

## Creation of a connection to a database

- ❑ SQLJ needs a reference (a context) to an existing database:

**#sql context** connect

- ❑ Afterwards *connect* can be used like a class which especially contains the following constructor:

```
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:

```
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

```
Name y;
```

- binding to an SQL command

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

- access to the result set

```
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;
```

```
...
```

```
#sql 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.

## 6.4 PL/SQL

### Introduction

- ❑ PL/SQL is Oracle's procedural/imperative language extension to SQL.
- ❑ Syntax is very similar to the programming language ADA.
- ❑ PL/SQL offers software engineering features like data capsuling, information hiding, overloading, and exception handling.
- ❑ PL/SQL is a block-structured language. Basic units like procedures, functions and anonymous blocks are logical blocks that can contain a number of nested subblocks. A block or subblock groups declarations and statements that logically belong together. Declarations are valid only locally to the block and do not exist any more, if the block has been executed.
- ❑ advantages with respect to a host language
  - homogeneous connection of the imperative concepts to SQL
  - type conversions are not needed
  - platform independent execution
- ❑ disadvantage: imperative concepts are not sufficient for a complete development of APs

## PL/SQL block

- ❑ PL/SQL block consists of three parts
  - an optional declaration part where variables and objects can be declared,
  - an executable part where variables are manipulated,
  - an optional exception handling part where exception and errors can be dealt with that arise during execution.

- ❑ definition of an PL/SQL block

```
[DECLARE <declarations>]
BEGIN
    <statements>
    [EXCEPTION <exceptions>]
END
```

## Declaration part

- ❑ type declarations
  - Variables can be of an SQL data type or of an additional PL/SQL data type (e.g. *boolean*).
  - Variables can be assigned values.

- PL/SQL also supports the definition of records

**type** person\_type **is record** (name *varchar*(50), salary *int*);

#### □ variable declarations

- specialty: data types of the relations can be used for the declaration of variables  
example: myBook books%rowtype
- example: yourBook myBook%type

Program variables can but need not be identical to the corresponding attribute names. The **%type** notation in each variable declaration means that this variable is of the same type as the corresponding attribute in the relation. That is, a variable of the type of the variable *myBook* is declared.

#### □ cursor declarations

- The introduction of a cursor (logical pointer to a tuple within a relation) allows the sequential processing of all tuples that form the result of a query
- constant cursor  
**cursor** current-book **is** **select** \* **from** books;
- parameterized cursor  
**cursor** average-earner(from *int*, to *int*) **is**  
**select** \* **from** persons where salary > from and salary < to;

## Control structures

- ❑ imperative flow control
  - conditional statement
    - if** <condition> **then** <PL/SQL statement> **else** <PL/SQL statement> **end if**;
  - loops
    - for** <index variable> **in** <range> **loop** <PL/SQL statement> **end loop**;
  - while-loop, exit-when
- ❑ processing of a cursor
  - opening a cursor
    - open** current-book;
    - open** average-earner(1000, 2000);
  - processing of a result set
    - special loop construct:
      - for** myBook **in** current-book **loop** <PL/SQL statement> **end loop**;

## Procedures and functions

- ❑ In PL/SQL it is also possible to declare procedures and functions.
- ❑ A procedure is a block provided with a name and a parameter list.
- ❑ A function always yields a result with the aid of the command **return**.
- ❑ example

```
function totalSalary(int from, int to) return int is  
begin  
    declare p Personen%rowtype;  
    int total;  
    open average-earner(from, to);  
    ...  
    return total;  
end;
```

- ❑ The parameters of procedures and functions can be provided with one of the following three options: **in**, **out**, **in out**

```
procedure work(par1 in type1, par2 out type2, par3 in out type3) is  
<PL/SQL statements with assignments to the in out and out parameters>
```