Lab Manual

Data Warehousing Mining

CT-463

Computer Science & Information Technology

Name: Fatima Shehzad
Roll No: CT-20036
Section: A

Lab 1

Q1) Now create tables and insert at least 5 campuses in Campus_degree, 5 student in each campus and 10 course marks for each student.

QUERY FOR CREATING TABLE:

CREATE TABLE Campus_degree (

Campus_id NUMBER PRIMARY KEY,

Campus VARCHAR2(50),

Degree VARCHAR2(50)

);

QUERY FOR INSERTION IN TABLE:

INSERT INTO Campus_degree (Campus_id, Campus, Degree) VALUES (1, 'Karachi Campus', 'Computer Science');

INSERT INTO Campus_degree (Campus_id, Campus, Degree) VALUES (2, 'Lahore Campus', 'Electrical Engineering');

INSERT INTO Campus_degree (Campus_id, Campus, Degree) VALUES (3, 'Islamabad Campus', 'Business

Administration');

INSERT INTO Campus_degree (Campus_id, Campus, Degree) VALUES (4, 'Faisalabad Campus', 'Mechanical

Engineering');

CREATE TABLE Student_Performance (Sid

NUMBER,

Course_code VARCHAR2(10),

Marks NUMBER,

```sql
 PRIMARY KEY (Sid, Course_code),
 FOREIGN KEY (Sid) REFERENCES Campus_degree (Campus_id)
);

CREATE TABLE Student_Campus ( Sid
 NUMBER PRIMARY KEY,
 Campus_id NUMBER,
 FOREIGN KEY (Sid) REFERENCES Campus_degree (Campus_id),
 FOREIGN KEY (Campus_id) REFERENCES Campus_degree (Campus_id)  );
  INSERT INTO Student_Campus (Sid, Campus_id) VALUES (1, 1);
  INSERT INTO Student_Campus (Sid, Campus_id) VALUES (2, 2);
   INSERT INTO Student_Campus (Sid, Campus_id) VALUES (3, 3);
  INSERT INTO Student_Campus (Sid, Campus_id) VALUES (4, 4);
  INSERT INTO Student_Campus (Sid, Campus_id) VALUES (5, 5);
  INSERT INTO Student_Performance (Sid, Course_code, Marks) VALUES (1, 'CS101', 90);
  INSERT INTO Student_Performance (Sid, Course_code, Marks) VALUES (2, 'EE101', 85);
  INSERT INTO Student_Performance (Sid, Course_code, Marks) VALUES (3, 'BA201', 78);
  INSERT INTO Student_Performance (Sid, Course_code, Marks) VALUES (4, 'ME101', 92);
  INSERT INTO Student_Performance (Sid, Course_code, Marks) VALUES (5, 'MED101', 88);
```

Q2) Perform equijoin, outer join on above table

EQUIJOIN

SELECT Student_Campus.Sid, Campus_degree.Campus, Campus_degree.Degree FROM

Student_Campus

JOIN Campus_degree ON Student_Campus.Campus_id =
Campus_degree.Campus_id;

LEFT JOIN

SELECT Student_Campus.Sid, Campus_degree.Campus, Campus_degree.Degree FROM

Student_Campus

LEFT JOIN Campus_degree ON Student_Campus.Campus_id =
Campus_degree.Campus_id;

RIGHT OUTER JOIN

SELECT Student_Campus.Sid, Campus_degree.Campus, Campus_degree.Degree FROM

Student_Campus

RIGHT JOIN Campus_degree ON Student_Campus.Campus_id =
Campus_degree.Campus_id;

Q3) Perform group by clause using Student_Performance

QUERY:

SELECT Course_code, AVG(Marks) AS AverageMarks

FROM Student_Performance

GROUP BY Course_code;

Lab 2

Q1) Create sql query of all above denormalization techniques by using HR or student_performance schema

DENORMALIZATION TECHNIQUES USING student_performance SCHEMA:

CREATE TABLE Denormalized_Student_Performance AS

SELECT SP.Sid, SP.Course_code, SP.Marks, SC.Campus_id, CD.Campus, CD.Degree

FROM Student_Performance SP

JOIN Student_Campus SC ON SP.Sid = SC.Sid

JOIN Campus_degree CD ON SC.Campus_id = CD.Campus_id;

COLLAPSING:

CREATE TABLE Denormalized_Employee_Manager AS
SELECT e.employee_id, e.first_name, e.last_name, e.salary * 12 AS annual_salary, m.employee_id AS manager_id, m.first_name AS manager_first_name, m.last_name AS manager_last_name FROM employees e
LEFT JOIN employees m ON e.manager_id = m.employee_id;

HORIZONTAL SPLITTING:

CREATE TABLE Employees_Hired_Before AS SELECT * FROM employees
WHERE hire_date < TO_DATE('01-JAN-2000', 'DD-MON-YYYY');

VERTICAL SPLITTING:

CREATE TABLE Employee_Basic_Info AS

SELECT employee_id, first_name, last_name, email, hire_date FROM employees

Lab 3

Q No 1: Create a query to display all data in sal_fact table.

```
SELECT
    MONTH_ID, CATEGORY_ID,
    STATE_PROVINCE_ID, UNITS,
    SALES
FROM AV.SALES_FACT;
```

Q No 2: Write a query to display quarterly sales of departments in different region.

```
SELECT
    G.REGION_NAME, P.DEPARTMENT_NAME,
    T.QUARTER_NAME, SUM(S.SALES)
    QUATERLY_SALES
FROM
    AV.SALES_FACT S, AV.TIME_DIM
    T, AV.GEOGRAPHY_DIM G,
    AV.PRODUCT_DIM P
WHERE
    S.MONTH_ID = T.MONTH_ID AND S.STATE_PROVINCE_ID =
    G.STATE_PROVINCE_ID AND S.CATEGORY_ID = P.CATEGORY_ID
GROUP BY
    G.REGION_NAME,
    P.DEPARTMENT_NAME,
    T.QUARTER_NAME,
    T.QUARTER_END_DATE
ORDER BY
    G.REGION_NAME,
    P.DEPARTMENT_NAME,
    T.QUARTER_END_DATE;
```

Q No 3: Write a query to display quarterly sales of country.

```
SELECT
    G.COUNTRY_NAME,
    T.QUARTER_NAME, SUM(S.SALES)
    QUATERLY_SALES
FROM
    AV.SALES_FACT S,
    AV.TIME_DIM T,
    AV.GEOGRAPHY_DIM G
WHERE
    S.MONTH_ID = T.MONTH_ID AND S.STATE_PROVINCE_ID
    = G.STATE_PROVINCE_ID
GROUP BY
    G.COUNTRY_NAME,
    T.QUARTER_NAME,
    T.QUARTER_END_DATE
ORDER BY
    G.COUNTRY_NAME,
    T.QUARTER_END_DATE;
```

Q No 4: Write a query to display yearly unit sold in different region.

```
SELECT
    G.REGION_NAME,
    T.YEAR_NAME,
    SUM(S.UNITS) YEARLY_UNITS_SOLD FROM
    AV.SALES_FACT S,
    AV.TIME_DIM T,
    AV.GEOGRAPHY_DIM G
WHERE
    S.MONTH_ID = T.MONTH_ID AND S.STATE_PROVINCE_ID
    = G.STATE_PROVINCE_ID
GROUP BY
    G.REGION_NAME,
    T.YEAR_NAME
ORDER BY
    G.REGION_NAME,
    T.YEAR_NAME;
```

Q No 5: Write a query to display monthly sales and unit sold in different region.

```
SELECT
    G.REGION_NAME,
    T.MONTH_NAME,
      SUM(S.SALES) MONTHLY_SALES,
    SUM(S.UNITS) MONTHLY_UNITS_SOLD
FROM
    AV.SALES_FACT S,
    AV.TIME_DIM T,
    AV.GEOGRAPHY_DIM G
WHERE
    S.MONTH_ID = T.MONTH_ID AND S.STATE_PROVINCE_ID
    = G.STATE_PROVINCE_ID
GROUP BY
    G.REGION_NAME,
    T.MONTH_NAME,
    T.MONTH_END_DATE
ORDER BY
    G.REGION_NAME,
    T.MONTH_END_DATE;
```

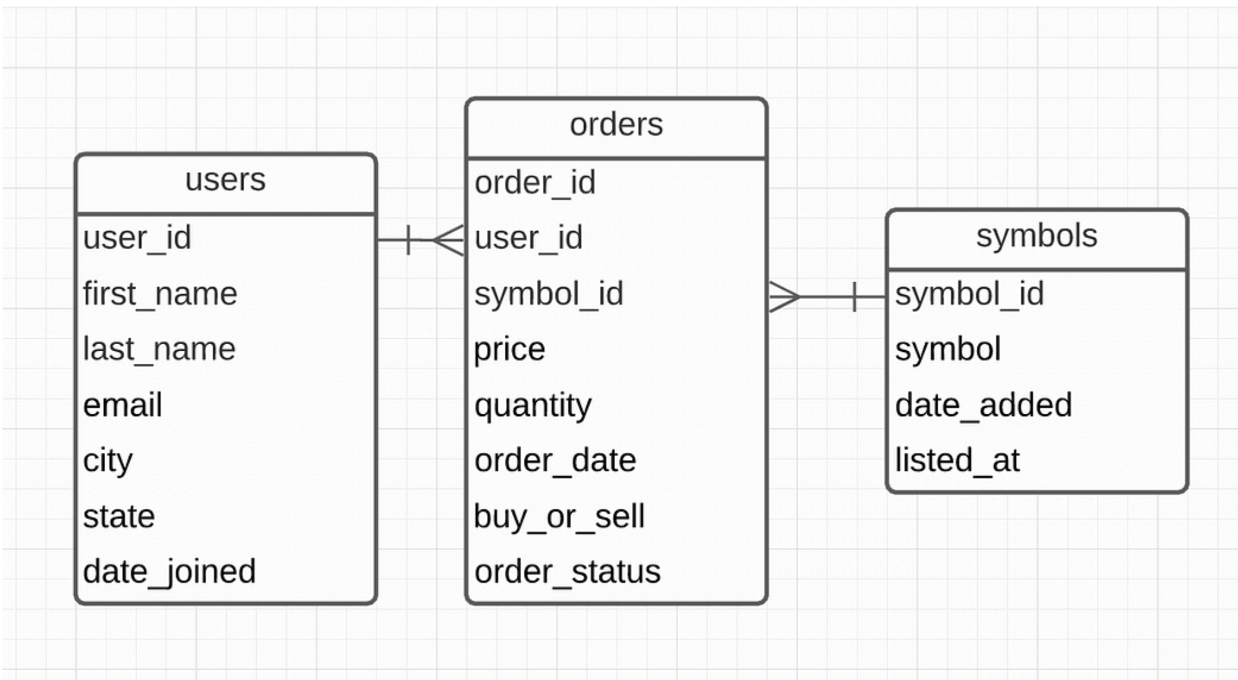
Q No 6: Write a query to display seasonally sales in different province.

```
SELECT
    G.STATE_PROVINCE_NAME,
    T.SEASON,
      SUM(S.SALES) SEASONAL_SALES
FROM
    AV.SALES_FACT S,
    AV.TIME_DIM T,
    AV.GEOGRAPHY_DIM G
WHERE
    S.MONTH_ID = T.MONTH_ID AND S.STATE_PROVINCE_ID
    = G.STATE_PROVINCE_ID
GROUP BY
    G.STATE_PROVINCE_NAME, T.SEASON
ORDER BY
    G.STATE_PROVINCE_NAME,
    T.SEASON;
```

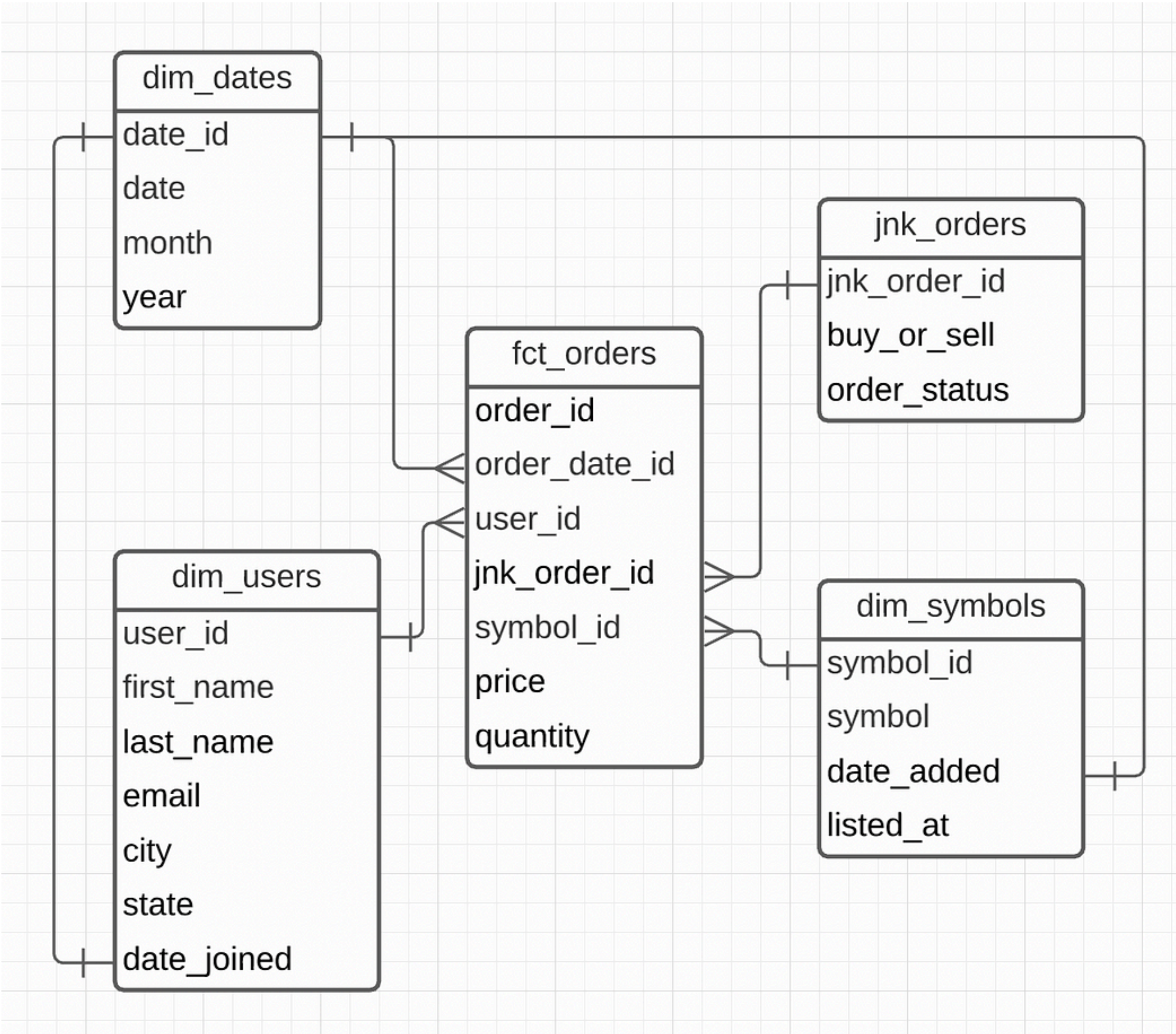
Lab. 4

Excersize: Refers to what has been seen in class. Using ERDPLUS , design star schema using any ERD except HR or class lecture erd and generate code and execute this on sql live or oracle apex.

Entity Realtionship Diagram of Stock Market Trading:



Star Schema:



dim_dates
date_id
date
month
year
jnk_orders
jnk_order_id
buy_or_sell
order_status
fct_orders
order_id
order_date_id
user_id
jnk_order_id
symbol_id
price
quantity
dim_users
user_id
first_name
last_name
email
city
state
date_joined
dim_symbols
symbol_id
symbol
date_added
listed_at

SQL Queries :

```
 CREATE TABLE fct_orders (SELECT
  order_id,
  order_date as order_date_id,
  user_id,
  SHA256(CONCAT(buy_or_sell, order_status)) as jnk_order_id,
  symbol_id,
  price,
  quantity
FROM `gcpuser-project.raw_hands_on_star_schema.orders`
)

CREATE TABLE jnk_orders (WITH values AS (
  SELECT DISTINCT buy_or_sell, order_status
  FROM `gcpuser-project.raw_hands_on_star_schema.orders`
),apply_surrogate_key AS (
  SELECT
    SHA256(CONCAT(buy_or_sell, order_status)) AS jnk_orders_id,
    buy_or_sell,
    order_status
  FROM values
)SELECT *
FROM apply_surrogate_key
)

CREATE TABLE  dim_dates (
  SELECT
    date AS date_id,
    date,
    EXTRACT(MONTH FROM date) AS month,
    EXTRACT(YEAR FROM date) AS year
  FROM UNNEST(
    GENERATE_DATE_ARRAY('2014-01-01', CURRENT_DATE('America/New_York'), INTERVAL 1 DAY)
  ) AS date
```

```
CREATE TABLE dim_users  (
   user_id INT NOT NULL,
 first_name VARCHAR(20) NOT NULL,
 last_name VARCHAR(50) NOT NULL,
 email VARCHAR(20) NOT NULL,
 city VARCHAR(20) NOT NULL,
 state VARCHAR(20) NOT NULL,
  dated_joined VARCHAR(20) NOT NULL,
 PRIMARY KEY (user_id)

 )

CREATE TABLE dim_symbols (
   symbol_id INT NOT NULL,
 symbol VARCHAR(20) NOT NULL,
 date_added VARCHAR(50) NOT NULL,
 listed_at VARCHAR(20) NOT NULL
 PRIMARY KEY (symbol_id)

 )
```

Lab 5

Q 1. Create a query to perform roll up operations using the av.sales_fact table. (Note:-examine all data)

```
SELECT MONTH_ID,
   CATEGORY_ID,
   STATE_PROVINCE_ID,
   SUM(UNITS) AS TOTAL_UNITS,
   SUM(SALES) AS TOTAL_SALES
FROM AV.SALES_FACT
GROUP BY ROLLUP(MONTH_ID, CATEGORY_ID, STATE_PROVINCE_ID);
```

Q 2. Create a query to perform cube operation using av.sales_fact table. (Note: Also examine the data)

```
SELECT MONTH_ID, CATEGORY_ID,
STATE_PROVINCE_ID, SUM(UNITS) AS TOTAL_UNITS, SUM(SALES) AS TOTAL_SALES
FROM AV.SALES_FACT
GROUP BY CUBE(MONTH_ID, CATEGORY_ID, STATE_PROVINCE_ID);
```

Q 3. Why do we use roll up, cube and pivot function and what are the differences between them?

The usage of rollup, cube and pivot functions are as follows along with their significant differences:

Roll-up:

- Used for hierarchical aggregations.
- Provides subtotals and grand totals.
- Performs aggregation on a set of columns.
- Result has progressively higher levels of aggregation.
- Suitable for examining data at different levels of granularity in a hierarchy.

Cube:

- Similar to roll-up but more comprehensive.
- Generates all possible subtotals.
- Produces a result set with all combinations of aggregations.
- Valuable for multi-dimensional analysis.
- Useful when exploring data from various dimensions.

Pivot:

- Used to change the view of data.
- Transforms rows into columns based on unique values.
- Involves rotating a table-valued expression.
- Performs aggregations where required.
- Enhances the view of data and allows for concise summarization.

Q 4. Perform pivot operation using AV schema.

```
SELECT * FROM (
SELECT MONTH_ID, CATEGORY_ID, UNITS FROM AV.SALES_FACT
) PIVOT(
SUM(UNITS) FOR MONTH_ID IN ('Jan', 'Feb', 'Mar')
);
```

Q 5. Create pivot operation using SH schema.

```
SELECT *
FROM (
SELECT MONTH_ID, CATEGORY_ID, UNITS FROM SH.SALES_FACT
) PIVOT(
SUM(UNITS) FOR MONTH_ID IN ('Jan', 'Feb', 'Mar')
);
```

Q 6. Create a view using cube operation.

```
CREATE VIEW AV.CubeView AS SELECT
MONTH_ID, CATEGORY_ID, STATE_PROVINCE_ID,
SUM(UNITS) AS TOTAL_UNITS, SUM(SALES) AS TOTAL_SALES
FROM AV.SALES_FACT
GROUP BY CUBE(MONTH_ID, CATEGORY_ID, STATE_PROVINCE_ID);
```

Lab 6

```
CREATE TABLE EMPLOYEES1 AS SELECT * FROM HR.EMPLOYEES;
```

Q No 1: Create a materialized view that stores the last name and salary of employees earning more than $12,000.

```
CREATE MATERIALIZED VIEW mv_12k_salary_employees AS
    SELECT LAST_NAME, SALARY
    FROM EMPLOYEES1 WHERE
    SALARY > 12000;
```

Q No 2: Create a materialized view called EMPLOYEES_VU on the employee numbers, employee names, and department numbers from the EMPLOYEES table

```
CREATE MATERIALIZED VIEW EMPLOYEES_VU AS
    SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, DEPARTMENT_ID FROM
    EMPLOYEES1;
```

Q No 3: Insert one employee into the employees table then Display the contents of the EMPLOYEES_VU view also what is the difference between employees table and materialized view.

```
-- Inserting a new employee into the EMPLOYEES1 table
INSERT INTO EMPLOYEES1
    (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE,
JOB_ID, SALARY, DEPARTMENT_ID)
    VALUES
    (50, 'John', 'Doe', 'julia@example.com', '515.123.4951',
TO_DATE('23-DEC-23', 'DD-MON-RR'), 'IT_PROG', 25000, 60);

-- Displaying the contents of EMPLOYEES1
SELECT * FROM EMPLOYEES1 WHERE EMPLOYEE_ID = 50;

-- -- Displaying the contents of the EMPLOYEES_VU materialized view SELECT
* FROM EMPLOYEES_VU WHERE EMPLOYEE_ID = 50;

The materialized view is not updated with the new inserted data (i.e
EMPLOYEE_ID = 50)
```

Q No 4: Write a query to manually refresh EMPLOYEES_VU.

```
EXEC DBMS_MVIEW.refresh('EMPLOYEES_VU')
```

Q No 5: Create a materialized view that stores the number of people with the same job.

```
CREATE MATERIALIZED VIEW job_count_mv AS SELECT
    JOB_ID, COUNT(*) AS employee_count FROM
    EMPLOYEES1
    GROUP BY JOB_ID;
```

Q No 6: Modify Lab 4 3.sql create a materialized view that Oracle refreshes automatically.

```
-- MODIFIED
CREATE TABLE EMPLOYEES_MODIFIED AS SELECT * FROM HR.EMPLOYEES;

-- Create materialized view log on EMPLOYEES_MODIFIED
CREATE MATERIALIZED VIEW LOG ON EMPLOYEES_MODIFIED WITH
ROWID;

-- Create materialized view EMPLOYEES_VU_MODIFIED with fast refresh on commit
CREATE MATERIALIZED VIEW EMPLOYEES_VU_MODIFIED REFRESH
    FAST ON COMMIT
    WITH ROWID
    AS
    SELECT * FROM EMPLOYEES_MODIFIED;

INSERT INTO EMPLOYEES_MODIFIED
    (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE,
JOB_ID, SALARY, DEPARTMENT_ID)
    VALUES (150, 'John', 'Doe', 'julia@example.com', '515.123.4951',
TO_DATE('23-DEC-23', 'DD-MON-RR'), 'IT_PROG', 25000, 60); SELECT
* FROM EMPLOYEES_VU_MODIFIED WHERE FIRST_NAME = 'John'
```

Q No 7: Create a materialized view that stores the difference between the highest and lowest salaries. Label the column DIFFERENCE.

```
CREATE MATERIALIZED VIEW salary_difference_mv AS SELECT
    MAX(SALARY) - MIN(SALARY) AS DIFFERENCE FROM
    EMPLOYEES1;
SELECT * FROM salary_difference_mv;
```

Lab 7-8

Q No 1: Create a query to display geography, product, and time dimension data. (Note:-examine all data)

```
SELECT
    t.YEAR_ID, t.YEAR_NAME, t.YEAR_END_DATE,
    t.QUARTER_ID, t.QUARTER_NAME, t.QUARTER_END_DATE, t.QUARTER_OF_YEAR,
    t.MONTH_ID, t.MONTH_NAME, t.MONTH_END_DATE, t.MONTH_OF_YEAR,
    t.MONTH_LONG_NAME, t.SEASON, t.SEASON_ORDER, t.MONTH_OF_QUARTER,
    p.DEPARTMENT_ID, p.DEPARTMENT_NAME,
    p.CATEGORY_ID, p.CATEGORY_NAME,
    g.REGION_ID, g.REGION_NAME, g.COUNTRY_ID,
    g.COUNTRY_NAME,
    g.STATE_PROVINCE_ID, g.STATE_PROVINCE_NAME
    FROM av.sales_fact s
    JOIN av.time_dim t ON s.MONTH_ID = t.MONTH_ID
    JOIN av.product_dim p ON s.CATEGORY_ID = p.CATEGORY_ID JOIN
    av.geography_dim g ON s.STATE_PROVINCE_ID =
g.STATE_PROVINCE_ID;
```

Q No 2: Select 40 rows form sale fact table. (Note:-examine data)

```
SELECT * FROM AV.SALES_FACT WHERE ROWNUM <= 40;
```

Q No 3: Why we use analytical view and what is the difference between simple analytical view and materialize analytical view.

Use of Analytic View:

Analytic Views are used to perform complex calculations and Aggregate functions such as Sum, Count, Min, Max, etc. Analytic Views are designed to run Star schema queries. Each Analytic View has one Fact table surrounded by multiple dimension tables. Fact table contains a primary key for each Dim table and measures.

Difference between Analytical View and Materialized View:

Views are generally used when data is to be accessed infrequently and data in table get updated on frequent basis. On other hand Materialized Views are used when data is to be accessed frequently and data in table not get updated on frequent basis.

Q No 4: Create attribute dimension of time, geography and product dimension.

Dimension Attribute of Time:

```
CREATE OR REPLACE ATTRIBUTE DIMENSION time_attr_dim DIMENSION TYPE TIME
USING av.time_dim
ATTRIBUTES
    (year_id, year_name, quarter_id, quarter_name, month_id, month_name,
    month_long_name)
    LEVEL MONTH LEVEL
    TYPE MONTHS
    KEY month_id DETERMINES(quarter_id) LEVEL
    QUARTER
    LEVEL TYPE QUARTERS
    KEY quarter_id DETERMINES(year_id) LEVEL
    YEAR
    LEVEL TYPE YEARS
    KEY year_id;
```

Dimension Attribute of Geography:

```
CREATE OR REPLACE ATTRIBUTE DIMENSION geography_attr_dim USING
av.geography_dim
    ATTRIBUTES
    (region_id, region_name, country_id, country_name,
state_province_id, state_province_name )
    LEVEL REGION
    KEY region_id
    LEVEL COUNTRY
    KEY country_id DETERMINES(region_id)
    LEVEL STATE_PROVINCE
    KEY state_province_id DETERMINES(country_id);
```

Dimension Attribute of Product:

```
CREATE OR REPLACE ATTRIBUTE DIMENSION product_attr_dim USING
av.product_dim
    ATTRIBUTES
    (department_id, department_name, category_id, category_name) LEVEL
    DEPARTMENT
```

```
    KEY department_id
    LEVEL CATEGORY
    KEY category_id DETERMINES(department_id);
```

Q No 5: Create hierarchy of time, geography and product dimension.

```
-- Hierarchy of Time:
CREATE OR REPLACE HIERARCHY time_hier
USING time_attr_dim
(month CHILD OF
 quarter CHILD OF
 year);

-- Hierarchy of Geography:
CREATE OR REPLACE HIERARCHY geography_hier
USING geography_attr_dim
 (state_province
 CHILD OF country
 CHILD OF region);

-- Hierarchy of Product:
CREATE OR REPLACE HIERARCHY product_hier
USING product_attr_dim
 (CATEGORY
 CHILD OF department);
```

Q No 6: Using attribute dimension and hierarchy of time and geography dimension, create an analytical view which measures average and count of sale facts.Select sales at the year level.

```
-- Analytical View:
CREATE OR REPLACE ANALYTIC VIEW sales_av
USING av.sales_fact
DIMENSION BY
 (time_attr_dim
 KEY month_id REFERENCES month_id HIERARCHIES
 (
 time_hier DEFAULT),
 geography_attr_dim
 KEY state_province_id REFERENCES state_province_id HIERARCHIES
 (
```

```
 geography_hier DEFAULT)
 )
MEASURES
(sales FACT sales,
 units FACT units,
 avg_sales FACT sales AGGREGATE BY AVG,
 count_sales FACT sales AGGREGATE BY COUNT
)
DEFAULT MEASURE SALES;
-- Select Statement:
SELECT
 time_hier.member_name AS time,
 geography_hier.member_name AS geography,
 sales,
 units,
 ROUND(avg_sales, 3) AS yearly_average_sales,
 count_sales AS yearly_sales_count
FROM sales_av
 HIERARCHIES (
 time_hier,
geography_hier)
WHERE
 time_hier.level_name IN ('YEAR') ORDER
BY
 time_hier.member_name,
 geography_hier.member_name;
```

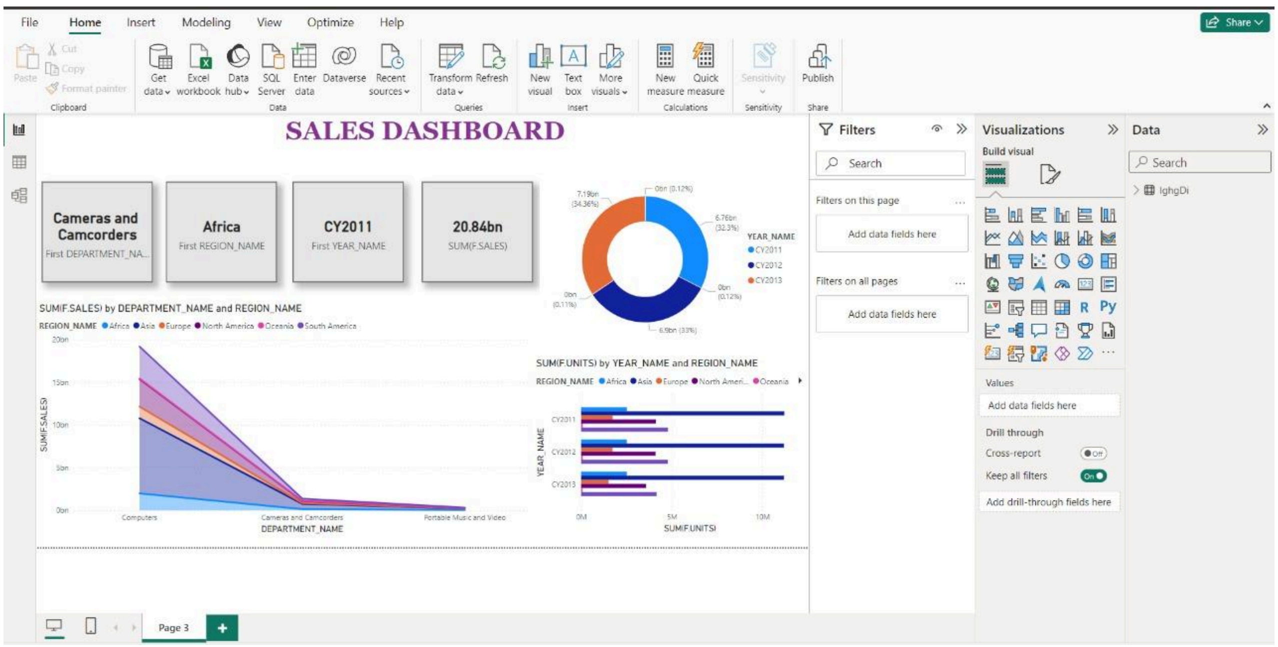
Q No 7: Using attribute dimension and hierarchy of time, geography and product dimension, create analytical view which measure max, min, count, standard deviational and variation of unit's fact.Select sales at the REGION_IDI level and CATEGORY_ID.

```
-- Analytical View:
CREATE OR REPLACE ANALYTIC VIEW sales_av
    USING av.sales_fact
    DIMENSION BY
    (time_attr_dim
    KEY  month_id  REFERENCES  month_id
    HIERARCHIES  (time_hier  DEFAULT),
    product_attr_dim
    KEY category_id REFERENCES category_id
    HIERARCHIES (product_hier DEFAULT),
    geography_attr_dim
    KEY state_province_id REFERENCES state_province_id
```

```
    HIERARCHIES (geography_hier DEFAULT)
    ) MEASURES
    (sales FACT sales,
    units FACT units,
    max_units FACT units AGGREGATE BY MAX, min_units
    FACT units AGGREGATE BY MIN, count_units FACT
    units AGGREGATE BY COUNT,
    standard_deviation_units FACT units AGGREGATE BY STDDEV,
    variance_units FACT units AGGREGATE BY VARIANCE
    )
    DEFAULT MEASURE SALES;

-- Select Statement:
SELECT
 time_hier.member_name AS time,
 geography_hier.member_name AS geography,
 product_hier.member_name AS product, sales,
 units,
 max_units,
 min_units,
 count_units,
 ROUND(standard_deviation_units, 3) AS standard_deviation_units,
 ROUND(variance_units, 3) AS variance_units
FROM sales_av
 HIERARCHIES (
 time_hier,
geography_hier,
product_hier) WHERE
 geography_hier.level_name IN ('REGION') AND
product_hier.level_name IN ('CATEGORY') ORDER
BY
 time_hier.member_name,
 geography_hier.member_name,
 product_hier.member_name;
```

Lab 9-10

Theory refers to what has been discussed in Lab (using Av schema). create dashboard using power BI and publish to all.



Lab 11-12

Theory refers to what has been discussed in Lab (using Weka) , Using different data samples, Appley classification, clustering, and association is performed with a discussion of the results.

1. DATASET OF DIABETES:

a. CLASSIFICATION RESULTS USING SMO:

```
=== Summary ===

Correctly Classified Instances         207               79.3103 %
Incorrectly Classified Instances        54               20.6897 %
Kappa statistic                          0.4902
Mean absolute error                      0.2069
Root mean squared error                  0.4549
Relative absolute error                 45.8735 %
Root relative squared error             97.1692 %
Total Number of Instances              261

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.904    0.446    0.813      0.904    0.856      0.499    0.729     0.801     tested_negative
                 0.554    0.096    0.730      0.554    0.630      0.499    0.729     0.546     tested_positive
Weighted Avg.    0.793    0.334    0.787      0.793    0.784      0.499    0.729     0.720

=== Confusion Matrix ===

   a   b   <-- classified as
 161  17 |   a = tested_negative
  37  46 |   b = tested_positive
```

DISCUSSION OF RESULT:

- The classifier achieved reasonably good accuracy.
- Class tested_negative has higher precision, recall, and F-measure compared to class tested_positive.
- The confusion matrix provides insights into how well the model is performing on each class.

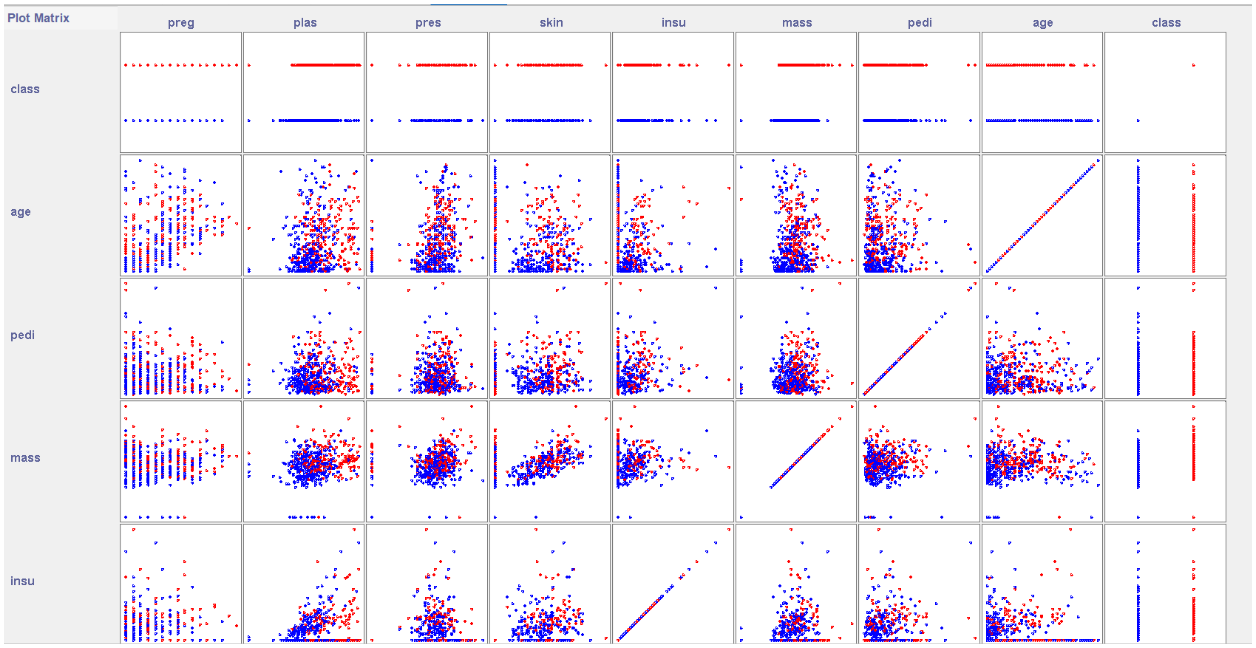
b. CLUSTERING RESULTS USING SIMPLE K MEANS:

```
Final cluster centroids:
                                          Cluster#
Attribute                  Full Data             0                1
                             (506.0)         (184.0)          (322.0)
==================================================================
preg                           3.919          4.7826           3.4255
plas                         121.164        139.3967         110.7453
pres                         69.1779         70.8587          68.2174
skin                         19.9486          21.962          18.7981
insu                          78.585         90.0217          72.0497
mass                         32.0275         35.2766          30.1708
pedi                          0.4798          0.5516           0.4388
age                          33.7925         37.4674          31.6925
class                  tested_negative tested_positive tested_negative




Time taken to build model (percentage split) : 0 seconds

Clustered Instances

0       84 ( 32%)
1      178 ( 68%)
```

DISCUSSION OF RESULT:

- The dataset is divided into two clusters with distinct centroids.
- The clusters represent groups of instances with similar attribute values.
- Clustering might reveal patterns or groups within the data.

VISUALIZATION:



2. DATASET OF LABOR:

a. CLASSIFICATION RESULTS USING RANDOM FOREST:

```
=== Summary ===

Correctly Classified Instances          17               89.4737 %
Incorrectly Classified Instances         2               10.5263 %
Kappa statistic                          0.7564
Mean absolute error                      0.2236
Root mean squared error                  0.2973
Relative absolute error                 49.2623 %
Root relative squared error             63.4471 %
Total Number of Instances               19

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.833    0.077    0.833      0.833    0.833      0.756    0.962     0.931     bad
                 0.923    0.167    0.923      0.923    0.923      0.756    0.962     0.984     good
Weighted Avg.    0.895    0.138    0.895      0.895    0.895      0.756    0.962     0.967

=== Confusion Matrix ===

  a  b   <-- classified as
  5  1 |  a = bad
  1 12 |  b = good
```

DISCUSSION OF RESULT:

- The Random Forest classifier achieved high accuracy and a substantial Kappa statistic.
- Both classes (bad and good) have good precision, recall, and F-Measure.
- The confusion matrix indicates that the model performs well in classifying instances into the correct classes.

b. CLUSTERING RESULTS USING EM:

```
vacation
  below_average                         9.9493  9.0507
  average                                6.902   5.098
  generous                              9.9949  2.0051
  [total]                              26.8462 16.1538
longterm-disability-assistance
  yes                                  24.8326  8.1674
  no                                    1.0135  6.9865
  [total]                              25.8462 15.1538
contribution-to-dental-plan
  none                                  2.0015  4.9985
  half                                  4.9806  7.0194
  full                                 19.8641  4.1359
  [total]                              26.8462 16.1538
class
  bad                                   1.0687 13.9313
  good                                 24.7775  1.2225
  [total]                              25.8462 15.1538


Time taken to build model (percentage split) : 0.05 seconds

Clustered Instances

0       12 ( 60%)
1        8 ( 40%)


Log likelihood: -7.82877
```

DISCUSSION OF RESULT:

- The dataset is divided into two clusters, each with distinct attribute distributions.
- Clustering can help identify patterns or subgroups within the data.

c. <u>ASSOCIATION RESULTS USING FILTERED ASSOCIATOR:</u>

```
=== Run information ===

Scheme:       weka.associations.FilteredAssociator -F "weka.filters.MultiFilter -F \"weka.filters.unsupervised.a
Relation:     labor-neg-data-weka.filters.unsupervised.attribute.Remove-R1,4,6-10,15-16
Instances:    57
Attributes:   8
              wage-increase-first-year
              wage-increase-second-year
              cost-of-living-adjustment
              statutory-holidays
              vacation
              longterm-disability-assistance
              contribution-to-dental-plan
              class
```

DISCUSSION OF RESULT:

- Association rules can provide insights into relationships between different attributes.
- The rules can help understand how attributes are associated within the dataset.

VISUALIZATION:

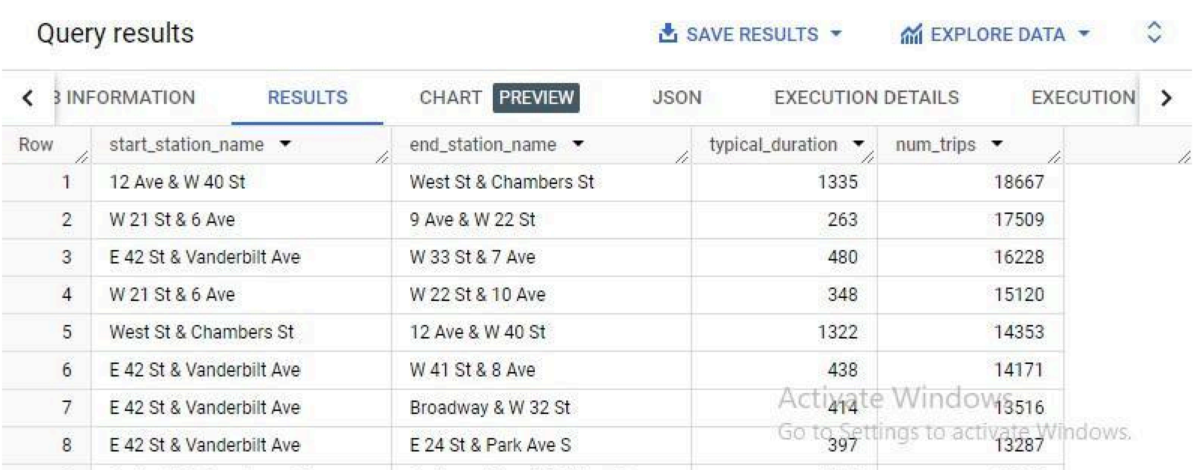
Lab 13

Q No 1: Using BigQuery to perform six queries on different data sets.

Bicycle Rental Dataset

Query # 1

```
SELECT
MIN(start_station_name) AS start_station_name,

MIN(end_station_name) AS end_station_name,
APPROX_QUANTILES(tripduration, 10)[OFFSET (5)] AS typical_duration,
COUNT(tripduration) AS num_trips
FROM
`bigquery-public-data.new_york_citibike.citibike_trips`
WHERE
start_station_id != end_station_id GROUP
BY
start_station_id,
end_station_id ORDER
BY
num_trips DESC LIMIT
10
```



Row	start_station_name	end_station_name	typical_duration	num_trips
1	12 Ave & W 40 St	West St & Chambers St	1335	18667
2	W 21 St & 6 Ave	9 Ave & W 22 St	263	17509
3	E 42 St & Vanderbilt Ave	W 33 St & 7 Ave	480	16228
4	W 21 St & 6 Ave	W 22 St & 10 Ave	348	15120
5	West St & Chambers St	12 Ave & W 40 St	1322	14353
6	E 42 St & Vanderbilt Ave	W 41 St & 8 Ave	438	14171
7	E 42 St & Vanderbilt Ave	Broadway & W 32 St	414	13516
8	E 42 St & Vanderbilt Ave	E 24 St & Park Ave S	397	13287
9	Centre St & Chambers St	Cadman Plaza E & Tillary St	1505	12280

Query # 2

```sql
WITH
trip_distance AS (
SELECT
bikeid,
ST_Distance(ST_GeogPoint(s.longitude,
s.latitude), ST_GeogPoint(e.longitude,
e.latitude)) AS distance
FROM
`bigquery-public-data.new_york_citibike.citibike_trips`,
`bigquery-public-data.new_york_citibike.citibike_stations` as s,
`bigquery-public-data.new_york_citibike.citibike_stations` as e WHERE
start_station_name = s.name AND
end_station_name = e.name)
SELECT
bikeid,
SUM(distance)/1000 AS total_distance FROM
trip_distance
GROUP BY
bikeid ORDER
BY
total_distance DESC LIMIT
5
```

Query results

B INFORMATION | RESULTS | CHAI

Row	bikeid	total_distance
1	19455	7481.830252382...
2	15731	7178.329343452...
3	17955	7130.726600576...
4	17747	7129.427058716...
5	15029	7108.773107499...

Weather Dataset

Query # 3

```
SELECT
wx.date, wx.value/10.0
AS prcp FROM
`bigquery-public-data.ghcn_d.ghcnd_2015` AS wx WHERE
id = "USW00094728"
AND qflag IS NULL AND
element = "PRCP" ORDER
BY
wx.date
```

Query results

JOB INFORMATION RESULTS

Row	date	prcp
1	2015-01-01	0.0
2	2015-01-02	0.0
3	2015-01-03	18.0
4	2015-01-04	7.6
5	2015-01-05	0.0
6	2015-01-06	1.3
7	2015-01-07	0.0
8	2015-01-08	0.0

Correlation Between Rain And Bicycle Rentals

Query # 4

```
WITH bicycle_rentals AS (
SELECT
COUNT(starttime) as num_trips, EXTRACT(DATE
from starttime) as trip_date
FROM `bigquery-public-data.new_york_citibike.citibike_trips` GROUP
BY trip_date
),
rainy_days AS (
SELECT
date,
(MAX(prcp) > 5) AS rainy FROM
(
SELECT
wx.date AS date,
IF (wx.element = "PRCP", wx.value/10, NULL) AS prcp FROM
`bigquery-public-data.ghcn_d.ghcnd_2015` AS wx WHERE
wx.id = "USW00094728"
)
GROUP BY
date
) SELECT
ROUND(AVG(bk.num_trips)) AS num_trips,
wx.rainy
FROM bicycle_rentals AS bk
JOIN rainy_days AS wx
ON wx.date = bk.trip_date
GROUP BY wx.rainy
```

Query results

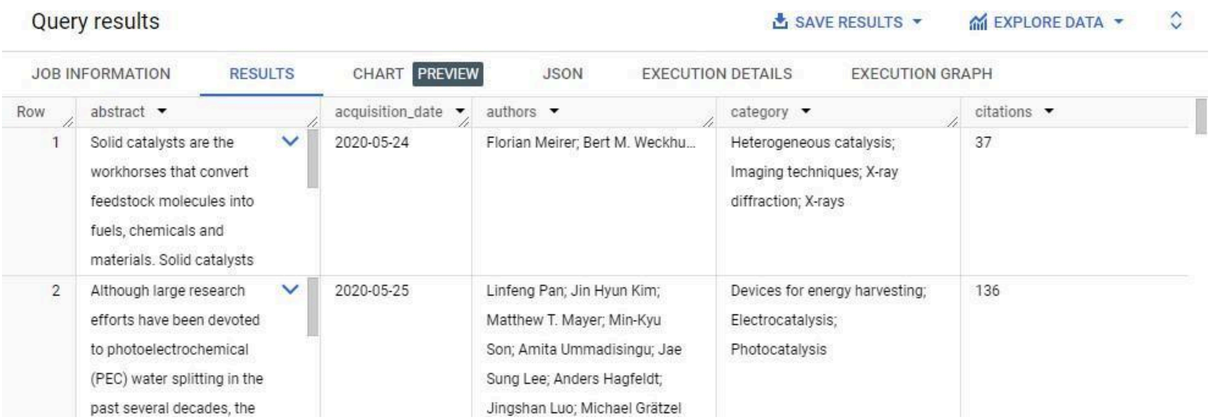
B INFORMATION RESULTS

Row	num_trips	rainy
1	28598.0	false
2	19503.0	true

Breathe

Query # 5

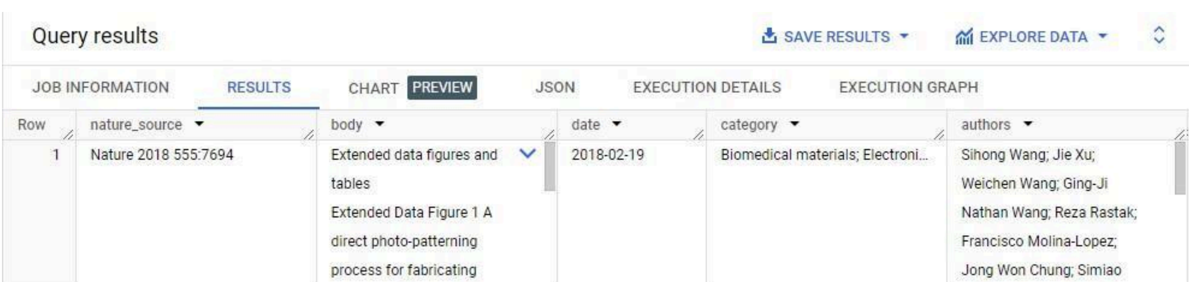
```
SELECT * FROM `bigquery-public-data.breathe.nature` LIMIT 1000
```



Row	abstract	acquisition_date	authors	category	citations
1	Solid catalysts are the workhorses that convert feedstock molecules into fuels, chemicals and materials. Solid catalysts	2020-05-24	Florian Meirer; Bert M. Weckhu...	Heterogeneous catalysis; Imaging techniques; X-ray diffraction; X-rays	37
2	Although large research efforts have been devoted to photoelectrochemical (PEC) water splitting in the past several decades, the	2020-05-25	Linfeng Pan; Jin Hyun Kim; Matthew T. Mayer; Min-Kyu Son; Amita Ummadisingu; Jae Sung Lee; Anders Hagfeldt; Jingshan Luo; Michael Grätzel	Devices for energy harvesting; Electrocatalysis; Photocatalysis	136

Query # 6

```
SELECT nature_source, body, date, category, authors FROM
`bigquery-public-data.breathe.nature`
WHERE authors="Sihong Wang; Jie Xu; Weichen Wang; Ging-Ji Nathan Wang; Reza Rastak;
Francisco Molina-Lopez; Jong Won Chung; Simiao Niu; Vivian R. Feig; Jeffery Lopez; Ting
Lei; Soon-Ki Kwon; Yeongin Kim; Amir M. Foudeh; Anatol Ehrlich; Andrea Gasperini;
Youngjun Yun; Boris Murmann; Jeffery B.-H. Tok; Zhenan Bao"
```



Row	nature_source	body	date	category	authors
1	Nature 2018 555:7694	Extended data figures and tables Extended Data Figure 1 A direct photo-patterning process for fabricating	2018-02-19	Biomedical materials; Electroni...	Sihong Wang; Jie Xu; Weichen Wang; Ging-Ji Nathan Wang; Reza Rastak; Francisco Molina-Lopez; Jong Won Chung; Simiao