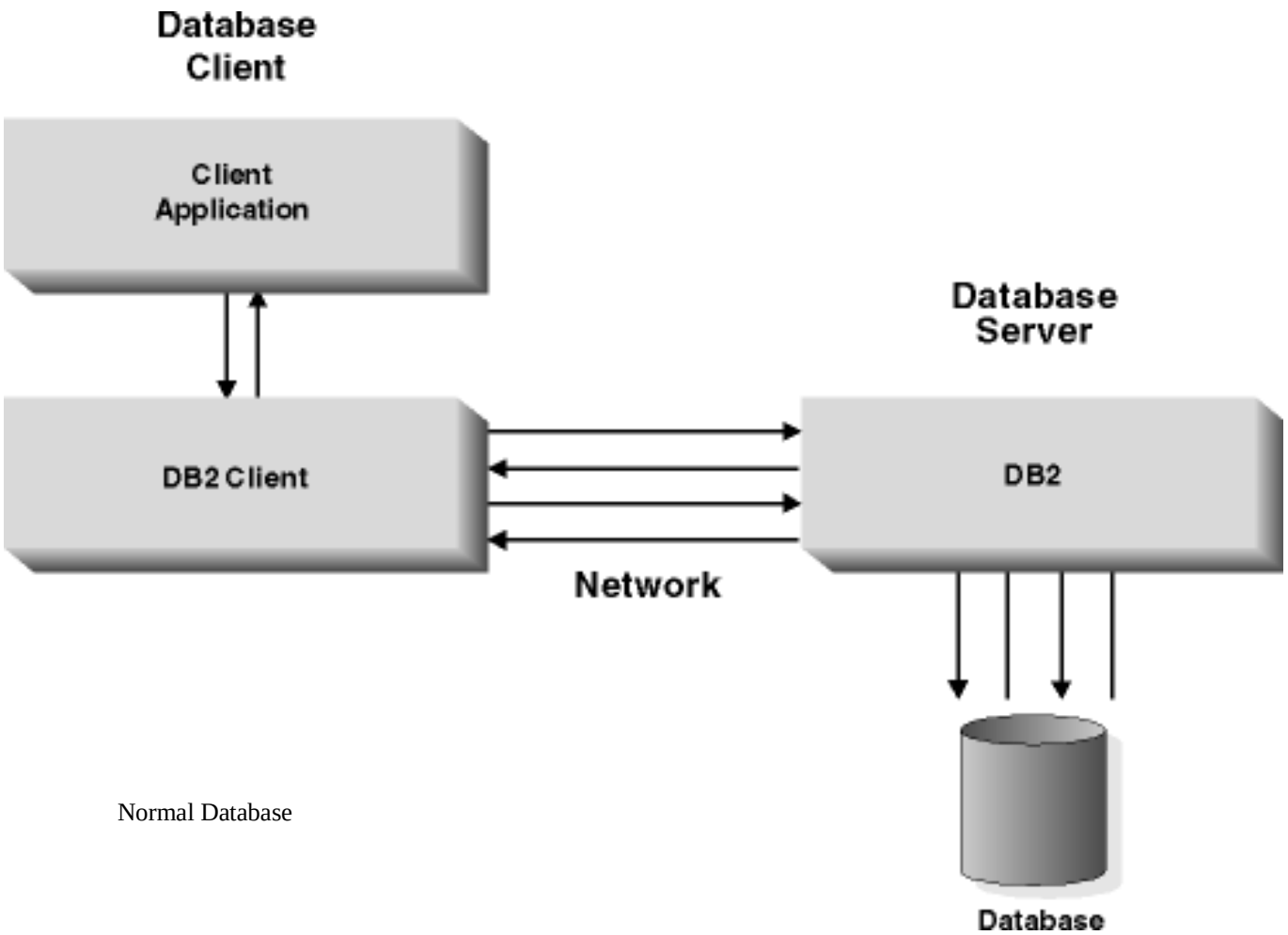
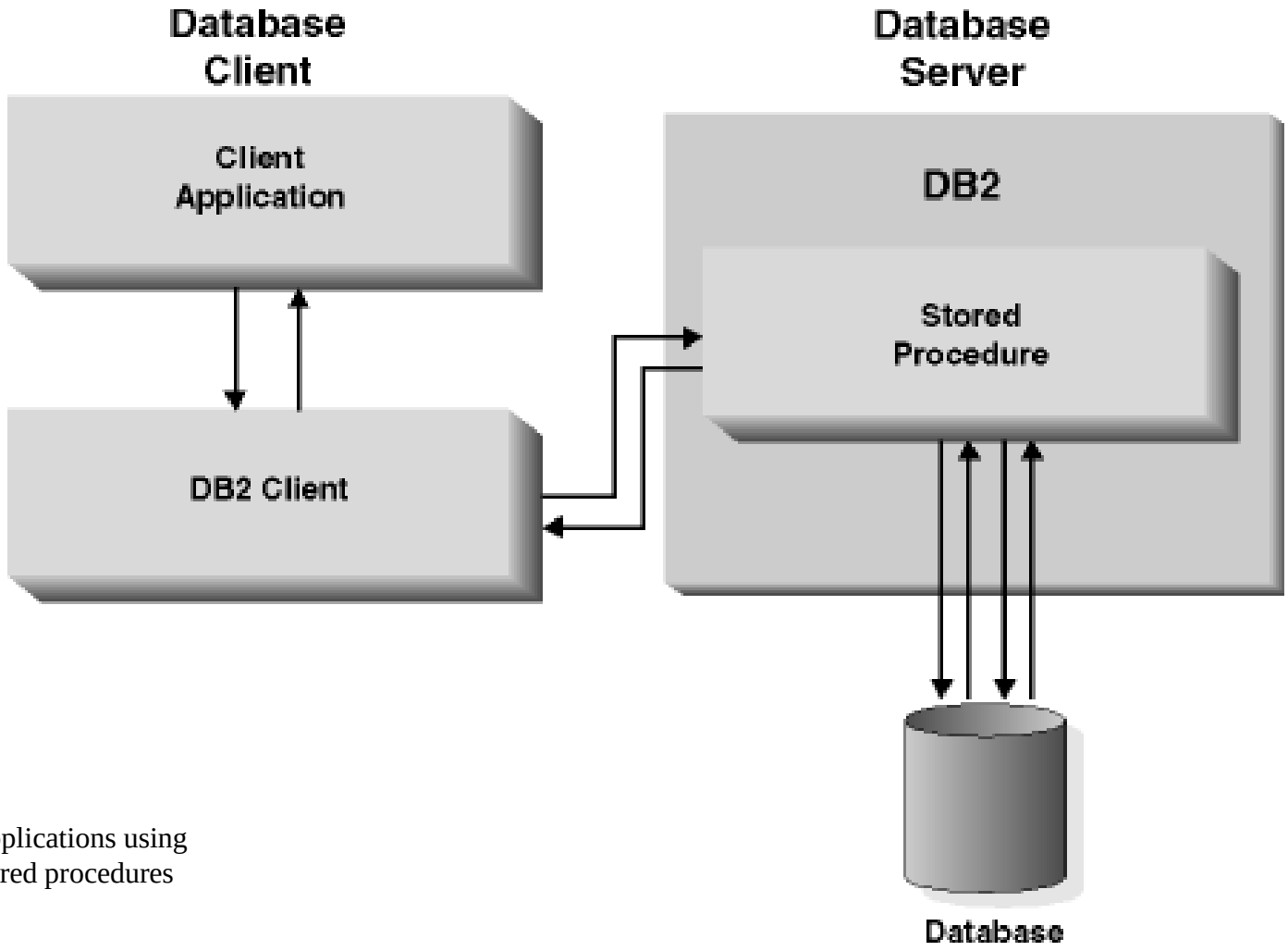
Stored Procedure Language

Stored Procedure Overview

- Stored Procedure is a function in a shared library accessible to the database server
- can also write stored procedures using languages such as C or Java
- Advantages of stored procedure : Reduced network traffic
- The more SQL statements that are grouped together for execution, the larger the savings in network traffic



Normal Database



Applications using
stored procedures

Writing Stored Procedures

- Allows local variables, loops, procedures, examinationof one tuple ar a time.

Rough Form

DECLARE

Declarations

BEGIN

Executable statements

END

@

- The DECLARE portion is optional
- Alternate terminating character in DB2 CLP scripts: ('@'), needed for ending and running.

Simplest Form : Sequence of Modifications

Employee(name , ssn, salary)

```
BEGIN
        INSERT INTO EMPLOYEE VALUES ('Sharad', 123, 234);
        DELETE FROM EMPLOYEE WHERE ssn = 234;
END
@
```

To process the DB2 CLP script from the command line:

db2 -tdterm-char -vf script-name

Writing Stored Procedures

- Tasks performed by the client application
- Tasks performed by the stored procedure, when invoked
- The CALL statement
- Explicit parameter to be defined :
  - **IN**: Passes a value to the stored procedure from the client application
  - **OUT**: Stores a value that is passed to the client application when the stored procedure terminates.
  - **INOUT :** Passes a value to the stored procedure from the client application, and returns a value to the Client application when the stored procedure terminates

```
CREATE OR REPLACE PROCEDURE <name> (<arglist>) AS <declarations>
        BEGIN
        <procedure statements>
        END
      @
```

EXAMPLE:

```
CREATE PROCEDURE UPDATE_SALARY_1        (1)
  (IN EMPLOYEE_NUMBER CHAR(6),          (2)
  IN RATE INTEGER)                      (2)
  LANGUAGE SQL                          (3)
  BEGIN
    UPDATE EMPLOYEE                     (4)
    SET SALARY = SALARY * (1.0 * RATE / 100.0 )
    WHERE SSN = EMPLOYEE_NUMBER;
  END
```

LANGUAGE value of SQL and the BEGIN...END block, which forms the procedure body, are particular to an SQL procedure

1)The stored procedure name is UPDATE_SALARY_1.

2)The two parameters have data types of CHAR(6) and INTEGER. Both are input parameters.

3)LANGUAGE SQL indicates that this is an SQL procedure, so a procedure body follows the other parameters.

4)The procedure body consists of a single SQL UPDATE statement, which updates rows in the employee table.

Some Valid SQL Procedure Body Statements

- CASE statement
- FOR statement
- GOTO statement
- IF statement
- ITERATE statement
- RETURN statement
- WHILE statement

- **Invoking Procedures**

  Can invoke Stored procedure stored at the location of the database by using the SQL CALL statement

- **Nested SQL Procedures:**

  To call a target SQL procedure from within a caller SQL procedure, simply include a CALL statement with the appropriate number and types of parameters in your caller.

```
CREATE PROCEDURE NEST_SALES(OUT budget DECIMAL(11,2))
                LANGUAGE SQL
                BEGIN
                  DECLARE total INTEGER DEFAULT 0;
                  SET total = 6;
                  CALL SALES_TARGET(total);
                  SET budget = total * 10000;
                END
```

CONDITIONAL STATEMENTS:

```
IF <condition> THEN
    <statement(s)>
ELSE
    <statement(s)>
END IF;
```

Loops

```
LOOP
    ……
    EXIT WHEN <condition>
    ……
END LOOP;
```

EXAMPLE :

```
CREATE PROCEDURE UPDATE_SALARY_IF
        (IN employee_number CHAR(6), IN rating SMALLINT)
        LANGUAGE SQL
        BEGIN
    SET counter = 10;
    WHILE (counter > 0) DO
        IF (rating = 1)
                                THEN UPDATE employee
                SET salary = salary * 1.10, bonus = 1000
                WHERE empno = employee_number;
          ELSEIF (rating = 2)
                                THEN UPDATE employee
                SET salary = salary * 1.05, bonus = 500
                WHERE empno = employee_number;
          ELSE UPDATE employee
                SET salary = salary * 1.03, bonus = 0
                WHERE empno = employee_number;
          END IF;
  SET counter = counter – 1;
          END WHILE;
        END
     @
```

EXAMPLE :

The procedure receives a department number as an input parameter. A WHILE statement in the procedure body fetches the salary and bonus for each employee in the department. An IF statement within the WHILE statement updates salaries for each employee depending on number of years of service and current salary. When all employee records in the department have been processed, the FETCH statement that retrieves employee records receives SQLSTATE 20000. A not_found condition handler makes the

search condition for the WHILE statement false, so execution of the WHILE statement ends.

```
CREATE PROCEDURE BUMP_SALARY_IF (IN deptnumber SMALLINT)
  LANGUAGE SQL
  BEGIN
    DECLARE v_salary DOUBLE;
    DECLARE v_years SMALLINT;
    DECLARE v_id SMALLINT;
    DECLARE at_end INT DEFAULT 0;
    DECLARE not_found CONDITION FOR SQLSTATE '02000';

    -- CAST salary as DOUBLE because SQL procedures do not support DECIMAL
    DECLARE C1 CURSOR FOR
      SELECT id, CAST(salary AS DOUBLE), years
      FROM staff;
    DECLARE CONTINUE HANDLER FOR not_found
      SET at_end = 1;
```

```
OPEN C1;
        FETCH C1 INTO v_id, v_salary, v_years;
        WHILE at_end = 0 DO
          IF (v_salary < 2000 * v_years)
            THEN UPDATE staff
              SET salary = 2150 * v_years
              WHERE id = v_id;
            ELSEIF (v_salary < 5000 * v_years)
              THEN IF (v_salary < 3000 * v_years)
                THEN UPDATE staff
                  SET salary = 3000 * v_years
                  WHERE id = v_id;
              ELSE UPDATE staff
                SET salary = v_salary * 1.10
                WHERE id = v_id;
              END IF;
            ELSE UPDATE staff
              SET job = 'PREZ'
              WHERE id = v_id;
          END IF;
          FETCH C1 INTO v_id, v_salary, v_years;
        END WHILE;
        CLOSE C1;
      END
```