6. Application Programming

6.1 Introduction

Database and programming languages

- SQL is a powerful declarative query language. The formulation of queries in SQL is usually simpler than the coding of the same queries in an all-purpose programming language. There are at least two reasons for accessing a database with a programming language from a user perspective:
  - Not all queries can be expressed in SQL (little functionality for "everyday" programming) since SQL does not have the full expressive power of a programming language. In order to be able to express such queries, SQL can be embedded into a more powerful language. Applications are usually developed in imperative and object-oriented languages (C, Cobol, Fortran, Java, C++, ...).
  - Non-declarative actions like printing, interaction with the user, or transmission of query results to a graphical user interface are outside of SQL. Task sharing: query processing and updates with SQL, all other tasks with the aid of a programming language.
SQL queries can be automatically optimized and efficiently executed. The use of a programming language only makes an automatic optimization extraordinarily difficult.

Ad hoc queries are posed mostly by experts and more seldom. Frequently non-interactive batch applications are needed. Often the possibilities of the DBMS for representing data are limited and unsuitable for user requirements.

Special integration problem (impedance mismatch):
- programming language supports the processing of single data records (tuple-oriented approach).
- SQL supports the processing of data records, i.e., of relations (set-oriented approach).

consequently the question: How can the programming of database tasks be combined with the “usual” tasks without abandoning the benefits of SQL?
Alternatives for coupling

- **loose coupling**: Constructs of the database language are embedded into a program of a programming language and specially marked. Particular methods are employed to migrate from the set-oriented processing of SQL to the processing of single variables/records of the programming language.
  - CALL interface
    + provision of libraries
  - embedding with preprocessor (**embedded SQL**)
    + static: structure of the SQL commands is predefined
    + dynamic: arbitrary SQL commands are allowed

- **integration**: A special database language is developed which incorporates the usual programming language concepts and the set-oriented operations of the relational DBMS in a most possible unified way.
  - language extensions
    + of SQL
    + of an imperative or object-oriented programming language
  - script languages
    + languages similar to BASIC without type concept
    + simple connection to windows- and graphic-oriented interfaces
6.2 Procedural CALL Interface

Use of a library by employing the Oracle Call Interface (OCI)
Components of the CALL interface

- Data structures shared by the AP (application program) and the database server
  - For establishing the communication
  - For processing a query

- **Cursor concept**: Data structure used in the AP for accessing the relations of the DB

- In AP storage of SQL queries in a string

- Type checking only possible in the AP

- Binding of the variables of the AP to the data structures of the DBMS server

- Running an AP:
  - Establishing the connection to the DBMS server
  - Initialising a cursor
  - Parsing an SQL statement
  - Binding input variables to an SQL statement
  - Executing an update or a query
  - Closing the cursor
  - Decoupling from the server
executing a query

- requesting the output parameters
- binding the output to variables of the AP
- positioning the cursor
- abort of the query

drawbacks of the CALL interface
- complicated programming
- error-prone

advantage: high flexibility

JDBC - a CALL interface in Java

database programming in Java together with a use of SQL
- **JDBC (Java Data Base Connectivity)** protocol allows Java applications to access relational databases, independently of a particular DBMS.
- Queries are transmitted as uninterpreted strings to the DBMS.
- Results are sent through objects of a class `ResultSet` from the DBMS to the AP.
- **client-server concept**
  - DBMS runs as server for several clients on another computer than the AP.

- **uniform interface for different DBMS**

- **use of strong typing (whenever possible)**

- **support of important concepts**
  - static queries
  - SQL queries that can be parameterized
  - support of very large objects
  - dynamic queries and metadata

**Client-server coupling in JDBC**

- **Native coupling**

  ![Diagram of client-server coupling in JDBC]

  Special net software of the DBMS is installed on the client. Methods in JDBC are based on the functions of the underlying net software.
6. Application Programming

− disadvantages
  + installation of the net software on all clients
  + no independence from the database system

❑ ODBC coupling

![Diagram of ODBC coupling]

- ODBC: The standard **Open Database Connectivity** is an *application program interface (API)* that was suggested by Microsoft in 1991. ODBC supports a special “limited” SQL which is characterized by a minimal set of instructions.
- advantage: independence from special database systems
- disadvantages:
  + installation of ODBC + database drivers on all clients
  + overhead of ODBC
native pure Java client

- advantage: no installation of special software on the client
- disadvantage: dependence on database manufacturer

native protocol pure Java clients

- advantages
  + database independence
  + no software on the client

similar to the first solution, but the net software is reimplemented in Java and downloaded from the server on request (see e.g. the current coupling to Oracle).
Establishing a connection

- creation of a connection object

- `Connection con = DriverManager.getConnection("jdbc:oracle:thin:@venus.mathematik.uni-marburg.de:1521:Init_DB", "scott", "tiger");`
  - First string corresponds to an URL to the database.
  - Second string is the user name.
  - Third string is the password.

- Before creating the connection object the corresponding driver class has to selected.
  - `Class.forName("oracle.jdbc.driver.OracleDriver");`

Interpreted queries

- SQL query is interpreted (translated and at the same time executed). The result of the query is transferred to an object of class `ResultSet`. A repeated execution of the query requires a new interpretation. The query itself cannot be parameterized.
example:

```java
// Creation of a new object of class Statement
Statement stmt = con.createStatement();

// Translation of the query and creation of a new object of class ResultSet
ResultSet rs = stmt.executeQuery("select count(*) as number from user_tables");

// Operation next provides the functionality of an iterator.
rs.next();

// Access to the attribute values with get functions
System.out.println("Number of tables: ", rs.getInt(1));
System.out.println("Number of tables: ", rs.getString("number")); // alternatively
```

Precompiled queries

example:

```java
// An SQL query is translated with two parameters.
PreparedStatement stmt =
    con.PrepareStatement("select x, y from Points where x < ? and x > ?");
```
// The parameters of the query are set.
stmt.setInt(1, 20);
stmt.setInt(2, 10);

// The query is executed.
ResultSet rs = stmt.executeQuery();

- advantages
  - If queries are executed several times in a similar way, time is saved for the repeated translation process.
  - high optimization costs only once due to one translation

Dynamic SQL in Java

- JDBC permits to pose queries dynamically, since an arbitrary object of class String is expected as input for the execution of an SQL statement.

- example:
  String str;
  ...
  ResultSet rs = con.createStatement().executeQuery(str);

- problem: type of the result is unknown at run time
In order to provide such type information at run time, the class `ResultSetMetaData` is used. This class offers operations to query for metadata like the number of attributes and the database types of the result. An object of the class is then created by

```java
ResultSetMetaData rsmd = rs.getMetaData();
```

Afterwards the number of attributes of the result relation can be determined, for example, with the statement

```java
int count = rsmd.getColumnCount();
```

and with

```java
for (int i = 0; i < count; i++) {
    int sqlType = rsmd.getColumnType(i);
    ...
}
```

an integer is returned in each loop, which yields the type of the ith attribute. For each type the corresponding `get` function can then be called, for example.
6.3 Embedded SQL in Java (eSQL, SQLJ)

Basic principles

- use of a preprocessor
- static determination of database operations at translation time
- type checking between AP and database through the preprocessor
- simple transmission of data from the database into the AP
- use of the cursor principle for traversing relations
Syntactical tagging of database operations in Java APs

- syntax: `#sql{<SQL statement>}`
- An SQL statement relates to database objects. An exception are the so-called **host variables** that are used for the data transfer between the database and the AP.
- A host variable can be declared and used like a usual variable in Java.
- A host variable can be used in an SQL statement by preceding the variable name with a “:”.
- The purpose of host variables is to receive the results of a query. Only one result may be assigned at a time.
- extension of the **select** clause by the keyword **into** followed by the host variable
- examples:
  - `#sql{select A, B from R where B > :x}`
    The value of the variable `x` is inserted into the SQL command.
  - `#sql{select A, B into :a, :b from R where Id = 7}`
    The result is bound to the host variables `a` and `b` (assumption: `Id` is key candidate).