

# **ALAGAPPA UNIVERSITY**

[Accredited with 'A+' Grade by NAAC (CGPA:3.64) in the Third Cycle and Graded as Category–I University by MHRD-UGC](A State University Established by the Government of Tamil Nadu)



KARAIKUDI – 630 003

# **Directorate of Distance Education**

# Master of Computer Applications II - Semester 315 24

# **RDBMS LAB**

#### Author

Dr. Kavita Saini, Associate Professor, School of Computer Science & Engineering, Galgotias University, Greater Noida.

"The copyright shall be vested with Alagappa University" All rights reserved. No part of this publication which is material protected by this copyright notice may be reproduced or transmitted or utilized or stored in any form or by any means now known or hereinafter invented, electronic, digital or mechanical, including photocopying, scanning, recording or by any information storage or retrieval system, without prior written permission from the Alagappa University, Karaikudi, Tamil Nadu.

Information contained in this book has been published by VIKAS<sup>®</sup> Publishing House Pvt. Ltd. and has been obtained by its Authors from sources believed to be reliable and are correct to the best of their knowledge. However, the Alagappa University, Publisher and its Authors shall in no event be liable for any errors, omissions or damages arising out of use of this information and specifically disclaim any implied warranties or merchantability or fitness for any particular use.



Vikas<sup>®</sup> is the registered trademark of Vikas<sup>®</sup> Publishing House Pvt. Ltd.

VIKAS<sup>®</sup> PUBLISHING HOUSE PVT. LTD. E-28, Sector-8, Noida - 201301 (UP) Phone: 0120-4078900 • Fax: 0120-4078999 Regd. Office: 7361, Ravindra Mansion, Ram Nagar, New Delhi 110 055 • Website: www.vikaspublishing.com • Email: helpline@vikaspublishing.com

Work Order No. AU/DDE/DE1-291/Preparation and Printing of Course Materials/2018 Dated 19.11.2018 Copies - 500

# **RDBMS LAB**

## Syllabi

#### **BLOCK 1: TABLE MANIPULATION**

- 1. Table Creation, Renaming a Table, Copying another Table, Dropping a Table
- 2. Table Description: Describing Table Definitions, Modifying Tables, Joining Tables, Number and Date Functions.

#### **BLOCK 2: SQLQUERIES AND SUB QUERIES**

- 3. SQL Queries: Queries, Sub Queries, and aggregate functions
- 4. DDL: Experiments using database DDL SQL statements
- 5. DML: Experiment using database DML SQL statements
- 6. DCL: Experiment using database DCL SQL statements

#### **BLOCK 3: INDEX AND VIEW**

- 7. Index : Experiment using database index creation, Renaming a index, Copying another index, Dropping a index
- 8. Views: Create Views, Partition and locks

## **BLOCK 4: EXCEPTION HANDLING AND PL/SQL**

- 9. Exception Handling: PL/SQL Procedure for application using exception handling
- 10. Cursor: PL/SQL Procedure for application using cursors
- 11. Trigger: PL/SQL Procedure for application using triggers
- 12. Package: PL/SQL Procedure for application using package
- 13. Reports: DBMS programs to prepare report using functions

#### **BLOCK 5: APPLICATION DEVELOPMENT**

14. Design and Develop Application: Library information system, Students mark sheet processing, Telephone directory maintenance, Gas booking and delivering, Electricity bill processing, Bank Transaction, Pay roll processing. Personal information system, Question database and conducting Quiz and Personal diary Introduction

## **INTRODUCTION**

## NOTES

Rapid globalization coupled with the growth of the Internet and information technology has led to a complete transformation in the way organizations function today. Organizations require those information systems that would provide them a 'competitive strength' by handling online operations, controlling operational and transactional applications, and implementing the management control tools. All this demands the Relational Database Management System or RDBMS which can serve both the decision support and the transaction processing requirements. Technically, the present RDBMS handles the distributed heterogeneous data sources, software environments and hardware platforms. Precisely, RDBMS is a Database Management System or DBMS that is based on the relational model introduced by E. F. Codd.

The most widely used commercial and open source databases are based on the relational model. Characteristically, a RDBMS is a DBMS in which data is stored in tables and the relationships among the data are also stored in tables. This stored data can be accessed or reassembled in many different ways without having to change the table forms. RDBMS program lets you create, update and manage a relational database. In spite of repeated challenges by competing technologies, as well as the claim by some experts that no current RDBMS has fully implemented relational principles, the majority of new corporate databases are still being created and managed with an RDBMS. So, understanding RDBMS through lab manuals has become extremely important.

This Lab Manual is intended for the students of MCA in the subject of RDBMS. This manual typically contains practical/Lab Sessions related to RDBMS, covering various aspects related to the subject to enhanced understanding. Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the textbooks.

Self-Instructional Material A **Database Management System** (RDBMS) is a collection of database and stored procedures. A RDBMS enables you to store, extract and manage important information from a database. It is software that is used to maintain data security and data integrity in a structured database.

As mentioned earlier in section RDBMS helps in maintaining and retrieving data in different form. There are various tools available for RDBMS such as Oracle, INGRES, Sybase, Microsoft SQL Server, MS-Access, IBM-DB-II, and My SQL.

## **Application of DBMS in various fields**

In day to day life, various applications are in use. Few of the application are given below where database is used:

- Banking: For account holder information, amount with draw and deposit, load and other transactions.
- Airlines: For reservations, cancelation, fare detail and airline schedules.
- Universities: For student registration, examination, fee detail, course detail and other information.
- Manufacturing: For inventory, production, sale and purchase orders
- Human Resources: Employee records, salaries, tax deductions, allowances
- Multimedia application
- Real Time Application
- Graphical Information System (GIS)

## **Introduction to Oracle**

**Oracle** is a secure portable and powerful database management system of Oracle Corporation. Oracle Corporation is an American multinational computer technology corporation headquartered in Redwood Shores, California. Oracle Database is compatible and connectable with almost all operating systems and machine. It is based on relational data model and a non-procedural language called structure query language (SQL). It is a tool that supports storing managing and organization the data.

## Getting Started with SQL:

To work with SQL \*Plus Oracle should to be installed on computer system. The following steps are required to follow to invoke SQL plus:

- 1. Click on Start button
- 2. Click on All Programs
- 3. Click on Oracle Database 10g Express Edition
- 4. Click on Go to Database Home Page

Self-Instructional Material

1

RDBMS Lab

## NOTES



2 Material

The following Screen will appear:

RDBMS Lab

NOTES

ORACLE' Database Express Edition



Self-Instructional Material

3

After clicking on **SQL Command**, following command screen will appear, where we can type and run all SQL commands:

## NOTES

Autocommit Display 10 •	Save

Enter SQL statement or PL/SQL command and click Run to see the results.

## **Data Types in Oracle**

When you define any table, it is required to specify the data type of fields. The main categories of data types are:

Data Type	Size
Char (size)	Maximum size of 2000 bytes
Varchar2 (size)	Maximum size of 4000 bytes
Long	Maximum size of 2GB
Raw (size)	Maximum size of 2000 bytes
long raw (size)	Maximum size of 2GB
Number(p,s)	Precision can range from 1 to 38.
	Scale can range from -84 to 127.
Date	A date between Jan 1, 4712 BC
	and Dec 31, 9999 AD.

## **Operators in Oracle**

Operators are the special characters that manipulate data items to produce some result. These data items are called *operands*. Operators are classified into two categories:

- 1. Unary Operators
- 2. Binary Operators

## 1. Unary Operators

A unary operator operates only on one operand. A unary operator is used as shown below:

## Syntax:

Operator operand

Self-Instructional Material

4

RDBMS Lab

## 2. Binary Operators

A binary operator operates on two operands. A binary operator is used as shown below:

## Syntax:

Operand1 operator operand2

There are various types of operators to cater different purpose which includes:

- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Set Operators
- Concatenates Operator

## **Creating a Table**

DDL (Data Definition Language) is the subset of SQL commands used to modify, create or remove ORACLE database objects, including tables.it is used to define the structure of a table. In a table structure you define various fields, their data types and constraints as per the requirement.

## Syntax:

```
Create table <table_name >
    (column_name data type(size), column_name data
type(size),.....);
```

**Example 1:** Create a table *Course* with the fields, data types and constraints as shown below.

Column Name	Data Type	Size
c_code	varchar2	15
c_name	varchar2	15
duration	number	8
free	number	10,2

The window below shows the query for creating the table as specified and Oracle will prompt a message.

Autocommit Display 10 🔹		
Create Table Course ( C_code <u>varchar</u> (15), Duration humber (8), );	C_name <u>varchar</u> (15), Fee number (10,2)	

RDBMS Lab

NOTES

Self-Instructional Material

5

Table created.

# *RDBMS Lab* **Example 2:** Create a table *Student* with the fields, data types and constraints as shown below.

## NOTES

Column Name	Data Type	Size
Roll_No	Varchar	10
Name	Varchar	10
Address	Varchar	35
C_Code	Varchar	8

The window below shows the query for creating the table as specified and Oracle will prompt a message.

Autocommit Display	10 🔹			
reate Table Student ( Roll_No Name Address C_Code );	varchar varchar varchar varchar	(10), (10), (35), (8)		
Results Explain Descrit	e Saved SQ	L History		

Table created.

## **Rename Tables**

The *Syntax* for renaming the table name is:

Rename old\_table\_name to new\_table\_name;

**Example 3:** Write a query to rename table *student* to *student MCA*.

User: KAVITA



Statement processed.

Self-Instructional Material

6

## **Dropping a Table**

When a SQL table is no more required, you can delete it using DROP command. Drop command is used to drop any object such as table, index, view, package and function.

## Syntax:

Drop table <table\_name >

**Example 4:** Write a query to drop table *course*.



Results Explain Describe Saved SQL History

Table dropped.

## Truncate a Table

This command will remove all the records from a table. But structure will remain same.

#### Syntax:

Truncate Table <Table name>

Example 5: Write a query to truncate table *Student*.



## NOTES

Run

RDBMS Lab

Self-Instructional Material

7

Table truncated.

#### RDBMS Lab

## **Describe the Table**

Describe command is used to describe the structure of a table created in the database.

## NOTES

## Syntax:

Describe <table\_name> Or Desc <table\_name>

**Example 6:** Write a query to see the structure of *course* table.

Results       Explain       Describe       Saved SQL       History         Object Type       TABLE Object       COURSE       Table       Column       Data Type       Length       Precision       Scale       Primary Key       Nullable       Default       Com         COURSE       0.002E       Vercher2       15       - <th></th>											
Results       Explain       Describe       Saved SQL       History         Object Type       TABLE Object       COURSE       Scale       Primary Key       Nullable       Default       Com         COURSE       C.CODE       Varchar2       15       -       -       ✓       -											
Results       Explain       Describe       Saved SQL       History         Object Type       TABLE Object       COURSE       - <t< th=""><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th></t<>											
Results       Explain       Describe       Saved SQL       History         Object Type       TABLE Object       COURSE       - <t< th=""><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th></t<>											
Results       Explain       Describe       Saved SQL       History         Object Type       TABLE Object       COURSE											
Results       Explain       Describe       Saved SQL       History         Object Type       TABLE Object       COURSE											
Results       Explain       Describe       Saved SQL       History         Object Type       TABLE Object       COURSE       Scale       Primary Key       Nullable       Default       Con         COURSE       C.COOP       Varchar2       15       -											
Object Type TABLE Object COURSE Table Column Data Type Length Precision Scale Primary Key Nullable Default Com COURSE C.CODE Varchar2 15	Results	Explain Des	cribe Saved	SQL His	tory						
Table       Column       Data Type       Length       Precision       Scale       Primary Key       Nullable       Default       Con         C_NAME       Varchar2       15       -       -       ✓       -<	Obiect Tvo	e TABLE C	biect COUR	SE							
COURSE         C. CODE         Varchar(2         15         -         -         ✓         -	Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Com	m
C.NAME       Varchar(2       15       -	COURSE	C CODE	Varchar2	15		-	5	~			
DURATION         Number         0         0         -         ✓         -		<u>C NAME</u>	Varchar2	15	12	20	2	~	-	- C.	
EEE       Number       10       2       ✓       .       .       1-4         Or         User: KAVITA         Home > SQL > SQL Commands         Mutocommit Display 10 ▼       Save         Desc Course       Save       Save         Results Explain Describe Saved SQL History       Cobject Type TABLE Object COURSE       Table       Course         Table       Column       Detatype       Length       Precision       Scale       Primary Key       Nullable       Default       Commit Course         COURSE       Course		DURATION	Number		0	D	2	~			
1-4       Or       User: KAVITA       Home > SQL > SQL Commands       Image: Market Display 10 • Save       Desc Course       Desc Course       Course       Object Course       Cobject Type TABLE Object COURSE       Table Column Data Type Length Precision Scale Primary Key Nullable Default Comm       COURSE COURSE       COURSE 0. 1 · · · · · · · · · · · · · · · · · ·		<u>FEE</u>	Number	ал. С	10	2	2	~	- 20		
Or         Just: KAVITA         Mome > SQL > SQL Commands         Image: Autocommit Display 10       Save         Desc Course       Save         Colorse         Colorse Saved SQL History         Object Type TABLE Object COURSE         Table Column Data Type Length Precision Scale Primary Key Nullable Default Comm         COURSE 0 CODE Varchar2 15											_
Results       Explain       Describe       Save         Results       Explain       Describe       Save         Object Type       TABLE Object       COURSE         Table       Column       Data Type       Length         Precision       Scale       Primary Key       Nullable         Default       Common       Common       Common         COURSE       -       -       -         QUASE       15       -       -       -         QUASE       15       -       -       -											
Matter Course       Save         Desc Course       Save         Results Explain Describe Saved SQL History         Object Type TABLE Object COURSE         Table       Column Data Type         Length       Precision         Scale       Primary Key         Nullable       Default         COURSE       -	User: KAVI		ana and a					_	_	_	
Desc Course         Results Explain Describe Saved SQL History         Object Type TABLE Object COURSE         Table Column Data Type Length Precision Scale Primary Key Nullable Default Comm         COURSE C. CODE Varchar2         00 NAME Varchar2         15       -         -       -         0 Diable Varchar2         0 Diable Varchar2         0	User: KAVI Home > SC	TA QL > <mark>SQL Con</mark>	nmands	7771							
Results Explain Describe Saved SQL History         Object Type TABLE Object COURSE         Table       Column       Data Type       Length       Precision       Scale       Primary Key       Nullable       Default       Comm         COURSE       Column       Data Type       Length       Precision       Scale       Primary Key       Nullable       Default       Comm         COURSE       C       CODE       Varchar2       15       -	User: KAVIT Home > SC	TA QL > <mark>SQL Con</mark> commit Disp	o <mark>mands</mark> play 10	•					Sa	ive	
Results Explain Describe Saved SQL History         Object Type TABLE Object COURSE         Table       Column       Data Type       Length       Precision       Scale       Primary Key       Nullable       Default       Comm         COURSE       Column       Data Type       Length       Precision       Scale       Primary Key       Nullable       Default       Comm         COURSE       C       CODE       Varchar2       15       -	User: KAVIT Home > SC I Autoc Desc Con	TA QL > <mark>SQL Con</mark> commit Disp unse	nmands play 10						Sa	ive	
Results Explain Describe Saved SQL History         Object Type TABLE Object COURSE         Table       Column       Data Type       Length       Precision       Scale       Primary Key       Nullable       Default       Comm         COURSE       C       CODE       Varchar2       15       -<	User: KAVIT Home > SO Ø Autoo Desc Con	TA QL > <mark>SQL Con</mark> commit Disp unse	nmands play 10	V					Sa	ive	and a second sec
Results Explain Describe Saved SQL History         Object Type TABLE Object COURSE         Table       Column       Data Type       Length       Precision       Scale       Primary Key       Nullable       Default       Comm         COURSE       C       CODE       Varchar2       15       -<	User: KAVIT Home > SC Autoc Desc Con	ra QL> <mark>SQL Con</mark> commit Disp unse	nmands play 10	T					Sa	ive	and
Results Explain Describe Saved SQL History         Object Type TABLE Object COURSE         Table       Column       Data Type       Length       Precision       Scale       Primary Key       Nullable       Default       Comm         COURSE       C       CODE       Varchar2       15       -<	User: KAVIT Home > SC Ø Autoc Desc Cou	rA QL> <mark>SQL Com</mark> commit Disp unse	nmands play 10	▼.					Sa	ive	and the second se
Results Explain Describe Saved SQL History         Object Type TABLE Object COURSE         Table       Column       Data Type       Length       Precision       Scale       Primary Key       Nullable       Default       Comm         COURSE       C       CODE       Varchar2       15       -<	User: KAVIT Home > SC I Autoc Desc Cou	rA QL> <mark>SQL Com</mark> commit Disp unse	omands play 10	•					Sa	ive	Laryound
Results Explain Describe Saved SQL History         Object Type TABLE Object COURSE         Table       Column       Data Type       Length       Precision       Scale       Primary Key       Nullable       Default       Comm         COURSE       Column       Data Type       Length       Precision       Scale       Primary Key       Nullable       Default       Comm         COURSE       C       CODE       Varchar2       15       -	User: KAVIT Home > SC I Autoc Desc Cou	ra DL > <mark>SQL Com</mark> commit Disp unse	omands play 10	•					Sa	ive	
Results Explain Describe Saved SQL History         Object Type TABLE Object COURSE         Table       Column       Data Type       Length       Precision       Scale       Primary Key       Nullable       Default       Comm         COURSE       C       CODE       Varchar2       15       -<	User: KAVIT Home > SC Autoc Desc Cou	TA DL > <mark>SQL Com</mark> commit Disp urse	omands play 10	•					Sa	ive	
Results Explain Describe Saved SQL History         Object Type TABLE Object COURSE         Table       Column       Data Type       Length       Precision       Scale       Primary Key       Nullable       Default       Comm         COURSE       Cocope       Varchar2       15       -	User: KAVIT Home > SC I Autoc Desc Cou	TA DL > <mark>SQL Com</mark> commit Disj urse	olay 10	T					Sa	ive	
Object Type TABLE Object COURSE           Table         Column         Data Type         Length         Precision         Scale         Primary Key         Nullable         Default         Comm           COURSE         C         CODE         Varchar2         15         - <td>User: KAVIT Home &gt; SC Autoc Desc Cou</td> <td>TA DL&gt; <mark>SQL Com</mark> commit Disj urse</td> <td>olay 10</td> <td>•</td> <td></td> <td></td> <td></td> <td></td> <td>Sa</td> <td>ive</td> <td></td>	User: KAVIT Home > SC Autoc Desc Cou	TA DL> <mark>SQL Com</mark> commit Disj urse	olay 10	•					Sa	ive	
Table         Column         Data Type         Length         Precision         Scale         Primary Key         Nullable         Default         Comm           COURSE         C. CODE         Varchar2         15         -	User: KAVIT Home > SC Pesc Cou Results	TA Commit Disp unse Explain De	nmands play 10 scribe Saveo	▼ I SQL His	tory				Sa	ive	
COURSE         C CODE         Varchar2         15         -	User: KAVIT Home > SC Desc Cou   Results Object Ty	Explain De	nmands play 10 scribe Saved Object COUF	T SQL His	tory				Sa	ive	
C NAME Varchar2 15	User: KAVIT Home > SC Desc Cou   Results Object Ty Table	Explain De Column	scribe Saved Digit COUF	▼ I SQL His SSE Length	tory	Scale	Primary Key	Nullable	Default	IVE	e
DUDATION Number 0 2	User: KAVIT Home > SC Desc Cou Course Object Ty Table COURSE	Explain De Column Column Column <u>C CODE</u>	scribe Saved Digit COUF Data Type Varchar2	▼ I SQL His SE Length 15	tory Precision	Scale	Primary Key	Nullable	Default	Comm	e
	User: KAVIT Home > SC Desc Cou Course Object Ty Table COURSE	Explain De Column Column Column Column Column Column Column	scribe Saved Digit COUF Data Type Varchar2 Varchar2	▼ I SQL His SE Length 15 15	tory Precision	Scale -	Primary Key	Nullable ✓	Default -	Comm -	e
	ts t t t t t t t t t t t t t t	Explain De pre TABLE ( Column C CODE C.NAME DURATION FEE	scribe Saver Digital To Digital COUF Data Type Varchar2 Varchar2 Number Number	V I SQL His RSE Length 15 15 -	tory Precision - - 8 10	Scale - - 0 2	Primary Key	Nullable v v v v v	Default - -	Comm - -	
1-4	User: KAVIT Home > SC Desc Cou Course Object Ty Table COURSE	Explain De Column Column Column Column Column Column Column Column DURATION FEE	scribe Saved Digit COUF Data Type Varchar2 Varchar2 Number Number	V I SQL His SSE Length 15 15 - -	tory Precision - - 8 10	Scale - - 2	Primary Key	Nullable V V V V	Default - - -	Comm - - -	e

Self-Instructional Material

I

## Modifying a Table

SQL provides an ALTER command to modify a table structure. It is a Data Definition Language (DDL) command. Following are the few examples to modify a table structure.

## Add a New Column

## Syntax:

Alter table <table\_name >

ADD (column\_name data type(length), column\_name data type(length), ...);

**Example 7:** Write a query to add new column (mobile Number (10)) in table *student*.



RDBMS Lab

## NOTES

Self-Instructional Material RDBMS Lab

NOTES

You can see the new structure of *student* table as shown below.

Home > SQL > SQL Commands

OLL NO AME DDRESS CODE IOBILE	Varchar2 Varchar2 Varchar2 Varchar2 Varchar2 Number	10 20 30 8 -	- - - 10	- - -	2 * •	~ ~ ~ ~	- -	• •
AME DDRESS CODE IOBILE	Varchar2 Varchar2 Varchar2 Number	20 30 8 -	- - - 10	-	-	~	•	•
DORESS CODE IOBILE	Varchar2 Varchar2 Number	30 8 -	- - 10	-	-	~	-	··· • ··
Data	varchar2 Number	-	10	**	-			
Data	ANING	92	19	0		~		
Data					2	v	1958	57. 1. juli
E Dat	tabase Exp	ress Edit	ion					U
							Home	Logout
SQL Com	mands							
nit Displ	lay 10	•					Sav	e
course	modify c_c	ode char	(15);					
1	tabl n da 8: W1 5). © Dat it Disp	table <tab. n data type 8: Write a que 5). Control Database Exp Course modify c_c</tab. 	table <table_nam n data type (ler 8: Write a query to cl 5). Commands it Display 10 course modify c_code char</table_nam 	<pre>table <table_name> mod: n data type (length), 8: Write a query to change the 5). © Database Express Edition Course modify c_code char (15);</table_name></pre>	<pre>table <table_name> modify ( n data type (length),); 8: Write a query to change the data to 5).</table_name></pre>	<pre>table <table_name> modify (column da n data type (length),); 8: Write a query to change the data type of colu 5). © Database Express Edition Course modify c_code char (15);</table_name></pre>	<pre>table <table_name> modify (column data typ n data type (length),); 8: Write a query to change the data type of column c_c 5). © Database Express Edition Course modify c_code char (15);]</table_name></pre>	<pre>table <table_name> modify (column data type (ler n data type (length),); 8: Write a query to change the data type of column c_code fro 5). © Database Express Edition CL Commands it Display 10 • Sav course modify c_code char (15);]</table_name></pre>

Self-Instructional 10 Material

## Modify the Length of an Existing Column

## Syntax:

Alter table <table\_name> modify (column data type (length),

column data type (length),...);

**Example 9:** Write a query to change the length of columns **name varchar (20)**, **address Varchar(40)** in table *student*.

# Home > SQL > SQL Commands Alter table student modify (name yercher (20), address Yercher (40));] Results Explain Describe Saved SQL History Table altered. After altering student table structure will look like as shown below: Home > SQL > SQL Commande Mutocommit Display 10 • desc student

Results Explain Describe Saved SQL History Object Type TABLE Object STUDENT Table Column Data Type Length Precision Scale Primary Key Nullable Default Comment STUDENT ROLL NO Varchar2 10 -~ NAME Varchar2 20 40 ADDRESS Varchar2 ~ C CODE Varchar2 8 10 0 MOBILE Number -1-5

## Important points to Remember

• If table column contains the values, then the length of column could be increase.

NOTES

RDBMS Lab

Self-Instructional Material

11

## • To change the data type column should be empty.

• To decrease the size of data type column should be empty.

## **Delete** any Column

Syntax:

## NOTES

RDBMS Lab

Alter table drop column column name;

Example 10: Write a query to drop column mobile in table student.

ORACLE Database Express Edition	Home Logout Hels
User: KAVITA	
Home > SQL > <mark>SQL Commands</mark>	
Autocommit Display 10	Save
Alter table student drop column mobile;	

Table dropped.

## **Data Constraints**

It is very important that whatever you store into your tables is as per the need of your organization. No false or incorrect data is stored by the user even intentionally or accidentally. Constraints are the restriction that you could put on your data to maintain data integrity. For example employer's salary should not be negative value, two students should not have the same enrollment number etc. The constraints helps in maintaining data integrity. Constraints could be specified when a table is created or even after the table is created with the ALTER TABLE command.

Oracle provides various types constraints as listed below:

- Primary Key
- Foreign Key or Reference Key
- Not Null
- Unique
- Check
- Default

Self-Instructional 12 Material Constraint could be defined at column level or at the table level. The only difference between these two is the syntax of these two.

Note: Drop all table created previously in this manual.

## Not Null constraint

In database, NULL is a special value that is different from zero, space or blank. It represents an unknown value for the column. The NOT NULL constraint ensures that the value in column is not missing (NULL). This constraint enforce user to enter data into a specified column. A column with this constraint could have duplicate values but could not be NULL or empty.

You must have created your e-mail ID. When you create an e-mail ID, it is mandatory to fill certain entries (the field with \*), these fields are the fields with the not null constraint.

**Example 11:** Create a table book with the NOT NULL constraint with the structure as shown below.

Column Name	Data Type	Size	Constraint
B_Code	varchar2	15	
Title	varchar2	40	NOT NULL
Author	varchar2	15	NOT NULL
Price	number	7,2	

The SQL command to create table with NOT NULL constraint is given in window shown below.

Autocommit Display 10 •	Save Run
Create Table Book ( B code <u>varchar</u> (15), Title <u>varchar</u> (40) NOT NULL, Author <u>varchar2</u> (15) NOT NULL, Price number (7,2) );	

Table created.

NOTES

RDBMS Lab

Self-Instructional Material

13

#### RDBMS Lab

The structure of table *book* is given below.

#### Home > SQL > SQL Commands Autocommit Display 10 Save Run Desc Book Results Explain Describe Saved SQL History Object Type TABLE Object BOOK Table Column Data Type Length Precision Scale Primary Key Nullable Default Comment Varchar2 15 BOOK B CODE ~ 40 TITLE Varchar2 AUTHOR Varchar2 15 Number ~ 1 - 4

The above SQL command will create a table book where Title and Author have NOT NULL constraints. These constraints would make it sure that both the columns have some values during inserting and updating of data to these columns.

*Note: NOT NULL constraints can be set at column level only.* 

## Unique Constraint

Sometimes, it is required that column must have unique values only. The unique constraint ensures that data to the specified column data is not duplicate but it could contain the NULL values. Let us take an example of contact number and e-mail ID; it is not necessary that every student has a contact number and an e-mail ID, if they have that will be unique only.

**Example 12:** Create a table *student* with the UNIQUE constraint with the structure as shown below.

Column Name	Data Type	Size	Constraint
Roll_No	Varchar	10	
Name	Varchar	10	
Address	Varchar	35	
E_Mail	Varchar	20	Unique
Mobile	Number	10	Unique

## NOTES

Self-Instructional 14 Material The SQL command to create table with Unique constraint is given in window shown below.

RDBMS Lab

Autocommit	Display 10 V	Save Ru
reate Table : ( Roll_) );	Student No <u>varchar</u> (10), Name <u>varchar</u> (10), Address <u>varchar</u> (35), E_Mail <u>varchar</u> (30), Mobile Number (10), Unique (E_Mail), Unique (Mobile)	
	- Describe Secol COL History	

## **Primary Key Constraint**

A primary key constraint is used to uniquely identify each and every record in a table. A primary key has properties of unique and not null constraints.

A primary key constraint has the following properties:

- A primary key column allows unique values only.
- It does not allow NULL value in column.
- A primary key column could be used for a reference in another table (child table).

**Example 13:** Create a table *course* having the Primary Key constraint with the structure as shown below.

Column Name	Data Type	Size	Constraint
c_code	varchar2	15	Primary Key
c_name	varchar2	15	
duration	number	8	
fee	number	10,2	

## NOTES

The SQL command to create table with Primary Key constraint is given in window shown below.

NOTES

RDBMS Lab

Autocommit Display	10 🔻	Save Run
Create Table Course ( C_code C_name Duration Fee );	varchar (15) Primary key, varchar (15), number (8), number (10,2)	
Results Explain Describ	e Saved SQL History	

Table created.

Harman FOL N FOL

The query in window will create table course which contains a primary key field course code. Here primary key constraint will enforce the end user to enter unique and not null values only.

**Example 14:** Create a table book with the Primary Key constraint with the structure as shown below.

Column Name	Data Type	Size	Constraint
B_Code	varchar2	15	Primary Key
Title	varchar2	40	
Author	varchar2	15	
Price	number	7,2	

The SQL command to create table with Primary Key constraint is as follows:



Table created.

Note: A table can have only one primary key.

Self-Instructional 16 Material

## Foreign Key Constraint or Reference Key Constraint

A foreign key column in a table derived values from a primary key of another table that helps in establishing relationship between tables.

A table having primary key column is called a Master Table or a parent table and a table with the reference key is known as a Transaction Table or a child table.

C\_code and B\_code are the primary key of the tables *course* and *book* respectively. These columns can be used to as a reference key in another table.

#### **Important Points to Remember**

- Reference key column in a table must have the same data type be as specified in primary key column in another table.
- Size of data type must be the same or more as defined in a primary key column.
- Name of reference key column could be same or different as defined in primary key column.
- A table may contain more than one reference keys.
- Reference keys column values could be duplicate or not NULL.
- Reference keys column can have the same values as stored in primary key column.

Suppose that students can enrolled in the course which are offered by the university. Course table contains the detail of all the courses offered by the university, so C\_code column in *student* table must have reference of C\_code column of *course* table.

**Example 15:** Create a table student with the Reference Key constraint with the structure as shown below.

Column Name	Data Type	Size	Constraint
Roll_No	Varchar	10	
Name	Varchar	10	
Address	Varchar	35	
C_code	Varchar	15	Reference Key

The SQL command to create table with REFERENCE KEY constraint is as follows:

RDBMS Lab

## NOTES

Self-Instructional Material

## Note: drop student table

NO	ГES

Autocommit Display 10 T		Save	Ru
op Table Student			
Results Explain Describe Saved SQL History			
able dropped. Now create Student table again with refere	nce key as shown be	elow:	
able dropped. Now create Student table again with refere	nce key as shown be	elow:	
able dropped. Now create Student table again with refere	nce key as shown be	elow:	Run
able dropped. Now create Student table again with refere tome > SQL > SQL Commands Autocommit Display 10 • Create Table Student ( Roll_No varchar (10), Name varchar (10), Address varchar (35), C_code varchar (15) references course (C_code) );	nce key as shown be	ave	Run

The above command will create table *student* which contains a reference key column course code. This column will create reference of course code of *course* table, when record in *student* table will be inserted or updated by the user.

*Note:* A table can have more than one reference keys.

## **Check Constraint**

A check constraint enforce user to enter data as specified condition. For example marks in any subject should be between the ranges 0 to 100, fee should not be negative, book code must start with 'B', and book price should be between the ranges 1 to 15000 and employee HRA could not be more than 40% of basic salary and so on.

**Example 16:** Create a table book with the Check constraint with the structure as shown below:

Column Name	Data Type	Size	Constraint
B_Code	varchar2	15	Check
Title	varchar2	40	
Author	varchar2	15	
Price	number	7,2	Check

## NOTES

RDBMS Lab

*Note: drop table book created earlier.* 

The SQL command to create table with Check constraint is given in window shown below.

Autocom	mit Display 10 🔻	Save	Run
Create Tab ( B_ Ti Au Pr );	le Book code <u>varchar</u> (15) check ( B_code like 'B%') , tle <u>varchar</u> (40). thor <u>varchar2</u> (15), ice number (7,2) check ( Price >1 and price< = 15000)		
Doculte Ex	nian Describe Saved SOL Liston		

Table created.

## **Default Constraint**

Sometimes, the value of any column for every new record is same. To maintain the status of book in a library either it is available to issue or not, you must keep the status of book as 'T' (available) or 'F' (Issued). Every new book purchased for library, the status of book is required to be 'T'. Default value concept is suitable for these types of situations.

**Example 17:** Create a table book with the Default constraint with the structure as shown below.

Column Name	Data Type	Size	Constraint
B_Code	varchar2	15	
Title	varchar2	40	
Author	varchar2	15	
Price	number	7,2	
Status	Char	1	Default

The SQL command to create table with Default constraint is given in window shown below.

Home > SQL > SQL > SQL Command

NOTES

RDBMS Lab

Autocommit Display 10 🔹	Save Run
Create Table Book ( B_code varchar (15), Title varchar (40), Author varchar2 (15), Price number (7,2), Status char (1) default 'T' );	
Results Explain Describe Saved SQL History	

Table created.

**Example 18:** Create a table student with multiple constraints having the structure as shown below:

Column Name	Data Type	Size	Constraint
Roll_No	Varchar	10	Primary Key
Name	Varchar	10	Not Null
Address	Varchar	35	
C_code	Varchar	15	Reference Key
Mobile	Number	10	Unique

Note: drop student table then create student table again

The SQL command to create table as specified above is shown below:



20 Self-Instructional Material

## Data Manipulation Language (DML)

Data Manipulation Language (DML) commands are used to insert, manipulate and access data. The data manipulation language statements are Insert, Delete, and Update.

## **Insert Records in a Table**

## Syntax:

Insert into values (value1, value2, ...); Example 19: Insert (course code – PG001, course name- MCA, duration- 3, fee-32000) in the *course* table.

Iome > SQL > SQL Commands		
Autocommit Display 10 •	Save	Run
Insert into course values (' <u>PG001</u> ',' <u>MCA</u> ',3,	32000.00)	
Results Explain Describe Saved SQL History Dutput:		
Results Explain Describe Saved SQL History		
1 row(s) inserted.		
0.08 seconds		
	Application Express 2	2.1.0.00.39

After executing the above command system will prompt a message **1 row** inserted.

*Note:* All char, varchar and date values should be enclosed in single quotes (') for example 'MCA', '07-Sept-09', 'A-08-02', ...

RDBMS Lab

## NOTES

Self-Instructional Material

#### RDBMS Lab

## NOTES

## Try yourself:

1. Insert into course values ('PG003','M Sc-IT',3,32000.00)

2. Insert into course values ('PG002', 'MBA', 2,40000.00)

3. Insert into course values ('UG002', 'B SC-IT', 3,25000.00)

4. Add five records in *course* table.

5. Create a new table *Book* with the following fields and data types.

Field Name	Data Type	Size
B_Code	varchar	15
Title	varchar	30
Author	varchar	15
Price	Number	6,2
6. View the structure of <i>Boo</i>	<i>k</i> table.	
7. Add five records in <i>Book</i>	table.	

## **Insert Data into Specific Fields**

In the insert command shown above, it is necessary to insert data in all the fields in the same sequence as defined in the table. But sometimes, few fields are required to update later on. For example, student's subjects marks are inserted in the table and total, percentage or grade is required to calculate later on.

Syntax: (to insert data into selected fields only)

Insert into (column1, column2, ...)

values (value1, value2, ...);

**Example 20:** Write a query to insert (roll\_no='A-08-20', name='John', address= 'delhi') in the *student* table.





## **Insert Data with User Interaction**

If hundreds or thousands of records are to be inserted in a table, it will be very tedious job to do it with the constant values. The other ways to insert records into table is take input from the user and repeat the command.

**Example 21:** Insert into course values ('&C\_code', '&C\_name', &duration, &fee);



The same command cab be repeated to insert more records by putting / and pressing enter key at SQL prompt.

You can also insert records interactively into specific fields as shown below. **Example 22:** Insert into student (roll\_no, name, address) values ('&roll\_no', '&name', '&address');





*Note:* The & symbol would prompt user to input data to the various variable. The variable name that is written after & is not required to the same as field names. NOTES

RDBMS Lab

Self-Instructional Material

23

#### RDBMS Lab

## Try yourself:

## NOTES

table.	C	
C_code	C_name	Duration
UG001	BCA	3
UG002	B Sc-IT	3
PG003	M Sc-IT	2
2. Add 10 reco	ords into studer	nt table with the user interaction.
3. Add data int	ob code, title	, and author fields of book table with the user

1. Add the following data into C code, C name and duration fields of Course

## **Display Table Records**

interaction.

Select command is used to display the records in the table. All the fields and records could be displayed or only selective records and fields could be retrieved.

## To view all the Records

To retrieve all the recoeds use "\*" as shown below:

Syntax:

Select \* from ;

Example 23: Write a query to display all the records in the *course* table.



## **To View Selected Columns**

To view only selective columns, enter column names separated by comma (,) as shown below:

Syntax:

Select field1, field2, ....from ;

**Example 24:** Write a query to display the column c\_*name* and *fee* in the *course* table.

	(C Commundo			
Autocommit	Display 10		Save	Run
Select c_name	:, fee from	course		
Results Expla	in Describe S	aved SQL History		
C_NAME FI	E			
M Sc-CS 32	000			
BCA 29	000			
	000			
B SC-IT 25	000			
B SC-IT 25 MBA 40				
B SC-IT         25           MBA         40           M Sc-IT         32	000			

## **Update Table Records**

Update command is used to change or update the records in a table. For example, the contact no. or address of any person has been changed or course fee is changed by the university.

#### Syntax:

## NOTES

RDBMS Lab

Self-Instructional Material

RDBMS Lab	<b>Example 25:</b> Write a query to update fee=32000 having course code 'UG001'
	in the <i>course</i> table.

	Save Run
Update course set fee=32000 where c_cod	e= <mark>'UG001</mark> ';
Results Explain Describe Saved SQL Histor	у
1 row(s) updated.	
1 row(s) updated. D.19 seconds	
1 row(s) updated. 0.19 seconds	
1 row(s) updated. 0.19 seconds	Application Express 2.1.0.00.3

**Where** clause is used to specify the condition for which this fee should be changed. Without any condition all the records will be updated with the new fee Rs. 32,000.

More than one columns can also be updated by specifying multiple columns and there new values after set keyword.

NOTES

**Example 26:** Write a query to change address to madras and course code to 'PG001' having roll no= 'A-08-20' in the *student* table.

Autocommit Display 10	Save	Run
Jpdate student set ADDRESS='Madras', C_CODE=' <u>PG</u>	01' where ROLL_NO='A-0	98-20°;
		;
Results Explain Describe Saved SQL History		
Results Explain Describe Saved SQL History		

## NOTES

RDBMS Lab

## Try yourself:

- 1. Display name and c\_code of students.
- 2. Change the address from Madras to Delhi of student whose roll number is A-08-20.
- 3. Change the fee from Rs. 32000 to Rs. 38000 of course where c\_code is PG001.

## **Delete Records**

Delete command is used to delete records from the table. One or more or all the records can be deleted from the table depending upon the where condition.

## Syntax:

Self-Instructional Material

27

RDBMS Lab **Example 27:** Write a query to delete a record from the *course* table where course code is 'PG002'.

- norocommit bishidy to .	Save
Delete from course where c_code - 'PG00	2*;
Results Explain Describe Saved SQL Histor	у
row(s) deleted	
0.06 seconds	
	Application Express 2.1
Language: en-us	Application Express 2.1 Copyright © 1999, 2008, Oracle. All rights r
Language: en-us	Application Express 2.1 Copyright © 1999, 2006, Oracle. All rights r
Language: en-us To delete all the records from a ta	Application Express 2.1 Copyright © 1999, 2008, Oracle. All rights r able, you can write the delete con
Language: en-us To delete all the records from a ta without where clause as given below:	Application Express 2 1 Copyright © 1999, 2008, Oracle. All rights r uble, you can write the delete con
To delete all the records from a ta without where clause as given below: Delete from course;	Application Express 2.1 Copyright © 1999, 2008, Oracle. All rights r able, you can write the delete con
To delete all the records from a ta without where clause as given below: Delete from course;	Application Express 2.1 Copyright © 1999, 2008, Oracle. All rights r able, you can write the delete con
To delete all the records from a ta without where clause as given below: Delete from course; Or	Application Express 2.1 Copyright © 1999, 2008. Oracle. All rights r able, you can write the delete con
To delete all the records from a ta without where clause as given below: Delete from course; Or Delete course;	Application Express 2.1 Copyright © 1999, 2008, Oracle. All rights r uble, you can write the delete con
Language: en-us To delete all the records from a ta without where clause as given below: Delete from course; Or Delete course; Home > SQL > SQL Commands	Application Express 2.1 Copyright © 1999, 2008, Oracle. All rights r able, you can write the delete con
To delete all the records from a ta without where clause as given below: Delete from course; Or Delete course; Home > SQL > SQL Commands Autocommit Display 10	Application Express 2.1 Copyright © 1999, 2008. Oracle. All rights r uble, you can write the delete con
To delete all the records from a ta without where clause as given below: Delete from course; Or Delete course; Home > SQL > SQL Commands Pelete from course	Application Express 2 1 Copyright © 1999, 2006. Oracle. All rights r uble, you can write the delete con Save Rur
To delete all the records from a ta vithout where clause as given below: Delete from course; Or Delete course; Home > SQL > SQL Commands Autocommit Display 10 • Delete from course	Application Express 2.1 Copyright © 1999, 2008, Oracle. All rights r uble, you can write the delete con
To delete all the records from a ta vithout where clause as given below: Delete from course; Or Delete course; Home > SQL > SQL Commands Autocommit Display 10 • Delete from course	Application Express 2.1 Copyright © 1999, 2008, Orscle. All rights r uble, you can write the delete con
Language: en-us To delete all the records from a ta without where clause as given below: Delete from course; Or Delete course; Home > SQL > SQL Commands ✓ Autocommit Display 10 • Delete from course	Application Express 2.1 Copyright © 1999, 2008, Oracle. All rights r uble, you can write the delete con
To delete all the records from a ta without where clause as given below: Delete from course; Or Delete course; Home > SQL > SQL Commands Autocommit Display 10 • Delete from course	Application Express 2.1 Copyright © 1999, 2008, Oracle. All rights r uble, you can write the delete con save Rur
Language: en-us To delete all the records from a ta without where clause as given below: Delete from course; Or Delete course; Home > SQL > SQL Commands @Autocommit Display 10 Delete from course	Application Express 2 1 Copyright © 1999, 2006. Oracle. All rights r uble, you can write the delete con
To delete all the records from a ta without where clause as given below: Delete from course; Or Delete course; Home > SQL > SQL Commands @ Autocommit Display 10 • Delete from course	Application Express 2 1 Copyright © 1999, 2006. Oracle. All rights r uble, you can write the delete con
Language: en-us To delete all the records from a ta vithout where clause as given below: Delete from course; Or Delete course; Home > SQL > SQL Commands Velete from course	Application Express 2 1 Copyright © 1999, 2006. Oracle. All rights r uble, you can write the delete con
Language: en-us To delete all the records from a ta vithout where clause as given below: Delete from course; Or Delete course; Home > SQL > SQL Commands Autocommit Display 10 Delete from course	Application Express 2.1 Copyright © 1999, 2008, Oracle. All rights r uble, you can write the delete con Save Rur
Language: en-us To delete all the records from a ta vithout where clause as given below: Delete from course; Or Delete course; Home>SQL>SQL Commands  ✓ Autocommit Display 10  Delete from course	Application Express 2 Copyright © 1999, 2008, Oracle. All rights r uble, you can write the delete con Save Rur
<pre>anguage: en-us To delete all the records from a ta ithout where clause as given below: Delete from course; Or Delete course; Home&gt;SQL&gt;SQL Commands Velete from course </pre>	Application Express Copyright © 1999, 2006, Oracle, All right uble, you can write the delete co Save R

Self-Instructional Material

I

NOTES

The above command will delete all the records from the course table.

## View the Existing Tables

To view all the existing tables in database, you can use **Tab**. Tab is a view which displays the name and type of object such as table, view, or synonym.

Example 28: Write a query to display all the tables in the database.

## NOTES

RDBMS Lab

			Save	Run
Select * from tab				
Results Explain Describe Save	TABTYPE	CLUSTERID		
Results Explain Describe Save TNAME TEST	ed SQL Hist TABTYPE TABLE	CLUSTERID -		
Results Explain Describe Save TNAME TEST EMPLOYEE	ed SQL Hist TABTYPE TABLE TABLE	CLUSTERID -		
Results Explain Describe Save TNAME TEST EMPLOYEE EMPLOYEE1	TABTYPE TABLE TABLE TABLE TABLE	CLUSTERID - -		
Results Explain Describe Save TNAME TEST EMPLOYEE EMPLOYEE1 TEST_TABLE	TABTYPE TABLE TABLE TABLE TABLE TABLE	CLUSTERID - -		
Results Explain Describe Save TNAME TEST EMPLOYEE EMPLOYEE1 TEST_TABLE STUDENT	TABTYPE TABLE TABLE TABLE TABLE TABLE TABLE	CLUSTERID - - - -		
Results Explain Describe Save TNAME TEST EMPLOYEE EMPLOYEE1 TEST_TABLE STUDENT BINS30KzsysySJm1ecILRRZXXA==\$0	TABTYPE TABLE TABLE TABLE TABLE TABLE TABLE TABLE TABLE	CLUSTERID - - - - -		
Results Explain Describe Save TNAME TEST EMPLOYEE EMPLOYEE1 TEST_TABLE STUDENT BINS30KzsysySJm1ecILRRZXXA==\$0 BINSaoNjk9ifQ/mDmD5qp6GQ2Q==\$0	TABTYPE TABLE TABLE TABLE TABLE TABLE TABLE TABLE TABLE TABLE	CLUSTERID - - - - - -		
Results Explain Describe Save TNAME TEST EMPLOYEE EMPLOYEE1 TEST_TABLE STUDENT BIN\$30KzsysySJm1eclLRRZXXA==\$0 BIN\$aoNjk9ifQ/mDmD5qp6GQ2Q==\$0 BOOK	TABTYPE TABLE TABLE TABLE TABLE TABLE TABLE TABLE TABLE TABLE TABLE	CLUSTERID - - - - - - - -		
Results     Explain     Describe     Save       TNAME     TEST       EMPLOYEE       EMPLOYEE1       TEST_TABLE       STUDENT       BINS30KzsysySJm1ecILRRZXXA==\$0       BINSaoNjk9ifQ/mDmD5qp6GQ2Q==\$0       BOOK       BINSLIDRUI5JTeObJAvWXC3ubw==\$0	TABTYPE TABLE TABLE TABLE TABLE TABLE TABLE TABLE TABLE TABLE TABLE TABLE	CLUSTERID - - - - - - - -		

**TNAME** is a column which displays the object name as table, view, index, or synonym.

**TABTYPE** is a column which displays the type of object. The type of object may be any table, view, index, or synonym.

## **Filtering Records using Where Conditions**

A university can have thousands of records but all these records are not required to view every time. Many users might need to view different records from the same table at different time. To filter various records of table, **where** clause can be used with conditional, logical and other operators.

#### Syntax:

Select \* from [where <condition>];

Self-Instructional Material

29

The following is the *course* table contains 8 records. Let us filter records from this table with different conditions.

NOTES

RDBMS Lab

C_NAME	DURATION	FEE
MCA	3	55000
M Sc-CS	2	50000
BCA	3	32000
B Sc-IT	3	25000
M Sc-IT	2	48000
B Tech-CS	4	60000
B Tech-EC	4	64000
B Tech-IT	4	58000
	C_NAME MCA M Sc-CS BCA B Sc-IT M Sc-IT B Tech-CS B Tech-EC B Tech-IT	C_NAMEDURATIONMCA3M Sc-CS2BCA3B Sc-IT3M Sc-IT2B Tech-CS4B Tech-EC4B Tech-IT4

## **Conditional Operators in SQL**

## Equal to (=)

10

E.

To see the detail of course where course code equal to PG003 then the query will be:

Select \* from course where c\_code='PG003';

Output of the above query is shown below:

Autoco	mmit Displ	ay 10 🔹		Save	Run
Select *	from cours	e where c_c	ode='PG003'		
Results E	C_NAME	Cribe Saved S	SQL History		
Results E C_CODE PG003	C_NAME M Sc-IT	cribe Saved S DURATION 3	SQL History FEE 32000		
Results E C_CODE PG003 1 rows retur	C_NAME M Sc-IT rned in 0.00 s	DURATION 3 seconds	SQL History FEE 32000 CSV Export		

Self-Instructional 30 Material

## Not Equal to (<>, !=)

To see the detail of course where course duration is not 4 years then the query will be:

Select \* from course where duration <> 4;

Output of the above query is shown below:

## NOTES

RDBMS Lab

elect * from co	urse where dur	ation /> /:	
		ación (7 4,	
Results Explain [	Describe Saved	SQL History	
C_CODE C_NAM	E DURATION	FEE	
C_CODE C_NAM PG007 M Sc-CS UG001 BCA	E DURATION	FEE 32000 32000	
C_CODE C_NAM PG007 M Sc-CS UG001 BCA UG002 B SC-IT	IE DURATION 3 3 3	FEE 32000 32000 25000	
C_CODE C_NAN PG007 M Sc-CS UG001 BCA UG002 B SC-IT PG003 M Sc-IT	IE DURATION 3 3 3 3 3	FEE 32000 32000 25000 32000	

## Greater Than (>)

To see the detail of course where course fee is greater than Rs. 50000 then the query will be:

Select \* from course where fee >50000;

Self-Instructional Material RDBMS Lab

**NOTES** 

Output of the above query is shown below:

M Autoco	ommit Displ	lay 10 🔹		Save	Run
Select *	from cours	se w <mark>h</mark> ere fea	e >=∃0000		
Results	Explain Des	cribe Saved S	QL History		S
Results C_CODE	Explain Des	cribe Saved S	GQL History		
Results C_CODE PG007	Explain Des C_NAME M Sc-CS	cribe Saved S DURATION 3	GQL History FEE 32000		
Results C_CODE PG007 UG001	Explain Des C_NAME M Sc-CS BCA	cribe Saved S DURATION 3 3	QL History FEE 32000 32000		
Results C_CODE PG007 UG001 PG003	Explain Des C_NAME M Sc-CS BCA M Sc-IT	cribe Saved S DURATION 3 3 3	GL History FEE 32000 32000 32000		

Similar to operators, equal to, not equal to and greater than operators are used to filer records. Other operators like less than, less than equal to, greater than equal to can be used.

## **Other Operators in SQL**

## BETWEEN

The BETWEEN operator filters the records between a given range. Suppose you want to filter the courses where fee is in between Rs. 45000 to Rs. 58000. The query to retrieve such records is given below:

Select  $\star$  from course where fee between 45000 and 58000

Self-Instructional 32 Material
Output of the above query is shown below:

#### RDBMS Lab

NOTES

Autocomm	it Displa	sy 10	•		Save	Run
elect * fr	om cours	e where fe	e between 450	00 and 58000		
Results Exp	ain Desc	ribe Saved	I SQL History			
Results Exp C_CODE C PG006 N	ain Desc _NAME IBA	ribe Saved DURATION 2	I SQL History			
Results Exp C_CODE C PG006 M rows returne	Iain Desc _NAME IBA d in 0.00 s	ribe Saved DURATION 2 econds	I SQL History			

The between operators can filter the numbers, text, or date values.

### NOT BETWEEN

The NOT BETWEEN operator filters the records where the data is not in between a given range.

Select  $\star$  from course where fee not between 45000 and 58000

Self-Instructional Material

Output of the above query is shown below:

Autocor	nmit Displ	lay 10 🔻				Save	Ru
Select * ·	from cours	se where fee	not betwee	n 45000 a	nd 58000		
Results E	xplain Des	cribe Saved S	SQL History				
Results E	xplain Des C_NAME	cribe Saved S DURATION	FEE				
Results E C_CODE PG002 PG007	xplain Des C_NAME MBA M Sc-CS	cribe Saved S DURATION 2 3	GQL History FEE 40000 32000				
Results         E           C_CODE         PG002           PG007         UG001	Xplain Des C_NAME MBA M Sc-CS BCA	cribe Saved S DURATION 2 3 3	SQL History FEE 40000 32000 32000				
Results         E           C_CODE         PG002           PG007         UG001           UG002         PG002	xplain Des C_NAME MBA M Sc-CS BCA B SC-IT	cribe Saved § DURATION 2 3 3 3 3	FEE 40000 32000 32000 25000				
Results         E           C_CODE         PG002           PG007         UG001           UG002         PG003	Xplain Des C_NAME MBA M Sc-CS BCA B SC-IT M Sc-IT	cribe Saved S DURATION 2 3 3 3 3 3 3	FEE         40000           32000         32000           25000         32000				

#### **Oracle Functions**

Oracle provides various built in functions for different purpose such as calculation, comparison and conversion of data. Functions may or may not have the arguments (input) and have the capability to return a value.

Basically there are two types of function:

- Aggregate Functions
- Scalar functions

#### **Aggregate Functions**

Aggregate functions work on a group of values (a column values) and returns a single value.

Few aggregate functions are listed below:

- SUM()
- MAX()
- MIN()
- AVG()
- COUNT()

34 Material

Self-Instructional

NOTES

RDBMS Lab

#### **Scalar functions**

SQL scalar functions return a single value, based on the input value.

Few scalar functions are listed below:

- MID()
- LEN()
- Upper()
- Lower()

Consider a table *course* with the following records:

C_CODE	C_NAME	DURATION	FEE
PG002	MBA	2	40000
PG006	MBA	2	50000
PG007	M Sc-CS	3	32000
UG001	BCA	3	32000
UG002	B SC-IT	3	25000
PG003	M Sc-IT	3	32000
PG001	MCA	3	32000

**Example 29:** Write a query to find the total fee received in MBA course.

Autocommit Display 10	•	Save	Run
elect sum (fee) from course	where C_NAME='MBA';		
			Z
coulte Evolain Describe Saver	Listory		
esults Explain Describe Saved	I SQL History		
Explain Describe Saved	I SQL History		
SUM(FEE) rows returned in 0.01 seconds	CSV Export		
esults Explain Describe Saved	<u>CSV Export</u>		
SUM(FEE) 20000 rows returned in 0.01 seconds	<u>CSV Export</u>		
Results Explain Describe Saved SUM(FEE) 90000 rows returned in 0.01 seconds	CSV Export		

NOTES

RDBMS Lab

Self-Instructional Material

# *RDBMS Lab* **Example 30:** Write a query to find the minimum fee received in MBA course from the *course* table.

	Save
elect min(fee) from course where C_NAME='	мва';
Results Explain Describe Saved SQL History	
MIN(FEE) 40000	
rows returned in 0.00 seconds <u>CSV Export</u>	

**Example 31:** Write a query to find the maximum fee received in MBA course from the *course* table.



Self-Instructional 36 Material

NOTES

**Example 32:** Write a query to count the number of records in *course* table where c name= 'MBA'.

Autocommit Display 10 •	Save	Run
elect count(*) from course where C_NAME='MBA'	;	
Results Explain Describe Saved SQL History		//
2		
rows returned in 0.00 seconds <u>CSV Export</u>		

NOTES

RDBMS Lab

**Example 33:** Write a query to converts the text (i.e. Computer) to uppercase.



Self-Instructional Material

### *RDBMS Lab* **Example 34:** Write a query to converts the text (i.e. COMPUTER) to lowercase.

Home > SQL > SQL > Commands

NOTES
-------

Autocommit Display 10 🔻	Save Run
Select Lower ('COMPUTER') from dual;	
Results Explain Describe Saved SQL History	
LOWER('COMPUTER') computer	
1 rows returned in 0.00 seconds <u>CSV Export</u>	

**Example 35:** Write a query to round the figure (i.e. 1.23456).

Select round (1.23456,2) from dual; Results Explain Describe Saved SQL History ROUND(1.23456,2) 1.23	Autocommit Display 10 •	Save Ru
Results Explain Describe Saved SQL History ROUND(1.23456,2) 1.23	Select round (1.23456,2) from dual;	



**Example 36:** Write a query to find the square root of 49.

•

#### NOTES

Save

Run

RDBMS Lab

# **Join Commands**

1 rows returned in 0.02 seconds

SQRT(49) 7

Home > SQL > SQL Commands

Autocommit Display 10

Select sqrt(49) from dual;

Results Explain Describe Saved SQL History

Table 1:

create table student1(rno number(10), name char(30), course char(30), fee number(10)); insert into student1 values(101, 'NAMAN', 'B.tech', 59000); insert into student1 values(102, 'AMAN', 'B.tech', 59000); insert into student1 values(102, 'SITA', 'BCA', 49000); insert into student1 values (105, 'GITA', 'MCA', 59000); select \* from student1

CSV Export

RNO	NAME	COURSE	FEE
101	NAMAN	B.tech	59000
102	AMAN	B.tech	59000
103	SITA	BCA	49000
105	GITA	MCA	59000

#### Table 2:

create table marks1(rno number(10), sub1 number(10), sub2 number(10), sub3 number(10), total number(10));

insert into marks1 values (101, 50, 40, 40, 130);

insert into marks1 values (103, 60, 40, 40, 140);

insert into marks1 values (105, 50, 40, 50, 140);

select \* from marks1

RNO	SUB1	SUB2	SUB3	TOTAL
101	50	40	40	130
103	60	40	40	140
105	50	40	50	140

Self-Instructional Material

#### **EQUIJOIN**

**Example 37:** Write a query to display roll no., name, sub1, sub2, sub3 and total form the table student1 and marks1 where student1.rno=marks1.rno.

#### NOTES

RDBMS Lab

Select student1.rno, name, sub1, sub2, sub3, total from student1,marks1 where student1.rno=marks1.rno;

#### **Output:**

RNO	NAME	SUB1	SUB2	SUB3	TOTAL
101	NAMAN	50	40	40	130
103	SITA	60	40	40	140
105	GITA	50	40	50	140

#### Left Outer Join

select student1.rno,name,sub1,sub2,sub3,total from student1 left outer join marks1 on student1.rno=marks1.rno; OR

select student1.rno,name,sub1,sub2,sub3,total from student1,marks1 where student1.rno=marks1.rno(+);

RNO	NAME	SUB1	SUB2	SUB3	TOTAL
101	NAMAN	50	40	40	130
103	SITA	60	40	40	140
105	GITA	50	40	50	140
102	AMAN	-	-	-	-

#### **Table Project:**

insert into project values(102, 'Railway', 'Manager'); insert into project values(106, 'AI', 'Coder');

select \* from project

RNO	PNAME	ROLE
102	Railway	Manager
106	AI	Coder

#### **Right Outer Join**

Select student1.rno, project.rno, name, pname from student1 right outer join project on student1.rno=project.rno; OR

Select student1.rno, name, sub1, sub2, sub3, total from student1, marks1 where student1.rno(+)=marks1.rno;

RNO	RNO	NAME	PNAME
102	102	AMAN	Railway
-	106	-	AI

#### **Full Outer Join**

Select student1.rno, project.rno, name, pname from student1
full outer join project on student1.rno= project.rno;

RNO	RNO	NAME	PNAME
102	102	AMAN	Railway
103	-	SITA	-
105	-	GITA	-
101	-	NAMAN	-
-	106	-	AI

#### Data Control Language (DCL)

Data Control Language are the commands that allow authorized database users to share the data with other users. The shared data can be accessed or manipulated by other users as per the permission granted.

The data manipulation language statements are GRANT and REVOKE

- **GRANT**-provides user's access privileges to database.
- **REVOKE**-withdraw user's access privileges given by the GRANT command.

#### **Oracle Transactions**

All the changes made through DML commands are known as transaction. A transaction is a logical group of work. Transactions that you do on a database temporarily stored on the client machine that can be make permanent or canceled by the user. Oracle provides few commands to control the transactions as given below:

- Commit
- Savepoint
- Rollback

#### Commit

The commit command is used to make the transaction permanent to the database. The commit command ends the current transactions.

SQL > Commit;

#### Rollback

The rollback command is used to terminate the current transaction. All the changes made to the rollback database can be undone by rollback. It is generally used when a session disconnects from the database without completing the current transaction.

SQL > rollback;

When rollback command is executed, Oracle prompts a message as shown below:

Rollback complete.

\* Rollback undone the whole transaction made after the last committed transaction.

Self-Instructional Material

NOTES

RDBMS Lab

## RDBMS Lab Index An index is a performance-tuning method of allowing faster retrieval of records. An index creates an entry for each value that appears in the indexed columns. By default, Oracle creates B-tree indexes. **NOTES** Syntax: The syntax for creating an index in Oracle/PLSQL is: CREATE [UNIQUE] INDEX index name ON table name (column1, column2, ... column n) [ COMPUTE STATISTICS ]; UNIQUE refers to the combination of values in the indexed columns must be unique, Compute Statistics tells Oracle to collect statistics during the creation of the index. The statistics are then used by the optimizer to choose a "plan of execution", when SQL statements are executed. Example 38: An example to create an index in Oracle/PLSQL. Create index employee idx ON employee (name); In this example, we've created an index on the employee table called employee idx. We can also create an index with more than one field as in the example below: CREATE INDEX student idx ON student (name); Home > SQL > SQL Commands Save Run Autocommit Display 10 ۲ CREATE INDEX student\_idx ON student(name); Results Explain Describe Saved SQL History Index created. 0.26 seconds

Application Express 2.1.0.00.39

We car	n also cho	ose to collect s	statist	tics up	pon creation of the inc	lex as follows:
CREATE	INDEX	student_i	idx	ON	student(name)	COMPUTE
STATIST	ICS;					

Autocom	mit Display	10 🔹	Save	Run
REATE IND	EX student <u>i</u> c	dx ON student(name) COMPU	UTE STATISTICS;	
Results Exp	plain Describe	Saved SQL History		3
.00 seconds				
			- 100 -	

### Syntax:

The syntax for renaming an index in Oracle/PLSQL is:

ALTER INDEX index\_name RENAME TO new\_index\_name; NOTES

RDBMS Lab

Self-Instructional Material

### *RDBMS Lab* **Example 39:** An example of how to rename an index in Oracle/PLSQL.

	Save
ALTER INDEX student <u>idx</u> RENAME TO student <u>idx</u>	_new;
Results Explain Describe Saved SQL History	
Statement processed. 0.14 seconds	Application Expre
Statement processed. 0.14 seconds <b>p an Index</b>	Application Expres
Statement processed. 0.14 seconds o an Index ax: The syntax for dropping an index in O	Application Expres
Statement processed. 0.14 seconds p an Index rax: The syntax for dropping an index in Or DROP INDEX index_name;	Application Expres



NOTES

RDBMS Lab

NOTES

### View

A view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

A view is a virtual table, which consists of a set of columns from one or more tables. It is similar to a table but it doesn't not store in the database. View is a query stored as an object.

#### Syntax:

```
CREATE VIEW view_name AS SELECT set of fields FROM relation name WHERE (Condition)
```

**Example 41:** Write a query to create a view Student\_view having fields roll number, name, mobile using table *student*.



Self-Instructional Material

## Display Records from View

**Example 42:** To display the records from view.

	Home > SQL > SQL Commands		
NOTES	Autocommit Display 10  select * from student_view ;	Save	Run
	Results       Explain       Describe       Saved SQL       History         ROLL_NO       NAME       MODILE       A-08-20       John       -         A-08-20       John       -       -       -       -         &a       &b       -       -       -       -         3 rows returned in 0.00 seconds       CSV Export       -       -		
Drop V Syntax Dro	View x: pp View View_name;	Application Express 2	: <u>1.0.00.39</u>
Exam	Home > SQL > SQL Commands		
	Autocommit Display 10  drop view student_view ;  Results Explain Describe Saved SQL History	Save	Run
	View dropped. 0.62 seconds		

RDBMS Lab

#### PL/SQL

PL/ SQL is also known as an embedded SQL and is a superset of SQL. PL/ SQL is an acronym of Procedural Language/Structure Query Language. It supports procedural features and SQL commands.

#### Structure of PL/ SQL Program

PL/SQL program block is divided in three sections.

- 1. Declaration section
- 2. Execution section
- 3. Exception handling section



#### **Declaration Section**

In declaration section, variables, constants, user defined exceptions, cursor and other objects are declared. This is an optional section. This section begins with the keyword DECLARE.

#### **Execution Section**

All the executable statements such as SQL statements, control statements, loops are written under this section. This is a mandatory section. This section begins with the keyword BEGIN and ends with the keyword END.

#### **The Exception Handling Section**

During program execution many abnormal situations may occur. To handle these situations, statements are written in this block. These situations are known as errors which occur due to the logical error, syntax error or system error. This is an optional section.

NOTES

RDBMS Lab

RDBMS Lab

Syntax: DECLARE declaration statements

NOTES

BEGIN executable\_statements

EXCEPTION

exception\_handling\_statements

... END ;

...

••

### PL/ SQL Engine

Oracle uses a PL/ SQL engine to processes the PL/ SQL statements. Either the PL/ SQL program is stored on the client side or on the server side. PL/ SQL engine is used by Oracle to execute the program statements.

#### Data Types in PL/ SQL

A program has many inputs and outputs in the form of variable and constant. These variable and constant specifies the storage format, type of value and a range of the values that can be stored. PL/ SQL provides various data types which are system defined and also gives the flexibility to the programmer to create their own data types.

#### **Classification of Data Types**

- Scalar Data Types
- Composite Data Types





#### Comments in PL/ SQL

In Oracle, comments may be introduced either for single line or for multiple lines.

- 1. /\*...\*/ is used for multiple line comments.
- 2. -- is used for single line comments.

The example for single line comment is given below :

-- This is a PL/ SQL program to calculate employee salary

### Variables in PL/ SQL

Variables are the identifiers of data type. These variables could be the identifiers of either system defined (scalar) data types or the identifiers of user defined (composite) data type i.e. record, table or Varray.

Variable declaration can be of any data type. For example:

```
Name char (30);
```

Salary Number (8, 2);

Date\_of\_join Date;

Constants can be of any data type. For example:

Pi constant number (3, 2) := 3.5;

Status Booleans := TRUE;

Pi and Status are assigned with a value during declaration, makes them constant.

Example 44: Write a PL/SQL program to display 'First PL/SQL Program'.

Autocommit Display 10 T	Save	Run
JECLARE		
BEGIN		
)BMS_OUTPUT.PUT_LINE ( 'First PL/SQL Program' END ;	; (	

Click on Run button to run program.

#### **Output:**

Results Explain Describe Saved SQL History

First PL/SQL Program

Statement processed.

Self-Instructional Material

49

RDBMS Lab

#### NOTES

*RDBMS Lab* **Example 45:** Write a PL/SQL program to display sum of two numbers given at run time.

Home > SQL > SQL Commands Autocommit Display 10 Save Run • DECLARE \* NUMBER (10) ; NUMBER (10) ; NUMBER (10) ; number\_1 number\_2 res BEGIN number\_1 := :number\_1 ; number\_1 := number\_1 ; number\_2 := number\_2 ; res := number\_1 + number\_2 ; DBMS\_OUTPUT\_PUT\_LINE ( 'Sum is ' || res ) ; END ; Results Explain Describe Saved SQL History

After running this program it will show input screen as shown below:

			Submi
:NUMBER	1		
NUMBER	_2		
inom DEn			

Enter values in text boxes and click on Submit button.

#### **Output:**

Results	Explain	Describe	Saved SQL	History		
5um is 1	10					
S <mark>tatemen</mark>	nt proce	ssed.				
0.00 seco	nds					
1.22	22.W2				Application	Express 2.1.0.00.39

Self-Instructional 50 Material

NOTES

Example 46: Write a PL/SQL Program to print Prime Number.

RDBMS Lab

Home > SQL > SQL > Commands

```
Autocommit Display 50
                             ۲
declare
n number;
i number;
        flag number;
begin
        i:=2;
        flag:=1;
        n:=:n;
        for i in 2...n/2
        loop
                if mod(n,i)=0
                then
                        flag:=0;
                        exit;
                end if;
        end loop;
        if flag=1
        then
                dbms_output.put_line('Number is Prime');
        else
                dbms_output.put_line('Number is not Prime');
        end if;
end;
```

### Input:

) Enter Bind Variables - Google Chrome	
127.0.0.1:8080/apex/f?p=4500:138:522288652727	8247:::
N13	Submit

NOTES

RDBMS Lab	Output:	
	Results Explain Describe Saved SQL History	
NOTES	Number is Prime Statement processed.	
	0.00 seconds	
		Application Express 2.1.0.00.

### **Example 47:** Write a PL/SQL Program to find factorial of a number given number.

Autocommit Display 10 •	Save	Run
declare		1
n number;		
<pre>fact number:=1;</pre>		
1 number;		
begin		
n:=:n;		
for i in 1n		
loop		
<pre>fact:=fact*i;</pre>		
end loop;		
<pre>dbms_output.put line('factorial='  fact);</pre>		
end;		

Enter SQL statement or PL/SQL command and click Run to see the results.

#### Input:



#### **Output:**

Results	Explain	Describe	Saved SQL	History
factoria	al=120			
Statemer	nt proces	ssed.		
0.00 seco	onds			
Languaga	00.115			Application Express 2.1.0.00.39

#### **Try Yourself:**

- 1. Write PL/SQL program to display demonstrate all sections of PL/SQL program.
- 2. Write PL/SQL program to display HELLO.

#### **Exception Handling**

In PL/ SQL, error is called as exception. Error may occur due to various reasons such as semantic error, hardware failure, system resources problems and many other reasons. Due to these errors program terminates abnormally.

#### **Types of Exception**

- 1. Internal exception
- 2. User-defined exceptions

#### Table: Internal Exceptions

Exceptions	Explanation
ZERO_DIVIDE	This exception raised when PL/SQL program attempts to divide a
	number by zero.
NO_DATA_FOUND	This exception raised when SELECT INTO statement returns no
	rows while expected to return.
CURSOR_ALREADY_OPEN	This exception raised when you try to open a cursor which is
	already.
INVALID_NUMBER	This exception raised when, the conversion of a string into a number
	fails because the string does not represent a valid number.
LOGIN_DENIED	This exception raised when PL/SQL program attempts to log on to
	Oracle with an invalid username and/or password.
NOT_LOGGED_ON	This exception raised when PL/SQL program issues a database call
	without being connected to Oracle.
STORAGE_ERROR	This exception raised when PL/SQL runs out of memory.
TOO_MANY_ROWS	A SELECT INTO statement returns more than one row while
	expected only one.
VALUE_ERROR	This exception raised when data type or data size is invalid.
PROGRAM_ERROR	This exception raised when PL/SQL has an internal problem.
OTHERS	This exception raised when error is unknown or not explicitly
	defined.

#### NOTES

RDBMS Lab

Self-Instructional Material

RDBMS Lab

Example 48: Write a program to demonstrate exception handling.

NOTES

Save Run Autocommit Display 10 . Declare b\_title varchar (40) ; Begin Select title into b\_title from book where title = 'DS' ; Exception when NO\_DATA\_FOUND then dbms\_output.put\_line ( 'No Record Found' ) ; when TOO MANY ROWS then dbms\_output.put\_line ( 'Query Returns More Than One Query' ) ; End ;

Query returns more than one records then TOO\_MANY\_ROWS exception:

Results Explain Describe Saved SQL History

Query Returns More Than One Query

Home > SQL > SQL Commands

Statement processed.

0.02 seconds

In the above program, select query is used to select book title into variable B title. Two internal exceptions are handled named NO DATA FOUND and TOO MANY ROWS. If query returns more than one records then TOO MANY ROWS exception would be raised by the system, if no record matches then NO DATA FOUND exception would be raised.

#### **User Defined Exceptions**

You can assign a name to unnamed system exceptions using a **Pragma** called **Exception Init as shown below:** 

Pragma Exception Init (exception name, Oracle error number);

In the above example, exception name is the user defined name of the exception that will be associated with Oracle error number.

#### Syntax:

. .

```
DECLARE
   exception name EXCEPTION ;
   PRAGMA EXCEPTION INIT (exception_name, Err_code);
BEGIN
   Executable statement;
```

Self-Instructional Material

```
EXCEPTION
WHEN exception_name THEN
Handle the exception
END;
```

Example 49: Write PL/SQL program to the given scenario given below:

Let's consider the student table and course tables.

The c\_code is a primary key in course table and c\_code is a foreign key in student table.

If you try to delete a c\_code from course table and it has a corresponding child records in student table an exception will be thrown with oracle code number -2292.



child\_record\_exception is a user defined name of exception in the above example. RAISE\_APPLICATION\_ERROR()

A user can assign an error message by using

Raise\_application\_error() to make the error message more descriptive for the end-user. It is a build-in procedure.

Example 50: Write a PL/SQL program to demonstrate User-defined Exceptions.

Other than the pre-defined exceptions, you can define your own exception to validate data against business requirements. For example, if user wants to update

#### NOTES

RDBMS Lab

Self-Instructional Material

total marks of student but subject marks are NULL, an error must be raised by the system to alert the user.

A user defined exceptions must be declared within declaration section by the keyword EXCEPTION and must be raised explicitly by RAISE statement within the executable section.

#### **Create Table Marks:**

Create table marks ( roll no number(3), subl number(3), sub2 number(3), sub3 number(3), total number(3) )

#### Insert values in roll no, sub1, sub2, sub3 fields only:

insert into marks (101,34,54,43)	(roll_no,	sub1	,sub2,	sub3)	values
insert into marks (102,54,54,50)	(roll_no,	sub1	,sub2,	sub3)	values
insert into marks (104,65,44,40)	(roll_no,	sub1	,sub2,	sub3)	values

Select \* from marks;

ROLL_NO	SUB1	SUB2	SUB3	TOTAL
101	34	54	43	-
102	54	54	50	-
104	65	44	40	-

### Home > SQL > SQL Commands

- / 10100	commit Display	10	•		Save	e Run
DECLARE						
	null_marks EX	CEPTION	;			
	rno Number	(3);				
	<pre>s1 Number (3)</pre>	;				
	<pre>s2 Number (3)</pre>	3				
	<pre>s3 Number (3)</pre>	;				
BEGIN						
	Select sub1,	sub2, s	ub3 into	<u>\$1, 52, 53</u> fr	rom marks where	roll_no
= 102 ;						
	If s1 is NULL	or s2	is NULL c	r <u>s3</u> is NULL	then	
	RAISE null_ma	rks;				
	End if ;					
	Update marks	set tot	al = <u>51</u> +	<u>s2 + s3</u> when	re ro <mark>ll_</mark> no = 102	2;
EXCEPTIO	DN					
	WHEN null mar	ks THEN		washing even the s		
12002	DBMS_OUTPUT.P	UT_LINE	( 'Subje	ct marks are M	UULL') ;	-
END ;						
Results	Explain Descrit	e Save	d SQL His	tory		
-						
1 row(s)	updated.					
0.02 seco	nds					

Self-Instructional Material

RDBMS Lab

NOTES

In the above example, **null\_marks** is a user defined exception which must be raised explicitly using RAISE statement. This exception would be raised, when marks in any subject would be NULL.

Check student's marks, after executing the above program:

select \*from marks;

ROLL_NO	SUB1	SUB2	SUB3	TOTAL
101	34	54	43	-
102	54	54	50	158
104	65	44	40	-

#### **Try Yourself:**

- 1. Write a PL/SQL code block that will accept an account number from the user and debit an amount of Rs. 2000 from the account, if the account has a minimum balance of 500 after the amount is debited.
- 2. Write a PL/SQL code block to calculate the area of the circle for a value of radius varying from 3 to 7. Store the radius and the corresponding values of calculated area in a table Areas.

Areas-radius, area.

- 3. Write a PL/SQL block of code for inverting a number 5639 or 9365.
- 4. Write a PL/SQL block of code to achieve the following: if the price of Product 'p00001' is less than 4000, then change the price to 4000. The Price changes to be recorded in the old\_price\_table along with Product\_no and the date on which the price was last changed. Tables involved: product\_master-product\_no, sell\_price.

Old\_price\_table-product\_no,date\_change,Old\_price

#### Cursor

Oracle allocates a memory known as the context area for the processing of the SQL statements. A cursor is a pointer or handle to the context area. Through the cursor, a PL/SQL program can control the context area and what happens to it as the statement is processed.

The three types of the cursors are:

- 1. Static cursors
- 2. Dynamic cursors
- 3. REF cursors

Static cursors are the ones whose select statements are known at the compile time. These are further classified into:

- Explicit cursors
- Implicit cursors

NOTES

RDBMS Lab

Self-Instructional Material

*RDBMS Lab* **Example 51:** Create a cursor to show roll number and total marks of students from *marks* table using cursor.

Home > SQL > SQL Commands

#### NOTES

	Save Run
DECLARE	
CURSOR cur_marks IS	
<pre>SELECT ROLL_NO, sub1, sub2, sub3, total</pre>	
FROM marks;	
BEGIN	
OPEN cur marks:	
LOOP	
FETCH cur_marks INTO rec_marks_detail	j
EXIT WHEN cur_marks%NOTFOUND;	
DBMS_OUTPUT.PUT_LINE('Marks Details :	'  ' '  rec_marks_detail.roll_no  '
'  rec_marks_detail.total);	
END LOOP;	
DDWG OUTDUT DUT I THE (IT the) work of	
DBMS_OUTPUT.PUT_LINE('Total number of	rows : '  cur_marks%ROWCOUNT);
DBMS_OUTPUT.PUT_LINE('Total number of CLOSE cur_marks; END:	rows : '  cur_marks% <u>ROWCOUNT</u> );
DBMS_OUTPUT.PUT_LINE('Total number of CLOSE cur_marks; END;	rows : '  cur_marks% <u>ROWCOUNT</u> );
DBMS_OUTPUT.PUT_LINE('Total number of CLOSE cur_marks; END; Results Explain Describe Saved SOL History	rows : '  cur_marks% <u>ROWCOUNT</u> );
DBMS_OUTPUT.PUT_LINE('Total number of CLOSE cur_marks; END; Results Explain Describe Saved SQL History	rows : '  cur_marks% <u>ROWCOUNT</u> );
DBMS_OUTPUT.PUT_LINE('Total number of CLOSE cur_marks; END; Results Explain Describe Saved SQL History	rows : '  cur_marks% <u>ROWCOUNT</u> );
DBMS_OUTPUT.PUT_LINE('Total number of CLOSE cur_marks; END; Results Explain Describe Saved SQL History Marks Details : 101 131 Marks Details : 102 158	rows : '  cur_marks% <u>ROWCOUNT</u> );
DBMS_OUTPUT.PUT_LINE('Total number of CLOSE cur_marks; END; Results Explain Describe Saved SQL History Marks Details : 101 131 Marks Details : 102 158 Marks Details : 104 149	rows : '  cur_marks% <u>ROWCOUNT</u> );
DBMS_OUTPUT.PUT_LINE('Total number of CLOSE cur_marks; END; Results Explain Describe Saved SQL History Marks Details : 101 131 Marks Details : 102 158 Marks Details : 104 149 Total number of rows : 3	rows : '  cur_marks% <u>ROWCOUNT</u> );
DBMS_OUTPUT.PUT_LINE('Total number of CLOSE cur_marks; END; Marks Details : 101 131 Marks Details : 102 158 Marks Details : 104 149 Total number of rows : 3 Statement processed.	rows : '  cur_marks% <u>ROWCOUNT</u> );

#### Trigger

A trigger is a PL/ SQL code block that runs automatically when an event occurs. An event in PL/ SQL is the data definition language such as INSERT, UPDATE or DELETE.

#### Uses of a Trigger

A database trigger helps in maintaining the organization's database in such a manner that without executing the PL/ SQL code explicitly, it update and validate the data. Triggers have the capabilities to provide a customized management system of your database.

Database trigger can be used to cater the following purposes:

- To enforce integrity constraints (e.g. check the referenced data to maintain referential integrity) across the clients in a distributed database
- To prevent generate invalid transactions in database.
- To update data automatically to one or more tables or views without user interaction

Automatically generate derived column values
• To customize complex security authorizations.
• To permit insert, update or delete operations to a associated table only during predetermined a date and time.
Provide auditing
Provide transparent event logging
• Helps in prompting information about various events taken on database, events of users, and SQL statements to subscribe applications.
Helps in maintaining replication of synchronous table
• Helps in gathering statistics on various table accesses.
Structure of PL/SQL Trigger
Syntax:
CREATE [OR REPLACE ]
TRIGGER <trigger_name></trigger_name>
BEFORE (or AFTER)
INSERT OR UPDATE [OF COLUMNS] OR DELETE
ON table_name
[FOR EACH ROW [WHEN (condition)]]
DECLARE.
Declaration statements
Dectatation statements
BEGIN
Executable statements
EXCEPTION
Exception handling statement
 END:
A database trigger can also have declarative and exception handling parts
How to A nnly a Trigger
A detabase trians has three sections nor alive trians retations at the section of
and a trigger restriction.

Three of Parts of Trigger are:

1. A Trigger Statement

- 2. A Trigger Body Action
- 3. A Trigger Restriction

Self-Instructional Material

RDBMS Lab

### NOTES

#### **Example 52:** To Create a Trigger.

A company XYZ has the employee detail in employee table. Company wants to have the history of all the employees who have left the organization. To store the employee history, a new table emp\_history is created with the same structure as employee table.

NOTES

RDBMS Lab

The structure of employee table is shown below:

Column Name	Data Type	Size
EMP_CODE	NUMBER	10
E_NAME	Varchar2	15
DESIGNATION	Varchar2	35
SALARY	NUMBER	10,2
DEPTNO	NUMBER	2

The employee table contains the following records:

EMP_CODE	E_NAME	DESIGNATION	SALARY	DEPTNO
7369	SMITH	CLERK	15000	20
7499	ALLEN	SALESMAN	35000	30
7521	WARD	SALESMAN	32000	30
7566	JONES	MANAGER	55000	20
7654	MARTIN	SALESMAN	30000	30
7698	BLAKE	MANAGER	60000	30
7782	CLARK	MANAGER	64000	10
7788	SCOTT	ANALYST	58000	20
7839	KING	PRESIDENT	70040	10
7844	TURNER	SALESMAN	30430	30
7876	ADAMS	CLERK	23000	20

Create a Duplicate Table of Employee

To maintain the employee history, a table emp\_history can be created with the SQL command given below:

Create table emp\_history as select \* from employee where emp\_code is null;

You can see the structure of new table emp\_history by giving command as written below:

Desc emp\_history;

Column Name	Data Type	Size
EMP_CODE	NUMBER	10
E_NAME	Varchar2	15
DESIGNATION	Varchar2	35
SALARY	NUMBER	10,2
DEPTNO	NUMBER	2

### NOTES

RDBMS Lab

Whenever any employee leaves the organization his or her detail will be deleted from the employee table and the same record should be inserted into emp\_history table. A trigger can be associated on table employee on the event delete.

The code for trigger is given below:

Autocommit	Display	10	•		Save	Run
Create or rep before Delete for each row DECLARE Declare t ENAME VARCHA DESIGNATION Y SALARY NUMBER DEPTNO NUMBER BEGIN Copy the da EMP_CODE:=:01 E_NAME :=:01d DESIGNATION:= SALARY:=:01d. DEPTNO:=:01d. insert the insert into g deptno);	he varial ER(10); R2(15); ARCHAR2( (10,2); (2); ta to be d. emp_cr. .E_NAME; :old. de: salary; deptno; delete rn mp_histor	gger emp oyee bles. 35); deleted ode; signatic ecord in ry value	f from employe on; to employee h s ( <u>emp_</u> code,	e table into va istory table e_name, designa	riables	γ,

In the above example, **emp\_history** is a trigger which is associated with the employee table. This is a trigger which should be fired with delete command on *employee* table and will store the deleted record in *emp\_history* table.

#### **Application:**

To test whether the trigger is fired and insert the deleted record in emp\_history table delete few records from employee table as shown below:

```
SQL> delete from employee where emp_code = 7782;
SQL> delete from employee where emp_code = 7876;
SQL> delete from employee where emp_code = 7844;
```

After executing the above queries, display all the records from the *emp history* table.

Select \*from emp\_history;

#### NOTES

RDBMS Lab

The above command would prompt the record as shown below:

EMP_CODE	E_NAME	DESIGNATION	SALARY	DEPTNO
7782	CLARK	MANAGER	64000	10
7876	ADAMS	CLERK	23000	20
7844	TURNER	SALESMAN	30430	30

#### Example 53: Before Insert Trigger

In the below example, a trigger is associated with the employee table. This trigger would fire before inserting a new record in the table.

Autocommit Display 10 🔹		Save	Rur
reate or replace trigger insert <u>emp</u>			
pefore Insert on employee for each row			
pegin			
DBMS_OUTPUT.PUT_LINE('New employee Code inserte DBMS_OUTPUT.PUT_LINE('New employee Name inserte	is: '     is :'	:NEW. <u>emp_</u> co   :NEW.e_name	ode); );
end;			

In the above example, **insert\_emp** is a trigger which is associated with the employee table. This is a trigger would fire on insert command on *employee* table and would prompt new employee code and employee name before inserting it in to employee table.

#### **Application:**

To test whether the trigger is fired and display message on screen, insert new record into *employee* table as shown below:

SQL> Insert into employee (emp\_code, e\_name) values
(321,'Scott');

When new record is inserted into *employee* table, system prompts the message as shown below:

New employee Code inserted is :321

New employee Name inserted is :Scott

Note: The trigger would execute even if you insert data in all the fields of employee table.

Example 54: To create IF Statement in Trigger.

A database trigger also use if statement. If statements in database triggers is used to determine what statement caused the execution of the trigger, such as inserting, updating or deleting a data from the associated table.

The general form of if statements in trigger are:

- If Inserting Then
- If Deleting Then
- If Updating Then

	10 •	Save	Run
create or replace tri before insert or upda for each row begin /* the trigger would record from employee if inserting then dbms_output.p elsif deleting then dbms_output.p elsif updating then dbms_output.p elsif updating then dbms_output.p   :new.e_name); end if; end;	gger emp_trigger te or Delete on employe fire either by insertin table and the following ut_line(' Inserting Emp ut_line(' Deleting Empl ut_line(' Updating Empl	e g, updation or deleting t ; conditions would be chec ployee '    :new.e_name); oyee '    :old.e_name); oyee '    :old.e_name	he ked */

RDBMS Lab

#### NOTES

In the above example, **emp\_trigger** is a database trigger which is associated with the employee table. This is a trigger having three if conditions to determine what statement invoked it, and prompts an appropriate message in various cases.

#### NOTES

RDBMS Lab

### | Viewing Triggers

To view all the triggers created by the user, a data dictionary named USER\_TRIGGERS can be used.

To see all the triggers use select statement on USER\_TRIGGERS as shown below:

alts Explain Describe Saved SQL History		
Ilts Explain Describe Saved SQL History		
Ilts Explain Describe Saved SQL History		
Ilts Explain Describe Saved SQL History		
Its Explain Describe Saved SQL History		
Ilts Explain Describe Saved SQL History		
GGER_NAME		
GGER_NAME		
GGER_NAME		
HISTORY		
is returned in 0.00 seconds		
for more description, you can also write the follow	ing command	1.
or more description, you can also write the following	ing command	1.
<pre>&gt; Select * from user_triggers;</pre>		
ig a Trigger		
/e .e		
: Noron triggor < triggor name >		
: > Drop trigger < trigger name >		
: > Drop trigger < trigger name >		
: > Drop trigger < trigger name >		
: > Drop trigger < trigger name >		

Example 55: Write a query to delete a trigger from emp\_history.

Home > SQL > SQL Commands Autocommit Display 10 ▼ Save Run Drop trigger emp\_history ;] Results Explain Describe Saved SQL History Trigger dropped.

### 1.10 seconds

#### PL/ SQL Package

A package is \_\_\_\_\_\_a database object. It is a collection of various database objects as procedures, functions, cursors, variables and constants.

There are two types of packages:

- 1. Built-in Packages
- 2. User defined Packages

#### **Built-in Packages**

Built-in Packages such as DBMS\_OUTPUT, DBMS\_SQL, DBMS\_DDL, DBMS\_TRANSACTION etc. caters pre-defined functionality.

#### **User defined Packages**

User defined package serve the user as per the changed business needs.

A package consists of two parts:

- Package Specification
- Package Body

## NOTES

RDBMS Lab

#### RDBMS Lab

#### **Package Specification**

In package specification one can declare variables, constants, exceptions, cursors, sub-procedures and other database objects.

**NOTES** 

#### Syntax:

```
CREATE [or Replace] Package < package_name > IS <
declarations >
Begin
   (Executable statements)
```

END <package\_name >;

The sub-procedures declared in package specification must be declared in package body.

#### **Package Body**

The actual implementation of declared sub-procedures and cursors is done in package body. The sub-procedures declared in package specification must be declared in package body.

#### *Syntax:* The CREATE BODY statement is as follows:

```
CREATE [or Replace] package < package_name > IS <
declarations >
Procedure < procedure_name > (variable data type);
Function < function_name > (variable data type) return
data type;
END < body_name >;
```

#### **A Package Function**

The example given below declares a function getGrade which would accept an argument of varchar data type and would return a value of varchar data type.

Example 56: To create or replace a package. RDBMS Lab Step 1:



The above code will create a package with the name pkg\_marksheet. This package contains a function named getGrade. This function will accept an argument of varchar type and will return a value of varchar type.

#### Package created.

#### Step 2:

The function pkg\_marksheet is declared in package body as shown below:

```
create or replace package body {\tt pkg\_marksheet} as function getgrade (rno varchar ) return varchar IS
```

```
s1 number (3) ;
s2 number (3) ;
s3 number (3) ;
s4 number (3) ;
total number (3) ;
per number (3) ;
```

NOTES

Self-Instructional Material

```
RDBMS Lab
                        begin
                            select sub1, sub2, sub3, sub4 into s1, s2, s3 , s4
                        from marks where roll no = rno ;
                            total := s1 + s2 + s3 + s4;
      NOTES
                            per := total / 4 ;
                        if per \geq= 90 then
                            return `A+' ;
                        elsif per >= 80 then
                            return 'A' ;
                        elsif per >= 70 then
                            return 'A-';
                        elsif per >= 60 then
                            return 'B+' ;
                        elsif per \geq 50 then
                            return 'B' ;
                        elsif per >= 40 then
                            return 'B-' ;
                        elsif per >= 30 then
                            return 'C' ;
                        else
                            return 'F' ;
                        end if ;
                        end getgrade ;
                        end pkg_marksheet ;
                        /
                           The output of the above PL/ SQL code, when compiled is given below:
                         Results Explain Describe Saved SQL History
                         Package Body created.
                         0.50 seconds
                                                                         Application Express 2.1.0.00.39
                        Language: en-us
                                                              Copyright © 1999, 2008, Oracle. All rights reserved.
```
#### **Calling Package Function**

To call the function declared in package specification, the reference of package name need to give as given below:

An example to call a package function is as follows:

pkg marksheet.getGrade ('A-08-12');

Where, pkg\_marksheet is a package name in which a function getGrade is declared which takes a varchar argument A-08-12.

#### **A Package Procedure**

The example given below declares a procedure show\_book\_price which would accept an argument of varchar data type.

Example 57: To create a package procedure. Step 1:



The above code will create a package with the name book\_price. This package contains a procedure named show\_book\_price. This procedure will accept an argument of varchar type.

Note: Procedure cannot return any value.

NOTES

RDBMS Lab

Self-Instructional Material

69

RDBMS Lab

The output of the above PL/ SQL code when compiled is given below: **Package created.** 

Step-2

## NOTES

Autocommit Display 10	Save	Run
<pre>create or replace package book_price as</pre>	e;	
Results Explain Describe Saved SQL History		

Save the above program with the any name (let us suppose show\_price) and then run it.

The output of the above PL/ SQL code when compiled is given below:

Package body created.

## **Calling Package Procedure**

To call the procedure declared in package specification, the reference of package name need to give as shown below:

The *Syntax* to call a package procedure is as follows:

Package name.procedure name;

The example to call a package procedure is as follows :

book price. show book price ('B003');

Where, book\_price is a package name in which a procedure show\_book\_price is declared which takes a varchar argument B003.

Self-Instructional 70 Material

## **Reports using functions**

A stored function always returns a result and can be called inside an SQL statement just like ordinary SQL function. A function parameter is the equivalent of the in procedure parameter, as functions use the RETURN keyword to determine what is passed back. User-defined functions or stored functions are the stored procedures which have the features of all procedures. They can accept parameters, perform calculations based on data retrieved and return the result to the calling SQL statement, procedure, function or PL/SQL program.

## **Create a Function**

The syntax to create a function is as follows:

CREATE OR REPLACE FUNCTION function\_name (function\_params)

RETURN return\_type IS

Declaration statements

BEGIN

Executable statements

RETURN something\_of\_return\_type;

EXCEPTION

Exception section

END;

**Description of the Syntax** 

## **CREATE Function:**

This is used to create a function, if no other function with the given name exists.

## **OR REPLACE Function:**

OR REPLACE is used to re-create the function if the given function name already exists. If no function exists with the given name, it creates the new function. You can also use OR REPLACE clause to change the definition of an existing function without dropping, re-creating and regranting privileges previously granted on the function to other users. If you redefine a function, then Oracle Database recompiles it.

## IS:

It is similar to DECLARE in PL/SQL Blocks. Variables could be declared between IS and BEGIN.

## RETURN

Clause Function returns a value. The RETURN clause is used to specify the data type of the return value of the function. Since every function must return a value,

RDBMS Lab

## NOTES

Self-Instructional Material this clause is mandatory to use. The return value can have any data type supported by PL/SQL.

**Example 58:** Consider table given below, which contains the detailed of accounts of account holders of bank.

NOTES

RDBMS Lab

ACC_NO	NAME	TYPE_OF_AC	CONTACT_NO	AC_BALANCE
120040	Tom	Saving	98978800	15620
120040	Merlisa	Saving	98981600	26500
120041	George	Saving	8787700	16560
120041	Smith	Saving	6050234	25500
120042	Loise	Current	6050234	26660
120043	marry	Current	38042342	70080

 Table:
 Account\_holder

A stored function is given to return the balance of an account holder. The account number is passed as a parameter in this function.

```
Function: get balance ()
```

/\* This is a stored function which returns
the total balance of all saving accounts\*/
CDEATE or replace FUNCTION get balance ( no IN

```
CREATE or replace FUNCTION get_balance ( no IN NUMBER)
```

RETURN NUMBER

```
IS acc_bal NUMBER ( 11 , 2 ) ;
BEGIN
SELECT sum ( ac_balance ) INTO acc_bal from
account_holder WHERE acc_no = no ;
RETURN ( acc_bal ) ;
END;
/
```

The given function, get\_balance () has a parameter of number type to accept the account holder's account number. The acc\_bal is a variable in which the balance of the given account holder is stored and returned to the caller program.

Save	the above file with the name account_balance.SQL
Com	pile Function
To ex proce	accute any stored procedure it is necessary to compile it. To compile a dure the following command is used:
	The syntax is as follows:
	SQL> @ function_name ;
	For example,
	SQL>@ account_balance;
	Example 59: Based on library information system.
	List of tables:
	Book_Details
	Binding_Details
	Category_Details
	Borrower_Details
	Student_Details
	Staff_Details
	Student_Details
	Shelf_Details
Libra	ary Management System (SQL Commands)
Creat	ing table "Book_Details":
1.	CREATE TABLE Book_Details
2.	(
3.	ISBN_Codeint PRIMARY KEY,
4.	Book_Titlevarchar(100),
5.	Language varchar(10),
6.	Binding_Idint,
7.	No_Copies_Actualint,
8.	No_Copies_Currentint,

Save file

RDBMS Lab

# NOTES

Self-Instructional Material

RDBMS Lab	9. Category_idint,		
	10. Publication yearint		
	11.)		
NOTES	Inserting Some Data in "Book_Details":		
	1. INSERT INTO Book_details		
	<pre>2. VALUES(`0006','Programming Concept', 'English',2,20,15,2,2006);</pre>		
	Creating table "Binding_Details":		
	1. CREATE TABLE Binding_details		
	2. (		
	3. Binding_idint PRIMARY KEY,		
	4. Binding_Namevarchar(50)		
	5.)		
	Describe Binding table:		
	Describe binding_details;		
	Inserting Some data in Binding Table:		
	<pre>1.INSERT INTO Binding_DetailsVALUES    (1,'McGraw Hill);</pre>		
	<pre>2.INSERT INTO Binding_DetailsVALUES    (2,'BPB Publication');</pre>		
	All Data of Binding Table:		
	1. select *from binding_Details		
	BINDING ID BINDING NAME		
	1 McGraw Hill		
	2 BPB Publication		
	Creating Relationship between Book and Binding Table:		
	1. ALTER TABLE Book details		
	2. ADD CONSTRAINT Binding_ID_FK FOREIGN KEY( Binding_Id) REFERENCES Binding_Details(Binding_Id);		

Self-Instructional 74 Material

```
Checking Relationship:
 1. selectb.Book Title, e.binding name
 2. fromBook Detailsb, Binding Details e
 3. whereb.binding id = e.binding id;
                              BINDING NAME
                BOOK_TITLE
              Introduction to database McGraw Hill
              Programming Concept
                              BPB Publication
Creating Category Table:
 1. CREATE TABLE Category Details
 2. (
 3. Category_Idint PRIMARY KEY,
 4.
      Category Namevarchar(50)
 5.)
Inserting some data in Category Table:
 1. INSERT INTO CATEGORY DETAILS VALUES
    (1, 'Database');
 2. INSERT INTO CATEGORY DETAILS VALUES (
    2, 'Programming Language');
Building Relationship between Book & Category Table:
 1. ALTER TABLE Book details
 2. ADD CONSTRAINT Category Id FK FOREIGN KEY
    (Category Id) REFERENCES Category Details (Category Id);
Checking Relationship:
 1. Select b.Book Title, e.Category Name
 2. From Book Detailsb, Category Details e
 3. whereb.binding id = e.Category id;
```

BOOK_TITLE	CATEGORY_NAME	
Introduction to database	Database	
Programming Concept	Programming Language	

Self-Instructional Material

#### RDBMS Lab

#### NOTES

RDBMS Lab	Creating Borrower Table:		
	1. CREATE TABLE Borrower_Details		
	2. (		
NOTES	3. Borrower_Idint PRIMARY KEY,		
	4. Book_Idint,		
	5. Borrowed_From date,		
	6. Borrowed_TO date,		
	7. Actual_Return_Date date,		
	8. Issued_byint		
	9.)		
	Inserting some data in Category Table:		
	1. Insert into BORROWER_DETAILS VALUES(1,0004,' 01-Aug-2014','7-Aug-2014','7-Aug-2014',1)		
	2. Insert into BORROWER_DETAILS VALUES(2,6,'02- Aug-2014','8-Aug-2014',NULL,1)		
	Building Relation between Book & Borrower Table:		
	1. ALTER TABLE Borrower_details ADD CONSTRAINT Book_Id_FK FOREIGN KEY(Book_Id) REFERENCES Book Details(ISBN Code);		
	Checking Relationship:		
	1. Select Borrower_Details.Borrower_id, Book_Details.Book_title		
	2. From Borrower_Details, Book_Details		
	3. Where Borrower_Details.book_id=Book_Details. ISBN_Code		
	BORROWER_ID BOOK_TITLE		
	1 Introduction to database		
	2 Programming Concept		

Self-Instructional 76 Material

```
2. ADD CONSTRAINT Issued by \_{\tt FK} Foreign Key(
    Issued by) REFERENCES Staff Details (Staff Id);
Creating Staff Table:
 1. CREATE TABLE Staff Details
 2. (
 3. Staff Idint PRIMARY KEY,
 4. Staff Namevarchar(50),
 5. Password varchar(16),
 6. Is Adminbinary float,
 7. Designation varchar(20)
 8.)
Inserting some data in Staff Table:
 1. Insert into STAFF DETAILS values
    (1,'Tarek Hossain','1234asd',0,'Lib mgr');
 2. Insert into STAFF DETAILS values
    (2,'Md.Kishor Morol','iloveyou',0,'Lib clr');
All Data of Staff table:
 1. select * from staff details
   STAFF_ID STAFF_NAME
                       PASSWORD IS_ADMIN
                                           DESIGNATION
   1
            Tarek Hossain
                        1234asd
                                 1.0E+000
                                           Lib_mgr
   2
            Md.Kishor Morol
                       iloveyou
                                  0
                                           Lib_clr
Creating Student Table:
  1. Create TABLE Student Details
 2. (
```

1. ALTER TABLE Borrower Details

```
3. Student Idvarchar(10) PRIMARY KEY,
```

```
4. Student_Namevarchar(50),
```

5. Sex Varchar(20),

NOTES

RDBMS Lab

Self-Instructional Material

77

```
6. Date_Of Birth date,
 7. Borrower Idint,
 8. Department varchar(10),
 9. contact Numbervarchar(11)
10.)
Inserting Some Data in Student Table:
 1. Insert into STUDENT DETAILS values ('13-23059-
     1', 'Ahmed, Ali', 'Male', '05-Oct-
    1995',1,'CSSE','01681849871');
 2. Insert into STUDENT DETAILS values ('13-23301-
     1','MOrol MD.Kishor','Male','03-Jan-
    1994',2,'CSE','01723476554');
All Data of Student Table:

    select *from student details;

                  SEX DATE_OF_BIRTH BORROWER_ID DEPARTMENT CONTACT_NUMBER
 STUDENT_ID STUDENT_NAME
 13-23059-1
         Ahmed Ali
                      05-OCT-95
                                          CSSE
                                                  01681849871
                   Male
                                 1
 13-23301-1
         MOrol MD.Kishor
                   Male 03-JAN-94
                                2
                                          CSE
                                                  01723476554
Building Relationship between student and Borrower table:
 1. ALTER TABLE student details
 2. ADD CONSTRAINT borrower id FK FOREIGN KEY(
    Borrower Id) REFERENCES Borrower Details (Borrower Id);
Checking Full Relationship:
 1. select student.student id, student.student
    name, book.Book Title, staff.staff name, b.Borrowed To
 2. fromstudent Detailsstudent, Staff
    Detailsstaff, Borrower Detailsb, book details book
 3. wherestudent.Borrower id = b.Borrower i
     d and book.ISEN Code = b.book id and b.Issued by = staff.Staff id;
 STUDENT ID
           STUDENT_NAME
                          BOOK TITLE
                                       STAFF_NAME
                                                 BORROWED_TO
 13-23059-1
           Ahmed,Ali
                        Introduction to database Tarek Hossain
                                                 07-AUG-14
 13-23301-1
           MOrol MD.Kishor
                        Programming Concept
                                                 08-AUG-14
                                       Tarek Hossain
```

NOTES

RDBMS Lab

Self-Instructional 78 Material

```
Adding Shelf Table:
```

```
1. Create Table Shelf_Details
```

```
2. (
```

- 3. Shelf idint PRIMARY KEY,
- 4. Shelf\_Noint,
- 5. Floor\_Noint

6.);

## Inserting Some Data from Shelf Table:

- 1. Insert into Shelf DetailsValues(1, 1, 1);
- 2. Insert into Shelf\_DetailsValues (2, 2, 10001);
- 3. Insert into Shelf DetailsValues (3, 1, 10002);

## All Data in Shelf Table:

1. select\*from Shelf\_Details;

SHELF_ID	SHELF_NO	FLOOR_NO
1	1	1
2	2	10001
3	1	10002

Adding Relationship between Shelf and Book Table:

```
1. ALTER TABLE Book_Details
```

```
2. ADD(Shelf_Idint);
```

```
3. UPDATE Book_Details set Shelf_Id = 1
```

- 4. where ISBN CODE = 4;
- 5. UPDATE Book Details set Shelf Id = 2
- 6. where ISBN CODE = 6;
- 7. ALTER TABLE Book\_Details
- 8. ADD CONSTRAINT Shelf\_Id\_FK FOREIGN KEY
   (Shelf\_Id) REFERENCES Shelf\_Details(Shelf\_Id);

## NOTES

RDBMS Lab

Self-Instructional Material

RDBMS Lab	Combine all Relationship:
NOTES	<pre>1. select student.student_id, student.student     _name, book.Book_Title, staff.staff_name, b.Borrowed_To,     shelf.shelf_No</pre>
	2.fromstudent_Detailsstudent, Staff_Detailsstaff, Borrower_Detailsb, book_detailsbook, Shelf_Details shelf
	<pre>3. wherestudent.Borrower_id = b.Borrower_ id and book.ISBN_Code = b.book_id and b.Issued_by = staff.Staff_id and book.Shelf_Id = shelf.Shelf_Id;</pre>

Self-Instructional 80 Material