Lab 5
Advanced Data Manipulation

CSE 4308

DATABASE MANAGEMENT SYSTEMS LAB

SEPTEMBER 10, 2023

Contents

1 Some Date Funtions **2**
1.1 Current_date . . . 2
1.2 TO_DATE . . . 2
1.3 TO_CHAR . . . 2
1.4 Extraction . . . 2
1.5 Last_Day . . . 2
1.6 Next_Day . . . 2
1.7 Months_Between . . . 2
1.8 Add_Months . . . 3

2 Some String Funtions **3**
2.1 Length . . . 3
2.2 Lower . . . 3
2.3 Upper . . . 3
2.4 Initcap . . . 3
2.5 Trim . . . 4
2.6 Lpad . . . 4
2.7 Rpad . . . 4
2.8 Replace . . . 4

3 Handling Null Value **4**

4 Outer Join **5**

5 Nested DDL and DML **5**
5.1 Table Creation . . . 5
5.2 Insertion . . . 5
5.3 Update . . . 6
5.4 Delete . . . 6

6 SQL CASE Examples **6**

7 Task **7**

1 Some Date Funtions

1.1 Current_date

To get the current, Oracle provides two default functions namely **CURRENT_DATE** and **sysdate** the syntax is following;

```
SELECT CURRENT_DATE FROM DUAL;
```

or,

```
SELECT sysdate FROM DUAL;
```

1.2 TO_DATE

This function is used to convert a date from a DATE value to a specified date format. Example,

```
SELECT TO_DATE('20 APR 2020', 'DD MON YYYY')CONVERTED_DATE
FROM dual;
```

1.3 TO_CHAR

This function converts a date which is in string type to date value. Example,

```
SELECT TO_CHAR(sysdate, 'DD-MM-YYYY')NEW_DATE
FROM dual;
```

1.4 Extraction

To extract day, month or year, one can use **EXTRACT ()** function. For example,

```
SELECT EXTRACT(YEAR FROM TO_DATE('29-Apr-2020 05:30:20',
                                 'DD-Mon-YYYY HH24:MI:SS')) YEAR
FROM DUAL;
```

Similarly, we can extract month and day too.

1.5 Last_Day

This function is used to return the last day of the month of the particular date. For instance,

```
SELECT LAST_DAY(sysdate) LAST_DAY
FROM dual;
```

1.6 Next_Day

This function is used to return the date of next day particular day. For example,

```
SELECT NEXT_DAY(SYSDATE,'FRIDAY')
FROM DUAL;
```

1.7 Months_Between

This function is used to measure the months between two dates and the syntax is as follows;

```
SELECT MONTHS_BETWEEN( sysdate, DATE '2011-04-02' ) MONTH_DIFFERENCE
FROM DUAL;
```

1.8 Add_Months

This function adds N months to a date and returns the same day N month after.

```
SELECT ADD_MONTHS( sysdate, 2 ) NEWDATE
FROM dual;
```

To add a year, we have to convert it into months and to add day we can simple add the day. Let's say,

```
SELECT sysdate+10 as NEWDATE
FROM dual;
```

2 Some String Funtions

2.1 Length

The String **LENGTH()** function in Oracle is used to return the length of a given string. For example;

```
SELECT LENGTH('HELLO') FROM DUAL;
```

2.2 Lower

The string **LOWER()** function in Oracle is used to return a specified character expression in lowercase letters. The following is the syntax to use the LOWER function in Oracle.

```
SELECT LOWER('Hello') FROM DUAL;
```

2.3 Upper

The string **UPPER()** function in Oracle is used to return a specified character expression in uppercase letters. The following is the syntax to use the UPPER function in Oracle.

```
SELECT UPPER('Hello') FROM DUAL;
```

2.4 Initcap

The string **INITCAP()** function in Oracle is used to set the first letter of each word in uppercase and rest all other letters in lowercase. For example;

```
SELECT INITCAP('HELLO') FROM DUAL;
```

2.5 Trim

The string TRIM function in Oracle is used to remove the leading or trailing characters (or both) from a character string. If trim_character or trim_source is a character literal, then you must enclose it in single quotes. If you specify **LEADING**, then Oracle removes any leading characters equal to trim_character and for **TRAILING**, it removes any trailing characters equal to trim_character. If you specify BOTH or none of the three, then Oracle removes leading and trailing characters equal to trim_character. Lastly, if you do not specify trim_character, then the default value is a blank space. For example;

```
SELECT TRIM('     Removing Leading White Spaces  ') LRTRIM FROM DUAL;
```

```
SELECT TRIM(LEADING '6' FROM '660123') LRTRIM FROM DUAL;
```

```
SELECT TRIM(TRAILING '5' FROM '123455') LRTRIM FROM DUAL;
```

2.6 Lpad

LPAD function is used to fill a string with a specific character on the left side of a given string.

```
SELECT LPAD('Hello',10,'+') PADL FROM DUAL;
```

It will produce a 10-character string left padded with '+'.

2.7 Rpad

Similar to LPAD, RPAD function is used to fill a string with a specific character on the right side.

```
SELECT RPAD('Hello',10,'+') PADL FROM DUAL;
```

It will produce a 10-character string right padded with '+'.

2.8 Replace

The string REPLACE function in Oracle is used to return a string with every occurrence of search_string replaced with replacement_string. For example;

```
SELECT REPLACE('JACK and JUE','J','BL') "New String" FROM DUAL;
```

Here, 'J' will be replaced by 'BL'.

3 Handling Null Value

The Oracle NVL function lets you substitute a value when a null value is encountered. For example,

```
SELECT NVL(supplier_city, 'n/a') FROM suppliers;
```

```
SELECT NVL(commission, 0) FROM sales;
```

4 Outer Join

An extension of the join operation that avoids loss of information. Sometimes it is necessary to know both matched and non-matched records. The non-matched values are replaced by NULL. This type of joining is termed as "Outer Join". There are three types of outer joins:

- **Left Outer Join**: All records from the first table and only the matched part from the second one.

```
SELECT Name, course_id
FROM STUDENT LEFT JOIN TAKES
ON STUDENT.ID=TAKES.ID;
```

Alternatively,

```
SELECT Name, course_id
FROM STUDENT, TAKES
WHERE STUDENT.ID=TAKES.ID (+);
```

- **Right Outer Join**: All records from the second table and only the matched part from the first one.

```
SELECT course_id, ID
FROM TAKES RIGHT JOIN COURSE
ON TAKES.course_id=COURSE.course_id;
```

Alternatively,

```
SELECT course_id, ID
FROM TAKES, COURSE
WHERE TAKES.course_id (+) =COURSE.course_id;
```

- **Full Outer Join**: All records from both tables. The example is as follows;

```
SELECT a Name, b. department_name
FROM STUDENT a FULL OUTER JOIN DEPT b
ON a.department_id = b.department_id;
```

5 Nested DDL and DML

5.1 Table Creation

When we need to create a similar table structure, we often seek a shortcut to lend the structure from the existing one. Default copies both structure and data. To exclude data we often need to trick in where clause.

```
CREATE TABLE NEW_STUDENTS
AS SELECT ID, NAME, GPA FROM STUDENT;
```

5.2 Insertion

We might want to insert tuples dynamically on the basis of the result of a query. In that case, we can use sub-query in insert statement.

```
INSERT INTO INSTRUCTOR
SELECT ID, name, dept_name
FROM STUDENT
WHERE total_cred > 144;
```

5.3 Update

Sometimes we need to update information based on other information stored in the same or another table In that case, we can use a sub-query in the update statement.

```
UPDATE INSTRUCTOR
SET salary = salary * 1.05
WHERE salary < (SELECT avg (salary)
FROM INSTRUCTOR);
```

5.4 Delete

Similar to update operation, we can use a sub-query in the delete statement.

```
DELETE FROM INSTRUCTOR
WHERE salary < (SELECT avg (salary)
FROM INSTRUCTOR);
```

6 SQL CASE Examples

The following SQL goes through conditions and returns a value when the first condition is met:

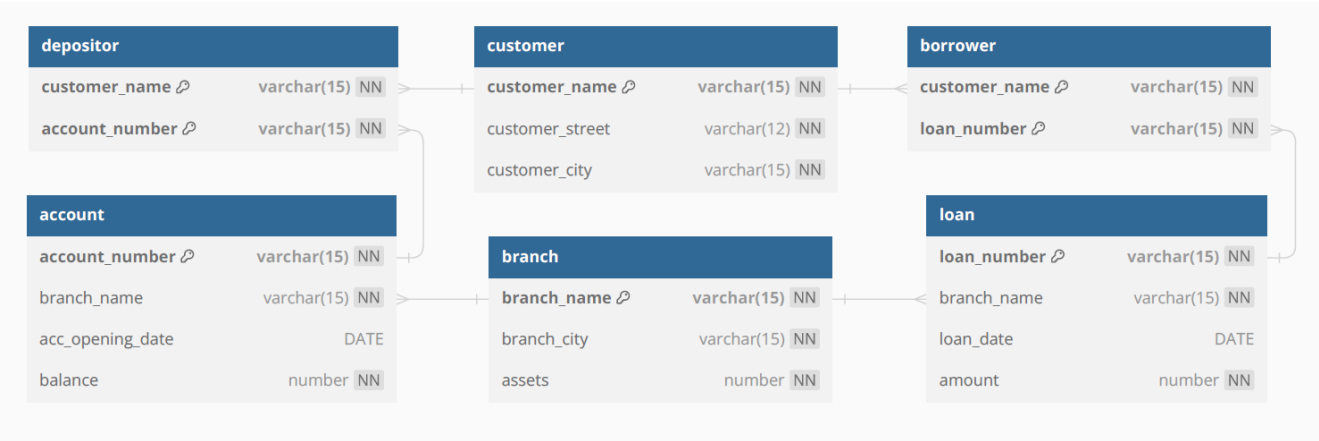
```
SELECT ID, Name,
CASE
    WHEN Marks > 40 THEN 'Pass'
    ELSE 'Fail'
END AS status
FROM STudent;
```

We can use conditions in the update statement too. For example;

```
UPDATE INSTRUCTOR
SET salary = CASE
WHEN salary <= 100000 THEN salary * 1.05
ELSE salary*1.03
END;
```

7 Task

Execute the `movie.sql` script using command. It creates a set of tables along with values that maintain the following schema:



Here, the boldfaces denote the primary keys and the arcs denote the foreign key relationships.
In this lab, you have to write all SQL statements in an editor first and save them with .sql extension. Then execute the SQL script.
Write SQL statements for the following queries:

1. Find all customer names and their cities who have a loan but not an account.
2. Find all customer names who have an account, as well as a loan.
3. Show the count of accounts that were opened in each month along with the month.
4. Find the months between the last acc_opening_date and last loan_date of customer 'Smith'.
5. Find the customer name who has the highest total balance (may have multiple accounts).
6. Find all the name of the customers whose total balance is sufficient to pay off at least one of their loans and display their names with the suffix 'Eligible'.
7. Show branch names suffixed with 'Elite' for total balances > (average balance + 500), 'Moderate' for balances within (average balance + 500) to (average balance - 500), else 'Poor'.
8. Find the branch information for cities where at least one customer lives who does not have any account or any loans. The branch must have given some loans and has accounts opened by other customers.
9. Create a new customer_new table using a similar structure to the customer table.
10. In the customer_new table insert only those customers who have either an account or a loan.
11. Add a new column Status in customer_new table of varchar2(15) type.
12. For each customer if his/her total balance is greater than the total loan then set the status 'In savings', if the vice versa then 'In loan' and if both of the amounts are the same then 'In breakeven'.