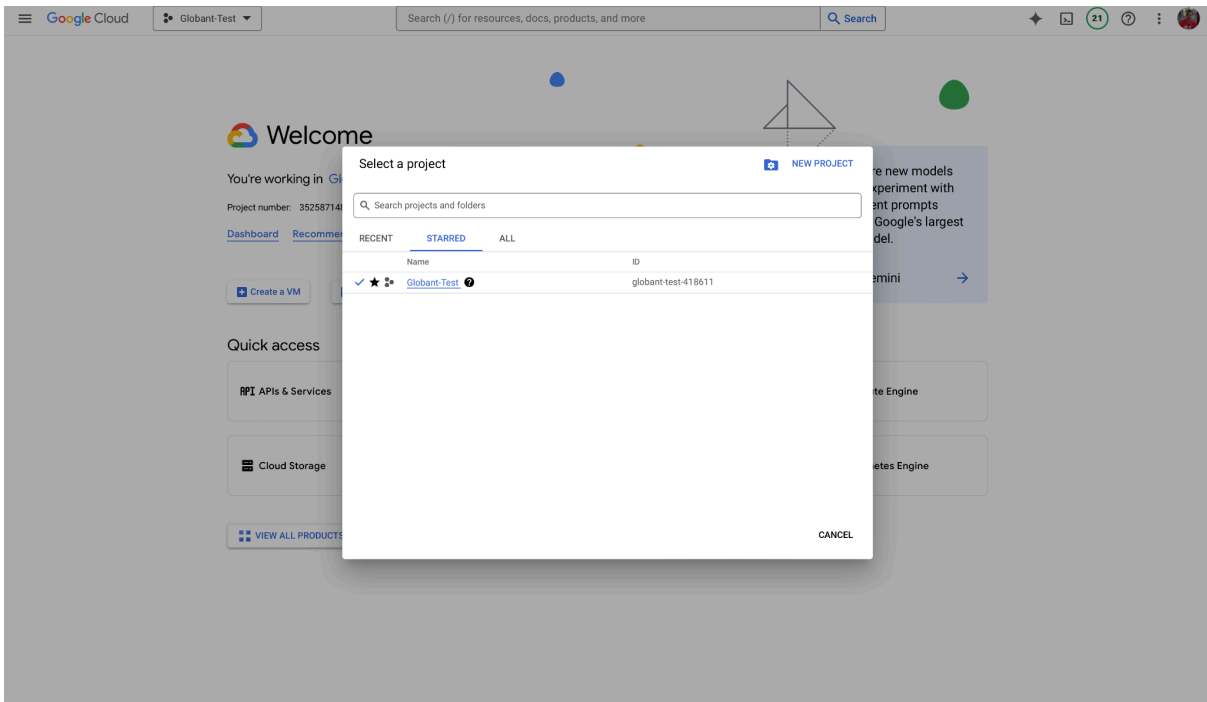


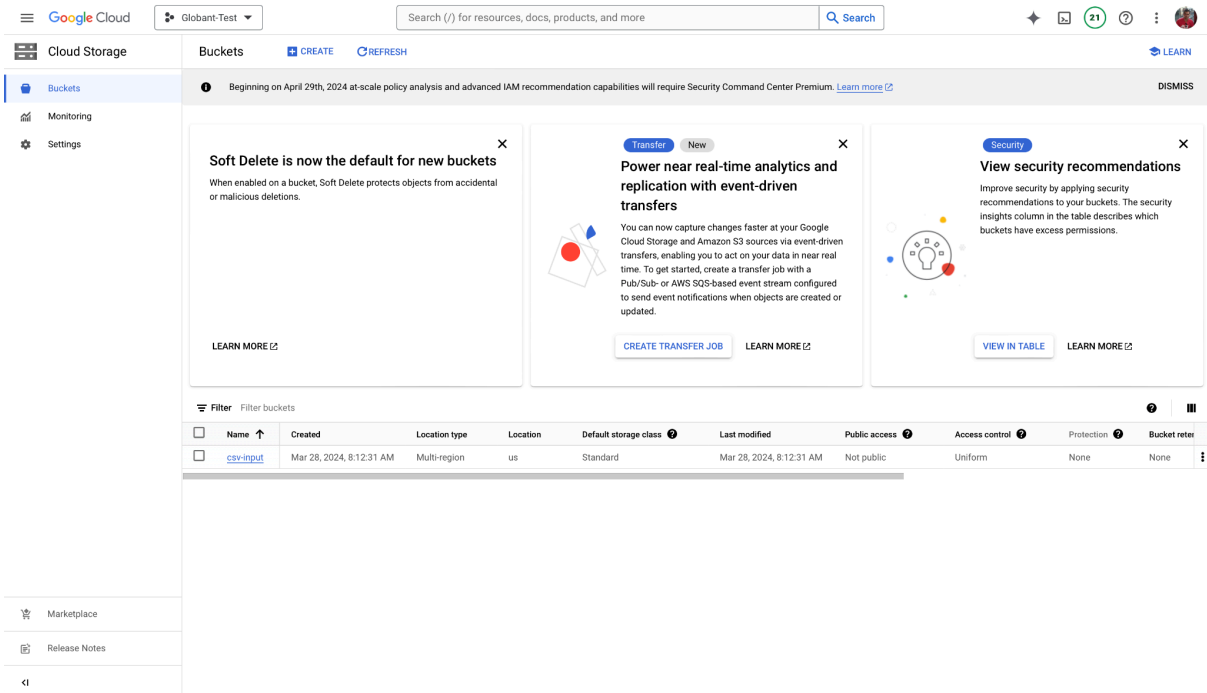
Github: <https://github.com/martinmvelez/Globant>

With the purpose of developing this exercise, i created Globant-Test project in GCP.



Buckets

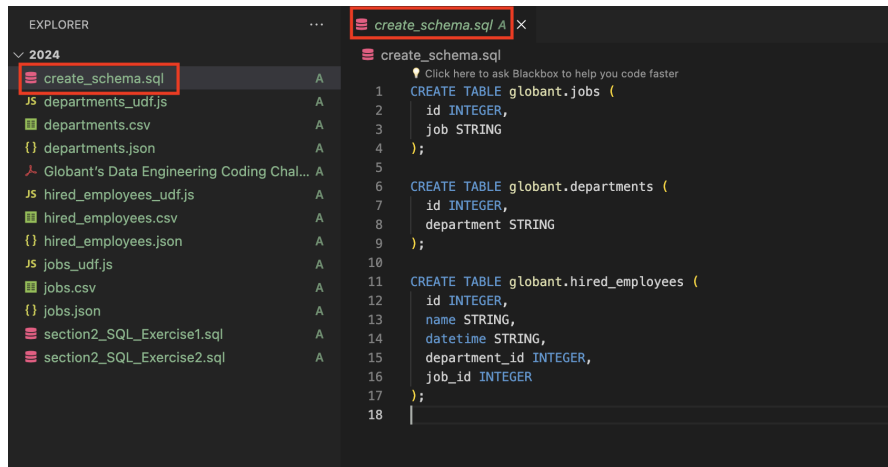
To upload the files departments.csv, jobs.csv, hired_employees.csv i created the bucket csv-input



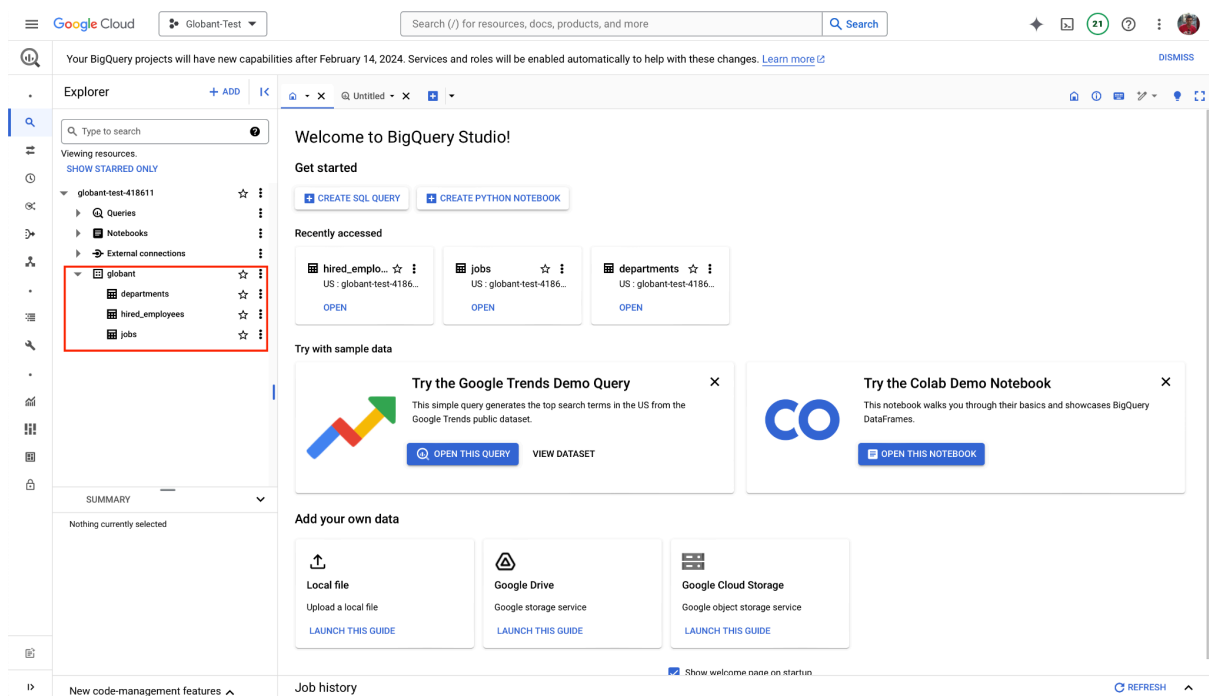
Bigquery

I created the dataset globant and the tables

Note: code of table creation create_schema.sql



```
1 CREATE TABLE globant.jobs (  
2   id INTEGER,  
3   job STRING  
4 );  
5  
6 CREATE TABLE globant.departments (  
7   id INTEGER,  
8   department STRING  
9 );  
10  
11 CREATE TABLE globant.hired_employees (  
12   id INTEGER,  
13   name STRING,  
14   datetime STRING,  
15   department_id INTEGER,  
16   job_id INTEGER  
17 );  
18
```



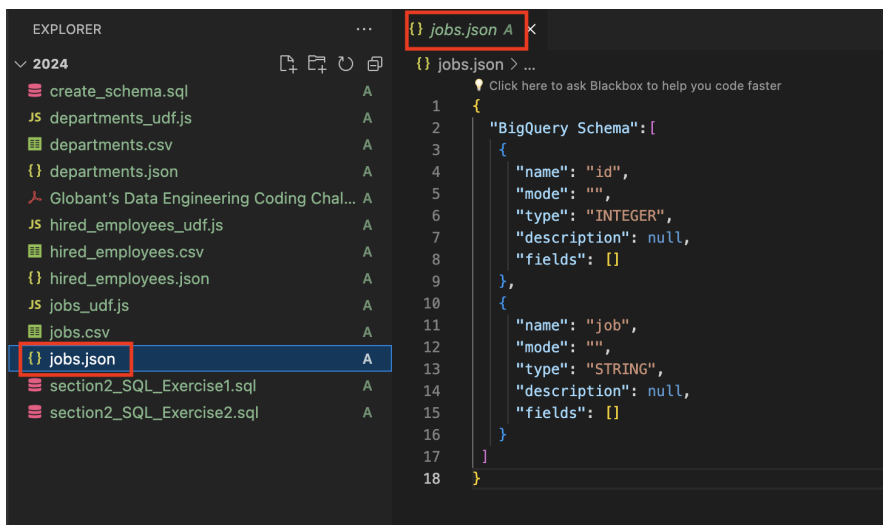
APIs

In the following i will describe the ingestion process for each one.

Jobs.csv

I created jobs.json, jobs_udf.js

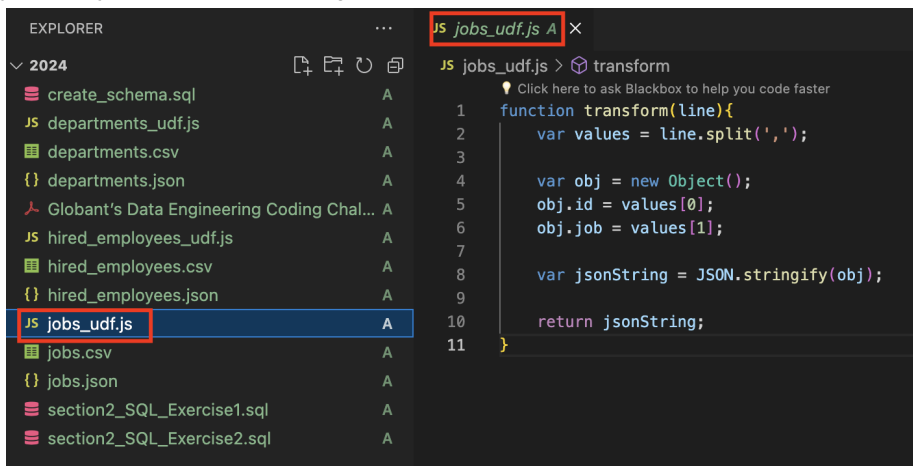
jobs.json: is the schema of the table



The screenshot shows the VS Code interface. In the Explorer on the left, the file `jobs.json` is selected and highlighted with a red box. The Editor on the right shows the content of `jobs.json`, which is a BigQuery schema definition. The schema is a JSON object with a key `"BigQuery Schema": [` followed by an array of field objects. Each field object has `"name"`, `"mode"`, `"type"`, `"description"`, and `"fields"` properties. The first field is `"id"` of type `"INTEGER"`. The second field is `"job"` of type `"STRING"`. The schema is enclosed in curly braces and ends with a closing bracket for the array.

```
{
  "BigQuery Schema": [
    {
      "name": "id",
      "mode": "",
      "type": "INTEGER",
      "description": null,
      "fields": []
    },
    {
      "name": "job",
      "mode": "",
      "type": "STRING",
      "description": null,
      "fields": []
    }
  ]
}
```

jobs_udf.js: Script for the csv ingestion



The screenshot shows the VS Code interface. In the Explorer on the left, the file `jobs_udf.js` is selected and highlighted with a red box. The Editor on the right shows the content of `jobs_udf.js`, which is a JavaScript User Defined Function (UDF) for BigQuery. The function is named `transform` and takes a `line` parameter. It splits the line by commas, creates a JavaScript object with `id` and `job` properties, and returns the object as a JSON string.

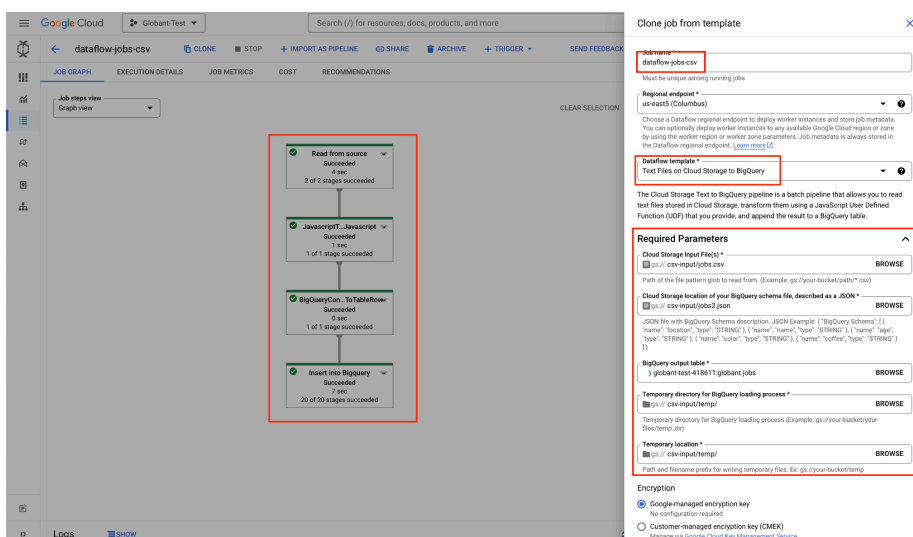
```
function transform(line){
  var values = line.split(',');

  var obj = new Object();
  obj.id = values[0];
  obj.job = values[1];

  var jsonString = JSON.stringify(obj);

  return jsonString;
}
```

Dataflow Job



The screenshot shows the Google Cloud Dataflow console. On the left, the 'JOB GRAPH' tab is selected, showing a pipeline with four stages: 'Read from source', 'JavaScript...JavaScript', 'BigQueryConn. ToTableflow', and 'Insert into Bigquery'. The 'Read from source' stage is highlighted with a red box. On the right, the 'Clone job from template' dialog is open, showing the configuration for the 'dataflow-jobs-csv' job. The 'Regional endpoint' is set to 'us-east1 (Columbus)'. The 'Dataflow template' is set to 'Text Files on Cloud Storage to BigQuery'. The 'Required Parameters' section is expanded, showing fields for 'Cloud Storage input file(s)', 'Cloud Storage location of your BigQuery schema file', 'BigQuery output table', 'Temporary directory for BigQuery loading process', and 'Temporary location'.

Clone job from template

Job name: dataflow-jobs-csv

Regional endpoint: us-east1 (Columbus)

Dataflow template: Text Files on Cloud Storage to BigQuery

Required Parameters

Cloud Storage input file(s): gs://cav-input/jobs-csv

Cloud Storage location of your BigQuery schema file, described as a JSON: gs://cav-input/jobs2.json

BigQuery output table: globant-test-418611:globant.jobs

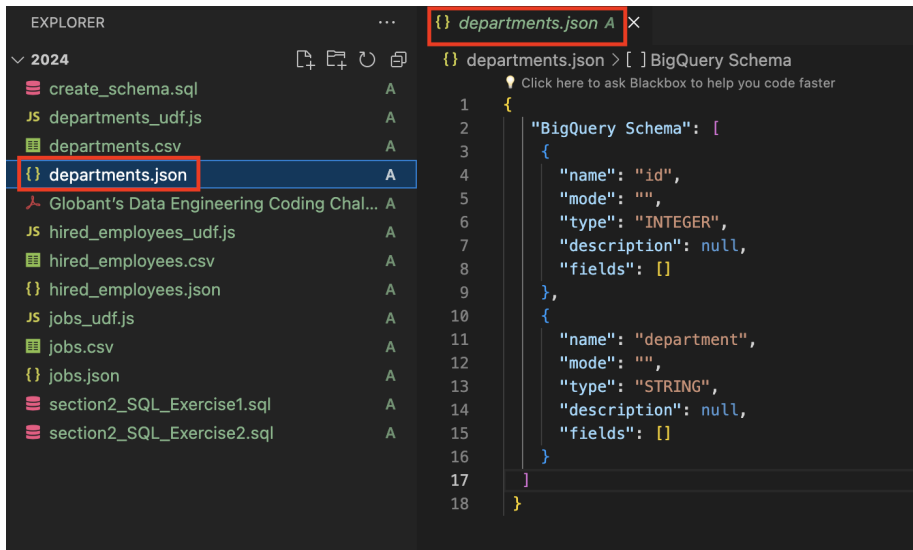
Temporary directory for BigQuery loading process: gs://cav-input/temp

Temporary location: gs://cav-input/temp

Departments.csv

I created departments.json, departments_udf.js

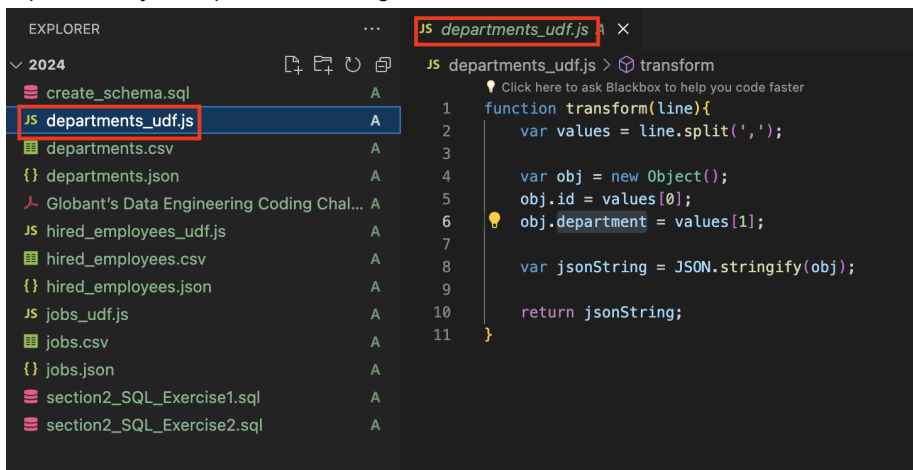
departments.json: is the schema of the table



The screenshot shows the VS Code interface. In the Explorer on the left, the file `departments.json` is selected and highlighted with a red box. The editor on the right displays the JSON schema for the `departments` table. The schema is a BigQuery Schema with two fields: `id` (INTEGER) and `department` (STRING). The JSON is as follows:

```
{
  "BigQuery Schema": [
    {
      "name": "id",
      "mode": "",
      "type": "INTEGER",
      "description": null,
      "fields": []
    },
    {
      "name": "department",
      "mode": "",
      "type": "STRING",
      "description": null,
      "fields": []
    }
  ]
}
```

departments.js: Script for the csv ingestion



The screenshot shows the VS Code interface. In the Explorer on the left, the file `departments_udf.js` is selected and highlighted with a red box. The editor on the right displays the JavaScript User Defined Function (UDF) script for transforming CSV data into JSON. The script is as follows:

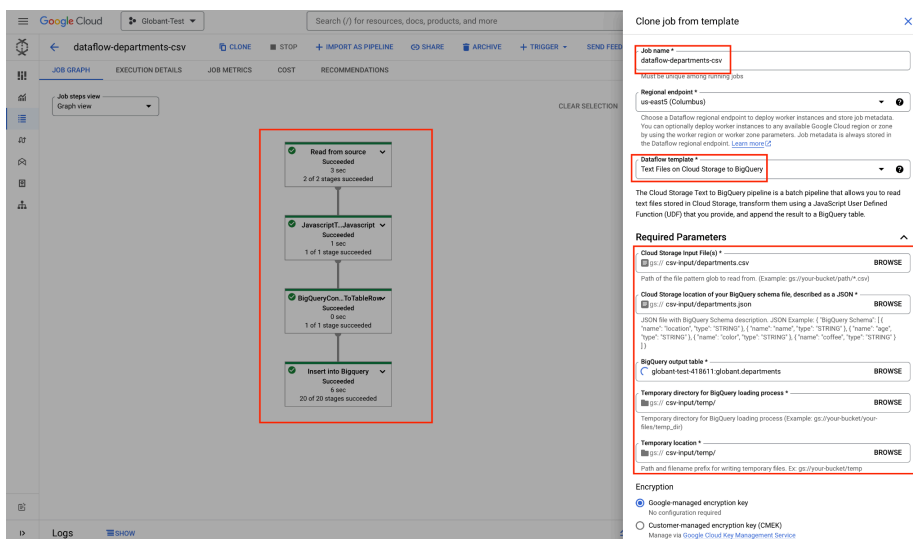
```
function transform(line){
  var values = line.split(',');

  var obj = new Object();
  obj.id = values[0];
  obj.department = values[1];

  var jsonString = JSON.stringify(obj);

  return jsonString;
}
```

Dataflow Job



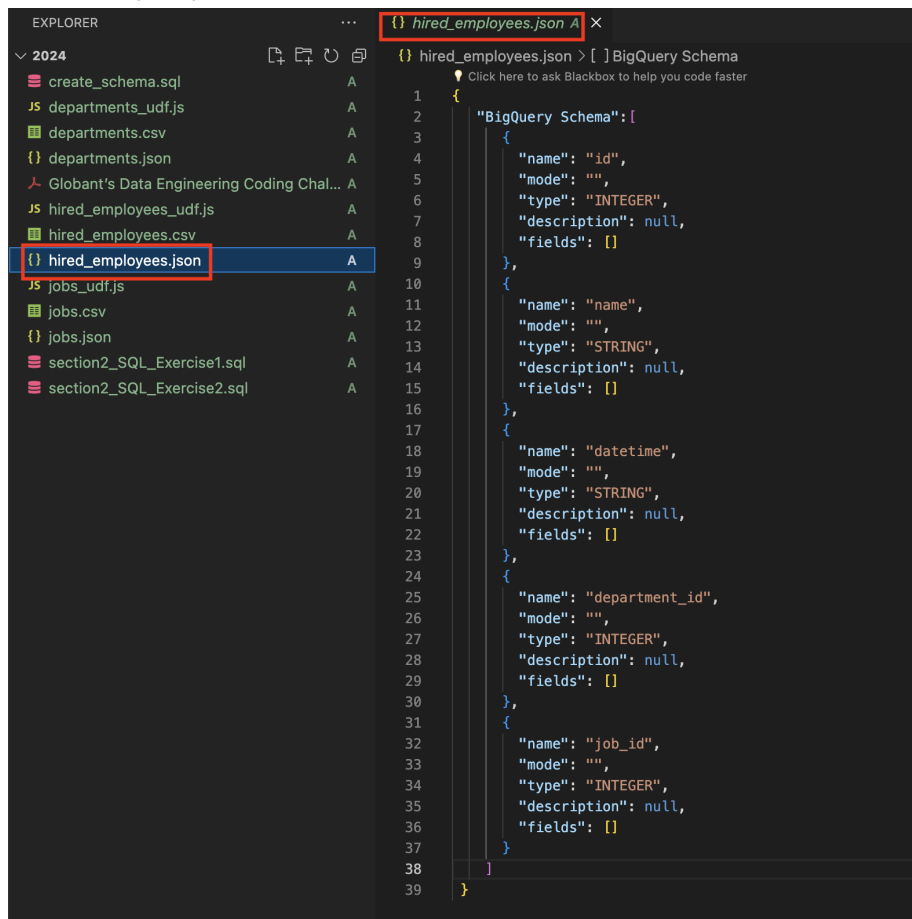
The screenshot shows the Google Cloud Dataflow console. On the left, the 'JOB GRAPH' tab is selected, showing a pipeline with four stages: 'Read from source', 'JavaScript...JavaScript', 'BigQueryConn. ToTableflow', and 'Insert into BigQuery'. The 'JavaScript...JavaScript' stage is highlighted with a red box. On the right, the 'Clone job from template' dialog is open, showing the configuration for the 'dataflow-departments-csv' job. The 'Job name' is 'dataflow-departments-csv', the 'Regional endpoint' is 'us-east1 (Columbus)', and the 'Dataflow template' is 'Text Files on Cloud Storage to BigQuery'. The 'Required Parameters' section is expanded, showing the following parameters:

- Cloud Storage Input File(s) ***: `gs://csv-input/departments.csv`
- Cloud Storage location of your BigQuery schema file, described as a JSON ***: `gs://csv-input/departments.json`
- BigQuery output table ***: `globant-test-418611:globant.departments`
- Temporary directory for BigQuery loading process ***: `gs://csv-input/temp/`
- Temporary location ***: `gs://csv-input/temp/`

hired_employees.csv

I created hired_employees.json, hired_employees_udf.js

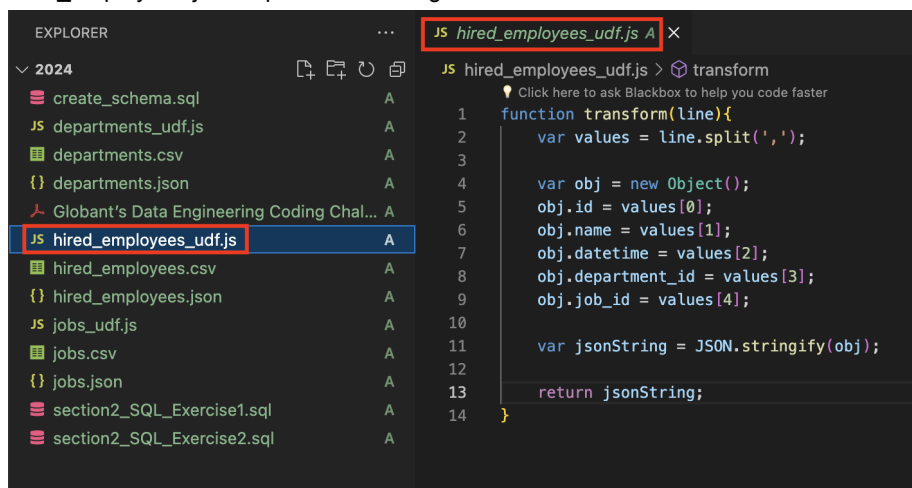
hired_employees.json: is the schema of the table



The screenshot shows the VS Code interface. In the Explorer on the left, the file `hired_employees.json` is selected and highlighted with a red box. The Editor on the right displays the content of `hired_employees.json`, which is a BigQuery Schema definition. The schema includes fields: `id` (INTEGER), `name` (STRING), `datetime` (STRING), `department_id` (INTEGER), and `job_id` (INTEGER). The file is named `hired_employees.json` and is located in the `2024` directory.

```
{
  "BigQuery Schema": [
    {
      "name": "id",
      "mode": "",
      "type": "INTEGER",
      "description": null,
      "fields": []
    },
    {
      "name": "name",
      "mode": "",
      "type": "STRING",
      "description": null,
      "fields": []
    },
    {
      "name": "datetime",
      "mode": "",
      "type": "STRING",
      "description": null,
      "fields": []
    },
    {
      "name": "department_id",
      "mode": "",
      "type": "INTEGER",
      "description": null,
      "fields": []
    },
    {
      "name": "job_id",
      "mode": "",
      "type": "INTEGER",
      "description": null,
      "fields": []
    }
  ]
}
```

hired_employees.js: Script for the csv ingestion



The screenshot shows the VS Code interface. In the Explorer on the left, the file `hired_employees_udf.js` is selected and highlighted with a red box. The Editor on the right displays the content of `hired_employees_udf.js`, which is a JavaScript function named `transform` that takes a line of CSV data and returns a JSON string. The function splits the line by commas and maps the values to the fields defined in the schema: `id`, `name`, `datetime`, `department_id`, and `job_id`. The file is named `hired_employees_udf.js` and is located in the `2024` directory.

```
function transform(line){
  var values = line.split(',');

  var obj = new Object();
  obj.id = values[0];
  obj.name = values[1];
  obj.datetime = values[2];
  obj.department_id = values[3];
  obj.job_id = values[4];

  var jsonString = JSON.stringify(obj);

  return jsonString;
}
```

Dataflow Job

The screenshot shows the Google Cloud Dataflow console. On the left, the 'JOB GRAPH' tab is active, displaying a pipeline with four stages: 'Read from source' (Succeeded, 7 sec, 2 of 2 stages succeeded), 'JavaScript...JavaScript' (Succeeded, 2 sec, 1 of 1 stage succeeded), 'BigQueryCon...ToTableflow' (Succeeded, 6 sec, 1 of 1 stage succeeded), and 'Insert into Bigquery' (Succeeded, 8 sec, 20 of 20 stages succeeded). On the right, the 'Clone job from template' panel is open, showing the job name 'dataflow-hired_employees-csv', the regional endpoint 'us-east5 (Columbus)', and the Dataflow template 'Text Files on Cloud Storage to BigQuery'. Below these, the 'Required Parameters' section is visible, including 'Cloud Storage Input File(s)', 'Cloud Storage location of your BigQuery schema file', 'BigQuery output table', 'Temporary directory for BigQuery loading process', and 'Temporary location'.

Exercise 1

The screenshot shows a SQL editor with a file explorer on the left and a query editor on the right. The file explorer lists various files, including 'section2_SQL_Exercise1.sql', which is highlighted. The query editor contains the following SQL code:

```
with
source as (
  SELECT hired.id,
         case when substr(hired.datetime, 0,10) <> ""
              then EXTRACT(QUARTER FROM date(substr(hired.datetime, 0,10)))
              else null
         end as quarter,
         job.job,
         dep.department
  FROM `globant-test-418611.globant.hired_employees` hired
  left join `globant-test-418611.globant.departments` dep on hired.department_id = dep.id
  left join `globant-test-418611.globant.jobs` job on hired.job_id = job.id
  WHERE date(substr(hired.datetime, 0,10)) BETWEEN '2021-01-01' AND '2021-12-31'
      AND substr(hired.datetime, 0,10) <> ""
)

SELECT DEPARTMENT,
       JOB,
       COUNT(CASE WHEN QUARTER = 1 THEN 1 ELSE NULL END) AS Q1,
       COUNT(CASE WHEN QUARTER = 2 THEN 1 ELSE NULL END) AS Q2,
       COUNT(CASE WHEN QUARTER = 3 THEN 1 ELSE NULL END) AS Q3,
       COUNT(CASE WHEN QUARTER = 4 THEN 1 ELSE NULL END) AS Q4
FROM SOURCE
GROUP BY 1,2
ORDER BY 1,2
```

Exercise 2

```
EXPLORER
2024
create_schema.sql
departments_udf.js
departments.csv
departments.json
Globant's Data Engineering Coding Chal...
hired_employees_udf.js
hired_employees.csv
hired_employees.json
jobs_udf.js
jobs.csv
jobs.json
section2_SQL_Exercise1.sql
section2_SQL_Exercise2.sql

section2_SQL_Exercise2.sql
Click here to ask Blackbox to help you code faster
1
2 -- List of ids, name and number of employees hired of each department that hired more
3 -- employees than the mean of employees hired in 2021 for all the departments, ordered
4 -- by the number of employees hired (descending).
5
6
7
8 SELECT
9     dep.id,
10    dep.department,
11    count(*) as hired
12 FROM `globant-test-418611.globant.hired_employees` hired
13 left join `globant-test-418611.globant.departments` dep on hired.department_id = dep.id
14 left join `globant-test-418611.globant.jobs` job on hired.job_id = job.id
15 GROUP BY 1,2
16 having count(*) > (SELECT AVG(CNT) AS AVG_CNT
17                    FROM(
18                        SELECT DEPARTMENT_ID,COUNT(ID) AS CNT
19                        FROM `globant-test-418611.globant.hired_employees` hired
20                        where date(substr(hired.datetime, 0,10)) BETWEEN '2021-01-01' AND '2021-12-31'
21                        AND substr(hired.datetime, 0,10)<>""
22                        group by 1
23                    ))
24 order by 3 desc
```