

9 Embedding SQL into Programming languages

- 9.1 Introduction: using SQL from programs
- 9.2 Embedded SQL
 - Static and dynamic embedding
 - Cursors
 - ESQL / C
 - Positioned Update
- 9.3 SQL and Java
 - JDBC
 - SQLJ
- 9.4 OR mapping and components
- 9.5 Transactions in application programs
 - Definition
 - Isolation levels

Lit.: Kemper / Eickler: chap. 4.19-4.23;
Melton: chap. 12,13,17-19, Widom, Ullman, Garcia-Molina: chapt.8
Christian Ullenboom Java ist auch eine Insel, Kap. 20, Galileo Comp.

Using SQL from Programs



- SQL is a **data sublanguage**
- Easy **data** access but no way to define "business rules"
- **Business rule**: Algorithm which implements an application specific use cases
Example (Uni-DB): student finished her studies and leaves university
- Needs a **host language** offering
 - Control structures**
 - User interface**: output formatting, forms
 - Transaction**: more than one DB interaction as a unit of work for implementing business rules.

© HS-2010

9-Embedded-2

SQL and Programming Languages



Issues

Language mismatch ("impedance mismatch")

- Set oriented operations versus manipulation of individuals -> Objects vs. Relations
- How to interconnect program variables and e.g attributes in SQL statements?
- Should an SQL-statement as part of a program be compiled, when?

© HS-2010

9-Embedded-3

SQL from Programs: Introduction



Overview of language / DB integration concepts

Interface of standard programming languages

- **Call level interface**, proprietary library routines, API
Standardized: SQL CLI **Open Database connection (ODBC)**,
- **Embedded C / Java / ..**
Standardized language extensions
- Standardized API
Java DBC

Object-Relational Mapping

- JPA (Java Persistence Architecture), Hibernate

Component architectures: hiding the details of DB interaction, Enterprise Java Beans (EJB)

© HS-2010

9-Embedded-4

Using SQL from Programs:CLI



Call level interface

Language and DBS specific library of procedures to access the DB

Example: MySQL C API

- Communication buffer for transferring commands and results
- API data types like
 - MYSQL** handle for db connections
 - MYSQL_RES** structure which represents result set
- API functions
 - `mysql_real_query()`
 - `mysql_real_query (MYSQL *mysql, const char * query, unsigned int length)`
 - query of `length` of character string in buffer and many more....

Standard : Open Database Connection (**ODBC**)
Predecessor of Java Database Connection (**JDBC**), see below

© HS-2010

9-Embedded-5

SQL Call level interface (SQL/CLI)

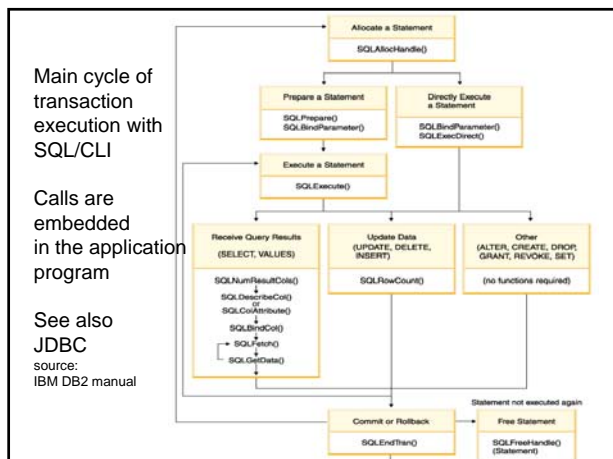


Standardized Interface to C / C++ defined by **X/OPEN** and **SQL Access group**

- Main advantages
 - DBS-independent**
Application development independent from DBS
(as opposed to Embedded SQL precompiler approach, see below)
 - Easy to connect to multiple DB
- Microsoft implementation
 - ODBC** (= Open Database Connectivity) de facto standard, available not only for MS products

© HS-2010

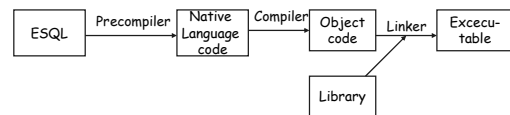
9-Embedded-6



9.2 Embedded SQL

Embedded SQL – the most important(?) approach Concepts

- Program consists of "native" and SQL-like statements (e.g. Embedded C, SQLJ)
- Precompiler compiles it to native code, includes calls to DBS resources
- Employs call level interface in most implementations



© HS-2010

9-Embedded-8

Embedded SQL (ESQL): what needed?

- Well defined type mapping (for different languages)
- Syntax for embedded SQLJ statements
`#sql [[<context>]] { <SQL-Anweisung> }`
- Binding to host language variables
`#sql {SELECT m# FROM M WHERE titel = :titleString};...`
`#sql {FETCH ...INTO :var1}`
hypothetical syntax, like SQLJ
- Exception handling (WHENEVER condition action)
`SQLSTATE, SQLCODE (deprecated)`

© HS-2010

9-Embedded-9

ESQL

C / Java embedding

ESQL/C

```
EXEC SQL UPDATE staff SET job = 'Clerk'
WHERE job = 'Mgr';
if ( SQLCODE < 0 printf( "Update Error: ... );
```

SQLJ

```
try { #sql { UPDATE staff SET job = 'Clerk'
WHERE job = 'Mgr' }; }
catch (SQLException e)
{ println( "Update Error: SQLCODE = " + ... );
```

© HS-2010

9-Embedded-10

ESQL Static / dynamic embedding

Static versus dynamic SQL:

Static: all SQL commands are known in advance, SQL-compilation and language binding at precompile time

Dynamic

- SQL-String executed by DBS:
Operator tree, optimization, code binding....
- SQL-String *prepared* (compiled) at runtime.
Performance gain in loops etc.

© HS-2010

9-Embedded-11

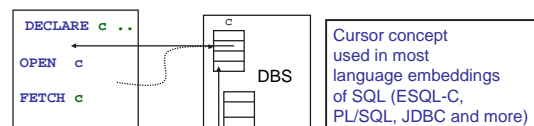
ESQL Cursors

Cursor concept

How to process a result set one tuple after the other?

CURSOR: name of an SQL statement and a handle for processing the result set record by record

Cursor is defined, *opened at runtime* (= SQL-statement is executed) and used for **FETCHing** single result records



Buffers for application program cursors
DBS may determine result set in a lazy or eager way

© HS-2010

9-Embedded-12

ESQL Cursors

Cursor : just like cursors in PL/xxSQL, but typically more positioning operations

```
Declare curs for Select c#, lname, m.title
from C, R, M where ....
```

Cursor curs

7369	SMITH	To be or ..	Active set
7566	JONES	Metropolis	
7788	SCOTT	Forest Gump	Current row
7876	ADAMS	Forest Gump	
7902	FORD	Star Wars I	

© HS-2010

9-Embedded-13

ESQL

Fetch

```
FETCH curs INTO :x, :nameVar, :titleVar;
```

Cursor scrolling (Declare c SCROLL cursor.. in SQL 92):

```
FETCH [NEXT | PRIOR | FIRST | LAST |
[ABSOLUTE | RELATIVE expression] ]
FROM cursor INTO target-variables
```

```
FETCH curs PRIOR INTO :x, :nameVar, :titleVar;
```

=

```
FETCH curs RELATIVE -1 INTO :x, :nameVar, :titleVar;
```

Single row **SELECT** does not need a FETCH but result is bound to variables: SELECT a,b FROM... INTO :x,:y WHERE

© HS-2010

9-Embedded-14

SQLJ - Example

```
#sql private static iterator EmployeeIterator(String,
String, BigDecimal);
...
EmployeeIterator iter;
#sql [ctx] iter = {
    SELECT LASTNAME , FIRSTNAME , SALARY
    FROM DSN879.EMP
    WHERE SALARY
    BETWEEN :min AND :max };
while (true)
{ #sql {
    FETCH :iter
    INTO :lastname, :firstname, :salary };
if (iter.endFetch())
    break; ..... }
iter.close();
```

© HS-2010

9-Embedded-15

ESQL

Opening

```
OPEN cursor_name;
```

In a **compiled language** environment (e.g. embedded C):

- bind input variables
- execute query
- put (first) results into communication (context) area
- no exception if result is empty
- has to be checked when fetching the results
- positions the cursor before the first row of the result set (" -1 ")

First steps in an **interpreted language** (e.g. 4GL PL/SQL) :

- allocate context area
- parse query

© HS-2010

9-Embedded-16

9.2.3 ESQL

```
#include <stdio.h>

/* declare host variables
*/
char userid[12] =
"ABEL/xyz";
char emp_name[10];
int emp_number;
int dept_number;
char temp[32];
void sql_error();

/* include the SQL
Communications Are
*/ #include <sqlca.h>
```

```
main()
{ emp_number = 7499;
/* handle errors */
EXEC SQL WHENEVER SQLERROR
do sql_error("Oracle
error");

/* connect to Oracle */
EXEC SQL CONNECT :userid;

/* declare a cursor */
EXEC SQL DECLARE
emp_cursor CURSOR FOR
SELECT ename
FROM emp
WHERE deptno =
:dept_number;
```

ESQLExample: Embedded C

```
printf("Department number? ");
gets(temp);
dept_number = atoi(temp);
/* open the cursor and identify the active
set */
EXEC SQL OPEN emp_cursor; ...
/* fetch and process data in a loop
exit when no more data */
EXEC SQL WHENEVER NOT FOUND DO break;
while (1)
{ EXEC SQL FETCH emp_cursor INTO
:emp_name; ..
}
EXEC SQL CLOSE emp_cursor;
EXEC SQL COMMIT WORK RELEASE;
exit(0); }
```

Close cursor before another SQL statement is executed

© HS-2010

9-Embedded-18

ESQL Exception handling

Exception handling: error routine (C)

```
void sql_error(msg)
char *msg;
{
    char buf[500];
    int buflen, msglen;
    EXEC SQL WHENEVER
        SQLERROR CONTINUE;
    EXEC SQL ROLLBACK WORK
        RELEASE;

    buflen = sizeof (buf);
    sqlglm(buf, &buflen, &msglen);
    printf("%s\n", msg);
    printf("%*.s\n", msglen, buf);
    exit(1);
}
```

© HS-2010

9-Embedded-19

ESQL Exception handling

```
EXEC SQL WHENEVER SQLERROR GOTO sql_error;
...
sql_error:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL ROLLBACK WORK RELEASE;
...
```

Without the WHENEVER SQLERROR CONTINUE statement, a ROLLBACK error would invoke the routine again, starting an infinite loop.

© HS-2010

9-Embedded-20

Positioned Update

Update / Delete statements in general use search predicates to determine the rows to be updated

```
Update M
set price_Day = price_Day+1 where price_Day <= 1
```

Often useful: step through a set of rows and update some of them ⇒ positioned update

```
DECLARE myCurs FOR SELECT ppd, title FROM M
FOR UPDATE ON ppd
UPDATE M SET ppd = ppd + 1
WHERE CURRENT OF myCurs /* delete in a
                        /*similar way
```

A cursor may declared **FOR READ ONLY**

- which basically results in some performance gains

© HS-2010

9-Embedded-21

ESQL Cursor sensitivity

Which state has the database during processing?

```
EXEC SQL DECLARE myCurs FOR SELECT price_Day, title
FROM M FOR UPDATE ON price_Day
WHERE price_Day < 2
EXEC SQL OPEN...
...
EXEC SQL FETCH myCurs INTO .....
UPDATE M SET price_Day = price_Day + 2
WHERE CURRENT OF myCurs /* similar for
                        /* delete
```

Is the row under the cursor still in the result set?

Yes and No !

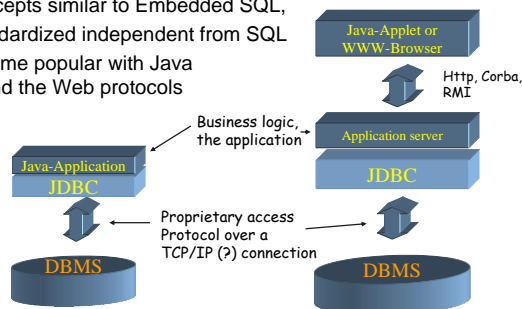
A cursor declared **INSENSITIVE** does not make visible any changes (update, delete) until the cursor is closed and reopened.

© HS-2010

9-Embedded-22

9.3 SQL and Java

Concepts similar to Embedded SQL, Standardized independent from SQL became popular with Java and the Web protocols



© HS-2010

9-Embedded-23

SQL & Java: JDBC und SQLJ

SQLJ

- essentially embedded SQL for Java
- Compiles to JDBC method call, more interweaved with programming language.
- Defined and implemented by major DBS companies (Oracle in particular)

JDBC

- Java call-level interface (API) for SQL DBS
- API DB vendor independent, but often not implemented fully
- Supports static and dynamic SQL
- Implemented by nearly all DB vendors

© HS-2010

9-Embedded-24

9.4.1 JDBC

Preparation

```
import java.sql.*;
```

(1) **Load a driver** (included in java.sql.*), many vendor products
`Class.forName("oracle.jdbc.driver.OracleDriver");`
url is a variable holding the JDBC-Driver and host information

(2) **Set up the connection** to one or more database(s)
`Connection con = DriverManager.getConnection("jdbc:oracle:thin:@tarent.mi.fu-berlin.de:1521:hstarent", username, password);`
Several connections at a time may be used.

Opening a connection takes time! A solution... see below.

© HS-2010

9-Embedded-25

Creating a connection

(3a) Create a statement object

```
Statement stmt = con.createStatement();
```

something like a channel, through which queries are sent to the DB

Note: `stmt` this is NOT a statement, but a "statement channel".

© HS-2010

9-Embedded-26

Executing a query from Java

Processing

Queries may now be "sent through the statement channel" of a connection to the DBS and executed:

```
ResultSet rs = stmt.executeQuery("Select * from myTab");
```

Note: Only `String` (The SQL query as a string) is allowed
no parameters within SQL command!

`ResultSet` object manages result table:

`boolean next()`, `boolean last()`, `boolean previous()`,... similar to cursor in ESQL,
but cursor is "hidden".

Many methods for extracting column values from
`int getInt(int ColumnIndex)`, `String getString(...)`,

© HS-2010

9-Embedded-27

SQL & Java: JDBC

Processing

```
ResultSet rs = stmt.executeQuery("SELECT * FROM Movies WHERE pricePD > 2");
```

Process the results one after the other

```
while (rs.next())
{ // Loop through each column, get the
  // column data and display
  for (i = 1; i <= numCols; i++)
  {
    if (i > 1) System.out.print(",");
    System.out.print(rs.getString(i));
  }
}
```

Variable binding
by position

© HS-2010

9-Embedded-28

JDBC: variable binding, result set iterator

Variable binding by name

```
java.sql.Statement stmt = con.createStatement();
ResultSet r = stmt.executeQuery("SELECT a, b, c FROM Table1");

while (r.next())
{
  // print the values for the current row.
  int i = r.getInt("a");
  String s = r.getString("b");
  float f = r.getFloat("c");
  System.out.println("ROW = " + i + " " + s + " " + f);
}
```

Compare
variable binding
by position

© HS-2010

9-Embedded-29

SQL and Java Exceptions

```
try {
  // Code that could generate an exception goes here.
  // If an exception is generated, the catch block below
  // will print out information about it.
} catch (SQLException ex) {
  System.out.println("\n--- SQLException caught ---\n");
  while (ex != null) {
    System.out.println("Message: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("ErrorCode: " + ex.getErrorCode());
    ex = ex.getNextException();
    System.out.println("");
  }
}
```

process all exceptions

```
--- SQLException caught ---
Message: There is already an object named 'Meier' in the database.
Severity 16,
State 1, Line 1
SQLState: 42501 -- Defined as X/Open standard
ErrorCode: 2714 -- Vendor specific
```

© HS-2010

9-Embedded-30

JDBC

Accessing columns

The class `ResultSet` has methods for each type to access result data by position or by name

By position:

```
String s = rs.getString(2); // the second attribute to be bound.
```

By name:

```
String rs.getString("b");  
// get the value of the attribute b of the  
// row under the (implicit) cursor
```

Input parameters for queries? (... where attr = :val)

JDBC

Prepared statements

```
String mTitle; ....  
try {  
    java.sql.PreparedStatement prepStmt =  
        con.prepareStatement("SELECT count(*)  
        FROM M,T where M.title = ? and T.mId = M.mId);  
    prepStmt.setString(1, mTitle);  
    ResultSet r = prepStmt.executeQuery();  
    while (r.next())  
    {  
        int i = r.getInt(1);  
        // must get by position, no name available  
        System.out.println("Number of tapes for " +  
            movieTitle + " is: " + i)  
    }  
} catch(SQLException {...}
```

Subclass of statement
will be compiled,
? Indicates parameter
Bind value to position (!)
in statement and execute

Prepared vs non-prepared

- Prepared statement have IN parameters
- Overhead for compiling SQL-statement basically constant
- Ratio of compile Time / processing time for queries important
- Simple queries: prepare when executed frequently – e.g. in a loop
- Complex queries in a loop: no performance gain
- Many more factors to be analyzed