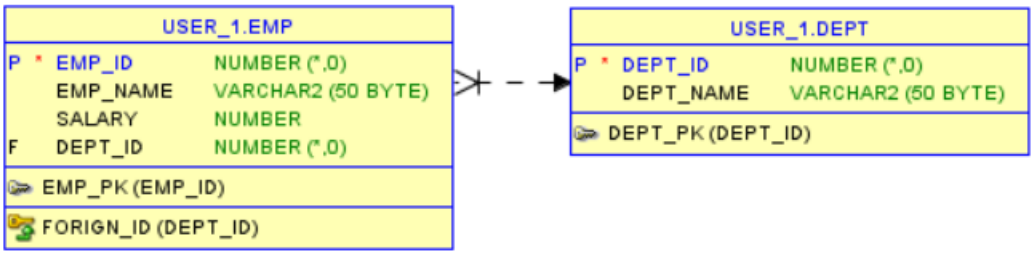
DATABASE 2
ORACLE TASK

ID	Name
20210265	حازم أحمد عبد العال شاذلي

ORACLE TASK:

An organization has a "Department" table and an "Employees" table in Oracle. The "Department" table contains information about different departments in the organization, and the "Employees" table contains information about the employees, including the department they belong to. The department table contains ID as a primary key and department name. The departments are HR, IT, and finance. The Employee table contains ID as a primary key and name, salary, and department ID as a foreign key.

ER diagram.



a) Create a Manager User and grant them a role of privileges to create two users. Let User 1 create the Employee and the Department table. Let User 2 insert 5 rows of employees. *[2 Marks]*

Sys Connection :

Create a Manager User and grant them a role of privileges to create two users.

```
--sys

create user manager identified by 123;
grant create session to manager;
grant create user to manager;
```

Manager Connection :

```
--manager
create user user_1 identified by 123;
create user user_2 identified by 123;
```

Sys Connection :

```
--sys
grant create session to user_1;
grant create table to user_1;
alter user user_1 quota 20m on system;

grant create session to user_2;
grant insert on user_1.emp to user_2;
```

Note *execute the last line after creation of user_1.emp table*

User 1 Connection :

Let User 1 create the Employee and the Department table.

```
--user 1
create table dept(
  dept_id INTEGER PRIMARY KEY,
  dept_name varchar(50)
);

create table emp(
  emp_id integer primary key,
  emp_name varchar(50),
  salary number,
  dept_id integer,
  CONSTRAINT forign_id FOREIGN KEY (dept_id) REFERENCES  dept(dept_id)
);


insert into dept values (1, 'HR');
insert into dept values (2, 'IT');
insert into dept values (3, 'Finance');

commit;
```

User_2 Connection :

```
-- user 2

-- Question 1
insert into user_1.emp values (1,'Hazem',5000,1);
insert into user_1.emp values (2,'Hossam',5000,2);
insert into user_1.emp values (3,'Hamid',5000,2);
insert into user_1.emp values (4,'Hafez',5000,2);
insert into user_1.emp values (5,'Zeyad',5000,1);
commit;
```

. Let User 2 insert 5 rows of employees.

b) Demonstrate generating a blocker-waiting situation using two transactions by user 1 and user 2. The Transaction is calling a function that raises the rate of salary by 10% for department 1. *[2 Marks]*

Sys Connection :

```
CREATE OR REPLACE FUNCTION RaiseSalary
RETURN NUMBER AS
    count_updated NUMBER := 0;
BEGIN
    FOR counter IN (SELECT salary FROM user_1.emp WHERE dept_id = 1)
    LOOP
        UPDATE user_1.emp SET salary = counter.salary * 1.1 WHERE dept_id = 1;
        count_updated := count_updated + 1;
    END LOOP;
    RETURN count_updated;
END;
```

To make the transaction between user_1 and user_2

Function should be granted to them.

```
grant execute on RaiseSalary to user_1;
grant execute on RaiseSalary to user_2;
```

User_1 conn :

```
DECLARE
    updated_count NUMBER;
BEGIN
    updated_count := sys.RaiseSalary();
END;
```

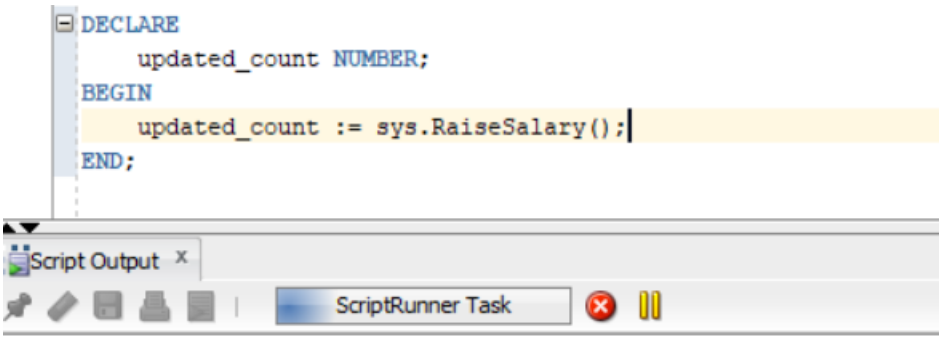
Script Output

Task completed in 0 seconds

```
anonymous block completed
```

With no commit !

User_2 conn :



User_1 (Block)

User_2(Waiting)

We can solve this problem by commit or rollback on the user_1

c) Identify the sessions in the situation using SID and serial# for both blocker and waiting sessions. *[2 Marks]*

Sys Connection :

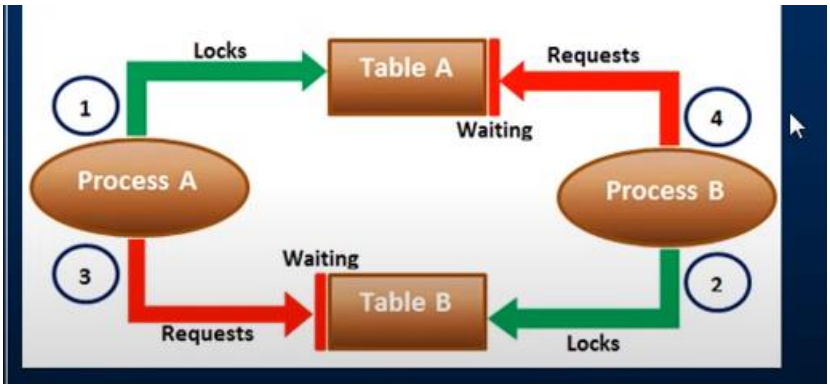
```
SELECT w.sid "Waiting Session",
       w.serial# "Waiting Serial Id",
       w.blocking_session "Blocker session id",
       w.seconds_in_wait "Waiting Session Period",
       v.sql_fulltext "Waiting Sql Statement",
       waiting.serial# "Waiting Serial",
       blocking.serial# "Blocking Serial"
FROM v$session w
JOIN v$sql v ON w.sql_id = v.sql_id
LEFT JOIN v$session waiting ON w.sid = waiting.sid
LEFT JOIN v$session blocking ON w.blocking_session = blocking.sid
WHERE w.blocking_session IS NOT NULL;
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0 seconds

	Waiting Session	Waiting Serial Id	Blocker session id	Waiting Session Period	Waiting Sql Statement	Waiting Serial	Blocking Serial
1	137	211	122	8	UPDATE USER_1.EMP SET SALARY = :B1 * 1.1 WHERE DEPT_ID = 1	211	187

d) Demonstrate a deadlock scenario and display the expected result. *[2 Marks]*



User_1 Connection :

```
update emp set SALARY = 5000 where EMP_ID = 1;

update dept set DEPT_NAME = 'AI' where DEPT_ID = 1;

rollback;
```

```
update dept set DEPT_NAME = 'CS' where DEPT_ID = 1;

update emp set SALARY = 6000 where EMP_ID = 1;
```

2 4 1 3

DEADLOCK

```
update dept set DEPT_NAME = 'CS' where DEPT_ID = 1;

update emp set SALARY = 6000 where EMP_ID = 1;

rollback;
```

Script Output | Task completed in 12.151 seconds

```
1 rows updated.
Error starting at line : 4 in command -
update emp set SALARY = 6000 where EMP_ID = 1
Error report -
SQL Error: ORA-00060: deadlock detected while waiting for resource
00060. 00000 -  "deadlock detected while waiting for resource"
*Cause:    Transactions deadlocked one another while waiting for resources.
*Action:   Look at the trace file to see the transactions and resources
           involved. Retry if necessary.
```

Expected results

IF TRANSACTION 1 COMMITED

RESULT WILL BE 'CS '

AND TRANSACTION 3 WILL NOT EXECUTE (DEADLOCK OCCURS)

IF CONNECTION 2 COMMITED , BOTH TRANSACTIONS WILL OCCUR

THE RESULT WILL BE SALARY 5000 AND DEPT WILL BE 'AI'

e) Perform the following functions *[2 Marks]*

i. Create a function that calculates the average salary for any department

Values of table to be tested on

```
insert into user_1.emp values (1,'Hazem',300,1);
insert into user_1.emp values (2,'Hossam',200,2);
insert into user_1.emp values (3,'Hamid',100,2);
insert into user_1.emp values (4,'Hafez',500,2);
insert into user_1.emp values (5,'Zeyad',400,1);
commit;
```

SYS CONNECTION :

```
CREATE OR REPLACE FUNCTION avg_salary(desired_dept_id INTEGER)
RETURN NUMBER
IS
  avg_sal NUMBER;
BEGIN
  SELECT AVG(user_1.emp.salary) INTO avg_sal
  FROM user_1.emp
  WHERE dept_id = desired_dept_id;

  RETURN avg_sal;
END;


-- set serveroutput on
DECLARE
  avg_salary_for_dept NUMBER;
BEGIN
  avg_salary_for_dept := avg_salary(2);
  DBMS_OUTPUT.PUT_LINE('Average Salary for Department ' || avg_salary_for_dept);
END;
```

Script Output x | Query Result x

Task completed in 0 seconds

```
FUNCTION AVG_SALARY compiled
anonymous block completed
Average Salary for Department 266.66666666666666666666666666666666666667
```

ii. Create a function that calculates the Total Salary in a Department.

```
CREATE OR REPLACE FUNCTION total_salary(desired_dept_id INTEGER)
RETURN NUMBER
IS
  total_sal NUMBER;
BEGIN
  SELECT SUM(user_1.emp.salary) INTO total_sal
  FROM user_1.emp
  WHERE dept_id = desired_dept_id;

  RETURN total_sal;
END;


DECLARE
  total_salary_for_dept NUMBER;
BEGIN
  total_salary_for_dept := TOTAL_SALARY(2);
  DBMS_OUTPUT.PUT_LINE('Total Salary for Department ' || total_salary_for_dept);
END;
```

Script Output | Query Result

Task completed in 0 seconds

```
FUNCTION TOTAL_SALARY compiled
anonymous block completed
Total Salary for Department 800
```

iii. Create a function that calculates the maximum Salary.

```
CREATE OR REPLACE FUNCTION getMax
RETURN NUMBER
IS
  max_sal NUMBER;
BEGIN
  SELECT MAX(user_1.emp.salary) INTO max_sal
  FROM user_1.emp;

  RETURN max_sal;
END;


DECLARE
  MAX_NUMBER NUMBER;
BEGIN
  MAX_NUMBER := getMAX();
  DBMS_OUTPUT.PUT_LINE('Maximum Salary Across All Departments: ' || MAX_NUMBER);
END;
```

Script Output | Query Result

Task completed in 0 seconds

```
FUNCTION GETMAX compiled
anonymous block completed
Maximum Salary Across All Departments: 500
```