Establishing a connection

- creation of a connection object
  
  ```java
  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.
  ```java
  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

example:

// 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.
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
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:
  ```java
  #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:
  ```java
  #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
  ```java
  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:
  ```java
  while (not x.endFetch()) {
    ```sql
    {fetch :x into :a, :b};
    ```
    System.out.println(a + " earns " + b + " Dollars.");
  }
  ```

Name-related iterators (example)
- declaration
  ```sql
  public iterator Name(string A, int B);
  ```
- declaration of a variable of that type
  ```java
  Name y;
  ```
binding to an SQL command

```sql
y = {select A, B from R where B > 10};
```

access to the result set

```java
while (y.next()) System.out.println(y.A() + " earns " + y.B() + " Dollars.");
```

access to the values is done by calling methods where the name of the method corresponds to the name of the attribute.

Method `next` accesses the next tuple.

Set-valued operations for change and deletion

Such operations also employ iterators. The data set to be changed or deleted is bound to the iterator. Then changes can be executed.

```sql
public iterator Name(String A, int B);
Name y;
...
```

```java
y = {select A, B from R where B > 10};
...
while (y.next()) #sql {update R set B = B + 10 where current of :y};
```

The currently addressed data record is changed.