Advanced Teradata

In this lecture

- Macros.
- Stored Procedures.
- OLAP functions.

Macros

- A **macro** is a set of SQL statements which are stored and executed by calling the macro name.
- The definition of macros is stored in Data Dictionary. Users only need `EXEC` privilege to execute the macro.
- Users don't need separate privileges on the database objects used inside the macro.
- Macro statements are executed as a single transaction:
    - If one of the SQL statements fails, then all the statements are rolled back.
    - Macros can accept parameters.
    - Macros can contain DDL statements, but that should be the last statement.

Create Macros

```
CREATE MACRO <macroname> [(parameter1, parameter2,...)]
(
<sql statements>
);
```

Example

```
CREATE MACRO Get_Emp AS
(
SELECT
EmployeeNo,
FirstName,
LastName
FROM
employee
ORDER BY EmployeeNo;
);
```

```
EXEC Get_Emp;
```

Parameterized Macros

Macro parameters are referenced with `:Param;` .

```
CREATE MACRO Get_Emp_Salary(EmployeeNo INTEGER) AS
(
SELECT
EmployeeNo,
NetPay
FROM
Salary
WHERE EmployeeNo = :EmployeeNo;
);
```

```
EXEC Get_Emp_Salary(101);
```

Stored Procedures

Stored Procedures

- A stored procedure contains a set of SQL statements and procedural statements.
- The definition of stored procedure is stored in database and the parameters are stored in data dictionary tables.

Stored Procedures (cont.)

Advantages

- Stored procedures reduce the network load between the client and the server.
- Provides better security since the data is accessed through stored procedures instead of accessing them directly.
- Gives better maintenance since the business logic is tested and stored in the server.

Example

```
CREATE PROCEDURE <procedurename> ( [parameter 1 data type, parameter 2 data type..] ) BEGIN <SQL or SPL statements>; END;
```

Example (cont.)

```
CREATE PROCEDURE InsertSalary(
 IN in_EmployeeNo INTEGER, IN in_Gross INTEGER,
 IN in_Deduction INTEGER, IN in_NetPay INTEGER
)
BEGIN

INSERT INTO Salary
( EmployeeNo, Gross, Deduction, NetPay )
VALUES
(:in_EmployeeNo, :in_Gross, :in_Deduction, :in_NetPay);

END;
```

```
CALL InsertSalary(105,20000,2000,18000);
```

Differences between macros and procedures

- The macro contains only SQL and maybe dot commands that are only for use in BTEQ.
- A marco is normally a SELECT results in rows being returned to the user.
- A stored procedure does not return rows to the user like a macro. Instead, the selected column or columns must be used within the procedure.

Differences between macros and procedures (cont.)

- Like a macro, stored procedures allow parameter values to be passed to it at execution time.
- Unlike a macro that allows only input values, a stored procedure also provides output capabilities.
- A stored procedure only returns output values to a user client as output parameters, not as rows.

OLAP

OLAP functions

- OLAP functions are similar to aggregate functions.
    - Aggregate functions will return only one value.
    - OLAP function will provide the individual rows in addition to the aggregates.
- Aggregation functions can be:
- `SUM`, `COUNT`, `AVG`, `MIN`, `MAX`, `MSUM`, `MAVG`, `MDIFF`, `CSUM`
- `RANK`, `DENSE_RANK`, `ROW_NUMBER`, `LAG`, `LEAD`, `FIRST_VALUE`, `LAST_VALUE`

```
analytical_function_name([column_name])
OVER (
[PARTITION BY COLUMN1] [ORDER BY COLUMN2][DESC/ASC]
[ROWS BETWEEN n FOLLOWING|PRECEDING(start window)
AND m FOLLOWING|PRECEDING|CURRENT ROW)(end window)]
)
```

Window features (optional)

- **PARTITION BY:** Perform analysis within sub categories.
    - **Example:** Calculate salary per department.
- **ORDER BY:** In which order should columns be processed.
- **ROWS BETWEEN:** Check within a window of rows. Must specify start/end (or `UNBOUNDED`).

Example

Consider the following Salary table.

EmployeeNo	Gross	Deduction	NetPay
101	40,000	4,000	36,000
102	80,000	6,000	74,000
103	90,000	7,000	83,000
104	75,000	5,000	70,000

Example (cont.)

- Calculate running total of `NetPay`

```
SELECT
EmployeeNo, NetPay,
SUM(Netpay)
OVER(ORDER BY EmployeeNo
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
as RunningSalary
FROM Salary;
```

RANK

- `RANK` function orders the records based on the column provided.
- `RANK` function can also filter the number of records returned based on the rank.

```
RANK() OVER
([PARTITION BY columnnlist]
 [ORDER BY columnlist][DESC|ASC])
```

Example

Consider the following Employee table.

EmployeeNo	JoinedDate	DepartmentID	BirthDate
101	3/27/2005	1	1/5/1980
102	4/25/2007	2	3/5/1983
103	3/21/2007	2	4/1/1983
104	2/1/2008	2	11/6/1984
105	1/4/2008	3	12/1/1984

Example (cont.)

The following query orders the records of the employee table by `Joined Date` and assigns the ranking on `JoinedDate`.

```
SELECT EmployeeNo, JoinedDate,RANK()
OVER(ORDER BY JoinedDate) as Seniority
FROM Employee;
```

Example (cont.)

Running the above query we get:

```
EmployeeNo JoinedDate Seniority
---------- ---------- ----------
101        2005-03-27             1
103        2007-03-21             2
102        2007-04-25             3
105        2008-01-04             4
104        2008-02-01             5
```

PARTITION BY

- `PARTITION BY` clause groups the data by the prescribed columns and performs the OLAP function within each group.
- Following is an example of the query that uses PARTITION BY clause.

```
SELECT EmployeeNo, JoinedDate, RANK()
OVER(PARTITION BY DeparmentNo ORDER BY JoinedDate)
as Seniority
FROM Employee;
```

Result

```
EmployeeNo DepartmentNo JoinedDate Seniority
---------- ------------ ---------- -----------
101 1 2005-03-27 1
103 2 2007-03-21 1
102 2 2007-04-25 2
104 2 2008-02-01 3
105 3 2008-01-04 1
```

Filtering: `QUALIFY`

- Similar to `WHERE` or `HAVING`

```
SELECT EmployeeNo, JoinedDate
,RANK() OVER(ORDER BY JoinedDate) as Seniority
FROM Employee
QUALIFY (
        RANK() OVER(ORDER BY JoinedDate)
) < 3;
```

Your turn!

Exercise

- Use the script `04 OLAP.sql` to create a sample table.
- Write queries for:
    i. Total sum of salary within department against each of employee of that department
    ii. Cumulative salary within each department
    iii. Total count of employees within department against each of employee of that department.
    iv. Find employees whose total department salary is greater than 9000.