Homework | module 2 > week 7 > day 18

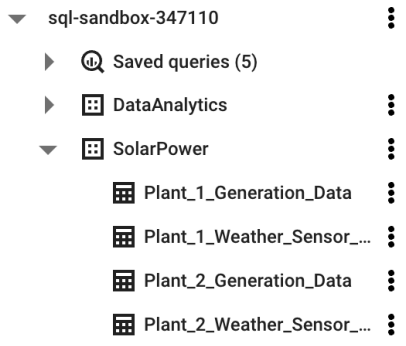
Topics covered: UNION + DDL (CREATE, INSERT, UPDATE, DELETE)

Standard Exercise:

UNION, DATA LOAD & CREATE TABLE AS

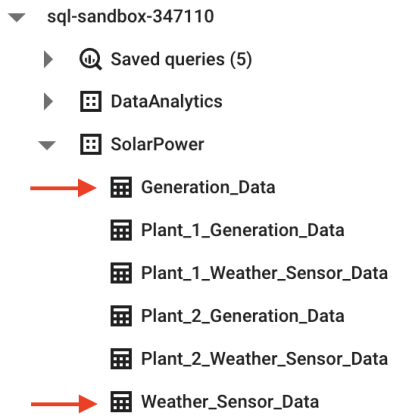
Go over to this Kaggle page, read carefully the description of the dataset as well as the description of each column in the data. There are two sets of data collected for 2 power plants: the “Generation Data” and the ”Weather Sensor Data”:

1. Download the four csv files using the Download button at the top of the page (it will download them into a zipped folder, so make sure you unzip the folder); **FATTO su Big Query**
2. Move over to BigQuery, below your personal project create a new data set and name it “SolarPower”;**FATTO su Big Query**
3. In the newly created data set, upload the four files into four separate tables (give to each table the corresponding file name); in the end you should have a situation similar to the one the screenshot below:



FATTO su Big Query

4. Open a new editor window and write a query that combines the two pairs of datasets to have Plant 1 and Plant 2 in the same table and a total of two resulting tables. CREATE those two TABLEs AS the result of the UNION between the above mentioned tables and call them "Generation_Data" and "Weather_Sensor_Data"; your SolarPower data set should now look like this:




```
Create table SolarPower.Generation_Data as
Select *
From `adept-bond-365418.SolarPower.Plant_1_Generation_Data`
Union All
SELECT *
FROM `adept-bond-365418.SolarPower.Plant_2_Generation_Data`;

Create table SolarPower.Weather_Data as
Select *
From `adept-bond-365418.SolarPower.Plant_1_Sensor_Data`
Union All
SELECT *
FROM `adept-bond-365418.SolarPower.Plant_2_Weather_Sensor_Data`
```

5. How many inverters (*hint*: *source_key*) are there in each plant?

SELECT plant_id, count(distinct source_key) as nr_inverters
FROM SolarPower.Generation_Data
GROUP BY plant_id;

6. How many days of observations do we have for each plant?

SELECT plant_id, count(distinct extract(date from date_time)) as nr_days
FROM SolarPower.Generation_Data
GROUP BY plant_id;

7. Which inverter generated the highest total yield? Which plant does it belong to? *Hint: careful with the aggregation function you use with the total_yield field, read carefully its description on the kaggle page.*

```
SELECT plant_id, source_key, max(total_yield) as total_yield
FROM SolarPower.Generation_Data
GROUP BY plant_id, source_key
ORDER BY total_yield desc;
```

DATA DEFINITION LANGUAGE

Using the w3school environment, create the following two tables (from the previous lesson on Joins) specifying the correct schemas (column names, data types, etc)

1. Create the "students" table:

   students

Row	nmStudent	idCourse
1	Mark	1
2	Jack	2
3	Ivan	3
4	Beth	3
5	Sara	6

```
CREATE TABLE Students(
row int ,
nmStudent varchar(255),
id_Course int not null
);

Insert into Students ( row, nmStudent, id_Course)
values ( '1', 'Mark', '1' ),
    ( '2', 'Jack', '2' ),
    ( '3', 'Ivan','3' ),
    ('4', 'Beth', '3' ),
    ( '5', 'Sara', '6' );
```

2. Create the “courses” table:

Row	idCourse	nmCourse
1	1	Math
2	2	English
3	3	Physics
4	4	Business
5	5	History

```
CREATE TABLE Courses(
row int ,
id_Course int not null ,
nmCourse varchar(255)
);
```

```
Insert into Courses ( row, id_Course, nmCourse)
values ( '1', '1', 'Math' ),
    ( '2', '2', 'English' ),
    ( '3', '3','Physics' ),
    ('4', '4', 'Business' ),
    ( '5', '5', 'History' );
```

3. Use all four types of JOINs as we saw them in lesson 2.7.17 and familiarise yourself with how JOINs work and when null values are generated and think about what that happens.

```
Select *
From Students
Join Courses
on Students.id_Course = Courses.id_Course

Select *
From Students
LEFT Join Courses
on Students.id_Course = Courses.id_Course

Select *
From Students
RIGHT Join Courses
on Students.id_Course = Courses.id_Course


Select *
From Students
FULL OUTER Join Courses
on Students.id_Course = Courses.id_Course
```

4. There is an error in the students table, change Sara's idCourse from 6 to 4.
   Update Students
   set id_Course = '4'
   Where id_Course = '6'

5. There is an error also in the courses table, change the name of idCourse = 5 from "History" to "Economics".
   Update Courses
   set nmCourse = 'Economics'
   Where id_Course = '5'

6. There is a new student in the class, his name is "George" and he is going to follow the Economics course; add a new row of data to the students table containing the new student's information.
   Insert into Students (row, nmStudent, id_Course)
   Values ('6', 'George', '5');

7. How have these changes affected the relationship between the two tables?
   a. Think about it and try to visualise in your head how the results from the four types of JOINs will change now.
   b. Then replicate the work you did at point 3. with the new tables.

Select *
From Courses
Left Join Students
on Courses.id_Course = Students.id_Course

Advanced Exercise (optional):

Using the "bigquery-public-data.thelook_ecommerce" data set, perform the following tasks:

1. JOIN the order_item table with the products table and return the joined table

```
SELECT *
 FROM `bigquery-public-data.thelook_ecommerce.order_items` as order_items
 LEFT JOIN `bigquery-public-data.thelook_ecommerce.products` as products
 on  order_items.product_id = products.id
```

2. Write a query that shows each product categories sorted by *number of orders* per category

```
SELECT category, count(distinct order_id) as orders_d
FROM `bigquery-public-data.thelook_ecommerce.order_items` a
LEFT JOIN `bigquery-public-data.thelook_ecommerce.products` b
 on a.product_id = b.id
group by category
order by orders_d desc;
```

3. Using the last query as the base, add the average margin per order for each category. Notice how top selling categories may not have the highest relative margins. (*Hint*: *margin = retail_price - cost)*

```
SELECT category,
 count(distinct order_id) as orders_d,
 (SUM(b.retail_price) - SUM(b.cost))/count(distinct order_id) as avg_margin_per_order
FROM `bigquery-public-data.thelook_ecommerce.order_items` a
LEFT JOIN `bigquery-public-data.thelook_ecommerce.products` b
on a.product_id = b.id
group by category
order by avg_margin_per_order desc;
```

4. Which product(s) is/are the most popular (number sold)?

```
SELECT name,
 count(product_id) as nr_count
FROM `bigquery-public-data.thelook_ecommerce.order_items` a
LEFT JOIN `bigquery-public-data.thelook_ecommerce.products` b
 on a.product_id = b.id
GROUP BY name
ORDER BY nr_count desc;
```

5. Which product takes the longest time (in terms of number of minutes) from shipping to delivery?

```
SELECT name,
 avg(date_diff(delivered_at, shipped_at, minute)) as avg_hr_ship_deliver,
--  avg((TIME(delivered_at) - TIME(shipped_at))/60),
 count(product_id) as nr_count
FROM `bigquery-public-data.thelook_ecommerce.order_items` a
LEFT JOIN `bigquery-public-data.thelook_ecommerce.products` b
 on a.product_id = b.id
GROUP BY name
ORDER BY avg_hr_ship_deliver desc;
```

6. Which product takes the shortest time?

```
SELECT name,
 avg(date_diff(delivered_at, shipped_at, minute)) as avg_hr_ship_deliver,
 count(product_id) as nr_count
FROM `bigquery-public-data.thelook_ecommerce.order_items` a
LEFT JOIN `bigquery-public-data.thelook_ecommerce.products` b
 on a.product_id = b.id
GROUP BY name
ORDER BY avg_hr_ship_deliver asc;
```

7. Did you notice something strange in the last result? You should have seen some *null* values among the "avg_hr_ship_deliver" variable, why do you think that is? Write a query that checks if there are any null values in "delivered_at". What do you think a *null* value means in such a column?

8. Back to the query at point 6, add a WHERE condition that excludes *items that have not been delivered* yet:

```
SELECT name,
 avg(date_diff(delivered_at, shipped_at, minute)) as avg_hr_ship_deliver,
 count(product_id) as nr_count
FROM `bigquery-public-data.thelook_ecommerce.order_items` a
LEFT JOIN `bigquery-public-data.thelook_ecommerce.products` b
 on a.product_id = b.id
WHERE delivered_at is not null
GROUP BY name
ORDER BY avg_hr_ship_deliver asc;
```

9. When you answered question five, all those items that took the longest time from shipping to delivery were only ordered one or two times; answer to the same question but only for those products that were sold at least 5 times.

```
SELECT name,
 avg(date_diff(delivered_at, shipped_at, minute)) as avg_min_ship_deliver,
 count(product_id) as nr_count
FROM `bigquery-public-data.thelook_ecommerce.order_items` a
LEFT JOIN `bigquery-public-data.thelook_ecommerce.products` b
 on a.product_id = b.id
WHERE delivered_at is not null
GROUP BY name
HAVING count(product_id) > 5
ORDER BY avg_min_ship_deliver desc;
```