8.4 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

```plsql
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
    ```sql
cursor current-book is select * from books;
```
  - parameterized cursor
    ```sql
cursor average-earner(from int, to int) is
      select * from persons where salary > from and salary < to;
```
Control structures

- imperative flow control
  - conditional statement
    \[
    \text{if} \ <\text{condition}> \ \text{then} \ <\text{PL/SQL statement}> \ \text{else} \ <\text{PL/SQL statement}> \ \text{end if};
    \]
  - loops
    \[
    \text{for} \ <\text{index variable}> \ \text{in} \ <\text{range}> \ \text{loop} \ <\text{PL/SQL statement}> \ \text{end loop};
    \]
- while-loop, exit-when

- processing of a cursor
  - opening a cursor
    \[
    \text{open} \ \text{current-book};
    \]
    \[
    \text{open} \ \text{average-earner}(1000, 2000);
    \]
  - processing of a result set
    special loop construct:
    \[
    \text{for} \ \text{myBook} \ \text{in} \ \text{current-book} \ \text{loop} \ <\text{PL/SQL statement}> \ \text{end loop};
    \]
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

  ```plsql
  function totalSalary(int from, int to) return int is
  begin
    declare p Persons%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`

  ```plsql
  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>
  ```
Stored Procedures

- With the command **create**, functions and procedures can be stored in the DBMS in a translated form and called on request.
- advantage: no anew translation of the query necessary
- Declaration is done according to the aforementioned pattern.
- cursor variables
  - Frequently, it is favorable to transmit the results of a stored procedure through cursor variables to the calling PL/SQL program.
- A cursor is a reference to a list of data records.
- two types of cursor variables
  - strong type
    ```
    type personCursorType is ref cursor persons%rowtype;
    ```
  - weak type
    ```
    type allCursorType is ref cursor;
    ```
- variable declaration as usual
- At the time of its declaration the cursor variable does not have a relationship to a query.
Bindung of a cursor variable to queries

When opening a cursor, the variable is bound to a query.

```sql
open personCursor for select * from persons where salary > 2000
```

usual application

- opening of a cursor variable in the stored procedure/function
- handing over of the cursor to the AP, which processes the records

Despite many advantages of cursor variables, there are still many limitations:

- A cursor variable may not be opened in the update mode.
- Type `ref cursor` is only known in PL/SQL but not in SQL.

stored functions in SQL

- Stored functions can be declared and called in SQL with the following limitations:
  - The functions do not contain grouping operations.
  - All data types of the input and of the output must be known in the DBMS.
- example for the declaration of a stored function

```sql
create function simple (x in int) return int as begin return x/101; end simple;
```

- example of an SQL query using this function

```sql
select Name, simple(salary) from Persons;
```
example: program segment which yields the information about the employee with the highest salary

```sql
declare
    lname   employee.lastname%type;
    fname   employee.firstname%type;
    addr    employee.address%type;
    esalary employee.salary%type;
begin
    select lastname, firstname, address, salary
    into   lname, fname, addr, esalary
    from   employee
    where  salary = (select max(salary) from employee);
    dbms_output.put_line(lname, fname, addr, esalary);
exception
    when others then
        dbms_output.put_line(“Error detected!”);
end;
```
example: program segment which increases the salary of employees, whose salary is below the average salary, by 10% and which outputs the average salary, if it exceeds 30000 Dollar after the previous update.

```sql
declare
    avg-salary number;
begin
    select avg(salary) into avg-salary from employee;
    update employee
        set salary = salary * 1.1
        where salary < avg-salary;
    select avg(salary) into avg-salary from employee;
    if avg-salary > 30000 then
        dbms_output.put_line("Average salary is " || avg-salary);
    end if;
    commit;
exception
    when others then dbms_output.put_line("Update error!"); rollback;
end;
```
example: Calculate the salary increases depending on the current salaries of employees.

```sql
declare
cursor EmpCursor is select salary from employee for update of salary;
EmpSal employee.salary%type;
begin
    open EmpCursor;
    fetch EmpCursor into EmpSal;
    while not EmpCursor%notfound
        if EmpSal > 60000 then
            update employee set salary = salary * 1.1
            where current of EmpCursor;
        elsif EmpSal > 50000 then
            update employee set salary = salary * 1.15
            where current of EmpCursor;
        else
            update employee set salary = salary * 1.20
            where current of EmpCursor;
        end if;
        fetch EmpCursor into EmpSal;
    end loop;
end;
```
9. Data Integrity

9.1 Introduction

Integrity constraints

- **Integrity constraints (ICs)**
  - are an instrument to ensure that changes of the database by authorized users do not lead to a loss of data consistency.
  - serve for a restriction of the database states to those ones that really exist in the real world.
  - are semantically derivable from the posed data model and can therefore already be specified during the creation of the schema.

- **advantages**
  - Consistency conditions are specified only once.
  - Consistency conditions are checked automatically by the DBMS.
  - APs do not need to care about a check of the conditions.

- **Static integrity constraints** relate to restrictions of the possible database states, **dynamic integrity constraints** to restrictions of the possible database state transitions.
– example for static IC: German professors may only have ranks C2, C3 or C4.
– example for dynamic IC: Professors may only be promoted, but not demoted. Their rank may not be set from C4 to C3, for example.

Examples for ICs

- No customer name may appear more than once in the relation “customers”.
- Each customer name in the relation “orders” must appear in the relation “customer”.
- No account of a customer is allowed to be less than USD -100.00.
- The account of Mr White may not be overchecked.
- Only those products can be ordered for which at least one supplier exists.
- The bread price may not be increased.