



5th Edition

Elmasri / Navathe

Chapter 9

Introduction to SQL Programming Techniques



5th Edition

Elmasri / Navathe

Chapter Outline

- 9.1 General Constraints as Assertions
- 9.2 Views in SQL
- 9.3 Database Programming
- 9.4 Embedded SQL
- 9.5 Functions Calls, SQL/CLI
- 9.6 Stored Procedures, SQL/PSM
- 9.7 Summary

Chapter Objectives

- Specification of more general **constraints** via assertions
- SQL facilities for defining views (virtual tables)
- Various techniques for accessing and manipulating a database via programs in general-purpose languages
 - E.g., Java, C++, etc.

Constraints as Assertions

- General constraints: constraints that do not fit in the basic SQL categories (presented in chapter 8)
- Mechanism: **CREATE ASSERTION**
 - Components include:
 - a constraint name,
 - followed by `CHECK`,
 - followed by a condition

Assertions: An Example

- “The salary of an employee must not be greater than the salary of the manager of the department that the employee works for”

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

constraint
name,
CHECK,
condition

Using General Assertions

- Specify a query that violates the condition; include inside a `NOT EXISTS` clause
- Query result must be empty
 - if the query result is not empty, the assertion has been violated

SQL Triggers

- Objective: to monitor a database and take initiate action when a condition occurs
- Triggers are expressed in a syntax similar to assertions and include the following:
 - **Event**
 - Such as an insert, deleted, or update operation
 - **Condition**
 - **Action**
 - To be taken when the condition is satisfied

SQL Triggers: An Example

- A trigger to compare an employee's salary to his/her supervisor during insert or update operations:

```
CREATE TRIGGER INFORM_SUPERVISOR
BEFORE INSERT OR UPDATE OF
  SALARY, SUPERVISOR_SSN ON EMPLOYEE
FOR EACH ROW
  WHEN
    (NEW.SALARY > (SELECT SALARY FROM EMPLOYEE
                  WHERE SSN=NEW.SUPERVISOR_SSN) )
  INFORM_SUPERVISOR (NEW.SUPERVISOR_SSN, NEW.SSN) ;
```

Views in SQL

- A view is a “virtual” table that is derived from other tables
- Allows for limited update operations
 - Since the table may not physically be stored
- Allows full query operations
- A convenience for expressing certain operations

Specification of Views

- SQL command: **CREATE VIEW**
 - a table (view) name
 - a possible list of attribute names (for example, when arithmetic operations are specified or when we want the names to be different from the attributes in the base relations)
 - a query to specify the table contents

SQL Views: An Example

- Specify a different WORKS_ON table

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

Using a Virtual Table

- We can specify SQL queries on a newly create table (view):

```
SELECT FNAME, LNAME  
FROM WORKS_ON_NEW  
WHERE PNAME= 'Seena' ;
```

- When no longer needed, a view can be dropped:

```
DROP WORKS_ON_NEW;
```

Efficient View Implementation

- Query modification:
 - Present the view query in terms of a query on the underlying base tables
- Disadvantage:
 - Inefficient for views defined via complex queries
 - Especially if additional queries are to be applied to the view within a short time period

Efficient View Implementation

- View materialization:
 - Involves physically creating and keeping a temporary table
- Assumption:
 - Other queries on the view will follow
- Concerns:
 - Maintaining correspondence between the base table and the view when the base table is updated
- Strategy:
 - Incremental update

Update Views

- Update on a single view without aggregate operations:
 - Update may map to an update on the underlying base table
- Views involving joins:
 - An update *may* map to an update on the underlying base relations
 - Not always possible

Un-updatable Views

- Views defined using groups and aggregate functions are not updateable
- Views defined on multiple tables using joins are generally not updateable
- **WITH CHECK OPTION:** must be added to the definition of a view if the view is to be updated
 - To allow check for updatability and to plan for an execution strategy

Database Programming

- Objective:
 - To access a database from an application program (as opposed to interactive interfaces)
- Why?
 - An interactive interface is convenient but not sufficient
 - A majority of database operations are made thru application programs (increasingly thru web applications)

Database Programming Approaches

- Embedded commands:
 - Database commands are embedded in a general-purpose programming language
- Library of database functions:
 - Available to the host language for database calls; known as an *API*
 - *API* standards for Application Program Interface
- A brand new, full-fledged language
 - Minimizes impedance mismatch

Impedance Mismatch

- Incompatibilities between a host programming language and the database model, e.g.,
 - type mismatch and incompatibilities; requires a new binding for each language
 - set vs. record-at-a-time processing
 - need special iterators to loop over query results and manipulate individual values

Steps in Database Programming

1. Client program *opens a connection* to the database server
2. Client program *submits queries to and/or updates* the database
3. When database access is no longer needed, client program *closes (terminates) the connection*

Embedded SQL

- Most SQL statements can be embedded in a general-purpose *host* programming language such as COBOL, C, Java
- An embedded SQL statement is distinguished from the host language statements by enclosing it between **EXEC SQL** or **EXEC SQL BEGIN** and a matching **END-EXEC** or **EXEC SQL END** (or semicolon)
 - Syntax may vary with language
 - *Shared variables* (used in both languages) usually prefixed with a colon (:) in SQL

Example: Variable Declaration in Language C

- Variables inside **DECLARE** are shared and can appear (while prefixed by a colon) in SQL statements
- **SQLCODE** is used to communicate errors/exceptions between the database and the program

```
int loop;
```

```
EXEC SQL BEGIN DECLARE SECTION;
```

```
    varchar dname[16], fname[16], ...;
```

```
    char ssn[10], bdate[11], ...;
```

```
    int dno, dnumber, SQLCODE, ...;
```

```
EXEC SQL END DECLARE SECTION;
```

SQL Commands for Connecting to a Database

- Connection (multiple connections are possible but only one is active)

```
CONNECT TO server-name AS connection-name  
AUTHORIZATION user-account-info;
```

- Change from an active connection to another one

```
SET CONNECTION connection-name;
```

- Disconnection

```
DISCONNECT connection-name;
```

Embedded SQL in C

Programming Examples

```
loop = 1;
while (loop) {
    prompt ("Enter SSN: ", ssn);
    EXEC SQL
        select FNAME, LNAME, ADDRESS, SALARY
        into :fname, :lname, :address, :salary
        from EMPLOYEE where SSN == :ssn;
    if (SQLCODE == 0) printf(fname, ...);
    else printf("SSN does not exist: ", ssn);
    prompt("More SSN? (1=yes, 0=no): ", loop);
    END-EXEC
}
```

Embedded SQL in C

Programming Examples

- A **cursor** (iterator) is needed to process multiple tuples
- **FETCH** commands move the cursor to the *next* tuple
- **CLOSE CURSOR** indicates that the processing of query results has been completed

Dynamic SQL

- Objective:
 - Composing and executing new (not previously compiled) SQL statements at run-time
 - a program accepts SQL statements from the keyboard at run-time
 - a point-and-click operation translates to certain SQL query
- Dynamic update is relatively simple; dynamic query can be complex
 - because the type and number of retrieved attributes are unknown at compile time

Dynamic SQL: An Example

```
EXEC SQL BEGIN DECLARE SECTION;  
varchar sqlupdatestring[256];  
EXEC SQL END DECLARE SECTION;  
...  
prompt ("Enter update command:", sqlupdatestring);  
EXEC SQL PREPARE sqlcommand FROM :sqlupdatestring;  
EXEC SQL EXECUTE sqlcommand;
```

Embedded SQL in Java

- SQLJ: a standard for embedding SQL in Java
- An SQLJ translator converts SQL statements into Java
 - These are executed thru the *JDBC* interface
- Certain classes have to be imported
 - E.g., `java.sql`

Java Database Connectivity

- JDBC:
 - SQL connection function calls for Java programming
- A Java program with JDBC functions can access any relational DBMS that has a JDBC driver
- JDBC allows a program to connect to several databases (known as *data sources*)

Steps in JDBC Database Access

1. Import JDBC library (`java.sql.*`)
2. Load JDBC driver:
`Class.forName("oracle.jdbc.driver.OracleDriver")`
3. Define appropriate variables
4. Create a connect object (via `getConnection`)
5. Create a statement object from the `Statement` class:
 - 1. `PreparedStatement` 2. `CallableStatement`
6. Identify statement parameters (designated by question marks)
7. Bound parameters to program variables
8. Execute SQL statement (referenced by an object) via JDBC's `executeQuery`
9. Process query results (returned in an object of type `ResultSet`)
 - `ResultSet` is a 2-dimensional table

Embedded SQL in Java: An Example

```
ssn = readEntry("Enter a SSN: ");
try {
    #sql{select FNAME< LNAME, ADDRESS, SALARY
    into :fname, :lname, :address, :salary
    from EMPLOYEE where SSN = :ssn};
}
catch (SQLException se) {
    System.out.println("SSN does not exist: ",+ssn);
    return;
}
System.out.println(fname + " " + lname + ... );
```

Multiple Tuples in SQLJ

- SQLJ supports two types of iterators:
 - *named iterator*: associated with a query result
 - *positional iterator*: lists only attribute types in a query result
- A **FETCH** operation retrieves the next tuple in a query result:
`fetch iterator-variable into program-variable`

Database Programming with Functional Calls

- Embedded SQL provides static database programming
- API: Dynamic database programming with a library of functions
 - **Advantage:**
 - No preprocessor needed (thus more flexible)
 - **Disadvantage:**
 - SQL syntax checks to be done at run-time

SQL Call Level Interface

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

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

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

Database Stored Procedures

- Persistent procedures/functions (modules) are stored locally and executed by the database server
 - As opposed to execution by clients
- Advantages:
 - If the procedure is needed by many applications, it can be invoked by any of them (thus reduce duplications)
 - Execution by the server reduces communication costs
 - Enhance the modeling power of views
- Disadvantages:
 - Every DBMS has its own syntax and this can make the system less portable

Stored Procedure Constructs

- A stored procedure

```
CREATE PROCEDURE procedure-name (params)  
local-declarations  
procedure-body;
```

- A stored function

```
CREATE FUNCTION fun-name (params) RETRUNS return-  
type  
local-declarations  
function-body;
```

- Calling a procedure or function

```
CALL procedure-name/fun-name (arguments);
```

SQL Persistent Stored Modules

- SQL/PSM:
 - Part of the SQL standard for writing persistent stored modules
- SQL + stored procedures/functions + additional programming constructs
 - E.g., branching and looping statements
 - Enhance the power of SQL

SQL/PSM: An Example

```
CREATE FUNCTION DEPT_SIZE (IN deptno INTEGER)
RETURNS VARCHAR[7]
DECLARE TOT_EMPS INTEGER;

SELECT COUNT (*) INTO TOT_EMPS
      FROM SELECT EMPLOYEE WHERE DNO = deptno;
IF TOT_EMPS > 100 THEN RETURN "HUGE"
  ELSEIF TOT_EMPS > 50 THEN RETURN "LARGE"
  ELSEIF TOT_EMPS > 30 THEN RETURN "MEDIUM"
  ELSE RETURN "SMALL"
ENDIF;
```

Summary

- Assertions provide a means to specify additional constraints
- Triggers are assertions that define actions to be automatically taken when certain conditions occur
- Views create temporary (virtual) tables
- A database may be accessed in an interactive mode
- Most often, however, data in a database is manipulated via application programs
- Several methods of database programming:
 - Embedded SQL
 - Dynamic SQL
 - Stored procedure and function