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
8.2 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:
  
  1. establishing the connection to the DBMS server
  2. initialising a cursor
  3. parsing an SQL statement
  4. binding input variables to an SQL statement
  5. executing an update or a query
  6. closing the cursor
  7. 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
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:

// 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

In these queries the translation of an SQL query is separated from the execution of the query.

element:

// 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:
  ```java
  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.
8.3 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

![Diagram](image-url)
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).
Creation of a connection to a database

- SQLJ needs a reference (a context) to an existing database:
  ```sql
  #sql context connect
  ```
- Afterwards `connect` can be used like a class which especially contains the following constructor:
  ```java
  connect connectionObject =
  new connect("jdbc:oracle:thin:@venus.mathematik.uni-marburg.de:1521:Init_DB", "scott", "tiger");
  ```
- This context object is optionally part of the SQL statement:
  ```sql
  (connectionObject){select A, B from R where B > :x}
  ```
- During the translation of an SQLJ program, checks are feasible which can be performed with JDBC only later at run time.

Query formulation with iterators

- For SQL statements that yield more than one answer, iterators (cursors) can be defined.
- distinction between position-related and name-related iterators
Position-related iterators (example)
- declaration of an iterator type Pos with two components
  \#sql public iterator Pos(string, int);
- declaration of a variable of that type
  Pos x;
- binding of an SQL command to that variable
  \#sql x = {select A, B from R where B > 10};
- The access to the result set is then performed in a loop:
  \begin{verbatim}
  while (not x.endFetch()) {
    \#sql {fetch :x into :a, :b};
    System.out.println(a + " earns " + b + " Dollars.");
  }
  \end{verbatim}

Name-related iterators (example)
- declaration
  \#sql public iterator Name(string A, int B);
- declaration of a variable of that type
  Name y;