

**CCS341**

**DATA WAREHOUSING LABORATORY**

**L T P C**

**2 0 2 3**

**COURSE OBJECTIVES:**

- To know the details of data warehouse Architecture
- To understand the OLAP Technology
- To understand the partitioning strategy
- To differentiate various schema
- To understand the roles of process manager & system manager

**LIST OF EXPERIMENTS:**

**Note: The lab instructor is expected to design problems based on the topics listed. The Examination shall not be restricted to the sample experiments designed.**

1. Data exploration and integration with WEKA
2. Apply weka tool for data validation
3. Plan the architecture for real time application
4. Write the query for schema definition
5. Design data ware house for real time applications
6. Analyse the dimensional Modeling
7. Case study using OLAP
8. Case study using OTLP
9. Implementation of warehouse testing

**TOTAL: 30 PERIODS**

**COURSE OUTCOMES:**

**Upon completion of the course, the students will be able to**

CO1: Design data warehouse architecture for various Problems

CO2: Apply the OLAP Technology

CO3: Analyse the partitioning strategy

CO4: Critically analyze the differentiation of various schema for given problem

CO5: Frame roles of process manager & system manage

**TEXT BOOKS:**

1. Alex Berson and Stephen J. Smith "Data Warehousing, Data Mining & OLAP", Tata McGraw – Hill Edition, Thirteenth Reprint 2008.
2. Ralph Kimball, "The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling", Third edition, 2013.

## REFERENCES:

1. Paul Raj Ponniah, "Data warehousing fundamentals for IT Professionals", 2012.
2. K.P. Soman, Shyam Diwakar and V. Ajay "Insight into Data mining Theory and Practice", Easter Economy Edition, Prentice Hall of India, 2006.

### List of Experiments

Sl.No	Ex.No.	Date	Title of the Experiments	Page No.	Staff sign
1.	1		Data exploration and integration with WEKA		
2.	2		Apply weka tool for data validation		
3.	3		Plan the architecture for real time application		
4.	4		Write the query for schema definition		
5.	5		Design data ware house for real time applications		
6.	6		Analyse the dimensional Modeling		
7.	7		Case study using OLAP		
8.	8		Case study using OTLP		
9.	9		Implementation of warehouse testing.		

EX.NO.:1

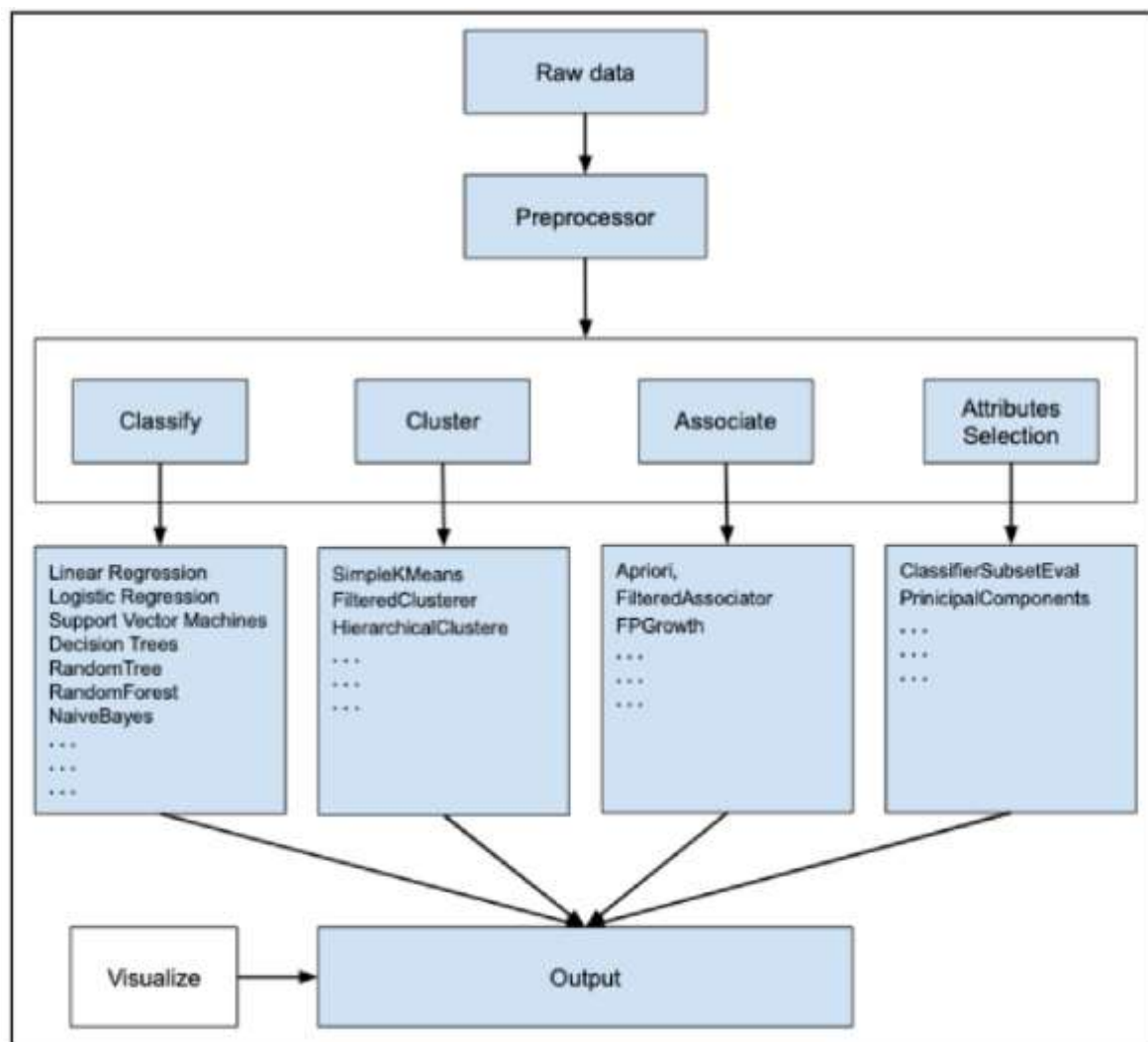
## DATA EXPLORATION AND INTEGRATION WITH WEKA

### AIM:

To exploring the data and performing integration with weka

### PROCEDURE:

WEKA - an open-source software provides tools for data preprocessing, implementation of several Machine Learning algorithms, and visualization tools so that you can develop machine learning techniques and apply them to real-world data mining problems. What WEKA offers is summarized in the following diagram:



## **WEKA Installation**

To install WEKA on your machine, visit WEKA's official website and download the installation file. WEKA supports installation on Windows, Mac OS X and Linux. You just need to follow the instructions on this page to install WEKA for your OS.

The WEKA GUI Chooser application will start and you would see the following screen

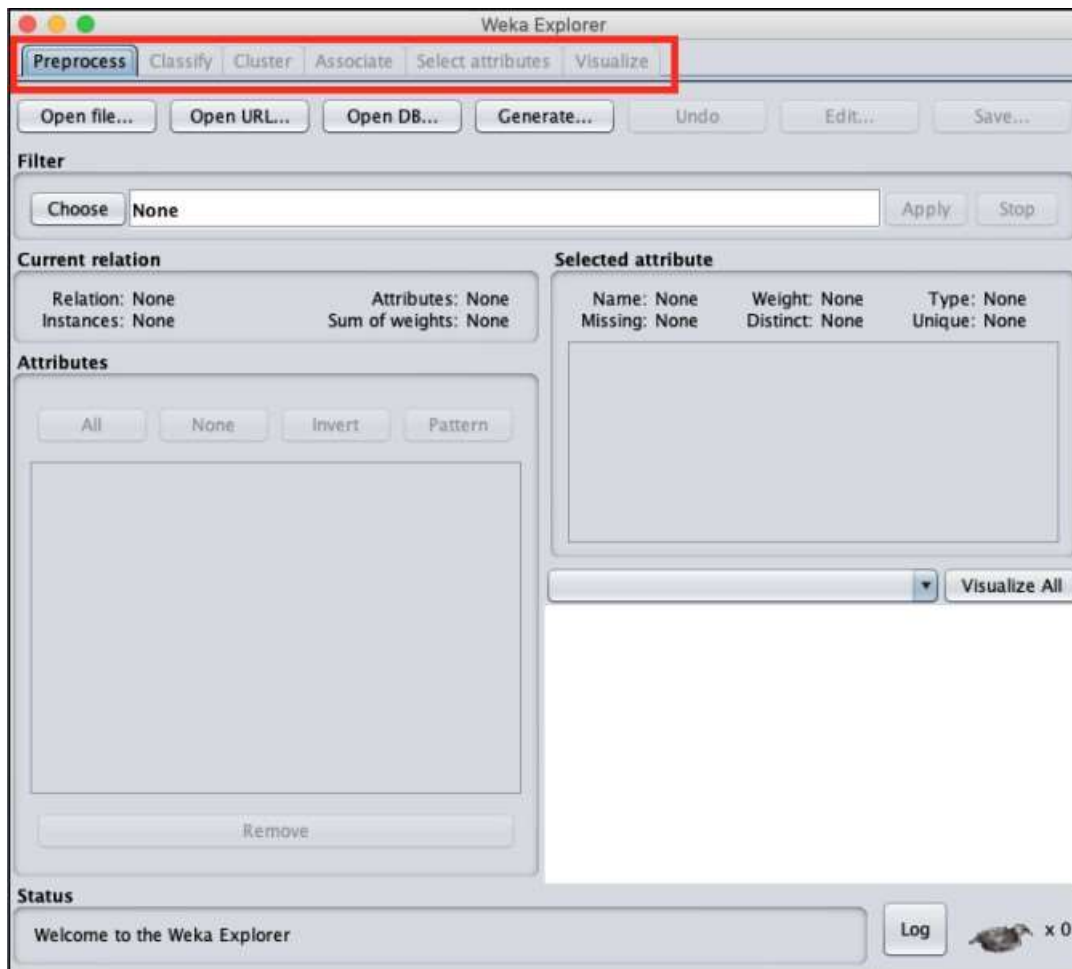


The GUI Chooser application allows you to run five different types of applications as listed here:

- Explorer
- Experimenter
- KnowledgeFlow
- Workbench
- Simple CLI

### **WEKA – Launching Explorer**

let us look into various functionalities that the explorer provides for working with big data. When you click on the Explorer button in the Applications selector, it opens the following screen:



On the top, you will see several tabs as listed here:

- Preprocess
- Classify
- Cluster
- Associate
- Select Attributes
- Visualize

Under these tabs, there are several pre-implemented machine learning algorithms. Let us look into each of them in detail now.

### **Preprocess Tab**

Initially as you open the explorer, only the Preprocess tab is enabled. The first step in machine learning is to preprocess the data. Thus, in the Preprocess option, you will select the data file, process it and make it fit for applying the various machine learning algorithms.

### **Classify Tab**

The Classify tab provides you several machine learning algorithms for the classification of your data. To list a few, you may apply algorithms such as Linear Regression, Logistic Regression, Support Vector Machines, Decision Trees, RandomTree, RandomForest, NaiveBayes, and so on. The list is very exhaustive and provides both supervised and unsupervised machine learning algorithms.

## Cluster Tab

Under the Cluster tab, there are several clustering algorithms provided - such as SimpleKMeans, FilteredClusterer, HierarchicalClusterer, and so on.

## Associate Tab

Under the Associate tab, you would find Apriori, FilteredAssociator and FPGrowth.

## Select Attributes Tab

Select Attributes allows you feature selections based on several algorithms such as ClassifierSubsetEval, PrincipalComponents, etc.

## Visualize Tab

Lastly, the Visualize option allows you to visualize your processed data for analysis. As you noticed, WEKA provides several ready-to-use algorithms for testing and building your machine learning applications. To use WEKA effectively, you must have a sound knowledge of these algorithms, how they work, which one to choose under what circumstances, what to look for in their processed output, and so on. In short, you must have a solid foundation in machine learning to use WEKA effectively in building your apps.

## Loading Data

The data can be loaded from the following sources:

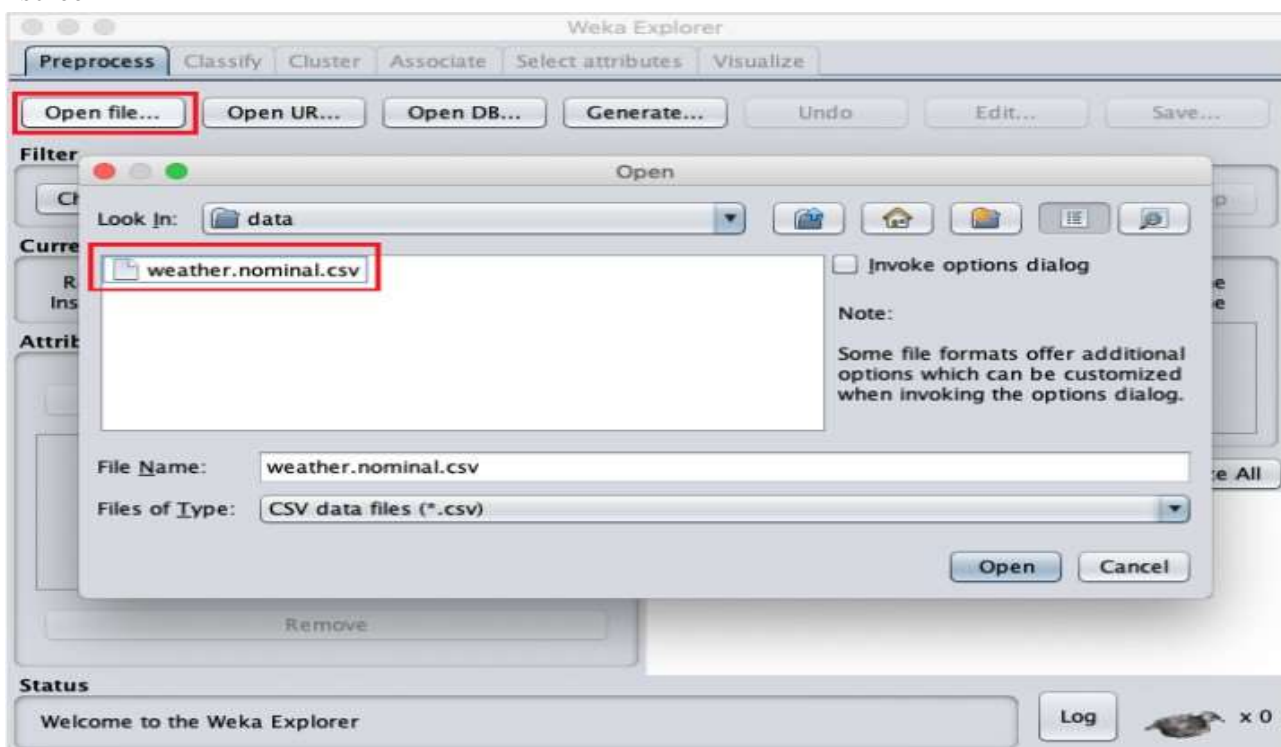
- Local file system
- Web
- Database

### Loading Data from Local File System

There are three buttons

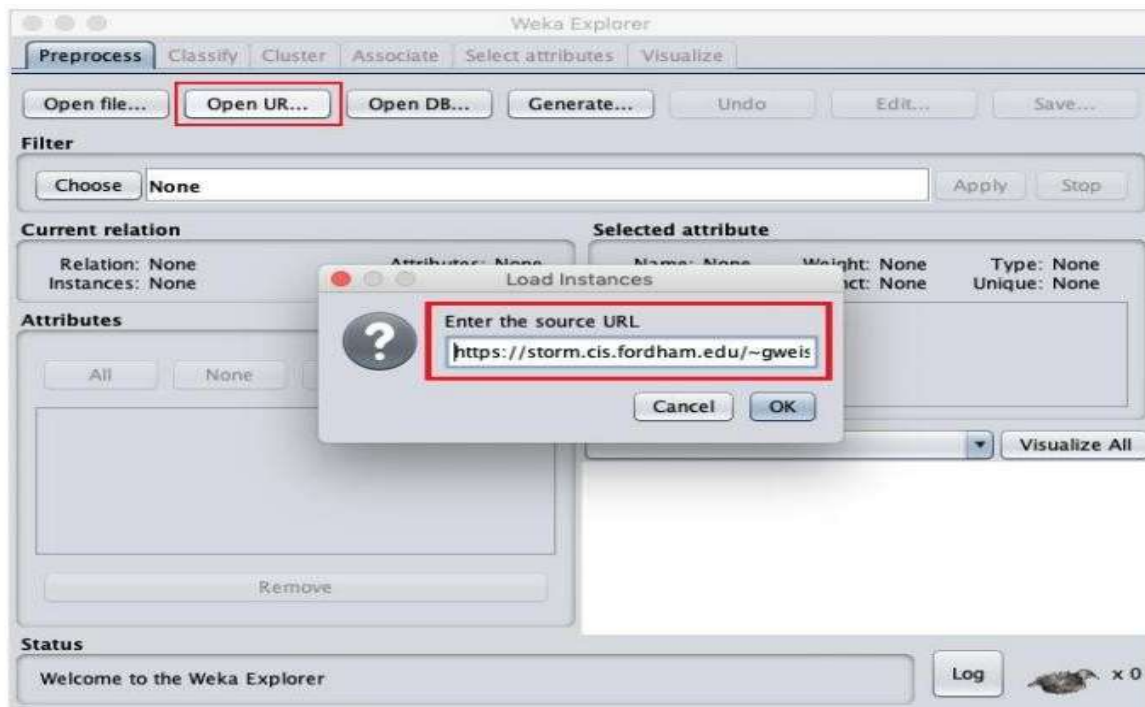
- Open file
- Open URL
- Open DB

Click on the Open file ... button. A directory navigator window opens as shown in the following screen



## Loading Data from Web

Once you click on the Open URL button, you can see a window as follows:



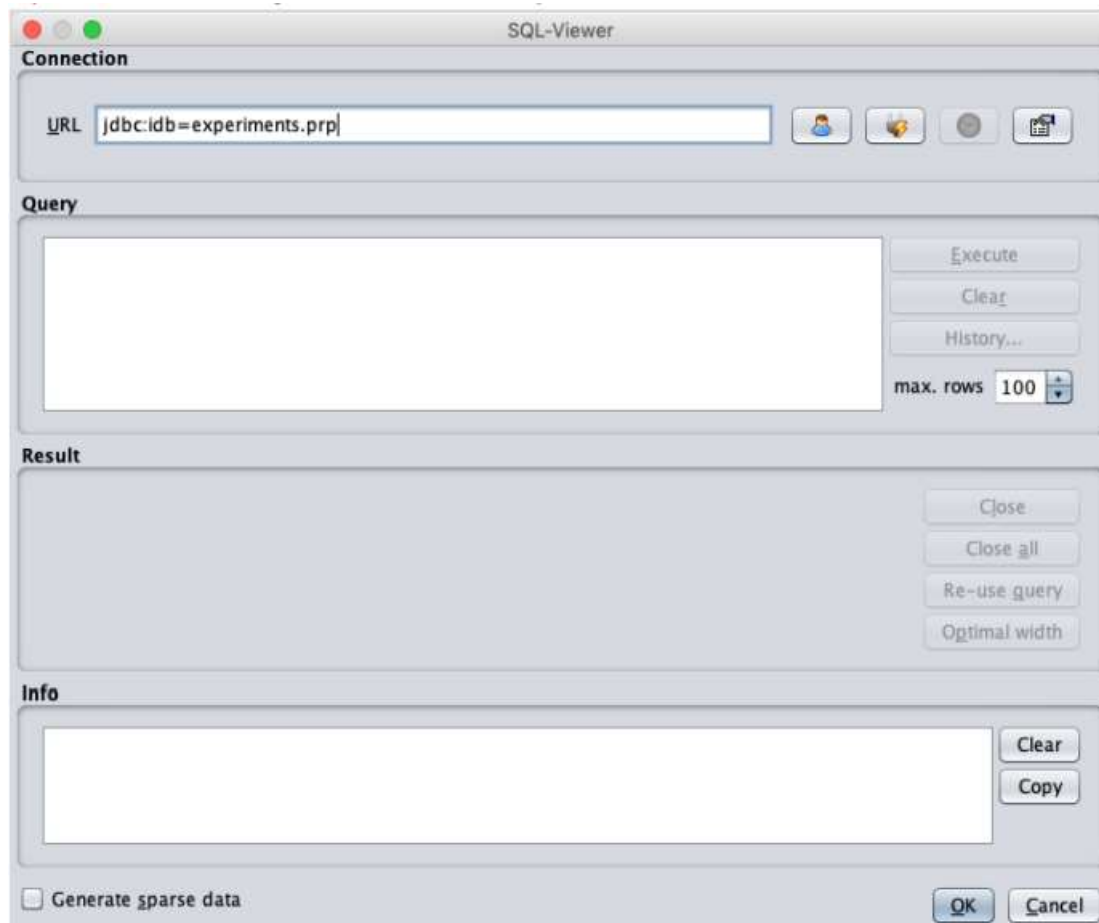
We will open the file from a public URL Type the following URL in the popup box:

<https://storm.cis.fordham.edu/~gweis/data-mining/weka-data/weather.nominal.arff>

You may specify any other URL where your data is stored. The Explorer will load the data from the remote site into its environment.

## Loading Data from DB

Once you click on the Open DB button, you can see a window as follows:



Set the connection string to your database, set up the query for data selection, process the query and load the selected records in WEKA.

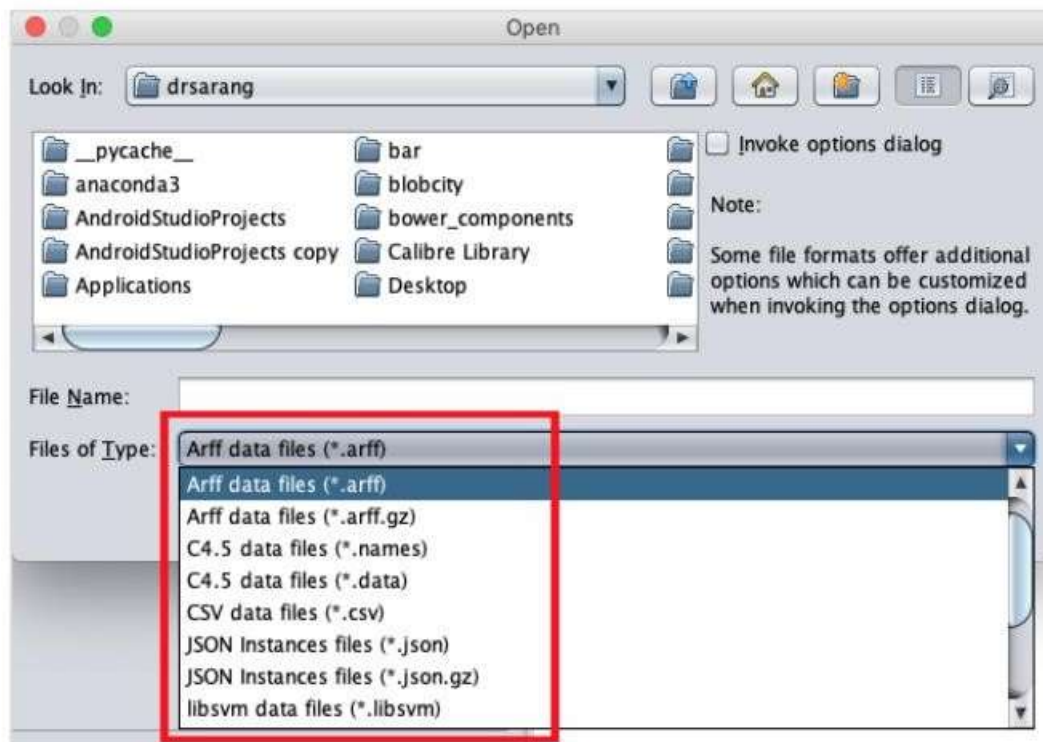
### **WEKA File Formats**

WEKA supports a large number of file formats for the data. Here is the complete list:

- arff
- arff.gz
- bsi
- csv
- dat
- data
- json
- json.gz
- libsvm
- m
- names
- xrf
- xrf.gz

The types of files that it supports are listed in the drop-down list box at the bottom of the screen. This is shown in the screenshot given below.





As you would notice it supports several formats including CSV and JSON. The default file type is Arff.

### Arff Format

An Arff file contains two sections - header and data.

- The header describes the attribute types.
- The data section contains a comma separated list of data.

As an example for Arff format, the Weather data file loaded from the WEKA sample databases is shown below:

From the screenshot, you can infer the following points:

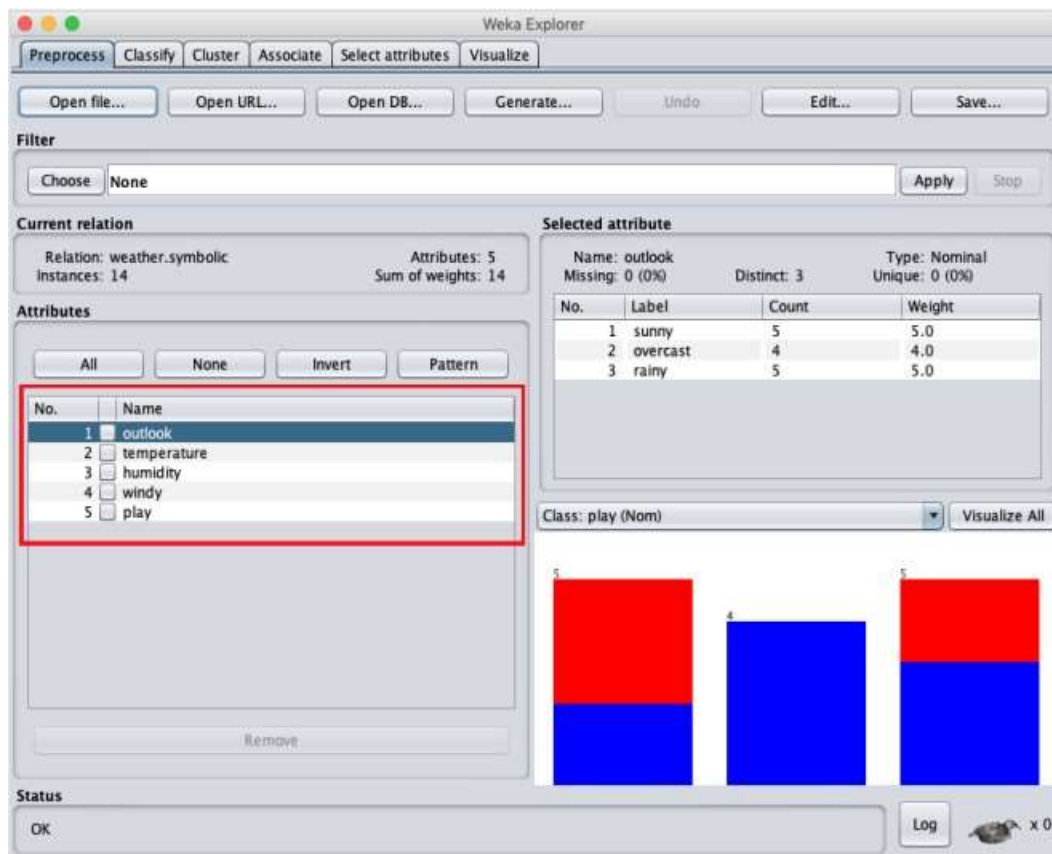
- The @relation tag defines the name of the database.
- The @attribute tag defines the attributes.
- The @data tag starts the list of data rows each containing the comma separated fields.
- The attributes can take nominal values as in the case of outlook shown here:  
@attribute outlook (sunny, overcast, rainy)
- The attributes can take real values as in this case:  
@attribute temperature real
- You can also set a Target or a Class variable called play as shown here:  
@attribute play (yes, no)
- The Target assumes two nominal values yes or no.

### Understanding Data

Let us first look at the highlighted Current relation sub window. It shows the name of the database that is currently loaded. You can infer two points from this sub window:

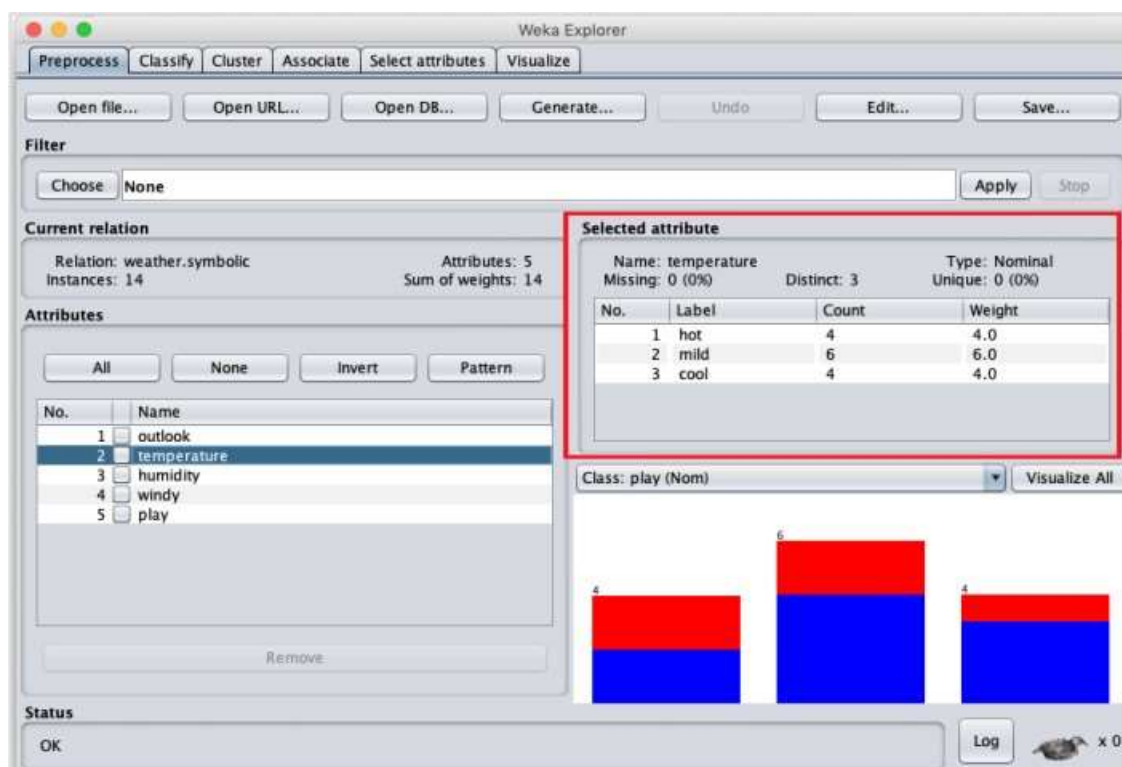
- There are 14 instances - the number of rows in the table.
- The table contains 5 attributes - the fields, which are discussed in the upcoming sections.

On the left side, notice the Attributes sub window that displays the various fields in the database.



The weather database contains five fields - outlook, temperature, humidity, windy and play. when you select an attribute from this list by clicking on it, further details on the attribute itself are displayed on the right hand side.

Let us select the temperature attribute first. When you click on it, you would see the following screen:

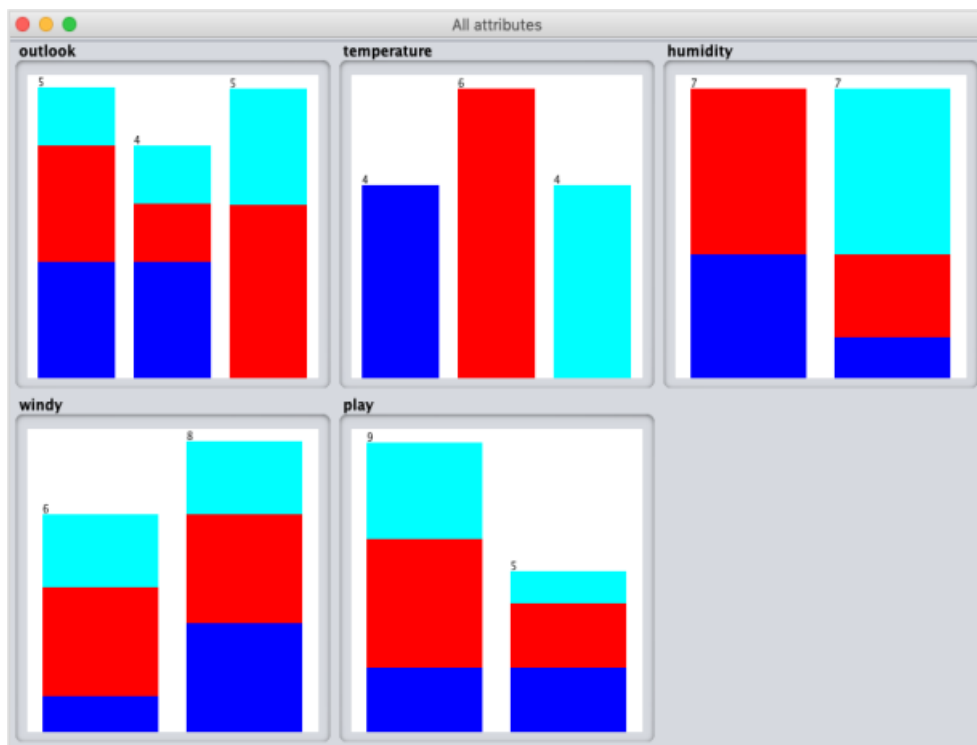


In the Selected Attribute subwindow, you can observe the following:

- The name and the type of the attribute are displayed.
- The type for the temperature attribute is Nominal.
- The number of Missing values is zero.
- There are three distinct values with no unique value.
- The table underneath this information shows the nominal values for this field as hot, mild and cold.
- It also shows the count and weight in terms of a percentage for each nominal value.

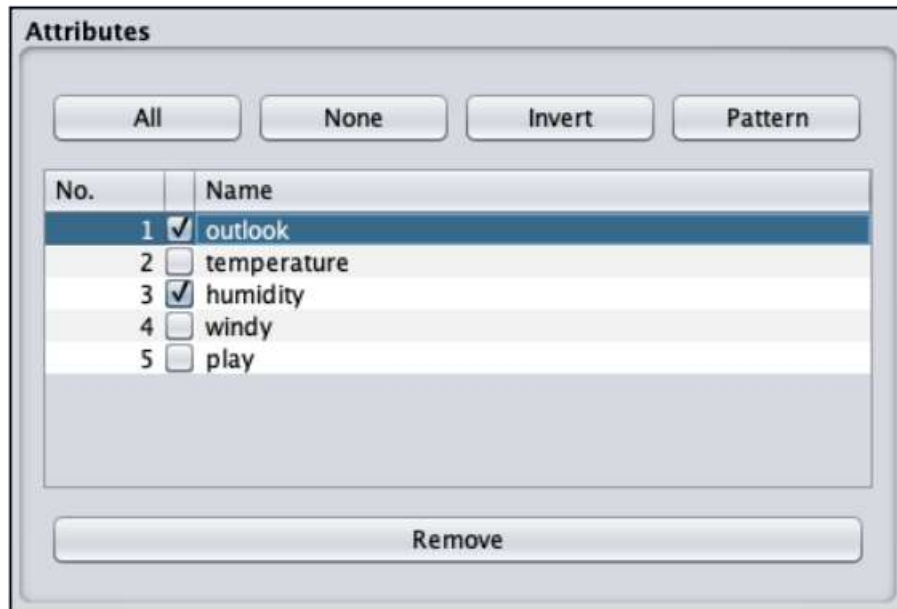
At the bottom of the window, you see the visual representation of the class values

If you click on the Visualize All button, you will be able to see all features in one single window as shown here:



## Removing Attributes

Many a time, the data that you want to use for model building comes with many irrelevant fields. For example, the customer database may contain his mobile number which is relevant in analysing his credit rating



To remove Attribute/s select them and click on the Remove button at the bottom.

The selected attributes would be removed from the database. After you fully preprocess the data, you can save it for model building.

Next, you will learn to preprocess the data by applying filters on this data.

## **Data Integration**

Suppose you have 2 datasets as below and need to merge them together

file1.csv	file2.csv
1 Name, Age	1 Gender, Cuntry
2 Lahiru, 26	2 M, SL
3 Alice, 35	3 F, US
4 Bob, 20	4 M, UK
5 eve, 40	5 F, China

Open a Command Line Interface


- Run the following command replacing values as needed
- `java -cp weka.jar weka.core.Instances merge <path to file1> <path to file 2> <path to result file>`

### Example

```
java weka.core.Instances merge C:\Users\Ram\Downloads\file1.csv  
C:\Users\Ram\Downloads\file2.csv > C:\Users\Ram\Downloads\results.csv
```

Finished redirecting output to 'C:\Users\Ram\Downloads\results.csv'.

Now you can see results.csv or results.csv file in your given location as below.



```
file1.csv  file2.csv  result.arff x  
1  @relation file1_file2  
2  
3  @attribute Name {Lahiru,Alice,Bob,eve}  
4  @attribute Age numeric  
5  @attribute Gender {M,F}  
6  @attribute Cuntry {SL,US,UK,China}  
7  
8  @data  
9  Lahiru,26,M,SL  
10 Alice,35,F,US  
11 Bob,20,M,UK  
12 eve,40,F,China  
13 |
```

### RESULT:

Thus the weka software are installed and performed data exploration and integration successfully

**EX.NO.:2**

## **APPLY WEKA TOOL FOR DATA VALIDATION**

### **AIM:**

To validate the data stored in data warehouse using Weka tool.

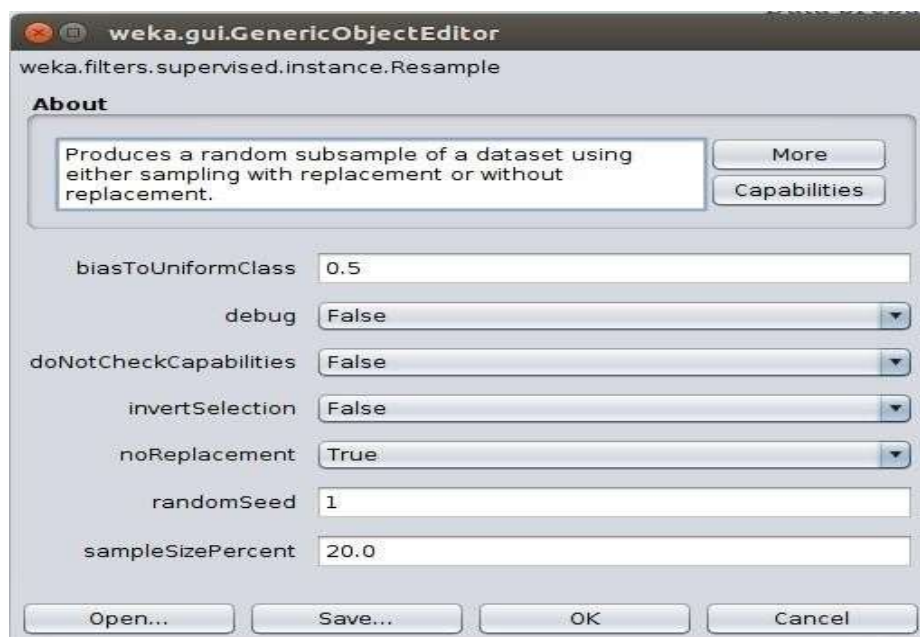
### **PROCEDURE:**

Data validation is the process of verifying and validating data that is collected before it is used. Any type of data handling task, whether it is gathering data, analyzing it, or structuring it for presentation, must include data validation to ensure accurate results.

#### **1. Data Sampling**

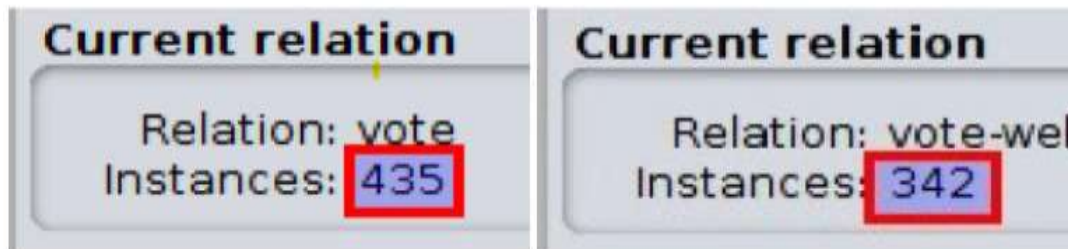
After loading your dataset

- Click on choose ( certain datasets in sample datasets does not allow this operation. I used Brest-cancer dataset for this experiment )
- Filters -> supervised -> Instance -> Re-sample
- Click on the name of the algorithm to change parameters
- Change biasToUniformClass to have a biased sample. If you set it to 1 resulting dataset will have equal number of instances for each class. Ex:- Brest-cancer positive 20 negative 20.
- Change noReplacement accordingly.
- Change sampleSizePercent accordingly. ( self explanatory )



## 2. Removing duplicates

- Choose filters -> unsupervised -> instance -> RemoveDuplicates
- Compare the results as below



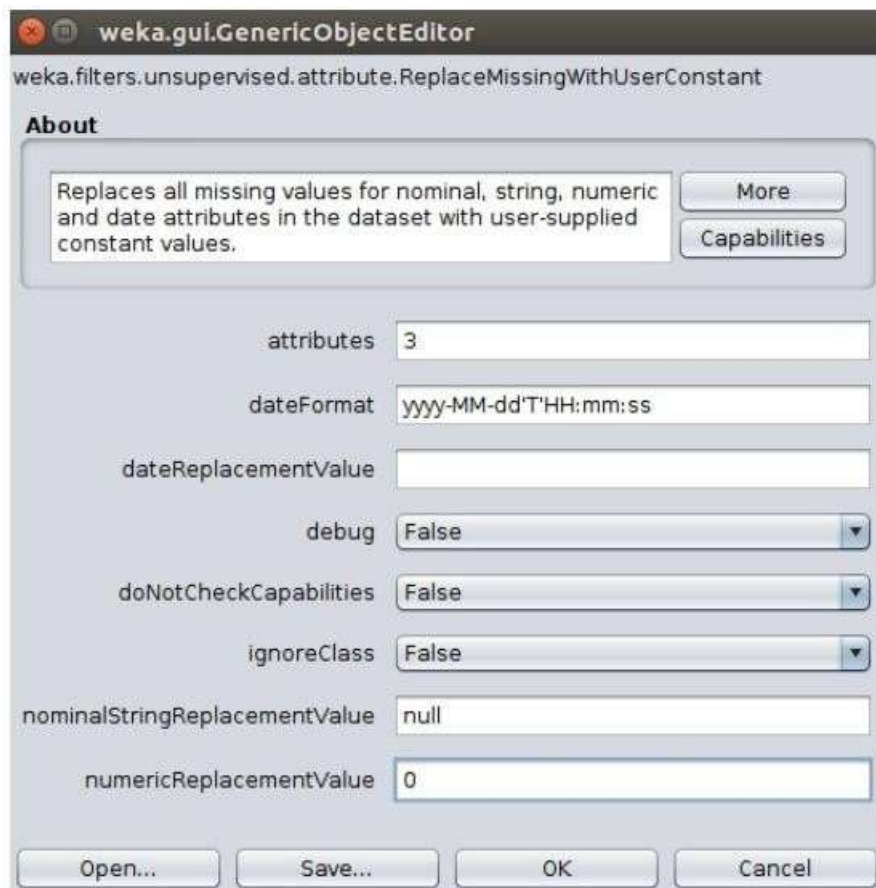
### 2.2 Dealing with missing values

- Open labor.arff file using weka. ( which has missing values )
- Click on edit button on the top bar to get a view of dataset as blow. Here you can clearly see the missing values in gray areas.

The screenshot shows the 'Viewer' window for the 'labor-neg-data' dataset. The table has 31 rows and 14 columns. Missing values are indicated by gray cells. The columns are: No., duration, wage-incr, wage-incr, wage-incr, cost-of-living, working-ho, u, pension, standby-shift-differ, education, statutory, vacation, longterm, contribut, bereave, and average. The rows contain numerical and categorical data, with some cells being grayed out to represent missing values.

- Filters -> unsupervised -> attribute -> replaceMissingValuesWithUserConstants





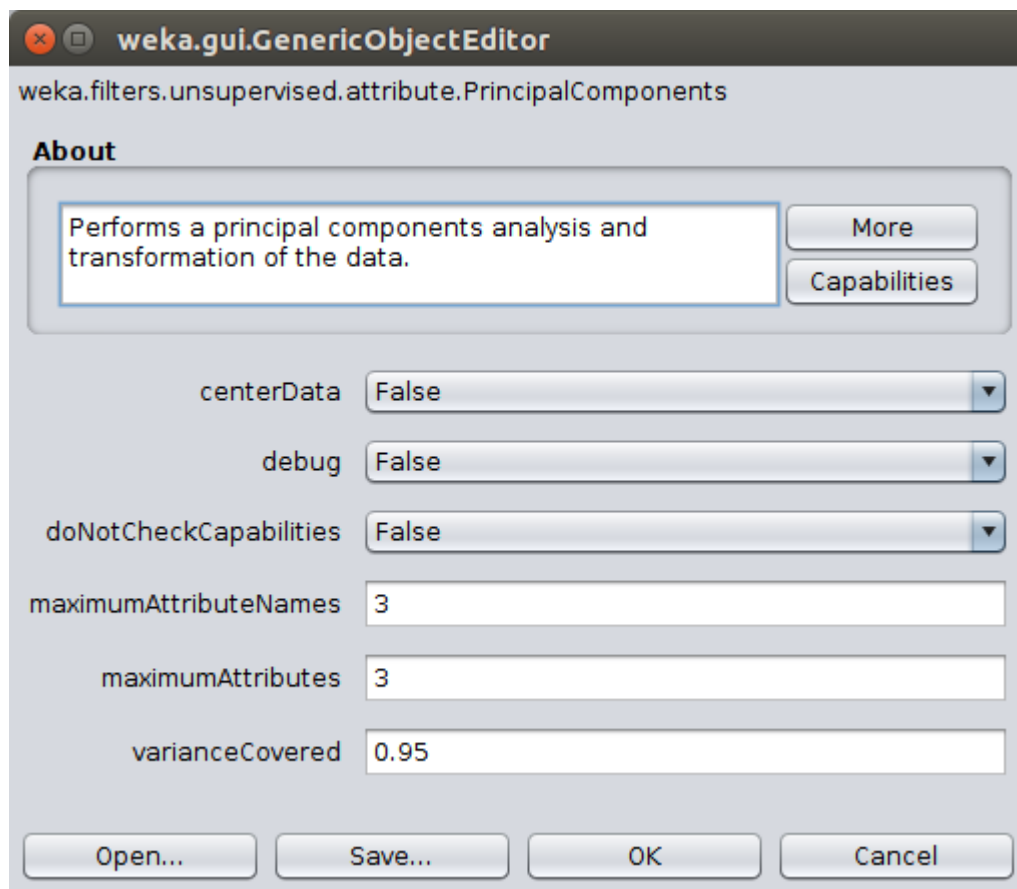
- In attributes set the column number you want to replace values.
- Set nominalStringReplacementValue to a suitable string if selected column is nominal.
- Set numericReplacementValue to 0, -1 depending on your requirement if selected column is numeric.
- fill all replacement values, add “first-last” to attribute column to apply for all columns at once.
- ReplaceMissingValues filter which replace numeric values with mean and nominal values with mode.

### 3. Data Reduction

#### PCA

- Load iris dataset
- Filters -> unsupervised -> attribute -> PrincipleComponents





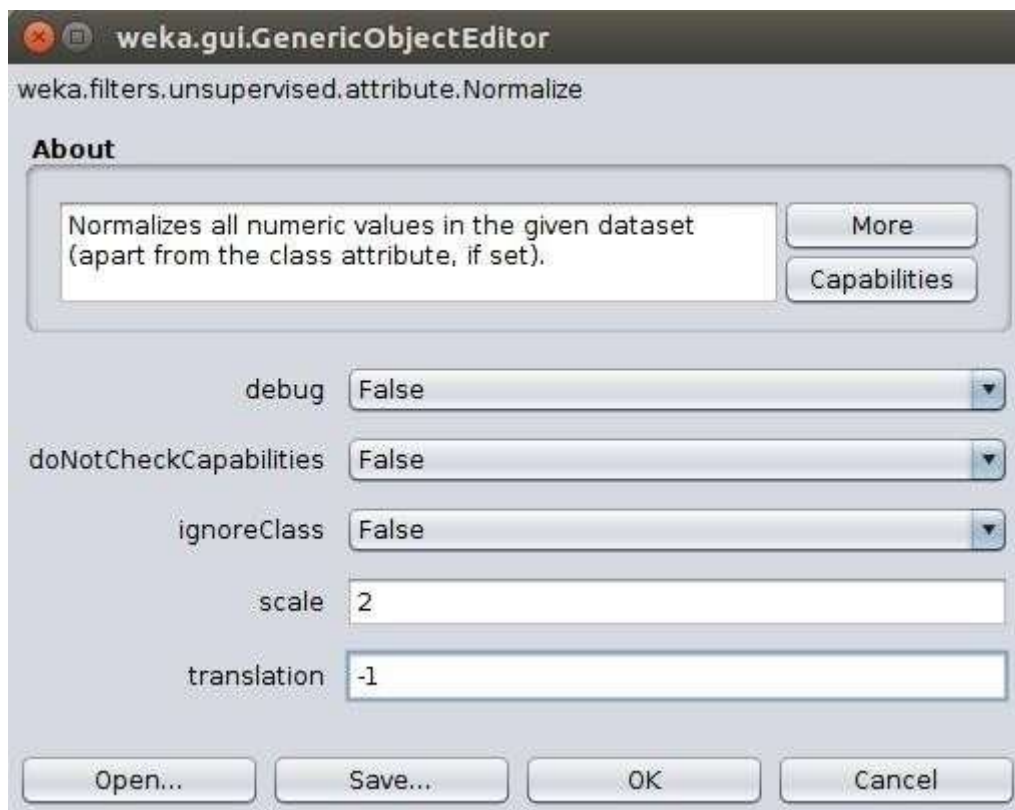
- Original iris dataset have 5 columns. ( 4 data + 1 class ). Lets reduce that to 3 columns ( 2 data + 1 class ).
- maximumAttributes – No of attributes we need after reduction.
- maximumAttributeNames – PCA algorithm calculates 4 principle components for this dataset. Upon them we are selecting the 2 components which have the most variance ( PC1, PC2 ). Then we need to re-represent data again using these selected components ( reducing 4D plot to 2D plot ). In this process we can select how many principle components we are using when re-generating values. See the final result below where you can see new columns are created using 3 principle components multiplied by respective bias values.

No.		Name
1	<input type="checkbox"/>	-0.581petallength-0.566petalwidth-0.522sepallength...
2	<input type="checkbox"/>	0.926sepalwidth+0.372sepallength+0.065petalwidth...
3	<input type="checkbox"/>	class

#### 4. Data transformation

##### Normalization

- Load iris dataset
- Filters -> unsupervised -> attribute -> normalize



- Normalization is important when you don't know the distribution of data beforehand.
- Scale is the length of number line and translation is the lower bound.
- Ex :- scale 2 and translation -1 => -1 to 1, scale 4 and translation -2 => -2 to 2
- This filter get applied to all numeric columns. You can't selectively normalize.

### Standardization

- Load iris dataset.
- Used when dataset known to be in Gaussian (bell curve) distribution.
- Filters -> unsupervised -> attribute -> standardize
- This filter get applied to all numeric columns. You can't selectively standardize.

### Discretization

- Load diabetes dataset.
- Discretization comes in handy when using decision trees.
- Suppose you need to change weight column to two values like low and high.

weka.filters.unsupervised.attribute.Discretize

**About**

An instance filter that discretizes a range of numeric attributes in the dataset into nominal attributes.

More  
Capabilities

attributeIndices 6

binRangePrecision 6

bins 2

debug False

desiredWeightOfInstancesPerInterval -1.0

doNotCheckCapabilities False

findNumBins False

ignoreClass False

invertSelection False

makeBinary False

spreadAttributeWeight False

useBinNumbers False

useEqualFrequency True

Open... Save... OK Cancel

- Set column number 6 to AttributeIndices.
- Set bins to 2 ( Low/ High)
- When you set equal frequency to true there will be equal number of high and low entries in the final column.

**RESULT:**

Thus the software as performed and apply weka tool for data validation as successfully validate.

EX.NO.:3

## PLAN THE ARCHITECTURE FOR REAL TIME APPLICATION

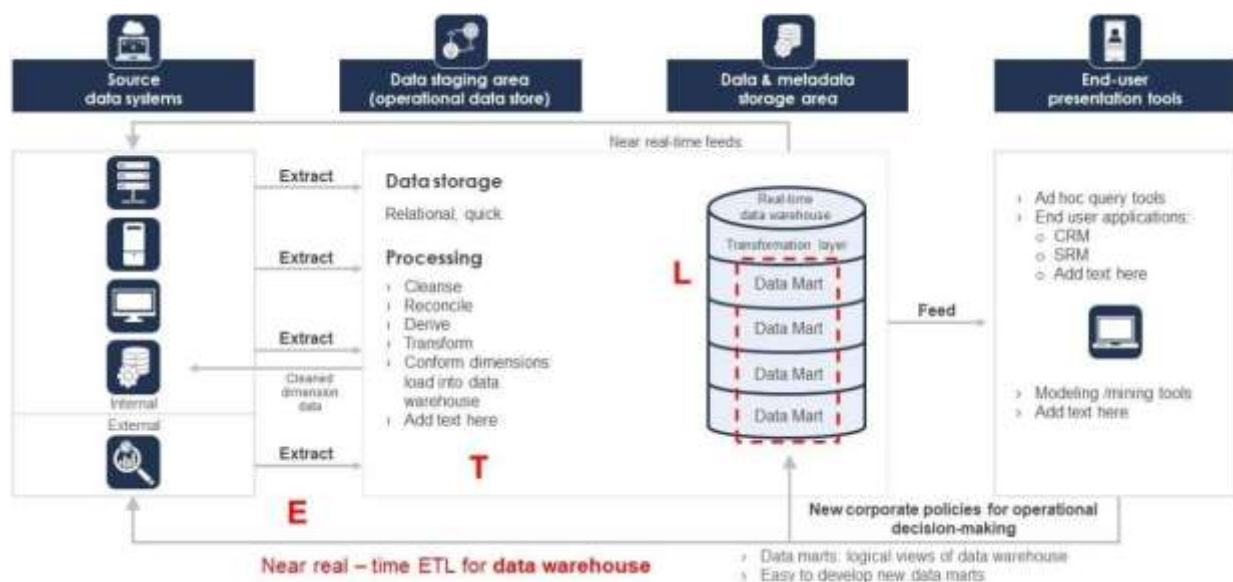
### AIM:

To plan the architecture for real time application.

### PROCEDURE:

### DESIGN STEPS:

1. **Gather Requirements:** Aligning the business goals and needs of different departments with the overall data warehouse project.
2. **Set Up Environments:** This step is about creating three environments for data warehouse development, testing, and production, each running on separate servers
3. **Data Modeling:** Design the data warehouse schema, including the fact tables and dimension tables, to support the business requirements.
4. **Develop Your ETL Process:** ETL stands for Extract, Transform, and Load. This process is how data gets moved from its source into your warehouse.
5. **OLAP Cube Design:** Design OLAP cubes to support analysis and reporting requirements.
6. **Reporting & Analysis:** Developing and deploying the reporting and analytics tools that will be used to extract insights and knowledge from the data warehouse.
7. **Optimize Queries:** Optimizing queries ensures that the system can handle large amounts of data and respond quickly to queries.
8. **Establish a Rollout Plan:** Determine how the data warehouse will be introduced to the organization, which groups or individuals will have access to it, and how the data will be presented to these users.



**EX.NO.:4**

## **QUERY FOR SCHEMA DEFINITION**

### **AIM:**

To Write a query for Star, Snowflake and Galaxy schema definitions.

### **PROCEDURE:**

#### **STAR SCHEMA**

- Each dimension in a star schema is represented with only one-dimension table.
- This dimension table contains the set of attributes.
- There is a fact table at the center. It contains the keys to each of four dimensions.
- The fact table also contains the attributes

#### **SNOWFLAKE SCHEMA**

- Some dimension tables in the Snowflake schema are normalized.
- The normalization splits up the data into additional tables.
- Unlike Star schema, the dimensions table in a snowflake schema are normalized.

#### **FACT CONSTELLATION SCHEMA**

- A fact constellation has multiple fact tables. It is also known as galaxy schema.
- The sales fact table is same as that in the star schema.
- The sales fact table is same as that in the star schema.
- The shipping fact table also contains two measures, namely dollars sold and units sold.

### **SAMPLE PROGRAM:**

Table Creation: (Galaxy Schema)

```
CREATE TABLE ProductCategory (  
    CategoryID INT PRIMARY KEY,  
    CategoryName VARCHAR(255)  
);  
  
CREATE TABLE Product (  
    ProductID INT PRIMARY KEY,  
    ProductName VARCHAR(255),  
    CategoryID INT,  
    FOREIGN KEY (CategoryID) REFERENCES ProductCategory(CategoryID)  
);  
  
CREATE TABLE CustomerLocation (  
    LocationID INT PRIMARY KEY,  
    City VARCHAR(255),  
    State VARCHAR(255)  
);
```

```
CREATE TABLE Customer (  
    CustomerID INT PRIMARY KEY,  
    CustomerName VARCHAR(255),  
    LocationID INT,  
    FOREIGN KEY (LocationID) REFERENCES CustomerLocation(LocationID)  
);
```

```
CREATE TABLE Sales (  
    SaleID INT PRIMARY KEY,  
    Date DATE,  
    ProductID INT,  
    CustomerID INT,  
    SalesAmount DECIMAL(10, 2),  
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID),  
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)  
);
```

```
CREATE TABLE OrderFact (  
    OrderID INT PRIMARY KEY,  
    Date DATE,  
    ProductID INT,  
    CustomerID INT,  
    Quantity INT,  
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID),  
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)  
);
```

### INSERT SAMPLE DATA

```
INSERT INTO ProductCategory (CategoryID, CategoryName) VALUES  
(1, 'Electronics'),  
(2, 'Clothing'),  
(3, 'Home & Garden');
```

```
INSERT INTO Product (ProductID, ProductName, CategoryID) VALUES  
(101, 'Smartphone', 1),  
(102, 'Laptop', 1),  
(201, 'T-Shirt', 2),  
(202, 'Jeans', 2),  
(301, 'Coffee Maker', 3),  
(302, 'Gardening Tools', 3);
```

```
INSERT INTO CustomerLocation (LocationID, City, State) VALUES  
(501, 'New York', 'NY'),  
(502, 'Los Angeles', 'CA'),  
(503, 'Chicago', 'IL');
```

```
INSERT INTO Customer (CustomerID, CustomerName, LocationID) VALUES
(1001, 'John Doe', 501),
(1002, 'Jane Smith', 502),
(1003, 'Bob Johnson', 503);
```

```
INSERT INTO Sales (SaleID, Date, ProductID, CustomerID, SalesAmount) VALUES
(10001, '2024-02-01', 101, 1001, 500.00),
(10002, '2024-02-02', 201, 1002, 30.00),
(10003, '2024-02-03', 301, 1003, 150.00),
(10004, '2024-02-04', 102, 1001, 800.00),
(10005, '2024-02-05', 202, 1002, 50.00);
```

```
INSERT INTO OrderFact (OrderID, Date, ProductID, CustomerID, Quantity)
VALUES
(20001, '2024-02-01', 101, 1001, 2),
(20002, '2024-02-02', 201, 1002, 3),
(20003, '2024-02-03', 301, 1003, 1),
(20004, '2024-02-04', 102, 1001, 5),
(20005, '2024-02-05', 202, 1002, 2);
```

### **STAR SCHEMA DEFINITION**

```
SELECT
    s.date,
    s.sales_amount,
    p.product_name,
    c.customer_name
FROM
    sales s
JOIN
    product p ON s.product_id = p.product_id
JOIN
    customer c ON s.customer_id = c.customer_id;
```

OUTPUT:

```
+-----+-----+-----+-----+
| date   | sales_amount | product_name | customer_name |
+-----+-----+-----+-----+
| 2024-02-01 |    500.00 | Product A   | Customer X   |
| 2024-02-02 |    750.00 | Product B   | Customer Y   |
| 2024-02-03 |    600.00 | Product C   | Customer Z   |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

## **SNOWFLAKE SCHEMA DEFINITION**

```
SELECT
  s.Date,
  s.SalesAmount,
  p.ProductName,
  pc.CategoryName,
  c.CustomerName,
  cl.City
FROM
  Sales s
JOIN
  Product p ON s.ProductID = p.ProductID
JOIN
  ProductCategory pc ON p.CategoryID = pc.CategoryID
JOIN
  Customer c ON s.CustomerID = c.CustomerID
JOIN
  CustomerLocation cl ON c.LocationID = cl.LocationID;
```

OUTPUT:

Date	SalesAmount	ProductName	categoryName	CustomerName	City
2024-02-01	500.00	Smartphone	Electronics	John Doe	New York
2024-02-04	800.00	LAPTOP	ELECTRONICS	JOHN DOE	NEW YORK
2024-02-02	30.00	T-Shirt	Clothing	Jane Smith	Los Angeles
2024-02-05	50.00	Jeans	Clothing	Jane Smith	Los Angeles
2024-02-03	150.00	Coffee Maker	Home & Garden	Bob Johnson	Chicago

## **FACT CONSTELLATION SCHEMA DEFINITION**

```
SELECT
  s.Date AS SalesDate,
  s.SalesAmount,
  o.Date AS OrderDate,
  o.Quantity,
  p.ProductName,
  pc.CategoryName,
  c.CustomerName,
  cl.City
FROM
  Sales s
JOIN
  Product p ON s.ProductID = p.ProductID
JOIN
  ProductCategory pc ON p.CategoryID = pc.CategoryID
JOIN
  Customer c ON s.CustomerID = c.CustomerID
```



JOIN

CustomerLocation cl ON c.LocationID = cl.LocationID

JOIN

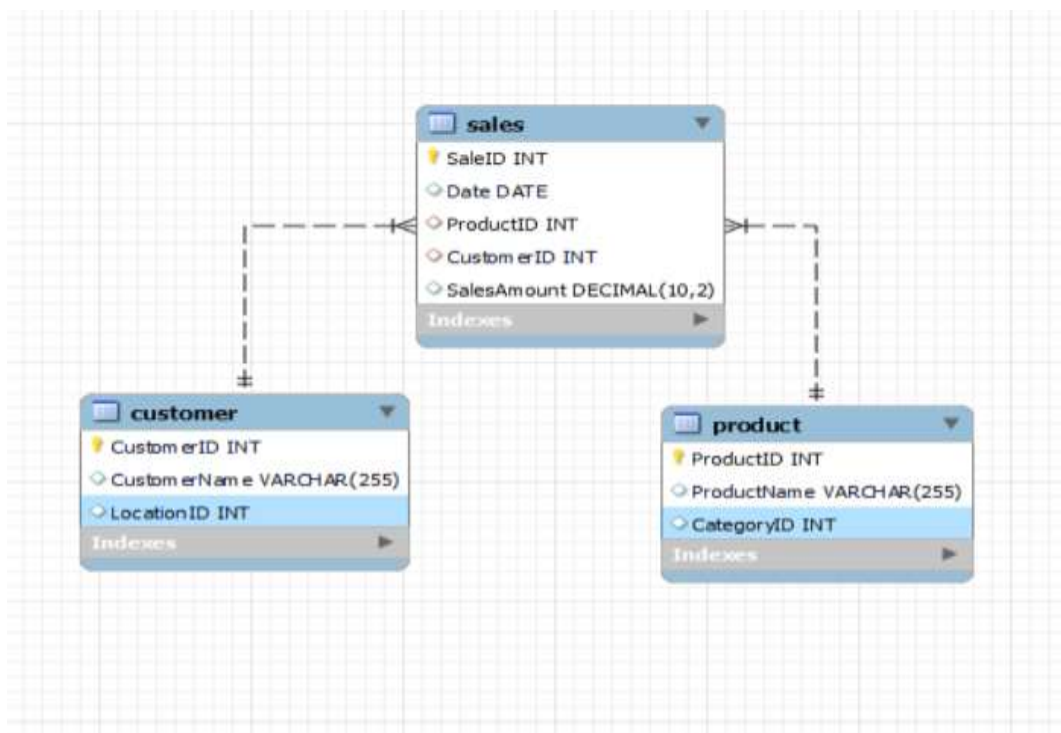
OrderFact o ON s.ProductID = o.ProductID AND s.CustomerID = o.CustomerID AND s.Date = o.Date;

OUTPUT:

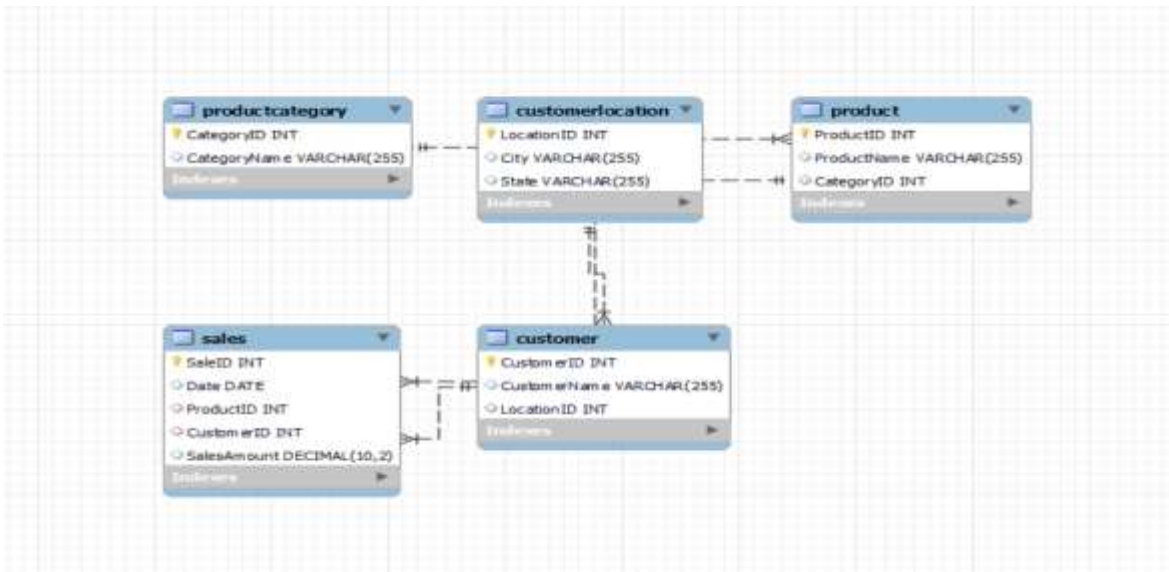
SalesDate	SalesAmount	OrderDate	Quantity	ProductName	CategoryName	CustomerName	City
2024-02-01	500.00	2024-02-01	2	Smartphone	Electronics	John Doe	New York
2024-02-04	800.00	2024-02-04	5	Laptop	Electronics	John Doe	New York
2024-02-02	30.00	2024-02-02	3	T-Shirt	Clothing	Jane Smith	Los Angeles
2024-02-05	50.00	2024-02-05	2	Jeans	Clothing	Jane Smith	Los Angeles
2024-02-03	150.00	2024-02-03	1	Coffee Maker	Home & Garden	Bob Johnson	Chicago

5 rows in set (0.00 sec)

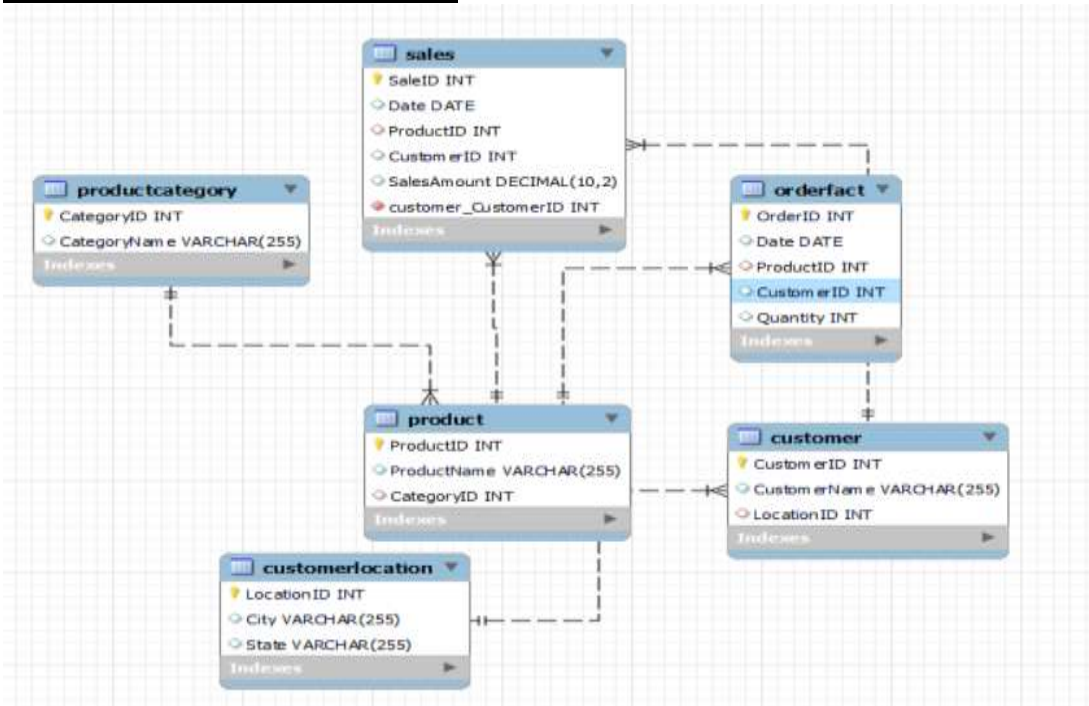
## STAR SCHEMA



## SNOWFLAKE SCHEMA



## FACT CONSTELLATION SCHEMA



## RESULT:

Thus the query for star, Snowflake and Galaxy schema was written Successfully.

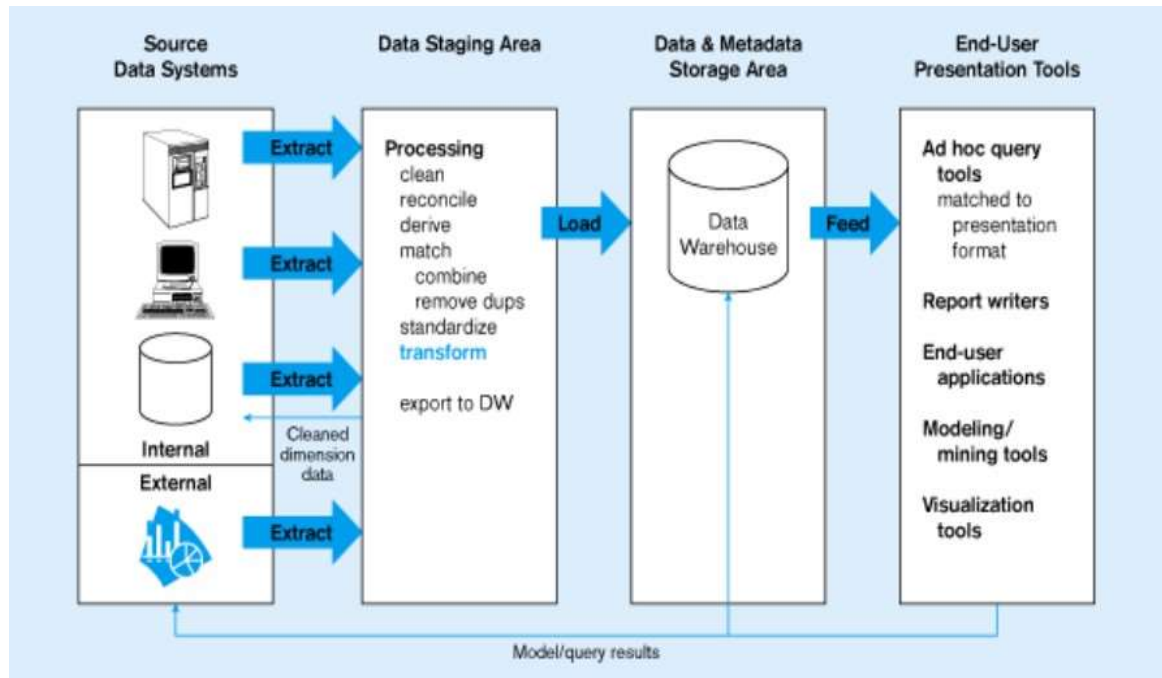
**EX.NO.:5**

## **DESIGN DATA WARE HOUSE FOR REAL TIME APPLICATIONS**

### **AIM:**

To design a data warehouse for real time applications

### **PROCEDURE:**



### **1. Understand Requirements:**

- Identify the business requirements and goals of the data warehouse.
- Determine the types of queries that will be run in real-time.
- Consider the volume of data and expected growth over time.

### **2. Data Modeling:**

- Create a star or snowflake schema based on the business requirements.
- Identify dimensions and facts in your data model.
- Normalize or denormalize tables based on query patterns and performance considerations.

### **3. Indexing:**

- Use appropriate indexes to speed up query performance.
- Consider indexing columns used in WHERE clauses, JOIN conditions, and ORDER BY clauses.
- Regularly analyze and optimize indexes to ensure their effectiveness.

#### **4. Partitioning:**

- Partition large tables to improve query performance and manage data more efficiently.
- Consider partitioning by date, range, or key, depending on the nature of the data and queries.
- Regularly review and optimize partitioning strategies as data grows.

#### **5. Data Loading:**

- Implement an ETL (Extract, Transform, Load) process for efficient data loading.
- Consider using tools like Apache Kafka for real-time data streaming and loading.
- Optimize the data loading process to minimize downtime and ensure real-time updates.

#### **6. Caching:**

- Implement caching mechanisms to reduce the load on the database for frequently accessed data.
- Consider using an in-memory caching solution like Redis or Memcached for improved performance.

#### **7. Concurrency and Locking:**

- Optimize the database for concurrent access by multiple users or applications.
- Use appropriate isolation levels and locking strategies to manage concurrent transactions effectively.

#### **8. Query Optimization:**

- Regularly analyze and optimize SQL queries for better performance.
- Use EXPLAIN statements to understand and optimize query execution plans.
- Consider denormalization for read-heavy queries to avoid JOIN operations.

#### **9. Backup and Recovery:**

- Implement a robust backup and recovery strategy to ensure data integrity.
- Regularly test backup and recovery processes to validate their effectiveness.

#### **10. Security:**

- Implement proper access controls and authentication mechanisms to secure the data warehouse.
- Regularly audit and monitor database activity to detect and address potential security issues.

#### **11. Monitoring and Performance Tuning:**

- Use monitoring tools to track the performance of the data warehouse.
- Regularly analyze performance metrics and make adjustments as needed.

#### **12. Scaling:**

- Plan for scalability by considering options such as sharding or vertical scaling.
- Monitor system resource usage and scale the infrastructure as needed.

**CODE:**

```
CREATE DATABASE a;  
USE a;
```

```
-- Creating the tables
```

```
CREATE TABLE Achievements (  
    ach_id INT PRIMARY KEY,  
    ach_desc VARCHAR(255),  
    ach_type VARCHAR(255),  
    ach_date DATE,  
    ach_st_ssn INT  
);
```

```
CREATE TABLE RegisteredActivities (  
    reg_id INT PRIMARY KEY,  
    reg_activityName VARCHAR(255),  
    reg_year YEAR,  
    reg_st_ssn INT  
);
```

```
CREATE TABLE Alumni (  
    al_GPA DECIMAL(3,2),  
    al_date DATE,  
    al_st_ssn INT  
);
```

```
CREATE TABLE Receipt (  
    re_id INT PRIMARY KEY,  
    re_amount DECIMAL(10,2),  
    re_paidFlag BOOLEAN,  
    re_paidOnDueAfterLag INT,  
    re_ac_id INT  
);
```

```
CREATE TABLE Account (  
    ac_id INT PRIMARY KEY,  
    ac_fees DECIMAL(10,2),  
    ac_semester VARCHAR(255),  
    ac_totalCredits INT,  
    ac_modeOfPayment VARCHAR(255),  
    ac_st_ssn INT  
);
```

```
CREATE TABLE Student (  
    st_ssn INT PRIMARY KEY,  
    st_name VARCHAR(255),  
    st_sex CHAR(1),  
    st_birthDate DATE,  
    st_address VARCHAR(255),  
    st_major VARCHAR(255)  
);
```

```
CREATE TABLE Instructor (  
    in_id INT PRIMARY KEY,  
    in_name VARCHAR(255),  
    in_phone VARCHAR(10),  
    in_birthDate DATE,  
    in_certificate VARCHAR(255),  
    in_address VARCHAR(255)  
);
```

```
CREATE TABLE TranscriptFactTable (  
    tr_st_ssn INT,  
    tr_in_id INT,  
    tr_courseName VARCHAR(255),
```

```
tr_courseDifficulty VARCHAR(255),
tr_grade CHAR(2),
tr_semester VARCHAR(255),
PRIMARY KEY (tr_st_ssn, tr_in_id)
);
```

-- Adding the foreign key constraints

```
ALTER TABLE Achievements ADD FOREIGN KEY (ach_st_ssn) REFERENCES Student(st_ssn);
ALTER TABLE RegisteredActivities ADD FOREIGN KEY (reg_st_ssn) REFERENCES Student(st_ssn);
ALTER TABLE Alumni ADD FOREIGN KEY (al_st_ssn) REFERENCES Student(st_ssn);
ALTER TABLE Receipt ADD FOREIGN KEY (re_ac_id) REFERENCES Account(ac_id);
ALTER TABLE Account ADD FOREIGN KEY (ac_st_ssn) REFERENCES Student(st_ssn);
ALTER TABLE TranscriptFactTable ADD FOREIGN KEY (tr_st_ssn) REFERENCES Student(st_ssn);
ALTER TABLE TranscriptFactTable ADD FOREIGN KEY (tr_in_id) REFERENCES Instructor(in_id);-- Inserting
data into the Achievements table
INSERT INTO Achievements (ach_id, ach_desc, ach_type, ach_date, ach_st_ssn)
VALUES (1, 'Won coding competition', 'Academic', '2024-02-01', 123456789);
```

-- Inserting data into the Registered\_Activities table

```
INSERT INTO Registered_Activities (reg_id, reg_activityName, reg_year, reg_st_ssn)
VALUES (1, 'Coding Club', 2024, 123456789);
```

-- Inserting data into the Alumni table

```
INSERT INTO Alumni (al_GPA, al_date)
VALUES (3.5, '2024-05-01');
```

-- Inserting data into the Receipt table

```
INSERT INTO Receipt (re_id, re_amount, re_paidFlag, re_paidOnDueAfterLag, re_ac_id)
VALUES (1, 5000, true, 0, 1);
```

-- Inserting data into the Account table

```
INSERT INTO Account (ac_id, ac_fees, ac_semester, ac_totalCredits, ac_modeOfPayment, ac_st_ssn)
VALUES (1, 10000, 'Spring 2024', 15, 'Credit Card', 123456789);
```

-- Inserting data into the Student table

```
INSERT INTO Student (st_ssn, st_name, st_sex, st_birthDate, st_address, st_major)
VALUES (123456789, 'John Doe', 'M', '2000-01-01', '123 Main St', 'Computer Science');
```

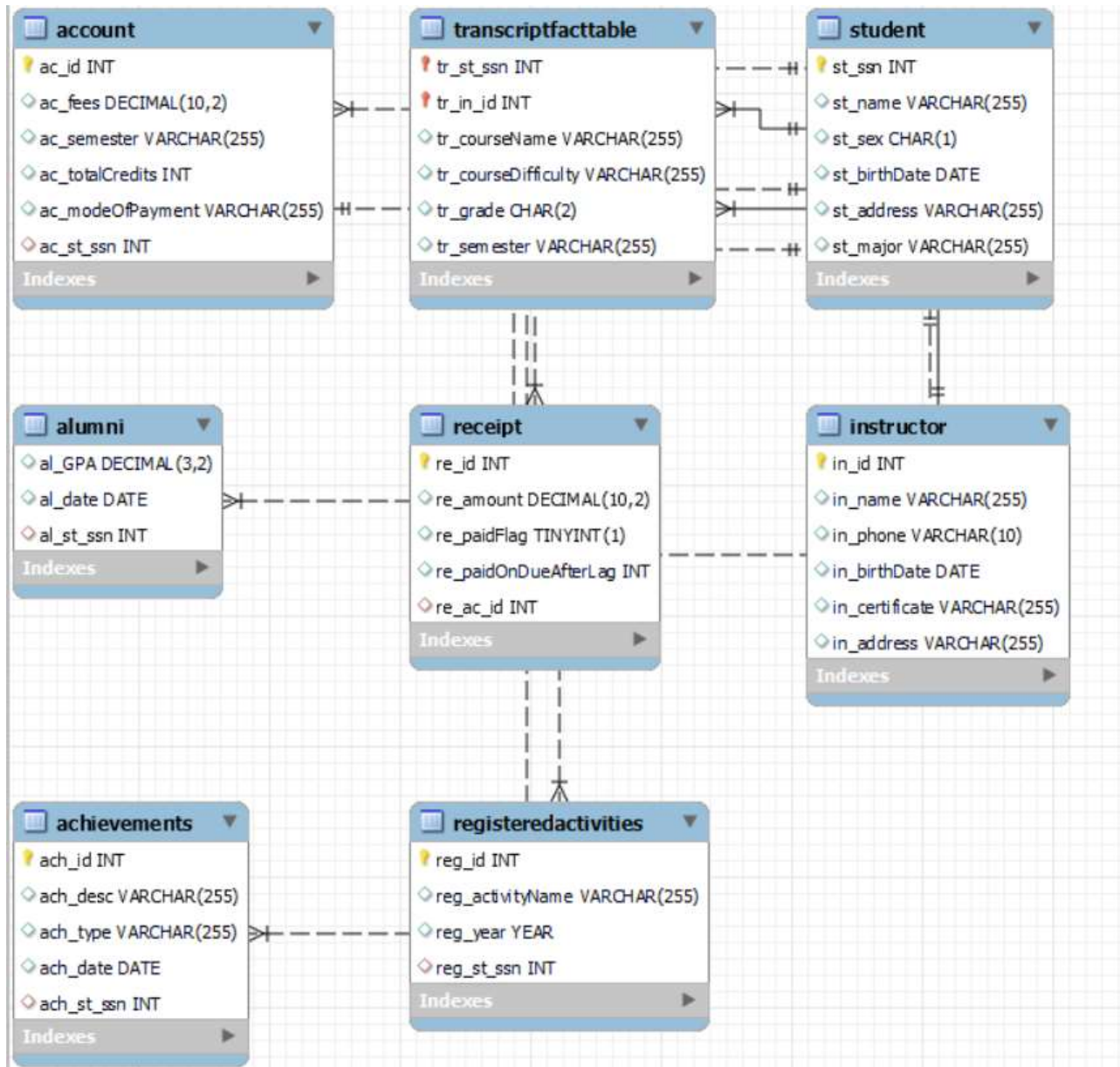
-- Inserting data into the Instructor table

```
INSERT INTO Instructor (in_id, in_name, in_phone, in_birthDate, in_certificate, in_address)
VALUES (987654321, 'Jane Smith', '555-555-55', '1980-01-01', 'PhD in Computer Science', '456 Main St');
```

-- Inserting data into the Transcript\_Fact\_Table

```
INSERT INTO Transcript_Fact_Table (tr_st_ssn, tr_in_id, tr_courseName, tr_courseDifficulty, tr_grade, tr_semester)
VALUES (123456789, 987654321, 'Intro to Computer Science', 'Easy', 'A', 'Spring 2024');
```

### Output:



### RESULT:

Thus the program was written and executed successfully to design a data warehouse for real time applications

**EX.NO: 6**

## **ANALYSES THE DIMENSIONAL MODELING**

**AIM:**

To Analyses the dimensional Modeling using MYSQL.

**PROCEDURE:**

To analyze the dimensional modeling, you can follow this procedure:

### **1. Understand Dimensional Modeling Concepts:**

Review the concept of dimensional modeling, which involves creating dimension tables and a fact table to organize and structure data in a way that supports efficient querying and reporting.

### **2. Examine Dimension Tables:**

Look at the dimension tables (dim\_time, dim\_product, dim\_location) and understand their structure.

dim\_time includes time-related information such as date, year, quarter, month, and day.

dim\_product includes product-related information such as product name, category, and subcategory.

dim\_location includes location-related information such as country, city, and state.

### **3. Analyze Fact Table:**

Examine the fact table (fact\_sales) and understand how it relates to dimension tables through foreign key references.

The fact table includes sales-related information such as sales ID, time ID, product ID, location ID, amount, and quantity.

Foreign key relationships link the fact table to the corresponding records in dimension tables.

### **4. Review Data Population:**

Observe the sample data inserted into dimension tables and the fact table.

Understand how the data in dimension tables is related to the fact table through foreign keys.

Verify that the sample sales data in the fact table corresponds to existing records in dimension tables.

### **5. Evaluate Data Types:**

Review the data types chosen for columns (e.g., DECIMAL for amount, INT for identifiers) to ensure they are appropriate for the data they store.

### **6. Check for Data Integrity:**

Ensure that foreign key relationships are correctly established to maintain data integrity.

Confirm that each foreign key in the fact table corresponds to a primary key in its respective dimension table.



## **7. Consider Indexing:**

Evaluate whether indexes are applied appropriately, especially on columns used for joins and filtering, to enhance query performance.

## **8. Verify Constraints:**

Check for constraints like primary keys, auto-incrementing values, and other constraints to maintain data consistency and integrity.

## **9. Plan for Updates:**

Consider how updates, deletions, and insertions will be handled, especially in a production environment. Ensure that these operations do not compromise data consistency.

## **10. Documentation:**

Document the dimensional model, including relationships, data types, and constraints, for future reference.

## **PROGRAM:**

-- Create dimension tables

```
CREATE TABLE dim_time (  
    time_id INT AUTO_INCREMENT PRIMARY KEY,  
    date DATE,  
    year INT,  
    quarter INT,  
    month INT,  
    day INT  
);
```

```
CREATE TABLE dim_product (  
    product_id INT AUTO_INCREMENT PRIMARY KEY,  
    product_name VARCHAR(255),  
    category VARCHAR(50),  
    subcategory VARCHAR(50)  
);
```

```
CREATE TABLE dim_location (  
    location_id INT AUTO_INCREMENT PRIMARY KEY,  
    country VARCHAR(100),  
    city VARCHAR(100),
```

```
state VARCHAR(100)
);
```

```
-- Create fact table
```

```
CREATE TABLE fact_sales (
    sales_id INT AUTO_INCREMENT PRIMARY KEY,
    time_id INT,
    product_id INT,
    location_id INT,
    amount DECIMAL(10, 2),
    quantity INT,
    FOREIGN KEY (time_id) REFERENCES dim_time(time_id),
    FOREIGN KEY (product_id) REFERENCES dim_product(product_id),
    FOREIGN KEY (location_id) REFERENCES dim_location(location_id)
);
```

```
-- Populate dimension tables (sample data)
```

```
INSERT INTO dim_time (date, year, quarter, month, day) VALUES
('2024-01-01', 2024, 1, 1, 1),
('2024-01-02', 2024, 1, 1, 2),
```

```
-- Populate other time data...
```

```
INSERT INTO dim_product (product_name, category, subcategory) VALUES
('Product A', 'Category 1', 'Subcategory 1'),
('Product B', 'Category 2', 'Subcategory 2'),
```

```
-- Populate other product data...
```

```
INSERT INTO dim_location (country, city, state) VALUES
('USA', 'New York', 'NY'),
('USA', 'Los Angeles', 'CA'),
```

```
-- Populate other location data...
```

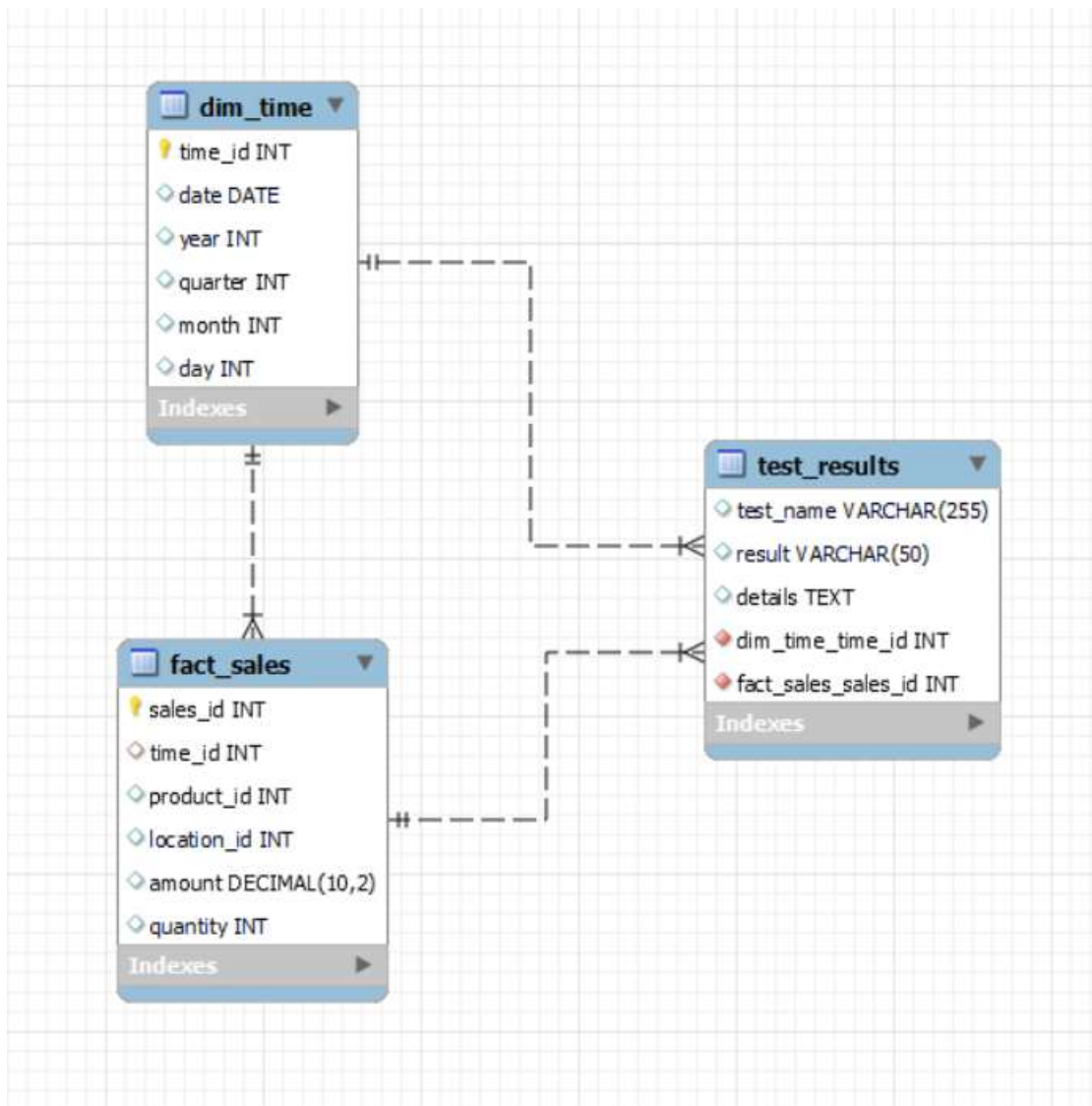
```
-- Populate fact table (sample sales data)
```

```
INSERT INTO fact_sales (time_id, product_id, location_id, amount, quantity) VALUES
(1, 1, 1, 100.00, 2),
```

(2, 2, 2, 150.00, 3),

-- Populate other sales data...

## OUTPUT:



## RESULT:

Thus the program was written and executed successfully for analyze the dimensional modeling

**EX.NO: 7**

## **CASE STUDY USING OLAP**

### **AIM:**

case study scenario for using OLAP (Online Analytical Processing) in a data warehousing environment:

**Case Study:** Retail Sales Analysis

### **Background:**

A retail company operates multiple stores across different regions. They have been collecting transactional data from their point-of-sale (POS) systems for several years. The company wants to gain insights into their sales performance, customer behavior, and product trends to make informed business decisions.

### **Objective:**

The objective is to design a data warehousing solution using OLAP for analyzing retail sales data to uncover actionable insights.

### **Data Sources:**

1. Transactional data: Includes information such as sales date, store ID, product ID, quantity sold, unit price, and total sales amount.
2. Customer data: Includes demographics, loyalty program membership status, and purchase history.
3. Product data: Includes product attributes such as category, brand, and price.

### **Solution:**

1. Data Integration:  
Extract data from various sources and load it into a centralized data warehouse. Transform and cleanse the data to ensure consistency and quality.
2. Dimensional Modeling:  
Design a star schema or snowflake schema to organize the data into fact tables (e.g., sales transactions) and dimension tables (e.g., store, product, time, customer).
3. OLAP Cube Creation:  
Build OLAP cubes based on the dimensional model to provide multi-dimensional views of the data. Dimensions such as time, product, store, and customer can be sliced and diced for analysis.
4. Analysis:

#### **- Sales Performance Analysis:**

Analyze total sales revenue, units sold, and average transaction value by store, region, product category, and time period.

#### **- Customer Behavior Analysis:**

Explore customer segmentation based on demographics, purchase frequency, and purchase amount. Identify high-value customers and their purchasing patterns.

**- Product Trend Analysis:**

Identify top-selling products, analyze product performance over time, and detect seasonality or trends in sales.

**- Cross-Selling Analysis:**

Analyze associations between products frequently purchased together to optimize product placement and marketing strategies.

**5. Visualization:**

Create interactive dashboards and reports using OLAP cube data to present insights to business users. Visualization tools like Tableau, Power BI, or custom-built dashboards can be used.

**6. Decision Making:**

Use insights gained from analysis to make data-driven decisions such as inventory management, marketing campaigns, and product assortment planning.

**Benefits:**

1. Improved Decision Making: Provides timely and relevant insights to stakeholders for making informed decisions.
2. Enhanced Operational Efficiency: Optimizes inventory management, marketing strategies, and resource allocation based on data-driven insights.
3. Competitive Advantage: Enables the company to stay ahead of competitors by understanding customer preferences and market trends.
4. Scalability: The OLAP solution can scale to handle large volumes of data and accommodate evolving business needs.

**Conclusion:**

By leveraging OLAP technology within a data warehousing environment, the retail company can gain deeper insights into their sales data, customer behavior, and product performance, ultimately driving business growth and profitability.

**EX.NO: 8**

## **CASE STUDY USING OLTP**

### **AIM:**

case study scenario for using OLTP (Online Transaction Processing) in a data warehousing environment:

**Case Study:** Online Retail Order Management System

### **Background:**

A retail company operates an e-commerce platform where customers can purchase products online. They need to manage a high volume of transactions efficiently while ensuring data integrity and real-time processing.

### **Objective:**

The objective is to design an OLTP system within a data warehousing environment to handle online retail orders, manage inventory, process payments, and maintain customer information.

### **Solution:**

#### 1. Database Design:

- Design a relational database schema optimized for transactional processing.
- Tables include entities such as customers, orders, products, inventory, and payments.
- Implement normalization to minimize data redundancy and maintain data integrity.

#### 2. Online Order Management:

- Capture customer orders in real-time through the e-commerce platform.
- Validate orders for product availability, pricing, and customer information.
- Generate order IDs and track order status (e.g., pending, shipped, delivered).

#### 3. Inventory Management:

- Update inventory levels upon order placement and fulfillment.
- Implement mechanisms to prevent overselling and ensure accurate stock levels.
- Trigger alerts for low inventory levels to facilitate replenishment.

#### 4. Payment Processing:

- Integrate with payment gateways to securely process customer payments.
- Record payment transactions and associate them with corresponding orders.
- Handle payment authorization, capture, and settlement in real-time.

#### 5. Customer Management:

- Maintain customer profiles with information such as contact details, shipping addresses, and order history.
- Enable customers to update their profiles and track order statuses.
- Implement authentication and authorization mechanisms to ensure data security.

## 6. Scalability and Performance:

- Optimize database performance for handling concurrent transactions and high throughput.
- Implement indexing, partitioning, and caching strategies to improve query performance.
- Scale the system horizontally or vertically to accommodate increasing transaction volumes.

### **Benefits:**

#### 1. Real-time Transaction Processing:

Enables the company to process online orders, manage inventory, and handle payments in real-time, providing a seamless shopping experience for customers.

#### 2. Data Integrity:

Ensures data consistency and accuracy by enforcing constraints and validations within the OLTP system.

#### 3. Efficient Order Fulfillment:

Facilitates efficient order fulfillment and inventory management, reducing order processing times and minimizing stockouts.

#### 4. Improved Customer Experience:

Enhances customer satisfaction by providing timely order updates, personalized recommendations, and secure payment processing.

#### 5. Insight Generation:

Captures transactional data that can be used for business intelligence and analytics purposes, such as identifying sales trends, customer preferences, and market opportunities.

### **Conclusion:**

By implementing an OLTP system within a data warehousing environment, the retail company can effectively manage online retail operations, process transactions in real-time, and maintain data integrity, ultimately driving customer satisfaction and business growth.

**EX.NO: 9****IMPLEMENTATION OF WAREHOUSE TESTING****AIM:**

Implementing warehouse testing involves several steps to ensure the efficiency and accuracy of warehouse operations.

**PROCEDURE:****1. Define Testing Objectives:**

Determine the specific objectives and goals of the warehouse testing. This could include ensuring inventory accuracy, optimizing picking and packing processes, improving order fulfillment times, etc.

**2. Identify Test Scenarios:**

Define a set of test scenarios that cover different aspects of warehouse operations, such as receiving goods, put-away, picking, packing, shipping, and inventory counts. These scenarios should be based on real-world scenarios and should cover both normal and edge cases.

**3. Set Testing Criteria:**

Establish criteria for evaluating the success of each test scenario. This could include accuracy rates, time taken to complete tasks, error rates, etc.

**4. Prepare Test Data:**

Gather or generate the necessary test data to simulate real-world warehouse operations. This could include product data, inventory levels, customer orders, shipping information, etc.

**5. Allocate Testing Resources:**

Assign personnel and equipment necessary to conduct the warehouse testing. This may involve coordinating with warehouse staff, IT personnel, and any external vendors or consultants as needed.

**6. Execute Test Scenarios:**

Conduct the warehouse testing by executing the predefined test scenarios using the allocated resources. Ensure that each scenario is executed according to the defined criteria, and record the results of each test.

**7. Analyze Test Results:**

Analyze the results of the warehouse testing to identify any areas of improvement or areas where issues were encountered. Determine the root causes of any issues and prioritize them based on their impact on warehouse operations.

**8. Implement Remedial Actions:**

Take corrective actions to address any issues or deficiencies identified during the testing process. This could involve updating procedures, modifying system configurations, providing additional training to warehouse staff, etc.



## 9. Document Findings:

Document the findings of the warehouse testing process, including test results, corrective actions taken, and any recommendations for future improvements. This documentation will serve as a reference for future testing cycles and continuous improvement efforts.

## 10. Iterate and Refine:

Continuously iterate and refine the warehouse testing process based on feedback and results from previous testing cycles. Make adjustments as needed to improve the efficiency and effectiveness of warehouse.

### **PROGRAM:**

-- Create a testing schema

```
CREATE SCHEMA IF NOT EXISTS testing;
```

-- Switch to the testing schema

```
USE testing;
```

-- Create the dim\_time table

```
CREATE TABLE dim_time (  
    time_id INT AUTO_INCREMENT PRIMARY KEY,  
    date DATE,  
    year INT,  
    quarter INT,  
    month INT,  
    day INT  
);
```

-- Create the fact\_sales table

```
CREATE TABLE fact_sales (  
    sales_id INT AUTO_INCREMENT PRIMARY KEY,  
    time_id INT,  
    product_id INT,  
    location_id INT,  
    amount DECIMAL(10, 2),  
    quantity INT,  
    FOREIGN KEY (time_id) REFERENCES dim_time(time_id)  
    -- Add foreign key constraints for other dimensions if needed  
);
```

-- Insert sample data into dim\_time and fact\_sales tables (replace with your own data)

```
INSERT INTO dim_time (date, year, quarter, month, day) VALUES  
( '2024-01-01', 2024, 1, 1, 1),  
( '2024-01-02', 2024, 1, 1, 2);
```

```
INSERT INTO fact_sales (time_id, product_id, location_id, amount, quantity) VALUES  
(1, 1, 1, 100.00, 2),  
(2, 2, 2, 150.00, 3);
```

```

-- Create a table to store test results
CREATE TABLE test_results (
    test_name VARCHAR(255),
    result VARCHAR(50),
    details TEXT
);

-- Perform testing and insert results into the test_results table
-- Test 1: Data Completeness Testing
INSERT INTO test_results (test_name, result, details)
SELECT 'Data Completeness Testing',
    CASE WHEN COUNT(*) > 0 THEN 'Pass' ELSE 'Fail' END AS result,
    CASE WHEN COUNT(*) > 0 THEN 'Data completeness test passed for dim_time table.' ELSE
'Data completeness test failed for dim_time table.' END AS details
FROM dim_time;

-- Test 2: Data Accuracy Testing
INSERT INTO test_results (test_name, result, details)
SELECT 'Data Accuracy Testing',
    CASE WHEN SUM(amount < 0) = 0 THEN 'Pass' ELSE 'Fail' END AS result,
    CASE WHEN SUM(amount < 0) = 0 THEN 'Data accuracy test passed for fact_sales table.'
ELSE 'Data accuracy test failed for fact_sales table.' END AS details
FROM fact_sales;

-- Test 3: Data Consistency Testing
INSERT INTO test_results (test_name, result, details)
SELECT 'Data Consistency Testing',
    CASE WHEN COUNT(*) = 0 THEN 'Pass' ELSE 'Fail' END AS result,
    CASE WHEN COUNT(*) = 0 THEN 'Data consistency test passed.' ELSE 'Data consistency
test failed: Duplicate records found.' END AS details
FROM (
    SELECT t.date, COUNT(*) AS record_count
    FROM dim_time t
    JOIN fact_sales f ON t.time_id = f.time_id
    GROUP BY t.date
    HAVING COUNT(*) > 1
) AS duplicates;

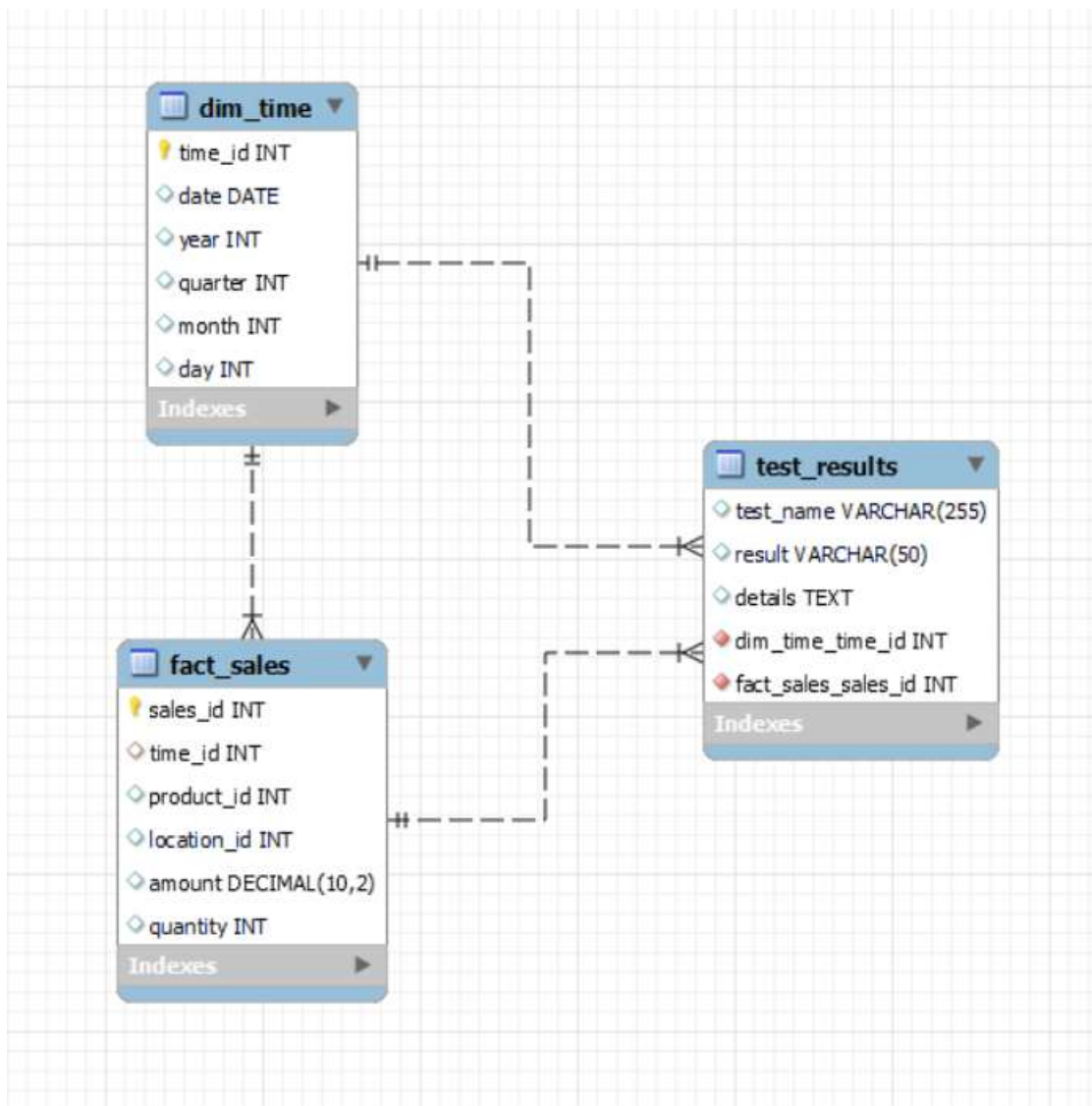
-- Test 4: Business Logic Validation
INSERT INTO test_results (test_name, result, details)
SELECT 'Business Logic Validation',
    CASE WHEN SUM(total_sales) IS NOT NULL THEN 'Pass' ELSE 'Fail' END AS result,
    CASE WHEN SUM(total_sales) IS NOT NULL THEN 'Business logic validation passed.'
ELSE 'Business logic validation failed.' END AS details
FROM (
    SELECT year, month, SUM(amount) AS total_sales
    FROM dim_time t
    JOIN fact_sales f ON t.time_id = f.time_id
    GROUP BY year, month
) AS sales_by_month;

-- View test results
SELECT * FROM test_results;

```

## OUTPUT:

Test_ name	Result	Details
Data Completeness Testing	Pass	Data completeness test passed for dim_time table.
Data Accuracy Testing	Pass	Data accuracy test passed for fact_sales table.
Data Consistency Testing	Pass	Data consistency test passed.
Business Logic Validation	Pass	Business logic validation passed.



## RESULT:

Thus the program was written and executed successfully for Implementation of warehouse testing