

DBMS RECORD

Page No: 1

INTRODUCTION

Database Management System

This model is like a hierarchical tree structure, used to construct a hierarchy of

records in the form of nodes and branches. The data elements present in the structure have

Parent-Child relationship. Closely related information in the parent-child structure is stored

together as a logical unit. A parent unit may have many child units, but a child is restricted

to have only one parent.

The drawbacks of this model are:



The hierarchical structure is not flexible to represent all the relationship proportions,

which occur in the real world.



It cannot demonstrate the overall data model for the enterprise because of the non-

availability of actual data at the time of designing the data model.



It cannot represent the Many-to-Many relationship.

Network Model

It supports the One-To-One and One-To-Many types only. The basic objects in this

model are Data Items, Data Aggregates, Records and Sets.

It is an improvement on the Hierarchical Model. Here multiple parent-child relationships

are used. Rapid and easy access to data is possible in this model due to multiple access paths

to the data elements.

Relational Model









Does not maintain physical connection between relations

Data is organized in terms of rows and columns in a table

The position of a row and/or column in a table is of no importance

The intersection of a row and column must give a single value

Features of an RDBMS









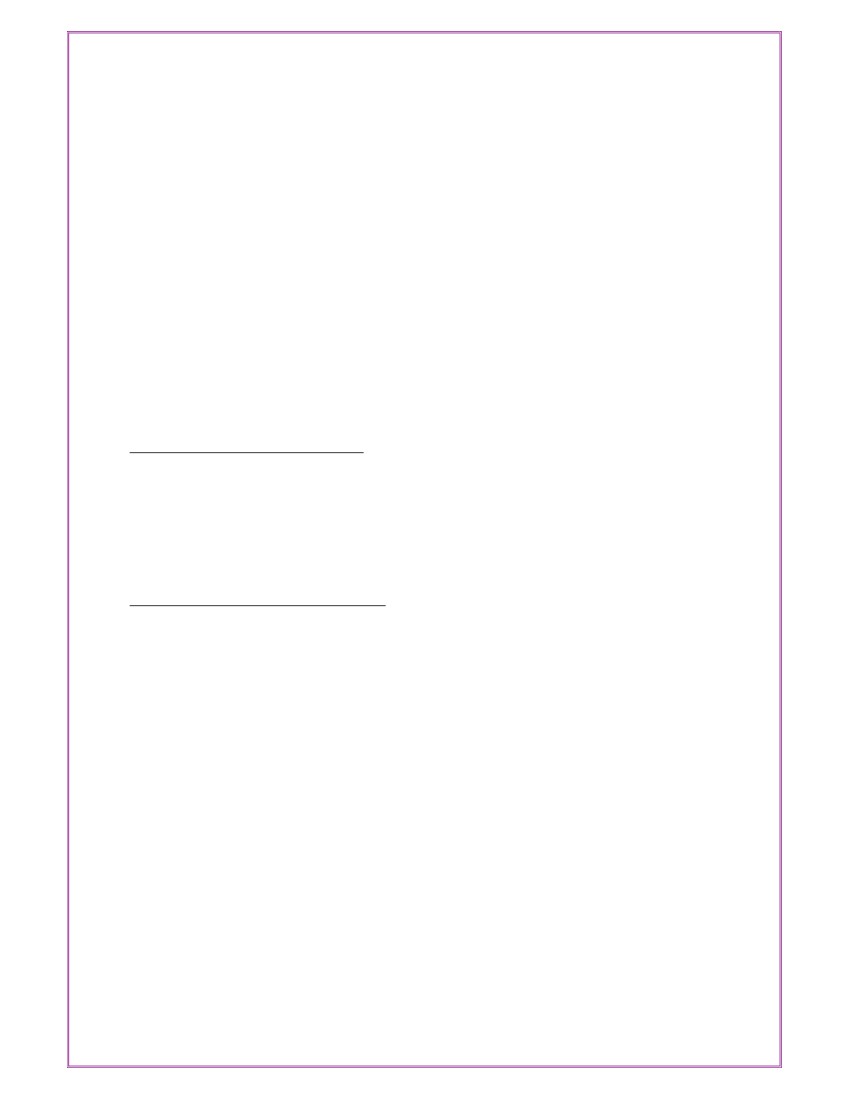
The ability to create multiple relations and enter data into them

An attractive query language

Retrieval of information stored in more than one table

An RDBMS product has to satisfy at least Seven of the 12 rules of Codd to be

accepted as a full- fledged RDBMS.



DBMS RECORD

Page No: 2

Relational Database Management System

RDBMS is acronym for Relation Database Management System. Dr. E. F. Codd first

introduced the Relational Database Model in 1970. The Relational model allows data to be

represented in a simple row- column. Each data field is considered as a column and each

record is considered as a row. Relational Database is more or less similar to Database

Management System. In relational model there is relation between their data elements. Data

is stored in tables. Tables have columns, rows and names. Tables can be related to each

other if each has a column with a common type of information. The most famous RDBMS

packages are Oracle, Sybase and Informix.

Simple example of Relational model is as follows :

Student Details Table

Roll\_no

1

2

3

Sname

Rahul

Sachin

Saurav

S\_Address

Satelite

Ambawadi

Naranpura

Student Marksheet Table

Rollno

1

2

3

Sub1

78

54

23

Sub2

89

65

78

Sub3

94

77

46

Here, both tables are based on students details. Common field in both tables is Rollno. So

we can say both tables are related with each other through Rollno column.

Degree of Relationship







One to One (1:1)

One to Many or Many to One (1:M / M: 1)

Many to Many (M: M)

The Degree of Relationship indicates the link between two entities for a specified

occurrence of each.



DBMS RECORD

Page No: 3

One to One Relationship : (1:1)

11

Student Has Roll No.

One student has only one Rollno. For one occurrence of the first entity, there can be, at the

most one related occurrence of the second entity, and vice-versa.

One to Many or Many to One Relationship: (1:M/M: 1)

1M

Course Contains Students

As per the Institutions Norm, One student can enroll in one course at a time however, in one

course, there can be more than one student.

For one occurrence of the first entity there can exist many related occurrences of the second

entity and for every occurrence of the second entity there exists only one associated

occurrence of the first.

Many to Many Relationship: (M:M)

MM

Students Appears Tests

The major disadvantage of the relational model is that a clear-cut interface cannot be

determined. Reusability of a structure is not possible. The Relational Database now accepted

model on which major database system are built.

Oracle has introduced added functionality to this by incorporated object-oriented

capabilities. Now it is known is as Object Relational Database Management System

(ORDBMS). Object- oriented concept is added in Oracle8.

Some basic rules have to be followed for a DBMS to be relational. They are known as

Codd’s rules, designed in such a way that when the database is ready for use it encapsulates

the relational theory to its full potential. These twelve rules are as follows.



DBMS RECORD

Page No: 4

E. F. Codd Rules

1. The Information Rule

All information must be store in table as data values.

2. The Rule of Guaranteed Access

Every item in a table must be logically addressable with the help of a table name.

3. The Systematic Treatment of Null Values

The RDBMS must be taken care of null values

inapplicable information.

4. The Database Description Rule

A description of database is maintained using the same logical structures with

which data was defined by the RDBMS.

5. Comprehensive Data Sub Language

According to the rule the system must support data definition, view definition,

data

manipulation,

integrity

constraints,

authorization

and

transaction

to represent missing or

management operations.

6. The View Updating Rule

All views that are theoretically updateable are also updateable by the system.

7. The Insert and Update Rule

This rule indicates that all the data manipulation commands must be

on sets of rows having a relation rather than on a single row.

8. The Physical Independence Rule

Application programs must remain unimpaired when any changes are

storage representation or access methods.

9. The Logical Data Independence Rule

The changes that are made should not affect the user’s ability to work with the

data.The change can be splitting table into many more tables.

10. The Integrity Independence Rule

The integrity constraints should store in the system catalog or in the database.

11. The Distribution Rule

The system must be access or manipulate the data that is distributed in other

operational

made in



DBMS RECORD

Page No: 5

systems.

12. The Non-subversion Rule

If a RDBMS supports a lower level language then it should not bypass any

integrity constraints defined in the higher level.

Object Relational Database Management System

Oracle8 and later versions are supported object-oriented concepts. A structure once created

can be reused is the fundamental of the OOP’s concept. So we can say Oracle8 is supported

Object Relational model, Object – oriented model both. Oracle products are based on a

concept known as a client-server technology. This concept involves segregating the

processing of an application between two systems. One performs all activities related to the

database (server) and the other performs activities that help the user to interact with the

application (client). A client or front-end database application also interacts with the

database by requesting and receiving information from database server. It acts as an

interface between the user and the database.

The database server or back end is used to manage the database tables and also respond to

client requests.

Introduction to ORACLE

ORACLE is a powerful RDBMS product that provides efficient and effective solutions for

major database features. This includes:























Large databases and space management control

Many concurrent database users

High transaction processing performance

High availability

Controlled availability

Industry accepted standards

Manageable security

Database enforced integrity

Client/Server environment

Distributed database systems

Portability



DBMS RECORD

Page No: 6





Compatibility

Connectivity

An ORACLE database system can easily take advantage of distributed processing by using

its Client/ Server architecture. In this architecture, the database system is divided into two

parts:

A front-end or a client portion

The client executes the database application that accesses database information and interacts

with the user.

A back-end or a server portion

The server executes the ORACLE software and handles the functions required for

concurrent, shared data access to ORACLE database.



DBMS RECORD

Page No: 7

ROADWAY TRAVELS

“Roadway Travels” is in business since 1977 with several buses connecting different

places in India. Its main office is located in Hyderabad.

The company wants to computerize its operations in the following areas:







Reservations

Ticketing

Cancellations

Reservations :

Reservations are directly handeled by booking office.reservations can be made 60

days in advance in either cash or credit. In case the ticket is not available,a wait listed ticket

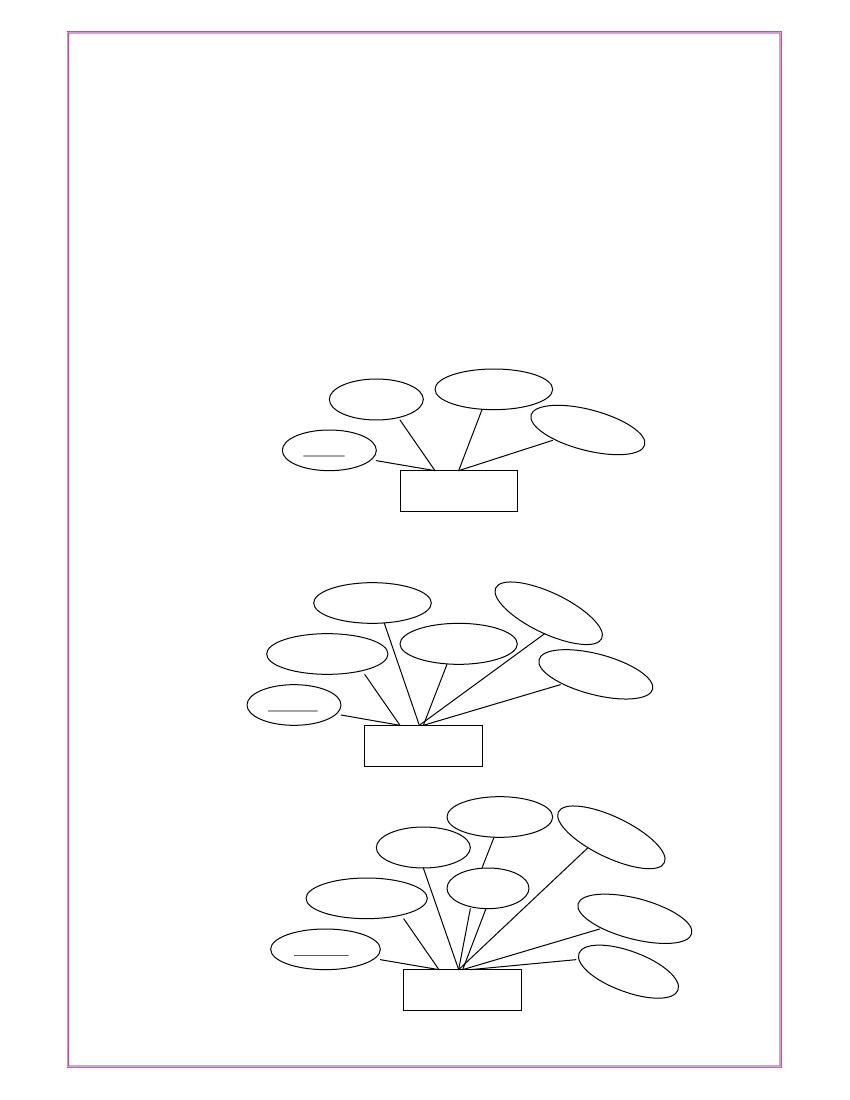
is issued to the customer. This ticket is confirmed against the cancellation.

Cancellation and modification:

Cancellations are also directly handed at the booking office. Cancellation charges

will be charged.

Wait listed tickets that do not get confirmed are fully refunded.



DBMS RECORD

Page No: 8

Exp No: 1

Date: \_ \_/\_ \_/ \_ \_

AIM: Analyze the problem and come with the entities in it. Identify what Data has to be

persisted in the databases.

The Following are the entities:

1 .Bus

2. Reservation

3. Ticket

4. Passenger

5. Cancellation

The attributes in the Entities:

Bus:( Entity)

Source

Destination

Couch Type

Bus No

Bus

Reservation (Entity)

Contact No

No-of-Seats

Address

PNR NO

Bus No

Journey date

Reservation

Ticket :(Entity)

Dep- Time

Age

Sex

Destination

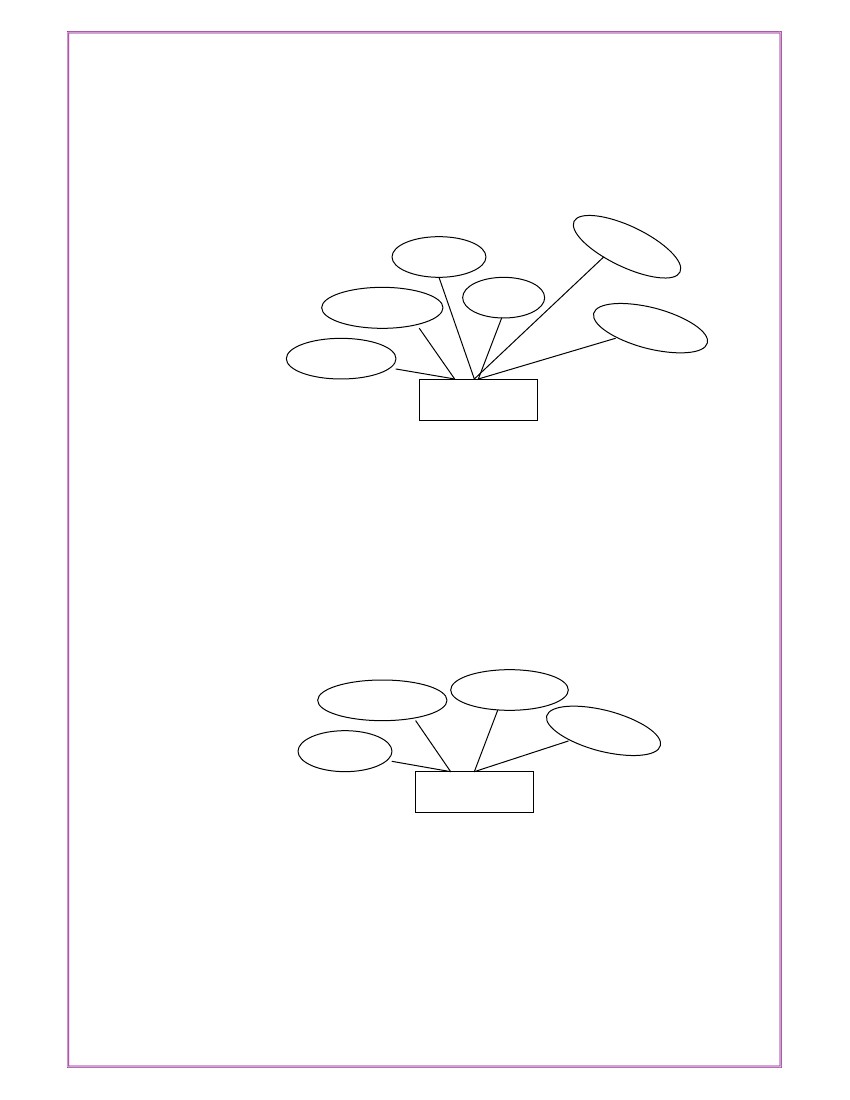
Ticket No

Bus No

Source

Journey date

Ticket



DBMS RECORD

Page No: 9

Passenger:

Age

Sex

Contact NO

PNR NO

Name

Ticket No

Passenger

Cancellation (Entity)

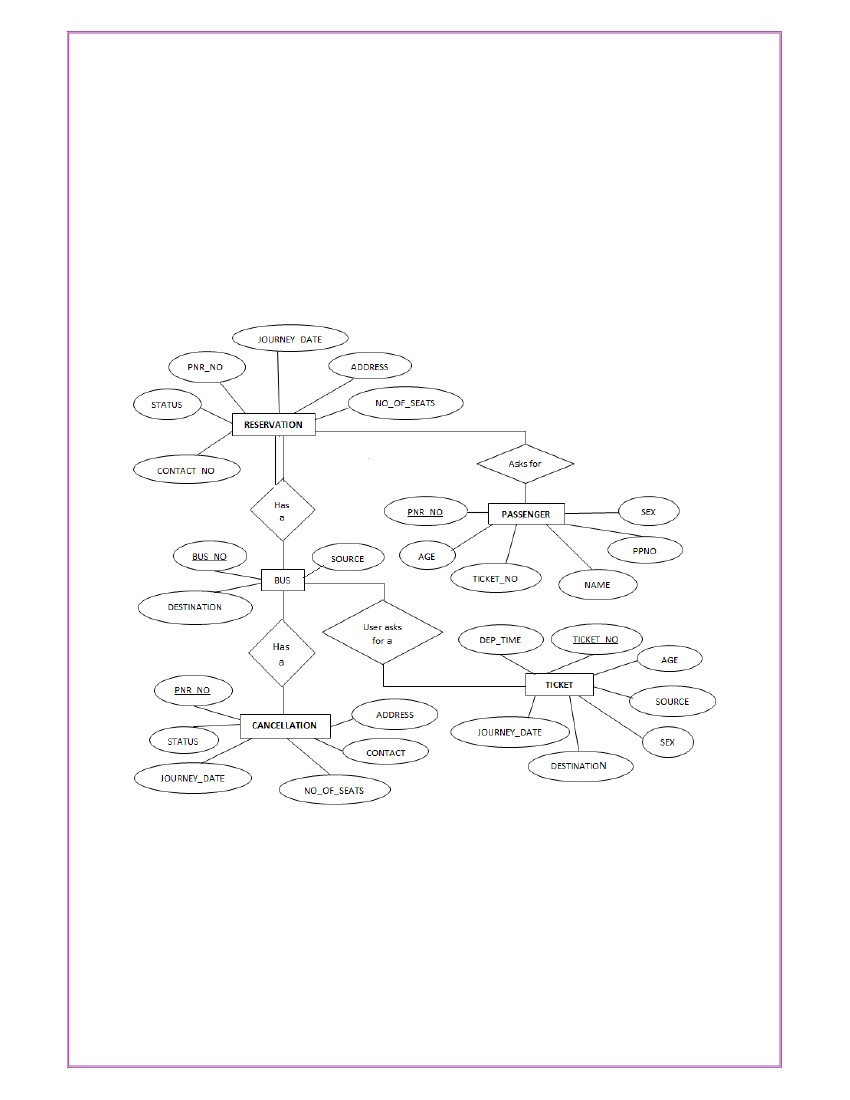
Journey date

Seat No

Contact No

PNR NO

Cancellation



DBMS RECORD

Page No: 10

Concept design with E-R Model:



DBMS RECORD

Page No: 11

Exp No: 2

Date: \_ \_/\_ \_/ \_ \_

AIM: Represent all entities in a tabular fashion. Represent all relationships in a tabular

fashion.

What is SQL and SQL\*Plus

Oracle was the first company to release a product that used the English-based Structured

Query Language or SQL. This language allows end users to manipulate information of

table(primary database object). To use SQL you need not to require any programming

experience. SQL is a standard language common to all relational databases. SQL is database

language used for storing and retrieving data from the database. Most Relational Database

Management Systems provide extension to SQL to make it easier for application developer.

A table is a primary object of database used to store data. It stores data in form of rows and

columns.

SQL\*Plus is an Oracle tool (specific program ) which accepts SQL commands and PL/SQL

blocks and executes them. SQL \*Plus enables manipulations of SQL commands and

PL/SQL blocks. It also performs additional tasks such as calculations, store and print query

results in the form of reports, list column definitions of any table, access and copy data

between SQL databases and send messages to and accept responses from the user. SQL

\*Plus is a character based interactive tool, that runs in a GUI environment. It is loaded on

the client machine.

To communicate with Oracle, SQL supports the following categories of commands:

1. Data Definition Language

Create, Alter, Drop and Truncate

2. Data Manipulation Language

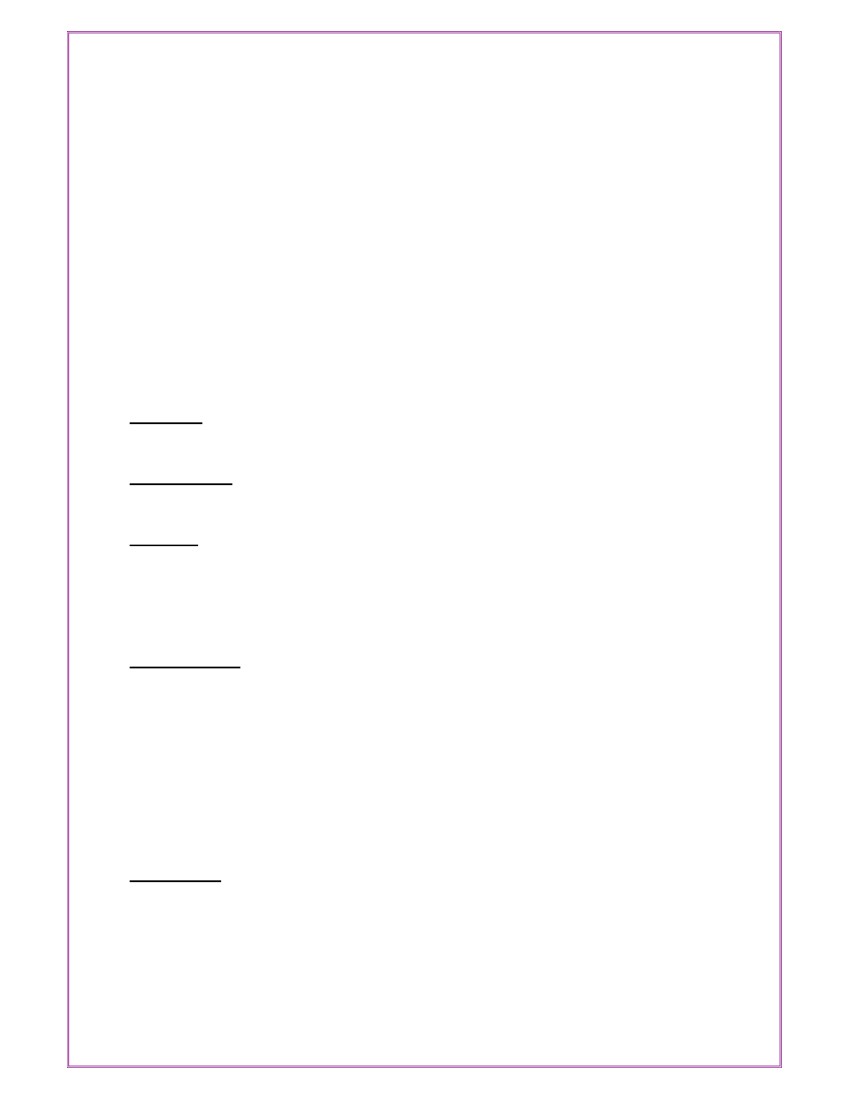
Insert, Update, Delete and Select

3. Transaction Control Language

Commit, Rollback and Save point

4. Data Control Language

Grant and Revoke



DBMS RECORD

Page No: 12

Before we take a look on above-mentioned commands we will see the data types available in

Oracle.

Oracle Internal Data types

When you create a table in Oracle, a few items should be important, not only do you have to

give each table a name(e.g. employee, customer), you must also list all the columns or fields

(e.g. First\_name, Mname, Last\_name) associated with the table. You also have to specify

what type of information thattable will hold to the database. For example, the column

Empno holds numeric information. An Oracle database can hold many different types of

data.

Data type Description

Char(Size) Stores fixed-length character data to store alphanumeric values, with a

maximum size of 2000 bytes. Default and minimum size is 1 byte.

Varchar2(Size) Stores variable-length character data to store alphanumeric values,

with maximum size of 4000 bytes.

char(Size) Stores fixed-length character data of length size characters or bytes,

depending on the choice of national character set. Maximum size if determined by the

number of bytes required storing each character with an upper limit of 2000 bytes. Default

and minimum size is 1 character or 1 byte, depending on the character set.

Nvarchar2(Size) Stores variable-length character string having maximum length size

characters or bytes, depending on the choice of national character set. Maximum size is

determined by the number of bytes required to store each character, with an upper

limit of 4000 bytes.

Long Stores variable-length character data up to 2GB(Gigabytes). Its lenth would be

restricted based on memory space available in the computer.

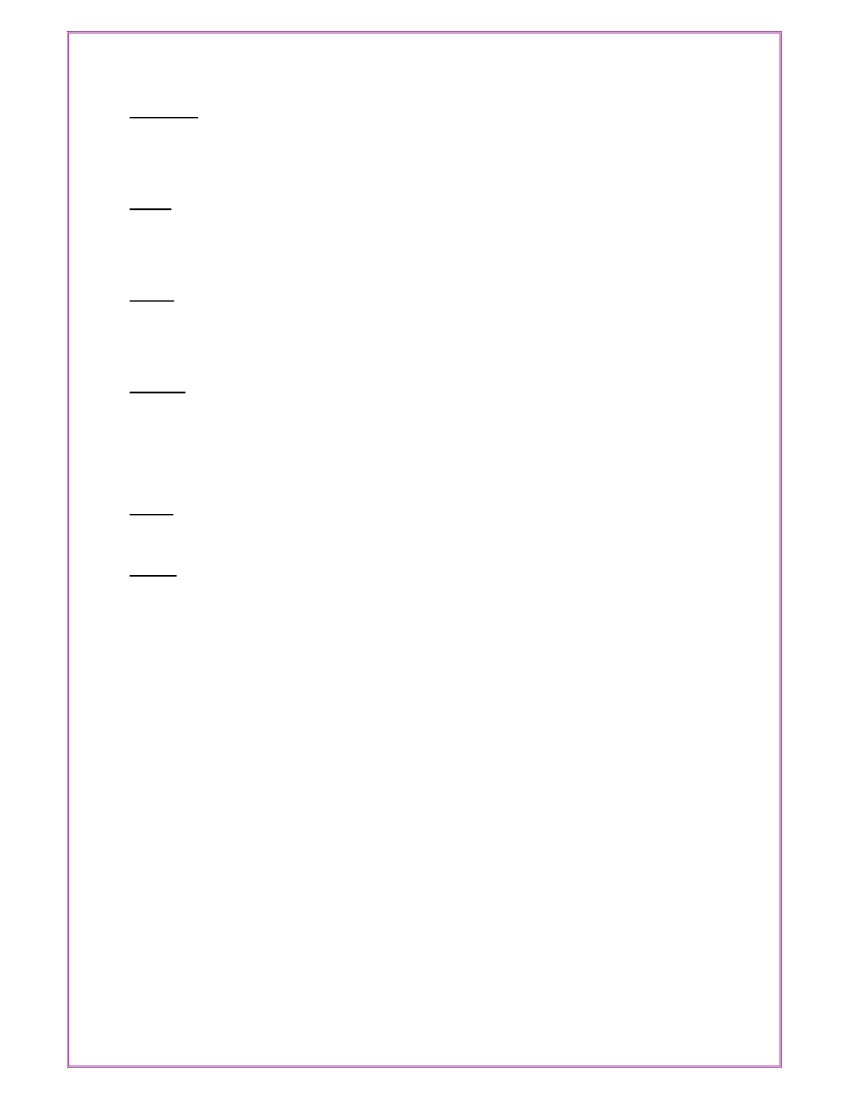
Number [p,s] Number having precision p and scale s. The precision p indicates total

number of digit varies from 1 to 38. The scale s indicates number of digit in fraction part

varies from –84 to 127.

Date Stores dates from January 1, 4712 B.C. to December 31, 4712 A.D. Oracle

predefine format of Date data type is DD-MON-YYYY.



DBMS RECORD

Page No: 13

Raw(Size) Stores binary data of length size. Maximum size is 2000 bytes. One must

have to specify size with RAW type data, because by default it does not specify any size.

Long Raw Store binary data of variable length up to 2GB(Gigabytes).

LOBS – LARGE OBJECTS

LOB is use to store unstructured information such as sound and video clips, pictures upto 4

GB size.

CLOB A Character Large Object containing fixed-width multi-byte characters.

Varying-

width character sets are not supported. Maximum size is 4GB.

NCLOB A National Character Large Object containing fixed-width multi-byte

characters.

Varying-width character sets are not supported. Maximum size is 4GB. Stores

national character set data.

BLOB To store a Binary Large Object such a graphics, video clips and sound files.

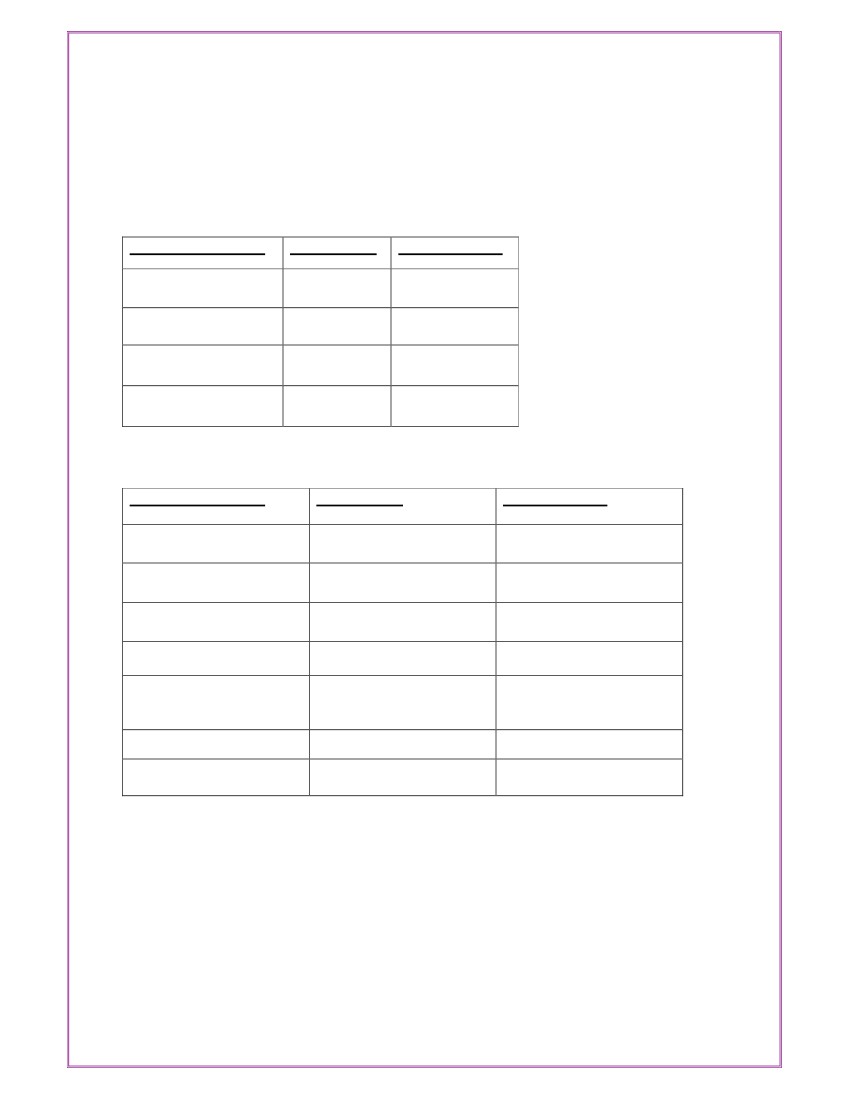
Maximum size is 4GB.

BFILE Contains a locator to a large Binary File stored outside the database. Enables

byte stream I/O access to external LOBs residing on the database server. Maximum

size is 4GB.Apart from oracle internal data types, user can create their own data type, which

is used in database and other database object. We will discuss it in the later part.



DBMS RECORD

Page No: 14

The following are tabular representation of the above entities and relationships

BUS:

COLOUMN NAME

Bus No

Source

Destination

Couch Type

DATA TYPE

varchar2(10)

varchar2(20)

varchar2(20)

varchar2(20)

CONSTRAINT

Primary Key

Reservation:

COLOUMN NAME

PNRNo

Journey date

No-of-seats

Address

Contact No

DATA TYPE

number(9)

Date

integer(8)

varchar2(50)

Number(9)

CONSTRAINT

Primary Key

BusNo

Seat no

varchar2(10)

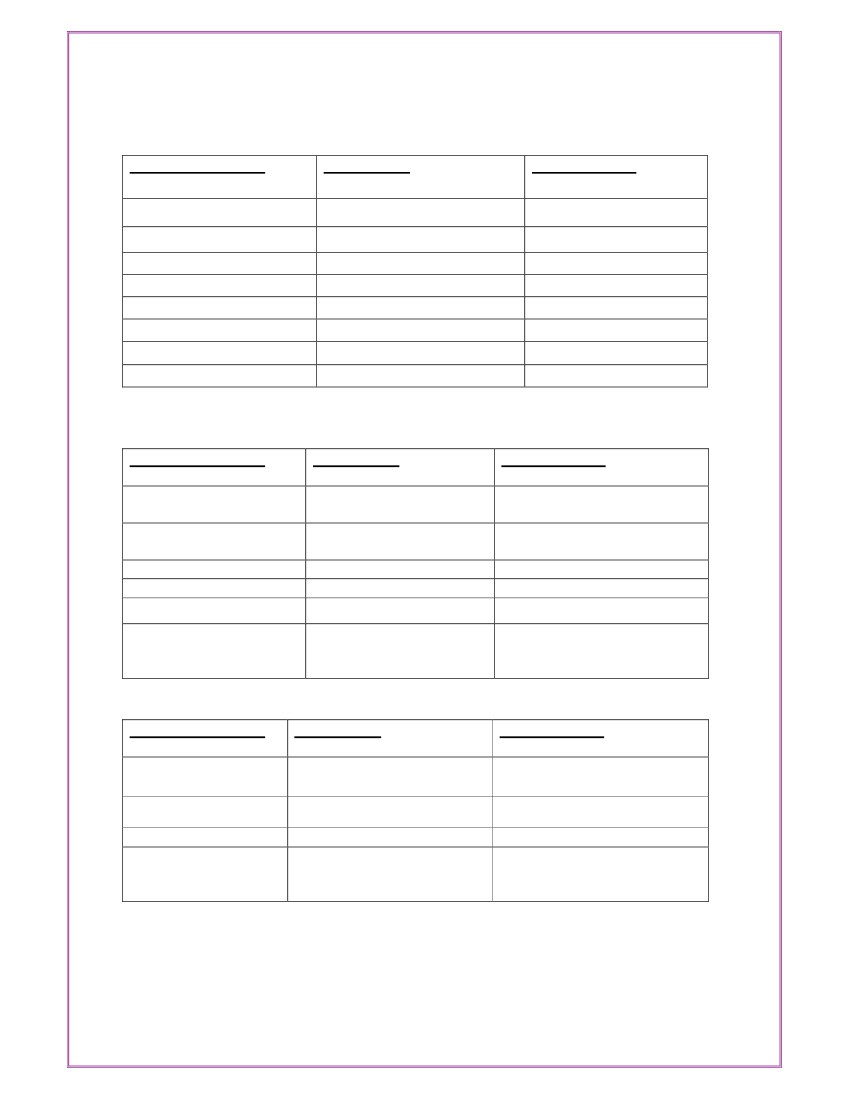
Number

Should be equal to 10

numbers and not allow

other than numeric

Foreign key



DBMS RECORD

Page No: 15

Ticket:

COLOUMN NAME

Ticket\_No

Journey date

Age

Sex

Source

Destination

Dep-time

Bus No

DATA TYPE

number(9)

Date

int(4)

Char(10)

varchar2(10)

varchar2(10)

varchar2(10)

Number2(10)

CONSTRAINT

Primary Key

Passenger:

COLOUMN NAME

PNR No

Ticket No

Name

Age

Sex

Contact no

DATA TYPE

Number(9)

Number(9)

varchar2(15)

integer(4)

char(10)

Number(9)

CONSTRAINT

Primary Key

Foreign key

(Male/Female)

Should be equal to 10 numbers

and not allow other than

numeric

Cancellation:

COLOUMN NAME

PNR No

Journey-date

Seat no

Contact\_No

DATA TYPE

Number(9)

Date

Integer(9)

Number(9)

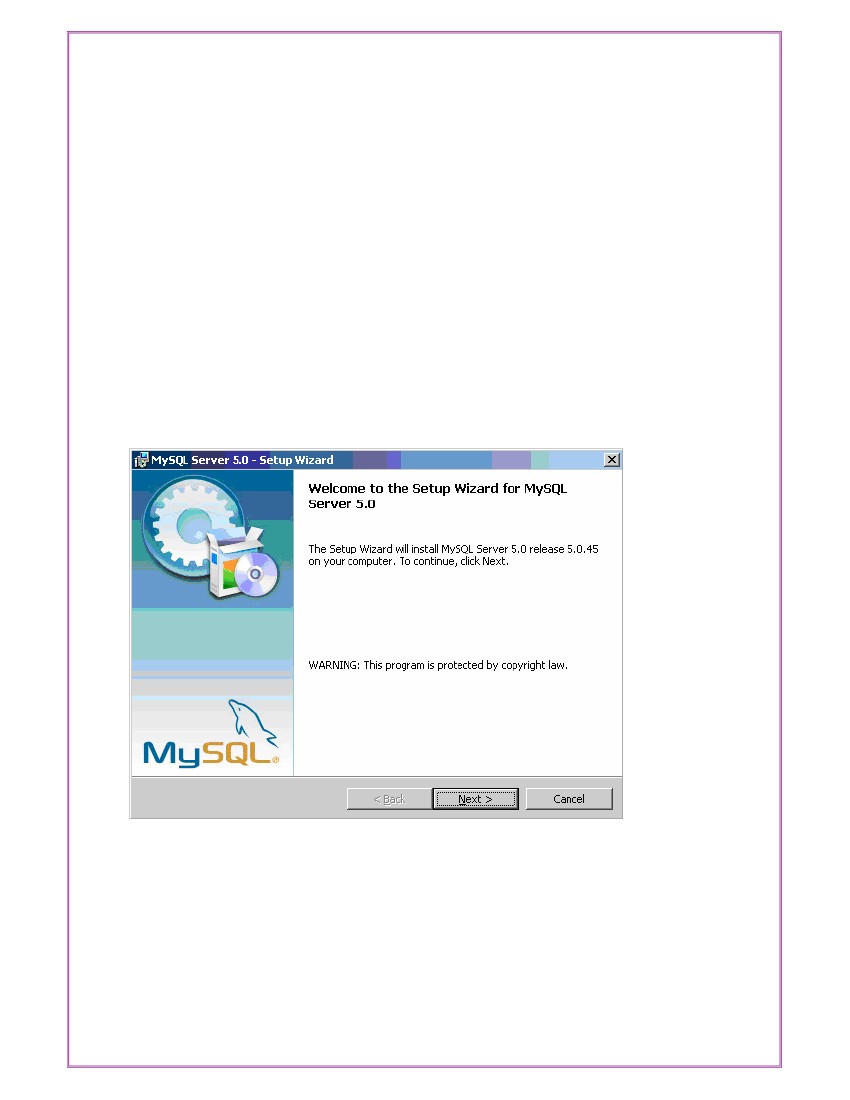
CONSTRAINT

Foriegn-key

Should be equal to 10 numbers

and not allow other than

numeric



DBMS RECORD

Page No: 16

Exp No: 3

Date: \_ \_/\_ \_/ \_ \_

AIM: Installation of MySQL and practicing DDL & DML commands.

1. Steps for installing MySQL

Step11

Make sure you already downloaded the MySQL essential 5.0.45 win32.msi file. Double

click on the .msi file.

Step22

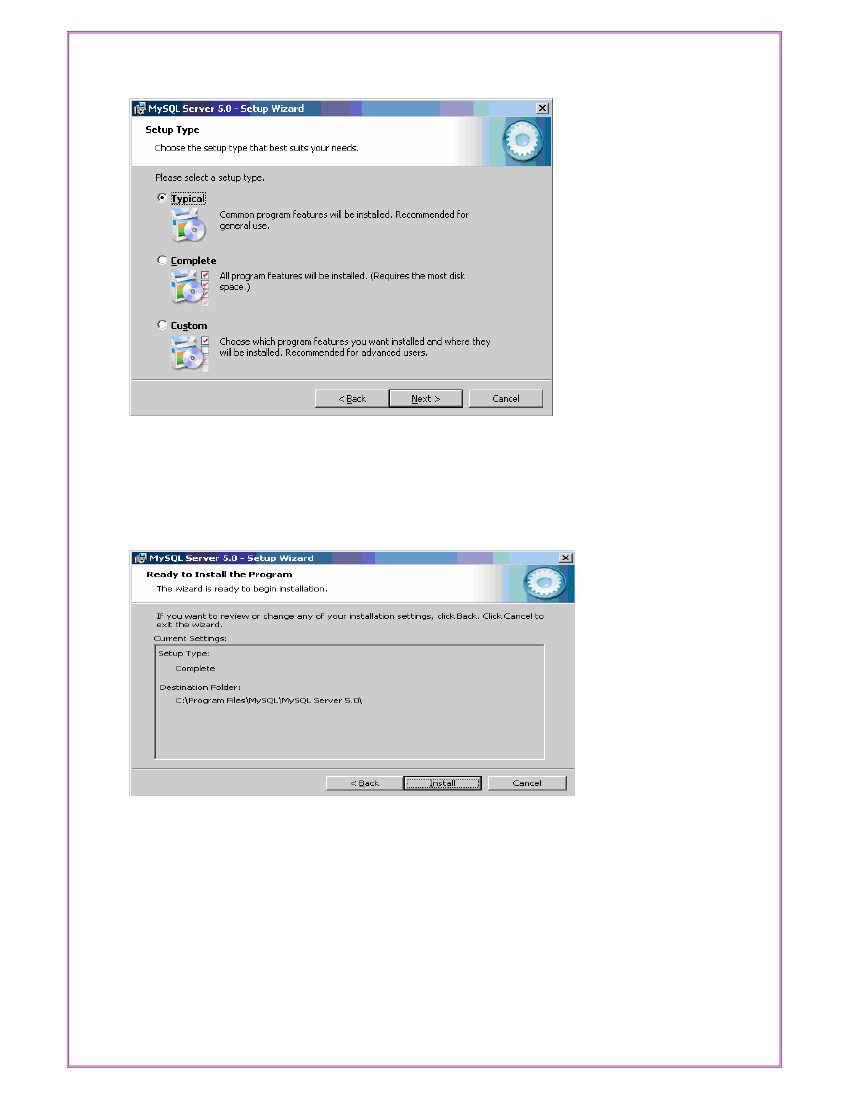
This is MySQL Server 5.0 setup wizard. The setup wizard will install MySQL Server 5.0

release 5.0.45 on your computer. To continue, click next.

Step33

Choose the setup type that best suits your needs. For common program features select Typical

and it’s recommended for general use. To continue, click next.



DBMS RECORD

Page No: 17

Step4

4

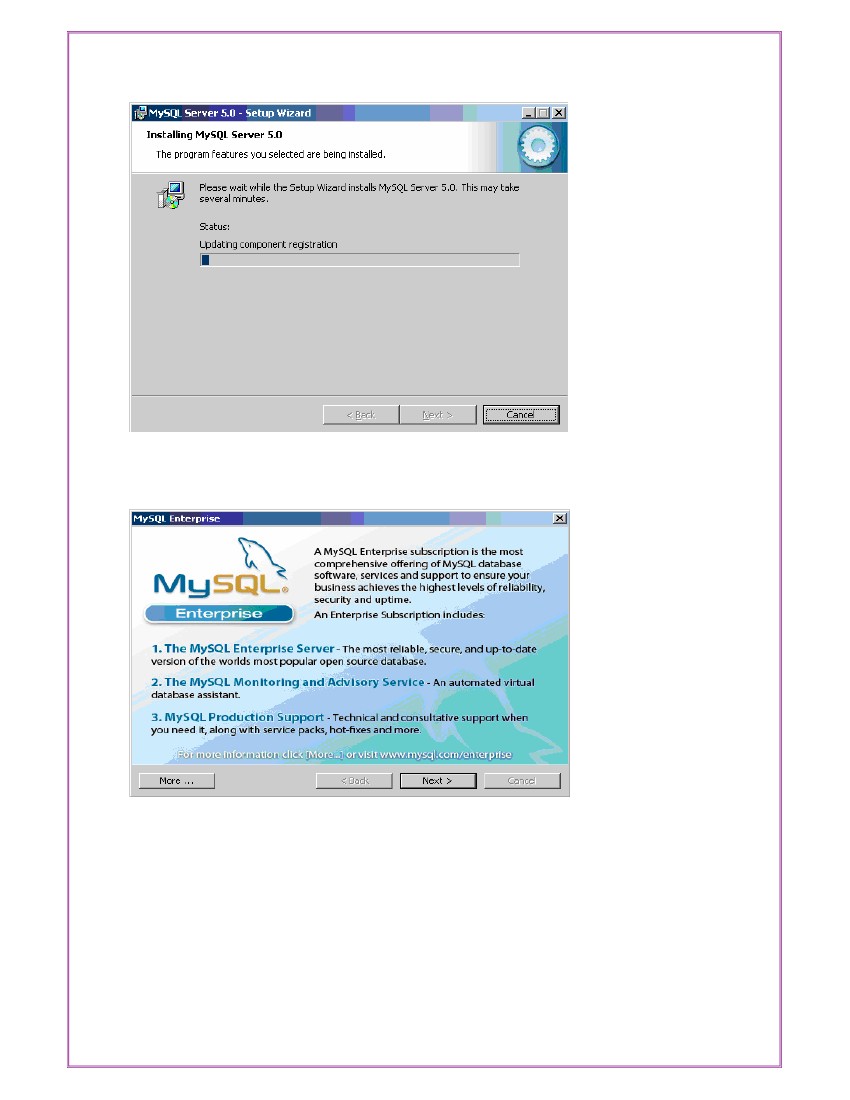
This wizard is ready to begin installation. Destination folder will be in C:\Program

Files\MySQL\MySQL Server 5.0\. To continue, click next.

Step55

The program features you selected are being installed. Please wait while the setup wizard

installs MySQL 5.0. This may take several minutes.



DBMS RECORD

Page No: 18

Step6

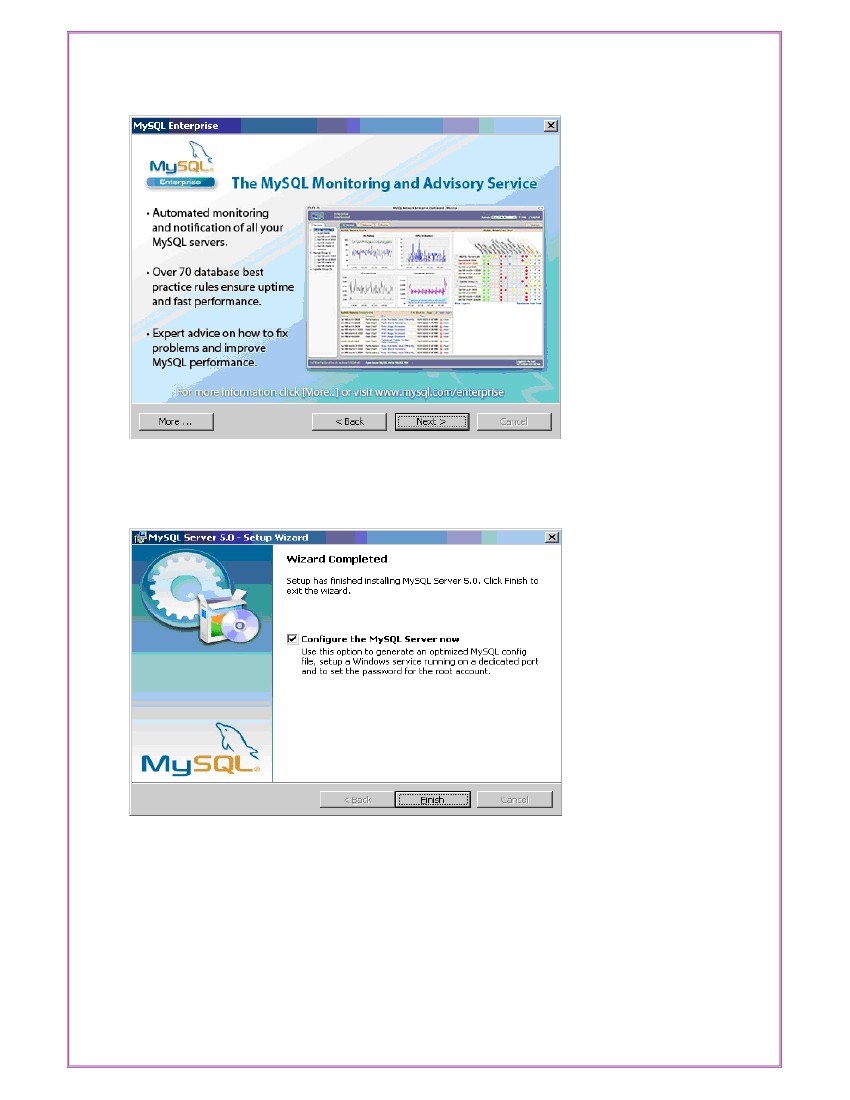
To continue, click next.

6

Step7

To continue, click next.

7



DBMS RECORD

Page No: 19

Step88

Wizard Completed. Setup has finished installing MySQL 5.0. Check the configure the

MySQL server now to continue. Click Finish to exit the wizard

d.

Step99

The configuration wizard will allow you to configure the MySQL Server 5.0 server instance.



DBMS RECORD

Page No: 20

To continue, click next.

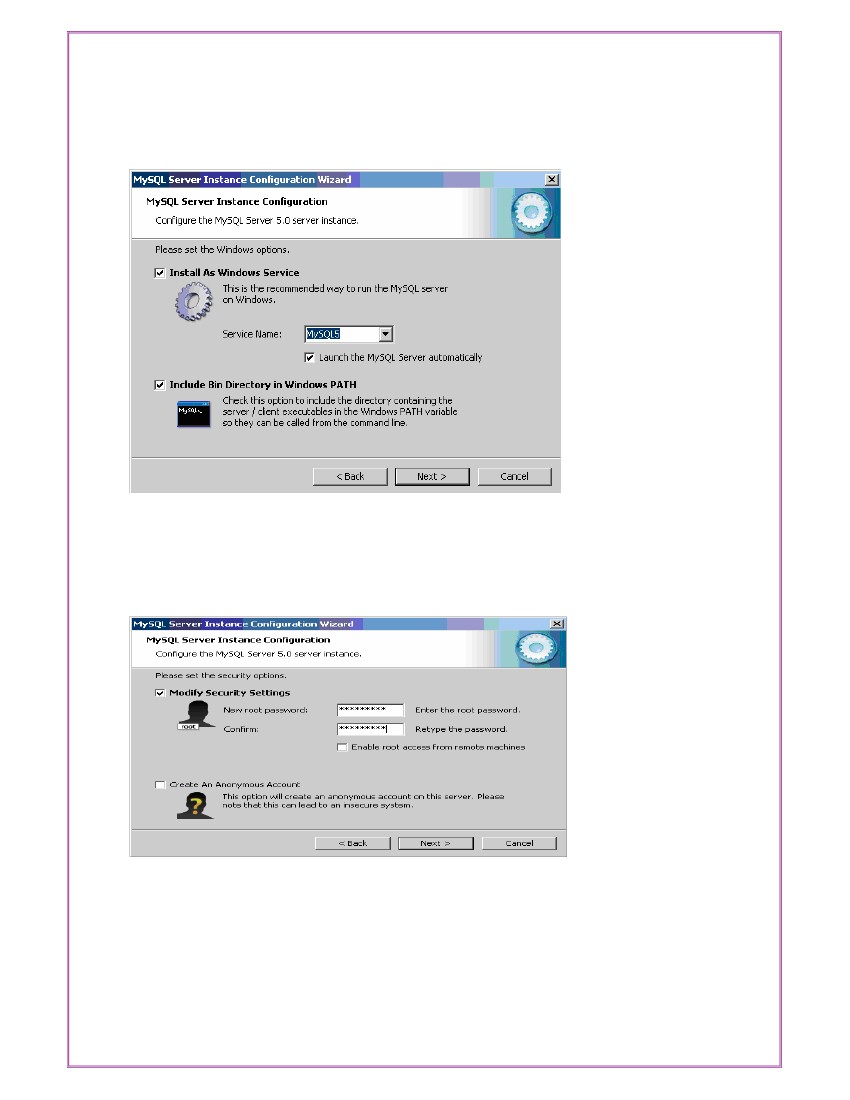
Step1010

Select a standard configuration and this will use a general purpose configuration for the

serverthatcanbetunedmanually.Tocontinue,clicknext.

Step11

11



DBMS RECORD

Page No: 21

Check on the install as windows service and include bin directory in windows path. To

continue, click next.

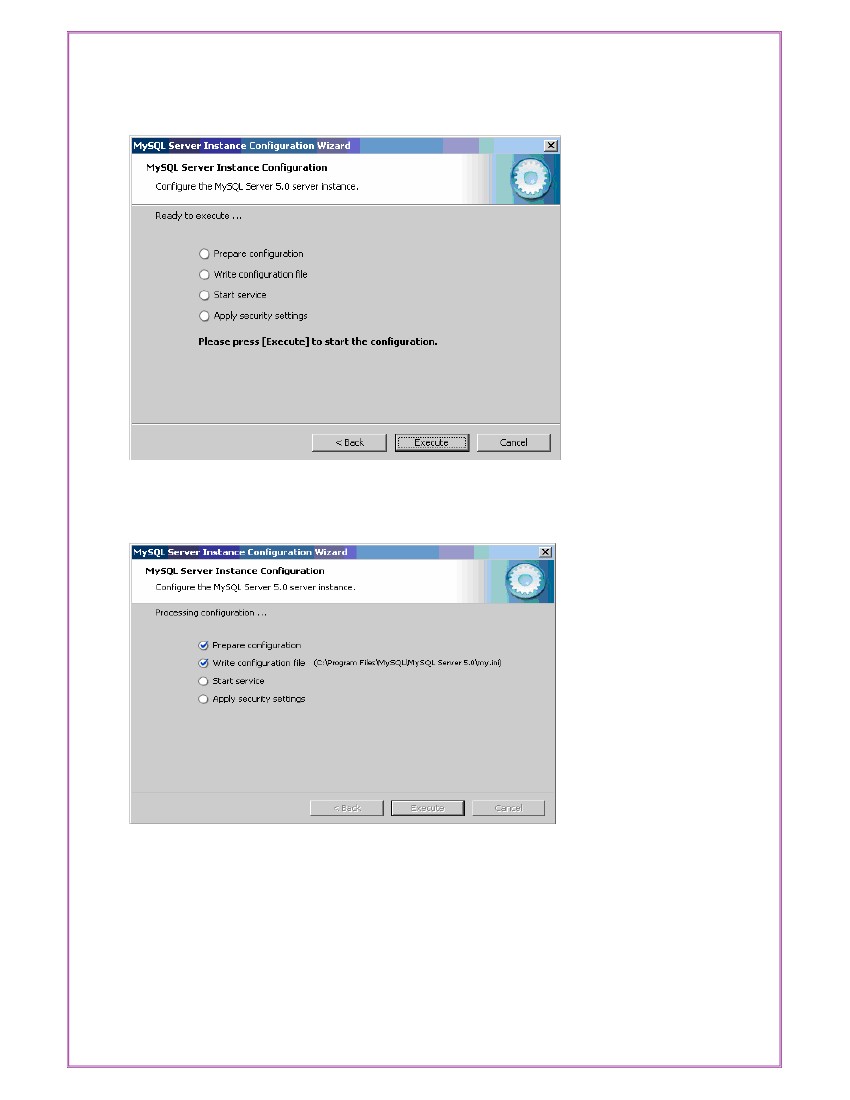
Step1212

Please set the security options by entering the root password and confirm retype the

password. To continue, click next.

Step13

13



DBMS RECORD

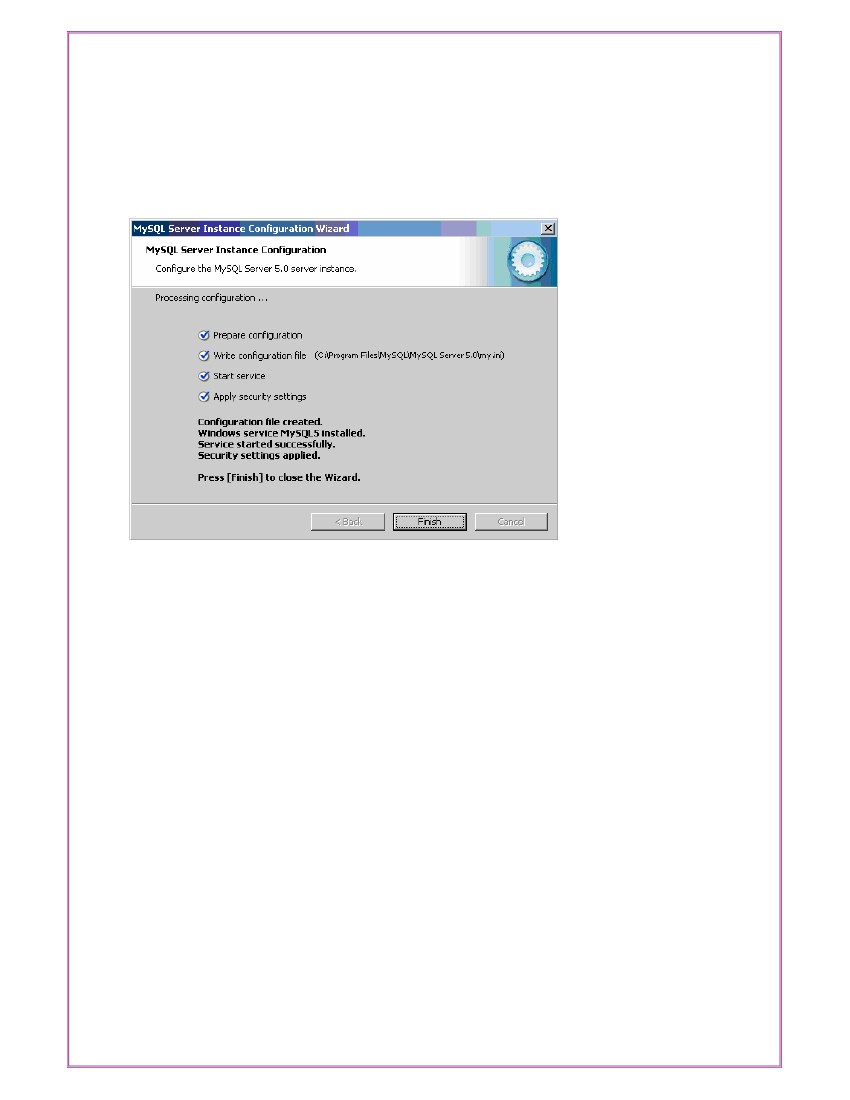
Page No: 22

Ready to execute? Clicks execute to continue.

Step14

Processing configuration in progress.

14



DBMS RECORD

Page No: 23

Step15

15

Configuration file created. Windows service MySQL5 installed. Press finish to close the

wizard.



DBMS RECORD

Page No: 24

2. Practicing DDL & DML Commands

Data Definition Language

The data definition language is used to create an object, alter the structure of an object and

also drop already created object. The Data Definition Languages used for table definition

can be classified into following:

 Create table command

 Alter table command

 Truncate table command

 Drop table command

Creating of Tables on ROAD WAY TRAVELS:

Table is a primary object of database, used to store data in form of rows and columns. It is

created using following command:

Create Table <table\_name> (column1 datatype(size), column2 datatype(size),

・\_,column(n) datatype(size));

Where, table\_name is a name of the table and coulumn1, column2 \_ column n is a name of

the column available in table\_name table.

R Each column is separated by comma.

Pointes to be remember while creating a table.

 Table Name must be start with an alphabet.

 Table name and column name should be of maximum 30 character long.

 Column name should not be repeated in same table.

 Reserve words of Oracle cannot be used as a table and column name.

 Two different tables should not have the same name.

 Underscores, numerals and letters are allowed but not blank space or single quotes.

Example:

SQL> create table Bus(Bus\_No varchar(5),

varchar(20),CouchType varchar2(10),fair number);

Table Created.

Above definition will create simple table. Still there are more additional option related with

source

varchar(20),

destination



DBMS RECORD

Page No: 25

create table for the object-relation feature we will discuss it afterwards.

Desc command

Describe command is external command of Oracle. The describe command is used to view

the structure of a table as follows.

Desc <table name>

SQL> desc bus;

NameNull?

----------------------------------------- --------

BUS\_NONOT NULL

SOURCE

DESTINATION

CouchType

FAIR

Reservation Table:

SQL> create table Reservation(PNR\_NO Numeric(9), No\_of\_seats Number(8), Address

varchar(50), Contact\_No Numeric(9), Status char(3));

Table created.

SQL> desc Reservation

NameNull?Type

----------------------------------------- -------- ----------------------------

PNR\_NONUMBER(9)

NO\_OF\_SEATSNUMBER(8)

ADDRESSVARCHAR2(50)

CONTACT\_NONUMBER(9)

STATUSCHAR(3)

Type

----------------------------

INTEGER2(5)

VARCHAR2(20)

VARCHAR2(20)

VARCHAR2(10)

NUMBER

Cancellation Table:

SQL> create table Cancellation(PNR\_NO Numeric(9), No\_of\_seats Number(8), Address

varchar(50), Contact\_No Numeric(9), Status char(3));

Table created.

SQL> desc Cancellation

NameNull? Type

----------------------------------------- -------- ----------------------------

PNR\_NONUMBER(9)

NO\_OF\_SEATSNUMBER(8)

ADDRESSVARCHAR2(50)

CONTACT\_NONUMBER(9)

STATUSCHAR(3)



DBMS RECORD

Page No: 26

Ticket Table:

SQL> create table Ticket(Ticket\_No Numeric(9) primary key, age number(4), sex char(4)

Not null, source varchar(2), destination varchar(20), dep\_time varchar(4));

Table created.

SQL> desc Ticket

NameNull?Type

----------------------------------------- ------------------------------------

TICKET\_NONOT NULL NUMBER(9)

AGENUMBER(4)

SEXNOT NULL CHAR(4)

SOURCEVARCHAR2(2)

DESTINATIONVARCHAR2(20)

DEP\_TIMEVARCHAR2(4)

Alteration of Table

Once Simple Table is created, if there is a need to change the structure of a table at that time

alter command is used. It is used when a user want to add a new column or change the width

of datatype or datatype itself or to add or drop integrity constraints or column. (we will see

about constraints very soon.).i.e. table can be altered in one of three way : by adding

column, by changing column definition or by dropping column.

Addition of Column(s)

Addition of column in table is done using:

Alter table <table\_name> add(column1 datatype, column2 datatype ・\_);

Add option is used with alter table\_ when you want to add a new column in existing

table. If you want to Add more than one column then just write column name, data type and

size in brackets. As usual Comma sign separates each column. For Example, suppose you

want to add column comm in emp\_master, then you have to perform the following

command.

SQL> ALTER TABLE Passenger ADD FOREIGN KEY (PNR\_NO) REFERENCES

Reservation(PNR\_NO);

Table altered.

SQL> ALTER TABLE Cancellation ADD FOREIGN KEY (PNR\_NO) REFERENCES

Reservation(PNR\_NO);

Table altered.



DBMS RECORD

Page No: 27

SQL> alter table Ticket modify tiketnonumber(10);

Table altered.

Deletion of Column

Till Oracle8 it is not possible to remove columns from a table but in Oracle8i, drop option is

used withAlter table\_ when you want to drop any existing column.

Alter table <table\_name> drop column <column name>;

Using above command you cannot drop more than one column at a time.

For Example, suppose you want to delete just before created column comm from the

emp\_master, then you have to apply following command.

SQL>Alter Table Emp\_master drop column comm;

Table altered.

Dropping column is more complicated than adding or modifying a column, because of the

additional work that Oracle has to do. Just removing the column from the list of columns in

the table actually recover the space that was actually taken up by the column values that is

more complex, and potentially very time- consuming for the database. For this reason, you

can drop a column using unused clause. Column can be immediately remove column by

drop clause, the action may impact on performance or one make marked column as unused

using unused caluse, there will be no impact on performance. When unused caluse is used

the column can actually be dropped at a later time when the database is less heavily used.

One can marked column as a unused using :

Alter table <table\_name> set unused column <column name>;

For Example,

SQL>Alter Table Emp\_master set unused column comm;

Table altered.

Making a column as Unused\_ does not release the spcace previously used by the colum,

until you drop the unused columns. It can be possible using:

Alter table <table\_name> drop unused columns;

Once you have marked column as unused\_ you cannot access that column

You can drop multiple columns at a time using single command as per follows



DBMS RECORD

Page No: 28

Alter table <table\_name> drop (Column1, Column2,\_);

The multiple columns name must be enclosed in parentheses.

Modification in Column

Modify option is used with Alter table\_ when you want to modify any existing column.

If you want to

modify data type or size of more than one column then just write column name, data type

and size in brackets and each column is separated by comma sign as per follows:

Alter table <table name> modify (column1 datatype,・\_);

For Example, if you want to change size of salary column of emp\_master the following

command is performed.

SQL> Alter table emp\_master modify salary number(9,2);

Table altered.

It will change size of salary column from 7 to (9,2).

When you want to decrease the size of column, table must be empty. If table has any rows

then it will not allow decrement in the column width.

Truncate Table

If there is no further use of records stored in a table and the structure is required then only

data can be deleted using Truncate command. Truncate command will delete all the records

permanently of specified table as follows.

Truncate table <table name> [Reuse Storage];

Example

Following command will delete all the records permanently from the table.

SQL>Truncate Table Emp\_master;

Or

SQL>Truncate Table Emp\_master Reuse Storage;

Table truncated.



DBMS RECORD

Page No: 29

Exp No: 4

AIM: Applying Constraints on Road Way Travels Tables.

Constraints

Date: \_ \_/\_ \_/ \_ \_

Maintaining security and integrity of a database is the most important factor. Such

limitations have to be enforced on the data, and only that data which satisfies the conditions

will actually be stored for analysis.

If the data gathered fails to satisfy the conditions set, it is rejected. This technique ensures

that the data that is stored in the database will be valid, and has integrity. Rules, which are

enforced on data being entered and prevents user from entering invalid data into tables are

called constraints.

permanent storage.

Oracle permits data constraints to be attached to table columns via SQL syntax that will

check data for integrity. Once data constraints are part of a table column construction, the

Oracle engine checks the data being entered into a table column against the data constraints.

If the data passes this check, it is stored in the table, else the data is rejected. Even if a single

column of the record being entered into the table fails a constraint, the entire record is

rejected and not stored in the table. Both the Create table\_ and Alter table\_ SQL verbs can

be used to write SQL sentences that attach constraints to a table column.

Until now we have created table without using any constraint, Hence the tables have not

been given any instructions to filter what is being stored in the table.

The following are the types of integrity constraints

 Domain Integrity constraints

 Entity Integrity constraints

 Referential Integrity constraint

Oracle allows programmers to define constraints

 Column Level

 Table Level

Thus, constraints are super control data being entered in tables for



DBMS RECORD

Page No: 30

Column Level constraints

If data constraints are defined along with the column definition when creating or altering

a table structure, they are column level constraints. Column level constraints are applied to

the current column. The current column is the column that immediately precedes the

constraints i.e. they are local to a specific column.Column level constraints cannot be

applied if the data constraints span across the multiple columns in a table.

Table Level Constraint

If the data constraints are defined after defining all the table columns when creating or

altering a table structure, it is a table level constraint. Table Level constraints mostly used

when data constraints spans across multiple columns in a table.

Domain Integrity Constraints

These constraints set a range and any violations that take place will prevent the user from

performing the manipulations that caused the breached.

Domain integrity constraints are classified into two types

• Not Null constraint

• Check constraint

Not Null Constraint

Often there may be records in a table that do not have values for every field, either because

the information is not available at the time of data entry or because field is not applicable in

every case. Oracle will place a null value in the column in the absence of a user-defined

value. By default every column will accept null values.A Null values is different from a

blank or a zero. We can say that Null means undefined. Null are treated specially by Oracle.

When a column is defined as not null, then that column becomes mandatory column. It

implies that a value must be entered into the column. Remember that not null constraints can

be applied on column level only.

Example

SQL> create table Ticket(Ticket\_No Numeric(9) , age number(4), sex char(4) Not null,

source varchar(2), destination varchar(20), dep\_time varchar(4));



DBMS RECORD

Page No: 31

Table created.

Check Constraint

Business rule validations can be applied on a column using Check constraint. Check

constraint must be specified by logical or boolean expressions.

Create a client\_master table with following check constraints.

 Data values being inserted into the column client\_no must starts with the capital

letter ‘C’

 Data values being inserted into the column name should be in upper case only.

 Only allow Mumbai\_ or Ahmedabad\_ values for the city column.

Check constraint defined at column level as per follows:

SQL> create table Reservation(PNR\_NO Numeric(9), No\_of\_seats Number(8), Address

varchar(50), Contact\_No Numeric(10) constraint ck check(length(contact\_no)=10), Status

char(3));

Table created.

Check constraint with alter command

SQL> alter table Ticket add constraint check\_age check(age>18);

Table altered.

Entity Integrity Constraints

This type of constraints are further classified into

 Unique Constraint

 Primary Key Constraint

Unique Constraint

The purpose of unique key is to ensure that information in the column(s) is unique i.e. the

value entered in column(s) defined in the unique constraint must not be repeated across the

column. A table may have many unique keys. If unique constraint is defined in more than

one column (combination of columns), it is said to be composite unique key. Maximum

combination of columns that a composite unique key can contain is 16.

Example:

SQL> create table Ticket(Ticket\_No Numeric(9) unique, age number(4), sex char(4) l,

source varchar(2), destination varchar(20), dep\_time varchar(4));



DBMS RECORD

Page No: 32

Table created.

Unique constraint with alter command

Example:

SQL> Alter table ticket add constraint uni1 Unique (ticket\_no);

Primary Key Constraint

A primary key is one or on more columns(s) in a table to uniquely identify each row in the

table. A primary key column in a table has a special attribute. It defines the column, as a

mandatory column i.e. the column cannot be left blank and should have a unique value. Here

by default not null constraint is attached with the column.A multicolumn primary key is

called a Composite primary key. The only function of a primary key in a table is to uniquely

identify a row. A table can have only one primary key.

Primary key constraint at the column level

Example:

SQL> create table Ticket(Ticket\_No Numeric(9) constraint pk primary key, age number(4),

sex char(4) l, source varchar(2), destination varchar(20), dep\_time varchar(4));

Table created.

SQL> insert into ticket values (1001,26, ‘M’, ‘KPHB’,’MTM’,’20:00’);

1 row created.

SQL> insert into ticket values (1001,26, ‘M’, ‘KPHB’,’MTM’,’20:00’);

18

ERROR at line 1:

ORA-00001: unique constraint (SCOTT.PR\_ONO) violated

If we insert already inserted ticket\_no the above mentioned error will generate.

SQL> insert into ticket values (null,26, ‘M’, ‘KPHB’,’MTM’,’20:00’);

ERROR at line 1:

ORA-01400 :cannot insert NULL into (SCOTT\_.ORDER\_MASTER\_ .ORDER\_NO”)

If we try to insert null values it will generate error message.

Composite Primary key defined at the table level: Composite Primary key is a primary key

created with the combination of more than one key and combination values of both the fields

should be unique



DBMS RECORD

Page No: 33

Example:

SQL> create table vendor\_master (ven\_code varchar(5), ven\_name varchar(20), venadd1

varchar(15), venadd2 varchar(15),vencity varchar(15), constraint pr\_com primary key

(ven\_code,ven\_name));

Table created.

Primary key with alter command:

SQL> alter table bus add constraint pr primary key (busno);

Table altered.

Referential Integrity Constraint

In this category there is only one constraint and it is Foreign Key & References

To establish a Parent-child\_ or a Master-detail\_ relationship between two tables having a

common column, we make use of referential integrity constraint.Foreign key represent

relationships between tables. A foreign key is a column whose values are derived from the

primary key or unique key. The table in which the foreign key is defined is called a foreign

table or Detail table. The table that defines the primary or unique keys and is referenced by

the foreign key is called the Primary table or Master table.The master table can be

referenced in the foreign key definition by using references keyword. If the column name is

not specified, by default, Oracle references the primary key in the master table.

The existence of a foreign key implies that the table with the foreign key is related to the

master table from which the foreign key is derived. A foreign key must have a

corresponding primary key or a unique key value in a master table.

Principles of Foreign Key Constraint

 Rejects an insert or update of a value in a particular column, if a corresponding

value does not exist in the master table.

 Deletion of rows from the Master table is not possible if detail table having

corresponding values.

 Primary key or unique key must in Master table.

 Requires that the Foreign key column(s) and reference column(s) have same data

type



DBMS RECORD

Page No: 34

References constraint defined at column level

Example:

SQL> create table Passenger(PNR\_NO Numeric(9) references reservation , Ticket\_NO

Numeric(9) references ticket, Name varchar(20), Age Number(4), Sex char(10), PPNO

varchar(15));

Table created.

Foreign Key Constraint with alter command

SQL> alter table reservation add constraint fk\_icode foreign key (busno) references

bus(bus\_no);

Table altered.

Remember that when we add constraint at table level foreign key keyword is must.

SQL> delete from bus where bus\_no = 2011;

ERROR at line 1:

ORA-02292: integrity constraint (SCOTT.FK\_ONO) violated - child record found



DBMS RECORD

Page No: 35

Exp No: 5

AIM : Applying DML commands on Road Way Travels Tables.

Data Manipulation Command

Date: \_ \_/\_ \_/ \_ \_

Data Manipulation commands are most widely used SQL commands and they are

 Insert

 Update

 Delete

 Select

a) Insert command

After creation of table, it is necessary it should have data in it. The insert command is used

to add data in form of one or more rows to a table as per follows:

Insert into <table name> values(a list of data values);

In a list of data values you have to specify values for each and every column in the same

order as they are defined. A value of each column is separated by comma in the list.

The value of char, nchar, varchar2, nvarchar2, raw, long and date data types are enclosed in

single quotes.

Using insert command one can insert values in specific columns as follows:

Insert into <table name>(column list) values(a list of data);

Here number of column and a list of data should be same and list of data should be in order

to column list.

SQL> insert into emp\_master (empno,ename,salary) values (1122,‘Smith’,8000);

1 row created.

Above command insert one row but values are inserted in only three columns. Remaining

four columns have null values.If you have define not null constraint in any of remaining

columns it want allow you to insert data in a table.

Adding values in a table using Variable method

Till now we have seen static method to insert data. One can add data in a table using

DBMS RECORD

Page No: 36

variable method with & (ampersand) sign. It will prompt user to enter data of mention field.

Generally It is used to add more than one row in a table without typing whole command

repetitively using / sign.

SQL> insert into Passenger values(&PNR\_NO,&TICKET\_NO, '&Name', &Age, '&Sex',

'&PPNO');

Enter value for pnr\_no: 1

Enter value for ticket\_no: 1

Enter value for name: SACHIN

Enter value for age: 12

Enter value for sex: m

Enter value for ppno: sd1234

old

1: insert into Passenger values(&PNR\_NO,&TICKET\_NO, '&Name', &Age, '&Sex',

'&PPNO')

new 1: insert into Passenger values(1,1,'SACHIN',12,'m','sd1234')

1 row created.

SQL> /

Enter value for pnr\_no: 2

Enter value for ticket\_no: 2

Enter value for name: rahul

Enter value for age: 34

Enter value for sex: m

Enter value for ppno: sd3456

old 1: insert into Passenger values(&PNR\_NO,&TICKET\_NO, '&Name', &Age, '&Sex',

'&PPNO')

new 1: insert into Passenger values(2,2,'rahul',34,'m','sd3456')

1 row created.

SQL> /

Enter value for pnr\_no: 3

Enter value for ticket\_no: 3

Enter value for name: swetha

Enter value for age: 24

Enter value for sex: f

Enter value for ppno: sdqw34

old 1: insert into Passenger values(&PNR\_NO,&TICKET\_NO, '&Name', &Age, '&Sex',

'&PPNO')

new 1: insert into Passenger values(3,3,'swetha',24,'f','sdqw34')

1 row created.

SQL> /

Enter value for pnr\_no: 4

Enter value for ticket\_no: 4

Enter value for name: ravi



DBMS RECORD

Page No: 37

Enter value for age: 56

Enter value for sex: m

Enter value for ppno: sdqazx

old 1: insert into Passenger values(&PNR\_NO,&TICKET\_NO, '&Name', &Age, '&Sex',

'&PPNO')

new 1: insert into Passenger values(4,4,'ravi',56,'m','sdqazx')

1 row created.

SQL> /

Enter value for pnr\_no: 4

Enter value for ticket\_no: 5

Enter value for name: asif

Enter value for age: 33

Enter value for sex: m

Enter value for ppno: iuyhjk

old 1: insert into Passenger values(&PNR\_NO,&TICKET\_NO, '&Name', &Age, '&Sex',

'&PPNO')

new 1: insert into Passenger values(4,5,'asif',33,'m','iuyhjk')

insert into Passenger values(4,5,'asif',33,'m','iuyhjk')

\*

ERROR at line 1: ORA-00001: unique constraint (SYSTEM.SYS\_C004023) violated

SQL> insert into Bus values('&Bus\_No','&source','&destination');

Enter value for bus\_no: 1

Enter value for source: hyd

Enter value for destination: ban

old 1: insert into Bus values('&Bus\_No','&source','&destination')

new 1: insert into Bus values('1','hyd','ban')

1 row created.

SQL> /

Enter value for bus\_no: 2

Enter value for source: hyd

Enter value for destination: chn

old 1: insert into Bus values('&Bus\_No','&source','&destination')

new 1: insert into Bus values('2','hyd','chn')

1 row created.

SQL> /

Enter value for bus\_no: 4

Enter value for source: hyd

Enter value for destination: mum

old 1: insert into Bus values('&Bus\_No','&source','&destination')

new 1: insert into Bus values('4','hyd','mum')

1 row created.



DBMS RECORD

Page No: 38

SQL> /

Enter value for bus\_no: 5

Enter value for source: hyd

Enter value for destination: kol

old 1: insert into Bus values('&Bus\_No','&source','&destination')

new 1: insert into Bus values('5','hyd','kol')

1 row created.

SQL> /

Enter value for bus\_no: 5

Enter value for source: sec

Enter value for destination: ban

old 1: insert into Bus values('&Bus\_No','&source','&destination')

new 1: insert into Bus values('5','sec','ban')

insert into Bus values('5','sec','ban')

\*

ERROR at line 1:

ORA-00001: unique constraint (SYSTEM.SYS\_C004025) violated

SQL> insert into Reservation values(&PNR\_NO, &No\_of\_seats, '&Address', &Contact\_No ,

'&Status');

Enter value for pnr\_no: 1

Enter value for no\_of\_seats: 2

Enter value for address: masabtank

Enter value for contact\_no: 9009897812

Enter value for status: s

old1: insert into Reservation values(&PNR\_NO, &No\_of\_seats, '&Address',

&Contact\_No,' &Status')

new 1: insert into Reservation values(1,2,'masabtank',9009897812,'s')

1 row created.

SQL>insertinto

values(&PNR\_NO,&No\_of\_seats,'&Address',&Contact\_No,'&Status');

Enter value for pnr\_no: 8

Enter value for no\_of\_seats: 3

Enter value for address: cbt

Enter value for contact\_no: 9090887753

Enter value for status: s

Reservation

old 1: insert into Reservation values(&PNR\_NO, &No\_of\_seats, '&Address', &Contact\_No,

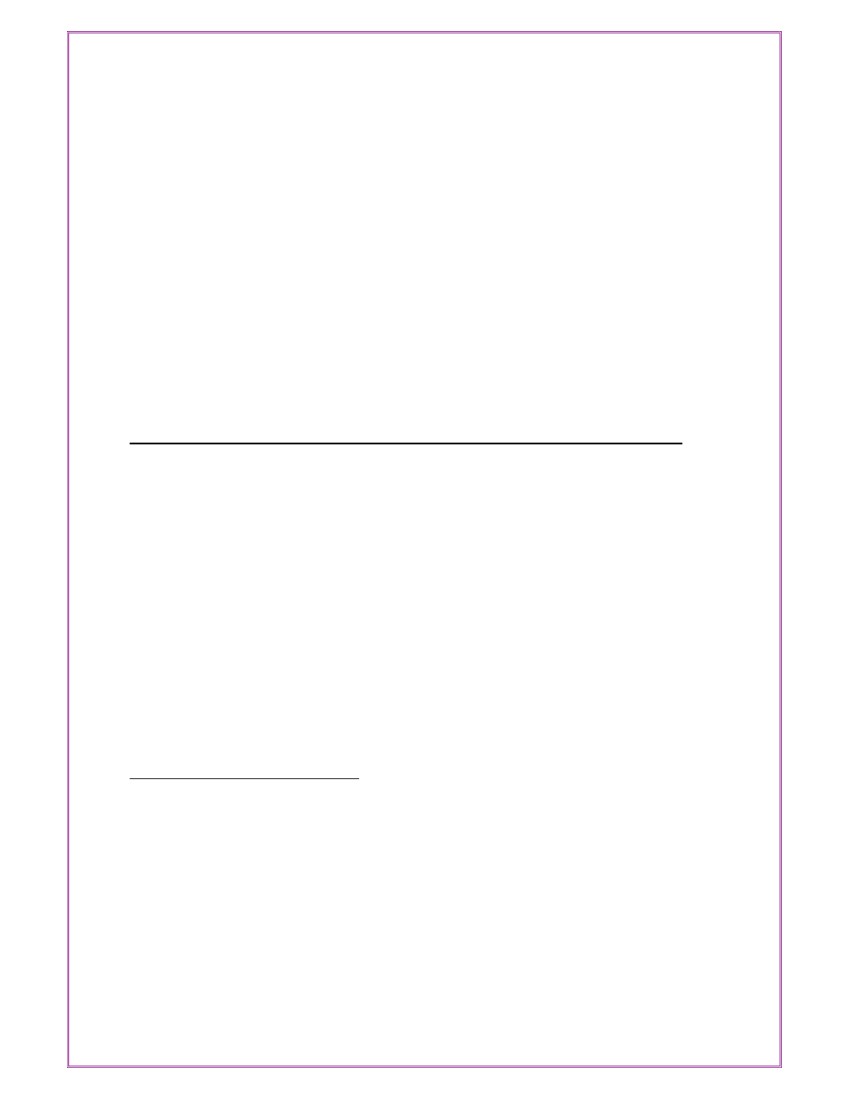
'&Status')

new 1: insert into Reservation values(8,3,'cbt',9090887753,'s')

insert into Reservation values(8,3,'cbt',9090887753,'s')

\*

ERROR at line 1:



DBMS RECORD

Page No: 39

ORA-02291: integrity constraint (SYSTEM.SYS\_C004024) violated - parent key not found

b) Simple Select Command

Stored information can be retrieved from the table through select command. Select is the

most frequently used command, as access to information is needed all the time. Syntax of

simple select command is as per follows:

Select <column1>,<column2>,\_,<column(n)> from <table name>;

The following command will select all the rows and columns from emp\_master.

SQL> select \* from emp\_master;

EMPNO

1122

1122

1123

1124

1125

ENAME

Allen

Smith

King

Martin

Tanmay

JOB

HIREDATE

SALARY

10000

8000

3400

7000

DEPTNO

10

COMM

1000

Manager 1-JAN-00

1-JAN-00

Clerk

30-JUN-00

20

20

300

1000

Manager 30-AUG-00

16-SEP-00 10

5 rows selected.

The ‘\*’ will indicate all the columns. But If you want to retrieve only specific columns from

a table then you have to specify column names with select commands.

SQL> select empno,ename,salary from emp\_master;

This query will give information from only three columns.

EMPNO

1122

1122

1123

1124

ENAME

Allen

Smith

King

Martin

SALARY

10000

8000

3400

7000

5 rows selected.

SQL> select \* from Passenger;

PNR\_NO

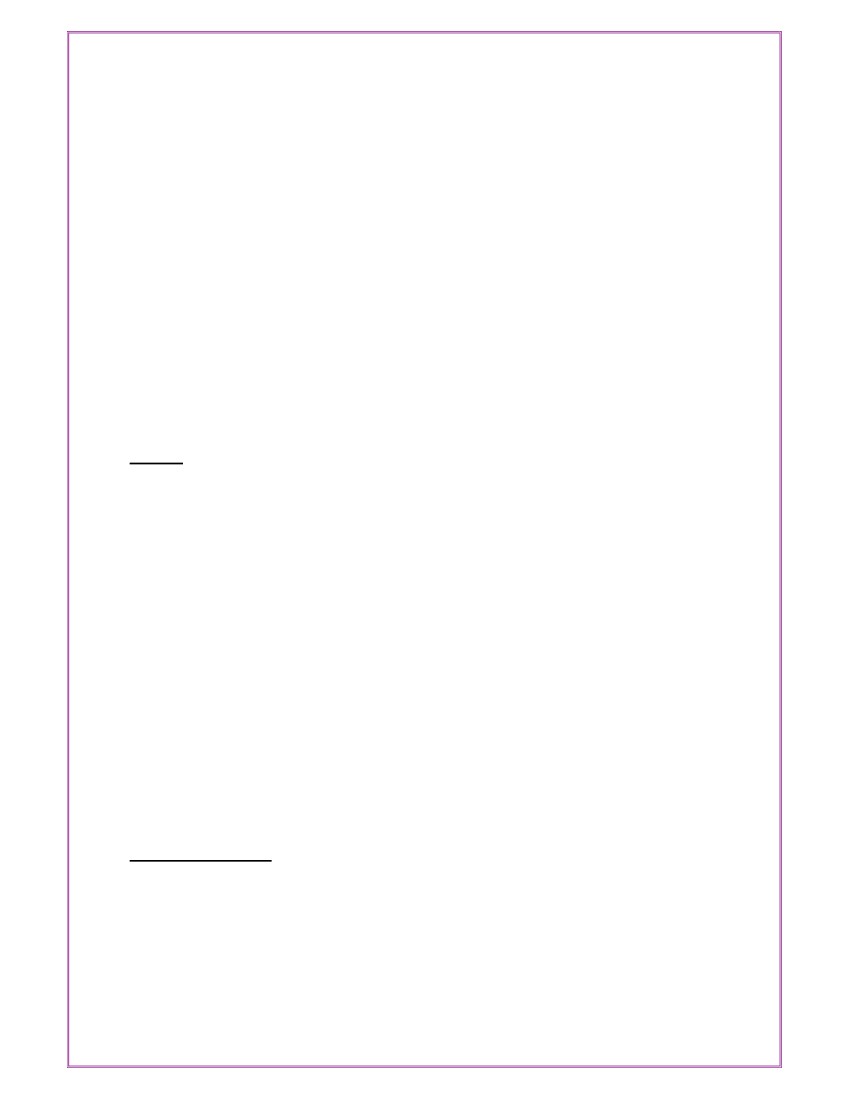
TICKET\_NO

NAME

AGE

SEX

PPNO



DBMS RECORD

Page No: 40

----------

1

2

3

4

----------

1

2

3

4

--------------------

SACHIN

rahul

swetha

ravi

---------- ---------- ---------------

12msd1234

34msd3456

24fsdqw34

56msdqazx

Select Command

Previously we have seen simple use of select statement to retrieve the data from the table.

Now we have look further use of Select statement.

Distinct Clause

To prevent the selection of distinct rows, we can include distinct clause with select

command.The following command will exclude duplicate empno.

SQL> select distinct deptno from emp\_master;

DEPTNO

10

20

2 rows selected.

Select command with where clause:

To list out specific rows from a table we can include where clause. We have to specify

conditions with where clause to filter the records. The where clause is similar which we have

used with delete and update command. It can be done using :

Select <column(s)> from <table name> where [condition(s)];

Example

Suppose you want to view only those rows where HireDate is 1-JAN-00.

SQL> select empno,ename from emp\_master where hiredate = ‘1-jan- 00’;

EMPNO

1122

ENAME

Allen

1 row selected.

UPDATE Table

SQL> update Passenger set age='43' where PNR\_NO='2';



DBMS RECORD

Page No: 41

1 row updated.

SQL> select \* from Passenger;

PNR\_NOTICKET\_NO

--------------------

11

22

33

44

NAME

--------------------

SACHIN

rahul

swetha

ravi

AGE

----------

12

43

24

56

SEXPPNO

---------- ---------------

msd1234

msd3456

fsdqw34

msdqazx

DELETE

SQL> delete from Passenger where PNR\_NO='4';

1 row deleted.

SQL> select \* from Passenger;

PNR\_NOTICKET\_NONAME

---------- ------------------------------

11SACHIN

22rahul

33swetha

AGE SEXPPNO

---------- ---------- ---------------

12msd1234

43msd3456

24fsdqw34

DROP Table

SQL> drop table Cancellation;

Table dropped.

Select command with DDL and DML command.

Select command is used to provide information of the table. But apart from retrieving data it

is used with some DDL and DML commands.

Table Creation with select statement

One can create table using select statement as per follows :

create table <table name> as select <columnname(s)> from <existing table name>;

Example

Using following command we can copy emp\_master table into emp\_master\_copy.

SQL>create table emp\_master\_copy as select \* from emp\_master;

It will create emp\_master\_copy table with the same Structure and data of emp\_master

table. Suppose if you want to create new table with some specific columns only, then you

have to specify column name in select statement as per follows:



DBMS RECORD

Page No: 42

SQL>create table emp\_master\_copy1 (eno,nm) as select empno,ename from emp\_master;

This command will create a new table emp\_master\_copy1 with only two columns eno and

nm similar to empno and ename available in emp\_master If you want to make a copy of

table without any data i.e. only structure of table, one have to specifywrong condition (like

1=2, 2=3,11=13).SQL>create table emp\_copy as select \* from emp\_master where 1=2;

The condition 1=2 will never satisfy so select statement will retrieve none row and only

structure will copy.

Insert data using Select statement

Inserting records from one table to another table can also possible through select statement.

Syntax:

Inert into <tablename> (select <columns> from <tablename>);

Example

SQL> insert into emp\_copy (select \* from emp\_master);

This will insert all the rows of emp\_master.

R When you insert data into one table from another table data structure should be same of

both the table.

If you want to make copy of selected columns data from one table to another table the data

structure of both the columns should be same.

Example

SQL> insert into emp\_copy(nm) (select name from emp\_master);

Change Table Name

One can change the existing table name with a new name.

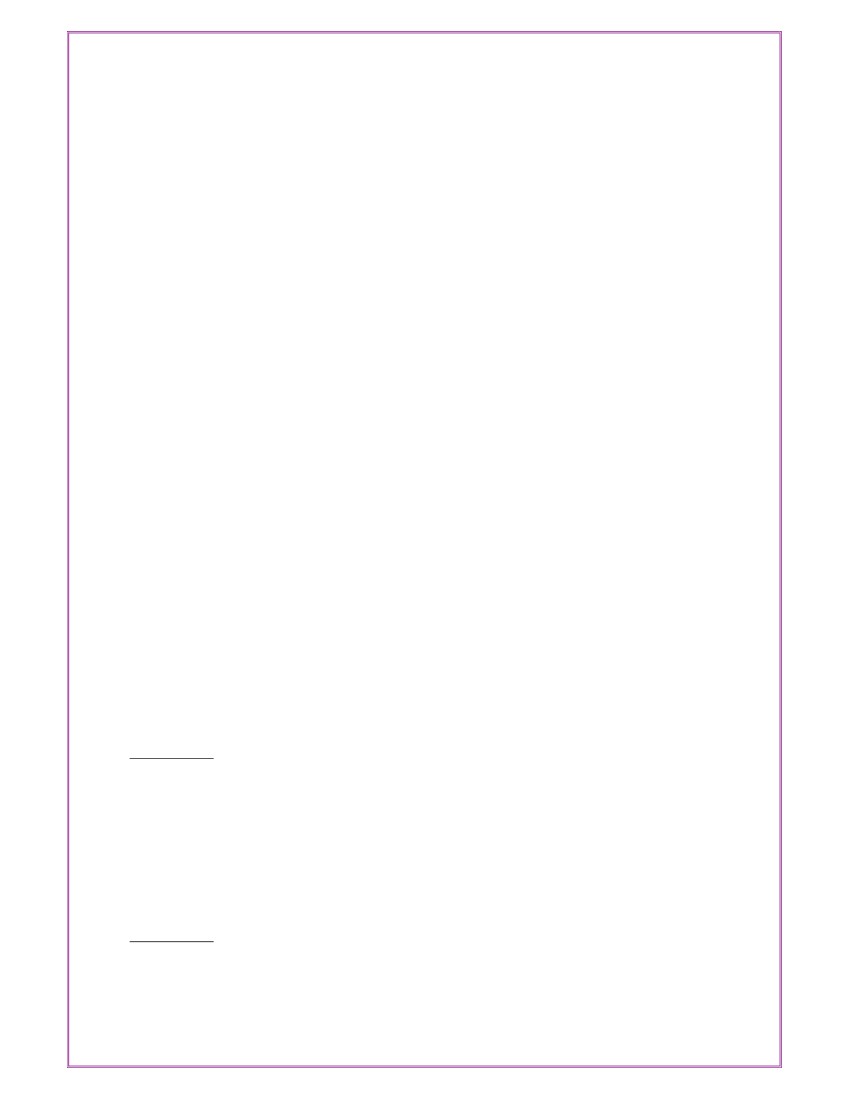
Syntax

Rename <OldName> To <NewName>;

Example:

SQL> Rename emp\_master\_copy1 To emp\_master1;

Table Renamed.



DBMS RECORD

Page No: 43

Exp No: 6

Date: \_ \_/\_ \_/ \_ \_

Aim: Practice Queries using ANY, ALL, IN, EXISTS, UNION, INTERSECT

Set Operators

Set operators combine the results of two queries into a single one. The following set

operators are availablein SQL.

 Union

 Union All

 Intersect

 Minus

While we are using set operators the following points must be keep in mind The queries,

which are related by a set operator should have the same number of column and the

corresponding columns, must be of the data types.Such a query should not contain any

columns of long data type.The label under which the rows are displayed are those from the

first select statement.

Union: The union operator returns all distinct rows selected by two or more queries.

The following example

combines the result of two queries with the union operator, which eliminates duplicate rows.

SQL> select order\_no from order\_master;

ORDER\_NO

O001

O002

O003

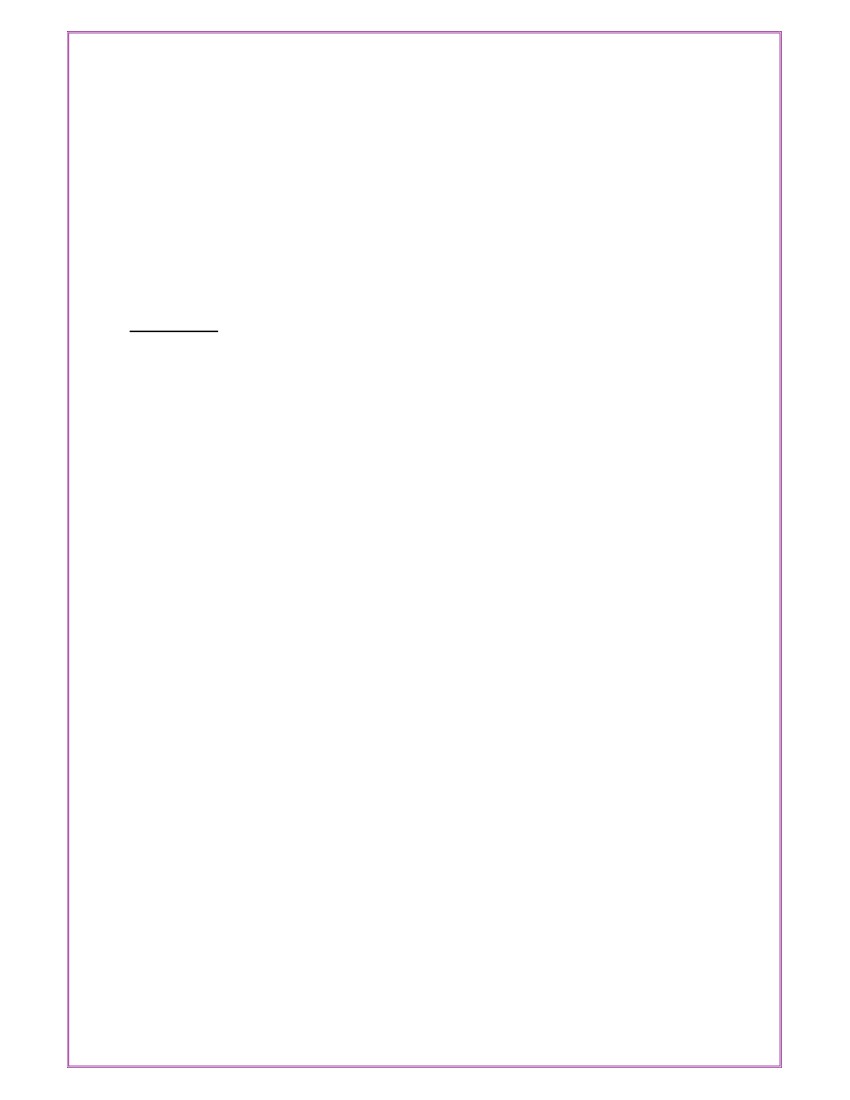
O004

SQL> select order\_no from order\_detail;

ORDER\_NO

O003

O004



DBMS RECORD

Page No: 44

O005

O006

O007

Now we check output using union operator.

Example:

SQL>select order\_no from order\_master union select order\_no from

order\_detail;

ORDER\_NO

O001

O002

O003

O004

O005

O006

O007

Union All : The union all operators returns all rows selected by either query including

duplicates. The following example combines the result with the aid of union all operator,

which does not eliminates duplicate rows.

Example:

SQL> select order\_no from order\_master union all select order\_no from

order\_detail

ORDER\_NO

O001

O002

O003

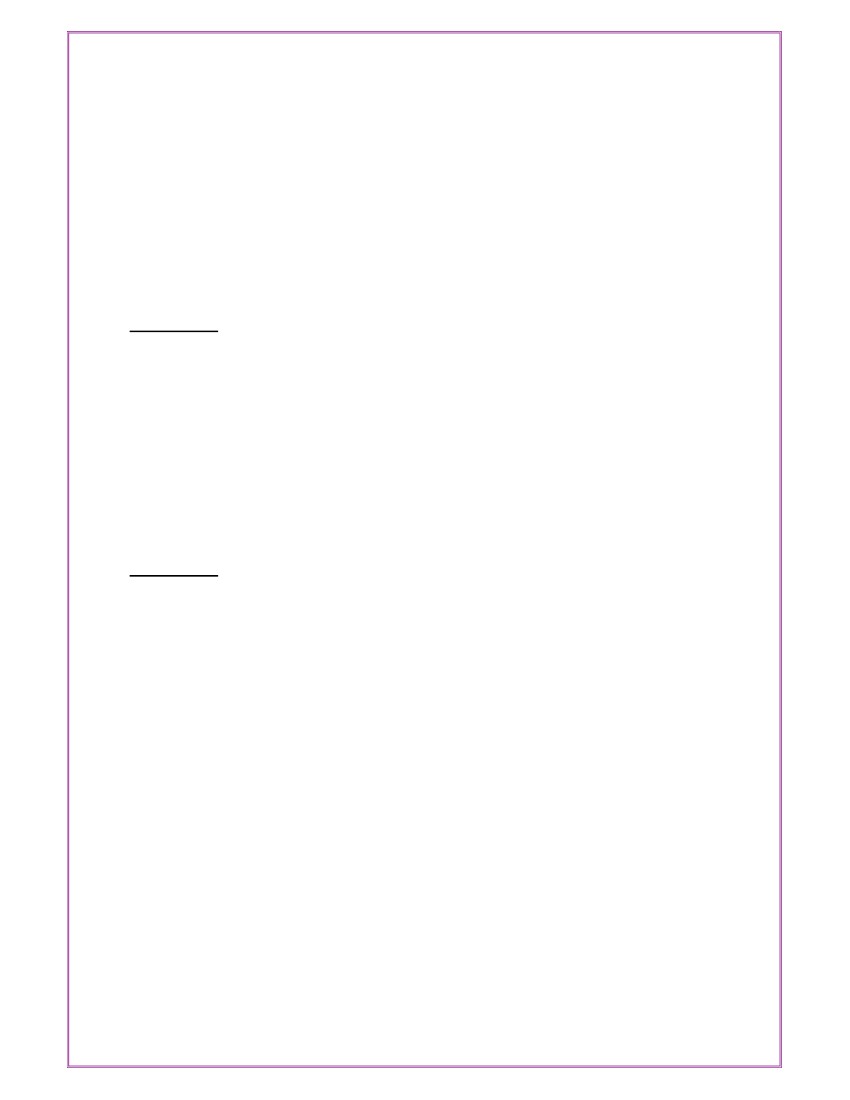
O004

O003

O004

O005

O006



DBMS RECORD

Page No: 45

O007

Intersect : The intersect operator outputs only rows produced by both the queries

intersected i.e. the output in an intersect clause will include only those rows that are

retrieved by both the queries.

Example:

SQL> select order\_no from order\_master intersect select order\_no from

order\_detail;

ORDER\_NO

O003

O004

Minus : The Minus operator outputs the rows produced by the first query, after filtering the

rows retrieved by the second query.

Example:

SQL> select order\_no from order\_master minus select order\_no from order\_detail;

ORDER\_NO

O001

O002

EXAMPLE QUERIES:

1. Display Unique PNR\_NO of all Passengers

SQL> select PNR\_NO from Passenger;

PNR\_NO

----------

1

2

3

4

5

6

7

7 rows selected.

2. Display all the names of male Passengers



DBMS RECORD

Page No: 46

SQL> select Name from Passenger where Sex='m';

NAME

--------------------

SACHIN

rahul

rafi

salim

riyaz

3. Display Ticket numbers and names of all Passengers

SQL> select Ticket\_NO,Name from Passenger;

TICKET\_NONAME

------------------------------

1SACHIN

2rahul

3swetha

23rafi

12salim

34riyaz

21neha

7 rows selected.

4. Display the source and destination having journey time more than 10 hours.

SQL> select source, destination from Ticket where Journey\_Dur>10;

SOURCEDESTINATION

------------------------------

HYDBAN

SECBAN

HYDMUM

5. Find the ticket number of passenger whose name starts with ‘S’ and ends with ‘H’

SQL> select Ticket\_NO from Passenger where Name like'S%'and name like'%N';

TICKET\_NO

----------

1

6. Find the names of the passenger whose age is between 20 and 40

SQL> select Name from Passenger where age between 20 and 40;

NAME

--------------------

swetha

rafi

riyaz

neha



DBMS RECORD

Page No: 47

7. Display all the name of the passengers beginning with ‘r’

SQL> select Name from Passenger where Name like 'r%';

NAME

--------------------

rahul

rafi

riyaz

8. Display the sorted list of Passenger Names

SQL> select Name from Passenger ORDER BY Name;

NAME

--------------------

SACHIN

neha

rafi

rahul

riyaz

salim

swetha

7 rows selected.



DBMS RECORD

Page No: 48

Exp No: 7

Date: \_ \_/\_ \_/ \_ \_

AIM: Practice Queries using Aggregate functions, Group By, Having

Order Clause.

Group Functions:

Clause and

A group functions returns a result based on a group of rows. Some of these are just purely

mathematical functions. The group functions supported by Oracle are summarized below:

1) Avg (Average): This function will return the average of values of the column specified in

the argument of the column.

Example:

SQL> select avg(comm) from emp\_master;

AVG(COMM)

————-

766.66667

2) Min (Minimum): The function will give the least of all values of the column present in

the argument.

Example:

SQL>Select min(salary) from emp\_master;

MIN(SALARY)

—————-

3400

3) Max (Maximum): To perform an operation, which gives the maximum of a set of values

the max, function can be made use of.

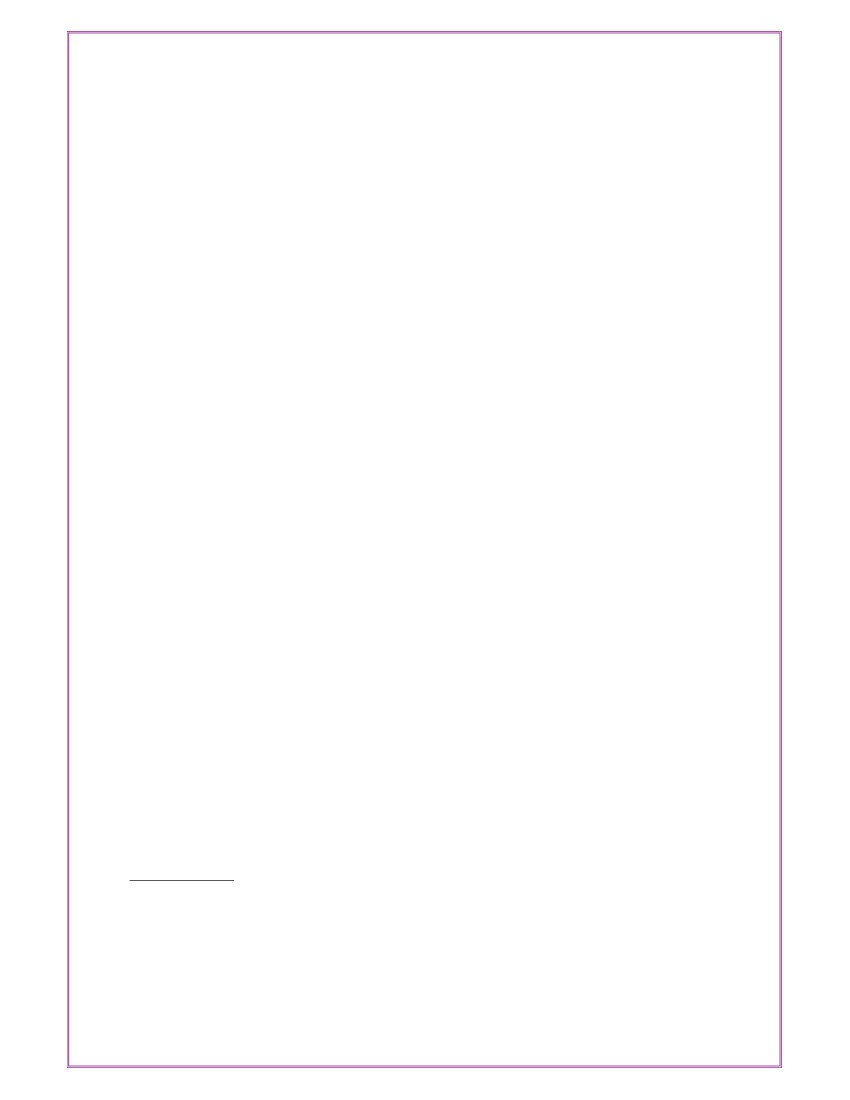
Example:

SQL>select max(salary) from emp\_master;

This query will return the maximum value of the column specified as the argument.

MAX(SALARY)

—————-



DBMS RECORD

Page No: 49

10000

4) Sum: The sum function can be used to obtain the sum of a range of values of a record set.

Example:

SQL>Select sum(comm) from emp\_master;

SUM(COMM)

————-

2300

5) Count: This function is used to count number rows. It can take three different arguments,

which mentioned below.

Syntax:

Count(\*)

Count(column name)

Count(distinct column name)

Count (\*): This will count all the rows, including duplicates and nulls.

Example:

SQL>Select count(\*) from emp\_master;

COUNT(\*)

————

4

Count (Column name) : It counts the number of values present in the column without

including nulls.

Example:

SQL> select count(comm) from emp\_master;

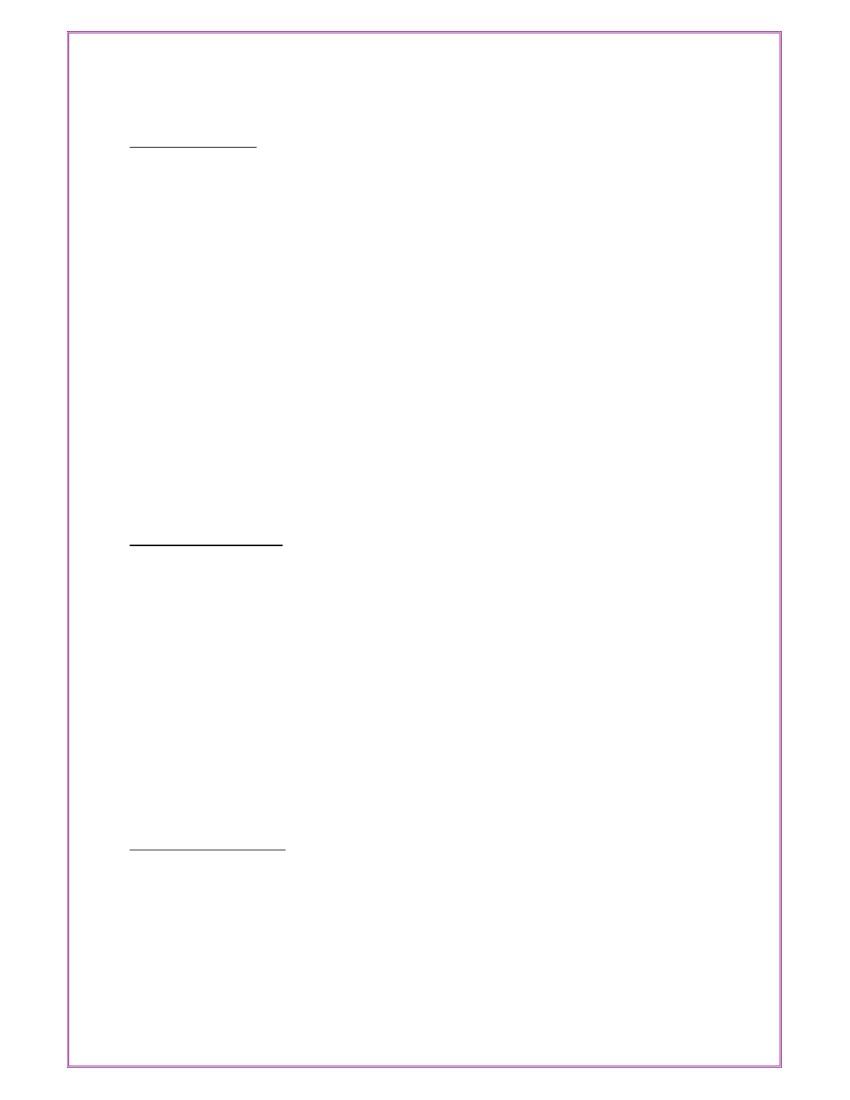
COUNT(comm)

3

Count (distinct column name) : It is similar to count(column name) but eliminates duplicate

values while counting.

Example:



DBMS RECORD

Page No: 50

SQL>Select count(distinct deptno) from emp\_master;

COUNT(DEPTNO)

2

Group By Clause

Group by clause is used with group functions only. Normally group functions returns

only one row. But group by clause will group on that column. The group by clause tells

Oracle to group rows based on distinct values for specified columns, i.e. it creates a data set,

containing several sets of records grouped together based on a condition.

Select group function from table name group by column name

Example:

SQL>select deptno,count(\*) from emp\_master group by deptno;

DEPTNO COUNT(\*)

10

20

2

2

Having Clause

The having clause is used to satisfy certain conditions on rows, retrieved by using group by

clause.Having clause should be preceding by a group by clause. Having clause further filters

the rows return by group by clause.

Example

SQL> select deptno,count(\*) from emp\_master group by deptno having Deptno is not null;

DEPTNO

10

20

COUNT(\*)

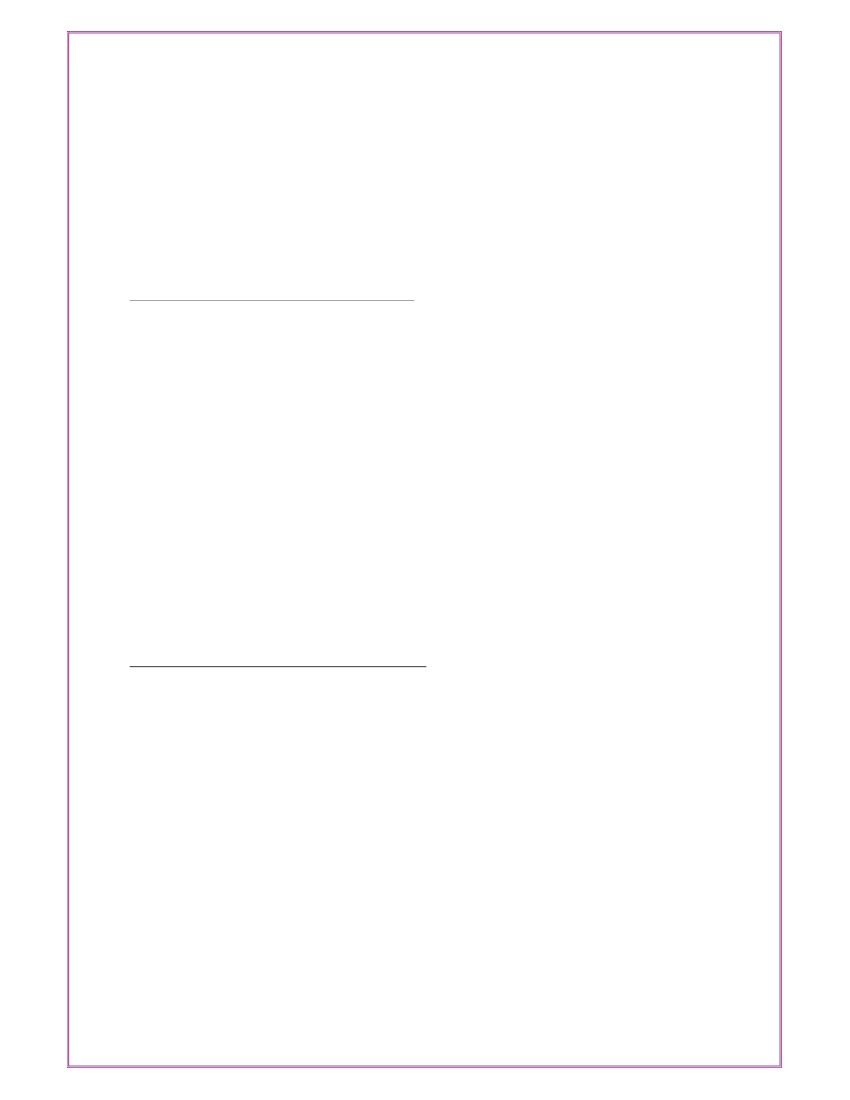
2

2

Order By Clause

Order by clause is used to arrange rows in either ascending or descending order. The order

by clause can also be used to arrange multiple columns. The order by clause should be the



DBMS RECORD

Page No: 51

last clause in select statement.It is used as per follows :

select

<column(s)> from <TableName> where [condition(s)]

[order by <column

name> [asc/]desc];

Example

If you want to view salary in ascending order the following command can performed:

SQL> select empno,ename,salary from emp\_master order by salary;

EMPNO

1123

1124

1122

1125

4 rows selected.

If you have not specify any order by default it will consider ascending order and salary will

be displayed in ascending order. To retrieve data in descending order the desc keyword is

used after order by clause.

SQL> select empno,ename,salary from emp\_master order by salary desc;

And the output will opposite from above.

ENAME

King

Martin

Allen

Tanmay

SALARY

3400

7000

10000

10000

EMPNO

1122

1125

1124

1123

4 rows selected.

ENAME

Allen

Tanmay

Martin

King

SALARY

10000

10000

7000

3400

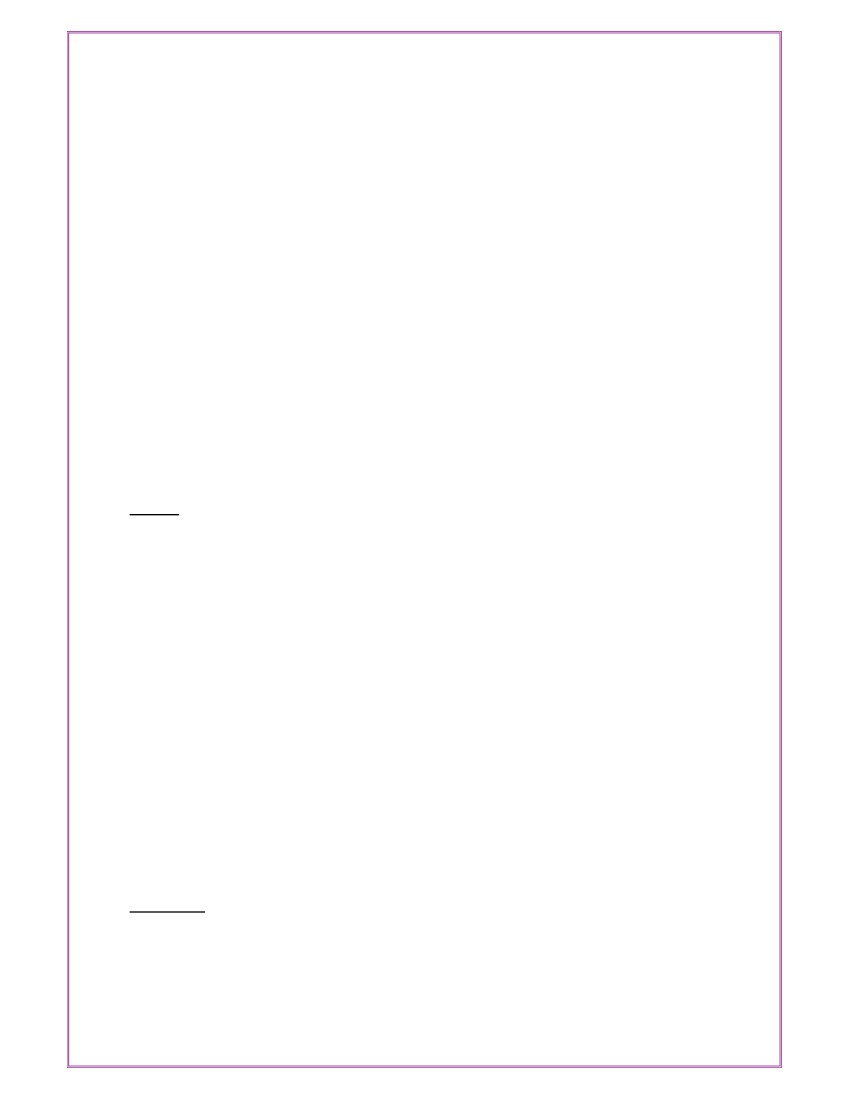
SQL \*Plus having following operators.

 Arithmetic Operators

 Comparison Operators

 Logical Operator

Arithmetic Operator



DBMS RECORD

Page No: 52

Arithmetic operators are used to perform calculations based on number values. The

arithmetic operators are + (addition), - (subtraction), \* (multiplication) and / (division). We

can include them in sql command.

Example

SQL> select salary+comm from emp\_master;

Salary+comm

11000

3700

8000

(Null)

4 rows selected.

Example:

SQL> select salary+comm net\_sal from emp\_master;

NET\_S

11000

3700

8000

(Null)

4 rows selected.

In above query, it will give output of salary+comm and net\_sal is column alias, which is

used to change column heading. So the output will be displayed under the net\_sal heading. If

you calculate any number value with null value, it will always return null value.

In arithmetic operators \* and / have equal higher precedence. And + and – have equal lower

precedence.

Check the following illustrates the precedence of operators.

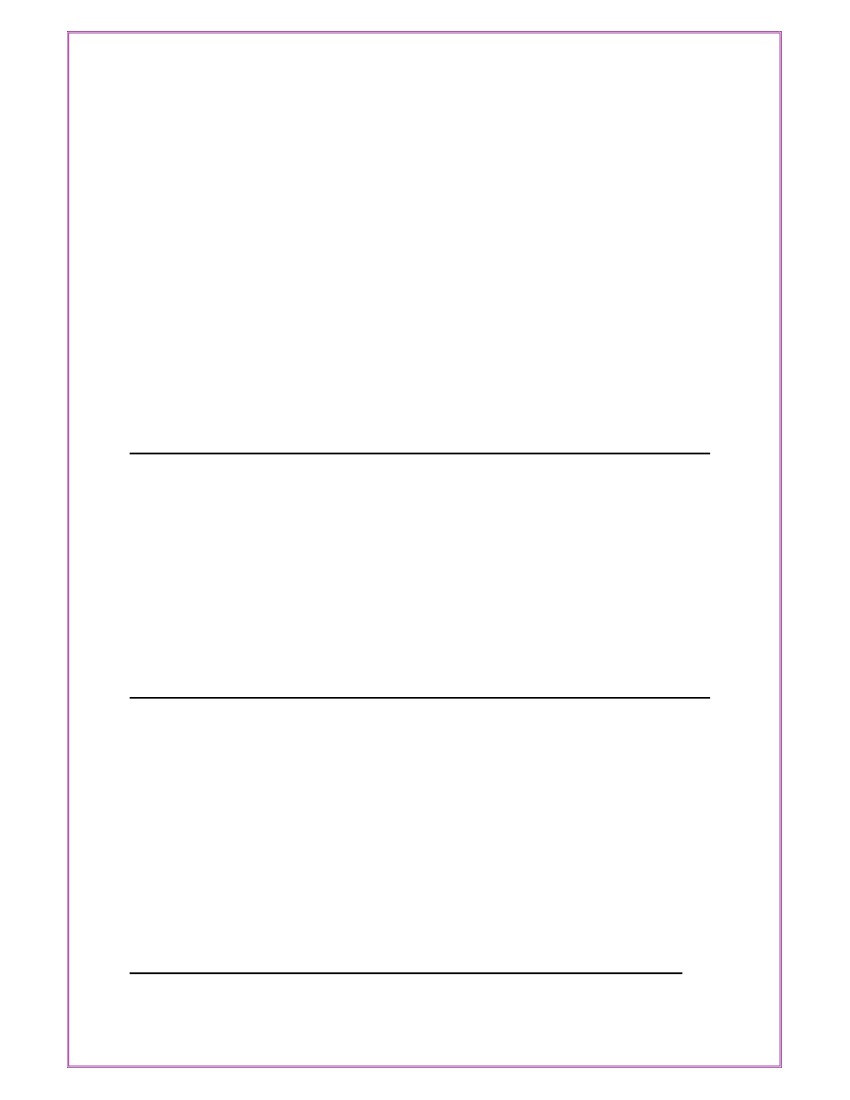
SQL> Select 12\*(salary+comm) annual\_netsal from emp\_master;

ANNUAL\_

132000

44400

96000



DBMS RECORD

Page No: 53

(Null)

4 rows selected.

If the parenthesis is omitted then multiplication will be performed first followed by addition.

We can change the order of evaluation by using parenthesis.

Comparison Operators:

Comparison operators are used in condition to compare one expression with other. The

comparison operators are =, >, <, >=, <=, !=, between, like , is null and in operators.

Between operator is used to check between two values.

33

Example:

SQL> select \* from emp\_master where salary between 5000 and 8000;

EMPNO

1124

ENAME

Martin

JOB

HIREDATE

SALARY

7000

DEPTNO COMM

20

1000

Manager 30-aug-00

1 row selected.

The above select statement will display only those rows where salary of employee is

between 5000 and 8000.

IN Operator:

The in operator can be used to select rows that match one of the values in a list.

SQL>Select \* from emp\_master where deptno in(10,30);

EMPNO

1122

1125

ENAME

Allen

Tanmay

JOB

HIREDATE

SALARY

10000

10000

DEPTNO COMM

10

10

1000

Manager 1-JAN-00

16-SEP-00

2 rows selected.

The above query will retrieve only those rows where deptno is either in 10 or 30.

LIKE Operator:

Like operator is used to search character pattern, we need not know the exact character

value. The like operator is used with special character % and \_ (underscore).

SQL> select \* from emp\_master where job like ‘M%’;

EMPNO

1122

ENAME

Allen

JOB

HIREDATE

SALARY

10000

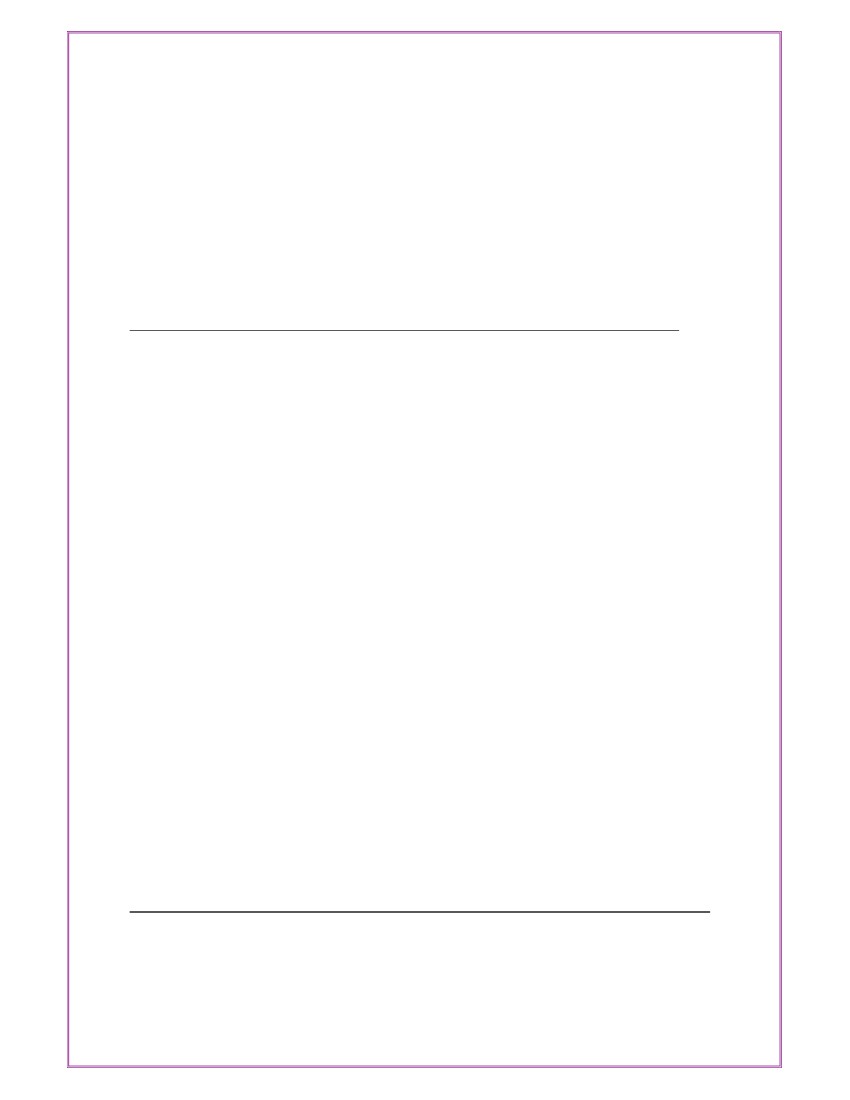
DEPTNO

10

COMM

1000

Manager 1-jan-00



DBMS RECORD

Page No: 54

1124

Martin

Manager 30-aug-00

7000

20

1000

2 rows selected.

The above select statement will display only those rows where job is starts with ‘M’

followed by any number of any characters. % sign is used to refer number of characters (it

similar to \* asterisk wildcard in DOS), while \_ (underscore) is used to refer single

character(it similar to ? question wildcard in DOS).

SQL>Select \* from emp\_master where job like ‘\_lerk’;

EMPNO

1123

1 row selected.

In above query, it will display only those rows where job is start with any single character

but ends with ‘clerk’.

Logical Operators:

Logical operators are used to combine the results of two conditions to produce a single

result. The logical operators are AND, NOT and OR.

AND Operator:

The Oracle engine will process all rows in a table and display the result only when all the

conditions specified using the AND operator are satisfied.

SQL> select \* from emp\_master where salary > 5000 and comm < 750 ;

No rows selected.

The select statement will return only those rows where salary is greater than 5000 and comm

is less than 750. If both the conditions are true then only it will retrieve rows.

OR Operator:

The Oracle engine will process all rows in a table and display the result only when any of

the conditions specified using the OR operators are satisfied.

SQL>select \* from emp\_master where salary > 5000 or comm < 750;

EMPNO

1122

1123

1124

ENAME

Allen

King

JOB

Manager

Clerk

HIREDATE

1-jan-00

30-jun-00

30-aug-00

SALARY

10000

3400

7000

10

20

20

DEPTNO

COMM

ENAME

King

JOB

Clerk

HIREDATE

30-jun-00

SALARY

3400

DEPTNO

20

COMM

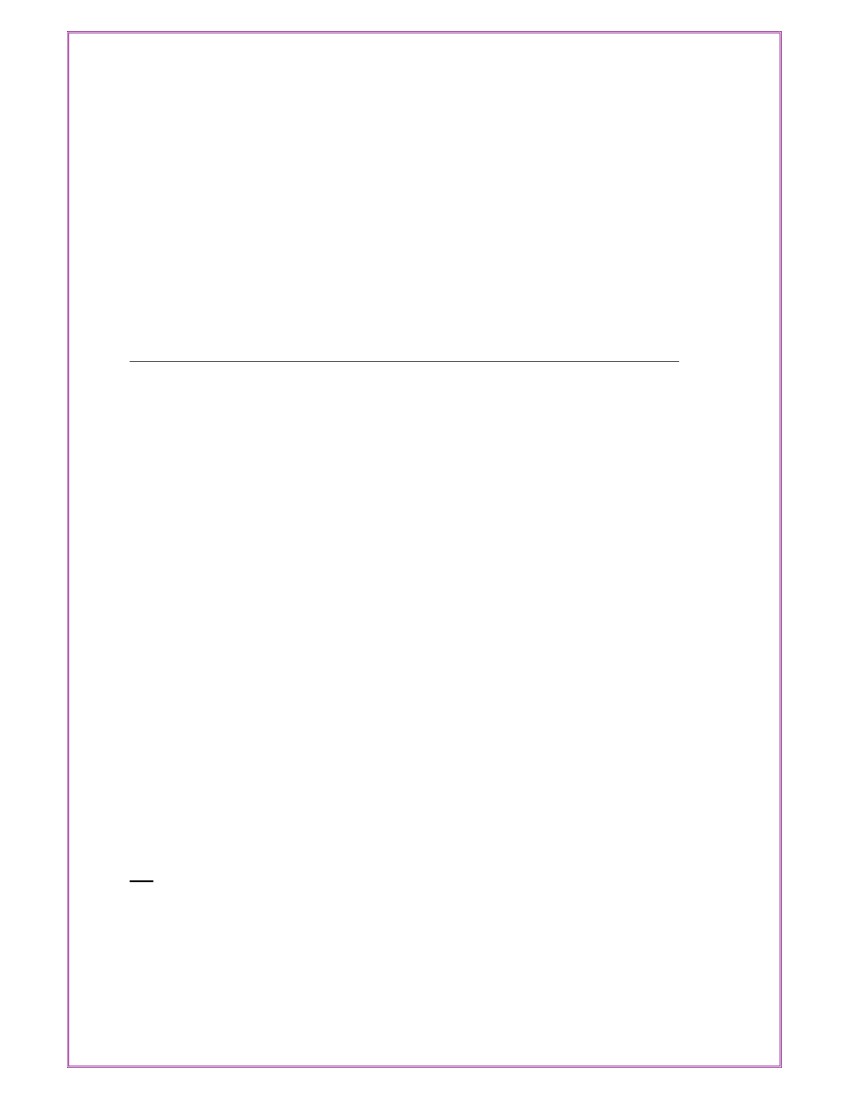
300

1000

300

1000

Martin Manager



DBMS RECORD

Page No: 55

1125

Tanmay

16-SEP-00

10000

10

4 rows selected.

This select statement will check either salary is greater than 5000 or comm is less than 750.

ie it will return all the records either of any one condition returns true.

NOT Operator:

The Oracle engine will process all rows in a table and display the result only when none of

the conditions specified using the NOT operator are satisfied.

SQL> select \* from emp\_master where not salary = 10000;

EMPNO

1123

1124

ENAME

King

Martin

JOB

Clerk

HIREDATE

30-jun-00

SALARY

3400

7000

DEPTNO

20

20

COMM

300

1000

Manager 30-aug-00

2 rows selected.

This select statement will return all the records where salary is NOT equal to 10000.

Pre-define Functions

Oracle functions serve the purpose of manipulating data items and returning a result.

Functions are also capable of accepting user-supplied variables or constants and operations

on them. Such variables and constants are called arguments.

Functions are classified into Group Functions and Single Row Functions (Scalar

Functions).Before we check single row function and group function, we will take a look on

Dual table”

The Oracle Table Dual”

Dual is a small oracle worktable, which consists of only one row and one column, and

contains the value x in that column. Besides arithmetic calculations, it also supports date

retrieval and it’s formatting.

SQL> select 2\*2 from dual;

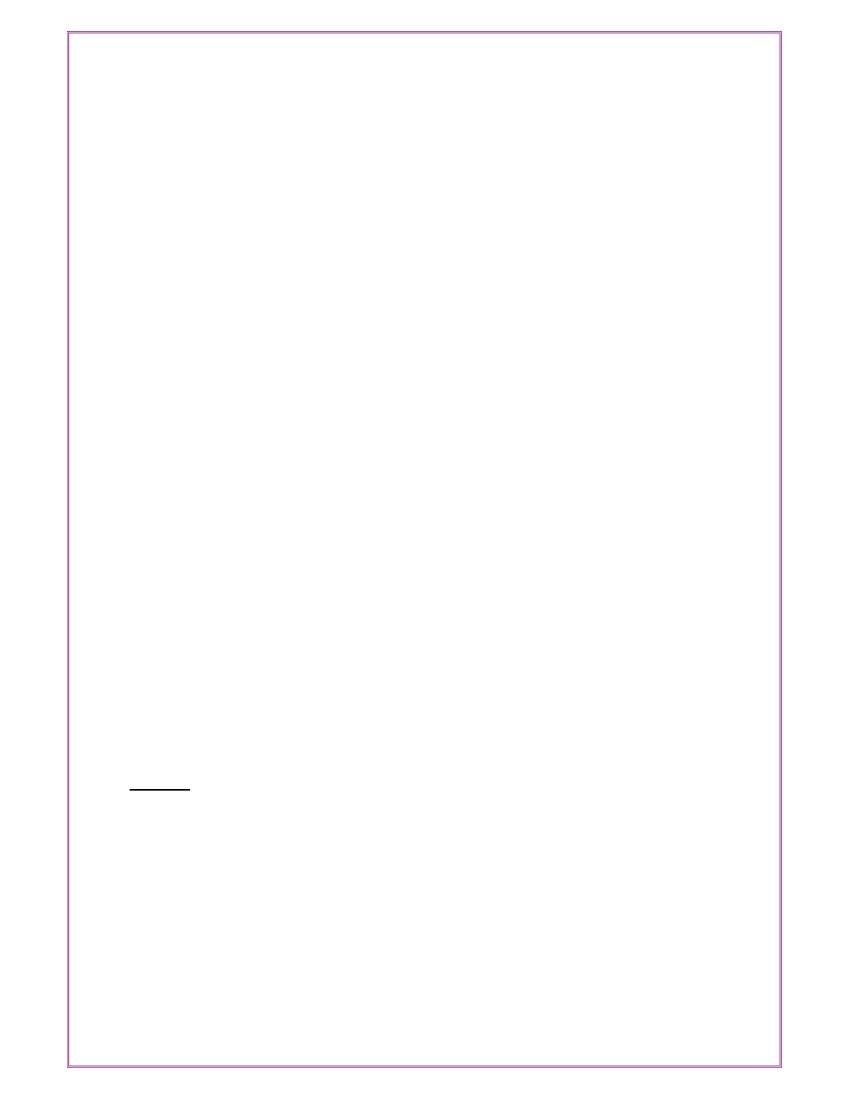
2\*2

4

Single Row Functions (Scalar Functions):

Functions that act on only one value at a time are called as Single Row Functions. A

Single Row function returns one result for every row of a queried table or view.



DBMS RECORD

Page No: 56

Single Row functions can be further grouped together by the data type of their arguments

and return values. Functions can be classified corresponding to different data types as :

 String Functions : Work for String Data type

 Numeric Functions : Work for number Data type

 Conversion Functions : Work for conversion of one data type to another

 Date Functions : Work for Date Data type 36

String Functions:

String functions accept string input and return either string or number values.

1) Initcap (Initial Capital): This String function is used to capitalize first character of the

input string.

Syntax:

initcap(string)

Example:

SQL> select initcap(‘azure’) from dual;

INITC

Azure

2) Lower: This String function will convert input string in to lower case.

Syntax:

Lower(string)

Example:

SQL> select lower(‘AZURE’) from dual;

LOWER

azure

3) Upper: This string function will convert input string in to upper case.

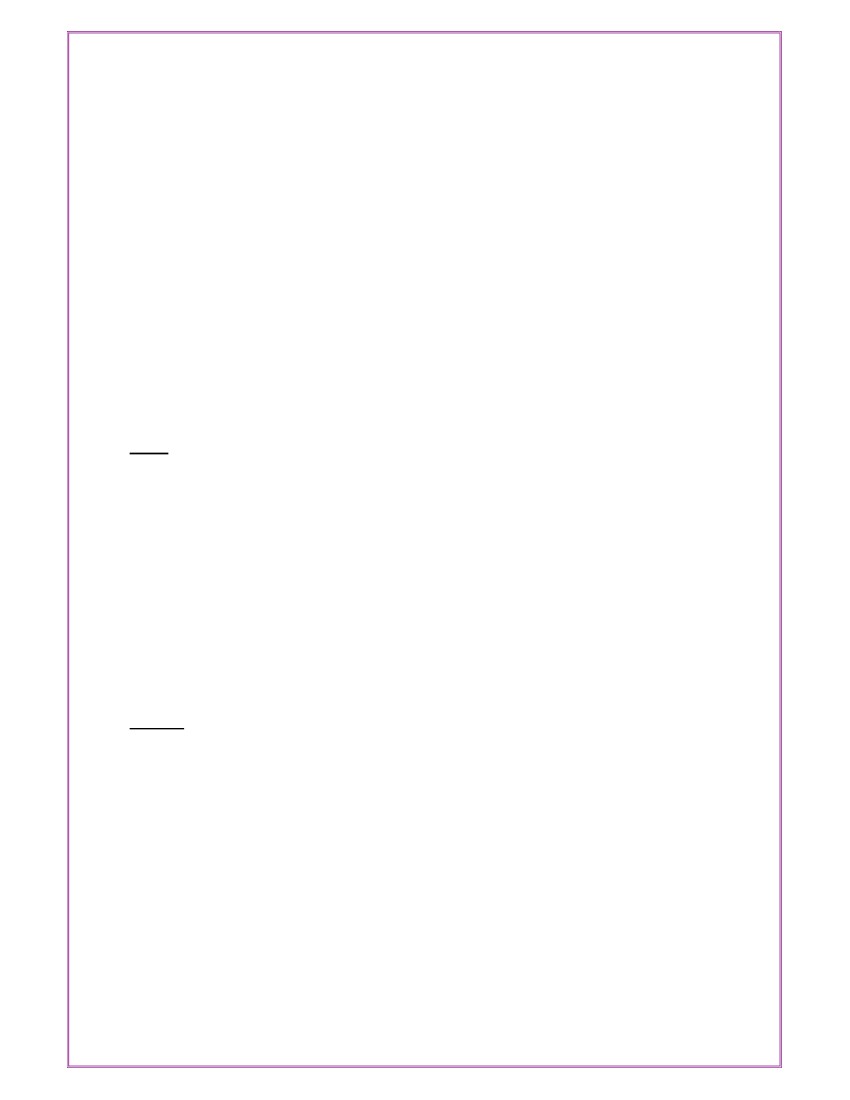
Syntax:

Upper(string)

Example:

SQL> select upper(‘azure’) from dual;

UPPER



DBMS RECORD

Page No: 57

——-

AZURE

4) Ltrim (Left Trim): Ltrim function accepts two string parameters; it will fetch only those

set of characters from the first string from the left side of the first string, and displays only

those characters which are not present in second string. If same set of characters are not

found in first string it will display whole string

Syntax:

Ltrim(string,set)

Example:

SQL>select ltrim(‘azuretech’,’azure’) from dual;

LTRI

tech

5) Rtrim (Right Trim): Rtrim function accepts two string parameters; it will fetch only

those characters from the first string, which is present in set of characters in second string

from the right side of the first string.

Syntax:

Rtrim(string,set)

Example:

SQL>select rtrim(‘azuretrim’,’trim’) from dual;

RTRIM

azure

6) Translate: This function is useful when you want to encrypt string. It will take first

character from string1 and search the same character in string2 if that character is found than

it replaces that character out of string3 on base of position of character found in string2. In

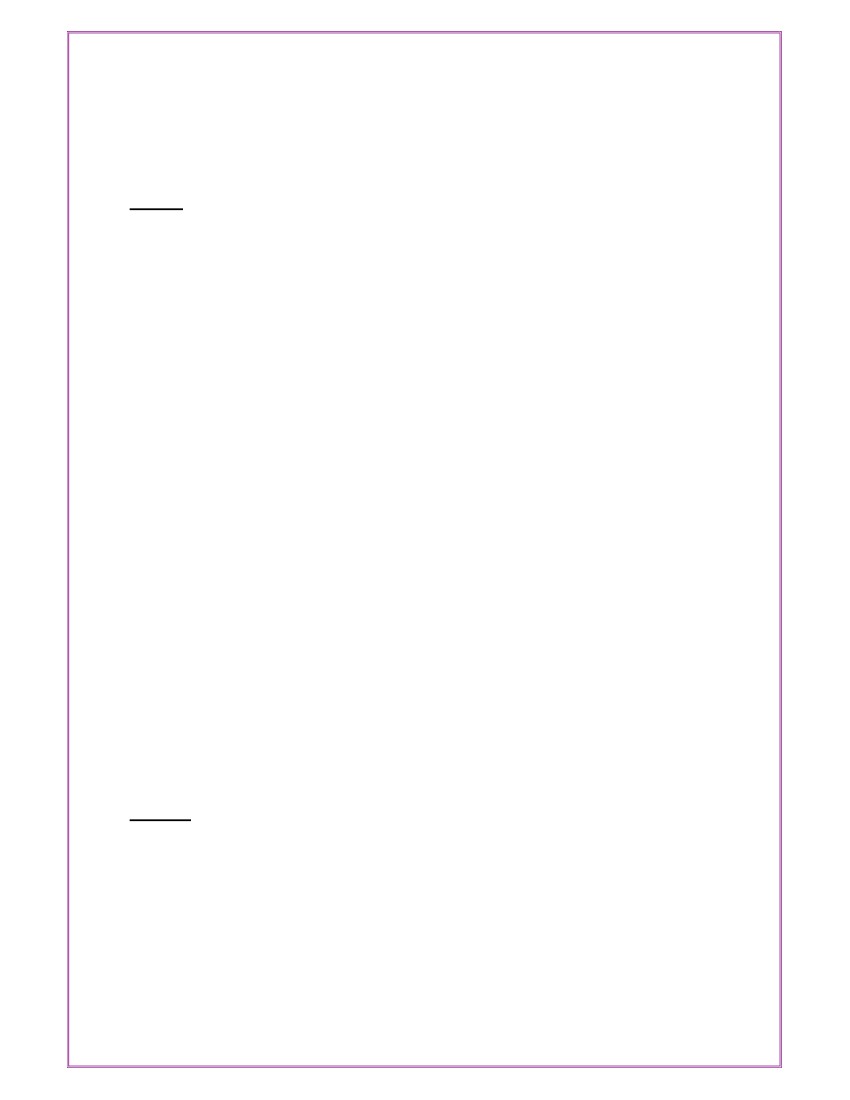
below given example first character of string1 is found at position no 2 in string2, so it will

extract second character from string3. Same way second character is found at position

number 4 in string2, so it will extract fourth character from string3 and so on. If any

character in string1 is not found in string2 then it is kept unchanged.

Syntax:



DBMS RECORD

Page No: 58

Translate(string1, string2, string3)

Example:

SQL>select translate(‘abcde’,’xaybzcxdye’,’tanzmulrye’) from dual;

TRANS

azure

7) Replace: This function is useful when you want to search a specified string and replace it

with particular string form the string provided. For example, you want to search ‘A’ from

the ‘TACHNOLOGIAS’ and replace it with ‘E’ to make it ‘TECHNOLOGIES’. Replace

function accepts three arguments first argument is, from which string you want to search,

second argument is what you want to search from the first argument and third argument is

replace string, value of second argument, if found will be replaced with value passed in third

argument.

Syntax:

Replace(string, searchstring, replacestring)

Example:

SQL> select replace(‘jack and jue’,’j’,’bl’) from dual;

8) Substr: Substring fetches out a piece of the string beginning at start and going for count

characters, if count is not specified, the string is fetched from start and goes till end of the

string.

Syntax:

Substr(string, starts [, count])

Example:

SQL>select substr(‘azuretechnology’,4,6) from dual;

SUBSTR

retech

9) Chr: Character function except character input and return either character or number

values. The first among character function is chr. This returns the character value of given

number within braces.

Syntax:

Chr(number)



DBMS RECORD

Page No: 59

Example:

SQL>select chr(65) from dual;

A

10) Lpad (Left Pad): This function takes three arguments. The first argument is character

string, which has to be displayed with the left padding. Second is a number, which indicates

total length of return value and third is the string with which left padding has to be done

when required.

Syntax:

Lpad(String,length,pattern)

Example:

Sql > select lpad(‘Welcome’,15,’\*’) from dual;

LPAD(‘WELCOME’,

———————-

\*\*\*\*\*\*\*\*Welcome

11) Rpad (Right Pad): Rpad does exact opposite then Lpad function.

Syntax:

Lpad(String,length,pattern)

Example:

SQL> select rpad(‘Welcome’,15,’\*’) from dual;

RPAD(‘WELCOME’,

———————-

Welcome\*\*\*\*\*\*\*\*

12) Length: When the length function is used in a query. It returns length of the input string.

Syntax:

Length(string)

Example:

SQL>select length(‘auzre’) from dual;

LENGTH(‘AUZRE’)

———————-

5



DBMS RECORD

Page No: 60

13) Decode: Unlike the translate function which performs character-by-character

replacement the decode function does a value-by-value replacement.

Syntax:

Select decode(column name,if,then,if,then\_....) from <tablename>;

Example:

SQL> select deptno,decode(deptno,10, ‘Sales’, 20, ‘Purchase’, ‘Account’)

DNAME from emp\_master;

DEPTNO DNAME

——— ————

10 Sales

20 Purchase

20 Purchase

10 Sales

4 rows selected.

14) Concatenation ( || ) Operator: This operator is used to merge two or more strings.

Syntax:

Concat(string1,string2)

SQL> select concat(‘Azure’,’ Technology’) from dual;

CONCAT(‘AZURE’,’

————————

Azure Technology

SQL> select ‘ename is ‘||ename from emp\_master;

‘ENAME IS’||ENAME

————————-

ename is Allen

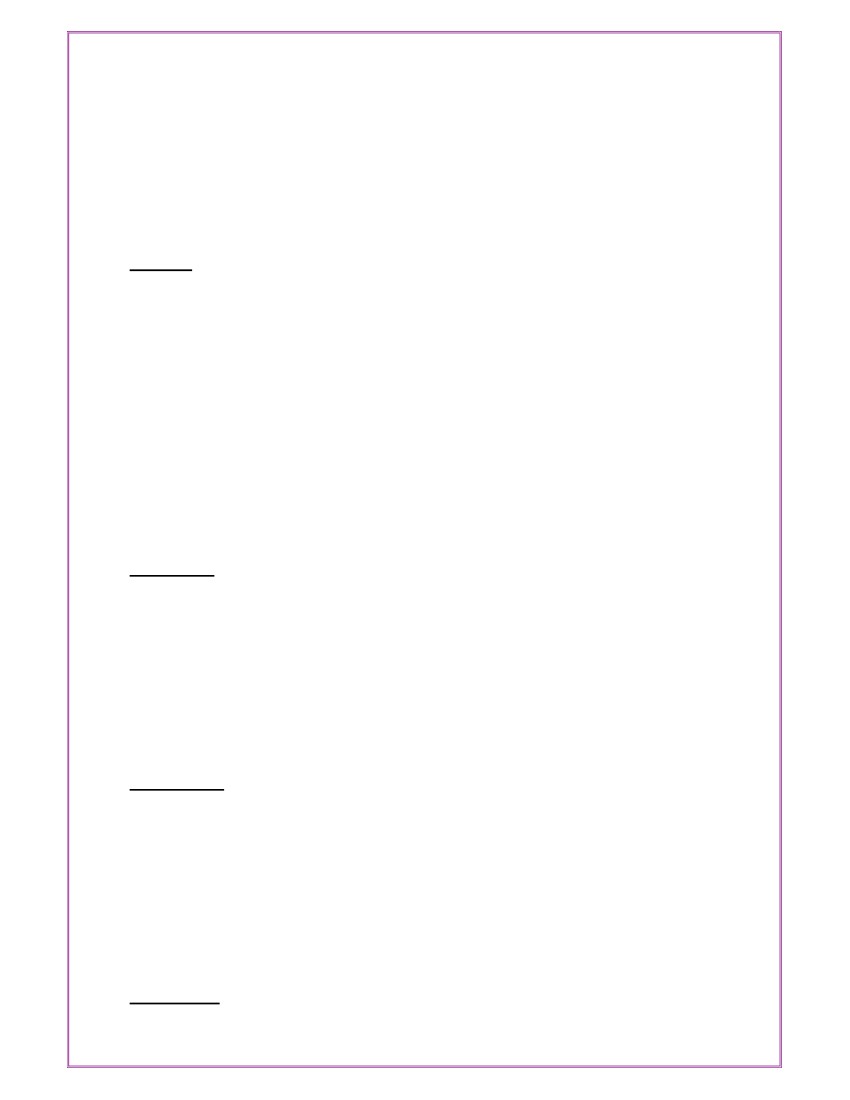
ename is King

ename is Martin

ename is Tanmay

4 rows selected.

Numeric Functions:



DBMS RECORD

Page No: 61

1) Abs (Absolute): Abs() function always returns positive number.

Syntax:

Abs(Negetive Number)

Example:

SQL> select Abs(-10) from dual;

ABS(-10)

10

2) Ceil: This function will return ceiling value of input number. i.e. if you enter 20.10 it will

return 21 and if you enter 20.95 then also it will return 21. so if there is any decimal value it

will add value by one and remove decimal value.

Syntax:

Ceil (Number)

Example:

SQL>select Ceil (23.77) from dual;

CEIL(23.77)

24

3) Floor: This function does exactely opposite of the ceil function.

Syntax:

Floor(Number)

Example:

SQL>select Floor(45.3) from dual;

FLOOR(45.3)

45

4) Power: This function will return power of raise value of given number.

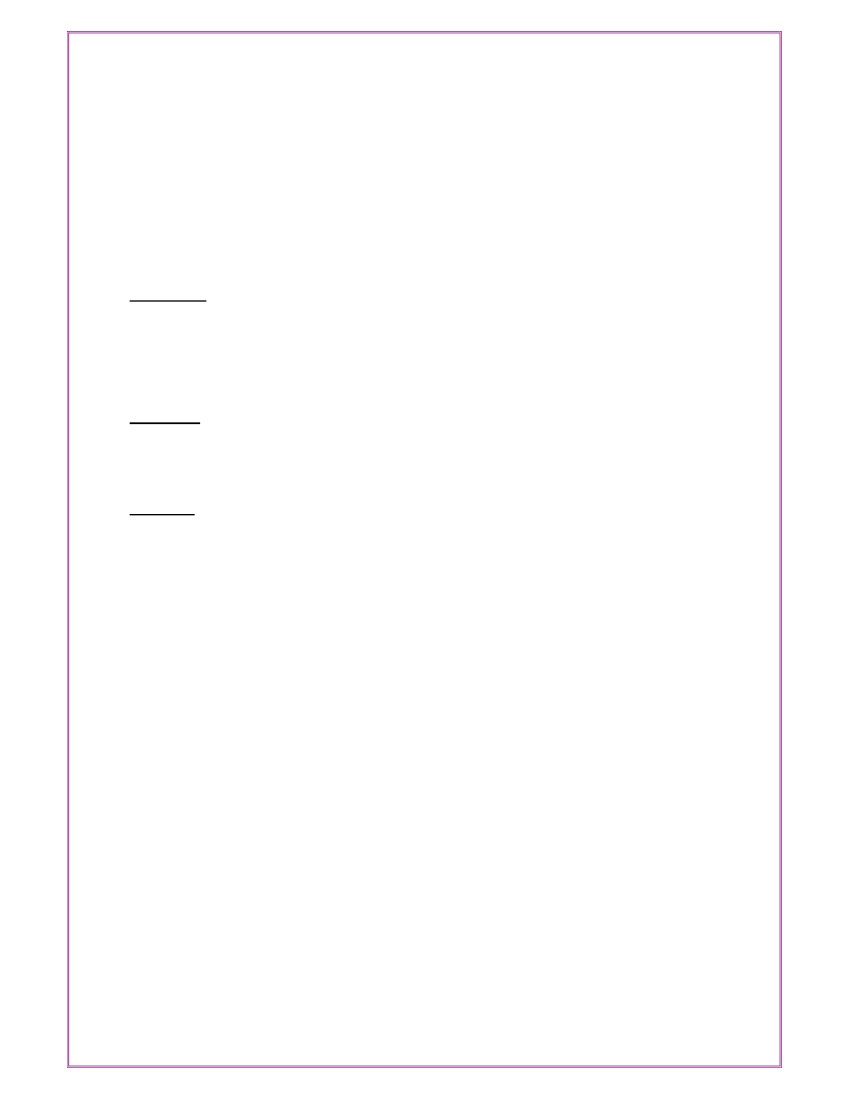
Syntax:

Power(Number, Raise)

Example:

SQL>Select power (5,2) from dual;

POWER(5,2)



DBMS RECORD

Page No: 62

25

5) Mod: The function gives the remainder of a value divided by another value.

Syntax:

Mod(Number, DivisionValue)

Example:

SQL>select Mod(10,3) from dual;

MOD(10,3)

1

6) Sign: The sign function gives the sign of a value without it’s magnitude.

SQL>select sign(-45) from dual;

SIGN(-45)

-1

SQL>Select sign(45) from dual;

SIGN(45)

1

Date Function:

1) Add\_Months: The add\_months data function returns a date after adding a specified data

with the specified number of months. The format is add\_months(d,n), where d is the date

and n represents the number of months.

Syntax:

Add\_Months(Date,no.of Months)

Example:

SQL> select Add\_Months(sysdate,2) from dual;

This will add two months in system date.

ADD\_MONTH

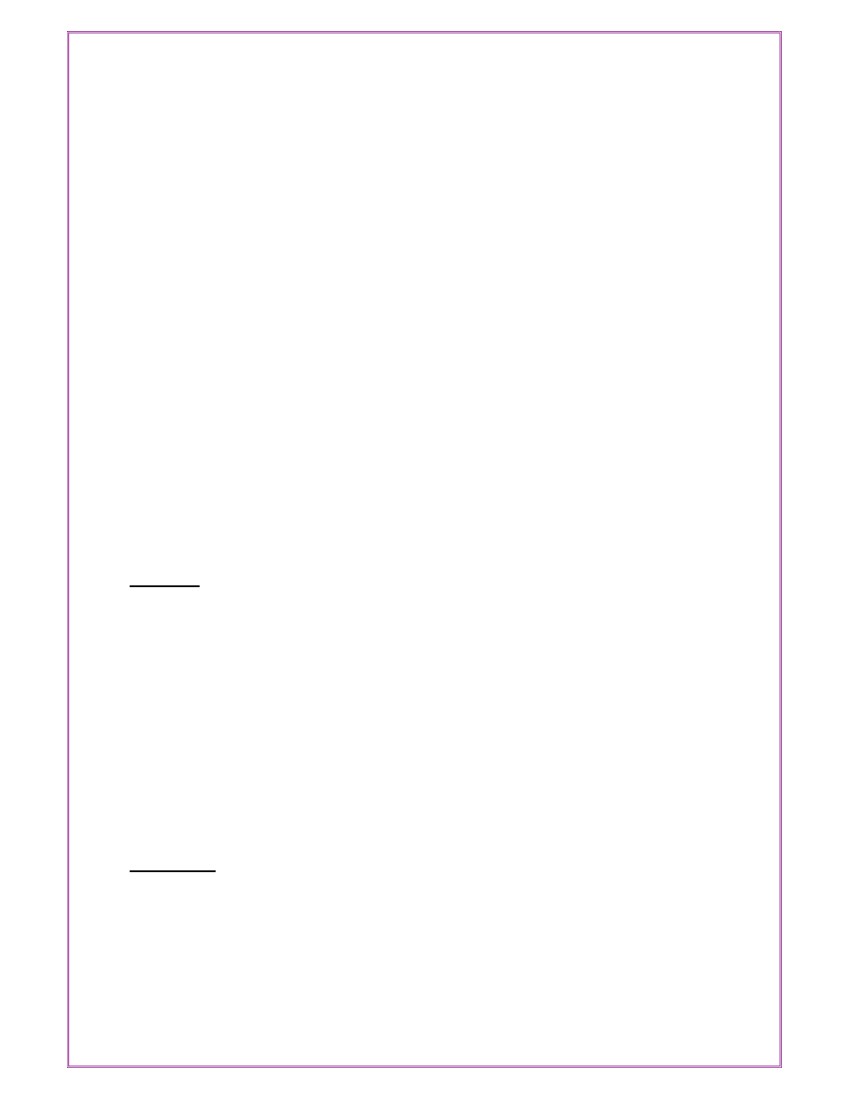
———

02-NOV-01

2) Last\_day: Returns the last date of month specified with the function.

Syntax:

Last\_day(Date)



DBMS RECORD

Page No: 63

Example:

SQL> select sysdate, last\_day(sysdate) from dual;

SYSDATE LAST\_DAY

———— ————-

02-SEP-01 30-SEP-01

3) Months\_Between: Where Date1, Date2 are dates. The output will be a number. If Date1

is later than Date2, result is positive; if earlier, negative. If Date1 and Date2 are either the

same days of the month or both last days of months, the result is always an integer;

otherwise Oracle calculates the fractional portion of the result based on a 31-day month and

considers the difference in time components of Date1 and Date2.

Syntax:

Months\_Between(Date1,Date2)

Example:

SQL>select months\_between(sysdate,’02-AUG-01’) 溺onths\_ from dual;

MONTHS

4

4) Next\_Day: Returns the date of the first weekday named by ‘char’ that is after the date

named by‘Date’. ‘Day’ must be the day of the week.

Syntax:

Next\_Day(Date,Day)

Example:

SQL>select next\_day(sydate, ‘sunday’) 哲ext\_ from dual;

This will return date of next sunday.

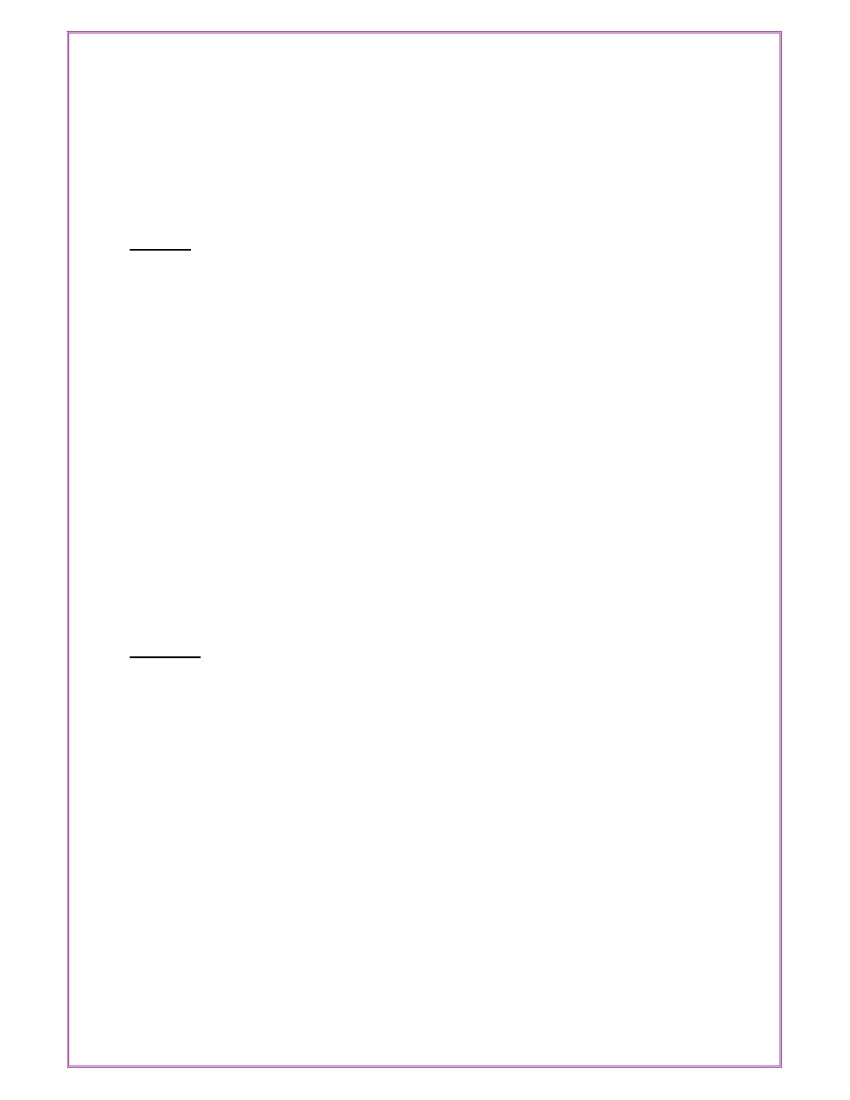
NEXT\_DAY

09-SEP-00

5) Round: This function returns the date, which is rounded to the unit specified by the

format.

Syntax:



DBMS RECORD

Page No: 64

Round (Date, [fmt])

If format is not specified by default date will be rounded to the nearest day.

Example:

SQL>Select round(‘4-sep-01’,’day’) 迭ounded\_ from dual;

Rounded

02-SEP-01

The date formats are ‘month’ and ‘year’.

If rounded with ‘month’ format it will round with nearest month.

If rounded with ‘year’ format it will round with nearest year.

6) Trunc (Truncate): This function returns the date, which is truncated to the unit specified

by the format.

Syntax:

Trunc(Date,[fmt])

If format is not specified by default date will be truncated.

Example:

This will display first day of current week.

SQL>Select Trunc(‘4-sep-01’,’day’) 典runcated\_ from dual;

Truncated

02-SEP-01

The date formats are ‘month’ and ‘year ’.

If rounded with ‘month’ format it will display first day of the current month.

If rounded with ‘year’ format it will display first day of the current year.

Conversion Functions:

Conversion functions convert a value from one data type to another. The conversion

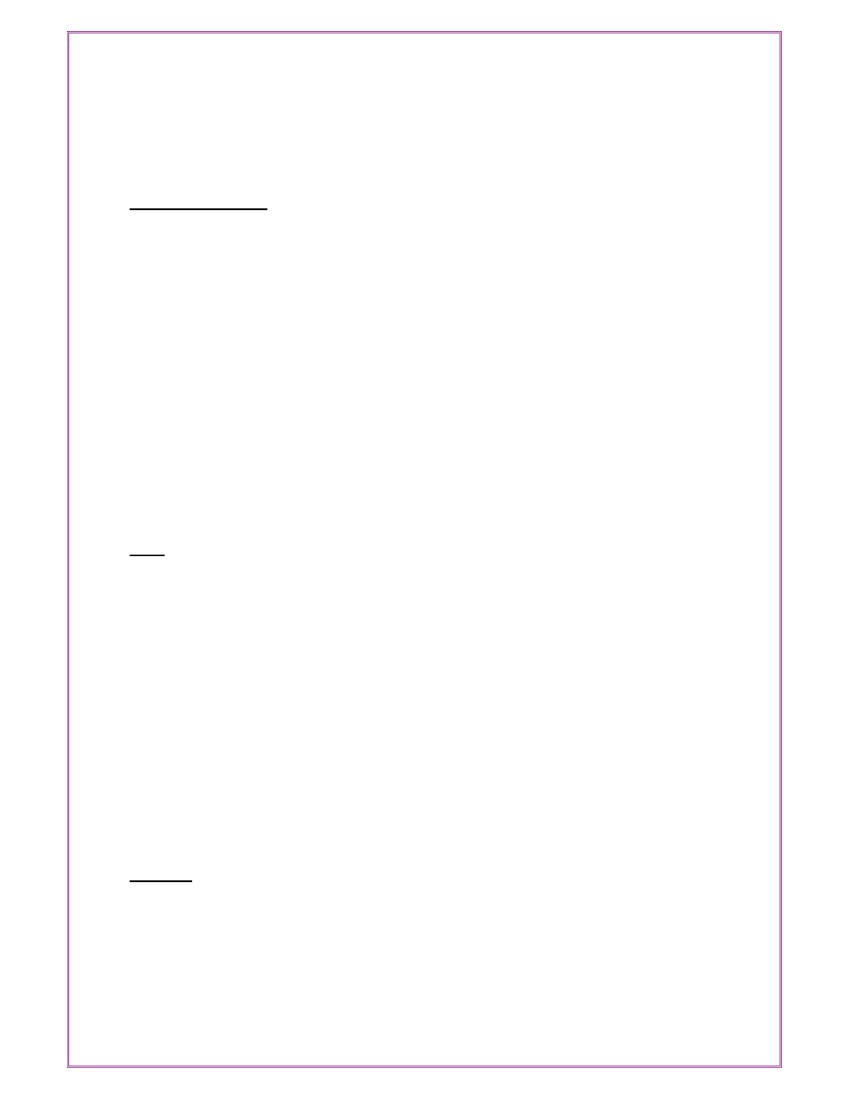
functions are classified into the following:

 To\_Number()

 To\_Char()

 To\_Date()

1) To\_Number: The to\_number function allows the conversion of string containing



DBMS RECORD

Page No: 65

numbers into the number data type on which arithmetic operations can be performed.

Example:

SQL>Select to\_number(‘50’) from dual;

TO\_NUMBER(‘50’)

50

2) To\_Char: To\_char function converts a value of number data type to a value of char data

type, using the optional format string. It accepts a number (no) and a numeric format (fmt) in

which the number has to appear. If ‘fmt’ is omitted, ‘no’ is converted to a char exactly long

enough to hold significant digits.

Syntax:

To\_char(no,[fmt])

Example:

SQL> select to\_char(17145,’$099,999’) 鼎har\_ from dual;

Char

$017,145

To\_char converts a value of date datatype to character value. It accpets a date, as well as

the format(fmt) in which the date has to appear. ‘fmt’ must be the date format. If ‘fmt’ is

omitted, ‘date’ is converted to a character value in the default date format 電d-mon-yy\_.

Syntax:

To\_char(Date,[fmt])

Example:

SQL>select to\_char(hiredate, ‘month dd yyyy’) 滴ireDate\_ from emp\_master

where salary = 10000;

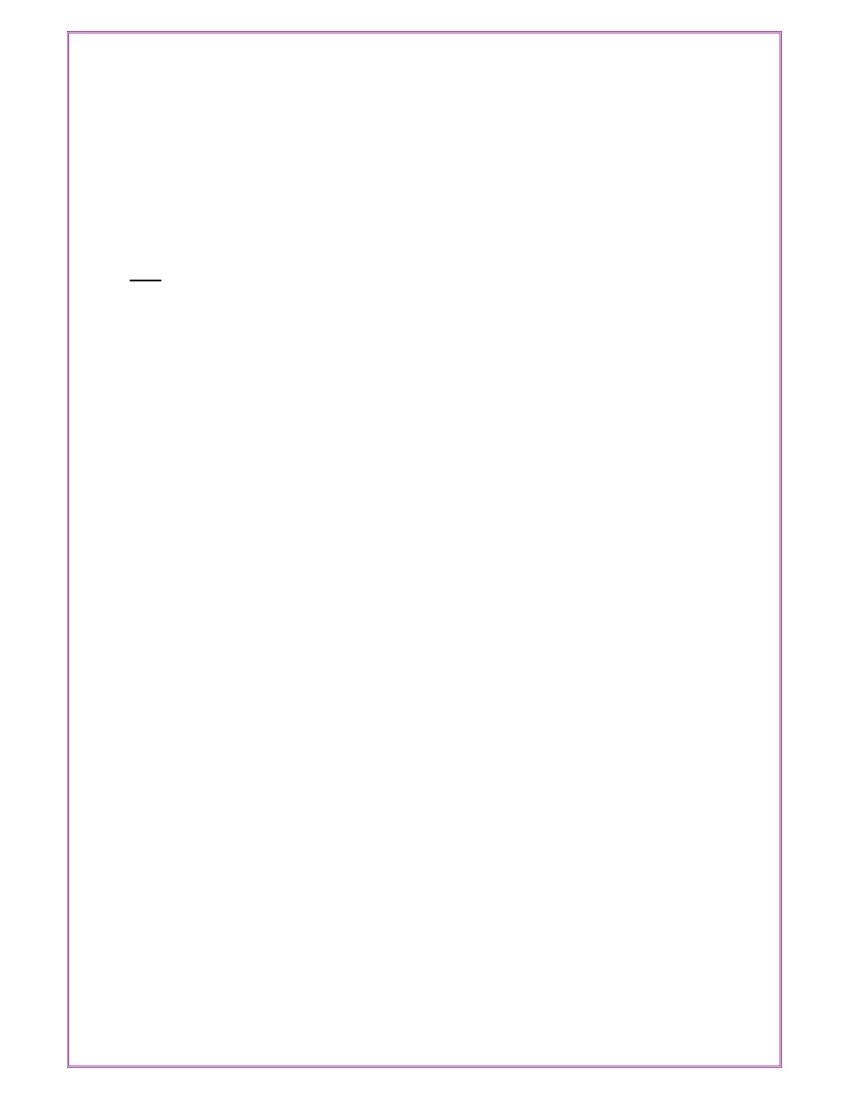
HireDate

January 01 2000

September 16 2000

3) To\_Date: The format is to\_date(char [,fmt]). This converts char or varchar datatype to

date datatype. Format model, fmt specifies the form of character. Consider the following



DBMS RECORD

Page No: 66

example which returns date for the string ‘January 27 2000’.

Syntax:

To\_date(char,[fmt])

Example:

SQL>select to\_date(’27 January 2000’,’dd/mon/yy’) 泥ate\_ from dual;

Date

27-JAN-00

Practice queries using Aggregate functions, Group by, having and Order By Clause.

1. Write a query to Display the information present in the Cancellation and Reservation

Tables

SQL> select \* from Reservation UNION select \* from Cancellation;

PNR\_NONO\_OF\_SEATSADDRESSCONTACT\_NOSTATUS

---------- ----------------------------------------- -----------------

12sdfgh1234543s

13msbtnk123456789s

22ldkp234567891s

22wertgfds12212121n

34dskng345678912n

35azxsdcvf13243546s

42ddfdsfsdfdsf3456789s

45abids567891234s

52allbd891234567s

511liopujth43256787s

61koti231456781s

PNR\_NONO\_OF\_SEATSADDRESSCONTACT\_NOSTATUS

----------------------------------------- ----------

631swebnht453212345s

72dbdhfdbhf90876543s

73jklhg2345671s

14 rows selected.

2. Find the distinct PNR\_NO that are present

SQL> SELECT PNR\_NO, COUNT(\*) AS NoOccurances FROM Passenger GROUP BY

PNR\_NO HAVING COUNT(\*)>0;

PNR\_NONOOCCURANCES

----------------------

11

21



DBMS RECORD

Page No: 67

31

41

51

61

71

7 rows selected.

3. Find the No of Seats booked for each PNR\_NO using GROUP BY Clause.

SQL> select PNR\_NO,sum(No\_of\_seats) from Reservation group by PNR\_NO;

PNR\_NOSUM(NO\_OF\_SEATS)

--------------------------

13

61

22

45

52

36

73

7 rows selected.

4. Find the number of seats booked in each class where the number of seats is greater than

1.

SQL> select class,sum(No\_of\_seats) from Reservation where class='a 'or class='b' or class=

'c' group by class having sum(No\_of\_seats)>1;

CLASSSUM(NO\_OF\_SEATS)

-------------------

a13

b7

c2

5. Find the total number of cancelled seats.

SQL> select sum(No\_of\_seats) from Cancellation;

SUM(NO\_OF\_SEATS)

----------------

22

6. Creating and dropping views



DBMS RECORD

Page No: 68

Exp No: 8

AIM : Implement Joins

Joins

Date: \_ \_/\_ \_/ \_ \_

Sometimes we require to treat multiple tables as though they were a single entity. Then a

single SQL sentence can manipulate data from all the tables. To achieve this, we have to join

tables. The purpose of join is to combine the data spread across tables. A join is actually

performed by the ‘where’ clause which combines the specified rows of tables.

Syntax for joining tables

select columns from table1, table2, ... where logical expression;

Basically there are three different types of joins:

 Simple Join

 Self Join

 Outer Join

Simple Join : This is the most frequently used join. It retrieves the rows from two tables

having a common column and is further classified into equi-join and non-equi join. Equi join

is based on equality and where clause uses comparison operator equal to (=) to perform a

join. Non-equi join specifies therelationship between columns belonging to different tables

by making use of relational operators( >,<,>=,<=, <>).

Example:

SQL> select \* from order\_master , order\_detail where Order\_master.order\_no =

order\_detail.order\_no;

This select statement will join the output of order\_master and order\_detail and display only

those rows where order\_master’s order\_no equals to order\_detail’s order\_no. In the example



DBMS RECORD

Page No: 69

the column name is prefixed by the table name because both the tables have the same

column name i.e. order\_no. Therefore to distinguish between them we use table names. If

the column names are unique, then we need not prefix it with the table name.

Example:

SQL> select a.\*, b.\* from itemfile a, order\_detail b where a.max\_level< b.qty\_ord

and a.itemcode = b.itemcode;

This select statement will retrieve rows from itemfile and order\_detail where qty\_ord of

order\_detail table is less than max\_level of itemfile and Itemcode are common in both the

table. Here a and b is indicating table aliases name. To prevent ambiguity in a query we

include table names in the select statements. Table aliases are used to make multiple table

queries shorter and more readable.

Self Join : In some situations, you may find it necessary to join a table to itself, as though

you were joining two separate tables. This is referred to as a self-join. In a self-join, two

rows from the same table combine to form a result row. To join a table to it self, two copies

of the very ame table have to be opened in memory. Hence in the fromclause, the table name

needs to be mentioned twice. Since the table names are same, the second table will overwrite

the first table and in effect, result in only one table being in memory. This is because a table

name is translated into specific memory location. To avoid this, each table to be opened

under an alias. Now these table aliases will cause two identical tables to be opened in

different memory locations. This will result in two identical tables to be physically present

in the computer’s memory.Display employees’salary with their manager’s salary. (Use

default emp table)

Example:

SQL>

select a.ename, a.salary, b.ename, b.salary from emp a, emp b where a.mgr =

b.empno;

This query will return employee name,salary with his manager’s name and salary.

Outer Join : Outer join extends the result of simple join. An outer join returns all the rows

returned by simple join as well as those rows from one table that do not match any row from

the other table. This cannot be with a simple join. The outer join is represented by (+) sign.

Example:



DBMS RECORD

Page No: 70

SQL> select \* from order\_master a, order\_detail b where a.order\_no = b.order\_no(+);

This select statement will return all the records from order\_master and only matching

records from order\_detail. If (+) is not specified then it is simple join and it will retrieve

only matching records from both tables.

Exp No: 9

AIM : Implement Sub Queries:

Subquery

Date: \_ \_/\_ \_/ \_ \_

A subquery is a form of an SQL statement that appears inside another SQL statement. It is

also termed as nested query. The statement containing a subquery is called parent query

statement. The parent statement uses the rows returned by the subquery. Subquery is always

enclosed within parenthesis. Subquery will be evaluated first followed by the main query.

Example:

SQL> select \* from order\_master where order\_no = (select order\_no from order\_detail

where order\_no = ‘O001’);

In this case subquery will execute first and the main query’s condition will work on

subquery’s output.Now check the following select statement, what will the output.

Example:

SQL> select \* from order\_master where order\_no = (select order\_no from order\_detail);

It will return an error, ‘single-row subquery returns more than one row’.

When subquery returns more than one row we have to use operators like any, all, in or not

in.‘=any’ is equivalent to ‘in’ operator and ‘!=all’ is equivalent to not in.

Example:

SQL>Select \* from order\_master where order\_no = any(select order\_no from order\_detail);

SQL> select \* from order\_master where order\_no in(select order\_no from order\_detail);

Both select statements are equal. After using ‘any’ operator it display records from

order\_master where any order\_no is equal to order\_no from order\_detail.

Example:



DBMS RECORD

Page No: 71

SQL> select \* from order\_detail where qty\_ord =all(select qty\_hand from itemfile where

itemrate =250);

In above example the subquery will display area that holds the itemrate that is less than 250.

The main query will display details about orders only if qty\_ord is greater than all the values

return by the subquery.

Exp No: 10

Date: \_ \_/\_ \_/ \_ \_

AIM : Implement Views:

Views

After a table is created and populated with data, it may become necessary to prevent all

users from accessing all columns of a table, for data security reasons. This would mean

creating several tables having the appropriate number of columns and assigning specific

users to each table, as required. This will answer data security requirements very well but

will give rise to a great deal of redundant data being resident in tables, in the database.To

reduce redundant data to the minimum possible, Oracle allows the creation of an object

called a View.

A View is mapped, to a SELECT sentence. The table on which the view is

based is described in theFROM clause of the SELECT statement. The SELECT clause

consists of a sub-set of the columns of the table. Thus a View, which is mapped to a table,

will in effect have a sub-set of the actual columns of the table from which it is built. This

technique offers a simple, effective way of hiding columns of a table.Some View’s are used

only for looking at table data. Other View’s can be used to Insert, Update and Delete table

data as well as View data. If a View is used to only look at table data and nothing else, the

View is called a Read-Only view. A View that is used to Look at table data as well as Insert,

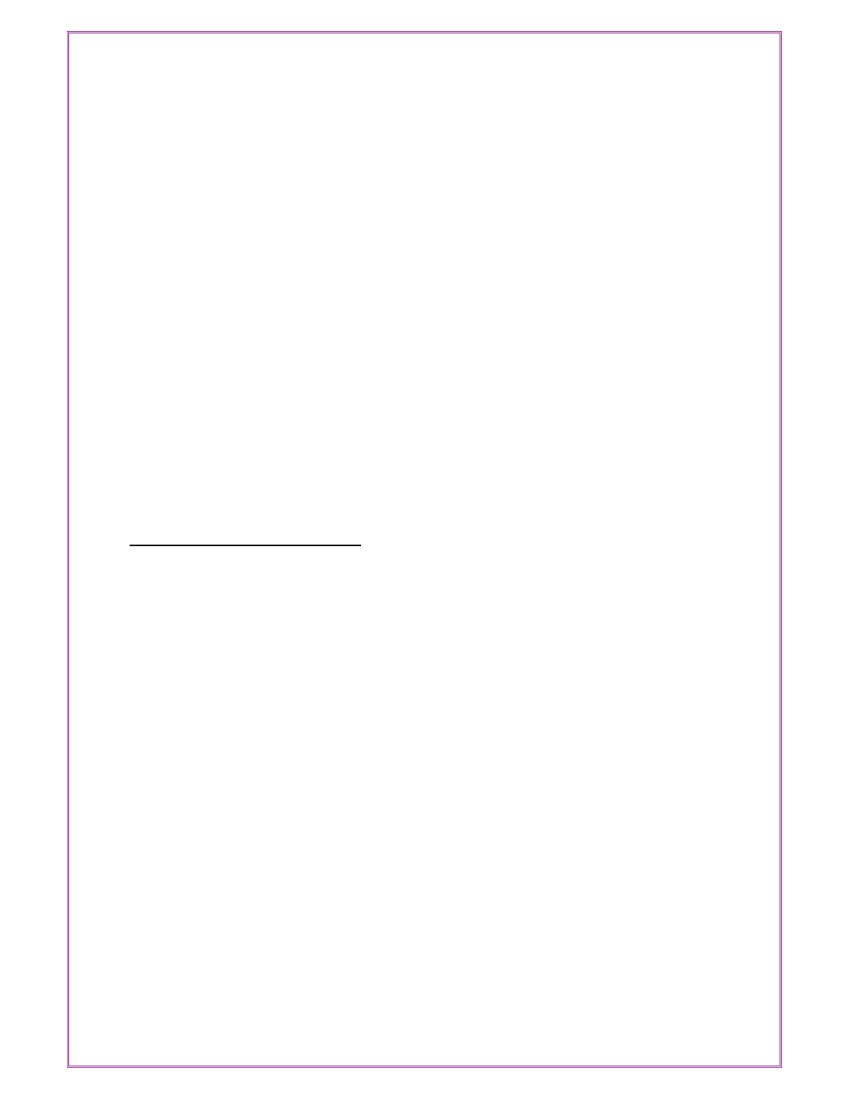
Update and

Delete table data is called an Updateable View.

The reasons why views are created are:

 When Data security is required

 When Data redundancy is to be kept to the minimum while maintaining data securiry



DBMS RECORD

Page No: 72

Lets spend some time in learning how a View is Created

 Used for only viewing and/or manipulating table data

i.e. a read-only or updateable view

 Destroyed

Syntax:

Create View <View\_Name> As Select statement;

Example:

SQL>Create View EmpView As Select \* from Employee;

View created.

Once a view has been created, it can be queried exactly like a base table.

Syntax:

Select columnname,columnname from <View\_Name>;

Example:

SQL>Select Empno,Ename,Salary from EmpView where Deptno in(10,30);

EMPNO

1122

1125

ENAME

Allen

Tanmay

SALARY

10000

10000

2 rows selected.

Updateable Views:

Views can also be used for data manipulation (i.e. the user can perform the Insert, Update

and Delete operations). Views on which data manipulation can be done are called

Updateable Views. When you give an updateable view name in the Update, Insert or Delete

SQL statement, modifications to data will be passed to the underlying table.

For a view to be updateable, it should meet the following criteria:

Views defined from Single Table:

If the user wants to INSERT records with the help of a view, then the PRIMARY KEY

column/s and all the NOT NULL columns must be included in the view.The user can

UPDATE, DELETE records with the help of a view even if the PRIMARY KEY column

and NOT NULL column/s are excluded from the view definition.

Example:



DBMS RECORD

Page No: 73

Table Name: Employee

Column Name Data Type Size Attributes

Empno Number 3 Primary Key

Ename Varchar2 30 Not Null

Salary Number 8,2 Not Null

Deptno Number 3

Syntax for creating an Updateable View:

Create View Emp\_vw As

Select Empno,Ename,Deptno from Employee;

View created.

When an INSERT operation is performed using the view:

SQL>Insert into Emp\_vw values(1126,’Brijesh’,20);

1 row created.

\_ When an MODIFY operation is performed using the view:

SQL>Update Emp\_vw set Deptno=30 where Empno=1125;

1 row updated.

\_ When an DELETE operation is performed using the view:

SQL>Delete from Emp\_vw where Empno=1122;

1 row deleted.

A view can be created from more than one table. For the purpose of creating the View

these tables will be linked by a join condition specified in the where clause of the View’s

definition. The behavior of the View will vary for Insert, Update, Delete and Select table

operations depending upon the following:

 Whether the tables were created using a Referencing clause

 Whether the tables were created without any Referencing clause and are actually

standalone tables not related in any way.

View defined from Multiple tables (Which have no Referencing clause):

If a view is created from multiple tables, which were not created using a ‘Referencing

clause’ (i.e. No logical linkage exists between the tables), then though the PRIMARY KEY

column/s as well as the NOT NULL columns are included in the View definition the view’s



DBMS RECORD

Page No: 74

behavior will be as follows:

The INSERT, UPDATE or DELETE operation is not allowed. If attempted Oracle displays

the followingerror message:

For insert/modify:

ORA-01779: cannot modify a column, which maps to a non-preserved table.

For delete:

ORA-01752: cannot delete from view without exactly one key-preservedtable.

View defined from Multiple tables (Which have been created with a Referencing

clause):

If a view is created from multiple tables, which were created using a ‘Referencing clause’

(i.e. a logicallinkage exists between the tables), then though the PRIMARY KEY Column/s

as well as the NOT NULL columns are included in the View definition the view’s behavior

will be as follows:

 An INSERT operation is not allowed.

 The DELETE or MODIFY operations do not affect the Master table.

 The view can be used to MODIFY the columns of the detail table included in the

view.

 If a DELETE operation is executed on the view, the corresponding records from the

detail table will be deleted.

Syntax for creating a Master/Detail View (Join View):

SQL>Create View EmpDept\_Vw As

Select a.Empno,a.Ename,a.Salary,a.Deptno,b.Dname From Employee a,DeptDet b

Where a.Deptno=b.Deptno;

View created.

When an INSERT operation is performed using the view

SQL>Insert into EmpDept\_Vw values(1127,’AbhayShah’,10000,10,’Technical’);

ORA-01776:cannot modify more than one base table through a join view When an

MODIFY operation is performed using the view

SQL>Update EmpDept\_Vw set salary=4300 where Empno=1125;



DBMS RECORD

Page No: 75

1 row updated.

When an DELETE operation is performed using the view

SQL>Delete From EmpDept\_Vw where Empno=1123;

1 row deleted.

Common restrictions on updateable views:

The following condition holds true irrespective of the view being created from a single table

or multiple tables.For the view to be updateable the view definition must not include:

 Aggregate functions.

 DISTINCT, GROUP BY or HAVING clause.

 Sub-queries.

 Constants, Strings or Value Expressions like Salary \* 2.25.

 UNION, INTERSECT or MINUS clause.

 If a view is defined from another view, the second view should be updateable.

If the user tries to perform any of INSERT, UPDATE, DELETE operation, on a view which

is created from a non-updateable view Oracle returns the following error message:

For insert/modify/delete:

ORA-01776:data manipulation operation not legal on this view

To Create Read-only View:

In this view, you cannot manipulate the records. Because of this view is created with read

only.

SQL>Create View EmpRO As select \* from Employee with Read Only;

View created.

To Create View With Check option:

In this view, you cannot change value of deptno column. Because of this view is created

with checkoption.

SQL>Create View EmpCk As Select \* from Employee Where Deptno=10 WithCheck

Option;

View created.

Destroying a view:



DBMS RECORD

Page No: 76

The DROP VIEW command is used to remove a view from the database.

Syntax:

Drop View <View\_Name>;

Example:

Remove the view Emp\_Vw from the database.

SQL>Drop View Emp\_Vw;

View dropped.

SQL> create view v1 as select \* from Passenger full natural join Reservation;

View created.

a) INSERT

SQL> insert into male\_pass values(&PNR\_NO,&age);

Enter value for pnr\_no: 12

Enter value for age: 22

old 1: insert into male\_pass values(&PNR\_NO,&age)

new 1: insert into male\_pass values(12,22)

1 row created.

b) DROP VIEW

SQL> drop view male\_pass;

View dropped.



DBMS RECORD

Page No: 77

Exp No: 11

AIM : Implement Indexes:

Indexes

Date: \_ \_/\_ \_/ \_ \_

When the user fires a SELECT statement to search for a particular record, the Oracle

engine must first locate the table on the hard disk. The Oracle engine reads system

information and locates the starting location of a table’s records on the current storage

media. The Oracle engine then performs a sequential search to locate records that match

user-defined criteria.

For example, to locate all the orders placed by client ‘C00001’ held in the sales\_order

table the Oracle engine must first locate the sales\_order table and then perform a sequential

search on the client\_no column seeking a value equal too ‘C00001’. The records in the

sales\_order table are stored in the order in which they are keyed in and thus to get all orders

where client\_no is equal to ‘C00001’ the Oracle engine must search the entire table

column.Indexing a table is an ‘access strategy’, that is, a way to sort and search records in

the table. Indexes are essential to improve the speed with which the record/s can be located

and retrieved from a table.

An index is an ordered list of the contents of a column, (or a group of columns) of a

table.

Indexing involves forming a two dimensional matrix completely independent of the

table on which the index is being created.A column, which will hold sorted data, extracted

from the table on which the index is being created.An address field that identifies the

location of the record in the Oracle database. This address field is called Rowid.

When data is inserted in the table, the Oracle engine inserts the data value in the index. For



DBMS RECORD

Page No: 78

every data value held in the index the Oracle engine inserts a unique rowid value. This is

done for every data value inserted into the index, without exception. This rowid indicates

exactly where the record is stored in the table.Hence once the appropriate index data values

have been located, the Oracle engine locates an associated record in the table using the

rowid found in the table.The records in the index are sorted in the ascending order of the

index column/s.

If the SELECT statement has a where clause for the table column that is indexed, the Oracle

engine will scan the index sequentially looking for a match of the search criteria rather than

the table column itself. The sequential search is done using an ASCII compare routine to

scan the columns of an index. Since the data is sorted on the indexed column/s, the

sequential search ends as soon as the Oracle engine reads an index data value that does not

meet the search criteria.

22245001111111

Example:

Select order\_no,order\_date,client\_no From Sales\_order Where client\_no=’C00001’;

When the above select statement is executed, since an index is created on client\_no column,

the Oracle engine will scan the index to search for a specific data value i.e. client\_no equal

to ‘C00001’. The Oracle engine will then perform a sequential search to retrieve records that

match the search criteria i.e. client\_no=‘C00001’. When ‘C00002’ is read, the Oracle engine

stops further retrieval from the index. For the three records retrieved, the Oracle engine

locates the address of the table records from the address of the table records from the

ROWID field and retrieves records stored at the specified address.

Client\_no ROWID

C00001 00000240.0000.00004

C00001 00000240.0002.00004

C00001 00000241.0002.00004

The Rowid in the current example indicates that the record with client\_no ‘C00001’ is

located in data file0004. Two records are stored in block 00000240 with record number 0000

and 0002. The third record is stored in block 00000241 with record number 0002.Thus, data

retrieval from a table by using an index is faster then data retrieval from the table



DBMS RECORD

Page No: 79

whereindexes are not defined.

Duplicate/Unique Index:

Oracle allows the creation of two types of indexes. These are:

Indexes that allow duplicate values for the indexed columns i.e. Duplicate Index

Indexes that deny duplicate values for the indexed columns i.e. Unique Index

Creation of Index:

An index can be created on one or more columns. Based on the number of columns included

in the index, an index can be:

simple Index

Composite Index

Creating Simple Index:

An index created on a single column of a table it is called Simple Index. The syntax for

creating simple index that allows duplicate values is:

Syntax:

Create Index <Index Name> On <Table Name>(ColumnName);

Example:

Create a simple index on client\_no column of the Client\_master table

SQL>Create Index idx\_client\_no On Client\_master (Client\_no) ;

Index Created.

Creating Composite Index:

An index created on a more than one column it is called Composite Index. The syntax for

creating a composite index that allows duplicate values is:

Syntax:

Create Index <Index Name> On <Table Name>(ColumnName, ColumnName);

Example:

Create a composite index on the sales\_order tables on column order\_no and product\_no.

SQL>Create Index idx\_sales\_order On Sales\_order (Order\_no,product\_no) ;

Index Created.

Note: The indexes in the above examples do not enforce uniqueness i.e. the columns

included in the index can have duplicate values. To create unique index, the keyword



DBMS RECORD

Page No: 80

UNIQUE should be included in the Create

Index command.

Creation of Unique Index:

A unique index can also be created on one or more columns. If an index is created on a

single column, it is called Simple Unique Index. The syntax for creating a simple unique

index is:

Syntax:

Create Unique Index <Index Name> On <Table Name> (Column Name);

If an index is created on more than one column it is called Composite Unique Index. The

syntax for creating a composite unique index is:

Syntax:

Create Unique Index <Index Name> On <Table Name> (ColumnName,ColumnName);

Example:

Create a unique index on client\_no column of the client\_master table.

SQL>Create Unique Index idx\_client\_no On Client\_master (Client\_no);

Index Created.

Note: When the user defines a primary key or a unique key constraint, the Oracle

engine automatically

creates a unique index on the primary key or unique key column/s.

Dropping Indexes:

Indexes associated with the tables can be removed by using the DROP INDEX command.

Syntax:

Drop Index <Index Name>;

Example:

Remove index idx\_client\_no created for the table client\_master.

SQL>Drop Index idx\_client\_no;

Index dropped.

Note: When a table, which has associated indexes (unique or non-unique) is dropped,



DBMS RECORD

Page No: 81

the Oracle engine automatically drops all the associated indexes as well.

Exp No: 12

Date: \_ \_/\_ \_/ \_ \_

Aim : Implementing Operations on relations using PL / SQL.

Basics of PL/SQL

About PL/SQL

PL/SQL is a Oracle’s Procedural Language (PL) extension to Structured Query Language

(SQL), that Oracle developed as an extension to standard SQL in order to provide a way to

execute proce- dural logic on the database. PL/SQL provides a mechanism for developers to

add a procedural component at the server level. It has been enhanced to the point where

developers now have access to all the features of a full-featured procedural language at the

server level. It also forms the basis for programming in Oracle’s continually evolving set of

client/server development tools, most nota- bly Developer/2000.

Use of PL/SQL

You can use PL/SQL to perform processing on the server rather than the client. You can use

PL/SQL to encapsulate business rules and other complicated logic. It provides for

modularity and ab- straction. Finally, it provides you with a level of platform independence.

Oracle is implemented on many platforms, but PL/SQL is the same on all of them. It makes

no difference whether you are running Personal Oracle on a laptop or Oracle 8 Server on

Windows NT In a nutshell, With PL/SQL you have the power to make your applications

more robust, more effi- cient, and most secure

Advantages of PL/SQL

PL/SQL is a block-structured language offered by Oracle to facilitate the use of the Oracle

RDBMS.It has the following properties and features that can be used to aid in application

development:PL/SQL is completely portable, high performance transaction processing,



DBMS RECORD

Page No: 82

which offers the follow- ing advantages.

• Without PL/SQL, Oracle must process SQL statements one at a time. With PL/SQL, an

entire block of statements can process in a single command line statement. This reduces the

time taken to communicate the application and the Oracle server. PL/SQL blocks are

portable to any operating system or platform.

• PL/SQL allows us to use of all SQL data manipulation commands, transactions control

commands, SQL functions (except group functions), operators and pseudo columns.

• Any DDL Statements are not allowed in PL/SQL Block.

• PL/SQL supports all the SQL data types and as well as it has its own.

These features make PL/SQL a powerful SQL processing language. Using PL/SQL has

several major advantages over using standard SQL statements (in addition to allowing the

use of stored procedures and functions). Among these are ease of use, portability, and higher

performance.

PL/SQL Block

PL/SQL code is grouped into structures called blocks. A Block contains three sections, as

de- scribed below.

declare

<declaration of variables, constants, function, procedure,

cursor etc.>;

begin

<executable statement(s)>;

exception

<exception handling>;

end;

/

Within a PL/SQL block, the first section is the Declaration section. Using Declaration

section, you can define variables and cursors that the block will use. The declaration section

starts with the keyword declare and ends when the Executable commands section starts. (as

indicated by the keyword begin). The executable commands section is followed by

exception handling section; the exception keyword signals the start of the exception



DBMS RECORD

Page No: 83

handling section. The PL/SQL block is termi- nated by the end keyword.

In PL/SQL Block only the executable section is required, the declarative and exception

handling sections are optional.

A Simple Block

Example

Begin

Insert into emp(empno,ename) values(100,’Shruti’);

Insert into emp(empno,ename) values(101,’Yesha’);

End;

/

/ forward slash executes the PL/SQL block.

When the PL/SQL block is executed it will inserted two rows in emp table.

dbms\_output.put\_line()

dbms\_output.put\_line() is used to displays the value of variable or any message on the

next line of the console. You might wonder, that, though the block is successfully completed

with dbms\_output.put\_line() it is not showing any output. For that we have to check out that

the serveroutput is on or not? If not, write the following line on the SQL prompt, and

execute the block again, now you get the output on the console.

SQL>Set Serveroutput On

R It is not necessary to write the above statement for the execution of each and every

block. This statement is written only once per session

You can omit line word from dbms\_output.put\_line().

dbms\_output.put()is used to displays the value of variable or any message on the same line

on the console.

Example

Begin

dbms\_output.put\_line(‘Starting of PL/SQL’);

dbms\_output.put\_line(‘Welcome to’)

dbms\_output.put(‘AZURE’);

End;



DBMS RECORD

Page No: 84

/

When the above PL/SQL block is executed, you will receive the following response from

Oracle.

SQL> PL/SQL Procedure successfully completed.

Starting of PL/SQL

Welcome to AZURE

R before executing above block make sure that Serveroutput must be on.

Datatypes

PL/SQL datatypes can be classified into two types.

• Scalar datatypes

• Composite datatypes

All SQL data types like number, char, varchar2, raw, long raw, lob, date and ANSI Standard

data type such as boolean, binary\_integer and number are categorized as a scalar datatype.

Boolean

Boolean data types can be used to store the values TRUE, FALSE or NULL.

Binary\_Integer

Binary\_integer is used to store signed integers. The range of binary\_integer value is ｱ 231

i.e. the range of binary\_integer is –2147483647 to 21474483647

Number

It is same as SQL number data types. In addition to this it includes ANSI standard types

which includes following datatypes

 Dec / Decimal

 Int / Integer

 Real

Identifiers

Identifiers are names, which the programmer used within the program. They can refer to

one of the following.

• Variable

• Constant



DBMS RECORD

Page No: 85

Some data are predetermined before a block is used, and its value remains unchanged during

the execution of block, these are constant. Other data may change or be assigned values, as

the block executed is known as variable.

Variables

Communication with the database takes place via variables in the PL/SQL block. Variables

are memory locations, which can store data values. As the program runs, the contents of

variables can and do change. Information from the database can be assigned to a variable, or

the contents of a variable can be inserted into the database. These variables are declared in

the declarative sections of the block. Every variable has a specific type as well, which

describes what kind of information can be stored in it.

Variables are declared in the declarative section of the block.

The general syntax for declaring a variable is

variable\_name type [:= value/expression];

where variable\_name is the name of the identifier, type is the any valid data type and value

is the value of the variable.

For example, the following are the legal variable declarations:

salary number(5);

Declares variable called salary to store maximum 5-digit number.

Assigning values to a variable

We can assign a value to the variable using followings.

• Assignment operator (:=)

• Default keyword

• By fetching method

Declare variable name to store maximum 10 characters and assign AZURE value through

assign- ment operator

name varchar2(10):=‘AZURE’;

Declared variable called salary to store maximum 5-digit number and assign value 3999 to it

using default keyword.

Salary number(5) default 3999;

Store an employee name in nm variable whose employee number is 2 using fetching method.



DBMS RECORD

Page No: 86

Select ename into nm from emp where empno = 2;

Constants

We can declare variable as a constant and use them in executable part. One cannot change

the value of constant throughout the program.

variable\_name CONSTANT type := value/expression;

where variable\_name is the name of the identifier, type is the any valid data type and value

is the value of the variable.

For example,

pi constant number(9,7):=3.1415926;

Here value of variable pi can’t be change throughout the program execution.

Example

Insert value into dept table using variables.

Declare

v\_deptno number := 10;

v\_dname varchar2(10) := ‘sales’;

v\_loc varchar2(10) := ‘bombay’;

Begin

insert into dept values(v\_deptno, v\_dname, v\_loc);

End;

/

Example

To get the area of the circle provided the radius is given.

Declare

pi constant NUMBER(9,7) := 3.1415926;

radius INTEGER(5);

area NUMBER(13,2);

Begin

radius := 3;

area := pi \* power(radius,2);

dbms\_output.put\_line(‘Area of the circle is ’,area);



DBMS RECORD

Page No: 87

End;

/

In the above example, Declaration section having constant identifier (the value of which can

not be reinitialized) named pi, which stores a constant value 3.1415926 and another two

variables of type Integer and Number respectively. In Executable Section, radius is initialize

with value 3 and then area is initialize with the calculated result according to the

mathematical formula for finding area of the Circle.

Example

To get the name and salary of specified employee.

Declare

nm varchar2(20);

sal number(9,2);

Begin

Select ename,salary into nm, sal from emp where empno=2;

dbms\_output.put\_line(‘Name : ’ || nm);

dbms\_output.put\_line(‘Salary : ’ || sal);

End;

/

In the above example we have use concatenation operator (||). Concatenation operator

attaches two or more strings together.

‘hello ’||‘world’ gives you ‘hello world’ output.

Comments in PL/SQL Block

As we know from very beginning of programming, Comment improves readability and

makes your program more understandable. The PL/SQL engine ignores them at the

compilation and execution time.

There are two kinds of comments:

Single-line comments

It is starts with two dashes and continues until the end of the line.

Example

Begin — — declaration section is ignored



DBMS RECORD

Page No: 88

Insert into emp(ename, empno) values(‘Tejas’,1234);

End;

/

Multiline comments

It is starts with /\* delimiter and end with the \*/ delimiter. This is the same style of comments

used in the C language.

Example

Begin /\* declaration section and exception

section is also neglected\*/

Insert into emp(ename, empno) values(‘Tanmay’,1234);

End;

/

R One can utilize multiline comments in Single line.

Example

To get the name and salary of specified employee.

Declare

nm varchar2(20);

sal number(9,2);

Begin

/\* Get employee name and Salary \*/

Select ename,salary into nm, sal from emp where empno=2;

/\* Display Employee name and Salry \*/

dbms\_output.put\_line(‘Name : ’ || nm);

dbms\_output.put\_line(‘Salary : ’ || sal);

End;

/

%type Attribute

In many cases, variable will be used to manipulate data stored in a database table. In such

case, the variable should have the same type as the table column.



DBMS RECORD

Page No: 89

For example, the ename column of emp table has type varchar2(20). Based on this, we have

to declare a variable as follows as discussed in above program.

nm varchar2(20);

This is fine, but what happens if the definition of ename column is changed? Say the table is

altered and ename now has type varchar2(25). Any PL/SQL code that uses this column

would have to be changed, as shown here:

nm varchar2(25);

If you have a large amount of PL/SQL code based on ename column of emp table, this can

be a time consuming and error prone process. Rather than hardcode the type of a variable in

this way, you can use the %type attribute.

This attribute is used to declare a variable’s data type as being equivalent to the specified

column’s datatype. So, you need not know the exact data type and size of a database

column.

For example:

nm emp.ename%type;

By using %type, nm will have whatever type and size the ename column of the emp table

has. If the database definition changes, the datatype in PL/SQL block is also changed

according to changes made in database.

Example

To get the name and salary of specified employee using %type attribute.

Declare

nm emp.ename%type;

sal emp.salary%type;

Begin

Select ename,salary into nm, sal from emp where empno=2;

dbms\_output.put\_line(‘Name : ’ || nm);

dbms\_output.put\_line(‘Salary : ’ || sal);

End;

/

%rowtype Attribute

%type attribute is used to declare a variable’s data type as being equivalent to the



DBMS RECORD

Page No: 90

specified column’s datatype , while %rowtype attribute is used to declare composite

variable that is equivalent to a row in the specified table. The composite variable is consist

of the column names and datatypes in the referenced table i.e. in the declaration of

%rowtype attribute with variable, variable inherits column and its datatype information for

all the columns of a table.

For example:

erec emp%rowtype;

By using %rowtype, erec will have all the columns with it of emp table.

To, access value of the particular column is done as follows.

erec.ename;

Where, erec is a composite variable, while ename is a column of emp table.

Example

To get the name and salary of specified employee using %type attribute.

Declare

emprec emp%rowtype;

Begin

Select \* into emprec from emp where empno=10;

dbms\_output.put\_line(‘Name : ’ || emprec.ename);

dbms\_output.put\_line(‘Salary : ’ || emprec.salary);

End;

/

%rowtype attribute specially used with Cursor, which will be discussed later.

PL/SQL Control Structures

Control structure is the most important in PL/SQL to change the logical flow of statements

within PL/SQL Block. PL/SQL has a variety of control structures that allow you to control

the behavior of the block as it runs. These structures include conditional statements and

iterative controls i.e. PL/SQL supports basic programming control structures.

• Sequence

• Selection / Condition

• Iteration



DBMS RECORD

Page No: 91

These structures combined with variables, gives PL/SQL its power and flexibility.

R dbms\_standard package provides a language facility to interact with Oracle.

Selection Control

Within PL/SQL block, Selection control is used when the execution of a particular set of

statement is based on a specific condition. Sequence of statements can be executed based on

some condition using the if statement. There are various form of if statement.

If-then form

The simple form of the if statement is the if-then statement as follows.

IF <boolean\_expression> THEN

statements;

END IF;

Where boolean\_expression is any expression that evaluates to a Boolean value.

Example

Accept Number from a User and display Hello message if the entered number is Positive.

Declare

num number;

Begin

num := &num;

if num > 0 then

dbms\_output.put\_line(‘Hello’);

end if;

end;

/

Example

Display Salary of a specified employee increasing by 500 if its salary is more than 3000.

Declare

sal number(9,2);

num emp.empno%type;

Begin

num := &num;



DBMS RECORD

Page No: 92

Select salary into sal from emp where empno=num;

If sal > 3000 then

sal := sal + 500;

end if;

dbms\_output.put\_line(‘Salary : ’ || sal);

End;

/

Above block will display salary of specific employee (as per entered employee number) by

increas- ing 500 if its salary is greater than 3000 otherwise it will display salary as it is.

If-then-else form

So far we have discussed the simplest form of the if statement, which gives us a choice of

executing a statement of block or skipping them. If-then-else form allows us to execute

either or blocks depend on condition using if-then-else form as follows.

IF <boolean\_expression> THEN

True block statements;

ELSE

False block statements;

END IF;

True block statements are executed only if the condition is satisfied otherwise the else

portion will be executed.

Example

Accept number from a user and find out whether it is Odd or Even.

Declare

num number;

Begin

num := &num;

if mod(num,2) = 0 then

dbms\_output.put\_line(no,’is even’);

else

dbms\_output.put\_line(no,’is Odd’);



DBMS RECORD

Page No: 93

end if;

End;

/

Example

Accept employee number from a user and increase its salary depends on the current salary as

follows.

Salary Increment

>= 5000 12.5%;

<5000 11%

Declare

sal number(9,2);

num emp.empno%type;

pf number(9,2);

Begin

num := &num;

Select salary into sal from emp where empno=num;

If sal >= 5000 then

update emp set salary = salary+(salary\*0.125)where empno=num;

else

update emp set salary = salary + (salary\*0.11) where empno=num;

end if;

End;

/

If-then-elsif form

This form is used to select one of multiple alternatives.

IF <boolean\_expression1> THEN

statements;

ELSIf <boolean\_expression2> THEN

statements;



DBMS RECORD

Page No: 94

ELSIf <boolean\_expression2> THEN

statements;

——————————

——————————

END IF;

Example

Declare

sal emp.sal%type;

eno emp.empno%type;

Begin

Eno := &eno;

Select salary into sal from emp where empno=eno;

if sal > 10000 then

dbms\_output.put\_line(‘Salary is more than 10000’);

elsif sal >= 7000 then

dbms\_output.put\_line(‘salary is between 7000 to 10000’);

else

dbms\_output.put\_line(‘Salary is less than 7000’);

end if;

End;

/

Iterative Control / Loops

PL/SQL provides a facility for executing statements repeatedly, via loops. In PL/SQL we

have three loops as follows to execute statements repetitively.

• Simple loop

• While loop

• For loop

Simple loop

The most basic kind of loops, simple loop have this Syntax:

LOOP



DBMS RECORD

Page No: 95

Sequence\_of\_statements;

END LOOP;

Sequence\_of\_statements will be executed infinitely, since this loop has no stopping

condition. We can exit this loop via EXIT statement. General form of exit statement is as

follows.

EXIT [WHEN condition];

Example

Declare

I number(2):=0;

Begin

Loop

dbms\_output.put\_line(I);

I:=I+1;

Exit when (I>10);

End loop;

End;

/

WHILE Loop

WHILE <condition>

LOOP

Sequence\_of\_statements;

END LOOP;

Before each iteration of the loop, condition is evaluated. If it evaluates to TRUE,

sequence\_of\_statements is executed. If condition evaluates to FALSE or NULL, the loop is

fin- ished and control resumes after the END LOOP statement.

The only difference between simple loop and while loop is simple execute first and then it

will check condition, so simple loop execute at least once and in while loop first it will

check condition and then execute.

Example



DBMS RECORD

Page No: 96

Declare

I number(2):=0;

Begin

While I > 50

loop

dbms\_output.put\_line(i);

I:=I+5;

End Loop;

End;

/

FOR Loop

The number of iterations for simple loops and while loops is not known in advance; it

depends on the condition. FOR loops, on the other hand, have a defined number of

iterations.

FOR loop\_counter IN [REVERSE] LowerBound..UpperBound

LOOP

Sequence\_of\_statements;

End LOOP;

Where loop\_counter is the implicitly declared index variable, lowerbound and upperbound

specify the number of iterations, and sequence\_of\_Statements is the contents of the loop.

Example

Declare

no number := 5;

Begin

For I in 1..10 loop

dbms\_output.put\_line(no||‘ \* ‘||I||‘ = ‘||no\*I);

End loop;

End;

/



DBMS RECORD

Page No: 97

Reverse keyword

If the REVERSE keyword is present in the FOR loop, then the loop index will iterate from

the high value to the low value. Notice that the syntax is the same; the low value is still

referenced first.

Example

Begin

For I in REVERSE 1..5 LOOP

dbms\_output.put\_line(I);

End LOOP;

End;

/

Goto Statment

The goto statement allows us to branch to a label unconditionally. The label, which is

enclosed within double angular brackets, must precede an executable SQL or a PL/SQL

block. When ex- ecuted, the goto statement transfers control to the labeled statement or a

block.

Example

Declare

no number:=1;

Begin

While no<=10 loop

dbms\_output.put\_line(no);

no := no+1;

If no = 5 then

goto lbl;

End if;

End loop;

<<lbl>>

dbms\_output.put\_line(‘Number Printing from lable ‘||no);

End;



DBMS RECORD

Page No: 98

/

In above example, when no is equal to 5 control transfer to label lbl and execute whatever

men- tioned after label and stop execution of for loop.

Exception Handling

Exception

One of the features of PL/SQL is the exception handling mechanism. By using exceptions

and exception handlers, you can make your PL/SQL programs robust and able to deal with

both unex- pected and expected errors during execution. What kind of errors can occur in a

PL/SQL program? Errors can be classified into run-time error and compile-time error.

Exceptions are designed for run-time error handling. Errors that occur during the

compilation time are detected by PL/SQL engine and reported back to the user. The program

cannot handle this, since this had yet to run. Exceptions and exception handlers are how the

program responds to run- time errors. When an error occurs, an exception is raised. When

this happens, control is passed to the exception handler, which is the separate section of the

PL/SQL Block. This separates the error handling from the rest of the block, which makes the

logic of the program easier to understand. In PL/SQL, the user can anticipate and trap for

certain runtime errors. there are two types of exceptions:

 Predefined Exceptions

 User-defined Exceptions

R Exceptions can be internally defined by Oracle or by the user.

Pre-defined Exceptions

The Pre-defined exception is raised implicitly (automatically) when PL/SQL block or any of

its statement violets Oracle rule. Those errors, which are frequently occur, are assign as a

predefined exception by Oracle.

Then General form Exception Handling is :

Declare

...

Begin

...

Exception



DBMS RECORD

Page No: 99

When Exception\_1 then

<Statement(s)>;

When Exception\_2 then

<Statement(s)>;

...

End;

/

In this syntax, Exception\_1 and Exception\_2 are the names of the predefined exceptions.

State- ments are the valid code that will be executed if the exception name is satisfied.

The Oracle server defines several errors with standard names. Although every Oracle error

has a number, the errors must be referenced by name. PL/SQL has predefined some common

Oracle errors and exceptions. Some of these predefined exception names are Sr. Exception

Description

1 No\_Data\_Found SELECT returned no rows

2 Too\_many\_rows SELECT into statement returned more than one row.

3 Invalid\_cursor This exception is raised when we violet cursor operation.

For example, when we try to close a cursor, which is not opened.

4 Value\_error Arithmetic, Conversion, Truncation or Constraint Error occurred. i.e.

Attempt to insert a value with larger precision.

5 Invalid\_Number Conversion of a character to a number is failed.

6 Zero\_divide Attempted to divide by zero.

7 Dup\_val\_on\_Index Attempted to insert a duplicate value into a column that has a unique

index.

8 Cursor\_already\_open Attempted to open a cursor that was previously opened.

9 Not\_logged\_on A database call was made without being logged into Oracle.

10 Transaction\_backed\_out Usually raised when a remote portion of a transaction is rolled

back

11 Login\_denied Login to Oracle failed because of invalid username and password.

12 Program\_error Raised if PL/SQL encounters an internal problem.

13 Storage\_error Raised if PL/SQL runs out of memory or if memory is corrupted.



DBMS RECORD

Page No: 100

14 Timeout\_on\_resource Timeout occurred while Oracle was waiting for a resource.

15 Others This is a catchall. If the error was not trapped in the previous

exception traps, this statement will trap the error.

Oracle declares predefined exceptions globally in the package standard. Therefore, you do

not need to declare them yourself.

Example

Write a PL/SQL Block to accept employee name from a user if it is exist display its salary

otherwise display appropriate message using exception handling.

Declare

erec emp%rowtype;

nm emp.ename%type;

Begin

nm:=&nm;

SELECT \* into erec from emp where ename=nm;

dbms\_output.put\_line(‘Employee Salary : ‘ || erec.salary);

Exception

When No\_Data\_Found then

dbms\_output.put\_line(‘Entred name is not found’);

When Others then

Null;

End;

/

Example

Write a PL/SQL Block to display the salary of that employee whose age is 45 year otherwise

dis- play appropriate message using exception handling.

Declare

erec emp%rowtype;

yr number;

Begin

yr := &yr;



DBMS RECORD

Page No: 101

SELECT \*into erec from emp Where round((SYSDATE-BDATE)/365),0)= 45;

dbms\_output.put\_line(‘Employee Salary : ‘ || erec.salary);

Exception

When No\_Data\_Found then

dbms\_output.put\_line(‘No Employee with 45 years age’);

When Too\_many\_rows then

dbms\_output.put\_line(‘More than one Employee with 45 years age’);

When Others then Null;

End;

/

Example

Write a PL/SQL Block to insert add one row in employee table with employee number and

name.Display appropriate message using exception handling on duplication entry of

employee number.

Declare

eno emp.empno%type;

nm emp.ename%type;

Begin

eno := &eno;

nm := ‘&nm’;

insert into emp(empno,ename) values(eno,nm);

Exception

When Dup\_val\_on\_index then

dbms\_output.put\_line(‘Employee Number already Exist’);

When Others then

Null;

End;

/

As you saw in the earlier examples of exception-handling blocks, the other exception was

used as a catchall exception handler. Others is normally used when the exact nature of the



DBMS RECORD

Page No: 102

exception isn’t important, when the exception is unnamed, or even when it’s unpredictable

User-Defined Exception

Unlike internal exceptions, user-defined exceptions should be explicitly specified. The user-

defined exception must be declared in the declaration part of the PL/SQL Block and it can

explicitly raised with the RAISE Statement. Declaration of user-defined cursor declares a

name for user\_defined error that the PL/SQL code block recognizes. The raise exceptions

procedure should only be used when Oracle does not raise its own exception or when

processing is undesirable or impossible to complete

Steps for trapping a user-defined error include the following:

1) Declare the name for the user-defined exception within the declaration section of the

block.

2) Raise the exception explicitly within the executable portion of the block using the RAISE

State- ment.

3) Reference the declared exception with an error-handling routine.

Example

Declare

Invalid\_Pay Exception;

Pay Number := &Pay;

Begin

If Pay Not Between 2000 and 5000 Then

RAISE Invalid\_Pay;

End If;

Exception

When Invalid\_Pay Then

dbms\_Output.Put\_Line(‘Salary should be between 2000 and 5000’);

End;

/

Example

Accept employee number and salary from a user and store it into the table if salary is greater

than zero otherwise display appropriate message using user-defined exception.



DBMS RECORD

Page No: 103

Declare

sal emp.salary%type;

eno emp.empno%type;

sal\_error exception;

Begin

eno := &eno;

sal := &sal;

if sal=0 then

raise sal\_error;

else

update emp set salary = sal where empno=eno;

end if;

Exception

when sal\_error then

dbms\_output.put\_line(‘Salary must be >0’);

End; /



DBMS RECORD

Page No: 104

Exp No: 13

Date: \_ \_/\_ \_/ \_ \_

Aim : Writing triggers

Database Triggers:

Trigger defines an action the database should take when some database-related

event occurs. Trig- gers may be used to supplement declarative referential integrity, to

enforce complex business rules, or to audit changes to data. The code within a trigger, called

a trigger body, is made up of PL/SQL blocks. It’s like a stored procedure that is fired when

an insert, update or delete command is issued against associated table.

The execution of triggers is transparent to the user. Triggers are executed by the database

when specific types of data manipulation commands are performed on specific tables. Such

commands may include insert, update, and delete. Updates of specific columns may also be

used as triggering events.Because of their flexibility, triggers may supplement referential

integrity; they should not be used toreplace it. When enforcing the business rules in an

application, you should first rely on the declara- tive referential integrity available with

Oracle; use triggers to enforce rules that cannot be coded through referential integrity.In

other words, a trigger is a PL/SQL block that is associated with a table, stored in a database

and executed in response to a specific data manipulation event. Triggers can be executed, or

fired, in response to the following events:







A row is inserted into a table

A row in a table is updated

A row in a table is deleted

R It is not possible to define a trigger to fire when a row is selected.

A database trigger has three parts namely a trigger statement, a trigger body and a trigger



DBMS RECORD

Page No: 105

restric- Tion Trigger statement specifies the DML statements like insert , update, delete and

it fires the trigger body. It also specifies the table to which the trigger associated.Trigger

body is a PL/SQL bock that is executed when a triggering statement is issued.Restrictions on

a trigger can be achieved using the WHEN clause as shown in the syntax for creat-ing

triggers. They can be included in the definition of a row trigger, wherein, the condition in

the WHEN clause is evaluated for each row that is effected by the trigger.A trigger is a

database object, like a table or an index. When you define a trigger, it becomes a part of the

database and is always executed when the event for which it is defined occurs. It doesn’t

matter if the event is triggered by someone typing in a SQL statement using SQL\* Plus,

running a Client/Server program that updates the database, or running a utility like Oracle’s

SQL Loader in order to bulk-load data. Because of this, triggers serves as a choke point,

allowing you to perform critical data validation or computations in response to database

changes, no matter what the source.

Types of Triggers

A trigger ’s type is defined by the type of triggering transaction and by the level at which

the trigger is executed. In the following sections, you will see descriptions of these

classifications, along with relevant restrictions.

Row-Level Triggers

Row-level triggers execute once for each row in a transaction. Row-level triggers are the

most common type of trigger; they are often used in data auditing applications. Row-level

triggers are also useful for keeping distributed data in sync. Row-level triggers are created

using the for each row clause in the create trigger command.

Statement-Level Triggers

Statement-level triggers execute once for each transaction. For example, if a single

transaction inserted 500 rows into a table, then a statement-level trigger on that table would

only be executed once. Statement-level triggers therefore are not often used for data-related

activities; they are nor- mally used to enforce additional security measures on the types of

transactions that may be per- formed on a table.Statement-level triggers are the default type

of trigger created via the create trigger command.

BEFORE and AFTER Triggers



DBMS RECORD

Page No: 106

Because triggers are executed by events, they may be set to occur immediately before or

after those events. Since the events that execute triggers include database transactions,

trigger can be executed immediately before or after insert, update and delete. For database-

level events, addi- tional restrictions apply; you cannot trigger an event to occur before a

logon or startup takes place.

Within the trigger, you can reference the old and new values invoked by the transaction. The

access required for the old and new data may determine which type of trigger you need.

徹ld\_ refers to the data, as it existed prior to the transaction; updates and deletes usually

reference old values. New\_ values are the data values that the transaction creates (such

as the columns in an inserted record).

If you need to set a column value in an inserted row via your trigger, then you need to

use a BE- FORE INSERT trigger to access the 渡ew\_ values. Using an AFTER INSERT

trigger would not allow you to set the inserted value, since the row will already have been

inserted into the table. AFTER row-level triggers are frequently used in auditing

applications, since they do not fire until the row has been modified. The row’s successful

modification implies that it has passed the referen- tial integrity constraints defined for that

table.Together with commonly used four types, it gives a total of 12 possible trigger types,

which are listed in the following Table. Note that the SELECT statement is the only data

manipulation state-ment for which no triggers can be defined.

• Before update row

• Before update statement

• Before insert row

• Before insert statement

• Before delete row

• Before delete statement

• After update row

• After update statement

• After insert row

• After insert statement



DBMS RECORD

Page No: 107

• After delete row

• After delete statement

Also note that one trigger can be defined to fire for more than one SQL statement.

INSTEAD OF Triggers

You can use INSTEAD OF triggers to tell Oracle what to do instead of performing the

actions that invoked the trigger. For example, you could use an INSTEAD OF trigger on a

view to redirect inserts into table or to update multiple tables that are part of a view. You can

use INSTEAD OF triggers on either object views or relational views. For example, if a view

involves a join of two tables, your ability to use the update command on records in the view

is limited. However, if you use an INSTEAD OF trigger, you can tell Oracle how to update,

delete, or insert records in the view’s underlying tables when a user attempts to change

values via the view. The code in the INSTEAD OF trigger is executed in place of the update,

delete,or insert command you enter.

Uses of Triggers

The possible uses for database triggers are varied and are limited only by your imagination.

Some common uses are listed below:• Enforcing business rules

• Maintaining referential integrity

• Enforcing security

• Maintaining a historical log of changes

• Generating column values, including primary key values

• Replicating data

Syntax:

Create [ or replace ] trigger [user.]trigger\_name{ before | after | instead of }

{ delete | insert | update [ of column [, column] ・ }

on [user.]{ Table | View }

for each { row | statement }

[ when (condition) ]

PL/SQL Block

Clearly, there is a great deal of flexibility in the design of a trigger. The before and after



DBMS RECORD

Page No: 108

keywords indicate whether the trigger should be executed before or after the triggering

transaction. If the instead of clause is used, the trigger’s code will be executed instead of the

event that caused the trigger to be invoked. The delete, insert, and update keywords (the last

of which may include a column list) indicate the type of data manipulation that will

constitute a trigger event. When the for each row clause is used, the trigger will be a row-

level trigger; otherwise, it will be a statement-level trigger. The when clause is used to

further restrict when the trigger is executed. The restrictions enforced in the when clause

may include checks of old and new data values. For example, suppose we want to monitor

any adjustments to Salary column value that are greater than 10 percent. The following row-

level BEFORE UPDATE trigger will be executed only if the new value of the salary column

is more than 10 percent greater than its old value and add transac- tion details in audit table.

This example also illustrates the use of the new keyword, which refers to the new value of

the column, and the old keyword, which refers to the original value of the column.

Example

Create or replace trigger emp\_salary\_update\_row

before update on emp

for each row

when (:New.Amount / :Old.Amount > 1.1)

usr varchar2(20);

Begin

Select user into usr from dual;

Insert into EMP\_AUDIT values (:Old.salary, :New.salary, :Old.eno,

usr, to\_char(sysdate,’HH:MI’),sysdate);

Commit;

End;

/

Trigger created.

Breaking the above created trigger command makes it easier to understand. Let’s do it:

Create or replace trigger emp\_salary\_update\_row the emp\_salary\_update\_row is the trigger

name, which indicates table name and it acts upon and the type of trigger. One can define



DBMS RECORD

Page No: 109

trigger with any valid name. before update on emp above statement indicates that this trigger

applies to the Emp table and it will executed before update transactions have been

committed to the database. for each row Because of above statement, the trigger will apply

to each row in the transaction. If this clause is not used, then the trigger will execute at the

statement level. The When clause adds further criteria to the triggering condition. The

triggering event not only must be an update of the Ledger table, but also must reflect an

increase of over 10 percent in the value of the Amount column when (New.Amount /

Old.Amount > 1.1)

The PL/SQL code shown in the following listing is the trigger body. The commands shown

here are to be executed for every update of the emp table that passes the when condition. For

this to suc- ceed, the EMP\_AUDIT table must exist, and the owner must have been granted

privileges on that table. This example inserts the old values from the emp record into

EMP\_AUDIT table before the employee record is updated in emp table. Structure of

EMP\_AUDIT table is as per follows.

EMP\_AUDIT

eno number(5)

old\_salary number(9,2)

new\_salary number(9,2)

user varchar2(20)

tr\_time varchar2(10)

tr\_date date

Begin

Select user into usr from dual;

Insert into EMP\_AUDIT values (:Old.eno, :Old.salary,

:New.salary, usr, to\_char(sysdate,’HH:MI’),sysdate);

commit;

End;

/

Above trigger makes the log of the emp table in the EMP\_AUDIT table and maintains the

track of updation in the table, the user name, Transaction Date, Transaction Time, All the



DBMS RECORD

Page No: 110

old and new values of columns.R When referencing the New and Old keywords in the

PL/SQL block, they are preceded by the colons(:)

Using :Old and :New Values in Row Level Triggers

When Row level trigger fires once per row processed by the triggering statement. Inside the

trigger, you can access the row that is currently being processed. This is done through

keywords :new and :old. Following describes what values are set to :Old and :New with the

given triggering statement.

Triggering Statement

INSERT

: Old Undefined – all fields are NULL

: New Values that will be inserted when the statement is complete

UPDATE

: Old Original values for the row before the update

: New New values that will be updated when the statement is complete

DELETE

: Old Original values before the row is deleted

: New Undefined – all fields are NULL

They are valid only within row level triggers and not in statement level triggers. :Old values

are not available if the triggering statement is INSERT and :new values are not available if

the triggering statement is DELETE. Each column is referenced by using the expression

:Old.ColumnName or :New.ColumnName. if a column is not updated by the triggering

update statement then :old and :new values remain the same.

An Example of a Trigger with :New

Suppose for a moment that you wanted to be sure that all department names were stored

using uppercase letters. Perhaps you are doing this to facilitate searching on that field.

Following example

shows one way to do this with a trigger.

Example

Create or replace trigger upperdname before insert or update on

dept for each row



DBMS RECORD

Page No: 111

Begin

:new.dname := upper(:new.dname);

End;

/

Trigger created.

Example

Following trigger does not allow the user to delete any row from a Try table.

Create Or Replace Trigger delete\_rest\_trig Before Delete On Try

Begin

Raise\_Application\_Error(-20011,’UNABLE TO DELETE’);

End;

/

In the above example, a new Term is used i.e. Raise\_Application\_Error()

Let’s discuss it in details:

Customizing Error Conditions

Oracle provides a procedure named raise\_application\_error that allows programmers to issue

user- defined error messages.

Syntax :

Raise\_Application\_Error(Error\_number, Message);

Where Error\_number is a negative integer in the range –20000 to –20999. and Message Is a

string up to 2048 bytes in length An application can call 迭aise\_Application\_Error\_ only

from an executing stored subprogram like stored procedures and functions, database

triggers.

Typically

迭aise\_Application\_Error\_

is

used

in

database

triggers.

迭aise\_Application\_Error\_ ends the subprogram, rolls back any database changes it made,

and returns a user-defined error number and message to the application Within a single

trigger, you may establish different error conditions. For each of the error conditions you

define, you may select an error message that appears when the error occurs. The error num

bers and messages that are displayed to the user are set via the Raise\_Application\_Error



DBMS RECORD

Page No: 112

proce - dure, which may be called from within any trigger. Following example shows a

statement-level BEFORE UPDATE or INSERT trigger on the Emptable. When a user

attempts to Insert, modify or delete a row in the Emp table, this trigger is executed and

checks two system conditions; that the day of the week is neither Saturday nor Sun-day, and

that the Oracle username is other than user ‘ADMIN’.

Example

Create or replace trigger emp\_rest before insert or update or delete on Emp

Declare

Weekend\_Error Exception;

Invalid\_User Exception;

Begin

If to\_char(SysDate, ‘DY’) in (‘sat’,’sun’) then

Raise Weekend\_Error;

End if;

If upper(User) != ‘ADMIN’ then

Raise Invalid\_User;

End if;

Exception

When Weekend\_Error then Raise\_Application\_Error(-20001,’No Insertion or

Updation on Weekends’);

When Invalid\_User then

Raise\_Application\_Error(-20002,’Insertion, Updation or

deletion only allowed to Admin Users’);

End;

/

Trigger created.

Study following code and find out use of following Trigger

Create or replace trigger check\_date\_time before insert or update

or delete on dept



DBMS RECORD

Page No: 113

Begin

If to\_number(to\_char(sysdate,’hh24'))>not in(10,18) or

to\_char(sysdate, ‘dy’) not in (‘sun’, ‘sat’) then

raise\_application\_error(-20001,’no manipulations allowed in

table in non-office working hours’);

End if;

End;

Firing Triggers selectively using Trigger Predicates Problem

We have a lot of tables in the database. Instead of writing three different triggers for

INSERT, UPDATE, and DELETE operations, we want to write one trigger for each table in

the system, and that trigger should handle any DML operations on that table individually.

We need to know how to create such triggers and in the trigger body how to recognize the

type of DML operation that caused the trigger to fire. How to fire triggers selectively using

trigger predicates?

Solution

You can write a single trigger to handle multiple DML operations on a table. For

instance, an INSERT,DELETE or UPDATE statement can fire the same trigger with the user

of the ON IN- SERT OR DELETE OR UPDATE OF clause while creating the trigger. The

trigger body can use the conditional predicates INSERTING,DELETING, AND

UPDATING to execute specific blocks of code, depending upon the triggering statement.

Example

Create or replace trigger find\_tran before insert or update or

delete on dept for each row

Begin

If Inserting then

raise\_application\_error(-20001,’Insertion Restricted’);

elsif updating then

raise\_application\_error(-20002,’Updation Restricted’);

elsif deleting then

raise\_application\_error(-20003,’Deletion Restricted’);



DBMS RECORD

Page No: 114

end if;

End;

/

Examples:

Create of insert trigger, delete trigger and update trigger.

1. To write a TRIGGER to ensure that Bus table does not contain duplicate of null values in

Bus\_No column.

a) CREATE OR RELPLACE TRIGGER trig1 before insert on Bus for each row

DECLARE a number;

BEGIN

if(:new.Bus\_No is Null) then

raise\_application\_error(-20001,'error:: Bus\_No cannot be null');

else

select count(\*) into a from Bus where Bus\_No =:new. Bus\_No;

if(a=1) then

raise\_application\_error(-20002,'error:: cannot have duplicate Bus\_No ');

end if;

end if;

END;

RESULT:

SQL> @trigger

Trigger created.

SQL> select \* from Bus;

BUS\_NO

-----

110

221

412

501

SOURCE

DESTINATION

-------------------- --------------------

hyd

hyd

hyd

hyd

ban

chn

mum

kol

SQL> insert into Bus values(&Bus\_No,'&source','&destination');



DBMS RECORD

Page No: 115

Enter value for Bus\_No: null

Enter value for source: Chen

Enter value for destination: hyd

old 1: insert into Bus values(&Bus\_No, '&source', '&destination')

new 1: insert into Bus values(null,Chen','hyd')

insert into Bus values(null,’Chen','hyd')

\*

ERROR at line 1:

ORA-20001: error::Bus\_No cannot be null

35

ORA-06512: at "SYSTEM.TRIG1", line 5

ORA-04088: error during execution of trigger 'SYSTEM.TRIG1'

SQL> /

Enter value for Bus\_No: 110

Enter value for source:KOL

Enter value for destination: hyd

old 1: insert into Bus values(&Bus\_No, '&source', '&destination')

new 1: insert into Bus values(110,KOL','hyd')

insert into Bus values(110,’KOL','hyd')

\*

ERROR at line 1:

ORA-20002: error:: cannot have duplicate Bus\_No

ORA-06512: at "SYSTEM.TRIG1", line 9

ORA-04088: error during execution of trigger 'SYSTEM.TRIG1'

b) Create Trigger updchek before update on Ticket

For Each Row

Begin

If New.Ticket\_No>60 Then

Set New.Ticket\_No=Ticket\_No;



DBMS RECORD

Page No: 116

Else

Set New.Ticket\_No=0;

End If

End.

SQL> @trigger

Trigger created.

c) CREATE OR RELPLACE TRIGGER trig1 before insert on Passenger for each row

DECLARE a number;

BEGIN

if(:new.PNR\_NO is Null) then

raise\_application\_error(-20001,'error:: PNR\_NO cannot be null');

else

select count(\*) into a from Passenger where PNR\_NO =:new. PNR\_NO;

if(a=1) then

raise\_application\_error(-20002,'error:: cannot have duplicate PNR\_NO ');

end if;

end if;

END;

SQL> @trigger

Trigger created.



DBMS RECORD

Page No: 117

Exp No: 14

AIM : Implement Cursors:

Date: \_ \_/\_ \_/ \_ \_

Cursors:

In SQL when you submit a query, it returns number of rows depends on query. It may be

zero or may be hundreds. While in PL/SQL if your select statement returns multiple rows

then oracle must return Too\_many\_rows error message (Exception). In Such circumstances

it is necessary to manipulate multiple rows through PL/SQL Block without raising

Exception. The resource that Oracle provides to accomplish this job is the Cursor. PL/SQL

cursors provide a way for your program to select multiple rows of data from the database

and then to process each row individually. Specifically, a cursor is a name assigned by

Oracle to every SQL statement processed. This is done in order to provide Oracle a means to

direct and control all phases of the SQL processing

Two kinds of cursor are used by Oracle: Implicit and Explicit. PL/SQL implicitly declares a

cursor for every SQL statement. Implicit cursors are declared by Oracle for each UPDATE,

DELETE, and INSERT SQL command. Explicit cursors are declared and used by the user to

process multiple rows returned by a SELECT statement. Explicitly defined cursors are

constructs that enable the user to name an area of memory to hold a specific statement for

access at a later time.

Explicit Cursor

User define cursor are known as Explicit cursor. Explicit cursor is one in which the cursor



DBMS RECORD

Page No: 118

explicitly assigned to the select statement. Processing of explicit cursor involves four steps.

1) Declare the cursor

2) Open the cursor

3) Fetch data from cursor

4) Close the cursor

Declaring the cursor

The first step is to declare cursor in order for PL/SQL to reference the returned data. This

must be done in the declaration portion of your PL/SQL block. Declaring a cursor

accomplishes two goals:





It names the cursor

It associates a query with a cursor

The name you assign to a cursor is an undeclared identifier, not a PL/SQL variable. You

cannot assign values to a cursor name or use it in an expression. This name is used in the

PL/SQL block to reference the cursor query.

Cursor <cursor\_name> is <select statement>;

Where, cursor\_name is the name you assign to the cursor. SELECT statement is the query

that returns row to the cursor active set. In the following example, the cursor named

cur\_emp is defined with a SELECT statement that queries the employee table.

cursor cur\_emp is select \* from emp where ename like ‘A%’;

The only constraint that can limit the number of cursors is the availability of memory to

manage the cursors.

Opening the Cursor

Opening the cursor activates the query and identifies the active set. When the OPEN

command is executed, the cursor identifies only the rows that satisfy the query used with

cursor definition. The rows are not actually retrieved until the cursor fetch is issued. OPEN

also initializes the cursor pointer to just before the first row of the active set. Syntax to

opening a cursor is :

Open <cursor\_name>;

In this syntax, cursor\_name is the name of the cursor that you have previously defined.

R After a cursor is opened, until the moment you close it, all fetched data in the active



DBMS RECORD

Page No: 119

set will remain static This means that the cursor will ignore all SQL DML commands

(INSERT, UPDATE, DELETE and SELECT) performed on that data after the cursor was

opened. Hence, you should open the cursor only when you need it.

Fetching Data from the Cursor

Getting data form the Activ Set is accomplished with FETCH command. The FETCH

command retrieves the rows from the active set one row at a time. The FETCH command is

usually used in conjunction with some type of iterative process. The first FETCH statement

sorts the active set as necessary. In the iterative process, the cursor advances to the next row

in the active set each time the FETCH command is executed. The FETCH command is the

only means to navigate through the active set. Syntax for Fetching Data from the Cursor is :

Fetch <cursor\_name> into <record\_list>;

In this syntax, cursor\_name is the name of the previously defined cursor from which you are

now retrieving rows-one at a time. record\_list is a list of variables that will receive the

columns from the active set. The FETCH command places the results of the active set into

these variables. The record\_list or Variable list is the structure that receives the data of

fetched row. For each column value retrieved by the cursors’ query must have corresponding

variable in the INTO list. Additionally, their datatypes must be compatible.

If you want to revisit a previously fetched row, you must close and reopen the cursor and

then fetch each row in turn. If you want to change the active set, you must assign new values

to the input variables in the cursor query and reopen the cursor. This re-creates the active set

with the results of the revised query statement.

Closing the Cursor

The CLOSE statement closes or deactivates the previously opened cursor and makes the

active set undefined. Oracle will implicitly close a cursor when the user’s program or session

is termi- nated. After closing the cursor, you cannot perform any operation on it or you will

receive and invalid\_cursor exception. Syntax for closing a cursor:

Close <cursor\_name>;

where, cursor\_name is the name of the previously opened cursor

Example

Declare



DBMS RECORD

Page No: 120

cursor cur\_emp is select ename,Salary from emp;

nm emp.ename%type;

sal emp.salary%type;

Begin

open cur\_emp;

fetch cur\_emp into nm,sal;

dbms\_output.put\_line(‘Name : ‘ || nm);

dbms\_output.put\_line(‘Salary : ‘ || sal);

close cur\_emp;

End;

/

Above cursor will store all the records of emp table into active set but display only first

employee details. Because no iterative control is used with fetch statement.

To display all the employee details you should aware little about Cursor Attribute.

Explicit Cursor Attributes

Each cursor, whether it is explicitly or implicitly defined, carries with it attributes that

provide useful data of the cursor. The four cursor attributes are %isopen, %rowcount,

%found and %notfound. These attributes can be used in any PL/SQL statement. Cursor

attributes cannot be used against closed cursors, an invalid\_cursor exception will be raised if

you attempt this.

The %isopen Attribute

The %isopen attribute indicates whether the cursor is open. If the named cursor is open, then

this attribute equates to true; otherwise, it will be false. The following example uses the

%isopen at- tribute to open a cursor if it is not already open:

Example

Declare

cursor c1 is select \* from emp;

Begin

open c1;

if c1%isopen then



DBMS RECORD

Page No: 121

dbms\_output.put\_line(‘cursor already open’);

else

open c1;

end if;

close c1;

End;

/

The %notfound Attribute

The %notfound attribute is useful in telling you whether a cursor has any rows left in it to be

fetched. The %notfound Attribute equates to true when last fetch statement return no row

(there are no more rows remaining in Active Set), while it equates to false if last fetch

statement retuns row. Prior to the first fetch, this attribute will equate to null. An error will

be returned if you evalu- ate %notfound on a cursor that is not opened.

Following example illustrates use of %notfound attribute.

Example

Declare

cursor cur\_emp is select \* from emp where ename like ‘A%’;

emp\_rec emp%rowtype;

Begin

open cur\_emp;

loop

fetch cur\_emp into emp\_rec;

exit when cur\_emp%notfound;

dbms\_output.put\_line(‘Name : ‘ || emp\_rec.ename);

dbms\_output.put\_line(‘Age : ‘ || round((sysdate-

emp\_rec.bdate)/30, 0);

end loop;

close cur\_emp;

End;

/



DBMS RECORD

Page No: 122

Above PL/SQL block of cursor will store all the records of employee into active set whose

name start with A and display their name with the age.

The %found Attribute:

The %found attribute equates to true if the last FETCH statement returns row. Therefore, the

%found attribute is a logical opposite of the %notfound attribute. The %found attribute

equates to false when no rows are fetched. Like the %notfound, this attribute also equates to

null prior to the first fetch.

The following example illustrates practical use of %found attribute

Example

Declare

cursor cur\_emp is select ename,salary from emp;

nm emp.ename%type;

sal emp.salary%type;

Begin

open cur\_emp;

loop

fetch cur\_emp into nm,sal;

if cur\_emp%found then

dbms\_output.put\_line(‘Name : ’|| nm);

dbms\_output.put\_line(‘Salary : ’|| sal);

else

exit;

end if;

end loop;

close cur\_emp;

End;

/

The %rowCount Attribute:

The %rowCount attribute returns the number of rows fetched so far for the cursor. Prior to

the first fetch, %rowcount is zero. There are many practical applications of the %rowcount



DBMS RECORD

Page No: 123

attribute. The following example will perform a commit after the first 250 employees’

salaries are processed.

Example

Declare

cursor c1 is select \* from emp where salary > 4000;

emp\_rec emp%rowtype;

Begin

open c1;

loop

fetch c1 into emp\_rec;

exit when c1%rowcount > 5;

dbms\_output.put\_line(emp\_rec.ename);

end loop;

close c1;

End;

/

Above PL/SQL block of cursor will display only first five records whose salary is greater

than 4000.

Automated Explicit Cursors (Cursor For Loop)

The previous section illustrated the basic mechanism of declaring and using cursors. In

many pro- gramming situations, there is more than one way to code your logic. This also

applies to PL/SQL cursors; there are opportunities to streamline or simplify the coding and

usage of them. An alternate way to open, fetch and close the cursor Oracle furnishes another

approach to place the cursor within a FOR Loop. This is known as a CURSOR FOR loop. A

CURSOR FOR loop will implicitly.









Declare the Loop Index

Open the Cursor

Fetch the next row from the cursor for each loop iteration

Close the cursor when all rows are processed or when the loop exits

The Syntax for cursor for loop :



DBMS RECORD

Page No: 124

For <record\_list> in <cursor\_name>

Loop

Statements;

End loop;

CURSOR FOR loops are ideal when you want all the records returned by the cursor. With

CUR- SOR FOR loops, you should not declare the record that controls the loop.

Example

Declare

cursor emp\_cursor is select \* from emp where deptno in (10,30);

emp\_rec emp%rowtype;

Begin

for emp\_rec in emp\_cursor

loop

update emp set salary = salary + (salary \* 0.10) where

empno = emp\_rec.empno;

end loop;

End;

/

Aim: To write a Cursor to display the list of Male and Female Passengers.

Analyst.

DECLARE

cursor c(jb varchar2) is select Name from Passenger where Sex=m;

pr Passenger.Sex%type;

BEGIN

open c('m');

dbms\_RESULT.put\_line(' Name of Male Passenger are:');

loop

fetch c into pr;

exit when c%notfound;



DBMS RECORD

Page No: 125

dbms\_RESULT.put\_line(pr);

end loop;

close c;

open c('f');

dbms\_RESULT.put\_line(' Name of female Passengers are:');

loop

fetch c into em;

exit when c%notfound;

dbms\_RESULT.put\_line(em);

end loop;

close c;

END;

RESULT:

Name of Male Passenger are:

SACHIN

rahul

rafi

salim

riyaz

Name of female Passengers are:

swetha

neha

PL/SQL procedure successfully completed.

b) To write a Cursor to display List of Passengers from Passenger Table.

DECLARE

cursor c is select PNR\_NO, Name, Age, Sex from Passenger ;

i Passenger.PNR\_NO%type;

j Passenger.Name%type;

k Passenger.Age%type;



DBMS RECORD

Page No: 126

l Passenger.Sex%type;

BEGIN

open c;

dbms\_RESULT.put\_line('PNR\_NO, Name, Age, Sex of Passengers are:= ');

loop

fetch c into i, j, k, l;

exit when c%notfound;

dbms\_RESULT.put\_line(i||' '||j||' '||k||' '||l);

end loop;

close c;

END;

RESULT:

SQL>@Passenger

PNR\_NO

----------

1

2

3

4

NAME

AGE SEX

-------------------- ---------- ----------

SACHIN

rahul

swetha

rafi

12

43

24

22

m

m

f

m

PL/SQL procedure successfully completed.



DBMS RECORD

Page No: 127

Exp No: 15

AIM : Implement SubPrograms in PL/SQL.

SUBPROGRAMS:

Date: \_ \_/\_ \_/ \_ \_

So, far we have seen PL/SQL Block (Anonymous Block) which are executed by

interactively enter- ing the block at the SQL prompt or by writing the PL/SQL statements in

a user\_named file and executing the block at SQL prompt using @ command. The block

needed to compile at every time it is run and only the user who created the block can use the

block. Any PL/SQL block consists of some hundreds statements, in such cases it is

necessary to break the entire block into smaller modules depending on your requirements.

So, your block became more easy to understand and efficient to perform operation and

maintenance. Stored procedures / Sub- programs are such kind of named PL/SQL Block.

Basically it is Sophisticated Business rules and application logic.

Stored subprogram are compiled at time of creation and stored in the database itself. The

source code is also stored in the database. Any user with necessary privileges can use the

stored subpro- gram.

Procedures and Functions are subprograms having group of SQL, PL/SQL and Java–

enables

Statements you to move code that enforce the business rules from your application to

database. It takes a set of parameters given to them by the calling program and perform a set

of actions. The only real difference between a procedure and a function is that a function

will include a single return value. Both functions and procedures can modify and return data



DBMS RECORD

Page No: 128

passed to them as a param- eter.

A procedure or function that has been stored in the library cache is referred to as a stored

proce- dure or a stored function. A stored procedure or stored function has the following

characteristics:

• It has a name: This is the name by which the stored procedure or function is called and

referenced.

• It takes parameters: These are the values sent to the stored procedure or function from the

application.

• It returns values: A stored procedure or function can return one or more values based on

the purpose of the procedure or function.

Procedures

A procedure is a one kind of subprogram, which is designed and created to perform a

specific operation on data in your database. A procedure takes zero or more input parameters

and returns 125 zero or more output parameters.

The syntax of a creation of procedure is as follows:

Syntax:

CREATE OR REPLACE PROCEDURE procedure\_name

[(argument1 [IN/OUT/IN OUT] datatype,

argument2 [IN/OUT/IN OUT] datatype,\_)] IS

[<local variable declarations>]

BEGIN

Executable Statements

[EXCEPTION

Optional Exception Handler(s)

]

END;

The procedure is made up of two parts: the declaration and the body of the procedure. The

declara- tion begins with the keyword PROCEDURE and ends with the last parameter



DBMS RECORD

Page No: 129

declaration. The body begins with the keyword IS and ends with the keyword END. The

procedure body is further divided into three parts : declarative, executable and exception part

same as PL/SQL blcok. The declaration section is used to assign name and define parameter

list, which variables are passed to the procedure and which values are returned from the

procedure back to the calling program.

Parameters can be define in following format

Argument [parameter mode] datatype

There are three types of parameters mode: IN, OUT and IN OUT

IN Mode

• Default parameter mode.

• Used to pass values to the procedure.

• Formal parameter can be a constant, literal, initialized variable or expression.

• Used for reading purpose

OUT Mode

• Used to return values to the caller.

• Formal parameter cannot be used in an expression, but should be assigned a value.

• Used for writing purpose

126 OUT Mode

• Used to pass values to the procedure as well as return values to the caller

• Formal parameter acts like an initialized variable and should be assigned a value.

• Used for both reading and writing purpose

R In a procedure declaration, it is illegal to constrain char and varchar parameter with

length and number parameter with precision and scale.

The body of the procedure is where the real work is done. The body is made up of the

PL/SQL statements that perform the desired task.

The EXCEPTION Section

In both procedures and functions, you can add optional exception handlers. These exception

han- dlers allow you to return additional information based on certain conditions (such as no

data found or some user-specified condition). By using exception handlers and allowing the

stored procedure to notify you of some special conditions, you can minimize the amount of



DBMS RECORD

Page No: 130

return-value checking that must be done in the application code. Because the work to

determine that no data has been selected has already been done by the RDBMS engine, you

can save on resources if you take advantage of this information.

Example

Create a procedure, which receives a number and display whether it is odd or even.

Create or replace procedure oddeven (num in number) is

a number(3);

Begin

a := mod(num,2);

If a = 0 then

dbms\_output.put\_line( num ||’ is even number’);

Else

dbms\_output.put\_line( num ||’ is odd number’);

End if;

End;

/

Procedure created.

127

Execution of procedure

Procedure is executed from a SQL prompt as per follows and One can execute procedure

from caller program also.

SQL > execute/exec procedure\_name(parameter list)

For example above created procedure is executed as follows

SQL> Exec example1(7)

7 is odd number

Example

Make a procedure, which will accept a number and return it’s Square.

Create or replace procedure square\_no(num in number,ans out number)

Is

Begin



DBMS RECORD

Page No: 131

ans:=num\*num;

End;

/

Procedure created.

To execute above procedure we make one block, which is known as a Caller Block

Declare

ret number(10);

no number(5) := &no;

Begin

square\_no (no, ret);

dbms\_output.put\_line(‘Square of ‘||no||’ is : ‘|| ret);

End;

/

Output

Enter value for no: 10

old 3: no number(5) := &no;

new 3: no number(5) := 10;

Square of 10 is : 100

128

Procedures Related to Table

Example

Pass employee no and name and store into employee table.

Create or replace procedure emp\_add (eno emp.empno%type, enm

emp.ename%type)

Is

Begin

Insert into emp(empno,ename) values(eno,enm);

Exception

When Dup\_val\_on\_index then

dbms\_output.put\_line(‘Employee Number already Exist’);



DBMS RECORD

Page No: 132

End;

/

To run above procedure

SQL> exec emp\_add(1013,’DAXESH’);

Example

Create a Procedure, which receives employee number and display employee name,

Designation and salary.

Create or replace procedure empdata(eno in number)

Is

enm varchar2(20);

jb varchar2(20);

sal number(10,2);

Begin

Select ename, job, salary into enm, jb, sal from emp where

empno= eno;

dbms\_output.put\_line(‘employee name

: ‘|| enm);

dbms\_output.put\_line(‘employee designation: ‘|| jb);

dbms\_output.put\_line(‘employee salary

End;

/

Procedure created.

Output

SQL>exec empdata(1051)

129 employee name

: RISHI

: ‘|| sal);

employee designation: ANALYST

employee salary

Example

Write a PL/SQL block, which will use a user-defined procedure, which accept employee

number and return employee name and department name in a out parameter.

Create or replace procedure emp\_data(eno number, enm out varchar2, dnm out varchar2)

: 5000



DBMS RECORD

Page No: 133

Is

Begin

select ename, dname into enm, dnm from emp, dept where emp.deptno

= dept.deptno and empno = eno;

End;

/

Procedure created

Block To execute procedure

Declare

employee varchar2(30);

department varchar2(20);

eno number(4);

Begin

eno :=&employeenumber;

emp\_data(eno, employee, department);

dbms\_output.put\_line(‘Employee Number ‘||eno);

dbms\_output.put\_line(‘Employee Name ‘||employee);

dbms\_output.put\_line(‘Department Name ‘||department);

End;

/

Output

Enter value for employeenumber: 1011

old 6: eno :=&EmployeeNumber;

new 6: eno :=1011;

Employee Number 1011

Employee Name TEJAS

Department Name RESEARCH

130

Example

Create a procedure, which receives department number and get total Salary of that



DBMS RECORD

Page No: 134

Department.

Create or replace procedure dept\_total(dno in numbe, total out number)

Is

Begin

Select sum(salary) into total from emp where deptno= dno;

dbms\_output.put\_line(‘Total salary of Department ‘|| dno ||

‘ is ‘ || total);

End;

/

Procedure created.

Block To execute procedure

Declare

dn number(5) := &no;

tot number;

Begin

dept\_total(dn,tot);

End;

/

Output

Enter value for dn: 10

old 2: dn number(5) := &dn;

new 2: dn number(5) := 10;

Total salary of Department 10 is 235300

Example

Write procedure to accept Department number and display Name, Designation and Age of

each employee belonging to such Department.

Create or replace procedure dept\_list(dno number)

Is

cursor c1 is select \* from emp where deptno = dno;

erec emp%rowtype;



DBMS RECORD

Page No: 135

Begin

For erec in c1

loop

131dbms\_output.put\_line(‘Emp. Name : ‘ || erec.ename);

dbms\_output.put\_line(‘Designation : ‘|| erec.desg);

dbms\_output.put\_line(‘Age : ‘|| round((sysdate-erec.bdate)/

365,0);

dbms\_output.put\_line(‘=============================’);

End loop;

End;

/

Procedure created.

Output

SQL>exec dept\_list(20);

Emp. Name : AANSHI

Designation : ANALYST

Age : 21

=========================

Emp. Name : TEJAS

Designation : MANAGER

Age : 27

=========================

Emp. Name : DAXESH

Designation : MANAGER

Age : 24

=========================

Example

Create a procedure, which will accept Deptno and Display no of employee under different

grade.

Create or replace procedure empcount(dno in number)



DBMS RECORD

Page No: 136

Is

Cursor c1 is select grade, count(\*) from emp where deptno = dno group by grade;

grd varchar2(3);

noofemp number(3);

Begin

Open c1;

dbms\_output.put\_line(‘Grade ‘||’No of employee’);

Loop

Fetch c1 into vgrade, noofemp;

132Exit when c1%notfound;

dbms\_output.put\_line(grd||’

End loop;

Close c1;

End;

/

Procedure created.

Output

SQL>exec empcount(30);

Grade No of employee

A1

B2

C2

D1

‘||noofemp);



DBMS RECORD

Page No: 137

Exp No: 16

AIM : Implement Functions of PL/SQL.

Functions:

Date: \_ \_/\_ \_/ \_ \_

A function, like a procedure, is a set of PL/SQL statements that form a subprogram. The

subpro- gram is designed and created to perform a specific operation on data. A function

takes zero or more input parameters and returns just one output value. If more than one

output value is required, a procedure should be used. The syntax of a function is as follows:

Syntax

CREATE OR REPLACE Function function\_name

[(argument1 [IN/OUT/IN OUT] datatype,

argument2 [IN/OUT/IN OUT] datatype,\_)] RETURN datatype IS

[<local variable declarations>]

BEGIN

PL/SQL Statements

[EXCEPTION

Optional Exception Handler(s)]

END [function\_name];

/

As with a procedure, a function is made up of two parts: the declaration and the body. The

declara- tion begins with the keyword Function and ends with RETURN statement. The

body begins with the keyword IS and ends with the keyword END.



DBMS RECORD

Page No: 138

The difference between a procedure and a function is the return value. A function has the

return declaration as well as a RETURN function within the body of that function that

returns a value. This RETURN function is used to pass a return value to the calling program.

133

R If you do not intend to return a value to the calling program, or you want to return

more than one value, use a procedure.

How Procedures and Functions Operate

Procedures and functions use the same basic syntax in the program body with the exception

of the RETURN keyword, which can only be used by functions. The body itself is made up

of PL/SQL blocks that perform the desired function and return the desired data to the calling

program. The goal of the body of the procedure is both to minimize the amount of data to be

transmitted across the network (to and from the calling program) and to perform the PL/SQL

statements in the most effi- cient manner possible.

Example

Create a function to get cube of passed number

Create or replace function cube(no number) return number

Is

ans number(4);

Begin

ans := no \* no \* no;

return ans;

End;

/

Function created.

Output

SQL> select cube(5) from dual;

CUBE(5)

———-

125

Example



DBMS RECORD

Page No: 139

Write a Function to find out maximum salary for the passed designation.

Create or replace function maxjob(des varchar2) return number Is

maxsal number(7,2);

Begin

select max(sal) into maxsal from emp where job = des;

return maxsal;

End;

/

134Function created.

Output

SQL> SELECT MAXJOB(‘ANALYST’) FROM DUAL;

MAXJOB(‘ANALYST’)

————————-

6725

Example

Create a Function to find out existence of employee whose name is passed as a parameter

Create or replace function empchk(enm varchar2) return boolean Is

erec emp%rowtype;

Begin

select \* into erec from emp where ename = enm;

return true;

Exception

When no\_data\_found then

return false;

When too\_manu\_rows then

Return true;

End;

/

Function created.

Block To execute procedure (Caller Program)



DBMS RECORD

Page No: 140

Declare

nm emp.ename%type;

b Boolean;

Begin

nm := &employeename

b :=empchk(nm);

if b = true then

dbms\_output.put\_line(‘Employee Exist’);

else

dbms\_output.put\_line(‘Employee not exist’);

end if;

End;

/

135

Example

Write a Function to find out Total salary for the passed department Name.

Create or replace function totsalary(dnm varchar2) return number

Is

totsal emp.sal%type;

Begin

select sum(sal) into totsal from emp,dept where dept.deptno

= emp.deptno and dname = dnm;

return totsal;

End;

/

Function created.

Block To execute procedure

Declare

tot number;

dnm dept.dname%type := &departmentname;



DBMS RECORD

Page No: 141

Begin

tot = totalsalary(dnm);

dbms\_output.put\_line(‘Total Salary of ‘|| dnm || ‘ is ‘ ||

tot);

End;

/

Example

Write a Function to find out No Of Employee who joined between dates passed.

Create or replace function noofemp(date1 date,date2 date) return

number

Is

noofemp number(7);

Begin

select count(\*) into noofemp from emp where hiredate between

date1 and date2;

return noofemp;

End;

/

Function created.

136

Output

SQL> select noofemp(’20-dec-80',’20-dec-81') from dual;

NOOFEMP(’20-DEC-80',’20-DEC-81')

————————————————

11

Example

Write a function to check whether passed number is Odd or Even.

Create or replace function oddeven(no number) return varchar2 is

Begin

If mod(no,2)=0 then



DBMS RECORD

Page No: 142

return

Else

return no || ‘ is odd’;

End if;

End;

/

Function created.

Output

SQL>Select oddeven(10) from dual;

ODDEVEN(10)

—————-

10 is Even

SQL> select oddeven(11) from dual;

ODDEVEN(11)

—————-

11 is Odd

Example

Write a Function to find out total annual income for the employee, who’s number we passed.

Create or replace function ann\_income(eno number) return number

Is total number(9,2);

Begin

137Select (salary + nvl(comm,0))\*12 into total from emp where

empno = eno;

return total;

End;

/

Function created.

Output

SQL> select ann\_income(1010) from dual;

ANN\_SAL(1010)



DBMS RECORD

Page No: 143

——————-

60000

Example

Create a function, which receives three arguments, first two as a number and third as a

arithmetic operator and return proper answer, In case of invalid operator display appropriate

message.

Create or replace function calc(a number, b number, c char, x boolean) return number is

d number; invalid\_opt exception;

Begin

If c =’+’ then

d := a + b;

Elsif c=’-’ then

d := a-b;

Elsif c=’/’ then

d := a/b;

Elsif c=’\*’ then

d := a\*b;

Else

Raise invaid\_opt;

End if;

x:=true;

Return d;

Exception

When value\_error then

dbms\_output.put\_line(‘cannot perform calculation’);

x:=false;

return 0;

When invalid\_opt then

x:=false;



DBMS RECORD

Page No: 144

return 0;

End;

/

Block To execute procedure

Declare

n1 number;

n2 number;

ans number;

op varchar2;

valid Boolean;

Begin

n1 := &firstnumber;

n2 := &secondnumber;

op := &operator

ans = (n1,n2,op,valid);

if valid then

dbms\_output.put\_line(n1 op n2 || ‘ is ‘ || ans);

end if;

End;

/