COMPUTER SCIENCE & ENGINEERING DEPARTMENT

DBMS LAB MANUAL

A3 REGULATION

Author:

Mrs ArunaSrinivas, Professor , MVGRCE, VZM.

Version:

2.2

Revision:

Revised as per A3 regulation and made other minor changes

M V G R COLLEGE OF ENGINEERING

(Permanently Affiliated to JNTU-K& Approved by AICTE New Delhi)

Vijayaram Nagar Campus, Chintalavalasa,

VIZIANAGARAM. PIN: 535003.

INDEX

LAB OBJECTIVES

A vast majority of the software applications being built use some kind of a data store mechanism to permanently store the data generated/used by the software. Database management systems are complex software systems that provide a wide variety of functionality for users to structure, organize and access data quickly and efficiently. Database systems have been there since the early 1970's but have grown in complexity and size. Relational database management systems use relational algebra as the basis for representation of data. RDBMS as relational database management systems are the most popular and widely used DBMS in the market place.

In this lab, you would be exposed to one popular RDBMS. The broad one line objective of the lab is to get familiar with the functionality and support provided by commercially popular RDBMS and understand how to use it to meet your data storage and organization requirements.

Detailed objectives of the lab are:

- Students will learn SQL (Structured Query Language) which would provide functionality to:
    - Learn how to create tables which are fundamental storage blocks of data.
    - Learn how to place constraints on data that is entered on tables to ensure data integrity.
    - Learn how to add, change and remove data from tables.
    - Learn how to select a subset of the data you want to see from the collection of tables and data.
    - Learn how to combine table and group multiple rows of data in table.
- Students will study and learn PL/SQL which would provide the ability to do iterative programming at database level to:
    - Write programming blocks with conditionals, assignments, loops, etc
    - Exception Handling.
    - Transaction oriented programs
    - Stored procedures, functions, packages.
    - Cursors which would allow row wise access of data.
    - Triggers which would allow you define pre and post actions when something changes in the database tables.

- At the end of the lab, student should be comfortable using any popular RDBMS for data access and updating and should be comfortable writing PL/SQL programs at database level using Oracle.

Lab Resource Requirements:

Software Requirements: ORACLE 9I,D2K.

Hardware Requirements:

- Desktops with 1GB RAM, 200GB Hard Disk, Pentium+ Processor.

List of Experiments	
Unit – I	
Experiment – 1	Creation of tables
Experiment – 2	Alter table with changes in columns
Experiment – 3	Alter table with constraints
Experiment – 4	Dropping Tables
Experiment – 5	Insert, Update & Delete Data into Tables
Experiment – 6	Simple SELECT queries
Experiment – 7	Selection with conditions
Experiment – 8	Using Character functions
Experiment – 9	Using number functions
Experiment – 10	Using date functions
Experiment – 11	single row sub-queries
Experiment – 12	Multiple row sub-queries
Experiment – 13	Equal joins
Experiment – 14	Correlated sub-queries
Experiment – 15	Aggregate functions
Experiment – 16	Grouping clauses
Experiment – 17	Select groups with having
Experiment – 18	Union/Intersection statements
Experiment – 19	Creating and dropping views
Additional Exercises for Unit – I	
Unit – II	

MODEL LAB PLAN

Introduction:

Week 1: Oracle 9i setup & connecting to Oracle from iSQL*Plus.

Unit – I (SQL):

Week 2: Creating, Altering, Dropping tables with Constraints, Insert Table.

- Experiment 1: Create Tables
- Experiment 2: Alter table with changes in columns
- Experiment 3: Alter table with constraints
- Experiment 4: Dropping Tables
- Experiment 5: Inserting Data into Tables.

Week 3: Inserting, Simple Select, Char, Number, Date functions

- Experiment 6: Simple Select
- Experiment 7: Select with conditions.
- Experiment 8: Using character functions.
- Experiment 9: Using number functions.
- Experiment 10: Using date functions.

Week 4: Detailed SELECT with sub-queries, EQUI-JOINS, correlated sub-queries.

- Experiment 11: Single row sub-queries.
- Experiment 12: Multiple row sub-queries.
- Experiment 13: Equal joins.
- Experiment 14: correlated sub-queries.

Week 5: GROUPING, SET, UPDATE, DELETE, VIEWS

- Experiment 15: Aggregate functions.
- Experiment 16: Grouping clauses
- Experiment 17: Select groups with having
- Experiment 18: Union/Intersection statements
- Experiment 19: Creating and dropping views.

Week 6: Back Logs, if any and/or Additional Exercises for Unit – I.

Unit – II (PL/SQL: Program Development):

Week 7: Iterative PL/SQL Blocks and functions.

- Experiment 20: Simple PL/SQL Blocks
- Experiment 21: Nested IF and CASE in PL/SQL
- Experiment 22: NULLIF and COALESCE functions
- Experiment 23: WHILE & FOR Loops

Week 8: Transaction support in PL/SQL

- Experiment 24: COMMIT & ROLLBACK
- Experiment 25: SAVEPOINTS

Week 9: Exception support in PL/SQL

- Experiment 26: Exception Blocks
- Experiment 27: BUILT-IN Exceptions
- Experiment 28: User defined Exceptions
- Experiment 29: Raising application error

Week 10: Functions, Procedures, Packages

- Experiment 30: Creating Stored Procedures with Parameters.
- Experiment 31: Creating Stored Functions with Parameters.
- Experiment 32: Grouping Stored Packages.

Week 11: Back Logs, if any and/or Additional Exercises for Unit – II.

Unit – III (PL/SQL: Cursors & Triggers)

Week 12: Declare, Fetch, Open, and Close Cursors

- Experiment 33: Declaring Cursors.
- Experiment 34: Opening & Closing Cursors.
- Experiment 35: Fetch from an open Cursor.
- Experiment 36: Accessing current row in the cursor.

Week 13: Before & After [Row and Statement Triggers], Instead of Triggers.

- Experiment 37: Creating Before Statement Trigger.
- Experiment 38: Creating After Statement Trigger.
- Experiment 39: Creating Before Row Trigger.
- Experiment 40: Creating After Row Trigger.
- Experiment 41: Creating Triggers with when condition.
- Experiment 42: Creating instead of triggers to replaces updating from views.

Week 14: Back Logs, if any and/or Additional Exercises for Unit – III

LAB MANUAL

Week-1 & Week-2:

Objective:

Student needs to learn Oracle 9i setup & connecting to Oracle from iSQL*Plus.
Students will also learn how to create, insert, alter, and dropping of the tables with constraints.

Experiment – I:
Problem Statement:

Create location, department, job_grade, and employee tables with the following columns.
Location: (location_id:number, city:string)
Department: (department_id:number, department_name:string, head:number, department_location:number)
Job_grade: (job_grade:string, lower_bound:number, upper_bound:number)
Employee: (employee_id:number, first_name:string, last_name:string, join_date:date, manager_id:number, salary:number)

Description of how the experiment- Commands/Solution:

1. Understand create table syntax.
2. Decide the name of the table.
3. Decide the name of each column and its data type.
4. Use the create table syntax to create the said tables.
5. Create primary key constraint for each table as understand from logical table structure.

Experiment 2:

Problem Statement:

Alter employee table to add job_grade column which is of string data type.

Solution:

1. Learn alter table syntax.
2. Define the new column and its data type.
3. Use the alter table syntax.

Experiment 3:

Problem Statement:

Alter employee table to make job_grade a foreign key to job_grade table, manager_id a foreign key to employee table, department_id a foreign key to department table.

Draw an ER diagram depicting the 4 tables from Experiment 1 with all the added constraints from Experiment 2 to Experiment 4.

Solution:

4. Learn alter table syntax.
5. Define the new constraint [type, name, columns effected]
6. Use the alter table syntax for adding constraints.

Experiment 4:

Problem Statement:

Create a dummy table called my_employee with the same definition as employee table and then drop the table.

Solution:

1. Use create table to create my_employee.
2. Use drop table to drop my_employee.

Experiment 5:

Problem Statement:

Insert data into location, department, job_grade & employee tables.

Solution:

1. Decide the data to add in location.
2. Add to location one row at a time using insert into syntax
3. Decide the data to add in department.
4. Add to department one row at a time using insert into syntax.
5. Decide the data to add in job_grade.
6. Add to job_grade one row at a time using the insert into syntax.
7. Decide the data to add in employee.
8. Add to employee one row at a time using the insert into syntax.

Outcome: Have the ability to interface with ORACLE 9i set up , understand the syntaxes and implementing them by creating tables along with constraints . And also how the relationship takes place by foreign keys and how to implement by using Alter command.

Week 3:

Objective:

Students will learn how to write simple select statement, select statement with conditions and date, character, number functions.

Experiment 6:

Problem Statement:

Give a list of all employees (names as first_name, last_name) who belong to one department_id.

Solution:

1. use SELECT FROM WHERE syntax.
2. select should include first_name and last_name in the given format.
3. from should include employee
4. where should include condition on department_id

Experiment – 7:

Problem Statement:

Select employee last_names from employee table who belong to a certain department_id and have a salary greater than 5000.

Solution:

1. Use SELECT FROM WHERE syntax.
2. Use AND operator for 2 conditions in WHERE.

Experiment – 8:

Problem Statement:

Select employee last_name with first letter in capital, all smalls and all capitals from employee table for all employees.

Solution:

1. Use select from where clause.
2. Use initcap, lower, and upper functions on last_name in select clause to get the result.

Experiment – 9:

Problem Statement:

Select the salary and additional HRA (7.5% of the salary) for each employee in employee table rounded to a whole number.

Solution:

1. Use select from where clause and arithmetic to compute HRA.
2. Use the round function to round off the computed HRA to the nearest whole number.

Experiment – 10:

Problem Statement:

Select employee last_name, join_date, and the number of days he/she has been working in the firm as of today.

Solution:

1. Use the select from where clause.
2. In the clause, get the current data using sysdate function, get the difference between sysdate and employee joining date and compute the days in between using the days_between date function.
3. Show the join_date using mm/dd/yyyy format to using to_char function.

Outcome:

Ability to retrieve the contents as given by the user and the usage of different functions.

Week 4:

Objective:

Students will gain knowledge on sub-queries and understand about joins

Experiment - 11:

Problem Statement:

Select employee last_name of all employees whose salary is greater than the salary of employee with id = 2.

Solution:

1. Use an inner query to first get the salary of employee_id = 2.
2. The result from the inner query is than given in the where clause of the outer query that selects all employees with salary greater than that.

Experiment - 12:

Problem Statement:

Select all employees whose salary is greater than the salaries of both employees with ids 2 & 3.

Solution:

1. Use an inner query to first get the salary of employee_id in (2,3).
2. The result from the inner query is than given in the where clause of the outer query using the all operator that selects all employees with salary greater than that.

Experiment - 13:

Problem Statement:

Select employee lastname and the corresponding department_name for all employees in employees table.

Solution:

1. Use select from where clause with the from coming from employee and department tables and a condition joining these two table using the foreign key department_id which connects both the tables with an equality condition.

Experiment - 14:

Problem Statement:

Select all employees whose salary is lesser than all employees in the same job grade.

Solution:

1. Use the correlated sub-query where the inner query will be made based on the data coming from the upper query. In this case, the salary and grade of each employee row is used to execute the inner query to compare if there are any employees in the same grade who have lesser salary using exists clause.

Outcome:

Ability to grasp knowledge on joins and write sub-queries.

Week 5:

Objective:

Students will get an exposure to clauses, views and aggregate functions.

Experiment – 15:

Problem Statement:

Select the average salary of all employees in department with department_id = 2.

Solution:

1. Use the AVG aggregate function in select clause.
2. Use the department_id = 2 condition in where clause.

Experiment – 16:

Problem Statement:

Select AVG salary of each department which has employees in employee table.

Solution:

1. Use the AVG aggregate function in select clause.
2. Since we want average per department, have department_id in select clause.
3. Include a grouping clause by department_id to indicate you want to aggregate data by department_id

Experiment – 17:

Problem Statement:

Select minimum salary of all departments where the minimum salary is less than 1000.

Solution:

1. Use the MIN aggregate function in select clause.
2. Since we want average per department, have department_id in select clause.
3. Include a grouping clause by department_id to indicate you want to aggregate data by department_id.
4. Exclude groupings which have a minimum salary greater than 1000 using having clause.

Experiment – 18:

Problem Statement:

Give a list of all employees who earn a salary greater than 10000 or work in job grade MANAGER.

Solution:

1. Write a select from where which gives all employees with salary greater than 10000.
2. Write a select from where which gives all employees whose job grade in manager.
3. Combine them using UNION clause.

Experiment - 19:

Problem Statement:

Create a view that shows all employees of department_id = 10 and select from the view.

Solution:

1. Create view with the select clause on employees table with the said condition.
2. Write a select statement using the view created instead of table whenever you need employees of department_id = 10.

Outcome:
Ability to grasp the knowledge of views

Week 6:
Back Logs, if any and/or Additional Exercises for Unit - I.

Week 7:

Objective:

Students will gain an exposure of Iterative Blocks in PL/SQL

Experiment – 20:

Problem Statement:

Write a PL/SQL block which declares a variable and reads the last name of employee with id = 5 and outputs that to standard output.

Solution:

1. Declare a varchar variable
2. Write the select into clause inside PL/SQL Begin END block.

Experiment – 21:

Problem Statement:

Write a PL/SQL block which declares a variable with a value and prints in all capitals if the value starts with 'S', in all smalls if it starts with 'R', and in initial capitals if otherwise.

Solution:

1. Use IF or CASE.

Experiment – 22:

Problem Statement:

Write a PL/SQL block which declares two variables with values and prints first value if it is not null and prints second value if first is null.

Solution:

Use NULLIF or COALESCE.

Experiment – 23:

Problem Statement:

Write a PL/SQL block which declares a variable and reads the last name of employees and outputs that to standard output.

Solution:

1. Declare a varchar variable

2. Write a LOOP.

Outcome:

Ability to understand the knowledge of Iterative blocks like Nested IFS, WHILE and FOR loops

Week 8:

Objective:

Students will gain an exposure to give a transaction support in PL/SQL

Experiment – 24:

Problem Statement:

Write a PL/SQL block which updates 2 tables. If the second update fails the first update also has to be reversed.

Solution:

1. Use BEGIN transaction, END transaction.
2. Check for success of a query.
3. Use ROLLBACK or COMMIT depending on query success.

Experiment – 25:

Problem Statement:

Write a PL/SQL block which updates 3 tables. If the third update fails the second update has to be reversed while first should not get effected.

Solution:

1. Use BEGIN transaction, END transaction.
2. Define SAVEPOINT at the end of first update.
3. Run 3rd update and check for success.
4. If 3rd fails, Use ROLLBACK to SAVEPOINT.

Outcome:

Ability to grasp the usage of rollback, commit and savepoints in PL/SQL.

Week 9:

Objective:

Students will gain an exposure of how to use Exceptions in oracle 9i.

Experiment – 26:

Problem Statement:

Write a PL/SQL block with a simple Exception Block.

Solution:

1. Refer to Exception block syntax.

Experiment – 27:

Problem Statement:

Write a PL/SQL block which declares a variable and reads the last name of employee with id = 5. If there is no employee with id = 5 an error should be thrown. Otherwise the name has to be printed.

Solution:

1. Use built-in exceptions.

Experiment – 28:

Problem Statement:

Write a PL/SQL block which declares a variable and reads the last name of employee with id = 5. If the name has 8 characters, raise an exception called too many characters and handle it by cutting the last_name to first eight characters.

Solution:

1. Use user defined exceptions.

Experiment – 29:

Problem Statement:

Think of a problem that generates an exception is caught but an application error has to be raised because the exception in fatal.

Solution:

1. As a part of exception handling, raise application error if the situation is not recoverable.

<u>Outcome:</u>

Ability to understand the usage of exceptions, like the built in exceptions, user defined exceptions.

Week 10:

Objective:

Students will gain an exposure to create procedures, functions and packages.

Experiment – 30:

Problem Statement:

Create a stored procedure which takes deparment_id as parameter, inserts all employees of that department in a table called dept_employee with the same structure.

Solution:

1. Learn how to use parameter in to stored procedure.

Experiment – 31:

Problem Statement:

Create a function which takes deparment_id as parameter and returns the name of the department.

Solution:

1. Learn how to use parameter in functions.

Experiment – 32:

Problem Statement:

Write a package which implements a set of functions that will compute HRA, DA based on rules given salary.

Solution:

1. Implement one stored function that will compute HRA given basic.
2. Implement one stored function that will compute DA given basic.
3. Put these in a package so it gets loaded at one time.

Outcome:

Ability to understand the concept of procedures, functions and packages.

Week 11:

Back Logs, if any and/or Additional Exercises for Unit – II.

Week 12:

Objective:

Students will learn about cursors like declaring it, opening and closing of cursor, fetching the rows in a corsor.

Experiment – 33:

Problem Statement:

Define a cursor which runs through all employees who belong to department with id = 2.

Solution:

1. Use Declare cursor syntax within the PL/SQL block to define the cursor with a simple query which is select from where using employee table and a condition department_id = 2

Experiment – 34:

Problem Statement:

Declare a cursor which runs through all employees who belong to department with id = 2, open the cursor and close the cursor without doing anything.

Solution:

1. Use open and close operations on cursors. This is similar to opening and closing file handles.

Experiment – 35:

Problem Statement:

Declare a cursor which runs through all employees who belong to department with id = 2, open the cursor and fetches one employee at a time, prints the last name and then closes the cursor after all employees are done.

Solution:

1. Declare cursor.
2. Open the cursor.
3. In a loop, fetch each row of a cursor into a variable.
4. Print the variable.
5. Continue the loop till there are no more rows.
6. Close the cursor.

Experiment – 36:

Problem Statement:

Use a cursor to look at each employee who belongs to department with id 10, check the job grade and append NEW_ to all job_grades.

Solution:

1. Use the WHERE CURRENT to identify the currently fetched row from cursor and run the update query.
2. This is useful for update and deletes.

Outcome:
Ability to grasp the concept of cursors.

Week – 13:

Objective:

Students will have to learn about before and after triggers i.e., Row and statement triggers, instead of triggers.

Experiment – 37:

Problem Statement:

Create a trigger which writes a record called "employees table being changed" with time in a log table whenever anyone attempts to change employees table.

Solution:

1. We want to write a statement trigger that records the fact that someone is changing the employees table in a log table using before trigger.

Experiment – 38:

Problem Statement:

Create a trigger which writes a record called "employees table has been changed" with time in a log table whenever someone successfully changes the employee table.

Solution:

1. We want to write a statement trigger that records the fact that some-one changed the employees table in a log table using after trigger that acts on insert, update and delete operations.

Experiment – 39:

Problem Statement:

Implement a solution that would record all attempted updates on salary in the employee table along with the employee id, old salary, new salary and the time when it was attempted in another table.

Solution:

1. Use row before trigger in update operation.

Experiment – 40:

Problem Statement:

Implement a solution that would record all updates on salary in the employee table along with the employee id, old salary, new salary and the time when it was updated in another table.

Solution:

1. Use row after trigger in update operation.

Experiment – 41:

Problem Statement:

Implement a solution to insert a log entry in log table whenever salary of employees with more than 20000 is revised.

Solution:

1. Write a row after trigger on employee record.
2. The trigger should work on the condition that the salary of the record is greater than 20000.

Experiment – 42:

Problem Statement:

Implement a solution which would transparently make the user feel like he is updating a view but in reality the view is read-only and a trigger is updating the base tables based on the view update given by user.

Solution:

1. Use the instead of option in triggers to make trigger work in place of view and reproduce the effect intended.

Outcome:

Have the ability to handle the creation of before statement triggers and after statement triggers, creation of before Row and after Row triggers, creation of triggers with When condition and creation Instead of triggers to replaces updating from views.

Unit – I Additional Exercises

1. Write a query that would give the employee name, his salary and the lower end and higher end salaries of his job grade?
2. Write a query that would give the employee name and his department name using natural join.
3. Write a query which would give all employee names and their department names. The query should also return employees who are currently not allotted to any department and departments that do not have any employees.
4. Write a query to merge department level employee table data changes to college level employee table on a periodic basis.
5. I want to insert a lot of records in employee table which has certain constraints on salary. I know all records I am inserting would be good in salary field. Trying inserting as is and with the salary constraint disable before bulk insert and enable after insert and see the time difference.
6. Create index of salary column of employee and search for all employees with a certain salary.
7. Instead of giving a unique ID yourself, what can be done to automatically generate unique ID.

Unit – II Additional Exercises

1. Write a PL/SQL procedure which takes department id as parameter and gives all employees belong to the department.
2. Write a PL/SQL procedure which takes lowest and highest salary as parameters and returns all employees with-in that salary range. If there re no employees, an exception has to be raised and an application error to be given?
3. Write a PL/SQL procedure where you define your own user exceptions and handle exceptions appropriately.

Unit – III Additional Exercises

1. Write a cursor which takes parameters and invoke a cursor from within a PL/SQL block.
2. Write an after trigger which acts at a row level on insert/update/delete and transfer old data to an audit trail table.

Supplementary Exercises

1. Design a database for library management in the college, populate data and write queries to return information like books issued, books issued to user, number of books issued, books reserved, etc.
2. Design a database for fee management in the college, populating data on fee payments, fee receipts and writing querying to return data like uncollected fees, which paid fee, etc.

References

Web-Sites:www.otn.oracle.com

Books:

1. Database Management Systems, Korth & Sudharshan.
2. Database Management Systems, Raghuram Krishnan.
3. Database Management Systems, Elmasri & Navathe.

Mapping of experiments to lab objectives

List of Experiments	Obj 1	Obj 2	Obj 3
Unit - I			
Experiment – 1 Creation of tables	×		×
Experiment – 2 Alter table with changes in columns	×		×
Experiment – 3 Alter table with constraints	×		×
Experiment – 4 Dropping Tables	×		×
Experiment – 5 Insert Data into Tables	×		×
Experiment – 6 Simple SELECT queries	×		×
Experiment – 7 Selection with conditions	×		×
Experiment – 8 Using Character functions	×		×
Experiment – 9 Using number functions	×		×
Experiment – 10 Using date functions	×		×
Experiment – 11 single row sub-queries	×		×
Experiment – 12 Multiple row sub-queries	×		×
Experiment – 13 Equal joins	×		×
Experiment – 14 Correlated sub-queries	×		×
Experiment – 15 Aggregate functions	×		×
Experiment – 16 Grouping clauses	×		×
Experiment – 17 Select groups with having	×		×
Experiment – 18 Union/Intersection statements	×		×
Experiment – 19 Creating and dropping views	×		×
Additional Exercises for Unit – I			

Unit - II			
Experiment – 20 Simple PL/SQL Blocks		×	×
Experiment – 21 Nested IF and CASE in PL/SQL		×	×
Experiment – 22 NULLIF and COALESCE functions		×	×
Experiment – 23 WHILE & FOR Loops		×	×
Experiment – 24 COMMIT & ROLLBACK		×	×
Experiment – 25 SAVEPOINTS		×	×
Experiment – 26 Exception Blocks		×	×
Experiment – 27 BUILT-IN Exceptions		×	×
Experiment – 28 User defined Exceptions		×	×
Experiment – 29 Raising applications error		×	×
Experiment – 30 Creating stored Procedures with parameters		×	×
Experiment – 31 Creating stored Functions with Parameters		×	×
Experiment – 32 Grouping stored Packages		×	×
Additional Exercises for Unit – II			
Unit - III			
Experiment – 33 Declaring Cursors		×	×
Experiment – 34 Opening & Closing Cursors		×	×
Experiment – 35 Fetch from an open Cursor		×	×
Experiment – 36 Accessing current row in the cursor		×	×
Experiment – 37 Creating Before statement Trigger		×	×

Experiment - 38 Creating After statement Trigger		×	×
Experiment - 39 Creating Before Row Trigger		×	×
Experiment - 40 Creating after row Trigger		×	×
Experiment - 41 Creating Triggers with when condition		×	×
Experiment - 42 Creating instead of Triggers to replaces updating from views		×	×