



Faculté de Sciences Économiques et de Gestion

Oracle

IUP ISEA

Année 1999-2000

Jérôme Darmont

jerome.darmont@univ-lyon2.fr

Plan du cours

- I. Introduction
- II. SQL*Plus
- III. PL/SQL
- IV. Les triggers

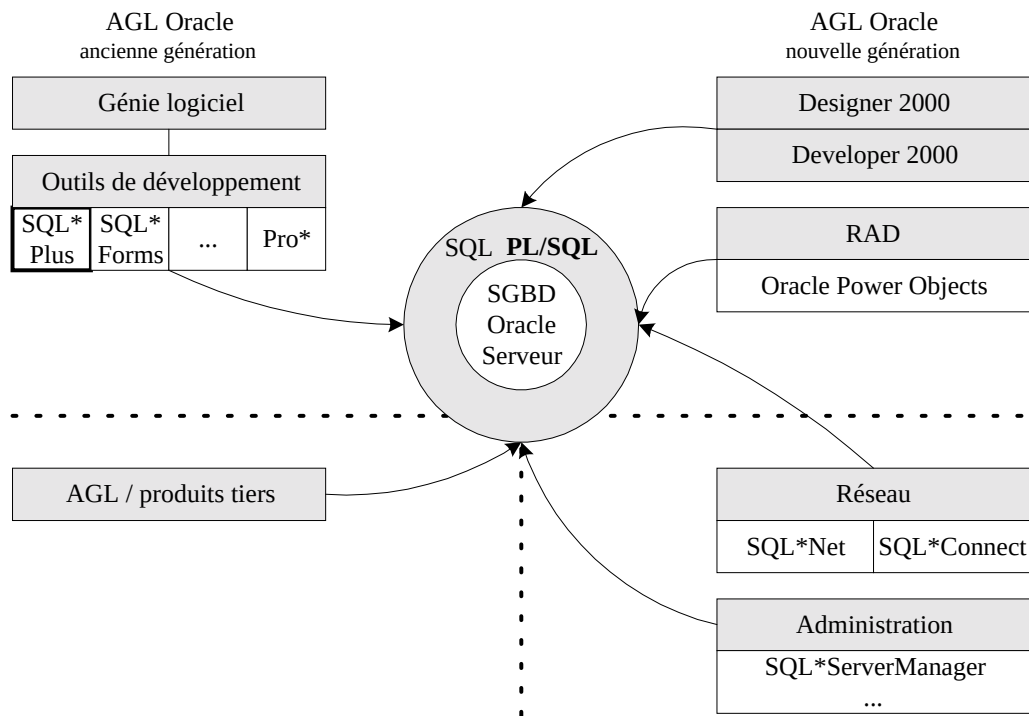
Plan du cours

- ☞ **I. Introduction**
- II. SQL*Plus
- III. PL/SQL
- IV. Les triggers

I.1. Qu'est-ce qu'Oracle ?

- **Oracle** : Système de Gestion de Bases de Données (SGBD) relationnel (SGBDR) édité par *Oracle Corporation* (<http://www.oracle.com/>)
- Basé sur **SQL** (*Structured Query Language*), langage de définition, de manipulation et de contrôle de bases de données relationnelles (standard ANSI depuis 1986)
- Première version d'Oracle : 1981
Version actuelle : Oracle 8i

I.2. Architecture d'Oracle



I.3. Base de données exemple

CLIENT (NumCli, Nom, Prénom, DateNaiss, Rue, CP, Ville)

PRODUIT (NumProd, Dési, PrixUni, NumFour)

FOURNISSEUR (NumFour, RaisonSoc)

COMMANDE (NumCli, NumProd, Date, Quantité)

Clés primaires

Clés étrangères

Plan du cours

I. Introduction

 **II. SQL*Plus**

III. PL/SQL

IV. Les triggers

II.1. Généralités

- **SQL :**
 - LDD (*Langage de Définition de Données*) : création, modification et suppression des définitions des tables
 - LMD (*Langage de Manipulation de Données*) : ajout, suppression, modification et interrogation des données
 - LCD (*Langage de Contrôle de Données*) : gestion des protections d'accès
- **Fin d'instruction :** ;
- **Commentaires :** /* ... */ ou --commentaire

II.2. Définition des données

Définitions des tables

```
CREATE TABLE (Attribut1 TYPE, Attribut2 TYPE, ...,  
               contrainte_intégrité1,  
               contrainte_intégrité2, ...);
```

Type des données :

- NUMBER(n) : Entier à n chiffres
- NUMBER(n, m) : Réel à n chiffres au total (virgule comprise), m après la virgule
- VARCHAR(n) : Chaîne de n caractères (entre ' ')
- DATE : Date au format 'JJ-MM-AAAA'

II.2. Définition des données

Définitions des contraintes d'intégrité

- *Clé primaire :*

```
CONSTRAINT nom_contrainte PRIMARY KEY  
(attribut_clé [, attribut_clé2, ...])
```

- *Clé étrangère :*

```
CONSTRAINT nom_contrainte FOREIGN KEY  
(attribut_clé_ét) REFERENCES table(attribut)
```

- *Contrainte de domaine :*

```
CONSTRAINT nom_contrainte CHECK (condition)
```

II.2. Définition des données

ex. CREATE TABLE Client (NumCli NUMBER(3),
 Nom CHAR(30),
 DateNaiss DATE,
 Salaire NUMBER(8,2),
 NumEmp NUMBER(3),

CONSTRAINT cle_pri PRIMARY KEY (NumCli),
CONSTRAINT cle_etr FOREIGN KEY (NumEmp)
 REFERENCES EMPLOYEUR(NumEmp),
CONSTRAINT date_ok CHECK (DateNaiss < SYSDATE));

II.2. Définition des données

- **Création d'index** (accélération des accès)

CREATE [UNIQUE] INDEX nom_index ON
nom_table (attribut [ASC|DESC], ...);

UNIQUE $\Rightarrow$ pas de double

ASC/DESC $\Rightarrow$ ordre croissant ou décroissant

ex. CREATE UNIQUE INDEX Icli ON Client
(NumCli);

- **Destructions :** DROP TABLE nom_table;
 DROP INDEX nom_index;

II.2. Définition des données

- **Ajout d'attributs**

ALTER TABLE nom_table ADD (attribut TYPE, ...);

ex. ALTER TABLE Client ADD (tel NUMBER(8));

- **Modifications d'attributs**

ALTER TABLE nom_table MODIFY (attribut TYPE, ...);

ex. ALTER TABLE Client MODIFY (tel NUMBER(10));

- **Suppression de contraintes**

ALTER TABLE nom_table DROP CONSTRAINT
nom_contrainte;

II.3. Mise à jour des données

- **Ajout d'un tuple**

INSERT INTO nom_table

VALUES (val_att1, val_att2, ...);

ex. INSERT INTO Produit

VALUES (400, 'Nouveau produit', 78.90);

- **Mise à jour d'un attribut**

UPDATE nom_table SET attribut=valeur
[WHERE condition];

ex. UPDATE Client SET Nom='Dudule'
WHERE NumCli = 3;

II.3. Mise à jour des données

- **Suppression de tuples**

DELETE FROM nom_table [WHERE condition];

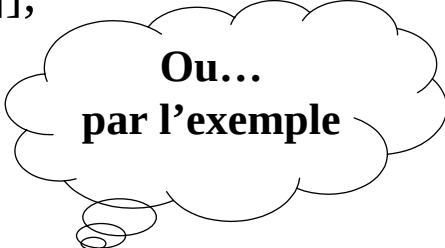
ex. DELETE FROM Produit;

DELETE FROM Client
WHERE Ville = 'Lyon';

II.4. Interrogation des données

SELECT [ALL|DISTINCT] attribut(s) FROM table(s)
[WHERE condition]
[GROUP BY attribut(s) [HAVING condition]]
[ORDER BY attribut(s) [ASC|DESC]];

- **Tous les tuples d'une table**
ex. SELECT * FROM Client;



Ou...
par l'exemple

- **Tri du résultat**
ex. *Par ordre alphabétique inverse de nom*
SELECT * FROM Client
ORDER BY Nom DESC;

II.4. Interrogation des données

- **Calculs** *ex. Calcul de prix TTC*
`SELECT PrixUni+PrixUni*0.206 FROM Produit;`
- **Projection** *ex. Noms et Prénoms des clients, uniquement*
`SELECT Nom, Prenom FROM Client;`
- **Restriction** *ex. Clients qui habitent à Lyon*
`SELECT * FROM Client
WHERE Ville = 'Lyon';`

II.4. Interrogation des données

- ex. Commandes en quantité au moins égale à 3*
`SELECT * FROM Commande
WHERE Quantite >= 3;`
- ex. Produits dont le prix est compris entre 50 et 100 F*
`SELECT * FROM Produit
WHERE PrixUni BETWEEN 50 AND 100;`
- ex. Commandes en quantité indéterminée*
`SELECT * FROM Commande
WHERE Quantite IS NULL;`

II.4. Interrogation des données

ex. *Clients habitant une ville dont le nom se termine par sur-Saône*

```
SELECT * FROM Client  
WHERE Ville LIKE '%sur-Saône';
```

'sur-Saône%' ⇒ commence par *sur-Saône*
'%sur%' ⇒ contient le mot *sur*

II.4. Interrogation des données

ex. *Prénoms des clients dont le nom est Dupont, Durand ou Martin*

```
SELECT Prenom FROM Client  
WHERE Nom IN ('Dupont', 'Durand', 'Martin');
```

NB : Possibilité d'utiliser la négation pour tous ces prédicats : NOT BETWEEN, NOT NULL, NOT LIKE, NOT IN.

II.4. Interrogation des données

- **Fonctions d'agrégat**

Elles opèrent sur un ensemble de valeurs.

- AVG(), VARIANCE(), STDDEV() : moyenne, variance et écart-type des valeurs
- SUM() : somme des valeurs
- MIN(), MAX() : valeur minimum, valeur maximum
- COUNT() : nombre de valeurs

ex. *Moyenne des prix des produits*

```
SELECT AVG(PrixUni) FROM Produit;
```

II.4. Interrogation des données

Opérateur DISTINCT

ex. *Nombre total de commandes*

```
SELECT COUNT(*) FROM Commande;  
SELECT COUNT(NumCli) FROM Commande;
```

ex. *Nombre de clients ayant passé commande*

```
SELECT COUNT( DISTINCT NumCli)  
FROM Commande;
```

II.4. Interrogation des données

Table COMMANDE (simplifiée)

<i>NumCli</i>	<i>Date</i>	<i>Quantite</i>
1	22/09/99	1
3	22/09/99	5
3	22/09/99	2

COUNT(NumCli) $\Rightarrow$ Résultat = 3

COUNT(DISTINCT NumCli) $\Rightarrow$ Résultat = 2

II.4. Interrogation des données

- **Jointure**

ex. *Liste des commandes avec le nom des clients*

```
SELECT Nom, Date, Quantite
FROM Client, Commande
WHERE Client.NumCli =
      Commande.NumCli;
```

II.4. Interrogation des données

ex. *Idem avec le numéro de client en plus*

```
SELECT C1.NumCli, Nom, Date, Quantite  
FROM Client C1, Commande C2  
WHERE C1.NumCli = C2.NumCli  
ORDER BY Nom;
```

NB : Utilisation d'*alias* (C1 et C2) pour alléger l'écriture + tri par nom.

II.4. Interrogation des données

Jointure exprimée avec le prédicat IN

ex. *Nom des clients qui ont commandé le 23/09*

```
SELECT Nom FROM Client  
WHERE NumCli IN (  
    SELECT NumCli FROM Commande  
    WHERE Date = '23-09-1999' );
```

NB : Il est possible d'imbriquer des requêtes.

II.4. Interrogation des données

- **Prédicats EXISTS / NOT EXISTS**

ex. *Clients qui ont passé au moins une commande
[n'ont passé aucune commande]*

```
SELECT * FROM Client C1
WHERE [NOT] EXISTS (
    SELECT * FROM Commande C2
    WHERE C1.NumCli = C2.NumCli    );
```

II.4. Interrogation des données

- **Prédicats ALL / ANY**

ex. *Numéros des clients qui ont commandé au
moins un produit en quantité supérieure à
chacune [à au moins une] des quantités
commandées par le client n° 1.*

```
SELECT DISTINCT NumCli FROM Commande
WHERE Quantite > ALL [ANY] (
    SELECT Quantite FROM Commande
    WHERE NumCli = 1    );
```

II.4. Interrogation des données

- **Groupement**

ex. *Quantité totale commandée par chaque client*

```
SELECT NumCli, SUM(Quantite)
FROM Commande
GROUP BY NumCli;
```

ex. *Nombre de produits différents commandés...*

```
SELECT NumCli, COUNT(DISTINCT NumProd)
FROM Commande
GROUP BY NumCli;
```

II.4. Interrogation des données

ex. *Quantité moyenne commandée pour les produits faisant l'objet de plus de 3 commandes*

```
SELECT NumProd, AVG(Quantite)
FROM Commande
GROUP BY NumProd
HAVING COUNT(*)>3;
```

Attention : La clause HAVING ne s'utilise qu'avec GROUP BY.

II.4. Interrogation des données

- **Opérations ensemblistes**

INTERSECT, MINUS, UNION

ex. *Numéro des produits qui soit ont un prix inférieur à 100 F, soit ont été commandés par le client n° 2*

```
SELECT NumProd FROM Produit WHERE PrixUni<100
UNION
SELECT NumProd FROM Commande WHERE NumCLi=2;
```

II.4. Interrogation des données

- **Fonctions SQL*Plus**

- | | |
|---|---|
| – ABS(n) : Valeur absolue de n | – LOWER(ch) : c en minuscules |
| – CEIL(n) : Plus petit entier $\geq n$ | – UPPER(ch) : c en majuscules |
| – FLOOR(n) : Plus grand entier $\leq n$ | – LTRIM(ch, n) : Troncature à gauche |
| – MOD(m, n) : Reste de m/n | – RTRIM(ch, n) : Troncature à droite |
| – POWER(m, n) : m^n | – REPLACE(ch, car) :
Remplacement de caractère |
| – SIGN(n) : Signe de n | – SUBSTR(ch, pos, lg) : Extraction
de chaîne |
| – SQRT(n) : Racine carrée de n | – SOUNDEx(ch) : Cp. Phonétique |
| – ROUND(n, m) : Arrondi à 10^{-m} | – LPAD(ch, lg, car) : Compléter à
gauche |
| – TRUNC(n, m) : Troncature à 10^{-m} | – RPAD(ch, lg, car) : Compléter à
droite |
| – CHR(n) : Caractère ASCII n° n | |
| – INITCAP(ch) : 1 ^{ère} lettre en maj. | |

II.4. Interrogation des données

- | | |
|--|--|
| <ul style="list-style-type: none">– ASCII(ch) : Valeur ASCII de ch– INSTR(ch, ssch) : Recherche de ssch dans ch– LENGTH(ch) : Longueur de ch– ADD_MONTHS(dte, n) : Ajout de n mois à dte– LAST_DAY(dte) : Dernier jour du mois– MONTHS_BETWEEN(dt1, dt2) : Nombre de mois entre dt1 et dt2– NEXT_DAY(dte) : Date du lendemain– SYSDATE : Date/heure système | <ul style="list-style-type: none">– TO_NUMBER(ch) : Conversion de ch en nombre– TO_CHAR(x) : Conversion de x en chaîne– TO_DATE(ch) : Conversion de ch en date– NVL(x, val) : Remplace par val si x a la valeur NULL– GREATEST(n1, n2...) : + grand– LEAST (n1, n2...) : + petit– UID : Identifiant numérique de l'utilisateur– USER : Nom de l'utilisateur |
|--|--|

II.5. Gestion des transactions

- **Transaction** : ensemble de modifications de la base
- **Début de transaction** : début de la session de travail ou fin de la transaction précédente
- **Validation** (et fin) d'une transaction : COMMIT;
- **Annulation** (et fin) d'une transaction : ROLLBACK;
- Fin de session de travail $\Rightarrow$ validation automatique

II.6. Les vues

- **Vue** : table *virtuelle* calculée à partir d'autres tables grâce à une requête
- **Définition d'une vue**
CREATE VIEW nom_vue AS requête;

ex. CREATE VIEW Noms AS
SELECT Nom, Prenom FROM Client;

II.6. Les vues

Intérêt des vues

- Simplification de l'accès aux données en masquant les opérations de jointure

ex. CREATE VIEW Prod_com AS
SELECT P.NumProd, Dési, PrixUni, Date, Quantite
FROM Produit P, Commande C
WHERE P.NumProd=C.NumProd;

SELECT NumProd, Dési FROM Prod_com
WHERE Quantite>10;

II.6. Les vues

Intérêt des vues

- Sauvegarde indirecte de requêtes complexes
- Présentation de mêmes données sous différentes formes adaptées aux différents usagers particuliers
- Support de l'indépendance logique
ex. Si la table Produit est remaniée, la vue Prod_com doit être refaite, mais les requêtes qui utilisent cette vue n'ont pas à être remaniées.

II.6. Les vues

Intérêt des vues

- Renforcement de la sécurité des données par masquage des lignes et des colonnes sensibles aux usagers non habilités

Problèmes de mise à jour, restrictions

La mise à jour de données via une vue pose des problèmes et la plupart des systèmes impose d'importantes restrictions.

II.6. Les vues

Problèmes de mise à jour, restrictions

- Le mot clé DISTINCT doit être absent.
- La clause FROM doit faire référence à une seule table.
- La clause SELECT doit faire référence directement aux attributs de la table concernée (pas d'attribut dérivé).
- Les clauses GROUP BY et HAVING sont interdites.

II.7. Sécurité et autorisation

- **Transmission de privilèges**

GRANT privilège ON table|vue
TO user|PUBLIC [WITH GRANT OPTION];

Privilèges :

SELECT : lecture

INSERT : insertion

UPDATE : mise à jour

DELETE : suppression

ALL : tous les privilèges

ALTER : destruction

INDEX : construction d'index

- **Suppression de privilèges**

REVOKE privilège ON table|vue FROM user|PUBLIC;

II.8. Catalogue du système

Contient sous forme relationnelle la définition de tous les objets créés par le système et les usagers.

Ces tables sont accessibles avec SQL (en mode consultation uniquement).

Quelques tables utiles gérées par Oracle

- USER_CATALOG (TABLE_NAME, TABLE_TYPE)
- USER_TAB_COLUMNS (TABLE_NAME, COLUMN_NAME, ...)
- USER_IND_COLUMNS (INDEX_NAME, TABLE_NAME, COLUMN_NAME, ...)
- ALL_TABLES (TABLE_NAME, OWNER, ...)

II.8. Catalogue du système

Exemples

- Tables qui contiennent un attribut *Intitule*

```
SELECT TABLE_NAME FROM USER_TAB_COLUMNS  
WHERE COLUMN_NAME='Intitule';
```
- Attributs de la table *Client*

```
SELECT COLUMN_NAME FROM USER_TAB_COLUMNS  
WHERE TABLE_NAME='Client';
```
- Tables de l'utilisateur *darmont*

```
SELECT TABLE_NAME FROM ALL_TABLES  
WHERE OWNER='darmont';
```

Plan du cours

I. Introduction

II. SQL*Plus

☞ **III. PL/SQL**

IV. Les triggers

III.1. Généralités

- **PL/SQL :** Langage procédural
Extension de SQL
 - Déclaration de variables et de constantes
 - Définition de sous-programmes
 - Gestion des erreurs à l'exécution (exceptions)
 - Manipulation de données avec SQL

III.2. Bloc PL/SQL

DECLARE

--Déclaration constantes/variables

BEGIN

--Commandes/instructions

EXCEPTION

--Traitement des erreurs à l'exé.

END;

III.3. Déclarations

- Partie déclarative d'un bloc PL/SQL ou d'un sous-programme
- **Types usuels** : INTEGER, REAL, STRING, DATE, BOOLEAN + types SQL
- **Variables**
ex. date_naissance DATE;
compteur INTEGER:=0; -- Valeur par défaut
compteur INTEGER DEFAULT 0; -- idem
id CHAR(5) NOT NULL:='AP001';

III.3. Déclarations

- **Constantes**
ex. euro CONSTANT REAL:=6.55957;
- **Type d'une autre variable**
ex. credit REAL;
debit credit%TYPE;
- **Type d'un tuple d'une table**
ex. un_client client%ROWTYPE;
- **Tableaux :** 1) Déclaration d'un type tableau
2) Déclaration d'une variable de ce type

III.3. Déclarations

- **Tableaux**

ex. TYPE Tab_entiers TABLE OF INTEGER
INDEX BY BINARY_INTEGER;
TYPE Tab_cli TABLE OF client.nom%TYPE
INDEX BY BINARY_INTEGER;

un_entier Tab_entiers;
un_client Tab_cli;

III.3. Déclarations

- **Enregistrements personnalisés**

1) Définition d'un type enregistrement

ex. TYPE Enr_four IS RECORD (
 numero fournisseur.numfour%TYPE,
 raison_sociale CHAR(20));

2) Définition d'une variable de ce type

ex. un_fournisseur Enr_four;

III.4. Instructions de base

- **Affectation**

ex. numero:=0;
 numero:=numero+1;
 SELECT numcli INTO numero
 FROM client WHERE numcli=numero+1;

- **Utilisation des tableaux**

ex. i:=1;
 un_entier(i):=i*2;
NB : i doit être de type BINARY_INTEGER.

III.4. Instructions de base

- **Utilisation des enregistrements personnalisés**

ex. un_fournisseur.numero:=4589;
un_fournisseur.raison_sociale:='COGIP';

```
SELECT numfour, raisonsoc  
INTO un_fournisseur  
FROM fournisseur  
WHERE numfour=4589;
```

III.5. Structures de contrôle

- **Instruction sélective**

IF-THEN, IF-THEN-ELSE ou IF-THEN-ELSIF

```
IF condition1 THEN  
  -- Instructions  
ELSEIF condition2 THEN  
  -- Instructions  
ELSE  
  -- Instructions  
END IF;
```

III.5. Structures de contrôle

- **Instructions itératives**

```
FOR compteur IN [REVERSE] min..max LOOP
  -- Instructions
END LOOP;
WHILE condition LOOP
  -- Instructions
END LOOP;
LOOP
  -- Instructions
END LOOP;
```

III.5. Structures de contrôle

- **Branchements**

```
-- Saut inconditionnel
GOTO étiquette;
```

```
-- Sortie de boucle
EXIT WHEN condition;
```

NB : À utiliser avec **énormément** de modération !

III.6. Curseurs

- **Curseur** : structure de données permettant de stocker le résultat d'une requête qui retourne plusieurs tuples
- **Déclaration** : `CURSOR nom_curs IS requête;`
ex. `CURSOR calcul IS`
 `SELECT numprod, prixuni*1.206 prixttc`
 `FROM produit;`

NB : Un tuple du curseur sera de type `calcul%ROWTYPE`.

III.6. Curseurs

- **Ouverture d'un curseur** : `OPEN nom_curs;`
- **Gestion automatique d'un curseur**
ex. `FOR tuple IN calcul LOOP`
 `var1:=tuple.numprod;`
 `var2:=tuple.prixttc;`
 `END LOOP;`
- **Gestion « manuelle »**
ex. `LOOP`
 `FETCH calcul INTO tuple;`
 `EXIT WHEN calcul%NOTFOUND;`
 `...`
 `END LOOP`

III.6. Curseurs

- **Attributs des curseurs**

- %NOTFOUND : FALSE si FETCH retourne un résultat
- %FOUND : opposé logique de %NOTFOUND
- %ROWCOUNT : Nombre de lignes lues
- %ISOPEN : TRUE si le curseur est ouvert

- **Fermeture d'un curseur : CLOSE nom_curs;**

III.7. Exceptions

- À chaque erreur à l'exécution, une *exception* est levée. Ces exceptions sont gérées par des routines séparées.
- **Fonctions PL/SQL pour la gestion d'erreurs**
 - SQLCODE : Code de la dernière exception levée
 - SQLERRM : Message d'erreur associé

III.7. Exceptions

- **Exceptions prédéfinies**

Nom	Code erreur	SQLCODE
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	-1403
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
STORAGE_ERROR	ORA-06500	-6500
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

III.7. Exceptions

- **Exceptions personnalisées**

➤ *Déclaration* : `nom_exc EXCEPTION;`
➤ *Lever l'exception* : `IF condition THEN`
`RAISE nom_exc;`
`END IF;`

- **Traitement des exceptions**

`WHEN nom_exc THEN -- Instruction`

ex. `WHEN probleme THEN`
`RAISE_APPLICATION_ERROR(-20501,'Erreur !');`

NB : $-20999 \leq \text{no d'erreur} \leq -20001$

III.8. Sous-programmes

- **Fonctions**

```
FUNCTION nomf (param1, param2...)
RETURN type_valeur_de_retour IS
    -- Déclarations locales
BEGIN
    -- Instructions
    RETURN valeur_de_retour;
EXCEPTION
    -- Traitement des exceptions
END;
```

III.8. Sous-programmes

- **Procédures**

```
PROCEDURE nomp (param1, param2...) IS
    -- Déclarations locales
BEGIN
    -- Instructions
EXCEPTION
    -- Traitement des exceptions
END;
```

- **Paramètres**

```
nom_param [IN|OUT|IN OUT] TYPE
ex. resultat OUT REAL
```

III.9. Exemple de programme PL/SQL

```
-- Calcul du prix TTC des produits  
-- et recopie dans la table PRODTTC
```

```
DECLARE
```

```
    nbp NUMBER(3);  
    aucun_produit EXCEPTION;  
  
    CURSOR calcul IS  
        SELECT numprod, prixuni*1.206 prixttc  
        FROM produit;  
  
    tuple calcul%ROWTYPE;
```

III.9. Exemple de programme PL/SQL

```
BEGIN
```

```
    -- Comptage des produits  
    SELECT COUNT(*) INTO nbp FROM produit;  
  
    -- Test « il existe des produits » ou pas ?  
    IF nbp = 0 THEN  
        RAISE aucun_produit;  
    END IF;
```

III.9. Exemple de programme PL/SQL

```
-- Recopie des valeurs dans la table prodttc
FOR tuple IN calcul LOOP
    INSERT INTO prodttc VALUES
        (tuple.numprod, tuple.prixttc);
END LOOP;

-- Validation de la transaction
COMMIT;

EXCEPTION

    WHEN aucun_produit THEN
        RAISE_APPLICATION_ERROR(-20501,
            'Erreur : table client vide');

END;
```

Plan du cours

I. Introduction

II. SQL*Plus

III. PL/SQL

 **IV. Les triggers**

IV.1. Définition

- **Trigger** (*déclencheur*) : routine déclenchée automatiquement par des événements liés à des actions sur la base
- Les triggers complètent les contraintes d'intégrité en permettant des contrôles et des traitements plus complexes.

IV.2. Types de triggers

	Insertion	Suppression	Mise à jour
Avant	1	2	3
Après	4	5	6

...d'une table de la base

IV.3. Création d'un trigger

- **En SQL*Plus :**

```
CREATE [OR REPLACE] TRIGGER nom_trig
  BEFORE|AFTER
  INSERT|DELETE|UPDATE
  ON nom_table
  FOR EACH ROW
  -- Bloc PL/SQL contenant le
  -- traitement à effectuer
```

- **Variables spécifiques**

:OLD.nom_attribut : valeur de l'attribut avant mise à jour
:NEW.nom_attribut : valeur de l'attribut après mise à jour

IV.4. Exemple de trigger

```
-- Test de clé primaire sur la table client
```

```
CREATE OR REPLACE TRIGGER trig_clep
  BEFORE INSERT ON client
  FOR EACH ROW
  DECLARE
    n INTEGER;
    cle_existe EXCEPTION;
    cle_nulle EXCEPTION;

  BEGIN
    -- Existence de la clé primaire
    SELECT COUNT(numcli) INTO n FROM client
      WHERE numcli=:NEW.numcli;
    IF n>0 THEN
      RAISE cle_existe;
    END IF
```

IV.4. Exemple de trigger

```
-- Valeur nulle
IF :NEW.numcli IS NULL THEN
    RAISE cle_nulle;
END IF;

EXCEPTION
    WHEN cle_existe THEN
        RAISE_APPLICATION_ERROR(-20501,
            'Clé primaire déjà utilisée !');
    WHEN cle_nulle THEN
        RAISE_APPLICATION_ERROR(-20502,
            'La clé primaire doit avoir une valeur!');

END;
```