# Mid Notes

Thursday, February 17, 2022 11:01 PM

```
CREATE TABLE Student (
      S_ID INT PRIMARY KEY,
      S_NAME VARCHAR2(20),
      CGPA FLOAT,
      SEMESTER INT
);

INSERT INTO Student VALUES (1, 'A', 3.91, 1);
INSERT INTO Student VALUES (4, 'D', 3.88, 3);
INSERT INTO Student VALUES (13, 'C', 3.51, 2);
INSERT INTO Student VALUES (12, 'B', 3.98, 4);
```

## Miscellaneous (Covers topics from previous year)

SYSDATE ->
Current date of the system.
cur_date := SYSDATE;
month_between := MONTHS_BETWEEN(cur_date, TO_DATE('12-02-1919', 'DD-MM-YYYY'));
Note that MONTHS_BETWEEN() returns a floating point number.

%TYPE ->
Gives the datatype of any variable or row.

%ROWTYPE ->
Returns all the datatypes of a table along with the row's alias. Useful for making a duplicate record.
Students_copy Students%ROWTYPE;
(Now a duplicate record named Students_copy has been created on the alias of all the rows in the Students table. If there is a row named S_ID in Students, it will also exist in Students_copy. Useful when dealing with cursors. A cursor is essentially a record pointer and the Students_copy record will contain the fetched values of the cursor in it.)

### Ans to question 1a ->

Why use %TYPE ->
To ensure that a variable that we will use to contain the value of a row, has the same datatype as that row. For example -> student_id Students.S_ID%TYPE; (the variable student_id will have the same type as the Students.S_ID)

Where to use SELECT....FOR ->
To lock a row for one thread, so that it doesn't get affected by other threads until this one tread is done. Applicable for the DML (Insert, Update and Delete)
SELECT Students.S_ID FROM STUDENTS FOR UPDATE;
UPDATE Students SET Student.CGPA = 3.71 WHERE Student.S_ID = 190042141;
(This will lock Student.S_ID and prevent it from being tamped with until updating is done in this thread.)
Note, you can use a cursor to lock the thread.
CURSOR my_cursor IS SELECT * FROM Student FOR UPDATE;

### Ans to question 1b ->

```
CREATE OR REPLACE FUNCTION find_cgpa(stud_id INT) RETURN INT
IS
    credit_sum INT := 0;
```

```
    credit_grade_sum FLOAT := 0;
    CURSOR myC IS SELECT Grades.S_ID, Grades.C_ID, Courses.CREDIT, Grades.LETTER_GRADE FROM Students, Grades, Courses
    WHERE Students.S_ID = stud_id AND Students.S_ID = Grades.S_ID AND Courses.C_ID = Grades.C_ID;
BEGIN
    FOR myCV in myC LOOP
        credit_sum := credit_sum + myCV.CREDIT;
        credit_grade_sum := credit_grade_sum + myCV.CREDIT*myCV.LETTER_GRADE;
    END LOOP;
    RETURN credit_grade_sum/credit_sum;
END;
```

## Ans to question 2a ->

```
DROP TABLE Customers;
DROP TABLE Accounts;
DROP TABLE Owners;

CREATE TABLE Customers(
    CID INT PRIMARY KEY,
    C_NAME VARCHAR2(50),
    DOB DATE,
    ADDRESS VARCHAR2(50),
    CONTACT VARCHAR2(15)
);

CREATE TABLE Accounts (
    AID INT PRIMARY KEY,
    ACC_TYPE INT,
    BALANCE FLOAT,
    I_RATE FLOAT,
    ICP INT,
    LID DATE -- Last interest date
);

CREATE TABLE Owners (
    CID INT,
    AID INT,
    PRIMARY KEY (CID, AID)
);

CREATE TABLE Transactions (
    TID NUMBER PRIMARY KEY,
    AID INT,
    AMOUNT FLOAT,
    T_TYPE INT,
    T_DATE DATE
);
```

## Ans to question 2b ->

```
CREATE OR REPLACE FUNCTION t_id_gen(acc_id INT, t_date DATE) RETURN NUMBER
IS
    cur_seq INT := 1;
    new_t_id VARCHAR2(50);
    acc_type INT;
```

```
BEGIN
    SELECT Accounts.ACC_TYPE INTO acc_type FROM Accounts WHERE Accounts.AID = acc_id;
    SELECT COUNT(TID) INTO cur_seq FROM Transactions;
    new_t_id := TO_CHAR(acc_type) || TO_CHAR(SYSDATE, 'YYMMDD') || '.' || TO_CHAR(cur_seq);
    RETURN new_t_id;
END;
/

CREATE OR REPLACE TRIGGER auto_t_id_gen BEFORE INSERT ON Transactions FOR EACH ROW
BEGIN
    :NEW.TID := t_id_gen(:NEW.AID, :NEW.T_DATE);
END;
/
```

## Ans to question 3a->

```
CREATE OR REPLACE FUNCTION calculate_interest(acc_id INT, new_balance FLOAT) RETURN FLOAT
IS
    net_interest FLOAT := 0;
    months_between_interest FLOAT := 0;
    last_interest_date DATE;
    icp_val INT;
    i_rate FLOAT;

BEGIN
    SELECT Accounts.ICP INTO icp_val FROM Accounts
    WHERE Accounts.AID = acc_id;

    SELECT Accounts.LID INTO last_interest_date FROM Accounts
    WHERE ACCOUNTS.AID = acc_id;

    SELECT Accounts.I_RATE INTO i_rate FROM Accounts
    WHERE ACCOUNTS.AID = acc_id;

    months_between_interest := MONTHS_BETWEEN(SYSDATE, last_interest_date);

    IF icp_val = 1 THEN
        net_interest := new_balance * i_rate * months_between_interest / 12;
    ELSIF icp_val = 2 THEN
        net_interest := new_balance * i_rate * months_between_interest / 12;
    END IF;

    return net_interest;

END;
/

CREATE OR REPLACE FUNCTION calculate_balance(acc_id INT) RETURN FLOAT
IS
    net_withdraw FLOAT := 0;
    net_deposit FLOAT := 0;
    new_balance FLOAT;
    net_interest FLOAT := 0;
BEGIN
```

```
    SELECT Accounts.BALANCE INTO new_balance FROM Accounts WHERE Accounts.AID = acc_id;

    -- 1 For deposit
    SELECT SUM(Transactions.AMOUNT) INTO net_deposit FROM Transactions
    WHERE Transactions.AID = acc_id and Transactions.T_TYPE = 1 and Transactions.T_DATE = SYSDATE;

    -- 0 For withdraw
    SELECT SUM(Transactions.AMOUNT) INTO net_withdraw FROM Transactions
    WHERE Transactions.AID = acc_id and Transactions.T_TYPE = 0 and Transactions.T_DATE = SYSDATE;

    new_balance := new_balance + net_deposit - net_withdraw;

    return new_balance;
END;
/

CREATE OR REPLACE PROCEDURE end_of_day
IS
    CURSOR acc_cursor IS SELECT * FROM Accounts FOR UPDATE;
    net_interest FLOAT := 0;
BEGIN
    FOR cursor_var IN acc_cursor LOOP
        net_interest := calculate_interest(cursor_var.AID, calculate_balance(cursor_var.AID));
        If net_interest = 0 THEN
            UPDATE Accounts SET Accounts.BALANCE = calculate_balance(cursor_var.AID)
            WHERE CURRENT OF acc_cursor;

        ELSE
            UPDATE Accounts SET Accounts.BALANCE = calculate_balance(cursor_var.AID)
            + net_interest
            WHERE CURRENT OF acc_cursor;

            UPDATE Accounts SET Accounts.LID = SYSDATE
            WHERE CURRENT OF acc_cursor;

        END IF;
    END LOOP;

END;
/
```

## Cursor

Implicit cursor ->

1. "SQL" is the keyword of that cursor

2. Runs after any DML (Insert, Update, Delete). Example :

   DELETE FROM Students WHERE Student.S_ID = 190042148;
   SQL%ROWCOUNT (Will give no of rows affected by the statement)
   SQL%ISFOUND (Will give TRUE if the row to be affected is found or the statement executed successfully.)
   SQL%ISOPEN (Will always give false for Implicit cursor as they are closed right after usage)

Explicit cursor ->

1. Declare like so : CURSOR my_cursor IS SELECT * FROM Students; (We can specify which attributes to take in the SELECT statement)
2. Open before using : OPEN my_cursor;
3. Use in a loop along with a predeclared record variable.
   ```
   my_record Students%ROWTYPE;
   LOOP
       FETCH my_cursor INTO my_record.S_ID, my_record.S_NAME, my_record.CGPA, my_record.SEMESTER;
       DBMS_OUTPUT.PRINT_LINE(my_record.S_ID || my_record.S_NAME || my_record.CGPA || my_record.SEMESTER);
       EXIT WHEN my_cursor%NOTFOUND;
   END LOOP;
   ```
4. Close after usage : CLOSE my_cursor;
5. Or, you may directly use the cursor in a for loop (Automatically opens and closes cursor. No exit condition is required in loop):
   ```
   FOR cursor_var in my_cursor LOOP
       DBMS_OUTPUT.PRINT_LINE(my_record.S_ID || my_record.S_NAME || my_record.CGPA || my_record.SEMESTER);
   END LOOP;
   ```
6. Explicit can't be used to update a table's contents directly, but we can use the cursor alongside the UPDATE statement. For example (Increasing all the SEMESTER value by 1):
   ```
   DECLARE
       CURSOR my_c IS SELECT * FROM Student FOR UPDATE;
   BEGIN

       FOR my_c_var IN my_c LOOP

           IF my_c_var.SEMESTER < 4 THEN
               -- Update the current semester by 1
               UPDATE Student SET Student.SEMESTER = my_c_var.SEMESTER + 1 WHERE Student.S_ID = my_c_var.S_ID;

           END IF;

       END LOOP;

       DBMS_OUTPUT.PUT_LINE('Done');

   END;
   /
   ```
   (Note that you can also use the update statement like so:
   `UPDATE Student SET Student.SEMESTER = my_c_var.SEMESTER + 1 WHERE CURRENT OF my_c; )`

## Table Space

1. The space hierarchy in Oracle database : Byte -> Block -> Extent -> Segments

2. Datafile extensions : .dbf and .ora

3. Tablespaces are logical units that contains one or more datafiles.

4. Creating a tablespace :
   ```
   CREATE TABLESPACE my_space
   DATAFILE 'C:/Oracle/my_datafile.dbf'
   SIZE 100M
   EXTENT MANAGEMENT AUTOALLCOATE;
   ```

(Default extent size: 64KB)

5. Altering the visibility of a tablespace:
   ALTER TABLESPACE my_space READ ONLY;
   ALTER TABLESPACE my_space READ WRITE;
   ALTER TABLESPACE my_space OFFLINE; (Makes it invisible)
   ALTER TABLESPACE my_space ONLINE;

6. Adding another datafile:
   ALTER TABLESPACE my_space ADD DATAFILE 'C:/Oracle/my_datafile2.dbf' SIZE 100M;

7. Adding a table into another tablespace (By default, the table is saved in the tablespace that the user is designated on):

   ```
   CREATE TABLE Student (
        S_ID INT PRIMARY KEY,
        S_NAME VARCHAR2(20),
        CGPA FLOAT,
        SEMESTER INT
   ) TABLESPACE my_space;
   ```

   You can access this table from other tablespace via the association name (my_space.Student)

8. Viewing the free space of a tablespace
   SELECT TABLESPACE_NAME, SUM(BYTES)/1024 'Remaining Space (KB)' FROM DBA_FREE_SPACE;

9. Granting a tablespace to a user:
   ALTER USER senpai GRANT QUOTA 10M ON my_space;

   For unlimited quota:
   ALTER USER senpai GRANT QUOTA UNLIMITED ON my_space;

10. Changing the default tablespace:
    ALTER USER senpai SET DEFAULT TABLESPACE my_space;

11. Changing the tablespace size(by increasing the size of the datafile)
    ALTER DATABSE
    DATAFILE "my_datafile" RESIZE 250M;