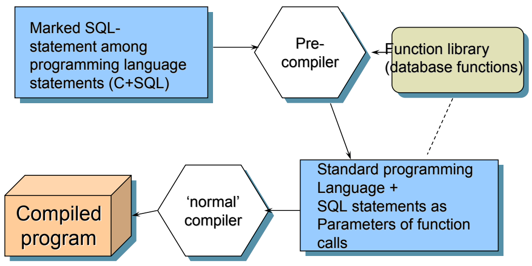
Programming Database Applications

- Databases are rarely used thru direct query interfaces
- Typically they are used thru application programs
- Techniques to use databases in application programs include:
  - Embedded SQL
  - Database Application Programming interface Library (database API)
  - Other interface that hide the database
  - A special database programming language

Embedded SQL

- SQL-statements are written among programming language statements. SQL-statements carry special (Exec SQL) marks that identify then as being SQL
- The marks are used by a pre-compiler that substitutes the SQL statements with calls of library functions
- The result of the pre-compiler is then compiled with a standard programming language compiler
- 2-phase compilation
- Pre-compilers are available from dbms suppliers for some programming languages (Ada, C, C++, Cobol, Pascal,…, Java)

Embedded SQL


Marked SQL-statement among programming language statements (C+SQL)
Pre-compiler
Function library (database functions)
Standard programming Language + SQL statements as Parameters of function calls
'normal' compiler
Compiled program

Example of Embedded SQL in Pascal

```
function avgSalary(dept:integer):real;
var
  n: integer;
  salSum: integer;
  #include SQLCA.INC
  EXEC SQL BEGIN DECLARE SECTION
      var  salary: integer;
           dpt: integer;
  EXEC SQL END DECLARE SECTION
```

Example of Embedded SQL in Pascal

```
begin
   EXEC SQL DECLARE salaryCursor CURSOR FOR
      SELECT salary from employee
      where department= :dpt;
   n:=0; salSum:=0; dpt:= dept;
   EXEC SQL  open salaryCursor;
   EXEC SQL fetch salaryCursor into :salary;
   while sqlcode = 0 do begin
      salSum := salSum + salary;
      n := n + 1;
      EXEC SQL fetch salaryCursor into :salary;
   end;
   EXEC SQL close salaryCursor;
   if n > 0 then  avgSalary := salSum/n
   else  avgSalary := 0;
end;.
```

Concepts in embedded SQL

- Cursor
  - Structure for processing the result of a query
    - Must be attached to a query (declare)
    - Is opened (open) = the attached query is executed using the variable values of the execution time
    - Is moved (fetch) = proceed to next record and transfer data from the current record to program variables
    - Is closed (close) = releases resources

Concepts in embedded SQL

- A database operation may fail. The success of the operation must be tested after each operation.
- Success may be tested by checking the value of the variable sqlstate (or sqlcode according to the older standard). Both return 0 if the operation succeeded and an error code it failed.
- Success may be also tested by defining error handlers (WHENEVER something do Handler)

Programming using a database API

- Application programming interface (API) is library that provides the services of the DBMS.
- Supplier specific libraries: **Native API**
  - For example OracleCLI = Oracle Call Level Interface
- Supplier independent libraries
  - For example ODBC (Microsoft Open Database Connection), JDBC Java database connection
  - Make it possible to change dbms and even to use many dbms in the same program

Programming using a database API

- Supplier independent libraries require a dbms specific driver to work with a particular dbms
- ODBC is the most common API
  - All suppliers provide ODBC drivers.
  - Parameters passed in C-language style
- JDBC provides the same ideas as ODBC but in Java
  - Some details of ODBC are hidden within class definitions.

JDBC

- Java database connection (JDBC) provides only a few classes:
- DriverManager
  - This class provides services for attaching the dbms specific driver and for establishing connections to the database,
  - DBms drivers are able to register themselves. Thus it is enough to load the driver, for example, using classByName service.
  - Here is the code for explicit registration of the driver

  -
    ```
    DriverManager.registerdriver(
            new  oracle.jdbc.OracleDriver());
    ```

JDBC

- Connection
  - This class establishes a database connection (session):
    - a connection between the application and the database – log in with some user account and password
  - All database services need a connection – Connection class connects service requests to the connection
  - Connection should be closed when it is no longer needed
  - DriverManager provides the method for creating a connection

JDBC

```
Connection con =
   DriverManager.getConnection(

  "jdbc:oracle:thin:@bodbacka.cs.helsinki.fi:1521:test",
    "scott","tiger");
```

Establishes a connection using Oracle thin driver via port 1521 to computer bodbacka.cs.helsinki.fi and its test- database using user account scott and password tiger

JDBC

- Statement
  - Environment for executing database operations
  - Provides, for example, methods
  - executeQuery - to execute queries
  - executeUpdate - to execute other operations
- Connection provides a methos for creationg statements

JDBC

- ResultSet
  - The answers obtained by executing a query
  - Corresponds to the cursor of embedded SQL
  - Method Statement.executeQuery() creates a ResultSet object
    - executeQuery accepts the actual query as its parameter

```
Statement stmt= con.createStatement();
ResultSet rs= stmt.executeQuery(
   "select name from employee");
```

JDBC

- Answer processing using the methods of ResultSet:
  - Method next() activates the next row of the answer. It returns *true* if such a row exists and otherwise *false*. First call activates the first row of the answer.

JDBC

- Data may be only in program variables. ResultSet provides data type specific methods getType to transfer data from active row to program getString for Strings, getBoolean for booleans, getInt, getDate,….
- These functions use the column name or the column sequence number as their parameter
- esim:
  - *String a= rs.getString("address");   // column address*
  - *Int p= rs.getInt(3);   // third column*

JDBC

- Get-functions are able to perform some data type conversions, for example to change integers to strings or vice versa. If conversion fails an SQLException is thrown. All error conditions cause SQLException to be thrown.
- Null values need special treatment. getString returns java.NULL in case of SQL null value, but, for example, getInt returns 0 (zero). To test whether the value was null there is method wasNull. It is parametrized like the get-methods.

JDBC

```
Statement stmt= con.createStatement();
ResultSet rs= stmt.executeQuery(
   "select name, address, salary from employee " +
   " order by name");
while (rs.next()) {
   System.out.println(rs.getString(1) +", "+
      rs.getString(2)+", "+rs.getString(3));
}
```

Output :

```
Lahtinen Kalle, Katu 6, 12000
Mäki Manu, Kuja5, 20000
```

JDBC

- Statement that do not produce a result (insert, delete, update, create, alter, …) are executed using **Statement.executeUpdate**-method.
- example

```
Int empUpdated=
   stmt.executeUpdate("update employee "+
     "set salary= salary + 10000 " +
     " where name= 'Laine Harri' ");
```

JDBC

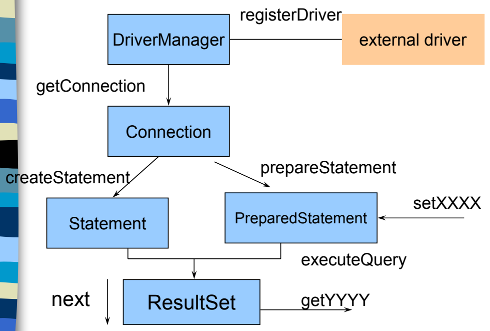
- **Parametrized operation:**
  - Database operations may always be executed by giving them as parameters for the methods executeUpdate and executeQuery. Sometimes almost the same operations is used repeatedly with only minor changes. In these cases it might be useful to compile the query only once and reuse the compiled query.
  - Class PreparedStatement supports this way of programming.

JDBC

```
PreparedStatement pst =
        con.prepareStatement(
            "select name,address,salary "+
            "from employee "+
            "where name like ?" );
```

- Question mark indicates a parameter, It can be used to substitute a constant value.
- PreparedStatement provides data type specific set-methods for assigning values for these parameters (setString for strings)
- Set methods have two parameters:
  - The sequence number of the question mark in the SQL-operation and
  - The value to substitute the parameter
  - Example;   pst.setString(1,"Smi%");
- Prepared statement has executeQuery and execute update methods, but these don't need any parameter.

JDBC


DriverManager
registerDriver
external driver
getConnection
Connection
createStatement
prepareStatement
Statement
PreparedStatement
setXXXX
executeQuery
next
ResultSet
getYYYY