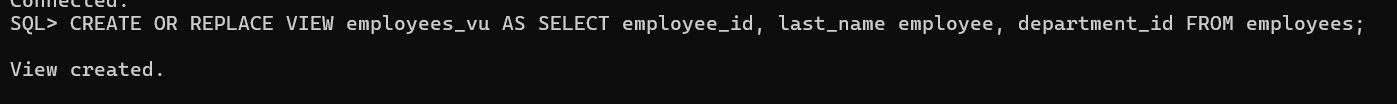
**DB 10 – SQL VOL. 2**

**22510064 (PARSHWA HERWADE)**

**P11:**

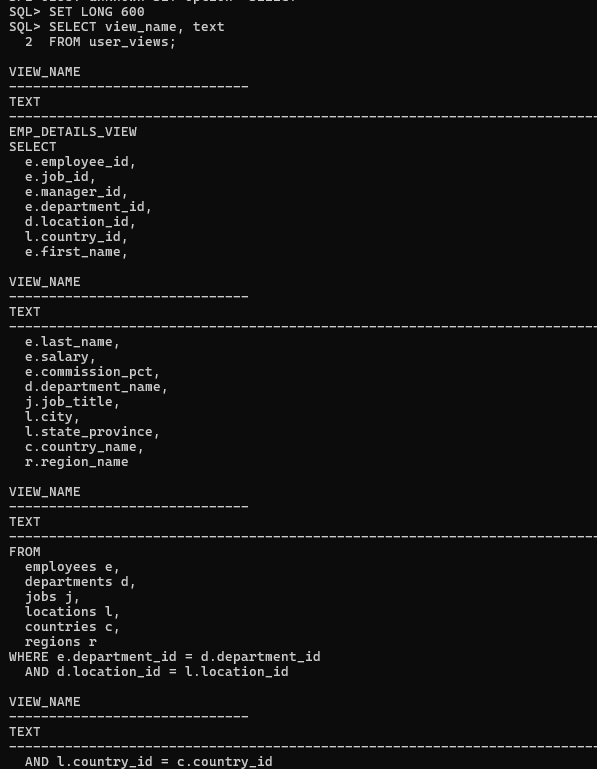
1.Create a view called EMPLOYEES\_VU based on the employee numbers, employee names, and department numbers from the EMPLOYEES table. Change the heading for the employee name to EMPLOYEE.



2. Display the contents of the EMPLOYEES\_VU view.



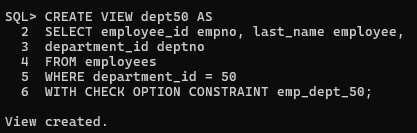
3. Select the view name and text from the USER\_VIEWS data dictionary view. Note: Another view already exists. The EMP\_DETAILS\_VIEW was created as part of your schema. Note: To see more contents of a LONG column, use the iSQL\*Plus command SET LONG n, where n is the value of the number of characters of the LONG column that you want to see.



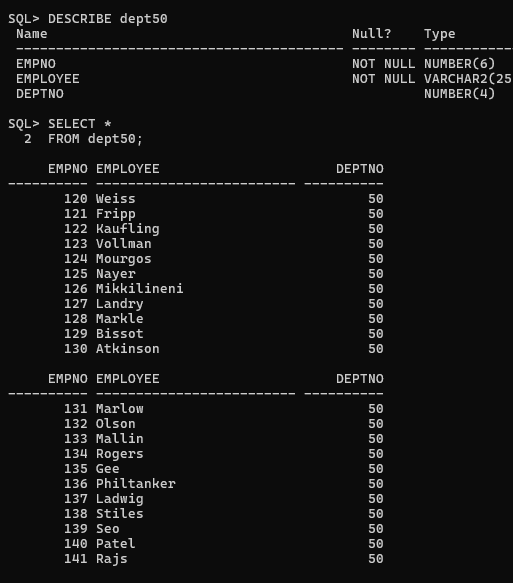
4. Using your EMPLOYEES\_VU view, enter a query to display all employee names and department numbers.



5. Create a view named DEPT50 that contains the employee numbers, employee last names, and department numbers for all employees in department 50. Label the view columns EMPNO, EMPLOYEE, and DEPTNO. Do not allow an employee to be reassigned to another department through the view.

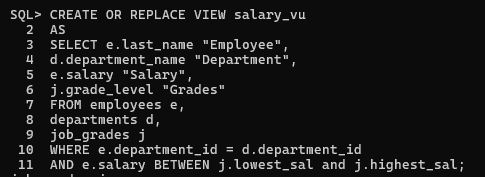


6. Display the structure and contents of the DEPT50 view.



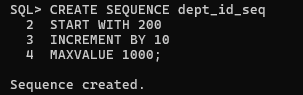
7. Attempt to reassign Matos to department 80.

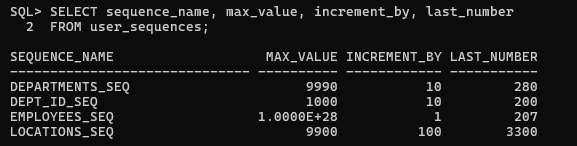


8. Create a view called SALARY\_VU based on the employee last names, department names, salaries, and salary grades for all employees. Use the EMPLOYEES, DEPARTMENTS, and JOB\_GRADES tables. Label the columns Employee, Department, Salary, and Grade, respectively. 

**P12:**

1. Create a sequence to be used with the primary key column of the DEPT table. The sequence should start at 200 and have a maximum value of 1000. Have your sequence increment by ten numbers. Name the sequence DEPT\_ID\_SEQ.



2. Write a query in a script to display the following information about your sequences: sequence name, maximum value, increment size, and last number. Name the script lab12\_2.sql. Run the statement in your script. 

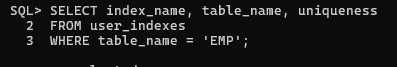
3. Write a script to insert two rows into the DEPT table. Name your script lab12\_3.sql. Be sure to use the sequence that you created for the ID column. Add two departments named Education and Administration. Confirm your additions. Run the commands in your script.



4. Create a nonunique index on the foreign key column (DEPT\_ID) in the EMP table.



5. Display the indexes and uniqueness that exist in the data dictionary for the EMP table. Save the statement into a script named lab12\_5.sql.

****

**P13:**

1. What privilege should a user be given to log on to the Oracle Server? Is this a system or an object privilege?

ANS. The CREATE SESSION system privilege

2. What privilege should a user be given to create tables?

ANS.The CREATE TABLE privilege  
 3. If you create a table, who can pass along privileges to other users on your table?

ANS.You can, or anyone you have given those privileges to by using the WITH GRANT OPTION.

4. You are the DBA. You are creating many users who require the same system privileges. What should you use to make your job easier?

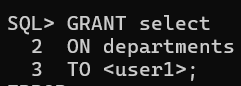
ANS.Create a role containing the system privileges and grant the role to the users

5. What command do you use to change your password?

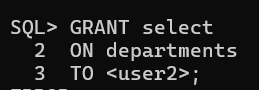
ANS.The ALTER USER statement

6. Grant another user access to your DEPARTMENTS table. Have the user grant you query access to his or her DEPARTMENTS table.

ANS.Team 2 executes the GRANT statement



Team 1 executes the GRANT statement



7. Query all the rows in your DEPARTMENTS table.



8. Add a new row to your DEPARTMENTS table. Team 1 should add Education as department number 500. Team 2 should add Human Resources department number 510. Query the other team’s table. ANS.Team 1 executes this INSERT statement.



Team 2 executes this INSERT statement.



9. Create a synonym for the other team’s DEPARTMENTS table.

Team 1 creates a synonym named team2.



Team 2 creates a synonym named team1

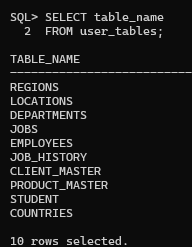


10. Query all the rows in the other team’s DEPARTMENTS table by using your synonym. Team 1 executes this SELECT statement.

ANS. SELECT \* FROM team2;

ANS. SELECT \* FROM team1;

11. Query the USER\_TABLES data dictionary to see information about the tables that you own.

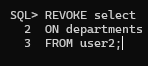


12. Query the ALL\_TABLES data dictionary view to see information about all the tables that you can access. Exclude tables that you own

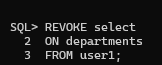


13. Revoke the SELECT privilege from the other team.

Team 1 revokes the privilege.

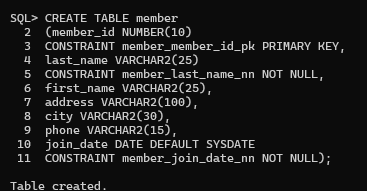


Team 2 revokes the privilege

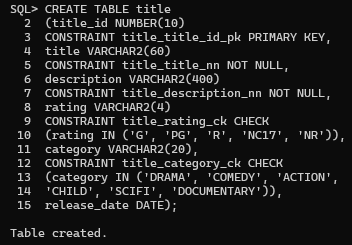
****

**P14:**

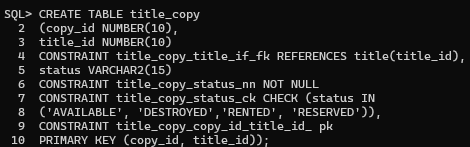
Create the tables based on the following table instance charts. Choose the appropriate data types and be sure to add integrity constraints. a. Table name: MEMBER



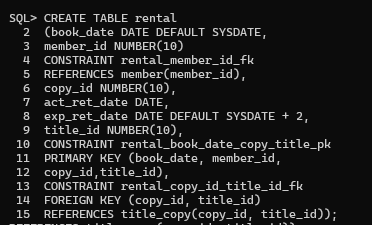
b. Table name: TITLE



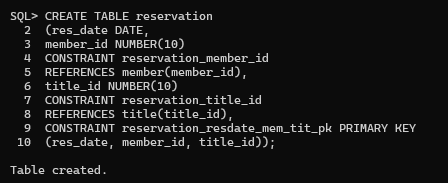
c. Table name: TITLE\_COPY

****

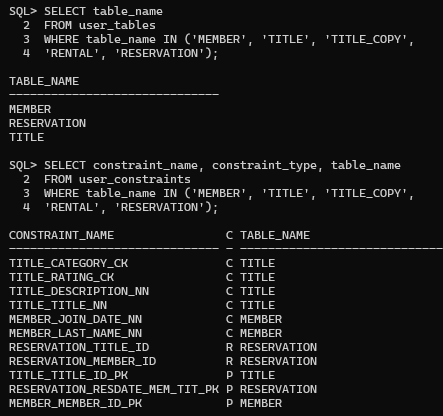
d. Table name: RENTAL



e. Table name: RESERVATION

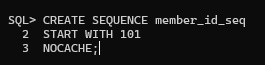


2. Verify that the tables and constraints were created properly by checking the data dictionary.

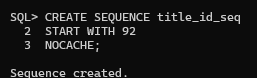


3. Create sequences to uniquely identify each row in the MEMBER table and the TITLE table.

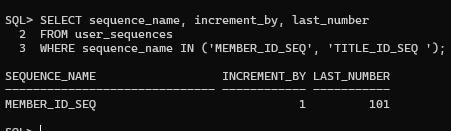
a. Member number for the MEMBER table: start with 101; do not allow caching of the values. Name the sequence MEMBER\_ID\_SEQ. CREATE SEQUENCE member\_id\_seq START WITH 101 NOCACHE;



b. Title number for the TITLE table: start with 92; no caching. Name the sequence TITLE\_ID\_SEQ. CREATE SEQUENCE title\_id\_seq START WITH 92 NOCACHE;

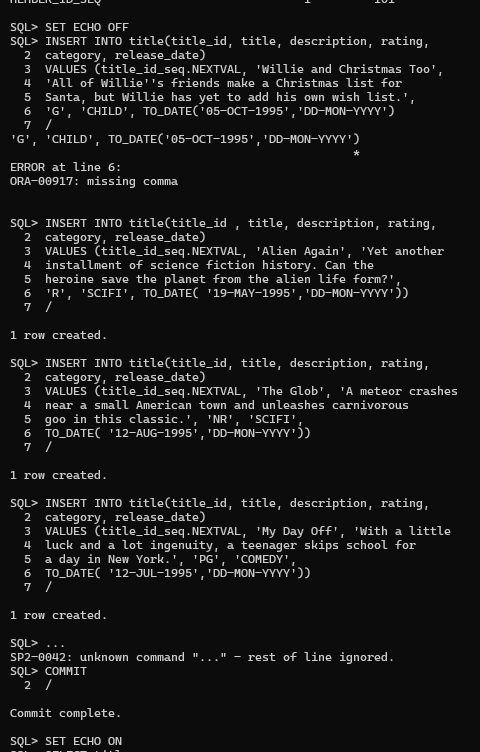


c. Verify the existence of the sequences in the data dictionary.

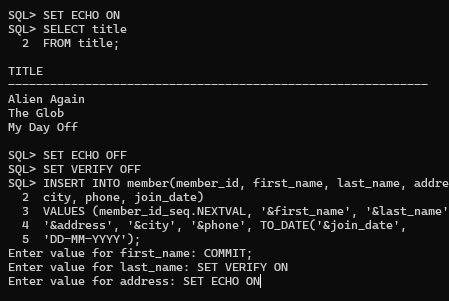


4. Add data to the tables. Create a script for each set of data to add.

a. Add movie titles to the TITLE table. Write a script to enter the movie information. Save the statements in a script named lab14\_4a.sql. Use the sequences to uniquely identify each title. Enter the release dates in the DD-MON-YYYY format. Remember that single quotation marks in a character field must be specially handled. Verify your additions.



b. Add data to the MEMBER table. Place the insert statements in a script named lab14\_4b.sql. Execute commands in the script. Be sure to use the sequence to add the member numbers.

****

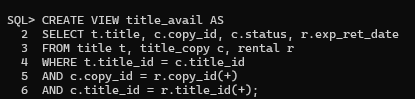
c. Add the following movie copies in the TITLE\_COPY table: Note: Have the TITLE\_ID numbers available for this exercise.



d. Add the following rentals to the RENTAL table: Note: Title number may be different depending on sequence number

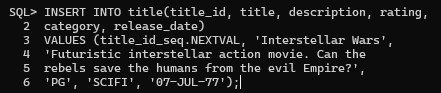


5. Create a view named TITLE\_AVAIL to show the movie titles and the availability of each copy and its expected return date if rented. Query all rows from the view. Order the results by title



6. Make changes to data in the tables

. a. Add a new title. The movie is “Interstellar Wars,” which is rated PG and classified as a scifi movie. The release date is 07-JUL-77. The description is “Futuristic interstellar action movie. Can the rebels save the humans from the evil empire?” Be sure to add a title copy record for two copies.



b. Enter two reservations. One reservation is for Carmen Velasquez, who wants to rent “Interstellar Wars.” The other is for Mark Quick-to-See, who wants to rent “Soda Gang.”

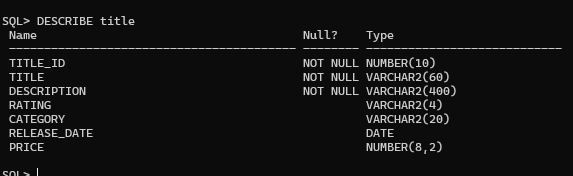


c. Customer Carmen Velasquez rents the movie “Interstellar Wars,” copy 1. Remove her reservation for the movie. Record the information about the rental. Allow the default value for the expected return date to be used. Verify that the rental was recorded by using the view you created.

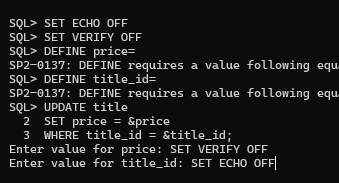


7. Make a modification to one of the tables.

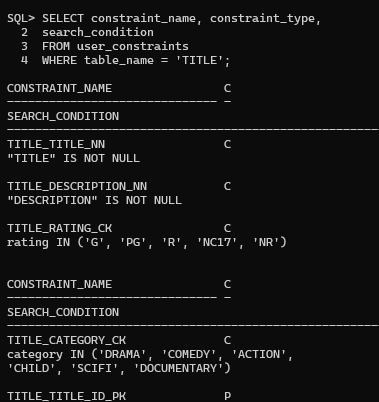
a. Add a PRICE column to the TITLE table to record the purchase price of the video. The column should have a total length of eight digits and two decimal places. Verify your modifications



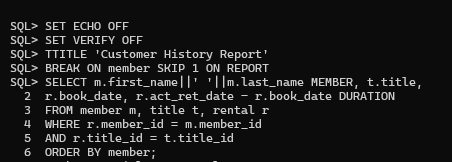
b. Create a script named lab14\_7b.sql that contains update statements that update each video with a price according to the following list. Run the commands in the script.



c. Ensure that in the future all titles contain a price value. Verify the constraint.

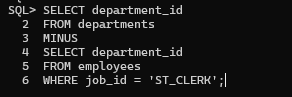


8. Create a report titled Customer History Report. This report contains each customer's history of renting videos. Be sure to include the customer name, movie rented, dates of the rental, and duration of rentals. Total the number of rentals for all customers for the reporting period. Save the commands that generate the report in a script file named lab14\_8.sql.

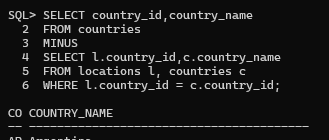
****

**P15:**

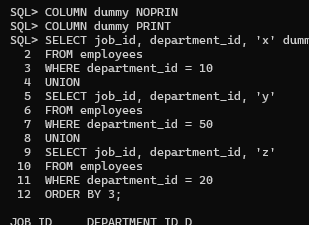
1. List the department IDs for departments that do not contain the job ID ST\_CLERK, using SET operators.



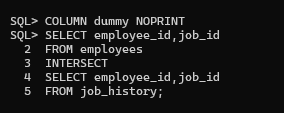
2. Display the country ID and the name of the countries that have no departments located in them, using SET operators



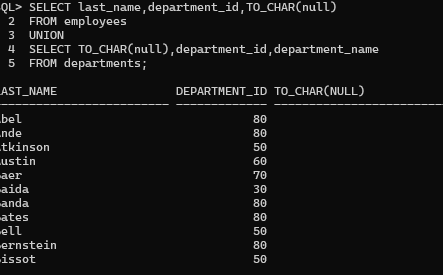
3. Produce a list of jobs for departments 10, 50, and 20, in that order. Display job ID and department ID, using SET operators.



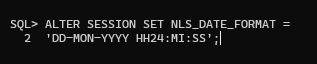
4. List the employee IDs and job IDs of those employees, who are currently in the job title that they have held once before during their tenure with the company



5. Write a compond query that lists the following : • Last names and department ID of all the employees from the EMPLOYEES table, irrespective of whether they belong to any department • Department ID and department name of all the departments from the DEPARTMENTS table, irrespective of whether they have employees working in them

****

**P16:**

1. Alter the session to set the NLS\_DATE\_FORMAT to DD-MON-YYYY HH24:MI:SS.
2. 

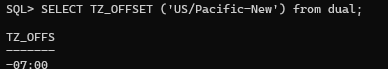
2. a. Write queries to display the time zone offsets (TZ\_OFFSET) for the following time zones. US/Pacific-New

b. Alter the session to set the TIME\_ZONE parameter value to the time zone offset of US/Pacific-New.

c. Display the CURRENT\_DATE, CURRENT\_TIMESTAMP, and LOCALTIMESTAMP for this session. Note: The output might be different based on the date when the command is executed.

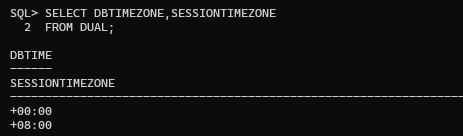
d. Alter the session to set the TIME\_ZONE parameter value to the time zone offset of Singapore.

e. Display the CURRENT\_DATE, CURRENT\_TIMESTAMP, LOCALTIMESTAMP for this session. Note: The output might be different based on the date when the command is executed.

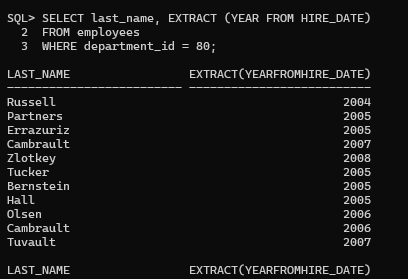




3. Write a query to display the DBTIMEZONE and SESSIONTIMEZONE

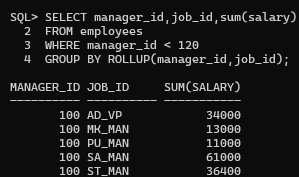


4. Write a query to extract the YEAR from HIRE\_DATE column of the EMPLOYEES table for those employees who work in department 80.

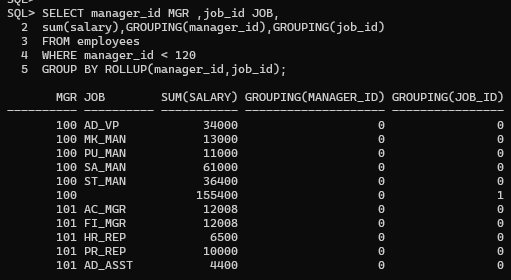
****

**P17:**

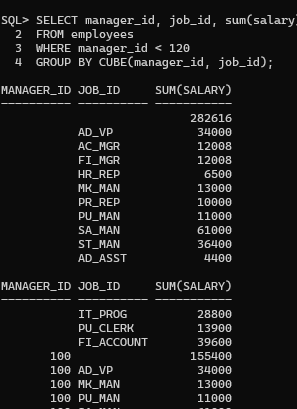
1. Write a query to display the following for those employees whose manager ID is less than 120: – Manager ID – Job ID and total salary for every job ID for employees who report to the same manager – Total salary of those managers – Total salary of those managers, irrespective of the job IDs.



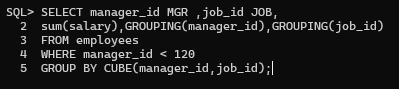
2. Observe the output from question 1. Write a query using the GROUPING function to determine whether the NULL values in the columns corresponding to the GROUP BY expressions are caused by the ROLLUP operation



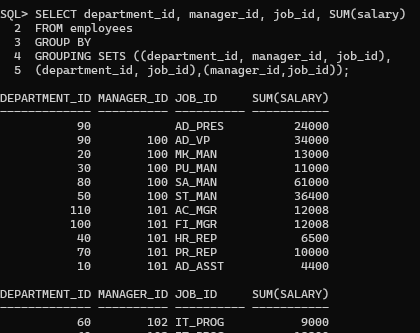
3. Write a query to display the following for those employees whose manager ID is less than 120 : – Manager ID – Job and total salaries for every job for employees who report to the same manager – Total salary of those managers – Cross-tabulation values to display the total salary for every job, irrespective of the manager – Total salary irrespective of all job titles



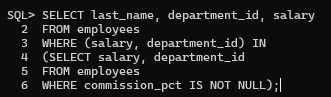
4. Observe the output from question 3. Write a query using the GROUPING function to determine whether the NULL values in the columns corresponding to the GROUP BY expressions are caused by the CUBE operation.



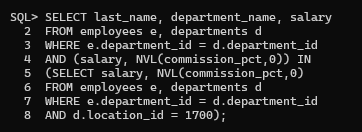
5. Using GROUPING SETS, write a query to display the following groupings : – department\_id, manager\_id, job\_id – department\_id, job\_id – Manager\_id, job\_id The query should calculate the sum of the salaries for each of these groups.

****

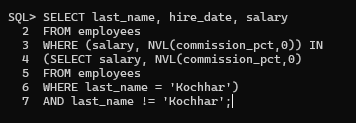
**P18:**

1. Write a query to display the last name, department number, and salary of any employee whose department number and salary both match the department number and salary of any employee who earns a commission.
2. 

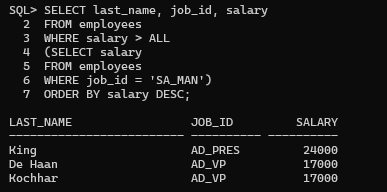
2. Display the last name, department name, and salary of any employee whose salary and commission match the salary and commission of any employee located in location ID1700



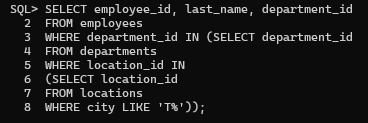
3. Create a query to display the last name, hire date, and salary for all employees who have the same salary and commission as Kochhar. Note: Do not display Kochhar in the result set.



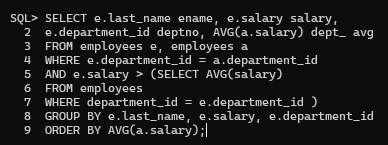
4. Create a query to display the employees who earn a salary that is higher than the salary of all of the sales managers (JOB\_ID = 'SA\_MAN'). Sort the results on salary from highest to lowest.



5. Display the details of the employee ID, last name, and department ID of those employees who live in cities whose name begins with T.

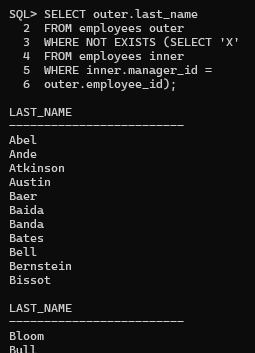


6. Write a query to find all employees who earn more than the average salary in their departments. Display last name, salary, department ID, and the average salary for the department. Sort by average salary. Use alises for the columns retrieved by the query as shown in the sample output.

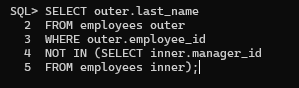


7. Find all employees who are not supervisors.

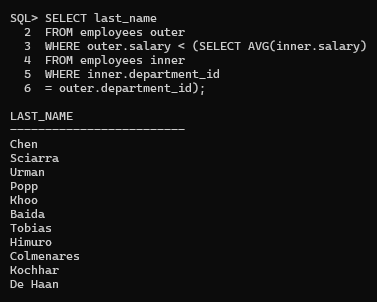
a. First do this by using the NOT EXISTS operator



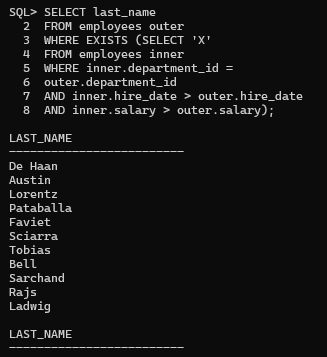
b. Can this be done by using the NOT IN operator? How, or why not?



8. Write a query to display the last names of the employees who earn less than the average salary in their departments.



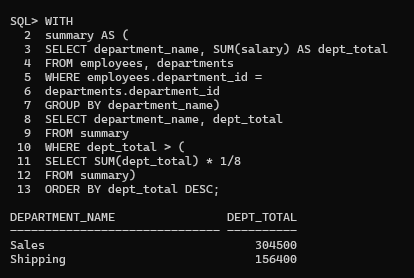
9. Write a query to display the last names who have one or more coworkers in their departments with later hire dates but higher salaries.



10. Write a query to display the employee ID, last names of the employees, and department names of all employees. Note: Use a scalar subquery to retrieve the department name in the SELECT statement.



11. Write a query to display the department names of those departments whose total salary cost is above one-eighth (1/8) of the total salary cost of the whole company. Use the WITH clause to write this query. Name the query SUMMARY

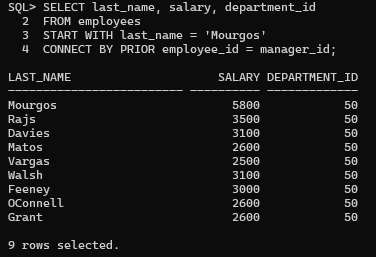
****

**P19:**

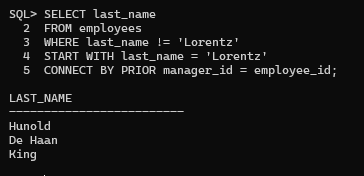
1. Look at the following output. Is this output the result of a hierarchical query? Explain why or why not.

ANS. a. Exhibit 1: This is not a hierarchical query; the report simply has a descending sort on SALARY. Exhibit 2: This is not a hierarchical query; there are two tables involved. Exhibit 3: Yes, this is most definitely a hierarchical query as it displays the tree structure representing the management reporting line from the EMPLOYEES table.

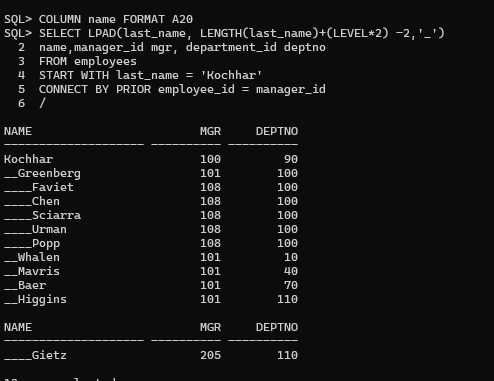
2. Produce a report showing an organization chart for Mourgos’s department. Print last names, salaries, and department IDs.



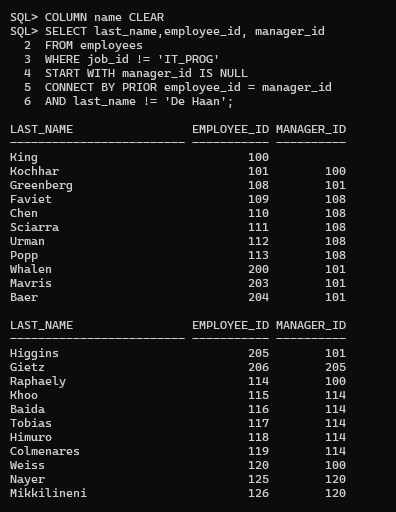
3. Create a report that shows the hierarchy of the managers for the employee Lorentz. Display his immediate manager first.



4. Create an indented report showing the management hierarchy starting from the employee whose LAST\_NAME is Kochhar. Print the employee’s last name, manager ID, and department ID. Give alias names to the columns as shown in the sample output.



5. Produce a company organization chart that shows the management hierarchy. Start with the person at the top level, exclude all people with a job ID of IT\_PROG, and exclude De Haan and those employees who report to De Hann

****

**P20:**

1. Run the cre\_sal\_history.sql script in the Labs folder to create the SAL\_HISTORY table.



2. Display the structure of the SAL\_HISTORY table.



3. Run the cre\_mgr\_history.sql script in the Labs folder to create the MGR\_HISTORY table



4. Display the structure of the MGR\_HISTORY table.



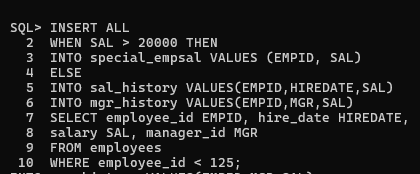
5. Run the cre\_special\_sal.sql script in the Labs folder to create the SPECIAL\_SAL table.



6. Display the structure of the SPECIAL\_SAL table.



7. a. Write a query to do the following: – Retrieve the details of the employee ID, hire date, salary, and manager ID of those employees whose employee ID is less than 125 from the EMPLOYEES table. – If the salary is more than $20,000, insert the details of employee ID and salary into the SPECIAL\_SAL table. – Insert the details of the employee ID, hire date, and salary into the SAL\_HISTORY table. – Insert the details of the employee ID, manager ID, and SYSDATE into the MGR\_HISTORY table.



b. Display the records from the SPECIAL\_SAL table.



c. Display the records from the SAL\_HISTORY table



d. Display the records from the MGR\_HISTORY table



8. a. Run the cre\_sales\_source\_data.sql script in the Labs folder to create the SALES\_SOURCE\_DATA table.



b. Run the ins\_sales\_source\_data.sql script in the Labs folder to insert records into the SALES\_SOURCE\_DATA table.



c. Display the structure of the SALES\_SOURCE\_DATA table



d. Display the records from the SALES\_SOURCE\_DATA table.



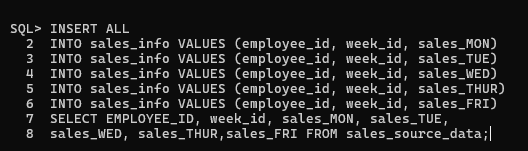
e. Run the cre\_sales\_info.sql script in the Labs folder to create the SALES\_INFO table.



f. Display the structure of the SALES\_INFO table.



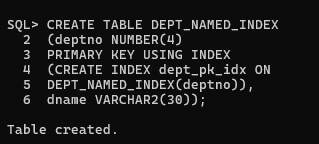
g. Write a query to do the following: – Retrieve the details of the employee ID, week ID, sales on Monday, sales on Tuesday, sales on Wednesday, sales on Thursday, and sales on Friday from the SALES\_SOURCE\_DATA table. – Build a transformation such that each record retrieved from the SALES\_SOURCE\_DATA table is converted into multiple records for the SALES\_INFO table. Hint: Use a pivoting INSERT statement



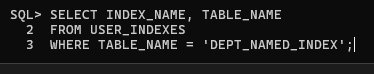
h. Display the records from the SALES\_INFO table



9. a. Create the DEPT\_NAMED\_INDEX table based on the following table instance chart. Name the index for the PRIMARY KEY column as DEPT\_PK\_IDX.



b. Query the USER\_INDEXES table to display the INDEX\_NAME for the DEPT\_NAMED\_INDEX table.

****