Chapter 5: Advanced SQL

Database System Concepts, 6th Ed.

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use

Chapter 5: Advanced SQL

- Accessing SQL From a Programming Language
  - JDBC and ODBC
- Functions and Procedural Constructs
- Triggers

JDBC and ODBC

- API (application-program interface) for a program to interact with a database server
- Application makes calls to
  - Connect with the database server
  - Send SQL commands to the database server
  - Fetch tuples of result one-by-one into program variables
- ODBC (Open Database Connectivity) works with C, C++, C#, and Visual Basic
  - Other API's such as ADO.NET sit on top of ODBC
- JDBC (Java Database Connectivity) works with Java

JDBC

- **JDBC** is a Java API for communicating with database systems supporting SQL.
- JDBC supports a variety of features for querying and updating data, and for retrieving query results.
- Model for communicating with the database:
  - Open a connection
  - Create a “statement” object
  - Execute queries using the Statement object and fetch results
  - Exception mechanism to handle errors

JDBC Code

```
public static void JDBCexample(String dbid, String userid, String passwd)
{
    try {
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        Connection conn = DriverManager.getConnection(
              "jdbc:oracle:thin:@db.yale.edu:2000:univdb", userid, passwd);
        Statement stmt = conn.createStatement();
           … Do Actual Work ….
        stmt.close();
        conn.close();
    }
    catch (SQLException sqle) {
        System.out.println("SQLException : " + sqle);
    }
}
```

JDBC Code (Cont.)

- Update to database

```
try {
    stmt.executeUpdate(
        "insert into instructor values('77987', 'Kim', 'Physics', 98000)");
} catch (SQLException sqle)
{
   System.out.println("Could not insert tuple. " + sqle);
}
```

- Execute query and fetch and print results

```
ResultSet rset = stmt.executeQuery(
                        "select dept_name, avg (salary)
                         from instructor
                         group by dept_name");
while (rset.next()) {
    System.out.println(rset.getString("dept_name") + " " +
                                rset.getFloat(2));
}
```

- Getting result fields:
  - rs.getString("dept_name") and rs.getString(1) equivalent if dept_name is the first argument of select result.

Procedural Extensions and Stored Procedures

- SQL provides a **module** language
  - Permits definition of procedures in SQL, with if-then-else statements, for and while loops, etc.
- Stored Procedures
  - Can store procedures in the database
  - then execute them using the **call** statement
  - permit external applications to operate on the database without knowing about internal details

Functions and Procedures

- SQL:1999 supports functions and procedures
  - Functions/procedures can be written in SQL itself, or in an external programming language.
- SQL:1999 also supports a rich set of imperative constructs, including
  - Loops, if-then-else, assignment
- Many databases have proprietary procedural extensions to SQL that differ from SQL:1999.

SQL Functions

- Define a function that, given the name of a department, returns the count of the number of instructors in that department.

```
create function dept_count (dept_name varchar(20))
returns integer
begin
    declare d_count integer;
    select count (* ) into d_count
    from instructor
    where instructor.dept_name = dept_name
    return d_count;
end
```

- Find the department name and budget of all departments with more that 12 instructors.

```
select dept_name, budget
from department
where dept_count (dept_name ) > 1
```

SQL Procedures

- The *dept_count* function could instead be written as procedure:

```
create procedure dept_count_proc (in dept_name varchar(20),
                                  out d_count integer)
begin
  select count(*) into d_count
  from instructor
  where instructor.dept_name = dept_count_proc.dept_name
end
```

- Procedures can be invoked either from an SQL procedure or from embedded SQL, using the **call** statement.

```
declare d_count integer;
call dept_count_proc( 'Physics', d_count);
```

Triggers

- A **trigger** is a statement that is executed automatically by the system as a side effect of a modification to the database.
- To design a trigger mechanism, we must:
  - Specify the **conditions** under which the trigger is to be executed.
  - Specify the **actions** to be taken when the trigger executes.

Trigger Example

- E.g., enforce foreign key constraint from *section* to *timeslot* although *time_slot_id* is not a primary key of *timeslot.*

```
create trigger timeslot_check1 after insert on section
referencing new row as nrow for each row

    /* nrow  iterates over the post-update version of each updated row */
when (nrow.time_slot_id not in (
            select time_slot_id
            from time_slot)) /* time_slot_id not present in time_slot */
begin
    rollback
end;
```

Chapter 5: Advanced SQL

- Accessing SQL From a Programming Language
  - JDBC and ODBC
- Functions and Procedural Constructs
- Triggers

End of Chapter

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use