

SQL and Applications

SQL and applications

SQL and applications

Embedded SQL

Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure

- Manipulating the content of a DB through SQL statements is uncomfortable.
- In general, an application is used to access the DB.
- Applications may be realized using different languages:
 - IV generation languages (4GL)
 - special solutions, often integrated with a specific system or DBMS (Access → Maschere / VBA ...)
 - Standard programming languages (C, C++, Java) that can embed SQL statements.
 - Problem: we need shared variables to allow communication between two languages.

Embedded SQL

- Applications embed SQL statements inside standard statements.
- SQL statements are contained between special statements:

EXEC SQL

... SQL statements ...

;

- A **preprocessor** converts the SQL statements in function calls to a DBMS-specific library.
- Sometimes the preprocessor is able to identify syntax errors of the embedded language. In general encapsulates the errors of the DBMS.

SQL and applications

Embedded SQL

Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure

Embedded SQL variables

- A **declare section** is used to define the **shared variables** that are used to exchange information between the program and the SQL environment.
- The variables are preceded by ‘:’ when used as SQL parameters.
- The environment offer a predefined structure called **sqlca** (*SQL Communication Area*) with a field **sqlcode** which is used to query the result of SQL statements.

SQL and applications

Embedded SQL

Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure

Example

```
void main() {
    exec sql begin declare section;
    char *DipName = "DEI";
    char *Dipcity = "Milan";
    int    DipNumber = 18;
    exec sql end declare section;

    exec sql connect to user@universityDB;
    if ( sqlca.sqlcode != 0 ) {
        printf("Connection failed!\n");
    } else {
        exec sql insert into Department
            values (:DipName, :DipCity, :DipNumber);
        exec sql disconnect all;
    }
}
```

SQL and applications

Embedded SQL

Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure

Cursors

- Problem: **impedance mismatch**:
 - traditional programming languages manage one record at a time. (i.e., tuple-oriented)
 - SQL handles sets of tuples (i.e., set-oriented)
- **cursors** are used to solve this problem:
 - access to the result of a query in a set-oriented fashion.
 - returns one tuple at a time to the program.

- Cursor definition syntax:

```
declare cursor_name [ scroll ] cursor for SQL-select  
[ for < read only | update [ of Attribute {, Attribute} ]>
```

SQL and
applications

Embedded
SQL

Dynamic
SQL

Call Level
Interface
(CLI)

JDBC

Stored
procedure

Operations on cursors

- Executing a query associated to a cursor:

open *cursor-name*

- Extract a tuple from a result-set:

fetch [**from** *position*] *cursor-name* **into** *fetch-list*

- Deallocating a cursor:

close *cursor-name*

- Accessing the current tuple:

current of *cursor-name*

(used in the **where** clause)

SQL and applications

Embedded SQL

Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure

Using cursors

```
void ShowDeptSalaries(char deptName[]) {
    exec sql begin declare section;
    char name[20], sname[20];
    long int sal;
    exec sql end declare section;

    exec sql declare deptEmp cursor for
        select name, surname, salary
        from EMPLOYEE
        where department = :deptName;
    exec sql open deptEmp;
    exec sql fetch deptEmp into :name, :sname, :sal;
    printf("Department %s\n", deptName);
    while ( sqlca.sqlcode == 0 ) {
        printf("Name: %s %s ", name, sname);
        printf("Salary: %d\n", sal);
        exec sql fetch deptEmp
into :name, :sname, :sal;
    }
    exec sql close deptEmp;
}
```

SQL and applications

Embedded SQL

Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure

Dynamic SQL

- Application know the meaning of a SQL statement only at runtime.
- Problem: handling **parameter passing** between the program and the SQL statement.
- Direct execution (without parameters):
execute immediate *SQL-statement*
- Prepared statements:
prepare *command-name* **from** *SQL-statement*
followed by:
execute *command-name* [**into** *target-list*]
[**using** *parameter-list*]

SQL and applications

Embedded SQL

Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure

Call Level Interface (CLI)

- An alternative to embedded SQL consists in offering an API to programmers to communicate with the DBMS.
- Standard procedure
 - create a **connection**
 - send a SQL statement through a connection
 - the result has a **relational structure (result-set)**
 - **close** the connection
- Runtime command. No optimization possible at runtime.

SQL and applications

Embedded SQL

Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure

ODBC

- Open DataBase Connectivity (MS ODBC) is one of the most diffused Call Level Interfaces.
- ODBC 4-level architecture:
 - **Application level**
 - specifies the driver to use to interact with the DBMS. The driver masks data-level, networking and enables separation of concerns.
 - **Driver manager**
 - load the specific driver
 - **Driver**
 - realizes the communication channel between the application and the data source.
 - **Datasource**
 - the specific DBMS. Executes the SQL statements.

SQL and applications

Embedded SQL

Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure

OLE DB and ADO

- OLE DB is a general interface to access datasources (*not only relational DBs*)
 - **Object Linking & Embedding DB**
 - It offers a, more or less, rich set of statements depending of the datasource's data model.
- Based on the OO paradigm MS COM.
- **ActiveX Data Object (ADO)** is an interface used to access to datasources that offer an OLE interface.

SQL and applications

Embedded SQL

Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure

ADO

- Object model based on 4 components:
 - *Connection*
 - communication channel between the application and the DBMS.
 - In order to be created it needs the DBMS location and user credential (e.g., `http://orsi@localhost:3306`).
 - *Command*
 - contains the SQL statement to be executed on the datasource.
 - *Record*
 - represents a single tuple.
 - *RecordSet*
 - represents a set of tuples.
 - **implementation of a cursor.**
 - Thanks to the object model we do not need to define it in SQL.

SQL and applications

Embedded SQL

Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure

JDBC

- Java DataBase Connectivity.
- Similar to ODBC. It is a java implementation of a CLI
 - The driver manager isolates the correct driver for a specific DBMS.
- JDBC Usage
 - Load the driver
 - Create a connection with the DBMS.
 - Creation of a SQL command (prepared statement).
 - Process the result-set.
 - Close the connection.

SQL and applications

Embedded SQL

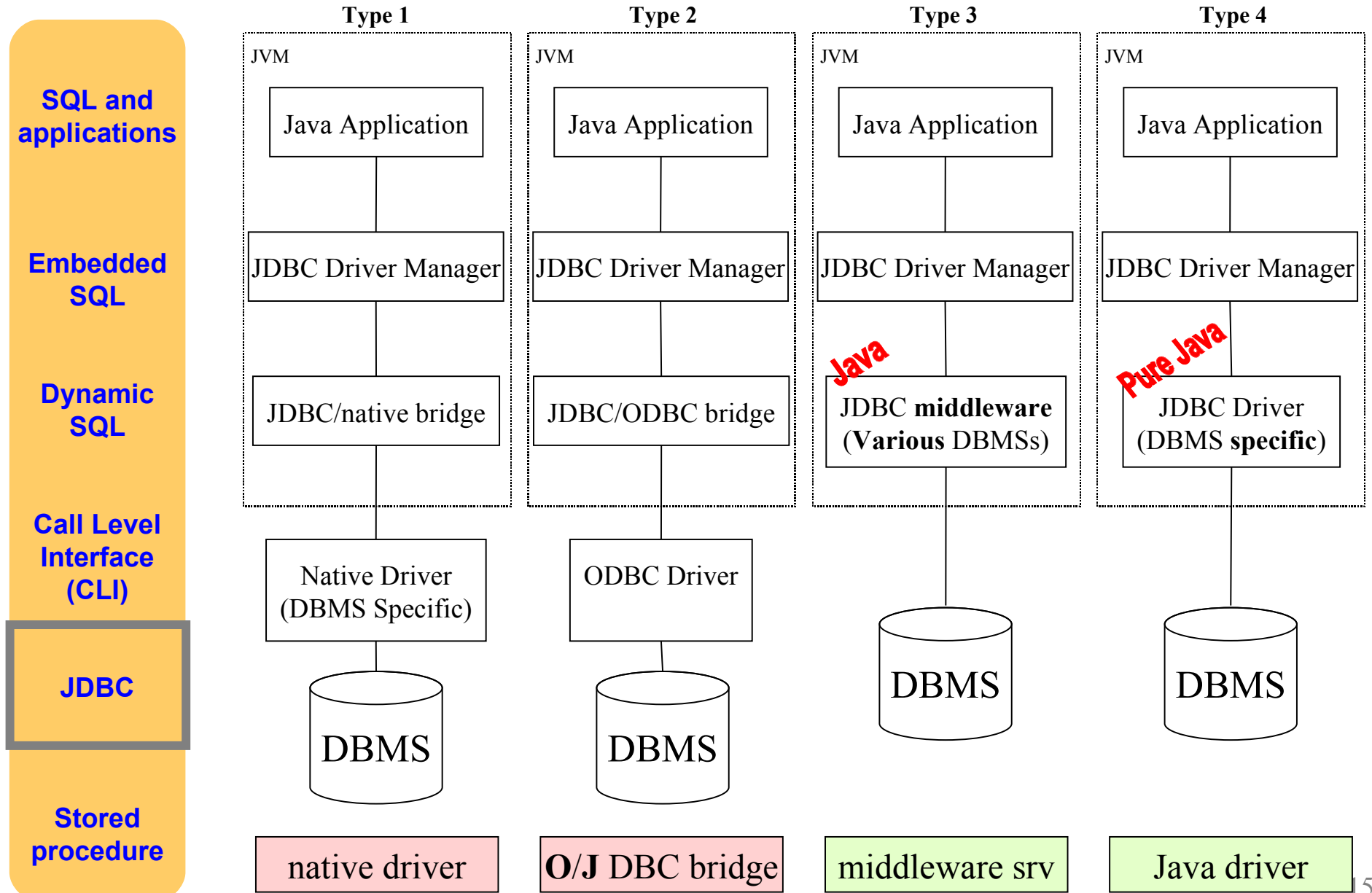
Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure

JDBC: Architecture



Creating a connection

SQL and applications

```
...  
//load the driver.  
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Embedded SQL

```
//prepare credentials.  
url = "jdbc:odbc:http://localhost:3300";  
String user = "userID";  
String pwd = "pwd";
```

Dynamic SQL

```
//create a connection.
```

Call Level Interface (CLI)

```
Connection c;  
c = DriverManager.getConnection(url, user, pwd);
```

JDBC

```
/*.....do your things.....*/
```

Stored procedure

```
//close the connection  
con.close();  
...
```

Statements

```
Statement stmt;  
stmt = con.createStatement();  
stmt.execute(  
    "INSERT INTO COFFEES " +  
    "VALUES ('Colombian',101,7.99,0,0)" );  
.....  
stmt.close();
```

SQL and applications

Embedded SQL

Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure

Query Statements

SQL and applications

```
String query = "SELECT * FROM COFFEES";
```

Embedded SQL

```
Statement stmt = con.createStatement();  
ResultSet rs = stmt.executeQuery(query);
```

Dynamic SQL

```
/*... do your things ...*/
```

Call Level Interface (CLI)

```
//de-allocate the cursor.  
rs.close();
```

JDBC

```
//close the statement.  
stmt.close();
```

Stored procedure

Showing a result-set

```
while ( rs.next() )
{
    String s = rs.getString("COF_NAME");
    float n = rs.getFloat("PRICE");

    System.out.println(s + " " + n);
}
```

SQL and applications

Embedded SQL

Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure

Exception Handling

Every operation you made on the DBMS may generate an exception. Exceptions are handled at application level.

```
try {  
    //statements  
} catch (SQLException s) {  
    //Exception handling  
}
```

SQL and applications

Embedded SQL

Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure

Stored Procedures

SQL and applications

- DB procedural statements that manipulates data directly on the DBMS.

Embedded SQL

- SQL-2 allows a limited usage of such procedures.

Dynamic SQL

- Most of the systems offer procedural extensions (e.g., Oracle PL/SQL) that are as expressive as normal programming languages.

Call Level Interface (CLI)

- Client-server architecture:

JDBC

- invoked by a client.
- stored and executed on a server.

Stored procedure

Stored Procedures

- Two steps:
 - procedure declaration (DDL)
 - procedure invocation (DML)
- stored procedure are part of the schema definition.
 - encapsulate the data and specify the DBMS *behavior*.

```
procedure Proc_name (val type, ...) is
begin
    // variables and exception declarations

    // SQL statements

end;
```

SQL and
applications

Embedded
SQL

Dynamic
SQL

Call Level
Interface
(CLI)

JDBC

Stored
procedure

Benefits

- Allows applications modular decomposition
- Increases:
 - efficiency
 - control
 - reuse
- Increase the responsibility of the DB administrator (w.r.t. the programmer)
 - DBA is responsible for a piece of the business logic.
 - business logic moves from the application to the DBMS.
- Data independence.

SQL and applications

Embedded SQL

Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure

Problems

- Efficient solution only if the problem requires simple operations on great amounts of data (OLTP).
- Not efficient if complex operations are executed. Other systems needed (OLAP).

SQL and applications

Embedded SQL

Dynamic SQL

Call Level Interface (CLI)

JDBC

Stored procedure