Introductory Google BigQuery with Gemini AI
Exercise 1
Solutions

1 BigQuery and Gemini reference / documentation

Basics of working with BigQuery via the console UI.

Quick start guide for working with loading and querying data as well as querying public datasets.

Official reference for GoogleSQL, the official SQL dialect for Google BigQuery.

Basic guide to using Gemini to assist writing queries,

2 Lab setup for exercises

We will start off by creating another dataset in the existing project which will hold the table that we will be using for this exercise.

We will create a dataset in the current project with the name: `exercise_dataset`
You can skip this step if you wish, since you will access to the dataset that I create, but you can also choose to create your dataset if you wish.

IMPORTANT NOTE: If you are sharing the same project with me (the trainer), all the datasets that you create will be within in the same project. To ensure that the datasets you create are distinguishable from the datasets of other participants of this workshop, please precede the dataset with your name, for e.g. `peter_exercise_dataset`, `jane_exercise_dataset`, etc (make sure you use underscore and not dashes to separate the words).

We will use the `sampletransactions.csv` file in the `data` subfolder of the downloaded workshop resources to populate the contents of a new table in this new dataset.

You can open `sampletransactions.csv` in Excel to quickly preview it first if you wish. This file contains dummy data for online trading transactions conducted by several users over several years. The fields / columns in this table are briefly explained as below. Note that the meaning of these fields / columns and their possible values may be different from a real life online trading transaction dataset.

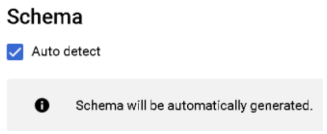
Column	Meaning
`trade_id`	Unique identifier for each trade
`user_id`	Unique identifier for each user. There are currently five users
`platform`	Trading platform. Here, we reference 5 of the main global stock exchanges: NYSE, Nasdaq, SSE (Shanghai Stock Exchange), LSE (London Stock Exchange), Euronext
`currency`	Currency used for the trade, 4 options: USD, EUR, CNY, GBP
`instrument`	Financial instrument being traded. The possible options include: stocks, bonds, ETF, futures, options, CFD, forex, commodities, REITS, mutual
`trade_type`	long or short
`entry_price`	Price at which the order was entered
`exit_price`	Price at which the order was closed
`trade_volume`	Volume or size of the trade
`open_time`	Moment when trader initiates a trade (buying stock, entering forex position, etc). Format: YYYY-MM-DD HH:MM:SS (this aligns with BigQuery's DateTime and also Timestamp data type format, which can optionally include time zone info).
`close_time`	Moment when trade is exited or position is closed (existing forex, selling stock, etc). Format: YYYY-MM-DD HH:MM:SS (this aligns with BigQuery's DateTime and also Timestamp data type format, which can optionally include time zone info).

Create a table in the newly created dataset with the following values in the dialog box that appears.

Create table from: `Upload`
Select file*: `sampletransactions.csv`

File Format: `CSV`
Destination: `intro-bigquery-workshop`
Dataset: `exercise_dataset`
Table: `sampletransactions`
Table type: `Native table`

Tick Auto detect for Schema.



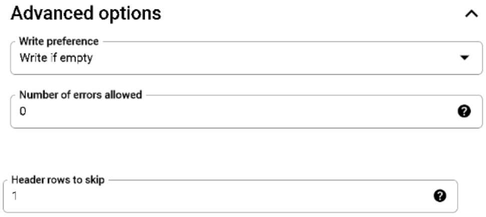
Schema

☑ Auto detect

Schema will be automatically generated.

BigQuery will scan the contents of each column and infer the data type for each column as it imports them into the table that it will create.

In Advanced Options, type 1 for Header Rows to skip as the first row in our CSV file is essentially a header row containing the names of the columns/fields for the table we are creating. You can leave the other options as they are.



Advanced options

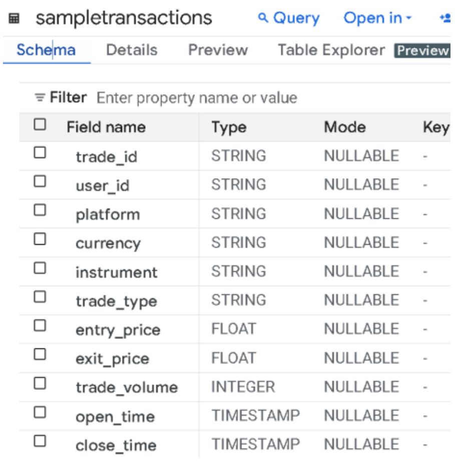
Write preference: Write if empty

Number of errors allowed: 0

Header rows to skip: 1

Finally, click Create Table.
A message about load job running should appear followed by notification about successful creation of the `sampletransactions` table.

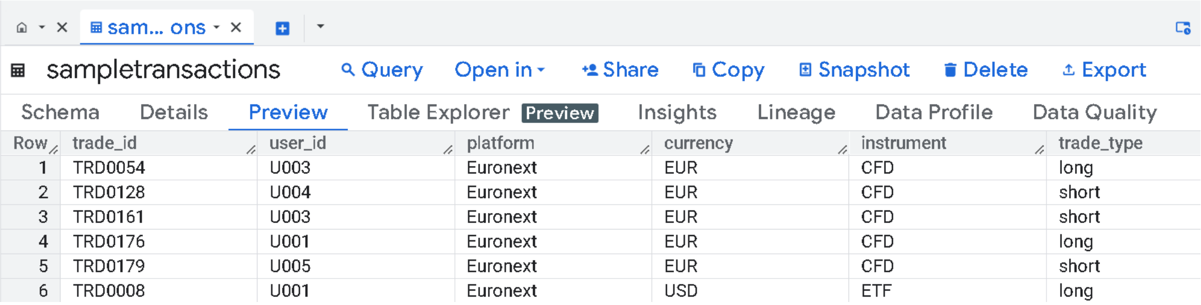
Selecting this table in the Explorer pane should show its Schema in the details pane, where you can see the data types that BigQuery has automatically assigned to each of the fields in the imported CSV file. The Nullable mode indicates this column can contain `null` values (to be covered in a later lab topic)



Field name	Type	Mode	Key
trade_id	STRING	NULLABLE	-
user_id	STRING	NULLABLE	-
platform	STRING	NULLABLE	-
currency	STRING	NULLABLE	-
instrument	STRING	NULLABLE	-
trade_type	STRING	NULLABLE	-
entry_price	FLOAT	NULLABLE	-
exit_price	FLOAT	NULLABLE	-
trade_volume	INTEGER	NULLABLE	-
open_time	TIMESTAMP	NULLABLE	-
close_time	TIMESTAMP	NULLABLE	-

Notice that the data type assigned to `open_time` and `close_time` column is the [TIMESTAMP datatype](), which optionally can include time zones info.

Click on Preview tab in the Details pane to view the first 50 rows in this table.



Row	trade_id	user_id	platform	currency	instrument	trade_type
1	TRD0054	U003	Euronext	EUR	CFD	long
2	TRD0128	U004	Euronext	EUR	CFD	short
3	TRD0161	U003	Euronext	EUR	CFD	short
4	TRD0176	U001	Euronext	EUR	CFD	long
5	TRD0179	U005	Euronext	EUR	CFD	short
6	TRD0008	U001	Euronext	USD	ETF	long

You will see that the `open_time` and `close_time` column both have the UTC time zone (the default time zone or GMT) assigned to them as no time zone data was specified in the original CSV data file.
If you need to specify time zone for date / time values in Google BigQuery, below are some examples of valid time zone values:

`2025-05-12 09:00:00+08:00` (UTC+8)
`2024-10-01 12:00:00-07:00` (UTC-7)
`2024-08-10 13:00:00 America/New_York` (EST)
`2024-09-11 15:00:00 America/Los_Angeles` (PST)
`2023-08-04 22:00:00 Asia/Shanghai` (CST)
`2023-08-04 22:00:00 Asia/Kuala_Lumpur` (CST)

Notice that the rows in this table (based on the TRDxxx sequences) do not appear in the same sequence as the initial data in `sampletransactions.csv`

This is because the job executed by BigQuery to load the data from this file executes in parallel to populate the table with the data, resulting in rows appearing out of the original sequence, just as in the case of the lab session.

2.1 Prompt template for other AI tools

If you are planning to use other AI tools such as ChatGPT, Grok or Claude for your prompts to generate queries, remember to specify the schema of the table first in this prompt template below:

```
I have created a table called sampletransactions in a dataset called exercise_dataset in Google BigQuery with the following schema and data types:

CustID: INTEGER
Date: DATE
FirstName: STRING
LastName: STRING
Region: STRING
State: STRING
ProdCategory: STRING
```

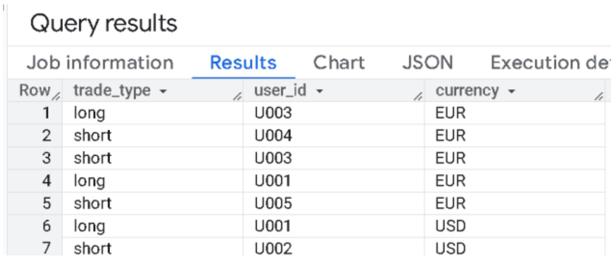
```
Price: FLOAT
Units: INTEGER

Create a GoogleSQL query that ……………………
```

3 Basic SELECT

Q1 Display the following columns for all rows in the table: `trade_type`, `user_id` and `currency` in that specific order.

Sample result:



Row	trade_type	user_id	currency
1	long	U003	EUR
2	short	U004	EUR
3	short	U003	EUR
4	long	U001	EUR
5	short	U005	EUR
6	long	U001	USD
7	short	U002	USD

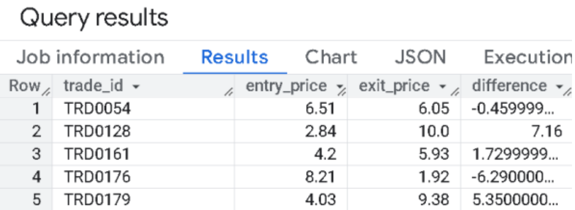
SAMPLE QUERY:

```
SELECT trade_type, user_id, currency
FROM exercise_dataset.sampletransactions;
```

3.1 SELECT with expressions and aliases

Q2 In evaluating a transaction, the price difference is computed as the difference between the exit and entry prices for a particular transaction. Compute this value for all rows in the table and give it a meaningful column name: `difference`

Sample result:



Row	trade_id	entry_price	exit_price	difference
1	TRD0054	6.51	6.05	-0.459999...
2	TRD0128	2.84	10.0	7.16
3	TRD0161	4.2	5.93	1.7299999...
4	TRD0176	8.21	1.92	-6.290000...
5	TRD0179	4.03	9.38	5.3500000...

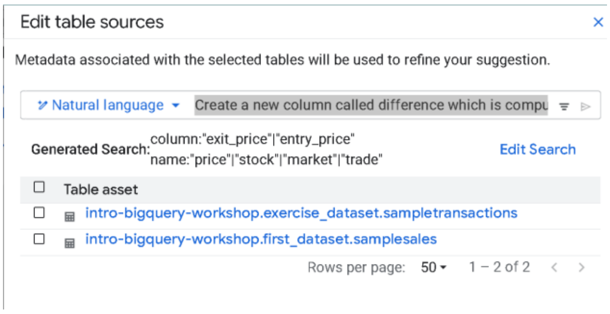
SAMPLE PROMPT:

```
Create a new column called difference which is computed as the
difference between the exit_price and entry_price columns for
sampletransactions. Show only the trade_id, entry_price, exit_price
and difference columns.
```

SAMPLE QUERY:

```
SELECT trade_id, entry_price, exit_price,
exit_price - entry_price as difference
FROM exercise_dataset.sampletransactions;
```

NOTE: The query returned from your prompt may look slightly different from the one shown above. If you do not specify the table name explicitly in your prompt, then the source table (the `FROM` portion of the query) will be by default the latest table that you interacted with.
If this is not the correct table, make sure to select `Edit Table Sources` and select the correct table (in this case `sampletransactions`).



Edit table sources

Metadata associated with the selected tables will be used to refine your suggestion.

Natural language: Create a new column called difference which is compu

Generated Search: column:"exit_price"|"entry_price" name:"price"|"stock"|"market"|"trade"

	Table asset
☐	intro-bigquery-workshop.exercise_dataset.sampletransactions
☐	intro-bigquery-workshop.first_dataset.samplesales

Another particular feature of Gemini is that they might provide queries which include operations that you did not explicitly specify in prompt. A common example might be the formatting of the `close_time` and `open_time` columns (which are in the Timestamp data type format) using the FORMAT_TIMESTAMP function. An example is shown below.

```
  FORMAT_TIMESTAMP('%F %T', close_time) AS close_time,
  FORMAT_TIMESTAMP('%F %T', open_time) AS open_time
```

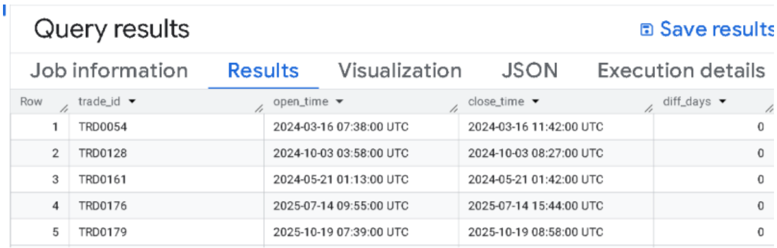
This may occur randomly in some of the query responses you get to your prompt: it does not occur all the time.

Q3 The trade duration is the length of time a trade is held open, essentially the difference between the closing time (`close_time`) and opening time (`open_time`) of a trade transaction. Compute the duration in terms of total hours or total minutes or total seconds for all rows in the table. You will need to write a separate query to compute the duration for each of these (i.e. total of 3 queries).

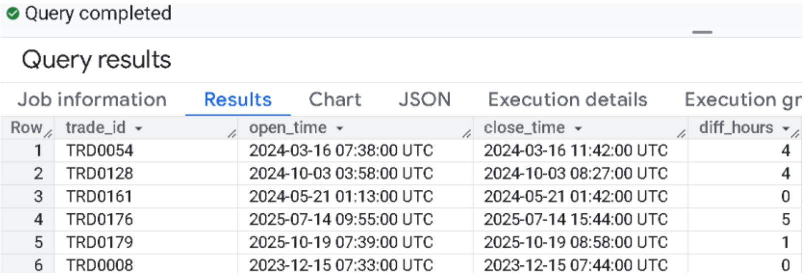
HINT: Both columns `close_time` and opening time `open_time` are of the TIMESTAMP data type in Google BigQuery. For this particular data type, BigQuery offers a large number of functions to work on values from this type. We can use the TIMESTAMP_DIFF function to specify the granularity (DAY, HOUR, MINUTE, SECOND, etc) between closing time (`close_time`) and opening time (`open_time`)
For e.g.

```
TIMESTAMP_DIFF(close_time, open_time, DAY)…..
TIMESTAMP_DIFF(close_time, open_time, HOUR)…..
TIMESTAMP_DIFF(close_time, open_time, MINUTE)…..
```

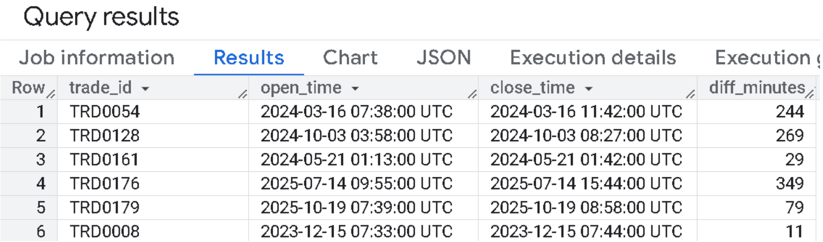
Sample result:



Row	trade_id	open_time	close_time	diff_days
1	TRD0054	2024-03-16 07:38:00 UTC	2024-03-16 11:42:00 UTC	0
2	TRD0128	2024-10-03 03:58:00 UTC	2024-10-03 08:27:00 UTC	0
3	TRD0161	2024-05-21 01:13:00 UTC	2024-05-21 01:42:00 UTC	0
4	TRD0176	2025-07-14 09:55:00 UTC	2025-07-14 15:44:00 UTC	0
5	TRD0179	2025-10-19 07:39:00 UTC	2025-10-19 08:58:00 UTC	0



Row	trade_id	open_time	close_time	diff_hours
1	TRD0054	2024-03-16 07:38:00 UTC	2024-03-16 11:42:00 UTC	4
2	TRD0128	2024-10-03 03:58:00 UTC	2024-10-03 08:27:00 UTC	4
3	TRD0161	2024-05-21 01:13:00 UTC	2024-05-21 01:42:00 UTC	0
4	TRD0176	2025-07-14 09:55:00 UTC	2025-07-14 15:44:00 UTC	5
5	TRD0179	2025-10-19 07:39:00 UTC	2025-10-19 08:58:00 UTC	1
6	TRD0008	2023-12-15 07:33:00 UTC	2023-12-15 07:44:00 UTC	0



Row	trade_id	open_time	close_time	diff_minutes
1	TRD0054	2024-03-16 07:38:00 UTC	2024-03-16 11:42:00 UTC	244
2	TRD0128	2024-10-03 03:58:00 UTC	2024-10-03 08:27:00 UTC	269
3	TRD0161	2024-05-21 01:13:00 UTC	2024-05-21 01:42:00 UTC	29
4	TRD0176	2025-07-14 09:55:00 UTC	2025-07-14 15:44:00 UTC	349
5	TRD0179	2025-10-19 07:39:00 UTC	2025-10-19 08:58:00 UTC	79
6	TRD0008	2023-12-15 07:33:00 UTC	2023-12-15 07:44:00 UTC	11

SAMPLE PROMPTS:

```
Create a new column called diff_days which is computed as the
difference between the close_time and open_time in days. Show only
the trade_id, open_time, close_time and diff_days columns.

Create a new column called diff_hours which is computed as the
difference between the close_time and open_time in hours. Show only
the trade_id, open_time, close_time and diff_hours columns.

Create a new column called diff_minutes which is computed as the
difference between the close_time and open_time in minutes. Show only
the trade_id, open_time, close_time and diff_minutes columns.
```

SAMPLE QUERY:

```
SELECT trade_id, open_time, close_time,
TIMESTAMP_DIFF(close_time, open_time, DAY) AS diff_days
FROM exercise_dataset.sampletransactions;
```

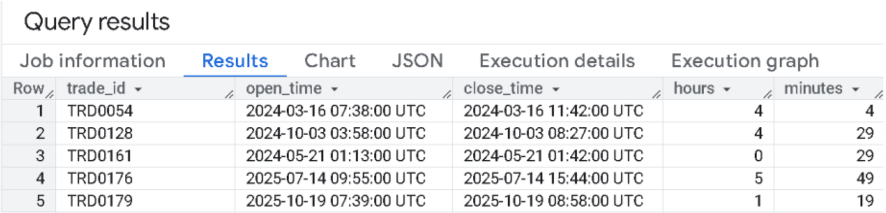
```
SELECT trade_id, open_time, close_time,
TIMESTAMP_DIFF(close_time, open_time, HOUR) AS diff_hours
FROM exercise_dataset.sampletransactions;


SELECT trade_id, open_time, close_time,
TIMESTAMP_DIFF(close_time, open_time, MINUTE) AS diff_minutes
FROM exercise_dataset.sampletransactions;
```

Q4. The previous queries provided the duration in terms of total hours or total minutes or total seconds. This is accurate for mathematical expressions, but may not be so intuitive for human comprehension. Write another query which adds on to your previous queries by using the MOD function in BigQuery to display the duration in terms of both hours and minutes (so for e.g. a duration of 269 minutes is displayed as 4 hours and 29 minutes instead).

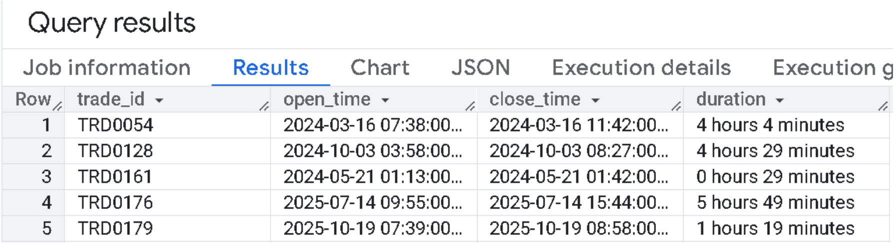
Hint: You can nest the `TIMESTAMP_DIFF` function within the `MOD` function so that the result from the `TIMESTAMP_DIFF` function is used by the MOD function.

Sample result:



Row	trade_id	open_time	close_time	hours	minutes
1	TRD0054	2024-03-16 07:38:00 UTC	2024-03-16 11:42:00 UTC	4	4
2	TRD0128	2024-10-03 03:58:00 UTC	2024-10-03 08:27:00 UTC	4	29
3	TRD0161	2024-05-21 01:13:00 UTC	2024-05-21 01:42:00 UTC	0	29
4	TRD0176	2025-07-14 09:55:00 UTC	2025-07-14 15:44:00 UTC	5	49
5	TRD0179	2025-10-19 07:39:00 UTC	2025-10-19 08:58:00 UTC	1	19

Alternatively, you could make it more readable by concatenating results together into a single string.



Row	trade_id	open_time	close_time	duration
1	TRD0054	2024-03-16 07:38:00...	2024-03-16 11:42:00...	4 hours 4 minutes
2	TRD0128	2024-10-03 03:58:00...	2024-10-03 08:27:00...	4 hours 29 minutes
3	TRD0161	2024-05-21 01:13:00...	2024-05-21 01:42:00...	0 hours 29 minutes
4	TRD0176	2025-07-14 09:55:00...	2025-07-14 15:44:00...	5 hours 49 minutes
5	TRD0179	2025-10-19 07:39:00...	2025-10-19 08:58:00...	1 hours 19 minutes

SAMPLE PROMPT:

```
Compute the difference between the close_time and open_time in terms
of both number of hours and minutes, rather than only hours or minutes.
This difference in terms of hours and minutes should be stored in 2
new columns: hours and mins. Show only the trade_id, open_time,
close_time, hours and mins columns.
```

SAMPLE QUERY:

```
SELECT trade_id, open_time, close_time,
  FLOOR(TIMESTAMP_DIFF(close_time,  open_time,  SECOND)  /  3600)  AS
hours,
  FLOOR(MOD(TIMESTAMP_DIFF(close_time,  open_time,  SECOND),  3600)  /
60) AS mins
FROM exercise_dataset.sampletransactions;
```

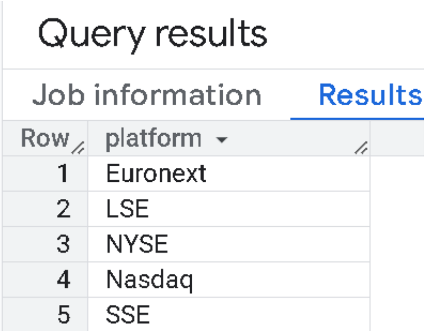
OR

```
SELECT trade_id, open_time, close_time,
TIMESTAMP_DIFF(close_time, open_time, HOUR)
|| ' hours ' ||
MOD(TIMESTAMP_DIFF(close_time, open_time, MINUTE),60)
|| ' minutes ' AS duration
FROM exercise_dataset.sampletransactions;
```

3.2 SELECT with DISTINCT and COUNT

Q5. Find all the distinct values possible for the `platform` column:

Sample result:



Row	platform
1	Euronext
2	LSE
3	NYSE
4	Nasdaq
5	SSE

SAMPLE PROMPT:

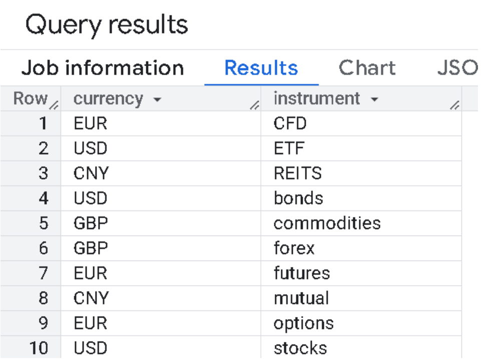
```
Show all the different unique platforms available from
sampletransactions
```

SAMPLE QUERY:

```
SELECT DISTINCT platform
FROM exercise_dataset.sampletransactions;
```

Q6. Select all the unique combination of values possible for the columns `currency` and `instrument`.

Sample result:



Row	currency	instrument
1	EUR	CFD
2	USD	ETF
3	CNY	REITS
4	USD	bonds
5	GBP	commodities
6	GBP	forex
7	EUR	futures
8	CNY	mutual
9	EUR	options
10	USD	stocks

Notice that there is no repetition of values for the instrument column, which means that each particular category of currency has a set of instruments associated with it that are not found in other currency category. This is important to note when we do hierarchical grouping later on.

SAMPLE PROMPT:

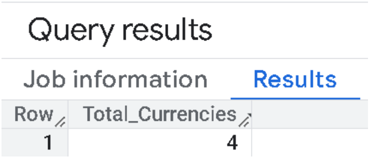
```
Show all the unique combination of values possible for currency and instrument from sampletransactions
```

SAMPLE QUERY:

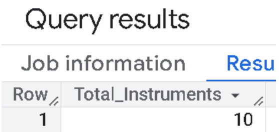
```
SELECT DISTINCT currency, instrument
FROM exercise_dataset.sampletransactions;
```

Q7. Count how many distinct values are available in the `currency` and `instrument` columns, without viewing these values.

Sample result:



Row	Total_Currencies
1	4



Row	Total_Instruments
1	10

SAMPLE PROMPT:

```
Count the total number of unique values possible for currency from sampletransactions

Count the total number of unique values possible for instrument from sampletransactions
```

SAMPLE QUERY:

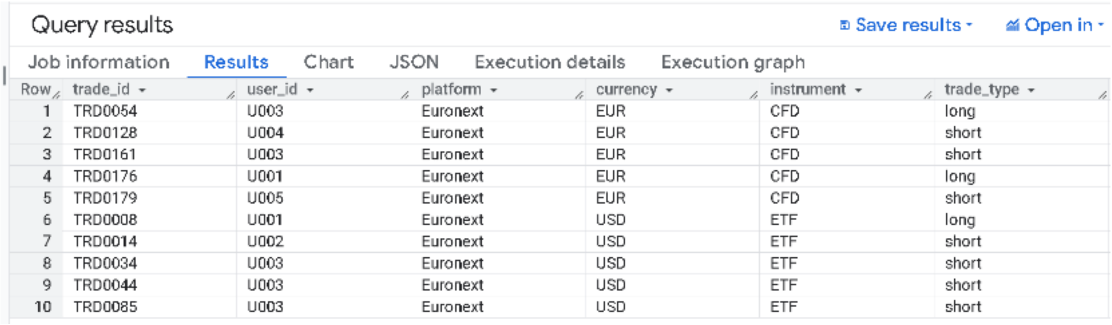
```
SELECT COUNT(DISTINCT currency) AS Total_Currencies
FROM exercise_dataset.sampletransactions;
```

```
SELECT COUNT(DISTINCT instrument) AS Total_Instruments
FROM exercise_dataset.sampletransactions;
```

3.3 SELECT with LIMIT

Q8. Show the first 10 rows with all the columns present from this table.

Sample result:



Row	trade_id	user_id	platform	currency	instrument	trade_type
1	TRD0054	U003	Euronext	EUR	CFD	long
2	TRD0128	U004	Euronext	EUR	CFD	short
3	TRD0161	U003	Euronext	EUR	CFD	short
4	TRD0176	U001	Euronext	EUR	CFD	long
5	TRD0179	U005	Euronext	EUR	CFD	short
6	TRD0008	U001	Euronext	USD	ETF	long
7	TRD0014	U002	Euronext	USD	ETF	short
8	TRD0034	U003	Euronext	USD	ETF	short
9	TRD0044	U003	Euronext	USD	ETF	short
10	TRD0085	U003	Euronext	USD	ETF	short

SAMPLE PROMPT:

```
Show the first 10 rows from sampletransactions
```

SAMPLE QUERY:

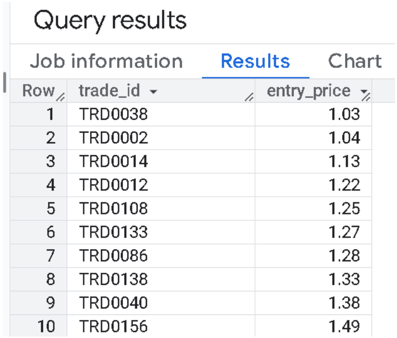
```
SELECT * FROM exercise_dataset.sampletransactions
LIMIT 10;
```

4 Using Gemini in BigQuery

5 Sorting rows with ORDER BY

Q1. Sort the rows in ascending order of the `entry_price` and show only the `trade_id` and `entry_price` columns. Limit the result returned to the first 10 rows.

Sample result:



Row	trade_id	entry_price
1	TRD0038	1.03
2	TRD0002	1.04
3	TRD0014	1.13
4	TRD0012	1.22
5	TRD0108	1.25
6	TRD0133	1.27
7	TRD0086	1.28
8	TRD0138	1.33
9	TRD0040	1.38
10	TRD0156	1.49

SAMPLE PROMPT:

```
Sort the rows in ascending order of the entry_price and show only the
trade_id and entry_price columns from sampletransactions. Limit the
result returned to the first 10 rows.
```

SAMPLE QUERY:

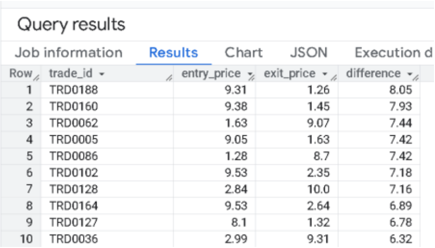
```
SELECT trade_id, entry_price
FROM exercise_dataset.sampletransactions
ORDER BY entry_price
LIMIT 10;
```

Q2. Earlier we had computed price difference as the difference between the exit and entry prices for a particular trade. Sort the rows in descending order based on the magnitude of the difference (i.e. we are not interested in the sign + or -, just the absolute value). Limit your result to the first 10 rows.

Hint: Google BigQuery has a large number of mathematical functions we can use in our queries. We can use the ABS function to get the magnitude of a number, regardless of its sign. The queries below are examples:

```
SELECT ABS(10) AS result;
SELECT ABS(-10) AS result;
```

Sample result:



Row	trade_id	entry_price	exit_price	difference
1	TRD0188	9.31	1.26	8.05
2	TRD0160	9.38	1.45	7.93
3	TRD0062	1.63	9.07	7.44
4	TRD0005	9.05	1.63	7.42
5	TRD0086	1.28	8.7	7.42
6	TRD0102	9.53	2.35	7.18
7	TRD0128	2.84	10.0	7.16
8	TRD0164	9.53	2.64	6.89
9	TRD0127	8.1	1.32	6.78
10	TRD0036	2.99	9.31	6.32

SAMPLE PROMPT:

```
Create a new column difference which is computed as the magnitude or
absolute value of the difference between exit_price and entry_price
from sampletransactions. Show only the trade_id, exit_price,
entry_price and difference columns and sort the result on descending
order of the difference column. Limit the result returned to the first
10 rows.
```

SAMPLE QUERY:

```
SELECT trade_id, entry_price, exit_price,
ABS(exit_price - entry_price) as difference
FROM exercise_dataset.sampletransactions
ORDER BY difference DESC
LIMIT 10;
```

You can further extend this answer to use the ROUND mathematical function to round the difference down to 2 decimal places to make the result more tidy.

```
SELECT trade_id, entry_price, exit_price,
ROUND(ABS(exit_price - entry_price), 2) as difference
FROM exercise_dataset.sampletransactions
ORDER BY difference DESC
LIMIT 10;
```

Q3. Sort the rows in descending order of the `currency` name. For transactions using the same currency, sort on ascending order of the `trade_volume`.

Sample result:

See file Topic 5 Q3 Results.csv in exercise-solutions.

SAMPLE PROMPT:

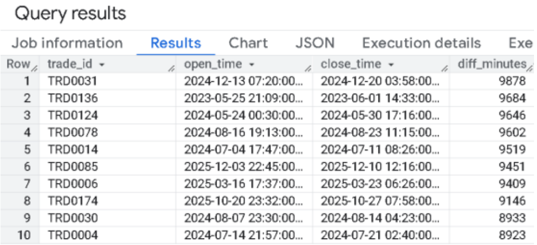
```
Sort the rows from sampletransactions in descending order of currency.
For rows with the same value of currency, sort on ascending order of
trade_volume.
```

SAMPLE QUERY:

```
SELECT trade_id, currency, trade_volume
FROM exercise_dataset.sampletransactions
ORDER BY currency DESC, trade_volume ASC;
```

Q4. Earlier we had computed the trade duration as the difference between the closing time (`close_time`) and opening time (`open_time`) in terms of total hours or total minutes or total seconds. Sort the rows in descending order of the trade duration in total minutes. Limit your result to the first 10 rows.

Sample result:



Row	trade_id	open_time	close_time	diff_minutes
1	TRD0031	2024-12-13 07:20:00...	2024-12-20 03:58:00...	9878
2	TRD0136	2023-05-25 21:09:00...	2023-06-01 14:33:00...	9684
3	TRD0124	2024-05-24 00:30:00...	2024-05-30 17:16:00...	9646
4	TRD0078	2024-08-16 19:13:00...	2024-08-23 11:15:00...	9602
5	TRD0014	2024-07-04 17:47:00...	2024-07-11 08:26:00...	9519
6	TRD0085	2025-12-03 22:45:00...	2025-12-10 12:16:00...	9451
7	TRD0006	2025-03-16 17:37:00...	2025-03-23 06:26:00...	9409
8	TRD0174	2025-10-20 23:32:00...	2025-10-27 07:58:00...	9146
9	TRD0030	2024-08-07 23:30:00...	2024-08-14 04:23:00...	8933
10	TRD0004	2024-07-14 21:57:00...	2024-07-21 02:40:00...	8923

SAMPLE PROMPT:

```
Create a new column diff_minutes which is the difference between
close_time and open_time in minutes.  Show the first 10 rows sorted
in descending order of diff_minutes
```

SAMPLE QUERY:

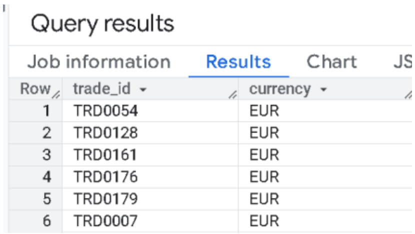
```
SELECT trade_id, open_time, close_time,
TIMESTAMP_DIFF(close_time, open_time, MINUTE) AS diff_minutes
FROM exercise_dataset.sampletransactions
ORDER BY diff_minutes DESC
LIMIT 10;
```

6 Saving queries, query results and viewing query history

7 Filtering with WHERE

Q1. Identify all the transactions that were made in EUR currency.

Sample result:



Row	trade_id	currency
1	TRD0054	EUR
2	TRD0128	EUR
3	TRD0161	EUR
4	TRD0176	EUR
5	TRD0179	EUR
6	TRD0007	EUR

SAMPLE PROMPT:

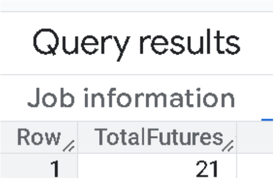
```
Show all rows where the currency column has the value EUR. Show only
trade_id and currency columns from the rows.
```

SAMPLE QUERY:

```
SELECT trade_id, currency
FROM exercise_dataset.sampletransactions
WHERE currency = 'EUR';
```

Q2. Count the total number of transactions which involve the instrument of type `futures`.

Sample result:



Row	TotalFutures
1	21

SAMPLE PROMPT:

```
Count the number of rows where the instrument is of type futures
```

SAMPLE QUERY:

```
SELECT COUNT(*) AS TotalFutures
FROM exercise_dataset.sampletransactions
WHERE instrument = 'futures';
```

Q3. List all the transactions whose trade volume is more or equals to 3000

Sample result:

See file Topic 7 Q3 Results.csv in exercise-solutions.

SAMPLE PROMPT:

```
Show all the rows where trade_volume is equal to or more than 3000.
Show only the trade_id and trade_volume columns
```

SAMPLE QUERY:

```
SELECT trade_id, trade_volume
FROM exercise_dataset.sampletransactions
WHERE trade_volume >= 30000;
```

Q4. Earlier we had computed price difference as the difference between the exit and entry prices for a particular trade. List all the rows where the price difference is more than 6.0

Sample result:

See file Topic 7 Q4 Results.csv in exercise-solutions.

SAMPLE PROMPT:

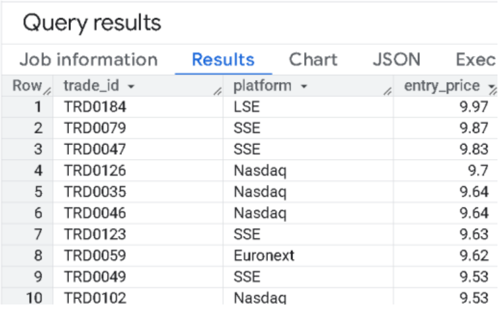
```
Create a new column difference which is computed as the magnitude or
absolute value of the difference between exit_price and entry_price
from sampletransactions. Show only the rows where difference is more
than 6.0. Show only the trade_id, exit_price, entry_price and
difference columns.
```

SAMPLE QUERY:

```
SELECT trade_id, entry_price, exit_price,
ABS(exit_price - entry_price) as difference
FROM exercise_dataset.sampletransactions
WHERE ABS(exit_price - entry_price) > 6.0;
```

Q5. Show all the transactions which were not made on the NYSE platform and sort them in descending order based on their `entry_price`. Limit your results to the top 10.

Sample result:



Row	trade_id	platform	entry_price
1	TRD0184	LSE	9.97
2	TRD0079	SSE	9.87
3	TRD0047	SSE	9.83
4	TRD0126	Nasdaq	9.7
5	TRD0035	Nasdaq	9.64
6	TRD0046	Nasdaq	9.64
7	TRD0123	SSE	9.63
8	TRD0059	Euronext	9.62
9	TRD0049	SSE	9.53
10	TRD0102	Nasdaq	9.53

SAMPLE PROMPT:

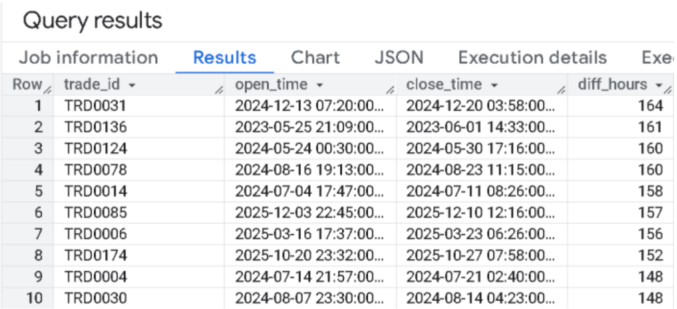
```
Show all rows which are not on the NYSE platform and sort them in descending order based on the entry_price. Limit this to the top 10 rows. Show only the trade_id, platform and entry_price columns in the results.
```

SAMPLE QUERY:

```
SELECT trade_id, platform, entry_price
FROM exercise_dataset.sampletransactions
WHERE platform != 'NYSE'
ORDER BY entry_price DESC
LIMIT 10;
```

Q6. Earlier we had computed the trade duration as the difference between the closing time (`close_time`) and opening time (`open_time`) in terms of total hours or total minutes or total seconds. Find all the trades whose duration is 5 hours or longer, and sort them in descending order. Limit your result to the first 10 rows.

Sample result:



Row	trade_id	open_time	close_time	diff_hours
1	TRD0031	2024-12-13 07:20:00...	2024-12-20 03:58:00...	164
2	TRD0136	2023-05-25 21:09:00...	2023-06-01 14:33:00...	161
3	TRD0124	2024-05-24 00:30:00...	2024-05-30 17:16:00...	160
4	TRD0078	2024-08-16 19:13:00...	2024-08-23 11:15:00...	160
5	TRD0014	2024-07-04 17:47:00...	2024-07-11 08:26:00...	158
6	TRD0085	2025-12-03 22:45:00...	2025-12-10 12:16:00...	157
7	TRD0006	2025-03-16 17:37:00...	2025-03-23 06:26:00...	156
8	TRD0174	2025-10-20 23:32:00...	2025-10-27 07:58:00...	152
9	TRD0004	2024-07-14 21:57:00...	2024-07-21 02:40:00...	148
10	TRD0030	2024-08-07 23:30:00...	2024-08-14 04:23:00...	148

SAMPLE PROMPT:

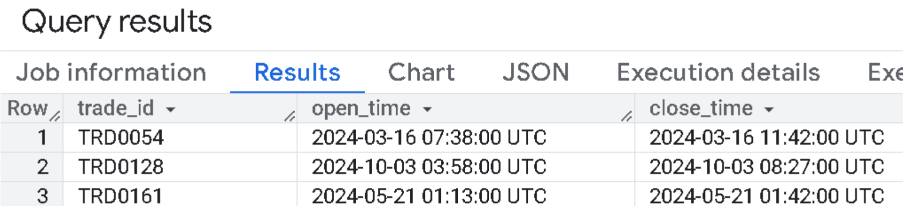
Create a new column diff_hours which is the difference between close_time and open_time in hours. Show all the rows where diff_hours is 5 or longer and sort these rows in descending order of diff_hours. Show only the first 10 rows and include only the trade_id, open_time, close_time and diff_hours columns.

SAMPLE QUERY:

```
SELECT trade_id, open_time, close_time,
TIMESTAMP_DIFF(close_time, open_time, HOUR) AS diff_hours
FROM exercise_dataset.sampletransactions
WHERE TIMESTAMP_DIFF(close_time, open_time, HOUR) >= 5
ORDER BY diff_hours DESC
LIMIT 10;
```

Q7. Show all the rows where the closing time (`close_time`) and opening time (`open_time`) occur on the same day (YYYY-MM-DD), irrespective of the time of the day.

Hint: You can use the DATE function to return the date portion (YYYY-MM-DD) of the entire time stamp value for both these columns.



Row	trade_id	open_time	close_time
1	TRD0054	2024-03-16 07:38:00 UTC	2024-03-16 11:42:00 UTC
2	TRD0128	2024-10-03 03:58:00 UTC	2024-10-03 08:27:00 UTC
3	TRD0161	2024-05-21 01:13:00 UTC	2024-05-21 01:42:00 UTC

SAMPLE PROMPT:

Show all the rows where close_time and open_time have the same day value, irrespective of the hours or minutes of the day. Show only trade_id, close_time and open_time columns in the result.

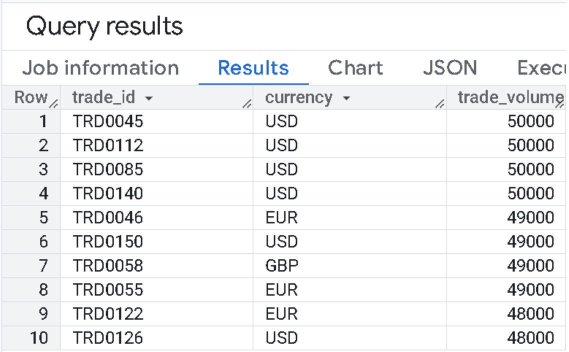
SAMPLE QUERY:

```
SELECT trade_id, open_time, close_time
FROM exercise_dataset.sampletransactions
WHERE DATE(open_time) = DATE(close_time);
```

7.1 Using the AND, OR and NOT operators

Q8. Show the top 10 highest transactions in terms of `trade_volume` that were made in any of these 3 currencies: USD, EUR, GBP. Give two possible alternative forms of the query that you can write.

Sample result:



Row	trade_id	currency	trade_volume
1	TRD0045	USD	50000
2	TRD0112	USD	50000
3	TRD0085	USD	50000
4	TRD0140	USD	50000
5	TRD0046	EUR	49000
6	TRD0150	USD	49000
7	TRD0058	GBP	49000
8	TRD0055	EUR	49000
9	TRD0122	EUR	48000
10	TRD0126	USD	48000

SAMPLE PROMPT:

```
Show the rows which have currency values of either: USD, EUR, GBP. Sort these in descending order of trade_volume and show the first 10 rows. Show only trade_id, currency and trade_volume columns. Use the OR clause in the query.
```

SAMPLE QUERY:

Version #1

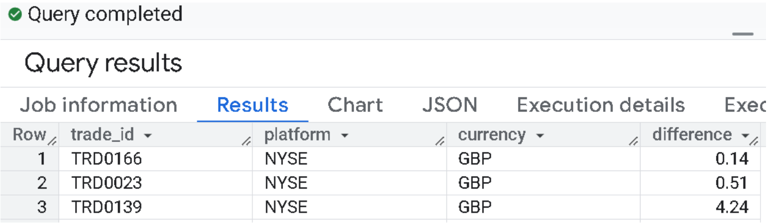
```
SELECT trade_id, currency, trade_volume
FROM exercise_dataset.sampletransactions
WHERE currency = 'GBP' OR currency = 'USD' OR currency = 'EUR'
ORDER BY trade_volume DESC
LIMIT 10;
```

Version #2 (if you leave out the explicit mention of using the OR clause in the query in your original prompt, it will use the shorter version of IN clause)

```
SELECT trade_id, currency, trade_volume
FROM exercise_dataset.sampletransactions
WHERE currency IN ('USD', 'EUR', 'GBP')
ORDER BY trade_volume DESC
LIMIT 10;
```

Q9. Earlier we had computed price difference as the difference between the exit and entry prices for a particular trade. Show the lowest 3 transactions in terms of this difference for trades that were transacted in GBP on the NYSE.

Sample result:



Row	trade_id	platform	currency	difference
1	TRD0166	NYSE	GBP	0.14
2	TRD0023	NYSE	GBP	0.51
3	TRD0139	NYSE	GBP	4.24

SAMPLE PROMPT:

```
Create a new column called difference which is computed as the difference between the exit_price and entry_price columns for sampletransactions. Show only the rows that have currency GBP and platform NYSE. Sort the rows on the difference column in ascending order and show the first 3 rows. Show only the trade_id, platform, currency and difference columns.
```

SAMPLE QUERY:

```
SELECT trade_id, platform, currency,
(exit_price - entry_price) as difference
FROM exercise_dataset.sampletransactions
WHERE currency = 'GBP' AND platform = "NYSE"
ORDER BY difference LIMIT 3;
```

7.2 Using BETWEEN for range tests

Q10. Find all transactions whose trade volume is between 20000 and 40000. Sort your results on the trade volume in descending order.

Sample result:

See file Topic 7 Q10 Results.csv in exercise-solutions.

SAMPLE PROMPT:

```
Find all rows where the trade_volume is between 20000 and 40000 and sort them on descending order of trade_volume. Show only the trade_id and trade_volume columns. Use the BETWEEN clause in the query.
```

SAMPLE QUERY:

```
SELECT trade_id, trade_volume
FROM exercise_dataset.sampletransactions
WHERE trade_volume BETWEEN 20000 AND 40000
ORDER BY trade_volume DESC;
```

Q11. List all transactions that took place between June 2024 and June 2025. We consider the transaction to have taken place when it was initiated, not when it closed.

Sample result:

See file Topic 7 Q11 Results.csv in exercise-solutions.

SAMPLE PROMPT:

```
Find all rows where open_time is between the start of June 2024 and
start of June 2025. Sort them on ascending order of open_time. Use
the BETWEEN clause in the query. Show only the trade_id and open_time
columns.
```

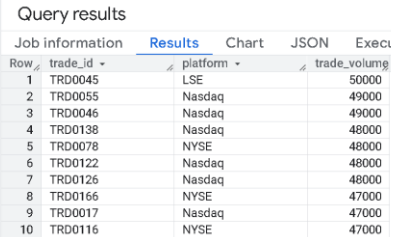
SAMPLE QUERY:

```
SELECT trade_id, open_time FROM exercise_dataset.sampletransactions
WHERE open_time BETWEEN '2024-06-01' AND '2025-06-01'
ORDER BY open_time;
```

7.3 Using IN to check for matching with other values

Q12. Show the top 10 highest transactions in terms of `trade_volume` that were made in any of these 3 platforms: `NYSE, Nasdaq, LSE`.

Sample result:



Row	trade_id	platform	trade_volume
1	TRD0045	LSE	50000
2	TRD0055	Nasdaq	49000
3	TRD0046	Nasdaq	49000
4	TRD0138	Nasdaq	48000
5	TRD0078	NYSE	48000
6	TRD0122	Nasdaq	48000
7	TRD0126	Nasdaq	48000
8	TRD0166	NYSE	47000
9	TRD0017	Nasdaq	47000
10	TRD0116	NYSE	47000

SAMPLE PROMPT:

```
Find all rows where platform is either NYSE, Nasdaq or LSE. Sort these
rows on descending order of trade_volume and show the first 10 rows.
Show only trade_id, platform and trade_volume columns.
```

SAMPLE QUERY:

```
SELECT trade_id, platform, trade_volume
FROM exercise_dataset.sampletransactions
WHERE platform IN ('NYSE', 'Nasdaq', 'SSE', 'LSE')
ORDER BY trade_volume DESC
LIMIT 10;
```

8 Using CASE to implement conditional logic to add columns

Q1. Assume we want to categorize the transactions into 3 categories based on the `trade_volume`.

Trade Volume	Category
10,000 and below	Low
10,001 – 30,000	Medium
30,001 and above	High

Create a new `category` column according to the table above.

Sample result:

See file Topic 8 Q1 Results.csv in exercise-solutions.

SAMPLE PROMPT:

```
Create a new category column which has 3 possible values depending on
the range of values in trade_volume for each given row.
When the trade volume is 10000 or less, the category column will have
the value: low
When the trade volume between 10001 and 30000, the category column
will have the value: medium
For any other value, the category column will have the value: High
Show the trade_id, trade_volume and category columns. Use the CASE
and WHEN clause in the query.
```

SAMPLE QUERY:

```
SELECT trade_id, trade_volume,
  CASE
    WHEN trade_volume <= 10000 THEN 'Low'
    WHEN trade_volume > 10000 AND trade_volume <= 30000 THEN 'Medium'
    ELSE 'High'
  END AS Category
FROM exercise_dataset.sampletransactions;
```

Q2. We will create a new column called `result` which determines whether a particular transaction results in a `profit` or `loss`.
In a long trade (`trade_type = long`), an asset is bought with an expectation that its price will increase – the goal is to buy low at entry and sell high when exiting. For this trade type, a profit occurs when the exit price > entry price.
In a short trade (`trade_type = short`), a borrowed asset is sold with an expectation that its price will fall – the goal is to sell high at entry and buy low when exiting. For this trade type, a profit occurs when the exit price < entry price

Sample result: See file Topic 8 Q2 Results.csv in exercise-solutions.

SAMPLE PROMPT:

```
Create a new column result which has either the value of profit or
loss.
For a given row, if the trade_type is long, result will be profit if
the exit_price is more than the entry_price, otherwise result will be
loss.
For a given row, if the trade_type is short, result will be profit if
the exit_price is less than the entry_price, otherwise result will be
loss.
Show only the trade_id, trade_type, entry_price, exit_price, result
columns. Use the CASE and WHEN clause in the query.
```

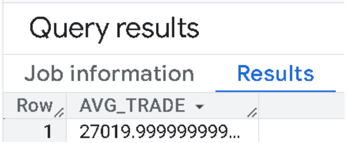
SAMPLE QUERY:

```
SELECT trade_id, trade_type, entry_price, exit_price,
  CASE
    WHEN trade_type = 'long' AND exit_price > entry_price THEN
'profit'
    WHEN trade_type = 'long'
  AND exit_price <= entry_price THEN 'loss'
    WHEN trade_type = 'short' AND exit_price < entry_price THEN
'profit'
    WHEN trade_type = 'short'
  AND exit_price >= entry_price THEN 'loss'
    ELSE 'unknown'
  END AS Result
FROM exercise_dataset.sampletransactions;
```

9 Aggregate functions: COUNT, SUM, AVG, MIN, MAX

Q1. Calculate the average volume of transactions that were performed using the USD currency.

Sample result:



Row	AVG_TRADE
1	27019.999999999...

SAMPLE PROMPT:

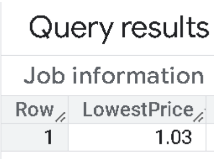
```
Find the average of trade_volume for the rows where currency has the
value of USD.
```

SAMPLE QUERY:

```
SELECT AVG(trade_volume) AS AVG_TRADE
FROM exercise_dataset.sampletransactions
WHERE currency = 'USD';
```

Q2. Find the lowest entry price for all transactions on either the LSE and SSE platform.

Sample result:



Row	LowestPrice
1	1.03

SAMPLE PROMPT:

```
Find the lowest value of entry_price for all rows which have the value
of LSE or SSE for the platform column.
```

SAMPLE QUERY:

```
SELECT MIN(entry_price) AS LowestPrice
FROM exercise_dataset.sampletransactions
WHERE platform IN ('LSE','SSE');
```

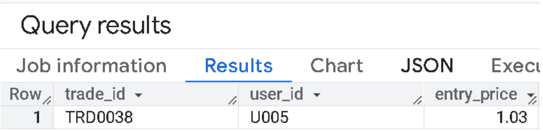
Or

```
SELECT MIN(entry_price) AS LowestPrice
FROM exercise_dataset.sampletransactions
WHERE platform = 'LSE' OR platform = 'SSE';
```

9.1 Refining queries with aggregate functions for column details

Q3. Find the `trade_id` and `user_id` for the transaction with the lowest entry price for all transactions on either the LSE and SSE platform
 Hint: You can use a subquery from the previous query.

Sample result:



Row	trade_id	user_id	entry_price
1	TRD0038	U005	1.03

SAMPLE PROMPT:

```
Find the row with the lowest value of entry_price for all rows which
have the value of LSE or SSE for the platform column. For this row,
show the trade_id, user_id and entry_price column. Use the MIN
function to achieve this.
```

SAMPLE QUERY:
Version #1

```
SELECT trade_id, user_id, entry_price
FROM exercise_dataset.sampletransactions
WHERE platform IN ('LSE','SSE')
AND entry_price = (

  SELECT MIN(entry_price) AS LowestPrice
  FROM exercise_dataset.sampletransactions
  WHERE platform IN ('LSE','SSE')

) LIMIT 1;
```

Version #2: If you do not explicitly specify to use the MIN function, the prompt will return a simpler version

```
SELECT trade_id, user_id, entry_price
FROM exercise_dataset.sampletransactions
WHERE platform IN ('LSE', 'SSE')
ORDER BY entry_price ASC LIMIT 1;
```

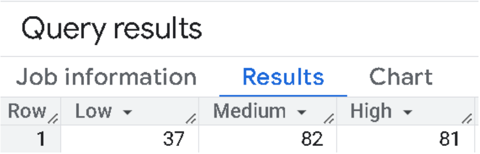
9.2 Aggregate functions with CASE clause

Q4. Earlier we had categorized the transactions into 3 categories based on the `trade_volume`.

Trade Volume	Category
10,000 and below	Low
10,001 – 30,000	Medium
30,001 and above	High

These 3 new values were placed in a new `category` column.
Find the total number of transactions in each of these categories (Low, Medium and High)

Sample result:



Row	Low	Medium	High
1	37	82	81

SAMPLE PROMPT:

```
Create a new category column which has 3 possible values depending on
the range of values in trade_volume for each given row.
When the trade volume is 10000 or less, the category column will have
the value: low
When the trade volume between 10001 and 30000, the category column
will have the value: medium
For any other value, the category column will have the value: High
Finally, compute the total of low, medium and high values respectively
in this new category column.
Use the SUM, CASE and WHEN clause to achieve this functionality in
the query you create.
```

SAMPLE QUERY:

Version #1:

```
SELECT
  SUM (
    CASE WHEN trade_volume <= 10000
    THEN 1 ELSE 0 END
  ) AS Low,

  SUM (
    CASE WHEN trade_volume > 10000 AND trade_volume <= 30000
    THEN 1 ELSE 0 END
  ) AS Medium,

  SUM (
    CASE WHEN trade_volume > 30000
    THEN 1 ELSE 0 END
  ) AS High

FROM exercise_dataset.sampletransactions;
```

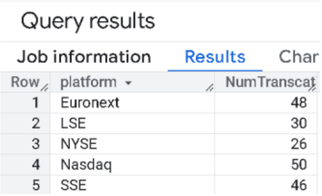
Version #2: Uses the GROUP BY clause (if you don’t explicitly specify to use the SUM clause) – this will be covered in the next lab

```
SELECT
  CASE
    WHEN trade_volume <= 10000 THEN 'low'
    WHEN trade_volume BETWEEN 10001
  AND 30000 THEN 'medium'
    ELSE 'High'
  END
 AS trade_volume_category,
 COUNT(trade_id) AS count_of_trades
FROM
  exercise_dataset.sampletransactions
GROUP BY
  trade_volume_category
ORDER BY
  trade_volume_category;
```

10 Aggregating and grouping with GROUP BY

Q1. Count the number of transactions performed in each of the 5 platforms.

Sample result:



Query results

Row	platform	NumTranscat
1	Euronext	48
2	LSE	30
3	NYSE	26
4	Nasdaq	50
5	SSE	46

SAMPLE PROMPT:

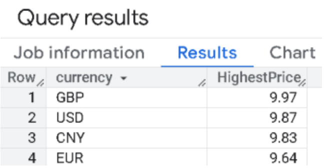
```
Find the total number of rows for each unique value in the platform
column.
```

SAMPLE QUERY:

```
SELECT platform, COUNT(platform) AS NumTranscations
FROM exercise_dataset.sampletransactions
GROUP BY platform;
```

Q2. Find the transaction with the highest entry price for each currency type. Sort these transactions in descending order based on these entry prices.

Sample result:



Row	currency	HighestPrice
1	GBP	9.97
2	USD	9.87
3	CNY	9.83
4	EUR	9.64

SAMPLE PROMPT:

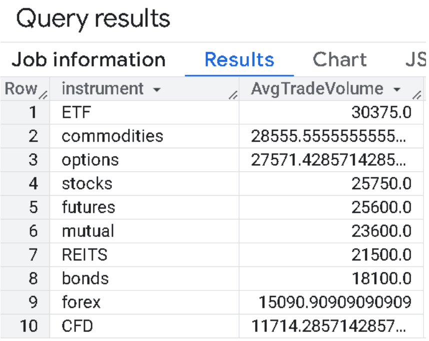
```
Find the rows with the highest value for entry_price for all unique
values in the currency column. Sort these rows in descending order
based on this value of entry_price. Show only the currency and the
highest value for entry_price in the result.
```

SAMPLE QUERY:

```
SELECT currency, MAX(entry_price) AS HighestPrice
FROM exercise_dataset.sampletransactions
GROUP BY currency
ORDER BY HighestPrice DESC;
```

Q3. Find the average trade volume for all long trade transactions for each particular instrument type. Sort these results in descending order of the average trade volume.

Sample result:



Row	instrument	AvgTradeVolume
1	ETF	30375.0
2	commodities	28555.5555555555...
3	options	27571.4285714285...
4	stocks	25750.0
5	futures	25600.0
6	mutual	23600.0
7	REITS	21500.0
8	bonds	18100.0
9	forex	15090.90909090909
10	CFD	11714.2857142857...

SAMPLE PROMPT:

```
Find all the rows where trade_type is long, and for these rows, compute
the average trade_volume for groupings corresponding to unique values
of the instrument column. Sort the results in descending order of the
average trade_volume. Show only the instrument and average
trade_volume in the results.
```

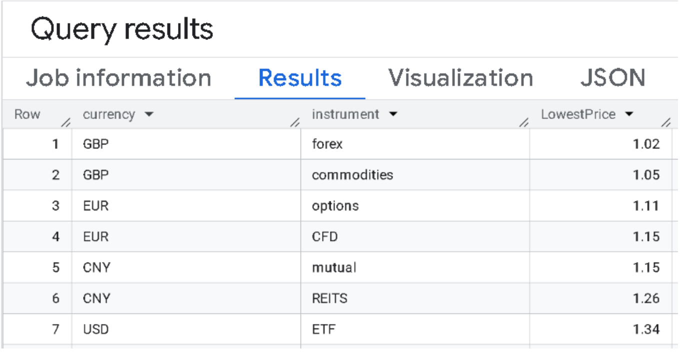
SAMPLE QUERY:

```
SELECT instrument, AVG(trade_volume) AS AvgTradeVolume
FROM exercise_dataset.sampletransactions
WHERE trade_type = 'long'
GROUP BY instrument
ORDER BY AvgTradeVolume DESC;
```

10.1 Grouping multiple columns

Q4. Find the lowest exit price of transactions for all groupings of currency and instrument. Order the results in ascending order of this lowest exit price.

Sample result:



Row	currency	instrument	LowestPrice
1	GBP	forex	1.02
2	GBP	commodities	1.05
3	EUR	options	1.11
4	EUR	CFD	1.15
5	CNY	mutual	1.15
6	CNY	REITS	1.26
7	USD	ETF	1.34

SAMPLE PROMPT:

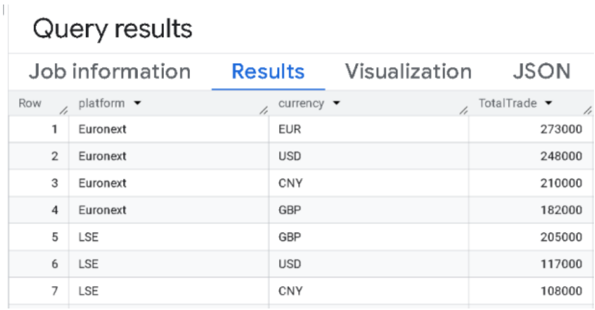
```
For all unique groupings of currency and instrument column values,
find the smallest value of exit_price for each grouping. Sort the
results in ascending order of this smallest value. Show only the
currency, instrument and smallest value in the results.
```

SAMPLE QUERY:

```
SELECT currency, instrument, MIN(exit_price) AS LowestPrice
FROM exercise_dataset.sampletransactions
GROUP BY currency, instrument
ORDER BY LowestPrice ASC;
```

Q5. Find the total trade volume for short trades on all unique groupings of platform and currency. Your result should show the platforms first with all the currencies associated with that platform listed in descending order of the total trade volume.

Sample result:



Row	platform	currency	TotalTrade
1	Euronext	EUR	273000
2	Euronext	USD	248000
3	Euronext	CNY	210000
4	Euronext	GBP	182000
5	LSE	GBP	205000
6	LSE	USD	117000
7	LSE	CNY	108000

SAMPLE PROMPT:

```
Find all the rows where trade_type is short, and for these rows, compute the total trade_volume for groupings corresponding to unique values of the platform and currency column. Sort the results in descending order of this total trade_volume. Show only the platform, currency and the total trade_volume in the results.
```

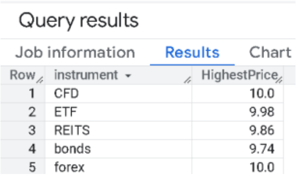
SAMPLE QUERY:

```
SELECT platform, currency, SUM(trade_volume) AS TotalTrade
FROM exercise_dataset.sampletransactions
WHERE trade_type = 'short'
GROUP BY platform, currency
ORDER BY platform, TotalTrade DESC;
```

10.2 Using HAVING clause to filter on groups

Q6. Find the highest exit price for transactions on all the different instruments. Exclude the instruments whose transaction with the highest exit price is less than 9.5

Sample result:



Row	instrument	HighestPrice
1	CFD	10.0
2	ETF	9.98
3	REITS	9.86
4	bonds	9.74
5	forex	10.0

SAMPLE PROMPT:

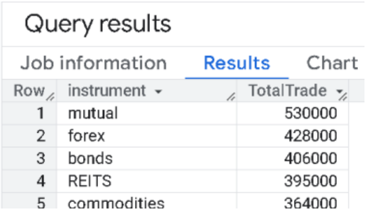
```
Find the highest exit_price in groupings for all unique values of instrument. Show the instrument and this highest exit_price in the results, and exclude instruments where this highest exit_price is less than 9.5.
```

SAMPLE QUERY:

```
SELECT instrument, MAX(exit_price) AS HighestPrice
FROM exercise_dataset.sampletransactions
GROUP BY instrument
HAVING HighestPrice > 9.5;
```

Q7. Earlier we have seen that the price difference is computed as the difference between the exit and entry prices for a particular transaction
We want to compute the total trade volume for transactions for all instruments, but exclude transactions whose price difference is 2.0 or less. For the final list, we only want to list instruments whose total trade volume is more than   300,000.

Sample result:



Row	instrument	TotalTrade
1	mutual	530000
2	forex	428000
3	bonds	406000
4	REITS	395000
5	commodities	364000

SAMPLE PROMPT:

```
For each row, the price difference is computed as the difference between the exit_price and entry_price. For rows where this price difference is 2.0 or more, compute the total trade_volume for grouping of rows corresponding to each unique value of instrument. Show the instrument and this total trade_volume in the results, and exclude instruments where this total trade_volume is less than 300000. Order the results in descending order of the total trade_volume
```

SAMPLE QUERY:

```
SELECT instrument, SUM(trade_volume) as TotalTrade
FROM exercise_dataset.sampletransactions
WHERE ABS(exit_price - entry_price)  >= 2.0
GROUP BY instrument
HAVING TotalTrade >= 300000
ORDER BY TotalTrade DESC;
```

Notice that the result you get would be quite different if you had removed the initial filter on the transactions that are to be grouped and aggregated, as shown below:

```
SELECT instrument, SUM(trade_volume) as TotalTrade
FROM exercise_dataset.sampletransactions
GROUP BY instrument
HAVING TotalTrade >= 300000
ORDER BY TotalTrade DESC;
```