A thick blue arrow originates from the right side of the two pink cylinders and points towards the word "SQL".

SQL

# Le Langage SQL

## Introduction

# Historique du Langage SQL



- E. F. CODD : premiers articles dans les années 70
- IBM crée le langage SEQUEL (*Structured English Query Language*) ancêtre du langage SQL
- Ce langage devient SQL (*Structured Query Language*, prononcer *sikuel*)
- En 1979, Relational Software Inc. (devenu depuis Oracle) met en place la première version commerciale de SQL
- Principaux SGBDR : Oracle, DB2, Informix, SQL-Server, Ingres, MySQL, Interbase, ....

# Norme et Standard du langage SQL



SQL

- Base du succès de SQL
- Fin des SGBD constructeurs
- ANSI (*American National Standards Institute*) et de l'ISO (*International Standards Organization*) qui est affilié à l'IEC (*International Electrotechnical Commission*)
- L'ANSI et l'ISO/IEC ont accepté SQL comme le langage standardisé. La dernière norme publiée par l'ANSI et l'ISO est SQL92 (SQL2)
- On attend la norme SQL3 ....

# Caractéristiques de la norme SQL



- Oracle et IBM participent activement au sein du groupe ANSI
- SQL92 définit quatre niveaux : *Entry*, *Transitional*, *Intermediate*, et *Full*
- Un SGBDR doit supporter au moins une implémentation de SQL de type *Entry*
- Oracle9i est totalement compatible *Entry* et a beaucoup de caractéristiques de type *Transitional*, *Intermediate*, et *Full*

# Les sous-langages de SQL



- LDD : Langage de Définition des Données
  - Création, Modification et Suppression des objets
  - Objets : tables, index, cluster, privilèges, ....
- LMD : Langage de Manipulation des Données
  - Ajout, Modification et Suppression des données
  - Notion de Transaction
- LID : Langage d'Interrogation des Données
  - Sélection (recherche) de l'information
  - Mise en œuvre du langage relationnel
- LCD : langage de Contrôle des Données
  - Notion de sous-schéma ou schéma externe
  - Notion de rôles et de privilèges

# SQL avancé



- **Langage de bloc pour augmenter la puissance de SQL :**
  - Fonctions itératives et alternatives
  - PL/SQL avec Oracle, Transact-SQL avec SQL-Server
- **Notion de Déclencheur ou Trigger**
  - MAJ automatique de colonnes dérivées
  - Contraintes complexes
- **Notion de Procédure Stockée**
  - Programme SQL stocké (compilé) dans la base
- **SQL encapsulé : SQL embarqué dans un langage externe**
  - Géré par le SGBD : PRO\*C, PRO\*ADA, ...
  - Extérieur au SGBD : VB, C#, ...

# Apprendre SQL avec Oracle



- **SGBD le plus répandu dans le monde (gros, moyens et petits systèmes)**
- **SGBD le plus normalisé**
- **Produit téléchargeable sur oracle.com à des fins d'apprentissage**
- **Interface SQL\*Plus pour dialoguer avec le langage SQL**



# Offre complète d'Oracle : Produits proposés

- Noyau Oracle Serveur
  - DBMS : gestionnaire de bases de données
  - Création d'une ou plusieurs instances
  - Licence serveur minimale
  - Toutes plates-formes acceptées
  - Driver SQL\*Net serveur
  - PL/SQL : langage de bloc propriétaire
- SQL\*Plus
  - Interface minimale pour exécuter des requêtes SQL

```
SQL> SELECT * FROM emp ;
```
  - Envoi de requêtes et retour des résultats sur écran
  - Appel de blocs, procédures, fonctions...

# Offre complète d'Oracle (suite)



SQL

- Enterprise Manager
  - Interface graphique pour administrer la base de données distante (DBA)
  - Administration système Oracle
  - Ajout, modification et suppression de tous les objets de la base
  - Surveillance des activités
- SQL\*Net
  - Driver propriétaire de communication client et serveur
  - Nécessaire en Client - Serveur et les BD réparties
- Oracle Application Server
  - Pilier de NCA (*Network Computing Architecture*)
  - Serveur Web Transactionnel
  - Cartouches PL/SQL, Java...
  - Intègre le standard CORBA et IIOP

# Offre complète d'Oracle (suite)



**SQL**

A large, stylized text element where the letters 'S', 'Q', 'U', and 'L' are stacked vertically, with a long pink arrow pointing from the right towards the text.

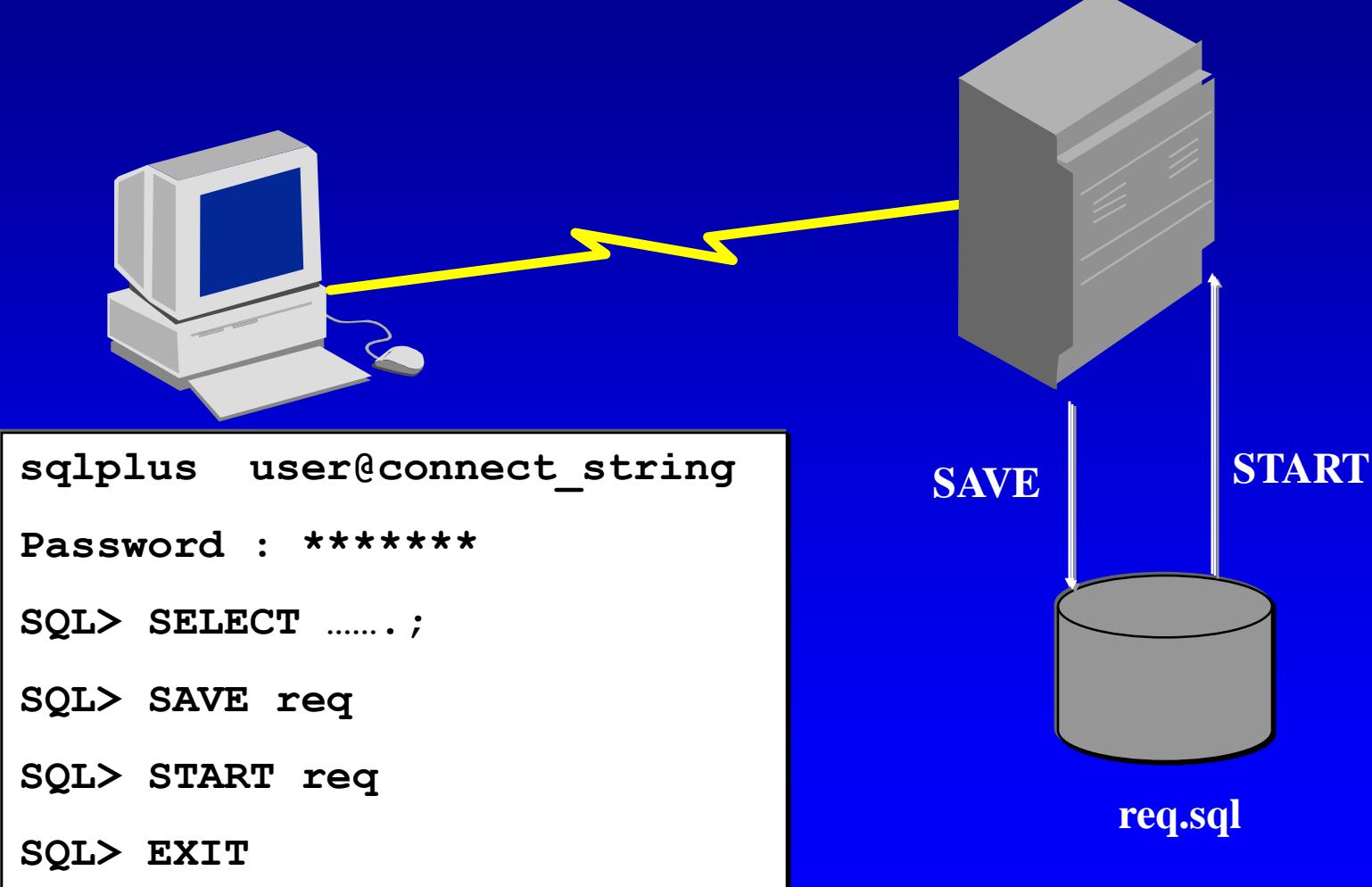
- **Oracle WareHouse**
  - Serveur OLAP
  - Analyse décisionnelle
- **Oracle Database Designer**
  - Atelier de génie logiciel
  - Construction du système d'information (données et programme)
  - Reverse engineering
- **Oracle Developer 2000**
  - Outil graphique de développement propriétaire
  - Intègre le produit SQL\*Forms

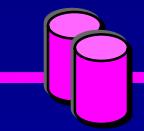
# Offre complète d'Oracle (fin)



- **Oracle Inter-Office**
  - Outil de Workflow
  - Gestion des flux de documents électronique
  - Concurrents : Lotus Notes et Exchange
- **Oracle Portal**
  - Portail d'entreprise
  - Point d'entrée unique de travail
- **Certains produits sont aujourd'hui proposés en standard avec Oracle 9i**

# SQL\*Plus



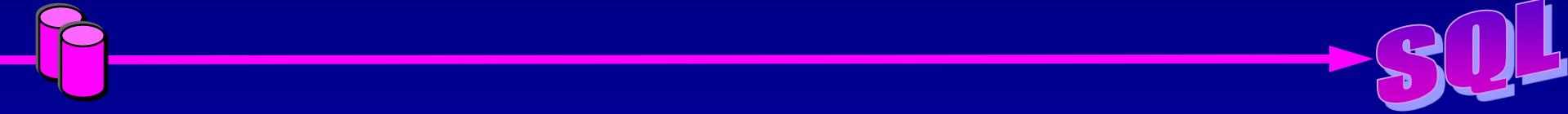


SQL

# Le Langage de Définition de Données

LDD

# Les ordres et les objets



- **Ordre CREATE**
  - Crédit de la structure de l'objet → DD
- **Ordre DROP**
  - Suppression des données et de la structure
- **Ordre ALTER**
  - Modification de la structure (contenant)
- **Syntaxe <Ordre> <Objet> < nom\_objet >**
- **Objet TABLE**
- **Objet INDEX**
- **Objet CLUSTER**
- **Objet SEQUENCE**

# Objet Table et Contraintes

## CREATE : Syntaxe



SQL

```
create table nom_table
  (colonne1           type1(longueur1),
   colonne2           type2(longueur2),
   .....
   constraint nom_constraint1
     type_constraint1,
   .....
  ) ;
```

3 Types de Contraintes

PRIMARY KEY

FOREIGN KEY

CHECK (NOT NULL, UNIQUE)

# Objet Table et Contraintes

## Les types de données



SQL

VARCHAR ( <i>size</i> )	Données caractères de longueur variable
CHAR ( <i>size</i> )	Données caractères de longueur fixe
NUMBER ( <i>p</i> , <i>s</i> )	Numérique de longueur variable
DATE	Valeurs de date et d'heure
LONG	Données caractères de longueur variable (2 Go)
CLOB	Données caractères (4 Go)
RAW	Binaire
BLOB	Binaire, jusqu'à 4 giga-octets
BFILE	Binaire, stocké dans un fichier externe, (4 Go)

# Objet Table et Contraintes

## CREATE : Exemples



SQL

```
-- Table 'Mère'

CREATE TABLE service
  (IdService           CHAR(3) ,
   NomService          VARCHAR(30) ,
   CONSTRAINT pk_service
     PRIMARY KEY (IdService)
  ) ;
```

# Objet Table et Contraintes

## CREATE : Exemples (suite)

```
-- Table 'Fille'  
CREATE TABLE employe  
  (IdEmploye          NUMBER(5) ,  
   NomEmploye         VARCHAR(30) ,  
   Indice             NUMBER(3) ,  
   DateEmbauche      DATE DEFAULT SYSDATE ,  
   IdService          CHAR(3)  
     CONSTRAINT nn_emp_ser NOT NULL,  
   CONSTRAINT pk_employe  
     PRIMARY KEY(IdEmploye) ,  
   CONSTRAINT fk_emp_ser FOREIGN KEY(IdService)  
     REFERENCES service(IdService) ,  
   CONSTRAINT ck_emp_indice CHECK  
     (indice BETWEEN 100 AND 900)  
 ) ;
```

# Objet Table : DROP



SQL

```
DROP TABLE nom_table;
```

Suppression complète de la table : définition et données

```
DROP TABLE nom_table CASCADE CONSTRAINTS;
```



Suppression aussi des contraintes de référence filles

# Modification de la structure

## ALTER TABLE



SQL

### Ajout de colonnes

```
ALTER TABLE nom_table  
    ADD (colonne1 type1, colonne2 type2);
```

### Modification de colonnes

```
ALTER TABLE nom_table  
    MODIFY (colonne1 type1, colonne2 type2);
```

### Suppression de colonnes

```
ALTER TABLE nom_table  
    DROP COLUMN (colonne1, colonne2);
```



# ALTER TABLE

## Exemples de modifications



SQL

```
ALTER TABLE client  
ADD ChiffreAffaire NUMBER (10,2);
```

```
ALTER TABLE client MODIFY nom VARCHAR(60);
```

```
ALTER TABLE etudiant  
MODIFY idDiplome CONSTRAINT nn_etu_dip NOT NULL;
```

```
ALTER TABLE client  
DROP COLUMN ChiffreAffaire ;
```

# Contraintes

```
constraint nomcontrainte
{ unique | primary key (col1[,col2]...)
           | foreign key (col1[,col2]...)
references [schema].table (col1[,col2]...)
[ON DELETE CASCADE]
           | check (condition) }
```



Attention : suppression de tous les fils !

# Modification des contraintes

## Ajout et Suppression



SQL

### Ajout de contraintes

```
ALTER TABLE nom_table  
    ADD CONSTRAINT nom_contrainte  
        type_contrainte;
```



Comme à la création d'une table

### Suppression de contraintes

```
ALTER TABLE nom_table  
    DROP CONSTRAINT nom_contrainte;
```

# Modification des contraintes

## Exemples



```
ALTER TABLE client
    ADD CONSTRAINT fk_client_cat
    FOREIGN KEY(idCat)
    REFERENCES categorie(idCat);
```

```
ALTER TABLE client
    DROP CONSTRAINT fk_client_cat;
```



# Activation et désactivation de contraintes

## Désactivation de contraintes

```
ALTER TABLE nom_table  
    DISABLE CONSTRAINT nom_constrainte;
```

```
ALTER TABLE nom_table  
    DISABLE CONSTRAINT PRIMARY KEY;
```

- Les contraintes existent toujours dans le dictionnaire de données mais ne sont pas actives
- Chargement de données volumineuses extérieures à la base



# Activation d'une contrainte désactivée

SQL

## Activation de contraintes

```
ALTER TABLE nom_table  
ENABLE CONSTRAINT nom_contrainte ;
```

```
ALTER TABLE nom_table  
ENABLE CONSTRAINT PRIMARY KEY;
```

# Ajout ou activation de contraintes : Récupération des lignes en erreur



SQL

## Création d'une table Rejets

```
CREATE TABLE rejets
(ligne           rowid,    ← Adresse ligne
 proprietaire   varchar(30),
 nom_table      varchar(30),
 constraint      varchar(30));
```

## Activation de contraintes

```
ALTER TABLE nom_table
    ENABLE CONSTRAINT nom_constrainte
    EXCEPTIONS INTO rejets;
```

# Vérification de Contraintes différées



SQL

- Une contrainte peut-elle être différée ?
  - NOT DEFERRABLE (par défaut)
  - DEFERRABLE
- Comportement par défaut de la contrainte :
  - INITIALLY IMMEDIATE (par défaut)
  - INITIALLY DEFERRED
- Utiliser la clause SET CONSTRAINTS ou ALTER SESSION SET CONSTRAINTS pour modifier le comportement d'une contrainte

# Vérification de Contraintes différées

## Exemples



```
CREATE TABLE emps ...
    dept_id      NUMBER(6)
        CONSTRAINT fk_emp_dept REFERENCES depts
            DEFERRABLE INITIALLY IMMEDIATE);
```

```
SQL> INSERT INTO emps VALUES (1, 'Laurent', 2)
*
ERROR at line 1:
ORA-02291: integrity constraint (MICHEL.FK_EMP_DEPT_ID)
violated - parent key not found ;
```

```
SET CONSTRAINTS ALL DEFERRED;
```

```
SQL> INSERT INTO emps VALUES (1, 'Laurent', 2);
1 row created.
```

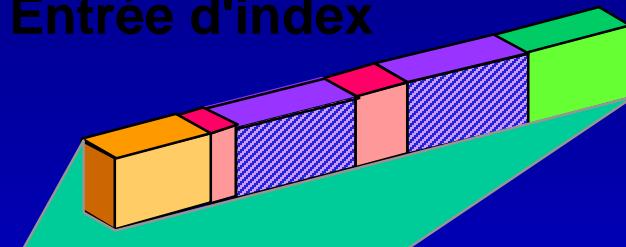
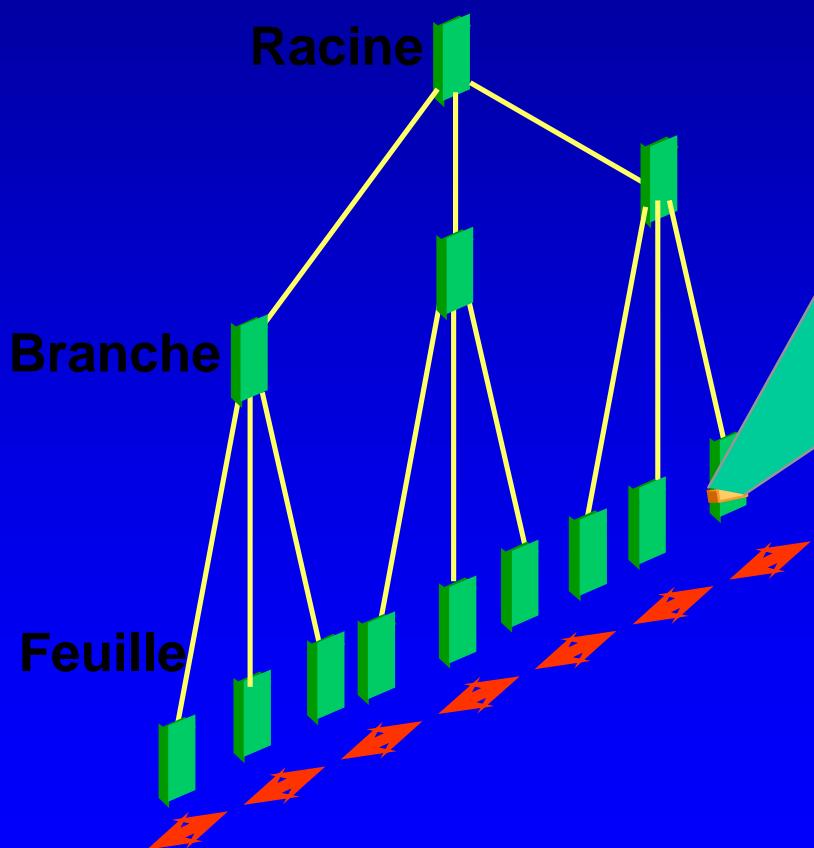
```
SQL> COMMIT;
COMMIT
*
ERROR at line 1:
ORA-02091: transaction rolled back
ORA-02291: integrity constraint (MICHEL.FK_EMP_DEPT_ID)
violated - parent key not found
```

# Les fichiers Index : organisation en B-Arbre



SQL

Entrée d'index



- En-tête d'entrée d'index
- Longueur de la colonne de la clé
- Valeur de la colonne de la clé
- ROWID

# Création et suppression d'Index



SQL

## Création d'un index Unique

```
CREATE UNIQUE INDEX nom_index  
ON nom_table(colonne[,colonne2 ...]);
```

## Création d'un index non Unique

```
CREATE INDEX nom_index  
ON nom_table(colonne[,colonne2 ...]);
```

## Suppression d'un index

```
DROP INDEX nom_index;
```

# Apport des Index



- Respect de la 4NF : clé primaire → index unique
- Amélioration des accès (sélection) sur les colonnes recherchées
- Optimisation des jointures (équi-jointure entre une clé primaire et sa clé étrangère)
- Attention aux clés primaires composées
- Table USER\_INDEXES du dictionnaire

# Cluster : Jointure physique



SQL

DEPT

no_dept	nom_dept	resp_dept
10	Info	Jean
20	Ventes	Xavier
30	Achats	Paul

EMP

No_emp	nom_emp	no_dept
100	Pierre	20
101	Sylvie	10
102	Michel	10
104	Corinne	20

Cluster Key  
(no\_dept)

10 Info Jean  
101 Sylvie  
102 Michel

20 Ventes Xavier  
100 Pierre  
104 Corinne

30 Achats Paul

Bloc1

Bloc2

Bloc3

Tables DEPT et EMP  
non-clusterisées

Tables DEPT et EMP  
clusterisées

# Création de Cluster (1)



SQL

## 1. Création du cluster

```
CREATE CLUSTER personnel
  (no_dept NUMBER(3))
  SIZE 200 TABLESPACE ts1
  STORAGE (INITIAL 5M NEXT 5M PCTINCREASE 0);
```

Taille du  
bloc logique

## 2. Création de l'index de cluster

```
CREATE INDEX idx_personnel
  ON CLUSTER personnel
  TABLESPACE tsx1
  STORAGE (INITIAL 1M NEXT 1M PCTINCREASE 0);
```

# Création de Cluster (2)



SQL

## 3- Création des tables dans le cluster

```
CREATE TABLE dept  
(no_dept NUMBER(3)  
CONSTRAINT pk_dept PRIMARY KEY,  
nom_dept VARCHAR(30), resp_dept VARCHAR(30))  
CLUSTER personnel(no_dept);
```

```
CREATE TABLE emp  
(no_emp NUMBER(3) CONSTRAINT pk_emp PRIMARY KEY,  
nom_emp VARCHAR(30),  
no_dept NUMBER(3) REFERENCES dept(no_dept))  
CLUSTER personnel(no_dept);
```

# Administration des Clusters



SQL

Modification des paramètres de stockage et  
de consommation d'espace d'un bloc

```
ALTER CLUSTER personnel  
SIZE 300K STORAGE (NEXT 2M);
```

Suppression des Clusters

```
DROP CLUSTER personnel  
INCLUDING TABLES;
```

OU

```
DROP TABLE emp;  
DROP TABLE dept;  
DROP CLUSTER personnel;
```

# Objet Séquence



- Permet d'obtenir des valeurs incrémentales
- Équivalent des colonnes AUTO\_INCREMENT de MySql ou IDENTITY de SqlServer
- N'est pas associée à une colonne particulière
- Verrouillage automatique en cas de concurrence d'accès
- Valeur suivante : <nom\_séquence>.NEXTVAL
- Valeur courante : <nom\_séquence>.CURRVAL



# Objet Séquence création et utilisation

SQL

```
CREATE SEQUENCE nom_séquence  
START WITH valeur_départ  
INCREMENT BY incrément;
```

```
INSERT INTO t1 VALUES  
(nom_séquence.NEXTVAL, ....);  
INSERT INTO t2 VALUES  
(....., nom_séquence.CURRVAL);
```

```
DROP SEQUENCE nom_séquence;
```

# Objet Séquence

## Exemple de mise en oeuvre



SQL

```
SQL> CREATE TABLE client (idClient NUMBER PRIMARY KEY,  
 2 nomClient VARCHAR(20));  
Table créée.  
SQL> CREATE TABLE compte (idCompte NUMBER PRIMARY KEY,  
 2 nomCompte VARCHAR(30), idClient REFERENCES client);  
Table créée.  
SQL> CREATE SEQUENCE seq_client START WITH 1 INCREMENT BY 1;  
Séquence créée.  
SQL> CREATE SEQUENCE seq_compte START WITH 1 INCREMENT BY 1;  
Séquence créée.  
SQL> INSERT INTO client VALUES(seq_client.NEXTVAL, 'Michel');  
1 ligne créée.  
SQL> SELECT seq_client.CURRVAL FROM dual;  
  CURRVAL  
-----  
 1
```

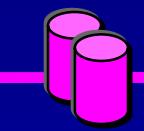
# Objet Séquence

## Exemple de mise en œuvre (suite)



SQL

```
SQL> INSERT INTO compte VALUES(seq_compte.NEXTVAL, 'Compte  
Courant Michel', seq_client.CURRVAL) ;  
1 ligne créée.  
  
SQL> INSERT INTO compte VALUES(seq_compte.NEXTVAL, 'Compte  
Epargne Michel', seq_client.CURRVAL) ;  
1 ligne créée.  
  
SQL> SELECT * FROM client;  
IDCLIENT NOMCLIENT  
-----  
1 Michel  
  
SQL> SELECT * FROM compte;  
IDCOMPTE NOMCOMPTE IDCLIENT  
-----  
1 Compte Courant Michel 1  
2 Compte Epargne Michel 1
```



**SQL**

A large, stylized, magenta/pink text representation of the acronym "SQL". It features a thick, three-dimensional font style and a slight shadow effect.

# Le Langage de Manipulation de Données

**LMD**

A large, stylized, cyan/light blue text representation of the acronym "LMD". It has a similar thick, three-dimensional font style as the SQL text above it.

# Les ordres SQL de manipulation



- **INSERT**
  - Insertion (ajout) de ligne(s) dans une table
  - Utiliser SQL\*LOAD pour des chargements externes
- **UPDATE**
  - Mise à jour (modification) de une ou plusieurs colonnes de une ou plusieurs lignes
- **DELETE**
  - Suppression de une ou plusieurs lignes
- **COMMIT / ROLLBACK**
  - Fin d'une transaction

# INSERT



SQL

```
INSERT INTO nom_table [(liste des colonnes)]  
VALUES (liste des valeurs);
```

Exemples :

```
INSERT INTO service (idSer, nomSer)  
VALUES (50, 'Réseaux et Systèmes');
```

```
INSERT INTO service  
VALUES (60, 'Analyse et Conception');
```

```
INSERT INTO service  
VALUES (60, NULL);
```



```
INSERT INTO service  
(idSer)  
VALUES (60);
```

# INSERT (suite)



SQL

Insert avec le contenu de une ou plusieurs tables

```
INSERT INTO etudiant_deug  
SELECT * FROM etudiant  
WHERE cycle = 1;
```

```
INSERT INTO etudiant_deug (nomd, prenomd, cycled)  
SELECT nom,prenom,1 FROM etudiant  
WHERE cycle = 1;
```

# UPDATE



SQL

```
UPDATE nom_table  
SET    colonne1 = valeur1  
       [,colonne2 = valeur2 ....]  
[ WHERE prédicat];
```

Exemples :

```
UPDATE employe  
SET    nom = 'Michel', adresse = 'Toulouse'  
WHERE idEmp = 100;
```

```
UPDATE employe  
SET    salaire = salaire * 1.1  
WHERE idSer = 'info';
```



UPDATE synchronisés : voir LID plus loin

# DELETE



SQL

```
DELETE FROM nom_table  
[WHERE prédicat];
```

Exemples :

```
DELETE FROM employe  
WHERE idEmp = 100;
```

```
DELETE FROM employe;
```



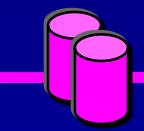
# COMMIT / ROLLBACK



- Notion de transaction : ensemble fini d'actions (update, delete et insert)
- Commit : point de confirmation dans la base
- Rollback ('retour arrière') : les actions sont 'défaites' jusqu'au dernier point de confirmation
- Le Commit peut être automatique (pas conseillé)



Voir la fonction d'Intégrité



SQL

Le Langage d'Interrogation  
de Données

LID

Mono-table

Mise en forme des résultats

# Sélection et Affichage Mono Table



SQL

- **Ordre SELECT**
- **Sélection : restriction sur les lignes**
- **Projection : fonctions de groupage et restrictions sur les groupages**
- **Ordres de Tris**
- **Fonctions Chaînes et Calculs**
- **Mise en page des résultats**

# Ordre SELECT



SQL

```
SELECT {DISTINCT|*|col1[,col2,...] [AS nom_col]}  
FROM nom_de_table
```

```
WHERE <prédicat sur les lignes>  
GROUP BY col1 [,col2,...]  
HAVING <prédicat sur les groupages>  
ORDER BY {col1 {ASC|DESC}  
[,col2 ...] | n°col }
```

# Exemples de SELECT MONO (1)



SQL

```
SELECT * FROM emp  
WHERE  
    idService IN (10,40,60)  
    AND (salaire BETWEEN 1000 AND 2000  
          OR  
          indice > 750 )  
    AND UPPER(adresse) LIKE '%TOULOUSE%' ;
```

Toutes les colonnes

Prédicat de restriction

```
SELECT nome AS "Nom Employé",  
       sal AS "Salaire Mensuel",  
       Sal*12 AS "Salaire Annuel" FROM emp  
WHERE idService NOT IN (10,40,60)
```

# Exemples de SELECT MONO (2) colonne virtuelle ROWNUM



SQL

```
SELECT * FROM emp  
WHERE ROWNUM < 50;
```

Colonne Virtuelle

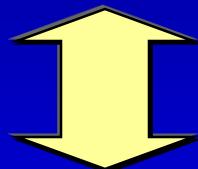
```
SELECT ROWNUM,e.* FROM emp e  
WHERE ROWNUM < 50;
```

# Exemples de SELECT MONO (3) tri du résultat



SQL

```
SELECT idService, nome, indice FROM emp  
ORDER BY idService , indice DESC, nome;
```

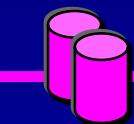


```
SELECT idService, nome, indice FROM emp  
ORDER BY 1 , 3 DESC, 2;
```

```
SELECT * FROM (  
SELECT idService, nome, indice FROM emp  
ORDER BY 1 , 3 DESC, 2)  
WHERE ROWNUM < 15;
```

Affichage des 15 premiers triés

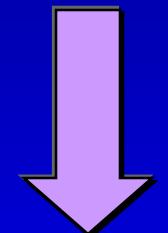
# Exemples de SELECT MONO (4) clause DISTINCT



SQL

```
SELECT DISTINCT idSer FROM emp;
```

<b>idEmp</b>	<b>nomEmp</b>	<b>idSer</b>	<b>salaire</b>
100	Michel	20	2000
200	Sylvie	10	3000
300	Bernard	20	1000
400	Claude	10	2000
500	Thomas	10	1000



<b>idSer</b>
-----
10
20

(Projection)

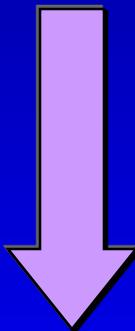
# Clause de groupage : GROUP BY fonctions de groupage



SQL

```
SELECT idSer, AVG(salaire), COUNT(*)  
FROM emp  
GROUP BY idSer;
```

idEmp	nomEmp	idSer	salaire
100	Michel	20	2000
200	Sylvie	10	3000
300	Bernard	20	1000
400	Claude	10	2000
500	Thomas	10	1000



idSer	AVG(salaire)	COUNT(*)
10	2000	3
20	1500	2

# Restriction sur les groupages : HAVING



SQL

```
SELECT idSer, AVG(salaire), COUNT(*)  
FROM emp  
GROUP BY idSer  
HAVING COUNT(*) >2;
```

idEmp	nomEmp	idSer	salaire
100	Michel	20	2000
200	Sylvie	10	3000
300	Bernard	20	1000
400	Claude	10	2000
500	Thomas	10	1000

restriction  
du groupage 20

idSer	AVG(salaire)	COUNT(*)
10	2000	3

# Mécanisme du GROUP BY et DISTINCT



SQL

- Si la colonne est indexée : la table n'est pas utilisée → optimisation
- Si la colonne n'est pas indexée : tri sur la colonne et regroupement → pas d'optimisation
- Utilisation d'un espace de travail si le volume est important (tablespace temporaire)
- Les résultats sont triés de façon ascendante

# Fonctions de groupages (agrégats)



SQL

AVG	Moyenne
COUNT	Nombre de lignes
GROUPING	Composer des regroupements ( <i>datawarehouse</i> )
MAX	Maximum
MIN	Minimum
STDDEV	Écart type
SUM	Somme
VARIANCE	Variance

# Fonctions numériques (1)



SQL

ABS	Valeur absolue
ACOS	Arc cosinus (de -1 à 1)
ADD_MONTHS	Ajoute des mois à une date
ATAN	Arc tangente
CEIL	Plus grand entier $\leq n$ , CEIL(15.7) donne 16
COS	Cosinus
COSH	Cosinus hyperbolique
EXP	Retourne e à la puissance n (e = 2.71828183)

# Fonctions numériques (2)



<b>FLOOR</b>	Tronque la partie fractionnaire
<b>CEIL</b>	Arrondi à l'entier le plus proche
<b>MOD (<math>m, n</math>)</b>	Division entière de $m$ par $n$
<b>POWER (<math>m, n</math>)</b>	$m$ puissance $n$
<b>ROUND (<math>m, n</math>)</b>	Arrondi à une ou plusieurs décimales
<b>SIGN (<math>n</math>)</b>	Retourne le signe d'un nombre
<b>SQRT</b>	Racine carré
<b>SIN</b>	Sinus

# Fonctions chaînes de caractères(1)



SQL

CHR	Retourne le caractère ASCII équivalent
CONCAT	Concatène
INITCAP	Premières lettres de chaque mot en majuscule
LOWER	Tout en minuscules
LPAD( <i>c1, n, c2</i> )	Cadrage à gauche de <i>c2</i> dans <i>c1</i> et affichage de <i>n</i> caractères
LTRIM( <i>c1, c2</i> )	Enlève <i>c2</i> à <i>c1</i> par la gauche
RTRIM( <i>c1, c2</i> )	Enlève <i>c2</i> à <i>c1</i> par la droite
TRIM( <i>c1, c2</i> )	Enlève <i>c2</i> à <i>c1</i> des 2 côtés

# Fonctions chaînes de caractères(2)



<b>REPLACE (c1 ,c2 ,c3)</b>	Remplace c2 par c3 dans c1
<b>LPAD (c1 ,n ,c2)</b>	Cadrage à gauche de c2 dans c1
<b>SOUNDEX</b>	Compare des chaînes phonétiquement ( <i>english only...</i> )
<b>SUBSTR(c ,d ,l)</b>	Extraction d'une sous-chaîne dans c à partir de d et d'une longueur de l
<b>UPPER</b>	Tout en majuscules
<b>TO_CHAR (entier)</b>	Transforme en chaîne
<b>TO_NUMBER (chaîne)</b>	Transforme en entier

# Fonctions pour les dates



<b>ADD_MONTHS (<i>d, n</i>)</b>	Ajoute <i>n</i> mois à la date <i>d</i>
<b>LAST_DAY (<i>d</i>)</b>	Retourne le dernier jour du mois
<b>MONTHS_BETWEEN (<i>d1, d2</i>)</b>	Retourne le nombre de mois entre 2 dates
<b>NEXT_DAY (<i>d, chaîne</i>)</b>	Retourne le prochain jour ouvrable
<b>ROUND (<i>FonctionDate</i>)</b>	Arrondie une date
<b>SYSDATE</b>	Date courante
<b>TRUNC (<i>FonctionDate</i>)</b>	Tronque une date
<b>TO_DATE (<i>chaîne, format</i>)</b>	Exemple : 'DD-MM-YYYY'

# Mise en forme des résultats



- **Principe**
  - Amélioration de la présentation du résultat d'une requête SQL
- **Formatage de colonnes**
  - Instruction COLUMN pour afficher des entêtes
- **Formatage de pages**
  - Instruction BREAK pour gérer les ruptures
  - Instruction COMPUTE avec les fonctions SUM, MIN, MAX, AVG, STD, VAR, COUNT, NUM pour des calculs
  - Directive SET avec les paramètres NEWPAGE, PAGESIZE, LINESIZE pour la mise en page des états de sortie
  - Instructions TTITLE, BTITLE pour l'affichage des titres des états de sortie
- **Ordres SQL\*PLUS**

# Affichage avec COLUMN



SQL

```
COLUMN nom_colonne [HEADING texte] -  
[JUSTIFY {LEFT | RIGHT | CENTER}] -  
[NOPRINT | PRINT] [FORMAT An]
```

```
COLUMN nome FORMAT A(30) -  
HEADING 'Nom des employés' -  
JUSTIFY CENTER
```

```
COLUMN adresse NOPRINT
```

```
CLEAR COLUMN
```

# Rupture de séquences : BREAK



SQL

```
BREAK ON nom_colonne SKIP n
```

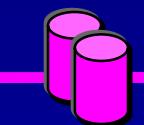
Rupture sur la colonne

Nul pour les valeurs dupliquées

Saut de n lignes à chaque rupture

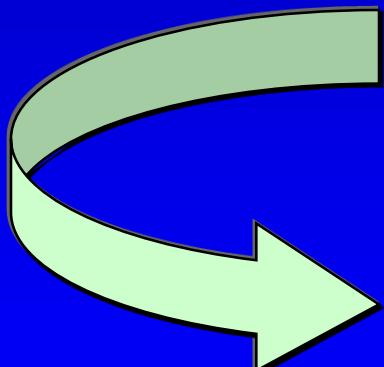
```
CLEAR BREAK
```

# Exemple de BREAK



SQL

```
BREAK ON specialite SKIP 2  
SELECT specialite,  
nom_chercheur  
AS "Nom" FROM chercheur  
ORDER BY 1,2;  
CLEAR BREAK
```



SPECIALITE	Nom
bd	michel
oo	christian claude gilles
rx	daniel françois
si	jacques jean

# Opérations de calculs



SQL

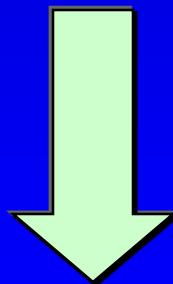
- Utilisation conjointe de **BREAK** et **COMPUTE** pour programmer des calculs en fonction de ruptures
- Syntaxe
  - **COMPUTE *fonction LABEL* 'label' OF *colonne* ON {*colonne* / ROW | REPORT}**
- ***fonction* : SUM, MIN, MAX, AVG, STD, VAR, COUNT, NUM**
  - **ROW** : le calcul est exécuté pour chaque ligne
  - **REPORT** : le calcul est exécuté à la fin du rapport
  - **CLEAR COMPUTE** efface les COMPUTES

# Exemple de COMPUTE



SQL

```
BREAK ON "Numéro Client "
compute sum label "Solde Client"      -
of SOLDE_COMPTE on "Numéro Client"
select num_client as "Numéro Client" ,
       libelle_compte,solde_compte from comptes
order by 1;
CLEAR BREAK
CLEAR COMPUTE
```



# Résultat de la requête



SQL

A large, stylized, three-dimensional text representation of the word "SQL" in pink and purple, with a black arrow pointing towards it from the left.

Numéro Client	LIBELLE_COMPTE	SOLDE_COMPTE
-----	-----	-----
1	compte courant Tuffery	1200
	compte épargne Tuffery	5800
*****	*****	-----
Solde Client		7000
2	compte courant Nonne	-250
	livret a Nonne	440
*****	*****	-----
Solde Client		190
3	compte courant Crampes	2000
	livret a Crampes	500
	compte épargne Crampes	750
*****	*****	-----
Solde Client		3250

# État de sortie



SQL

## Entêtes de page

```
TTITLE option {texte | variable}  
option : COL n, SKIP n, TAB n,  
LEFT, CENTER, RIGHT, BOLD FORMAT...
```

## Variables

SQL.LNO numéro de la ligne courante  
SQL.PNO numéro de la ligne courante  
SQL.RELEASE version d'Oracle  
SQL.USER nom du user

## Pieds de page

```
BTITLE option {texte | variable}
```

# Exemples d'état de sortie



SQL

```
SET NEWPAGE 1
SET PAGESIZE 30
SET LINESIZE 80
COL JOUR NOPRINT NEW_VALUE AUJOURDHUI
TTITLE LEFT 'PAGE' SQL.PNO SKIP LEFT AUJOURDHUI -
SKIP CENTER 'Comptes' SKIP CENTER 'par client' SKIP2
BTITLE CENTER 'Fin du rapport'
BREAK ON "Numéro Client"
COMPUTE SUM LABEL "Solde Compte" -
OF solde_compte ON "Numéro Client"
SELECT TO_CHAR(sysdate,('DD/MM/YYYY')) AS jour,
      num_client AS "Numéro Client" ,
      libelle_compte,solde_compte FROM comptes
ORDER BY 1;
TTITLE OFF
BTITLE OFF
CLEAR BREAKS
CLEAR COMPUTES
CLEAR COLUMNS
```

# Exemples d'état de sortie (suite)



SQL

PAGE

1

14/10/2003

## Comptes par client

Numéro Client	LIBELLE_COMPTE	SOLDE_COMPTE
1	compte courant Tuffery	1200
	compte épargne Tuffery	5800
*****		
Solde Compte		7000
2	compte courant Soutou	-250
	livret a Soutou	440
*****		
Solde Compte		190
3	compte courant Teste	2000
	livret a Teste	500
	compte épargne Teste	750
*****		
Solde Compte		3250

Fin du rapport

# Ordres d'entrée – sortie

## Variables paramétrées



SQL

Appuyer sur Entrée pour continuer

PAUSE 'Texte affiché avec arrêt'

PROMPT 'Texte affiché sans arrêt'

ACCEPT variable [NUMBER | CHAR] -  
[PROMPT [texte] | NOPROMPT] [HIDE]

```
SQL> ACCEPT nequipelue PROMPT 'Entrer le n°équipe : ' HIDE  
Entrer le n°équipe : **  
SELECT * FROM CHERCHEUR WHERE n_equipe='&nequipelue';  
N_CHERCHEUR NOM_CHERCH SPECIALITE UNIVERSITE N_EQUIPE  
----- ----- ----- -----  
c1      michel    bd          2 e1  
c3      claude    oo          3 e1  
c7      christian oo          2 e1
```



# Variables d'environnement : ordres SET variable valeur

**SQL**

TERMOUT { ON   OFF }	Affichage
ECHO { ON   OFF }	affichage des commandes
FEEDBACK <i>n</i>	écho du nombre de lignes traitées
SHOWMODE { ON   OFF }	affichage des paramètres
HEADING { ON   OFF }	affichage des titres
PAUSE { ON   OFF   <i>texte</i> }	arrêt en fin de page
SQLN { ON   OFF }	numéro de ligne du buffer
VERIFY { ON   OFF }	affichage des substitutions

Show all → variables courantes

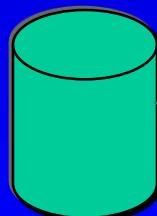
# Gestion des fichiers SPOOL

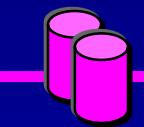


```
SQL> SPOOL fichier  
SQL> -- ordres sql  
SQL>  
SQL>  
SQL> SPOOL OFF
```

trace écran

fichier.lst





SQL

Le Langage d'Interrogation  
de Données

LID  
Multi-table  
Jointures

# Les différentes méthodes de Jointure



SQL

- Opérateurs ensemblistes
  - UNION, INTERSECT et MINUS
- Jointure en forme procédurale déconnectée
  - SELECT imbriqués déconnectés
- Jointure en forme procédurale synchronisée
  - SELECT imbriqués synchronisés
- Jointure en forme non procédurale ou relationnelle
  - Pas d'imbrication : 1 seul bloc SELECT-FROM-WHERE
  - Jointure complète ou externe (gauche ou droite)
- Requêtes hiérarchiques
  - Parcours d'arbre : association réflexive 1-N

# L'opérateur ensembliste UNION



SQL

```
SELECT liste de colonnes FROM table1  
UNION  
SELECT liste de colonnes FROM table2;
```

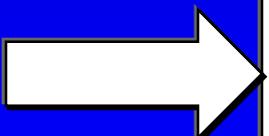
```
SELECT idPro FROM produits_fabriques  
UNION  
SELECT codePro FROM produits_achetes;
```

Même nombre de colonnes

Mêmes types de colonnes

Tri (ORDER BY) sur le n° de colonne

Le DISTINCT est automatique



# L'opérateur ensembliste INTERSECT



SQL

```
SELECT liste de colonnes FROM table1  
INTERSECT  
SELECT liste de colonnes FROM table2;
```

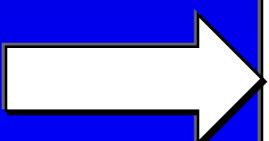
```
SELECT idPro FROM produits_fabriques  
INTERSECT  
SELECT codePro FROM produits_achetes;
```

Même nombre de colonnes

Mêmes types de colonnes

Tri (ORDER BY) sur le n° de colonne

Le DISTINCT est automatique



# L'opérateur ensembliste MINUS



SQL

```
SELECT liste de colonnes FROM table1  
MINUS  
SELECT liste de colonnes FROM table2;
```

```
SELECT idService FROM service  
MINUS  
SELECT idService FROM employe;
```



Attention à l'ordre des SELECT

# Composition d'opérateurs ensemblistes



```
SELECT liste de colonnes FROM table1  
INTERSECT  
→(SELECT liste de colonnes FROM table2  
MINUS  
SELECT liste de colonnes FROM table3);
```



→ Attention à l'ordre d'exécution → parenthèses

# Jointure forme procédurale déconnectée imbrication de SELECT



SQL

```
SELECT liste de colonnes (*) FROM table1
WHERE col1 [NOT] [IN] [<,>,!=,=,...] [ANY|ALL]
      (SELECT col2 FROM table2
       [WHERE prédicat ]);
```

- Col1 et col2 sont les colonnes de jointure
- [NOT] IN : si le sous-select peut ramener plusieurs lignes
- Ne sélectionne que des colonnes de la première table (\*)
- Exécute d'abord le sous-select puis 'remonte' en respectant le graphe relationnel

# Jointure forme procédurale déconnectée

## Exemple 1 : ouvrages empruntés



SQL

```
SELECT titre,auteur FROM ouvrage  
WHERE idOuvrage IN  
  (SELECT idOuvrage FROM emprunt  
   WHERE dateEmprunt = SYSDATE  
   AND idEtudiant IN  
     (SELECT idEtudiant FROM etudiant  
      WHERE ufr='Informatique' )) ;
```

- Du select le plus imbriqué vers le select le plus haut
- Opérateur 'IN' obligatoire

# Jointure forme procédurale déconnectée

## Exemple 2 : employés les mieux payés



SQL

```
SELECT nom, fonction FROM emp  
WHERE salaire =  
    (SELECT MAX(salaire) FROM emp)
```



- Le select imbriqué ne ramène qu'une valeur
- Signe ‘=’ autorisé
- Cette forme est obligatoire pour obtenir ce résultat

# Jointure forme procédurale déconnectée

## Exemple 3 : employé responsable



SQL

```
SELECT nom, indice, salaire FROM emp  
WHERE idEmp =  
    (SELECT idResp FROM emp  
     WHERE idEmp=&idlu) ;
```

- Signe '=' autorisé
- Jointure réflexive

# Jointure forme procédurale déconnectée Avantages et inconvénients



SQL

- Parfois obligatoire (exemple 2)
- Mise en œuvre naturelle du graphe de requêtes
- Affichage final : uniquement les colonnes de la première table
- Écriture lourde si le niveau d'imbrication est important : chemin d'accès à la charge du développeur
- Jointures
  - Équi-jointures avec = ou IN
  - Autres jointures (>,<, ....)
- ANY : au moins un élément ( $=\text{ANY} \Leftrightarrow \text{IN}$ )
- ALL : tous les éléments ( $!=\text{ALL} \Leftrightarrow \text{NOT IN}$ )

# Jointure forme procédurale déconnectée

## Exemple 4 : ALL



SQL

```
SELECT nom, indice, salaire FROM emp  
WHERE salaire > ALL  
    (SELECT salaire FROM emp  
     WHERE idService = 10) ;
```

- Employés gagnant plus que tous les employés du service 10 ?
- Requête plus ‘naturelle’ avec la fonction MAX ...

# Jointure en forme procédurale synchronisée

## Présentation



SQL

- Contrairement à la forme précédente, le sous-select est ‘synchronisé’ avec l’ordre du dessus (select ou update)
- Les deux ordres (dessus et dessous) fonctionnent comme deux boucles imbriquées
- Pour 1 pas de la boucle supérieure → n pas de la boucle inférieure



```
SELECT ..... FROM table t1  
WHERE .....  
  
(SELECT ..... FROM table t2  
WHERE t1.col1 = t2.col2);
```

# Jointure en forme procédurale synchronisée

## Exemple 1



SQL

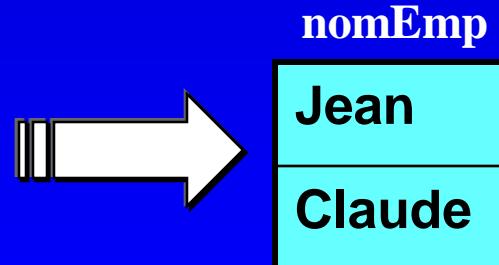
- Employés ne travaillant pas dans le même service que leur responsable ?

### EMPLOYEE

idEmp nomEmp idSer idResp

idEmp	nomEmp	idSer	idResp
100	Jean	50	200
200	Sylvie	25	
300	Xavier	25	200
400	Claude	30	200
500	Thomas	25	200

```
SELECT e.nomEmp FROM emp e  
WHERE e.idSer !=  
(SELECT r.idSer FROM emp r  
WHERE r.idEmp=e.idResp)  
AND e.idResp IS NOT NULL;
```



# Jointure en forme procédurale synchronisée

## Exemple 2 : UPDATE automatique



SQL

- Mise à jour automatique du nombre d'employés (nbEmp) de la table Service ?

SERVICE

idSer nomSer nbEmp

25	Info	3
30	Stat	1
50	Maths	1
60	Bd	0

```
UPDATE service s
SET s.nbEmp =
(SELECT COUNT(e.idEmp) FROM emp e
WHERE e.idSer=s.idSer);
```



# Jointure en forme procédurale synchronisée

## Test d'existence : [NOT] EXISTS



SQL

- Le prédictat ‘EXISTS’ est ‘vrai’ si le sous-select ramène au moins une ligne
- NOT EXISTS → ensemble vide
- Permet de mettre en œuvre l’opérateur relationnel de DIVISION
- Principe :

```
SELECT colonnes résultats FROM table1 t1
WHERE [NOT] EXISTS
    (SELECT {col | valeur} FROM table2 t2
     WHERE t1.col1=t2.col2);
```

# Jointure en forme procédurale synchronisée

## Exemple 3 : Test d'existence



SQL

- Les employés (nomEmp) travaillant dans un service contenant au moins un 'programmeur' ?

EMPLOYEE  
idEmp nomEmp idSer fonction

100	Jean	50	concepteur
200	Sylvie	25	analyste
300	Xavier	25	concepteur
400	Claude	30	analyste
500	Thomas	25	programmeur

```
SELECT e1.nomEmp FROM emp e1  
WHERE EXISTS  
(SELECT 'trouvé' FROM emp e2  
WHERE e1.idSer=e2.idSer  
AND  
e2.fonction='programmeur' );
```



nomEmp
Sylvie
Xavier
Thomas

# Jointure en forme procédurale synchronisée

## Mise en œuvre de la DIVISION



SQL

- DIVISION : un ensemble A est entièrement compris dans un autre (B)
- Un ensemble (A) est entièrement compris dans un autre (B) si :
  - $A - B = \emptyset$  ou NOT EXISTS
- Égalité parfaite de 2 ensembles (A et B) si :
  - $\left\{ \begin{array}{l} A - B = \emptyset \\ \text{et} \\ B - A = \emptyset \end{array} \right.$

# Jointure en forme procédurale synchronisée

## Exemple 4 : DIVISION



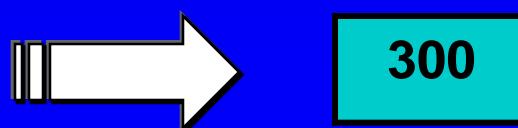
SQL

- Étudiants (idEtu) inscrits aux mêmes UV que l'étudiant '100' ?

COURS  
idEtu    UV

idEtu	UV
100	UV1
200	UV1
100	UV3
300	UV1
200	UV4
300	UV2
300	UV3

```
SELECT c1.idEtu FROM cours c1
WHERE NOT EXISTS
  (SELECT c100.UV FROM cours c100
   WHERE c100.idEtu=100
   MINUS
   SELECT c2.UV FROM cours c2
   WHERE c2.idEtu=c1.idEtu)
AND c1.idEtu <> 100;
```



# Jointure en forme Relationnelle



**SQL**

- Pas d'imbrication, colonnes résultats de toutes les tables intervenant
- Le chemin d'accès (graphe) est à la charge du SGBD
- Un seul bloc **SELECT – FROM – WHERE**
- Présentation d'une **EQUI-JOINTURE** :

```
SELECT colonnes résultats (des 3 tables)
FROM table1 t1, table2 t2, table3 t3
WHERE t1.col1=t2.col2 AND t2.col2=t3.col3;
```

**N tables → N-1 Jointures**

# Jointure en forme Relationnelle

## Exemple 1 : ouvrages empruntés



SQL

- Reprise du 1er exemple de la forme procédurale

```
BREAK ON nom
SELECT et.nom,o.titre,o.auteur
FROM ouvrage o, emprunt ep, etudiant et
WHERE et.ufr='Informatique' AND
et.idEtudiant=ep.idEtudiant AND
ep.dateEmprunt = SYSDATE AND
ep.idOuvrage=o.idOuvrage
ORDER BY 1,2;
CLEAR BREAK
```

Colonnes de tables différentes : impossible en procédural

# Jointure en forme Relationnelle

## Exemple 2 : employé responsable



SQL

- Nom de l'employé responsable d'un employé ?

```
SET HEADING OFF
ACCEPT idlu PROMPT 'Entrer le n° employé: '
SELECT er.nom || 'Responsable de' || ee.nom
FROM emp er, emp ee
WHERE er.idEmp = ee.idResp
  AND ee.idEmp = &idlu;
SET HEADING ON
```

# Jointure en forme Relationnelle

## Exemple 3 : autre jointure



SQL

- Autre jointure que l' EQUI-JOINTURE
- Employés gagnant plus que 'Michel' ?

```
COLUMN n HEADING 'Nom employé'  
COLUMN s1 HEADING 'Son salaire'  
COLUMN s2 HEADING 'Salaire de Michel'  
SELECT e.nom as n,e.salaire as s1,  
      m.salaire as s2  
FROM emp e, emp m  
WHERE m.nom='Michel'  
      AND e.salaire > m.salaire;  
CLEAR COLUMNS
```

# Jointure en forme Relationnelle

## Jointure Externe ou Complète



SQL

- Lignes de l'EQUI-JOINTURE plus les lignes n'ayant pas de correspondantes (+)
- Équivalent du LEFT JOIN de certains SGBD, OUTER JOIN pour Oracle

```
SELECT s.nomSer AS 'Nom du Service',  
       e.nomEmp AS 'Employés'  
FROM emp e, service s  
WHERE s.idSer = e.idSer(+);
```

Nom du Service    Employés

Info	Michel
Info	jean
Stat	xavier
Bd	

Aucun employé  
(valeur nulle)

# Jointure Externe ou Complète

## Exemple d'utilisation



SQL

- Services n'ayant pas d'employés ?

```
SELECT s.nomSer AS 'Nom du Service'  
FROM emp e, service s  
WHERE s.idSer = e.idSer(+)  
AND e.idEmp IS NULL;
```

# Requêtes hiérarchiques



- Extraction de données provenant d'un parcours d'arbre
- Association de type Hiérarchique (mère-fille) réflexif à plusieurs niveaux
- Syntaxe :

```
SELECT [LEVEL], colonne, expression...  
FROM table  
[WHERE condition(s)]  
[START WITH condition(s)]  
[CONNECT BY PRIOR condition(s)];
```



# Requêtes hiérarchiques

## Syntaxe d'une requête

- Les clauses **CONNECT BY** et **START WITH** identifient les requêtes hiérarchiques
- La pseudo-colonne **LEVEL** indique le numéro de niveau (1 pour le nœud racine, 2 représente un enfant de la racine...)
- **FROM table** : sélection à partir d'une seule table
- **START WITH** spécifie les lignes racine de la hiérarchie ou point de départ (clause obligatoire)
- **CONNECT BY** indique les colonnes où il existe une relation entre des lignes parent et enfant
- **PRIOR** indique la direction du parcours de l'arbre, permet également d'éliminer certaines branches de l'arborescence
- L'ordre **SELECT** ne peut pas contenir de jointure ou être basé sur une vue issue d'une jointure
- Remarque : Il manque un ordre de fin de parcours d'arbre



# Requêtes hiérarchiques exemple : table scott.emp

**SQL**

```
SELECT empno AS "Employé",  
ename AS "Nom", job AS "Travail",  
mgr AS "Responsable" FROM emp;
```

Employé	Nom	Travail	Responsable
7369	SMITH	CLERK	7902
7499	ALLEN	SALESMAN	7698
7521	WARD	SALESMAN	7698
7566	JONES	MANAGER	7839
7654	MARTIN	SALESMAN	7698
7698	BLAKE	MANAGER	7839
7782	CLARK	MANAGER	7839
7788	SCOTT	ANALYST	7566
7839	KING	PRESIDENT	
7844	TURNER	SALESMAN	7698
7876	ADAMS	CLERK	7788
7900	JAMES	CLERK	7698
7902	FORD	ANALYST	7566
7934	MILLER	CLERK	7782

14 ligne(s) sélectionnée(s).

# Requêtes hiérarchiques mise en oeuvre



SQL

- Parcours de l'arbre (`CONNECT BY PRIOR`)
  - Bas vers le haut : `clé père = clé fils`
  - Haut vers le bas : `clé fils = clé père`
- Point de départ (`START WITH`)
  - Prédicat simple (`START WITH ename = 'BLAKE'`)
  - Prédicat contenant une sous-interrogation :  
`(START WITH empno =  
(SELECT empno FROM emp  
WHERE ename = 'Michel'))`

# Requêtes hiérarchiques exemple (1)

```
SELECT empno,ename,job,mgr
  FROM emp
START WITH empno=7654
CONNECT BY PRIOR mgr = empno;
```

Départ

Du bas vers le haut

EMPNO	ENAME	JOB	MGR
7654	MARTIN	SALESMAN	7698
7698	BLAKE	MANAGER	7839
7839	KING	PRESIDENT	

# Utilisation de la fonction LPAD



SQL

```
SELECT LPAD('Texte à droite',30,'*-')  
AS "Ex. avec 30 car. affichés" FROM dual;
```

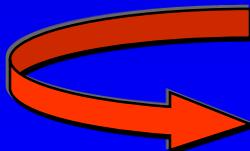


Ex. avec 30 car. affichés

-----

\*-\*-\*-\*-\*-\*-\*-\*-\*Texte à droite

```
SELECT LPAD(' ',3*2) || 'Indentation  
de 6' AS "ex. de LPAD" FROM DUAL;
```



ex. de LPAD

-----

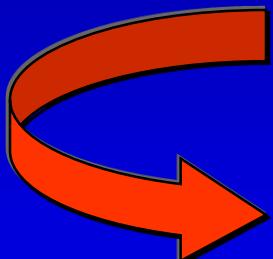
Indentation de 6

# Requêtes hiérarchiques exemple final



SQL

```
COLUMN organisation FORMAT A15 HEADING "Organigramme"
SELECT LPAD(' ', 3 * LEVEL-3) || ename AS organisation,
       LEVEL, empno FROM   emp
      CONNECT BY PRIOR empno = mgr
      START WITH mgr is NULL;
```



Organigramme	LEVEL	EMPNO
KING	1	7839
JONES	2	7566
SCOTT	3	7788
ADAMS	4	7876
FORD	3	7902
SMITH	4	7369
BLAKE	2	7698
ALLEN	3	7499
WARD	3	7521
MARTIN	3	7654
TURNER	3	7844
JAMES	3	7900
CLARK	2	7782
MILLER	3	7934

# 7 Façons de résoudre une requête .....



SQL

- Étudiants (nom) ayant emprunté un ouvrage particulier (&ouvlu) ?

```
1   SELECT e.nom  
      FROM etudiant e  
     WHERE e.idEtu IN  
           (SELECT o.idEtu FROM emprunt ep  
              WHERE ep.idOuv = '&ouvlu' );
```

```
2   SELECT e.nom  
      FROM etudiant e  
     WHERE e.idEtu = ANY  
           (SELECT o.idEtu FROM emprunt ep  
              WHERE ep.idOuv = '&ouvlu' );
```

## 7 Façons de résoudre une requête (suite)



SQL

3

```
SELECT e.nom  
FROM etudiant e  
WHERE EXISTS  
    (SELECT 'trouvé' FROM emprunt ep  
     WHERE ep.idOuv = '&ouvlu'  
      AND e.idEtu = ep.idEtu);
```

4

```
SELECT e.nom  
FROM etudiant e, emprunt ep  
WHERE ep.idOuv = '&ouvlu'  
  AND e.idEtu = ep.idEtu;
```

## 7 Façons de résoudre une requête (suite)



SQL

5

```
SELECT e.nom  
FROM etudiant e  
WHERE 0 <  
      (SELECT COUNT(*) FROM emprunt ep  
       WHERE ep.idOuv = '&ouvlu'  
         AND e.idEtu = ep.idEtu);
```

6

```
SELECT e.nom  
FROM etudiant e  
WHERE '&ouvlu' IN  
      (SELECT ep.idOuv FROM emprunt ep  
       WHERE e.idEtu = ep.idEtu);
```

# 7 Façons de résoudre une requête (fin)

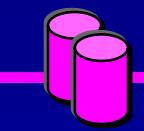


SQL

7

```
SELECT e.nom  
FROM etudiant e  
WHERE '&ouvlu' = ANY  
  (SELECT ep.idOuv FROM emprunt ep  
  WHERE e.idEtu = ep.idEtu);
```





**SQL**

# Optimisation des Requêtes

Plan d'exécution  
**EXPLAIN PLAN**



# Optimisation des requêtes : Visualisation du graphe

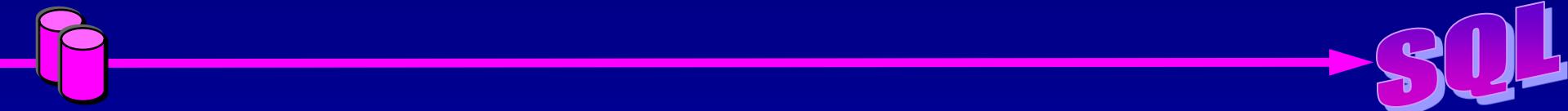
- Oracle conserve la trace du chemin d'accès d'une requête dans une table appelée « plan\_table »
- Tous les graphes des requêtes du LID et LMD sont conservées :
  - SELECT, UPDATE , INSERT et DELETE
- Chaque opération sur un objet est notée avec :
  - L'ordre d'exécution
  - Le parent (arbre relationnel ou algébrique)
  - L'objet consulté (table, index, cluster, view, ...)
  - Le type d'opération

# Structure de PLAN\_TABLE



STATEMENT_ID	VARCHAR2 (30)	id choisi
TIMESTAMP	DATE	date d'exécution
REMARKS	VARCHAR2 (80)	
OPERATION	VARCHAR2 (30)	opération (plus loin)
OPTIONS	VARCHAR2 (30)	option de l'opération
OBJECT_NODE	VARCHAR2 (128)	pour le réparti
OBJECT_OWNER	VARCHAR2 (30)	propriétaire
OBJECT_NAME	VARCHAR2 (30)	nom objet (table, index, ...)
OBJECT_INSTANCE	NUMBER (38)	(unique, ...)
OBJECT_TYPE	VARCHAR2 (30)	
OPTIMIZER	VARCHAR2 (255)	
SEARCH_COLUMNS	NUMBER	
ID	NUMBER (38)	n° nœud courant
PARENT_ID	NUMBER (38)	n° nœud parent
POSITION	NUMBER (38)	1 ou 2 (gauche ou droite)
COST	NUMBER (38)	
CARDINALITY	NUMBER (38)	
BYTES	NUMBER (38)	
OTHER_TAG	VARCHAR2 (255)	
PARTITION_START	VARCHAR2 (255)	
PARTITION_STOP	VARCHAR2 (255)	
PARTITION_ID	NUMBER (38)	
OTHER	LONG	

# Opérations / Options (1)



Operation	Option	Description
<b>AND-EQUAL</b>		Operation accepting multiple sets of rowids, returning the intersection of the sets, eliminating duplicates. Used for the single-column indexes access path.
	<b>CONVERSION</b>	TO ROWIDS converts bitmap representations to actual rowids that can be used to access the table. FROM ROWIDS converts the rowids to a bitmap representation COUNT returns the number of rowids if the actual values are not needed
	<b>INDEX</b>	SINGLE VALUE looks up the bitmap for a single key value in the index. RANGE SCAN retrieves bitmaps for a key value range. FULL SCAN performs a full scan of a bitmap index if there is no start or stop key.
	<b>MERGE</b>	Merges several bitmaps resulting from a range scan into one bitmap
	<b>MINUS</b>	Subtracts bits of one bitmap from another. Row source is used for negated predicates. Can be used only if there are nonnegated predicates yielding a bitmap from which the subtraction can take place
	<b>OR</b>	Computes the bitwise OR of two bitmaps
<b>CONNECT BY</b>		Retrieves rows in hierarchical order for a query containing a CONNECT BY clause

# Opérations / Options (2)



Operation	Option	Description
CONCATENATION		Operation accepting multiple sets of rows returning the union-all of the sets
COUNT		Operation counting the number of rows selected from a table
FILTER		Operation accepting a set of rows, eliminates some of them, and returns the rest
FIRST ROW		Retrieval on only the first row selected by a query
FOR UPDATE		Operation retrieving and locking the rows selected by a query containing a FOR UPDATE clause
HASH JOIN		Operation joining two sets of rows and returning the result
	SEMI	Hash semi-join

# Opérations / Options (3)



Operation	Option	Description
INDEX	UNIQUE SCAN	Retrieval of a single rowid from an index
	RANGE SCAN	Retrieval of one or more rowids from an index. Indexed values are scanned in ascending order
	RANGE SCAN DESCENDING	Retrieval of one or more rowids from an index. Indexed values are scanned in descending order
INLIST ITERATOR		Iterates over the operation below it for each value in the IN-list predicate
INTERSECTION		Operation accepting two sets of rows and returning the intersection of the sets, eliminating duplicates
MERGE JOIN		Operation accepting two sets of rows, each sorted by a specific value, combining each row from one set with the matching rows from the other, and returning the result
	OUTER	Merge join operation to perform an outer join statement

# Opérations / Options (4)



SQL

Operation	Option	Description
<b>MINUS</b>		Operation accepting two sets of rows and returning rows appearing in the first set but not in the second, eliminating duplicates
<b>NESTED LOOPS</b> (These are join operations)		Operation accepting two sets of rows, an outer set and an inner set. Oracle compares each row of the outer set with each row of the inner set, returning rows that satisfy a condition.
<b>SORT</b>	<b>AGGREGATE</b>	Retrieval of a single row that is the result of applying a group function to a group of selected rows
	<b>UNIQUE</b>	Operation sorting a set of rows to eliminate duplicates
	<b>GROUP BY</b>	Operation sorting a set of rows into groups for a query with a GROUP BY clause
	<b>JOIN</b>	Operation sorting a set of rows before a merge-join
	<b>ORDER BY</b>	Operation sorting a set of rows for a query with an ORDER BY clause

# Opérations / Options (5)



SQL

Operation	Option	Description
TABLE ACCESS	FULL	Retrieval of all rows from a table
	CLUSTER	Retrieval of rows from a table based on a value of an indexed cluster key
	HASH	Retrieval of rows from table based on hash cluster key value
	BY ROWID	Retrieval of a row from a table based on its rowid
	BY INDEX ROWID	If the table is nonpartitioned and rows are located using index(es)
UNION		Operation accepting two sets of rows and returns the union of the sets, eliminating duplicates
VIEW		Operation performing a view's query and then returning the resulting rows to another operation

# Visualisation du graphe

## Mise en œuvre (1)



SQL

- Calcul du plan d'exécution → remplissage dans PLAN\_TABLE

```
explain plan
set statement_id='r1'
for
select * from comptes
where num_client IN
    (select num_client from clients
     where nom_client = 'Tuffery');
column operation format a18
column options format a15
column object_name heading OBJET format      a13
column id format 99
column parent_id format 99
column position format 99
select operation,options,object_name,id,parent_id,position
from plan_table
where statement_id='r1'
order by id;
```

# Visualisation du graphe Mise en œuvre (2)

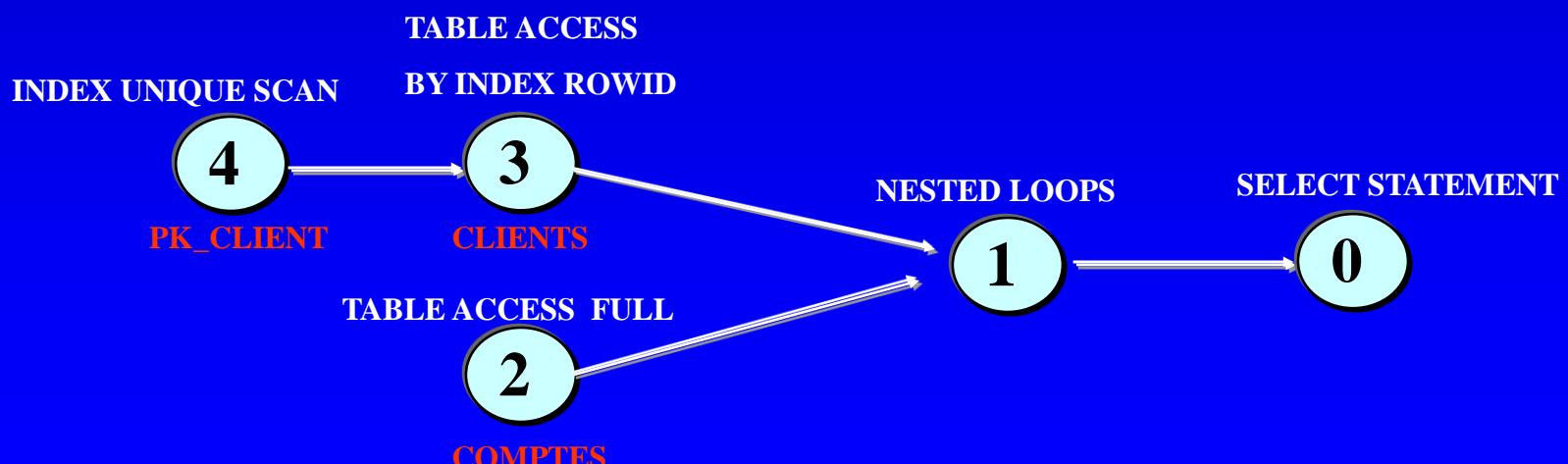


SQL

- Contenu de PLAN\_TABLE

OPERATION	OPTIONS	OBJET	ID	PARENT_ID	POSITION
SELECT STATEMENT			0		
NESTED LOOPS			1	0	1
TABLE ACCESS	FULL	COMPTE\$	2	1	1
TABLE ACCESS	BY INDEX ROWID	CLIENTS	3	1	2
INDEX	UNIQUE SCAN	PK_CLIENT	4	3	1

- Graphe correspondant



# Visualisation du graphe autres exemples (1)



SQL

```
select * from clients
where num_client NOT IN
    (select num_client from comptes);
```



OPERATION	OPTIONS	OBJET	ID	PARENT_ID	POSITION
SELECT STATEMENT			0		
FILTER			1	0	1
TABLE ACCESS	FULL	CLIENTS	2	1	1
TABLE ACCESS	FULL	COMPTEs	3	1	2

# Visualisation du graphe autres exemples (2)



SQL

```
select * from clients
where num_client IN
(select num_client from clients
minus
select num_client from comptes);
```



OPERATION	OPTIONS	OBJET	ID	PARENT_ID	POSITION
SELECT STATEMENT			0		
NESTED LOOPS			1	0	1
VIEW		VW_NS0_1	2	1	1
MINUS			3	2	1
SORT	UNIQUE		4	3	1
TABLE ACCESS	FULL	CLIENTS	5	4	1
SORT	UNIQUE		6	3	2
TABLE ACCESS	FULL	COMPTE\$	7	6	1
TABLE ACCESS	BY INDEX ROWID	CLIENTS	8	1	2
INDEX	UNIQUE SCAN	PK_CLIENT	9	8	1

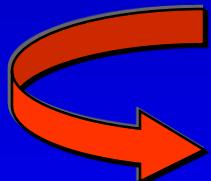
# Visualisation du graphe indentation de plan table



SQL

- Exemple précédent : indentation

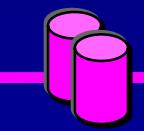
```
select lpad(' ',2*level)||operation||  
options||' ('||object_name||')'  
"plan de la requête"  
from plan_table where statement_id='r3'  
connect by prior id=parent_id  
start with id=0;
```



plan de la requête

---

```
SELECT STATEMENT ()  
NESTED LOOPS ()  
VIEW (VW_NSO_1)  
MINUS ()  
SORTUNIQUE ()  
TABLE ACCESSFULL (CLIENTS)  
SORTUNIQUE ()  
TABLE ACCESSFULL (COMPTE)  
TABLE ACCESSBY INDEX ROWID (CLIENTS)  
INDEXUNIQUE SCAN (PK_CLIENT)
```



**SQL**

# Le Langage de Contrôle de Données

**LCD**

# Contrôle des Données



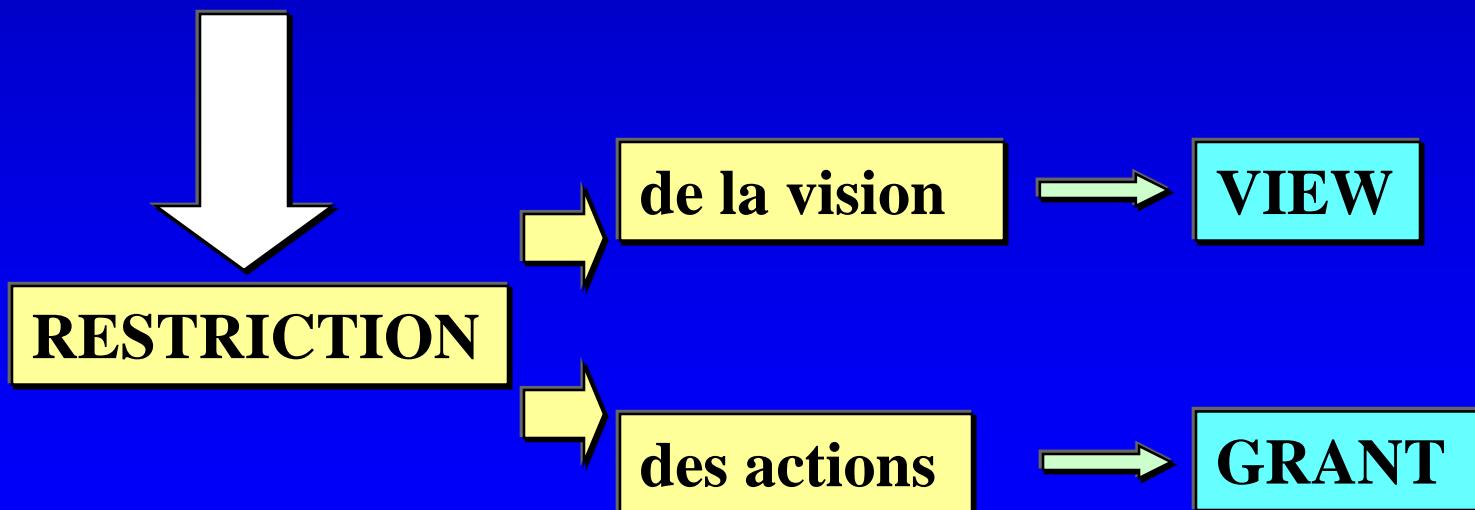
- **Notion de Sous-Schéma**
  - Restriction de la vision
  - Restriction des actions
- **Privilèges**
  - Systèmes
  - Objets
- **Rôles**
  - Regroupement de privilèges
- **Contraintes évènementielles : Trigger (plus loin)**
  - Contrôles avant une modification
  - Mises à jour automatique

# Restreindre les accès à une BD

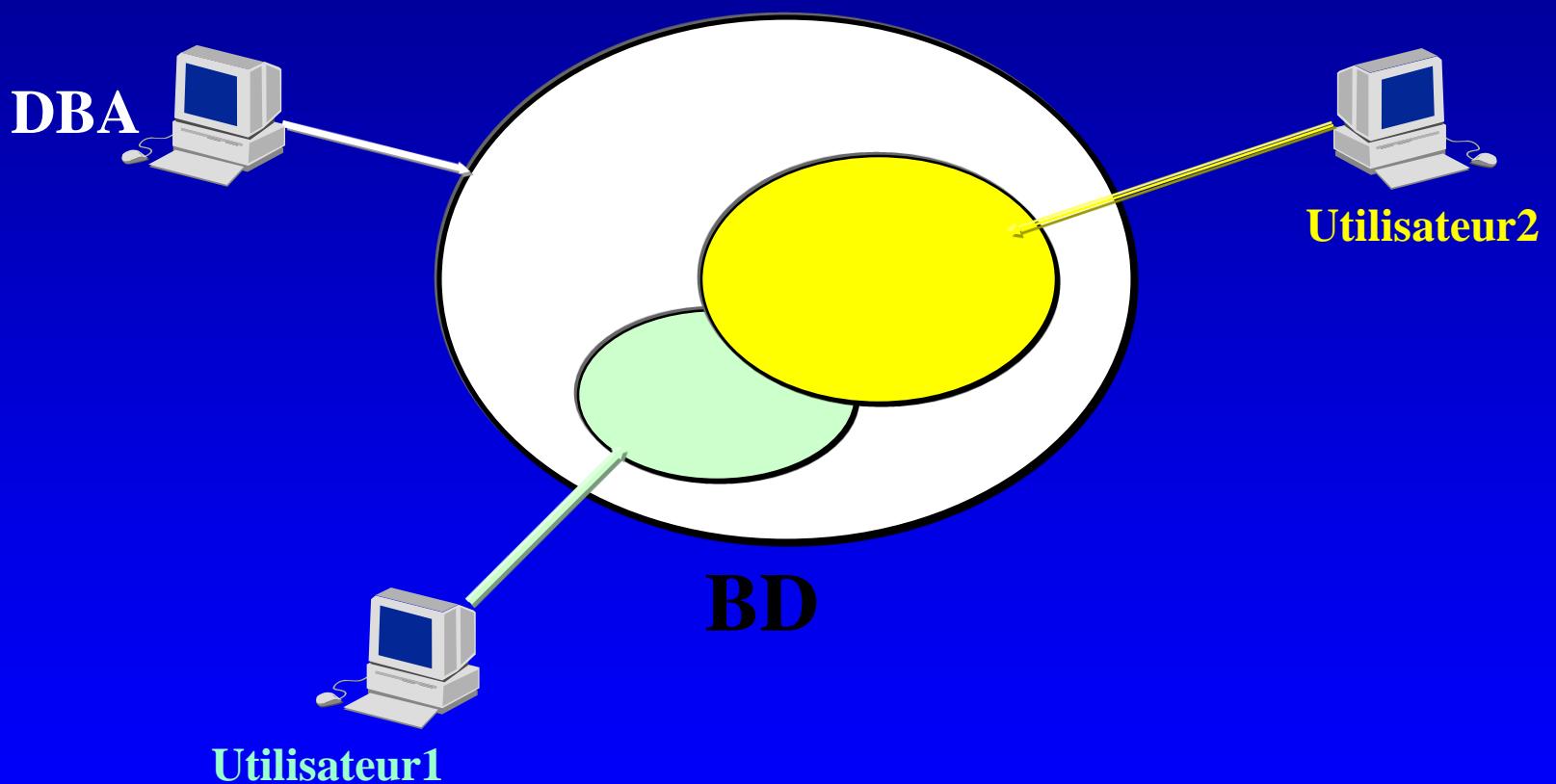


SQL

Tout le monde ne peut pas VOIR  
et FAIRE n'importe quoi



# Restriction des accès Sous-schéma ou schéma externe



# L'objet VUE



SQL

- Une VUE est une **table virtuelle** : aucune implémentation physique de ses données
- La définition de la vue est enregistrée dans le DD
- A chaque appel d'une vue : le SGBD réactive sa construction à partir du DD
- Vue mono-table : crée à partir d'une table
  - Modifications possibles → modifications dans la table
  - Jointure en forme procédurale autorisée
- Vue multi-table
  - Crée par une jointure en forme relationnelle
  - Aucune modification autorisée

# Utilisation d'une VUE



- **Simplification de requêtes pour des non spécialistes**
- **Création de résultats intermédiaires pour des requêtes complexes**
- **Présentation différente de la base de données : schéma externe**
- **Mise en place de la confidentialité (VOIR)**
- **Une vue mono-table pourra être mise à jour avec contraintes**

# Création et suppression d'une VUE



SQL

- Crédit d'une vue

```
CREATE [ ( OR REPLACE ) ] VIEW nom_vue
[ ( liste des colonnes de la vue ) ]
AS
SELECT .....
[WITH CHECK OPTION [CONSTRAINT nom_cont] ] ;
```

Sélection de lignes et  
colonnes

- Suppression d'une vue

```
DROP VIEW nom_vue;
```

- Pas de modification d'une vue

# Exemples de création de VUES (1)



SQL

- Vue mono-table avec restriction horizontale

```
CREATE VIEW enseignant_info AS
SELECT * FROM enseignant ← 1 Table
WHERE idDip IN
    (SELECT idDip FROM diplome
     WHERE UPPER(nomDiplome) LIKE '%INFO%');
```

- Vue mono-table avec restriction verticale

```
CREATE VIEW etudiant_scol AS
SELECT idEtu, nomEtu, adrEtu, idDip FROM etudiant;
```

# Exemples de création de VUES (2)



SQL

- Vue mono-table avec restriction mixte

```
CREATE VIEW etudiant_info
(numEtudiant,nomEtudiant,adrEtudiant,dip) AS
SELECT idEtu,nomEtu,adrEtu,idDip
FROM etudiant
WHERE idDip IN
(SELECT idDip FROM diplome
 WHERE UPPER(nomDiplome) LIKE '%INFO%');
```

A diagram consisting of a white rectangular box containing the text "Nouveaux noms". An upward-pointing arrow originates from the bottom right corner of this box and points to the "idDip" column in the second line of the SQL code.

# Exemples de création de VUES (3)



SQL

- Vue mono-table avec colonnes virtuelles

```
CREATE VIEW employe_salaire  
(ne,nome,mensuel,annuel,journalier) AS  
SELECT idEmp,nomEmp,sal,sal*12,sal/22  
FROM employe;
```

- Pas de modification sur les **colonnes virtuelles**
- Modifications autorisées sur les colonnes de base → mise à jour instantanée !

# Exemples de création de VUES (4)



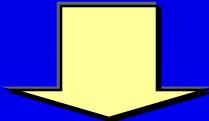
SQL

- Vue mono-table avec groupage

```
CREATE VIEW emp_service  
(ns, nombreEmp, moyenneSal) AS  
SELECT idService, COUNT(*), AVG(sal) FROM employe  
GROUP BY idService;
```

- Utilisation de la vue → reconstruction

```
SELECT * FROM emp_service WHERE nombreEmp > 5;
```



```
SELECT idService AS ns, COUNT(*) AS nombreEmp,  
AVG(sal) AS moyenneSal FROM employe  
GROUP BY ns HAVING COUNT(*) > 5;
```

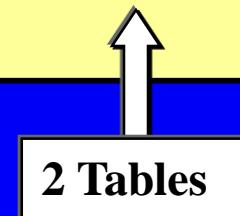
# Vues multi-tables



SQL

- Simplification de requêtes
- Pas de modifications possibles dans ce type de vue ( voir trigger ‘instead of’)
- Tables temporaires ‘virtuelles’ de travail
- Transformation de la présentation des données  
→ Schéma externe

```
CREATE VIEW emp_ser(nom_service, nom_employe)
AS SELECT s.noms,e.nome FROM emp e,service s
WHERE e.idSer=s.idSer;
```



# Exemples de vues multi-tables



SQL

- Reconstitution des clients (UNION)

```
CREATE VIEW clients(idCli,nom,...,secteur) AS
SELECT ct.*,'T' FROM clients_toulouse ct
UNION
SELECT cb.*,'B' FROM clients_bordeaux cb
UNION
SELECT cm.*,'M' FROM clients_montpellier cm;
```

- Reconstitution des étudiants (JOINTURE)

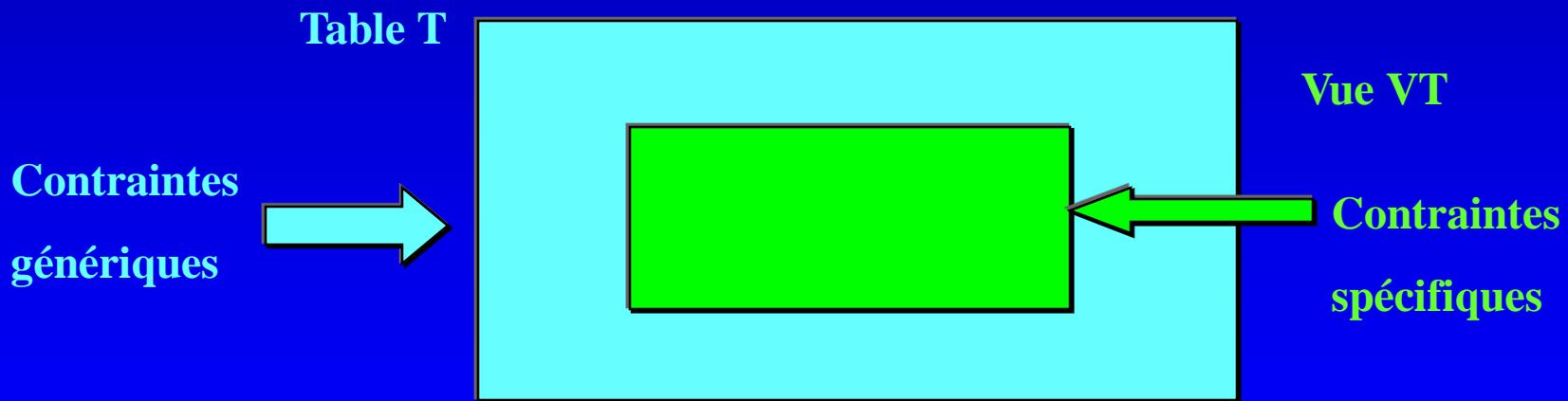
```
CREATE VIEW etudiants(idEtu,nom,adresse,
nomstage,entrstage) AS
SELECT e.id,e.nom,e.adr,s.nomS,s.entrS
FROM etudiant e,stage s WHERE e.id=s.id;
```

# Vues avec Contraintes WITH CHECK OPTION



SQL

- Principe : le prédicat de sélection de lignes se transforme en contrainte
- Mise en place de contraintes spécifiques :



# Exemple de vue avec Contrainte



SQL

- Les employés informaticiens ont des contraintes avantageuses ...

```
CREATE VIEW emp_info AS
SELECT * FROM emp
WHERE idSer IN
    (SELECT idSer FROM service
     WHERE UPPER(nomSer) LIKE '%INFO%')
AND sal > 3500
AND prime BETWEEN 500 AND 1000
WITH CHECK OPTION CONSTRAINT cko_emp_info;
```

- On ne pourra pas insérer un employé informaticien qui ne correspond pas aux contraintes du prédictat



# Restriction des Actions : Les privilèges

SQL

- **Privilèges Système et objet**
- **Contrôler l'accès à la base de données**
- **Sécurité système** : couvre l'accès à la base de données et son utilisation au niveau du système (nom de l'utilisateur et mot de passe, espace disque alloué aux utilisateurs et opérations système autorisées par l'utilisateur)
- **Sécurité données** : couvre l'accès aux objets de la base de données et leur utilisation, ainsi que les actions exécutées sur ces objets par les utilisateurs

# Privilèges système



SQL

- Plus de 100 privilèges système :
- Crédation d'utilisateurs (CREATE USER)
- Suppression de table (DROP ANY TABLE)
- Crédation d'espace disque (CREATE TABLESPACE)
- Sauvegarde des tables (BACKUP ANY TABLE)
- .....
- Les privilèges peuvent être regroupés dans des rôles (voir plus loin)

# Exemples de privilèges systèmes



SQL

Privilèges	ALTER	CREATE	DROP
PROCEDURE		×	×
ANY PROCEDURE		×	×
TABLE		×	
ANY TABLE	×	×	×
SESSION	×	×	
TABLESPACE	×	×	×
USER	×	×	×
VIEW		×	
.....			

# Délégation et suppression de privilèges



SQL

- Délégation : GRANT

```
GRANT {privilege_système | rôle}
[, {privilege2 | rôle2} ...]
TO {utilisateur1 | rôle | PUBLIC}
[, {utilisateur2 ...}]
[WITH ADMIN OPTION] ;
```

- Suppression : REVOKE

```
REVOKE {privilege_système | rôle}
[, {privilege2 | rôle2} ...]
FROM {utilisateur1 | rôle | PUBLIC}
[, {utilisateur2 ...}] ;
```

# Exemple de délégation et de suppression de privilèges Système



SQL

```
GRANT CREATE SESSION, CREATE TABLE  
DROP ANY TABLE TO michel;
```



Attention : suppression d'autres tables

```
REVOKE DROP ANY TABLE FROM michel;
```



On supprime que ce privilège

# Privilèges Objet



SQL

- **Contrôle les actions sur les objets**
  - Objets : tables, vues, séquences, ....
  - Actions : update, insert, execute, ....
- **Le propriétaire ('owner') peut donner ces privilèges sur ses propres objets**
- **Les privilèges peuvent être donnés avec l'option de délégation**

# Privilèges Objet

## Délégation et suppression



SQL

- Délégation : GRANT

```
GRANT privilège1 [,privilège2 ...]  
[ (colonne [,colonne2...]) ] ON schéma.objet  
TO {utilisateur1 | rôle | PUBLIC}  
[,{utilisateur2 ...}]  
[WITH GRANT OPTION] ;
```

- Suppression : REVOKE

```
REVOKE privilège1 [,privilège2 ...]  
[ (colonne [,colonne2...]) ]  
ON schéma.objet  
FROM {utilisateur1 | rôle | PUBLIC}  
[,{utilisateur2 ...}]  
[CASCADE CONSTRAINTS] ;
```

# Privilèges Objet Exemples



SQL

```
GRANT INSERT,UPDATE (adr,tel) ON etud_info  
TO Martine, Nicole ;
```

```
GRANT DELETE, UPDATE , INSERT ON etud_info  
TO Michel WITH GRANT OPTION;
```

```
REVOKE UPDATE (tel)   ON etud_info  
FROM Nicole ;
```

# Les privilèges Objet : objets et actions possibles



SQL

Actions \ Objets	Table	Vue	Séquence	Procédure Fonction Package	Snapshot
ALTER			X		
DELETE	X	X			
EXECUTE				X	
INDEX	X				
INSERT	X	X			
REFERENCES	X				
SELECT	X	X	X		X
UPDATE	X	X			

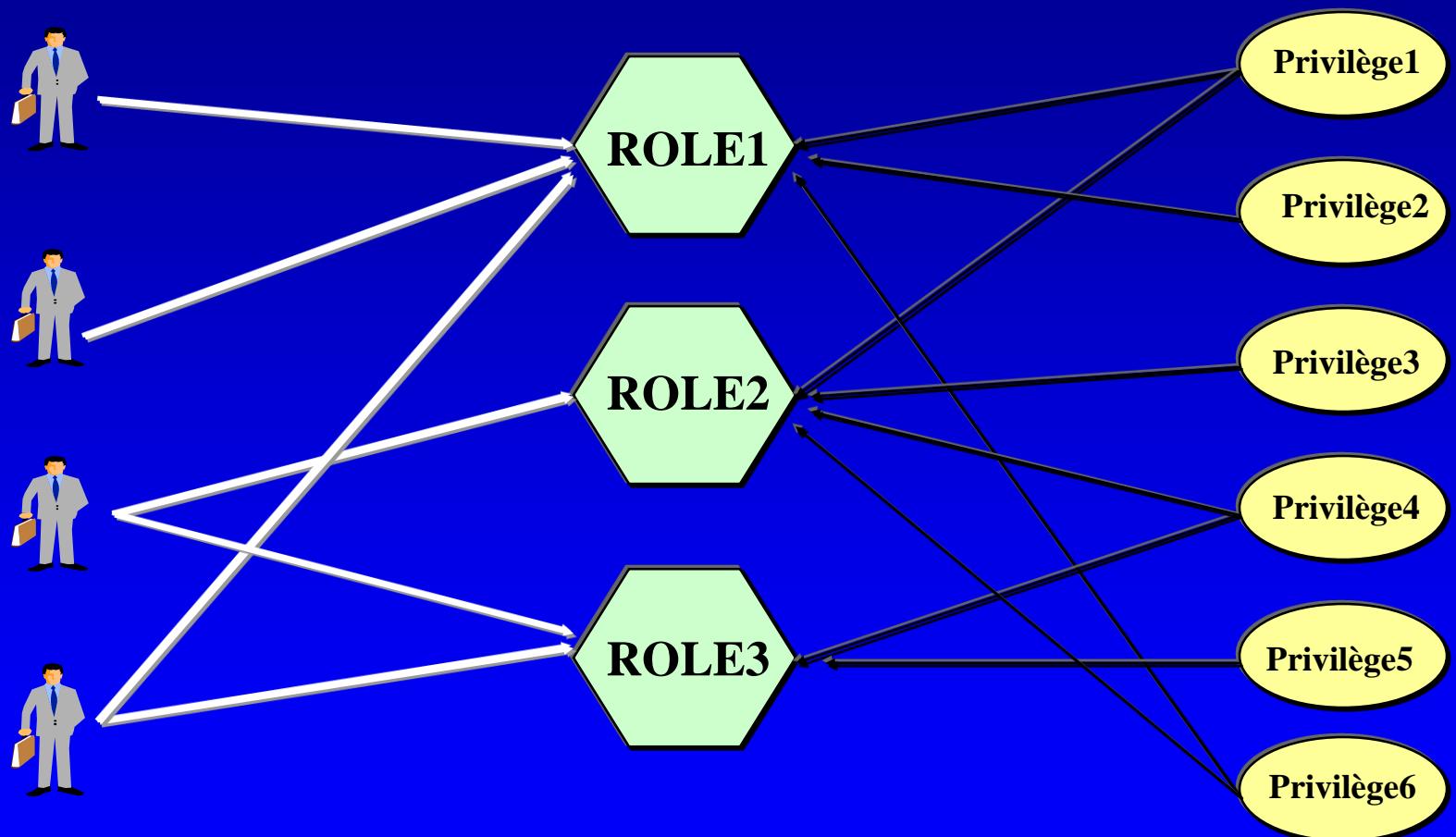
# Les Rôles



SQL

- **Regroupement de privilèges pour des familles d'utilisateur**
- **Facilitent la gestion des autorisations des privilèges objet en évitant les ordres GRANT**
- **Un rôle par défaut est donné à un utilisateur**
- **Un utilisateur peut posséder plusieurs rôles mais n'est connecté qu'avec un seul à la fois**
- **On peut donner un mot de passe pour certains rôles**

# Les Rôles : évitent le produit cartésien



# Manipulation des rôles : Ordres



SQL

- **Création / Modification d'un rôle**

```
{CREATE|ALTER} ROLE nom_rôle {NOT IDENTIFIED |  
IDENTIFIED {BY mot_de_passe | EXTERNALLY}} ;
```

- **Remplissage et attribution d'un rôle**

```
GRANT {privilege1 | rôle1} TO nom_rôle;  
GRANT {privilege2 | rôle2} TO nom_rôle;  
GRANT ROLE nom_role TO user;
```

- **Rôle par défaut ou activation**

```
SET ROLE nom_rôle [IDENTIFIED BY mot_de_passe];
```

- **Suppression / Révocation d'un rôle**

```
DROP ROLE nom_rôle;
```

```
REVOKE ROLE nom_rôle FROM user;
```

# Manipulation des rôles : Exemples



SQL

```
CREATE ROLE secretariat_info ;
```

```
GRANT SELECT,UPDATE (adr,tel)
ON ens_info TO secretariat_info;
GRANT SELECT,INSERT,UPDATE
ON etud_info TO secretariat_info;
GRANT SELECT,INSERT
ON cours_info TO secretariat_info;
```

```
GRANT secretariat_info TO
laurent, thomas, corinne ;
```

# Rôles prédéfinis

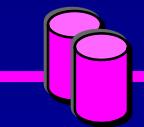


- **DBA**
  - Tous les privilèges système et objet
- **RESOURCE**
  - Crédit de tous les objets 'classiques'
  - Propriétaire des données ('owner')
- **CONNECT**
  - Connexion à la base
  - Attente de privilèges objet
- **EXP\_FULL\_DATABASE**
  - Exportation de tous les objets
- **IMP\_FULL\_DATABASE**
  - Importation d'objets

# Dictionnaire de données



- **ROLE\_SYS\_PRIVS**
  - privilèges systèmes accordés aux rôles
- **ROLE\_TAB\_PRIVS**
  - privilèges objets accordés aux rôles
- **USER\_ROLE\_PRIVS**
  - rôles accessibles par l'utilisateur



SQL

Le Langage de Contrôle  
de Données

TRIGGERS

# Généralités sur les Triggers



- **Programme évènementiel**
  - Bloc événement  
**(UPDATE, DELETE, INSERT)**
  - Bloc action  
**(bloc PL/SQL)**
- **Trois fonctions assurées**
  - Mise en place de contraintes complexes
  - Mise à jour de colonnes dérivées
  - Génération d'évènements
- **Associé à une table**
  - Suppression de la table → suppression des triggers

**SI évènement**  
**ALORS** action  
**SINON** rien  
**FINSI**

# 12 types de Triggers



SQL

- "Row" Trigger ou "Statement" Trigger
  - Row : le trigger est exécuté pour chaque ligne touchée
  - Statement : le trigger est exécuté une fois
- Exécution avant ou après l'événement
  - "before" : le bloc action est levé avant que l'événement soit exécuté
  - "after" : le bloc action est levé après l'exécution du bloc événement
- Trois évènements possibles
  - UPDATE : certaines colonnes
  - INSERT
  - DELETE

2

2

3

# Syntaxe de création



SQL

```
CREATE [OR REPLACE] TRIGGER <nom_trigger>
{BEFORE|AFTER}
{INSERT|DELETE|UPDATE [OF colonnes]}
ON <nom_table>
[FOR EACH ROW] ← row trigger si présent
[DECLARE]
-- déclaration de variables, exceptions
-- curseurs
BEGIN
-- bloc action
-- ordres SQL et PL/SQL
END ;
/
```

# Anciennes et nouvelles valeurs



SQL

- Pour les row trigger (triggers lignes) : accès aux valeurs des colonnes pour chaque ligne modifiée
- Deux variables : :NEW.colonne et :OLD.colonne

	Ancienne valeur :OLD.colonne	Nouvelle valeur :NEW.colonne
INSERT	NULL	Nouvelle valeur
DELETE	Ancienne valeur	NULL
UPDATE	Ancienne valeur	Nouvelle valeur



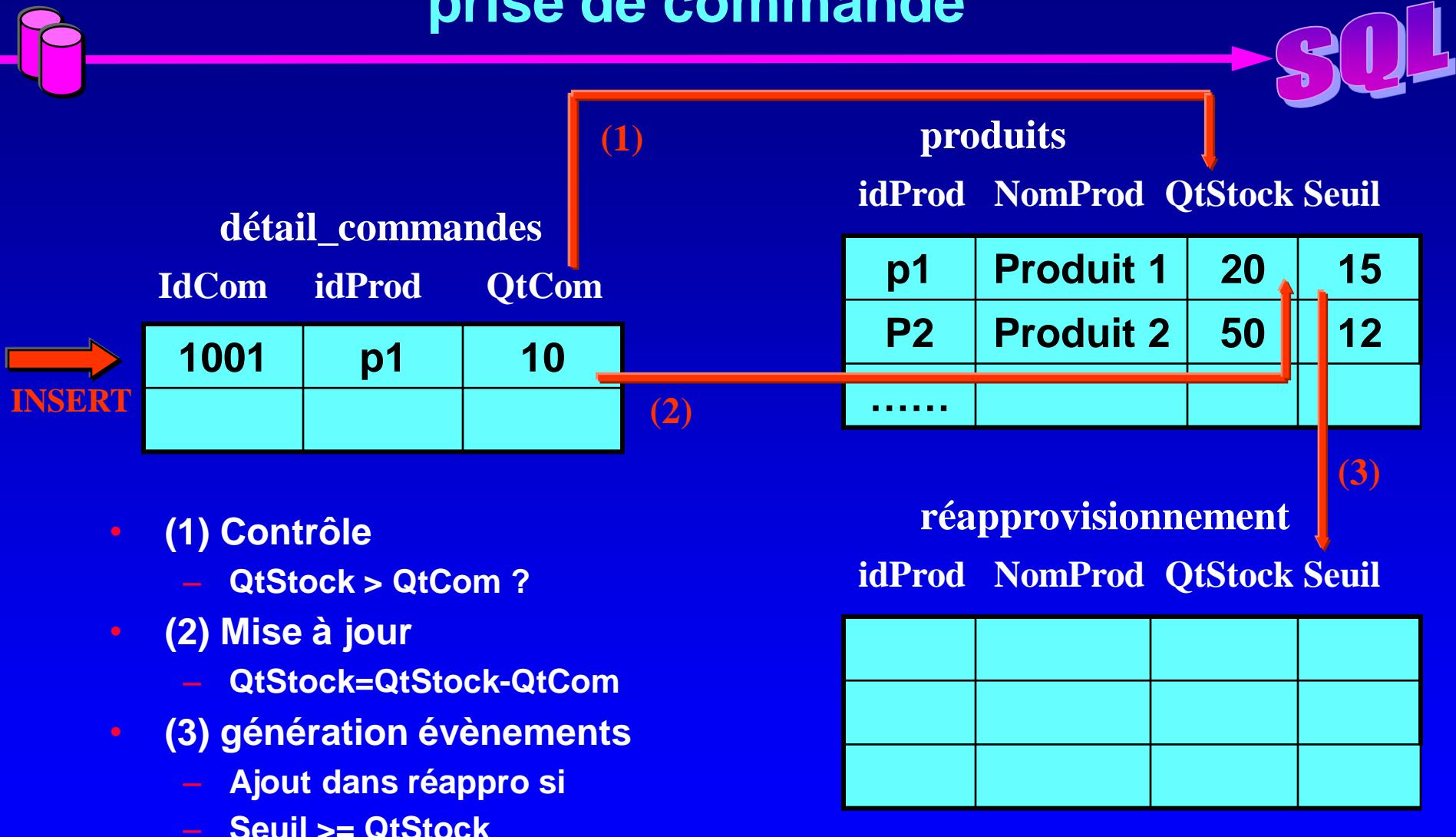
# Trigger de contraintes : ‘lever’ une erreur

- Test de **contraintes** : erreur ou pas
- Ordre : **RAISE\_APPLICATION\_ERROR**

```
RAISE_APPLICATION_ERROR(n°erreur, 'texte erreur')
```

- N°erreur : [-20000 , -20999] → SQLCODE
- Texte erreur : message envoyé → SQLERRM

# Exemples de row trigger : prise de commande



- **(1) Contrôle**
  - $QtStock > QtCom$  ?
- **(2) Mise à jour**
  - $QtStock=QtStock-QtCom$
- **(3) génération évènements**
  - Ajout dans réappro si
  - $Seuil \geq QtStock$

# Prise de commande : (1) contrôle quantité en stock ?



SQL

```
CREATE TRIGGER t_b_i_detail_commandes
BEFORE INSERT ON détail_commandes
FOR EACH ROW
DECLARE
v_qtstock NUMBER;
BEGIN
SELECT qtstock INTO v_qtstock FROM produits
WHERE idprod = :NEW.idprod;
IF v_qtstock < :NEW.qtcom THEN
RAISE_APPLICATION_ERROR(-20001,'stock insuffisant');
END IF;
END;
/
```



## Prise de commande : (2) Mise à jour quantité en stock

```
CREATE TRIGGER t_a_i_detail_commandes
AFTER INSERT ON detail_commandes
FOR EACH ROW
BEGIN
UPDATE produits p
SET p.qtstock = p.qtstock - :NEW.qtcom
WHERE idprod = :NEW.idprod;
END;
/
```

# Prise de commande :

## (3) génération d'un réapprovisionnement



SQL

```
CREATE TRIGGER t_a_u_produits
AFTER UPDATE OF qtstock ON produits
FOR EACH ROW
BEGIN
IF :NEW.qtstock <= :NEW.seuil THEN
INSERT INTO reapprovisionnement VALUES
(:NEW.idprod, :NEW.nomprod, :NEW.qtstock,
:NEW.seuil);
END IF;
END;
/
```

# Les prédictats dans un trigger



SQL

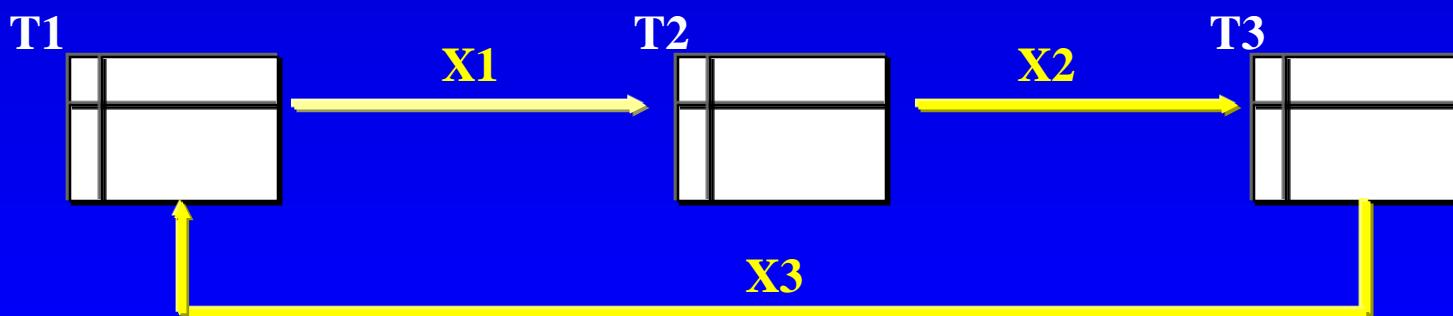
```
CREATE TRIGGER <nom_trigger>
{BEFORE|AFTER}
INSERT OR DELETE OR UPDATE [OF colonnes] }
ON <nom_table>
[FOR EACH ROW]
[DECLARE]
-- déclaration de variables, exceptions
BEGIN
IF UPDATING('colonne') THEN ..... END IF;
IF DELETING THEN ..... END IF;
IF INSERTING THEN ..... END IF;
END ;
/
```

# Limitation des Trigger



SQL

- Impossible d'accéder sur la table sur laquelle porte le trigger
- Attention aux effets de bords :
  - exemple : trigger x1 de la table T1 fait un insert dans la table T2 qui possède un trigger x2 qui modifie T3 qui possède un trigger x3 qui modifie T1





# Les triggers d'état ou ‘Statement trigger’

**SQL**

- Raisonnement sur la globalité de la table et non sur un enregistrement particulier
- TRIGGER BEFORE : 1 action avant un ordre UPDATE de plusieurs lignes
- TRIGGER AFTER : 1 action après un ordre UPDATE touchant plusieurs lignes

# Exemple de Statement Trigger



SQL

- Interdiction d'emprunter un ouvrage pendant le week-end

```
CREATE TRIGGER controle_date_emp
BEFORE UPDATE OR INSERT OR DELETE ON emprunt
BEGIN
IF TO_CHAR(SYSDATE,'DY')
IN ('SAT','SUN') THEN
RAISE APPLICATION_ERROR
(-20102,'Désolé les emprunts sont
interdits le week-end...') ;
END IF;
END;
/
```

# Les Triggers 'INSTEAD OF'



SQL

- Trigger faisant le travail 'à la place de' .....
- Posé sur une vue multi-table pour autoriser les modifications sur ces objets virtuels
- Utilisé dans les bases de données réparties pour permettre les modifications sur le objets virtuels fragmentés (cours BD Réparties)
- Permet d'assurer un niveau d'abstraction élevé pour les utilisateurs ou développeurs clients : les vraies mises à jour sont faites à leur insu ....

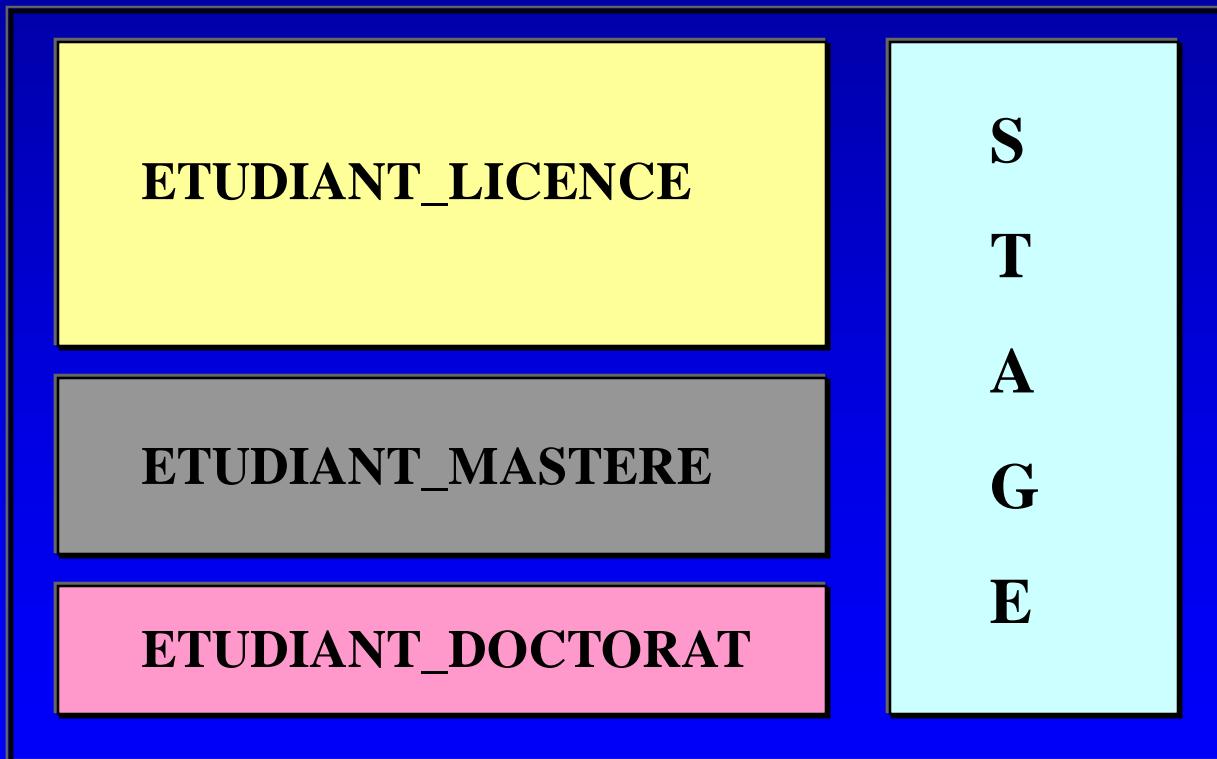
# Exemple de trigger ‘instead of’



SQL

- Vue étudiant résultant de 4 tables

ETUDIANT



# Exemple de trigger ‘instead of’ Construction de la vue ETUDIANT



SQL

Colonne virtuelle

```
CREATE VIEW etudiant
(ine,nom,adresse,cycle,nomstage,adstage)
AS SELECT el.ine,el.nom,el.adr,'L',s.noms,s.ads
FROM etudiant_licence el, stage s
WHERE el.ine=s.ine
UNION
SELECT em.ine,em.nom,em.adr,'M',s.noms,s.ads
FROM etudiant_mastere em, stage s
WHERE em.ine=s.ine
UNION
SELECT ed.ine,ed.nom,ed.adr,'D',s.noms,s.ads
FROM etudiant_doctorat ed, stage s
WHERE ed.ine=s.ine;
```

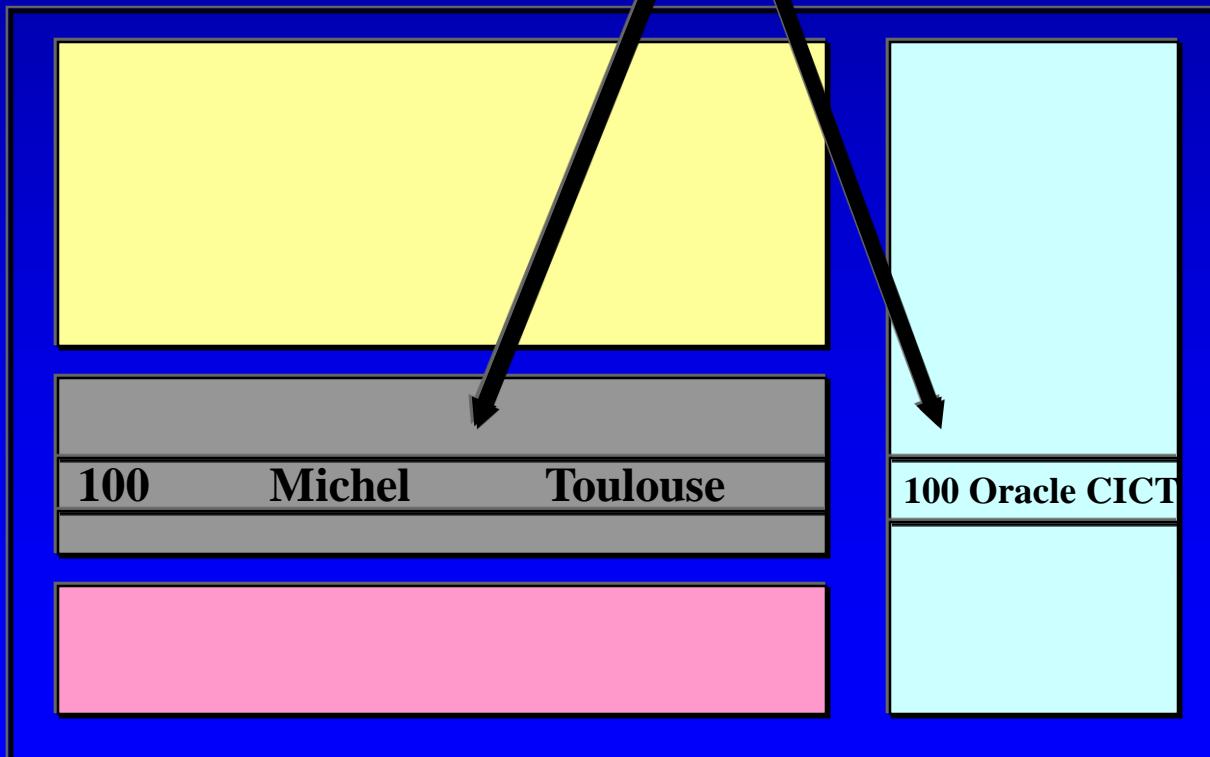
# Exemple de trigger ‘instead of’ utilisation de la vue : INSERT



SQL

```
INSERT INTO etudiant VALUES  
(100,'Michel','Toulouse','M','Oracle','CICT');
```

ETUDIANT



# Exemple de trigger ‘instead of’ trigger pour INSERT

 SQL

```
CREATE TRIGGER insert_etudiant
INSTEAD OF INSERT ON etudiant FOR EACH ROW
BEGIN
IF :NEW.cycle='L' THEN
INSERT INTO etudiant_licence VALUES
(:NEW.ine,:NEW.nom,:NEW.adresse);
INSERT INTO stage VALUES
(:NEW.ine,:NEW.nomstage,:NEW.adstage);
ELSIF :NEW.cycle='M' THEN
..... Idem pour M et D .....
ELSE RAISE_APPLICATION_ERROR
(-20455,'Entrer M, L ou D');
END IF;
END;
/
```

# Manipulation des Triggers



SQL

- Suppression d'un trigger

```
DROP TRIGGER <nomtrigger> ;
```

- Désactivation d'un trigger

```
ALTER TRIGGER <nomtrigger> DISABLE ;
```

- Réactivation d'un trigger

```
ALTER TRIGGER <nomtrigger> ENABLE ;
```

- Tous les triggers d'une table

```
ALTER TABLE <nomtable>
{ENABLE|DISABLE} ALL TRIGGERS ;
```

# Dictionnaire des Données



SQL

- **USER\_TRIGGERS**

TRIGGER_NAME	VARCHAR2 (30)
TRIGGER_TYPE	VARCHAR2 (16)
TRIGGERING_EVENT	VARCHAR2 (227)
TABLE_OWNER	VARCHAR2 (30)
BASE_OBJECT_TYPE	VARCHAR2 (16)
TABLE_NAME	VARCHAR2 (30)
COLUMN_NAME	VARCHAR2 (4000)
REFERENCING_NAMES	VARCHAR2 (128)
WHEN_CLAUSE	VARCHAR2 (4000)
STATUS	VARCHAR2 (8)
DESCRIPTION	VARCHAR2 (4000)
ACTION_TYPE	VARCHAR2 (11)
TRIGGER_BODY	LONG

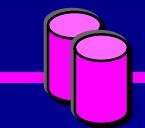
# Dictionnaire des Données (suite)



SQL

- **USER\_TRIGGER\_COLS**

TRIGGER_OWNER	VARCHAR2 (30)
TRIGGER_NAME	VARCHAR2 (30)
TABLE_OWNER	VARCHAR2 (30)
TABLE_NAME	VARCHAR2 (30)
COLUMN_NAME	VARCHAR2 (4000)
COLUMN_LIST	VARCHAR2 (3)
COLUMN_USAGE	VARCHAR2 (17)



**SQL**

A large, stylized, magenta/pink text representation of the word "SQL". It has a three-dimensional effect with shadows and highlights, and a thick, rounded font style.

**Le Langage de BLOC**

A large, bold, cyan text block containing the phrase "Le Langage de BLOC".

**PL/SQL**

A large, bold, cyan text block containing the phrase "PL/SQL".

# Le Langage de Bloc PL/SQL # SQL



- SQL : langage ensembliste
  - Ensemble de requêtes distinctes
  - Langage de 4ème génération : on décrit le résultat sans dire comment il faut accéder aux données
  - Obtention de certains résultats : encapsulation dans un langage hôte de 3ème génération
- PL/SQL
  - ‘Procédural Language’ : sur-couche procédurale à SQL, boucles, contrôles, affectations, exceptions, ....
  - Chaque programme est un bloc (BEGIN – END)
  - Programmation adaptée pour :
    - Transactions
    - Une architecture Client - Serveur

# Requêtes SQL



SQL

- Chaque requête ‘client’ est transmise au serveur de données pour être exécutée avec retour de résultats

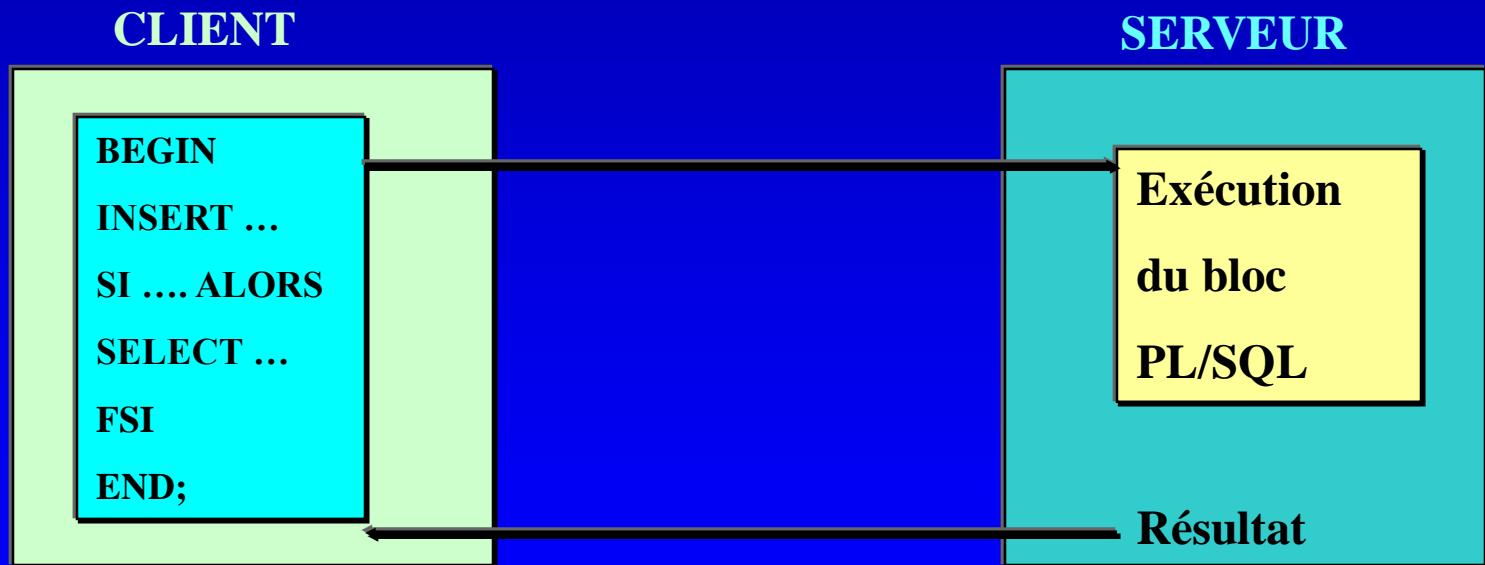


# Bloc PL/SQL



**SQL**

- **Le bloc de requêtes est envoyé sur le serveur. Celui-ci exécute le bloc et renvoie 1 résultat final.**



# Format d'un bloc PL/SQL



- **Section DECLARE : déclaration de**
  - Variables locales simples
  - Variables tableaux
  - cursors
- **Section BEGIN**
  - Section des ordres exécutables
  - Ordres SQL
  - Ordres PL
- **Section EXCEPTION**
  - Réception en cas d'erreur
  - Exceptions SQL ou utilisateur

```
DECLARE
    --déclarations
BEGIN
    --exécutions
EXCEPTION
    --erreurs
END;
/
```

# Variables simples



SQL

- Variables de type SQL

```
nbr      NUMBER (2) ;
nom      VARCHAR (30) ;
minimum CONSTANT INTEGER := 5 ;
salaire NUMBER (8,2) ;
debut    NUMBER NOT NULL ;
```

- Variables de type booléen (TRUE, FALSE, NULL)

```
fin      BOOLEAN ;
reponse BOOLEAN DEFAULT TRUE ;
ok      BOOLEAN := TRUE ;
```



# Variables faisant référence au dictionnaire de données

SQL

- Référence à une colonne (table, vue)

```
vsalaire    employe.salaire%TYPE;  
vnom        etudiant.nom%TYPE;  
Vcomm       vsalaire%TYPE;
```

- Référence à une ligne (table, vue)

```
vemploye   employe%ROWTYPE;  
vetudiant  etudiant%ROWTYPE;
```

- Variable de type ‘struct’
- Contenu d’une variable : variable.colonne

```
vemploye.adresse
```

# Tableaux dynamiques



SQL

- Déclaration d'un type tableau

```
TYPE <nom du type du tableau>
IS TABLE OF <type de l'élément>
INDEX BY BINARY_INTEGER;
```

- Affectation (héritage) de ce type à une variable

```
<nom élément> <nom du type du tableau>;
```

- Utilisation dans la section BEGIN : un élément du tableau :

```
<nom élément> (rang dans le tableau)
```

# Tableaux dynamiques variables simples



SQL

- Déclaration d'un tableau avec des éléments numériques

```
TYPE type_note_tab
IS TABLE OF NUMBER(4,2)
INDEX BY BINARY_INTEGER;
tab_notes type_note_tab;
i NUMBER;
```

```
i:=1;
tab_notes(i) := 12.5;
```

- Déclaration d'un tableau avec des éléments caractères

```
TYPE type_nom_tab
IS TABLE OF VARCHAR(30)
INDEX BY BINARY_INTEGER;
tab_noms type_nom_tab;
i NUMBER;
```

```
i:=1;
tab_noms(i) := 'toto';
```

# Tableaux dynamiques variables simples avec héritage



SQL

- Tableau avec éléments hérités

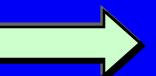
```
TYPE type_note_tab
IS TABLE OF partiel.note%TYPE
INDEX BY BINARY_INTEGER;
tab_notes type_note_tab;
i NUMBER;
```

```
i:=1;
tab_notes(i) := 12.5;
```



```
TYPE type_nom_tab
IS TABLE OF etudiant.nom%TYPE
INDEX BY BINARY_INTEGER;
tab_noms type_nom_tab;
i NUMBER;
```

```
i:=1;
tab_noms(i) := 'toto';
```



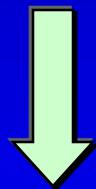
# Tableaux dynamiques avec des éléments de type RECORD



SQL

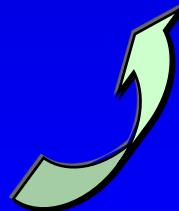
- Type RECORD : plusieurs variables dans un élément

```
TYPE type_emp_record  
(idEmp NUMBER,  
 nomEmp VARCHAR(30),  
 adrEmp VARCHAR(80));
```



```
i:=1;  
tab_emps(i).idEmp:= 100;  
tab_emps(i).nomEmp:= 'toto';  
tab_emps(i).adrEmp:= 'tlse';
```

```
TYPE type_emp_tab  
IS TABLE OF type_emp_record  
INDEX BY BINARY_INTEGER;  
tab_emps type_emp_tab;  
i NUMBER;
```



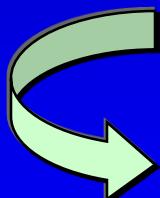
# Tableaux dynamiques avec des éléments de type ROW



SQL

- Type ROW : chaque élément est une variable ‘struct’

```
TYPE type_emp_tab
IS TABLE OF employe%ROWTYPE
INDEX BY BINARY_INTEGER;
tab_emps type_emp_tab;
i NUMBER;
```



```
i:=1;
tab_emps(i).idE:= 100;
tab_emps(i).nom:= 'toto';
tab_emps(i).adresse:= 'tlse';
```



# Variables paramétrées lues sous SQLPLUS : &

- Variables lues par un ACCEPT .... PROMPT

+ {

```
ACCEPT plu PROMPT 'Entrer la valeur : '
```

PL {

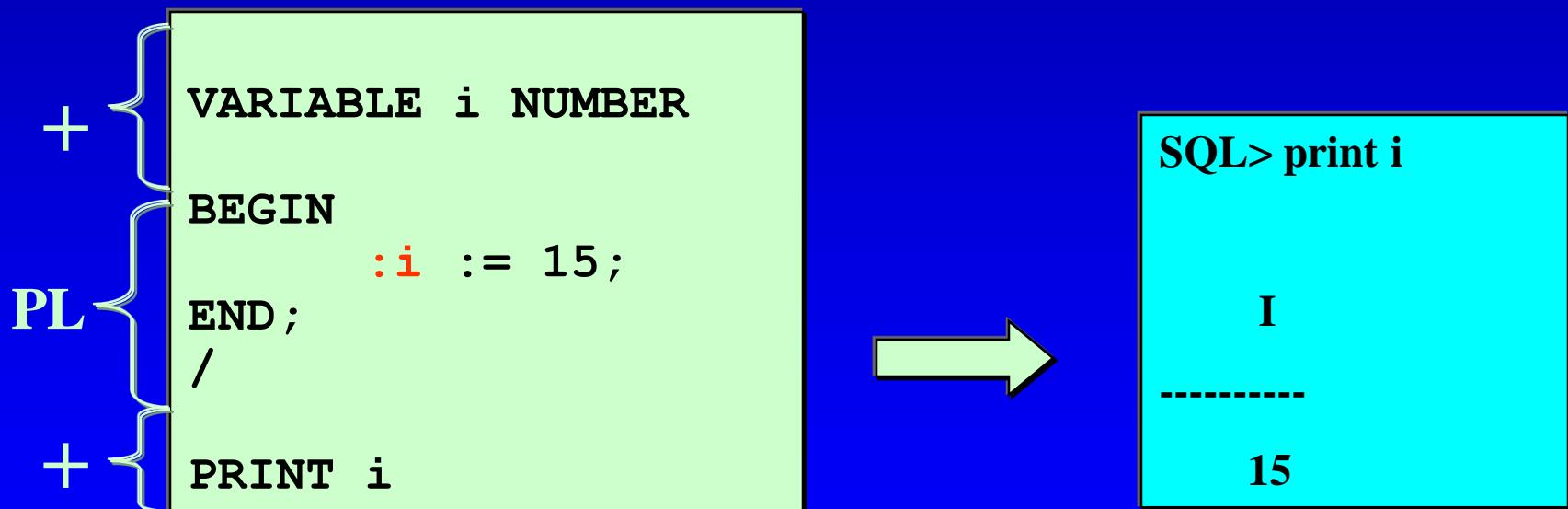
```
DECLARE
    -- déclarations
BEGIN
    -- travail avec le contenu de plu :
    -- &plu si numérique
    -- '&plu' si caractère
END;
/
-- Ordre SQL .....
```

+ }



# Variables en sortie sous SQLPLUS ::

- Variable déclarée sous sqlplus , utilisée dans le bloc PL puis affichée sous sqlplus



# Instructions PL



SQL

- **Affectation (:=)**
  - A := B;
- **Structure alternative ou conditionnelle**
  - Opérateurs SQL : >,<,....,OR,AND,....,BETWEEN,LIKE,IN
  - IF .... THEN ..... ELSE .....END IF;

```
IF condition THEN
    instructions;
ELSE
    instructions;
IF condition THEN instructions;
ELSIF condition THEN instructions;
ELSE instructions;
END IF;
```

# Structure alternative : CASE (1)



SQL

- Choix selon la valeur d'une variable

```
CASE variable
```

```
    WHEN valeur1 THEN action1;  
    WHEN valeur2 THEN action2;
```

```
.....
```

```
    ELSE    action;
```

```
END CASE;
```

# Structure alternative : CASE (2)



SQL

- Plusieurs choix possibles

**CASE**

```
WHEN expression1 THEN      action1;  
WHEN expression2 THEN      action2;  
.....  
ELSE    action;  
  
END CASE;
```

# Structure itérative



SQL

- **LOOP**

```
LOOP  
    instructions;  
    EXIT WHEN (condition);  
END LOOP;
```

- **FOR**

```
FOR (indice IN [REVERSE] borne1..borne2) LOOP  
    instructions;  
END LOOP;
```

- **WHILE**

```
WHILE (condition) LOOP  
    instructions;  
END LOOP;
```

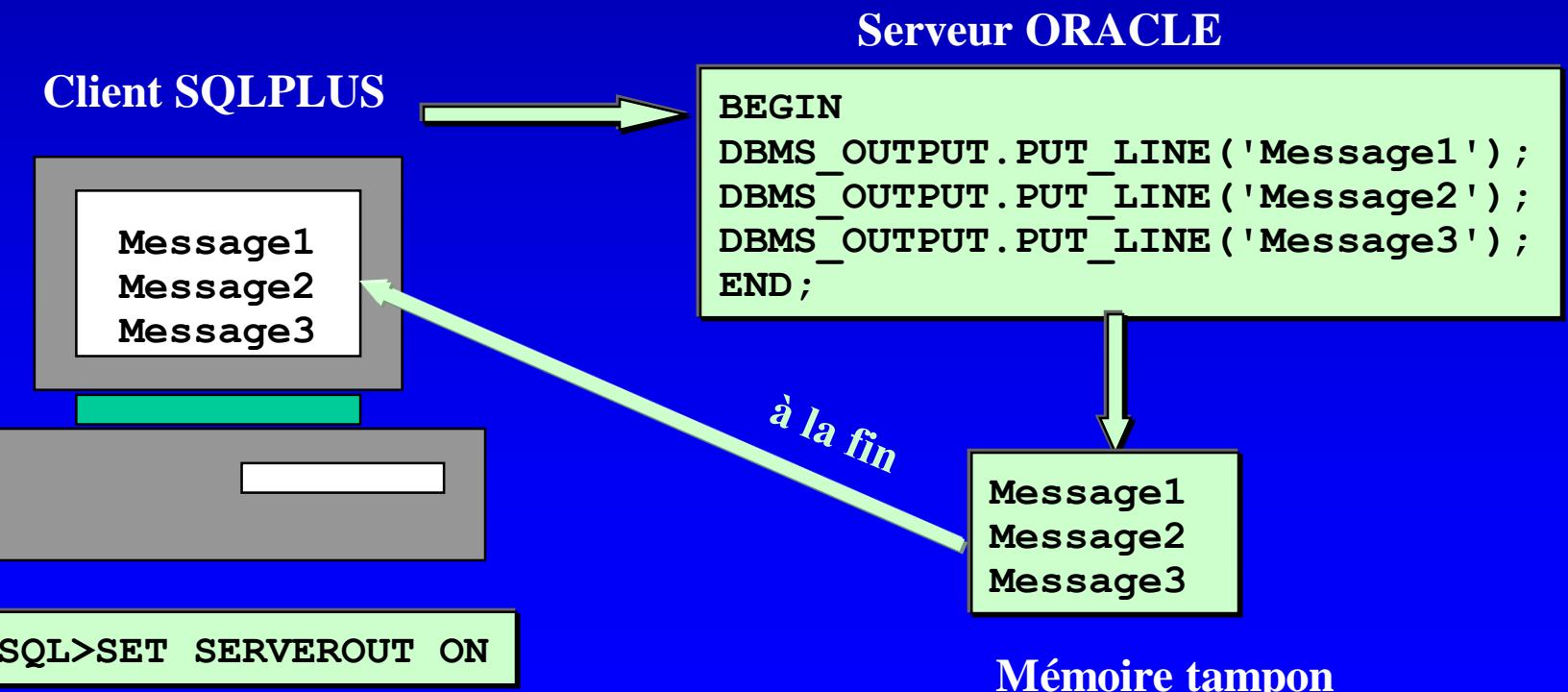
# Affichage de résultats intermédiaires

## Package DBMS\_OUTPUT



SQL

- Messages enregistrés dans une mémoire tampon côté serveur
- La mémoire tampon est affichée sur le poste client à la fin



# Le package DBMS\_OUTPUT



SQL

- Écriture dans le buffer avec saut de ligne
  - DBMS\_OUTPUT.PUT\_LINE(<chaîne caractères>);
- Écriture dans le buffer sans saut de ligne
  - DBMS\_OUTPUT.PUT(<chaîne caractères>);
- Écriture dans le buffer d'un saut de ligne
  - DBMS\_OUTPUT.NEW\_LINE;

```
DBMS_OUTPUT.PUT_LINE('Affichage des n premiers ');
DBMS_OUTPUT.PUT_LINE('caractères en ligne ');
FOR i IN 1..n LOOP
    DBMS_OUTPUT.PUT(tab_cars(i));
END LOOP;
DBMS_OUTPUT.NEW_LINE;
```



# Sélection mono – ligne

## SELECT .... INTO



- Toute valeur de colonne est rangée dans une variable avec INTO

```
SELECT nom,adresse,tel INTO vnom,vadresse,vtel  
FROM etudiant WHERE ine=&nolu;
```

```
SELECT nom,adresse,libDip INTO vnom,vadresse,vdip  
FROM etudiant e, diplôme d WHERE ine=&nolu  
AND e.idDip=d.idDip;
```

- Variable ROWTYPE

```
SELECT * INTO vretud FROM etudiant WHERE ine=&nolu;  
.....  
DBMS_OUTPUT.PUT_LINE('Nom étudiant : '||vretud.nom);  
.....
```

# Sélection multi – ligne : les CURSEURS

## Principe des curseurs



SQL

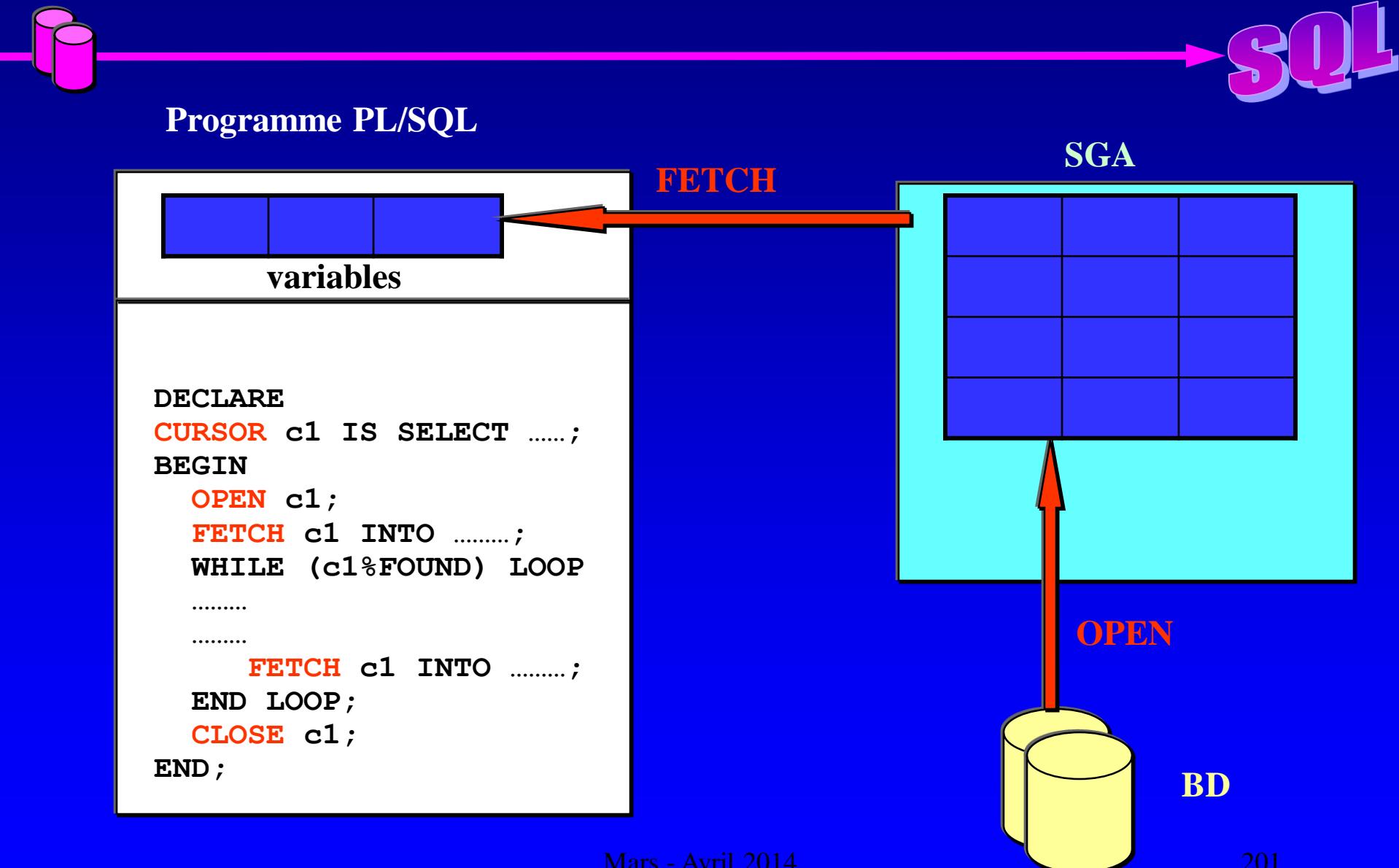
- Obligatoire pour sélectionner plusieurs lignes
- Zone mémoire (SGA : Share Global Area) partagée pour stocker les résultats
- Le curseur contient en permanence l'`@` de la ligne courante
- Curseur implicite
  - `SELECT t.* FROM table t WHERE .....`
  - `t` est un curseur utilisé par SQL
- Curseur explicite
  - `DECLARE CURSOR →`

# Démarche générale des curseurs



- Déclaration du curseur : **DECLARE**
  - Ordre SQL sans exécution
- Ouverture du curseur : **OPEN**
  - SQL ‘monte’ les lignes sélectionnées en SGA
  - Verrouillage préventif possible (voir + loin)
- Sélection d'une ligne : **FETCH**
  - Chaque FETCH ramène une ligne dans le programme client
  - Tant que ligne en SGA : FETCH
- Fermeture du curseur : **CLOSE**
  - Récupération de l'espace mémoire en SGA

# Traitement d'un curseur



# Gestion 'classique' d'un curseur

```
DECLARE
CURSOR c1 IS SELECT nom,moyenne FROM etudiant ORDER BY 1;
vnom          etudiant.nom%TYPE;
vmoyenne      etudiant.moyenne%TYPE;
e1 ,e2 NUMBER;
BEGIN
    OPEN c1;
    FETCH c1 INTO vnom,vmoyenne;
    WHILE c1%FOUND LOOP
        IF vmoyenne < 10 THEN e1:=e1+1;
            INSERT INTO liste_refus VALUES(vnom) ;
        ELSE      e2:=e2+1;
            INSERT INTO liste_refus VALUES(vnom) ;
        END IF;
        FETCH c1 INTO vnom,vmoyenne;
    END LOOP;
    CLOSE c1;
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(e2)||'Reçus ');
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(e1)||'Collés ');
    COMMIT;
END;
```

# Les variables système des Curseurs



- **Curseur%FOUND**
  - Variable booléenne
  - Curseur toujours ‘ouvert’ (encore des lignes)
- **Curseur%NOTFOUND**
  - Opposé au précédent
  - Curseur ‘fermé’ (plus de lignes)
- **Curseur%COUNT**
  - Variable number
  - Nombre de lignes déjà retournées
- **Curseur%ISOPEN**
  - Booléen : curseur ouvert ?

# Gestion ‘automatique’ des curseurs



SQL

```
DECLARE
CURSOR c1 IS SELECT nom,moyenne FROM etudiant ORDER BY 1;
-- PAS DE DECLARATION DE VARIABLE DE RECEPTION
e1 ,e2 NUMBER;
BEGIN
    --PAS D' OUVERTURE DE CURSEUR
    --PAS DE FETCH
    FOR c1_ligne IN c1 LOOP
        IF c1_ligne.moyenne < 10 THEN e1:=e1+1;
        INSERT INTO liste_refus VALUES(c1_ligne.nom);
        ELSE e2:=e2+1;
        INSERT INTO liste_refus VALUES(c1_ligne.nom);
        END IF;
    END LOOP;
    --PAS DE CLOSE
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(e2) || 'Reçus ');
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(e1) || 'Collés ');
    COMMIT;
END;
```

Variable STRUCT de réception

# Curseurs et Tableaux exemple final



SQL

```
DECLARE
CURSOR c1 IS SELECT nom,moyenne FROM etudiant
WHERE moyenne>=10 ORDER BY 2 DESC;
TYPE type_nom_tab IS TABLE OF etudiant.nom%TYPE
INDEX BY BINARY_INTEGER;
tab_noms type_nom_tab;
i,j NUMBER;
BEGIN /* Remplissage tableau */
i:=1;
FOR c1_ligne IN c1 LOOP
tab_noms(i) := c1.ligne.nom;
i:=i+1;
END LOOP; /* Affichage du tableau */
FOR j IN 1..i-1 LOOP
DBMS_OUTPUT.PUT_LINE('Rang : '||TO_CHAR(j)||'
Etudiant : '||tab_nom(j));
END LOOP;
END ;
```

# Gestion des Exceptions

## Principe



SQL

- Toute erreur (SQL ou applicative) entraîne automatiquement un débranchement vers le paragraphe EXCEPTION :

```
BEGIN
    instruction1;
    instruction2;
    .....
    instructionn;
EXCEPTION
    WHEN exception1 THEN
        .....
    WHEN exception2 THEN
        .....
    WHEN OTHERS THEN
        .....
END;
```

Débranchement involontaire (erreur SQL)  
ou volontaire (erreur applicative)

A red arrow points from the text "WHEN exception2 THEN" in the main block to the start of the exception handling block.

# Deux types d'exceptions



SQL

- Exceptions SQL

- Déjà définies (pas de déclaration)
  - DUP\_VAL\_ON\_INDEX
  - NO\_DATA\_FOUND
  - OTHERS
- Non définies
  - Déclaration obligatoire avec le n° erreur (sqlcode)

```
nomerreur EXCEPTION;  
PRAGMA EXCEPTION_INIT(nomerreur,n°erreur);
```

- Exceptions applicatives

- Déclaration sans n° erreur

```
nomerreur EXCEPTION;
```

# Exemple de gestion d'exception (1)



SQL

```
DECLARE
tropemprunt    EXCEPTION;
i NUMBER;
BEGIN
    i:=1;
    SELECT .....
    i:=2;
    SELECT .....
    IF ..... THEN RAISE tropemprunt; .....
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        IF i=1 THEN .....
            ELSE
        END IF;
    WHEN tropemprunt THEN
        .....
    WHEN OTHERS THEN
        .....
END;
```

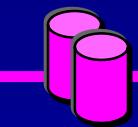
# Exemple de gestion d'exception (2)



SQL

```
DECLARE
enfant_sans_parent EXCEPTION;
PRAGMA EXCEPTION_INIT(enfant_sans_parent,-2291);
BEGIN
    INSERT INTO fils VALUES ( ..... );

EXCEPTION
    WHEN enfant_sans_parent THEN
        .....
    WHEN OTHERS THEN
        .....
END;
```



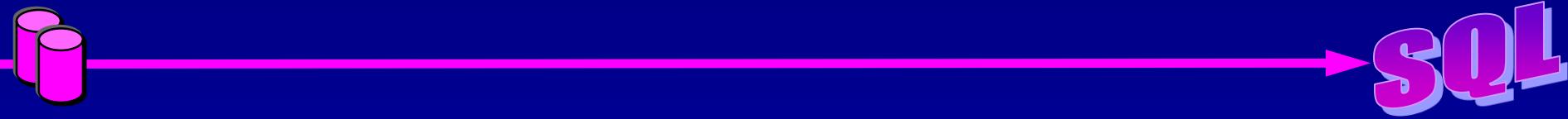
SQL

# Procédures Stockées

## Fonctions

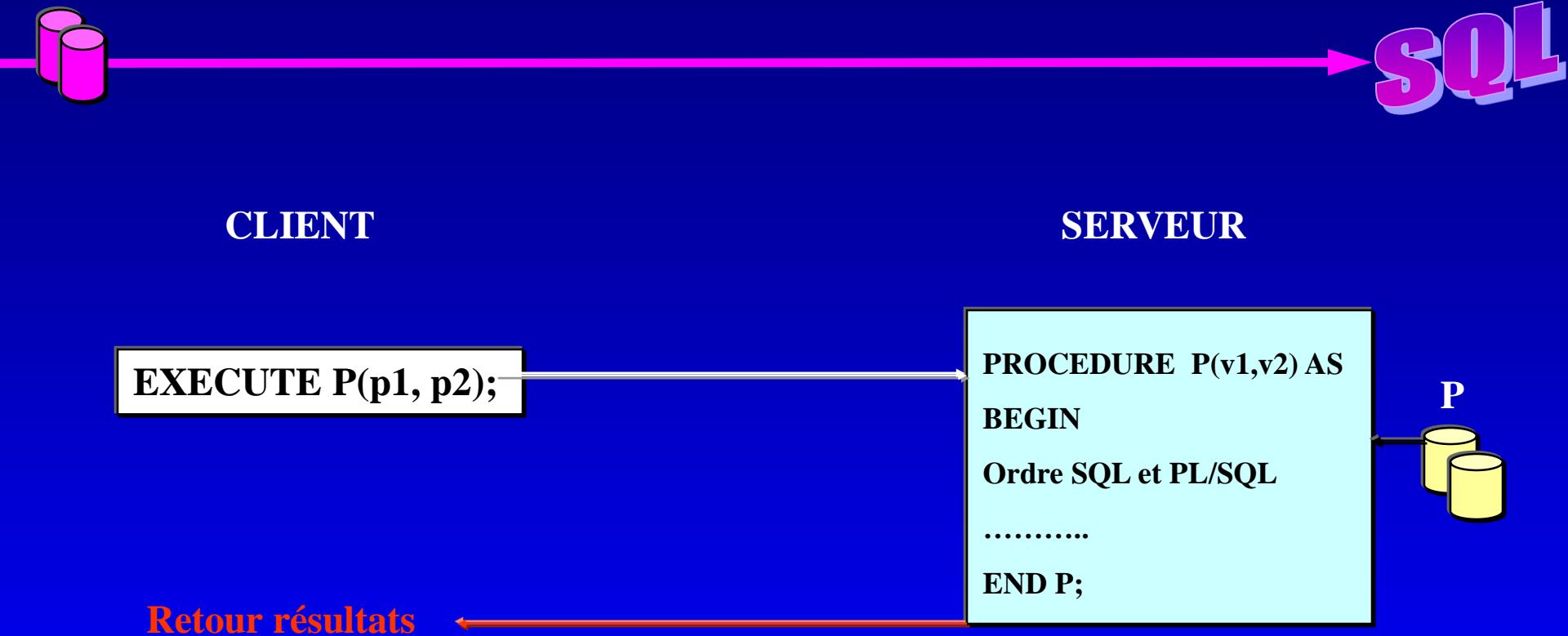
## Paquetages

# Procédures Stockées : Principe (1)



- Programme (PL/SQL) stocké dans la base
- Le programme client exécute ce programme en lui passant des paramètres (par valeur)
- Si le code est bon , le SGBD conserve le programme source (USER\_SOURCE) et le programme compilé
- Le programme compilé est optimisé en tenant compte des objets accélérateurs (INDEX, CLUSTER, PARTITION, ...)

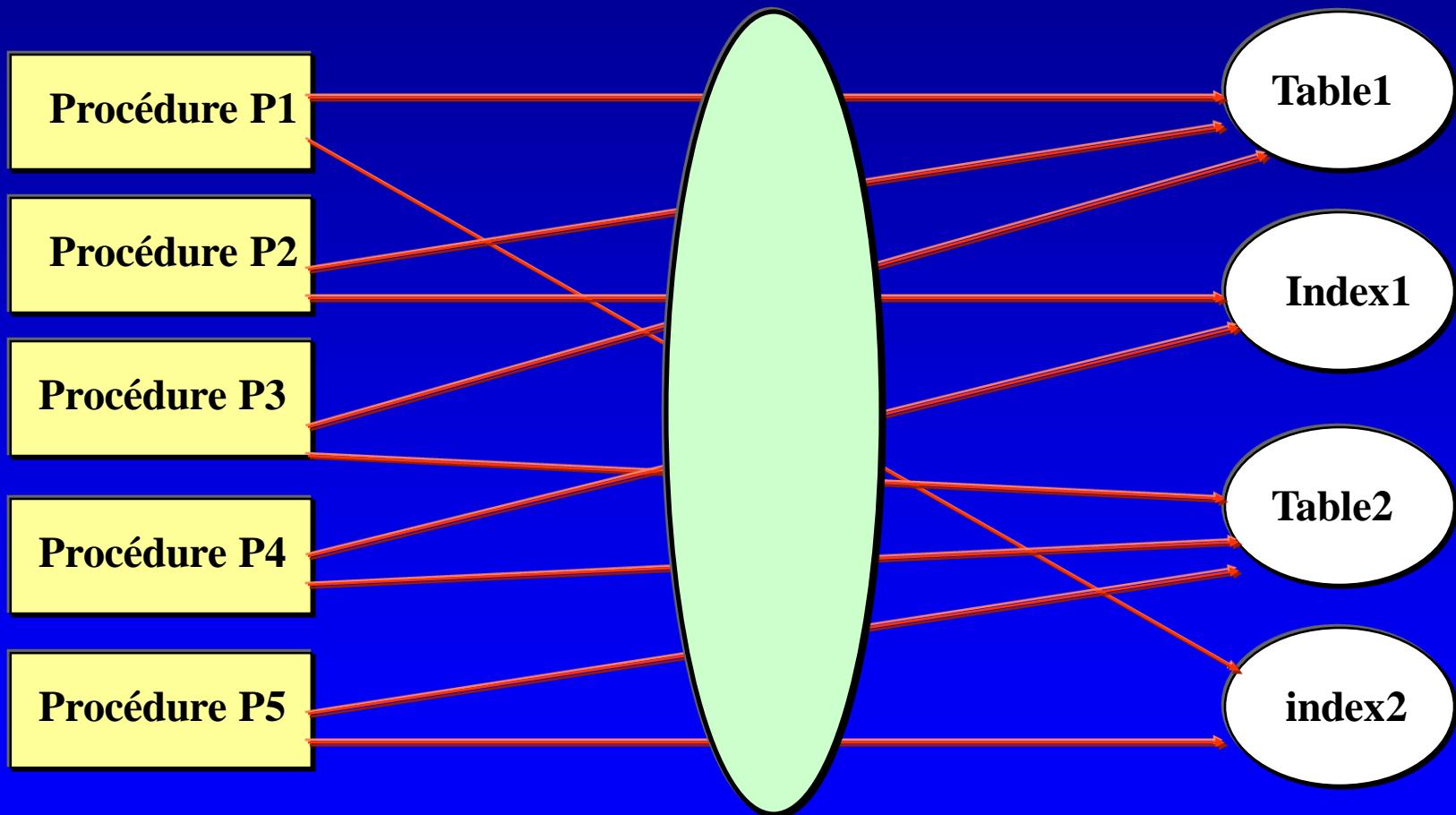
# Procédures Stockées : Principe (2)



# Optimisation des procédures liens avec les objets



Références croisées



# Optimisation des procédures



SQL

- Recompilation automatique d'une procédure si un objet est modifié
- Recompilation manuelle possible

```
ALTER PROCEDURE <nom_procédure> COMPILE;
```

# Avantages des procédures stockées



SQL

- **Vitesse** : programme compilé et optimisé
  - Une requête SQL normale est interprétée et optimisée à chaque exécution
- **Intégrité** : encapsulation des données
  - Vers le modèle Objet
  - Droit d'exécution et plus de manipulation
  - Les règles de gestion sont données sur le serveur en un seul exemplaire
- **Performance** : moins de transfert réseau
  - Plus de transfert de bloc de programme
  - Une procédure pour plusieurs utilisateurs
- **Abstraction** : augmentation du niveau d'abstraction des développeurs Client
- **Performance** :
  - Extensibilité, Modularité, Réutilisation, Maintenance

# Déclaration d'une procédure stockée



SQL

```
CREATE [OR REPLACE] PROCEDURE <nom_procédure>
[(variable1 type1, ..., variablen typen [OUT]) ] AS
...
-- déclarations des variables et
-- curseurs utilisées dans le corps de la procédure
BEGIN
...
-- instructions SQL ou PL/SQL
EXCEPTION
...
END;
/
```

# Exemple 1 de procédure stockée inscription d'un étudiant



SQL

```
CREATE PROCEDURE inscription (pnom etudiant.nom%TYPE,
... ,pdip diplome.idDip%TYPE)
AS
CURSOR uv_ins IS SELECT c.iduv AS uv FROM composition c
WHERE c.idDip=pdip;
BEGIN
DBMS_OUTPUT.PUT_LINE('Début inscription de ||pnom');
INSERT INTO etudiant VALUES(seqEtu.NEXTVAL,pnom,...,pdip);
FOR uv_1 IN uv_ins LOOP
INSERT INTO inscrire VALUES(seqEtu.CURRVAL,uv_1.uv);
END LOOP;
DBMS_OUTPUT.PUT_LINE('Transaction réussie');
COMMIT;
EXCEPTION
.....
END;
/
```

# Exemple 1 : appel de la procédure



SQL

- A partir de sqlplus

```
ACCEPT vnom PROMPT 'Entrer le nom : '
.....
EXECUTE inscription('&vnom',....., '&vdip');
```

- A partir de PL/SQL

```
inscription(nom,....., dip);
```

- A partir de pro\*c

```
EXEC SQL EXECUTE
BEGIN
    inscription(:nom, ..... ,:dip);
END;
END-EXEC;
```

## Exemple 2 : avec retour de valeurs suppression d'un étudiant



SQL

```
CREATE PROCEDURE suppression (pidEtu NUMBER,  
                           retour OUT NUMBER) AS  
  inscriptions EXCEPTION;  
  PRAGMA EXCEPTION_INIT(inscriptions,-2292);  
  vnom etudiant.nom%TYPE;  
  BEGIN  
    SELECT nom INTO vnom FROM etudiant WHERE idEtu=pidEtu;  
    DELETE FROM etudiant WHERE idEtu=pidEtu;  
    DBMS_OUTPUT.PUT_LINE('Etudiant'||vnom||' supprimé');  
    COMMIT;  
  retour:=0;
```

.../...

## Exemple 2 : avec retour de valeurs suppression d'un étudiant (suite)



SQL

```
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Etudiant' || TO_CHAR(pidEtu) || 'inconnu');
retour:=1;
WHEN inscriptions THEN
DBMS_OUTPUT.PUT_LINE('Encore des inscriptions');
retour:=2;
.....
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE(SQLERRM);
retour:=9;
END;
/
```

## Exemple 2 : appel avec retour



SQL

```
VARIABLE ret NUMBER  
ACCEPT vnom PROMPT 'Entrer le nom : '  
.....  
EXECUTE inscription('&vnom',....., '&vdip', :ret);  
PRINT ret
```

# Les Fonctions stockées



SQL

- **Comme une procédure mais qui ne retourne qu'un seul résultat**
- **Même structure d'ensemble qu'une procédure**
- **Utilisation du mot clé RETURN pour retourner le résultat**
- **Appel possible à partir de :**
  - **Une requête SQL normale**
  - **Un programme PL/SQL**
  - **Une procédure stockée ou une autre fonction stockée**
  - **Un programme externe comme pro\*c**

# Déclaration d'une fonction stockée



SQL

```
CREATE [OR REPLACE] FUNCTION nom_fonction  
[ (paramètre1 type1, ..... , paramètren typen) ]  
RETURN type_résultat IS  
-- déclarations de variables, curseurs et exceptions  
BEGIN  
-- instructions PL et SQL  
  
RETURN(variable);  
END;  
/
```

1 ou plusieurs RETURN

# Exemple 1 de fonction stockée



SQL

```
CREATE OR REPLACE FUNCTION moy_points_marques
(eqj joueur.ideq%TYPE)
RETURN NUMBER IS
moyenne_points_marques NUMBER(4,2);
BEGIN
SELECT AVG(totalpoints) INTO moyenne_points_marques
FROM joueur WHERE ideq=eqj;
RETURN(moyenne_points_marques);
END ;
/
```

# Utilisation d'une fonction



SQL

- A partir d'une requête SQL

```
SELECT moy_points_marques('e1') FROM dual;
```

```
SELECT nomjoueur FROM joueur WHERE  
totalpoints > moy_points_marques('e1');
```

- A partir d'une procédure ou fonction

```
BEGIN  
.....  
IF moy_points_marques(equipe) > 20 THEN .....  
END ;
```

## Exemple 2 de fonction stockée



SQL

```
CREATE OR REPLACE FUNCTION bon_client  
(pidclient NUMBER, pchiffre NUMBER)  
RETURN BOOLEAN IS  
total_chiffre NUMBER;  
BEGIN  
SELECT SUM(qte*prix_unit) INTO total_chiffre  
FROM commande WHERE idclient=pidclient;  
IF total_chiffre > pchiffre THEN  
    RETURN(TRUE);  
ELSE    RETURN(FALSE);  
END IF;  
END;
```



```
BEGIN
```

```
.....
```

```
IF bon_client(client,10000) THEN .....
```

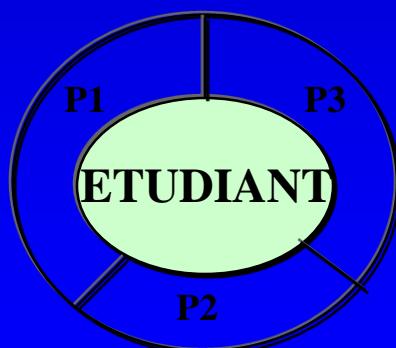
```
.....
```

# Les Paquetages



SQL

- Ensemble de programmes ayant un lien logique entre eux
- Exemple : package étudiant qui peut regrouper tous les programmes écrits sur les étudiants
- Début de l'approche objet avec les méthodes associées à une classe (MEMBER en Objet-Relationnel)



# Structure d'un paquetage



SQL

- Partie ‘visible’ ou spécification
  - Interface accessible au programme appelant
  - Ne contient que les déclarations des procédures ou fonctions publiques
  - Variables globales et session
  - Curseurs globaux
- Partie ‘cachée’ ou body
  - Corps des procédures ou des fonctions citées dans la partie spécification
  - Nouvelles procédures ou fonctions privées accessibles uniquement par des procédures ou fonctions du paquetage

# Déclaration d'un paquetage partie spécification



SQL

```
-- Partie Spécification

CREATE [OR REPLACE] PACKAGE nom_package AS
Procédure1(liste des paramètres);
.....
Fonction1(liste des paramètres);
.....
Variable_globale1 type1;
.....
CURSOR Curseur_global1 IS .....
.....
END nom_package;
/
```

# Déclaration d'un paquetage partie body



SQL

```
-- Partie body
CREATE [OR REPLACE] PACKAGE BODY nom_package AS
Procédure1(liste des paramètres) IS
.....
BEGIN
.....
END Procédure1;
Fonction1(liste des paramètres) RETURN type IS
.....
BEGIN
.....
RETURN(...);
END Fonction2;
END nom_package;
/
```

# Exemple : package 'étudiant' (1)



```
CREATE PACKAGE etudiant AS
-- Procédure publique inscription
PROCEDURE inscription (pnom etudiant.nom%TYPE,
... ,pdip diplome.idDip%TYPE);
-- Procédure publique suppression
PROCEDURE suppression(pidetu NUMBER);
END nom_package;
/
CREATE PACKAGE BODY etudiant AS
inscription (pnom etudiant.nom%TYPE,
... ,pdip diplome.idDip%TYPE) IS
CURSOR uv_ins IS SELECT c.iduv AS uv FROM
composition c
WHERE c.idDip=pdip;
BEGIN
```

## Exemple : package 'étudiant' (2)



SQL

```
INSERT INTO etudiant VALUES(seqEtu.NEXTVAL,pnom,...,pdip);
FOR uv_1 IN uv_ins LOOP
INSERT INTO inscrire VALUES(seqEtu.CURRVAL,uv_1.uv);
END LOOP;
DBMS_OUTPUT.PUT_LINE('Transaction réussie');
COMMIT;
EXCEPTION    ....
END inscription;
-- fonction privée inscrit_uv
FUNCTION inscrit_uv(pidetu NUMBER) RETURN BOOLEAN IS
nbre_ins NUMBER;
BEGIN
SELECT COUNT(*) INTO nbre_ins FROM inscrire WHERE
Idetu=pidetu;
IF nbre_ins>0 THEN RETURN(TRUE) ELSE RETURN(FALSE)
END IF;
END inscrit_uv;
```

## Exemple : package 'étudiant' (3)



SQL

```
PROCEDURE suppression (pidetu NUMBER) AS
BEGIN
IF inscrit_uv(pidetu) THEN
DBMS_OUTPUT.PUT_LINE('Cet étudiant est inscrit à des UV');
DBMS_OUTPUT.PUT_LINE('Impossible de le supprimer');
ELSE
DELETE FROM etudiant WHERE idetu=pidetu;
DBMS_OUTPUT.PUT_LINE('Etudiant supprimé');
COMMIT;
END IF;
END suppression;
END etudiant
```

# Appel d'un programme d'un package



SQL

- A partir des SQL

```
ACCEPT vnom PROMPT 'Entrer le nom : '  
.....  
EXECUTE etudiant.inscription('&vnom',....., '&vdip');
```

- A partir d'un autre package

```
etudiant.inscription(nom,....., dip);
```

- Uniquement les programmes PUBLICS