- binding to an SQL command

```sql
#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
#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.
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 encapsulation, 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
    ```plsql
    cursor current-book is select * from books;
    ```
  - parameterized cursor
    ```plsql
    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

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.

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.

```
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.
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.
9.2 Static Integrity Constraints

**Type integrity / domain constraints**

- Each attribute value must be atomic.
- Determination of upper and lower bounds of attribute values
  - SQL92 allows the explicit definition of domains (not implemented in Oracle)
- Enumeration of values
  - Such enumeration types can be modeled by an own relation with one attribute.
- Null values
  - The construct **not null** attached to an attribute declaration can be used to achieve that the value of the corresponding attribute exists for all tuples.

**Tuple constraints**

- Restriction of the values which a tuple can take with respect to its different attributes
- Example: In a relation *occupancy*, where a tuple stores the reservation of a seat from a train station *X* to a station *Y*, *X* should be unequal to *Y*.
- In SQL a tuple constraint can be defined by the **check** clause during the specification of the schema.
Relation constraints / key constraints

- refers to the set of all tuples of a relation
- **key integrity** by means of primary key and candidate key
  - no two tuples with equal keys
  - in SQL declaration of this constraint with *unique* and *primary key* during the specification of the schema
  - No primary key or candidate key can have the value *null*.

- aggregation constraints
  - example: An aggregate value shall be above or below a predefined bound.

- Recursive constraints
  - example: In a relation *TrainConnection* each station should be connected with each other station.

Referential constraints

- refer to several relations
9.3 Referential Integrity

Features

- **Referential ICs** are conditions for relations which especially model a relationship.
- They are used to maintain the consistency between tuples of two relations.
- A referential IC requires that a tuple of a relation that refers to another relation has to refer to an **existing** tuple of that relation.
- Let $R$ and $S$ be relations with the schemas $R$ and $S$. Let $K \subseteq R$ be a primary key of $R$. Then $F \subseteq S$ is called **foreign key** of $S$, if for each data record $s$ of relation $S$ one of the following conditions holds:
  - $\forall A \in F : s[A] = \text{null}$.
  - There is a data record $r$ of $R$ such that $s[F] = r[K]$.
- example: relations *customers*, *products*, *orders*  
  
  *(m:n*-relationship between *customers* and *products)*
- possible problems if referential integrity is not fulfilled:
  - Customer orders products which do not exist
  - Products can be ordered by a customer who does not exist.
Maintenance of referential integrity

- In a relation, which models a relationship, we should ensure that values are assigned to the foreign keys.
- Formulation of explicit integrity constraints in the query language, small extensions of the query language
- Formulation in relational algebra:
  - relation $R$ with primary key $K$
  - relation $S$ with foreign key $F$ that references $K$
  \[
  \pi_F(S) \subseteq \pi_K(R)
  \]
- Allowed changes
  - insertion of a tuple $s$ into $S$ if $s[F] \in \pi_K(R)$
  - change of an attribute value of a tuple $s$ if $s'[F] \in \pi_K(R)$ (let $s'$ be the new value of tuple $s$)
  - change of $r[K]$ of a tuple $r$ if $\sigma_{F=r[K]}(S) = \emptyset$
  - deletion of a tuple $r$ of $R$ if $\sigma_{F=r[K]}(S) = \emptyset$
Referential integrity in the ER model

- $N$-ary relationships contain the keys of all participating entities, i.e., all keys are foreign keys.
- A weak entity set must include the primary key of the entity set on which the weak entity set depends. That is, the weak entity set contains the primary key of the strong entity set as foreign key.

Integrity constraints in SQL

- domain constraints
  - **not null**
    + null value as attribute value not permitted
    + example: FamilyName `character(20)` **not null**
  - **default** `<value>`
    + initialization of an attribute value with the value `<value>` which must be a constant or the result of the evaluation of an expression. The value `<value>` must be compatible with the domain of the attribute.
    + example: NoOfChildren `smallint` default 0
  - **default** **null**: optional, default value
Tuple constraint

- **check** clause as an integrity constraint that can be represented as a simple Boolean expression

- typical applications: range queries, realization of enumeration types, handling of null values

- example for range query: limitation that students may study for a maximum of 8 semesters.
  
  check sem between 1 and 8

- example for enumeration type:
  
  check rank in ("C2", "C3", "C4")

- handling of null values
  
  - Null value fits for **check** or foreign key constraints to arbitrary values. If a part of a foreign key is **null**, the key relates to all data which have the same defined part.
  
  - If a foreign key is to be totally **null** or totally defined, this must be explicitly expressed by a **check** clause.
- example
  
  check $((S_1 \text{ is null and } S_2 \text{ is null and } ...) \text{ or }$
  
  $(S_1 \text{ is not null and } S_2 \text{ is not null and } ...))$

- The constraint
  
  check sem between 1 and 8

  is also fulfilled for a null value as semester.

Relation constraints/ key constraints

- **primary key** clause
  - automatically specified as **not null**
  - example:
    
    FirstName *character*(20) **not null**,
    LastName *character*(20) **not null**,
    Salary *numeric*(11,2),
    **primary key** (LastName, FirstName)
- candidate key: **unique**
  - example:
    
    FirstName `character(20)` **not null**,  
    LastName `character(20)` **not null**,  
    **unique** (LastName, FirstName)

Referential Integrity

- foreign key: **foreign key**
- **unique foreign key** models a 1:1-relationship
- Foreign key must be marked as primary key in the referenced relation.
- example:
  
  ```sql
  create table customers(cid integer, ..., primary key(cid))
  create table orders(..., cid integer, ..., foreign key(cid) references customers(cid))
  ```

- standard behavior when violating a referential integrity constraint: rejection of the action that triggered the violation
- If a foreign key is specified with a **cascade** clause, changes of the respective primary key are propagated.
example

+ create table lectures ( 
  ...
  foreign key held_by integer references professors(pers-id)
  on delete cascade
  on update cascade);

+ The deletion of a tuple with the key pers-id in professors has an effect on lectures. Also there the tuple is then deleted. This enables the realization of a dependent relationship.

+ A change of the attribute pers-id in professors is propagated to the foreign key held_by in lectures.

example

+ create table lectures (...,
  foreign key held_by integer references professors(pers-id)
  on delete set null
  on update set null);

+ The attribute value held_by in lectures is set to null, if the referenced tuple with the key pers-id in professors is deleted.

+ The attribute value held_by in lectures is set to null, if the referenced tuple with the key pers-id in professors is changed.
9.4 Management of Integrity Constraints

Adding ICs

- `alter table orders`  
  `add constraint limit check (price * number < 10000)`

- `alter table customers`  
  `add constraint unique_name unique cname`

- `alter table customers`  
  `drop constraint unique_name`

Enabling and disabling ICs

- `alter table customers enable constraint unique_name`
- `alter table customers disable constraint unique_name`
9.5 Complex Integrity Constraints

Assertions

- An **assertion** is a predicate that expresses a condition which is to be always satisfied by a database. Domain constraints and referential ICs are special kinds of assertions. There are also conditions which cannot be expressed with these two kinds like conditions with respect to several relations.

- **syntax:** `create assertion <assertion name> check <condition>`

- **example:** There must be at least four professors in order to maintain teaching.

  `create assertion AlwaysFourProfessors
  check (4 <= (select count(*) from professors))`

- The DBMS tests an assertion for validity. If the assertion is valid, future modifications of the database are only allowed if the assertion is not violated.

- Complex assertions can lead to an overhead.

- Assertions should be used with care.