Guru Nanak Dev Engineering College

Practical File

Relational Database Management Systems Laboratory(PGCA-2208)

Submitted To:
Prof. Satinderpal Singh

Associate Professor

Submitted by:
Sukhpreet Singh

University Roll no. 2303308
(MCA 1st Semester)
Batch 2023-2025

Department of Computer Application

INDEX

Practical-1

Comparative study of various Database Management Systems

What is a database management system: A Database Management System (DBMS) is a specialized software designed to store, retrieve, and manipulate data. It acts as a mediator between the database, applications, and user interfaces to manage and organize data effectively. The system provides a comprehensive suite of tools to govern databases, ensuring data security, consistency, and integrity.

Data modeling: A DBMS provides tools for creating and modifying data models, which define the structure and relationships of the data in a database.

Data storage and retrieval: A DBMS is responsible for storing and retrieving data from the database, and can provide various methods for searching and querying the data.

Concurrency control: A DBMS provides mechanisms for controlling concurrent access to the database, to ensure that multiple users can access the data without conflicting with each other.

Data integrity and security: A DBMS provides tools for enforcing data integrity and security constraints, such as constraints on the values of data and access controls that restrict who can access the data.

Backup and recovery: A DBMS provides mechanisms for backing up and recovering the data in the event of a system failure.

RELATIONAL VS NON-RELATIONAL DATABASES

Feature	Relational databases (SQL)	Non-relational databases (NoSQL)

Data structure	• Organize data into tables with rows and columns
	• Strict schema
	• Data resides in records and attributes
		• Use different data models like
		  ○ document-oriented,
		  ○ key-value,
		  ○ graph,
		  ○ wide-column
		• Store unstructured data
		• No fixed schema

Language	Structured Query Language (SQL)	Various query languages depending on the data model
Scalability	• Scale vertically (more computer power to a single server) • Horizontal scaling is challenging and requires additional effort	• Scale horizontally (add more servers) • Share data between servers, decreasing the request-per-second rate in each server
Performance	• Perform well with intensive read/write operations on small to medium datasets • Can suffer when data and user requests grow	• High performance with distributed design • Provide simultaneous access to a large number of users • Store unlimited data sets in various formats
Security	• The integrated structure provides better security • ACID compliance is preferred for applications where database integrity is critical	• Generally weaker security • ACID guarantees are often limited to a single database partition • Some DBMSs offer advanced security features for compliance.
Use cases	Complex software solutions, eCommerce, financial applications	Storing and scaling unstructured data, MVPs for startups, sprint-based Agile development

altexsoft

Practical-2

Data Definition Language (DDL), Data Manipulation Language (DML), and Data Control Language (DCL)

Data Definition Language(DDL)

Data Definition Language (DDL) is a subset of SQL. It is a language for describing data and its relationships in a database. You can generate DDL in a script for database objects to: Keep a snapshot of the database structure. Set up a test system where the database acts like the production system but contains no data.

DDL Commands:

- Create
- Alter
- truncate
- drop
- Rename

CREATE: This command is used to create a new table in SQL. The user has to give information like table name, column names, and their data types.

```
CREATE TABLE table_name
(
column_1 datatype,
column_2 datatype,
column_3 datatype,
....
);
```

ALTER

This command is used to add, delete or change columns in the existing table. The user needs to know the existing table name and can do add, delete or modify tasks easily.

ALTER TABLE table_name
ADD column_name datatype;

Data Manipulation Language(DML):

DML is an abbreviation for Data Manipulation Language. Represents a collection of programming languages explicitly used to make changes to the database, such as: CRUD operations to create, read, update and delete data. Using INSERT, SELECT, UPDATE, and DELETE commands.

DML Commands:

- INSERT
- SELECT
- UPDATE
- DELETE

Data Control Language(DCL):

Data Control Language (or DCL) consists of statements that control security and concurrent access to table data. Instructs the XDB Server to make permanent all data changes resulting from DML statements executed by a transaction. Connects the application process (or user) to a designated XDB Server or DB2 location.

DCL Commands:

- GRANT
- REVOKE
- COMMIT
- ROLLBACK

Practical-3

How to apply Constraints at various levels

Introduction

SQL constraints can be at a column or a table level. Column level constraints apply to specific columns in a table and do not specify a column name except the check constraints. They refer to the column that they follow. The names are specified by the Table-level constraints of the columns to which they apply.

The available constraints in SQL are:

- **NOT NULL:** This constraint tells us that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more.
- **UNIQUE:** This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated.
- **PRIMARY KEY:** A primary key is a field which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as the primary key.
- **FOREIGN KEY:** A Foreign key is a field which can uniquely identify each row in another table. And this constraint is used to specify a field as Foreign key.
- **CHECK:** This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition.
- **DEFAULT:** This constraint specifies a default value for the column when no value is specified by the user.

Primary Key Constraints

```
CREATE TABLE TableName (
    column1 datatype PRIMARY KEY,
    column2 datatype,
    -- other columns and constraints
);
```

```
ALTER TABLE TableName
ADD PRIMARY KEY (column1);
```

Foreign key Constraints

```
CREATE TABLE Table1 (
    column1 datatype PRIMARY KEY,
    -- other columns
);

CREATE TABLE Table2 (
    column2 datatype,
    foreign_key_column datatype,
    FOREIGN KEY (foreign_key_column) REFERENCES Table1(column1)
);
```

```
ALTER TABLE Table2
ADD CONSTRAINT fk_constraint
FOREIGN KEY (foreign_key_column) REFERENCES Table1(column1);
```

Unique Constraints

```sql
CREATE TABLE TableName (
    column1 datatype UNIQUE,
    -- other columns and constraints
);
```

```sql
ALTER TABLE TableName
ADD CONSTRAINT unique_constraint
UNIQUE (column1);
```

Default Constraints

```sql
CREATE TABLE TableName (
    column1 datatype DEFAULT default_value,
    -- other columns and constraints
);
```

```sql
ALTER TABLE TableName
ADD CONSTRAINT default_constraint
DEFAULT default_value FOR column1;
```

Check Constraints

```sql
ALTER TABLE TableName
ADD CONSTRAINT check_constraint
CHECK (column1 > 0);
```

Practical-4

View data in the required form using Operators, Functions and Joins

Using comparison operators with joins: In the lessons so far, you've only joined tables by exactly matching values from both tables. However, you can enter any type of conditional statement into the ON clause.

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    customer_name VARCHAR(50),
    -- other columns
);

CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    total_amount DECIMAL(10, 2),
    -- other columns
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

```
SELECT
    orders.order_id,
    orders.order_date,
    orders.total_amount,
    customers.customer_id,
    customers.customer_name
FROM
    orders
JOIN
    customers ON orders.customer_id = customers.customer_id;
```

Practical-5

Creating different types of Views for tailored presentation of data

SQL has a special version of tables called View, which is a virtual table that is compiled in runtime. A View is just an SQL statement, and the data associated with it is not physically stored in the view but is stored in the base tables of it.

Types of Views

There are two types of views in the SQL Server, namely System Defined Views and User Defined Views. This section contains a description of these two types.

System Defined Views: The System Defined Views are predefined views that already exist in the SQL Server database, such as Tempdb, Master, and temp. Each of the databases has its own properties and functions.

Information Schema: There are twenty different schema views in the SQL server. They are used to display the physical information of the database, such as tables, constraints, columns, and views. This view starts with INFORMATION_SCHEMA and is followed by the View Name. INFORMATION_SCHEMA.CHECK_CONSTRAINTS is used to receive information about any constraint available in the database.

Catalog View : These are used to return information used by the SQL server. Catalog views provide an efficient way to obtain, present, and transform custom forms of information. But they do not include any information about backup, replication, or maintenance plans, etc. These views are used to access metadata of databases, and the names and column names are descriptive, helping a user to query what is expected.

Dynamic Management: View These were introduced in the SQL server in 2005. The administrator can get information about the server state to diagnose problems, monitor the health of the server instance, and tune performance through these views. The Server-scoped Dynamic Management View is only stored in the Master database, whereas the Database-scoped Dynamic Management View is stored in each database.
User Defined Views These are the types of views that are defined by the users. There are two types under User Defined views, Simple View and Complex View.

Simple View: These views can only contain a single base table or can be created only from one table. Group functions such as MAX(), COUNT(), etc., cannot be used here, and it does not contain groups of data.

Complex View: These views can contain more than one base table or can be constructed on more than one base table, and they contain a group by clause, join conditions, an order by clause. Group functions can be used here, and it contains groups of data. Complex views cannot always be used to perform DML operations.

Basic View

```
CREATE VIEW basic_view AS
SELECT
    column1,
    column2
FROM
    your_table;
```

Joined View

```sql
CREATE VIEW joined_view AS
SELECT
    t1.column1,
    t2.column2
FROM
    table1 t1
JOIN
    table2 t2 ON t1.id = t2.id;
```

Aggregated View

```sql
CREATE VIEW aggregated_view AS
SELECT
    category,
    AVG(price) AS avg_price,
    MAX(quantity) AS max_quantity
FROM
    products
GROUP BY
    category;
```

Security View

```sql
CREATE VIEW security_view AS
SELECT
    sensitive_column1,
    non_sensitive_column2
FROM
    secure_table;
```

Practical-6

How to apply Conditional Controls in PL/SQL

Conditional Control: IF Statements. Often, it is necessary to take alternative actions depending on circumstances. The IF statement lets you execute a sequence of statements conditionally. That is, whether the sequence is executed or not depends on the value of a condition.

PL/SQL If

PL/SQL supports the programming language features like conditional statements and iterative statements. Its programming constructs are similar to how you use it in programming languages like Java and C++.

Syntax: (IF-THEN statement):

```
IF condition
THEN
Statement: {It is executed when condition is true}
END IF;
```

Syntax: (IF-THEN-ELSE statement):

```
IF condition
THEN
  {...statements to execute when condition is TRUE...}
ELSE
  {...statements to execute when condition is FALSE...}
END IF;
```

Syntax: (IF-THEN-ELSIF statement):

```
IF condition1
THEN
  {...statements to execute when condition1 is TRUE...}
ELSIF condition2
THEN
  {...statements to execute when condition2 is TRUE...}
END IF;
```

Syntax: (IF-THEN-ELSIF-ELSE statement):

```
IF condition1
THEN
  {...statements to execute when condition1 is TRUE...}
ELSIF condition2
THEN
  {...statements to execute when condition2 is TRUE...}
ELSE
  {...statements to execute when both condition1 and condition2 are FALSE...}
END IF;
```

Example of PL/SQL If Statement

```
DECLARE
  a number(3) := 500;
BEGIN
  -- check the boolean condition using if statement
  IF( a < 20 ) THEN
    -- if condition is true then print the following
    dbms_output.put_line('a is less than 20 ' );
  ELSE
    dbms_output.put_line('a is not less than 20 ' );
  END IF;
  dbms_output.put_line('value of a is : ' || a);
END;
```

```
a is not less than 20
value of a is : 500
PL/SQL procedure successfully completed.
```

Practical-7

Error Handling using Internal Exceptions and External Exceptions

Handling Errors in PL/SQL

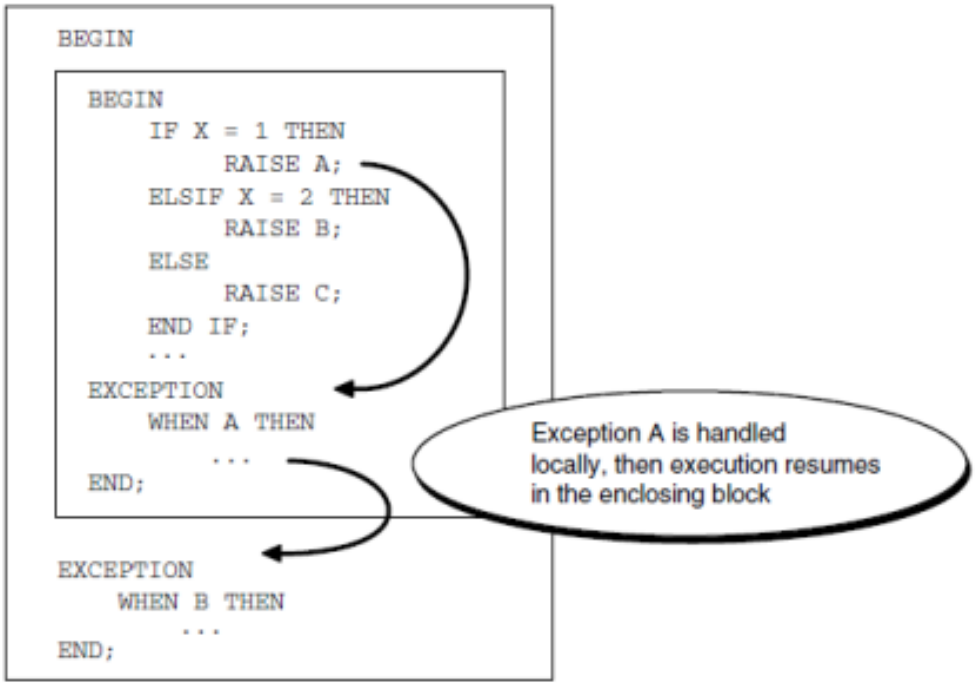
In PL/SQL, an error condition is called an exception. Exceptions can be internally defined (by the runtime system) or user-defined. Examples of internally defined exceptions include division by zero and out of memory.

To handle raised exceptions, you write separate routines called exception handlers. After an exception handler runs, the current block stops executing and the enclosing block resumes with the next statement. If there is no enclosing block, control returns to the host environment.

```
DECLARE
stock_price NUMBER := 9.73;
net_earnings NUMBER := 0;
pe_ratio NUMBER;
BEGIN
-- Calculation might cause division-by-zero error.
pe_ratio := stock_price / net_earnings;
DBMS_OUTPUT.PUT_LINE('Price/earnings ratio = ' || pe_ratio);
EXCEPTION -- exception handlers begin
-- Only one of the WHEN blocks is executed.
WHEN ZERO_DIVIDE THEN -- handles 'division by zero' error
DBMS_OUTPUT.PUT_LINE('Company must have had zero earnings.');
pe_ratio := NULL;
WHEN OTHERS THEN -- handles all other errors
DBMS_OUTPUT.PUT_LINE('Some other kind of error occurred.');
pe_ratio := NULL;
END; -- exception handlers and block end here
/
```

PL/SQL Exceptions Propagate

When an exception is raised, if PL/SQL cannot find a handler for it in the current block or subprogram, the exception propagates. That is, the exception reproduces itself in successive enclosing blocks until a handler is found or there are no more blocks to search. If no handler is found, PL/SQL returns an unhandled exception error to the host environment.



Handling Raised PL/SQL Exceptions

When an exception is raised, normal execution of your PL/SQL block or subprogram stops and control transfers to its exception-handling part, which is formatted as follows:

```
EXCEPTION
WHEN exception1 THEN -- handler for exception1
sequence_of_statements1
WHEN exception2 THEN -- another handler for exception2
sequence_of_statements2
...
WHEN OTHERS THEN -- optional handler for all other errors
sequence_of_statements3
END;
```

Internally defined exceptions (ORA-n errors) are described in Oracle Database Error Messages Reference. The runtime system raises them implicitly (automatically). An internally defined exception does not have a name unless either PL/SQL gives it one (see "Predefined Exceptions") or you give it one.

External Exception EEFFACE is an error message that appears when a C++ exception is not caught by C++ exception handlers, and the exception is eventually handled by Delphi exception handlers.

Practical-8

Using various types of Cursors

Cursor is a Temporary Memory or Temporary Work Station. It is Allocated by Database Server at the Time of Performing DML(Data Manipulation Language) operations on the Table by the User. Cursors are used to store Database Tables.

There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors. These are explained below.

1. Implicit Cursors: Implicit Cursors are also known as Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.
2. Explicit Cursors: Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner.

How To Create Explicit Cursor?

1. **Declare Cursor Object**

Syntax:

```
DECLARE cursor_name CURSOR FOR SELECT * FROM table_name
```

Query:

```
DECLARE s1 CURSOR FOR SELECT * FROM studDetails
```

2. Open Cursor Connection

Syntax:

OPEN cursor_connection

Query:

```
OPEN s1
```

Fetch Data from the Cursor There are a total of 6 methods to access data from the cursor. They are as follows:

1. FIRST is used to fetch only the first row from the cursor table.
2. LAST is used to fetch only the last row from the cursor table.
3. NEXT is used to fetch data in a forward direction from the cursor table.
4. PRIOR is used to fetch data in a backward direction from the cursor table.
5. ABSOLUTE n is used to fetch the exact n^{th} row from the cursor table.
6. RELATIVE n is used to fetch the data in an incremental way as well as a decremental way.

Syntax:

FETCH NEXT/FIRST/LAST/PRIOR/ABSOLUTE n/RELATIVE n FROM cursor_name

Query:

```
FETCH FIRST FROM s1
FETCH LAST FROM s1
FETCH NEXT FROM s1
FETCH PRIOR FROM s1
FETCH ABSOLUTE 7 FROM s1
FETCH RELATIVE -2 FROM s1
```

Close cursor connection

Syntax:

CLOSE cursor_name

Query:

```
CLOSE s1
```

Practical-9

How to run Stored Procedures and Functions

Step 1. Create a table or use an existing table. In my case, I use a table that has columns, username, and password. You also need to know how to create and execute a stored procedure. If you're new to SPs, read this before this article,

Step 2. The create function SQL query is used to create a new function. Create a function using the given query.

```
CREATE FUNCTION function_to_be_called (@username VARCHAR(200))
RETURNS VARCHAR(100)
AS
BEGIN
    DECLARE @password VARCHAR(200)
    SET @password = (SELECT [password] FROM [User] WHERE username = @username)
    RETURN @password
END
```

Step 3. The create procedure query is used to create a stored procedure. Create a procedure using the given query.

```
CREATE PROCEDURE chek_pass
    @user VARCHAR(200)
AS
BEGIN
    DECLARE @pass VARCHAR(200)
    SET @pass = dbo.function_to_be_called(@user)
    SELECT @pass
END
```

Step 4. Test it. Use exe to execute a stored procedure.

```
exec chek_pass 'username'
```

Practical-10

Creating Packages and applying Triggers

Triggers

Triggers provide a way of executing PL/SQL code on the occurrence of specific database events. For example, you can maintain an audit log by setting triggers to fire when insert or update operations are carried out on a table. The insert and update triggers add an entry to an audit table whenever the table is altered.

Packages

Packages are PL/SQL constructs that enable the grouping of related PL/SQL objects, such as procedures, variables, cursors, functions, constants, and type declarations. Informix Dynamic Server does not support the package construct.

A package can have two parts: a specification and a body. The specification defines a list of all objects that are publicly available to the users of the package. The body defines the code that is used to implement these objects, such as, the code behind the procedures and functions used within the package.

Stored Procedures

Stored procedures provide a powerful way to code application logic that can be stored on the server. Informix Dynamic Server and Oracle both use stored procedures. Oracle also uses an additional type of subprogram called a function.

Creating Triggers

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

PL/SQL Trigger Example

Let's take a simple example to demonstrate the trigger. In this example, we are using the following CUSTOMERS table:

Create table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Practical-11

Creating Arrays and Nested Tables.

Nested Tables

In the database, a nested table is a column type that stores an unspecified number of rows in no particular order.

```
DECLARE
  TYPE Roster IS TABLE OF VARCHAR2(15);  -- nested table type

  -- nested table variable initialized with constructor:

  names Roster := Roster('D Caruso', 'J Hamil', 'D Piro', 'R Singh');

  PROCEDURE print_names (heading VARCHAR2) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(heading);

    FOR i IN names.FIRST .. names.LAST LOOP  -- For first to last element
      DBMS_OUTPUT.PUT_LINE(names(i));
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('---');
  END;

BEGIN
  print_names('Initial Values:');

  names(3) := 'P Perez';  -- Change value of one element
  print_names('Current Values:');

  names := Roster('A Jansen', 'B Gupta');  -- Change entire table
  print_names('Current Values:');
END;
/
```

Result:

```
Initial Values:
D Caruso
J Hamil
D Piro
R Singh
---
Current Values:
D Caruso
J Hamil
P Perez
R Singh
---
Current Values:
A Jansen
B Gupta
```

Varrays

The Varrays store a fixed number of elements and are like a one-dimensional array. However, the number of elements can be modified at runtime. It is a consecutive set of elements of similar data types. It can be stored in a database table that can be handled with the help of SQL statements. But the process is not as easy and flexible as in a nested table.

Syntax of Varray:

```
TYPE <<type>> IS {VARRAY | VARYING ARRAY} (<<size>>)
 OF <<element>> [NOT NULL];
```

Varray Variables Declaration And Initialization

After creating a Varray, we can declare it in the way described below:

Syntax:

```
name type_n [:= type_n(...)];
```

Here,
‘name’ is the Varray name.
‘type_n’ is the type of Varray.
‘type_n(…)’ is the constructor of type Varray. The argument lists are mentioned by a comma separator and of type Varray.

Syntax:

```
name type_n := type_n();
```

This will initialize the variable with zero elements. In order to populate elements in the varray variables, **the syntax is:**

```
name type_n := type_n(e1, e2, ...);
```