Advanced Structured Query Language (SQL)-Part 2

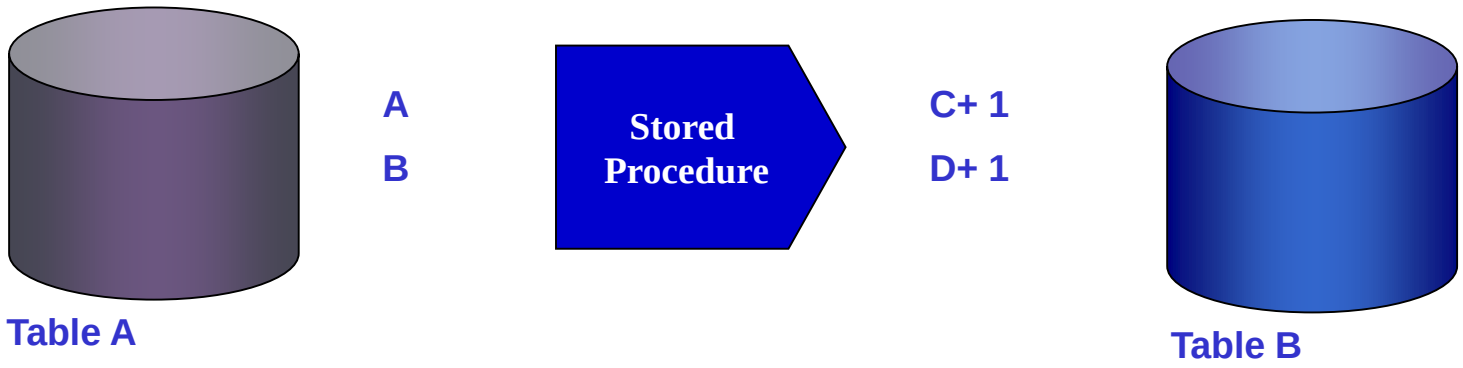
Lecture 9

Learning Outcomes

- In this chapter, students will learn:
  - How to create and use stored procedure
  - How to create and use user-defined function

Stored Procedures

- Named collection of procedural and SQL statements
- Just like triggers, stored procedures are stored in the database
  - *Difference*: can pass argument in stored procedures

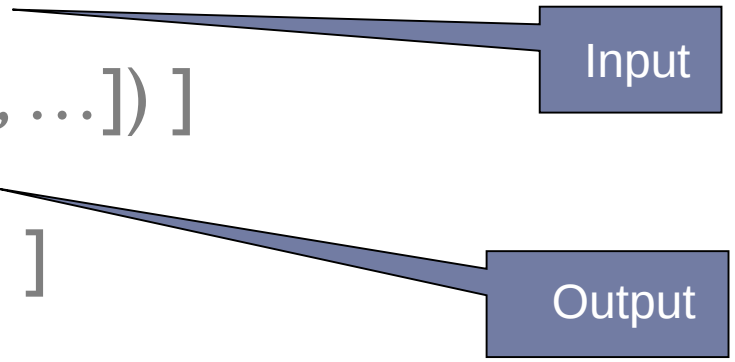


You can manipulate the whole set of records (adding, updating, deleting…) using stored procedure

Stored Procedures

- Syntax:

```
CREATE PROCEDURE  <procedure_name>
    [ (IN input_param1  data-type
            [, input_param2 data-type, …]) ]
        [ OUT (output_col1  data-type
            [, output_col2 data-type, … ]) ]
BEGIN
     SQL statements;
END
```



Stored Procedures

- Syntax to invoke a stored procedure:

  CALL <store_procedure_name (parameter,...)>

Stored Procedures Syntax- Parameter

- Each parameter can be in either IN, OUT, or INOUT mode.
- IN: read-only. You can reference an IN parameter inside a procedure, but you cannot change its value. *(DB2 uses IN as the default mode.)*
- OUT: writable. Typically, you set a returned value for the OUT parameter and return it to the calling program.
- INOUT: both readable and writable. The procedure can read and modify it.

Stored Procedures Syntax- Body

- The procedure body has two parts. Only the executable part is mandatory whereas the declarative part is optional.

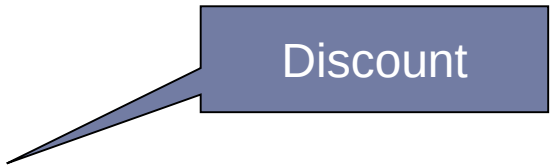
- 1) Declarative part: Declare variables, constants, cursors, etc.
- 2) Executable part: Contains at least one executable statement that implement specific business logic.

Stored Procedures

- Example:

```
CREATE PROCEDURE  Prc_Prod_Disc (IN Discount Decimal(7,2))
BEGIN
        UPDATE Product
        SET P_Discount = P_Discount + Discount
        WHERE P_QOH >= P_Min * 2;

       --more sql commands;
END
```



- To execute:

```
CALL Prc_Prod_Disc (0.05)
```

Stored Procedures

- Example:

```
CREATE PROCEDURE  Prc_Prod (IN Prod_Type Char(1),
                  Prod_Name Varchar(20), Prod_Price Decimal(7,2))
BEGIN
      IF (Prod_Type = 'X') THEN
         INSERT INTO Prod_A
            VALUES (Prod_Name, Prod_Price);
      ELSE
         INSERT INTO Prod_B
            VALUES (Prod_Name, Prod_Price);
      END IF;
END@

            CALL Prc_Prod ('X', 'Table', 59.90)
```

Stored Procedures

- Example:

```
CREATE PROCEDURE getInvoiceDetails (IN inv_ID int)
BEGIN

    SELECT * FROM Invoice WHERE Invoice_ID = inv_ID;

  OPEN c;

END@
```

```
CALL getInvoiceDetails (10001);
```

Store Procedure

- Drop procedure:

- DROP PROCEDURE proc_name;

- DROP PROCEDURE ProcProduct ;

Stored Procedures

- **Advantages**
  - Reduce network traffic and increase performance
    - Because the procedure is stored at the server, **No** transmission of individual SQL statements over network
  - Reduce code duplication by means of code isolation and code sharing
    - Create PL/SQL that are called by application programs
      - **Minimize** chance of **errors** and **cost** of application development and maintenance

Stored Procedures – Advantages vs Disadvantages

Advantages	Disadvantages
It is faster.	It is difficult to debug.
It is pre-compiled.	Need expert developer, since difficult to write code.
It reduces network traffic.	It is database dependent.
It is reusable.	It is non-portable.
It's security is high .	It is expensive.

PL/SQL Functions

- Consists of two sections: declarative section and executable section sections.
- Unlike a procedure, you must have at least one RETURN statement in the executable statement.

PL/SQL Functions

- **Create Function** – defines a **user-defined** function in a database server
- Syntax:

```
CREATE FUNCTION <function_name> (<parameter>)
        RETURNS TABLE <return_col-name return_data-type>
    LANGUAGE SQL
    READS SQL DATA
    NO EXTERNAL ACTION
    DETERMINISTIC
    RETURN
  <sql_commands>
```

PL/SQL Functions

- **Example**: Create a function that returns *Student_Name* and *Student_ID* that has CGPA greater than 2

```
CREATE FUNCTION getCGPAfunction (CGPA  int)
RETURNS TABLE (student_name  varchar(100),  student_ID  int)
LANGUAGE SQL
READS SQL DATA
NO EXTERNAL ACTION
DETERMINISTIC
RETURN
    Select student_name, student_ID from StudentCGPA
     where CGPA > getCGPAfunction.CGPA;
```

To execute: **Select * from** table(getCGPAfunction (2))

PL/SQL Functions

- **Example**: Create a function that returns *StudentID*and *StudentCGPA* where *StudentID*= 12345

```
CREATE FUNCTION getStudentDetails(stuID int)
RETURNS TABLE (StudentID int, StudentCGPA decimal(4,2))
LANGUAGE SQL
READS SQL DATA
NO EXTERNAL ACTION
DETERMINISTIC
RETURN
    SELECT StudentID, StudentCGPA FROM Student
   WHERE StudentID = getStudentDetails.stuID;
```

- To execute: SELECT * FROM table(getStudentDetails(12345))

PL/SQL Functions

- Remember **not** to be confuse between aggregate function (i.e., MIN, MAX, AVG, SUM, COUNT) and stored/build-in functions (YEAR(), MONTHNAME(), LENGTH(), ….

PL/SQL Functions

- Drop function:

- DROP FUNCTION function_name;

- DROP FUNCTION get_total_sales;