**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.**

**CREATE TABLE QUERY:**

CREATE TABLE Campus\_degree (

Campus\_id NUMBER PRIMARY KEY,

Campus VARCHAR2(50),

Degree VARCHAR2(50)

);

**INSERTION IN TABLE QUERY:**

INSERT INTO Campus\_degree (Campus\_id, Campus, Degree) VALUES (1, &#39;Karachi Campus&#39;,

&#39;Computer Science&#39;);

INSERT INTO Campus\_degree (Campus\_id, Campus, Degree) VALUES (2, &#39;Lahore Campus&#39;,

&#39;Electrical Engineering&#39;);

INSERT INTO Campus\_degree (Campus\_id, Campus, Degree) VALUES (3, &#39;Islamabad

Campus&#39;, &#39;Business

Administration&#39;);

INSERT INTO Campus\_degree (Campus\_id, Campus, Degree) VALUES (4,

&#39;Faisalabad Campus&#39;, &#39;Mechanical

Engineering&#39;);

CREATE TABLE Student\_Performance ( Sid

NUMBER,

Course\_code VARCHAR2(10),

Marks NUMBER,

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, &#39;CS101&#39;, 90);

INSERT INTO Student\_Performance (Sid, Course\_code, Marks) VALUES (2, &#39;EE101&#39;, 85);

INSERT INTO Student\_Performance (Sid, Course\_code, Marks) VALUES (3, &#39;BA201&#39;, 78);

INSERT INTO Student\_Performance (Sid, Course\_code, Marks) VALUES (4, &#39;ME101&#39;, 92);

INSERT INTO Student\_Performance (Sid, Course\_code, Marks) VALUES (5, &#39;MED101&#39;, 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**

**Q) 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 &lt; TO\_DATE(&#39;01-JAN-2000&#39;, &#39;DD-MON-YYYY&#39;);

VERTICAL SPLITTQUERY ING:

CREATE TABLE Employee\_Basic\_Info AS

SELECT employee\_id, first\_name, last\_name, email, hire\_date FROM

employees;

**LAB 3**

**Q 1) Create a query to display all data in sal\_fact table.**

SELECT \* from AV.&quot;SALES\_FACT&quot;;

**Q 2) Write a query to display quarterly sales of departments in different region.**

SELECT

MONTH\_ID,

CATEGORY\_ID,

SUBSTR(MONTH\_ID, 1, 3) AS Quarter,

STATE\_PROVINCE\_ID,

SUM(SALES) AS Quarterly\_Sales

FROM

AV.SALES\_FACT

GROUP BY

MONTH\_ID,

CATEGORY\_ID,

SUBSTR(MONTH\_ID, 1, 3),

STATE\_PROVINCE\_ID;

**Q 3) Write a query to display quarterly sales of country**

SELECT

MONTH\_ID,

SUBSTR(MONTH\_ID, 1, 3) AS Quarter,

SUBSTR(STATE\_PROVINCE\_ID, INSTR(STATE\_PROVINCE\_ID, &#39;\_&#39;) + 1) AS Country,

SUM(SALES) AS Quarterly\_Sales

FROM

AV.SALES\_FACT

GROUP BY

MONTH\_ID,

SUBSTR(MONTH\_ID, 1, 3),

SUBSTR(STATE\_PROVINCE\_ID, INSTR(STATE\_PROVINCE\_ID, &#39;\_&#39;) + 1);

**Q 4) Write a query to display yearly unit sold in different region**

SELECT

SUBSTR(MONTH\_ID, -2) AS Year,

STATE\_PROVINCE\_ID,

SUM(UNITS) AS Yearly\_Units\_Sold

FROM AV.SALES\_FACT

GROUP BY

SUBSTR(MONTH\_ID, -2),

STATE\_PROVINCE\_ID;

**Q 5) Write a query to display monthly sales and unit sold in different region.**

SELECT

MONTH\_ID,

STATE\_PROVINCE\_ID,

SUM(SALES) AS Monthly\_Sales,

SUM(UNITS) AS Monthly\_Units\_Sold

FROM AV.SALES\_FACT

GROUP BY MONTH\_ID,

STATE\_PROVINCE\_ID;

**Q 6) Write a query to display seasonally sales in different providence**

SELECT

Season,

STATE\_PROVINCE\_ID,

SUM(SALES) AS Seasonal\_Sales

FROM (

SELECT

CASE

WHEN MONTH\_ID IN (&#39;Dec&#39;, &#39;Jan&#39;, &#39;Feb&#39;) THEN &#39;Winter&#39;

WHEN MONTH\_ID IN (&#39;Mar&#39;, &#39;Apr&#39;, &#39;May&#39;) THEN &#39;Spring&#39;

WHEN MONTH\_ID IN (&#39;Jun&#39;, &#39;Jul&#39;, &#39;Aug&#39;) THEN &#39;Summer&#39;

WHEN MONTH\_ID IN (&#39;Sep&#39;, &#39;Oct&#39;, &#39;Nov&#39;) THEN &#39;Autumn&#39;

END AS Season,

STATE\_PROVINCE\_ID,

SALES

FROM

AV.SALES\_FACT

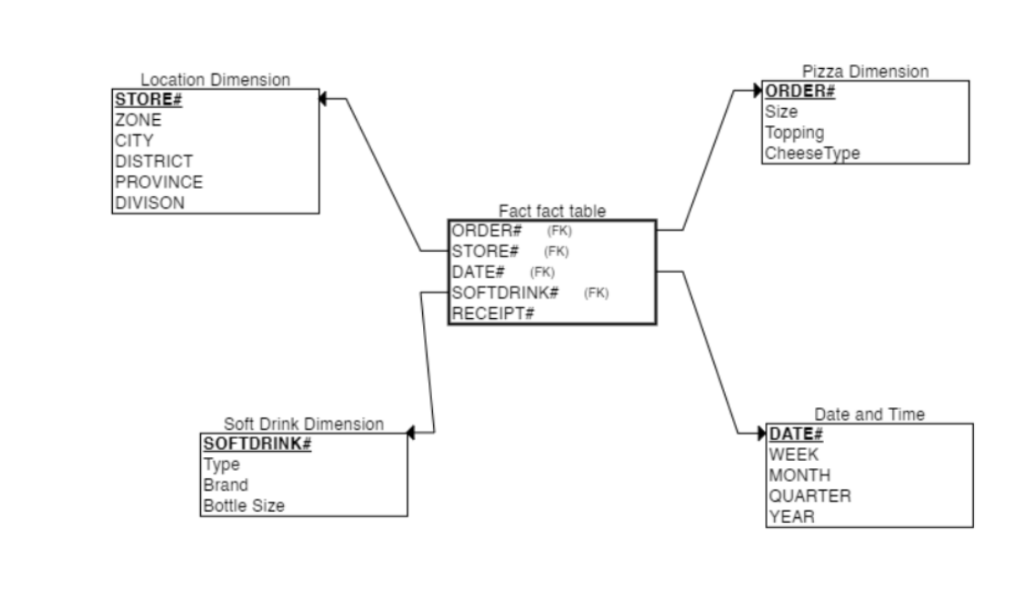
) AS SeasonalData

GROUP BY

Season,

STATE\_PROVINCE\_ID;

**LAB 4 - ERD Plus and Star Schema**



**Generated Query**

CREATE TABLE Location\_Dimension

(

STORE# INT NOT NULL,

CITY CHAR NOT NULL,

DISTRICT CHAR NOT NULL,

PROVINCE CHAR NOT NULL,

DIVISON INT NOT NULL,

PRIMARY KEY (STORE#)

);

CREATE TABLE Pizza\_Dimension

(

ORDER# INT NOT NULL,

Size CHAR NOT NULL,

Topping CHAR NOT NULL,

CheeseType CHAR NOT NULL,

PRIMARY KEY (ORDER#)

);

CREATE TABLE Date\_and\_Time

(

DATE# DATE NOT NULL,

WEEK CHAR NOT NULL,

MONTH CHAR NOT NULL,

QUARTER CHAR NOT NULL,

YEAR CHAR NOT NULL,

PRIMARY KEY (DATE#)

);

CREATE TABLE Soft\_Drink\_Dimension

(

SOFTDRINK# INT NOT NULL,

Type CHAR NOT NULL,

Brand CHAR NOT NULL,

Bottle\_Size CHAR NOT NULL,

PRIMARY KEY (SOFTDRINK#)

);

CREATE TABLE Fact

(

RECEIPT# INT NOT NULL,

ORDER# INT NOT NULL,

STORE# INT NOT NULL,

DATE# DATE NOT NULL,

SOFTDRINK# INT NOT NULL,

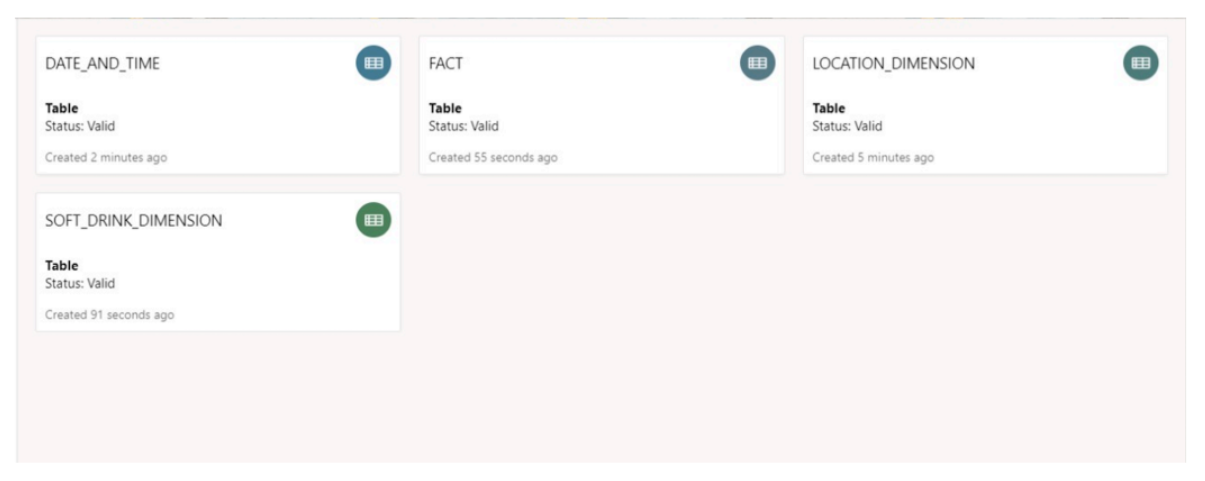
FOREIGN KEY (STORE#) REFERENCES Location\_Dimension(STORE#),

FOREIGN KEY (DATE#) REFERENCES Date\_and\_Time(DATE#),

FOREIGN KEY (SOFTDRINK#) REFERENCES Soft\_Drink\_Dimension(SOFTDRINK#)

);

**Schema**

****

**Practical No 5**

**Q 1. Create a query to perform roll up operation using av.sales\_fact table.**

**(Note:-examine all data)**

A.

SELECT

month\_id,

category\_id,

state\_province\_id,

SUM(sales) AS total\_sales,

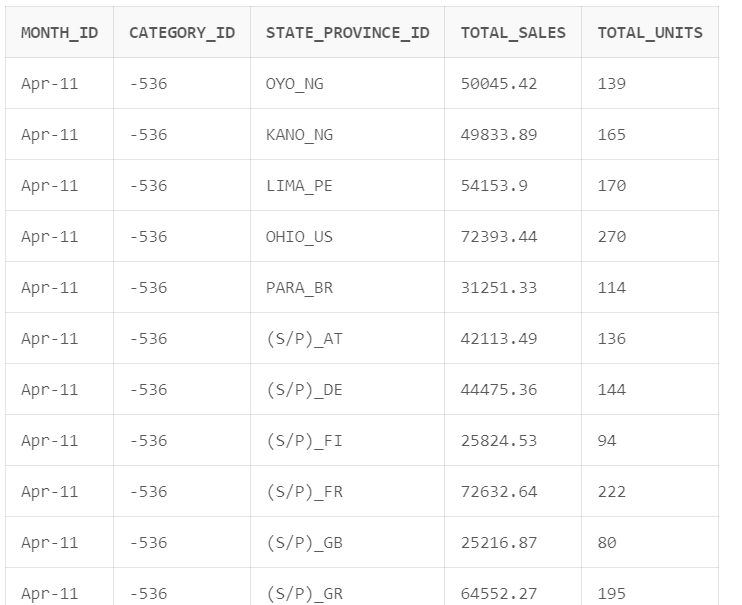
SUM(units) AS total\_units

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)**

A.

SELECT

month\_id,

category\_id,

state\_province\_id,

SUM(sales) AS total\_sales,

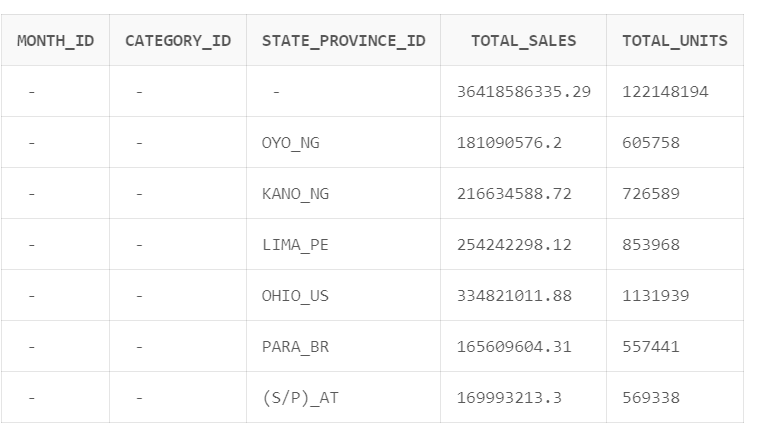
SUM(units) AS total\_units

FROM

av.sales\_fact

GROUP BY

CUBE(month\_id, category\_id, state\_province\_id);



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

A. Roll-up, cube, and pivot functions in SQL serve distinct purposes for data summarization. Roll-up is utilized to generate subtotals and a grand total for a specified set of columns, providing a hierarchical view. Cube, similar to roll-up, extends this by calculating subtotals and a grand total for all possible combinations of specified columns, offering a multidimensional perspective. Pivot, however, is employed to transform row data into a columnar format, enhancing readability, and is often used with aggregate functions for summarization. While roll-up and cube focus on hierarchical and dimensional analysis, providing subtotals at various levels, pivot primarily deals with restructuring data for better presentation. Each function is valuable in specific contexts, catering to diverse analytical needs.

**Q 4. Perform pivot operations uing AV schema.**

A.

SELECT

month\_id,

SUM(units) AS total\_units

FROM

av.sales\_fact

GROUP BY month\_id

ORDER BY month\_id;

|  |  |
| --- | --- |
| MONTH\_ID | TOTAL\_UNITS |
| Apr-11 | 2002303 |
| Apr-12 | 1998655 |
| Apr-13 | 2002657 |
| Apr-14 | 2003542 |
| Apr-15 | 2001030 |
| Aug-11 | 2042569 |
| Aug-12 | 2043836 |
| Aug-13 | 2043454 |
| Aug-14 | 2043335 |
| Aug-15 | 2043437 |
| Dec-11 | 2100209 |
| Dec-12 | 2100892 |
| Dec-13 | 2101709 |
| Dec-14 | 2101171 |
| Dec-15 | 2101992 |
| Feb-11 | 1866893 |
| Feb-12 | 1804964 |
| Feb-13 | 1804810 |
| Feb-14 | 1803959 |
| Feb-15 | 1869027 |
| Jan-11 | 1993656 |
| Jan-12 | 1990711 |
| Jan-13 | 1990777 |
| Jan-14 | 1991853 |
| Jan-15 | 1993658 |
| Jul-11 | 2077429 |
| Jul-12 | 2078935 |
| Jul-13 | 2080226 |
| Jul-14 | 2076820 |
| Jul-15 | 2078469 |
| Jun-11 | 2062078 |
| Jun-12 | 2064009 |
| Jun-13 | 2062598 |
| Jun-14 | 2061538 |
| Jun-15 | 2064798 |
| Mar-11 | 2031777 |
| Mar-12 | 2030877 |
| Mar-13 | 2030850 |
| Mar-14 | 2032628 |
| Mar-15 | 2032913 |
| May-11 | 2105096 |
| May-12 | 2105387 |
| May-13 | 2107216 |
| May-14 | 2106202 |
| May-15 | 2106537 |
| Nov-11 | 2044448 |
| Nov-12 | 2046881 |
| Nov-13 | 2044461 |
| Nov-14 | 2045638 |
| Nov-15 | 2046639 |

**Q 5. Create pivot operation using SH schema.**

A.

SELECT

prod\_id,

SUM(quantity\_sold) AS total\_items\_sold

FROM

sh.profits

GROUP BY prod\_id

ORDER BY prod\_id;

|  |  |
| --- | --- |
| PROD\_ID | TOTAL\_ITEMS\_SOLD |
| 13 | 6002 |
| 14 | 6010 |
| 15 | 5764 |
| 16 | 6928 |
| 17 | 6160 |
| 18 | 9589 |
| 19 | 10430 |
| 20 | 10903 |
| 21 | 4913 |
| 22 | 3441 |
| 23 | 19642 |
| 24 | 20948 |
| 25 | 19557 |
| 26 | 15651 |
| 27 | 11979 |
| 28 | 16796 |
| 29 | 7197 |
| 30 | 28979 |
| 31 | 23108 |
| 32 | 11253 |

**Q 6. Create a view using cube operation.**

A.

CREATE VIEW cube\_view AS

SELECT

month\_id,

category\_id,

SUM(units) AS total\_units

FROM

av.sales\_fact

GROUP BY

CUBE(month\_id, category\_id);

**Practical No: - 6**

*Theory: refers to what has been seen in class (using HR schema).*

**Q1. Create a materialized view that store last name and salary of employees earning more than $12,000.**

A:-

CREATE MATERIALIZED VIEW mv\_high\_salary\_employees

BUILD IMMEDIATE

REFRESH COMPLETE

AS

SELECT last\_name, salary

FROM hr.employees

WHERE salary > 12000;

**Q2. Create a materialized view called EMPLOYEES\_VU on the employee numbers, employee names, and department numbers from the EMPLOYEES table**

A:-

CREATE MATERIALIZED VIEW EMPLOYEES\_VU

BUILD IMMEDIATE

REFRESH COMPLETE

AS

SELECT employee\_id, first\_name || ' ' || last\_name AS employee\_name, department\_id

FROM hr.employees;

**Q3. Insert one employee in to employees table than Display the contents of the EMPLOYEES\_VU view also what is the difference between employees table and materialized view.**

A:-

-- Inserting a new employee

INSERT INTO hr.employees (employee\_id, first\_name, last\_name, department\_id)

VALUES (1001, 'John', 'Doe', 30);

-- Displaying contents of the materialized view

SELECT \* FROM EMPLOYEES\_VU;

A table is a permanent storage structure that always reflects the latest data, a materialized view is a precomputed, stored result set that can be used to improve query performance at the cost of potentially having slightly stale data.

**Q4.Write a query to manually refresh EMPLOYEES\_VU.**

A:-

EXEC DBMS\_MVIEW.REFRESH('EMPLOYEES\_VU', 'C');

Q5.Create a materialize view that store the number of people with the same job.

A:-

CREATE MATERIALIZED VIEW job\_count\_mv

BUILD IMMEDIATE

REFRESH COMPLETE

AS

SELECT job\_id, COUNT(\*) AS num\_people

FROM hr.employees

GROUP BY job\_id;

**Q5.Modify Lab 4 3.sql create a materialized view that Oracle refresh automatically.**

A:-

CREATE MATERIALIZED VIEW job\_count\_mv

BUILD IMMEDIATE

REFRESH FAST ON COMMIT

AS

SELECT job\_id, COUNT(\*) AS num\_people

FROM hr.employees

GROUP BY job\_id;

**Q6.Create a materialize view that that store the difference between the highest a nd lowest salaries. Label the column DIFFERENCE.**

A:-

CREATE MATERIALIZED VIEW salary\_difference\_mv

BUILD IMMEDIATE

REFRESH COMPLETE

AS

SELECT (MAX(salary) - MIN(salary)) AS DIFFERENCE

FROM hr.employees;

**LAB 7-8**

Theory refers to what has been discussed in class (using Av schema).

**Q 1. Create a query to display geography, product and time dimension data.**

 (Note:-examine all data)

A.

SELECT \* FROM av.geography\_dim;

SELECT \* FROM av.product\_dim;

SELECT \* FROM av.time\_dim;

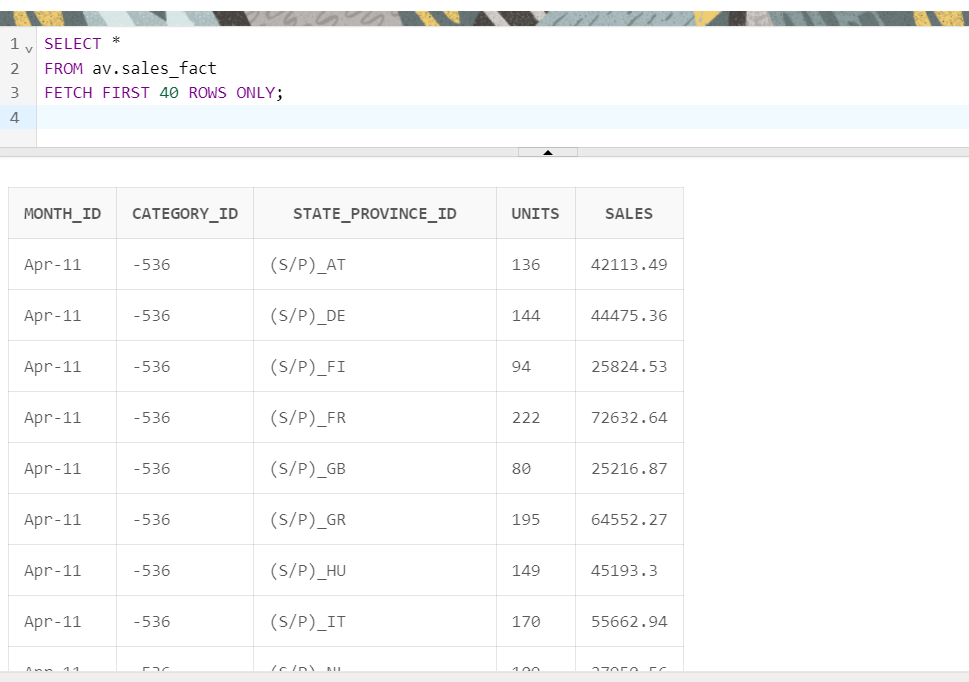
**Q 2. Select 40 rows form sale fact table. (Note:-examine data)**

A.

SELECT \*

FROM av.sales\_fact

FETCH FIRST 40 ROWS ONLY;



**Q 3. Why we use analytical view and what is the difference between simple analytical view and materialize analytical view.**

A.

Analytical views are used for complex analysis and reporting in a data warehouse environment. They provide a way to pre-aggregate and store results of complex queries for faster retrieval.

* **Simple Analytical View:**
  + Computed on-the-fly when queried.
  + Suitable for scenarios where real-time data is crucial.
  + May have a performance impact on large datasets.
* **Materialized Analytical View:**
  + Precomputed and stored, improving query performance.
  + Better for scenarios where query performance is a priority.
  + Requires periodic refresh to keep data up-to-date.

**Q 4. Create attribute dimension of time, geography and product dimension.**

A. **Time**:

CREATE OR REPLACE ATTRIBUTE DIMENSION time\_attr\_dim

USING av.time\_dim

ATTRIBUTES

(year\_id,

quarter\_id,

month\_id)

LEVEL MONTH

KEY month\_id

DETERMINES (quarter\_id)

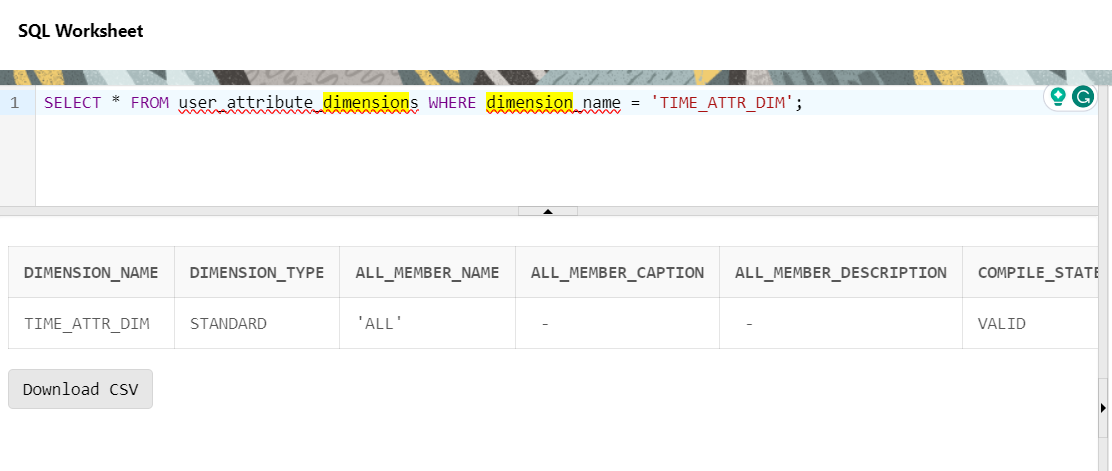
LEVEL QUARTER

KEY quarter\_id

DETERMINES (year\_id)

LEVEL YEAR

KEY year\_id;



**Geography**:

CREATE OR REPLACE ATTRIBUTE DIMENSION geography\_attr\_dim

USING av.geography\_dim

ATTRIBUTES

(region\_id,

country\_id,

state\_province\_id)

LEVEL STATE\_PROVINCE

KEY state\_province\_id

DETERMINES (country\_id)

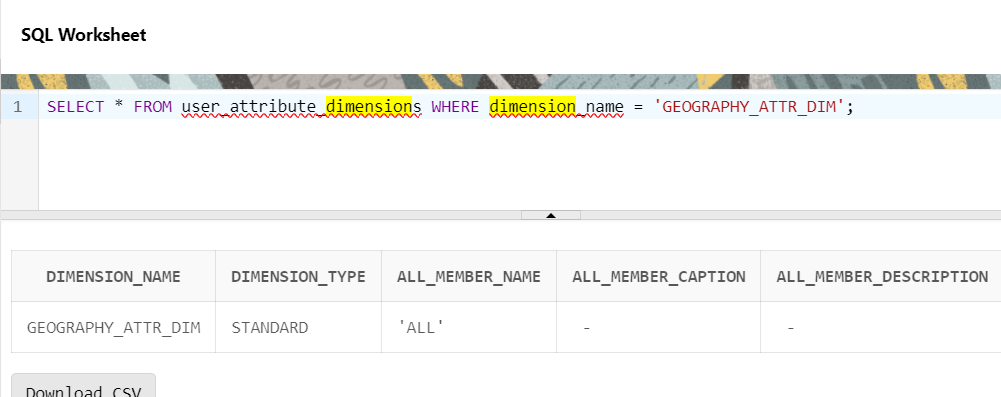
LEVEL COUNTRY

KEY country\_id

DETERMINES (region\_id)

LEVEL REGION

KEY region\_id;



**PRODUCT**:

CREATE OR REPLACE ATTRIBUTE DIMENSION product\_attr\_dim

USING av.product\_dim

ATTRIBUTES

(department\_id,

category\_id)

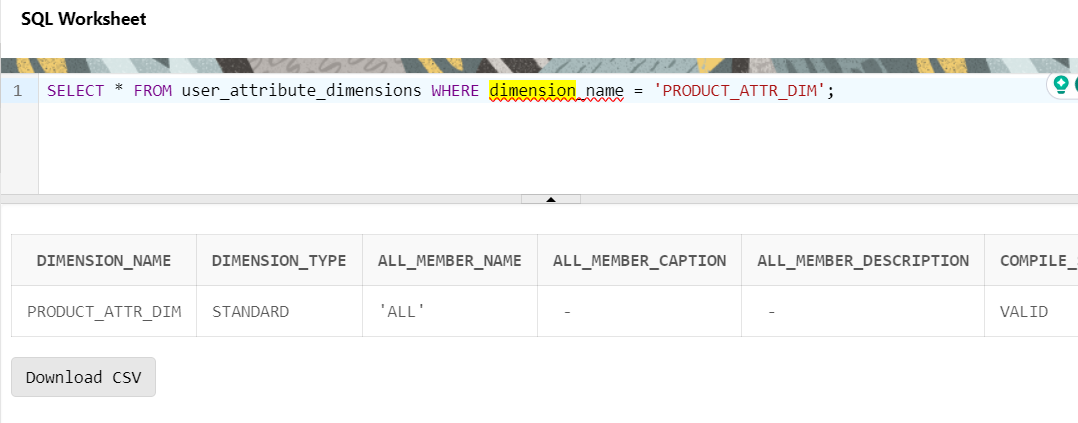
LEVEL CATEGORY

KEY category\_id

DETERMINES (department\_id)

LEVEL DEPARTMENT

KEY department\_id;



**Q 5. Create hierarchy of time, geography and product dimension.**

A.

**TIME HIERARCHY:**

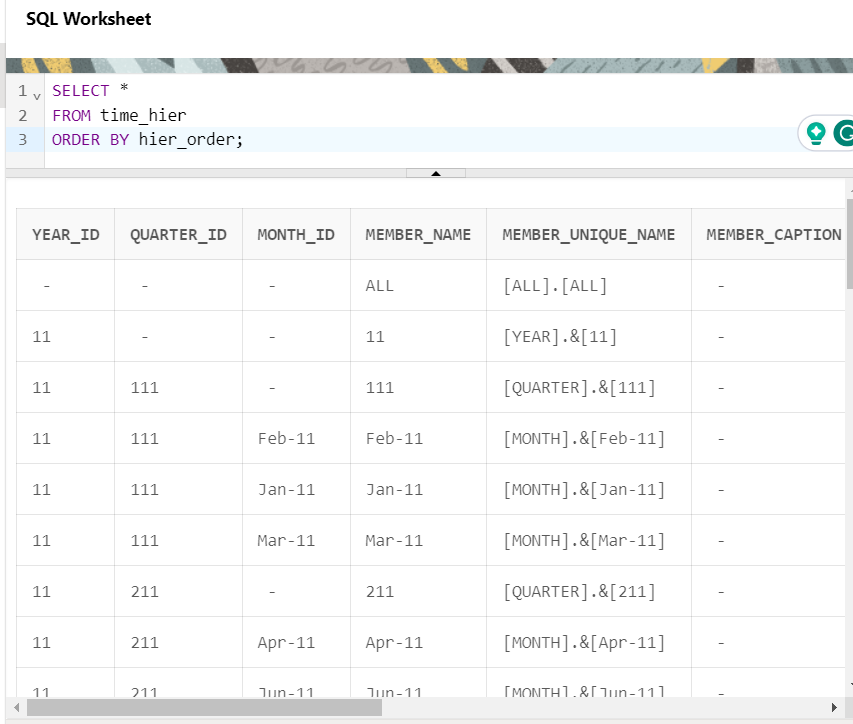
CREATE OR REPLACE HIERARCHY time\_hier

USING time\_attr\_dim

(month CHILD OF

quarter CHILD OF

year);



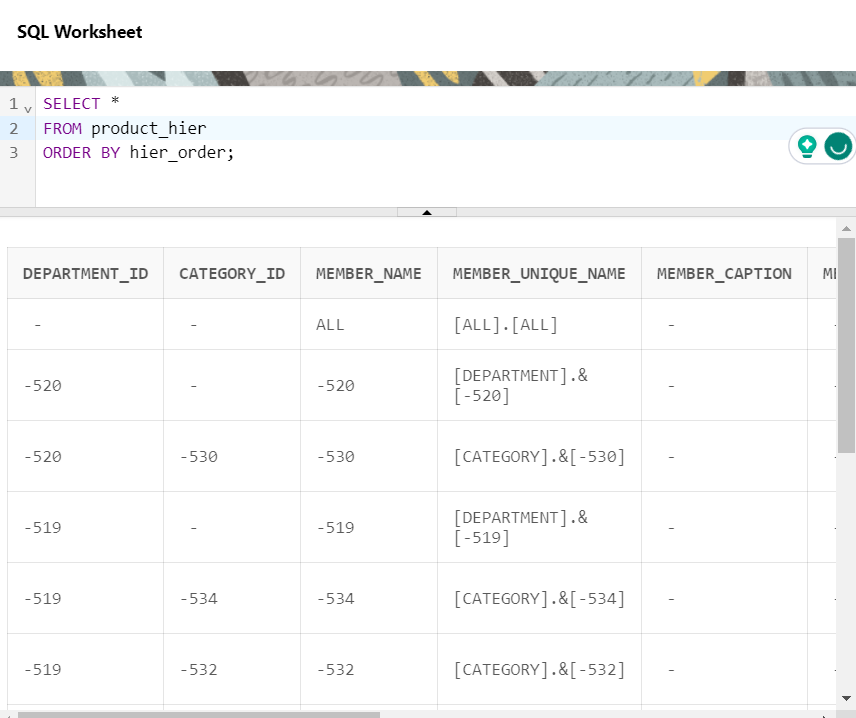
**PRODUCT HIERARCHY:**

CREATE OR REPLACE HIERARCHY product\_hier

USING product\_attr\_dim

(category CHILD OF

department);



**GEOGRAPHY HIERARCHY:**

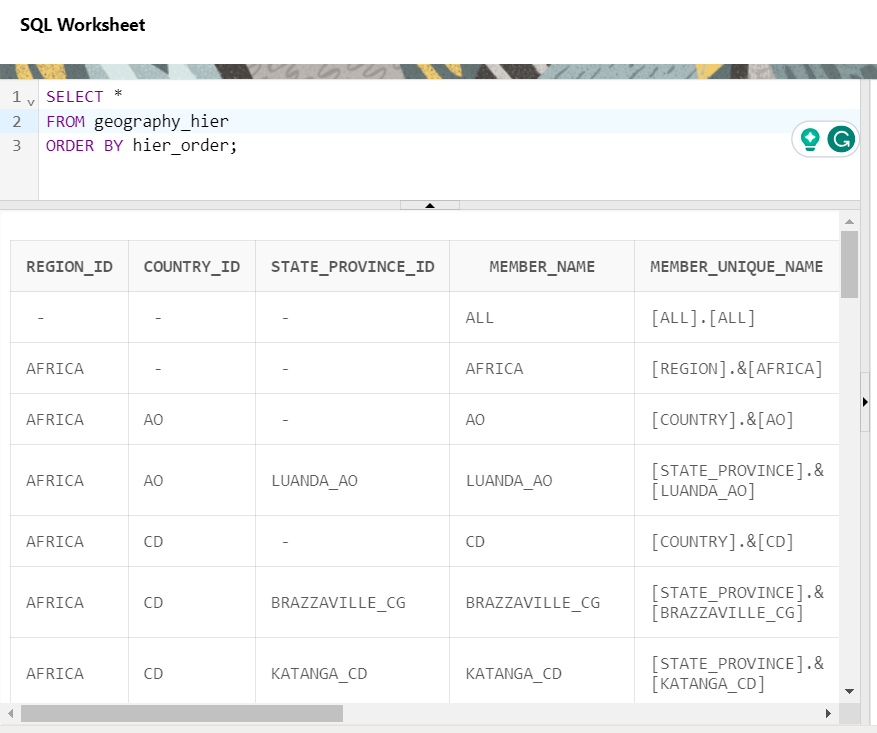
CREATE OR REPLACE HIERARCHY geography\_hier

USING geography\_attr\_dim

(state\_province CHILD OF

country CHILD OF

region);



**Q 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.**

A.

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))

MEASURES

(sales FACT sales,

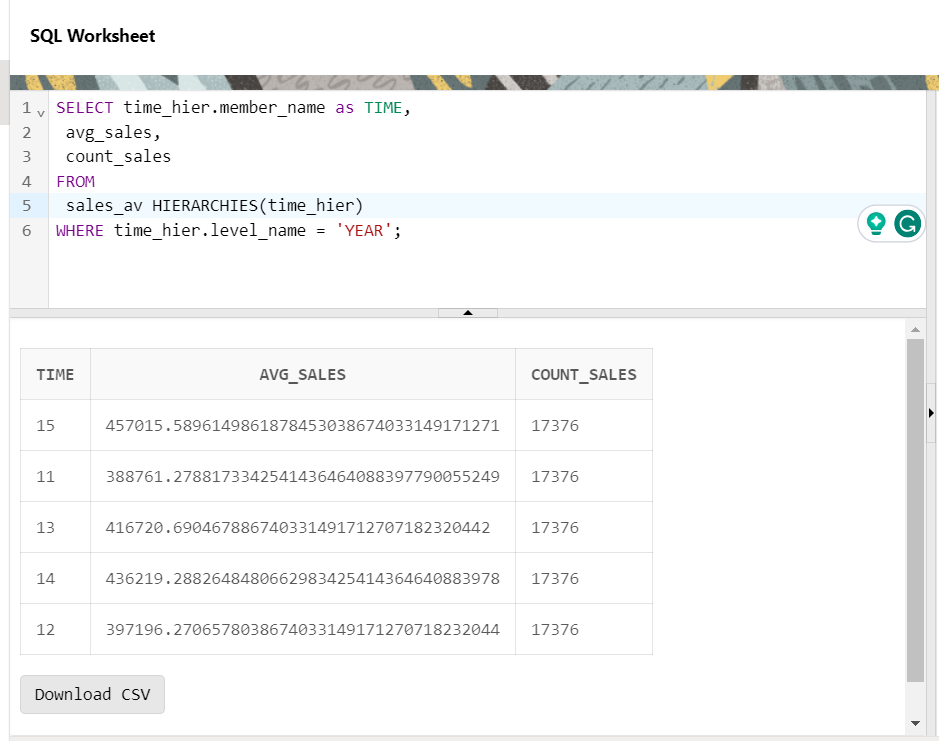
avg\_sales FACT sales AGGREGATE BY AVG,

count\_sales FACT sales AGGREGATE BY COUNT

)

DEFAULT MEASURE SALES

DEFAULT AGGREGATE BY SUM;



**Q 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\_IDl level and CATEGORY\_ID.**

A.

CREATE OR REPLACE ANALYTIC VIEW sales\_av\_min\_max

USING av.sales\_fact

DIMENSION BY

(time\_attr\_dim

KEY month\_id REFERENCES month\_id

HIERARCHIES (

time\_hier DEFAULT))

MEASURES

(sales FACT sales,

count\_sales FACT sales AGGREGATE BY COUNT,

max\_sales FACT sales AGGREGATE BY MAX,

min\_sales FACT sales AGGREGATE BY MIN,

stddev\_sales FACT sales AGGREGATE BY STDDEV,

variance\_sales FACT sales AGGREGATE BY VARIANCE

)

DEFAULT MEASURE SALES

DEFAULT AGGREGATE BY SUM;

SELECT

geography\_attr\_dim.REGION\_ID,

product\_attr\_dim.CATEGORY\_ID,

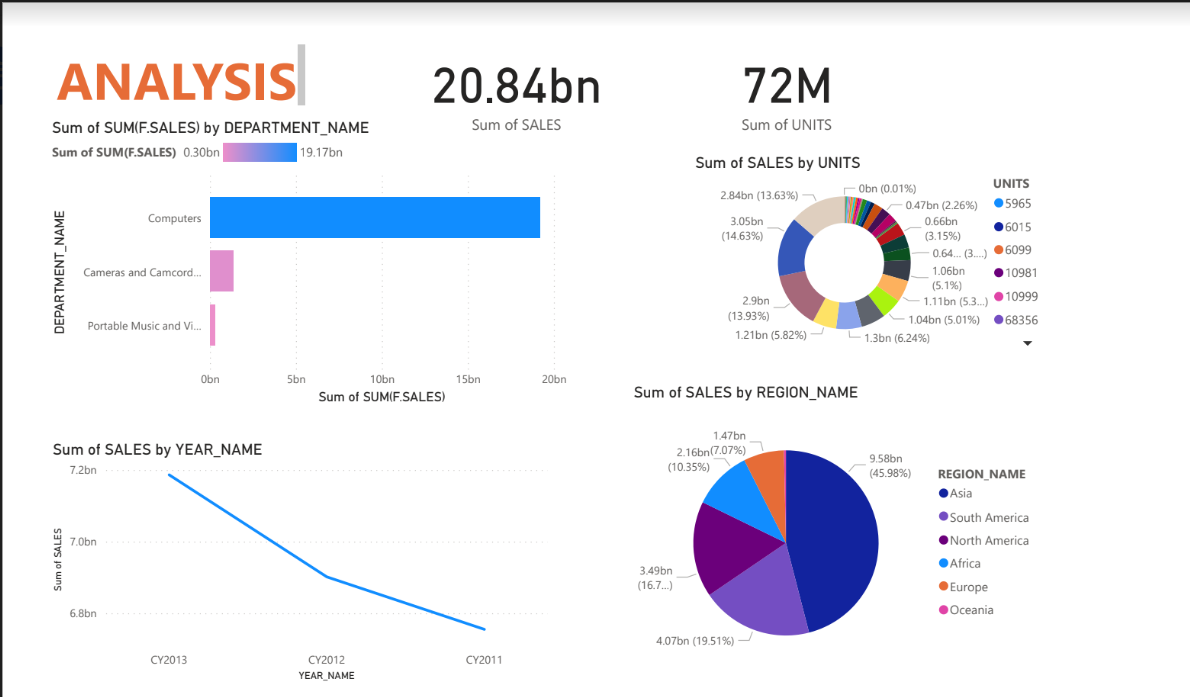
sale\_count,

avg\_sale

FROM

sales\_av\_min\_max;

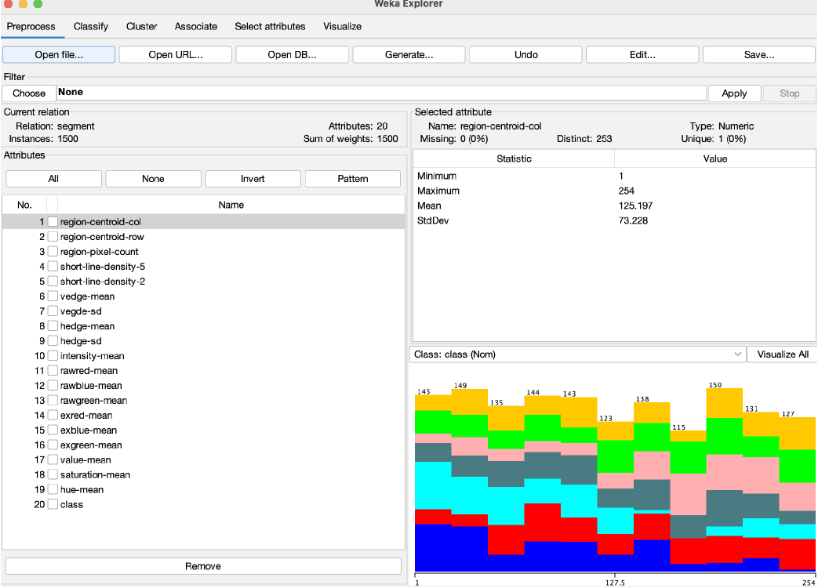
**LAB 9-10**

****

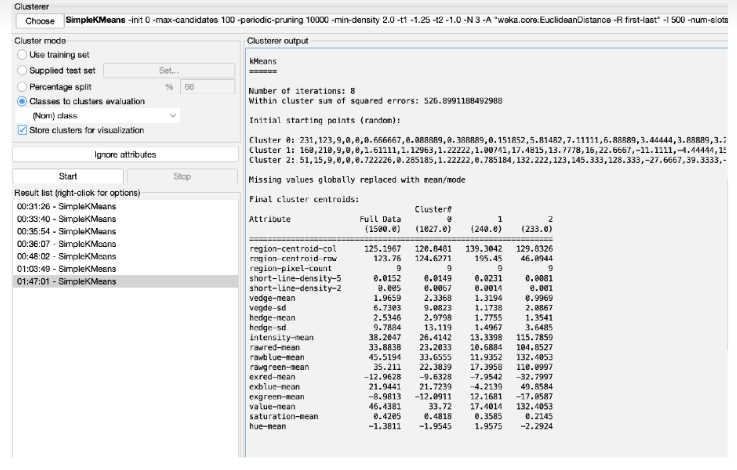
**LAB 11 & 12**

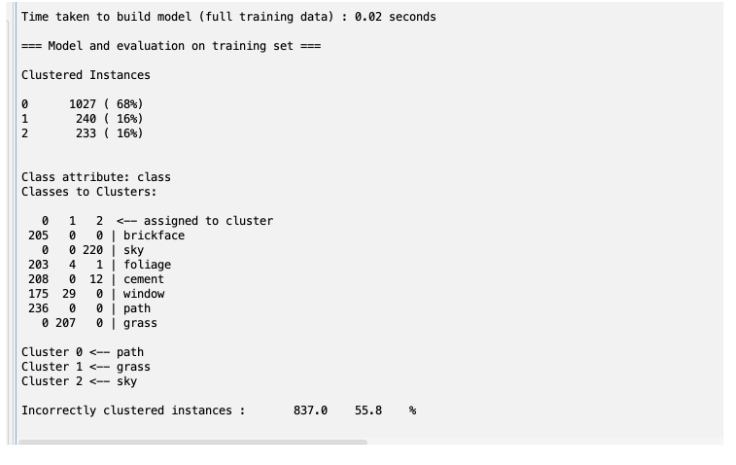
**1) Weather Integer Dataset**

**Selected Attributes:**

****

**Results Clustering ( SimpleKMeans clusterer):**

****

****

**Explanation Clustering:**

The clustering analysis was performed using the SimpleKMeans algorithm in Weka on a dataset

with 1500 instances and 20 attributes. The algorithm converged after 8 iterations, resulting in

three clusters. The within-cluster sum of squared errors was 526.90. The final cluster centroids

provide insights into the characteristics of each cluster across the attributes. The majority of

instances were assigned to Cluster 0 (68%), which represents the &#39;path&#39; class. However, the

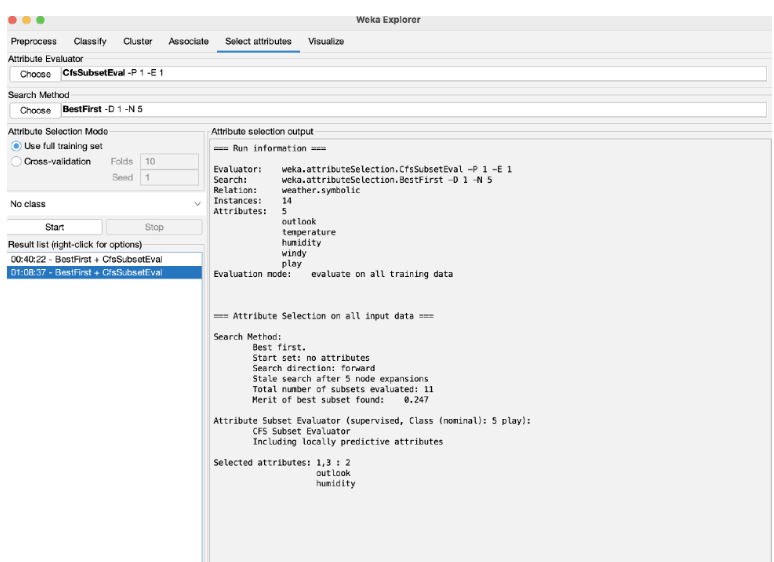
clustering was not entirely accurate, as 55.8% of instances were incorrectly clustered. Notably,

Cluster 1 corresponds to the &#39;grass&#39; class, and Cluster 2 represents the &#39;sky&#39; class,

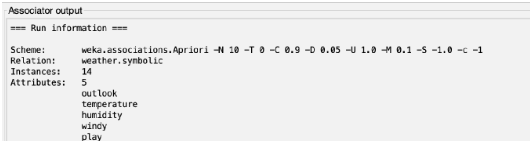
demonstrating the algorithm&#39;s ability to capture meaningful patterns within the data.

**2) Weather Nominal Dataset**

**Selected Attributes:**

****

**Results Association (Apiriori):**

****

**Explanation Association:**

The Apriori association analysis on the symbolic weather dataset revealed several interesting

rules with high confidence. For instance, the rule &quot;outlook=overcast ==&gt; play=yes&quot; exhibited a

confidence of 1, indicating a strong association between overcast conditions and playing.

Similarly, rules like &quot;temperature=cool ==&gt; humidity=normal&quot; and &quot;outlook=sunny play=no ==&gt;

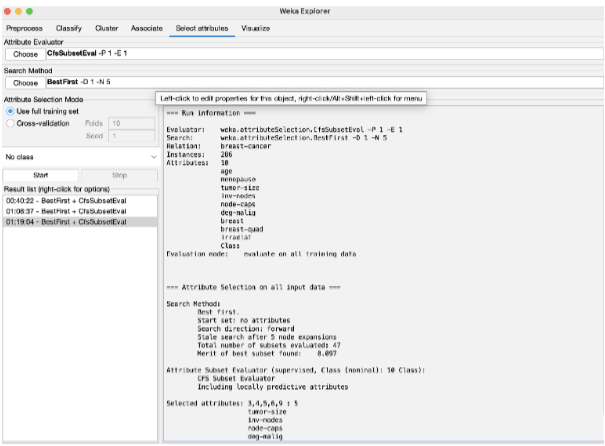
humidity=high&quot; demonstrated confident associations. The generated rules can provide valuable

insights into the relationships between different weather attributes and the likelihood of outdoor

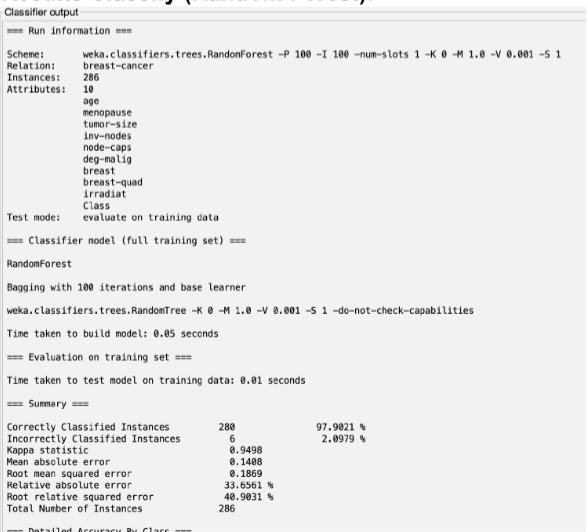
activities, offering practical implications for decision-making based on weather conditions.

**3) Breast Cancer Dataset**

**Selected Attributes:**

****

**Results Classify (Random Forest):**

****

**Explanation Classifier:**

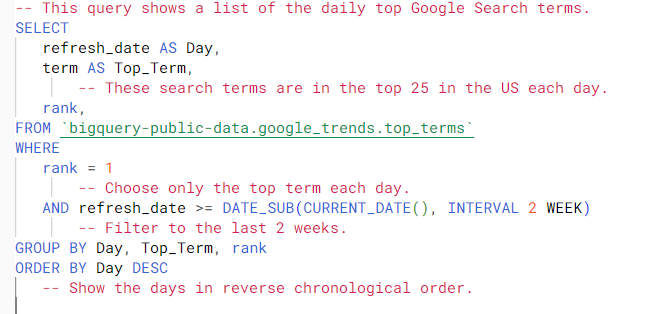
The Random Forest classification on the breast cancer dataset achieved an impressive accuracy of 97.90%. The model correctly classified instances into the recurrence event classes with high precision and recall values for both classes. The detailed accuracy by class shows that the model performed well in distinguishing between the two outcomes, with a weighted average F-measure of 0.979. The confusion matrix indicates that only a small number of instances were misclassified (6 out of 286), contributing to the model&#39;s overall reliability in predicting breast cancer recurrence based on the provided attributes.

**LAB 13**

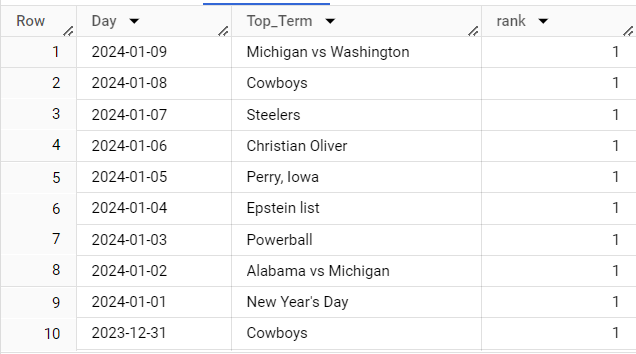
1. **Google Trends Data:** This data represents the top 25 google searched terms on specific days.

**Query:**

From top 25, we are displaying the rank 1 term of each day.

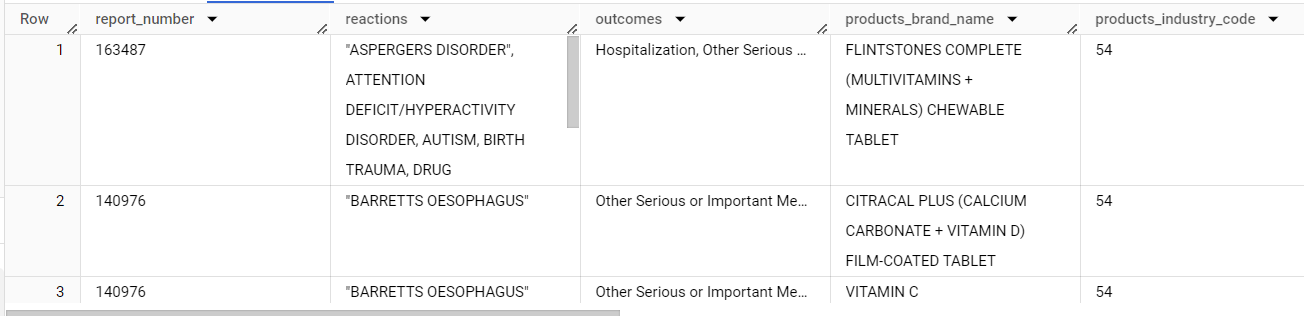


**Output**:



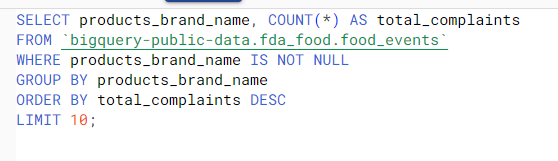
1. **Food Quality Data:**

This dataset contains the details of the consumer who eats the affected food, and the reactions (which shows that what happened to the consumer after eating that food), and the brand name of the food.

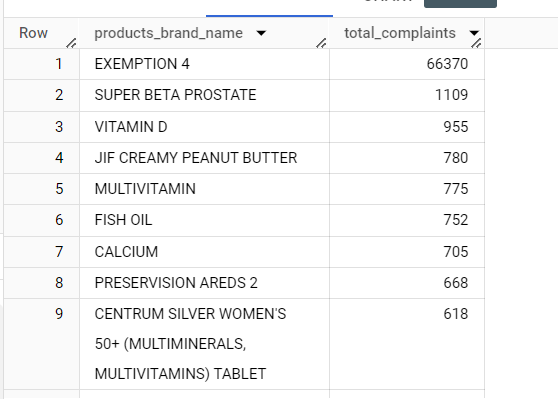


**Query**:

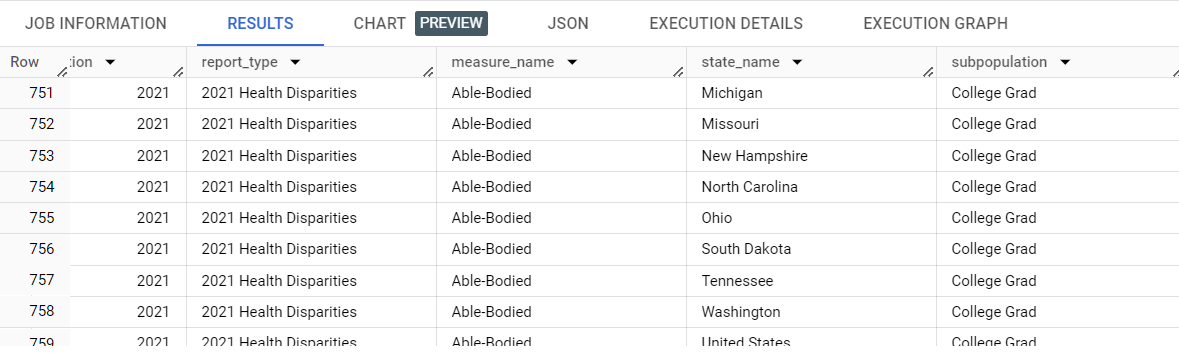
I have tried to show the brand name and their total no. of complaints.



**Output**:

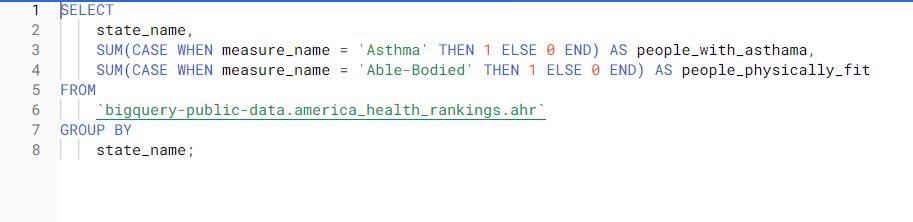


**3. Health data:** This data shows that a specific population of a certain state is physically fit, or having any disease. Like, row 751 shows that the College graduate categories of the state Michigan is physically fit.



**Query**:

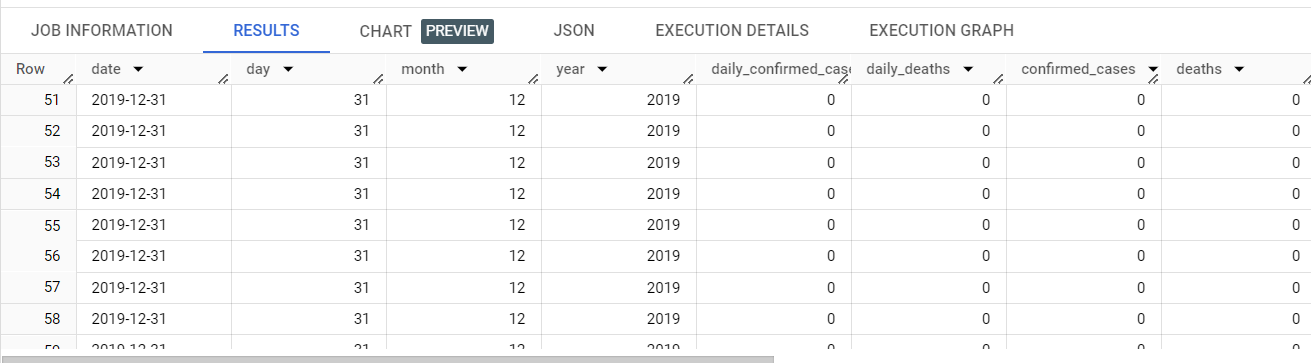
Now I am displaying the states with the no. of people having Asthma or physically fit.



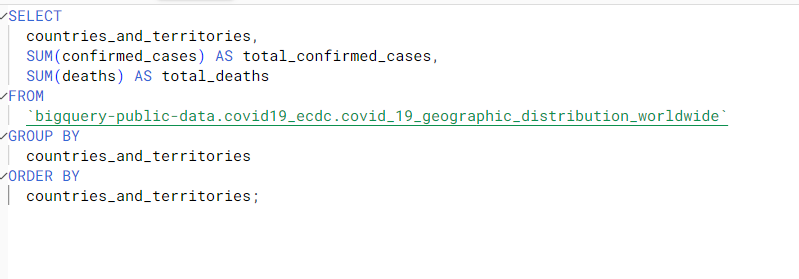
**Output**:



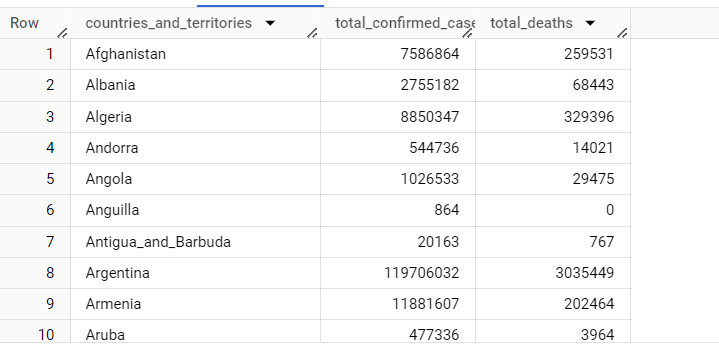
**4. Covid Data:** This data shows the daily covid cases and daily deaths and so many other information. And each country has multiple rows according to dates.



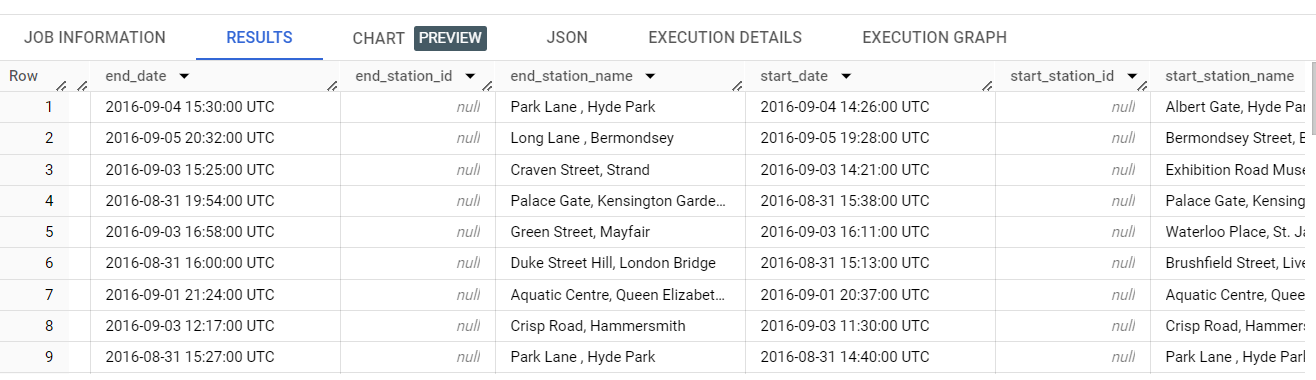
**Query**: I am displaying the overall covid cases and deaths of each country.



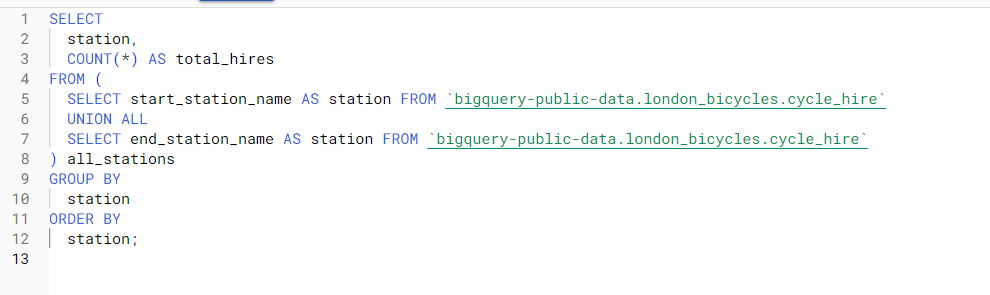
**Output**:



**5. London Cycle Data**: This dataset shows the hiring of London cycles, from where it is hired and also the ending stations, start and end date durations etc.



**Query**: I am displaying that from which station maximum cycle hiring occurs.



**Output:**

