Chap 9. More SQL: Assertions, Views, and Programming Techniques

Specifying General Constraints as Assertions

CREATE ASSERTION constraint_name

CHECK <condition> : <condition> must be true at all time

- 직원의 월급은 소속 부서장의 월급보다 적어야 한다

```
CREATE ASSERTION  SALARY_CONSTRAINT
CHECK (NOT EXISTS (SELECT * FROM EMPLOYEE E,
                                EMPLOYEE M,
                                DEPARTMENT D
                   WHERE E.SALARY>M.SALARY and
                         E.DNO=D.DNUMBER and
                         D.MGRSSN=M.SSN));
```

Specifying General Constraints as Assertions

```
CREATE DOMAIN D_NUM AS INTEGER
CHECK (D_NUM>0 AND D_NUM<21);
```

Specifying General Constraints as Assertions

- Trigger
  - 직원의 월급보다 적은 월급을 받는 부서장 찾아 전달함

DEFINE TRIGGER trigger_name ON table_names;
<condition>
ACTION_PROCEDURE procedure_name;

VIEWS

- View (virtual table)
  - derived from other tables
  - not stored in DB physically
  - 자주 사용하는 정보
  - 보안
  - 최근 정보 접근 가능

CREATE VIEW view_name
AS SELECT 문

Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).

VIEWS

- V1

```
CREATE VIEW WORKS_ON1
AS SELECT   FNAME, LNAME, PNAME, HOURS
FROM        EMPLOYEE, PROJECT, WORKS_ON
WHERE       SSN=ESSN AND PNO=PNUMBER;
```

- V2

```
CREATE VIEW DEPT_INFO(DEPT_NAME, NO_OF_EMPS,
                      TOTAL_SAL)
AS SELECT   DNAME, COUNT(*), SUM(SALARY)
FROM        DEPARTMENT, EMPLOYEE
WHERE       DNUMBER=DNO
GROUP BY    DNAME;
```

WORKS_ON1

FNAME	LNAME	PNAME	HOURS

DEPT_INFO

DEPT_NAME	NO_OF_EMPS	TOTAL_SAL

Figure 8.5 Two views specified on the database schema of Figure 7.5.

VIEWS

- View에 대한 질의 (base table과 동일)

  QV1

  - SELECT PNAME, FNAME, LNAME
  - FROM WORKS_ON1
  - WHERE PNAME='ProjectX';

- VIEW 삭제

  V1A

  - DROP VIEW WORKS_ON1;

View Implementation and Update

- View Update
  - multiple interpretation on base tables
  - view with a single base table without aggregate function: updatable
  - view with multiple base tables is not updatable
- View Implementation
  - query modification
    - view query => query on base tables (transformation)
  - view materialization
    - create temporary view table on the first query to view

Specifying General Constraints as Assertions

- INDEX
  - physical access structure
  - indexing attributes : 접근 속도 향상

```
CREATE [UNIQUE] INDEX LNAME_INDEX
        ON EMPLOYEE (LNAME);
        [CLUSTER]
```

  - unique : candidate key
  - clustering and unique : primary key
  - clustering but not unique – inverted list – nonclustering – secondary index
  -

```
DROP INDEX index_name;
```

Database Programming

- Approaches to Database Programming
  - Embedded commands
    - SQL are embedded in a general purpose programming language
  - Library of database functions
    - available to the host language for database calls (known as an API)
  - Designing a brand new, full-fledged language
    - with additional programming structures
    - minimizes impedance mismatch

Typical Sequence of Interaction in Database Programming

1) Client program opens a connection to the database server
   - Connection

     CONNECT TO <server-name> AS <connection-name>
     AUTHORIZATION <user-account-info>;
   - Change from an active connection to another one

     SET CONNECTION <connection-name>;

2) Client program submits queries to and/or updates the database

3) When database access is no longer needed, client program terminates the connection
   - Disconnection

     DISCONNECT <connection-name>;

Embedded SQL

- Most SQL statements can be embedded in a general purpose host programming language

- EXEC SQL and END-EXEC (or semicolon)
  - Distinguished from the host language statements

- SQLCODE/SQLSTATE
  - Communication between program and SQL
  - 0 & 00000 : SQL is successfully executed
  - Other codes : error
  - SQLSTATE is the variable in later versions of the SQL standard

Embedded SQL

- Shared Variables
  - Variables inside **DECLARE** are used in both languages
  - Usually prefixed with a colon in SQL
- Retrieving Single Tuples with Embedded SQL

```
//Program Segment E1:
0)   loop = 1 ;
1)   while (loop) {
2)     prompt("Enter a Social Security Number: ", ssn) ;
3)     EXEC SQL
4)       select FNAME, MINIT, LNAME, ADDRESS, SALARY
5)       into :fname, :minit, :lname, :address, :salary
6)       from EMPLOYEE where SSN = :ssn ;
7)     if (SQLCODE == 0) printf(fname, minit, lname, address, salary)
8)       else printf("Social Security Number does not exist: ", ssn) ;
9)     prompt("More Social Security Numbers (enter 1 for Yes, 0 for No): ", loop) ;
10)    }
```

FIGURE 9.3 Program segment E1, a C program segment with embedded SQL.

Embedded SQL

- Retrieving Multiple Tuples with Embedded SQL
  - CURSOR
    - a pointer that points to a single row
    - **OPEN CURSOR**
      - set cursor to a position *before the first row* in the result of a query
    - **CLOSE CURSOR**
      - indicates that the processing of query results has been completed
  - FETCH
    - copy the first row to program variables
    - move cursor to next row
    - return END_OF_CURSOR : last tuple

Embedded SQL

- Retrieving Multiple Tuples with Embedded SQL

```
//Program Segment E2:
0)   prompt("Enter the Department Name: ", dname) ;
1)   EXEC SQL
2)      select DNUMBER into :dnumber
3)      from DEPARTMENT where DNAME = :dname ;
4)   EXEC SQL DECLARE EMP CURSOR FOR
5)      select SSN, FNAME, MINIT, LNAME, SALARY
6)      from EMPLOYEE where DNO = :dnumber
7)      FOR UPDATE OF SALARY ;
8)   EXEC SQL OPEN EMP ;
9)   EXEC SQL FETCH from EMP into :ssn, :fname, :minit, :lname, :salary ;
10)  while (SQLCODE == 0) {
11)     printf("Employee name is:", fname, minit, lname)
12)     prompt("Enter the raise amount: ", raise) ;
13)     EXEC SQL
14)        update EMPLOYEE
15)        set SALARY = SALARY + :raise
16)        where CURRENT OF EMP ;
17)     EXEC SQL FETCH from EMP into :ssn, :fname, :minit, :lname, :salary ;
18)     }
19)  EXEC SQL CLOSE EMP ;
```

FIGURE 9.4 Program segment E2, a C program segment that uses cursors with embedded SQL for update purposes.

SQLJ

- Embedding SQL Commands in JAVA
  - SQLJ: a standard for embedding SQL in Java
  - An SQLJ translator converts SQL statements into Java
  - Using certain classes in **java.sql**
- Retrieving Multiple Tuples in SQLJ
  - SQLJ supports two types of iterators (like CURSOR)
    - named iterator: associated with a query result
    - positional iterator: lists only attribute types in a query result
  - FETCH operation retrieves the next tuple in a query result

    fetch <iterator-variable> into <program-variables>

Dynamic SQL

- Specifying Queries at Runtime Using Dynamic SQL
  - Objective : executing new (not previously compiled) SQL statements
  - A program accepts SQL statements from the keyboard at runtime
  - Dynamic query can be complex
    - the type and number of retrieved attributes are unknown at compile time
  - PREPARE : convert and generate a query
  - EXECUTE : execute a query
  - EXECUTE IMMEDIATE : PREPARE + EXECUTE

Dynamic SQL

- Dynamic SQL for updating a table

```
//Program Segment E3:
0)   EXEC SQL BEGIN DECLARE SECTION ;
1)   varchar sqlupdatestring [256] ;
2)   EXEC SQL END DECLARE SECTION ;
     ...
3)   prompt("Enter the Update Command: ", sqlupdatestring) ;
4)   EXEC SQL PREPARE sqlcommand FROM :sqlupdatestring ;
5)   EXEC SQL EXECUTE sqlcommand ;
     ...
```

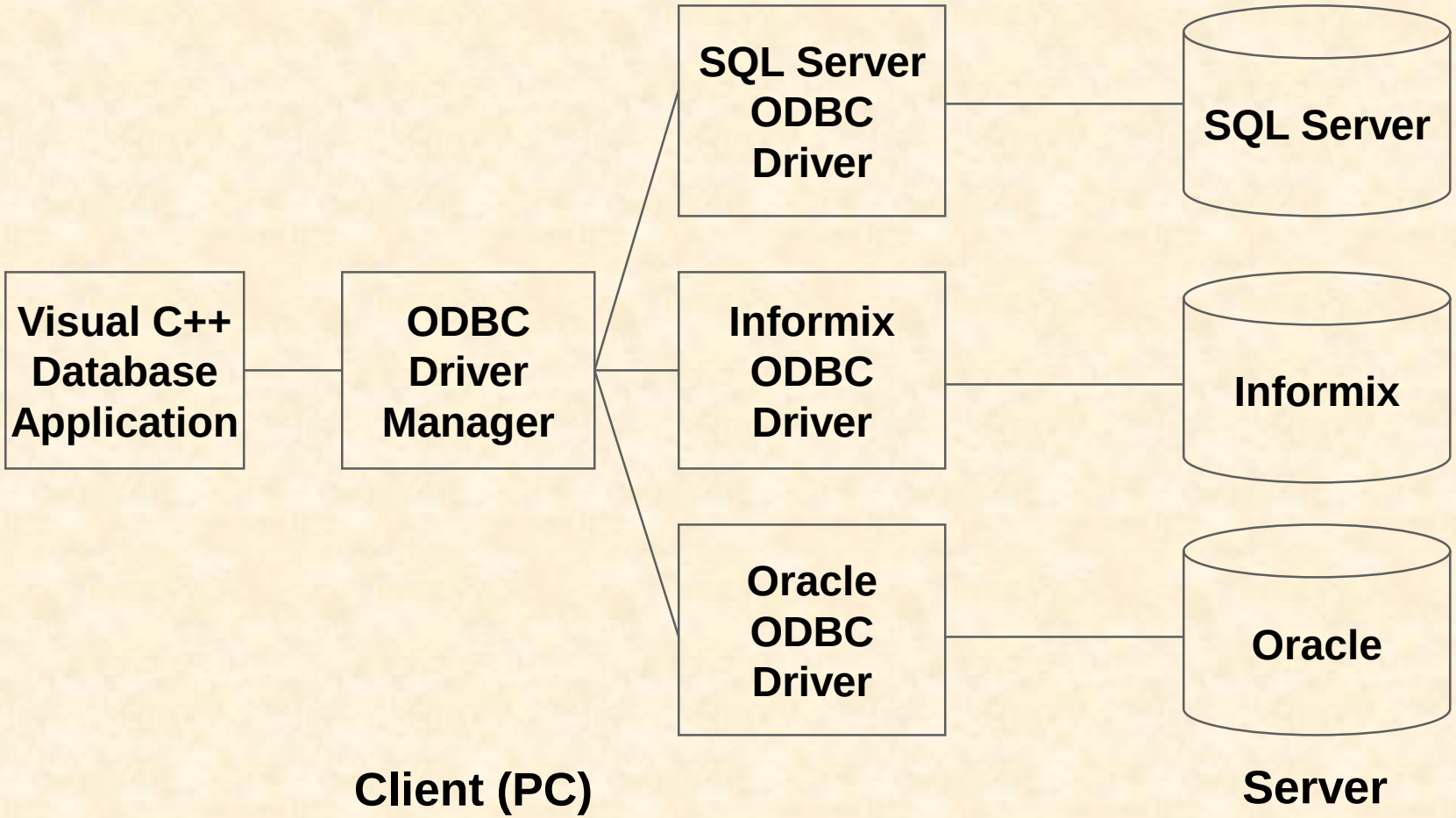
FIGURE 9.5 Program segment E3, a C program segment that uses dynamic SQL for updating a table.

Database Programming with Function Call (API)

- Components
  - Application : ODBC function들을 호출함
  - Driver Manager :
    - Load and unload drivers
    - ODBC function call을 처리 혹은 Driver에게 넘김
  - Driver :
    - ODBC function call 처리
    - SQL request를 data source에게 넘김.
    - 결과를 AP 에게 되돌림.
  - Data source :
    - user가 access하고자 하는 data와 data와 관련한 OS, DBMS, network platform들로 구성.

Database Programming with Function Call (API)

- ODBC Architecture

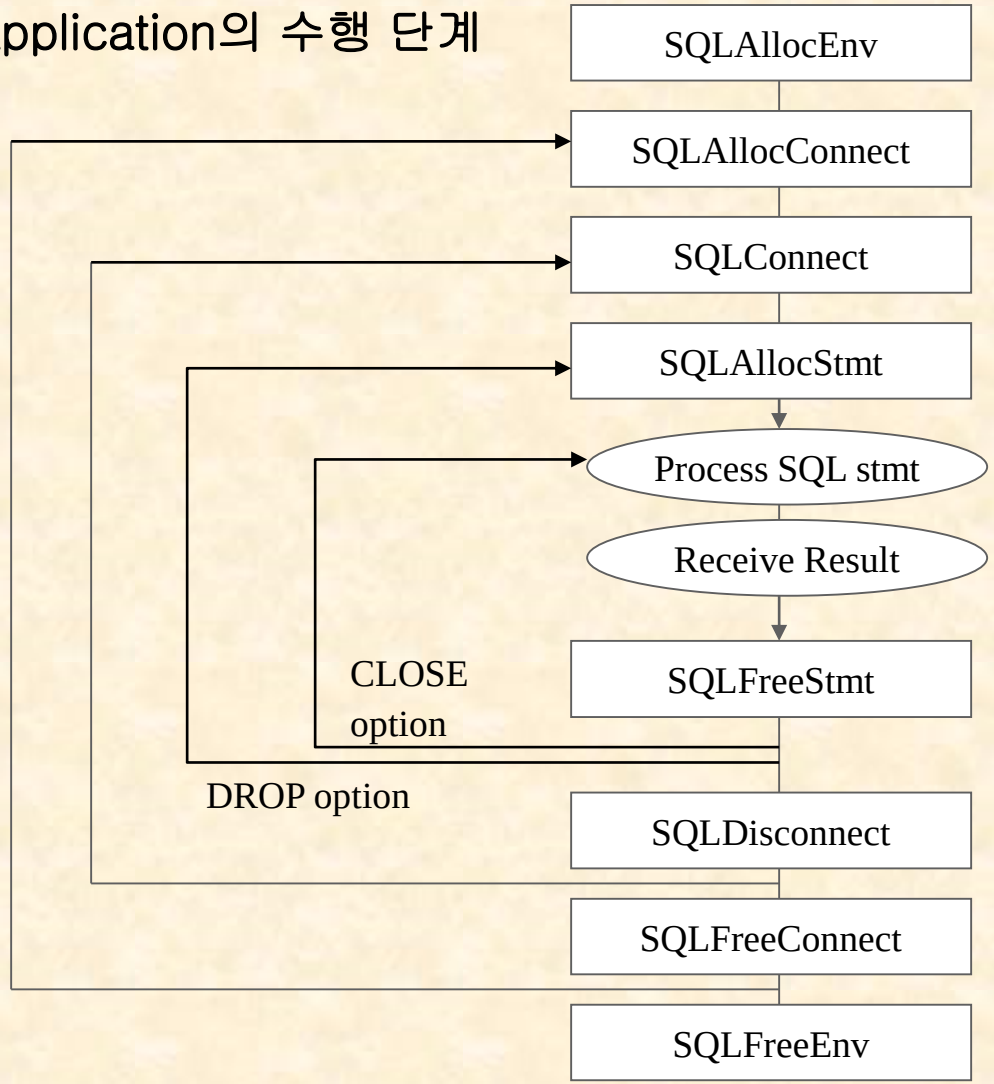


Database Programming with Function Call (API)

- Handle
  - 특정 item을 식별하기 위한 값
  - Four types
    1. Environment handle : 유효 연결 핸들이나 현재 가동 중인 연결 핸들 등, 광역 정보를 위한 기억장소를 식별
    2. Connection handle : 특정 연결에 관한 정보를 위한 기억장소를 식별
    3. Statement handle : SQL문에 관한 정보를 위한 기억장소를 식별
    4. Descriptor handle : version 3.0
       - SQL statement에 대한 여러 정보를 위한 기억장소를 식별
       - 자동으로 또는 명시적으로 할당

Database Programming with Function Call (API)

Application의 수행 단계



환경핸들을 위한 메모리 할당

연결핸들을 위한 메모리 할당

Driver를 로드하고 데이타소스와 연결

문장핸들을 위한 메모리를 할당

Database Programming with Function Call (API)

- JDBC (Java Database Connectivity)
  - SQL Function Calls for JAVA Programming
    - CLI와는 구문적(syntactic)인 차이 (not semantic)
  - allows a program to connect to several databases
  - JDBC Driver
    - An implementation of the function calls specified in the JDBC API for a particular vendor's RDBMS

Database Programming with Function Call (API)

- SQL/CLI (Call Level Interface)
  - A part of the SQL standard
  - Provides easy access to several databases within the same program
  - Certain libraries have to be installed and available
    - e.g., `sqlcli.h` for C
  - SQL statements in the calls
    - dynamically created
    - passed as string parameters

Database Programming with Function Call (API)

- Components of SQL/CLI
  - *Environment record*: keeps track of database connections
  - *Connection record*: keep tracks of info needed for a particular connection
  - *Statement record*: keeps track of info needed for one SQL statement
  - *Description record*: keeps track of tuples

Database Programming with Function Call (API)

- Steps in C and SQL/CLI Programming

1. Load SQL/CLI libraries
2. Declare record handle variables for the above components (called: **SQLHSTMT**, **SQLHDBC**, **SQLHENV**, **SQLHDEC**)
3. Set up an environment record using **SQLAllocHandle**
4. Set up a connection record using **SQLAllocHandle**
5. Set up a statement record using **SQLAllocHandle**

Database Programming with Function Call (API)

- Steps in C and SQL/CLI Programming

6. Prepare a statement using SQL/CLI function **`SQLPrepare`**
7. Bound parameters to program variables
8. Execute SQL statement via **`SQLExecute`**
9. Bound columns in a query to a C variable via **`SQLBindCol`**
10. Use **`SQLFetch`** to retrieve column values into C variables

Database Programming with Function Call (API)

- a C program segment with SQL/CLI.

```
//Program CLI1:
0)   #include sqlcli.h ;
1)   void printSal() {
2)   SQLHSTMT stmt1 ;
3)   SQLHDBC con1 ;
4)   SQLHENV env1 ;
5)   SQLRETURN ret1, ret2, ret3, ret4 ;
6)   ret1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env1) ;
7)   if (!ret1) ret2 = SQLAllocHandle(SQL_HANDLE_DBC, env1, &con1) else exit ;
8)   if (!ret2) ret3 = SQLConnect(con1, "dbs", SQL_NTS, "js", SQL_NTS, "xyz", SQL_NTS)
else exit ;
9)   if (!ret3) ret4 = SQLAllocHandle(SQL_HANDLE_STMT, con1, &stmt1) else exit ;
10)  SQLPrepare(stmt1, "select LNAME, SALARY from EMPLOYEE where SSN = ?", SQL_NTS) ;
11)  prompt("Enter a Social Security Number: ", ssn) ;
12)  SQLBindParameter(stmt1, 1, SQL_CHAR, &ssn, 9, &fetchlen1) ;
13)  ret1 = SQLExecute(stmt1) ;
14)  if (!ret1) {
15)      SQLBindCol(stmt1, 1, SQL_CHAR, &lname, 15, &fetchlen1) ;
16)      SQLBindCol(stmt1, 2, SQL_FLOAT, &salary, 4, &fetchlen2) ;
17)      ret2 = SQLFetch(stmt1) ;
18)      if (!ret2) printf(ssn, lname, salary)
19)          else printf("Social Security Number does not exist: ", ssn) ;
20)      }
21)  }
```

Database Programming with Function Call (API)

- a C program segment that uses SQL/CLI for a query with a collection of tuples in its result.

```
//Program Segment CLI2:
0)    #include sqlcli.h ;
1)    void printDepartmentEmps() {
2)    SQLHSTMT stmt1 ;
3)    SQLHDBC con1 ;
4)    SQLHENV env1 ;
5)    SQLRETURN ret1, ret2, ret3, ret4 ;
6)    ret1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env1) ;
7)    if (!ret1) ret2 = SQLAllocHandle(SQL_HANDLE_DBC, env1, &con1) else exit ;
8)    if (!ret2) ret3 = SQLConnect(con1, "dbs", SQL_NTS, "js", SQL_NTS, "xyz", SQL_NTS)
else exit ;
9)    if (!ret3) ret4 = SQLAllocHandle(SQL_HANDLE_STMT, con1, &stmt1) else exit ;
10)   SQLPrepare(stmt1, "select LNAME, SALARY from EMPLOYEE where DNO = ?", SQL_NTS) ;
11)   prompt("Enter the Department Number: ", dno) ;
12)   SQLBindParameter(stmt1, 1, SQL_INTEGER, &dno, 4, &fetchlen1) ;
13)   ret1 = SQLExecute(stmt1) ;
14)   if (!ret1) {
15)       SQLBindCol(stmt1, 1, SQL_CHAR, &lname, 15, &fetchlen1) ;
16)       SQLBindCol(stmt1, 2, SQL_FLOAT, &salary, 4, &fetchlen2) ;
17)       ret2 = SQLFetch(stmt1) ;
18)       while (!ret2) {
19)           printf(lname, salary) ;
20)           ret2 = SQLFetch(stmt1) ;
21)           }
22)       }
23)   }
```

Database Stored Procedures

- Stored Procedures and Functions
  - Stored procedures is stored locally and executed by the database server
- Advantages
  - if the program is needed by many applications, it can be invoked by any of them
    - reduces duplication and improves modularity
  - reduces communication costs
  - enhance the modeling power of views

Stored Procedure Constructs

- A stored procedure

  CREATE PROCEDURE <procedure-name> (<params>)

  <local-declarations>

  <procedure-body>;

- A stored function

  CREATE FUNCTION <function-name> (<params>)

  RETRUNS <return-type>

  <local-declarations>

  <function-body>;

- Calling a procedure or function

  CALL <procedure/function-name> (<arguments>);

SQL/PSM

- SQL/PSM (Persistent Stored Modules)
  - Part of the SQL standard for writing persistent stored modules
  - SQL + stored procedures/functions + additional programming constructs
    - branching and looping statements
    - enhance the power of SQL