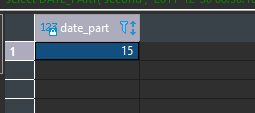
# Lecture 6

## 03/11/2021

Date part function

|  |
| --- |
| select DATE\_PART('second', '2011-12-30 08:56:10'::timestamp - '2011-12-30 08:54:55'::timestamp); |



Input:

* Varchar denoting the increment
* Two timestamp values between which the difference needs to be calculated

Note: In the example above the timestamp cast is being done by the ‘::’ operator

Output

An integer number of seconds giving the difference between the two timestamp values.

**Grain:** denotes the lowest denominator of the date; what a single row should represent in the dataset

Example: What is the daily trip count?

-grain/row: daily observation count

What date column to use to establish the grain?

Tpep\_pickup\_datetime → date only

|  |
| --- |
| select date\_trunc('day', tpep\_pickup\_datetime) from qcmath290.public."2018\_Yellow\_Taxi\_Trip\_Data\_gz" yttdg |

Input:

* Varchar value denoting the grain of your datetime
* Timestamp data type denoting the actual variable that needs to be truncated

Output: Timestamp data type truncated to the value that we established in the variable

Question: What is the daily trip count?

Sub Questions:

* What is the grain of the question?
  + tpep\_pickup\_datetime -> date only
    - How can I ensure that tpep\_pickup\_datetime returns date value only?
      * Using the date\_trunc()

|  |
| --- |
| select date\_trunc('day', tpep\_pickup\_datetime) as tpep\_pickup\_date  ,count(\*)  from qcmath290.public."2018\_Yellow\_Taxi\_Trip\_Data\_gz" yttdg  group by date\_trunc('day', tpep\_pickup\_datetime) |

Output: Two columns: Date truncated to day of the year, Count associated with the date

Question: What is the average daily trip count in 2018?

* Grain:
  + Year (top level)
  + Aggregate daily trip count

Note: Try to name CTE as explicitly as possible.

|  |
| --- |
| EXTRACT (YEAR FROM tpep\_pickup\_datetime) |

Input:

* Target extract date grain (ex: year, quarter, day, etc.).
* Timestamp column
  + If not timestamp use TIMESTAMP prefix to cast it as the timestamp

|  |
| --- |
| **SELECT** EXTRACT(**YEAR** **FROM** **TIMESTAMP** '2016-12-31 13:30:15'); |

Question: The average daily trip count in 2018?

* grain / row: year
  + aggregate daily trip count

|  |
| --- |
| with daily\_trip\_count as (  select  date\_trunc('day', tpep\_pickup\_datetime) as tpep\_pickup\_date ,  count(\*) trip\_count  from  qcmath290.public."2018\_Yellow\_Taxi\_Trip\_Data\_gz" yttdg  where  extract(year from tpep\_pickup\_datetime)= 2018  group by  date\_trunc('day', tpep\_pickup\_datetime) )  select  avg(trip\_count)  from  daily\_trip\_count |

Inner Query: Gives the aggregate daily trip count.

Outer Query: Calls the average function and averages those values using the information from the inner query. Sum of daily observations/ Number of days (366)

Question: The average daily distinct trip count in 2018?

* Grain/row: year
  + Aggregate distinct daily trip count
    - Distinct trips

|  |
| --- |
| with distinct\_trips as (  select  distinct \*  from  qcmath290.public."2018\_Yellow\_Taxi\_Trip\_Data\_gz"  where  extract(year from tpep\_pickup\_datetime)= 2018 ),  daily\_trip\_count as (  select  date\_trunc('day', tpep\_pickup\_datetime) as tpep\_pickup\_date ,  count(\*) trip\_count  from  distinct\_trips  where  extract(year from tpep\_pickup\_datetime)= 2018  group by  date\_trunc('day', tpep\_pickup\_datetime) )  select  avg(trip\_count)  from  daily\_trip\_count |

Step 1

* Using the distinct command, we established unique rows.
* Using the WHERE clause, we filtered out records that pertain to 2018 only.

Step 2

* We truncated the date values to obtain a daily column
* Used the count(\*) to calculate distinct trips.

Step 3

* Using the average function, we gained the average daily distinct count.

We essentially broke it into 3 steps and developed a CTE on that basis.

* Establish distinct\_trips
  + Referenced distinct\_trips in a subsequent CTE daily\_trip\_count
* We exited the CTE and referenced daily\_trip\_count with the average function.

Question: Average daily distinct trips as it pertains to April 2018

* grain / row: year
  + aggregate distinct daily trip count
    - distinct trips

|  |
| --- |
| select  distinct \*  from  qcmath290.public."2018\_Yellow\_Taxi\_Trip\_Data\_gz"  where  extract(year from tpep\_pickup\_datetime)= 2018  and extract(month from tpep\_pickup\_datetime)= 4 |

Note:

*Order:* [*https://www.postgresql.org/docs/9.5/sql-select.html*](https://www.postgresql.org/docs/9.5/sql-select.html)*: see the description for the order*

Step 1

* When a SQL statement is being executed, the order at which it is being executed is FROM first.
  + Identifying target object you are referencing
* The WHERE clause restricting the data followed by the SELECT and GROUP BY.
  + In the WHERE clause, we limited the number of records that the statement needs to do distinct on. This is why the query took a short amount of time.

**Lecture 07** (03/18/2021)

**Order of operations in a query**: See the [Postgres documentation](https://www.postgresql.org/docs/9.5/sql-select.html).

1. Evaluate CTE
2. From
3. Where
   1. The smaller the query, the faster it will run.
   2. If you are running into performance issues, make a tighter where clause.
   3. Even if using the where clause causes a slightly slower query, all future queries/operations that use the filtered result will be much faster.
4. Group By
   1. The smaller the data you’re grouping, the faster the query will run.
5. Having
6. Select
7. Select Distinct
   1. Note when Aliasing of columns occurs in regards to the order of operations.
   2. Aliasing of columns occurs in the select clause; if we try to use the alias in the: from, where, or group by clause (or anything earlier), it won’t know what the alias references (SQL Error [42703]).
8. Union/Intersect/Except
   1. Each of these operations returns a set.
9. Order by
   1. Can use column aliases in this statement.
   2. Ordering is expensive
      1. If the data is filtered, there are less rows, and the ordering will be cheaper.
   3. [Ordering in Postgresql](https://wiki.postgresql.org/images/5/59/Sorting_through_the_ages.pdf) and how it is done.
10. Limit
    1. Restricts the number of rows that are returned
11. Update

|  |
| --- |
| select \*,  tpep\_dropoff\_datetime :: timestamp as tpep\_dropoff\_timestamp  from "2018\_Yellow\_Taxi\_Trip\_Data\_gz"  where tpep\_dropoff\_datetime :: timestamp = timestamp'2018-04-12 20:52:08'  order by tpep\_dropoff\_timestamp |

|  |
| --- |
| select \*,  tpep\_dropoff\_datetime :: timestamp as tpep\_dropoff\_timestamp  from "2018\_Yellow\_Taxi\_Trip\_Data\_gz"  where tpep\_dropoff\_datetime :: timestamp >= timestamp'2018-04-12 00:00:00'  and tpep\_dropoff\_datetime :: timestamp < timestamp'2018-04-13 00:00:00'  order by tpep\_dropoff\_timestamp  limit 200 |

Observations and explaining the order of execution:

1. FROM
   1. "2018\_Yellow\_Taxi\_Trip\_Data\_gz" is knowing the \*, tpep\_dropoff\_datetime -> only the columns table
2. Where
   1. Filter out the rows that don’t match our criteria.
   2. The comparison will be executed.
   3. Converts the drop\_off time (varchar) and the boundary time (varchar) to a timestamp.
   4. Don’t compare timestamps as strings; there can be issues with the formatting[[1]](#footnote-0).
3. Select
   1. Select all columns using the \* operator, and we do a cast from varchar to timestamp, and then we alias the column name.
4. Order by
   1. Order the restricted results set by the aliased column name
5. Limit
   1. We only return the first 200 rows.
   2. This will be faster and can help with debugging[[2]](#footnote-1).

**Aggregation with View**

Once we have the logic down, we want to finalize that logic with a view

|  |
| --- |
| create view average\_daily\_distinct\_trip\_count as  with distinct\_trips as (  select  distinct \*  from  qcmath290.public."2018\_Yellow\_Taxi\_Trip\_Data\_gz"  where  extract(year from tpep\_pickup\_datetime)= 2018 ),  daily\_trip\_count as (  select  date\_trunc('day', tpep\_pickup\_datetime) as tpep\_pickup\_date ,  count(\*) trip\_count  from  distinct\_trips  where  extract(year from tpep\_pickup\_datetime)= 2018  group by  date\_trunc('day', tpep\_pickup\_datetime) )  select  avg(trip\_count)  from  daily\_trip\_count  ; |

When you create a view, what happens is that you associate the **logic** with a view, independent of the schema. I.e you have to recalculate the logic (sql script) each time on the associated table.

If we have a complex logic -> it will take longer for the view to execute

A view is not a materialized CTE; material is used in regard to whether the object is physical or not. Something that occupies a significant amount/calculable amount of storage.

In general, we want our views to be simple.

For recording purposes, it is not a good idea to use views.

Aggregation with Tables:

If we don’t like views, why not materialize the view into a table?

??

When to use a view:

* Time permits
  + Either quick, overnight it etc.
* When I only care about the results of the view once.
* Store **transformation logic**.
  + Cast a varchar to a timestamp, that is a transformation logic.
  + Calculations can be calculation logic.

|  |
| --- |
| create table average\_daily\_distinct\_trip\_count\_table as  with distinct\_trips as (  select  distinct \*  from  qcmath290.public."2018\_Yellow\_Taxi\_Trip\_Data\_gz"  where  extract(year from tpep\_pickup\_datetime)= 2018 ),  daily\_trip\_count as (  select  date\_trunc('day', tpep\_pickup\_datetime) as tpep\_pickup\_date ,  count(\*) trip\_count  from  distinct\_trips  where  extract(year from tpep\_pickup\_datetime)= 2018  group by  date\_trunc('day', tpep\_pickup\_datetime) )  select  avg(trip\_count)  from  daily\_trip\_count  ; |

To materialize a view, it will take a *very* long time the first time, but then each subsequent time it will be very quick since the data is there already.

Aggregation with Tables & Views

1. Build CTE
2. Build a View for the CTE
   1. CTE is just a string of code
   2. A View is a script/can be used in a select statement

CTE and View is based on the current status of the table.

* If the underlying table data is volatile, you should probably use a CTE/View
* If the data is slow moving, use a table
* If the data is big and fast, need to use an ETL

**Materializing Views**

create table <new table name> as

select \* from <view name>;

In a reporting setting, it is usually better to drop the table and then recreate the table from the view instead of updating. In an application ***DON’T DO THIS[[3]](#footnote-2)***. This pattern works in a limited fashion.

|  |
| --- |
| drop table average\_daily\_distinct\_trip\_count\_table\_from\_view;  create table average\_daily\_distinct\_trip\_count\_table\_from\_view as  select \* from average\_daily\_distinct\_trip\_count\_table; |

**Relationships in RDBMS**

* One-to-one(1:1)
  + A tuple in relation A can be associated with at most one (zero or 1) tuples in relation B
  + A tuple in relation B can be associated with at most one (zero or 1) tuples in relation A.
    - Student ID to ssn#
    - Student table, ssn table
      * Why not make it a new column in the student table?
        + Security reasons.
* One-to-many (1:n)
  + A tuple in relation A can be associated with any number (zero or more) tuples in relation B.
  + A tuple in relation B can be associated with at most one (zero or 1) tuples in relation A.
    - User to Grocery List
      * A user (relation A) can have many grocery lists
      * The grocery list (relation B) can only have 1 owner/user or be in the trash can (no owner).
* Many-to-one (n:1)
  + Same as one-to-many, but inverse relation A and B
* Many to Many (m:n)
  + A tuple in relation A can be associated with any number (zero or more) tuples in relation B
  + A tuple in relation B can be associated with any number (zero or more) tuples in relation A
    - Student ID to Interests
      * A student can have multiple interests, and an interest can be shared amongst multiple students

1. Also, if there is an error with the format of one of the strings, the error won’t be thrown. [↑](#footnote-ref-0)
2. I found this very useful. [↑](#footnote-ref-1)
3. Don’t you dare quote Prof. Zombory on this as good practice in your application. [↑](#footnote-ref-2)