Database systems project

Abdulaziz Hafiz

ID: 444003143

Group 1 (corrected version)

This document shows which parts of the code cover the main course topics.

1. Sub-query: used in ‘get\_employee\_details’ and views like ‘managers’.
2. With-clause: seen in ‘get\_employee\_details'.
3. Window-function: used in ‘get\_dept\_salary\_rank’ using partition by.
4. Case clause: found in ‘display\_emps\_salary\_type’.
5. Views: managers, and ‘dept\_location\_view’.
6. Execution-plans: used in the main menu (option 10 with hints to use 4 threads) and indexes that optimize the performance along with partitioning.
7. Indexes: used with foreign keys to optimize performance such as ‘idx\_loc\_c\_id’ and more can be found in the code section.
8. Partitioning: found in ‘job\_history\_partition’ that helps to optimize the performance.
9. Cursor: wide usage among the program such as ‘get\_employee\_hierarchy’ and ‘get\_employee\_details’.
10. Exceptions: an important role implemented in ‘main menu anonymous procedure’ , ‘add\_employee’ , ‘remove\_employee’ and more.
11. Procedure: an essential part of the project can be found in ‘display\_department\_loactions’, ‘get\_employee\_hierarchy’ and more.
12. Function: was used in ‘get\_emp\_job\_history’.
13. Triggers: found in ‘keep\_admins’ and ‘log\_deleted\_employees’.
14. Collections: used in ‘get\_emp\_job\_history’, and ‘phone\_varray’.

Note: this code must have its schema and data to operate correctly, schema and data can be found in code section.

Parts of code:

Subquery:

CREATE OR REPLACE VIEW managers AS

SELECT DISTINCT e.EMPLOYEE\_ID, e.FIRST\_NAME, e.LAST\_NAME, e.EMAIL, e.PHONE\_NUMBER, e.JOB\_ID, e.SALARY, e.HIRE\_DATE

FROM EMPLOYEES e

WHERE e.EMPLOYEE\_ID IN (SELECT DISTINCT MANAGER\_ID FROM EMPLOYEES WHERE MANAGER\_ID IS NOT NULL)

ORDER BY e.EMPLOYEE\_ID;

With clause:

CREATE OR REPLACE PROCEDURE get\_employee\_details AS

CURSOR emp\_cursor IS

WITH dept\_location AS (

SELECT d.DEPARTMENT\_ID, d.DEPARTMENT\_NAME, l.CITY

FROM DEPARTMENTS d

JOIN LOCATIONS l ON d.LOCATION\_ID = l.LOCATION\_ID),

employee\_info AS (

SELECT e.EMPLOYEE\_ID, e.FIRST\_NAME, e.LAST\_NAME, e.DEPARTMENT\_ID, e.JOB\_ID, e.SALARY

FROM EMPLOYEES e

ORDER BY e.EMPLOYEE\_ID)

SELECT ei.EMPLOYEE\_ID, ei.FIRST\_NAME, ei.LAST\_NAME, ei.SALARY, j.JOB\_TITLE, dl.DEPARTMENT\_NAME, dl.CITY

FROM employee\_info ei

JOIN dept\_location dl ON ei.DEPARTMENT\_ID = dl.DEPARTMENT\_ID

JOIN JOBS j ON ei.JOB\_ID = j.JOB\_ID;

BEGIN

FOR emp\_rec IN emp\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || emp\_rec.EMPLOYEE\_ID || ', Name: ' || emp\_rec.FIRST\_NAME || ' ' || emp\_rec.LAST\_NAME || ', Job Title: ' || emp\_rec.JOB\_TITLE ||

', Department: ' || emp\_rec.DEPARTMENT\_NAME || ', City: ' || emp\_rec.CITY || ', Salary: ' || emp\_rec.SALARY);

END LOOP;

END;

Window function:

CREATE OR REPLACE PROCEDURE get\_dept\_salary\_rank(p\_department\_id IN NUMBER) AS

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Company Departments:');

FOR dept\_rec IN (SELECT DEPARTMENT\_ID, DEPARTMENT\_NAME FROM DEPARTMENTS ORDER BY DEPARTMENT\_ID)

LOOP

DBMS\_OUTPUT.PUT\_LINE('Department ID: ' || dept\_rec.DEPARTMENT\_ID || ', Department Name: ' || dept\_rec.DEPARTMENT\_NAME);

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('---------------------------------');

DBMS\_OUTPUT.PUT\_LINE('---------------------------------');

DBMS\_OUTPUT.PUT\_LINE('Employee Info for Department ID: ' || p\_department\_id);

FOR emp\_rec IN (

SELECT e.EMPLOYEE\_ID, e.FIRST\_NAME, e.LAST\_NAME, e.SALARY, ROW\_NUMBER() OVER (PARTITION BY e.DEPARTMENT\_ID ORDER BY e.SALARY DESC) AS emp\_rank

FROM EMPLOYEES e WHERE e.DEPARTMENT\_ID = p\_department\_id ORDER BY e.SALARY DESC )

LOOP

DBMS\_OUTPUT.PUT\_LINE('ID: ' || emp\_rec.EMPLOYEE\_ID || ', Name: ' || emp\_rec.FIRST\_NAME || ' ' || emp\_rec.LAST\_NAME || ', Salary: ' || emp\_rec.SALARY || ', Rank: ' || emp\_rec.emp\_rank);

END LOOP;

END;

Case clause:

CREATE OR REPLACE PROCEDURE DISPLAY\_EMPS\_SALARY\_TYPE IS

AVG\_SALARY NUMBER;

TOTAL\_COUNT NUMBER := 0;

ABOVE\_COUNT NUMBER := 0;

BELOW\_COUNT NUMBER := 0;

BEGIN

SELECT AVG(SALARY) INTO AVG\_SALARY FROM EMPLOYEES;

FOR EMP IN (SELECT SALARY FROM EMPLOYEES) LOOP

TOTAL\_COUNT := TOTAL\_COUNT + 1;

CASE

WHEN EMP.SALARY > AVG\_SALARY THEN

ABOVE\_COUNT := ABOVE\_COUNT + 1;

WHEN EMP.SALARY < AVG\_SALARY THEN

BELOW\_COUNT := BELOW\_COUNT + 1;

END CASE;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('TOTAL EMPLOYEES: ' || TOTAL\_COUNT);

DBMS\_OUTPUT.PUT\_LINE('ABOVE AVERAGE SALARY: ' || ABOVE\_COUNT);

DBMS\_OUTPUT.PUT\_LINE('BELOW AVERAGE SALARY: ' || BELOW\_COUNT);

END DISPLAY\_EMPS\_SALARY\_TYPE;

Views:

CREATE OR REPLACE VIEW dept\_location\_view AS

SELECT d.DEPARTMENT\_NAME, l.CITY, r.REGION\_NAME

FROM DEPARTMENTS d

JOIN LOCATIONS l ON d.LOCATION\_ID = l.LOCATION\_ID

JOIN COUNTRIES c ON l.COUNTRY\_ID = c.COUNTRY\_ID

JOIN REGIONS r ON c.REGION\_ID = r.REGION\_ID;

Execution plans:

FOR job\_record IN (SELECT /\*+ PARALLEL(e 4) \*/ \* FROM TABLE(GET\_EMP\_JOB\_HISTORY(:EMP\_ID\_TO\_FIND\_HISTORY))) LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || job\_record.EMPLOYEE\_ID || ', Start Date: ' || job\_record.START\_DATE || ', End Date: ' || job\_record.END\_DATE || ', Job ID: ' || job\_record.JOB\_ID || ', Department ID: ' || job\_record.DEPARTMENT\_ID);

END LOOP;

Index:

CREATE INDEX IDX\_COUNTR\_REG ON COUNTRIES (REGION\_ID);

Partitioning:

CREATE TABLE JOB\_HISTORY\_PARTITION (

EMPLOYEE\_ID NUMBER(6) NOT NULL,

START\_DATE DATE NOT NULL,

END\_DATE DATE NOT NULL,

JOB\_ID VARCHAR2(10 BYTE) NOT NULL,

DEPARTMENT\_ID NUMBER(4),

...

) PARTITION BY RANGE (START\_DATE) (

PARTITION part\_2000\_2010 VALUES LESS THAN (TO\_DATE('2010-01-01', 'YYYY-MM-DD')),

PARTITION part\_2010\_2020 VALUES LESS THAN (TO\_DATE('2020-01-01', 'YYYY-MM-DD')),

PARTITION part\_2020\_2030 VALUES LESS THAN (TO\_DATE('2030-01-01', 'YYYY-MM-DD')),

PARTITION part\_future VALUES LESS THAN (MAXVALUE) );

Cursor:

CURSOR EMPLOYEE\_CURSOR IS

SELECT EMPLOYEE\_ID, MANAGER\_ID, LEVEL AS HIERARCHY\_LEVEL

FROM EMPLOYEES

START WITH MANAGER\_ID IS NULL

CONNECT BY PRIOR EMPLOYEE\_ID = MANAGER\_ID

ORDER BY LEVEL;

Procedure:

CREATE OR REPLACE PROCEDURE display\_managers\_info IS

BEGIN

FOR rec IN (SELECT \* FROM managers) LOOP

DBMS\_OUTPUT.PUT\_LINE('Manager ID: ' || rec.EMPLOYEE\_ID || ', Name: ' || rec.FIRST\_NAME || ' ' || rec.LAST\_NAME || ', Job: ' || rec.JOB\_ID || ', Email: ' || rec.EMAIL);

END LOOP;

END display\_managers\_info;

Function:

CREATE OR REPLACE FUNCTION GET\_EMP\_JOB\_HISTORY (p\_employee\_id IN NUMBER)

RETURN JOB\_HISTORY\_TABLE IS

job\_history JOB\_HISTORY\_TABLE := JOB\_HISTORY\_TABLE();

BEGIN

FOR RECORD IN (SELECT EMPLOYEE\_ID, START\_DATE, END\_DATE, JOB\_ID, DEPARTMENT\_ID

FROM JOB\_HISTORY\_PARTITION

WHERE EMPLOYEE\_ID = p\_employee\_id

ORDER BY START\_DATE)

LOOP

job\_history.EXTEND;

job\_history(job\_history.COUNT) := JOB\_HISTORY\_RECORD(RECORD.EMPLOYEE\_ID, RECORD.START\_DATE, RECORD.END\_DATE, RECORD.JOB\_ID, RECORD.DEPARTMENT\_ID);

END LOOP;

RETURN job\_history;

END GET\_EMP\_JOB\_HISTORY;

Trigger:

CREATE OR REPLACE TRIGGER KEEP\_ADMINS

BEFORE DELETE ON EMPLOYEES

FOR EACH ROW

DECLARE

v\_department\_name VARCHAR2(30);

BEGIN

SELECT DEPARTMENT\_NAME INTO v\_department\_name

FROM DEPARTMENTS

WHERE DEPARTMENT\_ID = :OLD.DEPARTMENT\_ID;

IF v\_department\_name = 'Administration' THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Cannot delete employees in the Administration department.');

END IF;

END KEEP\_ADMINS;

Collections:

CREATE OR REPLACE TYPE phone\_varray AS VARRAY(4) OF VARCHAR2(20);

Exception:

CREATE OR REPLACE PROCEDURE remove\_employee (p\_employee\_id IN NUMBER) IS

employee\_not\_found EXCEPTION;

BEGIN

DECLARE

v\_count NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_count FROM EMPLOYEES WHERE EMPLOYEE\_ID = p\_employee\_id;

IF v\_count = 0 THEN

RAISE employee\_not\_found;

END IF;

END;

DELETE FROM EMPLOYEES

WHERE EMPLOYEE\_ID = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN employee\_not\_found THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || p\_employee\_id || ' does not exist.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END remove\_employee;