Department of

Computer Science & Engineering

LAB MANUAL

RELATIONAL DATA BASE MANAGEMENT SYSTEM-2 LAB

B.Tech– VI Semester

KCT College OF ENGG AND TECH.

VILLAGE FATEHGARH

DISTT.SANGRUR

INDEX

S.No.	**Experiments**
1.	Case Study on normalization
2.	Study and usage of query optimization techniques
3.	Study and usage of backup and recovery features of database management software
4.	Study and usage of any object oriented or object relational database management software.
5.	Apply data mining for the data of CSV file format using WEKA
6.	Creating and use Web database in PHP

EXPERIMENT No.1:

Case study on normalization.

The WORK relation illustrates data about employees, their job title and the department they are assigned to. From examining sample data and discussions with management we have found that employees can have multiple job titles and can be assigned to more than one department. Each department is completely sited in a single location but a city could have more than one department at some time.

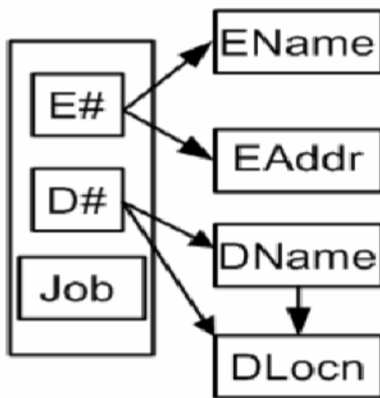
WORK

JOB	ENAME	EADDR	E#	D#	DNAME	DLOCN
HELPER	DAVIS	111 FIRST ST	12	1	PRESSING	ALCOA
HELPER	SPENCE	222 SECOND ST	78	1	PRESSING	ALCOA
ELECTRICIAN	MURPHY	100 MAIN ST	66	2	WELDING	NIOTA
FOREMAN	SMITH	300 BROAD ST	77	9	PACKING	LOUDON
CLERK	WILSON	111 FIRST ST	99	7	PAYROLL	MEMPHIS
CLERK	DAVIS	111 FIRST ST	12	1	PRESSING	ALCOA
CLERK	SPENCE	222 SECOND ST	78	1	PRESSING	ALCOA
CLERK	DAVIS	111 FIRST ST	12	5	MAILROOM	ONEIDA

For this relation, a composed key is required as no one attribute is a candidate. It turns out that the following SRN depicts the situation:

WORK (Job, EName, EAddr, E#, D#, DName, DLocn)

and the functional dependency diagrams would be:



There are numerous problems with the data model as it currently stands. We can not add new employees until they have a job title and a department assignment. We can easily lose department data by removing an employee who is the sole person assigned to a department. Certain updates require careful propagation of changes throughout the database.

Careful decomposition can take care of these problems. The employee data makes an obvious grouping and should be decomposed the get it into at least 2NF. It will actually go to BCNF as there are no further problems. It is ready to become a table.

EMPLOYEE

E#	ENAME	EADDR
12	DAVIS	111 FIRST ST
78	SPENCE	222 SECOND ST
66	MURPHY	100 MAIN ST
77	SMITH	300 BROAD ST
99	WILSON	111 FIRST ST

The Dept relation is another logical decomposition to remove the partial dependency and move to 2NF. Careful examination reveals the transitive dependency still exists so further decomposition is necessary.

DEPT

D#	DNAME	DLOCN
1	PRESSING	ALCOA
2	WELDING	NIOTA
9	PACKING	LOUDON
7	PAYROLL	MEMPHIS
5	MAILROOM	ONEIDA

Job-Worked winds up looking like the original relation's key. All three attributes are still the composed key. Since there are no dependencies, there is nothing to prevent this relation from being BSNF so it is ready too.

JOB-WORKED

E#	D#	JOB
12	1	HELPER
78	1	HELPER
66	2	ELECTRICIAN
77	9	FOREMAN
99	7	CLERK
12	1	CLERK
78	1	CLERK
12	5	CLERK

To remove the transitive dependency, we will decompose Dept into Department and Dept-Locn. Each of these is now in BCNF.

DEPARTMENT

D#	DNAME
1	PRESSING
2	WELDING
9	PACKING
7	PAYROLL
5	MAILROOM

DEPT-LOCN

D#	DLOCN
1	ALCOA
2	NIOTA
9	LOUDON
7	MEMPHIS
5	ONEIDA

EXPERIMENT No.2:

Study and usage of query optimization techniques.

Introduction: **Query optimization** is a function of many relational database management systems. The **query optimizer** attempts to determine the most efficient way to execute a given query by considering the possible query plans.

Generally, the query optimizer cannot be accessed directly by users: once queries are submitted to database server, and parsed by the parser, they are then passed to the query optimizer where optimization occurs. However, some database engines allow guiding the query optimizer with hints.

A query is a request for information from a database. It can be as simple as "finding the address of a person with SS# 123-45-6789," or more complex like "finding the average salary of all the employed married men in California between the ages 30 to 39, that earn less than their wives." Queries results are generated by accessing relevant database data and manipulating it in a way that yields the requested information. Since database structures are complex, in most cases, and especially for not-very-simple queries, the needed data for a query can be collected from a database by accessing it in different ways, through different data-structures, and in different orders. Each different way typically requires different processing time. Processing times of a same query may have large variance, from a fraction of a second to hours, depending on the way selected. The purpose of query optimization, which is an automated process, is to find the way to process a given query in minimum time. The large possible variance in time justifies performing query optimization, though finding the exact optimal way to execute a query, among all possibilities, is typically very complex, time consuming by itself, may be too costly, and often practically impossible. Thus query optimization typically tries to approximate the optimum by comparing several common-sense alternatives to provide in a reasonable time a "good enough" plan which typically does not deviate much from the best possible result.

.

Implementation

Most query optimizers represent query plans as a tree of "plan nodes". A plan node encapsulates a single operation that is required to execute the query. The nodes are arranged as a tree, in which intermediate results flow from the bottom of the tree to the top. Each node has zero or more child nodes—those are nodes whose output is fed as input to the parent node. For example, a join node will have two child nodes, which represent the two join operands, whereas a sort node would have a single child node (the input to be sorted). The leaves of the tree are nodes which produce results by scanning the disk, for example by performing an index scan or a sequential scan.

Join ordering

The performance of a query plan is determined largely by the order in which the tables are joined. For example, when joining 3 tables A, B, C of size 10 rows, 10,000 rows, and 1,000,000 rows, respectively, a query plan that joins B and C first can take several orders-of-magnitude more time to execute than one that joins A and C first. Most query optimizers determine join order via a dynamic programming algorithm pioneered by IBM's System R database project. This algorithm works in two stages:

1. First, all ways to access each relation in the query are computed. Every relation in the query can be accessed via a sequential scan. If there is an index on a relation that can be used to answer a predicate in the query, an index scan can also be used. For each relation, the optimizer records the cheapest way to scan the relation, as well as the cheapest way to scan the relation that produces records in a particular sorted order.
2. The optimizer then considers combining each pair of relations for which a join condition exists. For each pair, the optimizer will consider the available join algorithms implemented by the DBMS. It will preserve the cheapest way to join each pair of relations, in addition to the cheapest way to join each pair of relations that produces its output according to a particular sort order.
3. Then all three-relation query plans are computed, by joining each two-relation plan produced by the previous phase with the remaining relations in the query.

sort order can avoid a redundant sort operation later on in processing the query. Second, a particular sort order can speed up a subsequent join because it clusters the data in a particular way.

Query planning for nested SQL queries

A SQL query to a modern relational DBMS does more than just selections and joins. In particular, SQL queries often nest several layers of SPJ blocks (Select-Project-Join), by means of group by, exists, and not exists operators. In some cases such nested SQL queries can be flattened into a select-project-join query, but not always. Query plans for nested SQL queries can also be chosen using the same dynamic programming algorithm as used for join ordering, but this can lead to an enormous escalation in query optimization time. So some database management systems use an alternative rule-based approach that uses a query graph model.

Cost estimation

One of the hardest problems in query optimization is to accurately estimate the costs of alternative query plans. Optimizers cost query plans using a mathematical model of query execution costs that relies heavily on estimates of the cardinality, or number of tuples, flowing through each edge in a query plan. Cardinality estimation in turn depends on estimates of the selection factor of predicates in the query. Traditionally, database systems estimate selectivity through fairly detailed statistics on the distribution of values in each column, such as histograms. This technique works well for estimation of selectivities of individual predicates.

However many queries have conjunctions of predicates such as **SELECT COUNT**(*) **FROM** R **WHERE** R.make='Honda' **AND**R.model='Accord'. Query predicates are often highly correlated (for example, `model='Accord'` implies `make='Honda'`), and it is very hard to estimate the selectivity of the conjunct in general. Poor cardinality estimates and uncaught correlation are one of the main reasons why query optimizers pick poor query plans. This is one reason why a database administrator should regularly update the database statistics, especially after major data loads/unloads.

Query optimization techniques

The query processor optimizes queries that are written on the databases. However, this optimization is limited to the information that the query processor knows about in the databases and the organization of the databases.

Write effective queries and define logical tables to maximize query performance. Query optimization is based on the extent to which the database or file system performs the filtering that is required to obtain the final result set.

Example: A three-segment IMS™ HIDAM DBD contains 10,000 instances of the lowest level segment (also called the leaf segment).

- If the query processor can build a segment search argument (SSA), which contains a search argument for every segment, a single access is required. In this case, the query processor retrieves the final result set, and the connector or the query processor does not need to perform additional filtering.
- If the query processor cannot build an SSA, 10,000 IMS GET commands are issued. In this case, the connector or the query processor must filter the intermediate result set to obtain a single row result set.

Query processor optimization

The query processor optimizes SQL queries based on information in the WHERE clause, index information, and configuration parameters that activate optimization.

IMS access optimization

You can optimize access to IMS™ data with optimizing methods for queries, Program Communication Block (PCB) selection options, and Program Specification Block (PSB) scheduling.

VSAM access optimization

You can optimize access to VSAM data by using keyed access techniques, configuration parameters, and the VSAM service.

The techniques that you can use to optimize access to VSAM data only apply to VSAM KSDS data sets.

With ESDS and RRDS data sets, the entire contents of the files must be read to process a query. You cannot access the data directly, with the exception of accessing an ESDS data set through an alternate index.

Data server optimization

You can improve query performance with the dispatching priority of the data server and the Workload Manager (WLM) exit.

The dispatching priority at which the data server runs can have a strong affect on query performance. In addition, you can use the WLM system exit to place individual queries in WLM goal mode to control query resource

EXPERIMENT No. 3:

Study and usage of backup and recovery features of database management.

Purpose of Backup and Recovery

As a backup administrator, your principal duty is to devise, implement, and manage a backup and recovery trategy. In general, the purpose of a backup and recovery strategy is to protect the database against data loss and reconstruct the database after data loss. Typically, backup administration tasks include the following:

- ■ Planning and testing responses to different kinds of failures
- ■ Configuring the database environment for backup and recovery
- ■ Setting up a backup schedule
- ■ Monitoring the backup and recovery environment
- ■ Troubleshooting backup problems
- ■ Recovering from data loss if the need arises

As a backup administrator, you may also be asked to perform other duties that are related to backup and recovery:

- ■ Data preservation, which involves creating a database copy for long-term storage
- ■ Data transfer, which involves moving data from one database or one host to another

The purpose of this manual is to explain how to perform the preceding tasks.

Data Protection

As a backup administrator, your primary job is making and monitoring backups for data protection. A backup is a copy of data of a database that you can use to reconstruct data. A backup can be either a physical backup or a logical backup.

Physical backups are copies of the physical files used in storing and recovering a database. These files include data files, control files, and archived redo logs. Ultimately, every physical backup is a copy of files that store database information to another location, whether on disk or on offline storage media such as tape.

Logical backups contain logical data such as tables and stored procedures. You can use Oracle Data Pump to export logical data to binary files, which you can later import into the database. The Data Pump command-line clients expdp and impdp use the DBMS_DATAPUMP and DBMS_METADATA PL/SQL packages.

Physical backups are the foundation of any sound backup and recovery strategy. Logical backups are a useful supplement to physical backups in many circumstances but are not sufficient protection against data loss without physical backups.

Media Failures

A media failure is a physical problem with a disk that causes a failure of a read from or write to a disk file that is required to run the database.

User Errors

User errors occur when, either due to an error in application logic or a manua mistake, data in a database is changed or deleted incorrectly. User errors are estimated to be the greatest single cause of database downtime.

Application Errors
Sometimes a software malfunction can corrupt data blocks. In a **physical corruption**, which is also called a media corruption, the database does not recognize the block at all: the **checksum** is invalid, the block contains all zeros, or the header and footer of the block do not match. If the corruption is not extensive, then you can often repair it
easily with **block media recovery**.

Data Preservation
Data preservation is related to data protection, but serves a different purpose. For example, you may need to preserve a copy of a database as it existed at the end of a business quarter. This backup is not part of the disaster recovery strategy. The media to which these backups are written are often unavailable after the backup is complete.
You may send the tape into fire storage or ship a portable hard drive to a testing facility. RMAN provides a convenient way to create a backup and exempt it from your **backup retention policy**. This type of backup is known as an **archival backup**.
Data Transfer
In some situations you may need to take a backup of a database or database component and move it to another location. For example, you can use Recovery Manager (RMAN) to create a database copy, create a tablespace copy that can be imported into another database, or move an entire database from one platform to another. These tasks are not strictly speaking part of a backup and recovery strategy, but they do require the use of database backups, and so may be included in the duties of a backup administrator.

Backup and Recovery Solutions
When implementing a backup and recovery strategy, you have the following solutions available:
■ **Recovery Manager (RMAN)**
Recovery Manager is fully integrated with the Oracle database to perform a range of backup and recovery activities, including maintaining an **RMAN repository** of historical data about backups. You can access RMAN through the command line or through Oracle Enterprise Manager.
■ User-managed backup and recovery In this solution, you perform backup and recovery with a mixture of host operating system commands and SQL*Plus recovery commands.You are responsible for determining all aspects of when and how backups and recovery are done. These solutions are supported by Oracle and are fully documented, but RMAN is the preferred solution for database backup and recovery. RMAN provides a common interface for backup tasks across different host operating systems, and offers several backup techniques not available through user-managed methods Most of this manual focuses on RMAN-based backup and recovery. User-managed backup and recovery techniques are covered in Section VIII, "Performing User-Managed Backup and Recovery." The most noteworthy are the following:
■ Incremental backups
An **incremental backup** stores only blocks changed since a previous backup. Thus, they provide more compact backups and faster recovery, thereby reducing the need to apply redo during **data file media recovery**. If you enable **block change tracking**, then you can improve performance

by avoiding full scans of every input data file. You use the BACKUP INCREMENTAL command to perform incremental backups.

■ Block media recovery
You can repair a data file with only a small number of corrupt data blocks without taking it offline or restoring it from backup. You use the RECOVER BLOCK command to perform **block media recovery**.
■ Binary compression
A **binary compression** mechanism integrated into Oracle Database reduces the size of backups.
■ Encrypted backups
RMAN uses **backup encryption** capabilities integrated into Oracle Database to store backup sets in an encrypted format. To create encrypted backups on disk, the database must use the Advanced Security Option. To create encrypted backups directly on tape, RMAN must use the Oracle Secure Backup SBT interface, but does not require the Advanced Security Option.
■ Automated database duplication
Easily create a copy of your database, supporting various storage configurations, including direct duplication between ASM databases.
■ Cross-platform data conversion
Whether you use RMAN or user-managed methods, you can supplement physical backups with logical backups of schema objects made with Data Pump Export utility. You can later use Data Pump Import to re-create data after restore and recovery. Logical backups are mostly beyond the scope of the backup and recovery documentation.

EXPERIMENT No.4:

Study and usage of any object-oriented or object relational databases management software

4.1 Motivation

The relational model is the basis of many commercial relational DBMS products (e.g., DB2, Informix, Oracle, Sybase) and the structured query language (SQL) is a widely accepted standard for both retrieving and updating data. The basic relational model is simple and mainly views data as tables of rows and columns. The types of data that can be stored in a table are basic types such as integer, string, and decimal.

Relational DBMSs have been extremely successful in the market. However, the traditional RDBMSs are not suitable for applications with complex data structures or new data types for large, unstructured objects, such as CAD/CAM, Geographic information systems, multimedia databases, imaging and graphics. The RDBMSs typically do not allow users to extend the type system by adding new data types. They also only support first-normal-form relations in which the type of every column must be atomic, i.e., no sets, lists, or tables are allowed inside a column. Due to the new needs in database systems, a number of researches for OODBMS have begun in the early 80.s.

4.2 Concept & Features

While a relational database system has a clear specification given by Codd, no such specification existed for object-oriented database systems even when there were already products in the market. A consideration of the features of both object-oriented systems and database management systems has lead to a definition of an object-oriented database, which was presented at the First International Conference on Deductive, and Object-oriented Databases in the form of a manifesto in 1989. This ’manifesto’ distinguishes between the mandatory, optional and open features of an object-oriented database.

The mandatory features, which must be present if the system is to be considered (in the

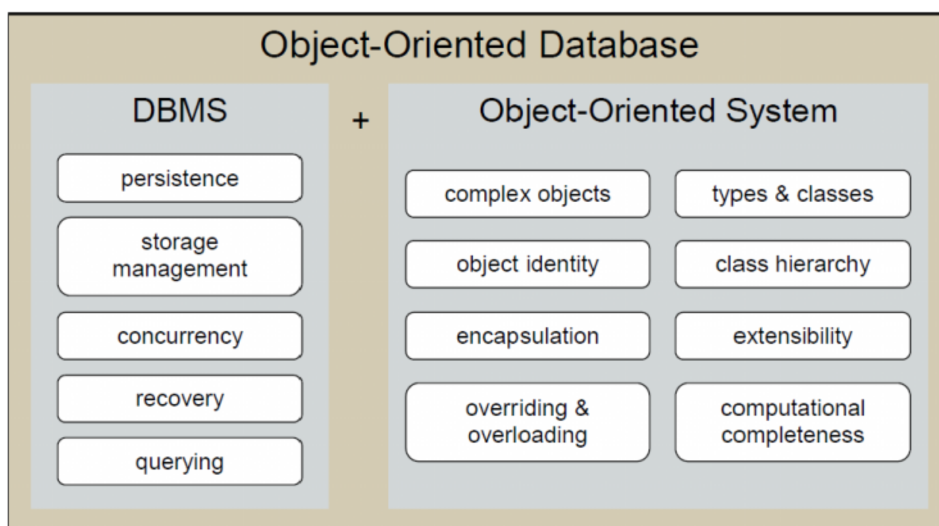


Figure 1 OODB feathues

opinion of the manifesto authors) to be an object-oriented database, are defined in the following two paragraphs. The first part describes features of object-oriented system, as the second part features of database system.

4.2.1 Mandatory features of object-oriented systems

Support for complex objects

A complex object mechanism allows an object to contain attributes that can themselves be objects. In other words, the schema of an object is not in first-normal-form. Examples of attributes that can comprise a complex object include lists, bags, and embedded objects.

Object identity

Every instance in the database has a unique identifier (OID), which is a property of an object that distinguishes it from all other objects and remains for the lifetime of the object. In object-oriented systems, an object has an existence (identity) independent of its value.

Encapsulation

Object-oriented models enforce encapsulation and information hiding. This means, the state of objects can be manipulated and read only by invoking operations that are specified within the type definition and made visible through the **public** clause.

In an object-oriented database system encapsulation is achieved if only the operations are visible to the programmer and both the data and the implementation are hidden.

Support for types or classes

Type: in an object-oriented system, summarizes the common features of a set of objects with the same characteristics. In programming languages types can be used at compilation time to check the correctness of programs.

Class: The concept is similar to type but associated with run-time execution. The term class refers to a collection of all objects with the same internal structure (attributes) and methods. These objects are called instances of the class.

Both of these two features can be used to group similar objects together, but it is normal for a system to support either classes or types and not both.

Class or type hierarchies

Any subclass or subtype will inherit attributes and methods from its superclass or supertype.

Overriding, Overloading and Late Binding

Overloading: A class modifies an existing method, by using the same name, but with a different list, or type, of parameters. Overriding: The implementation of the operation will depend on the type of the object it is applied to. Late binding: The implementation code cannot be referenced until run-time.

Computational Completeness

SQL does not have the full power of a conventional programming language. Languages such as Pascal or C are said to be computationally complete because they can exploit the full capabilities of a computer. SQL is only relationally complete, that is, it has the full power of relational algebra. Whilst any SQL code could be rewritten as a C++ program, not all C++ programs could be rewritten in SQL.

For this reason most relational database applications involve the use of SQL embedded within a conventional programming language. The problem with this approach is that whilst SQL deals with sets of records, programming languages tend to work on a record at a time basis. This difficulty is known as the impedance mismatch. Object-oriented databases attempt to provide a seamless join between program and database and hence overcome the

impedance mismatch. To make this possible the data manipulation language of an object-oriented database should be computationally complete.

4.2.2 Mandatory features of database systems

A **database** is a collection of data that is organized so that its contents can easily be accessed, managed, and updated. Thus, a database system contains the five following features:

Persistence

As in a conventional database, data must remain after the process that created it has terminated. For this purpose data has to be stored permanently on secondary storage.

Secondary Storage Management

Traditional databases employ techniques, which manage secondary storage in order to improve the performance of the system. These are usually invisible to the user of the system.

Concurrency

The system should provide a concurrency mechanism, which is similar to the concurrency mechanisms in conventional databases.

Recovery

The system should provide a recovery mechanism similar to recovery mechanisms in conventional databases.

Ad hoc query facility

The database should provide a high-level, efficient, application independent query facility. This needs not necessarily be a query language but could instead, be some type of graphical interface.

The above criteria are perhaps the most complete attempt so far to define the features of an object-oriented database in 1989. Further attempts to define an OODBS standard were made variables of researchers. One of them is a group called Object Data Management Group (ODMG). They have worked on an OODBS standard for the industry. The recent release is ODMG-2 in1997.

4.3 Making OOPL a Database

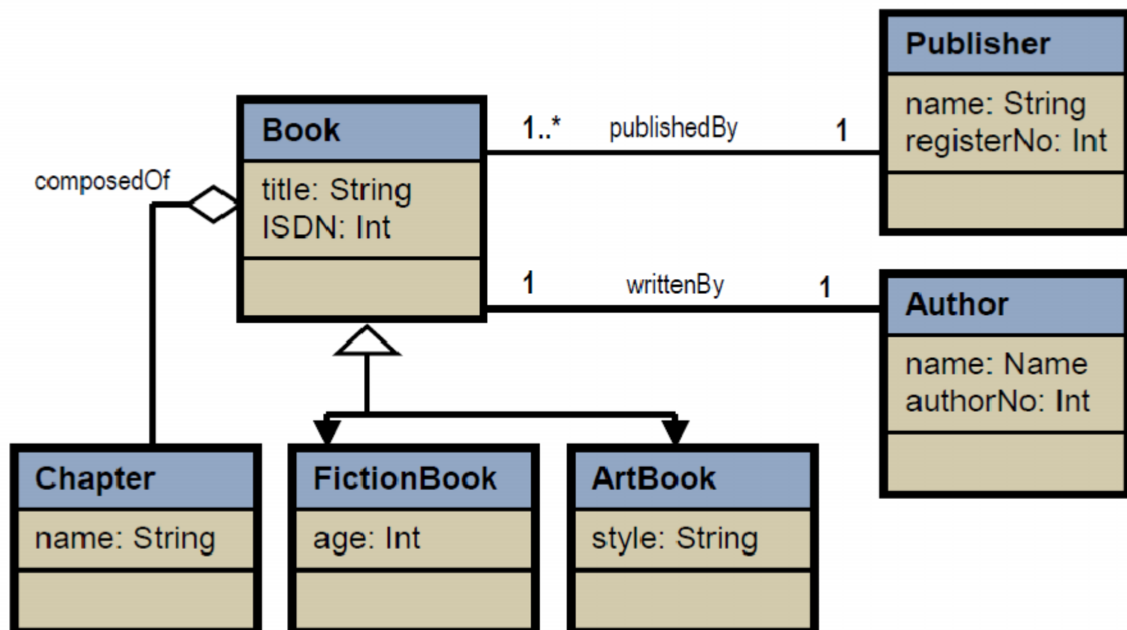
Basically, an OODBMS is an object database that provides DBMS capabilities to objects that have been created using an object-oriented programming language (OOPL). The basic principle is to add persistence to objects and to make objects persistent. Consequently application programmers who use OODBMSs typically write programs in a native OOPL such as Java, C++ or Smalltalk, and the language has some kind of Persistent class, Database class, Database Interface, or Database API that provides DBMS functionality as, effectively, an extension of the OOPL.

Object-oriented DBMSs, however, go much beyond simply adding persistence to any one object-oriented programming language. This is because, historically, many object-oriented DBMSs were built to serve the market for computer-aided design/computer-aided manufacturing (CAD/CAM) applications in which features like fast navigational access, versions, and long transactions are extremely important. Object-oriented DBMSs, therefore, support advanced object-oriented database applications with features like support for persistent objects from more than one programming language, distribution of data, advanced transaction models, versions, schema evolution, and dynamic generation of new types.

The following subsection describes object data modeling and the persistency concept in OODB.

4.3.1 Object data modeling

An object consists of three parts: structure (attribute, and relationship to other objects like aggregation, and association), behavior (a set of operations) and characteristic of types (generalization/serialization). An object is similar to an entity in ER model; therefore we begin with an example to demonstrate the structure and relationship.



The structure of an object Book is defined as following:

```
class Book {
title: String;
ISDN: Int;
publishedBy: Publisher inverse publish;
writtenBy: Author inverse write;
chapterSet: Set<Chapter>;
}
class Author {
name: String;
authorNo: Int;
write: Book inverse writtenBy;
}
```

Attributes are like the fields in a relational model. However in the Book example we have, for attributes publishedBy and writtenBy, complex types Publisher and Author, which are also objects. Attributes with complex objects, in RDNS, are usually other tables

linked by keys to the employee table.
Relationships: publish and writtenBy are associations with I:N and 1:1 relationship; composed_of is an aggregation (a Book is composed of chapters). The 1:N relationship is usually realized as attributes through complex types and at the behavioral level. For example,

```
class Publisher {
…
publish: Set<Book> inverse publishedBy;
…
Method insert(Book book){
publish.add(book);
}
```

Generalization/Serialization is the is_a relationship, which is supported in OODB through class hierarchy. An ArtBook is a Book, therefore the ArtBook class is a subclass of Book class. A subclass inherits all the attribute and method of its superclass.

```
class ArtBook extends Book {
style: String;
}
```

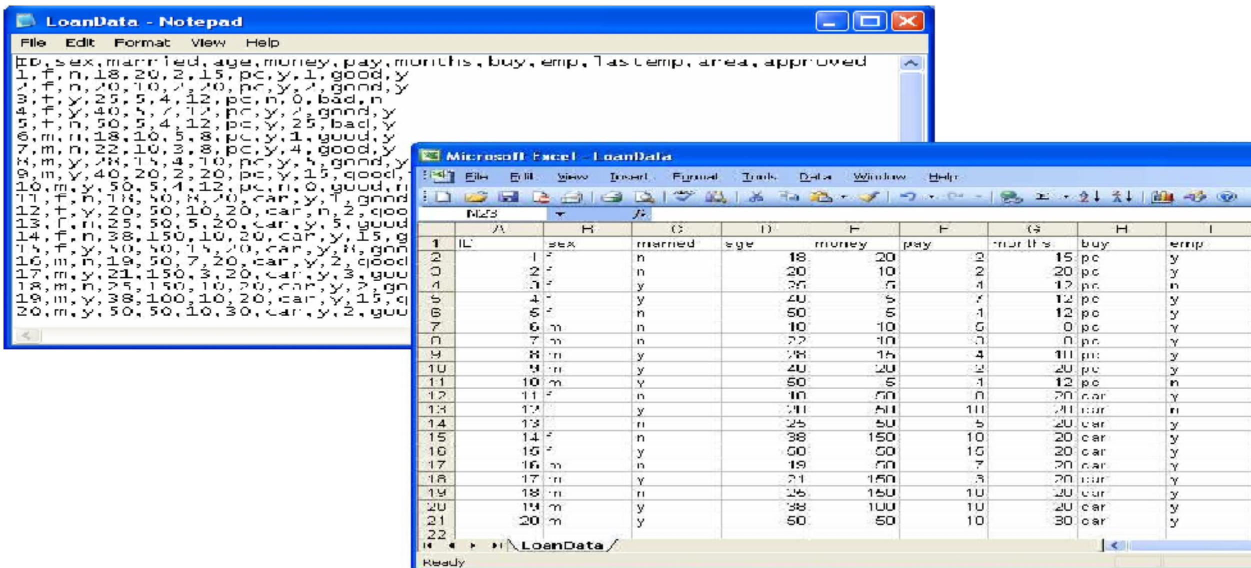
Message: means by which objects communicate, and it is a request from one object to another to execute one of its methods. For example:
Publisher_object.insert (”Rose”, 123,…)
i.e. request to execute the insert method on a Publisher object **)**
Method: defines the behavior of an object. Methods can be used
. to change state by modifying its attribute values
. to query the value of selected attributes
The method that responds to the message example is the method insert defined in the Publisher class.

4.3.2 Persistence of objects

Persistence, as mentioned before, means that certain program components .survive. the termination of the program. Thus these components have to be stored permanently on secondary storage.
Typically, persistence or non-persistence is specified at object creation time. There are two possible ways to make an object persistent:
(1) explicitly call built-in function ***persistence*** . certain objects are persistent
(2) automatically make object of persistent types persistent . all objects are persistent
There are several object-oriented DBMSs in the market (e.g., Gemstone, Objectivity/DB, ObjectStore, Ontos, O2, Itasca, Matisse). These products all support an object-oriented data model. Specifically, they allow the user to create a new class with attributes and methods, have the class inherit attributes and methods from superclasses, create instances of the class each with a unique object identifier, retrieve the instances either individually or collectively and load and run methods.
Most of these OODBs support a unified programming language and database language. That is, one language (e.g., C++ or Smalltalk) in which to do both general-purpose programming and database management.

Experiment-5

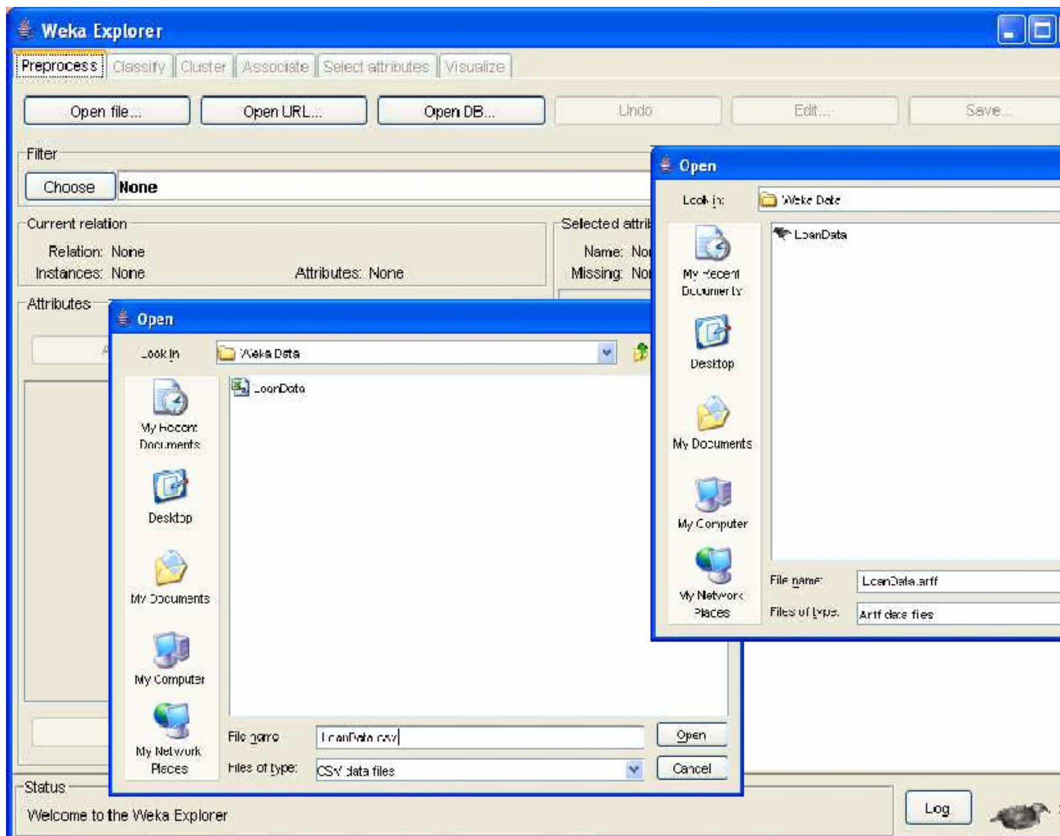
- Apply data mining for the data of CSV file format using



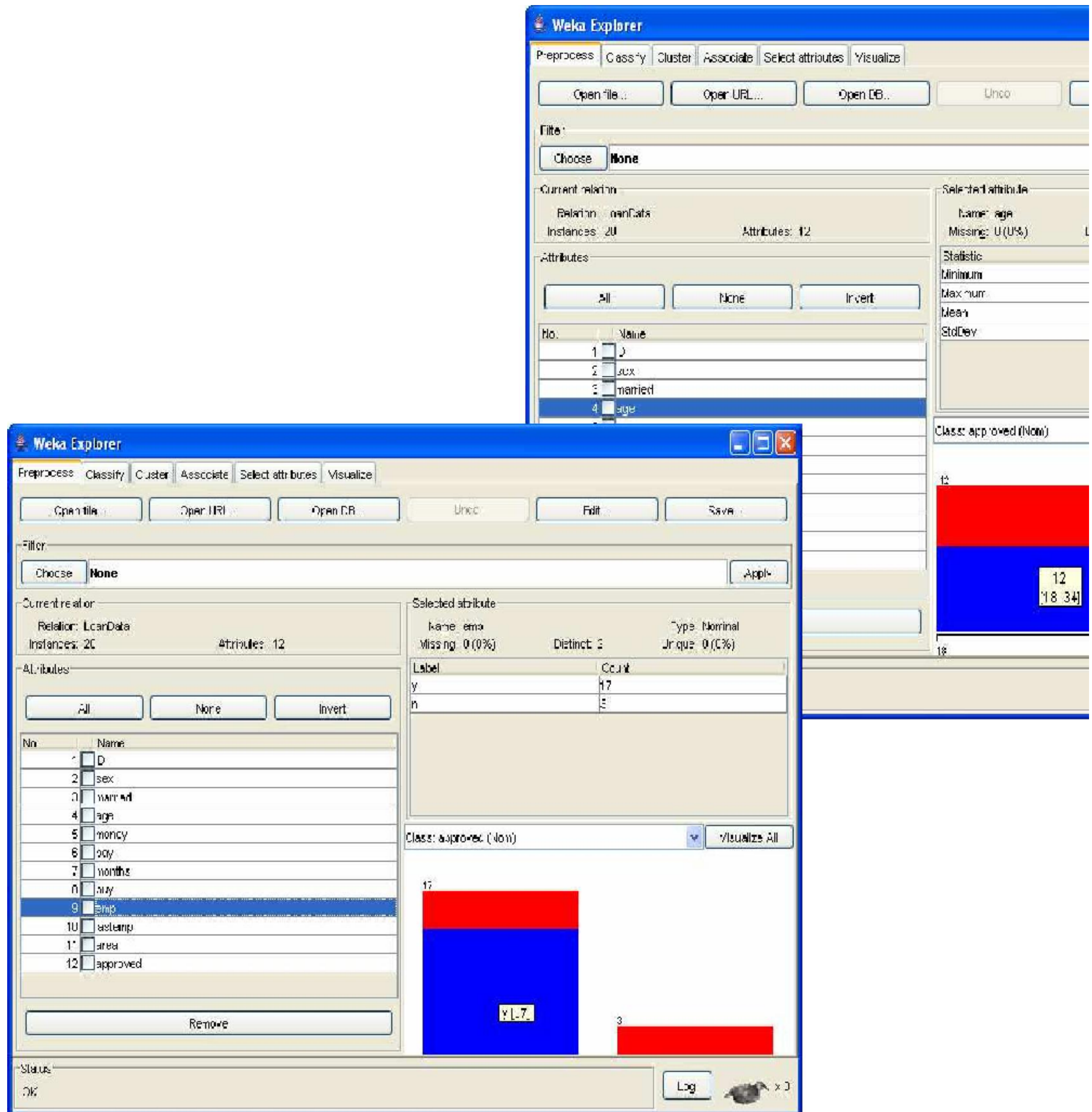
WEKA:-

Solution:-

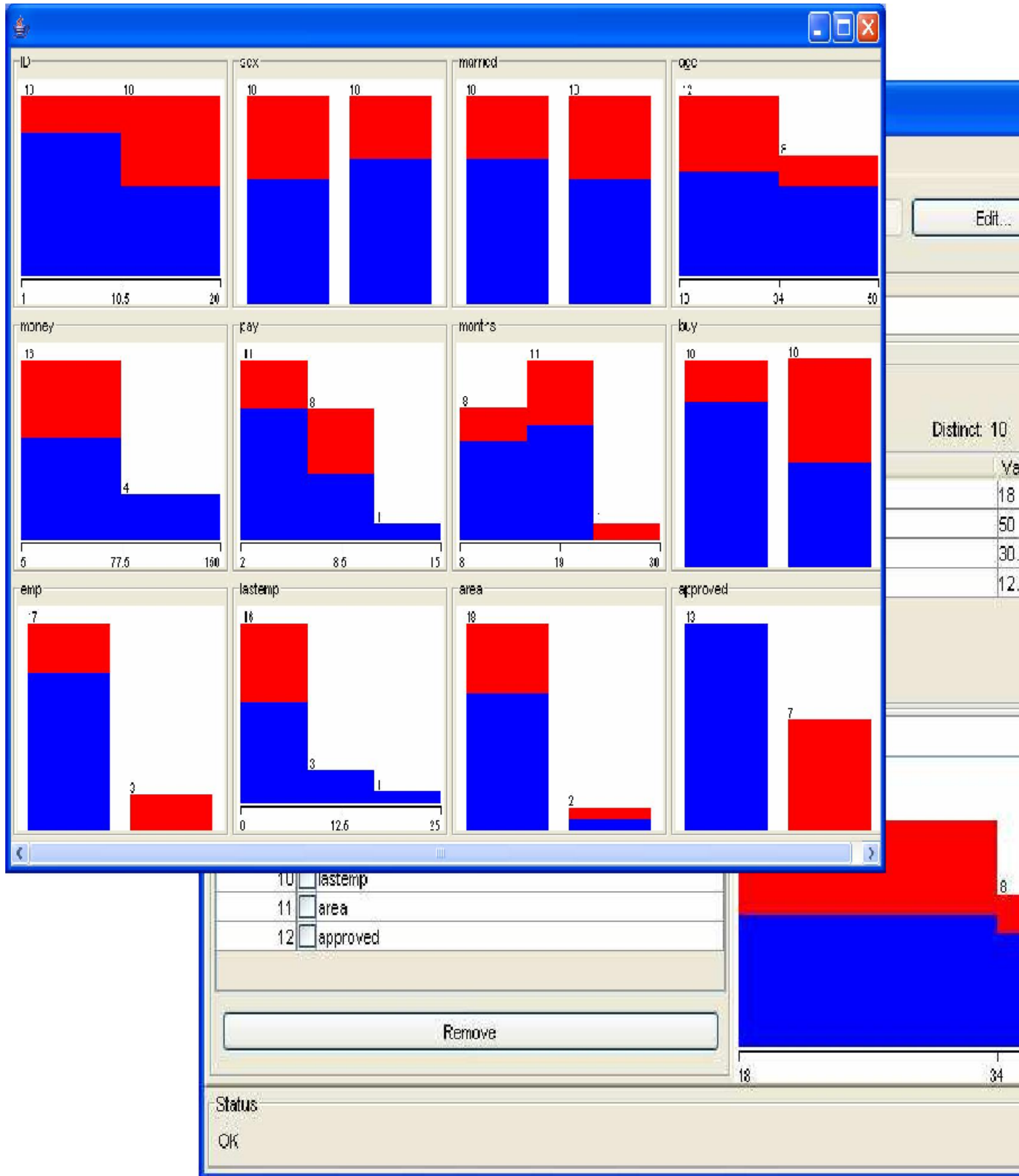
Step 1 - Choose the file by Open File dialog:-



Step 2 - Choose the field to visualize the results. Example - emp as shown below:-



By clicking at All the complete result is as shown below:-



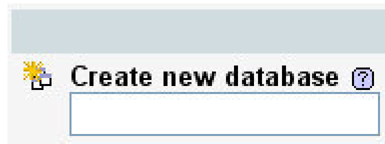
EXPERIMENT 6

- Creating and use Web database in PHP, shown below:-

ID	Title	First_Name	Surname
1	Mr	Test	Name
2	Mrs	Second	Test

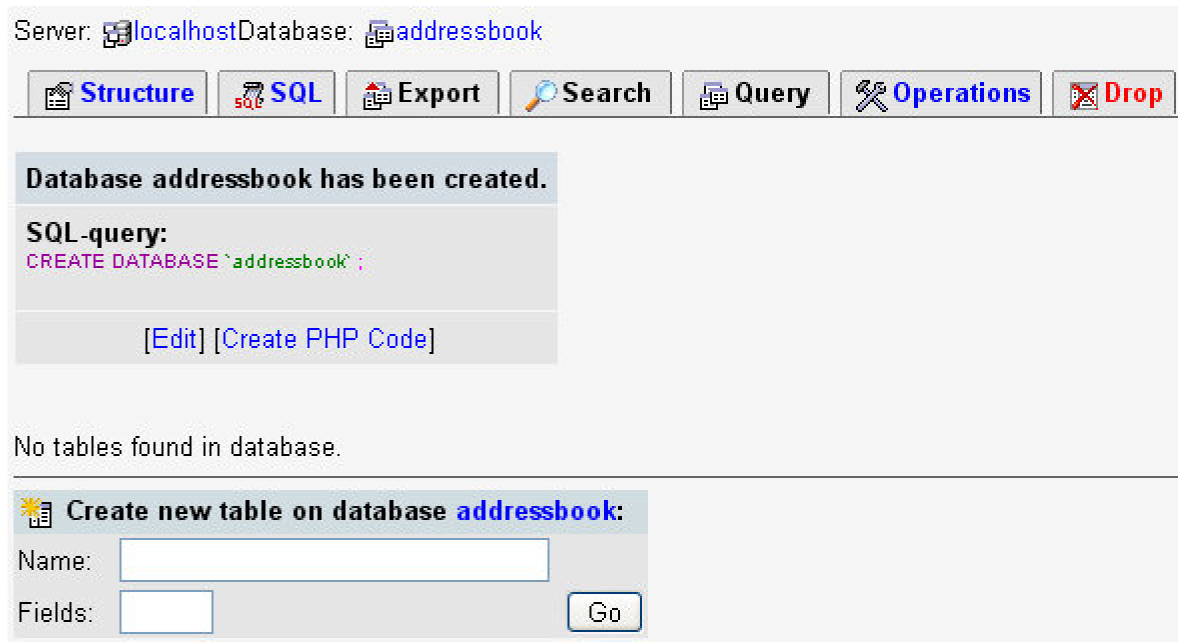
Solution:-

Creating a database in PHP using PHPmyadmin:-



This is where you type a name for your database. We're going to create a simple Address Book, so type that into the textbox:

After you have typed a name for your new database, click the "Create" button. You will be taken to a new area:



In this new area, you can create a Table to go in your database. At the moment, as it says, there are **No tables found in the database**. But the database itself has been created.

To create a new table, type a name for it in the box at the bottom. You can also type a number for the **Fields** textbox. The fields are the columns, remember, and will be things like first_name, surname, address, etc. You can always add more later, but just type 4 in there. In fact, type it out exactly as it is below:

Create new table on database addressbook:

Name: tbl_address_book

Fields: 4 Go

When you've finished, click the **Go** button. Another, more complex, area will appear:

Server: localhost Database: addressbook Table: tbl_address_book

Field	Type	Length/Values*	Collation	Attributes	Null	Default**	Extra
	VARCHAR				not null		
	VARCHAR				not null		
	VARCHAR				not null		
	VARCHAR				not null		

Table comments: Table type: Default Collation:

Add 1 field(s) Go

Save

* If field type is "enum" or "set", please enter the values using this format: 'a','b','c'...
If you ever need to put a backslash ("\") or a single quote ("'") amongst those values, backslash it (for example '\\xyz' or 'a\'b').

** For default values, please enter just a single value, without backslash escaping or quotes, using this format: a

In this new area, you set up the **fields** in your database. You can specify whether a field is for text, for numbers, for yes/no values, etc

Fill the data as shown below:-

Field	Type	Function	Null	Value
ID	smallint(6)			1
First_Name	varchar(50)			Test
Surname	varchar(50)			Name
Address	tinytext			12 Test Street

Using a database in PHP:-

```
<?PHP

        $user_name = "root";
        $password = "";
        $database = "addressbook";
        $server = "127.0.0.1";

        $db_handle = mysql_connect($server, $user_name, $password);
        $db_found = mysql_select_db($database, $db_handle);

        if ($db_found) {

                $SQL = "SELECT * FROM tb_address_book";
                $result = mysql_query($SQL);

                while ( $db_field = mysql_fetch_assoc($result) ) {

                        print $db_field['ID'] . "<BR>";
                        print $db_field['First_Name'] . "<BR>";
                        print $db_field['Surname'] . "<BR>";
                        print $db_field['Address'] . "<BR>";

                }

            mysql_close($db_handle);

        }
        else {

            print "Database NOT Found ";
            mysql_close($db_handle);

        }

?>
```