\_name from loan)*

1. Select execute query option or press F5.
2. **Union Operation:**

Gives all the values from two tables without duplicate value.

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: (Select customer\_name from Account)*

*Union*

*(Select customer\_name from loan)*

1. Select execute query option or press F5.
2. **Intersect Operation:**

Gives only the common values from two tables.

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: (Select customer\_name from Account)*

*intersect*

*(Select customer\_name from loan)*

1. Select execute query option or press F5.
2. **Except Operation:**

Gives only the common values from first query and subtract the values from second query.

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: (Select customer\_name from Account)*

*except*

*(Select customer\_name from loan)*

1. Select execute query option or press F5.

**[04]. SQL Aggregate Function:**

1. **SUM Function:**

Gives the summation of all values of an attribute.

*Syntax: sum (attribute)*

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: select sum (balance)*

*from account*

1. Select execute query option or press F5.
2. **AVG Function:**

Gives the average of all values of an attribute.

*Syntax: avg (attribute)*

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: select avg (balance)*

*from account*

1. Select execute query option or press F5.
2. **MAX Function:**

Gives only the maximum value from all the values of an attribute.

*Syntax: max (attribute)*

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: select max (balance)*

*from account*

1. Select execute query option or press F5.
2. **MIN Function:**

Gives only the minimum value from all the values of an attribute.

*Syntax: min (attribute)*

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: select min (balance)*

*from account*

1. Select execute query option or press F5.
2. **Count Function:**

Gives the total number of values from all the values of an attribute.

*Syntax: count (attribute)*

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: select count (customer\_name)*

*from account*

1. Select execute query option or press F5.

**[05]. Distinct Operator:**

Gives the values of an attribute without duplicate value.

*Syntax: select distinct A*

*from r*

R= Relation/Table

A= Attribute/Column name

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: select distinct customer\_name*

*from account*

1. Select execute query option or press F5.

**[06].AND Operator:**

Gives the result if both the first condition and second condition are true.

*Syntax: select A*

*from r*

*where p1*

*and p2*

R= Relation/Table

A= Attribute/Column name

P= Predicate/condition

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: select \* from account*

*where first\_name= ’Moshiur’*

*and last\_name= ‘Rahman’*

1. Select execute query option or press F5.

**[07].OR Operator:**

Gives the result if either the first condition or second condition is true.

*Syntax: select A*

*from r*

*where p1*

*or p2*

R= Relation/Table

A= Attribute/Column name

P= Predicate/condition

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: select \* from account*

*where first\_name= ’Moshiur’*

*or last\_name= ‘Rahman’*

1. Select execute query option or press F5.

**[08].Order by ASC Operator:**

Used to sort the values of an attribute in an ascending order.

*Syntax: select A*

*from r*

*order by A*

R= Relation/Table

A= Attribute/Column name

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: select first\_name*

*froom account*

*order by first\_name*

1. Select execute query option or press F5.

**[09].Order by DESE Operator:**

Used to sort the values of an attribute in an descending order.

*Syntax: select A*

*from r*

*order by A dese*

R= Relation/Table

A= Attribute/Column name

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: select first\_name*

*froom account*

*order by first\_name dese*

1. Select execute query option or press F5.

**[10]. Auto Generated Primary Key:**

Primary key is used to uniquely identify the records of a table.

*Syntax: create table r (A1 D1 primary key identify, A2 D2……………….An Dn)*

R= Relation/Table

A=Attribute/Column name

D= Domain/Data type

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: create table account*

*(*

*customer\_id int primary key,*

*customer\_name varchar(25)*

*customer\_city varchar(25)*

*balance numeric(12,2)*

*)*

1. Select execute query option or press F5.
2. After inserting values.

|  |  |  |  |
| --- | --- | --- | --- |
| customer\_id | customer\_name | customer\_city | balance |
| 433133 | Md. Moshiur Rahman | Gopalganj | 150000.00 |
| 433134 | Md. Asifur Rahman | Dinajpur | 210000.00 |

**[11]. Foreign Key:**

If the primary key of one table is used in another table them it is called foreign key.

*Syntax: create table r2 (A1 D1 foreign key references r1 , A2 D2……………….An Dn)*

R= Relation/Table

A=Attribute/Column name

D= Domain/Data type

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: create table loan*

*(*

*customer\_id int foreign key references account,*

*loan\_amount int,*

*loan\_date datetime,*

*loan\_category varchar(30)*

*)*

1. Select execute query option or press F5.
2. After inserting values.

|  |  |  |  |
| --- | --- | --- | --- |
| customer\_id | loan\_amount | loan\_date | loan\_category |
| 433133 | 100000 | 1/1/2013 | car loan |
| 433134 | 150000 | 1/5/2013 | house loan |

**[12]. Updating table value using the primary key and foreign key relationship:**

*Syntax: update r1*

*set (changed attribute values)*

*from r1, r2*

*where r1.a1 = r2.a2*

R= Relation/Table Name

A= Attribute

Steps:

Previously created two tables are.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | d |  | a1 | b1 | c1 | d1 | e1 |
| 1 | m | n | O |  | 1 | m1 | n1 | o1 | 40 |
| 2 | X | y | z |  | 2 | x1 | y1 | z1 | 60 |
| 3 | t | u | v |  | 3 | t1 | u1 | v1 | 80 |

Table A Table B

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: Update A*

*Set b=B.b1,*

*c=B.c1,*

*d=B.d1*

*From A,B*

*Where A.a=B.a1*

*and B.e1>60*

1. Select execute query option or press F5.
2. After updating.

|  |  |  |  |
| --- | --- | --- | --- |
| a | b | c | d |
| 1 | m | n | O |
| 2 | X | y | z |
| 3 | t1 | u1 | v1 |

**[13]. View Creation:**

*Syntax: create view view\_name as*

*select A1, A2*

*From r*

*Where p*

R= Relation/Table

A= Attribute/Column name

P= Predicate

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: create customer\_view as*

*Select customer\_name, customer\_city*

*From customer*

*Where customer\_city= ’Dhaka’*

1. Select execute query option or press F5.
2. To show the result write the sql command given below:

Select \* from customer\_view

[14]. Student Information Database:

Steps:

1. Right click on the database name.
2. Select new query option.
3. Write the sql command given below:

*Ex: create customer\_view as*

*Select customer\_name, customer\_city*

*From customer*

*Where customer\_city= ’Dhaka’*

1. Select execute query option or press F5.
2. To show the result write the sql command given below:

Select \* from customer\_view

**Index**

|  |  |  |
| --- | --- | --- |
| **Serial No** | **Topics** | **Page No** |
| 01 | **Data Definition Language (DDL):**   1. Create Statement: 2. Drop Statement: 3. Alter add Statement: 4. Alter add Statement: | 01 |
| 02 | **Data Manipulation language (DML):**   1. Select Statement: 2. Insert Statement: 3. Delete Statement: 4. Update Statement: | 03 |
| 03 | **SQL Set Operation:**   1. Union All Operation: 2. Union Operation: 3. Intersect Operation: 4. Except Operation: | 05 |
| 04 | **SQL Aggregate Function:**   1. SUM Function: 2. AVG Function: 3. MAX Function: 4. MIN Function: 5. Count Function: | 07 |
| 05 | Distinct Operator: | 09 |
| 06 | AND Operator: | 09 |
| 07 | OR Operator: | 10 |
| 08 | Order by ASC Operator: | 10 |
| 09 | Order by DESE Operator: | 11 |
| 10 | Auto Generated Primary Key: | 12 |
| 11 | Foreign Key: | 13 |
| 12 | Updating table value using the primary key and foreign key relationship: | 14 |
| 13 | View Creation: | 15 |
| 14 | Student Information Database: | 16 |

**Summary**

Joining data in CartoDB is a very common task. Not all joins are equal though, and the one you use is going to depend a lot on your data and what you want to create. Here we are going to walk you through some common JOIN methods on CartoDB. In each section we will show you how to join the data using SQL on-the-fly and then show you how to write the result of a join into your table.

**Join two tables by a shared value in each row**

This is the type of join you perform when you have a value in two tables (say country ISO codes) and you want to match rows from one table to the next (e.g. iso\_code='USA') from one table with the same value in a second table (e.g. iso='USA'). Column name doesn't matter, just the content!

table\_1

|  |  |  |
| --- | --- | --- |
| **cartodb\_id** | **the\_geom** | **iso\_code** |
| 1 | Polygon... | USA |
| 2 | Polygon... | BRA |
| 3 | Polygon... | BEL |
| 4 | Polygon... | GAB |

table\_2

|  |  |  |
| --- | --- | --- |
| **cartodb\_id** | **iso** | **population** |
| 1 | PBRA | 302 |
| 2 | USA. | 188 |
| 3 | GAB | 99 |
| 4 | BEL | 876 |

The above tables give a good example when you will want to use this method. Say you have some nice polygons for your world borders stored in table\_1, but you have a CSV with a value you need for your map and you've uploaded it seperately to table\_2. Here is the join command you would run:



SELECT table\_1.the\_geom,table\_1.iso\_code,table\_2.population

FROM table\_1, table\_2

WHERE table\_1.iso\_code = table\_2.iso

You'll see, it doesn't matter that the columns in the two tables were different, iso\_code versus just, iso, we still ran the join just fine. You could now use Table from query under the Options menu to create a new table with this data. Otherwise, we can just write it into the first table. Here's how.

First, create a new column in table\_1 called 'population' and make it of type 'number'. You can do this by clicking the drop down arrow at the top of any column and selecting Add new column.

Next, check in table\_2 and be sure that under the population column, the type is Number. If it says String, just change it. Click String and then choose Number, and accept the warning. Next run this SQL:



UPDATE table\_1

SET population = table\_2.population

FROM table\_2

WHERE table\_1.iso\_code = table\_2.iso

There is another variation that will do the same thing, but sometimes you will find it simpler to write:



UPDATE table\_1 as t1

SET population = (

SELECT population

FROM table\_2

WHERE

iso = t1.iso\_code

LIMIT 1

)

In the second exampe we run a subquery, so for each row of table\_1 it runs the query inside the parentheses. You'll notice some neat tricks in the example query. We use t1 as an alias for table\_1, so we don't have to write the full name multiple times. We also use LIMIT to make sure that the second query only gives back one result per row, otherwise we might get an error if the second table contained multiple values for some countries. Depending on what your data is, it may not be good to add a LIMIT because you will want to somehow combine all the resulting rows into one answer. If that is your case, keep reading!

**Join two tables by aggregating the shared values in a second table**

This is the case where you have many values in a second table, and you want to get some collection of their values where a shared column match the first table. Here we are talking about examples such as, SUM, AVG, MIN, MAX, etc. We will do just like we did above, but in this case we use a function to aggregate all values in the second table.

table\_1

|  |  |  |
| --- | --- | --- |
| **cartodb\_id** | **the\_geom** | **iso\_code** |
| 1 | Polygon... | USA |
| 2 | Polygon... | BRA |
| 3 | Polygon... | BEL |
| 4 | Polygon... | GAB |

table\_2

|  |  |  |  |
| --- | --- | --- | --- |
| **cartodb\_id** | **iso** | **day** | **total** |
| 1 | BRA | m | 4 |
| 2 | BRA | m | 5 |
| 3 | BRA | w | 2 |
| 4 | USA | m | 2 |
| 5 | USA | f | 1 |

Here is our new example data. We want to get the sum of all totals in each country. So, the SQL would look like this:



SELECT

table\_1.the\_geom,

table\_1.iso\_code,

SUM(table\_2.total) as total

FROM table\_1, table\_2

WHERE table\_1.iso\_code = table\_2.iso

GROUP BY table\_1.iso\_code, table\_2.iso

The biggest change now is the use of the GROUP BY method. This collapses all rows that have a shared 'iso' value, and then using SUM it sums up all the values in total from those collapsed rows! Nice, right? Now, lets add a column to table\_1 called total and make it numeric. Now to do the update version of the query:



UPDATE table\_1 as t1

SET total = (

SELECT SUM(total)

FROM table\_2

WHERE iso = t1.iso\_code

GROUP BY iso

)

Pretty simple. We can do this all day long!

[Here](http://www.postgresql.org/docs/9.1/static/functions-aggregate.html) are some other types of functions other than SUM you might be interested in.

**Join two tables by geospatial intersection!**

One of the most exciting joins is done by using geospatial intersections. If you have a set of points in one table, and a set of state polygons with iso\_codes in a second. You could use a geospatial intersection to give each point an ISO code based on the state they fall in. We can also of course combine COUNT, SUM, AVG, MIN, MAX and all that good stuff here too! Here is some example data:

table\_1

|  |  |  |
| --- | --- | --- |
| **cartodb\_id** | **the\_geom** | **iso\_code** |
| 1 | Polygon... | USA |
| 2 | Polygon... | BRA |
| 3 | Polygon... | BEL |
| 4 | Polygon... | GAB |

table\_2

|  |  |
| --- | --- |
| **cartodb\_id** | **the\_geom** |
| 1 | -58.299992, -33.989619 |
| 2 | -56.709986, -34.349959 |
| 3 | -56.48602, -30.415987 |
| 4 | -57.599957, -30.259614 |

Let's say we just want to know the total number of points from table\_2 that fall in table\_1. Easy, let's see the SQL:



SELECT table\_1.the\_geom,table\_1.iso\_code,COUNT(\*) as count

FROM table\_1, table\_2

WHERE ST\_Intersects(table\_1.the\_geom, table\_2.the\_geom)

GROUP BY table\_1.the\_geom, table\_1.iso\_code

The ST\_Intersects function is one you are going to use again and again. You'll love it. Now, let's do the same but insert the result into a new column. Start by adding a column to table\_1 called total and make it numeric. Next, run:



UPDATE table\_1 as t1

SET total = (

SELECT COUNT(\*)

FROM table\_2

WHERE ST\_Intersects(the\_geom, t1.the\_geom)

)

Nifty, right? You can now use this in combination with SUM, AVG, MAX and all that good stuff to get the values you need from one table into the next.

**Join two tables dynamically**

Just a note about doing the above for a map you build on Leaflet or Google Maps. You may not always want to write the result into table\_1, instead, you may want to query data live from the browser based on something the user is doing. In those cases, use the SELECT statements. If you are rendering a map with the results, you just need to remember to include the\_geom\_webmercator is all! It is our hidden reprojection of the\_geom that makes the final map speedy. Here is the above query modified to include it:



SELECT

table\_1.the\_geom,

table\_1.iso\_code,

COUNT(\*) as count,

table\_1.the\_geom\_webmercator

FROM table\_1, table\_2

WHERE ST\_Intersects(table\_1.the\_geom, table\_2.the\_geom)

GROUP BY table\_1.the\_geom, table\_1.iso\_code

**Join two real datasets**

Now, let's run through an example using a couple real datasets. Start by getting two tables ready in your CartoDB account. To find them, go to your account dashboard. In the top menu click the Common data option. Find the dataset called 'World Rivers' and click in the plus sign besides the element. This will load the data and take you to the resulting table. If you click on Map view, you'll see that it is a basic map of some of the worlds large rivers. This comes from [Natural Earth Data](http://naturalearthdata.com). Take note of the name of the table that was created, in our case it was table\_50m\_rivers\_lake\_cen

Next, go back to your Dashboard by clicking the back link in the upper left. Repeat the process importing a different table from the Common data. This time you have to import 'World Borders' dataset. When the table finishes loading click Map View, and you'll see that it is a dataset of all country borders. Let's join the rivers with the countries so we can make a choropleth of the total length of big rivers in countries around the world. Of course, our map is going to ignore all the little rivers not included in our dataset, but this is just an example!

**Joining the data**

From inside your country borders table create a new column to hold some numerical data. In the Table view, click the drop down arrow beside any regular column name and then click Add new column.

In the options, add big\_rivers as the column and select Number as the type. Finally, click 'Create column'.

**Join through a SELECT statement**

Now, let's do a SELECT to see the data joined:



SELECT

tm\_world\_borders\_s.cartodb\_id,

COUNT(table\_50m\_rivers\_lake\_cen.cartodb\_id)

FROM tm\_world\_borders\_s, table\_50m\_rivers\_lake\_cen

WHERE ST\_Intersects(

tm\_world\_borders\_s.the\_geom,

table\_50m\_rivers\_lake\_cen.the\_geom)

GROUP BY tm\_world\_borders\_s.cartodb\_id

In the above query, we are counting all the rivers that intersect a country. Be sure that your world borders table is named tm\_world\_borders\_s. We GROUP BY the country name so that as a result we get country name and a count of big rivers. Great!

**Updating a table using a JOIN**

Now lets run a similar query, but write the numeric value to the column we created earlier, big\_rivers. Run the following:



UPDATE tm\_world\_borders\_s wb

SET big\_rivers = (

SELECT COUNT(table\_50m\_rivers\_lake\_cen.cartodb\_id)

FROM table\_50m\_rivers\_lake\_cen

WHERE ST\_Intersects(

wb.the\_geom,

table\_50m\_rivers\_lake\_cen.the\_geom

))

In the above query, we are running the UPDATE to our new column big\_rivers, but running a nested query that selects the count of all rivers that intersect it. Like the above examples, we use an alias for the name of our world borders table name, wb. You can see that the alias is then used when we run the ST\_Intersects function, indicating that for every row in the wb table, we count the rivers that intersect the country geometry. We can check now the column shows the number of rivers that each country intersects with:

**Mapping the results**

Now, go back to the Map View for your world borders table. Click the Style option on the right hand side of your map. In the sidebar that slides out, click the overview image of the Choropleth map. In the menu, for Column select big\_rivers and customize the look & feel of the map.

Now take a look at the map and you can see which countries the most rivers in our dataset pass through.

**Index**

* [Summary](http://developers.cartodb.com/tutorials/joining_data.html#summary)
* [Join two tables by a shared value in each row](http://developers.cartodb.com/tutorials/joining_data.html#join_shared_value)
* [Join two tables by aggregating the shared values in a second table](http://developers.cartodb.com/tutorials/joining_data.html#join_aggregating)
* [Join two tables by geospatial intersection!](http://developers.cartodb.com/tutorials/joining_data.html#join_geospatial)
* [Join two tables dynamically](http://developers.cartodb.com/tutorials/joining_data.html#join_dynamic)
* [Join two real datasets](http://developers.cartodb.com/tutorials/joining_data.html#join_real)

**Materials**

* [**Natural Earth Data**](http://naturalearthdata.com)

**Next tutorial**

* [**Style maps based values in your table**](http://developers.cartodb.com/tutorials/conditional_styling.html)

ALTER TABLE table

ADD

FPK int IDENTITY(1,1),

CD varchar (50)

GO

CREATE VIEW dbo.a AS

(

SELECT

ROW\_NUMBER() OVER (ORDER BY C1 DESC, ID\_p, C2 DESC, C3 DESC, C4 DESC) AS ROWID,

1 AS CDa,

\*

FROM table

)

GO

CREATE VIEW dbo.b AS

(

SELECT DISTINCT

ID\_p, CDa AS NEW\_CD, MIN(ROWID) AS ROWID

FROM a

WHERE C1 = 1

GROUP BY ID\_p, CDa

)

GO

UPDATE c

SET c.CD = d.NEW\_CD

FROM

table AS c LEFT OUTER JOIN

(

SELECT

a.FPK, b.NEW\_CD

FROM

a LEFT OUTER JOIN b

ON a.ID\_p = b.ID\_p AND a.ROWID = b.ROWID

) AS d

ON c.FPK = d.FPK

GO

UPDATE table

SET CD = 0

WHERE table.CD IS NULL

DROP VIEW a, b

ALTER TABLE table DROP COLUMN FPK

GO