09. Programming Databases

Different facets of accessing databases in a program

Following are different types of programs that manipulate databases. Each type of programs has their own place and suitability.

- SQL queries
  - SQL queries are written as "declarative relation manipulation expressions" in SQL
  - Submitted from the client (a console, GUI client, or a program)
  - Executed on DBMS
  - SQL is not enough to perform complex data manipulations. Therefore, we often do some programming on top of SQL.
  - Following are such options.

- Stored Procedures
  - A Procedure (or function) sitting as database element
  - Has access to all database elements like tables, views, etc.
  - Run on DBMS server therefore no movement of data is required from database to the program, typically over network.
  - More coming ahead!

Triggers:

- Triggers are special types of Stored Procedures that are invoked automatically on event of a database update operation.
- Triggers can be defined to execute either before or after any INSERT, UPDATE, or DELETE operation.
- If a trigger event occurs, the trigger's function is called.

Types of Triggers

- Trigger could be **row-level** trigger or **statement-level** trigger
- A *row level* trigger is invoked once for each row that is affected by the statement that fired the trigger. In contrast, *statement-level* trigger is invoked only once when an appropriate statement is executed, regardless of the number of rows affected by that statement. A statement-level-trigger is executed even if it affects zero rows.
- Triggers are also classified as before triggers and after triggers.
- *Statement-level before* triggers naturally fire before the statement starts to do anything, while *statement-level after* triggers fire at the very end of the statement.
- *Row-level before* triggers fire immediately before a particular row is operated on, while *row-level after* triggers fire after working on the row (but before any statement-level after triggers).

- General programming language programs accessing databases through API
  - Most popular way of manipulating database.
  - Most application are written in Programming languages like Java, Python, C++, C#, PHP, or so on.
  - Programs written in these languages access (read/write) databases through standard programming interfaces JDBC, Python DB-API, MS-ADO, ODBC, etc.

- Embedded SQL: SQL is permitted in programming
    - Oracle's Pro*C, PostgreSQL's ECPG, and SQLJ are such environments

- Program accessing databases through Native Programming Interfaces:
    - Each DBMS typically provide a lower level access to databases – typically at physical level schema. This makes most efficient access of databases.
    - Programs written using standard API may typically have to be translated to this level calls by things like JDBC driver.
    - Oracle's OCI (Oracle Call Interface), PostgreSQL's (libpq) are such examples.

In any real application we write more than one type of programs. Considering following scenario of DA-IICT inviting applications for admission, you should be able to identify what functionality is implemented using what type of program or so-

- Student Applies: showing up blank application form - will require getting various inputs from databases.
- Applicant enters his/her details. On submission, various validation happens, may be by looking into databases.
- Finally, new tuples (for new applicant) are inserted into relevant relations.
- DA-IICT prepares its own merit list; a process updates few attributes of applications relation.
- DA-IICT allocates seats as per available sheet matrix (program wise, category wise, etc); updates again some fields of some relations.

Stored Procedures

Two terms here "stored" and "procedure" ? Or "Stored Functions"

"procedure": is a generic term used for "function" in a program. It has historic importance as many programming languages had function and procedures as separate constructs. With the time procedure, as a separate construct, has been phased out, and are not supported in most modern programming languages. A "cohesive" procedure basically turns out to be a function only.

"stored": procedures are stored in "database instance". They run in database instance context under the control of DBMS.

Why do we need stored Procedure? Here is an motivating example-

> Consider above scenario of DA-IICT inviting applications for B.Tech. admissions. Let us say we have a relation **Applications** as following-
> Applications (AppNo, Name, Marks_Phy, Marks_Chem, Marks_Math, DA_Rank)

Given this relation, if we want to allocate DA_Rank to each applicant, we cannot compute this merely by writing SQL queries. Below is some pseudo code typically performing required computation-

```
rank = 0
for each row in SELECT * FROM Applicants ORDER BY marks DESC
      rank = rank + 1
      UPDATE applicants set DA_RANK = rank
            WHERE application_no = row.application_no
end for
```

Above pseudo code can be implemented either as stored procedure or in a host programming language like java using appropriate access API.

Below is representative Java code for the same-

```
resultset = conn.execute("SELECT * FROM Applicants ORDER BY marks DESC");
rank = 0
while ( rs.next() ) {
      rank = rank + 1
      con.Execute("update applicants set da_rank = " + rank
            + " where application_no = " + resultset.application_no
   resultset.moveNext()
}
```

It is important to note here that java "host" program runs on the client, and require all tuples bringing to the client. This adds transportation cost, and lowers the program performance.

To appreciate the concern, let us understand what is movement of the data? Following representation with different highlights should help. Statements in red are executed on DBMS server. Their results are brought to client. We use variables (in blue) to store the results, and iteratively update related tuples.

```
//Let A be list of application numbers in the descending order of their marks
A = { SELECT applicant_no FROM Applicants ORDER BY marks DESC }
rank = 0
for a ∈ A
      rank = rank + 1
      UPDATE applicants set DA_RANK = rank WHERE application_no = a
end for
```

In certain applications like this, bringing of the data not for any use of the client, functionality can be better implemented in the form of "Stored Procedures" or "Stored Functions".

Stored Procedures are stored "program" as part of database instance, and run on the same computer where data are, avoiding data transportation and thus program performance.

Exercise: To make it more complex, let us say, we need to allocate Category Rank as well. Try figuring out appropriate solution for this.

Stored Procedures – applications and benefits

1. Programs runs in immediate vicinity of data, and does not require any data transportation over networks, and this speeds up data processing

2. Functionality that is "batch" in nature, that is, does not require any user intervention; need not to bring any data to the client. Examples
    a. Example above for allocating ranks to a large number of applicants is such a requirement. Once started, no intervention from user is needed.
    b. Computation of SPI and CPI at the end of semester; done once and as "batch" task.
    c. Run month-end database updates for an online Transaction Processing system
    d. Build various materialized views.
    e. Data import/export from other database sources

3. Helps in enabling various operational abstractions over data?

4. Stored procedure stores "application functionality" as part of database; this makes the functionality to be more shareable and reusable.

5. Database triggers are implemented as stored procedures

6. Complex constraints are also enforced using stored procedures.

Language features for Stored Procedures

- SQL is not enough?
- Stored procedures are written for requirements that cannot be expressed in expressive queries like SQL, and **require procedural constructs** like branching and iterations.
- Note that newer releases of SQL do support some branching expressive power in SQL queries (IF, SWITCH or so)
- ANSI added as SQL/PSM (SQL/Persistent Stored Module) as part of SQL extension with SQL-1999.
- RDBMS like Oracle had PL/SQL even before this standardization.
- Most RDBMS, today, provide their own procedural languages for creating stored procedures. For example: PL/SQL (Oracle), PL/PgSQL (postgresql), SQL/PL (DB2), TSQL(MS SQL)

Stored Procedures in PostgreSQL

- PostgreSQL provides an open architecture for creating stored procedures.
- It allows you creating stored procedures in many languages like C, PL/PgSQL, PL/perl, PL/Python, PL/Tcl, and PL/Java.
- PostgreSQL may require you to load the language before programming in that language.
- It also allows you to write simple stored procedures in SQL.
- PostgreSQL calls stored procedures as "stored functions".

For simplicity, we may be using term "function" for "stored functions" in further discussion.

Learning Stored Procedure Language (SPL)

To repeat: Stored Procedure Language (SPL) is "procedural extension" to SQL.

Let us see what those "extensions"? On the way you should also be checking that how does a stored procedure programming language defer from language like C?

Stored Procedure Language (SPL) = SQL + What ?

- Should allow to create and use variables
- Should allow mixing variables with attribute-names in SQL.
- Allow to submit a SQL statement to the database, and collect the responses into variables so that can be manipulated further
- Support for various procedure constructs like if .. then .. else, loops, exception handling, etc.

Example Stored Procedure - Partial code for ComputeSPI in PL/pgSQL

```
FOR stud IN SELECT DISTINCT student_id from registers WHERE AcadYr = 2010 and Semester=1
Loop
    sum_credit := 0;
    sum_points := 0;
    FOR course_taken IN SELECT * FROM registers WHERE AcadYr = 2010 and Semester=1
                                                AND student_id = stud.student_id
    Loop
    IF course.grade = 'AA' THEN
        p := 10;
    ELSEIF course.grade = 'AB' THEN
        p := 9;
    ...
    END IF;

    SELECT credit into cr FROM course WHERE courseno = course_taken.courseno;
        sum_credit := sum_credit + cr;
        sum_points := sum_point + cr * p;
    end Loop;
    mspi = sum_points/sum_credits;
    UPDATE result set spi = mspi WHERE AcadYr = 2010 and Semester=1 AND
                                       student_id = stud.student_id;
end Loop;
```

More constraints using stored procedures

Some constraints are complex to be defined in SQL-DDL, and sometimes it is not possible to be defined in standard DDL. Code below should be self-explanatory. It adds a constraint that an employee can work project controlled by its home department-

```
CREATE TABLE  works_on (
    eno  integer,
    pno  smallint,
    hours  integer,
    PRIMARY KEY  (eno, pno),
    CONSTRAINT  work_proj_check
    CHECK  ( valid_work_dept(pno, eno ) )
)
```

```
CREATE OR REPLACE FUNCTION valid_work_dept(ppno smallint, peno integer)
  RETURNS bool AS
$BODY$

DECLARE

    edno employee.dno%TYPE;
    pdno project.dno%TYPE;

BEGIN

    SELECT dno into edno FROM employee WHERE eno = peno;
    SELECT dno into pdno FROM project WHERE p.pno = ppno;

    IF pdno = edno THEN
        RETURN true;
    ELSE
        RETURN false;

END IF;
```

CREATE ASSERTION command

CREATE ASSERTION is another command that is used for creating complex constraints. Following is an example.

```
CREATE ASSERTION salary_constraint
    CHECK ( NOT EXISTS ( SELECT * FROM employee e NATURAL JOIN
        department d JOIN employee m ON m.ssn = d.mgrssn
        WHERE e.salary > m.salary)
);
```

API based access to databases

Another approach for accessing remote databases in host program is API based access. API based access are more canonical, powerful and popular way of manipulating remote databases.

Over the period of time, various API standards have evolved in most popular programming environments. Open Database Connectivity (ODBC) was first initiative by Microsoft towards standardization. Current popular standards are JDBC, Python DB-API, MS ADO.Net.

Like in case of embedded SQL, in this approach too most of the time we perform one or more of following operations using API-

1. Connect to remote Database
2. Submit SQL statements
3. If executed query returns a row-set, iterate through the result-set and do some processing for each row.

Examples

```
13  Class.forName( "org.postgresql.Driver" );
14  DB_URL = "jdbc:postgresql://10.100.71.21/test";
15  user = "test";
16  passwd = "test";
17  Connection con =  DriverManager.getConnection(DB_URL, user, passwd);
18
19  Statement stmt = con.createStatement();
20
21  ResultSet rs = stmt.executeQuery("SELECT * FROM company.Employee AS e "
22          + "JOIN company.Employee AS s ON (e.superssn=s.ssn) WHERE e.dno=4");
23
24  while ( rs. next() ) {
25      String fname, ss, sname="";
26      fname = rs.getString("fname");
27      ss = rs.getString("superssn");
28      System.out.println( fname +" " +  ss + " " + sname);
29  }
```

Embedded SQL

- We allow SQL statements in a host program.
- Host program run on “client”?
- SQL run on DBMS server and response is brought to the client – same way your PgAdmin-IV does?
- Below is a sample embedded SQL in ECPG (a C pre-processor for PostgreSQL) almost compatible to oracle’s Pro*C

```
#include <stdio.h>

EXEC SQL BEGIN DECLARE SECTION;
int c;
EXEC SQL END DECLARE SECTION;

int main() {

    EXEC SQL CONNECT TO test@10.100.71.21 USER test USING test;
    EXEC SQL set search_path to company;
    EXEC SQL SELECT count(*) INTO :c FROM employee;
    printf("emps: %d\n", c);
    return 0;
}
```

- In embedded SQL, there is notion of “host variables”, these are variables declared within SQL DECLARE SECTION.
- Host variables can be mixed in SQL statements by pre-fixing colon. Often used to supply value into SQL statements or are used to collect values returned by executed SQL statement. For example, above see use of host variable **c**.

Prepared Dynamic Query

- A dynamic Query is the one that uses some variables, and complete query expression is known only at run-time. For example
  `query = "SELECT * FROM EMPLOYEE WHERE dno = " + dno`
  Query is stored in a variable query, and becomes fully known at run-time?
- Sometime such a query is repeated run on DBMS with changed value of some parameter, for example dno in above case.
- When run repeatedly, we can ask dbms to create its "query execution plan" once and run it for following calls.
- Such queries are called prepared queries.

Benefits of Prepared Queries

- Faster Execution – since query plan can be created once and saved, this makes execution of dynamic query faster (gain is significant, when a dynamic query appears in a loop or so)
- Avoids Run time exceptions
- A popular technique to deal with SQL injections