

# PL / SQL



# PL / SQL

## Module 1 – Présentation de l'écosystème ORACLE



# Objectifs

- Découvrir Oracle
- Découvrir l'environnement logiciel
- Savoir créer un utilisateur Oracle fonctionnel
- Découvrir la notion de base de données Oracle



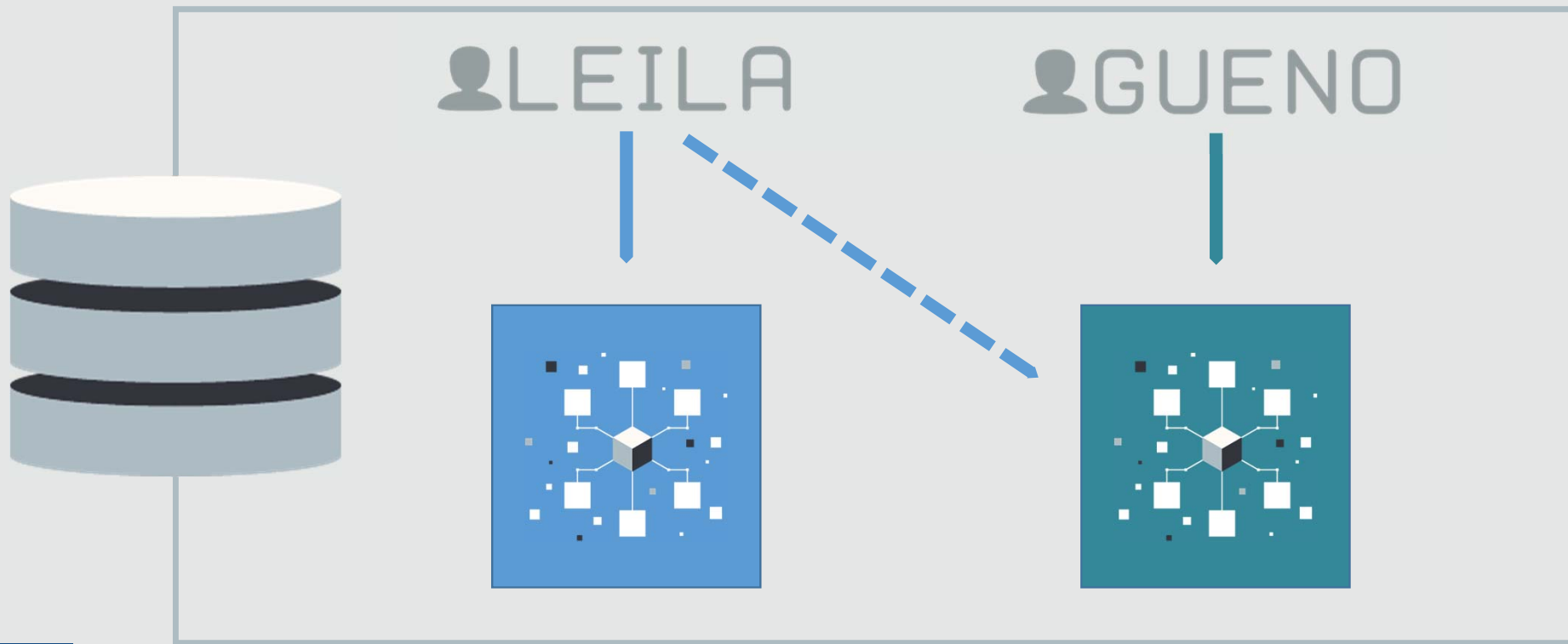
Présentation de l'écosystème ORACLE

# L'entreprise



Présentation de l'écosystème ORACLE

# Notion de base de données chez Oracle

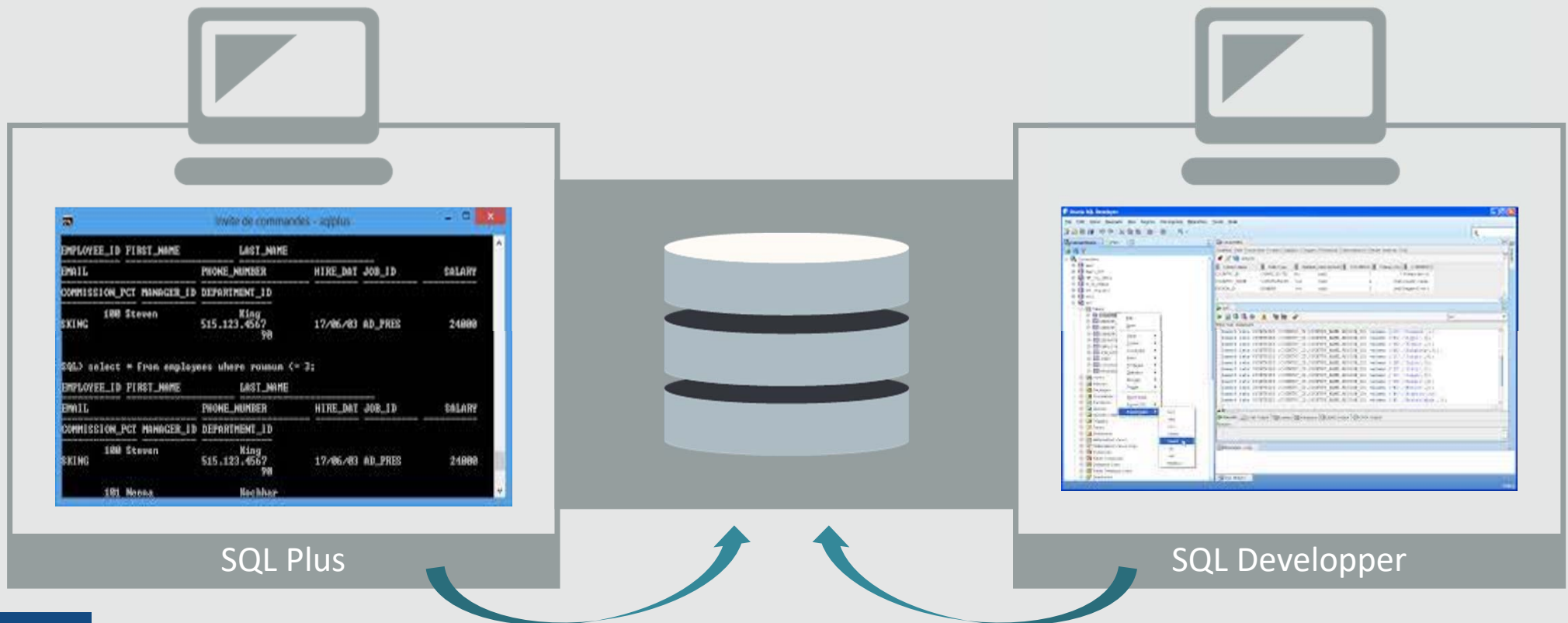


# Création d'un utilisateur avec des droits

```
create role eni;  
grant connect, resource to eni;  
grant create procedure to eni;  
grant create trigger to eni;  
  
create user u1  
identified by x  
default tablespace users  
temporary tablespace temp  
quota unlimited on users;  
  
grant eni to u1;
```

# Présentation de l'écosystème ORACLE

## L'environnement logiciel



Présentation de l'écosystème ORACLE  
SQL Plus

# Démonstration





Présentation de l'écosystème ORACLE  
SQL Developer

# Démonstration



# Conclusion

- Vous connaissez l'entreprise Oracle
- Vous avez compris le concept de schéma
- Vous avez découvert SQL Plus
- Vous savez créer un nouvel utilisateur
- Vous avez découvert SQL Developer



# PL / SQL

**Module de rappel – La gestion des tables sous Oracle**



# Objectifs

- Rappel des instructions du DDL
- Découverte des particularités Oracle



La gestion des tables sous Oracle

# La dette technique



Inventeur du concept de wiki et  
a permis l'élaboration de  
Wikipédia

*-Ward Cunningham-*

La gestion des tables sous Oracle

# La dette technique

Quand on code au plus vite et de manière non optimale, on contracte une dette technique que l'on rembourse tout au long de la vie du projet sous forme de temps de développement de plus en plus long et de bugs de plus en plus fréquents.



La gestion des tables sous Oracle

# Les conventions de nommage

- Mots clés SQL en majuscules
- Nom des tables au pluriel et en snake\_case
- Nom des colonnes explicites et en snake\_case



[https://docs.oracle.com/cd/A97630\\_01/appdev.920/a96624/02\\_funds.htm](https://docs.oracle.com/cd/A97630_01/appdev.920/a96624/02_funds.htm)

# Les commentaires de code

```
/* Je suis  
    un commentaire sur  
    plusieurs lignes */
```

```
--Je suis un commentaire sur une ligne
```



La gestion des tables sous Oracle

# Les principaux types caractères

CHAR (Size)

VARCHAR (Size)

LONG



La gestion des tables sous Oracle

# Le principal type numérique

`NUMBER (precision, scale)`

| Valeur réelle | Spécification de la colonne | Valeur stockée |
|---------------|-----------------------------|----------------|
| 123,89        | NUMBER                      | 123,89         |
| 123,89        | NUMBER(3)                   | 124            |
| 123,89        | NUMBER(3,2)                 | Impossible     |
| 123,89        | NUMBER(6,-2)                | 100            |
| 0,000127      | NUMBER(4,5)                 | 0,00013        |

La gestion des tables sous Oracle

# Les principaux types dates

DATE

TIMESTAMP



La gestion des tables sous Oracle

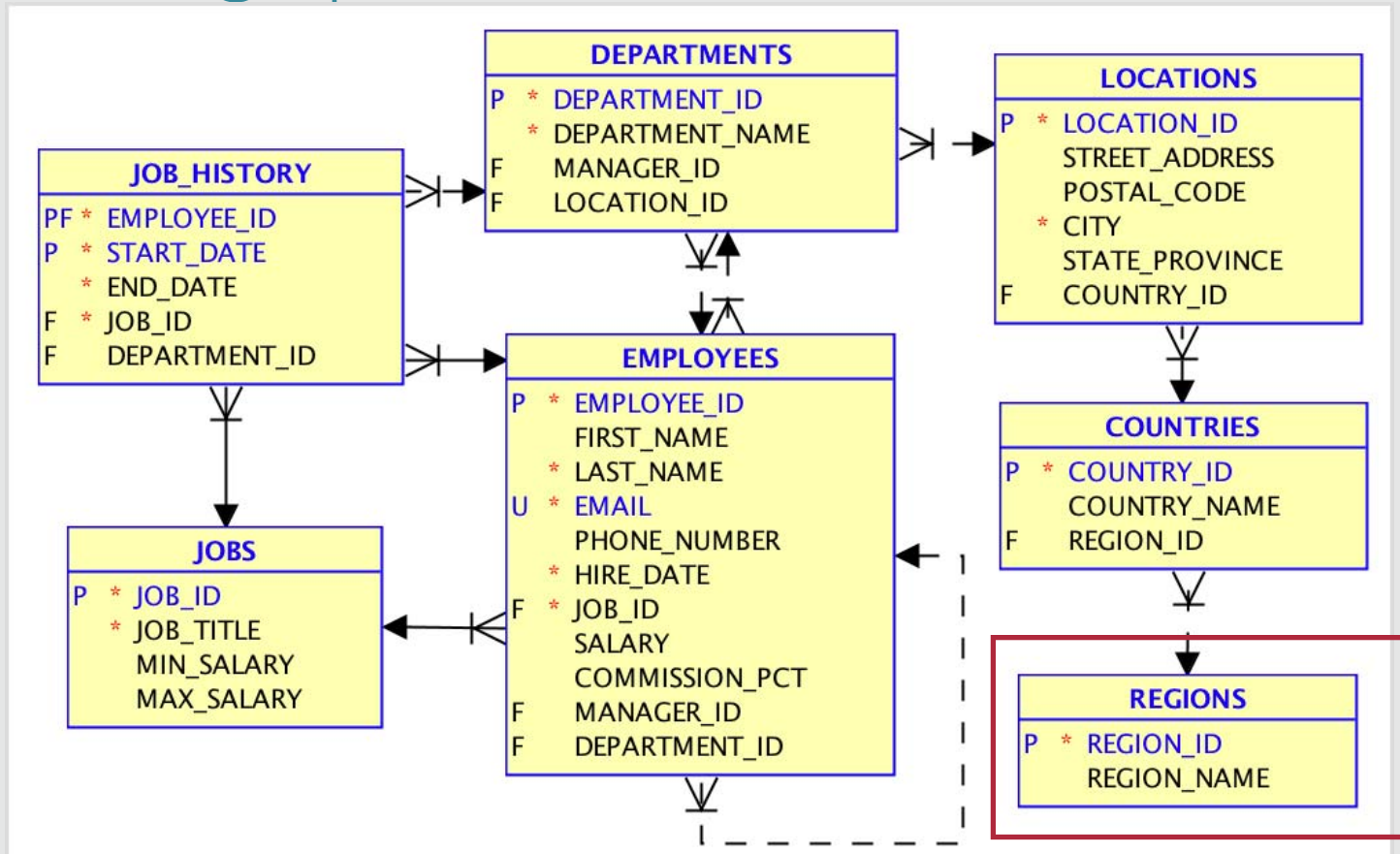
# Le principal type multimédia

BLOB



La gestion des tables sous Oracle

# Le schéma logique de base de données



La gestion des tables sous Oracle

## La création de la table REGIONS

```
CREATE TABLE regions
(
    region_id          NUMBER PRIMARY KEY,
    region_name        VARCHAR2(25) DEFAULT 'XXX'
);
```

La gestion des tables sous Oracle

# Les commentaires posés sur les objets

```
COMMENT ON TABLE regions
IS 'Regions table that contains region numbers and names. Contains 4 rows; references with the Countries table.'

COMMENT ON COLUMN regions.region_id
IS 'Primary key of regions table.'

COMMENT ON COLUMN regions.region_name
IS 'Names of regions. Locations are in the countries of these regions.'
```



La gestion des tables sous Oracle

## La consultation des commentaires

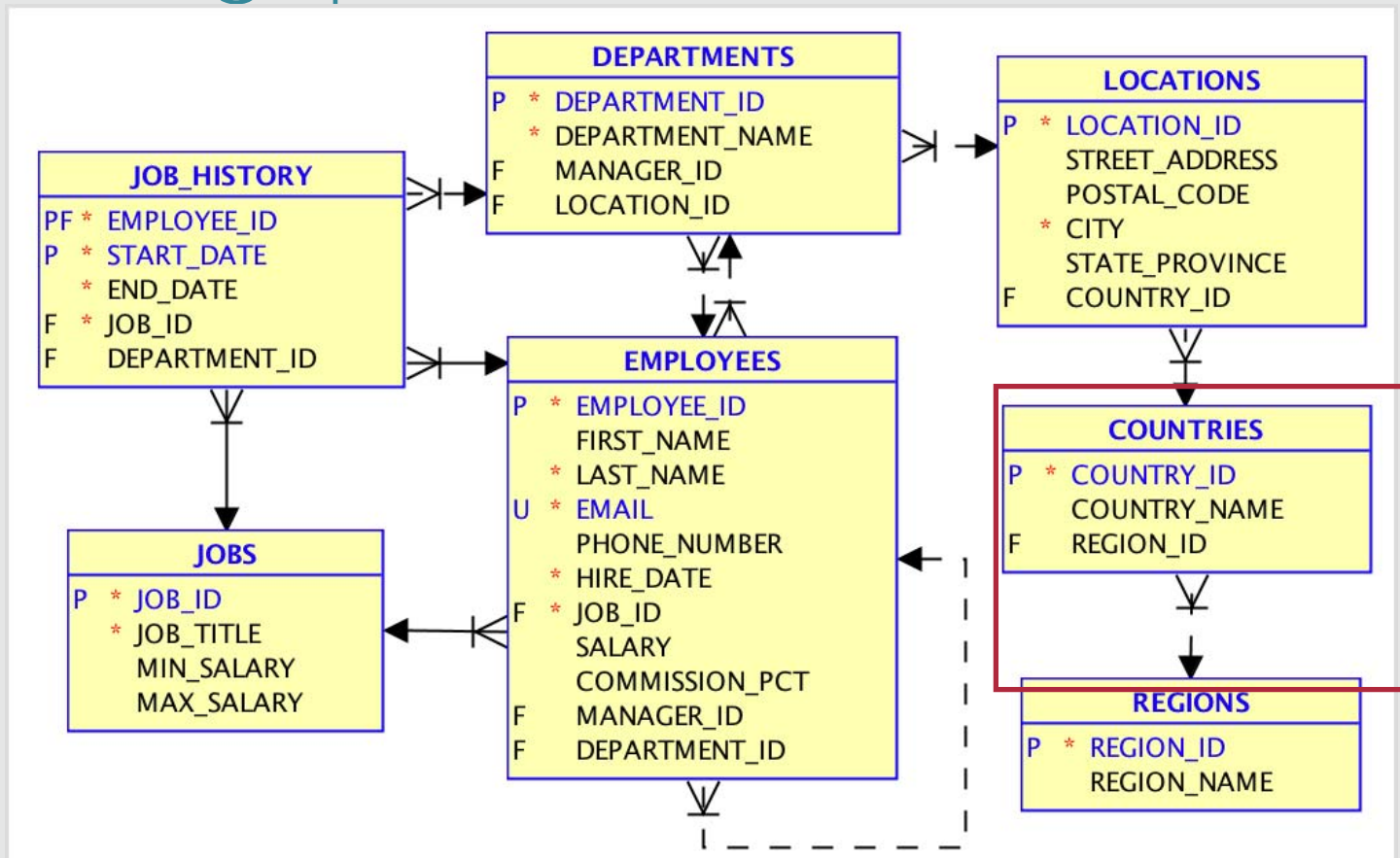
```
SELECT * FROM user_tab_comments;
```

```
SELECT * FROM user_col_comments;
```



La gestion des tables sous Oracle

# Le schéma logique de base de données



La gestion des tables sous Oracle

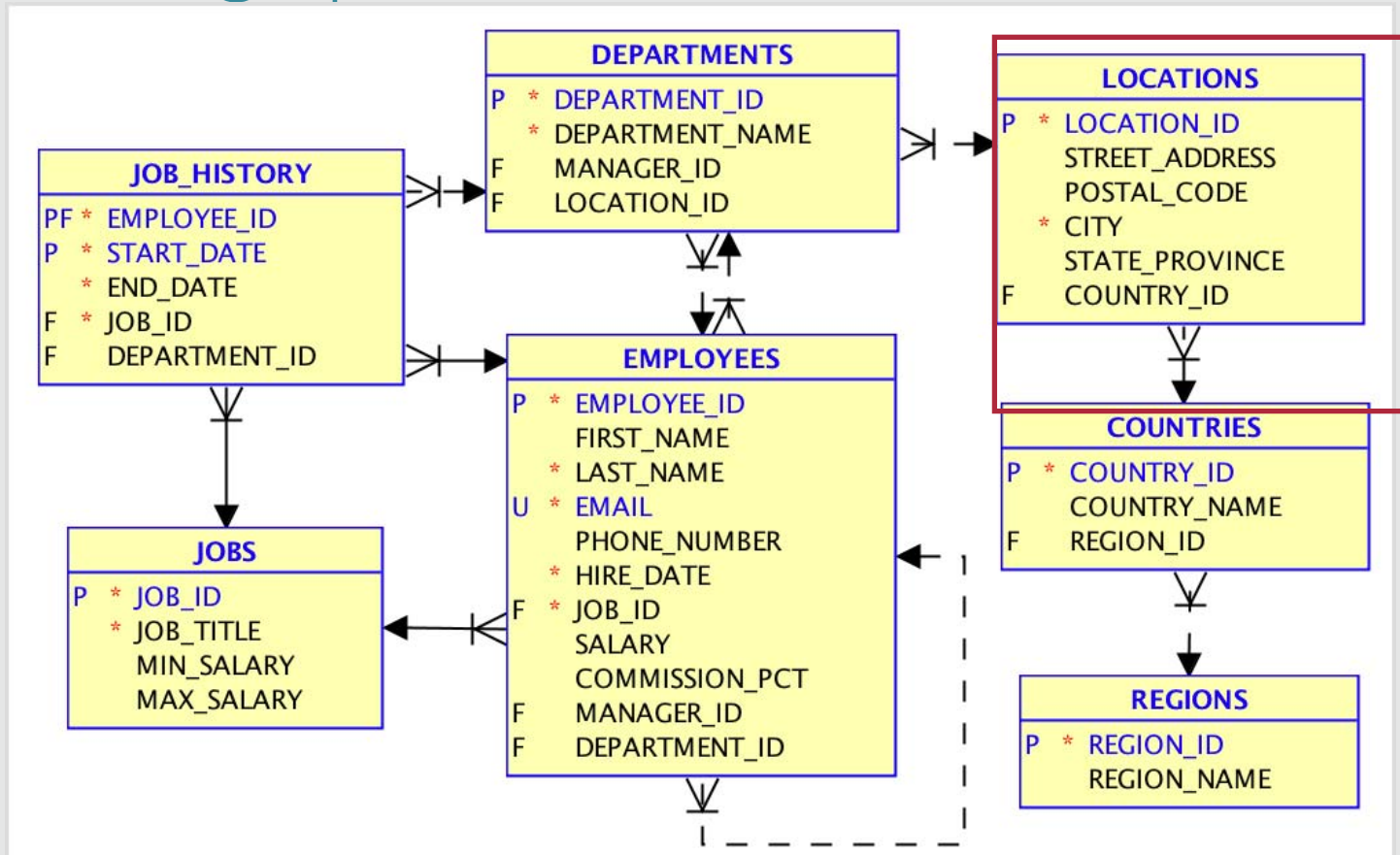
# La création de la table COUNTRIES

```
CREATE TABLE countries
(
    country_id CHAR(2) CONSTRAINT country_id_nn NOT NULL,
    country_name VARCHAR2(40) ,
    region_id   NUMBER ,
    CONSTRAINT country_c_id_pk PRIMARY KEY (country_id)
);

ALTER TABLE countries
ADD (
    CONSTRAINT countr_reg_fk
    FOREIGN KEY (region_id)
    REFERENCES regions(region_id)
);
```

La gestion des tables sous Oracle

# Le schéma logique de base de données



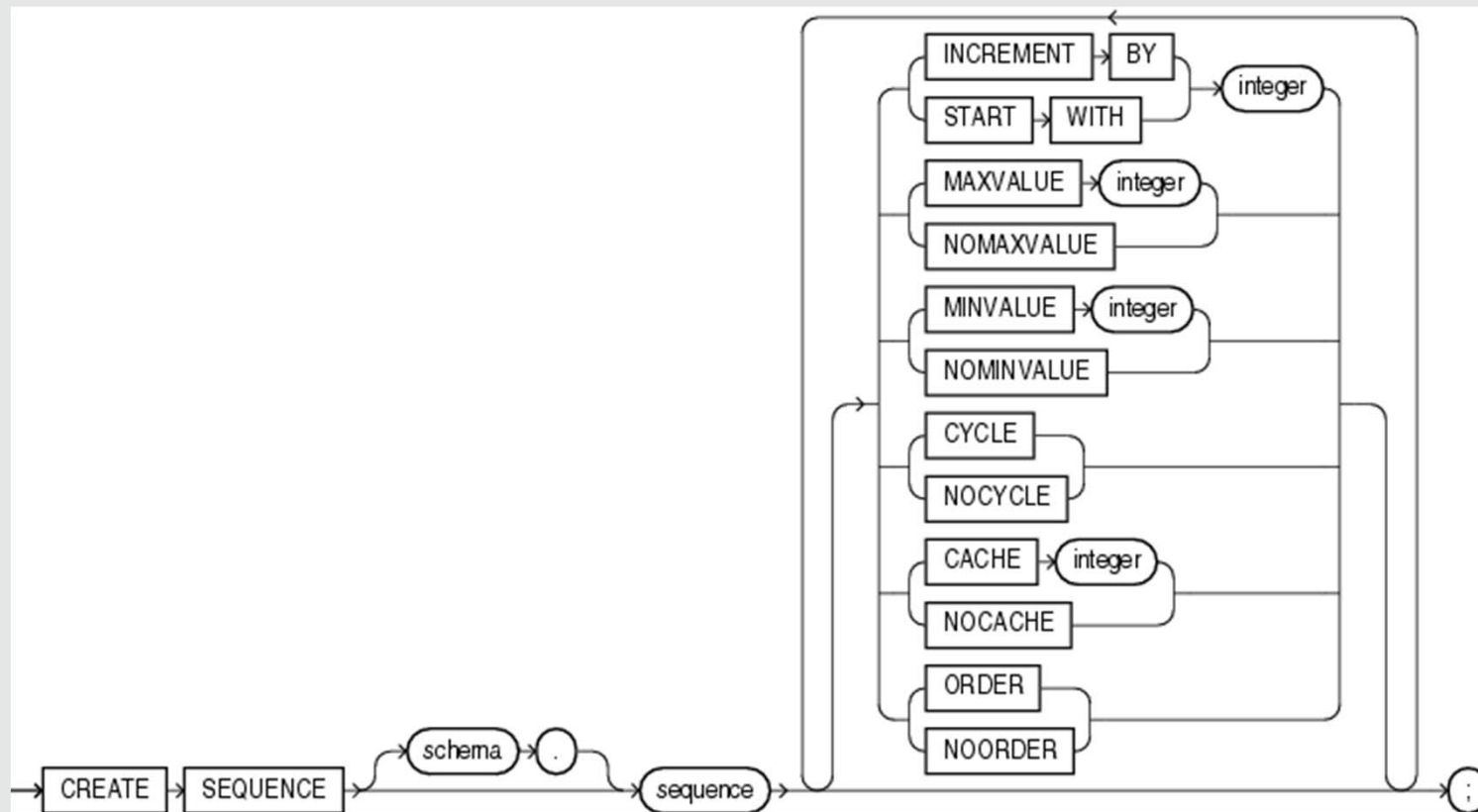
La gestion des tables sous Oracle

# La création de la table LOCATIONS

```
CREATE TABLE locations
(
    location_id      NUMBER(4) CONSTRAINT loc_id_pk PRIMARY KEY,
    street_address   VARCHAR2(40),
    postal_code      VARCHAR2(12),
    city             VARCHAR2(30) CONSTRAINT loc_city_nn NOT NULL,
    state_province   VARCHAR2(25),
    country_id       CHAR(2)  CONSTRAINT loc_c_id_fk REFERENCES countries(country_id)
);
```

La gestion des tables sous Oracle

## La création d'une séquence



La gestion des tables sous Oracle

## La création d'une séquence

```
CREATE SEQUENCE locations_seq  
  START WITH      3300  
  INCREMENT BY    100  
  MAXVALUE        9900  
  NOCACHE  
  NOCYCLE;
```

La gestion des tables sous Oracle

# L'utilisation d'une séquence

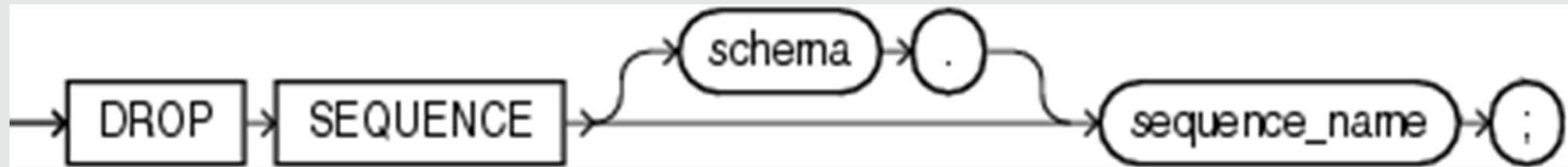
```
locations_seq.NEXTVAL
```

```
locations_seq.CURRVAL
```



La gestion des tables sous Oracle

# La suppression d'une séquence

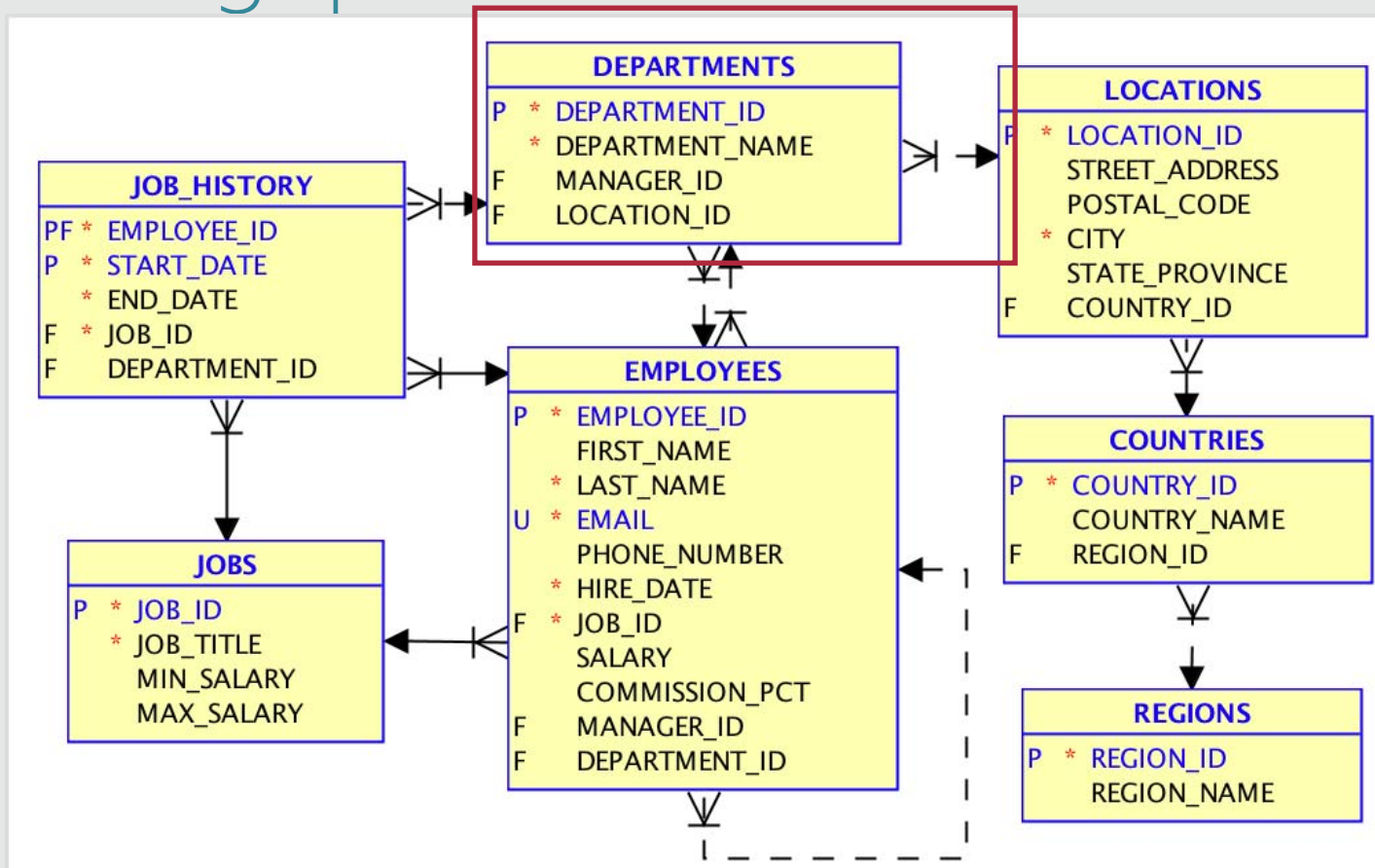


```
DROP SEQUENCE departments_seq;  
DROP SEQUENCE employees_seq;  
DROP SEQUENCE locations_seq;
```



La gestion des tables sous Oracle

# Le schéma logique de base de données



La gestion des tables sous Oracle

# La création de la table DEPARTMENTS

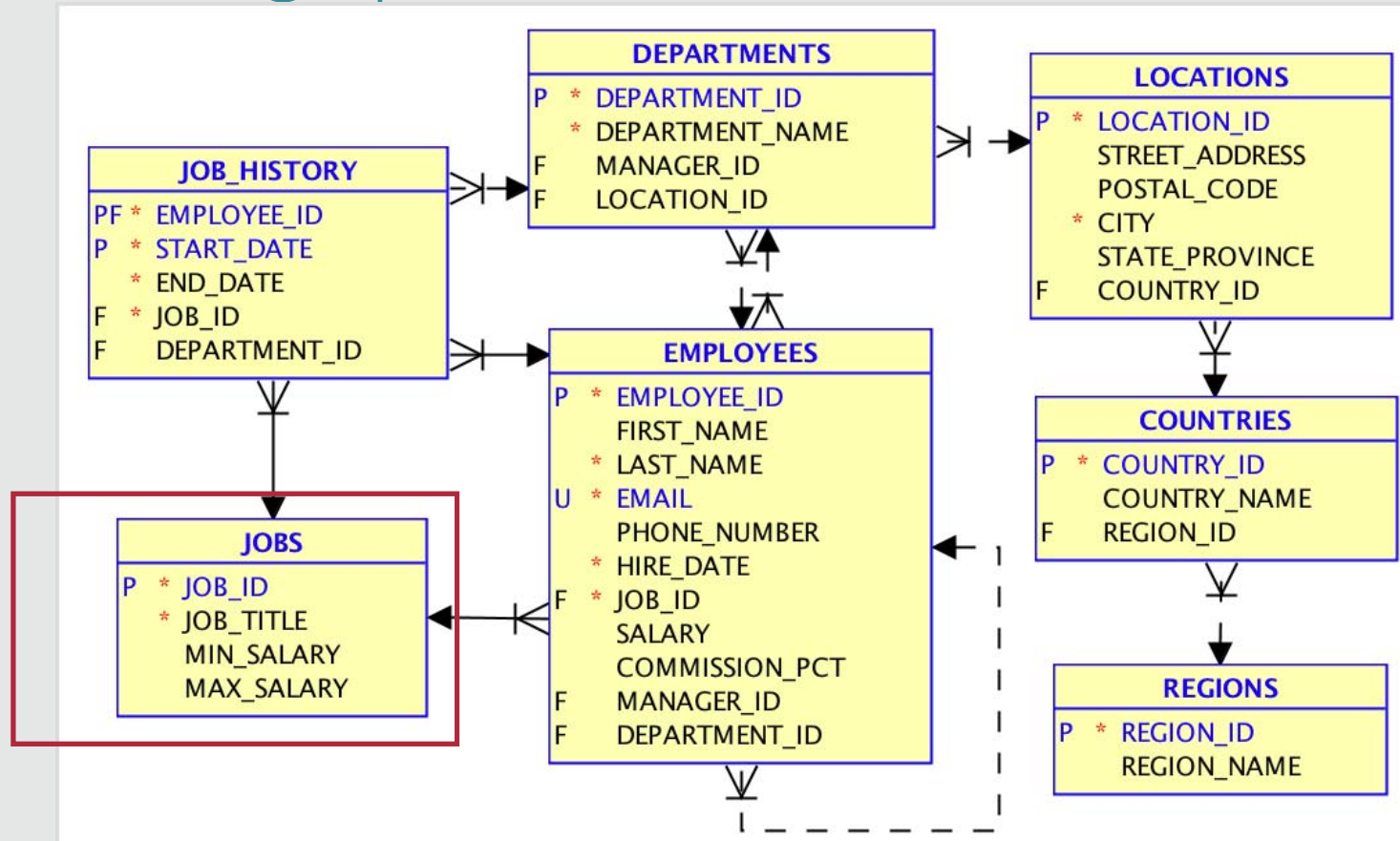
```
CREATE TABLE departments
(
    department_id    NUMBER(4),
    department_name   VARCHAR2(30) CONSTRAINT dept_name_nn NOT NULL,
    manager_id       NUMBER(6),
    location_id       NUMBER(4)
);

ALTER TABLE departments
ADD (
    CONSTRAINT dept_id_pk PRIMARY KEY (department_id),
    CONSTRAINT dept_loc_fk FOREIGN KEY (location_id) REFERENCES locations (location_id)
);

CREATE SEQUENCE departments_seq
START WITH          280
INCREMENT BY        10
MAXVALUE             9990
NOCACHE
NOCYCLE;
```

La gestion des tables sous Oracle

# Le schéma logique de base de données



La gestion des tables sous Oracle

# La création de la table JOBS

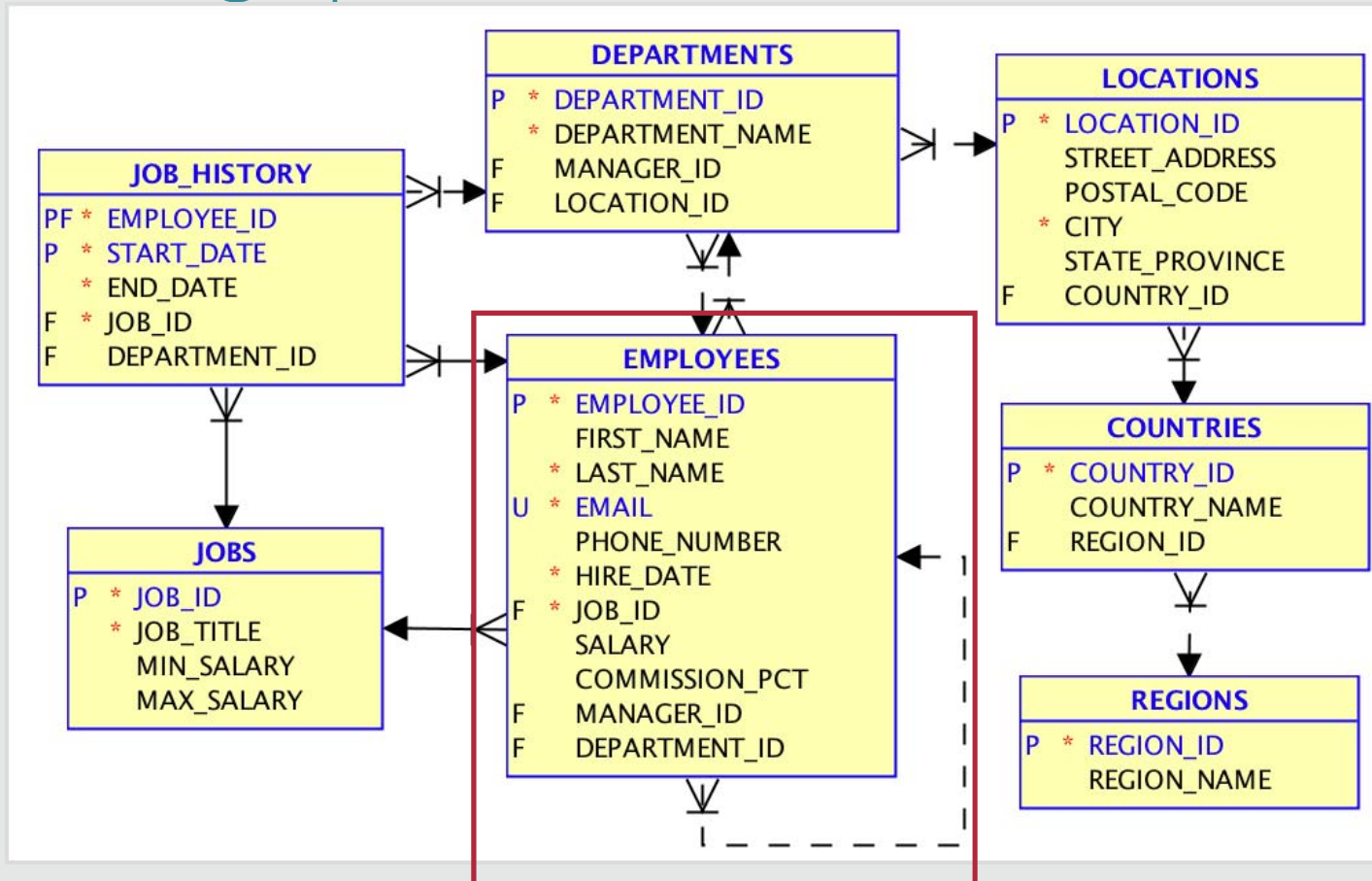
```
CREATE TABLE jobs
(
    job_id          VARCHAR2(10),
    job_title        VARCHAR2(35)
    CONSTRAINT job_title_nn NOT NULL,
    min_salary       NUMBER(6),
    max_salary       NUMBER(6)
);

CREATE UNIQUE INDEX job_id_pk ON jobs (job_id) ;

ALTER TABLE jobs
ADD (
    CONSTRAINT job_id_pk PRIMARY KEY(job_id)
) ;
```

La gestion des tables sous Oracle

# Le schéma logique de base de données



La gestion des tables sous Oracle

# La création de la table EMPLOYEES

```
CREATE TABLE employees
(
    employee_id      NUMBER(6),
    first_name       VARCHAR2(20),
    last_name        VARCHAR2(25) CONSTRAINT emp_last_name_nn NOT NULL,
    email            VARCHAR2(25) CONSTRAINT emp_email_nn NOT NULL,
    phone_number     VARCHAR2(20),
    hire_date        DATE CONSTRAINT emp_hire_date_nn NOT NULL,
    job_id           VARCHAR2(10) CONSTRAINT emp_job_nn NOT NULL,
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    manager_id       NUMBER(6),
    department_id    NUMBER(4),
    ,CONSTRAINT emp_salary_min CHECK (salary > 0)
    ,CONSTRAINT emp_email_uk UNIQUE (email)
);
```

La gestion des tables sous Oracle

# L'intégration de la table EMPLOYEES

```
ALTER TABLE employees
ADD (
    CONSTRAINT emp_emp_id_pk PRIMARY KEY (employee_id),
    CONSTRAINT emp_dept_fk FOREIGN KEY (department_id) REFERENCES departments,
    CONSTRAINT emp_job_fk FOREIGN KEY (job_id) REFERENCES jobs (job_id),
    CONSTRAINT emp_manager_fk FOREIGN KEY (manager_id) REFERENCES employees
) ;

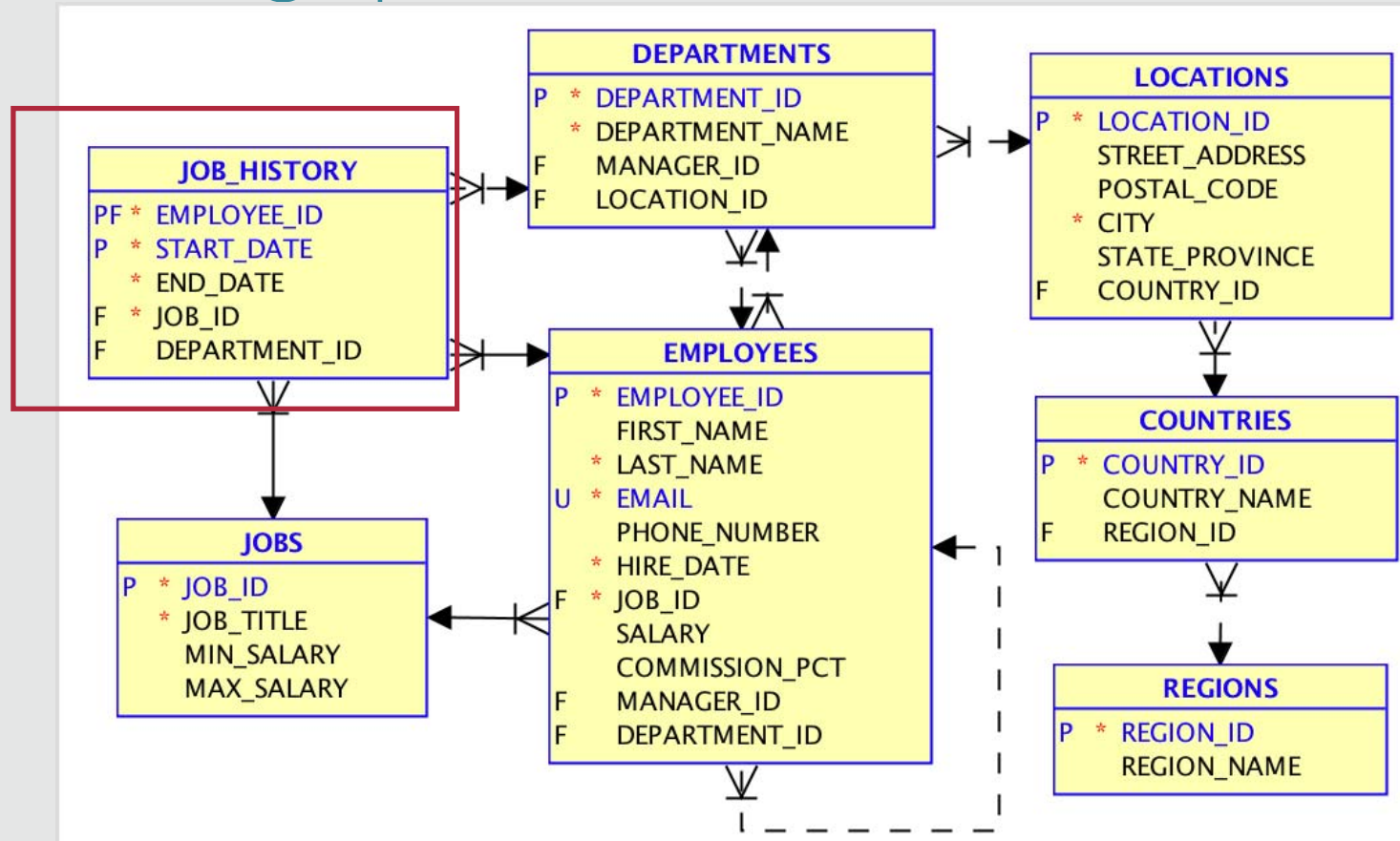
ALTER TABLE departments
ADD (
    CONSTRAINT dept_mgr_fk FOREIGN KEY (manager_id) REFERENCES employees (employee_id)
) ;

CREATE SEQUENCE employees_seq
START WITH 207
INCREMENT BY 1
NOCACHE
NOCYCLE;
```



La gestion des tables sous Oracle

# Le schéma logique de base de données





# Les contraintes portant sur plusieurs colonnes

```
CREATE TABLE job_history
(
    employee_id    NUMBER(6) CONSTRAINT    jhist_employee_nn  NOT NULL,
    start_date     DATE CONSTRAINT        jhist_start_date_nn  NOT NULL,
    end_date       DATE CONSTRAINT        jhist_end_date_nn    NOT NULL,
    job_id         VARCHAR2(10) CONSTRAINT    jhist_job_nn     NOT NULL,
    department_id  NUMBER(4),
    CONSTRAINT     jhist_date_interval CHECK (end_date > start_date)
);

CREATE UNIQUE INDEX jhist_emp_id_st_date_pk ON job_history (employee_id, start_date) ;

ALTER TABLE job_history
ADD (
    CONSTRAINT jhist_emp_id_st_date_pk PRIMARY KEY (employee_id, start_date),
    CONSTRAINT jhist_job_fk FOREIGN KEY (job_id) REFERENCES jobs,
    CONSTRAINT jhist_emp_fk FOREIGN KEY (employee_id) REFERENCES employees,
    CONSTRAINT jhist_dept_fk FOREIGN KEY (department_id) REFERENCES departments
) ;
```

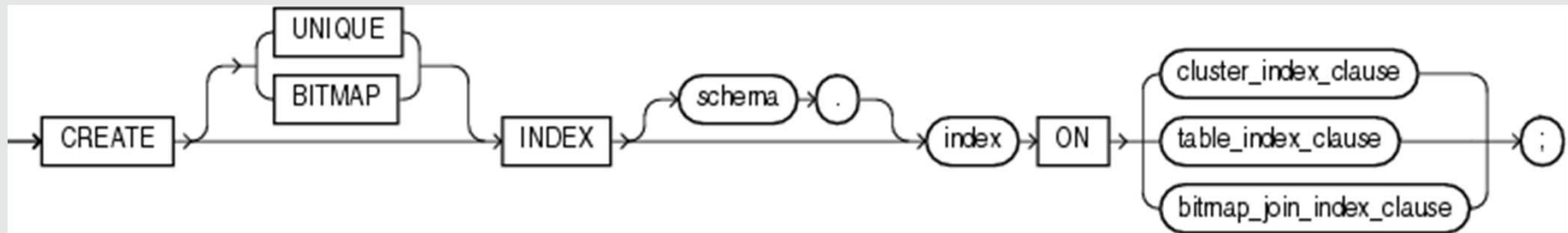
La gestion des tables sous Oracle

## Le vidage d'une table

```
TRUNCATE TABLE informations;
```

La gestion des tables sous Oracle

# La syntaxe de création d'un index



La gestion des tables sous Oracle

## Exemple de création d'un index

```
CREATE INDEX emp_department_ix ON employees (department_id);

CREATE INDEX emp_job_ix ON employees (job_id);

CREATE INDEX emp_manager_ix ON employees (manager_id);

CREATE INDEX emp_name_ix ON employees (last_name, first_name);

CREATE INDEX dept_location_ix ON departments (location_id);

CREATE INDEX jhist_job_ix ON job_history (job_id);

CREATE INDEX jhist_employee_ix ON job_history (employee_id);

CREATE INDEX jhist_department_ix ON job_history (department_id);

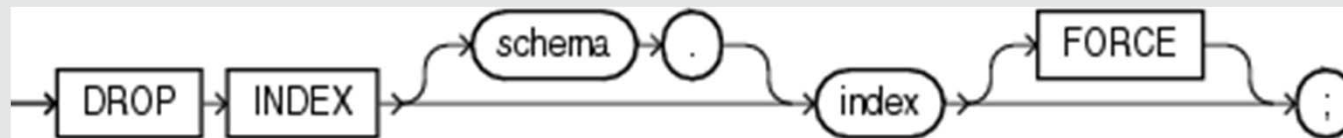
CREATE INDEX loc_city_ix ON locations (city);

CREATE INDEX loc_state_province_ix ON locations (state_province);

CREATE INDEX loc_country_ix ON locations (country_id);
```

La gestion des tables sous Oracle

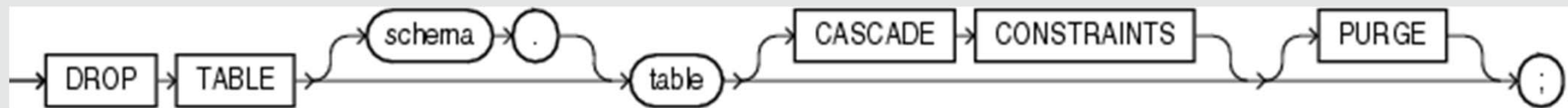
## La suppression d'un index



```
DROP INDEX dept_location_ix
```

La gestion des tables sous Oracle

# La suppression d'une table



```
DROP TABLE regions      CASCADE CONSTRAINTS;  
DROP TABLE departments  CASCADE CONSTRAINTS;  
DROP TABLE locations    CASCADE CONSTRAINTS;  
DROP TABLE jobs         CASCADE CONSTRAINTS;  
DROP TABLE job_history  CASCADE CONSTRAINTS;  
DROP TABLE employees    CASCADE CONSTRAINTS;  
DROP TABLE countries    CASCADE CONSTRAINTS;
```

La gestion des tables sous Oracle

# La suppression d'une contrainte

```
ALTER TABLE locations DROP CONSTRAINT loc_city_nn;
```



La gestion des tables sous Oracle

# L'ajout ou suppression d'une colonne

```
ALTER TABLE departments ADD dn VARCHAR2(300);
```

```
ALTER TABLE departments DROP COLUMN dn;
```



La gestion des tables sous Oracle

# Le changement de nom d'une colonne

```
ALTER TABLE regions  
RENAME COLUMN region_name TO region_true_name;
```

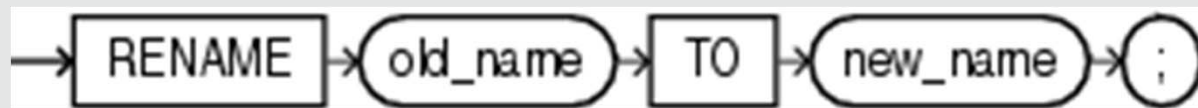
La gestion des tables sous Oracle

# La modification d'une colonne

```
ALTER TABLE  
    departments  
MODIFY  
    department_name VARCHAR2 (50) ;
```

La gestion des tables sous Oracle

## Le changement de nom d'une table



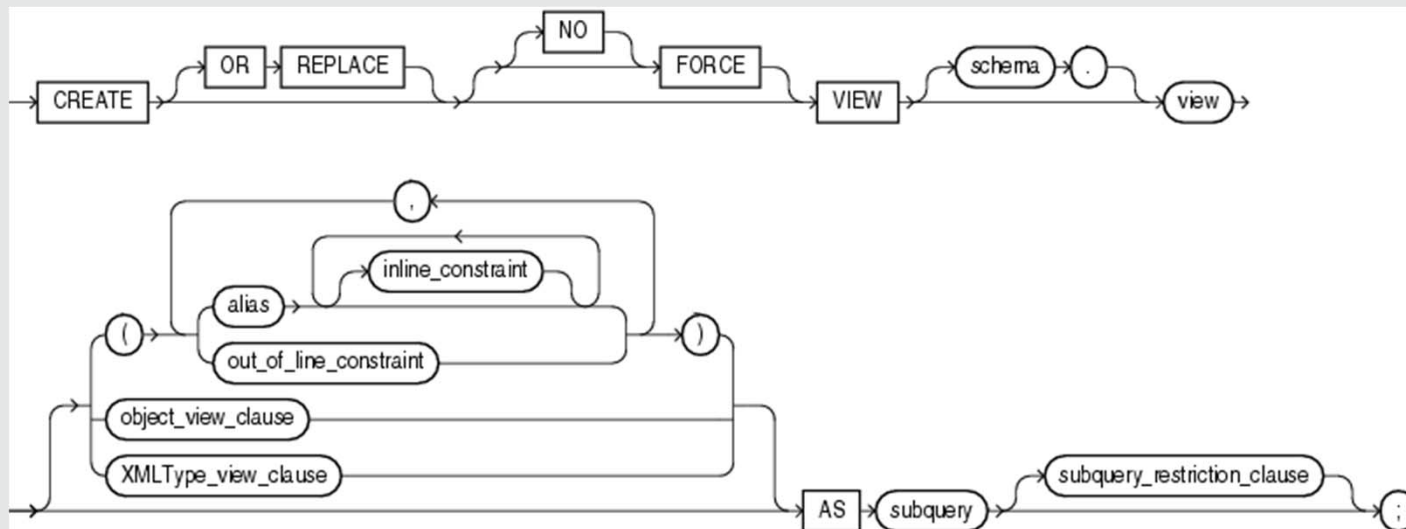
La gestion des tables sous Oracle

## La définition d'une colonne identité

```
CREATE TABLE informations
(
    informations_id NUMBER GENERATED AS IDENTITY,
    informations_text VARCHAR(500)
)
```

La gestion des tables sous Oracle

# La syntaxe de création d'une vue



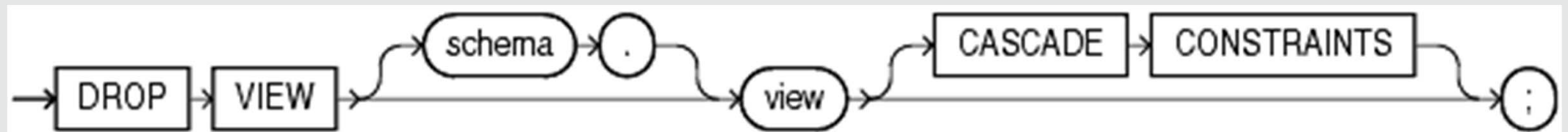
La gestion des tables sous Oracle

## Exemple de création d'une vue

```
CREATE OR REPLACE FORCE VIEW regions_countries_view
AS
SELECT
    r.region_name,
    c.country_name
FROM
    regions r
JOIN countries c ON r.region_id = c.region_id;
```

La gestion des tables sous Oracle

## La suppression d'une vue



La gestion des tables sous Oracle

## Exemple de suppression d'une vue

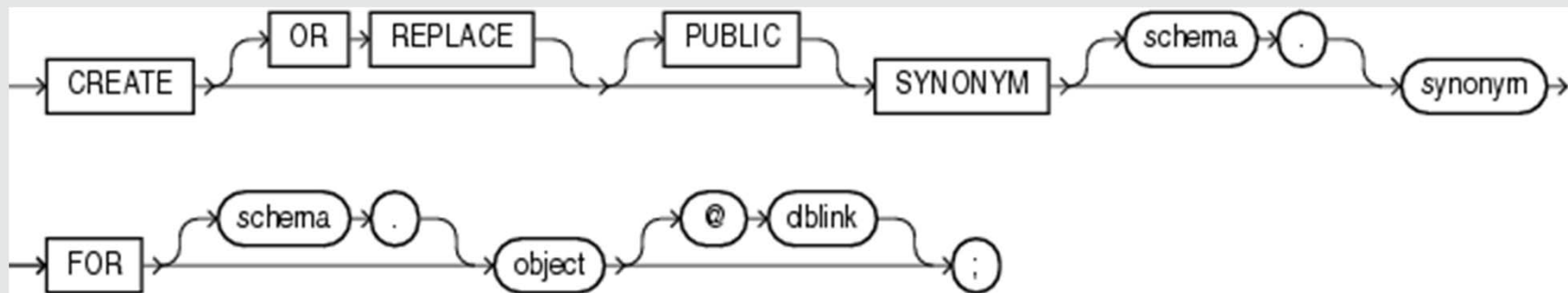
```
DROP VIEW regions_countries_view;
```





La gestion des tables sous Oracle

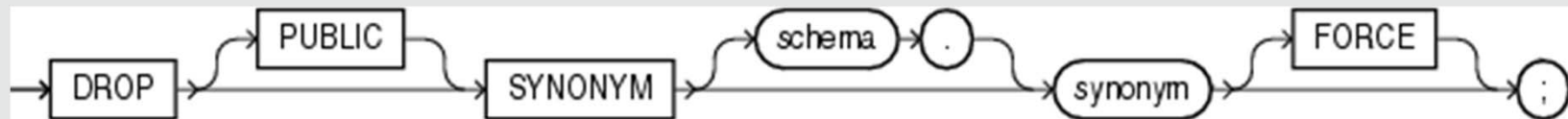
# La syntaxe de création d'un synonyme



```
CREATE SYNONYM history FOR job_history;
```

La gestion des tables sous Oracle

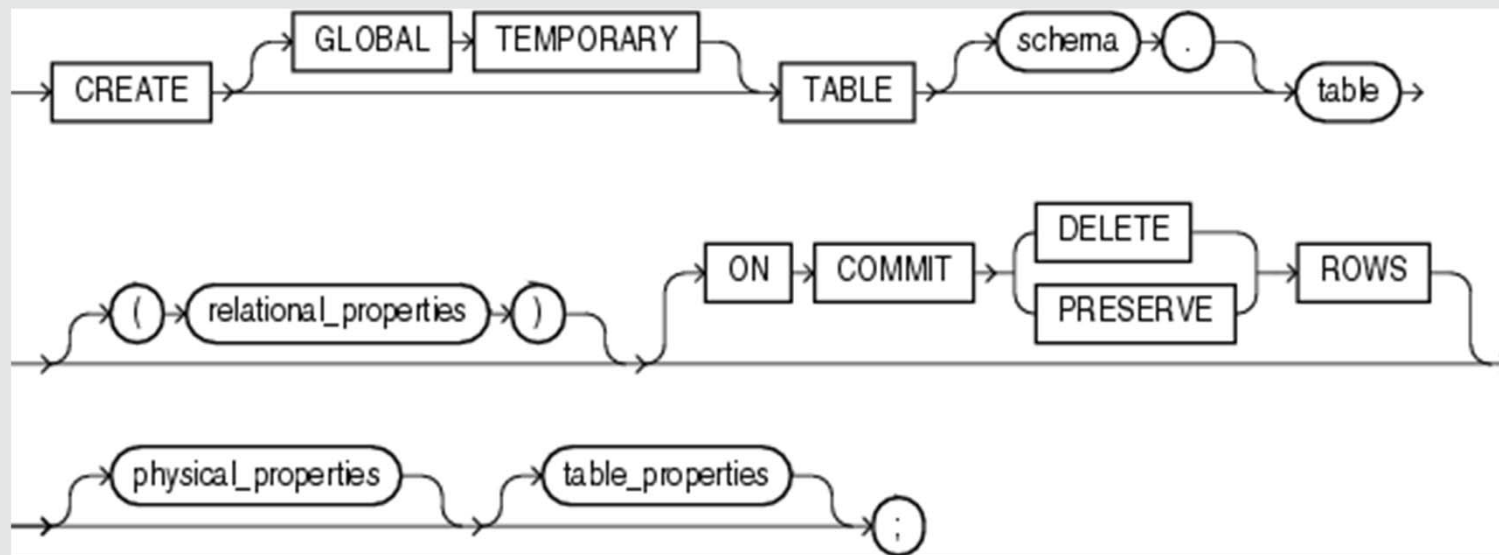
# La suppression d'un synonyme



```
DROP SYNONYM history;
```

La gestion des tables sous Oracle

# La syntaxe de création d'une table temporaire



La gestion des tables sous Oracle

## Exemple de création d'une table temporaire

```
CREATE GLOBAL TEMPORARY TABLE employees_sel  
(  
    id VARCHAR2(256),  
    end_date DATE  
) ON COMMIT PRESERVE ROWS ;
```

# Conclusion

- Vous savez mettre en place une base de données sous Oracle



# PL / SQL

**Module de rappel – La gestion des données sous Oracle**



# Objectifs

- Rappel du DML
- Découverte des particularités Oracle



La gestion des données sous Oracle

# L'insertion d'enregistrement

**INSERT  
INTO**

`INSERT INTO Table VALUES (valeur_1, valeur_2, valeur_3)`

**VALUES**

`INSERT INTO Table (colonne_1, colonne_3) VALUES (valeur_1, valeur_3)`



Il faut que tous les attributs NOT NULL soient définis lors d'un INSERT.



La gestion des données sous Oracle

# Exemples d'insertions d'enregistrements

```
INSERT INTO regions(region_id,region_name) VALUES (1, 'Europe' );  
  
INSERT INTO regions VALUES (2,'Americas');  
  
INSERT INTO regions VALUES ( 3, 'Asia');  
  
INSERT INTO regions VALUES ( 4, 'Middle East and Africa');
```

La gestion des données sous Oracle

# La mise à jour d'enregistrement(s)

**UPDATE**

**UPDATE** *Table*

**SET** *colonne = valeur*

**SET**

**UPDATE** *Table*

**SET** *colonne\_1 = valeur*

**WHERE**

**WHERE** *colonne\_2 = condition*

La gestion des données sous Oracle

# Exemples de mises à jour d'enregistrements

```
UPDATE regions SET region_name = UPPER(region_name)
```

```
UPDATE regions SET region_name = 'America' WHERE region_name = 'Americas'
```

La gestion des données sous Oracle

# La suppression d'enregistrement(s)

**DELETE**

DELETE FROM *Table*

**FROM**

DELETE FROM *Table* WHERE *colonne = condition*

**WHERE**

La gestion des données sous Oracle

# Exemples de suppressions d'enregistrements

```
DELETE FROM regions WHERE region_id = 1;
```

```
DELETE regions;
```

La gestion des données sous Oracle

# La validation des modifications

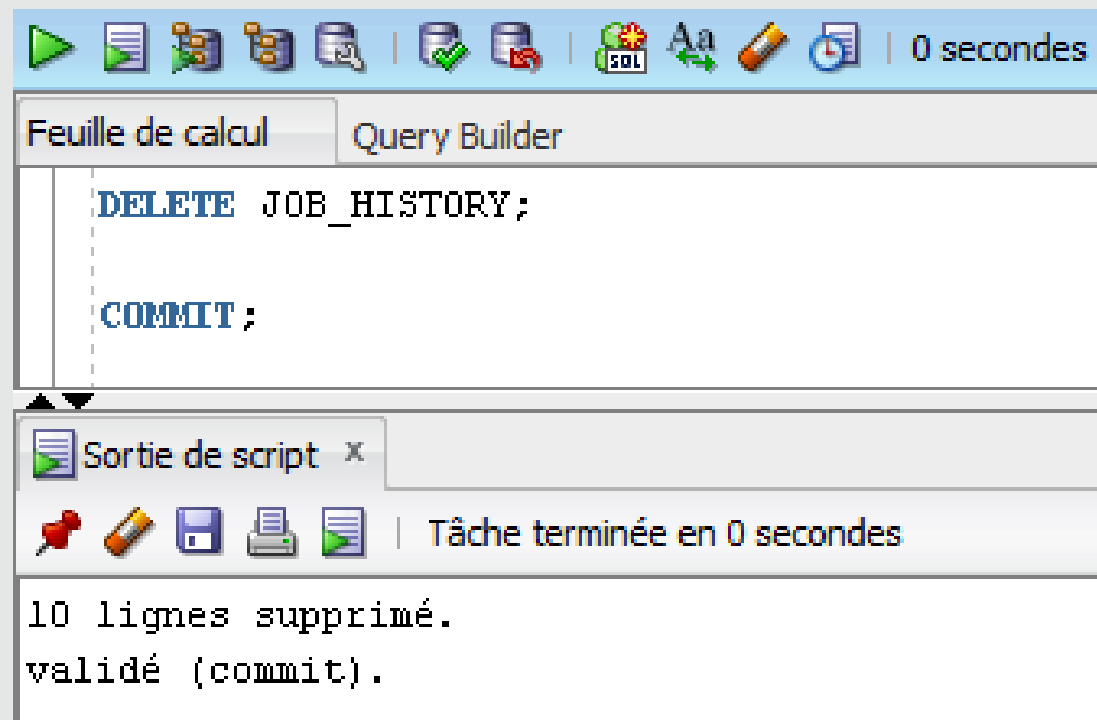
**Enregistre** en base toutes les insertions, modifications et suppressions réalisées depuis le début de la transaction.

Tant qu'il n'y a pas eu COMMIT, **seule la connexion courante** voit ses mises à jour.



La gestion des données sous Oracle

# La validation des modifications



La gestion des données sous Oracle

# L'invalidation des modifications

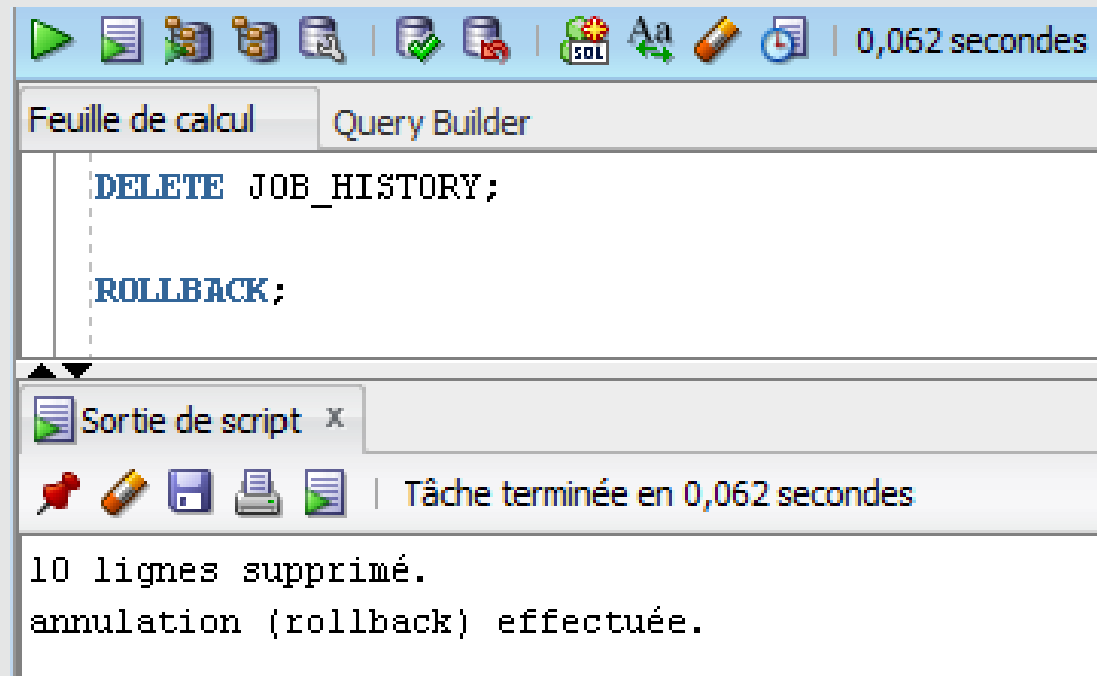
**Annule** toutes les insertions, modifications et suppressions réalisées depuis le début de la transaction.





La gestion des données sous Oracle

# L'invalidation des modifications



# Conclusion

- Vous savez modifier des données sous Oracle
- Vous savez valider vos modifications
- Vous savez annuler vos modifications



# PL / SQL

## Module 2 – Le langage PL / SQL



# Objectifs

- Savoir expliquer ce qu'est le PL / SQL
- Comprendre l'intérêt du PL / SQL



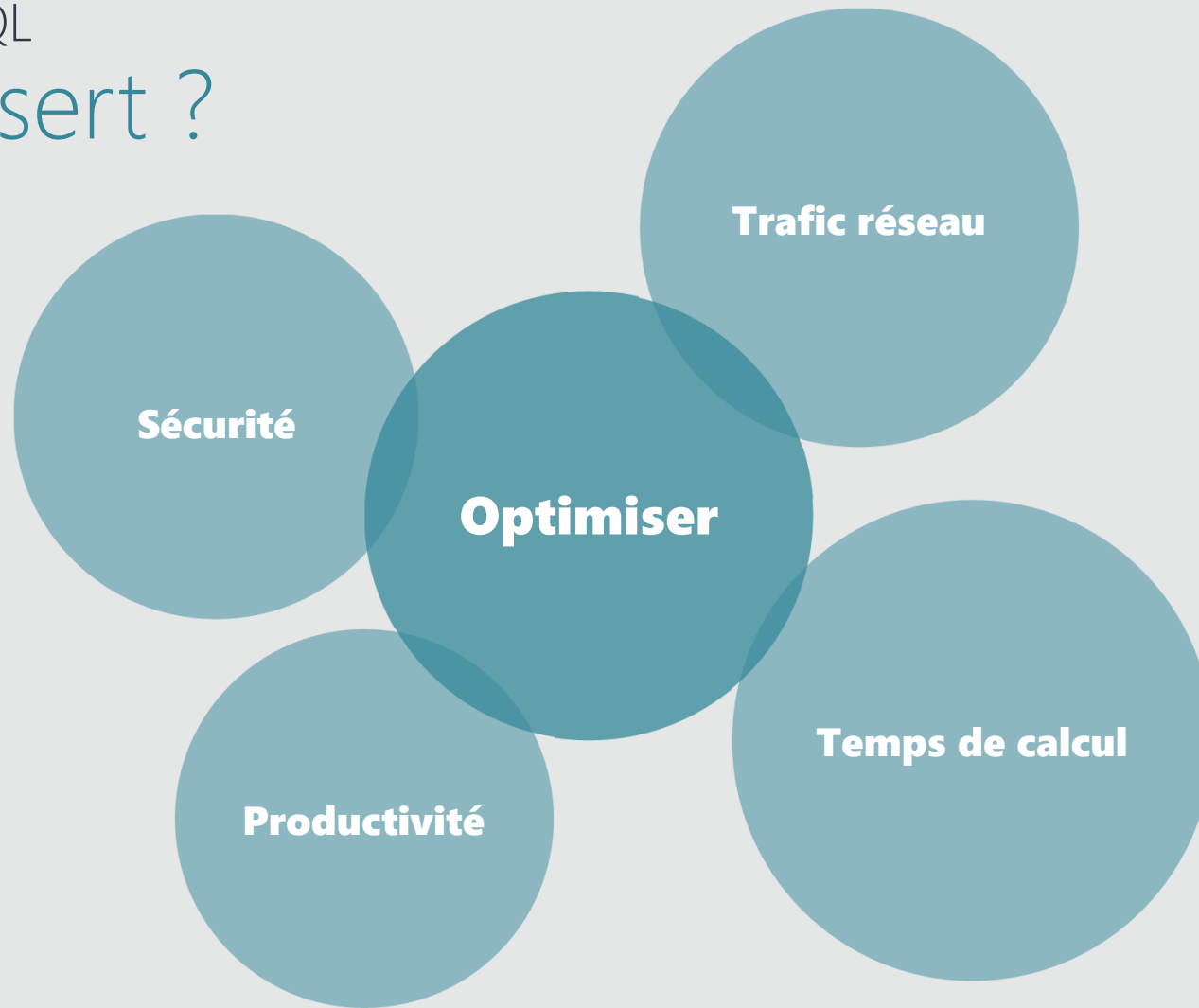
# Le langage PL / SQL C'est quoi ?



Procedural Language / Structured Query Language

Le langage PL / SQL

À quoi ça sert ?



# Conclusion

- Vous savez que le PL/SQL est un langage procédural
- Vous savez que le PL/SQL s'exécute sur le serveur
- Vous savez que le PL/SQL permet d'accéder aux données de manière optimisée



# PL / SQL

## Module 3 – Les blocs PL/SQL





# Objectifs

- Sensibilisation aux conventions de nommage
- Comprendre ce qu'est un bloc PL/SQL
- Connaître les différents types de blocs PL/SQL
- Découvrir le plus petit bloc PL/SQL du monde
- Connaître les sections possibles d'un bloc PL/SQL



Les blocs PL / SQL

# La dette technique

Quand on code au plus vite et de manière non optimale, on contracte une dette technique que l'on rembourse tout au long de la vie du projet sous forme de temps de développement de plus en plus long et de bugs de plus en plus fréquents.



# Les clés pour réduire la dette technique

```
nombre_adherents INT;
```

```
UPDATE  
    benevoles  
SET  
    actif = 1;
```

Les blocs PL / SQL

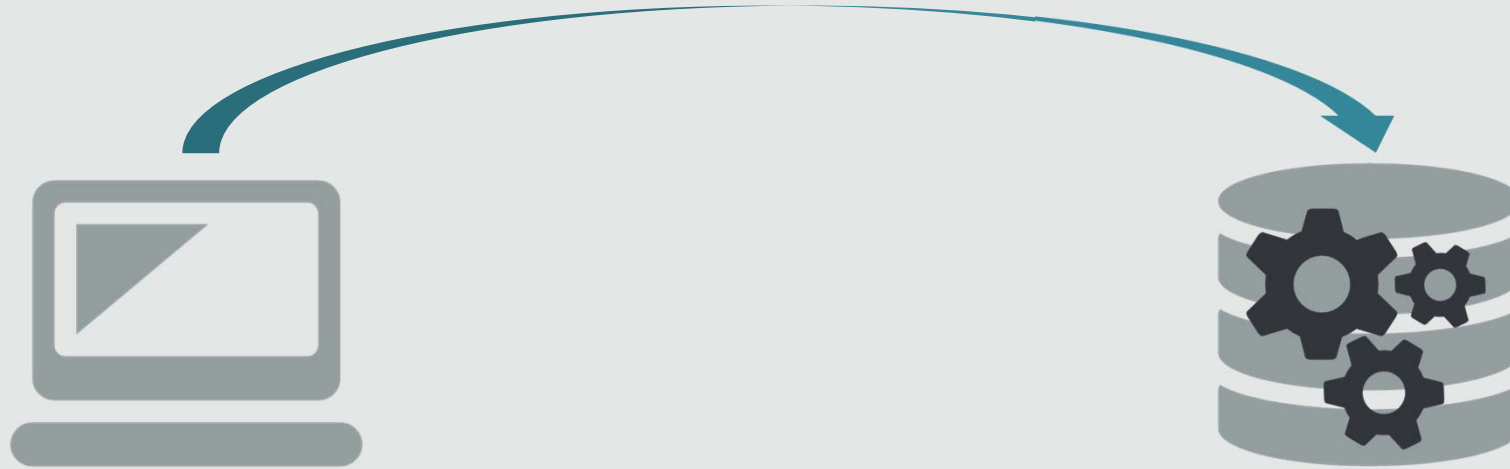
# Définition d'un bloc PL/SQL



Les blocs PL / SQL

# Le bloc interne

Envoi une requête avec le nom du bloc à exécuter

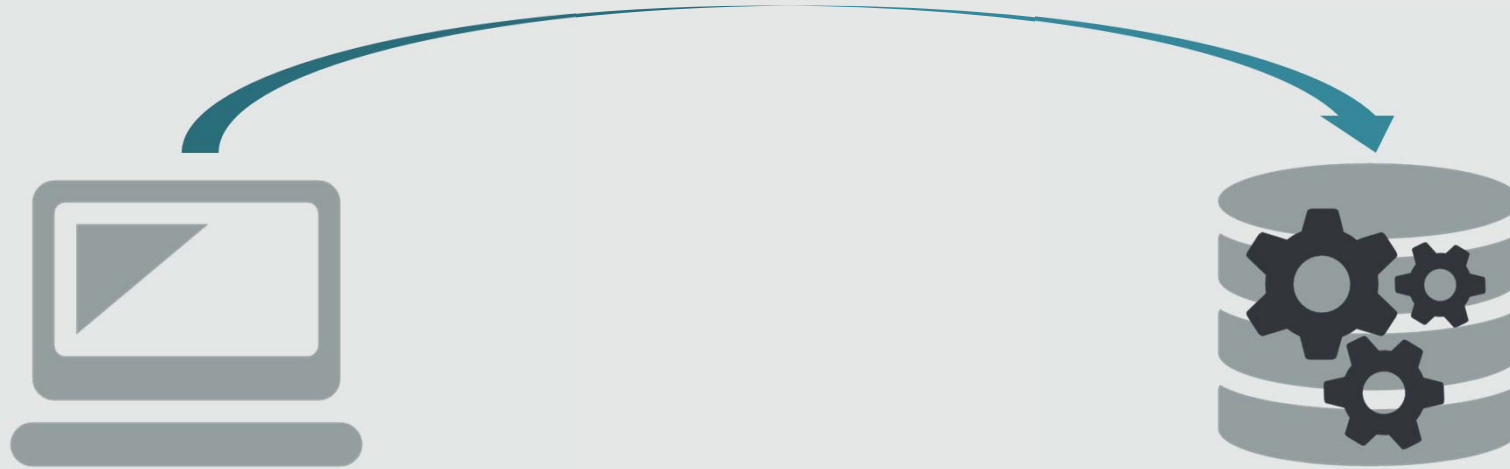


Le bloc est stocké et exécuté sur le serveur

Les blocs PL / SQL

## Le bloc externe

Envoi une requête avec le contenu du bloc à exécuter



Le bloc est exécuté sur le serveur

Les blocs PL / SQL

# Le plus petit bloc du monde

```
BEGIN  
    NULL;  --Traitement  
END;  
/
```

# La section DECLARE

```
DECLARE
    --Declaration des variables
BEGIN
    NULL; --Traitement
END;
/
```



# La section EXCEPTION

```
DECLARE
    --Déclaration des variables
BEGIN
    NULL; --Traitement
EXCEPTION
    --Section de gestion des erreurs
END;
/
```

# Conclusion

- Vous savez qu'un bloc PL/SQL est un programme
- Vous savez que les blocs PL/SQL sont exécutés sur le serveur de BDD
- Vous savez qu'un bloc associé à un nom est un bloc interne
- Vous savez qu'un bloc sans nom est un bloc externe ou anonyme
- Vous connaissez les sections DECLARE, BEGIN et EXCEPTION
- Vous savez que les sections DECLARE et EXCEPTION sont facultatives

# PL / SQL

## Module 4 – La section DECLARE



# Objectifs

- Connaître les types simples de variables
- Savoir déclarer une variable



La section DECLARE

# Les types de données

Tous les types SQL

BOOLEAN

PLS\_INTEGER

Sous-types



La section DECLARE

# La déclaration d'une variable

```
nom_de_la_variable [CONSTANT] TYPE [NOT NULL] [:= expression];
```



La section DECLARE

# Des exemples de déclarations

```
DECLARE
    compteur PLS_INTEGER;
    maximum CONSTANT PLS_INTEGER := 500;
    resultat BOOLEAN NOT NULL := TRUE;
    prenom VARCHAR2(30) := 'Anthony';
BEGIN
    NULL; --Traitement
END;
/
```

# Conclusion

- Vous connaissez les types simples de variables
- Vous savez déclarer une variable
- Vous avez découvert les types BOOLEAN et PLS\_INTEGER
- Vous savez qu'il existe des sous-types
- Vous savez que vous pouvez créer vos propres sous types





# PL / SQL

## Module 5 – La section BEGIN



# Objectifs

- Savoir utiliser des variables
- Savoir utiliser des structures de contrôle
- Savoir utiliser des types complexes
- Savoir afficher des messages pour déboguer

La section BEGIN

# Les commentaires de code

```
/* Je suis  
    un commentaire sur  
    plusieurs lignes */
```

```
--Je suis un commentaire sur une ligne
```

## La section BEGIN

# L'affectation



INTO

```
nombre_de_mots := 14540;
```

```
SELECT count(*) INTO total FROM regions;
```

La section BEGIN

# Le package DBMS\_OUTPUT

DBMS\_OUTPUT pour déboguer

```
DBMS_OUTPUT.PUT_LINE('Valeur de la variable nommée code_agence : ' || code_agence );
```

La commande **SET SERVEROUTPUT ON** active les fonctions du package DBMS\_OUTPUT.



La section BEGIN

# Le traitement conditionnel IF ... THEN ... ELSE

```
DECLARE
    couleur_drapeau VARCHAR(30) := 'VERT';
BEGIN
    IF couleur_drapeau = 'ROUGE' THEN
        | DBMS_OUTPUT.PUT_LINE('Baignade interdite');
    ELSIF couleur_drapeau = 'ORANGE' THEN
        | DBMS_OUTPUT.PUT_LINE('Baignade dangereuse');
    ELSE
        | DBMS_OUTPUT.PUT_LINE('Baignade autorisée youhouuu ! :)');
    END IF;
END;
```

La section BEGIN

# Le traitement conditionnel CASE avec valeur

```
DECLARE
    couleur_drapeau VARCHAR(30) := 'VERT';
BEGIN
    CASE couleur_drapeau
        WHEN 'ROUGE' THEN
            DBMS_OUTPUT.PUT_LINE('Baignade interdite !!');
        WHEN 'ORANGE' THEN
            DBMS_OUTPUT.PUT_LINE('Attention, la mer est dangereuse');
        WHEN 'VERT' THEN
            DBMS_OUTPUT.PUT_LINE('Tous à l'eau !!');
        WHEN 'NOIR' THEN
            DBMS_OUTPUT.PUT_LINE('Marée noire');
        ELSE
            DBMS_OUTPUT.PUT_LINE('Drapeau non repertorié');
    END CASE;
END;
```

La section BEGIN

# Le traitement conditionnel CASE avec condition

```
DECLARE
    couleur_drapeau VARCHAR(30) := 'VERT';
BEGIN
    CASE
        WHEN couleur_drapeau = 'VERT' THEN
            DBMS_OUTPUT.PUT_LINE('Le drapeau est vert. ');
        WHEN couleur_drapeau = 'ORANGE' THEN
            DBMS_OUTPUT.PUT_LINE('Le drapeau est orange ');
        WHEN couleur_drapeau = 'ROUGE' THEN
            DBMS_OUTPUT.PUT_LINE('Le drapeau est rouge. ');
        ELSE
            DBMS_OUTPUT.PUT_LINE('Je ne connais pas cette couleur. ');
        END CASE;
    END;
```



La section BEGIN

# Le traitement répétitif LOOP

```
DECLARE
    index_loop INTEGER := 0;
BEGIN
    LOOP
        IF index_loop = 3 THEN
            EXIT;
        ELSE
            DBMS_OUTPUT.PUT_LINE(index_loop);
        END IF;
        index_loop := index_loop + 1;
    END LOOP;
END;
```

La section BEGIN

# Le traitement répétitif FOR

```
BEGIN
    FOR i IN 0..3 LOOP
        DBMS_OUTPUT.PUT_LINE(i);
    END LOOP;
END;
```

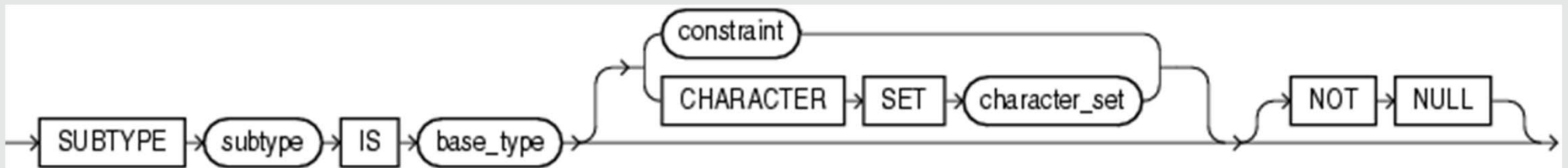
La section BEGIN

# Le traitement répétitif WHILE

```
DECLARE
    index_while INTEGER := 0;
BEGIN
    WHILE (index_while < 3) LOOP
        DBMS_OUTPUT.PUT_LINE(index_while);
        index_while := index_while + 1;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('FIN : index_while = ' || index_while);
END;
```

La section BEGIN

# Les sous-types



La section BEGIN

# Les types définis par l'utilisateur

```
SUBTYPE nom_sous_type IS type_de_base[(contrainte)][NOT NULL];
```



La section BEGIN

# Exemple de types définis par l'utilisateur

```
SET SERVEROUTPUT ON;

DECLARE
    SUBTYPE salaire IS NUMBER(4);
    SUBTYPE date_nn IS DATE NOT NULL;

    premiere_annee salaire := 2000;
    premier_jour date_nn := SYSDATE;

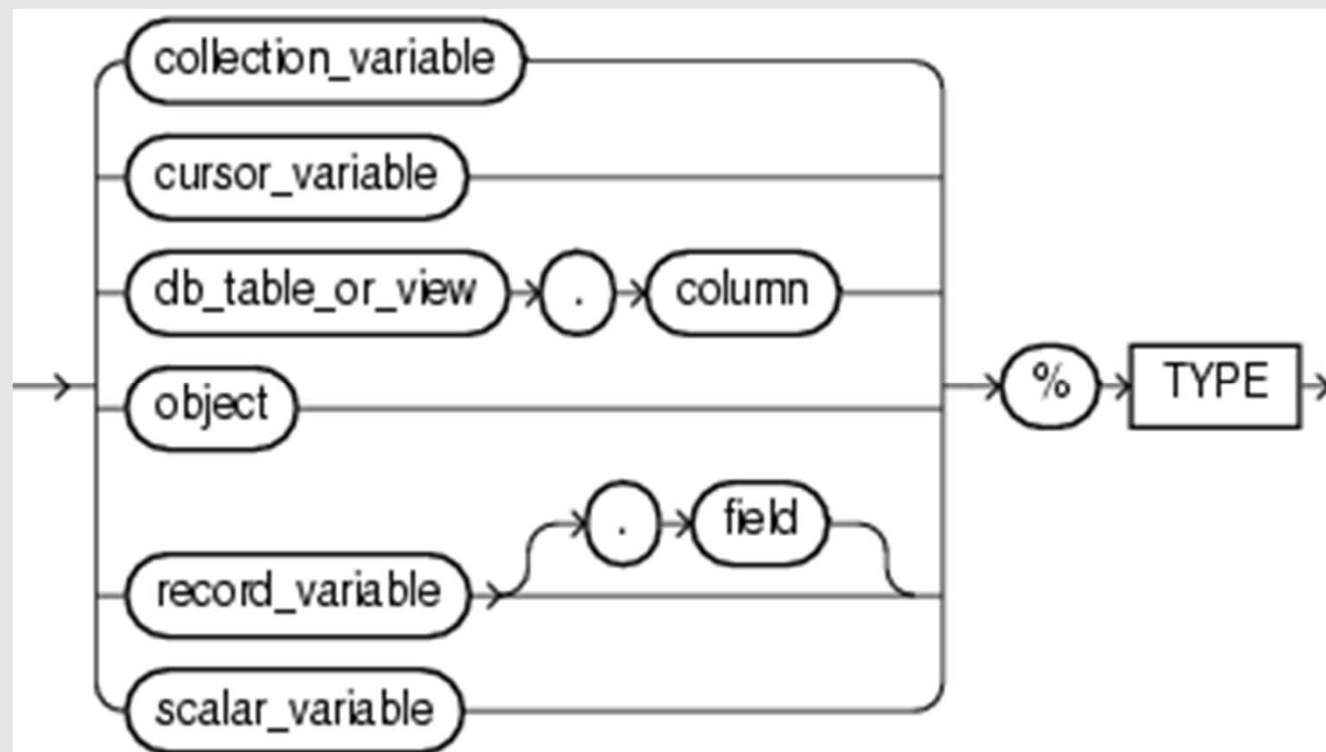
BEGIN

    DBMS_OUTPUT.PUT_LINE(premiere_annee);
    DBMS_OUTPUT.PUT_LINE(premier_jour);

END;
```

La section BEGIN

# L'attribut %TYPE



La section BEGIN

# L'attribut %TYPE

```
nom_variable nom_table.nom_colonne%TYPE;
```





La section BEGIN

# Exemple d'utilisation de l'attribut %TYPE

```
SET SERVEROUTPUT ON;

DECLARE

    continent regions.region_name%TYPE;

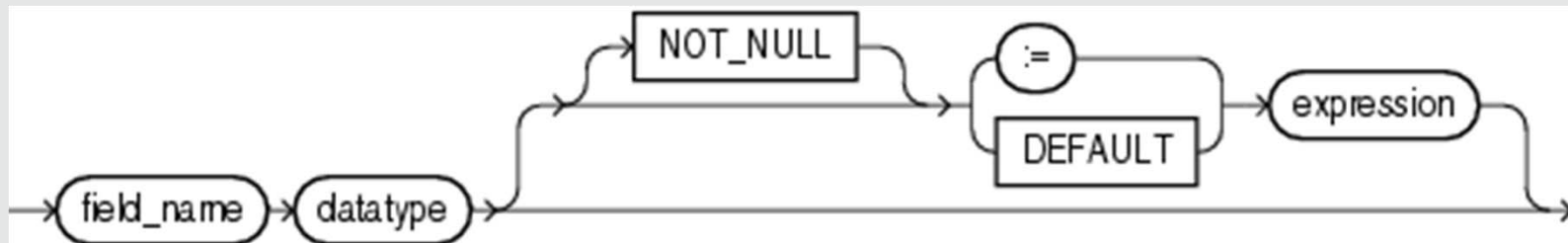
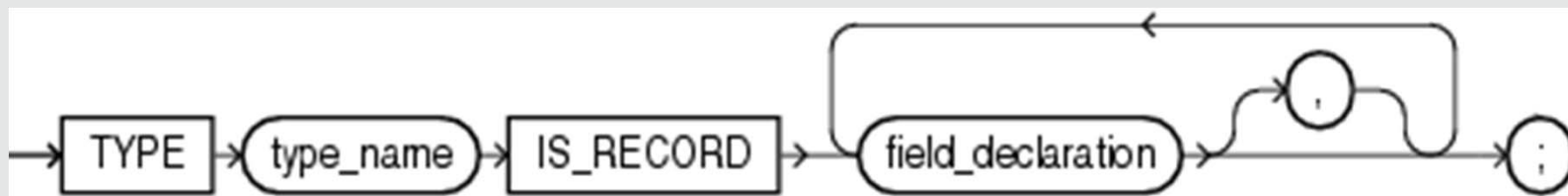
BEGIN

    continent := 'Europe';
    DBMS_OUTPUT.PUT_LINE(continent);

END;
```

La section BEGIN

# Les enregistrements de type RECORD



La section BEGIN

# Exemple d'utilisation du type RECORD

```
SET SERVEROUTPUT ON;

DECLARE

    TYPE fiche_parking IS RECORD
    (
        prenom VARCHAR2(50),
        nom VARCHAR2(50) NOT NULL := 'XXX',
        nombre NUMBER NOT NULL DEFAULT 1
    );

    fiche1 fiche_parking;

BEGIN

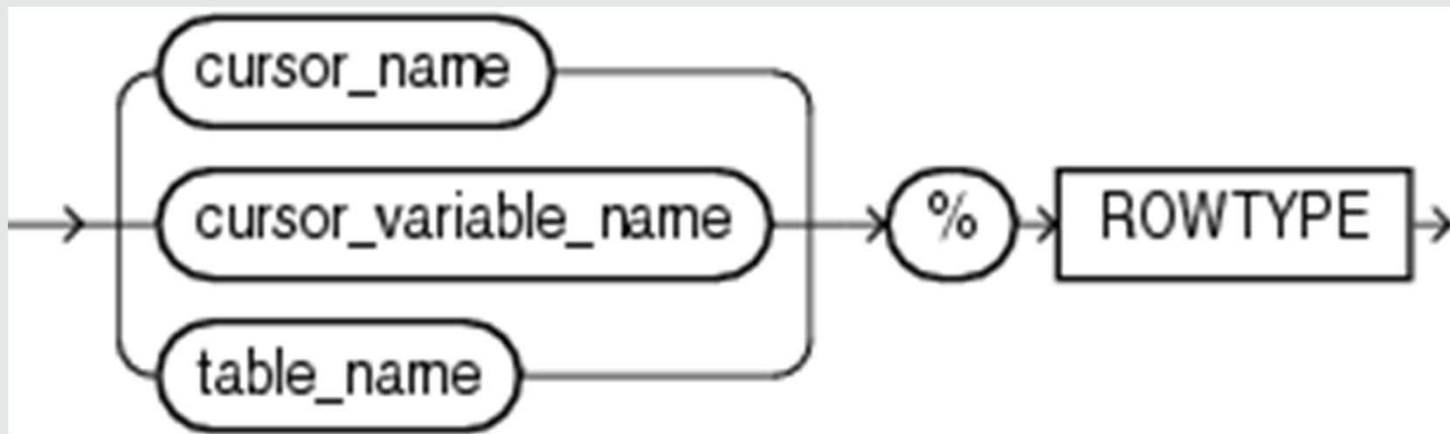
    fiche1.prenom := 'Anthony';
    fiche1.nom := 'Cosson';

    DBMS_OUTPUT.PUT_LINE('Prenom : ' || fiche1.prenom);
    DBMS_OUTPUT.PUT_LINE('Nom : ' || fiche1.nom);
    DBMS_OUTPUT.PUT_LINE('Nombre voiture : ' || fiche1.nombre);

END;
```

La section BEGIN

# L'attribut %ROWTYPE



La section BEGIN

# Exemple d'utilisation de l'attribut %ROWTYPE

```
SET SERVEROUTPUT ON;

DECLARE

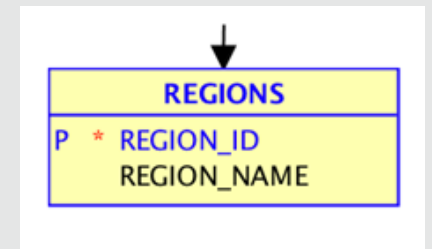
    region_rec regions%ROWTYPE;

BEGIN

    region_rec.region_id = 5;
    region_rec.region_name = 'Antarctique';

    DBMS_OUTPUT.PUT_LINE('Id de la region : ' || region_rec.region_id);
    DBMS_OUTPUT.PUT_LINE('Nom de la region : ' || region_rec.region_name);

END;
```



La section BEGIN

# Les collections

| TYPE   | Nombre d'éléments | Indexation     | Doit être initialisé |
|--|-------------------|----------------|----------------------|
| INDEX BY TABLE (Tableau associatif (clé/valeur)) | Illimité          | Alphanumérique | Non                  |
| NESTED TABLE (Tableau imbriqué)                  | Illimité          | Entier         | Oui                  |
| VARRAY (Taille variable)                         | Limité            | Entier         | Non                  |



La section BEGIN

# La collection de type INDEX BY TABLE

Ensemble de paires clé-valeur

Clé numérique ou alpha numérique

Taille dynamique

Stockage temporaire de données



La section BEGIN

# Exemple de collection INDEX BY TABLE

```
SET SERVEROUTPUT ON;

DECLARE

    TYPE population IS TABLE OF NUMBER INDEX BY VARCHAR(50);

    population_ville population;

    ville VARCHAR2(50);
BEGIN

    population_ville('Soliers') := 2151;
    population_ville('Caen') := 106538;
    population_ville('Limoges') := 134577;
    population_ville('Rennes') := 213454;

    ville := population_ville.FIRST;

    WHILE ville IS NOT NULL LOOP
        DBMS_Output.PUT_LINE('La population de ' || ville
        || ' est de ' || TO_CHAR(population_ville(ville)) || ' habitants');

        ville := population_ville.NEXT(ville);
    END LOOP;
END;
```



La section BEGIN

# La collection de type NESTED TABLE

Taille dynamique

Index numérique

Données parsemées

|                          |      |      |      |      |      |      |      |      |       |                         |
|--------------------------|------|------|------|------|------|------|------|------|-------|-------------------------|
| <b>Array of Integers</b> |      |      |      |      |      |      |      |      |       | Fixed<br>Upper<br>Bound |
| 321                      | 17   | 99   | 407  | 83   | 622  | 105  | 19   | 67   | 278   |                         |
| x(1)                     | x(2) | x(3) | x(4) | x(5) | x(6) | x(7) | x(8) | x(9) | x(10) |                         |

|                                     |  |      |      |  |      |      |      |  |       |                |
|-------------------------------------|--|------|------|--|------|------|------|--|-------|----------------|
| <b>Nested Table after Deletions</b> |  |      |      |  |      |      |      |  |       | Unbounded<br>→ |
| 321                                 |  | 99   | 407  |  | 622  | 105  | 19   |  | 278   |                |
| x(1)                                |  | x(3) | x(4) |  | x(6) | x(7) | x(8) |  | x(10) |                |

La section BEGIN

# Exemple de collection NESTED TABLE

```
SET SERVEROUTPUT ON;

DECLARE

    TYPE tableau_noms_objets IS TABLE OF VARCHAR2(50);

    tableau_musique tableau_noms_objets;

    objet_courant VARCHAR2(50);
BEGIN

    tableau_musique := tableau_noms_objets('Guitare','Tambour','Batterie','Basse');

    tableau_musique.DELETE(2);

    DBMS_OUTPUT.PUT_LINE(tableau_musique(1));
    --DBMS_OUTPUT.PUT_LINE(tableau_musique(2));
    DBMS_OUTPUT.PUT_LINE(tableau_musique(3));

END;
```

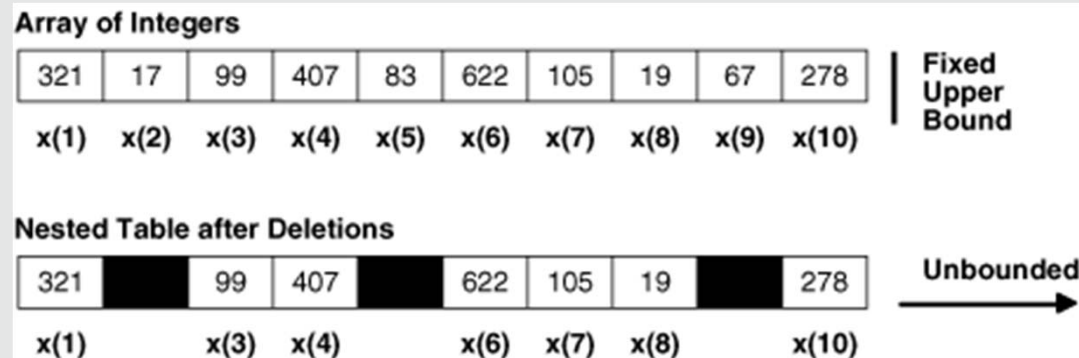
La section BEGIN

# La collection de type VARRAY

Taille fixe

Index numérique

Données denses



La section BEGIN

# Exemple de collection VARRAY

```
SET SERVEROUTPUT ON;

DECLARE

    TYPE tableau_couleur IS VARRAY(5) OF VARCHAR2(50);

    france tableau_couleur := tableau_couleur('Bleu','Blanc','Rouge');

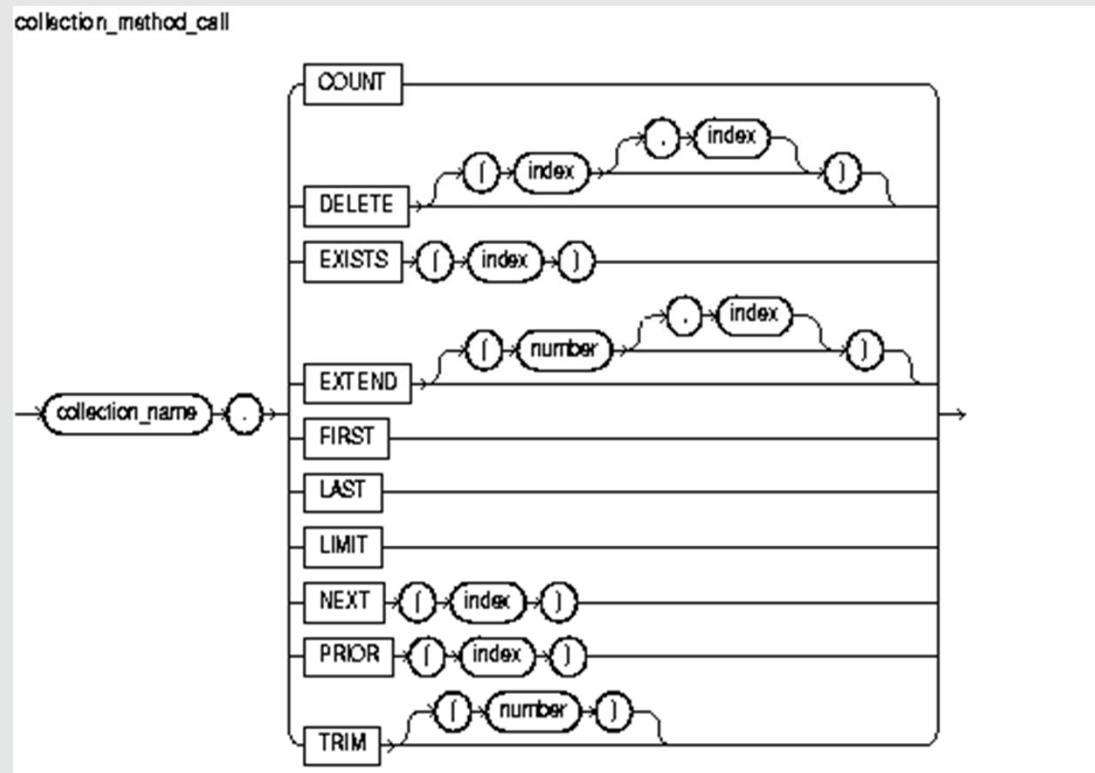
BEGIN

    DBMS_OUTPUT.PUT_LINE(france(1));
    DBMS_OUTPUT.PUT_LINE(france(2));
    DBMS_OUTPUT.PUT_LINE(france(3));

END;
```

La section BEGIN

# Méthodes associées aux collections



La section BEGIN

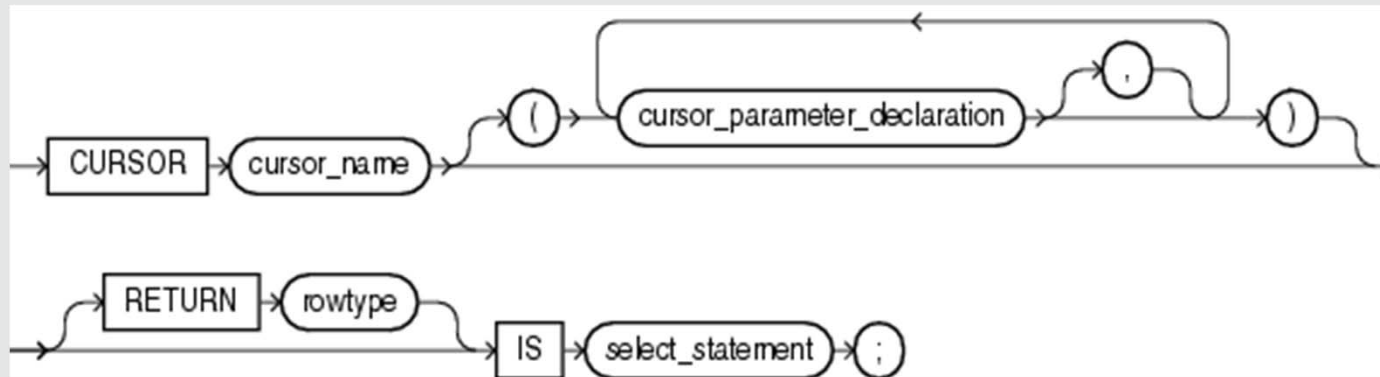
# Le curseur explicite

Un curseur permet de stocker le résultat d'une requête  
afin de le traiter ligne par ligne



La section BEGIN

# La déclaration d'un curseur explicite



La section BEGIN

# L'utilisation du curseur explicite : 1<sup>ère</sup> méthode

Fonctionnement :

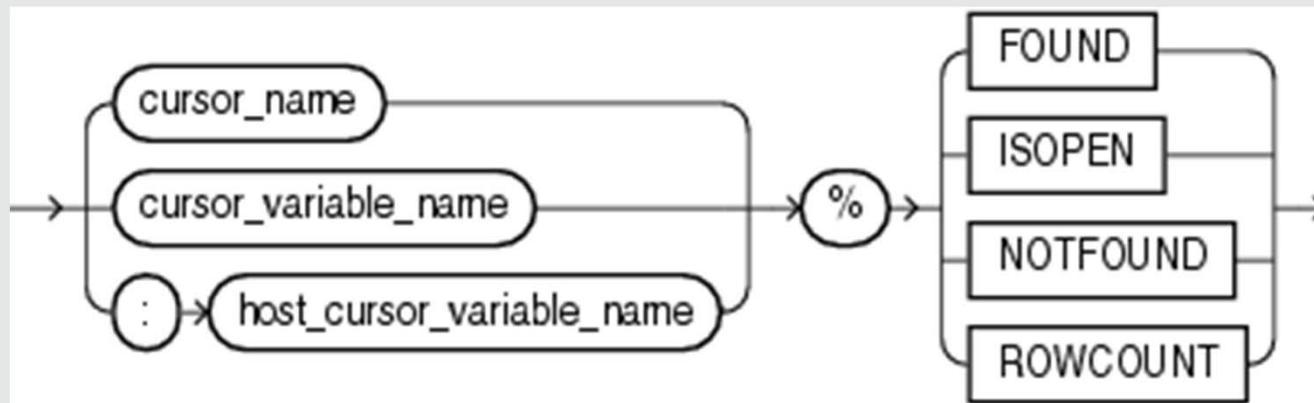
1. Déclaration du curseur
2. Ouverture du curseur
3. Récupération du résultat ligne par ligne
4. Fermeture du curseur





La section BEGIN

## Les attributs de curseur



La section BEGIN

# L'utilisation du curseur explicite : 1<sup>ère</sup> méthode

```
SET SERVEROUTPUT ON;

DECLARE
    CURSOR cursor_employees_it IS SELECT * FROM EMPLOYEES WHERE DEPARTMENT_ID = 60;

    employee employees%ROWTYPE;
BEGIN
    OPEN cursor_employees_it;

    LOOP
        FETCH cursor_employees_it INTO employee;
        DBMS_OUTPUT.PUT_LINE(employee.last_name);
        EXIT WHEN cursor_employees_it%NOTFOUND;
    END LOOP;

    CLOSE cursor_employees_it;
END;
```

La section BEGIN

# L'utilisation du curseur explicite : 2<sup>ème</sup> méthode

```
FOR ... IN ... LOOP
```



La section BEGIN

# L'utilisation du curseur explicite : 2<sup>ème</sup> méthode

```
SET SERVEROUTPUT ON;

DECLARE
    CURSOR cursor_employees_it IS SELECT first_name,last_name FROM EMPLOYEES WHERE DEPARTMENT_ID = 60;

    employee employees%ROWTYPE;
BEGIN

    FOR employee IN cursor_employees_it LOOP
        DBMS_OUTPUT.PUT_LINE(employee.first_name);
    END LOOP;

END;
```

La section BEGIN

# Le verrou d'intention FOR UPDATE

Permet de verrouiller la table ou les champs que l'on va mettre à jour à partir du curseur implicite



La section BEGIN

# Le verrou d'intention FOR UPDATE

```
SET SERVEROUTPUT ON;

DECLARE
    CURSOR cursor_employees_it IS SELECT * FROM EMPLOYEES WHERE DEPARTMENT_ID = 60 FOR UPDATE;

BEGIN

    FOR employee IN cursor_employees_it LOOP

        UPDATE employees SET last_name = UPPER(last_name) WHERE employee_id = employee.employee_id;
        DBMS_OUTPUT.PUT_LINE(employee.first_name || ' ' || employee.last_name);

    END LOOP;

    COMMIT;
END;
```

PL / SQL

# Le verrou d'intention FOR UPDATE OF

```
SET SERVEROUTPUT ON;

DECLARE
    CURSOR cursor_employees_it IS SELECT * FROM EMPLOYEES WHERE DEPARTMENT_ID = 60 FOR UPDATE OF last_name;
BEGIN

    FOR employee IN cursor_employees_it LOOP
        UPDATE employees SET last_name = UPPER(last_name) WHERE employee_id = employee.employee_id;
        DBMS_OUTPUT.PUT_LINE(employee.first_name || ' ' || employee.last_name);
    END LOOP;

    COMMIT;
END;
```

La section BEGIN

# La clause WHERE CURRENT OF du curseur explicite

Permet de simplifier la clause WHERE  
des requêtes de modification





La section BEGIN

# Exemple d'utilisation de la clause WHERE CURRENT OF

```
SET SERVEROUTPUT ON;

DECLARE
    CURSOR cursor_employees_it IS SELECT * FROM EMPLOYEES WHERE DEPARTMENT_ID = 60 FOR UPDATE OF last_name;
BEGIN

    FOR employee IN cursor_employees_it LOOP
        UPDATE employees SET last_name = UPPER(last_name) WHERE CURRENT OF cursor_employees_it;
        DBMS_OUTPUT.PUT_LINE(employee.first_name || ' ' || employee.last_name);
    END LOOP;

    COMMIT;

END;
```

La section BEGIN

# Le curseur explicite paramétrable

Un curseur peut être paramétrable



La section BEGIN

# Exemple de curseur explicite paramétrable

```
SET SERVEROUTPUT ON;

DECLARE
  CURSOR cursor_employees_it(dep NUMBER) IS SELECT * FROM EMPLOYEES WHERE DEPARTMENT_ID = dep FOR UPDATE OF last_name;
BEGIN

  FOR employee IN cursor_employees_it(60) LOOP
    UPDATE employees SET last_name = UPPER(last_name) WHERE CURRENT OF cursor_employees_it;
    DBMS_OUTPUT.PUT_LINE(employee.first_name || ' ' || employee.last_name);
  END LOOP;

  COMMIT;

END;
```



La section BEGIN

# L'utilisation du curseur explicite : 3<sup>ème</sup> méthode

```
FOR ... IN ... LOOP
```



La section BEGIN

# L'utilisation curseur explicite : 3<sup>ème</sup> méthode

```
SET SERVEROUTPUT ON;

BEGIN

  FOR employee IN (SELECT first_name,last_name FROM EMPLOYEES WHERE DEPARTMENT_ID = 60) LOOP

    DBMS_OUTPUT.PUT_LINE(employee.first_name);

  END LOOP;

END;
```

La section BEGIN

# Le curseur implicite

- C'est un curseur de session déclaré et géré implicitement par PL/SQL.
- Il contient des informations à propos des dernières instructions DML effectuées.
- Utilisation :
  - SQL%FOUND
  - SQL%ROWCOUNT



La section BEGIN

# Bloc anonyme

## Démonstration



# Conclusion

- Vous connaissez des types de données complexes
- Vous savez définir et manipuler un tableau
- Vous savez définir et manipuler une collection
- Vous connaissez la syntaxe des structures de contrôle du langage PL/SQL
- Vous savez définir et manipuler un curseur explicite
- Vous connaissez les attributs de curseur



# PL / SQL

## Module 6 – La section EXCEPTION



# Objectifs

- Savoir gérer les erreurs de programmation
- Savoir gérer les anomalies utilisateurs



La section EXCEPTION

# L'emplacement de la section Exception

```
DECLARE
    --Déclaration des variables
BEGIN
    --Traitement
EXCEPTION
    --Gestion des erreurs
END;
```

La section EXCEPTION

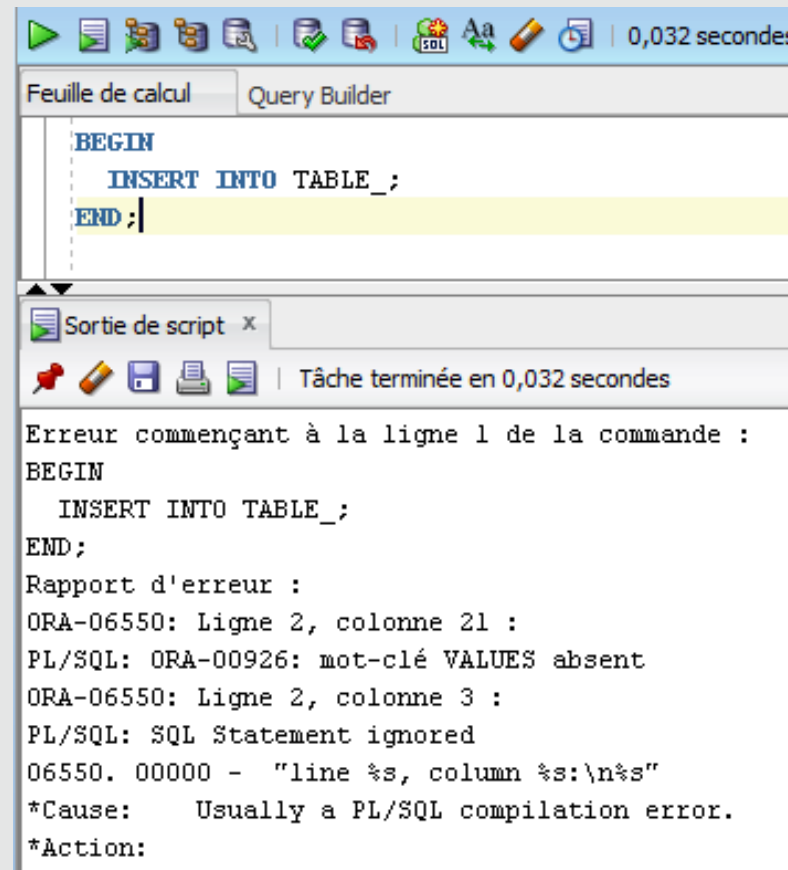
# Les différents types d'erreurs

- Erreur de compilation
- Erreur d'exécution
- Erreur utilisateur



La section EXCEPTION

# Les erreurs de compilation



The screenshot shows the Oracle SQL Developer interface. The top toolbar includes icons for running, saving, and other database operations, with a timer showing 0,032 secondes. Below the toolbar, there are two tabs: 'Feuille de calcul' and 'Query Builder'. The 'Query Builder' tab is active, displaying a SQL script in a text area. The script is as follows:

```
BEGIN
  INSERT INTO TABLE_ ;
END ;
```

Below the script, there is a 'Sortie de script' (Script Output) window. It shows the execution status: 'Tâche terminée en 0,032 secondes'. Below this, it displays an error message:

```
Erreur commençant à la ligne 1 de la commande :
BEGIN
  INSERT INTO TABLE_ ;
END ;
Rapport d'erreur :
ORA-06550: Ligne 2, colonne 21 :
PL/SQL: ORA-00926: mot-clé VALUES absent
ORA-06550: Ligne 2, colonne 3 :
PL/SQL: SQL Statement ignored
06550. 00000 - "line %s, column %s:\n%s"
*Cause:      Usually a PL/SQL compilation error.
*Action:
```

La section EXCEPTION

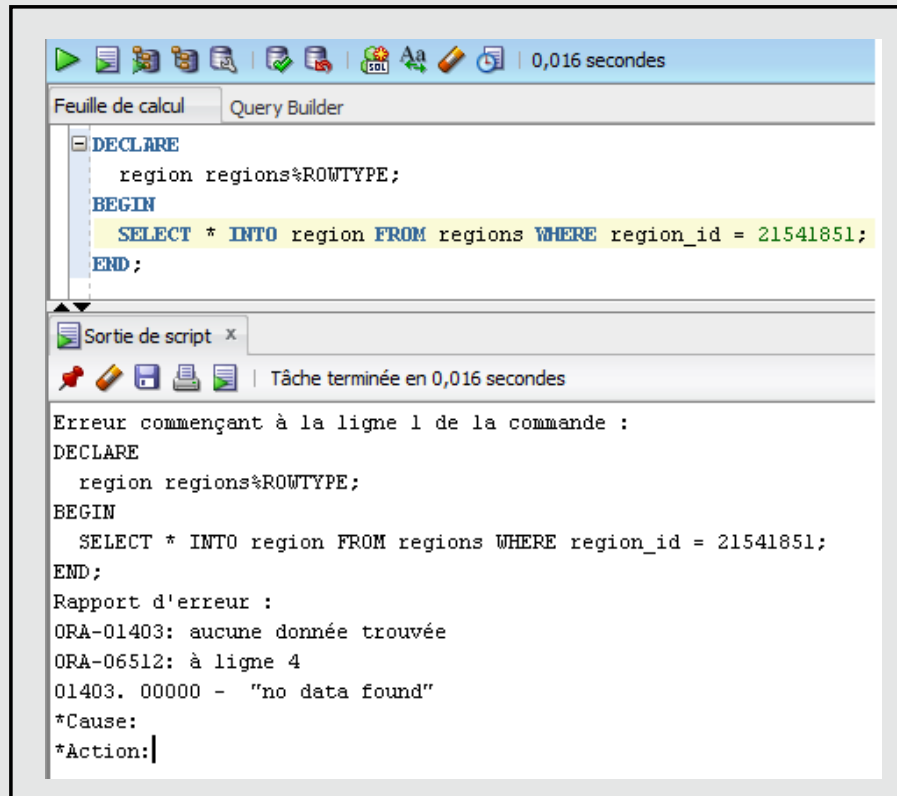
# Les erreurs d'exécution prédéfinies

- Erreur Oracle
- Exception numérotée
- Exception nommée



La section EXCEPTION

# Les erreurs d'exécution prédéfinies



Feuille de calcul | Query Builder | 0,016 secondes

```
DECLARE
    region regions%ROWTYPE;
BEGIN
    SELECT * INTO region FROM regions WHERE region_id = 21541851;
END;
```

Sortie de script x | Tâche terminée en 0,016 secondes

Erreur commençant à la ligne 1 de la commande :

```
DECLARE
    region regions%ROWTYPE;
BEGIN
    SELECT * INTO region FROM regions WHERE region_id = 21541851;
END;
```

Rapport d'erreur :

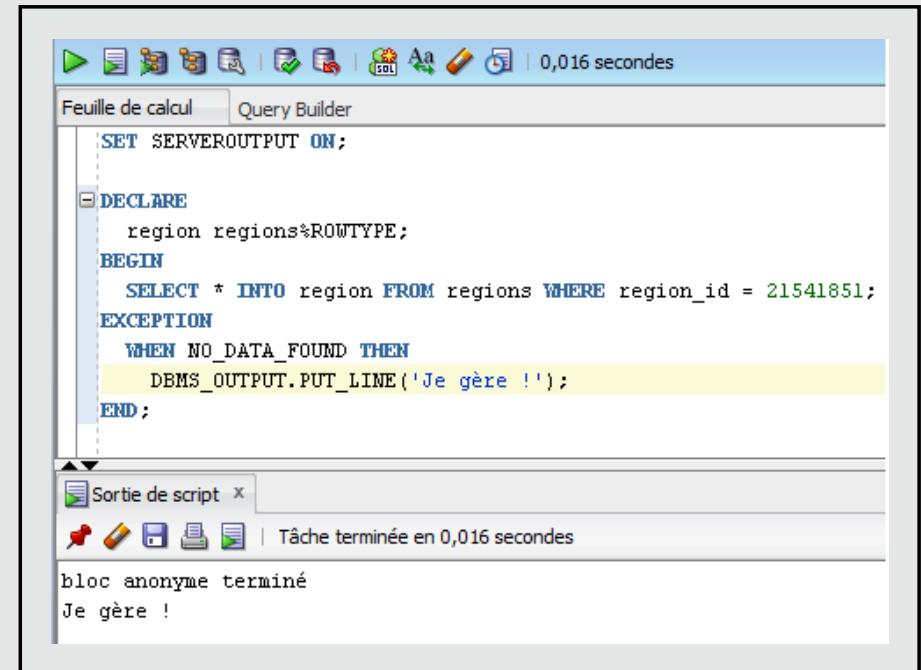
ORA-01403: aucune donnée trouvée

ORA-06512: à ligne 4

01403. 00000 - "no data found"

\*Cause:

\*Action:



Feuille de calcul | Query Builder | 0,016 secondes

```
SET SERVEROUTPUT ON;
```

```
DECLARE
    region regions%ROWTYPE;
BEGIN
    SELECT * INTO region FROM regions WHERE region_id = 21541851;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Je gère !');
END;
```

Sortie de script x | Tâche terminée en 0,016 secondes

bloc anonyme terminé

Je gère !

La section EXCEPTION

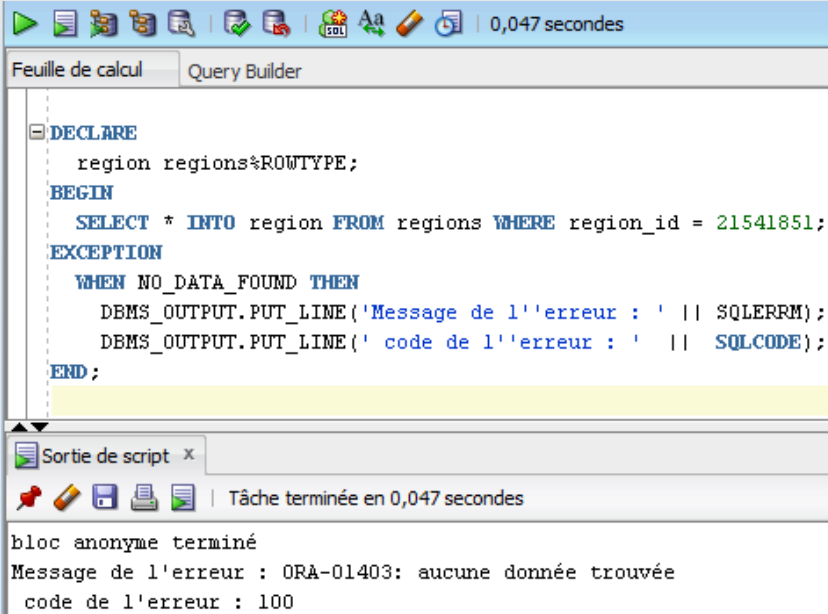
# La liste des exceptions prédéfinies

| NOM                     | NUMERO    | SQLCODE |
|-------------------------|-----------|---------|
| ACCESS_INTO_NULL        | ORA-06530 | -6530   |
| CASE_NOT_FOUND          | ORA-06592 | -6592   |
| COLLECTION_IS_NULL      | ORA-06531 | -6531   |
| CURSOR_ALREADY_OPENED   | ORA-06511 | -6511   |
| DUP_VAL_ON_INDEX        | ORA-00001 | -1      |
| INVALID_CURSOR          | ORA-01001 | -1001   |
| INVALID_NUMBER          | ORA-01722 | -1722   |
| NO_DATA_FOUND           | ORA-01403 | +100    |
| PROGRAM_ERROR           | ORA-06501 | -6501   |
| ROWTYPE_MISMATCH        | ORA-06504 | -6504   |
| STORAGE_ERROR           | ORA-06500 | -6500   |
| SUBSCRIPT_BEYOND_COUNT  | ORA-06533 | -6533   |
| SUBSCRIPT_OUTSIDE_LIMIT | ORA-06532 | -6532   |
| SYS_INVALID_ROWID       | ORA-01410 | -1410   |
| TOO_MANY_ROWS           | ORA-01422 | -1422   |
| VALUE_ERROR             | ORA-06502 | -6502   |
| ZERO_DIVIDE             | ORA-01476 | -1476   |



La section EXCEPTION

# Les fonctions SQLEERM & SQLCODE



The screenshot shows the Oracle SQL Developer Query Builder interface. The main window displays an SQL script with the following code:

```
DECLARE
    region regions%ROWTYPE;
BEGIN
    SELECT * INTO region FROM regions WHERE region_id = 21541851;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Message de l'erreur : ' || SQLEERM);
        DBMS_OUTPUT.PUT_LINE(' code de l'erreur : ' || SQLCODE);
END;
```

Below the script editor, the 'Sortie de script' (Script Output) window is visible, showing the execution results:

```
bloc anonyme terminé
Message de l'erreur : ORA-01403: aucune donnée trouvée
code de l'erreur : 100
```

The top status bar indicates the execution time as 0,047 secondes. The bottom status bar indicates the task is completed in 0,047 secondes.

La section EXCEPTION

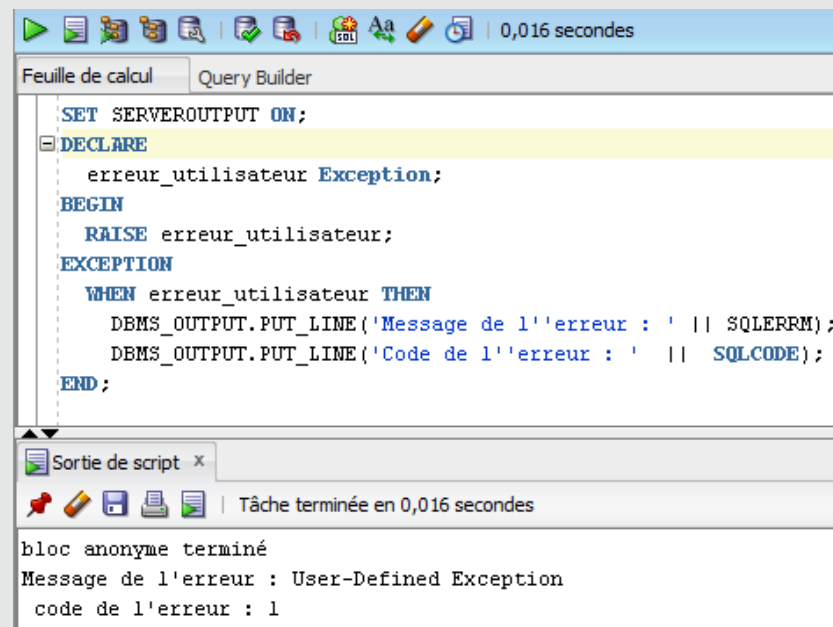
# Les erreurs utilisateur

- Définies grâce au type EXCEPTION
- Levées grâce au mot-clé RAISE
- Permettent de se protéger de traitements illogiques



La section EXCEPTION

# Les erreurs utilisateur



The screenshot shows a SQL query builder window with a toolbar at the top. The main area contains a script for handling a user-defined exception. The script is as follows:

```
SET SERVEROUTPUT ON;  
DECLARE  
    erreur_utilisateur Exception;  
BEGIN  
    RAISE erreur_utilisateur;  
EXCEPTION  
    WHEN erreur_utilisateur THEN  
        DBMS_OUTPUT.PUT_LINE('Message de l''erreur : ' || SQLERRM);  
        DBMS_OUTPUT.PUT_LINE('Code de l''erreur : ' || SQLCODE);  
END;
```

Below the script editor, there is a 'Sortie de script' (Script Output) window. It shows the execution status and the output of the script:

```
bloc anonyme terminé  
Message de l'erreur : User-Defined Exception  
code de l'erreur : 1
```

La section EXCEPTION

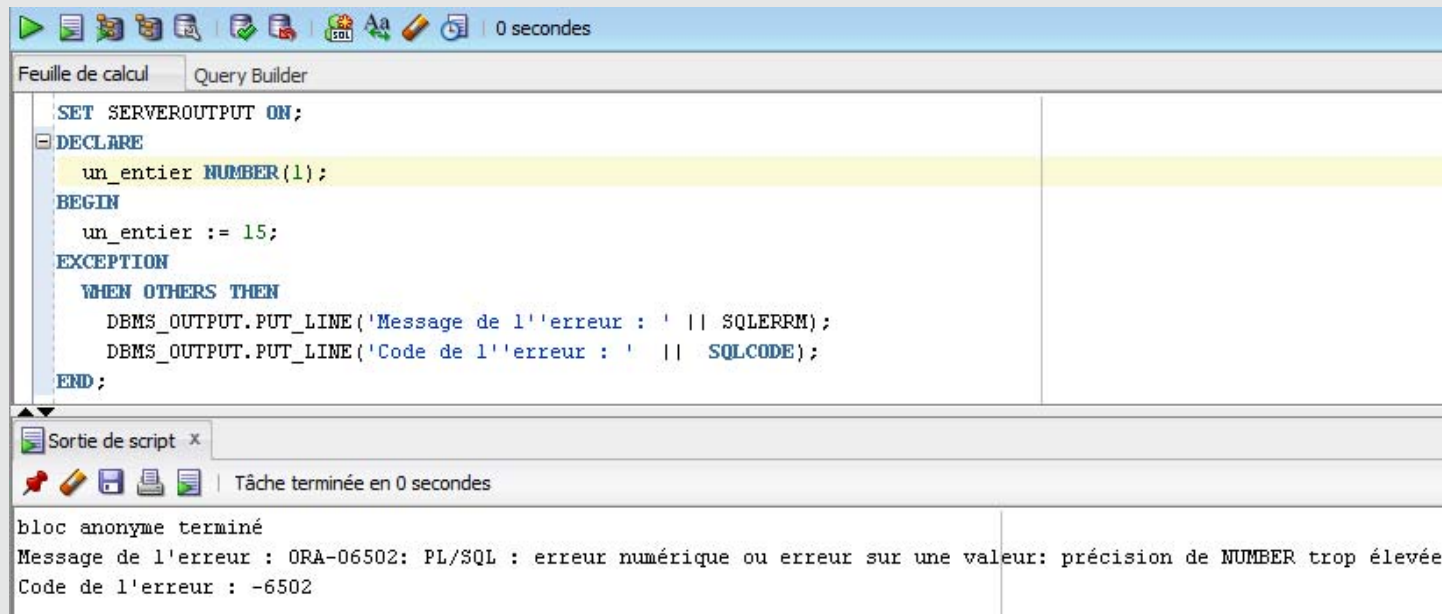
# Les erreurs d'exécution non prédéfinies

- Erreurs Oracle
- Définies par un code erreur numérique



La section EXCEPTION

# Les erreurs d'exécution non prédéfinies



The screenshot shows the Oracle SQL Developer interface. The top toolbar includes icons for running, saving, and other database operations, with a timer showing "0 secondes". The main window is titled "Feuille de calcul" and "Query Builder". It contains a PL/SQL script with the following code:

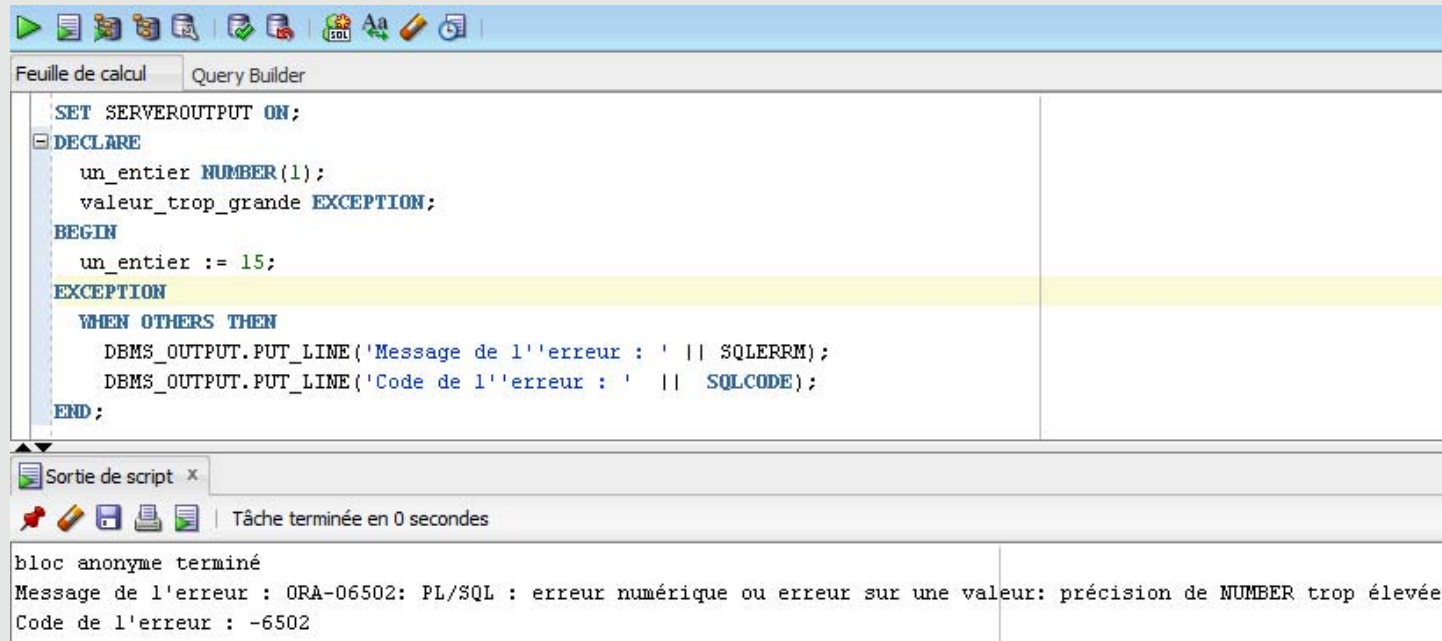
```
SET SERVEROUTPUT ON;  
DECLARE  
  un_entier NUMBER(1);  
BEGIN  
  un_entier := 15;  
EXCEPTION  
  WHEN OTHERS THEN  
    DBMS_OUTPUT.PUT_LINE('Message de l'erreur : ' || SQLERRM);  
    DBMS_OUTPUT.PUT_LINE('Code de l'erreur : ' || SQLCODE);  
END;
```

Below the script editor, there is a section titled "Sortie de script" with a timer showing "Tâche terminée en 0 secondes". It displays the execution results:

```
bloc anonyme terminé  
Message de l'erreur : ORA-06502: PL/SQL : erreur numérique ou erreur sur une valeur: précision de NUMBER trop élevée  
Code de l'erreur : -6502
```

La section EXCEPTION

# La directive de compilation PRAGMA EXCEPTION\_INIT



The screenshot shows the Oracle SQL Developer interface. The main window is titled 'Feuille de calcul' and 'Query Builder'. It contains a PL/SQL script with the following code:

```
SET SERVEROUTPUT ON;
DECLARE
  un_entier NUMBER(1);
  valeur_trop_grande EXCEPTION;
BEGIN
  un_entier := 15;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Message de l'erreur : ' || SQLERRM);
    DBMS_OUTPUT.PUT_LINE('Code de l'erreur : ' || SQLCODE);
END;
```

The script is executed, and the output is displayed in the 'Sortie de script' window. The output shows the message and code for the exception raised.

```
bloc anonyme terminé
Message de l'erreur : ORA-06502: PL/SQL : erreur numérique ou erreur sur une valeur: précision de NUMBER trop élevée
Code de l'erreur : -6502
```

La section EXCEPTION

# La directive de compilation PRAGMA EXCEPTION\_INIT

```
SET SERVEROUTPUT ON;
DECLARE
    un_entier NUMBER(1);
    valeur_trop_grande EXCEPTION;
    PRAGMA EXCEPTION_INIT(valeur_trop_grande,-6502);
BEGIN
    un_entier := 15;
EXCEPTION
    WHEN valeur_trop_grande THEN
        DBMS_OUTPUT.PUT_LINE('Nous sommes ici');
        DBMS_OUTPUT.PUT_LINE('Message de l''erreur : ' || SQLERRM);
        DBMS_OUTPUT.PUT_LINE('Code de l''erreur : ' || SQLCODE);
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Message de l''erreur : ' || SQLERRM);
        DBMS_OUTPUT.PUT_LINE('Code de l''erreur : ' || SQLCODE);
END;
```

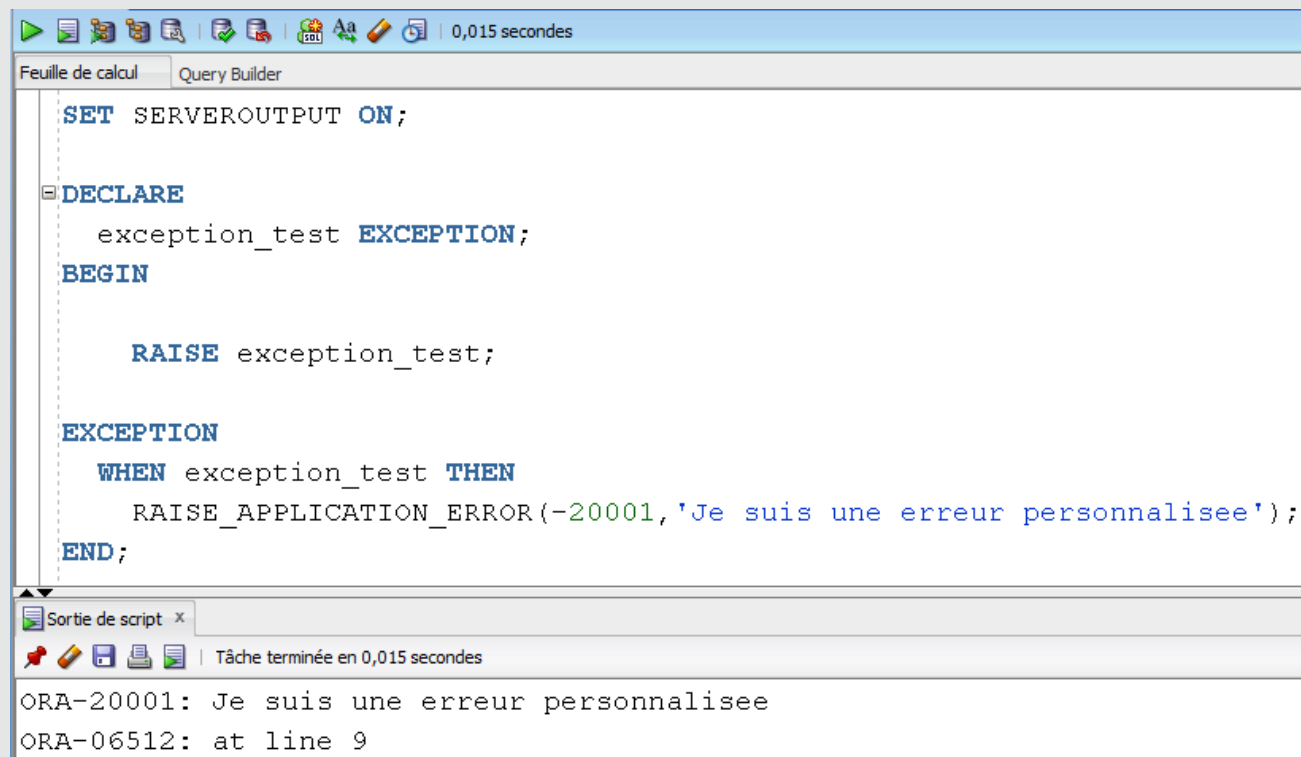
Sortie de script x

Tâche terminée en 0,015 secondes

bloc anonyme terminé  
Nous sommes ici  
Message de l'erreur : ORA-06502: PL/SQL : erreur numérique ou erreur sur une valeur: précision de NUMBER trop élevée  
Code de l'erreur : -6502

La section EXCEPTION

# La procédure RAISE\_APPLICATION\_ERROR



```
SET SERVEROUTPUT ON;

DECLARE
    exception_test EXCEPTION;
BEGIN

    RAISE exception_test;

EXCEPTION
    WHEN exception_test THEN
        RAISE_APPLICATION_ERROR(-20001, 'Je suis une erreur personnalisée');
END;
```

Sortie de script x

Tâche terminée en 0,015 secondes

ORA-20001: Je suis une erreur personnalisée  
ORA-06512: at line 9



La section EXCEPTION

# La propagation des exceptions 1/3

```
SET SERVEROUTPUT ON;

DECLARE
    exception_test EXCEPTION;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Je suis avant l''erreur');
    RAISE exception_test;
    DBMS_OUTPUT.PUT_LINE('Je suis apres l''erreur');
EXCEPTION
    WHEN exception_test THEN
        DBMS_OUTPUT.PUT_LINE('Je traite l''erreur');
END;
```

La section EXCEPTION

## La propagation des exceptions 2/3

```
SET SERVEROUTPUT ON;

DECLARE
    exception_test EXCEPTION;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Je suis avant le sous bloc');
    BEGIN
        DBMS_OUTPUT.PUT_LINE('Je suis avant l''erreur');
        RAISE exception_test;
        DBMS_OUTPUT.PUT_LINE('Je suis apres l''erreur');
    END;
    DBMS_OUTPUT.PUT_LINE('Je suis après le sous bloc');
EXCEPTION
    WHEN exception_test THEN
        DBMS_OUTPUT.PUT_LINE('Je traite l''erreur');
END;
```

La section EXCEPTION

# La propagation des exceptions 3/3

```
SET SERVEROUTPUT ON;

DECLARE
    exception_test EXCEPTION;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Je suis avant le sous bloc');
    BEGIN
        DBMS_OUTPUT.PUT_LINE('Je suis avant l''erreur');
        RAISE exception_test;
        DBMS_OUTPUT.PUT_LINE('Je suis apres l''erreur');
    EXCEPTION
        WHEN exception_test THEN
            DBMS_OUTPUT.PUT_LINE('Je traite l''erreur dans le sous bloc');
    END;
    DBMS_OUTPUT.PUT_LINE('Je suis après le sous bloc');
EXCEPTION
    WHEN exception_test THEN
        DBMS_OUTPUT.PUT_LINE('Je traite l''erreur');
END;
```

# Conclusion

- Vous savez ce qu'est une erreur utilisateur
- Vous savez ce qu'est une erreur prédéfinie & non prédéfinie
- Vous savez utiliser le type EXCEPTION
- Vous savez lever une erreur l'instruction avec RAISE
- Vous savez associer une erreur non prédéfinie à une variable EXCEPTION
- Vous savez associer un message à une erreur avec RAISE\_APPLICATION\_ERROR
- Vous connaissez le mécanisme de propagation des erreurs



# PL / SQL

## Module 7 – Les procédures stockées



# Objectifs

- Savoir créer une procédure stockée
- Savoir utiliser les différents types de paramètres pour les procédures stockées
- Savoir appeler une procédure stockée

Les procédures stockées

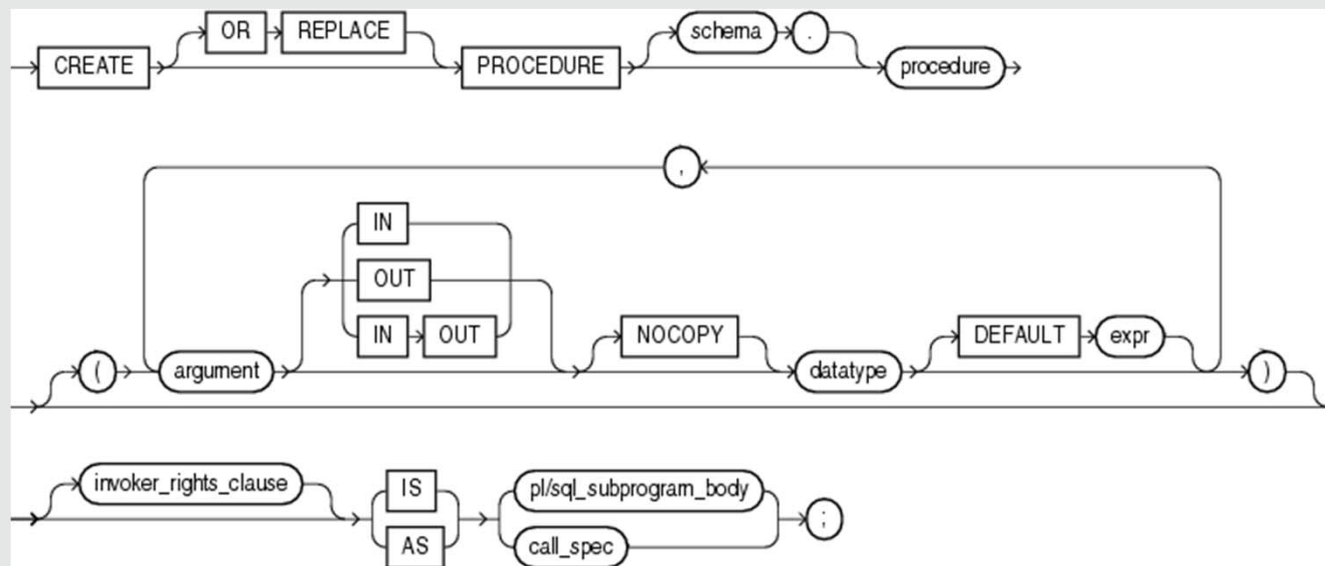
# Définition

- Programme défini par l'utilisateur
- Programme stocké sur le serveur
- Bloc PL/SQL associé à un nom



Les procédures stockées

# La syntaxe de création





Les procédures stockées

# Exemple de création d'une procédure stockée

```
SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE afficher_employes
AS
    CURSOR cursor_employees_it IS SELECT first_name,last_name FROM EMPLOYEES WHERE DEPARTMENT_ID = 60;
    employee employees%ROWTYPE;
BEGIN

    FOR employee IN cursor_employees_it LOOP
        DBMS_OUTPUT.PUT_LINE(employee.first_name);
    END LOOP;

END;
```

Les procédures stockées

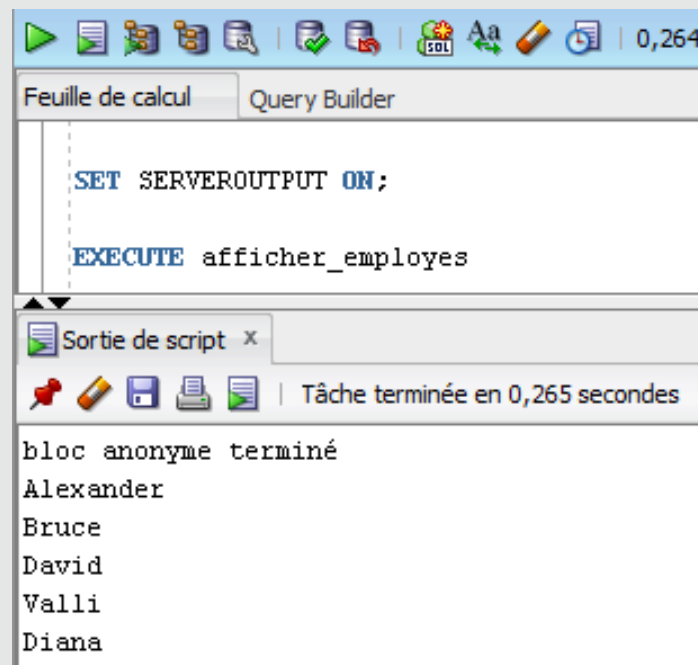
# L'appel à une procédure stockée

- En utilisant la commande EXECUTE
- En utilisant le nom de la procédure stockée



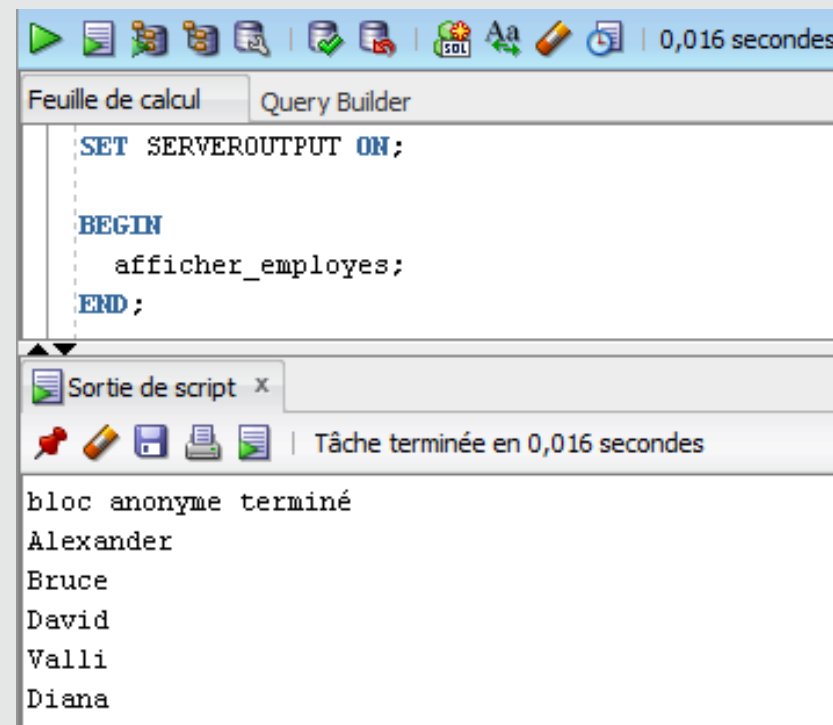
Les procédures stockées

# L'appel par la commande EXECUTE



Les procédures stockées

# L'appel à partir d'un autre bloc



The screenshot shows a SQL Query Builder window with a toolbar at the top. The main text area contains the following SQL code:

```
SET SERVEROUTPUT ON;  
  
BEGIN  
    afficher_employes;  
END;
```

Below the code editor is a tab labeled "Sortie de script x". The output area shows the result of the query:

```
bloc anonyme terminé  
Alexander  
Bruce  
David  
Valli  
Diana
```

The status bar at the bottom of the output window indicates "Tâche terminée en 0,016 secondes".

Les procédures stockées

# Les paramètres de procédure

- IN
- OUT
- IN OUT



Les procédures stockées

# Les paramètres IN

- Type par défaut
- Paramètre d'entrée



Les procédures stockées

# Exemple de paramètre d'entrée

```
SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE afficher_employes(id_department IN NUMBER)
AS
    CURSOR cursor_employees_it IS SELECT first_name,last_name FROM EMPLOYEES WHERE DEPARTMENT_ID = id_department;
    employee employees%ROWTYPE;
BEGIN
    FOR employee IN cursor_employees_it LOOP
        DBMS_OUTPUT.PUT_LINE(employee.first_name);
    END LOOP;
END;
```



Les procédures stockées

# Les paramètres OUT

- Paramètre de sortie





Les procédures stockées

# Exemple de paramètre de sortie

```
SET SERVEROUTPUT ON;

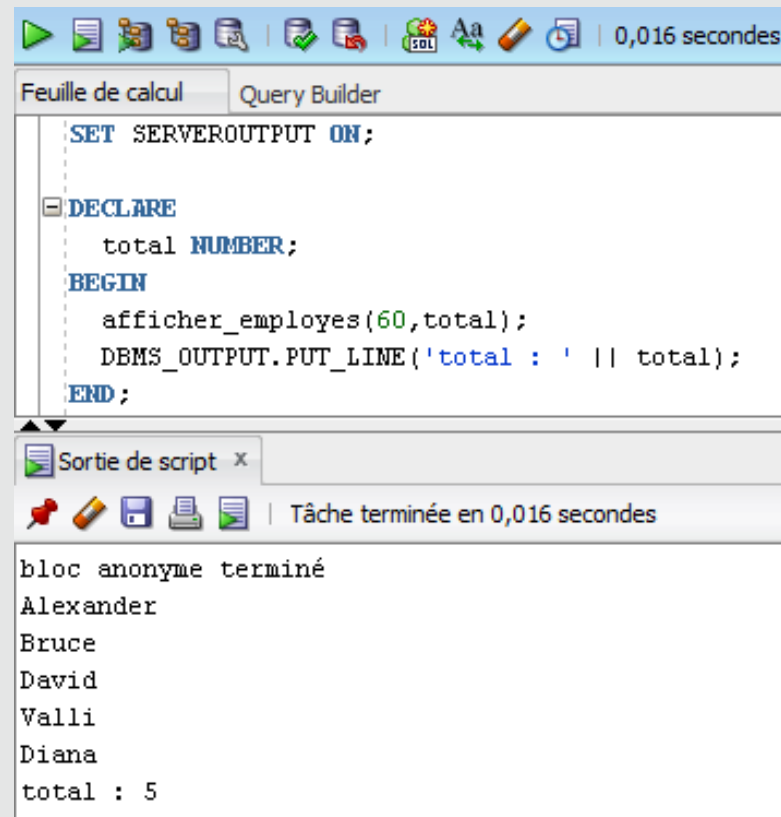
CREATE OR REPLACE PROCEDURE afficher_employes(id_department IN NUMBER, total_employes OUT NUMBER)
AS
    CURSOR cursor_employees_it IS SELECT first_name, last_name FROM EMPLOYEES WHERE DEPARTMENT_ID = id_department;
    employee employees%ROWTYPE;
BEGIN

    SELECT COUNT(*) INTO total_employes FROM employees WHERE DEPARTMENT_ID = id_department;

    FOR employee IN cursor_employees_it LOOP
        DBMS_OUTPUT.PUT_LINE(employee.first_name);
    END LOOP;
END;
```

Les procédures stockées

# Le passage de paramètres à une procédure



The screenshot shows a SQL Query Builder window with a toolbar at the top. The main area contains the following SQL code:

```
SET SERVEROUTPUT ON;  
  
DECLARE  
    total NUMBER;  
BEGIN  
    afficher_employes(60,total);  
    DBMS_OUTPUT.PUT_LINE('total : ' || total);  
END;
```

Below the code editor, there is a 'Sortie de script' (Script Output) window. It displays the results of the procedure execution:

```
bloc anonyme terminé  
Alexander  
Bruce  
David  
Valli  
Diana  
total : 5
```

The output window also shows a status bar at the bottom indicating 'Tâche terminée en 0,016 secondes' (Task completed in 0.016 seconds).

Les procédures stockées

# Les paramètres IN OUT

- Paramètres d'entrées et de sorties



Les procédures stockées

# Exemple de paramètre entrée / sortie

```
SET SERVEROUTPUT ON;

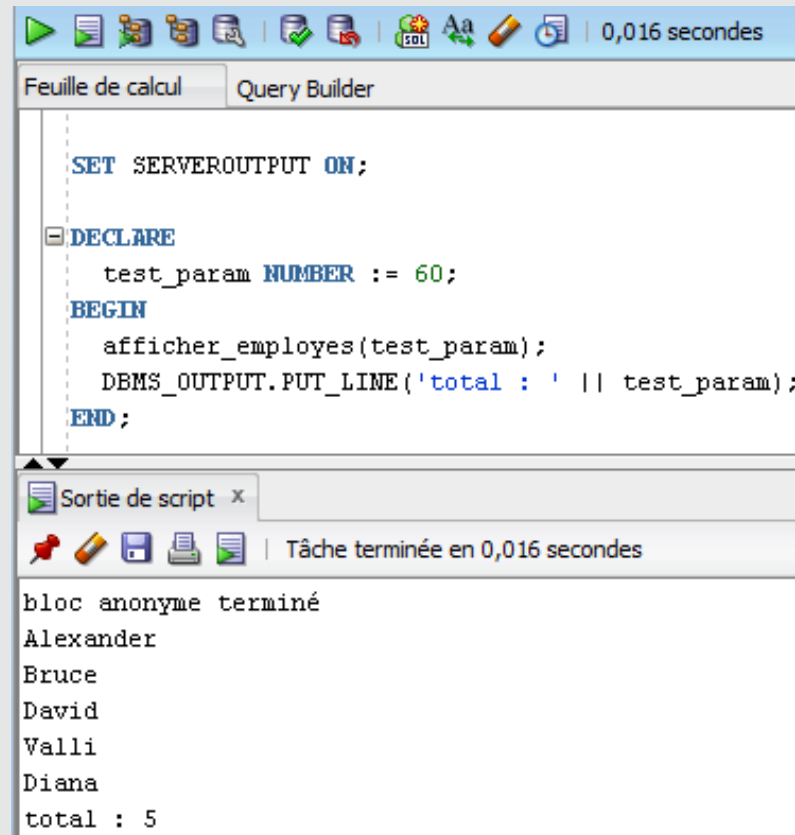
CREATE OR REPLACE PROCEDURE afficher_employes(info IN OUT NUMBER)
AS
    CURSOR cursor_employees_it IS SELECT first_name,last_name FROM EMPLOYEES WHERE DEPARTMENT_ID = info;
    employee employees%ROWTYPE;
BEGIN

    FOR employee IN cursor_employees_it LOOP
        DBMS_OUTPUT.PUT_LINE(employee.first_name);
    END LOOP;

    SELECT COUNT(*) INTO info FROM employees WHERE DEPARTMENT_ID = info;
END;
```

Les procédures stockées

# L'utilisation d'un paramètre entrée / sortie



```
SET SERVEROUTPUT ON;

DECLARE
    test_param NUMBER := 60;
BEGIN
    afficher_employes(test_param);
    DBMS_OUTPUT.PUT_LINE('total : ' || test_param);
END;
```

Sortie de script x

Tâche terminée en 0,016 secondes

bloc anonyme terminé  
Alexander  
Bruce  
David  
Valli  
Diana  
total : 5

Les procédures stockées

# Démonstration



# Conclusion

- Vous savez créer des procédures stockées
- Vous savez utiliser des procédures stockées
- Vous connaissez les différents types de paramètres applicables aux procédures stockées



# PL / SQL

## Module 8 – Les fonctions





# Objectifs

- Savoir créer une fonction utilisateur
- Savoir utiliser une fonction utilisateur
- Savoir exploiter la valeur retournée par une fonction



Les fonctions

# Définition

- Bloc de code nommé et stocké sur le serveur
- Retourne toujours une et seule une valeur



Les fonctions

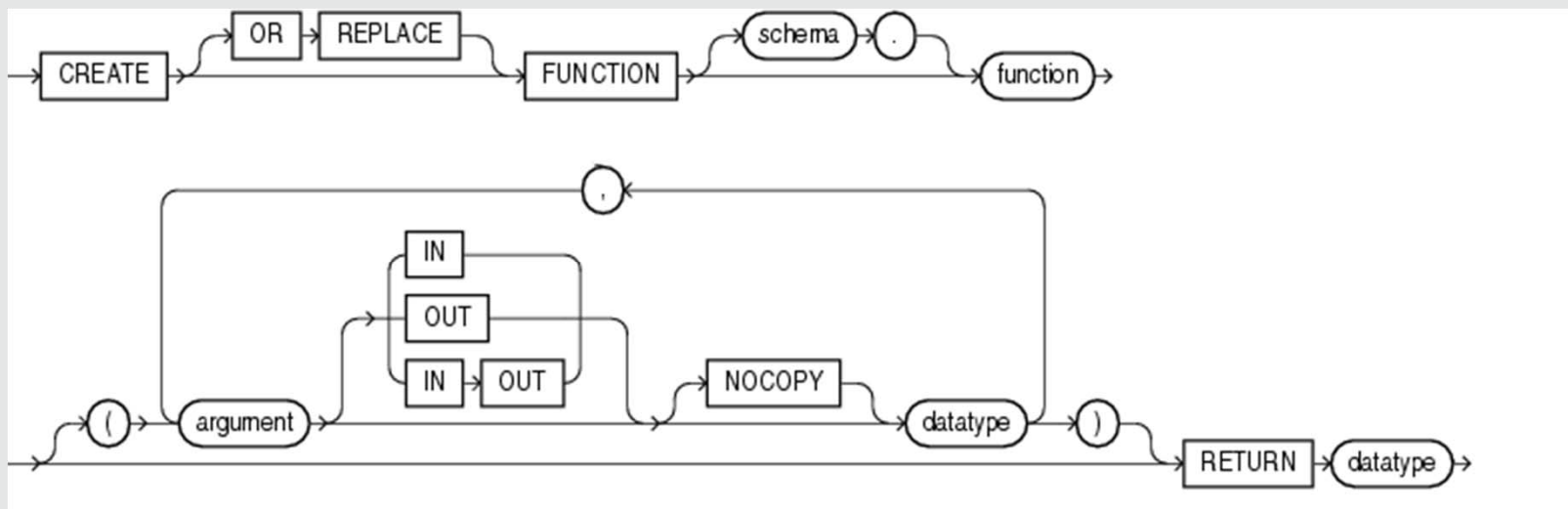
# Les spécificités

- Les paramètres d'entrées sont exclusivement de type IN
- Les paramètres d'entrées doivent être de type SQL et non PL/SQL
- Le paramètre de retour doit être de type SQL et non PL/SQL
- Les fonctions ne doivent pas faire de DML (INSERT, UPDATE, DELETE)



Les fonctions

# La syntaxe de création



Les fonctions

# Exemple de création d'une fonction

```
CREATE OR REPLACE FUNCTION multiplier_par_deux(nombre_a_multiplier IN NUMBER)
RETURN NUMBER
IS
resultat NUMBER;
BEGIN
    resultat := nombre_a_multiplier * 2;
    RETURN resultat;
END;
```

Les fonctions

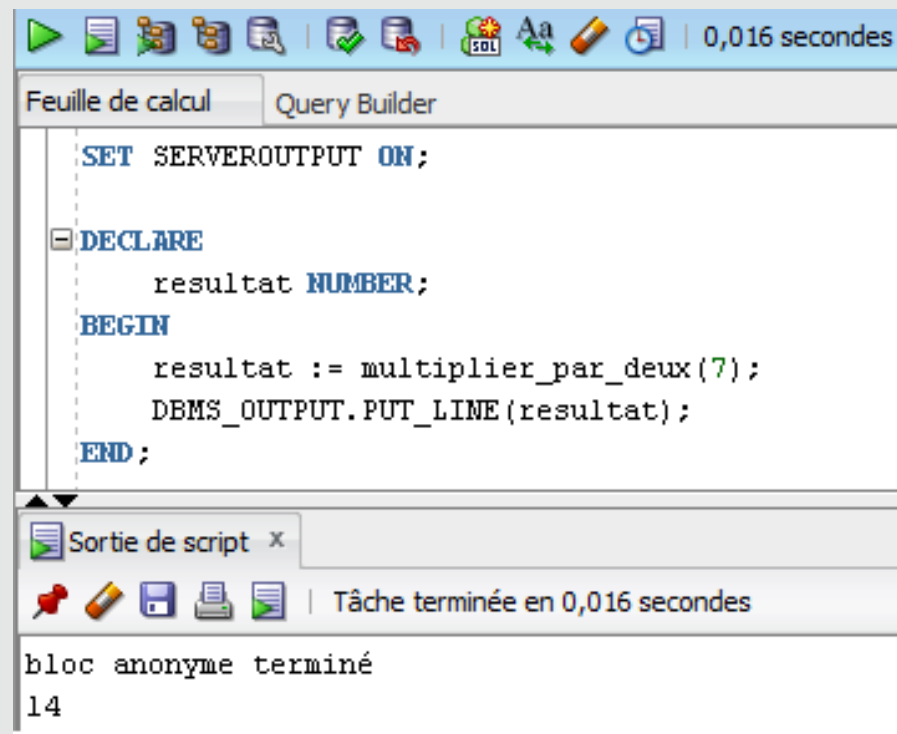
# L'appel à une fonction

- Depuis un bloc PL/SQL
- Depuis une requête SQL



Les fonctions

# L'appel à partir d'un bloc



The screenshot shows the Oracle SQL Developer interface. The top toolbar includes icons for running, saving, and other database operations, along with a timer showing 0,016 secondes. The main window is titled 'Feuille de calcul' and 'Query Builder'. It contains a PL/SQL block with the following code:

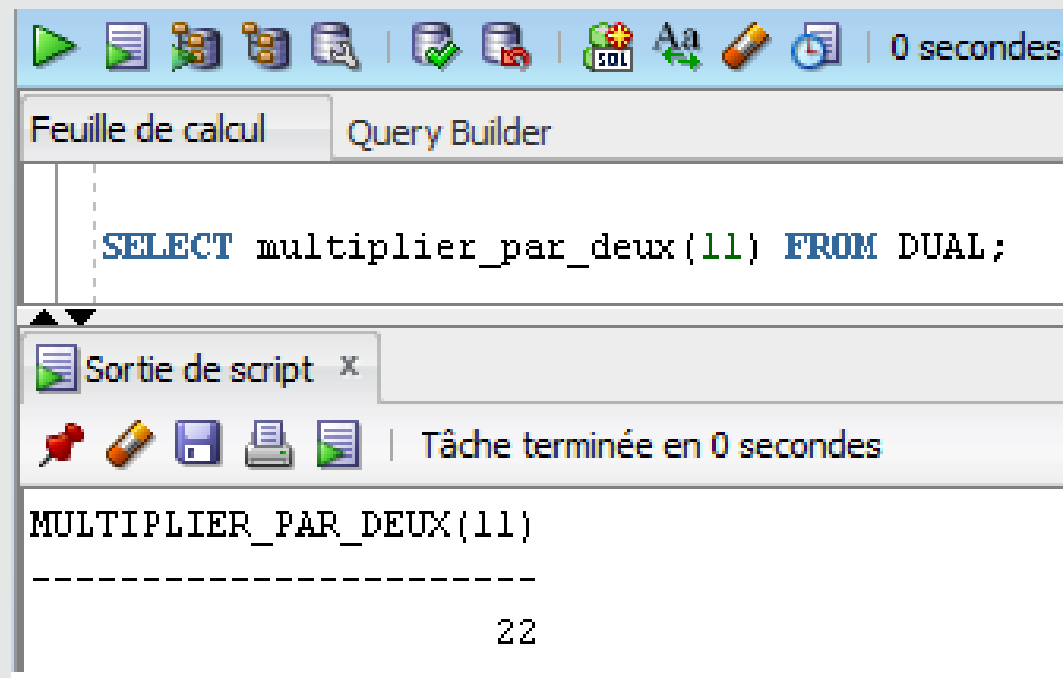
```
SET SERVEROUTPUT ON;  
  
DECLARE  
    resultat NUMBER;  
BEGIN  
    resultat := multiplier_par_deux(7);  
    DBMS_OUTPUT.PUT_LINE(resultat);  
END;
```

Below the code editor, there is a section titled 'Sortie de script' with a timer showing 'Tâche terminée en 0,016 secondes'. The output text is:

```
bloc anonyme terminé  
14
```

Les fonctions

# L'appel à partir d'une requête





Les fonctions

# Démonstration



# Conclusion

- Vous savez créer des fonctions
- Vous savez utiliser des fonctions
- Vous savez exploiter la valeur retournée par la fonction



# PL / SQL

## Module 9 – Les déclencheurs de base de données



# Objectifs

- Savoir expliquer ce qu'est un trigger
- Savoir créer un trigger



Les déclencheurs de base de données

# Définition

- Trigger = Déclencheur
- Traitement s'exécutant automatiquement lorsqu'un évènement (insertion, suppression, mise à jour) se produit sur une table ou une vue



Les déclencheurs de base de données

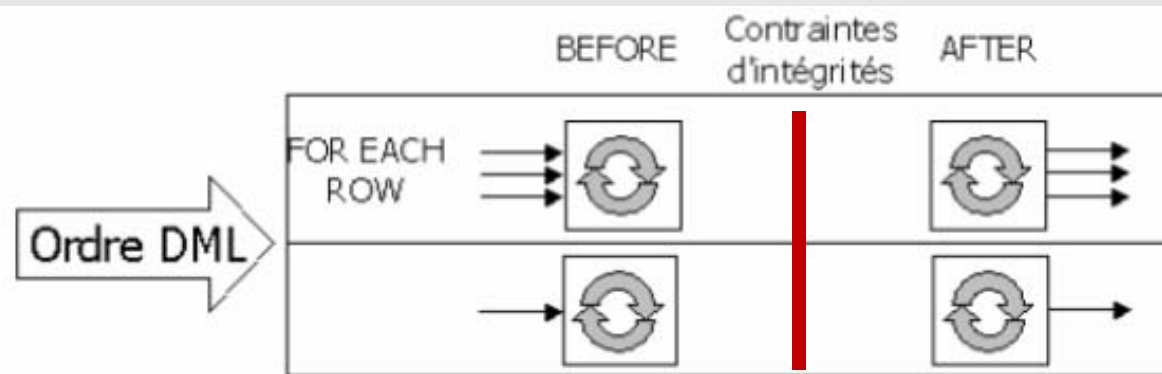
## Les spécificités

- Blocs associés à un nom
- Peuvent appeler des sous-programmes
- Non paramétrables
- COMMIT et ROLLBACK interdits



Les déclencheurs de base de données

# Le principe de fonctionnement



*Exécution avant ou après vérification des contraintes d'intégrité pour chaque ligne ou chaque ordre.*

Les déclencheurs de base de données

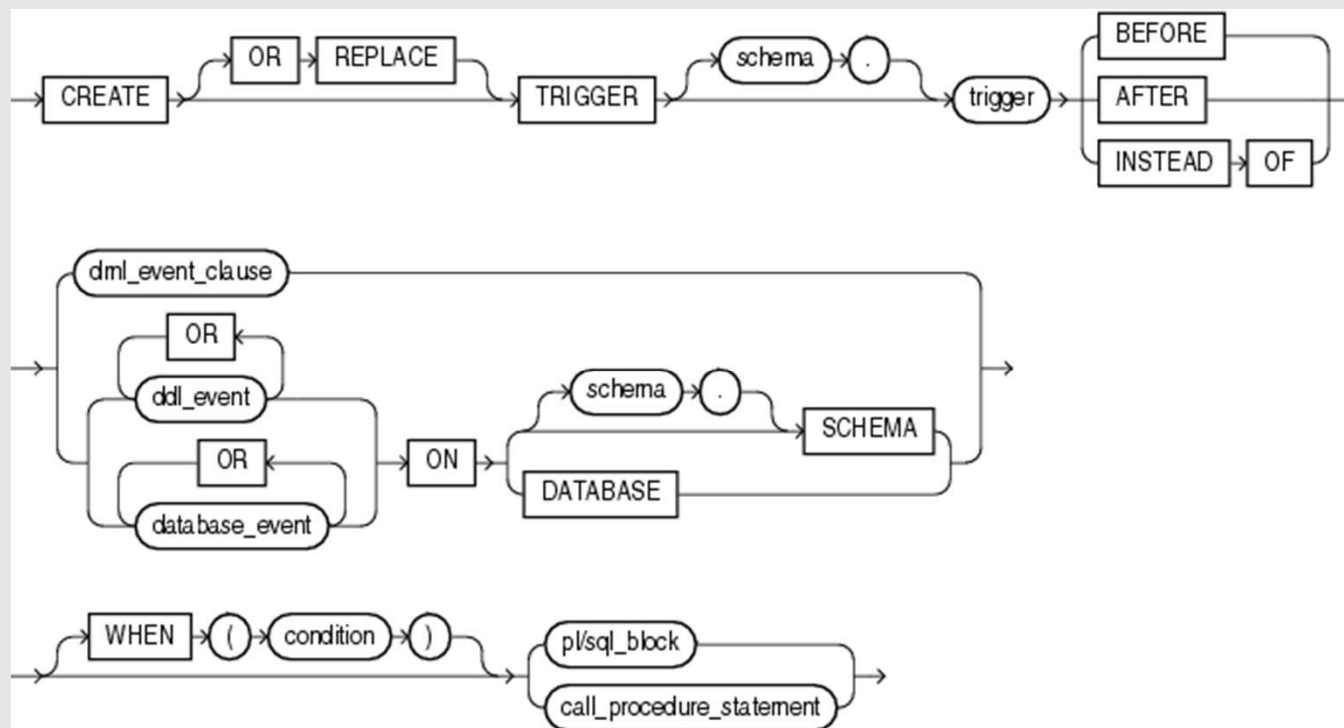
# Les différents types de trigger de table

| TRIGGER | BEFORE | STATEMENT | INSERT | SELECT |           |
|---------|--------|-----------|--------|--------|-----------|
|         |        |           | UPDATE | SELECT |           |
|         |        |           | DELETE | SELECT |           |
|         |        | ROW       | INSERT | SELECT | New       |
|         |        |           | UPDATE |        | New / Old |
|         |        |           | DELETE |        | New       |
|         | AFTER  | STATEMENT | INSERT | SELECT |           |
|         |        |           | UPDATE | SELECT |           |
|         |        |           | DELETE | SELECT |           |
|         |        | ROW       | INSERT |        | New       |
|         |        |           | UPDATE |        | New / Old |
|         |        |           | DELETE |        | New       |



Les déclencheurs de base de données

# La syntaxe de création 1/2



Les déclencheurs de base de données

## La syntaxe de création 2/2

