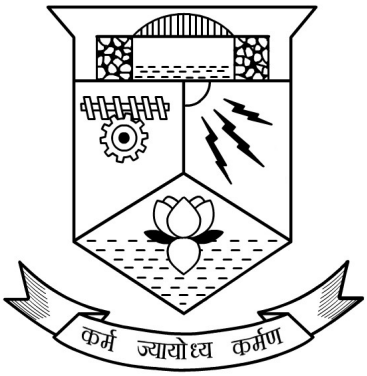
CS333 APPLICATION SOFTWARE DEVELOPMENT LAB

B.Tech V Semester Computer Science and Engineering
(APJ Abdul Kalam Technological University)



Department of Computer Science and Engineering
College of Engineering, Trivandrum

Vision and Mission of the Institution

Vision

National level excellence and international visibility in every facet of engineering research and education.

Mission

I. To facilitate quality transformative education in engineering and management.
II. To foster innovations in technology and its application for meeting global challenges.
III. To pursue and disseminate quality research.
IV. To equip, enrich and transform students to be responsible professionals for better service to humanity.

Vision and Mission of the Department

Vision

To be a centre of excellence in education and research in the frontier areas of Computer Science and Engineering.

Mission

I. To facilitate quality transformative education in Computer Science and Engineering.
II. To promote quality research and innovation in technology for meeting global challenges.
III. To transform students to competent professionals to cater to the needs of the society.

Program Educational Objectives (PEOs)

Consistent with the stated vision and mission of the institute and the Department, the faculty members of the Department strive to train the students to become competent technologists with integrity and social commitment.

Within a short span of time after graduation, the graduates shall:

PEO1. Achieve professional competency in the field of Computer Science and Engineering.

PEO2. Acquire domain knowledge to pursue higher education and research.

PEO3. Become socially responsible engineers with good leadership qualities and effective interpersonal
skills.

Program Specific Outcomes (PSOs):

PSO1: Model computational problems by applying mathematical concepts and design solutions using suitable data structures and algorithmic techniques.

PSO2: Design and develop solutions by following standard software engineering principles and implement by using suitable programming languages and platforms.

PSO3: Develop system solutions involving both hardware and software modules

Program Outcomes (POs)

PO1 : Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2 : Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4 : Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and

interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7 : Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice

PO9 : Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10 : Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Summary

Prerequisite and Course Description

The intention of this course is to build an understanding on database design and implementation. CS208 Principles of Database Design is the prerequisite. However introductory knowledge in database design is desirable.

Course Educational Objectives

1. To introduce basic commands and operations on database.
2. To introduce stored programming concepts (PL-SQL) using Cursors and Triggers .
3. To familiarize front end tools of database.

Course Outcomes (COs)

On completion of the course students will be able to

CO1 : Design and model relational database scheme for real-world scenarios using standard design and modeling approaches and implement it using standard DBMS software.

CO2 : Design and implement read-only and update queries on databases.

CO3 : Design and implement triggers and cursors.

CO4 : Implement of stored procedures and control structures (PL/SQL)

CO5 : Use front end tools to access database through db connectivity protocols.

CO6: Design and implement complete database projects with front-end and DB Back-end.

Evaluation Plan

Students will be evaluated based on the following criteria:
Practical record/Output = 30
Project =30
Regular lab viva = 10
Final written test/Quiz = 30

Syllabus

List of Exercises/Experiments: (Exercises/experiments marked with * are mandatory. Total 12 Exercises/experiments are mandatory)

1. Creation of a database using DDL commands and writes DQL queries to retrieve information from the database.
2. Performing DML commands like Insertion, Deletion, Modifying, Altering, and Updating records based on conditions.
3. Creating relationship between the databases. *
4. Creating a database to set various constraints. *
5. Practice of SQL TCL commands like Rollback, Commit, Savepoint.
6. Practice of SQL DCL commands for granting and revoking user privileges.

7. Creation of Views and Assertions *
8. Implementation of Build in functions in RDBMS *
9. Implementation of various aggregate functions in SQL *
10. Implementation of Order By, Group By& Having clause. *
11. Implementation of set operators, nested queries and Join queries *
12. Implementation of various control structures using PL/SQL *
13. Creation of Procedures and Functions *
14. Creation of Packages *
15. Creation of database Triggers and Cursors *
16. Practice various front-end tools and report generation.
17. Creating Forms and Menus
18. Mini project (Application Development using Oracle/ MySQL using Database connectivity)*

a. Inventory Control System.
b. Material Requirement Processing.
c. Hospital Management System.
d. Railway Reservation System.
e. Personal Information System.
f. Web Based User Identification System.
g. Timetable Management System.
h. Hotel Management System.

Lab Cycle

Postgresql Installation

Step 1: Update system and install dependencies

- sudo apt update
- sudo apt install -y wget

Step 2: Add PostgreSQL 11 APT repository

- Download the key for postgresql repository and add it to the package manager keyring
- wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -
- RELEASE=$(lsb_release -cs)
- echo "deb http://apt.postgresql.org/pub/repos/apt/ ${RELEASE}"-pgdg main | sudo tee /etc/apt/sources.list.d/pgdg.list
- cat /etc/apt/sources.list.d/pgdg.list.

Step 3: Install PostgreSQL 11 on Ubuntu 18.04 / Ubuntu 16.04

- sudo apt update
- sudo apt -y install postgresql-11

Step 4: Set PostgreSQL admin user’s password

- sudo su - postgres
- psql -c "alter user postgres with password '<YOUR PASSWORD>' "

Step 5: Post-Installation

- Postgres has been successfully installed .You can now use postgres by following the steps given below
- sudo su - postgres
- psql
- This will open the postgresql command line.

Exp No. 1

INTRODUCTION TO SQL

History of SQL

Dr. E. F. Codd published the paper, "A Relational Model of Data for Large Shared Data Banks", in June 1970 in the Association of Computer Machinery (ACM) journal, Communications of theACM. Codd's model is now accepted as the definitive model for relational database management systems (RDBMS). The language, Structured English Query Language ("SEQUEL") was developed by IBM Corporation, Inc., to use Codd's model. SEQUEL later became SQL (still pronounced "sequel"). In 1979, Relational Software, Inc. (now Oracle Corporation) introduced the first commercially available implementation of SQL. Today, SQL is accepted as the standard RDBMS language.

How SQL Works

The strengths of SQL provide benefits for all types of users, including application programmers, database administrators, managers, and end users. Technically speaking, SQL is a data sublanguage. The purpose of SQL is to provide an interface to a relational database such as Oracle, and all SQL statements are instructions to the database. In this SQL differs from general-purpose programming languages like C and BASIC. Among the features of SQL are the following:

1. It processes sets of data as groups rather than as individual units.

2. It provides automatic navigation to the data.

3. It uses statements that are complex and powerful individually, and that therefore stand alone. Flow-control statements were not part of SQL originally, but they are found in the recently accepted optional part of SQL, ISO/IEC 9075-5: 1996. Flow-control statements are commonly known as "persistent stored modules" (PSM), and Oracle's PL/SQL extension to SQL is similar to PSM.

Essentially, SQL lets you work with data at the logical level. You need to be concerned with the implementation details only when you want to manipulate the data. For example, to retrieve a set of rows from a table, you define a condition used to filter the rows. All rows satisfying the condition are retrieved in

a single step and can be passed as a unit to the user, to another SQL statement, or to an application. You need not deal with the rows one by one, nor do you have to worry about how they are physically stored or retrieved. All SQL statements use the optimizer, a part of Oracle that determines the most efficient means of accessing the specified data. Oracle also provides techniques you can use to make the optimizer perform its job better.

SQL provides statements for a variety of tasks, including:

1. Querying data
2. Inserting, updating, and deleting rows in a table
3. Creating, replacing, altering, and dropping objects
4. Controlling access to the database and its objects
5. Guaranteeing database consistency and integrity

SQL unifies all of the above tasks in one consistent language.

Common Language for All Relational Databases

All major relational database management systems support SQL, so you can transfer all skills you have gained with SQL from one database to another. In addition, all programs written in SQL are portable. They can often be moved from one database to another with very little modification.

Summary of SQL Statements

SQL statements are divided into these categories:

1. Data Definition Language (DDL) Statements
2. Data Manipulation Language (DML) Statements
3. Transaction Control Statements (TCL)
4. Session Control Statement
5. System Control Statement

Managing Tables

A table is a data structure that holds data in a relational database. A table is composed of rows and columns.

A table can represent a single entity that you want to track within your system. This type of a table could represent a list of the employees within your organization, or the orders placed for your company's products.

A table can also represent a relationship between two entities. This type of a table could portray the association between employees and their job skills, or the relationship of products to orders. Within the tables, foreign keys are used to represent relationships.

Creating Tables

To create a table, use the SQL command CREATETABLE.

Syntax:

CREATE TABLE <TABLE NAME>(<FIELD NAME ><DATA TYPE><[SIZE]>,.....)

Altering Tables

Alter a table in an Oracle database for any of the following reasons:

1. To add one or more new columns to the table
2. To add one or more integrity constraints to a table
3. To modify an existing column's definition (datatype, length, default value, and NOTNULL integrity constraint)
4. To modify data block space usage parameters (PCTFREE, PCTUSED)
5. To modify transaction entry settings (INITRANS, MAXTRANS)
6. To modify storage parameters (NEXT, PCTINCREASE, etc.)
7. To enable or disable integrity constraints associated with the table
8. To drop integrity constraints associated with the table

When altering the column definitions of a table, you can only increase the length of an existing column, unless the table has no records. You can also decrease the length of a column in an empty table. For columns of data type CHAR, increasing the length of a column might be a time consuming operation that requires substantial additional storage, especially if the table contains many rows. This is because the CHAR value in each row must be blank-padded to satisfy the new column length.

If you change the datatype (for example, from VARCHAR2 to CHAR), then the data in the column does not change. However, the length of new CHAR columns might change, due to blank-padding requirements.

Altering a table has the following implications:

1. If a new column is added to a table, then the column is initially null. You can add a column with a NOTNULL constraint to a table only if the table

does not contain any rows.

2. If a view or PL/SQL program unit depends on a base table, then the alteration of the base table might affect the dependent object, and always invalidates the dependent object.

Privileges Required to Alter a Table

To alter a table, the table must be contained in your schema, or you must have either the ALTER object privilege for the table or the ALTERANYTABLE system privilege.

Dropping Tables

Use the SQL command DROPTABLE to drop a table. For example, the following statement drops the EMP_TAB table:

If the table that you are dropping contains any primary or unique keys referenced by foreign keys to other tables, and if you intend to drop the FOREIGNKEY constraints of the child tables, then include the CASCADE option in the DROPTABLE command.

Oracle Built-In Datatypes

A datatype associates a fixed set of properties with the values that can be used in a column of a table or in an argument of a procedure or function. These properties cause Oracle to treat values of one datatype differently from values of another datatype. For example, Oracle can add values of NUMBER datatype, but not values of RAW datatype.

Oracle supplies the following built-in datatypes:character data types

- CHAR
- NCHAR
- VARCHAR2 and VARCHAR
- NVARCHAR2
- CLOB

- NCLOB
- LONG

1. NUMBER datatype
2. DATE datatype
3. Binary datatypes

- BLOB
- BFILE
- RAW
- LONG RAW

Another datatype, ROWID, is used for values in the ROWIDpseudocolumn, which represents the unique address of each row in a table.
Table summarizes the information about each Oracle built-in datatype.

Summary of Oracle Built-In Datatypes

Data Type	**Description**	**Column Length and Default**
CHAR (size)	Fixed-length Character data of length size bytes	Fixed for every row in the table (with trailing blanks); byte per row. Consider the character set (one-byte or multibyte) before setting size.
VARCHAR2 (size)	Variable-length Character data	Variable for each row, up to 4000 bytes per row:Consider the character set (one-byte or multibyte)before setting size: A maximum size must be specified.

NCHAR(size)	Fixed Length Character of length size , characters, bytes, depending on the character set	Fixed for every row in the table(with trailing blanks).Coloum size is the number of characters for a fixed width national character set or the number of bytes for a varying width national character set. Maximum size is determined ny the number of bytes required to store one character, with an upper limit of 2000 bytes per row. Default is 1 character or 1 byte, depending on the character set.
NVARCHAR2 (Size0	Variable-length Character data of length size, characters or bytes,depending on National character set: A maximum size must be specified	Variable for each row. Column size is the number of characters for a fixed-width national character set or the number of bytes for a varying-width national character set. Maximum size is determined by the number of byte, depending on the character set.
CLOB	Single-byte character data	
LONG	Variable-length Character data	Variable for each row in the table, up to 2^31 - 1 bytes, or 2 gigabytes, per row. Provided for backward

NUMBER(p, s)	Variable-length Numericdata.: Maximum precision p and/or scale s is 38	Variable for each row. The maximum space required
DATE	Fixed-length date and time data, ranging from Jan. 1, 4712 B.C.E. to Dec. 31, 4712 C.E.	Fixed at 7 bytes for each row in the table. Default format is a string (such as DD-MON-YY) specified by
BLOB	Unstructured binary data	
BFILE	Binary data stored in an external file	
RAW (size)	Variable-length Raw	Variable for each row in the table, up to 2000 bytes per row. A maximum size must be specified. Provided for binary data
LONG RAW	Variable-length Raw binary data	Variable for each row in the table, up to 2^31 - 1 bytes, or 2 gigabytes, per row. Provided for backward
ROWID	Binary data representing row addresses	Fixed at 10 bytes (extended ROWID) or 6 bytes (restricted ROWID) for each row in the table.
MLSLABEL	Trusted Oracle data type	

Using Character Data types

Use the character data types to store alphanumeric data.

1. CHAR and NCHAR data types store fixed-length character strings.

2. VARCHAR2 and NVARCHAR2 data types store variable-length character strings. (The VARCHAR datatype is synonymous with the VARCHAR2 datatype.)

3. CLOB and NCLOB data types store single-byte and multibyte character strings of up to four gigabytes.

4. The LONG datatype stores variable-length character strings containing up to two gigabytes, but with many restrictions.

5. This data type is provided for backward compatibility with existing applications; in general, new applications should use CLOB and NCLOB data types to store large amounts of character data.

When deciding which datatype to use for a column that will store alphanumeric data in a table,

consider the following points of distinction:

Space Usage

1. To store data more efficiently, use the VARCHAR2 datatype. The CHAR data type blank-pads and stores trailing blanks up to a fixed column length for all column values, while the VARCHAR2 datatype does not blank-pad or store trailing blanks for column values.

Comparison Semantics

1. Use the CHAR data type when you require ANSI compatibility in comparison semantics (when trailing blanks are not important in string comparisons). Use the VARCHAR2 when trailing blanks are important in string comparisons.

Future Compatibility

1. The CHAR and VARCHAR2 datatypes are and will always be fully supported. At this time, the VARCHAR datatype automatically corresponds to the VARCHAR2 datatype and is reserved for future use.

CHAR, VARCHAR2, and LONG data is automatically converted from the database character set to the character set defined for the user session by the NLS_LANGUAGE parameter, where these are different.

Using the NUMBER Datatype

Use the NUMBER datatype to store real numbers in a fixed-point or floating-point format. Numbers using this data type are guaranteed to be portable among different Oracle platforms, and offer up to 38 decimal digits of precision. You can store positive and negative numbers of magnitude 1 x 10^-130 to 9.99...x10^125, as well as zero, in a NUMBER column.

For numeric columns you can specify the column as a floating-point number:

Column_name NUMBER

Or, you can specify a precision (total number of digits) and scale (number of digits to the right of the decimal point):

Column_name NUMBER (<precision>, <scale>)

Although not required, specifying the precision and scale for numeric fields provides extra integrity checking on input. If a precision is not specified, then the column stores values as given. Table shows examples of how data would be stored using different scale factors.

Using the DATE Datatype

Use the DATE datatype to store point-in-time values (dates and times) in a table. The DATE datatype stores the century, year, month, day, hours, minutes, and seconds.

Oracle uses its own internal format to store dates. Date data is stored in fixed-length fields of seven bytes each, corresponding to century, year, month, day, hour, minute, and second.

Date Format

For input and output of dates, the standard Oracle default date format is DD-MON-YY.

For example: '13-NOV-92'

To change this default date format on an instance-wide basis, use the NLS_DATE_FORMAT parameter. To change the format during a session, use the ALTER SESSION statement. To enter dates that are not in the current default date format, use the TO_DATE function with a format mask. For example:

TO_DATE ('November 13, 1992', 'MONTH DD, YYYY')

If the date format DD-MON-YY is used, then YY indicates the year in the 20th century (for example, 31-DEC-92 is December 31, 1992). If you want to indicate years in any century other than the 20th century, then use a different format mask, as shown above.

Time Format

Time is stored in 24-hour format#HH:MM:SS. By default, the time in a date field is 12:00:00 A.M. (midnight) if no time portion is entered. In a time-only entry, the date portion defaults to the first day of the current month. To enter the time portion of a date, use the TO_DATE function with a format mask indicating the time portion, as in:

INSERT INTO Birthdays_tab (bname, bday) VALUES ('ANNIE',TO_DATE('13-NOV-92 10:56 A.M.','DD-MON-YY HH:MI A.M.'));

To compare dates that have time data, use the SQL function TRUNC if you want to ignore the time component. Use the SQL function SYSDATE to return the system date and time. The FIXED_DATE initialization parameter allows you to set SYSDATE to a constant; this can be useful for testing.

Exp No. 2

BASIC SQL QUERIES – I

AIM

To study the basic sql queries such as

1. SELECT
2. INSERT
3. UPDATE
4. DELETE.

QUESTIONS

Create a table named Employee and populate the table as shown below.

Emp_id	Emp_name	Dept	Salary (in US $)
1	Michael	Production	$2500
2	Joe	Production	$2500
3	Smith	Sales	$2250
4	David	Marketing	$2900
5	Richard	Sales	$1600
6	Jessy	Marketing	$1800
7	Jane	Sales	$2000
8	Janet	Production	$3000
9	Neville	Marketing	$2750
10	Richardson	Sales	$1800

1. Display the details of all the employees.

EMP_ID	EMP_NAME	DEPT	SALARY
1	Michael	Production	$2500
2	Joe	Production	$2500
3	Smith	Sales	$2250

2. Display the names and id’s of all employees.

EMP_ID	EMP_NAME
1	Michael
2	Joe
3	Smith
4	David
5	Richard
6	Jessy
7	Jane
8	Janet
9	Neville
10	Richardson

3. Delete the entry corresponding to employee id:10.
4. Insert a new tuple to the table. The salary field of the new employee should be kept NULL.
5. Find the details of all employees working in the marketing department.

EMP_ID	EMP_NAME	DEPT	SALARY
4	David	Marketing	$2900

6. Add the salary details of the newly added employee.

EMP_ID	EMP_NAME	DEPT	SALARY
1	Michael	Production	$2500
2	Joe	Production	$2500
3	Smith	Sales	$2250
4	David	Marketing	$2900
5	Richard	Sales	$1600
6	Jessy	Marketing	$1800
7	Jane	Sales	$2000
8	Janet	Production	$3000
9	Neville	Marketing	$2750
10	Richardson	Sales	$1900

7. Update the salary of Richard to 1900$.

EMP_ID	EMP_NAME	DEPT	SALARY
1	Michael	Production	$2500
2	Joe	Production	$2500
3	Smith	Sales	$2250
4	David	Marketing	$2900
5	Richard	Sales	$1900
6	Jessy	Marketing	$1800
7	Jane	Sales	$2000
8	Janet	Production	$3000
9	Neville	Marketing	$2750
10	Richardson	Sales	$1900

8. Find the details of all employees who are working for marketing and has a salary greater than 2000$.

EMP_ID	EMP_NAME	DEPT	SALARY
4	David	Marketing	$2900
9	Neville	Marketing	$2750

9. List the names of all employees working in the sales department and marketing department.

EMP_NAME

Smith

David

Richard

Jessy

Jane

Neville

Richardson

10. List the names and department of all employees whose salary is between2300$ and 3000$.

EMP_NAME	DEPT
Michael	Production
Joe	Production
David	Marketing
Janet	Production
Neville	Marketing

11. Update the salary of all employees working in production department 12%.

EMP_ID	EMP_NAME	DEPT	SALARY
1	Michael	Production	$2800
2	Joe	Production	$2800
3	Smith	Sales	$2250
4	David	Marketing	$2900
5	Richard	Sales	$1900
6	Jessy	Marketing	$1800
7	Jane	Sales	$2000
8	Janet	Production	$3360
9	Neville	Marketing	$2750
10	Richardson	Sales	$1900

12. Display the names of all employees whose salary is less than 2000$ or working for sales department.

EMP_NAME

Smith

Richard

Jessy

Jane

Richardson

RESULT

The query was executed and the output was obtained.

Exp No: 3

BASIC SQL QUERIES – II

AIM

Introduction to SQL statements

1. ALTER
2. RENAME
3. SELECT DISTINCT
4. SQL IN
5. SQL BETWEEN
6. Sql aliases
7. Sql AND
8. Sql OR

QUESTION

Create a table named car_details and populate the table as shown below.

ID	Name	company	country	ApproxPrice(in lakhs)
1	Beat	Chevrolet	USA	4
2	Swift	Maruti	Japan	6
3	Escort	Ford	USA	4.2
4	Sunny	Nissan	Japan	8
5	Beetle	Volkswagen	Germany	21
6	Etios	Toyota	Japan	7.2
7	Sail	Chevrolet	USA	5
8	Aria	Tata	India	7
9	Passat	Volkswagen	Germany	25

10	SX4	Maruti	Japan	6.7

1. List the names of all companies as mentioned in the database

COMPANY
Nissan
Ford
Toyota
Chevrolet
Maruti
Volkswagen
Tata

2. List the names of all countries having car production companies

COUNTRY
USA
Germany
India
Japan

3.List the details of all cars within a price range 4 to 7 lakhs

ID	NAME	COMPANY	COUNTRY	Approx Price
1	Beat	Chevrolet	USA	4
2	Swift	Maruti	Japan	6
3	Escort	Ford	USA	4.2
7	Sail	Chevrolet	USA	5
8	Aria	Tata	India	7
10	SX4	Maruti	Japan	6.7

4. List the name and company of all cars originating from Japan and having price<=6 lakhs

NAME	COMPANY
Swift	Maruti

5. List the names and the companies of all cars either from Nissan or having a price greater than 20 lakhs.

NAME	COMPANY
Sunny	Nissan
Beetle	Volkswagen
Passat	Volkswagen

6. List the names of all cars produced by (Maruti,Ford).Use SQL IN statement.

NAME
Swift
Escort
SX4

7. Alter the table cars to add a new field year (model release year).Upadate the year column for all the rows in the database.

Id	Name	Company	Country	Approx Price
1	Beat	Chevrolet	USA	4
2	Swift	Maruti	Japan	6
3	Escort	Ford	Usa	4.2
4	Sunny	Nissan	Japan	8
5	Beetle	Volkswagen	Germany	21
6	Etios	Toyota	Japan	7.2
7	Sail	Chevrolet	Usa	5
8	Aria	Tata	India	7
9	Passat	Volkswagen	Germany	25
10	SX4	Maruti	japan	6.7

8. Display the names of all cars as Car_name (while displaying the name attribute should be listed as car_aliases)

CAR_NAME

Beat
Swift

Escort
Sunny
Beetle
Etios
Sail
Aria
Passat
SX4

9. Rename the attribute name to car_name

Id	Car_name	Company	Country	Approx Price	Year
1	Beat	Chevrolet	USA	4	2015
2	Swift	Maruti	Japan	6	2015
3	Escort	Ford	USA	4.2	2015
4	Sunny	Nissan	Japan	8	2015
5	Beetle	Volkswagen	Germany	21	2015
6	Etios	Toyota	Japan	7.2	2015
7	Sail	Chevrolet	USA	5	2015
8	Aria	Tata	India	7	2015
9	Passat	Volkswagen	Germany	25	2015
10	SX4	Maruti	Japan	6.7	2015

10. List the car manufactured by Toyota(to be displayed as cars_Toyota)

CARS_TOYOTA
Etios

11. List the details of all cars in alphabetical order

ID	CAR_NAME	COMPANY	COUNTRY	APPROXPRICE	YEAR
8	Aria	Tata	India	7	2015
1	Beat	Cheverolet	USA	4	2015
5	Beetle	Volkswagen	Germany	21	2015
3	Escort	Ford	USA	4.2	2015

6	Etios	Toyota	Japan	7.2	2015
9	Passat	Volkswagen	Germany	25	2015
10	SX4	Maruti	Japan	6.7	2015
7	Sail	Cheverolet	USA	5	2015
4	Sunny	Nissan	Japan	8	2015
2	Swift	Maruti	Japan	6	2015

12. List the details of all cars from cheapest to costliest.

ID	CAR_NAME	COMPANY	COUNTRY	APPROXPRICE	YEAR
1	Beat	Cheverolet	USA	4	2015
3	Escort	Ford	USA	4.2	2015
7	Sail	Cheverolet	USA	5	2015
2	Swift	Maruti	Japan	6	2015
10	SX4	Maruti	Japan	6.7	2015
8	Aria	Tata	India	7	2015
6	Etios	Toyota	Japan	7.2	2015
4	Sunny	Nissan	Japan	8	2015
5	Beetle	Volkswagen	Germany	21	2015
9	Passat	Volkswagen	Germany	25	2015

RESULT

The query was executed successfully and output was verified.

Exp No: 4

AGGREGATE FUNCTIONS

AIM

Introduction to Aggregate functions

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

Description:
Aggregate Functions:
Sum()
Avg()
Count()
Max()
Min()

- Sum(fieldname)
  Returns the total sum of the field.

- Avg(fieldname)
  Returns the average of the field.

- Count( )
  Count function has three variations:

- Count(*) : returns the number of rows in the table including duplicates and those with null values

- Count(fieldname) : returns the number of rows where field value is not null
  Count (All): returns the total number of rows. It is same like count(*)

- Max(fieldname)
  Returns the maximum value of the field

- Min(fieldname)
  Returns the maximum value of the field

QUESTION

Create a table named student and populate the table as shown in the table.

The table contains the marks of 10 students for 3 subjects(Physics, Chemistry, Mathematics).The total marks for physics and chemistry is 25.while for mathematics it is 50.The pass mark for physics and chemistry is 12 and for mathematics it is 25.A student is awarded a ‘Pass’ if he has passed all the subjects.

Roll No	Name	Physics	Chemistry	Maths
1	Adam	20	20	33
2	Bob	18	9	41
3	Bright	22	7	31
4	Duke	13	21	20
5	Elvin	14	22	23
6	Fletcher	2	10	48
7	Georgina	22	12	22
8	Mary	24	14	31
9	Tom	19	15	24
10	Zack	8	20	36

ROLL_NO	NAME	PHYSICS	CHEMISTRY	MATHS
1	Adam	20	20	33
2	Bob	18	9	41
3	Bright	22	7	31
4	Duke	13	21	20
5	Elvin	14	22	23
6	Fletcher	2	10	48
7	Georgina	22	12	22
8	Mary	24	14	31
9	Tom	19	15	24
10	Zack	8	20	36

1. Find the class average for the subject ‘Physics’

AVG(PHYSICS)

15.125

2. Find the highest marks for mathematics (To be displayed as highest_marks_maths).

HIGHEST_MARKS_MATHS

48

3. Find the lowest marks for chemistry(To be displayed as lowest_mark_chemistry)

LOWEST_MARK_CHEMISRY

7

4. Find the total number of students who has got a ‘pass’ in physics.

COUNT_ROLLNO

8

5. Generate the list of students who have passed in all the subjects

ROLL_NO	NAME	PHYSICS	CHEMISTRY	MATHS
1	Adam	20	20	33
8	Mary	24	14	31

6. Generate a rank list for the class.Indicate Pass/Fail. Ranking based on total marks obtained by the students.

Roll No	Name	Physics	Chemistry	Maths	Total Marks	Result
1	Adam	20	20	33	73	P
8	Mary	24	14	31	69	P

2	Bob	18	9	41	68	F
10	Zack	8	20	36	64	F
3	Bright	22	7	31	60	F
6	Fletcher	2	10	48	60	F
5	Elvin	14	22	23	59	F
9	Tom	19	15	24	58	F
7	Georgina	22	12	22	56	F
4	Duke	13	21	20	54	F

7. Find pass percentage of the class for mathematics.

PASS_PERCENTAGE_MATHS

60

8. Find the overall pass percentage for all class.

PASS_PERCENTAGE

20

9. Find the class average.

CLASS_AVG

50.7

10. Find the total number of students who have got a Pass.

COUNT(RESULT)

2

RESULT

The query was executed successfully and output was obtained.

Exp No: 5

DATA CONSTRAINTS AND VIEWS

AIM

To study about various data constraints and views in SQL.

DESCRIPTION:

Using Referential Integrity Constraints

Whenever two tables are related by a common column (or set of columns), define a PRIMARY or

UNIQUE key constraint on the column in the parent table, and define a FOREIGNKEY constraint on the column in the child table, to maintain the relationship between the two tables.

Foreign keys can be comprised of multiple columns. However, a composite foreign key must reference a composite primary or unique key of the exact same structure (the same number of columns and datatypes). Because composite primary and unique keys are limited to 16 columns, a composite foreign key is also limited to 16 columns.

Nulls and Foreign Keys

By default (without any NOT NULL or CHECK clauses), and in accordance with the ANSI/ISO standard, the FOREIGNKEY constraint enforces the "match none" rule for composite foreign keys. The "full" and "partial" rules can also be enforced by using CHECK and NOTNULL constraints, as follows:

- To enforce the "match full" rule for nulls in composite foreign keys, which requires that all components of the key be null or all be non-null, define a CHECK constraint that allows only all nulls or all non-nulls in the composite

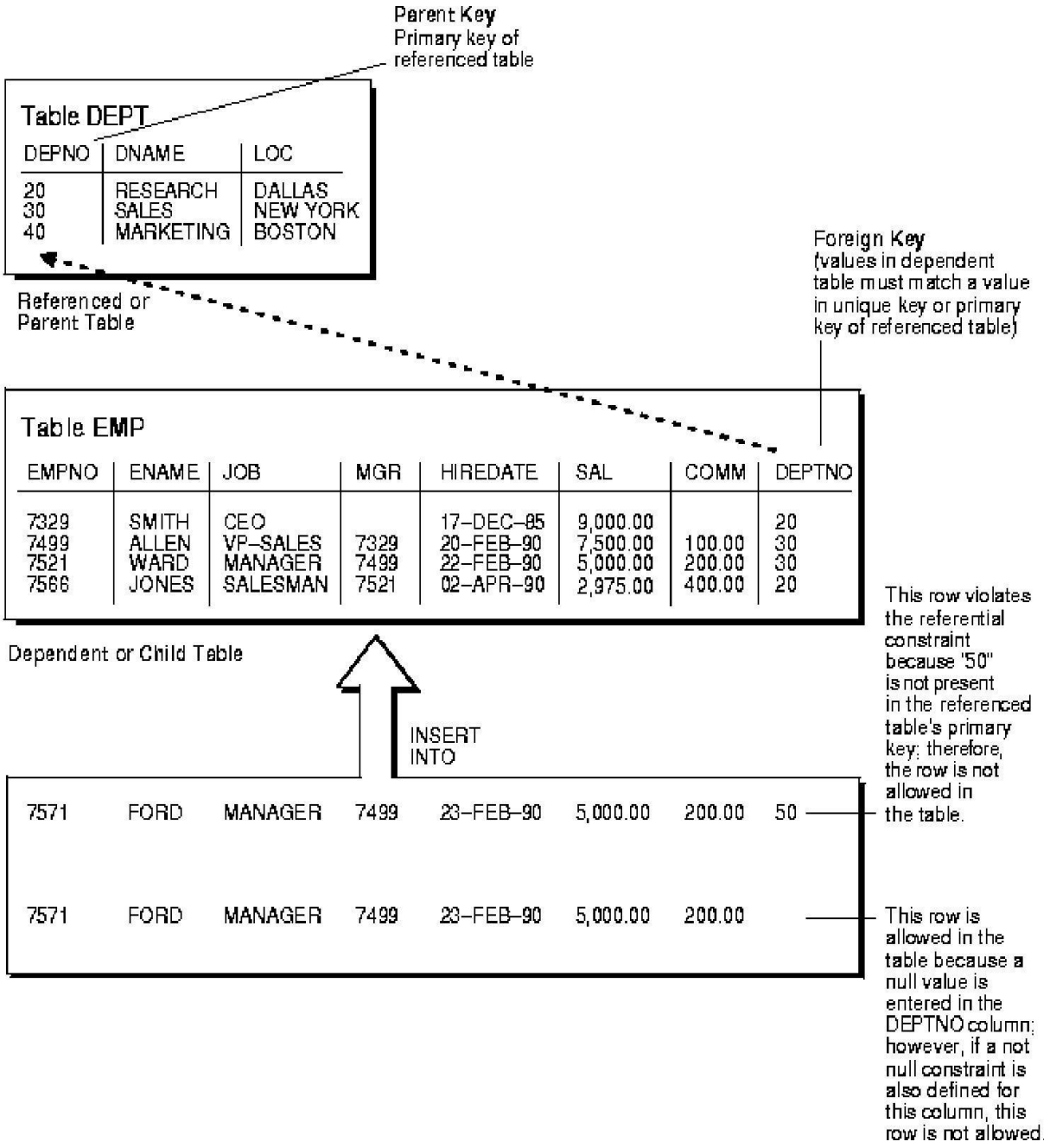
foreign key as follows, assuming a composite key comprised of columns A, B, and C:

CHECK ((A IS NULL AND B IS NULL AND C IS NULL) OR

(A IS NOT NULL AND B IS NOT NULL AND C IS NOT NULL))

- In general, it is not possible to use declarative referential integrity to enforce the "match partial" rule for nulls in composite foreign keys, which requires the non-null portions of the key to appear in the corresponding portions in the primary or unique key of a single row in the referenced table. You can often use triggers to handle this case.

Figure Referential Integrity Constraints:



Relationships Between Parent and Child Tables

Several relationships between parent and child tables can be determined by the other types of integrity constraints defined on the foreign key in the child table.

No Constraints on the Foreign Key

When no other constraints are defined on the foreign key, any number of rows in the child table can reference the same parent key value. This model allows nulls in the foreign key.

This model establishes a "one-to-many" relationship between the parent and foreign keys that allows undetermined values (nulls) in the foreign key. An example of such a relationship is shown in Figure 4-3 on page 8 between EMP_TAB and DEPT_TAB; each department (parent key) has many employees (foreign key), and some employees might not be in a department (nulls in the foreign key).

NOT NULL Constraint on the Foreign Key

When nulls are not allowed in a foreign key, each row in the child table must explicitly reference a value in the parent key because nulls are not allowed in the foreign key. However, any number of rows in the child table can reference the same parent key value.

This model establishes a "one-to-many" relationship between the parent and foreign keys. However, each row in the child table must have a reference to a parent key value; the absence of a value (a null) in the foreign key is not allowed. The same example in the previous section can be used to illustrate such a relationship. However, in this case, employees must have a reference to a specific department.

UNIQUE Constraint on the Foreign Key

When a UNIQUE constraint is defined on the foreign key, one row in the child table can reference a parent key value. This model allows nulls in the foreign key.

This model establishes a "one-to-one" relationship between the parent and foreign keys that allows undetermined values (nulls) in the foreign key. For example, assume that the EMP_TAB table had a column named MEMBERNO, referring to an employee's membership number in the company's insurance plan. Also, a table named INSURANCE has a primary key named MEMBERNO,

and other columns of the table keep respective information relating to an employee's insurance policy. The MEMBERNO in the EMP_TAB table should be both a foreign key and a unique key:

1. To enforce referential integrity rules between the EMP_TAB and INSURANCE tables (the FOREIGN KEY constraint)

2. To guarantee that each employee has a unique membership number (the UNIQUE key constraint)

UNIQUE and NOT NULL Constraints on the Foreign Key

When both UNIQUE and NOTNULL constraints are defined on the foreign key, only one row in the child table can reference a parent key value. Because nulls are not allowed in the foreign key, each row in the child table must explicitly reference a value in the parent key.

This model establishes a "one-to-one" relationship between the parent and foreign keys that does not allow undetermined values (nulls) in the foreign key. If you expand the previous example by adding a NOTNULL constraint on the MEMBERNO column of the EMP_TAB table, in addition to guaranteeing that each employee has a unique membership number, then you also ensure that no undetermined values (nulls) are allowed in the MEMBERNO column of the EMP_TAB table.

1. Create the following tables with given constraints

a. Create a table named Subjects with the given attributes

* Subid(Should not be NULL)

* Subname (Should not be NULL)

Populate the database. Make sure that all constraints are working properly.

SUB_ID	SUB_NAME
1	Maths
2	Physics
3	Chemistry
4	English

i) Alter the table to set subid as the primary key.

b. Create a table named Staff with the given attributes

-staffid (Should be UNIQUE)
-staffname
-dept
-Age (Greater than 22)
-Salary (Less than 35000)

Populate the database. Make sure that all constraints are working properly.

STAFF_ID	STAFF_NAME	DEPT	AGE	SALARY
1	John	Purchasing	24	30000
2	Sera	Sales	25	20000
3	Jane	Sales	28	25000

i)Delete the check constraint imposed on the attribute salary

ii)Delete the unique constraint on the attribute staffid

c. Create a table named Bank with the following attributes

-bankcode (To be set as Primary Key, type= varchar(3))

-bankname (Should not be NULL)

-headoffice

-branches (Integer value greater than Zero)

Populate the database. Make sure that all constraints are working properly.All constraints have to be set after creating the table.

BANKCODE	BANK NAME	HEADOFFICE	BRANCHOFFICE
AAA	SIB	Ernakulam	6
BBB	Federal	Kottayam	5
CCC	Canara	Trivandrum	3

d. Create a table named Branch with the following attributes

-branchid (To be set as Primary Key)

-branchname (Set Default value as ‘New Delhi’)

-bankid (Foreign Key:- Refers to bank code of Bank table)

i) Populate the database. Make sure that all constraints are working properly.

ii) During database population, demonstrate how the DEFAULT Constraint is satisfied.

BRANCH_ID	BRANCHNAME	BANKID
01	Kottayam	CCC
02	New Delhi	AAA

iii) Delete the bank with bank code ‘SBT’ and make sure that the corresponding entries are getting deleted from the related tables.

BANKCODE	BANK NAME	HEADOFFICE	Branch Office

AAA	SIB	Ernakulam	6
BBB	Federal	Kottayam	5
CCC	Canara	Trivandrum	3
SBT	Indian	Delhi	7

Branch ID	Branch Name	Bank ID
1	Kottayam	CCC
2	New Delhi	AAA
5	Calicut	SBT

After deletion

BANKCODE	BANK NAME	HEAD OFFICE	BRANCH OFFICE
AAA	SIB	Ernakulam	6
BBB	Federal	Kottayam	5
CCC	Canara	Trivandrum	3

BRANCH ID	BRANCH NAME	BANK ID
1	Kottayam	CCC
2	New Delhi	AAA

iv) Drop the Primary Key using ALTER command

BRANCH_ID	BRANCHNAME	BANKID
1	Kottayam	CCC
2	New Delhi	AAA
1	PPP	CCC

2. Create a View named sales_staff to hold the details of all staff working in sales Department

STAFF_ID	STAFF_NAME	DEPT	AGE	SALARY
1	Sera	Sales	25	20000
3	Jane	Sales	28	25000

3. Drop table branch. Create another table named branch and name all the constraints as given below:

Constraint name	Column	Constraint
Pk	branch_id	Primary key
Df	branch_name	Default :'New Delhi'
Fk	bankid	Foreign key/References

i) Delete the default constraint in the table

ii) Delete the primary key constraint

4. Update the view sales_staff to include the details of staff belonging to sales department whose salary is greater than 20000.

STAFF_ID	STAFF_NAME	DEPT	AGE	SALARY
3	Jane	Sales	28	25000

5. Delete the view sales_staff.

RESULT
The query was executed and the output was obtained.

Exp No: 6

STRING FUNCTIONS AND PATTERN MATCHING

AIM:

String Functions & Pattern Matching

- SUBSTR	- RPAD	- INITCAP
- INSTR	- LPAD	- CONCAT
- LTRIM	- UPPER	- LENGTH
- RTRIM	- LOWER	- REVERSE

Description

The main string functions are as follows

1. Length()
2. Lower()
3. Upper()
4. Concat()
5. Lpad()
6. Rpad()
7. Rtrim()
8. Instr()
9. Substr()

Length(field name/string)

Gives the length of the string.

Lower(field name/string)

Gives the content in lowercase letters

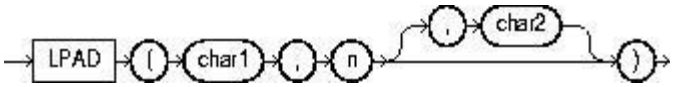
Upper(field name/string)

Gives the content in upper case letters

Concat (field name/string, field name/string)

Combines the first and second string into single one.

Lpad(field name/string,length,character)



Purpose

Returns char1, left-padded to length n with the sequence of characters in char2; char2 defaults to a single blank. If char1 is longer than n, this function returns the portion of char1 that fits in n.

The argument n is the total length of the return value as it is displayed on your terminal screen. In most character sets, this is also the number of characters in the return value. However, in some multibyte character sets, the display length of a character string can differ from the number of characters in the string.

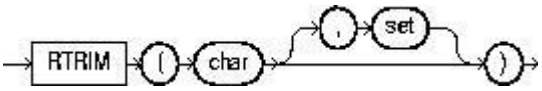
Rpad(field name/string,length,character)

Purpose

Returns char1, right-padded to length n with char2, replicated as many times as necessary; char2 defaults to a single blank. If char1 is longer than n, this function returns the portion of char1 that fits in n.

The argument n is the total length of the return value as it is displayed on your terminal screen. In most character sets, this is also the number of characters in the return value. However, in some multibyte character sets, the display length of a character string can differ from the number of characters in the string.

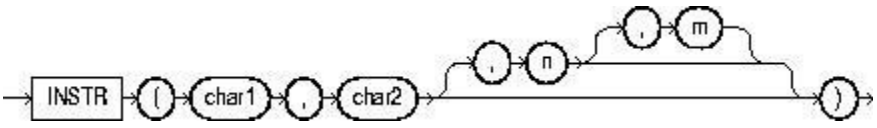
Rtrim(field name/string,substring)



Purpose

Returns char, with all the rightmost characters that appear in set removed; set defaults to a single blank. If char is a character literal, you must enclose it in single quotes. RTRIM works similarly to LTRIM.

Instr(field name/string,substring,n,m)

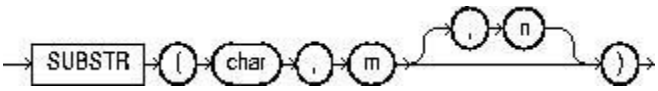


Purpose

Searches char1 beginning with its nth character for the nth occurrence of char2 and returns the position of the character in char1 that is the first character of this occurrence. If n is negative, Oracle counts and searches backward from the end of char1. The value of m must be positive. The default values of both n and m are 1, meaning Oracle begins searching at the first character of char1 for the first occurrence of char2. The return value is relative to the beginning of

char1, regardless of the value of n, and is expressed in characters. If the search is unsuccessful (if char2 does not appear m times after the nth character of char1) the return value is 0.

Substr(field name/string,n,m)



Purpose

Returns a portion of char, beginning at character m, n characters long.

1. If m is 0, it is treated as 1.

2. If m is positive, Oracle counts from the beginning of char to find the first character.

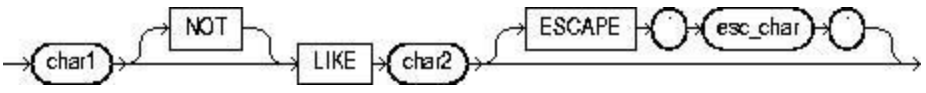
3. If m is negative, Oracle counts backwards from the end of char.

4. If n is omitted, Oracle returns all characters to the end of char. If n is less than 1, a null is returned.

Floating-point numbers passed as arguments to SUBSTR are automatically converted to integers.

LIKE Operator

The LIKE operator is used in character string comparisons with pattern matching. The syntax for a condition using the LIKE operator is shown in this diagram:



where:

char1 is a value to be compared with a pattern. This value can have datatype CHAR or VARCHAR2.

NOT logically inverts the result of the condition, returning FALSE if the condition evaluates to TRUE and TRUE if it evaluates to FALSE.

char2 is the pattern to which char1 is compared. The pattern is a value of datatype CHAR or VARCHAR2 and can contain the special pattern matching characters % and _.

ESCAPE identifies a single character as the escape character. The escape character can be used to cause Oracle to interpret % or _ literally, rather than as a special character.

If you wish to search for strings containing an escape character, you must specify this

character twice. For example, if the escape character is '/', to search for the string 'client/server', you must specify, 'client//server'.

Whereas the equal (=) operator exactly matches one character value to another, the LIKE operator matches a portion of one character value to another by searching the first value for the pattern specified by the second. Note that blank padding is not used for LIKE comparisons.

With the LIKE operator, you can compare a value to a pattern rather than to a constant. The pattern must appear after the LIKE keyword.

Patterns typically use special characters that Oracle matches with different characters in the value:

- An underscore (_) in the pattern matches exactly one character (as opposed to one byte in a multibyte character set) in the value.

- A percent sign (%) in the pattern can match zero or more characters (as opposed to bytes in a multibyte character set) in the value. Note that the pattern '%' cannot match a null.

Case Sensitivity and Pattern Matching:

Case is significant in all conditions comparing character expressions including

the LIKE and equality (=) operators. You can use the UPPER() function to perform a case-insensitive match, as in this condition:

UPPER(ename) LIKE 'SM%'

ESCAPE Option:

You can include the actual characters "%" or "_" in the pattern by using the ESCAPE option. The ESCAPE option identifies the escape character. If the escape character appears in the pattern before the character "%" or "_" then Oracle interprets this character literally in the pattern, rather than as a special pattern matching character.

Example:

To search for employees with the pattern 'A_B' in their name:

SELECT ename FROM empWHERE ename LIKE '%A_B%' ESCAPE '\';

The ESCAPE option identifies the backslash (\) as the escape character. In the pattern, the escape character precedes the underscore (_). This causes Oracle to interpret the underscore literally, rather than as a special pattern matching character.

A. Create a table named acct_details and populate the table as shown below.

Acct_No	Branch	Name	Phone
A40123401	Chicago	Mike Adams	(378) 400-1234
A40123402	Miami	Diana George	(372) 420-2345
B40123403	Miami	Diaz Elizabeth	(371) 450-3456
B40123404	Atlanta	Jeoffrey George	(370) 460-4567
B40123405	New York	Jennifer Kaitlyn	(373) 470-5678
C40123406	Chicago	Kaitlyn Vincent	(318) 200-3235
C40123407	Miami	Abraham Gottfield	(328) 300-2256
C50123408	New Jersey	Stacy Williams	(338) 400-5237

D50123409	New York	Catherine George	(348) 500-6228
D50123410	Miami	Oliver Scott	(358) 600-7230

SELECT *FROM acct_details

ACC_NO	BRANCH	NAME	PHONENO
A40123401	Chicago	Mike Adams	(378) 400-1234
A40123402	Miami	Diana George	(372) 420-2345
B40123403	Miami	Diaz Elizabeth	(371) 450-3456
B40123404	Atlanta	Jeoffrey George	(370) 460-4567
B40123405	New York	Jennifer Kaitlyn	(373) 470-5678
C40123406	Chicago	Kaitlyn Vincent	(318) 200-3235
C40123407	Miami	Abraham Gottfield	(328) 300-2256
C50123408	New Jersey	Stacy Williams	(338) 400-5237
D50123409	New York	Catherine George	(348) 500-6228
D50123410	Miami	Oliver Scott	(358) 600-7230

1. Find the names of all people starting on the alphabet ‘D’

NAME

Diana George
Diaz Elizabeth

2. List the names of all branches containing the substring ‘New’

BRANCH
New York

New Jersey
New York

3. List all the names in Upper Case Format

UPPER(NAME)

MIKE ADAMS

DIANA GEORGE

DIAZ ELIZABETH

JEOFFREY GEORGE

JENNIFER KAITLYN

KAITLYN VINCENT

ABRAHAM GOTTFIELD

STACY WILLIAMS

CATHERINE GEORGE

OLIVER SCOTT

4. List the names where the 4th letter is ‘n’ and last letter is ‘n’

NAME

Jennifer Kaitlyn

5. List the names starting on ‘D’ , 3rd letter is ‘a’ and contains the substring ‘Eli’

NAME

Diaz Elizabeth

6. List the names of people whose account number ends in ‘6’

NAME

Kaitlyn Vincent

7. Update the table so that all the names are in Upper Case Format

ACC_NO	BRANCH	NAME	PHONE NO
A40123401	Chicago	MIKE ADAMS	(378) 400-1234
A40123402	Miami	DIANA GEORGE	(372) 420-2345
B40123403	Miami	DIAZ ELIZABETH	(371) 450-3456
B40123404	Atlanta	JEOFFREY GEORGE	(370) 460-4567
B40123405	New York	JENNIFER KAITLYN	(373) 470-5678
C40123406	Chicago	KAITLYN VINCENT	(318) 200-3235
C40123407	Miami	ABRAHAM GOTTFIELD	(328) 300-2256
C50123408	New Jersey	STACY WILLIAMS	(338) 400-5237
D50123409	New York	CATHERINE GEORGE	(348) 500-6228
D50123410	Miami	OLIVER SCOTT	(358) 600-7230

8. List the names of all people ending on the alphabet ‘t’

NAME

KAITLYN VINCENT

OLIVER SCOTT

9. List all the names in reverse

REVERSE(NAME)

SMADA EKIM
EGROEG ANAID

HTEBAZILE ZAID

EGROEG YERFFOEJ

NYLTIAK REFINNEJ

TNECNIV NYLTIAK

DLEIFTTOG MAHARBA

SMAILLIW YCATS

EGROEG ENIREHTAC

TTOCS REVILO

10. Display all the phone numbers including US Country code (+1). For eg: (378)400-1234should be displayed as +1(378)400-1234. Use LPAD function

LPAD(PHONENO,16,'+1')

+1(378) 400-1234

+1(372) 420-2345

+1(371) 450-3456

+1(370) 460-4567

+1(373) 470-5678

+1(318) 200-3235

+1(328) 300-2256

+1(338) 400-5237

+1(348) 500-6228

+1(358) 600-7230

11. Display all the account numbers. The starting alphabet associated with the Account_Noshould be removed. Use LTRIM function.

ACC_NO	BRANCH	NAME	PHONE NO
4012340 1	Chicago	MIKE ADAMS	(378) 400-1234
4012340 2	Miami	DIANA GEORGE	(372) 420-2345
4012340 3	Miami	DIAZ ELIZABETH	(371) 450-3456
4012340 4	Atlanta	JEOFFREY GEORGE	(370) 460-4567
4012340 5	New York	JENNIFER KAITLYN	(373) 470-5678

12. Display the details of all people whose account number starts in ‘4’ and name contains the substring ‘Williams’.

ACC_NO	BRANCH	NAME	PHONE NO
5012340 8	New Jersey	STACY WILLIAMS	(338) 400-5237

B. Use the system table DUAL for the following questions:

1. Find the reverse of the string ‘ nmutuAotedOehT’
   REVERSE('NMUTUAOTEDOEHT')

The Ode to Autumn

2. Use LTRIM function on ‘123231xyzTech’ so as to obtain the output ‘Tech’
   LTRIM('123231XYZTECH','123XYZ')

Tech

3. Use RTRIM function on ‘Computer ‘ to remove the trailing spaces.
   RTRIM('COMPUTER')

Computer

4. Perform RPAD on ‘computer’ to obtain the output as ‘computerXXXX’
   RPAD('COMPUTER',12,'X')

   ComputerXXXX

5. Use INSTR function to find the first occurrence of ‘e’ in the string ‘Welcome to Kerala’

INSTR('WELCOME TO KERALA','E',1,1)

2

6. Perform INITCAP function on ‘mARKcALAwaY’

INITCAP('MARKCALAWAY')

Mark Calaway

7. Find the length of the string ‘Database Management Systems’.

LENGTH('DATABASE MANAGEMENT SYSTEMS')

27

8. Concatenate the strings ‘Julius’ and ‘Caesar’

CONCAT('JULIUS','CAESAR')

JuliusCaesar

9. Use SUBSTR function to retrieve the substring ‘is’ from the string ‘India is my country’.

   SUBSTR('INDIA IS MY COUNTRY',7,2)

   is

10. Use INSTR function to find the second occurrence of ‘k’ from the last. The string is ‘Making of a King’.

INSTR('MAKING OF A KING','K', -1,2)

0

RESULT
The query was executed and the output was obtained.

Exp No: 7

JOIN STATEMENTS, SET OPERATIONS, NESTED QUERIES AND GROUPING

AIM

To get introduced to

-UNION - JOIN
-INTERSECTION - NESTED QUERIES
-MINUS - GROUP BY & HAVING

Joins

A join is a query that combines rows from two or more tables, views, or materialized views ("snapshots"). Oracle performs a join whenever multiple tables appear in the query's FROM clause. The query's select list can select any columns from any of these tables. If any two of these tables have a column name in common, you must qualify all references to these columns throughout the query with table names to avoid ambiguity.

Join Conditions

Most join queries contain WHERE clause conditions that compare two columns, each from a different table. Such a condition is called a join condition. To execute a join, Oracle combines pairs of rows, each containing one row from each table, for which the join condition evaluates to TRUE. The columns in the join conditions need not also appear in the select list.

To execute a join of three or more tables, Oracle first joins two of the tables based on the join conditions comparing their columns and then joins the result to another table based on join conditions containing columns of the joined tables and the new table. Oracle continues this process until all tables are joined into the result. The optimizer determines the order in which Oracle joins tables based on the join conditions, indexes on the tables, and, in the case of the cost-based optimization approach, statistics for the tables.

In addition to join conditions, the WHERE clause of a join query can also contain

other conditions that refer to columns of only one table. These conditions can further restrict the rows returned by the join query.

Equijoins

An equijoin is a join with a join condition containing an equality operator. An equijoin combines rows that have equivalent values for the specified columns. Depending on the internal algorithm the optimizer chooses to execute the join, the total size of the columns in the equijoin condition in a single table may be limited to the size of a data block minus some overhead. The size of a data block is specified by the initialization parameter DB_BLOCK_SIZE.

Self Joins

A self join is a join of a table to itself. This table appears twice in the FROM clause and is followed by table aliases that qualify column names in the join condition. To perform a self join, Oracle combines and returns rows of the table that satisfy the join condition.

Cartesian Products

If two tables in a join query have no join condition, Oracle returns their Cartesian product. Oracle combines each row of one table with each row of the other. A Cartesian product always generates many rows and is rarely useful. For example, the Cartesian product of two tables, each with 100 rows, has 10,000 rows. Always include a join condition unless you specifically need a Cartesian product. If a query joins three or more tables and you do not specify a join condition for a specific pair, the optimizer may choose a join order that avoids producing an intermediate Cartesian product.

Outer Joins

An outer join extends the result of a simple join. An outer join returns all rows that satisfy the join condition and those rows from one table for which no rows from the other satisfy the join
condition. Such rows are not returned by a simple join. To write a query that performs an outer join of tables A and B and returns all rows from A, apply the outer join operator (+) to all columns of B in the join condition. For all rows in A that have no matching rows in B, Oracle returns NULL for any select list expressions containing columns of B.

Outer join queries are subject to the following rules and restrictions:

- The (+) operator can appear only in the WHERE clause or, in the context of left-correlation (that is, when specifying the TABLE clause) in the FROM clause, and can be

applied only to a column of a table or view.

- If A and B are joined by multiple join conditions, you must use the (+) operator in all of these conditions. If you do not, Oracle will return only the rows resulting from a simple

join, but without a warning or error to advise you that you do not have the results of an

outer join.

- The (+) operator can be applied only to a column, not to an arbitrary expression. However, an arbitrary expression can contain a column marked with the (+) operator.

- A condition containing the (+) operator cannot be combined with another condition using the OR logical operator.

- A condition cannot use the IN comparison operator to compare a column marked with the

(+) operator with an expression.

- A condition cannot compare any column marked with the (+) operator with a subquery.

If the WHERE clause contains a condition that compares a column from table B with a constant, the (+) operator must be applied to the column so that Oracle returns the rows from table A for which it has generated NULLs for this column. Otherwise Oracle will return only the results of a simple join.

In a query that performs outer joins of more than two pairs of tables, a single table can be the NULL-generated table for only one other table. For this reason, you cannot apply the (+) operator to columns of B in the join condition for A and B and the join condition for B and C.

Set Operators:

Set operators combine the results of two component queries into a single result. Queries containing set operators are called compound queries. Table lists SQL set operators.

Operator	Returns
UNION	All rows selected by either query.
UNION ALL	All rows selected by either query, including all duplicates.
INTERSECT	All distinct rows selected by both queries.
MINUS	All distinct rows selected by the first query but not the second

All set operators have equal precedence. If a SQL statement contains multiple set operators, Oracle evaluates them from the left to right if no parentheses explicitly specify another order.

The corresponding expressions in the select lists of the component queries of a compound query must match in number and datatype. If component queries select character data, the datatype of the return values are determined as follows:

- If both queries select values of datatype CHAR, the returned values have datatype

  CHAR.

- If either or both of the queries select values of datatype VARCHAR2, the returned

  values have datatype VARCHAR2.

Set Membership (IN & NOT IN):
NOT IN Operator

If any item in the list following a NOT IN operation is null, all rows evaluate to UNKNOWN (and no rows are returned). For example, the following statement returns the string 'TRUE' for each row:

SELECT 'TRUE'

FROM emp

WHERE deptno NOT IN (5,15);

However, the following statement returns no rows:

SELECT 'TRUE'

FROM emp

WHERE deptno NOT IN (5,15,null);

The above example returns no rows because the WHERE clause condition evaluates to:

deptno != 5 AND deptno != 15 AND deptno != null
Because all conditions that compare a null result in a null, the entire expression results in a null. This behavior can easily be overlooked, especially when the NOT IN operator references a subquery.

NESTED QUERIES

Subquery:

If a sql statement contains another sql statement then the sql statement which is inside another sql statement is called Subquery. It is also known as nested query. The Sql Statement which contains the other sql statement is called Parent Statement.

Nested Subquery:

If a Subquery contains another subquery, then the subquery inside another subquery is called nested subquery.

Correlated Subquery:

If the outcome of a subquery is depends on the value of a column of its parent query table then the Sub query is called Correlated Subquery.

QUESTION

Amazon is one of the largest online stores operating in the United States of America. They are maintaining four tables in their database. The Items table, Customers table, Orders table and Delivery table. Each of these tables contains the following attributes:

Items: - itemid (primary key) Itemname(type =varchar(50)) category

Price
Instock (type=int, greater than or equal to zero)

Customers:- custid (primary key) Custname

Address state

Orders:- orderid (primary key)
Itemid(refers to itemid of Items table)

Quantity (type=int)
Orderdate (type=date)

Delivery:- delivery id (primary key)
Custid (refers to custid in customers table)
Orderid (refers to ordered in orders table)

Create the above tables and populate them with appropriate data.

1. List the details of all customers who have placed an order

CUSTID	CUSTNAME	ADDRESS	STATE
111	elvin	202 jai street	delhi
113	soman	puthumana p.o	kerala
115	mickey	juhu	mumbai

2. List the details of all customers whose orders have been delivered

CUSTID	CUSTNAME	ADDRESS	STATE
115	mickey	juhu	mumbai
111	elvin	202 jai street	delhi
113	soman	puthumana p.o	kerala

3. Find the orderdate for all customers whose name starts in the letter ‘J’

ORDERDATE

22-DEC-14

4.Display the name and price of all items bought by the customer ‘Mickey’

ITEMNAME	PRICE
sony z5 premium	5005

5. List the details of all customers who have placed an order after January 2013 and not received delivery of items.

CUSTID	CUSTNAME	ADDRESS	STATE
114	jaise	kottarakara	kerala

6.Find the itemid of items which has either been ordered or not delivered. (Use SET

UNION)

ITEMID

1

3

64

4

5

7.Find the name of all customers who have placed an order and have their orders delivered.(Use SET INTERSECTION)

CUSTNAME

elvin

mickey

soman

8.Find the custname of all customers who have placed an order but not having their ordersdelivered. (Use SET MINUS)

CUSTNAME

jaise

9. Find the name of the customer who has placed the most number of orders.

CUSTID	CUSTNAME	ADDRESS	STATE
114	jaise	kottarakara	kerala

10. Find the details of all customers who have purchased items exceeding a price of 5000 $.

11.Find the name and address of customers who has not ordered a ‘Samsung Galaxy S4’

CUSTNAME	ADDRESS
Elvin	202 jai street
patrick	street 1 harinagar
Jaise	Kottarakara
mickey	juhu

12. Perform Left Outer Join and Right Outer Join on Customers & Orders Table.

CUSTID	CUSTNAME	ADDRESS	STATE	CUSTID	ITEMID	QUANTITY	ORDERDATE
111	elvin	202 jai street	delhi	111	1	2	11-OCT 14
113	soman	puthumanap.o	kerala	113	3	1	29-JAN-12
115	mickey	juhu	mumbai	115	5	1	16-MAY-13
114	jaise	kottarakara	kerala	114	4	3	22-DEC-14
112	Patrick	harinagar	Chennai	112	-	-	-

13. Find the details of all customers grouped by state

14.Display the details of all items grouped by category and having a price greater than the average price of all items.

ITEMID	ITEMNAME	CATEGORY	PRICE	INSTOCK

5	sony z5 premium	electronics	5005	1

RESULT

The query was executed and output was successfully obtained.

Exp No: 8

PL/SQL AND SEQUENCE

AIM

To study the basic pl/sql and sequence queries.

DESCRIPTION:

PL/SQL, Oracle's procedural extensions to SQL, is an advanced fourth-generation programming language (4GL). It offers modern features such as data encapsulation, overloading, collection types, exception handling, and information hiding. PL/SQL also offers seamless SQL access, tight integration with the Oracle server and tools, portability, and security.

Block Structure

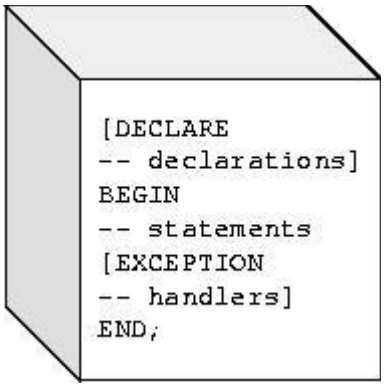
PL/SQL is a *block-structured* language. That is, the basic units (procedures, functions, and anonymous blocks) that make up a PL/SQL program are logical blocks, which can contain any number of nested sub-blocks. Typically, each logical block corresponds to a problem or subproblem to be solved. Thus, PL/SQL supports the divide-and-conquer approach to problem solving called *stepwise refinement*.

A block (or sub-block) lets you group logically related declarations and statements. That way, you can place declarations close to where they are used. The declarations are local to the block and cease to exist when the block completes.

As Figureshows, a PL/SQL block has three parts: a declarative part, an executable part, and an exception-handling part. (In PL/SQL, a warning or error condition is called an *exception*.) Only the executable part is required.

The order of the parts is logical. First comes the declarative part, in which items can be declared. Once declared, items can be manipulated in the executable part. Exceptions raised during execution can be dealt with in the exception-handling part.

Figure Block Structure:



You can nest sub-blocks in the executable and exception-handling parts of a PL/SQL block or subprogram but not in the declarative part. Also, you can define local subprograms in the declarative part of any block. However, you can call local subprograms only from the block in which they are defined.

Sequences

The sequence generator generates sequential numbers. Sequence number generation is useful to generate unique primary keys for your data automatically, and to coordinate keys across multiple rows or tables.

Without sequences, sequential values can only be produced programmatically. A new primary key value can be obtained by selecting the

most recently produced value and incrementing it. This method requires a lock during the transaction and causes multiple users to wait for the next value of the primary key; this waiting is known as *serialization.* If you have such constructs in your applications, then you should replace them with access to sequences. Sequences eliminate serialization and improve the concurrency of your application.

Example

To perform addition, subtraction, multiplication and division on two numbers.

```
DECLARE
        aint
        ;
        bin
        t;

BEGIN
        a:=:
        a;
        b:=:
        b;

        dbms_output.put_line('sum is'||
        (a+b));
        dbms_output.put_line('difference
        is'||(a-b));
        dbms_output.put_line('product
        is'||(a*b));
        dbms_output.put_line('qoutient
        is'||(a/b)); END

OUTPUT:
        A:30
        B:5


sum is35
difference is25
product is150
qoutient is6


        Statement processed.
```

QUESTIONS

Write PL/SQL programs for the following:

1. To print the first ‘n’ prime numbers.

OUTPUT:

2
3
5
7

11
13
17

19
23
29

31
37
41

43
47
53

59
61
67
71

Statement processed.

2. Display the Fibonacci series upto ‘n’ terms

OUTPUT
0
1

1

2
3

Statement processed.

3. Create a table named student_grade with the given attributes: roll, name ,mark1,mark2,mark3, grade. Read the roll, name and marks from the user. Calculate the grade of the student and insert a tuple into the table using PL/SQL. (Grade= 'PASS' if AVG >40, Grade ='FAIL' otherwi

select * from student_grade

ROLL	NAME	MARK1	MARK2	MARK3	GRADE
1	anu	50	45	48	PASS
2	manu	50	50	50	PASS
3	manu	35	40	40	FAIL

5. Create table circle_area (rad,area). For radius 5,10,15,20 &25., find the area and insert the corresponding values into the table by using loop structure in PL/SQL.

SELECT * from circle_area;

RADIUS	AREA
5	79
10	310
15	710
20	1300
25	2000

6. Use an array to store the names, marks of 10 students in a class. Using Loop structures in PL/SQL insert the ten tuples to a table named stud

SELECT * from stud

NAMES	MARKS
ARUN	25
AMAL	76
PETER	43

JOSE	45
ANNIE	67
MARY	57
JOSEPH	97
MARK	56
MIDHUN	89
KEVIN	8

7. Create a sequence using PL/SQL. Use this sequence to generate the primary key values for a table named class_cse with attributes roll,name and phone. Insert some tuples using PL/SQL programming.

ROLL	NAME	PHONE
1	ARUN	0482-239091
2	AMAL	0484-234562
3	PETER	0485-11234
4	JOSE	0489-43617
5	ANNIE	0481-23145

RESULT

The PL/SQL program was executed successfully and the output was obtained.

Exp No: 9

CURSOR

AIM

To study the use and implementation of cursors in PL/SQL.

DESCRIPTION

Oracle uses work areas to execute SQL statements and store processing information. A PL/SQL construct called a *cursor* lets you name a work area and access its stored information. There are two kinds of cursors: *implicit* and *explicit*. PL/SQL implicitly declares a cursor for all SQL data manipulation statements, including queries that return only one row. For queries that return more than one row, you can explicitly declare a cursor to process the rows individually.

QUESTIONS

Write PL/SQL programs for the following(using Cursors):

1. Create table student (id, name, m1, m2, m3, grade).Insert 5 tuples into it. Find the total, calculate grade and update the grade in the table.

create table stdnt(id int,sname varchar2(15),m1 int,m2

int,m3 int,gr char); insert into stdnt (id,sname,m1,m2,m3)

values(88,'anu',39,67,92);

1 row(s) inserted.

insert into stdnt (id,sname,m1,m2,m3) values(10,'jan',58,61,29);

1 row(s) inserted.

insert into stdnt (id,sname,m1,m2,m3) values(30,'karuna',87,79,77);

1 row(s) inserted.

insert into stdnt (id,sname,m1,m2,m3) values(29,'jossy',39,80,45);

1 row(s) inserted.

ID	SNAME	M1	M2	M3	GR
88	Anu	39	67	92	C
10	Jan	58	61	29	D
30	karuna	87	79	77	A
29	jossy	39	80	45	D

2. Create bank_details (accno, name, balance, adate).Calculate the interest of the amount and insert into a new table with fields (accno, interest). Interest= 0.08*balance.

create table bankdetails(accnoint,name varchar2(15),balanceint,adate date);

Table created.

insert into bankdetails values(1001,'aby',3005,'10-oct-15');

1 row(s) inserted.

insert into bankdetails values(1002,'alan',4000,'05-may-95');

1 row(s) inserted.

insert into bankdetails values(1003,'amal',5000,'16-mar-92');

1 row(s) inserted.

insert into bankdetails values(1004,'jeffin',3500,'01-apr-50');

1 row(s) inserted.

insert into bankdetails values(1005,'majo',6600,'01-jan-01');

1 row(s) inserted.

select * from bankdetails

ACCNO	NAME	BALANCE	ADATE
1001	Aby	3005	10-OCT-15
1002	Alan	4000	05-MAY-95
1003	Amal	5000	16-MAR-92
1004	Jeffin	3500	01-APR-50
1005	Majo	6600	01-JAN-01

create table banknew(accnoint,interesting)

Table created.

```
DECLARE

        cursor temp is select accno,name,balance,adate
        from bankdetails; tempvartemp%rowtype;

        intrin
t; BEGIN

        open
        temp;
        loop

        fetch temp into
        tempvar;
        intr:=.08*tempvar.ba
        lance;

        insert into banknew
        values(tempvar.accno,intr); exit when
        temp%notfound;

        end
        loop;
        close
        temp;
END;
```

select * from banknew

ACCNO	INTEREST
1001	240
1002	320
1003	400
1004	280
1005	528

3. Create table people_list (id, name, dt_joining, place).If person's experience is above 10 years, put the tuple in table exp_list (id, name, experience).

create table people_list(id INT, name varchar2(20),dt_joining DATE,place varchar2(20))

Table created.

create table exp_list(id INT, name varchar2(20),exp INT)

Table created.

select * from people_list

insert into people_list values(101,'Robert','03-APR-2005','CHY')

1 row(s) inserted.

insert into people_list values(102,'Mathew','07-JUN-2008','CHY')

1 row(s) inserted.

insert into people_list values(103,'Luffy','15-APR-2003','FSN')

1 row(s) inserted.

insert into people_list values(104,'Lucci','13-AUG-2009','KTM')

1 row(s) inserted.

insert into people_list values(105,'Law','14-APR-2005','WTC')

1 row(s) inserted.

insert into people_list values(101,'Vivi','21-SEP-2010','ABA')

1 row(s) inserted.

SELECT *FROM people_list;

ID	NAME	DT_JOINING	PLACE
102	Mathew	07-JUN-2008	CHY
103	Luffy	15-APR-2005	FSN
104	Lucci	13-AUG-2009	KTM
105	Law	12-APR-2005	WTC
101	Vivi	21-SEP-2010	ABA

select * from exp_list

ID	NAME	EXP
101	Robert	11
103	Luffy	12
105	Law	11

4. Create table employee_list(id,name,monthly salary). If:

annual salary<60000, increment monthly salary by 25% between 60000 and 200000, increment by 20%

between 200000 and 500000, increment by 15%
annual salary>500000, increment monthly salary by 10%

create table emp_list(id INT,Name varchar2(20),M_sal INT)

Table created.

insert into emp_list values(101,'Mathew',55000)

1 row(s) inserted

insert into emp_list values(102,'Jose',80000)
1 row(s) inserted

insert into emp_list values(103,'John',250000)

1 row(s) inserted

insert into emp_list values(104,'Ann',600000)

select * from emp_list

ID	NAME	M_SAL
101	Mathew	55000
102	Jose	80000
103	John	250000
104	Ann	600000

select * from emp_list

ID	NAME	M_SAL
101	Mathew	68750
102	Jose	96000
103	John	287500
104	Ann	660000

RESULT

The PL/SQL program was executed successfully and the output was obtained.

Exp No: 10

TRIGGER AND EXCEPTION HANDLING

AIM

To study PL/SQL trigger and exception handling.

DESCRIPTION:

TRIGGER

Triggers are procedures that are stored in the database and implicitly run, or *fired*, when something happens.

Syntax for writing triggers is: Create trigger <trigger name>
Before/after/instead of

insert/delete/update

of <column name> on <table name>

for each row
when
<condition>

<pl/sql block with declare—begin--end constructs>

EXCEPTION

It is used to handle run time errors in program

Syntax:

begin

<executable

statements>
exception

<exception
handling> end;

QUESTIONS

Write PL/SQL programs for the following:

Create a table customer_details (cust_id (unique) ,cust_name, address). Create a table emp_details(empid(unique), empname,salary)

Create table cust_count(count_row)

CREATE TABLE customer_details (cust_idintUNIQUE,cust_name varchar(25),address varchar(30));

Table created.

CREATE TABLE emp_details(empid INT UNIQUE,empname varchar(20),salary int)

Table created.

CREATE TABLE cust_count(count_rowint)

Table created.

insert into cust_countvaLUES(0)

1 row(s) inserted.

Example

1. Create a trigger whenever a new record is inserted in the customer_details table.

```
CREATE OR REPLACE TRIGGER trigger1 AFTER INSERT ON
customer_details FOR EACH ROW
BEGIN
        dbms_output.put_line('A row is inserted');

END;
```

Trigger created

INSERT INTO customer_details VALUES(1,'John','Ezhaparambbil')

A row is inserted

1 row(s) inserted.

2. Create a trigger to display a message when a user enters a value > 20000 in the salary

field of emp_details table.

INSERT INTO emp_details VALUES(1,'John',25000)

Employee has salary greater than 20000/-

1 row(s) inserted.

3. Create a trigger w.r.tcustomer_detailstable.Increment the value of count_row (in cust_count table) whenever a new tuple is inserted and decrement the value of count_row when a tuple is deleted. Initial value of the count_row is set to 0.

INSERT INTO customer_details VALUES(2,'Pretty','Thenganachalil')

1 row(s) inserted.

INSERT INTO customer_details VALUES(1,'John','Ezhaparambbil')

1 row(s) inserted.

select * from cust_count

COUNT_ROW

2

delete from customer_details where cust_id=1;

1 row(s) deleted.

select * from cust_count

COUNT_ROW
1

4. Create a trigger to insert the deleted rows from emp_details to another table and updated rows to another table. (Create the tables deleted and updatedT)

select * from emp_details

EMPID	EMPNAME	SALARY
1	John	25000
4	John	2000

UPDATE emp_details SET salary=salary+20000 WHERE empid=4;

1 row(s) updated.

select * from updated

EMPID	EMPNAME	SALARY
4	John	22000

DELETE FROM emp_details WHERE empid=1;

1 row(s) deleted.

select * from deleted

EMPID	EMPNAME	SALARY
1	John	25000

5. Write a PL/SQL to show divide by zero exception

OUTPUT:
3
84

9

3

Statement processed. 9
0

enter another divisor.
Statement processed.

6. Write a PL/SQL to show no data found exception

7. Create a table with ebill(cname,prevreading,currreading). If prevreading = currreading then raise an exception 'Data Entry Error'.

OUTPUT:

A: 4
B: 4

Data entry error

Statement peocessed.

A: 7
B: 8

select * from ebill

CNAME	PREVREADING	CURRREADING
melvy	7	8

RESULT

The PL/SQL program was executed successfully and the output was obtained.

Exp No. 11

PROCEDURES, FUNCTIONS AND PACKAGES

Functions

A function is a subprogram that computes a value. Functions and procedures are structured alike, except that functions have a RETURN clause. You write functions using the syntax

```
FUNCTION name [(parameter[, parameter, ...])] RETURN
  datatype IS [local declarations]

BEGIN

  executable
statements
[EXCEPTION

  exception
handlers] END
[name];
```

Procedures

A procedure is a subprogram that performs a specific action. You write procedures using the syntax

```
PROCEDURE name [(parameter[,
  parameter, ...])] IS [local declarations]

BEGIN

  executable
statements
[EXCEPTION

  exception
```

handlers] END
[name];

A procedure has two parts: the *specification* (*spec* for short) and the *body*. The procedure spec begins with the keyword PROCEDURE and ends with the procedure name or a parameter list.

Parameter declarations are optional. Procedures that take no parameters are written without parentheses.

The procedure body begins with the keyword IS and ends with the keyword END followed by an optional procedure name. The procedure body has three parts: a declarative part, an executable part, and an optional exception-handling part.

The declarative part contains local declarations, which are placed between the keywords IS and

BEGIN. The keyword DECLARE, which introduces declarations in an anonymous PL/SQL block, isnot used. The executable part contains statements, which are placed between the keywords

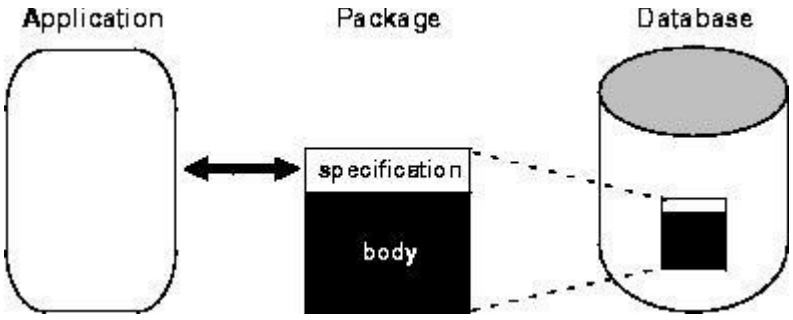
BEGIN and EXCEPTION (or END). At least one statement must appear in the executable part of aprocedure. The NULL statement meets this requirement. The exception-handling part contains exception handlers, which are placed between the keywords EXCEPTION and END.

Packages

A *package* is a schema object that groups logically related PL/SQL types, items, and subprograms. Packages usually have two parts, a specification and a body, although sometimes the body is unnecessary. The *specification* (*spec* for short) is the interface to your applications; it declares the types, variables, constants, exceptions, cursors, and subprograms available for use. The *body* fully defines cursors and subprograms, and so implements the spec.

As the figure shows, you can think of the spec as an operational interface and of the body as a "black box." You can debug, enhance, or replace a package body without changing the interface (package spec) to the package.

Figure Package Interface:



To create packages, use the CREATE PACKAGE statement, which you can execute interactively from SQL*Plus. Here is the syntax:

```
CREATE [OR REPLACE] PACKAGE
  package_name [AUTHID {CURRENT_USER |
  DEFINER}] {IS | AS} [type_definition
  [type_definition] ...]

  [cursor_spec [cursor_spec] ...]
  [item_declaration
  [item_declaration] ...]

  [{subprogram_spec | call_spec} [{subprogram_spec |
call_spec}]...] END [package_name];


[CREATE [OR REPLACE] PACKAGE BODY package_name {IS
  | AS} [type_definition [type_definition] ...]
```

[cursor_body [cursor_body] ...]
[item_declaration
[item_declaration] ...]

[{subprogram_spec | call_spec} [{subprogram_spec |
call_spec}]...] [BEGIN

sequence_of_state
ments] END
[package_name];]

The spec holds *public* declarations, which are visible to your application. The body holds implementation details and *private* declarations, which are hidden from your application. Following the declarative part of the package body is the optional initialization part, which typically holds statements that initialize package variables.

QUESTIONS

1. Create a function factorial to find the factorial of a number. Use this function in a PL/SQL Program to

display the factorial of a number read from the user.

OUTPUT:

NUM: 5
Factorial 5 is 120

Statement processed.

2. Create a table student_details(roll int,marksint, phone int). Create a procedure pr1 toupdate all rows in the database. Boost the marks of all students by 5%.

CREATE TABLE student_details(roll int,marksint, phone int);

Table created.

INSERT INTO student_detailsVALUES(1,70,9496947423);

1 row(s) inserted.

INSERT INTO student_detailsVALUES(2,85,9495941358);

1 row(s) inserted.

INSERT INTO student_detailsVALUES(3,78,8281865009);

1 row(s) inserted.

SELECT * FROM student_details;

ROLL	MARKS	PHONE	ROLL
1	70	9496947423	1
2	85	9495941358	2
3	78	8281865009	3

SELECT * FROM student_details;

ROLL	MARKS	PHONE
1	74	9496947423
2	89	9495941358
3	82	8281865009

3. Create table student (id, name, m1, m2, m3, total, grade).Create a function f1 to calculate grade. Create a procedure p1 to update the total and grade.

- **Read id,name,m1,m2,m3 from the user**
- **Insert the tuple into the database**
- **Using function f1 calculate the grade**

Using procedure p1, update the grade value for the tuple

CREATE TABLE student(id int, name varchar(10), m1 int, m2 int, m3 int, total int, grade varchar(1));

4. Create a package pk1 consisting of the following functions and procedures

- **Procedure proc1 to find the sum, average and**

product of two numbers

- **Procedure proc2 to find the square root of a number**
- **Function named fn11 to check whether a number is even or not**
- **A function named fn22 to find the sum of 3 numbers**

Use this package in a PL/SQL program. Call the functions f11, f22 and procedures pro1, pro2 within the program and display their results.

OUTPUT:
Sum:9

AVG:4.5
Product:18

Square root of 16 is 4 7 is
odd
Sum of 3,6 and 7 is 16

RESULT
The pl/sql program was executed and the output was obtained.