

# Question Solve - Batch 20

1] a) primary key has to be,

- unique
- not changable
- can Identify all the cols surely

in payroll management system employees must have id's  
now the id can be formulated in many ways,

- Automatic increament number value
- jobsection + name-prefix + joining-date

suppose      name → karcim  
                  section → sales  
                  date → 23 may

pk will be → salkar2305

this is very unlikely to match with another emp.  
being unique and fixed sized it also can be indexed

in btree to make the database efficient.

b) foreign key refers to the primary key of any other table. if the primary key do not exist then it will not be inserted.

so it is confirmed that all foreign key must take data from specific source.

c) named notation for a function is when you provide the variable name and value instead of providing only the value.

in positional notation

```
calculate_area(10, 5);
```

in named notation

```
calculate_area(len => 5, wid => 10);
```

d)

```
create function is_date_in(date_p date)
return boolean
IS
Start_date date := DATE '2023-01-01';
End_date date := DATE '2023-06-30';

BEGIN
    RETURN date_p >= Start_date AND date_p <= End_date;
END;
/

create function status(amount Number)
return VARCHAR2(20)
IS
BEGIN
    IF amount > 2000000 THEN
        RETURN 'Elite';
    ELSIF amount > 1000000 AND amount <= 2000000 THEN
        RETURN 'Royal';
    ELSE
        RETURN 'Ordinary';
    END IF;
END;
/

create function status_of_customer(CID TYPE%Customer)
return VARCHAR2(20)
IS
    diposit_amount Number := 0;
    withdraw_amount Number := 0;
    balance Number;

    diposit_now Number := 0;
    withdraw_now Number := 0;
    balance_now Number;

    Cursor trans
    IS
        Select * from TRANSACTION where CID = CID;

BEGIN
    For t in trans loop
        if t.TypeTransaction = 1 THEN
            diposit_amount := diposit_amount + t.Amount;
            if is_date_in(DateTranction) THEN
                diposit_now := diposit_now + t.Amount;
            END IF;
        ELSE
            withdraw_amount := withdraw_amount + t.Amount;
            if is_date_in(DateTranction) THEN
                withdraw_now := withdraw_now + t.Amount;
            END IF;
        END IF;
    END LOOP;

    balance := diposit_amount - withdraw_amount;
    balance_now := diposit_now - withdraw_now;

    DBMS_OUTPUT.PUT_LINE('Current balance is ' || balance);

    return status(balance_now);
END;
```

2 | a)

```
CREATE TABLE EMPLOYEE (  
    EMP_ID NUMBER PRIMARY KEY,  
    EMP_NAME VARCHAR(50),  
    EMP_SALARY NUMBER  
);  
  
CREATE TABLE TRANSACTION(  
    EMP_ID NUMBER,  
    AMOUNT NUMBER,  
    DATE_TRANSACTION DATE,  
    FOREIGN KEY (EMP_ID) REFERENCES EMPLOYEE(EMP_ID)  
);  
  
CREATE FUNCTION RULE(  
    AMOUNT NUMBER  
) RETURN NUMBER IS  
BEGIN  
    IF AMOUNT > 100000 THEN  
        RETURN AMOUNT * 0.02;  
    ELSIF AMOUNT > 50000 AND AMOUNT <= 100000 THEN  
        RETURN AMOUNT * 0.05;  
    ELSE  
        RETURN AMOUNT * 0.1;  
    END IF;  
END;  
  
CREATE PROCEDURE GIVE_BONUS() IS  
    CURSOR EMP_CUR IS  
    SELECT  
        *  
    FROM  
        EMPLOYEE;  
BEGIN  
    FOR EMP_REC IN EMP_CUR LOOP  
        INSERT INTO TRANSACTION(  
            EMP_ID,  
            AMOUNT,  
            DATE_TRANSACTION  
        ) VALUES(  
            EMP_REC.EMP_ID,  
            RULE(EMP_REC.EMP_SALARY),  
            SYSDATE  
        );  
    END LOOP;  
END;
```

b) this question do not make any sense. this process will never touch the salary attribute of employee.

3(a) %Rowtype takes the dynamic type of total rows. Each field will be same as in table.

Declare

```
dept_rec department%rowtype
```

```
cursor c is
```

```
select * from department;
```

Begin

```
Open c;
```

```
Loop
```

```
Fetch c into dept_rec;
```

```
Exit when c%Notfound;
```

```
  :  
  -- other codes
```

```
End loop;
```

```
close c;
```

```
End;
```

```
/
```

b) load data

\*

infile 'emp\_data.txt' ← source text file

badfile 'emp\_data.bad' ← database file

\*

fields terminated by '\$' ← separator

trailing blankquote EOL ← file handler (new line)

(

e\_id filter filter

e\_name char(50)

dept char(20)

salary int

)

structure of table

\*

Into table employee ← name of table

e) in key value pair all the key values are primary key by that we can search the values (data).

here we see two column. One the dept name and another is number of student.

So here dept is the key.

```
CREATE PROCEDURE PRINTER(  
  ) IS  
  CURSOR DEPTS IS  
  SELECT  
    *  
  FROM  
    DEPARTMENTS;  
BEGIN  
  FOR DEPT_REC IN DEPTS LOOP  
    DBMS_OUTPUT.PUT_LINE('Department ID: '  
                          || DEPT_REC.DEPARTMENT_ID  
                          || ' Department Name: '  
                          || DEPT_REC.DEPARTMENT_NAME);  
  END LOOP;  
END;
```

4) i)

```
CREATE TABLE EMPLOYEE(  
  E_ID NUMBER(5) PRIMARY KEY,  
  E_NAME VARCHAR2(50),  
  E_SALARY NUMBER(10, 2)  
);
```

```
CREATE TABLE LEAVE_REQUEST(  
  E_ID NUMBER(5),  
  LEAVE_START DATE,  
  LEAVE_END DATE,  
  LEAVE_REASON VARCHAR2(100),  
  LEAVE_STATUS VARCHAR2(20),  
  FOREIGN KEY (E_ID) REFERENCES EMPLOYEE(E_ID)  
);
```

```
CREATE TABLE YEARLY_HOLIDAY(  
  NAME VARCHAR2(50) PRIMARY KEY,  
  START_DATE DATE,  
  DURATION_DAYS NUMBER(3),  
);
```

ii)

```
CREATE FUNCTION COUNT_WEEKENDS(
    START_DATE DATE,
    END_DATE DATE
) RETURNS NUMBER IS
    WEEKEND_COUNT NUMBER := 0;
    C_DATE DATE := START_DATE;
BEGIN
    WHILE C_DATE <= END_DATE LOOP
        IF TO_CHAR(C_DATE, 'DY') IN ('SAT', 'SUN') THEN
            WEEKEND_COUNT := WEEKEND_COUNT + 1;
        END IF;

        C_DATE := C_DATE + 1;
    END LOOP;

    RETURN WEEKEND_COUNT;
END;

CREATE FUNCTION COUNT_HOLIDAYS(START_DATE DATE, END_DATE DATE)
RETURNS NUMBER IS
    HOLIDAY_COUNT NUMBER := 0;
    C_DATE DATE := START_DATE;
    CURSOR HOLIDAYS IS
        SELECT
            *
        FROM
            YEARLY_HOLIDAY;
BEGIN
    WHILE C_DATE <= END_DATE LOOP
        FOR H IN HOLIDAYS LOOP
            IF C_DATE >= H.START_DATE AND C_DATE < H.START_DATE +
H.DURATION_DAYS THEN
                HOLIDAY_COUNT := HOLIDAY_COUNT + 1;
            END IF;
        END LOOP;

        C_DATE := C_DATE + 1;
    END LOOP;

    RETURN HOLIDAY_COUNT;
END;

CREATE FUNCTION ACTUAL_LEAVE_DAYS(START_DATE DATE, END_DATE
DATE) RETURNS NUMBER IS
    TOTAL_DAYS NUMBER := END_DATE - START_DATE + 1;
    WEEKENDS NUMBER := COUNT_WEEKENDS(START_DATE, END_DATE);
    HOLIDAYS NUMBER := COUNT_HOLIDAYS(START_DATE, END_DATE);
BEGIN
    RETURN TOTAL_DAYS - WEEKENDS - HOLIDAYS;
END;
/
```



