BHARATI VIDYAPEETH DEEMED UNIVERSITY
COLLEGE OF ENGINEERING, PUNE - 43

DEPARTMENT OF COMPUTER ENGINEERING

Lab Manual

Database Management System

B.Tech (Computer)Sem-V

VISION OF THE INSTITUTE

"To be World Class Institute for Social Transformation through Dynamic Education"

MISSION OF THE INSTITUTE

- To provide quality technical education with advanced equipment's, qualified faculty Members, infrastructure to meet needs of profession and society.
- To provide an environment conducive to innovation, creativity, research, and entrepreneurial leadership.
- To practice and promote professional ethics, transparency and accountability for social community, economic and environmental conditions.

VISION OF THE DEPARTMENT

"To pursue and excel in the endeavour for creating globally recognized Computer Engineers through Quality education".

MISSION OF THE DEPARTMENT

1. To impart engineering knowledge and skills confirming to a dynamic curriculum.

2.To develop professional, entrepreneurial & research competencies encompassing continuous intellectual growth.

3. To produce qualified graduates exhibiting societal and ethical responsibilities in working environment.

PROGRAM EDUCATIONAL OBJECTIVES

1. Demonstrate technical and professional competencies by applying engineering fundamentals, computing principles and technologies.

2. Learn, Practice, and grow as skilled professionals/ entrepreneur/researchers adapting to the evolving computing landscape.

3. Demonstrate professional attitude, ethics, understanding of social context and interpersonal skills leading to a successful career.

PROGRAM SPECIFIC OUTCOMES

1. To apply fundamental knowledge and technical skills towards solving Engineering problems.
2. To employ expertise and ethical practise through continuing intellectual growth and adapting to the working environment.

PROGRAM OUTCOMES

1. To apply knowledge of computing and mathematics appropriate to the domain.
2. To logically define, analyse and solve real world problems.
3. To apply design principles in developing hardware/software systems of varying complexity that meet the specified needs.
4. To interpret and analyse data for providing solutions to complex engineering problems.
5. To use and practise engineering and IT tools for professional growth.
6. To understand and respond to legal and ethical issues involving the use of technology for societal benefits.
7. To develop societal relevant projects using available resources.
8. To exhibit professional and ethical responsibilities.
9. To work effectively as an individual and a team member within the professional environment.
10. To prepare and present technical documents using effective communication skills.
11. To demonstrate effective leadership skills throughout the project management life cycle.
12. To understand the significance of lifelong learning for professional development.

General Instructions

GENERAL INSTUCTIONS:

- Equipment in the lab is meant for the use of students. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care.
- Students are required to carry their reference materials, files and records with completed assignment while entering the lab.

- Students are supposed to occupy the systems allotted to them and are not supposed to talk or make noise in the lab.
- All the students should perform the given assignment individually.
- Lab can be used in free time/lunch hours by the students who need to use the systems should take prior permission from the lab in-charge.
- All the Students are instructed to carry their identity cards when entering the lab.
- Lab files need to be submitted on or before date of submission.
- Students are not supposed to use pen drives, compact drives or any other storage devices in the lab.
- For Laboratory related updates and assignments students should refer to the notice board in the Lab.

.

COURSE NAME: DATABASE MANAGEMENT SYSTEM

WEEKLY PLAN:

Week No.	Practical/Assignment Name	Problem Definition
1	PLSQL	Introduction to PLSQL, PLSQL Language Features,
2	PLSQL Program structure	Implementation of PLSQL Block with Variables, Datatypes, Control and Loop Structures, Operators, Functions, Procedures, Cursors and Triggers
3	PLSQL functions and Procedures	Implementation of PLSQL: Functions, Procedures, Cursors, and Triggers
4	ER Model, Mapping to Relations	Introduction to ER model and its mapping to Relational Schema
5	Normalization	Database Normalization
6	SQL Commands	Introduction to SQL and Implementation of DDL, DML, DCL Commands
7	Arithmetic and Set Operators with SQL	Implementation of Arithmetic and Set operations using SQL.
8	Aggregate Functions with SQL	Demonstration of SQL queries on Aggregate Functions
9	SQL subqueries and Nested Queries	Implementation of SQL subqueries and Nested queries
10	SQL Joins and Views	Implementation of SQL joins and views.
11	Case studies	Case studies

Examination Scheme

Practical Exam: 25 Marks
Term Work: 25 Marks
Total: 50 Marks
Minimum Marks required: 20 Marks

PROCEDURE OF EVALUATION

Each practical/assignment shall be assessed continuously on the scale of 25 marks. The distribution of marks as follows.

Sr. No	Evaluation Criteria	Marks for each Criteria	Rubrics
1	**Timely Submission**	**07**	➢ Punctuality reflects the work ethics. Students should reflect that work ethics by completing the lab assignments and reports in a timely manner without being reminded or warned.
2	**Presentation**	**06**	➢ Student are expected to write the technical document (lab report) in their own words. The presentation of the contents in the lab report should be complete, unambiguous, clear, understandable. The report should document approach/algorithm/design and code with proper explanation.
3	**Understanding**	**12**	➢ Correctness and Robustness of the code is expected. The Learners should have an in-depth knowledge of the practical assignment performed. The learner should be able to explain methodology used for designing and developing the program/solution. He/she should clearly understand the purpose of the assignment and its outcome.

LABORATORY USAGE–

Students use *Oracle 10g* Software to execute SQL queries on computers for executing the lab experiments, document the results and to prepare the technical documents for making the lab reports.

OBJECTIVES

1. To provide a strong formal foundation in database concepts, technology and practice to the students to groom them into well-informed database application developers.

2. To create a database and query it is using SQL, design forms and generate reports.
3. Understand the significance of integrity constraints, referential integrity constraints, triggers, assertions.
4. To make the students more practical oriented and also to pave the way for using SQL programming language for real time business applications.

PRACTICAL PRE-REQUISITE

1. Knowledge of Programming Languages
2. Discrete mathematics and Data structures.

SOFTWARE REQUIREMENT

- Operating System: Windows XP/7 3
- Front end VB/VC ++/JAVA
- Back end Oracle10g, my SQL, myAccess

COURSE OUTCOMES:

1. Differentiate significance of Database Management System over the file processing system.
2. Illustrate the fundamentals of data models and to conceptualize and depict a database system using data models.
3. Analyze and practice Relational Data Model.
4. Apply SQL queries for database definition and database manipulation.
5. Illustrate transaction management concepts like serializability, concurrency control and recovery system.
6. Investigate the knowledge about emerging trends in the area of database for unstructured data and applications for it

HOW OUTCOMES ARE ASSESSED?

Sr.No.	Outcome	Assignment Number	Level	Proficiency evaluated by
1	Differentiate significance of Database Management System over the file processing system.	3,4,5,6,7,8	3, 3, 3, 3, 3, 3	Problem definition Performing Practical and reporting results
2	Illustrate the fundamentals of data models and to conceptualize and depict a database system using data models.	2,3,4,5,6,7,8	3, 1, 1, 1, 1, 1, 1	Problem definition Performing Practical and reporting results
3	Analyze and practice Relational Data Model.	1,2,3,4,5,6,7,8	3, 3, 3, 3, 3, 3, 3	Problem definition Performing Practical and reporting results
4	Apply SQL queries for database definition and database manipulation.	1,2,3,4,5,6,7,8	3, 3, 3, 3, 3, 3, 3	Problem definition Performing Practical and reporting results
5	Illustrate transaction management concepts like serializability, concurrency control and recovery system.	1,2,3,4,5,6,7,8	2, 2, 2, 2, 2, 2, 2	Documentation.
6	Investigate the knowledge about emerging trends in the area of database for unstructured data and applications for it	9,10	3, 3	Documentation.

CONTRIBUTION TO PROGRAM OUTCOMES

CO Statements	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2
Differentiate significance of Database Management System over the file processing system.	**2**	**3**	**3**		**3**	**3**	**3**	**3**	**3**	**2**	**2**	**3**		
Illustrate the fundamentals of data models and to conceptualize and depict a database system using data models.	**1**	**3**	**3**	**3**	**3**	**2**	**3**	**3**	**3**	**2**	**2**	**3**	3	1
Analyze and practice Relational Data Model.	**2**	**3**	**3**	**2**	**3**	**3**	**3**	**3**	**3**	**2**	**2**	**3**	1	
Apply SQL queries for database definition and database manipulation.	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	3	

Illustrate transaction management concepts like serializability, concurrency control and recovery system.	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**		3
Investigate the knowledge about emerging trends in the area of database for unstructured data and applications for it	**2**	**1**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**		3

DESIGN EXPERIENCE GAINED

Students will be able to design and develop normalized database and report generation for real time applicationse.g.banking, reservation system, etc.

LEARNING EXPERIENCE GAINED

After undergoing this laboratory, the Student should be able to:

1. Understand, appreciate, and effectively explain the underlying concepts of database technologies
2. Design and implement a database schema for a given problem-domain
3. Normalize a database
4. Populate and query a database using SQL DML/DDL commands.
5. Declare and enforce integrity constraints on a database using a state-of-the-art RDBMS.
6. Programming PL/SQL including stored procedures, stored functions, cursors, packages

List of Practical Assignments

No	Detail of Assignment
1.	Write a PL/SQL block for a. Display prime numbers from 1 to 10. b. Calculate area of circle. c. Find reverse of given number. d. Display Factorial of given number. e. Display record in database table.
2.	Construct an Entity Relationship(ER) Model for given database and also normalized it.
3.	Implement various DDL, DML and DCL commands in SQL
4.	Implement Arithmetic and Set operations with SQL queries.
5.	Demonstrate SQL queries on aggregate functions.
6.	Implement SQL Sub queries and Nested queries.
7.	Demonstrate SQL queries on joins.
8.	Demonstrate the use of views in SQL.
9.	Case study on Data mining tool: WEKA tool.
10.	Case study on NOSQL database: MongoDB.

AssignmentNo.1

Title: Write a PL/SQL block for

a. Display prime numbers from 1 to 10.
b. Calculate area of circle.
c. Find reverse of given number.
d. Display Factorial of given number.
e. Display record in database table.

Aim: To practice and implement PLSQL blocks

Theory:PL/SQL is a combination of SQL along with the procedural features of programming languages. PLSQL programming elements as follows:

PLSQL variables and constants:PL/SQL allocates memory for the variable's value and the storage location is identified by the variable name.

```
counter binary_integer: = 0;
greetings varchar2(20) DEFAULT 'Have a Good Day';

PI CONSTANT NUMBER: = 3.141592654;
```

PLSQL Conditional and looping structures

1.IF THEN statment

```
IF condition THEN
   S;
END IF;
```

2 *IF-THEN-ELSE statement*

```
IF condition THEN
   S1;
ELSE
   S2;
END IF;
```

3.*IF-THEN-ELSIF statement*

It allows you to choose between several alternatives.

```
IF(boolean_expression 1)THEN
   S1; -- Executes when the boolean expression 1 is true
ELSIF( boolean_expression 2) THEN
```

```
   S2;  -- Executes when the boolean expression 2 is true
ELSIF( boolean_expression 3) THEN
   S3; -- Executes when the boolean expression 3 is true
ELSE
   S4; -- executes when the none of the above condition is true
END IF;
```

4.*Case statement*

Like the IF statement, the CASE statement selects one sequence of statements to execute.

```
CASE selector
   WHEN 'value1' THEN S1;
   WHEN 'value2' THEN S2;
   WHEN 'value3' THEN S3;
   ...
   ELSE Sn;  -- default case
END CASE;
```

5.*Searched CASE statement*

```
CASE
   WHEN selector = 'value1' THEN S1;
   WHEN selector = 'value2' THEN S2;
   WHEN selector = 'value3' THEN S3;
   ...
   ELSE Sn;  -- default case
END CASE;
```

6.*nested IF-THEN-ELSE*

```
IF( boolean_expression 1)THEN
   -- executes when the boolean expression 1 is true
   IF(boolean_expression 2) THEN
      -- executes when the boolean expression 2 is true
      sequence-of-statements;
   END IF;
ELSE
   -- executes when the boolean expression 1 is not true
   else-statements;
END IF;
```

Looping Structures

1.PL/SQL Basic LOOP

```
LOOP
   Sequence of statements;
END LOOP;
```

2.PL/SQL WHILE LOOP

```
WHILE condition LOOP
   sequence_of_statements
```

```
END LOOP;
```

3.PL/SQL FOR LOOP

```
FOR counter IN initial_value .. final_value LOOP
   sequence_of_statements;
END LOOP;
```

4.NESTED LOOPS IN PL/SQL

```
LOOP
   Sequence of statements1
   LOOP
      Sequence of statements2
   END LOOP;
END LOOP;
```

PLSQL comments: The PL/SQL single-line comments start with the delimiter -- (double hyphen) and multi-line comments are enclosed by /* and */.

PLSQL Procedures

A procedure is created with the CREATE OR REPLACE PROCEDURE statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows −

```
CREATE [OR REPLACE] PROCEDURE procedure_name

[(parameter_name [IN | OUT | IN OUT] type [, ...])]

{IS | AS}

BEGIN

< procedure_body >

END procedure_name;
```

DROP PROCEDURE procedure-name;

PLSQL Functions

A standalone function is created using the CREATE FUNCTION statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
< function_body >
```

```
END [function_name];
```

PLSQL cursor:

A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement.

- Implicit cursors
- Explicit cursors

In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has attributes such as **%FOUND, %ISOPEN, %NOTFOUND**, and **%ROWCOUNT**.

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE customers
  SET salary = salary +500;
  IF sql%notfound THEN
    dbms_output.put_line('no customers selected');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows ||' customers selected ');
END IF;
END;
/
```

Explicit cursor

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example

```
CURSOR c_customers IS
```

```
SELECT id, name, address FROM customers;
```

Opening the Cursor

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows −

```
OPEN c_customers;
```

Fetching the Cursor

Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows −

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

Closing the Cursor

Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows −

```
CLOSE c_customers;
```

Example:

```
DECLARE
   c_id customers.id%type;
   c_name customerS.No.ame%type;
   c_addr customers.address%type;
   CURSOR c_customers is
      SELECT id, name, address FROM customers;
BEGIN
   OPEN c_customers;
   LOOP
   FETCH c_customers into c_id, c_name, c_addr;
      EXIT WHEN c_customers%notfound;
      dbms_output.put_line(c_id ||' '|| c_name ||' '|| c_addr);
END LOOP;
   CLOSE c_customers;
END;
```

PL/SQL Triggers

Triggers are stored programs, which are automatically executed or fired when some events occur.

- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

The syntax for creating a trigger is −

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR]| UPDATE [OR]| DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
Declaration-statements
BEGIN
Executable-statements
```

Basic Syntax PLSQL Block:

```
DECLARE
        <declarations section>
BEGIN
<executable command(s)>
EXCEPTION
<exception handling>
END;
```

Sample Code:

```
DECLARE
  -- variable declaration
  message varchar2(20): = 'Hello, World!';
BEGIN
  /*
  *  PL/SQL executable statement(s)
  */
  dbms_output.put_line(message);
END;
```

/

Same structure will be used for writing PLSQL Blocks to calculate area, find reverse, display prime numbers etc. Students can refer theory given above and write PLSQL block for given. Only logic will change.

Assignment No.2

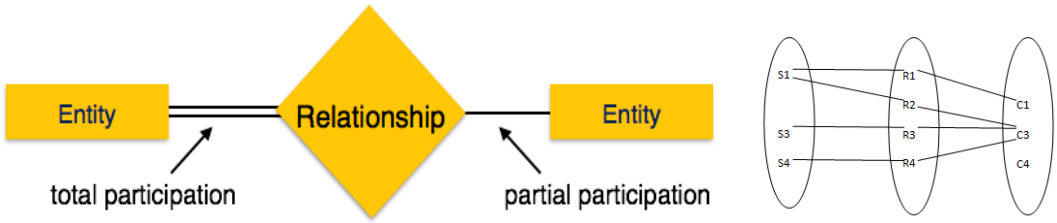
Title: Construct an Entity Relationship(ER) Model for given database and also normalized it.

Aim: Toconstruct an ER model for given system and normalized to Relational Schema.

Theory: Entity Relationship Model(ER)

An entity–relationship model (ER model) is a systematic way of describing and defining a business process. An ER model is typically implemented as a database. The main components of E-R model are: entity set and relationship set.

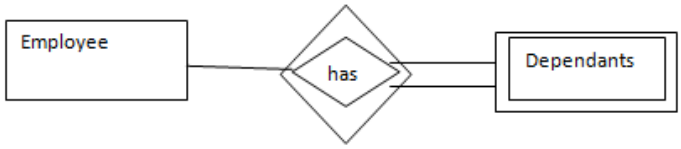
- Here are the geometric shapes and their meaning in an E-R Diagram –
- Rectangle: Represents Entity sets.
- Ellipses: Attributes
- Diamonds: Relationship Set (Unary, Binary and Ternary)
- Lines: They link attributes to Entity Sets and Entity sets to Relationship Set
- Double Ellipses: Multivalued Attributes
- Dashed Ellipses: Derived Attributes
- Double Rectangles: Weak Entity Sets
- Double Lines: Total participation of an entity in a relationship set
- A Total participation of an entity set represents that each entity in entity set must have at least one relationship in a relationship set. And if not then that is partial participation



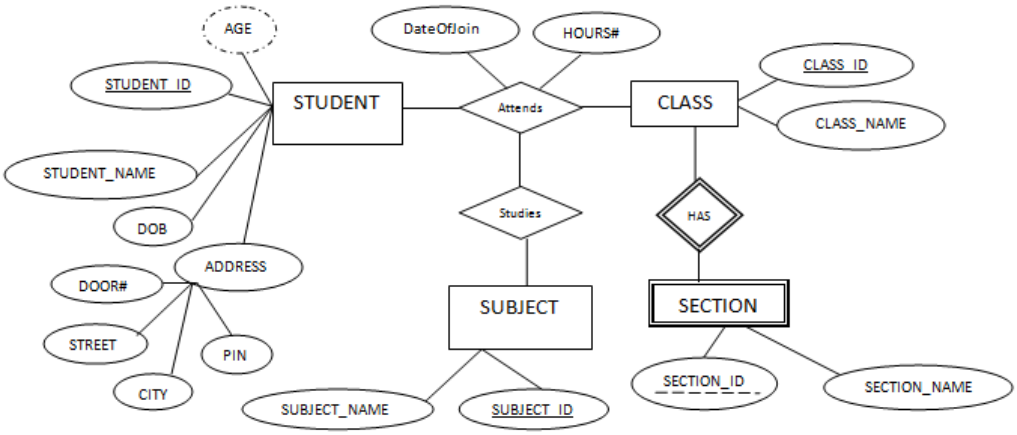
- Multivalued Attributes: An attribute that can hold multiple values is known as multivalued attribute. We represent it with double ellipses in an E-R Diagram. E.g. A person can have more than one phone numbers so the phone number attribute is multivalued.
- Derived Attribute: A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by dashed ellipses in an E-R Diagram. E.g. Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).
- Cardinality
  - One to One: When only one instance of an entity is associated with the relationship,
  - One to Many:When more than one instance of an entity is associated with a relationship
  - Many to One:When more than one instance of entity is associated with the relationship,

- Many to many:The following image reflects that more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship.

- Weak Entity Type and Identifying Relationship
  As discussed before, an entity type has a key attribute which uniquely identifies each entity in the entity set. But there exists **some entity type for which key attribute can't be defined**. These are called Weak Entity type.



Sample ER Model:



Mapping of ER Model to Relational Model

- Build a table for each entity set
- Build a table for each relationship set if necessary (more on this later)
- Make a column in the table for each attribute in the entity set
- Indivisibility Rule and Ordering Rule
- Primary Key

Transforming Entities.

- The figure shown below shows the CUSTOMER entity.

- Transform this entity to a relational table structure by creating a single table with a column for each attribute and the primary key attribute underlined as the primary key column for the table.

Transforming Entities with Multivalued Attributes.

The solution is to remove the multivalued attribute to a separate table along with the primary key (to link back to the original table). The attribute becomes part of a composite key.

Representation of Weak Entity Set

- Weak Entity Set Cannot exists alone
- To build a table/schema for weak entity set
- Construct a table with one column for each attribute in the weak entity set
- Remember to include discriminator
- Augment one extra column on the right side of the table, put in there the primary key of the Strong Entity Set (the entity set that the weak entity set is depending on)
- Primary Key of the weak entity set = Discriminator + foreign key

Representing Relationship Set Unary/Binary Relationship

1. *For one-to-one relationship w/out total participation*

Build a table with two columns, one column for each participating entity set's primary key. Add successive columns, one for each descriptive attributes of the relationship set (if any).

2. *For one-to-one relationship with one entity set having total participation*

Augment one extra column on the right side of the table of the entity set with total participation, put in there the primary key of the entity set without complete participation as per to the relationship.

3. *For one-to-many relationship w/out total participation*

Same thing as one-to-one for one-to-many/many-to-one relationship with one entity set having total participation on "many" side

Augment one extra column on the right side of the table of the entity set on the "many" side, put in there the primary key of the entity set on the "one" side as per to the relationship.

4. *For many-to-many relationship*

Same thing as one-to-one relationship without total participation. Primary key of this new schema is the union of the foreign keys of both entity sets. No augmentation approach possible…

Database Normalization

Database normalization is the process of efficiently organizing data in a database. There are two reasons of the normalization process:

- Eliminating redundant data, for example, storing the same data in more than one table.
- Ensuring data dependencies make sense.

Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored. Normalization consists of a series of guidelines that help guide you in creating a good database structure. Normalization guidelines are divided into normal forms; think of form as the format or the way a database structure is laid out. The aim of normal forms is to organize the database structure so that it complies with the rules of first normal form, then second normal form, and finally third normal form. It's your choice to take it further and go to fourth normal form, fifth normal form, and so on, but generally speaking, third normal form is enough.

1. First Normal Form (1NF)
2. Second Normal Form (2NF)
3. Third Normal Form (3NF).
4. Boyce-Codd Normal Form (BCNF

First Normal Form

First normal form (1NF) sets the very basic rules for an organized database:

1. Define the data items required, because they become the columns in a table. Place related data items in a table.
2. Ensure that there are no repeating groups of data.
3. Ensure that there is a primary key.

First Rule of 1NF:

You must define the data items. This means looking at the data to be stored, organizing the data into columns, defining what type of data each column contains, and finally putting related columns into their own table. For example, you put all the columns relating to locations of meetings in the Location table, those relating to members in the MemberDetails table, and so on.

Second Rule of 1NF:

The next step is ensuring that there are no repeating groups of data. Consider we have the following table

```
CREATE TABLE CUSTOMERS(
                ID INT NOT NULL,
                NAME VARCHAR (20) NOT NULL,
                AGE INT NOT NULL,
                ADDRESS CHAR (25),
                ORDERS VARCHAR(155)
                );
```

ID	NAME	AGE	ADDRESS	ORDERS
100	Sachin	36	Lower West Side	Cannon XL-200
100	Sachin	36	Lower West Side	Battery XL-200
100	Sachin	36	Lower West Side	Tripod Large

But as per 1NF, we need to ensure that there are no repeating groups of data. So let us break above table into two parts and join them using a key as follows:

CUSTOMERS table:

```
CREATE TABLE CUSTOMERS(
ID INT NOT NULL,
NAME VARCHAR (20) NOT NULL,
AGE INT NOT NULL,
ADDRESS CHAR (25),
PRIMARY KEY (ID)
);
```

This table would have the following record:

ID	NAME	AGE	ADDRESS
100	Sachin	36	Lower West Side

ORDERS table:

```
CREATE TABLE ORDERS(
ID INT NOT NULL,
CUSTOMER_ID INT NOT NULL,
ORDERS VARCHAR(155),

PRIMARY KEY (ID)
);
```

This table would have the following records

ID	CUSTOMER_ID	ORDERS

10	100	Cannon XL-200
11	100	Battery XL-200
12	100	Tripod Large

Third Rule of 1NF:
The final rule of the first normal form, create a primary key for each table which we have already created.

Second Normal Form
Second normal form states that it should meet all the rules for 1NF and there must be no partial dependences of any of the columns on the primary key: Consider a customer-order relation and you want to store customer ID, customer name, order ID and order detail, and date of purchase:

```
CREATE TABLE CUSTOMERS (
CUST_ID INT NOT NULL,
CUST_NAME VARCHAR (20) NOT NULL,
ORDER_ID INT NOT NULL,
ORDER_DETAIL VARCHAR (20) NOT NULL,
SALE_DATE DATETIME,
PRIMARY KEY (CUST_ID, ORDER_ID)
);
```

This table is in first normal form, in that it obeys all the rules of first normal form. In this table, the primary key consists of CUST_ID and ORDER_ID. Combined, they are unique assuming same customer would hardly order same thing. However, the table is not in second normal form because there are partial dependencies of primary keys and columns. CUST_NAME is dependent on CUST_ID, and there's no real link between a customer's name and what he purchased. Order detail and purchase date are also dependent on ORDER_ID, but they are not dependent on CUST_ID, because there's no link between a CUST_ID and an ORDER_DETAIL or their SALE_DATE. To make this table comply with second normal form, you need to separate the columns into three tables. First, create a table to store the customer details as follows:

```
CREATE TABLE CUSTOMERS (
CUST_ID INT NOT NULL,
CUST_NAME VARCHAR (20) NOT NULL,
PRIMARY KEY (CUST_ID)
);
```

Next, create a table to store details of each order:

```
CREATE TABLE ORDERS (
ORDER_ID INT NOT NULL,
ORDER_DETAIL VARCHAR (20) NOT NULL,
PRIMARY KEY (ORDER_ID)
```

```
);
```

Finally, create a third table storing just CUST_ID and ORDER_ID to keep track of all the orders for a customer:

```
CREATE TABLE CUSTMERORDERS (
CUST_ID INT NOT NULL,
ORDER_ID INT NOT NULL,
SALE_DATE DATETIME,
PRIMARY KEY (CUST_ID, ORDER_ID)
);
```

Third Normal Form

A table is in third normal form when the following conditions are met:

1. It is in second normal form.
2. All nonprimary fields are dependent on the primary key.

The dependency of nonprimary fields is between the data. For example, in the below table, street name, city, and state are unbreakably bound to the zip code.

```
CREATE TABLE CUSTOMERS (
CUST_ID INT NOT NULL,
CUST_NAME VARCHAR (20) NOT NULL,
DOB DATE,
STREET VARCAR (200),
```

```
CITY VARCHAR(100),
STATE VARCHAR(100),
ZIP VARCHAR(12),
EMAIL_ID VARCHAR(256),
PRIMARY KEY (CUST_ID)
);
```

The dependency between zip code and address is called a transitive dependency. To comply with third normal form, all you need to do is move the Street, City, and State fields into their own table, which you can call the Zip Code table:

```
CREATE TABLE ADDRESS(
ZIP VARCHAR(12),
STREET VARCHAR(200),
CITY VARCHAR(100),
STATE VARCHAR(100),
PRIMARY KEY (ZIP)
);
```

Next, alter the CUSTOMERS table as follows:

```
CREATE TABLE CUSTOMERS(
CUST_ID INT NOT NULL,
CUST_NAME VARCHAR (20) NOT NULL,
DOB DATE,
ZIP VARCHAR(12),
EMAIL_ID VARCHAR(256),
PRIMARY KEY (CUST_ID)
);
```

The advantages of removing transitive dependencies are mainly twofold. First, the amount of data duplication is reduced and therefore your database becomes smaller. The second advantage is data integrity. When duplicated data changes, there's a big risk of updating only some of the data, especially if it's spread out in a number of different places in the database. For example, if address and zip code data were stored in three or four different tables, then any changes in zip codes would need to ripple out to every record in those three or four tables.

Boyce-Codd Normal Form (BCNF)

When a table has more than one candidate key, anomalies may result even though the relation is in 3NF. Boyce-Codd normal form is a special case of 3NF. A relation is in BCNF if and only if every determinant is a candidate key.

Assignment No.3

Title: Implement various DDL, DML and DCL commands in SQL

Aim: Implementation of SQL queries.

Theory:

SQL Constraints: Constraints are the rules enforced on data columns on table

1. NOT NULL Constraint: Ensures that a column cannot have NULL value.

```
CREATE TABLE CUSTOMERS(
ID INT NOT NULL,
NAME VARCHAR (20) NOT NULL,
AGE INT NOT NULL,
ADDRESS CHAR (25) ,
SALARY DECIMAL (18, 2),
PRIMARY KEY (ID)
);
```

2. DEFAULT Constraint: Provides a default value for a column when none is specified.

```
CREATE TABLE CUSTOMERS (
ID INT NOT NULL,
NAME VARCHAR (20) NOT NULL,
AGE INT NOT NULL,
ADDRESS CHAR (25),
SALARY DECIMAL (18, 2) DEFAULT 5000.00,
PRIMARY KEY (ID)
);
```

3. UNIQUE Constraint: Ensures that all values in a column are different.

```
CREATE TABLE CUSTOMERS(
INT NOT NULL,
NAME VARCHAR (20) NOT NULL,
AGE ID INT NOT NULL UNIQUE,
ADDRESS CHAR (25) ,
SALARY DECIMAL (18, 2),
PRIMARY KEY (ID)
);
```

4. PRIMARY Key: Uniquely identified each rows/record in a database table.

```
CREATE TABLE CUSTOMERS(
```

```
ID INT NOT NULL,
NAME VARCHAR (20) NOT NULL,
AGE INT NOT NULL,
ADDRESS CHAR (25) ,
SALARY DECIMAL (18, 2),
PRIMARY KEY (ID)
);
```

5. FOREIGN Key: Uniquely identified a rows/records in any another database table.

```
CUSTOMERS table:
CREATE TABLE CUSTOMERS(
ID INT NOT NULL,
NAME VARCHAR (20) NOT NULL,
AGE INT NOT NULL,
ADDRESS CHAR (25) ,
SALARY DECIMAL (18, 2),
PRIMARY KEY (ID)
);
ORDERS table:
CREATE TABLE ORDERS (
ID INT NOT NULL,
DATE DATETIME,
CUSTOMER_ID INT references CUSTOMERS(ID),
AMOUNT double,
PRIMARY KEY (ID)
);
```

6. CHECK Constraint: The CHECK constraint ensures that all values in a column satisfy certain conditions.

```
CREATE TABLE CUSTOMERS(
ID INT NOT NULL,
NAME VARCHAR (20) NOT NULL,
AGE INT NOT NULL CHECK (AGE >= 18),
ADDRESS CHAR (25) ,
SALARY DECIMAL (18, 2),
PRIMARY KEY (ID)
);
```

7. INDEX: Use to create and retrieve data from the database very quickly.

```
CREATE TABLE CUSTOMERS(
ID INT NOT NULL,
NAME VARCHAR (20) NOT NULL,
AGE INT NOT NULL,
ADDRESS CHAR (25) ,
SALARY DECIMAL (18, 2),
PRIMARY KEY (ID)
```

);

Now, you can create index on single or multiple columns using the following syntax:

CREATE INDEX index_name
ON table_name (column1, column2.....);

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into groups based on their nature:

DDL -Data Definition Language: (DDL commands)	**Description**
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.
Data Manipulation Language: (DML commands)	**Description**
INSERT	Creates a record
UPDATE	Modifies records
DELETE	Deletes records
DCL -Data Control Language: (DCL Commands)	**Description**
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user
Data Query Language (DQL commands)	**Description**
SELECT	Retrieves certain records from one or more tables

SQL DDL DML DCL DQL Commands Syntax

- *SQL SELECT Statement:*

  SELECT column1, column2....columnN
  FROM table_name;

- *SQL DISTINCT Clause:*

  SELECT DISTINCT column1, column2....columnN
  FROM table_name;

- *SQL WHERE Clause:*

```
SELECT column1, column2....columnN
FROM table_name
WHERE CONDITION;
```

- *SQL AND/OR Clause:*

```
SELECT column1, column2....columnN
FROM table_name
WHERE CONDITION-1 {AND|OR} CONDITION-2;
```

- *SQL IN Clause:*

```
SELECT column1, column2....columnN
FROM table_name
WHERE column_name IN (val-1, val-2,...val-N);
```

- *SQL BETWEEN Clause:*

```
SELECT column1, column2....columnN
FROM table_name
WHERE column_name BETWEEN val-1 AND val-2;
```

- *SQL LIKE Clause:*

```
SELECT column1, column2....columnN
FROM table_name
WHERE column_name LIKE { PATTERN };
```

- *SQL ORDER BY Clause:*

```
SELECT column1, column2....columnN
FROM table_name
WHERE CONDITION
ORDER BY column_name {ASC|DESC};
```

- *SQL GROUP BY Clause:*

```
SELECT SUM(column_name)
FROM table_name
WHERE CONDITION
GROUP BY column_name;
```

- *SQL COUNT Clause:*

```
SELECT COUNT(column_name)
FROM table_name
WHERE CONDITION;
```

- *SQL HAVING Clause:*

```
SELECT SUM(column_name)
FROM table_name
WHERE CONDITION
```

GROUP BY column_name
HAVING (arithematic function condition);

- *SQL CREATE TABLE Statement:*
  CREATE TABLE table_name(column1 datatype,
  column2 datatype,
  column3 datatype,
  .....
  columnN datatype,
  PRIMARY KEY(one or more columns)
  );

- *SQL DROP TABLE Statement:*
  DROP TABLE table_name;

- *SQL CREATE INDEX Statement:*
  CREATE UNIQUE INDEX index_name
  ON table_name (column1, column2,...columnN);

- *SQL DROP INDEX Statement:*
  ALTER TABLE table_name
  DROP INDEX index_name;

- *SQL DROP TABLE Statement:*

  DROP TABLE table_name;
- *SQL CREATE INDEX Statement:*
  CREATE UNIQUE INDEX index_name
  ON table_name (column1, column2,...columnN);

- *SQL DROP INDEX Statement:*
  ALTER TABLE table_name
  DROP INDEX index_name;

- *SQL DESC Statement:*
  DESC table_name;
- *SQL TRUNCATE TABLE Statement:*
  TRUNCATE TABLE table_name;

- *SQL ALTER TABLE Statement:*
  ALTER TABLE table_name {ADD|DROP|MODIFY}
  column_name {data_ype};

- *SQLALTER TABLE Statement (Rename):*
  ALTER TABLE table_name RENAME TO new_table_name;

- *SQL INSERT INTO Statement:*

  INSERT INTO table_name(column1, column2....columnN)
  VALUES (value1, value2....valueN);

- *SQL UPDATE Statement:*
  UPDATE table_name SET column1 = value1, column2 = value2....columnN=valueN
  [WHERE CONDITION];

- *SQL DELETE Statement:*
  DELETE FROM table_name
  WHERE {CONDITION};

- *SQL CREATE DATABASE Statement:*
  CREATE DATABASE database_name;

- *SQL DROP DATABASE Statement:*
  DROP DATABASE database_name;

- *SQL USE Statement:*
  USE DATABASE database_name;

- *SQL COMMIT Statement:*
  COMMIT;

- *SQL ROLLBACK Statement:*
- *ROLLBACK;*

- *DISTINCT*
  SELECT DISTINCT column1, column2,.....columnN
  FROM table_name
  WHERE [condition

Assignment No.4

Title: Implement Arithmetic and Set operations with SQL queries.

Aim: Implementation of SQL queries with arithmetic and set Operations.

Theory:

Arithmetic Operators (+,,-,/, %, mod)*

All arithmetic operators and Expressions can be used in SQL query as follows

Select salary+increment as Sal from table_name;

Set Operations

UNION combines the result sets of two queries. Column data types in the two queries must match. UNION combines by column position rather than column name

UNION ALL
The UNION ALL set operator returns all rows selected by either query. That means any duplicates will remain in the final result set.

INTERSECT
The INTERSECT set operator returns all distinct rows selected by both queries. That means only those rows common to both queries will be present in the final result set.

MINUS
The MINUS set operator returns all distinct rows selected by the first query but not the second. This is functionally equivalent to the ANSI set operator EXCEPT DISTINCT.

Sample queries

```
SELECT product_id FROM order_items
UNION
SELECT product_id FROM inventories
ORDER BY product_id;
```

```
SELECT department_id, department_name
FROM   departments
WHERE  department_id <= 30
```

```
UNION ALL
SELECT department_id, department_name
FROM   departments
WHERE  department_id >= 20
ORDER BY 1;
```

```
SELECT department_id, department_name
FROM   departments
WHERE  department_id <= 30
INTERSECT
SELECT department_id, department_name
FROM   departments
WHERE  department_id >= 20
ORDER BY 1;
```

```
SELECT product_id FROM inventories
MINUS
SELECT product_id FROM order_items
ORDER BY product_id
```

AssignmentNo.5

Title: Demonstrate SQL queries on aggregate functions

Aim: Implementation of SQL queries on Aggregate Function.

Theory: Aggregate functions return a single value based on groups of rows, rather than single value for each row. You can use Aggregate functions in select lists and in ORDER BY and HAVING clauses. They are commonly used with the GROUP BY clause in a SELECT statement, where Oracle divides the rows of a queried table or view into groups.

MIN	returns the smallest value in a given column
MAX	returns the largest value in a given column
SUM	returns the sum of the numeric values in a given column
AVG	returns the average value of a given column
COUNT	returns the total number of values in a given column
COUNT(*)	returns the number of rows in a table

Sample code with syntax

- *SELECT AVG(salary)*
  FROM employee;

- *SELECT Count(*)*
  FROM employee;

- *SELECT AVG(MAX(salary))*
  FROM employees
  GROUP BY department_id;

- *SELECT AVG(salary)*
  FROM employee;

- *SELECT min(sal) "Min Salary" FROM emp;*

Assignment No.6

Title: Implement SQL Sub queries and Nested queries.

Aim: Implementation of SQL Sub queries and Nested queries

Theory:

Sql Subqueries:

- A subquery is a SQL query within a query.
- Subqueries are nested queries that provide data to the enclosing query.
- Subqueries can return individual values or a list of records
- Subqueries must be enclosed with parenthesis

here is no general syntax; subqueries are regular queries placed inside parenthesis. Subqueries can be used in different ways and at different locations inside a query: Here is a subquery with the IN operator

Use of WHERE ANY, ALL Clause in SQL Subqueries, and Nested Queries

1) ANY and ALL keywords are used with a WHERE or HAVING clause.
2) ANY and ALL operate on subqueries that return multiple values.
3) ANY returns true if any of the subquery values meet the condition.
4) ALL returns true if all of the subquery values meet the condition

A SQL nested query is a SELECT query that is nested inside a SELECT, UPDATE, INSERT, or DELETE SQL query. Here is a simple example of SQL nested query:

```
SELECT Model FROM Product
WHERE ManufacturerID IN (SELECT ManufacturerID
 FROM Manufacturer
 WHERE Manufacturer = 'Dell')
```

Sample Code:

```
SELECT column-names
 FROM table-name1
WHERE value IN (SELECT column-name
                FROM table-name2
               WHERE condition)
```

Subqueries can also assign column values for each record:

```
SELECT column1 = (SELECT column-name FROM table-name WHERE
condition),
      column-names
  FROM table-name
 WEHRE condition
```

Problem: List products with order quantities greater than 100.

```
SELECT ProductName
  FROM Product
 WHERE Id IN (SELECT ProductId
              FROM OrderItem
                    WHERE Quantity > 100)
```

Syntax of WHERE, ANY, ALL Clause Synatx

```
SELECT column-names
  FROM table-name
 WHERE column-name operator ANY
      (SELECT column-name
         FROM table-name
        WHERE condition)
```

```
SELECT column-names
  FROM table-name
 WHERE column-name operator ALL
      (SELECT column-name
         FROM table-name
        WHERE condition)
```

Assignment No.7

Title: Demonstrate SQL queries on joins.

Aim: Implementation of SQL queries on joins

Theory:

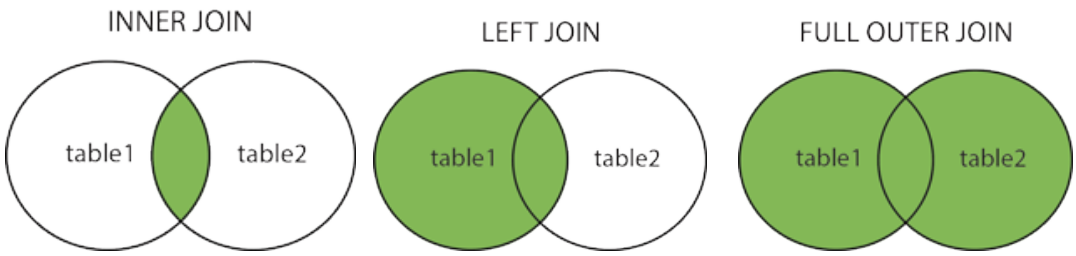
The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

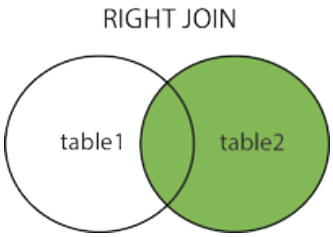
Different Types of SQL JOINs

1)SQL Equi joins
It is a simple sql join condition which uses the equal sign as the comparison operator. Two types of equi joins are SQL Outer join and SQL Inner join.
2) SQL Non Equi joins
It is a sql join condition which makes use of some comparison operator other than the equal sign like >, <, >=, <=

There are the different types of the JOINs in SQL:

- (INNER) JOIN: Returns records that have matching values in both tables
- LEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the right table
- RIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the left table
- FULL (OUTER) JOIN: Return all records when there is a match in either left or right table





The SQL Self JOIN syntax

- A self-JOIN occurs when a table takes a 'selfie'.
- A self-JOIN is a regular join, but the table is joined with itself.
- This can be useful when modeling hierarchies.
- They are also useful for comparisons within a table

```
SELECT column-names
  FROM table-name T1 JOIN table-name T2
  WHERE condition
```

T1 and T2 are different table aliases for the same table

Sample Codes:

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;

SELECT column-names
  FROM table-name1 LEFT JOIN table-name2
    ON column-name1 = column-name2
 WHERE condition

SELECT column-names
  FROM table-name1 RIGHT JOIN table-name2
    ON column-name1 = column-name2
 WHERE condition

SELECT column-names
  FROM table-name1 FULL JOIN table-name2
    ON column-name1 = column-name2
 WHERE condition
```

Assignment No.8

Title: Demonstrate the use of views in SQL.

Aim: Implementation of SQL Views.

Theory:

In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. A VIEW is a virtual table, through which a selective portion of the data from one or more tables can be seen. Views do not contain data of their own. They are used to restrict access to the database or to hide data complexity. A view is stored as a SELECT statement in the database. DML operations on a view like INSERT, UPDATE, DELETE affects the data in the original table upon which the view is based.

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

```
CREATE VIEW [Current Product List] AS
SELECT ProductID, ProductName
FROM Products
WHERE Discontinued = No;
```

```
CREATE OR REPLACE VIEW [Current Product List] AS
SELECT ProductID, ProductName, Category
FROM Products
WHERE Discontinued = No;
```

All DDL DML oprations that performed on table can also performed on Views.as Follows:

```
SELECT * FROM [Current Product List];
```

```
DROP VIEW view_name;
```

Inline Views

A common use for inline views in Oracle SQL is to simplify complex queries by removing join operations and condensing several separate queries into a single query. A subquery which is enclosed in parenthesis in the FROM clause may be given an alias name. The columns selected in the subquery can be referenced in the parent query, just as you would select from any normal table or view.

```
SELECT "column_name" FROM (Inline View);
```

Problem: Display the top five earner names and salaries from the EMPLOYEES table:

```
SELECT ROWNUM as RANK, last_name, salary
FROM (SELECT last_name, salary
    FROM employees
    ORDER BY salary DESC)
WHERE ROWNUM <= 5;
```

Assignment No.9

Title: Case study on Data mining tool: WEKA tool.

Aim: Detail study on WEKA tool

Requirements and Installation of Weka
We can install WEKA on Windows, MAC OS, and Linux. The minimum requirement is Java 8 or above for the latest stable versions of Weka.

As shown in the above screenshot, five options are available in the ***Applications*** category.

- The ***Explorer***is the central panel where most data mining tasks are performed. We will further explore this panel in upcoming sections.
- The tool provides an ***Experimenter*** In this panel, we can run experiments and also design them.
- WEKA provides the ***KnowledgeFlow*** panel. **It provides an interface to drag and drop components, connect them to form a knowledge flow and analyze the data and results.**
- The ***Simple CLI***panel provides the command line powers to run WEKA. For example, to fire up the ***ZeroR*** classifier on the ***arff*** data, we'll run from the command line:

java weka.classifiers.trees.ZeroR -t iris.arff

Weka Datatypes and Format of Data
Numeric (Integer and Real), String, Date, and Relational are the only four datatypes provided by WEKA. By default, WEKA supports the ARFF format. The ARFF, attribute-relation file format,

is an ASCII format that describes a list of instances sharing a set of attributes. Every ARFF file has two sections: header and data.

- The ***header*** section consists of attribute types,
- And the ***data*** section contains a comma-separated list of data for that attributes.

It is important to note that the declaration of the header (***@attribute***) and the declaration of the data (***@data***) are case-insensitive.

Let's look at the format with a weather forecast dataset:

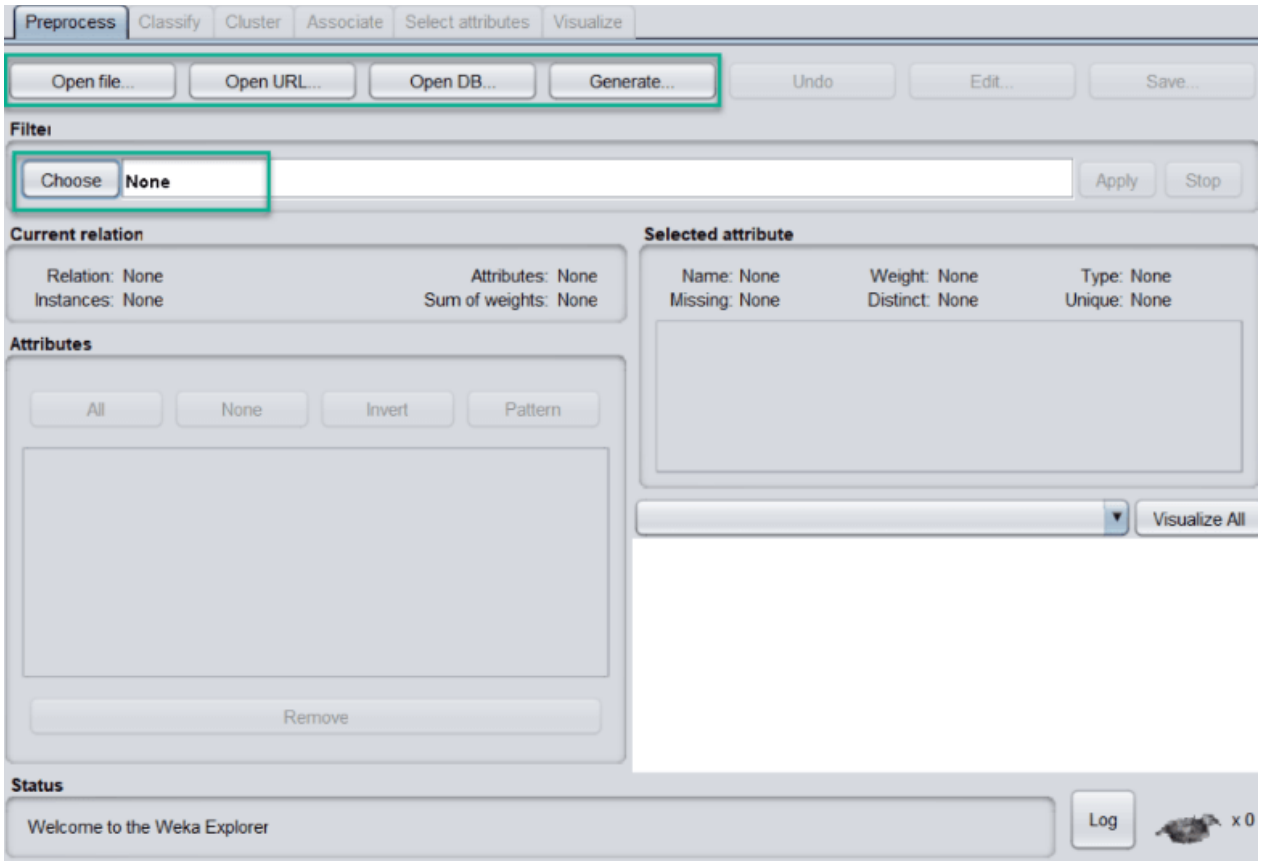
1. @attribute outlook {sunny,overcast,rainy}
2. @attribute tempreture {hot,mild,cool}
3. @attribute humidity {high,normal}
4. @attribute windy {TRUE,FALSE}
5. @attribute play {yes,no}
6.
7. @data
8. sunny,hot,high,FALSE,no
9. sunny,hot,high,TRUE,yes
10. overcast,hot,high,TRUE,yes
11. overcast,cool,normal,TRUE,yes
12. rainy,cool,normal,FALSE,no
13. rainy,cool,normal,TRUE,no

Besides ARFF, the tool supports different file formats such as **CSV, JSON**, and **XRFF**.

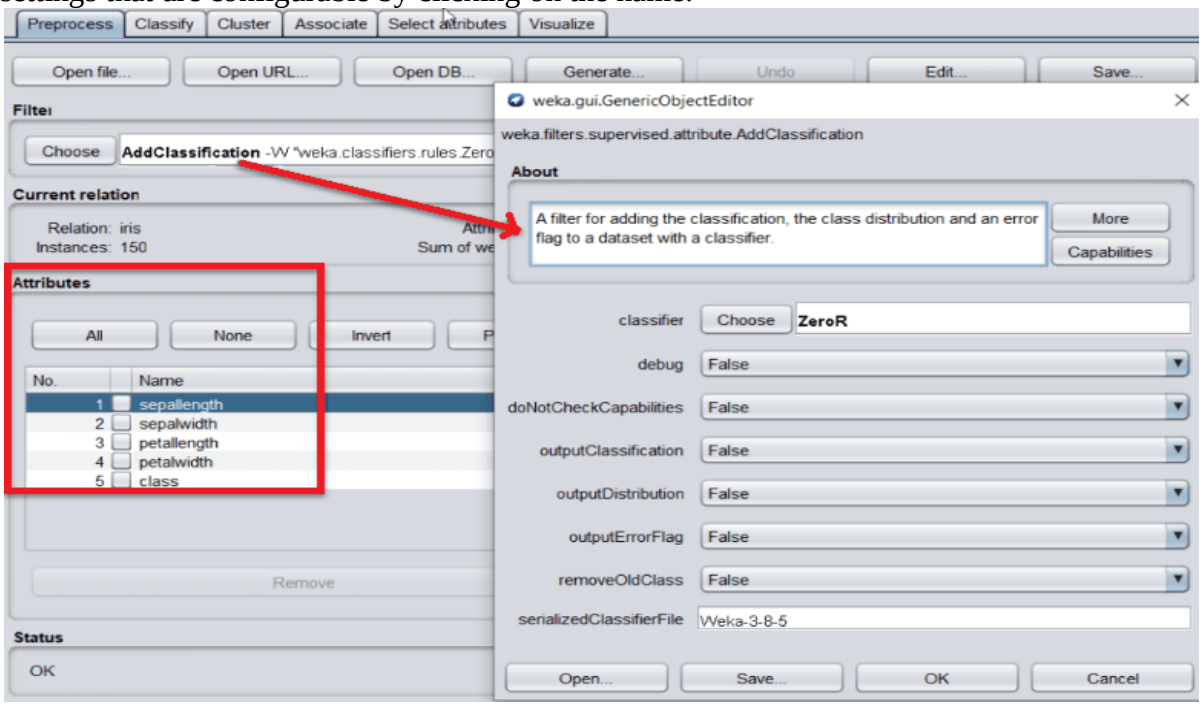
Loading of Data in Weka

WEKA allows you to load data from four types of sources:

1. The local file system
2. A public URL
3. Query to a database
4. Generate artificial data to run models



Once data is loaded from different sources, the next step is to preprocess the data. For this purpose, we can choose any suitable filter technique. All the methods come up with default settings that are configurable by clicking on the name:



If there are some errors or outliers in one of the attributes, such as ***sepallength***, in that case, we can remove or update it from the ***Attributes*** section.

Types of Algorithms by Weka

WEKA provides many algorithms for machine learning tasks. Because of their core nature, all the algorithms are divided into several groups. These are available under the ***Explorer*** tab of the WEKA. Let's look at those groups and their core nature:

- ***Bayes:*** consists of algorithms based on Bayes theorem like *Naive Bayes*
- ***functions:*** comprises the algorithms that estimate a function, including *Linear Regression*
- ***lazy:*** covers all algorithms that use lazy learning similar to *KStar, LWL*
- ***meta:*** consists of those algorithms that use or integrate multiple algorithms for their work like *Stacking, Bagging*
- ***misc:*** miscellaneous algorithms that do not fit any of the given categories
- ***rules:*** combines algorithms that use rules such as *OneR, ZeroR*
- ***trees:*** contains algorithms that use decision trees, such as *J48, RandomForest*

Each algorithm has configuration parameters such as *batchSize, debug,* etc. Some configuration parameters are common across all the algorithms, while some are specific. These configurations can be editable once the algorithm is selected to use.

Weka Extension Packages

In version 3.7.2, a package manager was added to allow the easier installation of extension packages. Some functionality that includes Weka before this version has moved into such extension packages, but this change also makes it easier for others to contribute extensions to Weka and maintain the software, as this modular architecture allows independent updates of the Weka core and individual extensions.

AssignmentNo.10

Title: Case study on NOSQL database: MongoDB.

Aim: Detail study on MongoDB

MongoDB tutorial provides basic and advanced concepts of SQL. Our MongoDB tutorial is designed for beginners and professionals.

MongoDB is a No SQL database. It is an open-source, cross-platform, document-oriented database written in C++.

Our MongoDB tutorial includes all topics of MongoDB database such as insert documents, update documents, delete documents, query documents, projection, sort() and limit() methods, create a collection, drop collection, etc. There are also given MongoDB interview questions to help you better understand the MongoDB database.

What is MongoDB

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.

In simple words, you can say that - Mongo DB is a document-oriented database. It is an open source product, developed and supported by a company named 10gen.

MongoDB is available under General Public license for free, and it is also available under Commercial license from the manufacturer.

The manufacturing company 10gen has defined MongoDB as:

"MongoDB is a scalable, open source, high performance, document-oriented database." - 10gen

MongoDB was designed to work with commodity servers. Now it is used by the company of all sizes, across all industry.

History of MongoDB

The initial development of MongoDB began in 2007 when the company was building a platform as a service similar to window azure.

Window azure is a cloud computing platform and infrastructure, created by Microsoft, to build, deploy and manage applications and service through a global network.

MongoDB was developed by a NewYork based organization named 10gen which is now known as MongoDB Inc. It was initially developed as a PAAS (Platform as a Service). Later in 2009, it is introduced in the market as an open source database server that was maintained and supported by MongoDB Inc.

The first ready production of MongoDB has been considered from version 1.4 which was released in March 2010.

MongoDB2.4.9 was the latest and stable version which was released on January 10, 2014.

Purpose of Building MongoDB

It may be a very genuine question that - "what was the need of MongoDB although there were many databases in action?"

There is a simple answer:

All the modern applications require big data, fast features development, flexible deployment, and the older database systems not competent enough, so the MongoDB was needed.

The primary purpose of building MongoDB is:

- Scalability
- Performance
- High Availability
- Scaling from single server deployments to large, complex multi-site architectures.
- Key points of MongoDB
- Develop Faster
- Deploy Easier
- Scale Bigger

First of all, we should know what is document oriented database?

Example of Document-Oriented Database

MongoDB is a document-oriented database. It is a key feature of MongoDB. It offers a document-oriented storage. It is very simple you can program it easily.

MongoDB stores data as documents, so it is known as document-oriented database.

1. FirstName = "John",
2. Address = "Detroit",
3. Spouse = [{**Name**: "Angela"}].
4. FirstName ="John",
5. Address = "Wick"

There are two different documents (separated by ".").

Storing data in this manner is called as document-oriented database.

Mongo DB falls into a class of databases that calls Document Oriented Databases. There is also a broad category of database known as No SQL Databases.

Features of MongoDB

These are some important features of MongoDB:

1. Support ad hoc queries

In MongoDB, you can search by field, range query and it also supports regular expression searches.

2. Indexing

You can index any field in a document.

3. Replication

MongoDB supports Master Slave replication.
A master can perform Reads and Writes and a Slave copies data from the master and can only be used for reads or back up (not writes)

4. Duplication of data

MongoDB can run over multiple servers. The data is duplicated to keep the system up and also keep its running condition in case of hardware failure.

5. Load balancing

It has an automatic load balancing configuration because of data placed in shards.

6. Supports map reduce and aggregation tools.

7. Uses JavaScript instead of Procedures.

8. It is a schema-less database written in C++.

9. Provides high performance.

10. Stores files of any size easily without complicating your stack.

11. Easy to administer in the case of failures.

12. It also supports:

- JSON data model with dynamic schemas
- Auto-sharding for horizontal scalability
- Built in replication for high availability
- Now a day many companies using MongoDB to create new types of applications, improve performance and availability.

Prerequisite

Before learning MongoDB, you must have the basic knowledge of SQL and OOPs.

Audience

Our MongoDB tutorial is designed to help beginners and professionals.

Problem

We assure that you will not find any problem in this MongoDB tutorial. But if there is any mistake, please post the problem in contact form.