INDEX

S.No.	Objective	Date	Sign
1.	Installing oracle		
2.	Creating Entity Relationship Diagram using case tools		
3.	Writing SQL statements Using ORACLE /MYSQL: a)Writing basic SQL SELECT statements. b) Restricting and sorting data. c)Displaying data from multiple tables. d)Aggregating data using group function. e)Manipulating data. f)Creating and managing tables.		
4.	Normalization		
5.	Creating cursor		
6.	Creating procedure and functions		
7.	Creating packages and triggers		
8.	Design and implementation of payroll processing system		
9.	Design and implementation of Library Information System		
10.	Design and implementation of Student Information System		
11	Automatic Backup of Files and Recovery of Files		

1. Lab Exercise

Aim: Installation of oracle 10g On Windows (Introduction to Database)

S/w Requirement: CD of the setup Oracle 10g.

Theory:

Database is a collection of information in a structured way. We can say that it is a collection of a group of facts. Your personal address book is a database of names you like to keep track of, such as personal friends and members of yourfamily.

Fig.1 contains required details about each student. There are six pieces of information on each student. They are Roll No, Name, Address and Subjects. Each piece of information in database is called a **Field**. We can define *field* as *the smallest unit in a database*. Each field represents one and only one characteristic of an event or item. Thus there are four fields in this database.

ROLL_NO	NAME	ADDRESS	SUBJECTS
1716510001	Abc	C-6, Kanpur.	DBMS
1716510002	Pqr	A-5, Kanpur.	.NET
1716510003	Dyz	R-2, Kanpur.	C
1716510004	Stq	M-1, Kanpur.	VB.NET
1716510005	Plm	Y-7, Kanpur.	CG

Fig. 1

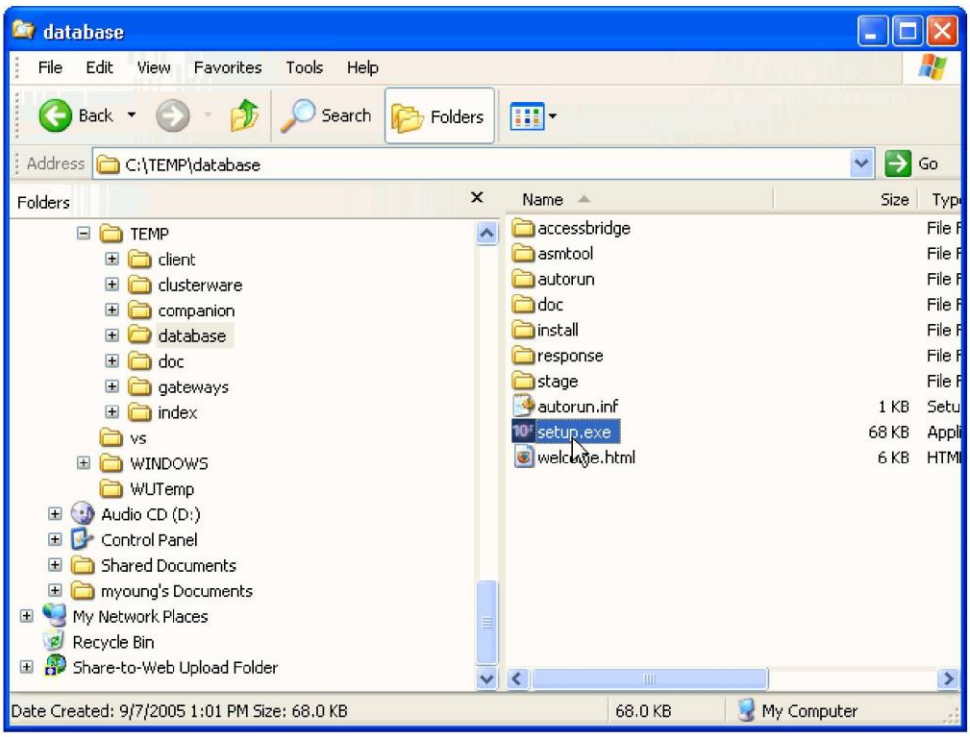
If you take a close look at all these fields, they are not of the same type. Name is character type Roll no is number type. In database there can be five categories of fields. They are:

- Numeric
- Character
- Logic
- Memo
- Date

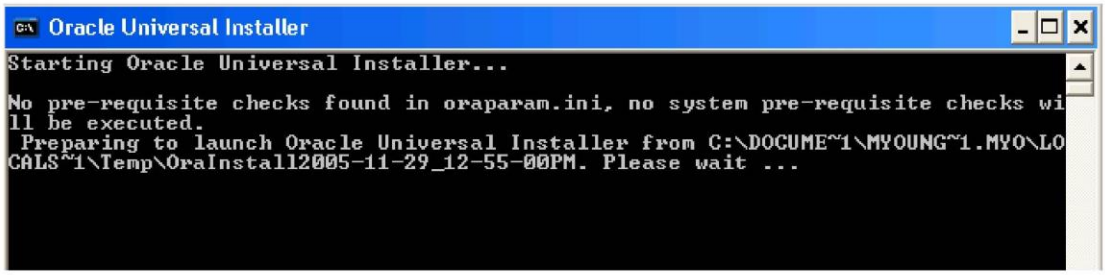
Installation of oracle 10g:

To install the Oracle software, you must use the Oracle Universal installer.

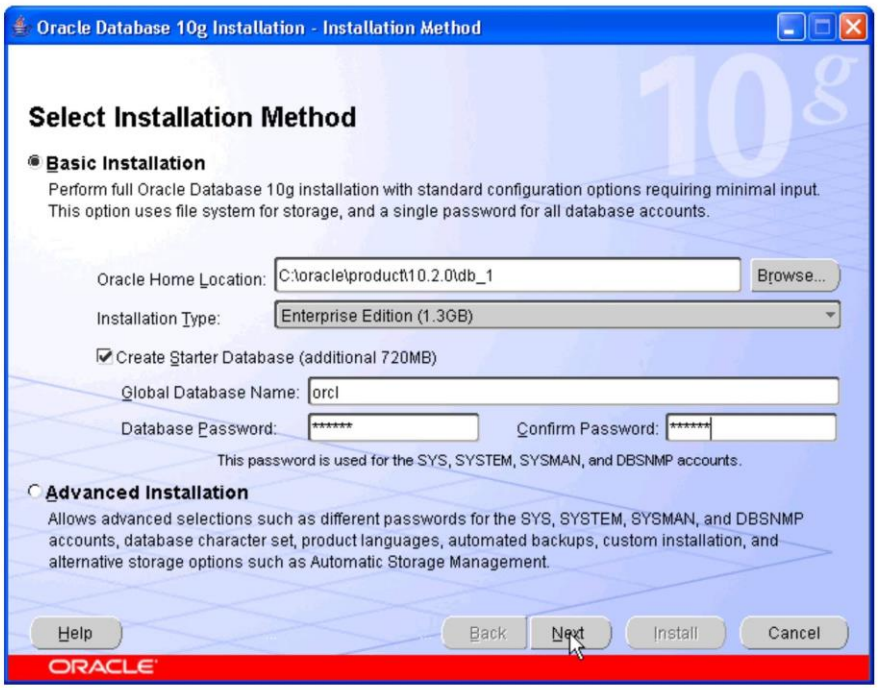
1. For this installation we need either the DVDs or a downloaded version of the DVDs. From the directory where the DVD files were unzipped, double-click **`setup.exe`**.



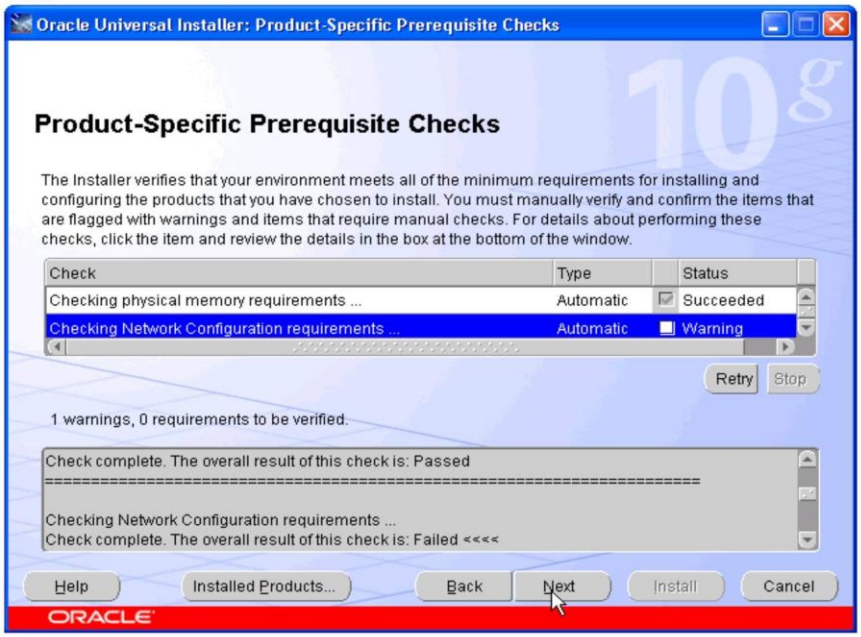
2. The Oracle Universal Installer starts.



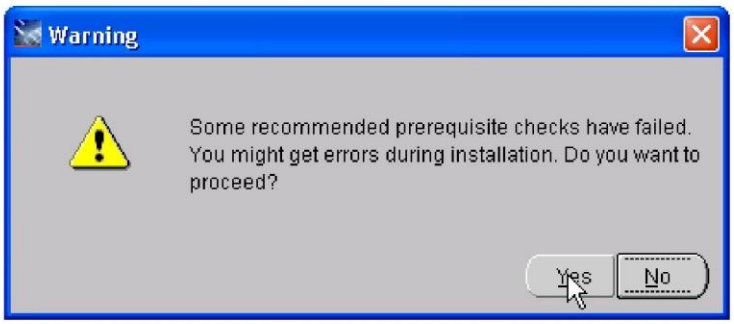
3. We have to perform a basic installation with a starter database. Enter **orcl** for the Global Database Name and **oracle** for the Database Password and Confirm Password. Then click **Next**.



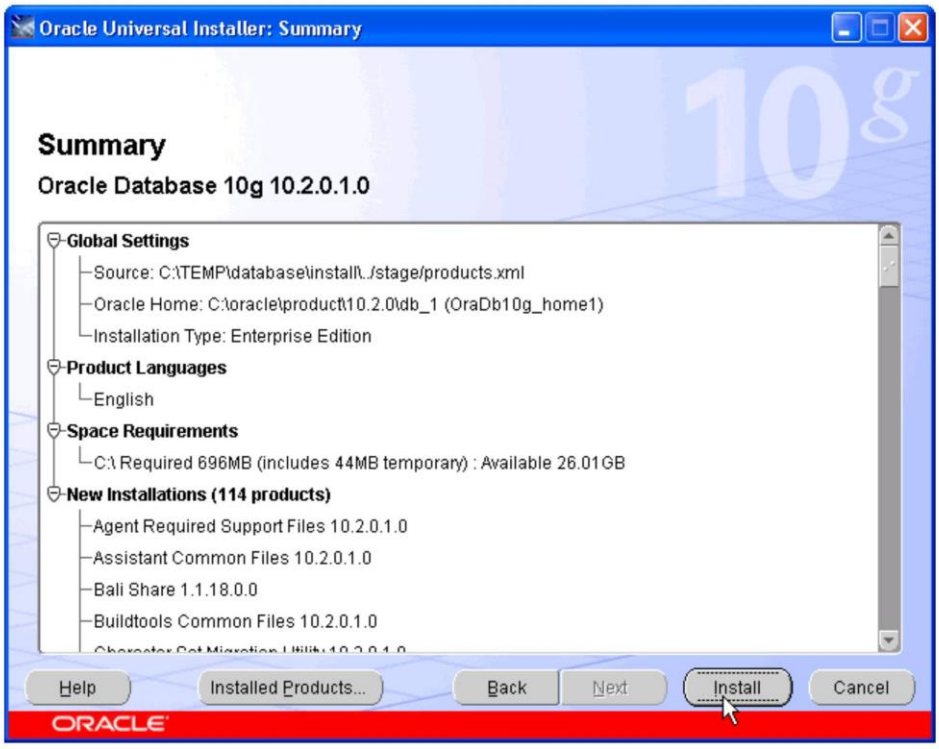
4. The installer now verifies that the system meets all the minimum requirements for installing and configuring the chosen product. We need to correct any reported errors (warnings are OK) before continuing. When the check successfully completes (with or without warnings), click **Next**.



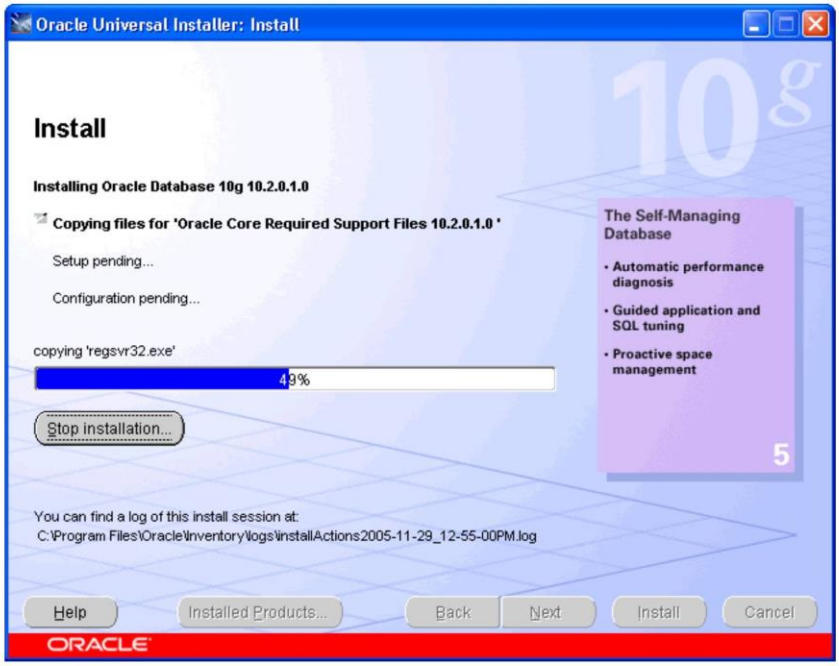
5. If we received any warnings, we can proceed. Click **Yes**



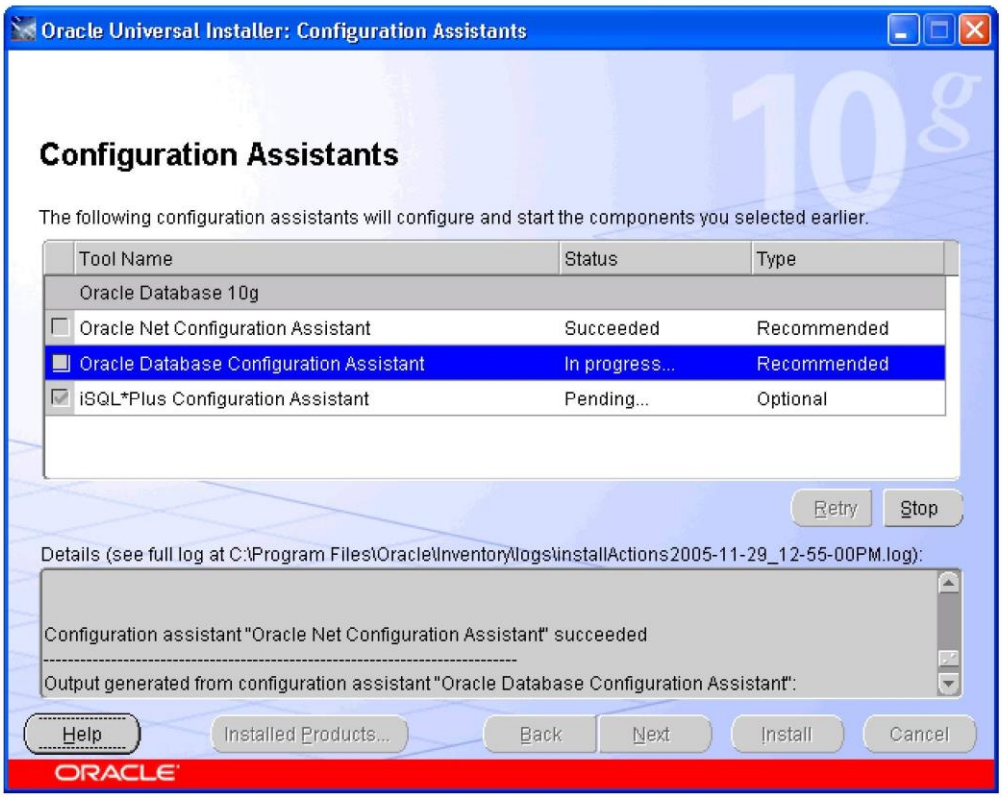
6. Review the Summary window to verify what is to be installed. Then, click **Install**.



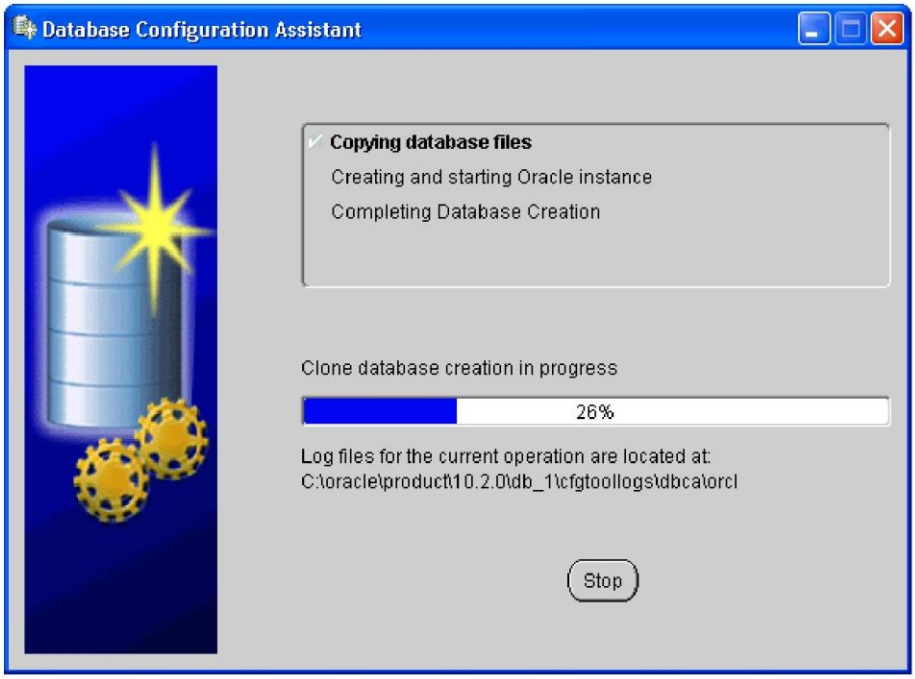
7. The progress window appears.



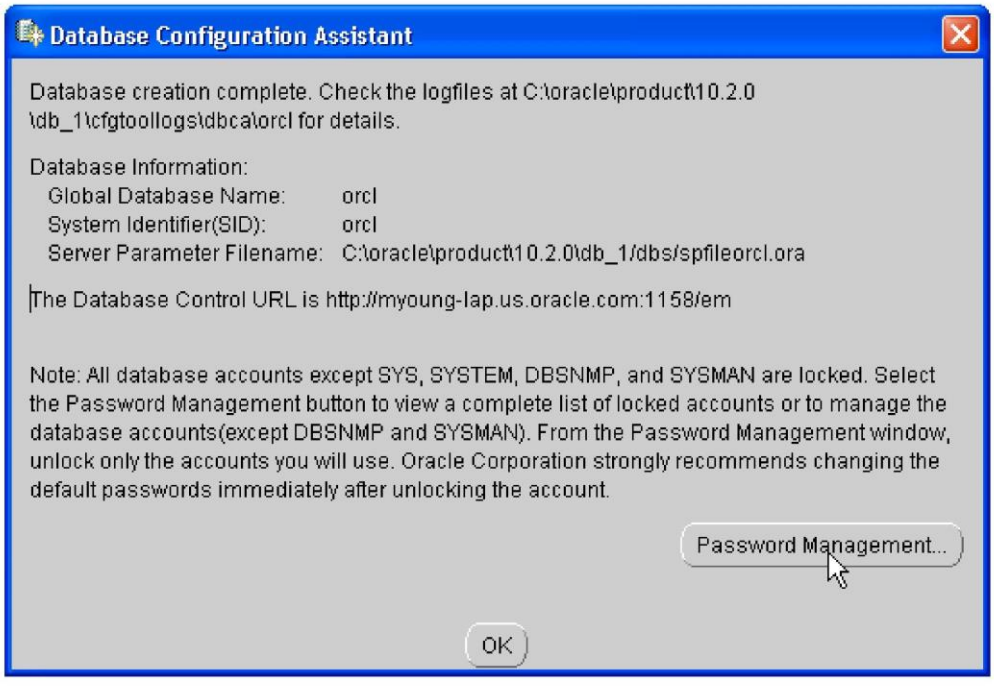
8. The Configuration Assistants window appears.



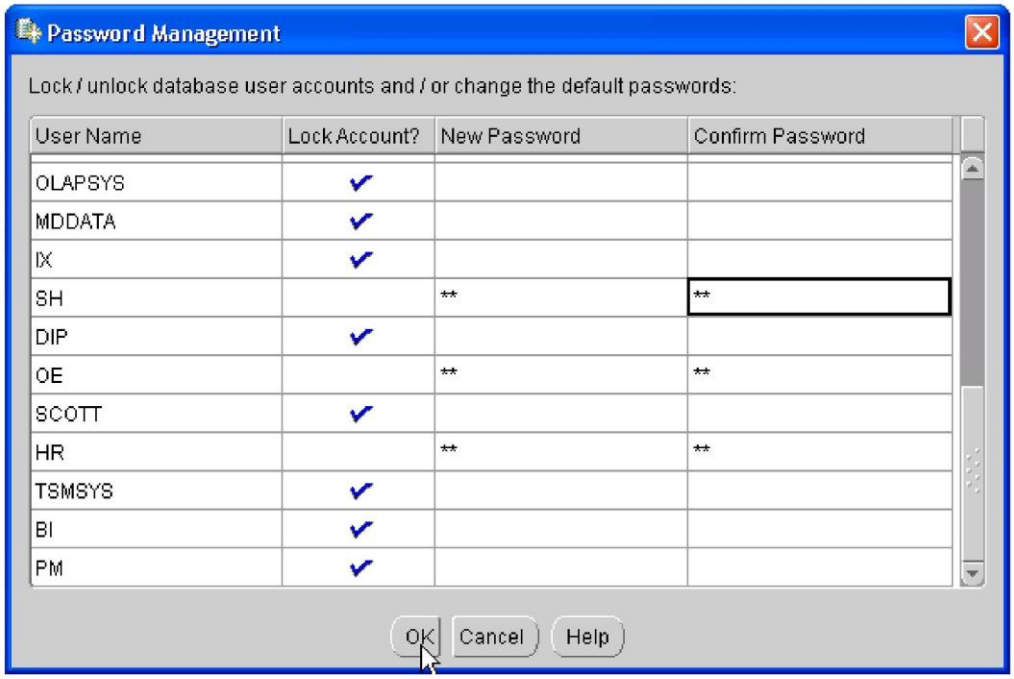
9. Your database is now being created.



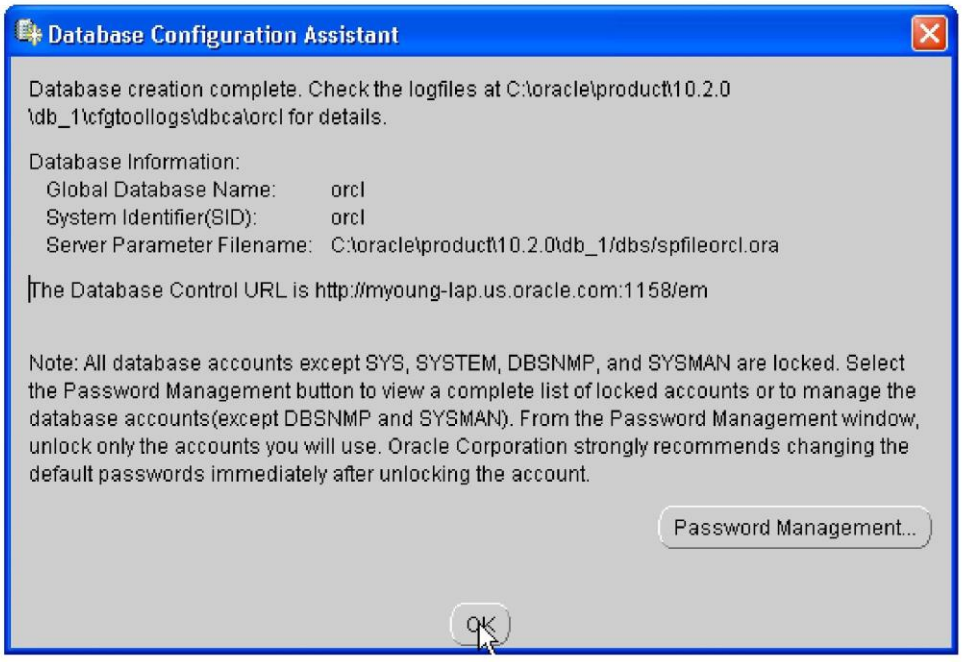
10. When the database has been created, you can unlock the users you want to use. Click **Password Management**.



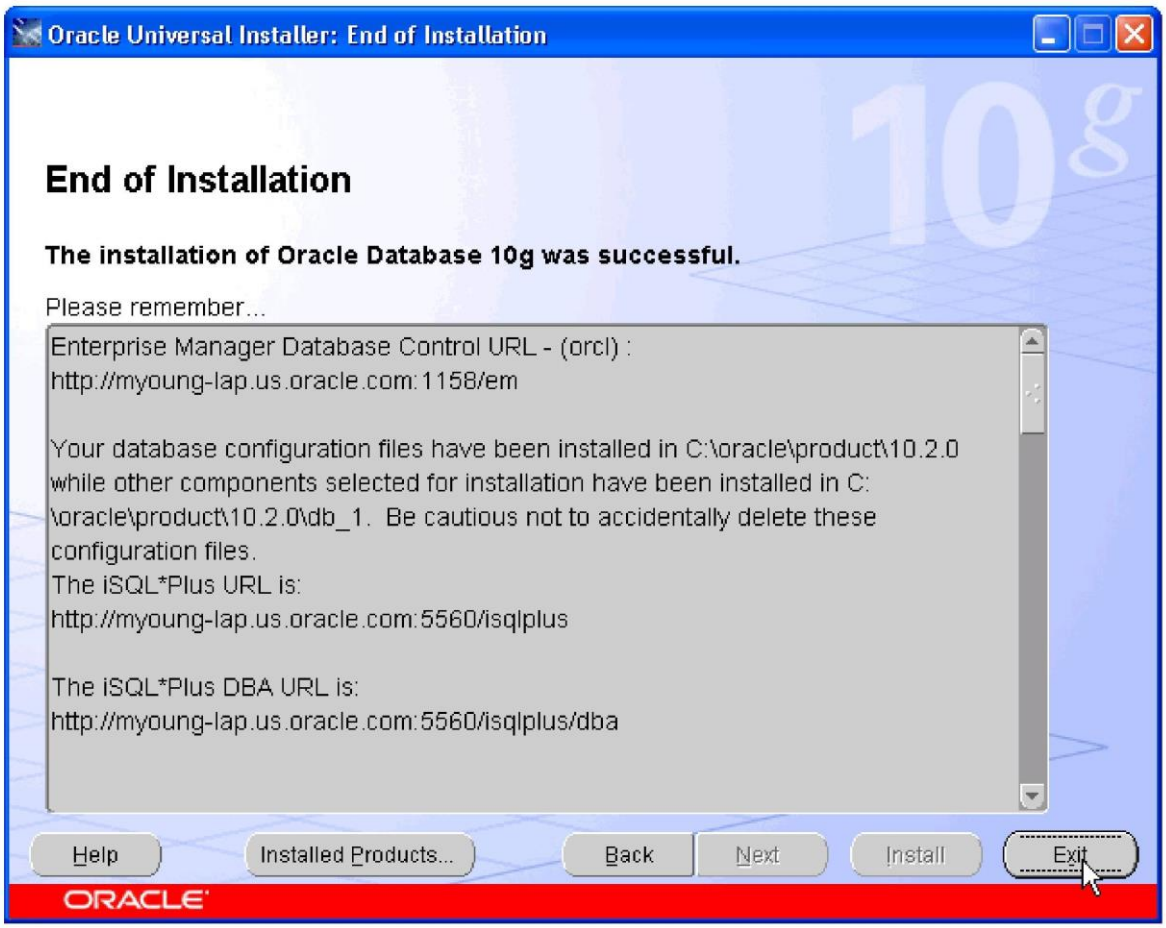
11. Unlock **SH**, **OE** and **HR** users by clicking on the check mark in the Lock Account? column. Enter the same name as the user in the New Password and Confirm Password fields. For example, to unlock **SH** user, enter **SH** in the New Password and Confirm Password fields. Then, click **OK.**



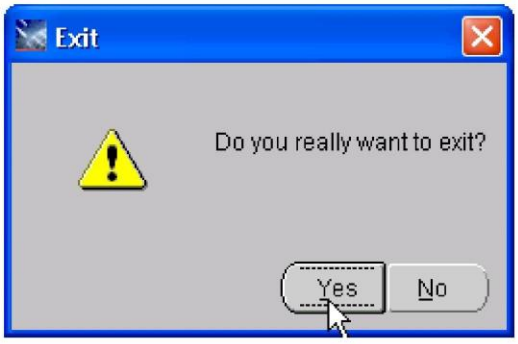
12. Click **OK** again.



13. Click **Exit**.



14. Click **Yes** to confirm exit.



2. Lab Exercise

Aim: Study of Entity- Relationship diagram.

S/w Requirement: Oracle 10g.

Theory:

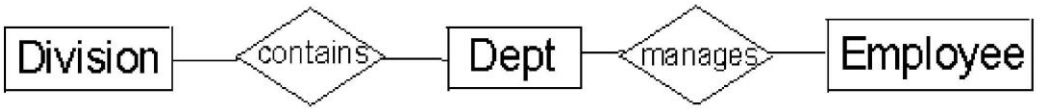
An entity-relationship (ER) diagram is a specialized graphic that illustrates the interrelationships between entities in a database. ER diagrams often use symbols to represent three different types of information. Boxes are commonly used to represent entities. Diamonds are normally used to represent relationships and ovals are used to represent attributes.

1 Entity
An *entity* is a person, place, concept, or thing about which the business needs data.

2 Relationship
A *relationship* is an association between entity types

Example of ER Diagram
we have identified three entity types (*Employee*, *Department*, *Division*) and two relationships among these entity types (*manages*, *contains*). Now we can begin to represent the problem in the language of ER modeling.

ER models are usually represented graphically. The language we are going to use represents entity types as rectangles and relationships as diamonds. Below is the representation of the situation we are working with.



Notice that the contains relationship is drawn between the two entities that it is associated with. Similarly for the manages relationship. This (simplified) ER model tells us that:

- Division is related to department through a relationship called *contains*.
- Departments are related to employees through a relationship called *manages*.
- Employees are not directly related to divisions.

Certainly, we know more about the problem than this. Consider the relationship between divisions and departments. We know that divisions have multiple departments and departments can only be contained within one division. Or, for every *one* division there can be many departments. In the

language of ER modeling this is called a **1: M** (read: “one to many”) relationship.

3. Cardinality: There are 4 types of cardinality:

   A. One to One
   B. One to Many
   C. Many to Many
   D. Many to One

A. One-to-one

One-to-one table relationships are a little more interesting and more underused than either of the other two types of relationships. The key indicator of a possible need for a one-to-one relationship is a table that contains fields that are only used for a certain subset of the records in that table.

Let's take a look at building a Catalog table for the items that your store sells. Odds are that you need to store some information about the individual items like catalog numbers, weight, and other common data. But if you're selling different kinds of items, books and CDs for example, you may want some item- specific information in the database. For example, you may want a page count, author, publish date, and ISBN for books, while you want playing time, number of tracks, artist, and label for the CDs. You could come up with some way to fit both sets of data into the same structure, but then when management decides you're also selling pet supplies, your system willprobably break!

A better solution would be a *one-to-one* relationship between the Item table and another table of item-specific data for each type of item. The resulting structure is essentially one "master" table (CatalogItems) with one or more "subtables" (CDs and Books in this example). You link the two subtables to the master table through the primary key of the master table.

Catalog Table

CatalogID	Price	Description	QuantityOnHand

CDs

CatalogID	PlayingTime	NumOfTracks	Artist	Label

Books

CatalogID	PageCount	Author	PublishDate	ISBN

It may take a few minutes for this design to sink in. As a comparison, here is what the proposed database table would look like as a single monolithic table.

The one-to-one relationship has saved us from doubling the number of fields in the Catalog table and, more importantly, helped us break the database into more discrete entities. In this scenario, we can get all the general information about an item from the Catalog table and can use the primary key of that table to pull up the appropriate information from the subtable.

B. One-to-many

The one-to-many relationship is the workhorse of relational databases as well as being the easiest relationship to understand. Let us say you need to build a shopping cart application for an e-commerce site. Your first draft of the database has columns for Item1, Item2, and Item3 with the corresponding Quantity1, Quantity2, and Quantity3 fields.

OrderNum	ShippingInfo	Item1	Quantity1	Item2	Quantity2	Item3	Quantity3

Of course, this immediately starts to break down with more than three orders! Any time you find yourself designing a database and adding similar fields like this to the same table, you need to break the table into two (or more!) related tables using a *one-to-many* relationship.

A one-to-many relationship allows records in Table 1 to be connected to an arbitrary number of records in Table 2 without the limitations imposed by resorting to redundant or limited numbers of fields in a single table. This reduces the size of the database and greatly increases the flexibility and performance of queries operating on that data. We can take our shopping cart example and break it into an Order table and an Item table quite simply.

Order Table

OrderID	ShippingInfo

OrderItem Table

OrderItemID	OrderID	Item	Quantity

The two tables are linked together using the OrderID field. The contents of any order in the Order table can easily be found by finding all the items with that value in the OrderID field. There is also the added advantage that the two pieces of data are independent and can easily be modified. If we now want to add an ItemNumber to the OrderItem table, we add a single column; in our

original monolithic data table, we'd be adding ItemNumber1, ItemNumber2, etc.

C. Many-to-many

Finally, there is the many-to-many table. This relationship is a little more complex than the one-to-many because, in addition to the two tables of data, we need another table to join the two tables of interest together. That's right, we're adding a table to the database -- but it is a simple table and saves us lots of effort down the road. As an example, let's say you want to add the ability to search for CDs by the musicians on any given song. From the musician side, you have one musician related to many songs.

Musician Table

MusicianID	MusicianName
44	Paul McCartney

Song Table

SongID	MusicianID	SongName
200	44	Sgt. Pepper's Lonely Heart's Club Band
201	44	Ebony and Ivory

But from the song side, you potentially have a song related to many musicians. The following visual represents that situation.

Song Table

SongID	SongName
200	Sgt. Pepper's Lonely Heart's Club Band

Musician Table

MusicianID	SongID	MusicianName
43	200	John Lennon
44	200	Paul McCartney

These two tables work individually, but when you try to put them together, you end up with this mish-mash table.

Song Table

SongID	MusicianID	SongName
200	43	Sgt. Pepper's Lonely Heart's Club Band
200	44	Sgt. Pepper's Lonely Heart's Club Band
201	44	Ebony and Ivory

Musician Table

MusicianID	SongID	MusicianName
43	200	John Lennon
44	200	Paul McCartney
44	201	Paul McCartney

This has saved us nothing -- in fact, it has complicated the structure by introducing lots of redundant data to manage. The way to handle this situation is to create **two** one-to-many relationships involving a *linking* table which we'll call Song_Musician, since it links those tables. We create a one-to-many from Song to Song_Musician since one song will have 0-N musicians and then another one-to-many from Musician to Song_Musician since any one musician will be in one or more songs. The results look like the following:

Musician Table

MusicianID	MusicianName
43	John Lennon
44	Paul McCartney

Song_Musician Table

SongID	MusicianID
200	43
200	44
201	44

Song Table

SongID	SongName
200	Sgt. Pepper's Lonely Heart's Club Band
201	Ebony and Ivory

This time around, all of the redundant data is in the Song_Musician table, which are only two columns of integers. Any changes to the structure of the Song or Musician table remain independent of their relationship, which is precisely what we're after.

D. Many to One

An employee can work in only one department; this relationship is *single- valued* for employees. On the other hand, one department can have many employees; this relationship is *multi-valued* for departments. The relationship between employees (single-valued) and departments (multi-valued) is a *one-to- many* relationship.

To define tables for each one-to-many and each many-to-one relationship:

1. Group all the relationships for which the "many" side of the relationship is the same entity.
2. Define a single table for all the relationships in thegroup.

In the following example, the "many" side of the first and second relationships is "employees" so an EMPLOYEE table is defined.

Table 3. Many-to-One Relationships

Entity	**Relationship**	**Entity**
Employees	are assigned to	departments
Employees	work at	jobs
Departments	report to	(administrative) departments

In the third relationship, "departments" is on the "many" side, so a department table, DEPARTMENT, is defined.

The following tables show these different relationships. The

EMPLOYEE table:

EMPNO	WORKDEPT	JOB
000010	A00	President
000020	B01	Manager

EMPNO	WORKDEPT	JOB
000120	A00	Clerk
000130	C01	Analyst
000030	C01	Manager
000140	C01	Analyst
000170	D11	Designer

The DEPARTMENT table:

DEPTNO	ADMRDEPT
C01	A00
D01	A00
D11	D01

3. Lab Exercise

Aim: Writing SQL statements Using ORACLS /MYSQL

S/w Requirement: Oracle 10g.

Theory:

1. DDL statements [Data Definition Language].

Statements are used to define the database structure or schema. Some examples:

A. CREATE - to create objects in the database

B. ALTER - alters the structure of the database

C. DROP - delete objects from the database

D.TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed

SYNTAX:

A. create table<table_name> (column definition 1, column definition 2,…);

B. alter table <table_name> modify (column definition….); alter

table <table_name> add (column definition….);

C. drop table <table_name>;

D. truncate table< table_name>;

2. DML Statements [Data Manipulation Language]

Statements are used for managing data within schema objects. Some examples:

A.SELECT - retrieve data from the a database

B. INSERT - insert data into a table

C. UPDATE - updates existing data within a table

D. DELETE - deletes all records from a table, the space for the records remain

SYNTAX:

A. select <column_name> from <table_name>;

B. insert into <table_name> values(alist of data values);

C. update <table_name> set <column_name>;

D. delete from <table_name> where <condition>;

3. DCL Statements [Data Control Language]

A. GRANT - gives user's access privileges to database

B. REVOKE - withdraw access privileges given with the GRANT command

SYNTAX:

A grant privileges on <object_name> to <user_name>;

B. revoke privileges on <object_name> to <user_name>;

4. Transaction Control [TCL]

Transaction Control (TCL) statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.

COMMIT - save work done

ROLLBACK - restore database to original since the last COMMIT

SYNTAX:

A. commit;

B. rollback;

5. Aggregating data using group function:

A group function returns a result based on group of rows. Some are purely mathematical function.

A. Avg function :

It returns average of values of the column specified in the arguments in the column.

SYNTAX: Select avg(column_name whose avg to find) from <table_name> where condition;

B. min function:

This function will give the least value of the column present in the argument.
SYNTAX: Select min(column_name whose min to find) from <table_name> where condition;

C. max function:

This function will give the maximum value of the column present in the argument.

Select min(column_name whose max to find) from <table_name> where condition;

D. sum:

This function will give the sum of value of the column present in the argument.
SYNTAX: Select sum(column_name whose sum to find) from <table_name> where condition;

E. count:

This functionis used to count the number of rows in function .
SYNTAX: Select count (*) from <table_name> ;

4. Lab Exercise

Aim: Normalization in ORACLE.

S/w Requirement: Oracle 10g.

Theory:

A. Trivial functional dependency

A trivial functional dependency is a functional dependency of an attribute on a superset of itself. {Employee ID, Employee Address} ⟶ {Employee Address} is trivial, as is {Employee Address} ⟶ {Employee Address}.

B. Full functional dependency

An attribute is fully functionally dependent on a set of attributes X if it is

- Functionaly dependent on X, and
- not functionally dependent on any proper subset of X. {Employee Address} has a functional dependency on {Employee ID, Skill}, but not a *full* functional dependency, because it is also dependent on {Employee ID}.

C. Transitive dependency

A transitive dependency is an indirect functional dependency, one in which $X \longrightarrow Z$ only by virtue of $X \longrightarrow Y$ and $Y \longrightarrow Z$.

D. Multivalued dependency

A multivalued dependency is a constraint according to which the presence of certain rows in a table implies the presence of certain other rows.

E. Join dependency

A table *T* is subject to a join dependency if *T* can always be recreated by joining multiple tables each having a subset of the attributes of *T*.

F. Superkey

A superkey is an attribute or set of attributes that uniquely identifies

rows within a table; in other words, two distinct rows are always guaranteed to have distinct superkeys. {Employee ID, Employee Address, Skill} would be a superkey for the "Employees' Skills" table; {Employee ID, Skill} would also be a superkey.

G. Candidate key

A candidate key is a minimal superkey, that is, a superkey for which we can say that no proper subset of it is also a superkey. {Employee Id, Skill} would be a candidate key for the "Employees' Skills" table.

H. Non-prime attribute

A non-prime attribute is an attribute that does not occur in any candidate key. Employee Address would be a non-prime attribute in the "Employees' Skills" table.

F. Primary key

Most DBMSs require a table to be defined as having a single unique key, rather than a number of possible unique keys. A primary key is a key which the database designer has designated for this purpose.

1 1st NF
2 2nd NF
3 3rd NF

Normal form	**Defined by**	**Brief definition**
First normal form (1NF)	Two versions: E.F. Codd (1970), C.J. Date (2003)	Table faithfully represents a relation and has no "repeating groups"
Second normal form (2NF)	E.F. Codd (1971)	No non-prime attribute in the table is functionally dependent on a part (proper subset) of a candidate key
Third normal form (3NF)	E.F. Codd (1971); see also Carlo Zaniolo's equivalent but differently-expressed definition (1982)	Every non-prime attribute is non-transitively dependent on everykey of the table

Boyce-Codd normal form (BCNF)	Raymond F. Boyce and E.F. Codd (1974)	Every non-trivial functional dependency in the table is a dependency on a superkey
Fourth normal form (4NF)	Ronald Fagin (1977	Every non-trivial multivalued dependency in the table is a dependency on a superkey
Fifth normal form (5NF)	Ronald Fagin (1979)	Every non-trivial join dependency in the table is implied by the superkeys of the table
Domain/key normal form (DKNF)	Ronald Fagin (1981)	Every constraint on the table is a logical consequence of the table's domain constraints and key constraints
Sixth normal form (6NF)	Chris Date, Hugh Darwen , and Nikos Lorentzos (2002)	Table features no non-trivial join dependencies at all (with reference to generalized join operator

Now we ready to come to grips with the ideas of normalization. The following table, containing information about some students at Enormous State University, is a table that is in 1st Normal Form, 1NF.

You can easily verify for yourself that this table satisfies the definition of 1NF: viz., it has no duplicated rows; each cell is single-valued (i.e., there are no repeating groups or arrays); and all the entries in a given column are of the same kind.

SocialSecurity Number	FirstName	LastName	Major
123-45-6789	Jack	Jones	Library and Information Science
222-33-4444	Lynn	Lee	Library and Information Science
987-65-4321	Mary	Ruiz	Pre-Medicine
123-54-3210	Lynn	Smith	Pre-Law
111-33-5555	Jane	Jones	Library and Information Science

Table 1

In Table 1 we can see that the key, SSN, functionally determines the other attributes; i.e., a given Social Security Number implies (determines) a particular value for each of the attributes FirstName, LastName, and Major (assuming, at least for the moment, that a student is allowed to have only one major). In the arrow notation: SSN ? FirstName, SSN ? LastName, and SSN ? Major.

A key attribute will, by the definition of key, uniquely determine the values of the other attributes in a table; i.e., all non-key attributes in a table will be functionally dependent on the key. However, there may be non-key attributes in a table that determine other attributes in that table. Consider the following table:

FirstName	LastName	Major	Level
Jack	Jones	LIS	Graduate
Lynn	Lee	LIS	Graduate
Mary	Ruiz	Pre- Medicine	Undergraduate
Lynn	Smith	Pre-Law	Undergraduate
Jane	Jones	LIS	Graduate

Table 2

In Table 2 the Level attribute can be said to be functionally dependent on the Major attribute. Thus, we have an example of an attribute that is functionally dependent on a non-key attribute. This statement is true in the table *per se*, and that is all that the definition of functional dependence requires; but the statement also reflects the real-world fact that Library and Information Science is a major that is open only to graduate students and that Pre-Medicine and Pre-Law are majors that are open only to undergraduatestudents.

The 2nd Normal Form (2NF)

Table 2 has another interesting aspect. Its key is a composite key, consisting of the paired attributes, FirstName and LastName. The Level attribute is functionally dependent on this composite key, of course; but, in addition, Level can be seen to be dependent on only the attribute LastName. (This is true because each value of Level is paired with a distinct value of LastName. In contrast, there are two occurrences of the value Lynn for the attribute FirstName, and the two Lynns are paired with different values of Level, so Level is not functionally dependent on FirstName.) Thus this table fails to qualify as a 2nd Normal Form table, since the definition of 2NF requires that all non-key attributes be dependent on the entire key. (Admittedly, this example of a partial dependency is artificially contrived, but nevertheless it illustrates theproblem of partial dependency.)

We can turn Table 2 into a table in 2NF in an easy way, by adding a column for the Social Security Number, which will then be the natural thing to use as the key.

SSN	FirstName	LastName	Major	Level
123- 45-6789	Jack	Jones	LIS	Graduate
222-33-4444	Lynn	Lee	LIS	Graduate
987-65-4321	Mary	Ruiz	Pre-Medicine	Undergraduate
123-54-3210	Lynn	Smith	Pre-Law	Undergraduate
111-33-5555	Jane	Jones	LIS	Graduate

Table 3

With the SSN defined as the key, Table 3 is in 2NF, as you can easily verify. This illustrates the fact that any table that is in 1NF and has a single-attribute (i.e., a non-composite) key is automatically also in 2NF.

Table 3 still exhibits some problems, however. For example, it contains some repeated information about the LIS-Graduate pairing.

Anomalies and Normalization

At this point it is appropriate to note that the main thrust behind the idea of normalizing databases is the avoidance of insertion and deletion anomalies in databases.

To illustrate the idea of anomalies, consider what would happen to our knowledge (at least, as explicitly contained in a table) of the level of the major, Pre-Medicine, if Mary Ruiz left Enormous State University. With the deletion of the row for Ms. Ruiz, we would lose the information that Pre-Medicine is an Undergraduate major. This is an example of a deletion anomaly. We may possess the real-world information that Pre-Medicine is an Undergraduate major, but no such information is explicitly contained in a table in our database.

As an example of an insertion anomaly, we can suppose that a new student wants to enroll in ESU: e.g., suppose Jane Doe wants to major in Public Affairs. From the information in Table 3 we cannot tell whether Public Affairs is an Undergraduate or a Graduate major; in fact, we do not even know whether Public Affairs is an established major at ESU. We do not know whether it is permissible to insert the value, Public Affairs, as a value of the attribute, Major, or what to insert for the attribute, Level, if we were to assume that Public Affairs is a valid value for Major. The point is that while we may possess real-world information about whether Public Affairs is a major at ESU and what its level is, this information is not explicitly contained in any table that we have thus far mentioned as part of our database.

A database-management system, a DBMS, can work only with the information that we put explicitly into its tables for a given database and into its rules for working with those tables, where such rules are appropriate and possible.

How do anomalies relate to normalization? The simple answer is that by arranging that the tables in a database are sufficiently normalized (in practice, this typically means to at least the 4th level of normalization), we can ensure that anomalies will not arise in our database. Anomalies are difficult to avoid directly, because with databases of typical complexity (i.e., several tables) the database designer can easily overlook possible problems. Normalization offers a rigorous way of avoiding unrecognized anomalies.

Normalization may look like a difficult process when one views it from the standpoint of the formal definitions of the various normal forms, as presented in Section 2 of this handout. But in practice, you can easily attain sufficient normalization in your database by simply ensuring that the tables in your database are what we can call "single-theme" tables. This idea will be illustrated as we proceed through the rest of the discussion in this handout.

Turning a Table with Anomalies (Table 3) into Single-Theme Tables

Although Table 3 is in 2NF, it is still open to the problems of insertion and deletion anomalies, as the discussion in the preceding section shows. The reason is that Table 3 deals with more than a single theme. What can we do to turn it into a set of tables that are, or at least come closer to being, single- theme tables?

A reasonable way to proceed is to note that Table 3 deals with both information about students (their names and SSNs) and information about majors and levels. This should strike you as two different themes. Presented below is one possible set of single-theme tables dealing with the information in Table 3. (To save space, the following tables also contain some information that is not in Table 3, and the discussion will deal with this added information.)

SSN	FirstName	LastName
123-45-6789	Jack	Jones
222-33-4444	Lynn	Lee
987-65-4321	Mary	Ruiz
123-45-4321	Lynn	Smith
111-33-5555	Jane	Jones
999-88-7777	Newton	Gingpoor

Table 4

Major	Level
LIS	Graduate
Pre-Medicine	Undergraduate
Pre-Law	Undergraduate
Public Affairs	Graduate

Table 5

SSN	Major
123-45-6789	LIS
222-33-4444	LIS
987-65-4321	Pre-Medicine
123-54-3210	Pre-Law
111-33-5555	LIS

Table 6

The three preceding tables should strike you as providing a better arrangement of the information in Table 3. For one thing, this arrangement puts the information about the students into a smaller table, Table 4, which happily fails to contain redundant information about the LIS-Graduate pairing. For another thing, this arrangement permits us to enter information about students (e.g., Newton Gingpoor) who have not yet identified themselves as pursuing a particular major. For still another thing, it puts the information about the Major-Level pairings into a separate table, Table 5, which can easily be expanded to include information (e.g., that the Public Affairs major is at the Graduate level) about majors for which, at the moment, there may be no

students registered. Finally, Table 6 provides the needed link between individual students and their majors (note that Newton Gingpoor's SSN is not in this Table 6, which tells us that he has not yet selected a major).

Tables 4 - 6 are single-theme tables and are in 2NF, as you can easily verify. (In fact, they are in DKNF, but we are not yet ready to discuss the latter level in detail.)

The 3rd Normal Form (3NF)

In order to discuss the 3rd Normal Form, we need to begin by discussing the idea of transitive dependencies.

In mathematics and logic, a transitive relationship is a relationship of the following form: "If A implies B, and if also B implies C, then A implies C." An example is: "If John Doe is a human, and if every human is a primate, then John Doe must be a primate." Another way of putting it is this: "If A functionally governs B, and if B functionally governs C, then A functionally governs C." In the arrow notation, we have:

$$[(A \rightarrow B) \text{ and } (B \rightarrow C)] \Rightarrow (A \rightarrow C)$$

The following table, Table 7, provides an example of how transitive dependencies can occur in a table in a relational database.

Author Last Name	Author First Name	Book Title	Subject	Collection or Library	Building
Berdahl	Robert	The Politics of the Prussian Nobility	History	PCL General Stacks	Perry-Library
Yudof	Mark	Child Abuse and Neglect	Legal Procedures	Law Library	Townes Hall
Harmon	Glynn	Human Memory and Knowledge	Cognitive Psychology	PCL General Stacks	Perry-Library
Graves	Robert	The Golden Fleece	Greek Literature	Classics Library	Waggener Hall
Miksa	Francis	Charles Ammi Cutter	Library Biography	Library and Information Science Collection	Perry-Library
Hunter	David	Music Publishing and Collecting	Music Literature	Fine Arts Library	Fine Arts Building
Graves	Robert	English and Scottish Ballads	Folksong	PCL General Stacks	Perry-Library

Table 7

What is wrong with having a transitive dependency or dependencies in a table? For one thing, there is duplicated information: from three different rows we can see that the PCL General Stacks are in the PCL building. For another thing, we have possible deletion anomalies: if the Yudof book were lost and its row removed from Table 7, we would lose the information that books on legal procedures are assigned to the Law Library and also the information the Law Library is in Townes Hall. As a third problem, we have possible insertion anomalies: if we wanted to add a chemistry book to the table, we would find that Table 7 nowhere contains the fact that the Chemistry Library is in Robert A.Welch Hall. As a fourth problem, we have the chance of making errors in updating: a careless data-entry clerk might add a book to the LISC but mistakenly enter Townes Hall in the building column.

The solution to the problem is, once again, to place the information in Table 7 into appropriate single-theme tables. Here is one such possible arrangement:

Author Last Name	Author First Name	Book Title
Berdahl	Robert	The Politics of the Prussian Nobility
Yudof	Mark	Child Abuse and Neglect
Harmon	Glynn	Human Memory and Knowledge
Graves	Robert	The Golden Fleece
Miksa	Francis	Charles Ammi Cutter
Hunter	David	Music Publishing and Collecting
Graves	Robert	English and Scottish Ballads

Table 8

Book Title	Subject
The Politics of the Prussian Nobility	History
Child Abuse and Neglect	Legal Procedures
Human Memory and Knowledge	Cognitive Psychology
The Golden Fleece	Greek Literature
Charles Ammi Cutter	Library Biography
Music Publishing and Collecting	Music Literature
English and Scottish Ballads	Folksong

Table 9

Subject	Collection or Library
History	PCL General Stacks
Legal Procedures	Law Library
Cognitive Psychology	PCL General Stacks
Greek Literature	Classics Library
Library Biography	Library and Information Science Collection
Music Literature	Fine Arts Library
Folksong	PCL General Stacks

Table 10

Collection or Library	Building
PCL General Stacks	Perry-Library
Law Library	Townes Hall
Classics Library	Waggener Hall
Library and Information Science Collection	Perry-Library
Fine Arts Library	Fine Arts Building

Table 11

You can verify for yourself that none of these tables contains a transitive dependency; hence, all of them are in 3NF (and, in fact, in DKNF).

We can note in passing that the fact that Table 8 contains the first and last names of Robert Graves in two different rows suggests that it might be worthwhile to replace it with two further tables, along the lines of:

Author Last Name	**Author First Name**	**Author Identification Number**
Berdahl	Robert	001
Yudof	Mark	002
Harmon	Glynn	003
Graves	Robert	004
Miksa	Francis	005
Hunter	David	006

Table 12

Author Identification Number	Book Title
001	The Politics of the Prussian Nobility
002	Child Abuse and Neglect
003	Human Memory and Knowledge
004	The Golden Fleece
005	Charles Ammi Cutter
006	Music Publishing and Collecting
004	English and Scottish Ballads

Table 13

Though Tables 12 and 13 together take a little more space than Table 8, it is easy to see that given a much larger collection, in which there would be many more authors with multiple works to their credit, Tables 12 and 13 would be more economical of storage space than Table 8. Furthermore, the structure of Tables 12 and 13 lessens the chance of making updating errors (e.g., typing Grave instead of Graves, or Miska instead of Miksa).

The Boyce-Codd Normal Form (BCNF)

The Boyce-Codd Normal Form (BCNF) deals with the anomalies that can occur when a table fails to have the property that every determinant is a candidate key. Here is an example, Table 14, that fails to have this property. (In Table 14 the SSNs are to be interpreted as those of students with the stated majors and advisers. Note that each of students 123-45-6789 and 987-65-4321 has two majors, with a different adviser for each major.)

SSN	Major	Adviser
123-45-6789	Library and Information Science	Dewey
123-45-6789	Public Affairs	Roosevelt
222-33-4444	Library and Information Science	Putnam
555-12-1212	Library and Information Science	Dewey
987-65-4321	Pre-Medicine	Semmelweis
987-65-4321	Biochemistry	Pasteur
123-54-3210	Pre-Law	Hammurabi

Table 14

We begin by showing that Table 14 lacks the required property, viz., that every determinant be a candidate key.

What are the determinants in Table 14? One determinant is the pair of attributes, SSN and Major. Each distinct pair of values of SSN and Major determines a unique value for the attribute, Adviser. Another determinant is the pair, SSN and Adviser, which determines unique values of the attribute, Major. Still another determinant is the attribute, Adviser, for each different value of Adviser determines a unique value of the attribute, Major. (These observations about Table 14 correspond to the real-world facts that each student has a single adviser for each of his or her majors, and each adviser advises in just one major.)

Now we need to examine these three determinants with respect to the question of whether they are candidate keys. The answer is that the pair, SSN and Major, is a candidate key, for each such pair uniquely identifies a row in Table
14. In similar fashion, the pair, SSN and Adviser, is a candidate key. But the determinant, Adviser, is not a candidate key, because the value Dewey occurs in two rows of the Adviser column. So Table 14 fails to meet the condition that every determinant in it be a candidate key.

It is easy to check on the anomalies in Table 14. For example, if student 987- 65-4321 were to leave Enormous State University, the table would lose the

information that Semmelweis is an adviser for the Pre-Medicine major. As another example, Table 14 has no information about advisers for students majoring in history.

As usual, the solution lies in constructing single-theme tables containing the information in Table 14. Here are two tables that will do the job.

SSN	Adviser
123-45-6789	Dewey
123-45-6789	Roosevelt
222-33-4444	Putnam
555-12-1212	Dewey
987-65-4321	Semmelweis
987-65-4321	Pasteur
123-54-3210	Hammurabi

Table 15

Major	Adviser
Library and Information Science	Dewey
Public Affairs	Roosevelt
Library and Information Science	Putnam
Pre-Medicine	Semmelweis
Biochemistry	Pasteur
Pre-Law	Hammurabi
History	Herodotus

Table 16

By way of an example of the value of separating Table 14 into single-theme tables, Table 16 includes information about at least one faculty member at ESU who could be the adviser of a student who wanted to major in history.

Tables 15 and 16 are in BCNF (in fact, they are in DKNF), since every determinant in them is also a candidate key. You can easily verify this statement if you note that the key in Table 15 is a composite key, SSN and Adviser.

5. Lab Exercise

Aim: Creating Cursor in Oracle.

S/w Requirement: Oracle 10g.

Theory:

There are two types of cursor
A. Explicit cursor
B. Implicit cursor

A. Explicit cursor:

An explicit cursor is one in which cursor name is explicitly assigned to select statement. An implicit cursor is used for all other sql statements. Processing of an explicit cursor involves four steps. Processing of an implicit cursor is taken care by PL/SQL .the declaration of the cursor is done in the declarative part of the block.

A cursor variable is a reference type. A reference type is similar to pointer. Explicit

cursor:
The set of rows return by query can contain zero or multiple rows depending upon the query defined. The rows are called active set. The cursor will point to the current row in the active set.
After declaring a cursor, we can use the following commands to control the cursor.

1. Open
2. Fetch
3. Close

```
declare
 c_rollno stud.rollno%type;
 cursor st is select rollno from stud where rollno=1; begin
 open st;
 loop
 fetch st into c_rollno;
 update stud set lastname='barde' where rollno=c_rollno; exit
 when st%NOTFOUND;
 end loop;
 dbms_output.put_line('table updated'); close st;
end;
/
```

outout:table updated

In the above cursor we are going to update the table. Name of table is stud and its attibutes are rollno which is number, firstname which is of datatype varchar2, lastname which is of datatype varchar2.
In the Declare section we declare variable that match to the attribute in the table %TYPE matches the datatype of that variable.In the above procedure c_rollno is the variable that we declare in the procedure to access the column rollno from table stud because of %type it directly provide matching of data.

In the begin section we open the cursor and
fetch the values.
Error handles by the exception section.
Close the cursor. And
end;

B. **Implicit cursor*:***
These are inbuilt cursor in oracle.

6. Lab Exercise

Aim: Creating Procedure and Functions in Oracle

S/w Requirement: Oracle 10g.

Theory:

PL/SQL supports the two type of programming:
1. Procedure.
2. Function.

Procedures are usually used to perform any specific task and functions are used to compute a value.

PROCEDURE:
The basic syntax for the creating procedure is:

CREATE OR REPLACE PRODURE procedure _name (arguments)AS/IS procedure

body;

The body of procedure is block of statements with declarative executable and exception sections.

The declarative section is located between the IS /AS keyword and BEGIN keyword.

The executable section is located between BEGIN and EXCEPTION keywords or between the BEGIN and END keywords if there is no EXCEPTION handling section.

If EXCEPTION handling is present, it is located between exception and END keywords.

Steps for Creating Procedure:

CREATE OR REPLACE PRODURE procedure _name (parameter list)AS/IS

(Declarative section)

BEGIN

(Executable section)

EXCEPTION

(Error handling or exception section).

END Procedure_name;

To execute the procedure we have to write a block of statement: Begin
Procedurename(data); End;
/
eg:

```
create or replace procedure addnewstud(
p_rollno stud.rollno%type,
p_firstname stud.firstname%type, p_lastname
stud.firstname%type) as begin
insert into stud(rollno,firstname,lastname)values(p_rollno,p_firstname,p_lastname);
end addnewstud;
/
```

output:Procedure created.

```
begin
addnewstud(2,'rohini','narwade');
end;
/
```

In the above procedure we are inserting data into the table. Here procedure name is addnewstud and take the variables like p_rollno, p_firstname, p_lastname having attribute in the table rollno, firstname, lastname respectively.in the begin section begin executable code is written. In the above procedure we are inserting the value so write the query to insert the data then end procedure by the end procedure then /.

FUNCTION:

Steps for Creating function:

CREATE OR REPLACE FUNCTION function _name (parameter list)AS/IS

Return datatype is/as

(local declaration)

BEGIN

(Executable section)

EXCEPTION

(Error handling or exception section).

END function_name;

A function has two parts, namely function specification and function body. The function specification begins with the keyword function and end with return clause. The function bodies begins with the keyword is/as and end with keyword end

```
create or replace function studentn( p_rollno
stud.rollno%type)
return boolean as v_firstname
varchar2(20); v_return
boolean;
begin
select firstname into v_firstname from stud where rollno=p_rollno; if
(firstname='seema')then
v_return =true;
else
v_return =false;
end if;
end studentn;
/
```

7. Lab Exercise

Aim: Writing packages and triggers in oracle.

S/w Requirement: Oracle 10g.

Theory:

1. Package.
2. Triggers

PACKAGES:

A package is a database object, which is an encapsulation of related PL/SQL types, subprogram, cursor, exception, variables and constants. It consists of two parts a specification and body. In a package specification we can declare types, subprogram, cursor, exception, variables and constants . A package body implements subprogram, cursor that are declare in package specification.

Package can be created with following commands:

1. Create package commands.
2. Create package body commands.

1 create package commands

CREATE PACKAGE package_name IS delaration

BEGIN

(executable statements)

END package_name;

The procedure and cursor declared in the 'create package' command is fully defined and implemented by package body, which can be achieved by using the following syntax:

CREATE PACKAGE BODY package_name IS delaration

BEGIN

(executable statements)

END packagebody_name;

In the create package and create package body commands, the keyword public and private denote the usage of object declaration in the package. Variables declare in the package body can be termed private to restrict their use in the package only. On the other hand public variables can be used in the package as well as outside the package.

Package specification:

The package specification contains public objects and types it also include subprogram. The specification contains the package resources required for our application. If a package specification declares only types, constants, variables, and exception, then they need not to include package body, because all information required for usage of types, constants, variables, and exception are specified in the specification.

Package body:
The package body contains the definition of every cursor and subprogram declare in the package specification and implements them private declaration can also be included in a package body.

Example:

1. create package commands:

```
create or replace package pack_me is
procedure order_proc(orno varchar 2);
function order_fun (ornos varchar 2 ) return
varchar2;
end pack_me;
/
```

2. create packeage body commands:

```
create or replace package body pack_me as
procedure order_proc(orno varchar 2)is
stat char (1);
begin
select ostatus into stat from order_master where
orderno= orno;
if stat= 'p' then
dbms_output.put_line('pending order'); else
dbms_output.put_line('completed order'); end if;
end order_proc;
```

```
function order_fun (ornos varchar 2 ) return
varchar2 is
icode varchar 2(5);
ocode varchar 2 (5);
qtyord number; qtydeld
number; begin
select qty_ord ,qty_deld, itemcode,ordernointo
qtyord,qtydeld,icode,ocode fromorder_detail where
order=ornos;
if qtyord<qtydeld then return
ocode;
else
return icode; end
if;
end order_fun;
end pack_me;
/
```

to execute the procedure in the package: exec
pack_me.order_no('001');

to execute the function in the package: declare
avarchar2(5); b
varchar2(5);
begin
b:=pack_me.order_fun('202');
dbms_output.put_line('the value is' || b); end;
/

Triggers:
Types of triggers

A. before
B. after
C. for each row
D. for each statement.

SYNTAX FOR TRIGGERS:
CREATE OR REPLACE TRIGGER trigger_name
BEFORE /AFTER INSERT/UPADTE/DELETE ON table_name
REFERENCING {OLD AS OLD/NEW AS NEW}
FOR EACH STATEMENT/ FOR EACH ROW when condition
PL/SQL_BLOCK
/

Before/After options:

The before/after options can be used to specify when the trigger body should be fired with respect to the triggering statement. If the user include a before option, then Oracle fires the triggers before executing the triggering statement. On the other hand if AFTER is used then , oracle files the trigger after executing the triggering statement.

For each row/ statement:

For each row/ statement option included in the 'create trigger ' syntax specifies that the triggers fires once per row . By defaults, database triggers a database triggers fires for each statement .

```
create or replace trigger orders
        before insert on order_detail for each row declare
        orno order_detail .orderno%type
begin
        select orderno into orno from order_detail where
        qty_ord <qyt_deld;
        if orno ='001' then
       raise_application_error(-200001,'enter some other name');
       end if;
       end;
       /
```

8. Design and implementation of Payroll Processing System

AIM: To create a database for payroll processing system using SQL and implement it using VB.

PROCEDURE:

1. Create a database for payroll processing which request the using SQL .
2. Establish ODBC connection
3. In the administrator tools open data source ODBC
4. Click add button and select oracle in ORA and click finish
5. A window will appear given the data source home as oracle and select source name and user id.
6. ADODC CONTROL FOR SALARY FORM
7. The above procedure must be follow except the table ,A select the table as salary
8. Write appropriate Program in form each from created in VB from each from created in VB form project.

PROGRAM:

SQL>create table emp(eno number primary key,enamr varchar(20),age number,addr varchar(20),DOB date,phno number(10));

Table created.

SQL>create table salary(eno number,edesig varchar(10),basic number,da number,hra number,pf number,mc number,met number,foreign key(eno) references emp);

Table created.

TRIGGER to calculate DA,HRA,PF,MC

```
SQL> create or replace trigger employ
2 after insert on salary
3 declare
4 cursor cur is select eno,basic from salary;
5 begin
6 for cur1 in cur loop
7 update salary set
8 hra=basic*0.1,da=basic*0.07,pf=basic*0.05,mc=basic*0.03 where hra=0;
 9 end loop;
10 end;
11 /
```

Trigger created.

PROGRAM FOR FORM 1

```
Private Sub emp_Click()
Form2.Show
End Sub
Private Sub exit_Click()
Unload Me
 End Sub
Private Sub salary_Click()
Form3.Show
```

```
End Sub
```

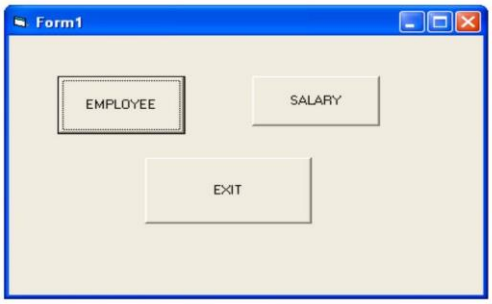
PROGRAM FOR FORM 2

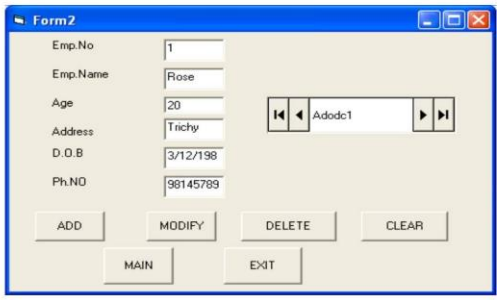
```
Private Sub add_Click()
Adodc1.Recordset.AddNew MsgBox "Record added"
End Sub
Private Sub clear_Click()
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
Text6.Text = ""
End Sub
 Private Sub delte_Click()
Adodc1.Recordset.Delete MsgBox "Record Deleted"
If Adodc1.Recordset.EOF = True Then
Adodc1.Recordset.MovePrevious
End If
 End Sub
Private Sub exit_Click()
Unload Me
 End Sub
Private Sub main_Click()
Form1.Show
End Sub
Private Sub modify_Click()
Adodc1.Recordset.Update
End Sub
```

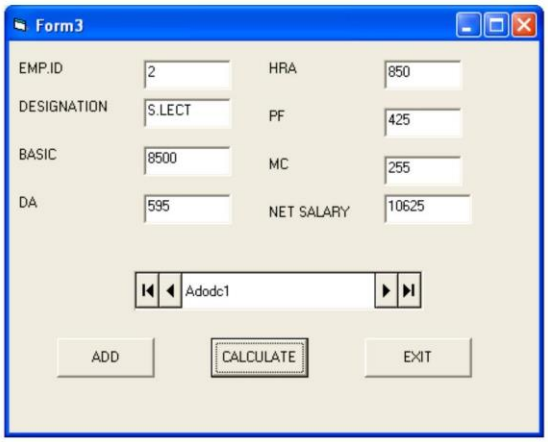
PROGRAM FOR FORM 3

```
Private Sub add_Click()
Adodc1.Recordset.AddNew MsgBox "Record added"
End Sub
 Private Sub clear_Click()
Text1.Text = "" Text2.Text = "" Text3.Text = "" Text4.Text = "" Text5.Text = "" Text6.Text = ""
End Sub
 Private Sub delte_Click()
Adodc1.Recordset.Delete MsgBox "Record Deleted"
If Adodc1.Recordset.EOF = True Then
Adodc1.Recordset.MovePrevious
End If
End Sub
Private Sub exit_Click()
Unload Me
End Sub
Private Sub main_Click()
Form1.Show
 End Sub
Private Sub modify_Click()
Adodc1.Recordset.Update
End Sub
```

Output:







RESULT: Thus payroll system was designed and implemented successfully.

9. Design and implementation of Library Information System

AIM: To create a database for Library Information System using SQL and implement it using VB.

PROCEDURE:

1. Create a database for library which request the using SQL
2. Establish ODBC connection
3. In the administrator tools open data source ODBC
4. Click add button and select oracle in ORA home 90, click finish
5. A window will appear given the data source home as oracle and select TNS source name as lion and give the used id as SWTT
6. ADODC CONTROL FOR library FORM
7. The above procedure must be follow except the table , A select the table as library
8. Write appropriate Program in form each from created in VB from each from created in VB form project.

Relational Database Schema							
Status	code	description					
Media	media_id	code					
Book	ISBN	title	author	year	dewey	price	
BookMedia	media_id	ISBN					
Customer	ID	name	addr	DOB	phone	username	password
Card	num	fines	ID				
Checkout	media_id	num	since	until			
Location	name	addr	phone				
Hold	media_id	num	name	until	queue		
Stored_In	media_id	name					
Librarian	eid	ID	Pay	name	since		
Video	title	year	director	rating	price		
VideoMedia	media_id	title	year				

```
CREATE TABLE Status ( code INTEGER, description CHAR(30), PRIMARY KEY (code) );

CREATE TABLE Media( media_id INTEGER, code INTEGER, PRIMARY KEY (media_id),
FOREIGN KEY (code) REFERENCES Status );

CREATE TABLE Book(ISBNCHAR(14), title CHAR(128), author CHAR(64), year INTEGER,
dewey INTEGER, price REAL, PRIMARY KEY (ISBN) );

CREATE TABLE BookMedia( media_id INTEGER, ISBN CHAR(14), PRIMARY KEY
(media_id), FOREIGN KEY (media_id) REFERENCES Media, FOREIGN KEY (ISBN)
REFERENCES Book);

CREATE TABLE Customer( ID INTEGER, name CHAR(64), addr CHAR(256), DOB
CHAR(10), phone CHAR(30), username CHAR(16), password CHAR(32), PRIMARY KEY
(ID), UNIQUE (username) );

CREATE TABLE Card( num INTEGER, fines REAL, ID INTEGER, PRIMARY KEY (num),
FOREIGN KEY (ID) REFERENCES Customer );

CREATE TABLE Checkout( media_id INTEGER, num INTEGER, since CHAR(10), until
CHAR(10), PRIMARY KEY (media_id), FOREIGN KEY (media_id) REFERENCES Media,
FOREIGN KEY (num) REFERENCES Card );

CREATE TABLE Location( name CHAR(64), addr CHAR(256), phone CHAR(30), PRIMARY
KEY (name) );

CREATE TABLE Hold( media_id INTEGER, num INTEGER, name CHAR(64), until
CHAR(10),
queue INTEGER, PRIMARY KEY (media_id, num), FOREIGN KEY (name) REFERENCES
Location, FOREIGN KEY (num) REFERENCES Card, FOREIGN KEY (media_id)
REFERENCES Media );

CREATE TABLE Stored_In( media_id INTEGER, name char(64), PRIMARY KEY (media_id),
FOREIGN KEY (media_id) REFERENCES Media ON DELETE CASCADE, FOREIGN KEY
(name) REFERENCES Location );

CREATE TABLE Librarian( eid INTEGER, ID INTEGER NOT NULL, Pay REAL, Loc_name
CHAR(64) NOT NULL, PRIMARY KEY (eid), FOREIGN KEY (ID) REFERENCES Customer
ON DELETE CASCADE, FOREIGN KEY (Loc_name) REFERENCES Location(name) );

CREATE TABLE Video( title CHAR(128), year INTEGER, director CHAR(64), rating REAL,
price REAL, PRIMARY KEY (title, year) );

CREATE TABLE VideoMedia( media_id INTEGER, title CHAR(128), year INTEGER,
PRIMARY KEY (media_id), FOREIGN KEY (media_id) REFERENCES Media, FOREIGN
```

KEY (title, year) REFERENCES Video);

INSERT INTO Customer(ID, name, addr, DOB, phone, username, password) VALUES (60201, 'Jason L. Gray', '2087 Timberbrook Lane, Gypsum, CO 81637', '09/09/1958', '970-273 9237', 'jlgray', 'password1');

INSERT INTO Customer(ID, name, addr, DOB, phone, username, password) VALUES (89682, 'Mary L. Prieto', '1465 Marion Drive, Tampa, FL 33602', '11/20/1961', '813-487-4873', 'mlprieto', 'password2');

INSERT INTO Customer(ID, name, addr, DOB, phone, username, password) VALUES (64937, 'Roger Hurst', '974 Bingamon Branch Rd, Bensenville, IL 60106', '08/22/1973', '847-221-4986', 'rhurst', 'password3');

INSERT INTO Customer(ID, name, addr, DOB, phone, username, password) VALUES (31430, 'Warren V. Woodson', '3022 Lords Way, Parsons, TN 38363', '03/07/1945', '731-845-0077', 'wvwoodson', 'password4');

INSERT INTO Customer(ID, name, addr, DOB, phone, username, password) VALUES (79916, 'Steven Jensen', '93 Sunny Glen Ln, Garfield Heights, OH 44125', '12/14/1968', '216-789-6442', 'sjensen', 'password5');

INSERT INTO Customer(ID, name, addr, DOB, phone, username, password) VALUES (93265, 'David Bain', '4356 Pooh Bear Lane, Travelers Rest, SC 29690', '08/10/1947', '864-610-9558', 'dbain', 'password6');

INSERT INTO Customer(ID, name, addr, DOB, phone, username, password) VALUES (58359, 'Ruth P. Alber', '3842 Willow Oaks Lane, Lafayette, LA 70507', '02/18/1976', '337-316-3161', 'rpalber', 'password7');

INSERT INTO Customer(ID, name, addr, DOB, phone, username, password) VALUES (88564, 'Sally J. Schilling', '1894 Wines Lane, Houston, TX 77002', '07/02/1954', '832-366-9035', 'sjschilling', 'password8');

INSERT INTO Customer(ID, name, addr, DOB, phone, username, password) VALUES (57054, 'John M. Byler', '279 Raver Croft Drive, La Follette, TN 37766', '11/27/1954', '423-592-8630', 'jmbyler', 'password9');

INSERT INTO Customer(ID, name, addr, DOB, phone, username, password) VALUES (49312, 'Kevin Spruell', '1124 Broadcast Drive, Beltsville, VA 20705', '03/04/1984', '703-953-1216', 'kspruell', 'password10');

INSERT INTO Card(num, fines, ID) VALUES (5767052, 0.0, 60201); INSERT INTO Card(num, fines, ID) VALUES (5532681, 0.0, 60201);

INSERT INTO Card(num, fines, ID) VALUES (2197620, 10.0, 89682);

```
INSERT INTO Card(num, fines, ID) VALUES ( 9780749, 0.0, 64937);
INSERT INTO Card(num, fines, ID) VALUES ( 1521412, 0.0, 31430);
INSERT INTO Card(num, fines, ID) VALUES ( 3920486, 0.0, 79916);
INSERT INTO Card(num, fines, ID) VALUES ( 2323953, 0.0, 93265);
INSERT INTO Card(num, fines, ID) VALUES ( 4387969, 0.0, 58359);
INSERT INTO Card(num, fines, ID) VALUES ( 4444172, 0.0, 88564);
INSERT INTO Card(num, fines, ID) VALUES ( 2645634, 0.0, 57054);
INSERT INTO Card(num, fines, ID) VALUES ( 3688632, 0.0, 49312);

INSERT INTO Location(name, addr, phone) VALUES ('Texas Branch', '4832 Deercove Drive,
Dallas, TX 75208', '214-948-7102');
INSERT INTO Location(name, addr, phone) VALUES ('Illinois Branch', '2888 Oak Avenue,
Des Plaines, IL 60016', '847-953-8130');
INSERT INTO Location(name, addr, phone) VALUES ('Louisiana Branch', '2063 Washburn
Street, Baton Rouge, LA 70802', '225-346-0068');
INSERT INTO Status(code, description) VALUES (1, 'Available');
INSERT INTO Status(code, description) VALUES (2, 'In Transit');
INSERT INTO Status(code, description) VALUES (3, 'Checked Out');
INSERT INTO Status(code, description) VALUES (4, 'On Hold');
INSERT INTO Media( media_id, code) VALUES (8733, 1);
INSERT INTO Media( media_id, code) VALUES (9982, 1);
INSERT INTO Media( media_id, code) VALUES (3725, 1);
INSERT INTO Media( media_id, code) VALUES (2150, 1);
INSERT INTO Media( media_id, code) VALUES (4188, 1);
INSERT INTO Media( media_id, code) VALUES (5271, 2);
INSERT INTO Media( media_id, code) VALUES (2220, 3);
INSERT INTO Media( media_id, code) VALUES (7757, 1);
INSERT INTO Media( media_id, code) VALUES (4589, 1);
INSERT INTO Media( media_id, code) VALUES (5748, 1);
INSERT INTO Media( media_id, code) VALUES (1734, 1);
INSERT INTO Media( media_id, code) VALUES (5725, 1);
INSERT INTO Media( media_id, code) VALUES (1716, 4);
INSERT INTO Media( media_id, code) VALUES (8388, 1);
INSERT INTO Media( media_id, code) VALUES (8714, 1);

INSERT INTO Book(ISBN, title, author, year, dewey, price) VALUES ('978-0743289412',
'Lisey"s Story', 'Stephen King', 2006, 813, 10.0);

INSERT INTO Book(ISBN, title, author, year, dewey, price) VALUES ('978-1596912366',
'Restless: A Novel', 'William Boyd', 2006, 813, 10.0);

INSERT INTO Book(ISBN, title, author, year, dewey, price) VALUES ('978-0312351588',
'Beachglass', 'Wendy Blackburn', 2006, 813, 10.0);

INSERT INTO Book(ISBN, title, author, year, dewey, price) VALUES ('978-0156031561', 'The
Places In Between', 'Rory Stewart', 2006, 910, 10.0);
```

```
INSERT INTO Book(ISBN, title, author, year, dewey, price) VALUES ('978-0060583002', 'The Last Season', 'Eric Blehm', 2006, 902, 10.0);

INSERT INTO Book(ISBN, title, author, year, dewey, price) VALUES ('978-0316740401', 'Case Histories: A Novel', 'Kate Atkinson', 2006, 813, 10.0);

INSERT INTO Book(ISBN, title, author, year, dewey, price) VALUES ('978-0316013949', 'Step on a Crack', 'James Patterson, et al.'2007, 813, 10.0);

INSERT INTO Book(ISBN, title, author, year, dewey, price) VALUES ('978-0374105235', 'Long Way Gone: Memoirs of a Boy Soldier', 'Ishmael Beah', 2007, 916, 10.0);

INSERT INTO Book(ISBN, title, author, year, dewey, price) VALUES ('978-0385340229', 'Sisters', 'Danielle Steel', 2006, 813, 10.0);

INSERT INTO BookMedia(media_id, ISBN) VALUES (8733, '978-0743289412');
INSERT INTO BookMedia(media_id, ISBN) VALUES (9982, '978-1596912366');
INSERT INTO BookMedia(media_id, ISBN) VALUES (3725, '978-1596912366');
INSERT INTO BookMedia(media_id, ISBN) VALUES (2150, '978-0312351588');
INSERT INTO BookMedia(media_id, ISBN) VALUES (4188, '978-0156031561');
INSERT INTO BookMedia(media_id, ISBN) VALUES (5271, '978-0060583002');
INSERT INTO BookMedia(media_id, ISBN) VALUES (2220, '978-0316740401');
INSERT INTO BookMedia(media_id, ISBN) VALUES (7757, '978-0316013949');
INSERT INTO BookMedia(media_id, ISBN) VALUES (4589, '978-0374105235');
INSERT INTO BookMedia(media_id, ISBN) VALUES (5748, '978-0385340229');

INSERT INTO Checkout(media_id, num, since, until) VALUES (2220, 9780749, '02/15/2007', '03/15/2007');
INSERT INTO Video(title, year, director, rating, price) VALUES ('Terminator 2: Judgment Day', 1991, 'James Cameron', 8.3, 20.0);
INSERT INTO Video(title, year, director, rating, price) VALUES ('Raiders of the Lost Ark', 1981, 'Steven Spielberg', 8.7, 20.0);
INSERT INTO Video(title, year, director, rating, price) VALUES ('Aliens', 1986, 'James Cameron', 8.3, 20.0);
INSERT INTO Video(title, year, director, rating, price) VALUES ('Die Hard', 1988, 'John McTiernan', 8.0, 20.0);

INSERT INTO VideoMedia(media_id, title, year) VALUES ( 1734, 'Terminator 2: Judgment Day', 1991);

INSERT INTO VideoMedia(media_id, title, year) VALUES ( 5725, 'Raiders of the Lost Ark', 1981);

INSERT INTO VideoMedia(media_id, title, year) VALUES ( 1716, 'Aliens', 1986);
INSERT INTO VideoMedia(media_id, title, year) VALUES ( 8388, 'Aliens', 1986);
```

```
INSERT INTO VideoMedia(media_id, title, year) VALUES ( 8714, 'Die Hard', 1988);

INSERT INTO Hold(media_id, num, name, until, queue) VALUES (1716, 4444172, 'Texas
Branch', '02/20/2008', 1);
INSERT INTO Librarian(eid, ID, pay, Loc_name) Values (2591051, 88564, 30000.00, 'Texas
Branch');
INSERT INTO Librarian(eid, ID, pay, Loc_name) Values (6190164, 64937, 30000.00, 'Illinois
Branch');
INSERT INTO Librarian(eid, ID, pay, Loc_name) Values (1810386, 58359, 30000.00,
'Louisiana Branch');

INSERT INTO Stored_In(media_id, name) VALUES(8733, 'Texas Branch');
INSERT INTO Stored_In(media_id, name) VALUES(9982, 'Texas Branch');
INSERT INTO Stored_In(media_id, name) VALUES(1716, 'Texas Branch');
INSERT INTO Stored_In(media_id, name) VALUES(1734, 'Texas Branch');
INSERT INTO Stored_In(media_id, name) VALUES(4589, 'Texas Branch');
INSERT INTO Stored_In(media_id, name) VALUES(4188, 'Illinois Branch');
INSERT INTO Stored_In(media_id, name) VALUES(5271, 'Illinois Branch');
INSERT INTO Stored_In(media_id, name) VALUES(3725, 'Illinois Branch');
INSERT INTO Stored_In(media_id, name) VALUES(8388, 'Illinois Branch');
INSERT INTO Stored_In(media_id, name) VALUES(5748, 'Illinois Branch');
INSERT INTO Stored_In(media_id, name) VALUES(2150, 'Louisiana Branch');
INSERT INTO Stored_In(media_id, name) VALUES(8714, 'Louisiana Branch');
INSERT INTO Stored_In(media_id, name) VALUES(7757, 'Louisiana Branch');
INSERT INTO Stored_In(media_id, name) VALUES(5725, 'Louisiana Branch');

/* Functions needed to view information about a customer */

SELECT C.ID, C.name, C.addr, C.DOB, C.phone, C.username, nvl((SELECT 'Librarian'
FROM Librarian L WHERE L.ID = C.ID), 'Customer') AS role FROM Customer C
WHERE C.username = <user input> AND C.password = <user input>;
/* Book search for customers */

SELECT B.ISBN, B.title, B.author, B.year, (SELECT COUNT(*) FROM BookMedia BM
WHERE BM.ISBN = B.ISBN AND BM.code = 1) AS num_available FROM Book B
WHERE B.title LIKE '%<user input>%' AND B.author LIKE '%<user input>%' AND B.year <=
<user input> AND B.year >= <user input>;

/* Find all copies of a book (used for placing holds or viewing detailed information). */

SELECT BM.media_id, S.description, nvl((SELECT SI.name FROM Stored_In SI
WHERE SI.media_id = BM.media_id), 'none') AS name FROM BookMedia BM, Media M,
Status S WHERE BM.ISBN = <user input> AND M.media_id = BM.media_id AND S.code =
M.code;

/* Video search for customers */
```

```
SELECT V.title, V.year, V.director, V.rating (SELECT COUNT(*) FROM VideoMedia VM
WHERE VM.ID = V.ID AND VM.code = 1) AS num_available FROM Video V
WHERE V.title LIKE '%<user input>%' AND V.year <= <user input> AND V.year <= <user
input> AND V.director LIKE '%<user input>%' AND V.rating >= <user input>;

/* Find all copies of a video (used for placing holds or viewing detailed information). */

SELECT VM.media_id, S.description, nvl((SELECT SI.name
FROM Stored_In SI

WHERE SI.media_id = VM.media_id), 'none') AS name FROM VideoMedia VM, Media M,
Status S WHERE VM.title = <user input> AND VM.year = <user input> AND M.media_id =
VM.media_id AND S.code = M.code;

/* Find the status of a given media item */

SELECT S.description FROM Status S, Media M WHERE S.code = M.code AND M.media_id
= <user input>;

/* Create a new Hold */

INSERT INTO Hold(media_id, num, name, until, queue) VALUES (<user input>, <user input>,
<user input>, <user input>, nvl((SELECT MAX(H.queue) FROM Hold H
WHERE H.media_id = <user input>), 0) + 1 );

/* Cancel Hold, Step 1: Remove the entry from hold */
DELETE FROM Hold WHERE media_id = <user input> AND num = <user input>


/* Cancel Hold, Step 2: Update queue for this item */
UPDATE Hold SET queue = queue-1 WHERE media_id = <user input> AND queue > <user
input>;

/* View the customer's card(s) */

SELECT CR.num, CR.fines FROM Card CR WHERE CR.ID = <user input>;

/* View media checked out on a given card */

SELECT B.title, B.author, B.year, BM.media_id, CO.since, CO.until FROM Checkout CO,
BookMedia BM, Book B WHERE CO.num = <user input> AND CO.media_id = BM.media_id
AND B.ISBN = BM.ISBN UNION SELECT V.title, V.director, V.year, VM.media_id,
CO.since, CO.until FROM Checkout CO, VideoMedia VM, Book B WHERE CO.num = <user
input> AND CO.media_id = VM.media_id AND VM.title = V.title AND VM.year = V.year;
```

```
/* View media currently on hold for a given card */

SELECT B.title, B.author, B.year, BM.media_id, H.until, H.queue, SI.name FROM Hold H,
BookMedia BM, Book B, Stored_In SI WHERE H.num = <user input> AND H.media_id =
BM.media_id AND B.ISBN = BM.ISBN AND SI.media_id = H.media_id UNION SELECT
V.title, V.director, V.year, VM.media_id, H.until, H.queue, SI.name FROM Hold H,
VideoMedia VM, Book B, Stored_In SI WHERE H.num = <user input> AND H.media_id =
VM.media_id AND VM.title = V.title AND VM.year = V.year AND SI.media_id = H.media_id;

/* View the total amount of fines the customer has to pay */
SELECT SUM(CR.fines) FROM Card CR WHERE CR.ID = <user input>;

/* Add new customer */

INSERT INTO Customer(ID, name, addr, DOB, phone, username, password) VALUES (<user
input>, <user input>, <user input>, <user input>, <user input>, <user input>, <user input>, );

/* Find a customer */

SELECT C.ID, C.name, C.addr, C.DOB, C.phone, C.username, nvl((SELECT 'Librarian' FROM
Librarian L WHERE L.ID = C.ID), 'Customer') AS role FROM Customer C WHERE
C.username = <user input> AND C.name LIKE '%<user input>%';

/* Add new card and assign it to a customer */

INSERT INTO Card(num, fines, ID) VALUES ( <user input>, 0, <user input>);

/* Create an entry in Checkout */

INSERT INTO Checkout(media_id, num, since, until) VALUES (<user input>, <user input>,
<user input>, <user input>);

/* Remove the entry for Stored_In */

DELETE FROM Stored_In WHERE media_id = <user input>;

/* Change the status code of the media */

UPDATE Media SET code = <user input> WHERE media_id = <user input>;

/* Remove the entry from Checkout */ DELETE FROM Checkout
WHERE media_id = <user input>;

/* Create the entry in Stored_In */

INSERT INTO Stored_In(media_id, name) VALUES (<user input>, <user input>);
```

/* Find the next Hold entry for a given media */ SELECT H.num, H.name, H.until
FROM Hold H WHERE H.queue = 1 AND H.media_id = <user input>;

/* Change the Stored_In entry to the target library branch */

UPDATE Stored_In SET name = <user input> WHERE media_id = <user input>;

/* Find the customer that should be notified about book arrival */

SELECT C.name, C.phone, CR.num FROM Customer C, Card CR, Hold H WHERE H.queue = 1 AND H.name = <user input> AND H.media_id = <user input> AND CR.num = H.num AND C.ID = CR.ID;

/* Add a new entry into the Book table */

INSERT INTO Book(ISBN, title, author, year, dewey, price) VALUES (<user input>, <user input>, <user input>, <user input>, <user input>, <user input>);

/* Add a new entry into the Video table */

INSERT INTO Video(title, year, director, rating, price) VALUES (<user input>, <user input>, <user input>, <user input>, <user input>);

/* Add a new Media object */

INSERT INTO Media(media_id, code) VALUES (<user input>, 1);

/* Add a new BookMedia object */

INSERT INTO BookMedia(media_id, ISBN) VALUES (<user input>, <user input>);

/* Add a new VideoMedia object */

INSERT INTO VideoMedia(media_id, title, year) VALUES (<user input>, <user input>, <user input>);

/* Remove an entry from the BookMedia table */

DELETE FROM BookMedia WHERE media_id = <user input>;

/* Remove an entry from the VideoMedia table */

DELETE FROM VideoMedia WHERE media_id = <user input>;

/* Remove an entry from the Media table */

DELETE FROM Media WHERE media_id = <user input>;

/* Remove an entry from the Book table */

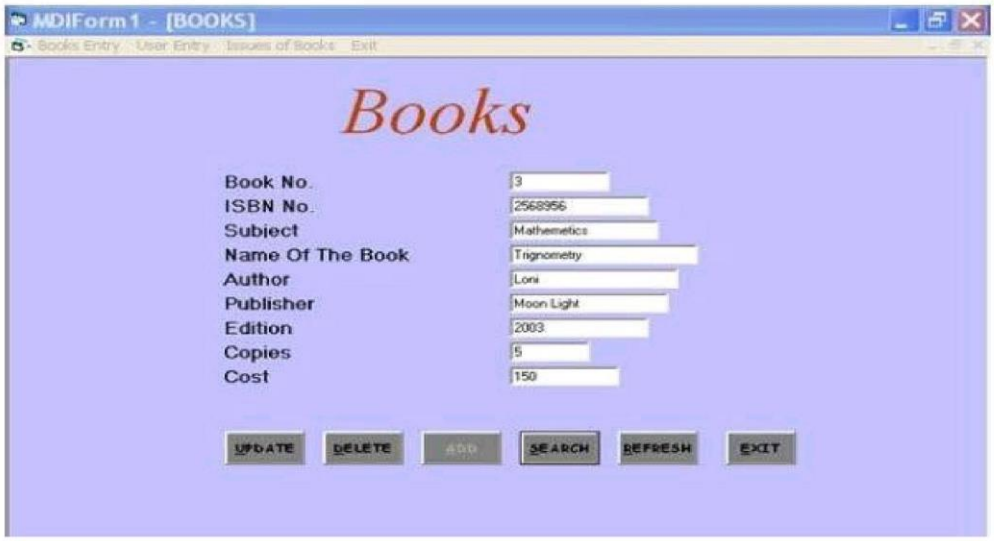
DELETE FROM Book WHERE ISBN = <user input>;

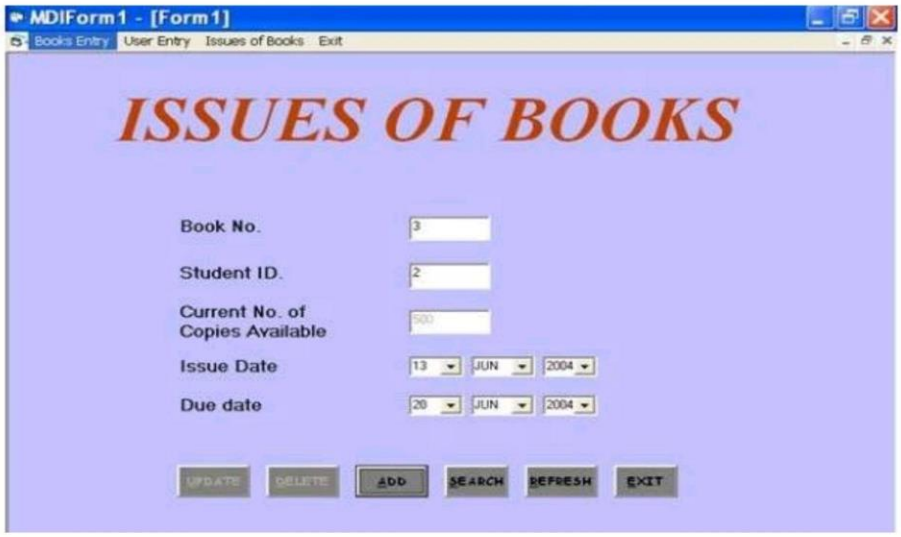
/* Remove an entry from the Video table */

DELETE FROM Video WHERE title = <user input> AND year = <user input>;

/* Update the customer's fines */
UPDATE Card SET fines = <user input> WHERE num = <user input>

Output:





RESULT- Thus the library information system was designed and implemented successfully.

10. Design and implementation of Student Information System

AIM- To create a database for Student Information System using SQL and implement it using VB.

PROCEDURE:

1. Create a database for library which request the using SQL
2. Establish ODBC connection
3. In the administrator tools open data source ODBC
4. Click add button and select oracle in ORA home 90, click finish
5. A window will appear given the data source home as oracle and select TNS source name as lion and give the used id as SWTT
6. ADODC CONTROL FOR library FORM
7. The above procedure must be follow except the table , A select the table as library
8. Write appropriate Program in form each from created in VB from each from created in VB form project.

i. ADMINISTRATOR Table

This table holds the profile information of the application super users otherwise known as system administrators. They have control of the software meaning that they can perform additional tasks that other users cannot ordinarily perform. Every software of this nature has such users and this one is no exception. The table contains the following columns; ADMIN_ID, TITLE, FRIST_NAME, LAST_NAME, and DEPARMENT_ID. The column ADMIN_ID is the primary key column (primary key disallows duplicate values and nulls in a column) every table should have a primary key column, as this acts like table indexing.

ii. ALL_COURCES Table

This table keeps the courses offered by students in different departments in the school. The table contains the following columns; COURCE_ID, COURCE_TITLE, and COURCE_CODE. The COURCE_ID is the primary key column.

iii. APP_USER_A Table

This table contains application login details for application administrators. The table columns are; USRNAME, PASSWD and ADMIN_ID. The column ADMIN_ID is the primary key column.

iv. APP_USER_L Table

This table contains application login details for application lecturers. The table columns are; USRNAME, PASSWD and LECTURER_ID. The column LECTURER_ID is the primary key column.

v. APP_USER_S Table

This table contains application login details for application students. The table columns are; USRNAME, PASSWD and MATRIG_NO. The column MATRIG_NO is the primary key column.

vi. DEPARTMENTS Table

This table holds information about the schools departments. The table contains the following columns; DEPARTMENT_ID and DEPARTMENT_NAME. The column DEPARTMENT_ID is the primary key column.

vii. GRADES Table

This is more like the main table in the database as all other tables relate to this table directly or in some other way. This table holds students examination records. The table contains the following columns; GRADES_ID, SESSION1, REG_NUMBER, DEPARTMENT_ID, LEVEL1, MATRIG_NO, FRIST_NAME, LAST_NAME, COURCE_CODE, GRADE, CREDIT_UNIT, SCORE, LECTURER_ID and GRADE_POINT. The column GRADES_ID is the primary key column.

viii. LECTURERS Table

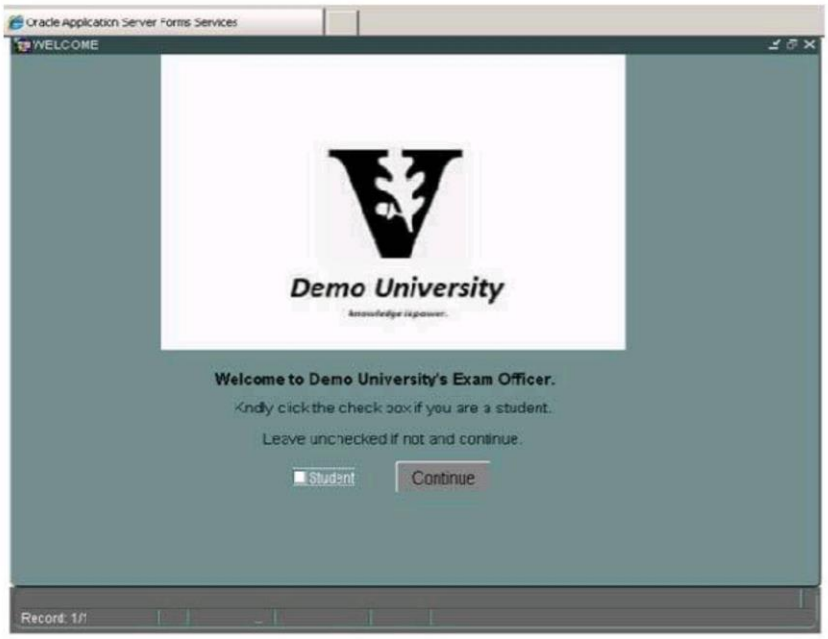
This table holds the profile information of the application lecturers. The table contains the following columns; LECTURER_ID, TITLE, FRIST_NAME, LAST_NAME, and DEPARMENT_ID. The column LECTUTER_ID is the primary key column.

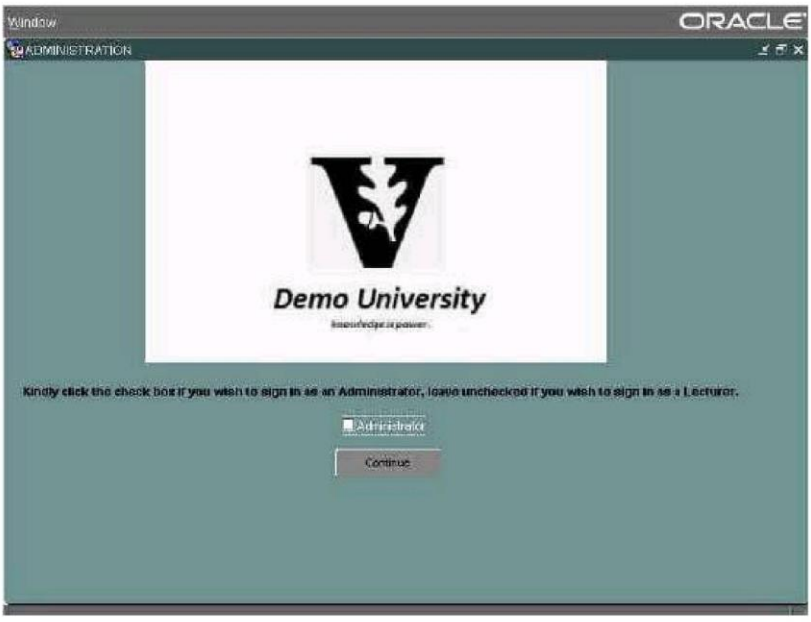
ix. REG_TABLE Table

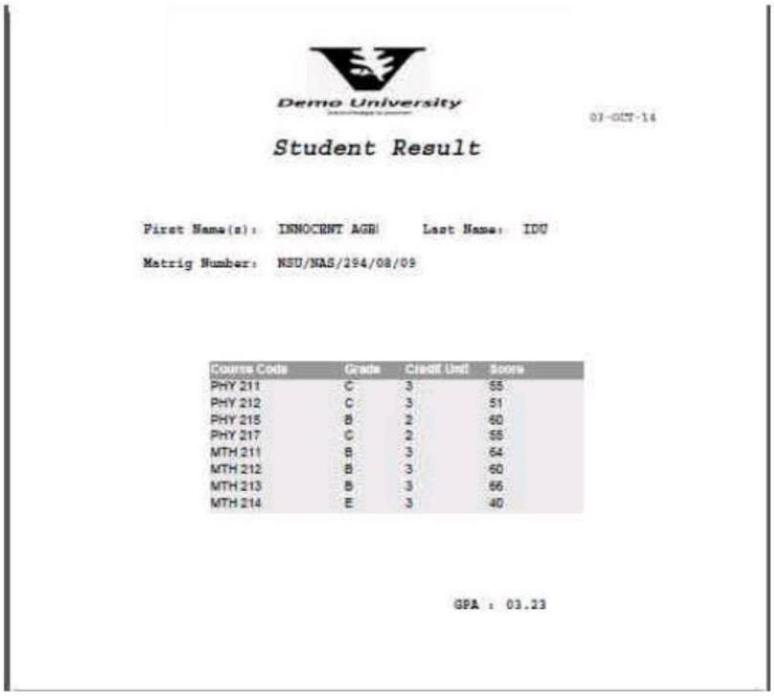
This table contains student's registration details i.e. if a student is registered for the semester this table is used to store that information. The table contains the following columns; REG_ID, REG_NUMBER, MATRIG_NO, FRIST_NAME, LAST_NAME, LEVEL1, DEPARTMENT_ID and SESSION1. The column REG_ID is the primary key column.

x. STUDENTS Table

This table holds the profile information of the application students. The table contains the following columns; MATRIG_NO, TITLE, FRIST_NAME, LAST_NAME, and DEPARMENT_ID. The column MATRIG_NO is the primary key column.







Demo University

07-OCT-14

Student Result

First Name(s): INNOCENT AGBI Last Name: IDU

Matrig Number: NSU/NAS/294/08/09

Course Code	Grade	Credit Unit	Score
PHY 211	C	3	55
PHY 212	C	3	51
PHY 215	B	2	60
PHY 217	C	2	55
MTH 211	B	3	64
MTH 212	B	3	60
MTH 213	B	3	66
MTH 214	E	3	40

GPA : 03.23

RESULT- Thus the student information system was designed and implemented successfully.

11. Automatic Backup of files and Recovery of files

AIM:- To study about automatic backup of files and recovery.

INTRODUCTION:

Because data is the heart of the enterprise, it's crucial to protect it. And to protect organization's data, one need to implement a data backup and recovery plan. Backing up files can protect against accidental loss of user data, database corruption, hardware failures, and even natural disasters. It's our job as an administrator to make sure that backups are performed and that backup tapes are stored in a secure location.

Creating a Backup and Recovery Plan

Data backup is an insurance plan. Important files are accidentally deleted all the time. Mission-critical data can become corrupt. Natural disasters can leave office in ruin. With a solid backup and recovery plan, one can recover from any of these.

Figuring Out a Backup Plan

It takes time to create and implement a backup and recovery plan. We'll need to figure out what data needs to be backed up, how often the data should be backed up, and more. To help, we create a plan, consider the following:

- How important is the data on systems? The importance of data can go a long way in helping to determine if one need to back it up—as well as when and how it should be backed up. For critical data, such as a database, one'll want to have redundant backup sets that extend back for several backup periods. For less important data, such as daily user files, we won't need such an elaborate backup plan, but 'll need to back up the data regularly and ensure that the data can be recovered easily.
- What type of information does the data contain? Data that doesn't seem important to we may be very important to someone else. Thus, the type of information the data contains can help we determine if we need to back up the data—as well as when and how the data should be backed up.
- How often does the data change? The frequency of change can affect our decision on how often the data should be backed up. For example, data that changes daily should be backed up daily.
- How quickly do we need to recover the data? Time is an important factor in creating a backup plan. For critical systems, we may need to get back online swiftly. To do this, we may need to alter our backup plan.
- Do we have the equipment to perform backups? We must have backup hardware to

perform backups. To perform timely backups, we may need several backup devices and several sets of backup media. Backup hardware includes tape drives, optical drives, and removable disk drives. Generally, tape drives are less expensive but slower than other types of drives.

- Who will be responsible for the backup and recovery plan? Ideally, someone should be a primary contact for the organization's backup and recovery plan. This person may also be responsible for performing the actual backup and recovery of data.
- What is the best time to schedule backups? Scheduling backups when system use is as low as possible will speed the backup process. However, we can't always schedule backups for off-peak hours. So we'll need to carefully plan when key system data is backed up.
- Do we need to store backups off-site? Storing copies of backup tapes off-site is essential to recovering our systems in the case of a natural disaster. In our off-site storage location, we should also include copies of the software we may need to install to reestablish operational systems.

The Basic Types of Backup

There are many techniques for backing up files. The techniques use will depend on the type of data we're backing up, how convenient we want the recovery process to be, and more.

If we view the properties of a file or directory in Windows Explorer, we'll note an attribute called Archive. This attribute often is used to determine whether a file or directory should be backed up. If the attribute is on, the file or directory may need to be backed up. The basic types of backups we can perform include

- Normal/full backups All files that have been selected are backed up, regardless of the setting of the archive attribute. When a file is backed up, the archive attribute is cleared. If the file is later modified, this attribute is set, which indicates that the file needs to be backed up.
- Copy backups All files that have been selected are backed up, regardless of the setting of the archive attribute. Unlike a normal backup, the archive attribute on files isn't modified. This allows us to perform other types of backups on the files at a later date.
- Differential backups Designed to create backup copies of files that have changed since the last normal backup. The presence of the archive attribute indicates that the file has been modified and only files with this attribute are backed up. However, the archive attribute on files isn't modified. This allows to perform other types of backups on the files at a later date.
- Incremental backups Designed to create backups of files that have changed since the

most recent normal or incremental backup. The presence of the archive attribute indicates that the file has been modified and only files with this attribute are backed up. When a file is backed up, the archive attribute is cleared. If the file is later modified, this attribute is set, which indicates that the file needs to be backed up.

- Daily backups Designed to back up files using the modification date on the file itself. If a file has been modified on the same day as the backup, the file will be backed up. This technique doesn't change the archive attributes of files.

In we backup plan we'll probably want to perform full backups on a weekly basis and supplement this with daily, differential, or incremental backups. We may also want to create an extended backup set for monthly and quarterly backups that includes additional files that aren't being backed up regularly.

Tip We'll often find that weeks or months can go by before anyone notices that a file or data source is missing. This doesn't mean the file isn't important. Although some types of data aren't used often, they're still needed. So don't forget that we may also want to create extra sets of backups for monthly or quarterly periods, or both, to ensure that we can recover historical data over time.

Differential and Incremental Backups

The difference between differential and incremental backups is extremely important. To understand the distinction between them. As it shows, with differential backups we back up all the files that have changed since the last full backup (which means that the size of the differential backup grows over time). With incremental backups, we only back up files that have changed since the most recent full or incremental backup (which means the size of the incremental backup is usually much smaller than a full backup).

Once we determine what data we're going to back up and how often, we can select backup devices and media that support these choices. These are covered in the next section.

Selecting Backup Devices and Media

Many tools are available for backing up data. Some are fast and expensive. Others are slow but very reliable. The backup solution that's right for our organization depends on many factors, including

- Capacity The amount of data that we need to back up on a routine basis. Can the backup hardware support the required load given our time and resource constraints?
- Reliability The reliability of the backup hardware and media. Can we afford to sacrifice reliability to meet budget or time needs?
- Extensibility The extensibility of the backup solution. Will this solution meet our needs as the organization grows?

- Speed The speed with which data can be backed up and recovered. Can we afford to sacrifice speed to reduce costs?
- Cost The cost of the backup solution. Does it fit into our budget?

Common Backup Solutions

Capacity, reliability, extensibility, speed, and cost are the issues driving our backup plan. If we understand how these issues affect our organization, we'll be on track to select an appropriate backup solution. Some of the most commonly used backup solutions include

- Tape drives Tape drives are the most common backup devices. Tape drives use magnetic tape cartridges to store data. Magnetic tapes are relatively inexpensive but aren't highly reliable. Tapes can break or stretch. They can also lose information over time. The average capacity of tape cartridges ranges from 100 MB to 2 GB. Compared with other backup solutions, tape drives are fairly slow. Still, the selling point is the low cost.
- Digital audio tape (DAT) drives DAT drives are quickly replacing standard tape drives as the preferred backup devices. DAT drives use 4 mm and 8 mm tapes to store data. DAT drives and tapes are more expensive than standard tape drives and tapes, but they offer more speed and capacity. DAT drives that use 4 mm tapes can typically record over 30 MB per minute and have capacities of up to 16 GB. DAT drives that use 8 mm tapes can typically record more than 10 MB per minute and have capacities of up to 36 GB (with compression).
- Auto-loader tape systems Auto-loader tape systems use a magazine of tapes to create extended backup volumes capable of meeting the high-capacity needs of the enterprise. With an auto-loader system, tapes within the magazine are automatically changed as needed during the backup or recovery process. Most auto-loader tape systems use DAT tapes. The typical system uses magazines with between 4 and 12 tapes. The main drawback to these systems is the high cost.
- Magnetic optical drives Magnetic optical drives combine magnetic tape technology with optical lasers to create a more reliable backup solution than DAT. Magnetic optical drives use 3.5-inch and 5.25-inch disks that look similar to floppies but are much thicker. Typically, magnetic optical disks have capacities of between 1 GB and 4 GB.
- Tape jukeboxes Tape jukeboxes are similar to auto-loader tape systems. Jukeboxes use magnetic optical disks rather than DAT tapes to offer high-capacity solutions. These systems load and unload disks stored internally for backup and recovery operations. Their key drawback is the high cost.
- Removable disks Removable disks, such as Iomega Jaz, are increasingly being used

as backup devices. Removable disks offer good speed and ease of use for a single drive or single system backup. However, the disk drives and the removable disks tend to be more expensive than standard tape or DAT drive solutions.

- Disk drives Disk drives provide the fastest way to back up and restore files. With disk drives, you can often accomplish in minutes what takes a tape drive hours. So when business needs mandate a speedy recovery, nothing beats a disk drive. The drawbacks to disk drives, however, are relatively high costs and less extensibility.

Before we can use a backup device, we must install it. When we install backup devices other than standard tape and DAT drives, we need to tell the operating system about the controller card and drivers that the backup device uses. For detailed information on installing devices and drivers, see the section of Chapter 2 entitled "Managing Hardware Devices and Drivers."

Buying and Using Tapes

Selecting a backup device is an important step toward implementing a backup and recovery plan. But we also need to purchase the tapes or disks, or both, that will allow we to implement our plan. The number of tapes we need depends on how much data we'll be backing up, how often we'll be backing up the data, and how long we'll need to keep additional data sets.

The typical way to use backup tapes is to set up a rotation schedule whereby we rotate through two or more sets of tapes. The idea is that we can increase tape longevity by reducing tape usage and at the same time reduce the number of tapes we need to ensure that we have historic data on hand when necessary.

One of the most common tape rotation schedules is the 10-tape rotation. With this rotation schedule, we use 10 tapes divided into two sets of 5 (one for each weekday). As shown in Table 14-2, the first set of tapes is used one week and the second set of tapes is used the next week. On Fridays, full backups are scheduled. On Mondays through Thursdays, incremental backups are scheduled. If we add a third set of tapes, we can rotate one of the tape sets to an off-site storage location on a weekly basis.

Tip The 10-tape rotation schedule is designed for the 9 to 5 workers of the world. If we're in a 24 x 7 environment, we'll definitely want extra tapes for Saturday and Sunday. In this case, use a 14-tape rotation with two sets of 7 tapes. On Sundays, schedule full backups. On Mondays through Saturdays, schedule incremental backups.

RESULT: Thus the study of automatic backup of files was performed successfully.