# Chapitre IV

# Introduction à PL/SQL

## 4 – Introduction à PL/SQL



## 4.1 - Mécanismes de PL / SQL

- Limites de la programmation déclarative
- PL / SQL = Programming Language for SQL
- Permet des interactions avec une base SQL
- Language de programmation qui inclut des ordres SQL

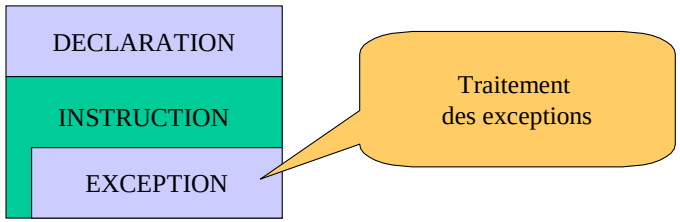
## Exemple introductif

```
DECLARE
   qty_on_hand  NUMBER(5);
BEGIN
   SELECT quantity INTO qty_on_hand FROM inventory
      WHERE product = 'TENNIS RACKET'
      FOR UPDATE OF quantity;
   IF qty_on_hand > 0 THEN  -- check quantity
      UPDATE inventory SET quantity = quantity - 1
         WHERE product = 'TENNIS RACKET';
      INSERT INTO purchase_record
         VALUES ('Tennis racket purchased', SYSDATE);
   ELSE
      INSERT INTO purchase_record
         VALUES ('Out of tennis rackets', SYSDATE);
   END IF;
   COMMIT;
END;
```

Récupération des valeurs

Validation

## Structure récursive de blocs




## Déclarations et affectations

Déclarations

```
part_no  NUMBER(4);
in_stock BOOLEAN;

credit_limit CONSTANT REAL := 5000.00;
```

```
pi     REAL := 3.14159;
radius REAL := 1;
area   REAL := pi * radius**2;

birthday DATE;
birthday DATE := NULL;
```

Affectations

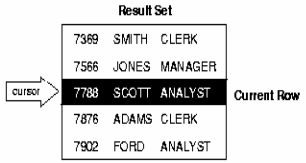
```
tax := price * tax_rate;
valid_id := FALSE;
bonus := current_salary * 0.10;
wages := gross_pay(emp_id, st_hrs, ot_hrs) - deductions;

SELECT sal * 0.10 INTO bonus FROM emp WHERE empno = emp_id;
```

## Curseurs

- Un curseur est une sorte de pointeur qui permet d'exploiter les lignes d'une table les unes après les autres.

```
DECLARE
   CURSOR c1 IS
      SELECT empno, ename, job FROM emp WHERE deptno = 20;
```




## Boucles et curseurs

```
DECLARE
   CURSOR c1 IS
      SELECT ename, sal, hiredate, deptno FROM emp;
   ...
BEGIN
   FOR emp_rec IN c1 LOOP
      ...
      salary_total :=  salary_total + emp_rec.sal;
   END LOOP;
```

```
OPEN ...
FETCH ...
CLOSE ...
```

## Types SQL et Types PL/SQL

```
%TYPE
my_title books.title%type

%ROWTYPE
DECLARE
  Dep_rec dept%ROWTYPE

  my_deptno := dept_rec.deptno
  curseur : c1%ROWTYPE
```

Affectation de structure

## Structure conditionnelle

```
-- available online in file 'examp2'
DECLARE
   acct_balance NUMBER(11,2);
   acct         CONSTANT NUMBER(4) := 3;
   debit_amt    CONSTANT NUMBER(5,2) := 500.00;
BEGIN
   SELECT bal INTO acct_balance FROM accounts
      WHERE account_id = acct
      FOR UPDATE OF bal;
   IF acct_balance >= debit_amt THEN
      UPDATE accounts SET bal = bal - debit_amt
         WHERE account_id = acct;
   ELSE
      INSERT INTO temp VALUES
         (acct, acct_balance, 'Insufficient funds');
            -- insert account, current balance, and message
   END IF;
   COMMIT;
END;
```

## Structure de choix multiples

```
-- This CASE statement performs different actions based
-- on a set of conditional tests.
CASE
  WHEN shape = 'square' THEN area := side * side;
  WHEN shape = 'circle' THEN
    BEGIN
      area := pi * (radius * radius);
      DBMS_OUTPUT.PUT_LINE('Value is not exact because pi is
irrational.');
    END;
  WHEN shape = 'rectangle' THEN area := length * width;
  ELSE
    BEGIN
      DBMS_OUTPUT.PUT_LINE('No formula to calculate area of a' ||
shape);
      RAISE PROGRAM_ERROR;
    END;
END CASE;
```

## Structure itérative

```
-- available online in file 'examp3'
DECLARE
   salary         emp.sal%TYPE := 0;
   mgr_num        emp.mgr%TYPE;
   last_name      emp.ename%TYPE;
   starting_empno emp.empno%TYPE := 7499;
BEGIN
    LOOP
   SELECT mgr INTO mgr_num FROM emp
      WHERE empno = starting_empno;
   WHILE salary <= 2500 LOOP
      SELECT sal, mgr, ename INTO salary, mgr_num, last_name
         FROM emp WHERE empno = mgr_num;
   END LOOP;
   INSERT INTO temp VALUES (NULL, salary, last_name);
   COMMIT;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      INSERT INTO temp VALUES (NULL, NULL, 'Not found');
      COMMIT;
END;
```

## Sortie **EXIT WHEN**

```
LOOP
   ...
   total := total + salary;
   EXIT WHEN total > 25000;  -- exit loop if condition is true
END LOOP;
-- control resumes here
```

## Traitement des exceptions

- Déclaration
  - **DECLARE ….**
  - **TOTO EXCEPTION**
- USAGE
  - **BEGIN ….**
  - **RAISE TOTO;**
- EXCEPTION
  - **EXCEPTION**
  - **WHEN TOTO THEN….**

## Transactions

- COMMIT
- ROLLBACK
- SAVEPOINT

## Procédure

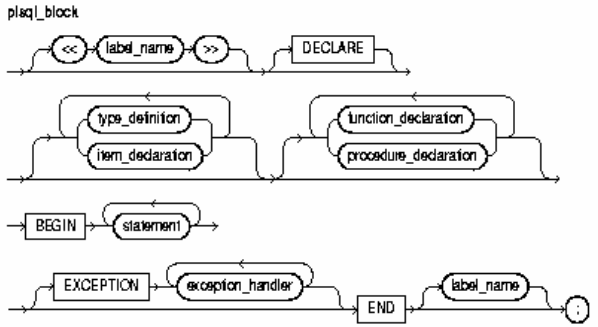
```
PROCEDURE award_bonus (emp_id NUMBER) IS
   bonus        REAL;
   comm_missing EXCEPTION;
BEGIN  -- executable part starts here
   SELECT comm * 0.15 INTO bonus FROM emp WHERE empno = emp_id;
   IF bonus IS NULL THEN
      RAISE comm_missing;
   ELSE
      UPDATE payroll SET pay = pay + bonus WHERE empno = emp_id;
   END IF;
EXCEPTION  -- exception-handling part starts here
   WHEN comm_missing THEN
      ...
END award_bonus;
```

## Package

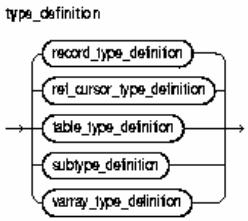
```
CREATE PACKAGE emp_actions AS  -- package specification
   PROCEDURE hire_employee (empno NUMBER, ename CHAR, ...);
   PROCEDURE fire_employee (emp_id NUMBER);
END emp_actions;

CREATE PACKAGE BODY emp_actions AS  -- package body
   PROCEDURE hire_employee (empno NUMBER, ename CHAR, ...) IS
   BEGIN
      INSERT INTO emp VALUES (empno, ename, ...);
   END hire_employee;
   PROCEDURE fire_employee (emp_id NUMBER) IS
   BEGIN
      DELETE FROM emp WHERE empno = emp_id;
   END fire_employee;
END emp_actions;
```

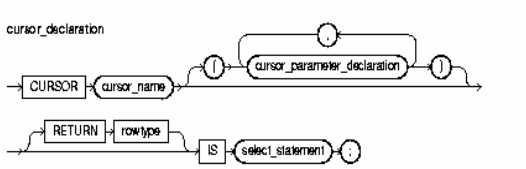
## 4.2 - Grammaire PL/SQL

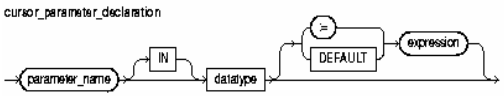



## Déclaration




## Déclaration de curseurs

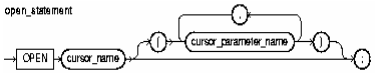


## Exemple

```
CURSOR c1 IS SELECT empno, ename, job, sal FROM emp
   WHERE sal > 2000;
CURSOR c2 RETURN dept%ROWTYPE IS
   SELECT * FROM dept WHERE deptno = 10;
CURSOR c3 (start_date DATE) IS
   SELECT empno, sal FROM emp WHERE hiredate > start_date;
```

## Open




```
CURSOR parts_cur IS SELECT part_num, part_price FROM parts;

OPEN parts_cur;

CURSOR emp_cur(my_ename VARCHAR2, my_comm NUMBER DEFAULT 0)
   IS SELECT * FROM emp WHERE ...

OPEN emp_cur('LEE');
OPEN emp_cur('BLAKE', 300);
OPEN emp_cur(employee_name, 150);
```

## Exemple curseur

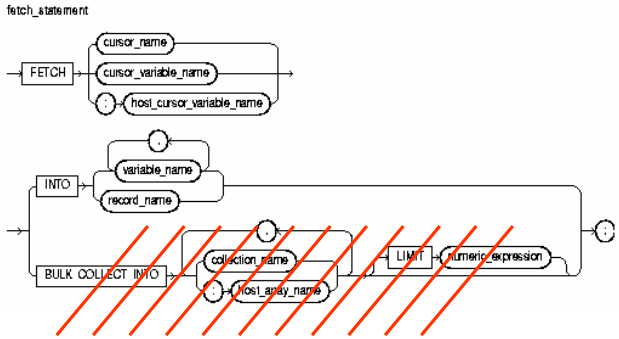
```
DECLARE
   CURSOR c1 IS
      SELECT ename, sal, hiredate, job FROM emp;
   emp_rec c1%ROWTYPE;  -- declare record variable that represents
                        -- a row fetched from the emp table

FETCH c1 INTO emp_rec;
```

emp_rec

| | |
|---|---|
| emp_rec.ename | JAMES |
| emp_rec.sal | 950.00 |
| emp_rec.hiredate | 03-DEC-95 |
| emp_rec.job | CLERK |

## FETCH

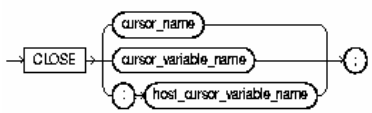


```
DECLARE
   my_sal NUMBER(7,2);
   n      INTEGER(2) := 2;
   CURSOR emp_cur IS SELECT  n*sal FROM emp;
BEGIN
   OPEN emp_cur;  --  n equals 2 here
   LOOP
      FETCH emp_cur INTO my_sal;
      EXIT WHEN emp_cur%NOTFOUND;
         -- process the data
       n :=  n + 1;  -- does not affect next FETCH; sal will be
multiplied by 2
   END LOOP;
```

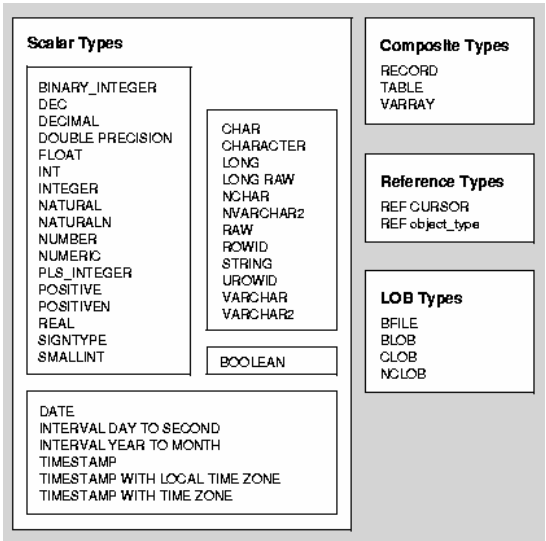
```
DECLARE
   TYPE EmpCurTyp IS REF CURSOR RETURN emp%ROWTYPE;
   emp_cv  EmpCurTyp;
   emp_rec emp%ROWTYPE;
BEGIN
   LOOP
      FETCH emp_cv INTO emp_rec;
      EXIT WHEN emp_cv%NOTFOUND;
      ...
   END LOOP;
END;
```

# CLOSE

close_statement




```
LOOP
   FETCH emp_cv INTO emp_rec;
   EXIT WHEN emp_cv%NOTFOUND;
   ...  -- process data record
END LOOP;
/* Close cursor variable. */
CLOSE emp_cv;
```

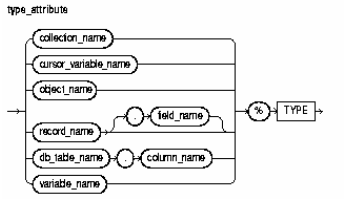


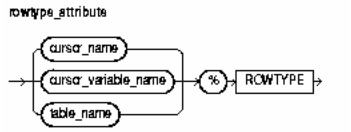

# Sous-type

```
SUBTYPE subtype_name IS base_type[(constraint)] [NOT NULL];
```

```
DECLARE
   SUBTYPE BirthDate IS DATE NOT NULL;  -- based on DATE type
   SUBTYPE Counter IS NATURAL;          -- based on NATURAL subtype
   TYPE NameList IS TABLE OF VARCHAR2(10);
   SUBTYPE DutyRoster IS NameList;      -- based on TABLE type
   TYPE TimeRec IS RECORD (minutes INTEGER, hours INTEGER);
   SUBTYPE FinishTime IS TimeRec;       -- based on RECORD type
   SUBTYPE ID_Num IS emp.empno%TYPE;    -- based on column type
```

## %TYPE et %ROWTYPE



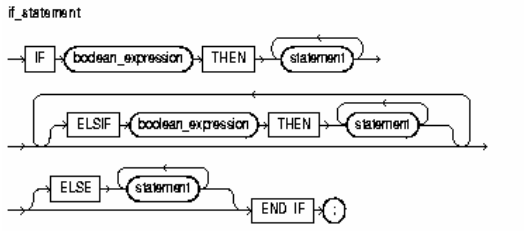




## Exemples

```
DECLARE
   emp_rec   emp%ROWTYPE;
   CURSOR c1 IS SELECT deptno, dname, loc FROM dept;
   dept_rec  c1%ROWTYPE;
```

```
DECLARE
   emp_rec  emp%ROWTYPE;
   ...
BEGIN
   SELECT * INTO emp_rec FROM emp WHERE empno = my_empno;
   IF (emp_rec.deptno = 20) AND (emp_rec.sal > 2000) THEN
      ...
   END IF;
END;
```

## IF

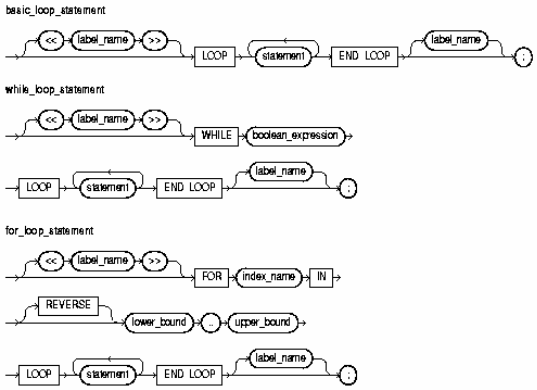



## Exemple

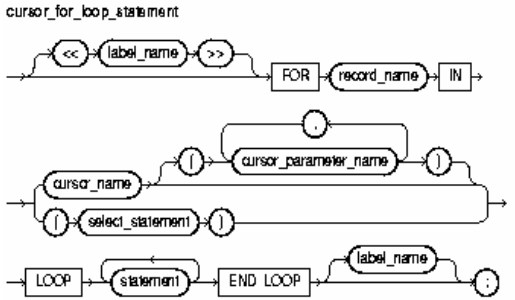
```
IF score < 70 THEN
   fail := fail + 1;
   INSERT INTO grades VALUES (student_id, 'Failed');
ELSE
   pass := pass + 1;
   INSERT INTO grades VALUES (student_id, 'Passed');
END IF;
```

```
IF shoe_count < 20 THEN
   order_quantity := 50;
ELSIF shoe_count < 30 THEN
   order_quantity := 20;
ELSE
   order_quantity := 10;
END IF;
```

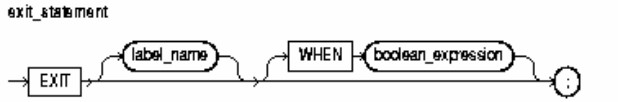
## LOOP



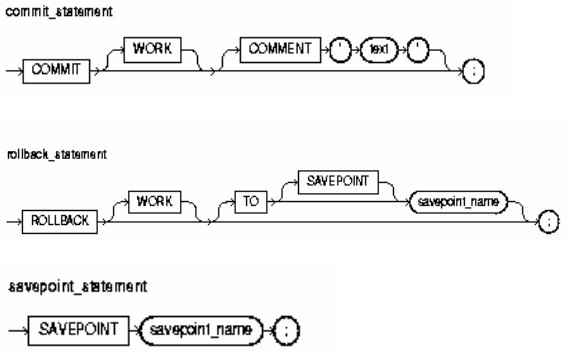

## Boucle avec curseur




## Sortie de boucle



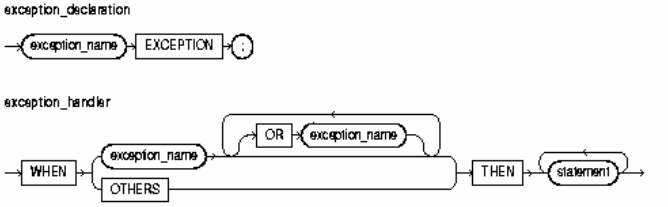

## COMMIT - ROLLBACK



## RAISE


```
IF quantity_on_hand = 0 THEN
   RAISE out_of_stock;
END IF;
```

## Sous-bloc EXCEPTION



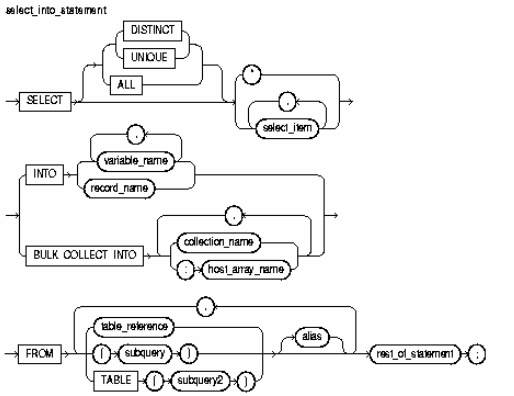

## Exemples

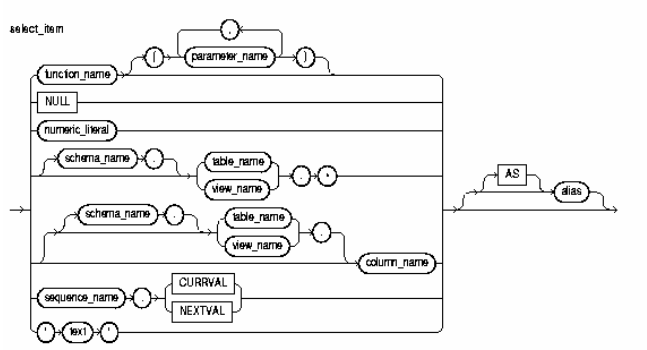
```
EXCEPTION
   WHEN invalid_number OR STANDARD.INVALID_NUMBER THEN ...
```

---

```
DECLARE
   bad_emp_id  EXCEPTION;
   bad_acct_no EXCEPTION;
   ...
BEGIN
   ...
EXCEPTION
   WHEN bad_emp_id OR bad_acct_no THEN  -- user-defined
      ROLLBACK;
   WHEN ZERO_DIVIDE THEN  -- predefined
      INSERT INTO inventory VALUES (part_number, quantity);
      COMMIT;
END;
```

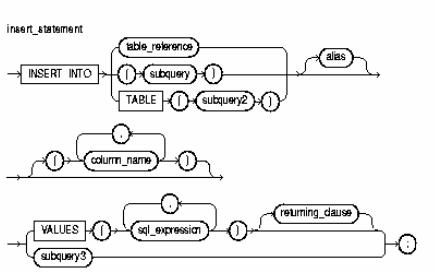
## SELECT… INTO...






```
SELECT ename, job, sal INTO my_ename, my_job, my_sal FROM emp
   WHERE empno = my_empno;
```

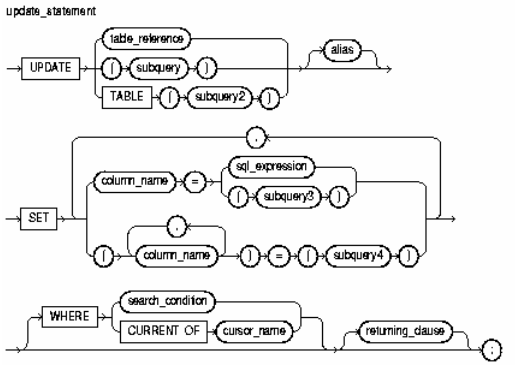
## INSERT




## Exemples

```
INSERT INTO bonus SELECT ename, job, sal, comm FROM emp
   WHERE comm > sal * 0.25;
...
INSERT INTO emp (empno, ename, job, sal, comm, deptno)
   VALUES (4160, 'STURDEVIN', 'SECURITY GUARD', 2045, NULL, 30);
...
INSERT INTO dept
   VALUES (my_deptno, UPPER(my_dname), 'CHICAGO');
```

## UPDATE



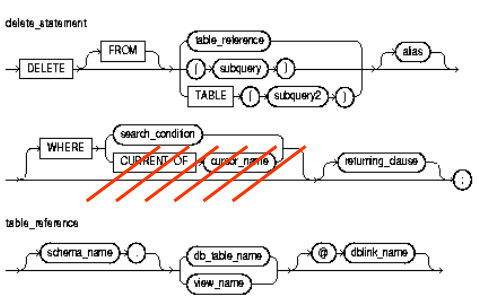
## Exemple

```
UPDATE inventory inv  -- alias
   SET (item_id, price) =
      (SELECT item_num, item_price FROM item_table
         WHERE item_name = inv.item_name);
```

```
UPDATE emp SET job = 'ANALYST', sal = sal * 1.15
   WHERE ename = 'FORD';
```

```
UPDATE emp SET sal = sal + 500 WHERE ename = 'MILLER'
   RETURNING sal, ename INTO my_sal, my_ename;
```

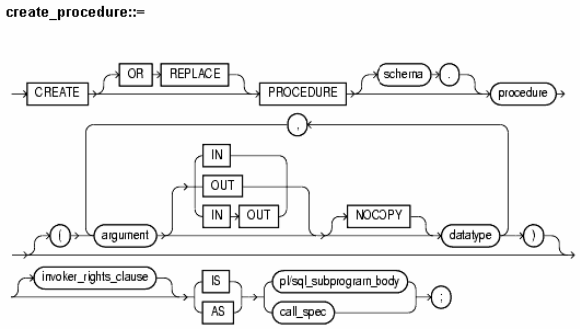
## DELETE



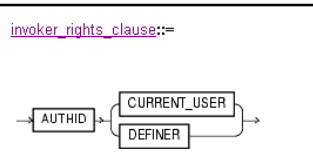

## 4.3 - Sous-programmes et packages

- Procédure
- Fonction
- Package

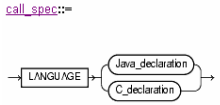
## CREATE PROCEDURE



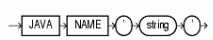
invoker_rights_clause::=



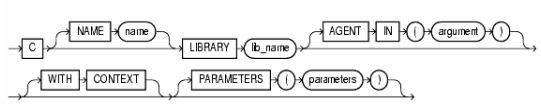

call_spec::=




**Java_declaration::=**




**C_declaration::=**



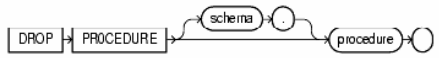

# EXEMPLES

```
CREATE PROCEDURE oe.credit
   (acc_no IN NUMBER, amount IN NUMBER) AS
   BEGIN
      UPDATE accounts
      SET balance = balance + amount
      WHERE account_id = acc_no;
   END;
```

```
CREATE PROCEDURE find_root
   ( x IN REAL )
   IS LANGUAGE C
      NAME "c_find_root"
      LIBRARY c_utils
      PARAMETERS ( x BY REFERENCE );
```

# DROP PROCEDURE

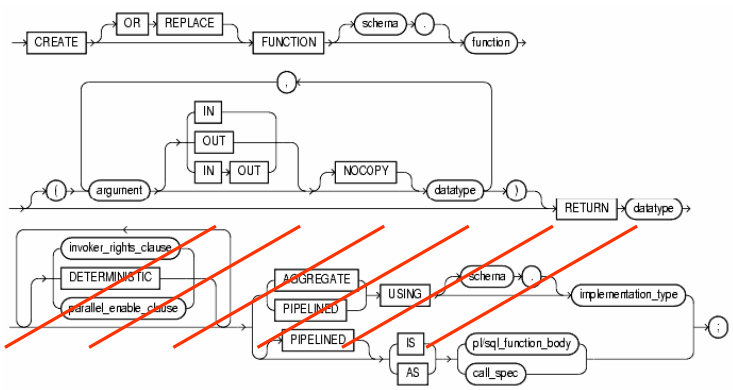
**drop_procedure::=**




```
DROP PROCEDURE oe.credit;
```

# CREATE FUNCTION

**create_function::=**



# Exemples

```
CREATE FUNCTION get_bal(acc_no IN NUMBER)
   RETURN NUMBER
   IS acc_bal NUMBER(11,2);
   BEGIN
      SELECT order_total
      INTO acc_bal
      FROM orders
      WHERE customer_id = acc_no;
      RETURN(acc_bal);
    END;
```

```
SELECT get_bal(165) FROM DUAL;

GET_BAL(165)
------------
        2519
```

```
CREATE FUNCTION get_val

( x_val IN NUMBER,
y_val IN NUMBER,
image IN LONG RAW )
RETURN BINARY_INTEGER AS LANGUAGE C
   NAME "c_get_val"
   LIBRARY c_utils
   PARAMETERS (...);
```

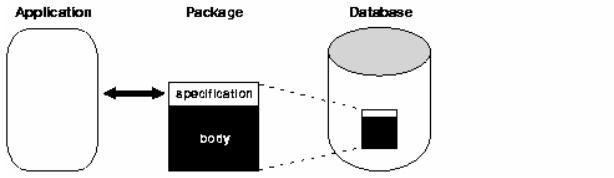
```
CREATE OR REPLACE FUNCTION text_length(a clob)
   RETURN NUMBER              IS
BEGIN
  RETURN DBMS_LOB.GETLENGTH(a);
END;
```

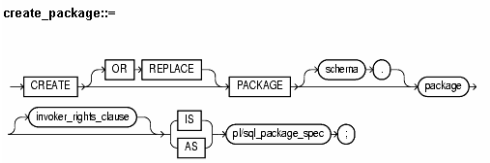
```
CREATE FUNCTION SecondMax (input NUMBER) RETURN NUMBER
    PARALLEL_ENABLE AGGREGATE USING SecondMaxImpl;
```

```
SELECT SecondMax(salary), department_id
      FROM employees
      GROUP BY department_id
      HAVING SecondMax(salary) > 9000;

SECONDMAX(SALARY) DEPARTMENT_ID
----------------- -------------
            13500            80
            17000            90
```

# CREATE PACKAGE




# Exemple

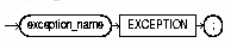
```
CREATE PACKAGE emp_mgmt AS
   FUNCTION hire (last_name varchar2(20), job_id varchar2(10),
      manager_id NUMBER(6), salary NUMBER(8,2),
      commission_pct NUMBER(2,2), department_id NUMBER(4)
      RETURN NUMBER;
   FUNCTION create_dept(department_id NUMBER(4), location NUMBER(4))
      RETURN NUMBER;
   PROCEDURE remove_emp(employee_id NUMBER(6));
   PROCEDURE remove_dept(department_id NUMBER(4));
   PROCEDURE increase_sal(employee_id NUMBER(4),
      salary_incr NUMBER);
   PROCEDURE increase_comm(employee_id NUMBER(4), comm_incr NUMBER);
      no_comm EXCEPTION;
      no_sal EXCEPTION;
   END emp_mgmt;
```

```
CREATE OR REPLACE PACKAGE emp_actions AS  -- spec
   TYPE EmpRecTyp IS RECORD (emp_id INT, salary REAL);
   CURSOR desc_salary RETURN EmpRecTyp;
   PROCEDURE hire_employee (
      ename  VARCHAR2,
      job    VARCHAR2,
      mgr    NUMBER,
      sal    NUMBER,
      comm   NUMBER,
      deptno NUMBER);
   PROCEDURE fire_employee (emp_id NUMBER);
END emp_actions;

CREATE OR REPLACE PACKAGE BODY emp_actions AS  -- body
   CURSOR desc_salary RETURN EmpRecTyp IS
      SELECT empno, sal FROM emp ORDER BY sal DESC;
   PROCEDURE hire_employee (
      ename  VARCHAR2,
      job    VARCHAR2,
      mgr    NUMBER,
      sal    NUMBER,
      comm   NUMBER,
      deptno NUMBER) IS
   BEGIN
      INSERT INTO emp VALUES (empno_seq.NEXTVAL, ename, job,
         mgr, SYSDATE, sal, comm, deptno);
   END hire_employee;
```

# 4.4 - Traitements des exceptions

exception_declaration




exception_handler


```
DECLARE
   bad_emp_id  EXCEPTION;
   bad_acct_no EXCEPTION;
   ...
BEGIN
   ...
EXCEPTION
   WHEN bad_emp_id OR bad_acct_no THEN  -- user-defined
      ROLLBACK;
   WHEN ZERO_DIVIDE THEN  -- predefined
      INSERT INTO inventory VALUES (part_number, quantity);
      COMMIT;
END;
```

| Exception | Oracle Error | SQLCODE Value |
|---|---|---|
| ACCESS_INTO_NULL | ORA-06530 | -6530 |
| CASE_NOT_FOUND | ORA-06592 | -6592 |
| COLLECTION_IS_NULL | ORA-06531 | -6531 |
| CURSOR_ALREADY_OPEN | ORA-06511 | -6511 |
| DUP_VAL_ON_INDEX | ORA-00001 | -1 |
| INVALID_CURSOR | ORA-01001 | -1001 |
| INVALID_NUMBER | ORA-01722 | -1722 |
| LOGIN_DENIED | ORA-01017 | -1017 |
| NO_DATA_FOUND | ORA-01403 | +100 |
| NOT_LOGGED_ON | ORA-01012 | -1012 |
| PROGRAM_ERROR | ORA-06501 | -6501 |
| ROWTYPE_MISMATCH | ORA-06504 | -6504 |
| SELF_IS_NULL | ORA-30625 | -30625 |
| STORAGE_ERROR | ORA-06500 | -6500 |
| SUBSCRIPT_BEYOND_COUNT | ORA-06533 | -6533 |
| SUBSCRIPT_OUTSIDE_LIMIT | ORA-06532 | -6532 |
| SYS_INVALID_ROWID | ORA-01410 | -1410 |
| TIMEOUT_ON_RESOURCE | ORA-00051 | -51 |
| TOO_MANY_ROWS | ORA-01422 | -1422 |
| VALUE_ERROR | ORA-06502 | -6502 |
| ZERO_DIVIDE | ORA-01476 | -1476 |

# DECLARATION et PORTEE

```
DECLARE
   past_due EXCEPTION;
```

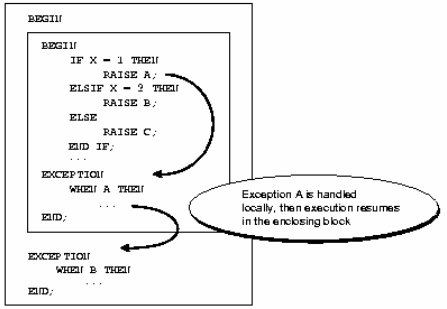
```
DECLARE
   past_due EXCEPTION;
   acct_num NUMBER;
BEGIN
   DECLARE  ---------- sub-block begins
      past_due EXCEPTION;  -- this declaration prevails
      acct_num NUMBER;
   BEGIN
      ...
      IF ... THEN
         RAISE past_due;  -- this is not handled
      END IF;
   END;  ------------- sub-block ends
EXCEPTION
   WHEN past_due THEN  -- does not handle RAISEd exception
      ...
END;
```

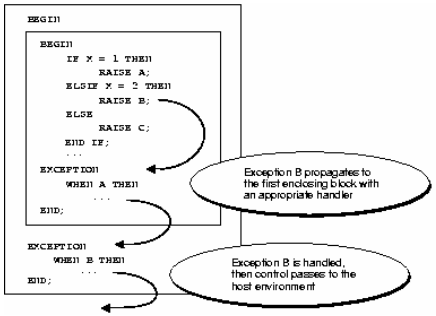
## Erreurs personnalisées

```
raise_application_error(error_number, message[, {TRUE | FALSE}]);
```

```
CREATE PROCEDURE raise_salary (emp_id NUMBER, amount NUMBER) AS
   curr_sal NUMBER;
BEGIN
   SELECT sal INTO curr_sal FROM emp WHERE empno = emp_id;
   IF curr_sal IS NULL THEN
      /* Issue user-defined error message. */
      raise_application_error(-20101, 'Salary is missing');
   ELSE
      UPDATE emp SET sal = curr_sal + amount WHERE empno = emp_id;
   END IF;
END raise_salary;
```

## Propagation des exceptions




```
DECLARE
   name   VARCHAR2(20);
   ans1   VARCHAR2(3);
   ans2   VARCHAR2(3);
   ans3   VARCHAR2(3);
   suffix NUMBER := 1;
BEGIN
   ...
   LOOP  -- could be FOR i IN 1..10 LOOP to allow ten tries
      BEGIN  -- sub-block begins
         SAVEPOINT start_transaction;  -- mark a savepoint
         /* Remove rows from a table of survey results. */
         DELETE FROM results WHERE answer1 = 'NO';
         /* Add a survey respondent's name and answers. */
         INSERT INTO results VALUES (name, ans1, ans2, ans3);
            -- raises DUP_VAL_ON_INDEX if two respondents
            -- have the same name
         COMMIT;
         EXIT;
      EXCEPTION
         WHEN DUP_VAL_ON_INDEX THEN
            ROLLBACK TO start_transaction;  -- undo changes
            suffix := suffix + 1;           -- try to fix problem
            name := name || TO_CHAR(suffix);
      END;  -- sub-block ends
   END LOOP;
END;
```

## 4.5 – Les Larges Objets

- 4.5.1 – Les CLOB
- 4.5.2 – Les BLOB
- 4.5.3 – Les BFILE

http://helyos.developpez.com/lob/

## Large Objects (LOB)

- LOB interne
  - CLOB (Character Large Object) : chaînes de caractères.
  - BLOB (Binary Large Object): données binaires
  - NCLOB (National Character Large Object) : les chaînes de caractères Unicode.
- LOB externe
  - BFILE (Binary File): pour les données stockées dans le système de fichier du système d'exploitation.

## LOB

- Le type LOB ressemble aux types LONG et LONG RAW.
- Différences entre LOB et LONG ou LONG RAW ?
  - La taille d'un LONG ne peut excéder 2Go alors que la taille d'un LOB peut monter jusqu'à 4Go.
  - En PL/SQL, pour récupérer une valeur de type LONG dans une variable, vous ne pourrez pas récupérer une valeur ayant une taille supérieure à 32760 Bytes (alors qu'une colonne de type LONG supporte jusqu'à 2Go).
  - Il n'est pas possible d'avoir une table avec plusieurs colonnes de type LONG ou LONG RAW, pour un LOB il n'existe pas cette limitation.
  - Il faut savoir que les accès à un LONG se font de manière séquentielle (vous êtes dans l'obligation de lire le LONG du début à la fin) alors que pour les LOB les accès se font de manière directe (d'où un gain de performances).
  - Il n'est pas possible de passer une valeur LONG à une fonction SQL, et en PL/SQL une variable LONG sera automatiquement convertie en VARCHAR2 (à moins que la taille du VARCHAR2 ne le permette pas, auquel cas la variable sera convertie en LONG).
  - Lors d'une requête SELECT l'intégralité du LONG est retournée alors que pour un LOB seul le pointeur sera retourné.

## Conversions de LONG ou un LONG RAW en CLOB ou en BLOB

```
CREATE TABLE t1(x INT, y LONG);
CREATE TABLE t2(x INT, y CLOB);
INSERT INTO t1 VALUES (1,
  rpad('*', 4000, '*'));
INSERT INTO t2
SELECT x, to_lob(y) FROM t1;
```

## 4.5.1 – Les CLOB

- Le type de données CLOB est un type de données qui va permettre de stocker d'importants volumes de **données de type "chaîne de caractères".** La taille maximale d'un CLOB est de 4Go.
- Il faut savoir que les CLOB conservent les mêmes règles transactionnelles que les types de données tels que VARCHAR2, NUMBER, etc. et qu'ils peuvent être restaurés sans aucune action complémentaire.

## Exemple SQL

```
create table t_test (id_clob number, texte clob);

INSERT INTO t_test VALUES (1,'Hello World');
INSERT INTO t_test VALUES (2,rpad('*',3200,'*'));
COMMIT;


SELECT * FROM t_test;
   ID_CLOB TEXTE
---------- --------------------------------------
         1 Hello World
         2 **********************************
```

## Exemple PL/SQL

```
CREATE OR REPLACE PROCEDURE insert_test (p_id
  NUMBER, p_text VARCHAR2)
IS
v_clob CLOB;

BEGIN
-- On insère la ligne avec un CLOB vide
INSERT INTO t_test VALUES (p_id, empty_clob())
  returning texte into v_clob;
-- On le remplit avec un contenu
DBMS_LOB.WRITE(v_clob, 1, length(p_text),p_text);
COMMIT;
END;
/
```

## Ou encore plus simple

```
CREATE OR REPLACE PROCEDURE
  insert_test (p_id NUMBER, p_text
  CLOB)
IS
BEGIN
INSERT INTO t_test VALUES (p_id,
  p_text);
COMMIT;
END;
/
```

## 4.5.2 – Les BLOB

- Le type de données BLOB va permettre de stocker d'important volume de **données de type binaire**. La taille maximale d'un BLOB ne peut excéder 4Go. Ce type de données nous permettra de stocker n'importe quel type de données dans la base (images, pdf, mp3, doc, etc.).
- Il faut savoir que les BLOB conservent les mêmes règles transactionnelles que les types de données tels que VARCHAR2, NUMBER, etc. et qu'ils peuvent être restaurés sans aucune action complémentaire.

## Création

```
CREATE TABLE t_blob (
   id number,
   image blob
   );


CREATE OR REPLACE PACKAGE sql_blob IS

   -- Procédure servant à ajouter un BLOB à notre table.
   -- p_id correspond à l'id
   -- p_name correspond au nom du fichier à inserer.
   PROCEDURE add_blob(p_id NUMBER, p_name VARCHAR2);

END sql_blob;
/
```

```
CREATE OR REPLACE PACKAGE BODY sql_blob IS

  PROCEDURE add_blob(p_id NUMBER, p_name VARCHAR2) IS
    v_blob  BLOB;
    v_bfile BFILE;
  BEGIN
    -- On insère la ligne avec un blob vide dont on récupère
   le pointeur
    INSERT INTO t_blob
    VALUES
      (p_id, empty_blob())
    RETURNING image INTO v_blob;
    -- On déclare un pointeur vers notre fichier
    v_bfile := bfilename(directory => 'BLOBDIR', filename =>
   p_name);
    -- On ouvre ce fichier
    dbms_lob.fileopen(v_bfile);
    -- On remplit l'emplacement du BLOB vide dans la table
   avec le contenu de notre fichier
    dbms_lob.loadfromfile(v_blob, v_bfile,
   dbms_lob.getlength(v_bfile));
    -- On ferme le fichier
    dbms_lob.fileclose(v_bfile);
  END;
END sql_blob;
/
```

## Lecture

```
PROCEDURE write_blob(p_id NUMBER, p_name VARCHAR2) IS
  v_file       utl_file.file_type;
  v_repertoire VARCHAR2(512) := 'c:\temp\';
  v_fichier    VARCHAR2(256) := p_name;
  v_buffer     RAW(32000);
  v_offset     PLS_INTEGER DEFAULT 1;
  v_taille     PLS_INTEGER;
  v_longueur   PLS_INTEGER;
  v_chunk      PLS_INTEGER;
  v_blob       BLOB;
BEGIN
 -- On récupère le BLOB
  SELECT image INTO v_blob FROM t_blob WHERE id = 1;
  -- On l'ouvre en lecture afin de pouvoir le parser plus facilement
  dbms_lob.OPEN(v_blob, dbms_lob.lob_readonly);
  -- On regarde la taille de Chunk idéale
  v_chunk := dbms_lob.getchunksize(v_blob);
  -- On regarde sa longueur
  v_longueur := dbms_lob.getlength(v_blob);
```

```
-- On crée le fichier sur le disque dur
  v_file     := utl_file.fopen(v_repertoire, v_fichier, 'w', 32767);
  -- On ecrit dans le fichier tant que l'on a pas fait tout le BLOB
  WHILE v_offset < v_longueur LOOP
    IF v_longueur - (v_offset - 1) > v_chunk THEN
      v_taille := v_chunk;
    ELSE
      v_taille := v_longueur - (v_offset - 1);
    END IF;
    v_buffer := NULL;
    -- On lit la partie du BLOB qui nous interesse
    dbms_lob.READ(v_blob, v_taille, v_offset, v_buffer);
    -- On écrit cette partie dans le fichier
    utl_file.put(file => v_file, buffer =>
   utl_raw.cast_to_varchar2(v_buffer));
    utl_file.fflush(file => v_file);
    v_offset := v_offset + v_taille;
  END LOOP;
  -- On ferme le BLOB
  dbms_lob.CLOSE(v_blob);
  -- On ferme le fichier
  utl_file.fclose(v_file);EXCEPTION
  WHEN OTHERS THEN
    IF dbms_lob.ISOPEN(v_blob) = 1 THEN
      dbms_lob.CLOSE(v_blob);
    END IF;
    IF utl_file.is_open(file => v_file) THEN
      utl_file.fclose(file => v_file);
    END IF;
END;
```

## 4.5.3 – Les BFILE

- Le type de données BFILE vous permet de stocker des objets de types binaires en dehors de la base de données.
- Le type BFILE est en fait un pointeur vers le fichier binaire (ce pointeur contient le path complet vers une fichier système).
- Les BFILE sont de types Read-only et ne peuvent donc être modifiés par le serveur. Leur taille, dépendante du système, ne pourra pas dépasser la taille de 2^32-1 Bytes.
- L'intégrité des données n'est plus assurée par Oracle mais par le système d'exploitation.
- Ne faisant pas partie de la base de données, les BFILE ne participent pas aux transactions, ne sont pas récupérables sans actions de sauvegarde complémentaire.
- Le nombre maximum de BFILES ouverts est déterminé par le paramètre SESSION_MAX_OPEN_FILES (qui lui aussi dépend du système d'exploitation).

## Création

```
CREATE TABLE t_bfile (id number,
  filename bfile);
```

- Ensuite voici la commande qui va permettre de rajouter une ligne avec un pointeur vers un fichier toto.txt (contenant hello world).

```
INSERT INTO t_bfile VALUES (1,
  bfilename('BLOBDIR', 'toto.txt'));
COMMIT;
```

## Affichage du contenu

```
CREATE OR REPLACE FUNCTION blob_to_char(p_file IN
   BFILE) RETURN VARCHAR2 AS
  v_raw   RAW(4000);
  v_bfile BFILE DEFAULT p_file;
BEGIN
  -- On ouvre notre fichier désigné par notre pointeur
  dbms_lob.fileopen(v_bfile);
  -- On récupère les 4000 premiers caractères
  v_raw := dbms_lob.substr(v_bfile, 4000, 1);
  -- On ferme notre fichier
  dbms_lob.fileclose(v_bfile);
  -- On convertit notre buffer en VARCHAR2
  RETURN utl_raw.cast_to_varchar2(v_raw);
END;
/
```

## Lecture

```
SQL> SELECT blob_to_char(filename)
  2  FROM t_bfile
  3  WHERE id=1;

BLOB_TO_CHAR(FILENAME)
--------------------------------
hello world
```

## Affichage Page Web

```
CREATE OR REPLACE PROCEDURE display_bfile(p_id NUMBER)
   IS
  v_amt   NUMBER DEFAULT 4096;
  v_off   NUMBER DEFAULT 1;
  v_raw   RAW(4096);
  v_bfile BFILE;
BEGIN
  -- On récupère le pointeur vers le fichier
  SELECT filename INTO v_bfile FROM t_bfile WHERE id =
   p_id;
  -- On ouvre le fichier
  dbms_lob.fileopen(v_bfile);
  -- On définit de manière arbitraire un mime/type
  owa_util.mime_header('image/gif');
  -- On affiche le contenu de notre fichier
```

## Suite

```
BEGIN
    LOOP
      dbms_lob.READ(v_bfile, v_amt, v_off, v_raw);
      htp.print(utl_raw.cast_to_varchar2(v_raw));
      v_off := v_off + v_amt;
      v_amt := 4096;
    END LOOP;
  EXCEPTION
    WHEN no_data_found THEN
      NULL;
  END;
  -- On ferme notre fichier
  dbms_lob.fileclose(v_bfile);
END;
/
```

## 4.6 – Impressions

- Initialement : écriture dans la base de données ou dans un fichier
  - Par exemple une table d'erreurs
- Sous l'exigence des utilisateurs
  - Création d'un package d'impression
  - DBMS_OUPUT

## DBMS_OUPUT

- Mise en service par la commande (SQL)
  - `SET SERVEROUTPUT ON`
  - `SET SERVEROUTPUT OFF`
- Mise en service par la commande (PL/SQL)
  - `DBMS_OUPUT.ENABLE`
  - `DBMS_OUPUT.DISABLE`

Buffer limité à 255 octets
Autrement (max 1 000 000 octets)
`SET SERVEROUTPUT ON SIZE 5000`

## Procédures

- `DBMS_OUPUT.NEW_LINE`
- `DBMS_OUPUT.PUT (data_to_display)`
- `DBMS_OUPUT.PUT_LINE (data_to_display)`

```
DECLARE
--Counter for the For Loop
     v_Counter NUMBER;
BEGIN
     FOR v_Counter IN 1..3 LOOP
-- This will cause two of each number to appear
  on same line as
-- PUT_LINE will flush PUT with it
          DBMS_OUTPUT.PUT(v_Counter);
          DBMS_OUTPUT.PUT_LINE(v_Counter);
     END LOOP;
--Demonstrate PUT with NEW_LINE
     DBMS_OUTPUT.PUT_LINE('We will now test with
  a newline character');
     FOR v_Counter IN 1..3 LOOP
          DBMS_OUTPUT.PUT(v_Counter);
          DBMS_OUTPUT.NEW_LINE;
     END LOOP;
END;
```

```
11
22
33
We will …
1
2
3
```

## Exceptions possibles

- ORU-10027 Buffer overflow
- ORU-10028 Line length overfow (limit of 255 characters per line)

## 4.7 – Conclusion

- Passage du déclaratif au procédural
- Permet de travailler dans les BD
- Large gamme d'opérateurs et de fonctions
- Multiples autres possibilités
- Liaisons JAVA, C++, etc.

- Langage propriétaire ????