

## MODULE : BASES DE DONNÉES AVANCÉES

### ÉLÉMENT 1 : BASES DE DONNÉES AVANCÉES



**SAIDI Rajaa**  
r.saidi@insea.ac.ma

### Objectifs du cours

1. Étudier comment écrire des déclencheurs en PL/SQL
2. Étudier comment dériver un modèle relationnel de données à partir d'un modèle orienté objets
3. Comprendre les concepts des bases de données objet-relationnelles et leurs intérêts
4. Comprendre pourquoi évaluer et optimiser une requête
5. Étudier comment obtenir « vite » des enregistrements satisfaisant un prédicat en utilisant des index

### Objectifs du module

- ❖ Étude approfondie des systèmes de gestion de bases de données, de leurs architectures et de leur évolution
  - **Premier élément** : consacré principalement à PL/SQL, au modèle objet-relationnel et aux techniques d'évaluation et d'optimisation des requêtes
  - **Deuxième élément** : consacré à l'administration du SGBD Oracle : architecture interne, maintenance, journalisation, restauration, sécurité et confidentialité des données

### Organisation du module

Module	VH	Pourcentage note finale	Détail note
Bases de données avancées	24h	50%	100% : Examen final
Administration des BD	24h	50%	?

## Plan

*Rappel cours BD1 : SQL*

Chapitre 1 – PL/SQL, Triggers et Vues

Chapitre 2 – de l'UML vers une BD relationnelle

Chapitre 3 – Bases de données objet-relationnelles

Chapitre 4 – Evaluation et optimisation des requêtes

Chapitre 5 – Indexation

## Rappel

## Cours BD1- SQL

5

## Exploitation des données dans une BD

- ❖ Différentes opérations pour manipuler les données dans une BD :
  - recherche d'information (ou interrogation)
  - insertion de données
  - mise à jour de données
  - suppression de données

- ❖ Exemple d'une BD relationnelle

EST-FACTURÉ				NUMFACT	REFPROD	QTE
PRODUIT	REFPROD	DESIGN	PRIXHT			
	1	Livre	19,20	6	1	7
	5	Cahier	2,30	6	5	3
	9	Crayon	4,75	2	5	10
	7	Stylo	2,20	2	1	5
				3	7	4
FACTURE	NUMFACT	DATACT	ANC-PRODUIT	REFPROD	DESIGN	PRIXHT
	6	13/10/2010		1	Livre	19,10
	2	16/10/2010		9	Crayon	4,75
	3	16/10/2010		4	Cartable	80
				5	Cahier	2,30
				2	Trousse	10,50

7

## Le langage SQL

- ❖ SQL comporte 5 mots-clés principaux
  - **SELECT** : recherche
  - **CREATE** : création
  - **INSERT** : ajout
  - **UPDATE** : mise à jour
  - **DELETE** : suppression

8

## La recherche de données : SELECT

PRODUIT	REFPROD	DESIGN	PRIXHT
	1	Livre	19,20
	5	Cahier	2,30
	9	Crayon	4,75
	7	Stylo	2,30

❖ Syntaxe d'une requête **SELECT** simple :

```
SELECT champ1, ..., champn
FROM table1, ..., tablem
WHERE condition
ORDER BY champ1, ..., champj
```

- champs présentés dans le résultat de la requête
- tables (et/ou requêtes) utilisées
- condition devant être vérifiée par un enreg. pour figurer dans le résultat
- ordre de présentation des enreg. du résultat de la requête

❖ Exemple : référence et désignation des produits coûtant moins de 5 dh triés par désignation

```
SELECT REFPROD, DESIGN
FROM PRODUIT
WHERE PRIXHT < 5
ORDER BY DESIGN
```

REFPROD	DESIGN
5	Cahier
9	Crayon
7	Stylo

sauts à la ligne facultatifs

9

## La clause SELECT – 1

PRODUIT	REFPROD	DESIGN	PRIXHT
	1	Livre	19,20
	5	Cahier	2,30
	9	Crayon	4,75
	7	Stylo	2,30

❖ **SELECT \*** : retourne tous les champs

• exemple :

```
SELECT *
FROM PRODUIT
WHERE PRIXHT < 5
```

REFPROD	DESIGN	PRIXHT
5	Cahier	2,30
9	Crayon	4,75
7	Stylo	2,30

❖ **SELECT DISTINCT** : supprime les valeurs identiques

• exemple :

```
SELECT PRIXHT
FROM PRODUIT
WHERE PRIXHT < 5
```

PRIXHT
2,30
4,75
2,30

```
SELECT DISTINCT PRIXHT
FROM PRODUIT
WHERE PRIXHT < 5
```

PRIXHT
2,30
4,75

⚠ **SELECT DISTINCT REFPROD, PRIXHT**  
FROM PRODUIT  
WHERE PRIXHT < 5

REFPROD	PRIXHT
5	2,30
9	4,75
7	2,30

10

## La clause WHERE – 1 / 3

PRODUIT	REFPROD	DESIGN	PRIXHT
	1	Livre	19,20
	5	Cahier	2,30
	9	Crayon	4,75
	7	Stylo	2,20

❖ Sélectionne les enregistrements vérifiant une condition, composée de :

- noms de champs (ex: **PRIXHT** ou **PRODUIT.PRIXHT**)
- constantes num. (ex: **2.3**), chaînes (ex: **'Dupont'**), dates (ex: **#2010/10/21#**)
- opérateurs (**=**, **<**, **<=**, **>**, **>=**, **+**, **-**, **\***, **/**, ...) et fonctions (**sin**, **log**, ...)
- opérateurs logiques : **OR**, **AND**, **NOT**

❖ Exemple : désignation des produits dont le prix est compris entre 3 et 5 dh

```
SELECT DESIGN
FROM PRODUIT
WHERE PRIXHT >= 3
AND PRIXHT <= 5
```

DESIGN
Crayon

noter la répétition du champ **PRIXHT**

⚠ Utiliser des parenthèses en cas d'utilisation conjointe de **AND** et **OR**

- exemple : désign. des produits dont le prix est entre 3 et 5 dh ou dont la réf. est supérieure à 2

```
SELECT DESIGN
FROM PRODUIT
WHERE (PRIXHT >= 3
AND PRIXHT <= 5)
OR REFPROD > 2
```

11

## La clause WHERE – 2 / 3

PRODUIT	REFPROD	DESIGN	PRIXHT
	1	Livre	19,20
	5	Cahier	2,30
	9	Crayon	4,75
	7	Stylo	2,20

❖ L'opérateur **BETWEEN** :

**champ BETWEEN x AND y**

est équivalent à :

**champ >= x AND champ <= y**

- Remarque : **x** et **y** peuvent être des expressions quelconques

❖ Exemple : désignation des produits dont le prix est compris entre 3 et 5 dh

```
SELECT DESIGN
FROM PRODUIT
WHERE PRIXHT BETWEEN 3 AND 5
```

DESIGN
Crayon

12

## La clause WHERE – 3 / 3

PRODUIT	REFPROD	DESIGN	PRIXHT
	1	Livre	19,20
	5	Cahier	2,30
	9	Crayon	4,75
	7	Stylo	2,20

- ❖ Opérateur **LIKE** : comparaison avec un motif

*champ* **LIKE** 'motif'

où *motif* contient :

- ? un caractère quelconque
- \* une suite quelconque de caractères (éventuellement vide)
- # un chiffre quelconque

- exemple : désignation produits dont le nom commence par 'c'

```
SELECT DESIGN
FROM PRODUIT
WHERE DESIGN LIKE 'c*'
```

DESIGN
Cahier
Crayon

- ❖ Opérateur **IS NULL** : champ non rempli

- exemple : réf. des produits dont la désignation n'est pas remplie

```
SELECT REFPROD
FROM PRODUIT
WHERE DESIGN IS NULL
```

REFPROD
7

13

## La clause ORDER BY

PRODUIT	REFPROD	DESIGN	PRIXHT
	1	Livre	19,20
	5	Cahier	2,30
	9	Crayon	4,75
	7	Stylo	2,20

- ❖ Trie les résultats sur un ou plusieurs champs :

```
SELECT ...
FROM ...
WHERE ...
ORDER BY champ1, ..., champn ASC ou DESC
```

optionnel

ASC : ordre croissant - par défaut  
DESC : ordre décroissant

- deux critères de tri: le plus à gauche est prioritaire ; le 2<sup>e</sup> utile en cas d'ex-aequo  
exemple : **ORDER BY NOM, PRENOM**

- ordre utilisé :

type champ	ordre
texte	lexicographique (ordre alphabétique étendu)
numérique	numérique
date / heure	chronologique

- ❖ Exemple : produits de moins de 5 dh triés par désignation croissante

```
SELECT REFPROD, DESIGN
FROM PRODUIT
WHERE PRIXHT < 5
ORDER BY DESIGN
```

REFPROD	DESIGN
5	Cahier
9	Crayon
7	Stylo

14

## La clause FROM – 1 / 5

- ❖ Exemple : désignation des produits facturés

PRODUIT(REFPROD, DESIGN, PRIXHT)  
EST-FACTURE(NUMFACT, REFPROD, QTE)

```
SELECT DESIGN
FROM EST-FACTURE, PRODUIT
WHERE EST-FACTURE.REFPROD = PRODUIT.REFPROD
```

jointure

champ REFPROD de la table EST-FACTURE      champ REFPROD de la table PRODUIT

PRODUIT	REFPROD	DESIGN	PRIXHT
	1	Livre	19,20
	5	Cahier	2,30
	9	Crayon	4,75
	7	Stylo	2,20

EST-FACTURE	NUMFACT	REFPROD	QTE
	6	1	7
	6	5	3
	2	5	10
	2	1	5
	3	7	4

- ❖ Lorsque plusieurs tables sont nécessaires

- il faut les mettre dans **FROM**
- il faut les "joindre" sur leur(s) champ(s) commun(s)

EST-FACTURE			PRODUIT		
NUMFACT	REFPROD	QTE	REFPROD	DESIGN	PRIXHT
6	1	7	1	Livre	19,20
6	5	3	5	Cahier	2,30
2	5	10	5	Cahier	2,30
2	1	5	1	Livre	19,20
3	7	4	7	Stylo	2,20

=

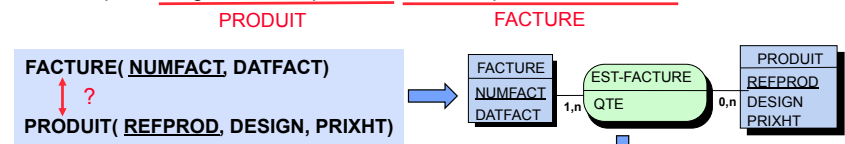
DESIGN
Livre
Cahier
Cahier
Livre
Stylo

15

## La clause FROM – 2 / 5

- ❖ Lorsque deux tables n'ont pas de champ commun, utiliser les tables intermédiaires pour effectuer la jointure : consulter le modèle entité-association

- exemple : désignation des produits facturés après le 14/10/2010



```
FACTURE(NUMFACT, DATACT)
EST-FACTURE(NUMFACT, REFPROD, QTE)
PRODUIT(REFPROD, DESIGN, PRIXHT)
```

```
SELECT DESIGN
FROM FACTURE, EST-FACTURE, PRODUIT
WHERE EST-FACTURE.NUMFACT = FACTURE.NUMFACT
AND EST-FACTURE.REFPROD = PRODUIT.REFPROD
AND DATACT > #2010/10/14#
```

jointures

16

## La clause FROM – 3 / 5

- ❖ Une requête peut être basée sur une (ou plusieurs) autres requête(s) :

```
SELECT ...
FROM req, ...
...
```

*req* peut être utilisée comme une table dont les champs sont ceux affichés dans la clause **SELECT** de *req*

- ❖ Intérêt

- écrire des requêtes qui peuvent être réutilisées dans d'autres requêtes
- décomposer une requête complexe à écrire en requêtes plus simples

17

## La clause FROM – 4 / 5

Désignation des produits ayant été facturés, dont le prix a augmenté par rapport à l'ancien catalogue (table ANC-PRODUIT)

PRODUIT	REFPROD	DESIGN	PRIXHT
	1	Livre	19,20
	5	Cahier	2,30
	9	Crayon	4,75
	7	Stylo	2,20

R1

```
SELECT REFPROD, PRIXHT
FROM EST-FACTURE, PRODUIT
WHERE EST-FACTURE.REFPROD = PRODUIT.REFPROD
```

EST-FACTURÉ	NUMFACT	REFPROD	QTE
	6	1	7
	6	5	3
	2	5	10
	2	1	5
	3	7	4

R1	REFPROD	PRIXHT
	1	19,20
	5	2,30
	5	2,30
	1	19,20
	7	2,20

ANC-PRODUIT	REFPROD	DESIGN	PRIXHT
	1	Livre	19,10
	9	Crayon	4,75
	4	Cartable	80
	5	Cahier	2,30
	2	Trousse	10,50

```
SELECT DESIGN
FROM R1, ANC-PRODUIT
WHERE R1.REFPROD = ANC-PRODUIT.REFPROD
AND R1.PRIXHT > ANC-PRODUIT.PRIXHT
```

DESIGN
Livre

18

## La clause FROM – 5 / 5

PRODUIT	REFPROD	DESIGN	PRIXHT
	1	Livre	19,20
	5	Cahier	2,30
	9	Crayon	4,75
	7	Stylo	2,20

- ❖ Il arrive que l'on utilise plusieurs fois la même table dans une requête

- exemple : liste des produits ayant un prix supérieur à celui du produit n° 9

```
SELECT P2.DESIGN
FROM PRODUIT AS P1, PRODUIT AS P2
WHERE P1.REFPROD = 9
AND P2.PRIXHT > P1.PRIXHT
```

DESIGN
Livre

P1

PRODUIT	REFPROD	DESIGN	PRIXHT
	1	Livre	19,20
	5	Cahier	2,30
	9	Crayon	4,75
	7	Stylo	2,20

P2

PRODUIT	REFPROD	DESIGN	PRIXHT
	1	Livre	19,20
	5	Cahier	2,30
	9	Crayon	4,75
	7	Stylo	2,20

19

## Calculs dans les requêtes

- ❖ Il est possible d'effectuer des calculs dans une requête
- ❖ Il existe deux types de calcul

- sur les lignes

	REFPROD	PRIXHT
max(PRIXHT)	1	19,20
	5	2,30
	9	5,75
	7	2,20

- sur les colonnes

	REFPROD	PRIXHT
x 1,196	1	19,20
	5	2,30
	9	5,75
	7	2,20

- ❖ Dans les deux cas, les opérations s'effectuent dans la clause SELECT

```
SELECT opération (champ)
FROM ...
...
```

20

## Calculs sur les lignes

PRODUIT	REFPROD	DESIGN	PRIXHT
	1	Livre	19,20
	5	Cahier	2,30
	9	Crayon	4,75
	7	Style	2,20

### ❖ Principales opérations

- **COUNT** : nombre d'enregistrements
- **SUM** : somme des valeurs du champ (numérique) sur un ensemble d'enreg.
- **AVG** : moyenne des valeurs du champ (numérique) sur un ensemble d'enreg.
- **MIN**, **MAX** : valeur min. et max. dans l'ensemble d'enregistrements
- opérateurs (+, -, \*, /, ...) et fonctions (**sin**, **exp**, ...)

### ❖ Exemple : prix le plus élevé parmi les produits

```
SELECT MAX(PRIXHT)
FROM PRODUIT
```

→ 19,20

### ❖ Ne pas confondre **COUNT** (nombre d'enreg.) et **SUM** (somme sur un champ) :

```
SELECT COUNT(PRIXHT)
FROM PRODUIT
WHERE PRIXHT < 5
```

→ 3 (3 enregistrements vérifiant la condition)

```
SELECT SUM(PRIXHT)
FROM PRODUIT
WHERE PRIXHT < 5
```

→ 9,25 (9,25 = 2,30 + 4,75 + 2,20)

21

## Nommage des colonnes

### ❖ On peut nommer une colonne affichée par une requête dans **SELECT**

- intérêt : réutiliser ce nom dans une autre requête

### ❖ Pour nommer une colonne :

```
SELECT expr AS nom
FROM ...
WHERE ...
```

### ❖ Exemple :

```
R1
SELECT REFPROD, PRIXHT * 1.196 AS PRIX TTC
FROM PRODUIT
```

R1	REFPROD	PRIX TTC
	1	22,96
	5	2,75
	9	6,87
	7	2,63

```
SELECT REFPROD, PRIX TTC
FROM R1
WHERE PRIX TTC > 6.10
```

	REFPROD	PRIX TTC
	1	22,96
	9	6,87

23

## Calculs sur les colonnes

PRODUIT	REFPROD	DESIGN	PRIXHT
	1	Livre	19,20
	5	Cahier	2,30
	9	Crayon	4,75
	7	Style	2,20

EST-FACTURÉ	NUMFACT	REFPROD	QTE
	6	1	7
	6	5	3
	2	5	10
	2	1	5
	3	7	4

### ❖ On peut effectuer des calculs sur une ou plusieurs colonnes en utilisant

- des constantes
- des noms de champ
- des opérateurs : +, -, \*, /, ...
- des fonctions : **sin**, **exp**, ...

### ❖ Exemple : réf. des produits avec leur prix TTC

```
SELECT REFPROD, PRIXHT * 1.196
FROM PRODUIT
```

→

REFPROD	
1	22,96
5	2,75
9	6,87
7	2,63

### ❖ On peut combiner calcul sur lignes et colonnes dans une même requête :

- Exemple : montant HT de la facture 6

```
SELECT SUM(PRIXHT * QTE)
FROM PRODUIT, EST-FACTURE
WHERE PRODUIT.REFPROD = EST-FACTURE.REFPROD
AND NUMFACT = 6
```

→ 141,3

22

## Regroupements d'enregistrements

### ❖ **Regroupement** : opération d'agrégation sur une ou plusieurs dimensions

### ❖ Exemples

- CA par client (regroupement par client)
- nombre de ventes par commercial et par mois (regroupement par commercial et par mois)
- ...

### ❖ Regrouper des enregistrements :

```
SELECT champ1, ..., champn, expr1, ..., exprm
FROM ...
WHERE ...
GROUP BY champ1, ..., champn
```

affiche les groupes  
calculs sur les groupes  
champ(s) sur le(s)quel(s) s'effectue les regroupements

### ❖ Exemple : quantités facturées par produit

EST-FACTURÉ	NUMFACT	REFPROD	QTE
	6	1	7
	6	5	3
	2	5	10
	2	1	5
	3	7	4

→

```
SELECT REFPROD, SUM(QTE)
FROM EST-FACTURE
GROUP BY REFPROD
```

→

REFPROD	
1	12
5	13
7	4

24

## Exemple

- ❖ Numéro de la facture ayant le montant le plus élevé

### R1 : Montant de chaque facture

```
SELECT  NUMFACT, SUM (PRIXHT *QTE) AS MONTANT
FROM    PRODUIT, EST-FACTURE
WHERE   EST-FACTURE.REFPROD = PRODUIT.REFPROD
GROUP BY NUMFACT
```

PRODUIT	REFPROD	DESIGN	PRIXHT
1	Livre		19,20
5	Cahier		2,30
9	Crayon		4,75
7	Stylo		2,20

EST-FACTURÉ	NUMFACT	REFPROD	QTE
	6	1	7
	6	5	3
	2	5	10
	2	1	5
	3	7	4

### R2 : Maximum des montants

```
SELECT MAX(MONTANT) AS MAXMONTANT
FROM R1
```

R1	NUMFACT	MONTANT
	2	119,00
	3	8,80
	6	141,30

### R3 : Numéro de la facture ayant le montant le plus élevé

```
SELECT NUMFACT
FROM R1,R2
WHERE R1.MONTANT=R2. MAXMONTANT
```

R2	MAXMONTANT
	141,30

R3	NUMFACT
	6

25

## Opérations de définition et de manipulation de données en SQL

- ❖ Création de table : **CREATE**
- ❖ Insertion d'enregistrements : **INSERT**
- ❖ Suppression d'enregistrements : **DELETE**
- ❖ Mise à jour d'enregistrements : **UPDATE**

26

## Création de table

### FACTURE (NUMFACT, DATFACT)

- ❖ Deux méthodes pour la création d'une table

- en utilisant le langage SQL

```
CREATE TABLE FACTURE (
  NUMFACT NUMBER IDENTITY
  PRIMARY KEY,
  DATEFACT DATE NOT NULL
);
```

- en utilisant une interface graphique spécifique à chaque SGBD

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT
1	NUMFACT	NUMBER	No	(null)
2	DATEFACT	DATE	Yes	(null)

27

## Contraintes d'intégrité

- ❖ Clause permettant de contraindre la modification de tables (conformité aux données attendues). Les contraintes doivent être exprimées dès la création de la table grâce aux mots clés suivants
  - **DEFAULT** : valeur par défaut
  - **NOT NULL** : Le champ doit être saisi
  - **UNIQUE** : valeur saisie pour un champ n'existe pas déjà dans la table
  - **CHECK** : faire un test sur un champ
  - **CONSTRAINT** : donner un nom à une contrainte

- ❖ Exemple

```
CREATE TABLE client(
  Numcli Number UNIQUE,
  Nom varchar2(30) NOT NULL,
  Prenom varchar2(30) NOT NULL,
  Age Number, check (age < 100),
  Email varchar2(50) NOT NULL, check (Email LIKE "%@%"),
  DateEnreg date DEFAULT sysdate );
```

28

## Contraintes : Définition des clés

- ❖ Une **clé primaire** se définit grâce à la clause **PRIMARY KEY** suivie de la liste de colonnes, séparées par des virgules, entre parenthèses : **PRIMARY KEY (colonne1, colonne2, ...)**
- ❖ Lorsqu'une liste de colonnes de la table en cours de définition permet de définir la clé primaire d'une table étrangère, on parle alors de **clé étrangère**, et on utilise la clause **FOREIGN KEY** suivie de la liste de colonnes de la table en cours de définition, séparées par des virgules, entre parenthèses, puis de la clause **REFERENCES** suivie du nom de la table étrangère et de la liste de ses colonnes correspondantes, séparées par des virgules, entre parenthèses : **FOREIGN KEY (colonne1, colonne2, ...) REFERENCES NomTable (colonne1, colonne2, ...)**

### ❖ Exemples

```
alter table client add CONSTRAINT pk_client  
PRIMARYKEY (numcli);
```

```
alter table livraison add CONSTRAINT fk_livraison  
FOREIGN KEY (numfou) REFERENCES fournisseur (numfou);
```

31

## Insertion d'enregistrements : INSERT – 2 / 2

- ❖ L'ajout d'enregistrements spécifiés par une requête s'effectue de la manière suivante :

```
INSERT INTO table (champ1, ... , champn)  
SELECT ... FROM ... WHERE ...
```

- le nombre et le type des champs dans la clause **SELECT** doivent être identiques à ceux des champs insérés dans **table**
- exemple : ajouter à la table ANC-PRODUIT tous les produits de PRODUIT

```
INSERT INTO ANC-PRODUIT (REFPROD, DESIGN, PRIXHT)  
SELECT REFPROD, DESIGN, PRIXHT FROM PRODUIT
```

31

## Insertion d'enregistrements : INSERT – 1 / 2

- ❖ L'insertion d'enregistrements permet
  - d'ajouter **un** enregistrement dans une table
  - d'ajouter **un ensemble** d'enreg. spécifié au moyen d'une requête SELECT
- ❖ L'ajout d'un seul enregistrement s'effectue de la manière suivante :

```
INSERT INTO table (champ1, ... , champn)  
VALUES (valeur1, ... , valeur_n)
```

- exemple : ajouter le produit de ref. n° 19 et de désignation 'Agrafeuse' dont le prix HT est 50,6 dh (PRODUIT (REFPROD, DESIGN, PRIXHT) )

```
INSERT INTO PRODUIT (REFPROD, DESIGN, PRIXHT)  
VALUES (19, 'Agrafeuse', 50.6)
```

- ❖ Remarque : l'insertion dans une table dont la clé est auto-incrémentée **ne doit pas** fixer de valeur pour ce champ
  - exemple : insertion en supposant le champ REFPROD auto-incrémenté

```
INSERT INTO PRODUIT (DESIGN, PRIXHT)  
VALUES ( 'Agrafeuse', 50.6)
```

30

## Suppression d'enregistrements : DELETE – 1 / 2

- ❖ La suppression d'enregistrements s'effectue de la manière suivante :

```
DELETE  
FROM table  
WHERE condition
```

- ⚠ les enregistrements concernés sont **immédiatement** supprimés

- ❖ Exemple : supprimer toutes les factures émises après le 16/10/2010

```
DELETE  
FROM FACTURE  
WHERE DATFACT > #2010/10/16#
```

32




## Mise à jour d'enregistrements : UPDATE – 1 / 2

---

- ❖ La mise à jour d'enregistrements s'effectue de la manière suivante :

```
UPDATE table
SET   champ1 = valeur1 , ... , champn = valeurn
WHERE condition
```

 les enregistrements sont définitivement mis à jour

- ❖ Exemple : remplacer la référence de produit 4 par 7 dans les factures

```
UPDATE EST-FACTURE
SET   REFPROD = 7
WHERE REFPROD = 4
```

33

## PL-SQL

---

- ❖ Langage Procédural pour SQL d'Oracle
- ❖ Proche de Pascal et Ada
- ❖ Facilités pour
  - Variables
  - Conditions
  - Boucles
  - Exceptions
  - Stockage de procédures
- ❖ Implémentations plus ou moins conformes
  - PostgreSQL : PL/pgSQL
  - SQL Server : Transact-SQL
  - DB2 : SQLPL
  - MySql : Stored procedure
  - etc.

35

## Chapitre 1

### PL/SQL, Triggers et Vues

## PL/SQL : Que peut-on faire avec ?

---

- ❖ Code pour automatiser un traitement
  - Processus périodiques
  - Processus ponctuels
  - Contraintes d'intégrité élaborées
  - Etc.
- ❖ Applications
  - Auditing
  - Sécurité
  - Vérification

36

## PL/SQL : Structure d'un bloc

- ❖ Un bloc contient 3 parties
  - Déclarations (optionnel)
  - Commandes exécutables (corps)
  - Gestion des exceptions (optionnel)

```
[DECLARE  
-- déclarations de types,  
-- variables locales au bloc,  
-- constantes,  
-- exceptions et curseurs]  
BEGIN [<nombloc>]  
-- instructions PL/SQL et SQL  
-- possibilité de blocs imbriqués  
[EXCEPTION  
-- Traitement des erreurs]  
END; /* ou END <nombloc> ; */
```

37

## PL/SQL : Déclarations

- ❖ Types
  - issus de SQL : number, date, varchar2, ...
  - boolean, integer, float, real,...
- ❖ Variables
  - De manière générale : <nom\_variable> <type\_variable> ;
  - Déclaration avec une valeur au départ <Nom\_variable> <type> **default** <valeur> ;
- ❖ Exemples
  - Nomcli **varchar2(25)** ;
  - N **number default 2** ;
- ❖ Visibilité d'une variable
  - dans le bloc où elle est déclarée
  - dans les blocs imbriqués (sauf si redéfinie dans bloc imbriqué)
- ❖ Déclarations de constantes  
    <Nom\_variable> **constant** <type> := <valeur>;

38

## PL/SQL : Corps

- ❖ Le corps peut comporter des instructions
  - d'affectation
  - SQL : commit, delete, insert, locktable, open, rollback, savepoint, select, set transaction, update...
  - de contrôle (conditionnelles, répétitives)
  - de gestion des erreurs
- ❖ Chaque instruction est terminée par « ; »
- ❖ L'imbrication de blocs est possible, mais pas recommandée
- ❖ Affectation d'une variable
  - Opérateur d'affectation (:=)
  - ex. Produit.LIBPROD := 'Livre' ;
- ❖ Option **into** de l'ordre select

```
Select LIBPROD into designation  
from Produit  
where REFPROD= 2;
```

39

## PL/SQL : Exemple

**FOURNISSEUR** ( numfou **NUMBER**, nomfou **VARCHAR2(30)**);

**LIVRAISON** (numli **NUMBER**, numfou **NUMBER**, dateli **DATE** default sysdate );

Nom du fournisseur de la livraison numéro 10 ?

```
DECLARE  
    num NUMBER:= 10;  
    nom VARCHAR2 (30) ;  
BEGIN  
    SELECT nomfou INTO nom  
    FROM Fournisseur, Livraison  
    WHERE numli= num and Fournisseur.numfou=Livraison.numfou;  
    DBMS_OUTPUT.PUT_LINE('La livraison numéro : ' || num || ' concerne le  
fournisseur : ' || nom);  
END;
```

40

## PL/SQL : Structures de contrôle

### ❖ Structure conditionnelle

```
IF <condition> THEN
<instruction>;...
  [ELSIF <condition> THEN <instruction>; ... <instruction>; ]
  [ELSE <instruction>; ... <instruction>]
END IF;
```

### ❖ Boucle répétitive simple

```
LOOP
<instruction>; ... <instruction>;
END LOOP;
```

#### *Sortie d'une boucle*

```
IF <condition> THEN EXIT ; END IF;
Ou EXIT WHEN <condition> ;
```

### ❖ Boucle Tant que

```
WHILE <condition>
LOOP <instruction>; ... <instruction>;
END LOOP;
```

### ❖ Boucle For

```
FOR <variable_boucle> IN <borne_inf> ..<borne_sup>
LOOP
<instruction>; ... <instruction>;
END LOOP;
```

41

## Structures de contrôle : Exemples

### ❖ Loop

```
DECLARE
    compteur NUMBER;
    somme NUMBER := 0;
    moyenne NUMBER;

BEGIN
    compteur := 1;

    LOOP
        somme := somme + compteur ;
        compteur := compteur + 1;
        EXIT WHEN compteur =11;
    END LOOP;
    moyenne := somme / 10;
    DBMS_OUTPUT.PUT_LINE ('La moyenne est '||TO_CHAR(moyenne));
END;
```

42

## Structures de contrôle : Exemples

### ❖ While

```
DECLARE
    compteur NUMBER;
    somme NUMBER := 0;
    moyenne NUMBER;

BEGIN
    compteur := 1;
    WHILE compteur <= 10 LOOP
        somme := somme + compteur ;
        compteur := compteur + 1;
    END LOOP;
    moyenne := somme / 10;
    DBMS_OUTPUT.PUT_LINE ('La moyenne est '||TO_CHAR(moyenne));
END;
```

43

## Structures de contrôle : Exemples

### ❖ For

```
DECLARE
    somme NUMBER := 0;
    moyenne NUMBER;

BEGIN
    FOR compteur IN 1 .. 10 LOOP
        somme := somme + compteur ;
    END LOOP;
    moyenne := somme / 10;
    DBMS_OUTPUT.PUT_LINE ('La moyenne est '||TO_CHAR(moyenne));
END;
```

44

## Procédures PL/SQL

❖ Utilisées pour enregistrer des traitements fréquemment utilisés au niveau du noyau **sans valeur de retour**

❖ Syntaxe

```
CREATE OR REPLACE PROCEDURE /* nom */ ( /* paramètres */ ) IS
    /* déclaration des variables locales */
BEGIN
    /* instructions */
END;
```

45

## Procédures PL/SQL : Exemple

```
CREATE OR REPLACE PROCEDURE compteAREbours (n NUMBER) IS
BEGIN
    IF n >= 0 THEN
        DBMS_OUTPUT.PUT_LINE ( n );
        compteAREbours ( n - 1 );
    END IF;
END;
```

❖ Invocation

- En PL/SQL, une procédure s'invoque tout simplement avec son nom
- Sous SQL+, on doit utiliser le mot-clé **CALL**. Par exemple, on invoque le compte à rebours sous SQL+ avec la commande **CALL** compteAREbours(20)

46

## Fonctions PL/SQL

❖ Utilisées pour enregistrer des traitements fréquemment utilisés au niveau du noyau **avec valeur de retour**

❖ Syntaxe

```
CREATE OR REPLACE FUNCTION /* nom */ ( /* paramètres */ )
RETURN /* types */
IS
    /* déclaration des variables locales */
BEGIN
    /* instructions */
END;
```

47

## Fonctions PL/SQL : Exemple

```
CREATE OR REPLACE FUNCTION Maximum( a NUMBER, b NUMBER)
RETURN NUMBER
IS
BEGIN
    IF a < b THEN RETURN b ;
    ELSE
        RETURN a ;
    END IF;
END;
```

❖ Invocation

- En PL/SQL, une fonction s'invoque tout simplement avec son nom
- Sous SQL+, On passe par une pseudo-table nommée **DUAL** de la façon suivante : **SELECT** Maximum(21, 12) **FROM** **DUAL**;

48

## Triggers

- ❖ Objectif : Comment assurer une cohérence sémantique plus complexe
  - Lorsqu'une table est modifiée, générer la modification d'une autre
  - Lorsqu'une valeur est insérée la modifier avant de la stocker dans la BD
  - Etc.
- ❖ Règle : Événement-Condition-Action (ECA)
  - Évènement
    - Insert, delete, update sur une table ou vue
  - Condition
    - Test ou prédicat logique
  - Action
    - Si la condition est satisfaite, code PL/SQL à exécuter

49

## Triggers : Exemple

CATALOGUE ( numcat NUMBER, datedeb DATE, datefin DATE)

❖ Trigger sur la relation CATALOGUE. On empêche l'insertion ou la modification d'un tuple si la date de début de la validité du catalogue est supérieure à celle de sa fin

```
CREATE OR REPLACE TRIGGER CATALOGUE_bef_ins_upd_row
BEFORE INSERT OR UPDATE
ON CATALOGUE
FOR EACH ROW
DECLARE
    dates_error EXCEPTION;
BEGIN
    IF :new.datedeb > :new.datefin THEN RAISE dates_error;
END IF;
EXCEPTION
    WHEN dates_error THEN
        RAISE_APPLICATION_ERROR (-20001, 'Date de début est supérieure à la date
de fin');
END;
```

51

## Syntaxe

```
create [or replace] trigger [schema .] trigger
{ before | after | instead of }
{ dml_event_clause Insert | Update | Delete
| { ddl_event [or ddl_event]...
  | database_event [or database_event]...
}
on { [schema .] schema | database }
}
[when ( condition ) ]
{ pl/sql_block | call_procedure_statement }
```

Les attributs des tables sont accessibles à travers les variable :NEW et :OLD

50

## Vues

- ❖ Une vue est une requête stockée qui est interrogée comme une table
- ❖ Création d'une vue

CREATE VIEW <nom-vue> AS <requête définissant la vue>

- ❖ Exemple

PRODUIT ( numprod NUMBER, nomprod VARCHAR2(30))

PROPOSER ( numfou NUMBER, numprod NUMBER, prix NUMBER NOTNULL)

- Créer une vue appelée « ProduitAffaire » sur les produits dont le prix ne dépasse pas 1000DH (Attributs de la vue : numprod, nomprod, prix)

```
CREATE VIEW ProduitAffaire AS
SELECT numprod, nomprod, prix
FROM PRODUIT,PROPOSER
WHERE prix<1000 and PRODUIT.numprod = PROPOSER.numprod;
```

52