

Le langage SQL

- I. **Introduction à SQL et au SGBD Oracle**
- II. Interrogation des données (SELECT)
- III. Mise à jour des données (INSERT, UPDATE, DELETE)
- IV. Définition des données (CREATE TABLE)
- V. Les vues (CREATE VIEW)
- VI. Notion de transaction
- VII. Notion de privilèges

I. Introduction à SQL et à Oracle

Généralités sur le langage SQL

- SQL = Structured Query Language
- SQL : langage pour les BDR basé sur l'algèbre relationnelle
- SQL est un standard ANSI/ISO depuis 1986
 - Version SQL-92 ou SQL2 (standard bien supporté)
 - Version SQL-99 ou SQL3 (dernier standard, peu supporté)
- Le SQL d'Oracle est proche de la norme
 - Oracle 8.i : Compatible SQL2
 - > Oracle 9.i : Compatible SQL3
 - Version actuelle : Oracle 11g

I. Introduction à SQL et à Oracle

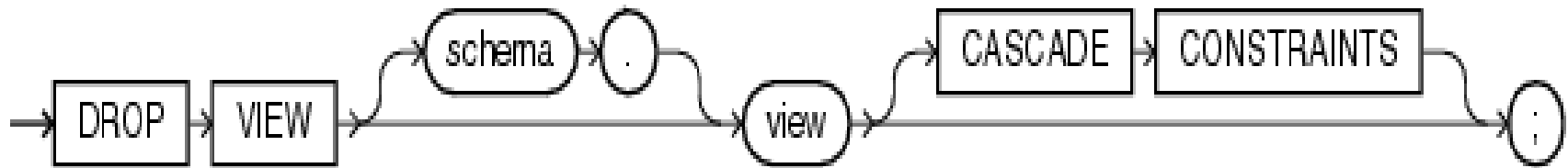
Généralités sur le langage SQL

SQL comporte trois parties :

- **LDD** (Langage de Définition de Données) pour créer, modifier et supprimer des définitions de tables (schémas)
 - Commandes : **create table, drop table, alter table**
- **LMD** (Langage de Manipulation de Données) pour rechercher, ajouter, supprimer, modifier les données
 - Commandes : **select, insert, delete, update**
 - Valider/annuler les mise à jour : **commit, rollback**
- **LCD** (Langage de Contrôle de Données) pour gérer les protections d'accès.
 - Commandes : **grant, revoke**

Notations pour la syntaxe dans la documentation ORACLE (format HTML)

drop_view::=



[Description of the illustration drop_view.gif](#)

DROP VIEW [schema.] view [CASCADE CONSTRAINTS] ;

Quelques précisions :

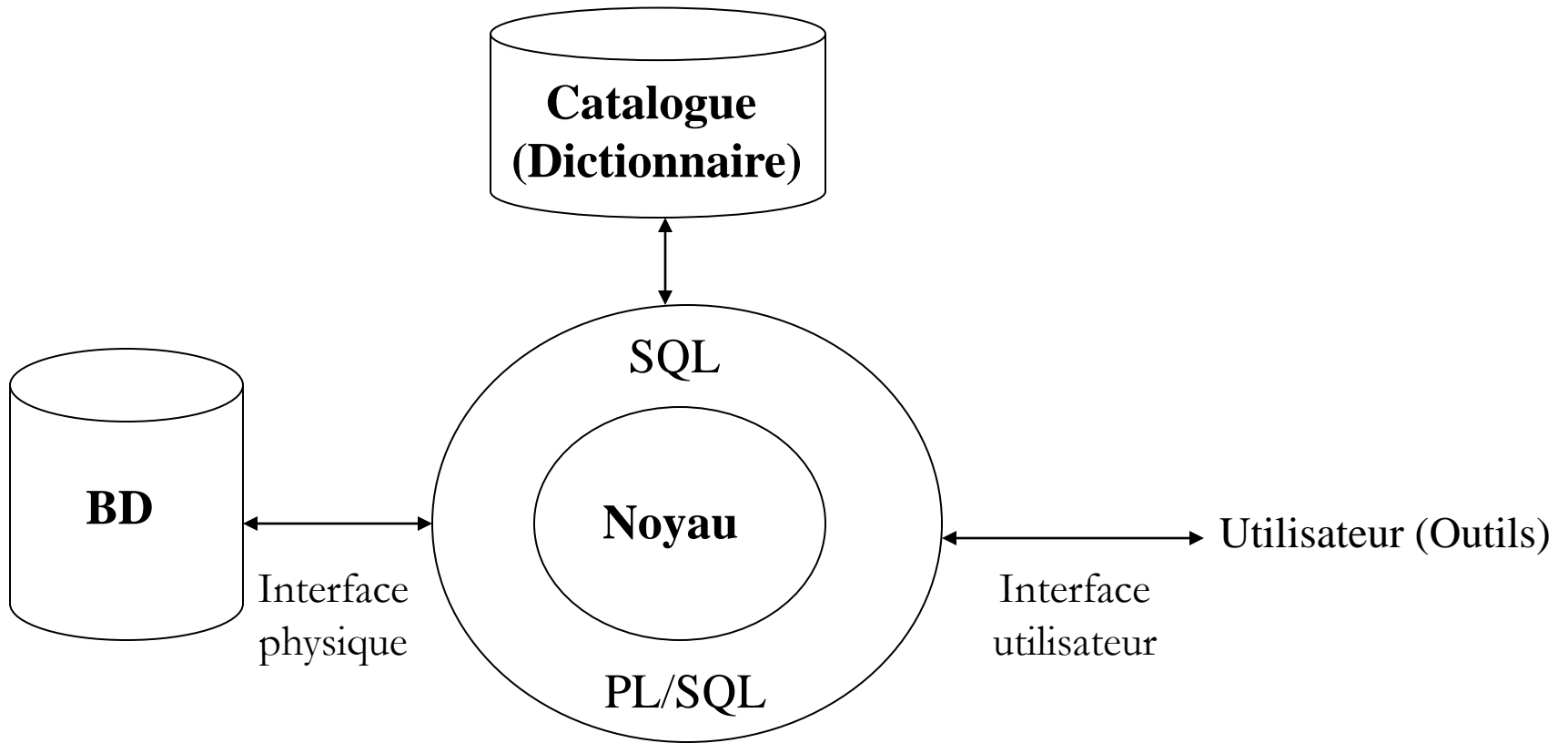
GGG : élément invariable (dans un rectangle)

ggg : paramètre ou élément variable (dans un ovale)

Signes de ponctuation (, ; ...) : dans un cercle

[ggg] -> élément optionnel

Architecture d'Oracle data server



SQL*plus : interface en ligne de commande pour soumettre des requêtes SQL ou PL/SQL à Oracle

Quelques concepts Oracle

- Serveur de données Oracle = une base + une *instance*
- *Instance* :
 - {processus d'arrière-plan}
 - Zone globale système (SGA) : tampons de données/journal partagés
- Dictionnaire de données : contient des informations sur les différents éléments d'une BD
 - Ensemble de tables (et de vues)
 - Créé dans chaque base (espace SYSTEM)
 - Appartient à sys (non modifiable, sauf sys.aud\$)

Dictionnaire de données (catalogue)

- Vues de la forme **USER_xxx**
 - Informations sur les objets dont l'utilisateur (courant) est propriétaire
- Vues de la forme **ALL_xxx**
 - Informations sur les objets auxquels a accès l'utilisateur
- Vues de la forme **DBA_xxx**
 - Informations sur les objets de tous les utilisateurs
- Principales vues du dictionnaire
 - USER_OBJECTS, ALL_OBJECTS, DBA_OBJECTS
 - USER_TABLES, ALL_TABLES, DBA_TABLES
 - USER_TAB_COLUMNS...
 - USER_TAB_COMMENTS
 - USER_VIEWS...
- Vue **DICTIONARY** : liste des tables et des vues du dictionnaire

Oracle-SQL et la casse des caractères (majuscule/minuscule)

- Il n'y a pas de différence entre des noms (ou identifiants) écrits en majuscules ou en minuscules
 - ex. noms d'attributs, de tables, de contraintes...
- Les noms sont stockés en majuscules dans le catalogue
- Seul cas où la casse est prise en compte : la comparaison de chaînes de caractères dans les données

SELECT	*		Select	*
FROM	Client	⇔	From	CLIENT
WHERE	nom = 'Dupont'		where	NOM = 'Dupont'

Quelques règles de nommage des objets dans Oracle

- Dénomination d'un objet = choix d'un nom pour désigner cet objet
- Un nom d'objet Oracle doit respecter certaines règles :
 - 1 à 30 caractères
 - pas de distinction majuscule/minuscule
 - le premier caractère est une lettre
 - caractères possibles :
 - caractères alphanumériques (a-z, A-Z, 0-9)
 - 3 caractères spéciaux permis : _ \$ #
 - ne doit pas être un mot réservé du langage SQL
 - doit être unique dans le schéma d'un utilisateur pour le même type d'objets

Nommage des relations/attributs dans Oracle

■ Propriétaire d'un objet = utilisateur l'ayant créé

■ Nommage d'une **relation**

- *nom-de-la-relation* si elle m'appartient
- *nom-du-propiétaire.nom-de-la-relation* sinon
- Possibilité de créer des synonymes

```
create synonym nom for toto.produit
```

■ Nommage d'un **attribut** dans une requête

- si pas d'ambiguïté : *nom-attribut*
- sinon : *nom-de-relation.nom-attribut*
- ou : *alias-de-relation.nom-attribut*

Le langage SQL (*SGBD ORACLE*)

- I. Introduction à SQL et à Oracle
- II. Interrogation des données**
- III. Mise à jour des données
- IV. Définition des données
- V. Les vues
- VI. Notion de transaction
- VII. Notion de privilèges

Base de données exemple pour le cours (1/2)

Client (noClient, nom, prénom, ddn, rue, CP, ville)

Produit (noProduit, libellé, prixUnitaire, *noFournisseur*)

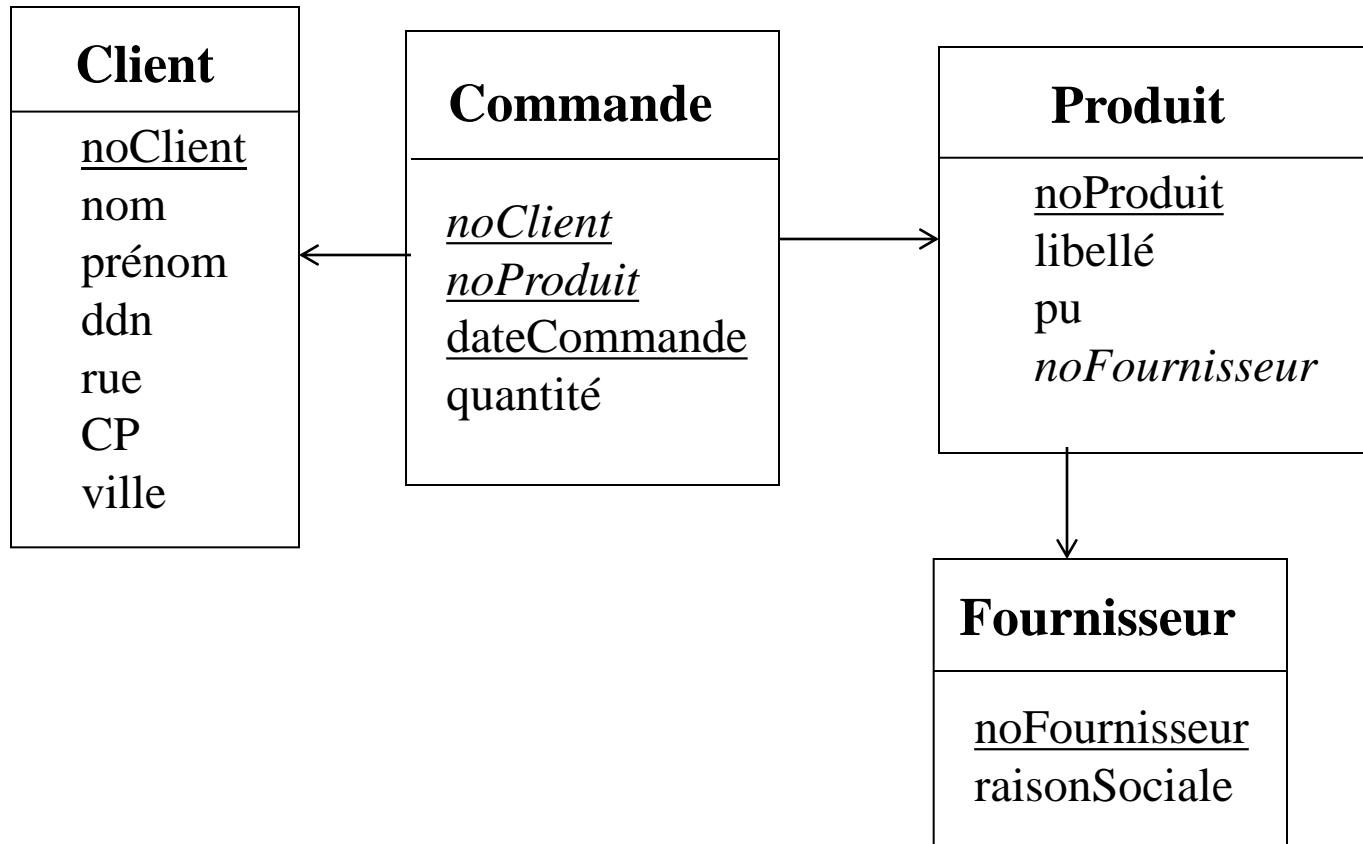
Fournisseur (noFournisseur, raisonSociale)

Commande (*noClient*, *noProduit*, dateCommande, quantité)

Clés primaires soulignées

Clés étrangères en italique

Base de données exemple pour le cours (2/2)



Syntaxe générale de la commande SELECT

```
SELECT * | { [ALL | DISTINCT]
            expression [[AS] nomColonne]
            [, expression [[AS] nomColonne]]... }

FROM  relation [alias] [, relation [alias] ...]
[WHERE condition]
[GROUP BY nomColonne [, nomColonne]...]
[HAVING condition]
[ORDER BY  nomColonne [ASC | DESC]
            [, nomColonne [ASC | DESC] ...]
```

Si CommandeSelect1 et CommandeSelect2 sont des commandes SELECT :

```
CommandeSelect1 {UNION | INTERSECT | EXCEPT} CommandeSelect2
```

Projection d'une relation et la clause **DISTINCT**

- Trouver les *noClient* et *dateCommande* de toutes les *Commandes*

```
SELECT      noClient, dateCommande
FROM        Commande
```

noClient	dateCommande
100	04-MAY-03
200	12-APR-03
100	05-JAN-04
300	25-FEB_04
400	30-JAN-04
400	30-JAN-04

Commande équivalente :

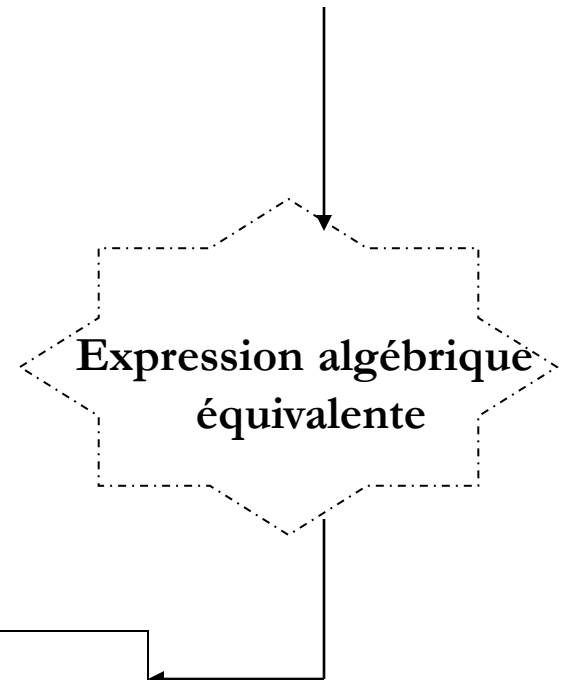
```
SELECT      ALL noClient, dateCommande
FROM        Commande
```

La clause **DISTINCT**

- Trouver les *noClient* et *dateCommande* de toutes les *Commandes*

```
SELECT      DISTINCT noClient, dateCommande
FROM        Commande
```

noClient	dateCommande
100	04-MAY-03
200	12-APR-03
100	05-JAN-04
300	25-FEB-04
400	30-JAN-04



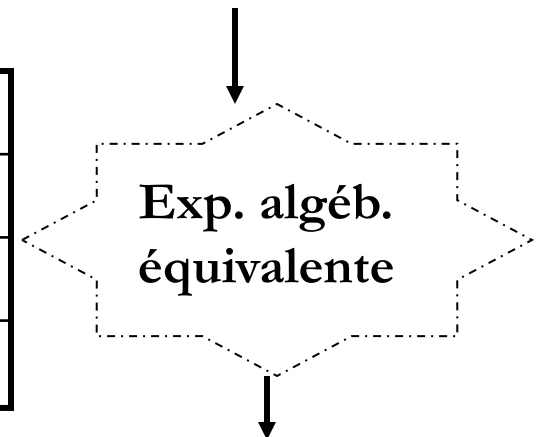
$\pi_{\{noClient, dateCommande\}}(Commande)$

Restriction d'une relation sur une condition

- *Sélectionner les Articles dont le prix est inférieur à 20€ et le numéro est supérieur à 30*

```
SELECT      *  
FROM        Article  
WHERE       prixUnitaire < 20 AND noArticle > 30
```

noArticle	libellé	prixUnitaire
31	Brosse	12
40	Tableau	9.99
50	Visseuse	19.99

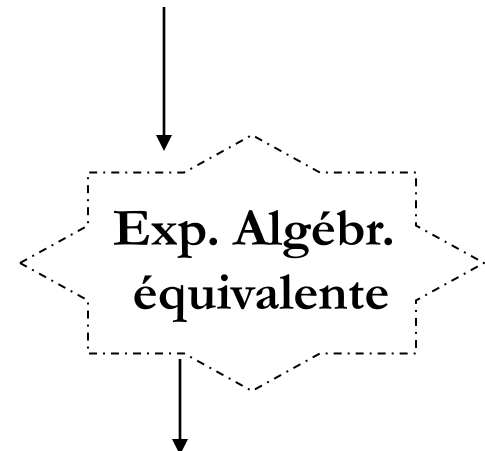

$$\sigma_{\text{prixUnitaire} < 20 \wedge \text{noArticle} > 30} (\text{Article})$$

Restriction et projection

- *Produire les noClient et dateCommande des Commandes dont la date est postérieure au 01/01/2004*

```
SELECT    noClient, dateCommande
FROM      Commande
WHERE     dateCommande > '01-JAN-04'
```

noClient	dateCommande
100	05-JAN-04
300	25-FEB-04
400	30-JAN-04


$$\pi_{\{noClient, dateCommande\}} (\sigma_{dateCommande > '01/01/2004'} (Commande))$$

Restriction : syntaxe d'une condition

SELECT * FROM nom-relation

WHERE condition

- Sélectionne les tuples de la relation pour lesquels condition est évaluée à Vrai (True)

- Syntaxe de condition

conditionSimple | (condition) | NOT (condition) |
condition {AND|OR} condition

- Syntaxe de conditionSimple

expression {= | < | > | <= | >= | <> | !=} expression |
expression [NOT] BETWEEN expression AND expression |
expression IS [NOT] NULL
expression [NOT] IN listeConstantes |
expression [NOT] LIKE patron

Notion de valeur indéfinie (NULL)

Et évaluation d'expressions

- Valeur d'un attribut inconnue ou indéfinie notée **NULL** dans l'extension d'une relation
- Seules opérations permises sur la valeur NULL
 - IS NULL
 - IS NOT NULL
- Expression arithmétique : Si un opérande est NULL, le résultat de l'évaluation est NULL
- Expression de comparaison ou condition simple : Si un opérande est NULL, le résultat de l'évaluation est NULL (UNKNOWN)
- Condition avec connecteurs logiques : cf. tables de vérité étendues

Notion de valeur indéfinie (NULL) : Extension des tables de vérité traditionnelles

a	b	a AND b	a OR b	NOT a
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	NULL	NULL	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	NULL	FALSE	NULL	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE
NULL	TRUE	NULL	TRUE	NULL
NULL	NULL	NULL	NULL	NULL
NULL	FALSE	FALSE	NULL	NULL

a, b : expressions booléennes à valeurs dans {TRUE, FALSE, NULL}

Restriction : Exemples

- *Produits dont le prix est compris entre 50 et 100€*

```
SELECT  *  
FROM    Produit  
WHERE   prixUnitaire >= 50  AND  prixUnitaire <= 100
```

```
SELECT  *  
FROM    Produit  
WHERE   prixUnitaire BETWEEN 50  AND 100
```

- *Produits dont le prix est inférieur à 50€ ou supérieur à 100 €*

```
SELECT  *  
FROM    Produit  
WHERE   prixUnitaire < 50  OR  prixUnitaire > 100
```

Restriction : Opérateurs IS NULL et LIKE

■ *Commandes en quantité indéterminée (null)*

```
SELECT      *  
FROM        Commande  
WHERE       quantité IS NULL
```

■ *Clients dont le nom commence par B, se termine par B et contient au moins 3 caractères*

```
SELECT      *  
FROM        Client  
WHERE       nom LIKE 'B_%B'
```

LIKE recherche des chaînes de caractères correspondant à un *patron* où :

% : désigne une suite de **zéro à n** caractères quelconques

_ : désigne **un et un seul** caractère quelconque

Possibilité de définir un caractère d'échappement :

```
SELECT dname FROM dept WHERE dname LIKE 'A\%S%' ESCAPE '\'
```

Restriction : Opérateur IN

- *Prénom des clients dont le nom est Dupont, Durant ou Martin*

```
SELECT    prénom
FROM      Client
WHERE     nom IN ('Dupond', 'Durant', 'Martin')
```

- *Prénom des clients dont le nom n'est pas dans l'ensemble {Dupont, Durant, Martin}*

```
SELECT    prénom
FROM      Client
WHERE     nom NOT IN ('Dupond', 'Durant', 'Martin')
```


Expression de calcul sur les colonnes dans la liste de projection

- *Liste des noArticle avec le prixUnitaire avant et après inclusion d'une taxe de 15%.*

```
SELECT noArticle, prixUnitaire,  
       prixUnitaire*1.15 as prixTTC  
FROM   Article
```

noArticle	prixUnitaire	prixTTC
10	10.99	12.64
20	12.99	14.94
40	25.99	29.89
50	22.99	26.44
60	15.99	18.39
70	10.99	12.64
80	26.99	31.04
81	25.99	29.89
90	25.99	29.89
95	15.99	18.39

Expression de calcul sur les colonnes dans la condition (du WHERE)

- Une condition peut comporter une expression de calcul

*ex: Chercher les tuples de T pour lesquels $X < Y^{**3} + Z/40$*

```
SELECT *  
FROM T  
WHERE X < Y**3+Z/40
```

- Une expression peut aussi faire appel à des **fonctions**

ex: Liste des commandes de la journée

```
SELECT *  
FROM Commande  
WHERE dateCommande = CURRENT_DATE
```

Liste des principales fonctions sous Oracle (1/2)

- `ABS(n)` : Valeur absolue
- `CEIL(n)` : plus petit entier $\geq n$
- `FLOOR(n)` : plus grand entier $\leq n$
- `MOD(m,n)` : Reste de la division euclidienne de m par n
- `POWER(m,n)` : m élevé à la puissance n
- `SIGN(n)` : Signe de n
- `SQRT(n)` : Racine carrée de n
- `INITCAP(ch)` : Première lettre en majuscule
- `LOWER(ch)` : Toutes les lettres en minuscules
- `UPPER(ch)` : Toutes les lettres en majuscules
- `LTRIM(chaine[,ens_car])` : Suppression de caractères par la gauche

- `RTRIM(chaine[,ens_car])` : Suppression de caractères par la droite
- `REPLACE(ch,car[,ch])` : Remplacement de caractères
- `SUBSTR(ch,position[,long])` : Extraction d'une chaîne
- `TRANSLATE(ch,car1,car2)` : Transcodage
- `SOUNDEX(ch)` : Comparaison phonétique
- `LPAD(ch,lg[,car])` : Compléter par la gauche
- `RPAD(ch ,lg[,car])` : Compléter par la droite
- `ASCII(ch)` : Donne la valeur ASCII
- `INSTR(ch,sous_ch(,position(n)))` : Recherche la sous-chaîne dans la chaîne

Liste des principales fonctions sous Oracle (2/2)

- `LENGTH(ch)` : Longueur de chaîne
- `ADD_MONTHS(TO_DATE(date),n)` : Addition de mois
- `LAST_DAY(TO_DATE(date))` : Dernier jour du mois
- `MONTHS_BETWEEN(TO_DATE(date1), TO_DATE(date2))` : Nombre de mois entre deux dates
- `NEXT_DAY(TO_DATE(date),jour)` : Date du prochain jour
- `SYSDATE` : Date et heure système
- `ROUND(TO_DATE(date)[,précision])` : Arrondi d'une date
- `TRUNC(TO_CHAR(date)[,précision])` : Troncature d'une date
- `TO_NUMBER(ch)` : Conversion d'une chaîne en nombre

- `TO_CHAR(exp[, 'format'])` : Conversion d'une expression en chaîne
- `TO_CHAR(date, 'mm')` : extrait le mois de date ('dd' pour le jour, 'yyyy' l'année)
- `TO_DATE(ch[, 'format'])` : Conversion d'une chaîne en date
- `NVL(expr,valeur)` : Protège de la manipulation d'une valeur NULL
`NVL (exp1, exp2)` : retourne `exp1` si `exp1` n'est pas null et `exp2` sinon. `exp1` et `exp2` doivent être des chaînes de caractères.
- `DECODE(expr,valeur1,résultat1[,valeur2,résultat2 ...],default)` : Incursion procédurale dans SQL ...
- `GREATEST(n1,n2,...)` : Le plus grand
- `LEAST(n1,n2,...)` : Le plus petit de la liste
- `VSIZE(expression)` : Nombre d'octets nécessaires pour le format interne
- `UID` : Identifiant numérique de l'utilisateur
- `USER` : Nom de l'utilisateur

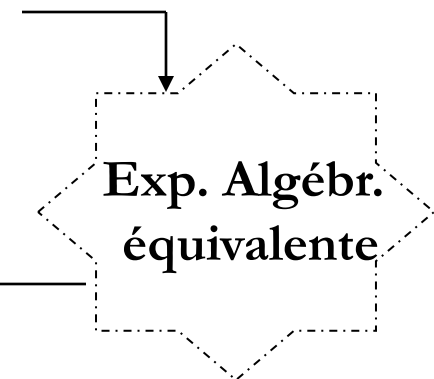
Produit cartésien

```
SELECT      *  
FROM relation1, relation2
```

Ex. Produire toutes les combinaisons possibles de Client et de Commande

```
SELECT      *  
FROM        Client, Commande
```

Client X Commande



Jointure(s) implicite(s)

```
SELECT  attribut1 [,attribut2, ...]  
FROM    relation1,relation2 [,relation3,...]  
WHERE   condition
```

Cette commande SELECT combine **produit cartésien**,
restriction et **projection**



Exp. Algèbr.
équivalente

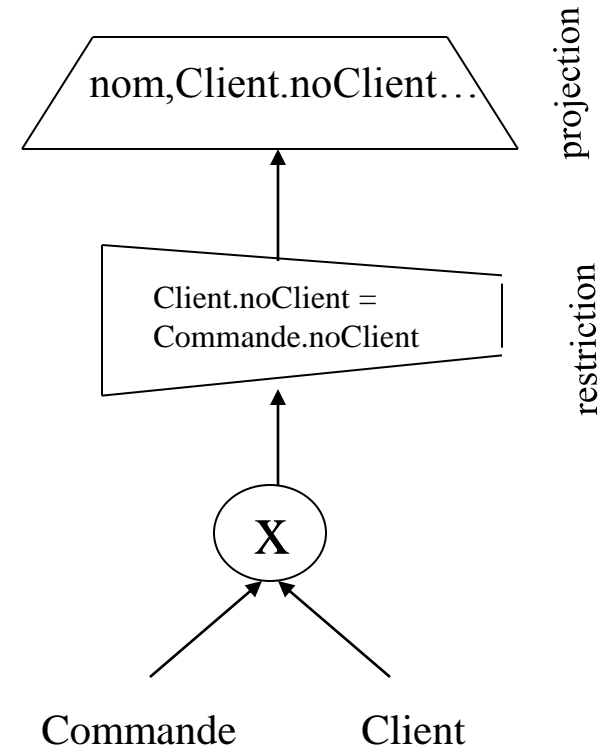
$$\pi_{\{attribut1, attribut2...\}} (\sigma_{condition} ((relation1 \times relation2) \times relation3) \dots)$$

N.B. Nécessité de préfixer le nom d'un attribut par sa relation
en cas d'ambiguïté

Jointure implicite : exemple de requête

ex. *Liste des commandes avec le nom du client*

```
SELECT    nom, Client.noClient,  
noProduit,dateCommande,quantité  
FROM      Commande, Client  
WHERE     Client.noClient =  
          Commande.noClient
```

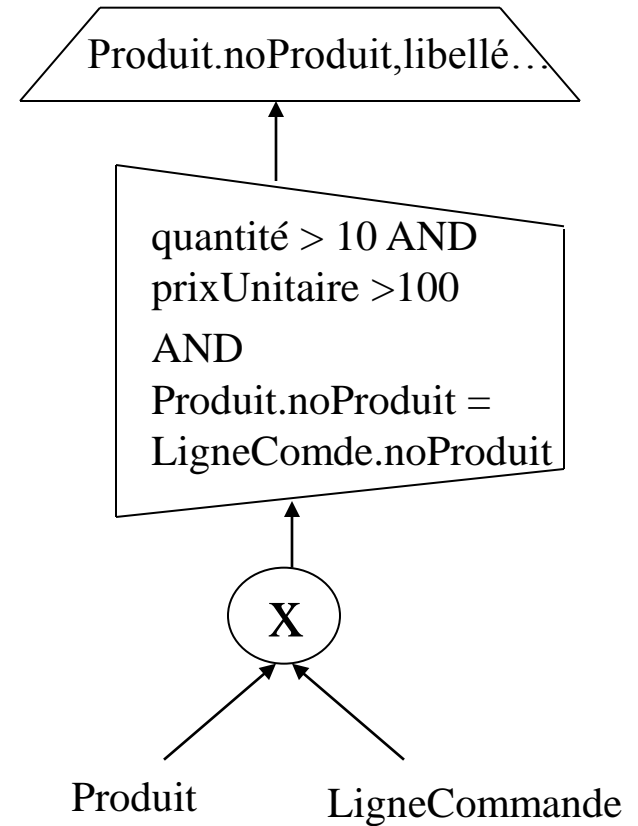


**Arbre algébrique canonique
correspondant à l'expression
SQL**

Jointure implicite : autre exemple

ex. *Produits commandés en quantité supérieure à 10 et dont le prix dépasse 100 €. Afficher les numéros de produit, leur libellé, leur prix unitaire ainsi que numéro de la commande.*

```
SELECT  Produit.noProduit, libellé,  
        prixUnitaire, noCommande  
FROM    Produit, LigneCommande  
WHERE   quantité > 10  
AND     prixUnitaire >100  
AND     Produit.noProduit =  
        LigneCommande.noProduit
```



Arbre algébrique canonique

Utilisation d'alias

Utilisation d'**alias** pour alléger l'écriture d'une requête utilisant plusieurs tables

ex. *Liste des commandes avec le nom et le numéro du client*

```
SELECT    C2.noClient, nom, date, quantité  
FROM      Commande C1 , Client C2  
WHERE     C1.noClient = C2.noClient
```

- *Commande* alias *C1*
- *Client* alias *C2*

Jointures explicites

Jointure sur égalité d'attribut(s) de même nom

```
SELECT liste-projection
FROM   Table1 JOIN Table2 USING (attr,attr...)
      [JOIN Relation3 USING (attr, attr...)... ]
[WHERE condition]
```

Jointure sur condition (égalité d'attributs de noms différents)

```
SELECT liste-projection
FROM   Table1 JOIN Table2 ON (condition1)
      [JOIN relation3 ON (condition2) ... ]
[WHERE condition]
```

Jointure externe (gauche ou droite)

```
SELECT liste-projection
FROM   Table1 [LEFT|RIGHT] JOIN Table2
      {ON (condition) | USING (attr, attr...))}
[WHERE condition]
```

Exemples de Jointures explicites

- ```
SELECT *

FROM Commande JOIN Client USING (noClient)
```
  
- ```
SELECT      *  
  
FROM        Film JOIN Programmation USING (titre)  
            JOIN Salle USING (nom_cinema, noSalle)  
            JOIN Cinema USING (nom_cinema)  
  
WHERE        realiseur like 'Barn%'
```

- ```
SELECT p1.nom, p1.prenom, p2.nom as "nom du père", p2.prenom
FROM Personne p1 JOIN Personne p2 ON (p1.id_pere = p2.id)

WHERE p1.statut = 'Etudiant'
```

# Opérations ensemblistes : UNION, INTERSECT, EXCEPT

- Sous ORACLE, **MINUS** au lieu de **EXCEPT**
- *Trouver les noms et prénoms des employés qui sont aussi des passagers*

Employé

| noEmployé | nomEmp  | prénomEmp |
|-----------|---------|-----------|
| 10        | Henry   | John      |
| 15        | Conrad  | James     |
| 35        | Jenqua  | Jessica   |
| 46        | Leconte | Jean      |

Passager

| noPassager | nomPass | prénomPass |
|------------|---------|------------|
| 4          | Harry   | Peter      |
| 78         | Conrad  | James      |
| 9          | Land    | Robert     |
| 466        | Leconte | Jean       |

```
(SELECT nomEmp as nom, prénomEmp as prénom
FROM Employé)
```

**INTERSECT**

```
(SELECT nomPass as nom, prénomPass as prénom
FROM Passager)
```

| nom     | prénom |
|---------|--------|
| Conrad  | James  |
| Leconte | Jean   |

# Fonctions d'agrégation

Une fonction d'agrégation opère sur les valeurs d'un attribut d'une relation et produit une valeur résultat unique (extension de l'algèbre relationnelle)

```
SELECT fctAgrégation
FROM relation(s) [WHERE condition]
```

`fctAgrégation` opère sur les lignes de la relation résultat (de la commande) où :

- ✓ **COUNT (\*)** : retourne le nombre de tuples de la relation résultat
- ✓ **COUNT** ([distinct] A) : nombre de valeurs non NULL (distinctes) de la colonne A
- ✓ **MAX** (A) : plus grande valeur de la colonne N
- ✓ **MIN** (N) : plus petite valeur de la colonne N
- ✓ **SUM** ([distinct] N) : somme des valeurs (distinctes) de la colonne N (ignore les valeurs NULL)
- ✓ **AVG** ([distinct] N) : moyenne des valeurs (distinctes) de la colonne N (ignore les valeurs NULL)

Où **N** est un attribut numérique et **A** un attribut quelconque

# Fonctions d'agrégation

*Nombre total d'articles et prix unitaire moyen*

```
SELECT COUNT(*) AS nbArticles,
 AVG (prixUnitaire) AS prixMoyen
FROM Article
```

| nbArticles | prixMoyen |
|------------|-----------|
| 15         | 9.50      |

*Nombre de prixUnitaires non null*

```
SELECT COUNT(prixUnitaire) AS nbPrixNonNull
FROM Article
```

| nbPrixNonNull |
|---------------|
| 15            |

*Nombre de prixUnitaires non null différents*

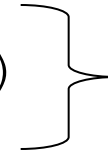
```
SELECT COUNT(DISTINCT prixUnitaire) AS nbPrix
FROM Article
```

| nbPrix |
|--------|
| 8      |

# Fonctions d'agrégation : Contraintes d'utilisation

- Une fonction d'agrégation doit être utilisée dans une clause SELECT sans résultats individuels

```
SELECT noProduit, max(prixUnitaire)
FROM Produit
```



**Faux !!**

Requête **invalid**e puisque plusieurs noProduit et un seul maximum.

- Une fonction d'agrégation peut être utilisée dans une sous-requête  
⇒ sélection de résultats individuels dans la requête englobante

```
SELECT noProduit, libellé
FROM Produit
WHERE prixUnitaire =
 (SELECT max (prixUnitaire)
 FROM Produit)
```

# Groupement de relations (GROUP BY)

- Par défaut, 1 relation forme 1 groupe
- Il est possible de partitionner une relation en groupes selon les valeurs de une ou plusieurs colonnes (attributs) : GROUP BY

ex. *Nombre de produits commandés par client*

```
SELECT noClient, COUNT(*) AS totalProduits
FROM Commande
GROUP BY noClient
```

- 1) Les commandes sont groupées par numéro de client
- 2) pour chaque groupe, afficher le numéro du client concerné par le groupe et le nombre de commandes.

*N.B.* : chaque expression du SELECT doit avoir une valeur **unique** par **groupe**.



# Grouperment de relations (GROUP BY)

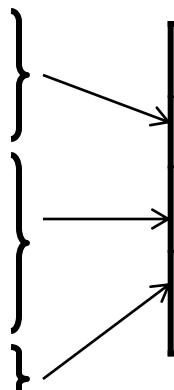
ex. *Nombre de produits commandés par client*

```
SELECT noClient, COUNT(*) AS totalProduits
FROM Commande
GROUP BY noClient
```

| noProduit | dateCommande | noClient |
|-----------|--------------|----------|
| 4         | 05-JAN-03    | 10       |
| 5         | 05-JAN-03    | 10       |
| 20        | 14-MAY-03    | 12       |
| 28        | 15-AUG-03    | 12       |
| 68        | 15-AUG-03    | 12       |
| 59        | 20-SEP-03    | 15       |

| noClient | totalProduits |
|----------|---------------|
| 10       | 2             |
| 12       | 3             |
| 15       | 1             |



# Grouperement de relations (GROUP BY)

ex. *Quantité totale de produits commandés par client en dehors du produit F565*

```
SELECT noClient, SUM(quantité)
FROM Commande
WHERE noProduit <> 'F565'
GROUP BY noClient
```

- 1) Les tuples de Commande ne vérifiant pas la condition sont exclus
- 2) Les commandes restantes sont groupées par numéro de client
- 3) pour chaque groupe, afficher le numéro du client concerné par le groupe et la somme des quantités.

Une clause **HAVING** permet de restreindre les groupes

# Groupement de relations (GROUP BY)

ex. *Quantité moyenne commandée par produit pour les produits ayant fait l'objet de plus de 3 commandes. Ignorer les commandes concernant le client C47.*

```
SELECT noProduit, AVG(quantité)
FROM Commande
WHERE noClient != 'C47'
GROUP BY noProduit
HAVING COUNT(*) > 3
```

- 1) Les tuples de Commande ne vérifiant pas la condition WHERE sont exclus
- 2) Les commandes restantes sont groupées par numéro de produit
- 3) pour chaque groupe, compter le nombre d'éléments et éliminer les groupes à moins de 3 éléments.
- 4) pour les groupes restants, afficher le numéro de produit et la quantité moyenne.

**N.B. : La clause HAVING ne s'utilise qu'avec un GROUP BY.**

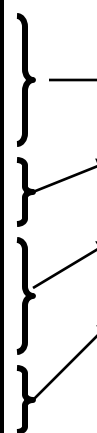
# Grouperment de relations (GROUP BY)

- Il est possible de partitionner sur plusieurs colonnes (attributs)

ex. *Nombre de produits commandés par client et par date*

```
SELECT noClient, dateCommmande,
 COUNT(noProduit) AS nbProduits
FROM Commande
GROUP BY noClient, dateCommmande
```

| noProduit | dateCommmande | noClient |
|-----------|---------------|----------|
| 4         | 05-JAN-03     | 10       |
| 5         | 05-JAN-03     | 10       |
| 20        | 14-MAY-03     | 12       |
| 28        | 15-AUG-03     | 12       |
| 68        | 15-AUG-03     | 12       |
| 59        | 20-SEP-03     | 15       |



| noClient | dateCommmande | nbProduits |
|----------|---------------|------------|
| 10       | 05-JAN-03     | 2          |
| 12       | 14-MAY-03     | 1          |
| 12       | 15-AUG-03     | 2          |
| 15       | 20-SEP-03     | 1          |

# Ordonner les résultats d'une requête (ORDER BY)

- Possibilité de trier les résultats d'une requête par rapport à une ou plusieurs de ses colonnes

```
SELECT colonne(s)
FROM relation(s) [WHERE condition]
ORDER BY colonne [ASC|DESC], [,colonne ASC|DESC]..]
```

Où

**ASC** : ordre ascendant (par défaut)

**DESC** : ordre descendant

*ex. liste des commandes par ordre croissant du numéro de client et par ordre chronologique inverse de la date*

```
SELECT *
FROM Commande
ORDER BY noClient, dateCommande desc
```

# Requêtes imbriquées : opérateur **IN** / **NOT IN** (test d'appartenance à un ensemble)

- Le résultat d'une requête peut être utilisé dans une condition de la clause **WHERE** d'une commande **SELECT** (sous-requête /requête imbriquée)

```
SELECT liste-colonnes-ou-expressions
FROM liste-relations
WHERE
 expression1 [NOT] IN (SELECT expression2
 FROM ... [WHERE ...])
```

- Attention : **expression1** et **expression2** doivent avoir le même type
- **IN** retourne **NULL** si **expression1** est **NULL**
- **NOT IN** retourne **NULL** si le résultat de la sous-requête contient une valeur **NULL**

## Requêtes imbriquées : opérateur IN / NOT IN (test d'appartenance à un ensemble)

*Nom des clients ayant passé commande le 24/10/2000*

```
SELECT nom
FROM Client
WHERE noClient IN (SELECT noClient
 FROM Commande
 WHERE dateCommande='24-OCT-00')
```

## Requêtes imbriquées : opérateur **EXISTS** / **NOT EXISTS** (test d'ensemble vide)

- Le résultat d'une requête peut être utilisé dans une condition de la clause **WHERE** d'une commande **SELECT** (sous-requête / requête imbriquée)

```
SELECT colonne(s)
FROM relation(s)
WHERE
 {EXISTS | NOT EXISTS} (sous-requête)
```

Si le résultat de la sous-requête est non vide alors **EXISTS** retourne **TRUE** sinon retourne **FALSE**



# Requêtes imbriquées : opérateur EXISTS / NOT EXISTS (test d'ensemble vide)

*Clients ayant passé au moins une commande*

```
SELECT *
FROM Client C1
WHERE EXISTS (SELECT *
 FROM Commande C2
 WHERE C1.noClient=C2.noClient)
```

*Clients n'ayant passé aucune commande*

```
SELECT *
FROM Client C1
WHERE NOT EXISTS (SELECT *
 FROM Commande C2
 WHERE C1.noClient=C2.noClient)
```

# Nom d'une colonne dans la relation résultat d'une commande **SELECT**

- Par défaut, nom de l'attribut ou de l'expression dont la colonne est issue

```
SELECT * FROM Produit
```

| noProduit | libellé | prixUnitaire |
|-----------|---------|--------------|
|           |         |              |

```
SELECT AVG(prixUnitaire) FROM Produit
```

- **Renommage possible**

| AVG(prixUnitaire) |
|-------------------|
|                   |

```
SELECT noProduit AS NumeroProduit
FROM Produit
```

| NumeroProduit |
|---------------|
|               |

```
SELECT noProduit AS "Numero Produit"
FROM Produit
```

| Numero Produit |
|----------------|
|                |

# **Le langage SQL (*SGBD ORACLE*)**

- I. Introduction à SQL et à Oracle
- II. Interrogation des données
- III. **Mise à jour des données**
- IV. Définition des données
- V. Les vues
- VI. Notion de transaction
- VII. Notion de privilèges

# Mise à jour des données

## a. Ajout de tuples (INSERT)

```
INSERT INTO nom-relation [(attribut1, attribut2...)]
{VALUES (valeur1[, valeur2, ...])} | {commande-SELECT}
```

*Ajout d'un produit*

```
INSERT INTO Produit VALUES(430, 'lecteur DVD', 9.99)
```

*Ajout du résultat d'un SELECT*

```
INSERT INTO Commande (noClient, noProduit)
 SELECT noClient, noProduit FROM Produit, Client
```

- Les attributs non mentionnés sont positionnés à NULL  
(si pas de valeur par défaut)

# Mise à jour des données

## b. Modification de tuples (UPDATE) (1/2)

```
UPDATE nom_relation
SET attribut1 = {expression1 | NULL | clause-SELECT1}
 [, attribut2 = {expression2 | NULL | clause-SELECT2}
 ...]
[WHERE condition]
```

*Positionner à 'Durand' le nom du client n° 3 :*

```
UPDATE Client SET nom = 'Durand' WHERE noClient=3
```

*Augmenter de 5% le prix des produits dont le libellé est dans une liste :*

```
UPDATE Produit
SET prixUnitaire = prixUnitaire*1.05
WHERE libellé IN ('CD-ROM', 'DVD', 'ZIP')
```

# Mise à jour des données

## b. Modification de tuples (UPDATE) (2/2)

*Augmenter de 10% les prix des produits fournis par le fournisseur 'FFF' :*

```
UPDATE Produit p SET prixUnitaire = prixUnitaire*1.1
WHERE p.noFournisseur IN
 (SELECT f.noFournisseur
 FROM Fournisseur f
 WHERE f.raisonSociale = 'FFF')
```

*Positionner le libellé du produit n° 99 à 'produitTest' et son prix au prix moyen :*

```
UPDATE Produit
SET
libellé = 'produitTest' ,
prixUnitaire = (SELECT AVG(prixUnitaire) FROM Produit)
WHERE noProduit = 99
```

# Mise à jour des données

## c. Suppression de tuples (DELETE)

**DELETE** [**FROM**] relation [**WHERE** condition]

*Supprimer les clients de Metz :*

```
DELETE Client WHERE ville = 'Metz'
```

*Supprimer tous les tuples de la table Client :*

```
DELETE Client
```

- ✓ suppression de **tous** les tuples de la table Client
- ✓ le **schéma** de la table Client existe toujours (dans le dictionnaire)

# **Le langage SQL (*SGBD ORACLE*)**

- I. Introduction à SQL et à Oracle
- II. Interrogation des données
- III. Mise à jour des données
- IV. **Définition des données**
- V. Les vues
- VI. Notion de transaction
- VII. Notion de privilèges



# Définition de données (schémas)

1. Création de schéma de relation (**CREATE TABLE**)
2. Création de schéma + instanciation (**CREATE TABLE ... AS...**)
3. Modification de schéma de relation (**ALTER TABLE**)
4. Suppression de schéma de relation (**DROP TABLE**)
5. Index pour l'accélération des accès (**CREATE/DROP Index**)

# 1. Création de schéma de relation

```
CREATE TABLE Client
 (noClient INTEGER,
 nom VARCHAR2 (50) ,
 prénom VARCHAR2 (15)
)
```

- Transmise à l'interprète du LDD
  - i. vérification
  - ii. création de la table
    - schéma stocké dans le dictionnaire de données
    - allocation des structures physiques

# Schéma et dictionnaire de données Oracle avec SQL\*plus

```
SQL> CREATE TABLE Client
 2 (noCLIENT INTEGER,
 3 nomClient VARCHAR(15),
 4 noTéléphone VARCHAR(15))
 5 /
```

Table créée.

```
SQL> SELECT Table_Name
 2 FROM USER_TABLES
 3 /
```

TABLE\_NAME

-----

CLIENT

```
SQL> SELECT Column_Name, Data_Type
 2 FROM USER_TAB_COLUMNS
 3 WHERE Table_Name = 'CLIENT'
 4 /
```

COLUMN\_NAME

-----

-----

NOCLIENT

NOMCLIENT

NOTÉLÉPHONE

DATA\_TYPE

-----

NUMBER

VARCHAR2

VARCHAR2

# Syntaxe générale du CREATE TABLE (1/2)

```
CREATE TABLE nomTable
 (définitionAttribut
 [, définitionAttribut] ...
 [, définitionContrainte] ...)) ;
```

## ■ Syntaxe de **défini**tionAttribut

```
nomAttribut typeDeDonnées
 [DEFAULT valeur] [NULL|NOT NULL]
 [UNIQUE|PRIMARY KEY]
 [REFERENCES nomTableBis (nomAttributBis)]
 [[CONSTRAINT nomContrainte] CHECK (condition)]
```

*Rappel : Les noms d'attributs ou de tables sont mémorisés en majuscules dans le catalogue*

# Syntaxe générale du CREATE TABLE (2/2)

## ■ Syntaxe de **définitionContrainte**

```
[CONSTRAINT nomContrainte]
{PRIMARY KEY (listeAttributs)} |
{UNIQUE (listeAttributs)} |
{FOREIGN KEY (listeAttributs) REFERENCES
 nomTableBis[(listeAttributs)]
 [ON DELETE {NO ACTION}|CASCADE|{SET NULL}|
 {SET DEFAULT}]}
[ON UPDATE {NO ACTION}|CASCADE|{SET NULL}|
 {SET DEFAULT}]
}|
{CHECK (condition)}
```

# Syntaxe compacte du CREATE TABLE

```
CREATE TABLE nomTable (
 {nomAttribut typeDeDonnées [DEFAULT valeur][NULL|NOT NULL]
 [UNIQUE|PRIMARY KEY] [REFERENCES
 nomTableBis(nomAttributBis)][CHECK (condition)]}*
 [,PRIMARY KEY (listeAttributs)]
 [,UNIQUE (listeAttributs)]*
 [,FOREIGN KEY (listeAttributs) REFERENCES
 nomTableBis[(listeAttributs)]
 [ON DELETE {NO ACTION |CASCADE|SET NULL |SET DEFAULT]
 [ON UPDATE [NO ACTION|CASCADE|SET NULL| SET DEFAULT]]*
 [,CHECK (condition)]*)
```

# Types de données dans ORACLE (1/6)

## ■ *Numérique*

### NUMBER ou NUMBER (p) ou NUMBER(p, c)

- NUMBER(p) : Nombre entier à  $p$  chiffres ( $p \leq 38$ )
  - ex : NUMBER (3): entier à 3 chiffres maximum (de -999 à +999)
- NUMBER(p,c : Nombre entier ou décimal à  $p$  chiffres (excluant la virgule) dont  $c$  chiffres après la virgule (si  $c > 0$ )
  - ex: NUMBER (5,2) : réel à 5 chiffres au total dont 2 après la virgule; valeur maxi = 999.99
  - Si  $c < 0$ , les données réelles sont arrondies à  $c$  positions à gauche de la virgule
- NUMBER : nombres à virgule flottante (précision et échelle maximales); ex.  $1.666 \text{ e}^{-30}$

## Types de données dans ORACLE (2/6)

| Donnée réelle | Type spécifié | Stocké en ...            |
|---------------|---------------|--------------------------|
| 9645123.88    | NUMBER        | 9645123.88               |
| 9645123.88    | NUMBER(9)     | 9645123                  |
| 9645123.88    | NUMBER(9,2)   | 9645123.88               |
| 9645123.88    | NUMBER(9,1)   | 9645123.9                |
| 9645123.88    | NUMBER(6)     | dépassement de précision |
| 9645123.88    | NUMBER (7,-2) | 9645100                  |
| 9645123.88    | NUMBER (7,2)  | dépassement de précision |



# Types de données dans ORACLE (3/6)

## ■ *Numérique*

- D'autres types numériques de la norme ANSI sont reconnus dans Oracle
  - **INT(EGER), SMALLINT, FLOAT, REAL...**
  - Ces types sont "convertis " dans le type NUMBER

| Type SQL ANSI                 | Type Oracle  |
|-------------------------------|--------------|
| NUMERIC(p,c)<br>DECIMAL (p,c) | NUMBER (p,c) |
| INT(EGER), SMALLINT           | NUMBER (38)  |
| FLOAT (b), REAL               | NUMBER       |

# Types de données dans ORACLE (4/6)

## ■ *Chaîne de caractères*

- CHARACTER(n) ou **CHAR(n)**
  - Chaîne de caractère de taille fixe égale à  $n$  ; avec  $n \leq 2000$
  - Exemples : 'G. Lemoyne-Allaire', 'Paul L"Heureux '
- CHARACTER VARYING (n) ou **VARCHAR2(n)**
  - Taille variable (max de  $n$  caractères) ;  $n \leq 4000$

# Types de données dans ORACLE (5/6)

- *Un type pour représenter la date **et** l'heure*

## DATE

- Une date comporte l'information sur l'année (2 ou 4 chiffres), le mois (2 chiffres), le jour (2 chiffres), l'heure (0-23), la minute(0-59), et la seconde (0-59)
- Format par défaut : DD-MON-YY ex: 01-FEB-11
- Utilisation des fonctions **to\_date** et **to\_char** pour entrer/afficher une date selon un format différent
  - Ex : `to_date('1998/05/31:12:00:00AM', 'yyyy/mm/dd:hh:mi:ssam')`
  - Ex : `to_char(sysdate, 'Dy DD-Mon-YYYY HH24:MI:SS')`
    - Résultat affiché : Tue 21-Apr-1998 21:18:27

# Types de données dans ORACLE (6/6)

- *Binaire long*
  - BINARY LARGE OBJECT (n) ou **BLOB(n)**
    - *n* : taille en octets (ex: 1024, 5K, 3M, 2G)
  
- *Longue chaîne de caractères*
  - CHARACTER LARGE OBJECT (n) ou **CLOB(n)**

# Contraintes d'intégrité (CI)

- Plusieurs types de contraintes statiques déclarées avec `CREATE TABLE`
  - contraintes sur le domaine d'un attribut
  - contraintes de clé primaire
  - contraintes d'intégrité référentielle
  
- D'autres contraintes peuvent être programmées et/ou porter sur des changements d'états de la BD, sur plusieurs tables...
  - `CREATE TRIGGER` (non traité ici)

# CI sur le domaine d'un attribut (1/4)

- Type de données (number, char...)
- Contrainte NOT NULL
  - L'utilisateur doit fournir une valeur définie à l'attribut (si pas de DEFAULT)

```
CREATE TABLE Client
 (noClient INTEGER NOT NULL,
 nom VARCHAR(50) NOT NULL,
 ddn VARCHAR(15) NULL,
 téléphone VARCHAR(15))
```

- Par défaut : NULL
  - Le SGBD insère la valeur NULL si absence de valeur et pas de DEFAULT

## CI sur le domaine d'un attribut (2/4)

### ■ Contrainte CHECK sur un attribut

*Le numéro de client est positif et inférieur à 1000*

```
CREATE TABLE Client
 (noClient INTEGER NOT NULL
 CHECK (noClient>0 AND noClient<1000) ,
 nom VARCHAR(50) NOT NULL,
 ddn VARCHAR(15) NULL)
```

```
CREATE TABLE Client
 (noClient INTEGER NOT NULL,
 nom VARCHAR(50) NOT NULL,
 ddn VARCHAR(15) NULL,
 CHECK (noClient>0 AND noClient<1000))
```

## CI sur le domaine d'un attribut (3/4)

- Contrainte CHECK sur plusieurs attributs du même tuple

*Les produits dont le numéro est supérieur à 100 ont un prix supérieur à 15 €.*

```
CREATE TABLE Produit
 (noProduit INTEGER NOT NULL,
 libellé VARCHAR(50),
 prixUnitaire NUMBER(10,2) NOT NULL,
 CHECK (noProduit<=100 OR prixUnitaire>15))
```



# CI sur le domaine d'un attribut (4/4)

- Valeur d'un attribut par défaut (DEFAULT)
- *Le numéro de téléphone est 'confidentiel' par défaut*

```
CREATE TABLE employé
 (numEmp INTEGER,
 numTel VARCHAR(15) NOT NULL
 DEFAULT 'confidentiel' ...)
```

# Contrainte de clé primaire

- Deux tuples ne peuvent pas avoir la même valeur de la clé
- Le(s) attribut(s) clé ne peuvent pas être NULL
- *La clé primaire de la table Client est noClient* (clé atomique)

```
CREATE TABLE Client
 (noClient INTEGER PRIMARY KEY,
 nom VARCHAR(50) NOT NULL,
 ddn VARCHAR(15) NULL);
```

```
CREATE TABLE Client
 (noClient INTEGER ,
 nom VARCHAR(50) NOT NULL,
 ddn VARCHAR(15) NULL,
 PRIMARY KEY (noClient))
```

# Contrainte de clé primaire (clé composée)

- *La clé primaire de la table Commande est constituée des 3 attributs noClient, noProduit et dateCde.*

```
CREATE TABLE Commande
(noClient INTEGER,
 noProduit INTEGER,
 dateCommande DATE,
 quantité INTEGER,
PRIMARY KEY
 (noClient,noProduit,dateCommande)
)
```

- Lorsque la clé est composée de plusieurs attributs, définir la contrainte PRIMARY KEY au niveau de la table

# Contrainte d'unicité (UNIQUE)

- Une seule clé primaire par table mais d'autres attributs peuvent avoir des valeurs uniques pour chaque tuple.

```
CREATE TABLE Citoyen
(numSécu INTEGER PRIMARY KEY ,
 nom VARCHAR(50) ,
 prénom VARCHAR(50) ,
 ddn DATE NULL, --date de naissance
 noPassport INTEGER NULL UNIQUE)
```

# Attribut auto-incrémentés – Notion de séquence (1/2)

- Les champs *incrémentables automatiquement (auto increment)* n'existent pas dans le SGBD Oracle. Pour simuler ce comportement, il est possible d'utiliser la notion de séquence.
- -- Création d'une séquence

```
CREATE SEQUENCE ma_sequence
START WITH 1
MAXVALUE 9999
MINVALUE 1;
```

# Attribut auto-incrémentés \_

## Notion de séquence (2/2)

- `ma_sequence.nextval` retourne la valeur suivante de la séquence `ma_sequence`
- Exemple d'utilisation : insérer une ligne dans la table `ma_table`, avec la valeur suivante de `ma_sequence` comme valeur de l'attribut `colonne`

```
INSERT INTO ma_table (colonne, ...)
VALUES (ma_sequence.NextVal, ...);
```

- On peut noter que la séquence n'est pas liée à un champ ou à une table et qu'il est possible de l'utiliser pour plusieurs attributs/tables

# Contrainte d'intégrité référentielle (FOREIGN KEY / REFERENCES)

*noClient* dans *Commande* fait référence à la clé primaire dans *Client*.

```
CREATE TABLE Commande
(noCommande INTEGER PRIMARY KEY,
 noClient INTEGER,
 noProduit INTEGER,
 dateCommande DATE,
 quantité INTEGER,
FOREIGN KEY (noClient) REFERENCES Client(noClient),
FOREIGN KEY (noProduit) REFERENCES Produit(noProduit))
```

- Les tables *Client* et *Produit* doivent avoir été créées
- On ne peut ajouter des commandes que si le client et le produit existent déjà
- **N.B. : La cible d'une référence doit être PRIMARY KEY ou UNIQUE**

# Cl référentielle : conduite à tenir en cas de mise à jour

```
[CONSTRAINT nomContrainte]
```

```
FOREIGN KEY listeAttributs REFERENCES
 nomTableBis(listeAttributs)
```

```
[ON DELETE {NO ACTION|CASCADE|SET NULL| SET DEFAULT}]
```

```
[ON UPDATE {NO ACTION|CASCADE|SET NULL| SET DEFAULT}]
```



# CI référentielle : conduite à tenir en cas de mise à jour (DELETE)

## ■ Tentative de suppression de la clé primaire

```
CREATE TABLE Commande
(noCommande INTEGER PRIMARY KEY,
 noClient INTEGER, ...
 FOREIGN KEY (noClient) REFERENCES
 Client(noClient))
```

```
DELETE FROM Client where noClient=45
```

*Que deviennent les commandes du client numéro 45 ?*

### ■ quatre options pour la conduite à tenir :

- NO ACTION
- SET NULL
- CASCADE
- SET DEFAULT

# CI référentielle : conduite à tenir en cas de suppression de la clé primaire

## ■ NO ACTION

*s'il existe des commandes du client 45, la suppression est refusée (violation contrainte)*

## ■ CASCADE

*s'il existe des commandes du client 45, supprimer ces commandes puis supprimer le client.*

## ■ SET NULL

*s'il existe des commandes du client 45, y positionner noClient à NULL puis supprimer le client.*

## ■ SET DEFAULT

*s'il existe des commandes du client 45, y positionner noClient à la valeur par défaut (lorsqu'elle existe) puis supprimer le client.*

# Cl référentielle : conduite à tenir en cas de mise à jour (UPDATE)

## ■ Tentative de modification de la clé primaire

```
CREATE TABLE Commande
(noCommande INTEGER PRIMARY KEY,
noClient INTEGER, ...
FOREIGN KEY (noClient) REFERENCES Client(noClient))
```

```
UPDATE Client SET noClient=100 where noClient=45
```

- *Que deviennent les commandes du client numéro 45 ?*
- quatre options pour la conduite à tenir :
  - NO ACTION
  - SET NULL
  - CASCADE
  - SET DEFAULT

# CI référentielle : conduite à tenir en cas de mise à jour (UPDATE)

## ■ NO ACTION

*s'il existe des commandes du client 45, la modification est refusée (violation contrainte)*

## ■ CASCADE

*s'il existe des commandes du client 45, y modifier noClient puis modifier noClient dans Client*

## ■ SET NULL

*s'il existe des commandes du client 45, y positionner noClient à NULL puis opérer la modification de noClient dans Client.*

## ■ SET DEFAULT

*s'il existe des commandes du client 45, y positionner noClient à la valeur par défaut (lorsqu'elle existe) puis supprimer le client.*

# CI référentielle : conduite à tenir en cas de mise à jour

```
CREATE TABLE Commande
(noCommande INTEGER PRIMARY KEY,
noClient INTEGER, ...
FOREIGN KEY (noClient) REFERENCES Client(noClient)
ON DELETE CASCADE
ON UPDATE CASCADE)
```

## ■ *N.B.* : Sous Oracle

✓ par défaut :

- ON DELETE NO ACTION
- ON UPDATE NO ACTION

✓ mais seules possibilités dans CREATE TABLE

- ON DELETE CASCADE
- ON DELETE SET NULL

## 2. Création de schéma + instanciation (1/2)

- CREATE TABLE : ne permet **que** la définition d'un schéma de relation
- L'instanciation de la relation se fait par une suite d'insertions de tuples (commande INSERT)
- Il est possible de créer et d'instancier une table à partir d'une requête :

```
CREATE TABLE ... AS SELECT ...
```

## 2. Création de schéma + instanciation (2/2)

```
CREATE TABLE nomTable[(nomAtt1, nomAtt2 ...)] AS SELECT ...
```

**a-** créer un schéma de relation

**b-** remplir la relation par évaluation de la requête

```
CREATE TABLE Client
 (noClient integer not null,
 nom varchar(50) not null,
 CP number(5) check CP>=1000 and CP<=99999);
INSERT INTO Client values (1, 'A', 54000);
INSERT INTO Client values (2, 'B', 75800);
...
CREATE TABLE Client54 AS
 (SELECT noClient, nom
 from Client
 where CP between 54000 and 54999);
```

# 3. Modification de schéma de relation(1/3)

- Une fois un schéma de table créé et des données entrées, on peut :
  - ajouter un attribut (nom, type, défaut, NOT NULL uniquement)
  - modifier ou supprimer le type ou la valeur par défaut d'un attribut
  - supprimer un attribut ou ajouter une contrainte
  - supprimer une contraintes nommée ou de clé primaire ou d'unicité

**ALTER TABLE nomTable**

```
ADD (attribut type [DEFAULT valeur] [contrainte]) |
MODIFY (attribut [type] [DEFAULT valeur] [contrainte]) |
DROP COLUMN attribut |
ADD [CONSTRAINT nom] contrainte |
DROP {PRIMARY KEY} | UNIQUE (attributs)} |{CONSTRAINT nom}
```

- On peut renommer une table

**RENAME ancienNomDeTable to nouveauNomDeTable**



### 3. Modification de schéma de relation(2/3)

```
ALTER TABLE client ADD (TEL_PORTABLE CHAR(10))
```

```
ALTER TABLE client MODIFY (ADR_CLIENT CHAR(70))
```

```
ALTER TABLE produit ADD CONSTRAINT PRIX_POSITIF
check (PRIXUNITAIRE >= 0)
```

```
ALTER TABLE produit DROP CONSTRAINT PRIX_POSITIF
```

```
ALTER TABLE client DROP COLUMN TEL_PORTABLE
```

```
RENAME CLIENT to ACHETEUR
```

### 3. Modification de schéma de relation(3/3)

- La commande ALTER TABLE permet de **désactiver** une contrainte nommée

```
ALTER TABLE nom_table DISABLE CONSTRAINT
nom_contrainte;
```

- La commande ALTER TABLE permet **d'activer** une contrainte nommée

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

## 4. Suppression d'un schéma de relation (DROP TABLE)

**DROP TABLE** *nom-relation* [**CASCADE CONSTRAINTS**]

- ✓ suppression du **schéma** de la relation
- ✓ suppression des **tuples** de la relation
- ✓ Suppression des **index**, désactivation des **vues** liées

Si la clé primaire de la relation à supprimer est référencée dans d'autres tables, l'option **CASCADE CONSTRAINTS** permet de supprimer ces CI dans ces tables.

# 5. Index

- Structure permettant d'accélérer les accès aux données d'une table

- **Création d'index**

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

UNIQUE : pas de double sur les clés de l'index (ni pour les valeurs de l'attribut)

ASC|DESC : ordre croissant/décroissant des clés dans l'index.

```
CREATE INDEX idx_prod_lib on Produit (libellé)
```

- **Suppression d'index**

```
DROP INDEX nom_index
```

**N.B.** : un index est automatiquement créé pour la clé primaire

# **Le langage SQL (*SGBD ORACLE*)**

- I. Introduction à SQL et à Oracle
- II. Interrogation des données
- III. Mise à jour des données
- IV. Définition des données
- V. Les vues**
- VI. Notion de transaction
- VII. Notion de privilèges

# Les Vues (1/3)

Une **vue** est une relation virtuelle :

- ses tuples ne sont pas stockés
- la requête de définition est stockée
- ses tuples sont générés à chaque appel de la vue

```
CREATE [OR REPLACE] VIEW nom-vue[(attributs)]
 AS commande-SELECT
```

*Définir, comme une vue, la liste des produits (nom et prix) **chers**, c-à-d dont le prix dépasse 500€*

```
CREATE OR REPLACE VIEW produits-chers (nomProduit,prix) AS
 (SELECT libellé, prixUnitaire
 FROM Produit
 WHERE prixUnitaire > 500)
```

## Les Vues (2/3)

- Une fois créée, une vue est utilisable comme une table quelconque.

*Liste des produits chers dont le libellé comporte 'de luxe'*

```
SELECT *
FROM ProduitsChers
WHERE nomProduit like '%de luxe%'
```

- Suppression d'une vue

```
DROP VIEW nom-vue
```

# Les Vues (3/3)

## Intérêt des vues

- Masquage des opérations de jointure
- Sauvegarde (indirecte) de requêtes complexes
- Support de l'indépendance logique
  - si les tables sont modifiées, les vues doivent être réécrites mais les requêtes utilisant les vues ne subissent pas de changement
- Support de la confidentialité en combinaison avec des privilèges d'accès adéquats
  - Masquer les lignes ou colonnes sensibles pour les utilisateurs non autorisés

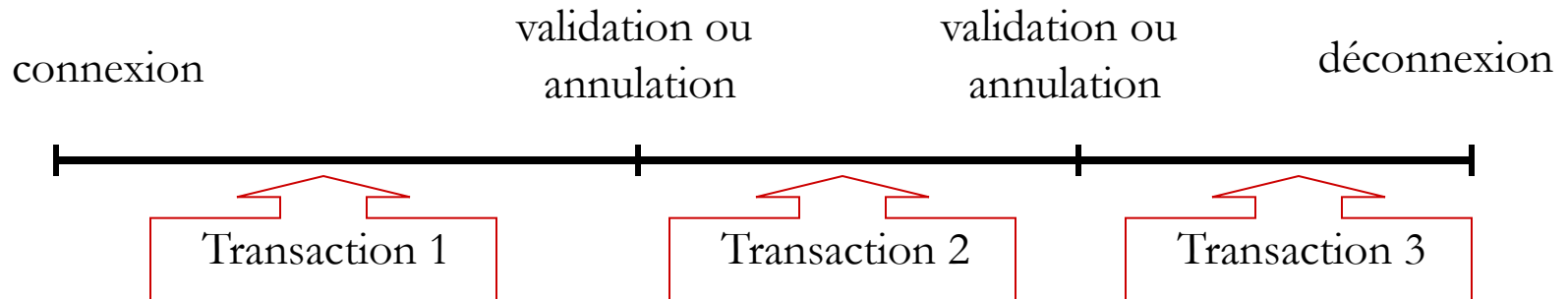


# **Le langage SQL (*SGBD ORACLE*)**

- I. Introduction à SQL et à Oracle
- II. Interrogation des données
- III. Mise à jour des données
- IV. Définition des données
- V. Les vues
- VI. Notion de transaction**
- VII. Notion de privilèges

## IV. Notion de transaction (1/2)

- Transaction = succession d'opérations de mises à jour (*insert*, *delete*, *update*) d'une BD.
- Début d'une transaction : début de la session de travail ou fin de la transaction précédente.



## IV. Notion de transaction (2/2)

- Validation et fin d'une transaction : les mises à jour deviennent permanentes et visibles des autres utilisateurs

**COMMIT [TRANSACTION]**

- Annulation et fin d'une transaction : les mises à jour sont annulées

**ROLLBACK [TRANSACTION]**

- A la déconnexion : validation automatique de la transaction en cours
- Les commandes du LDD (`create table`, `create view...`) sont validées automatiquement (commandes auto-commit)

# **Le langage SQL (SGBD ORACLE)**

- I. Introduction
- II. Interrogation des données
- III. Mise à jour des données
- IV. Définition des données
- V. Les vues
- VI. Notion de transaction
- VII. Notion de privilèges**

## VII. Notion de privilèges (1/2)

- Privilège : droit d'accès à un objet (table, vue...)
- Privilèges possibles sur les objets
  - SELECT : lecture
  - INSERT : ajout de tuples
  - UPDATE : modification de tuples
  - DELETE : suppression de tuples
  - INDEX : construction
  - ALL : tous les privilèges

## VII. Notion de privilèges (2/2)

### ■ Transmission de privilèges

```
GRANT privilège ON {table|vue} TO
 {utilisateur|PUBLIC} [WITH GRANT OPTION]
```

*Donner le droit de lecture sur la table Commande à tout utilisateur*

```
GRANT SELECT ON Commande TO PUBLIC
```

### ■ Suppression de privilèges

```
REVOKE privilège ON {table|vue} FROM
 {utilisateur|PUBLIC}
```

*Retirer le droit de lecture sur Commande à public*

```
REVOKE SELECT ON Commande FROM PUBLIC
```