Bases de Données relationnelles et leurs systèmes de Gestion

Chapitre IV :
Langages du modèle relationnel de données : L'essentiel de SQL

Nacer.Boudjlida@loria.fr Nancy Université, UHP Nancy 1

Le langage SQL (SG*BD ORACLE*)

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

Nacer.Boudjlida@loria.fr Nancy Université, UHP Nancy 1

I. Introduction à SQL et à Oracle

1. Généralités sur le langage SQL
2. Introduction au SGBD ORACLE
   a) Architecture
   b) Quelques concepts
   c) L'interface SQL*Plus (cf. TP)

Nacer.Boudjlida@loria.fr Nancy Université, UHP Nancy 1

1. Généralités sur SQL(1/2)

- SQL : basé sur l'algèbre relationnelle et le calcul des prédicats à variables tuples
- SQL : standard ANSI/ISO depuis 1986
  - SQL-86 (version préliminaire)
  - SQL-89 ou SQL1(niveau minimal supporté)
  - SQL-92 ou SQL2 (standard courant bien supporté)
  - SQL-99 ou SQL3 (extensions majeures de SQL2, peu supporté)
- SQL-Oracle proche de la norme
  - Oracle 8.i : Compatible SQL2
  - Oracle 9.i : Compatible SQL3

Nacer.Boudjlida@loria.fr Nancy Université, UHP Nancy 1

1. Généralités sur SQL (2/2)

1. **LDD** (**L**angage de **D**éfinition de **D**onnées) pour créer, modifier et supprimer des définitions d'objets (*schema objects*)
2. **LMD** (**L**angage de **M**anipulation de **D**onnées) pour:
   - Ajouter, supprimer, modifier, rechercher des instances de données
   - Valider/annuler des actions
3. **LCD** (**L**angage de **C**ontrôle de **D**onnées) pour gérer les droits d'accès

Nacer.Boudjlida@loria.fr Nancy Université, UHP Nancy 1

2. Introduction au SGBD ORACLE

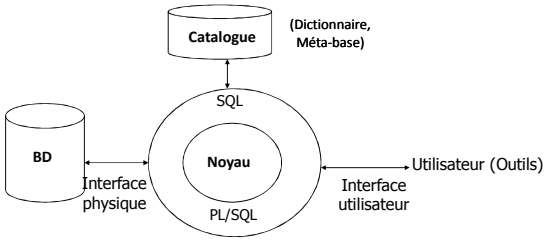
- Edité par *Oracle Corporation* (www.oracle.com) depuis 1981 : > 60% du marché
- Version actuelle : 11g (mi 2007)
- 10g = Version installée au Département et à l'ESIAL
- Large éventail d'outils « autour »:
  - Serveurs de données et utilitaires (sql*plus, import/export loader…)
  - Environnements de développement (Web, java, pré-compilateurs, …)
  - Logiciels d'aide à la décision (data warehouse, Oracle Discoverer…)
  - Progiciels client serveurs et Internet (Oracle Application & Oracle e-business suite)
  - …

Nacer.Boudjlida@loria.fr Nancy Université, UHP Nancy 1

2. Introduction au SGBD ORACLE

a) Architecture
b) Quelques concepts
c) L'interface SQL*Plus: Concepts (mise en œuvre en TP)

2.a) Architecture d'Oracle data server



SQL*plus : interface pour soumettre des requêtes SQL ou PL/SQL à Oracle

2.b) 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 : {tables} et {vues}
  - Créé dans chaque base, dans l'espace SYSTEM
  - Appartient à *sys*, non modifiable (sauf sys.aud$)

Dictionnaire de données (catalogue)

- Vues de nom **USER_xxx**
  - Informations sur les objets dont l'utilisateur est propriétaire
- Vues de nom **ALL_xxx**
  - Informations sur les objets auxquels a accès l'utilisateur
- Vues de nom **DBA_xxx**
  - Informations sur les objets de tous les utilisateurs
- Exemples :
  - USER-TABLES: « mes » tables
  - ALL_TABLES: « mes » tables + celles auxquelles j'ai accès
  - DBA_TABLES : Toutes les tables (uniquement pour DBA)

Dictionnaire de données (catalogue)

- Quelques vues du dictionnaire :
  - USER_OBJECTS, ALL_OBJECTS, DBA_OBJECTS
  - USER_TABLES, ALL_TABLES, DBA_TABLES
  - USER_TAB_COLUMNS, USER_TAB_COMMENTS
  - …
- Vue **DICTIONARY** (ou **DICT**) : liste des tables et des vues du dictionnaire
- **Note** : describe *Nom-de-Table*

2.c) SQL*Plus : interface SQL et PL/SQL

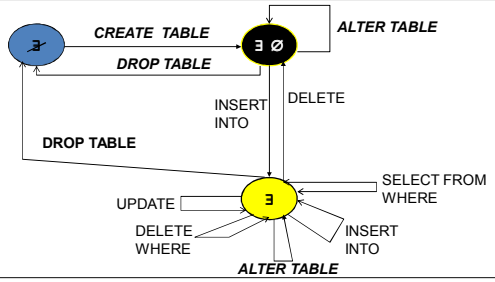
- SQL*Plus : interface ligne de commande entre l'utilisateur et le SGBD Oracle.
- Fonctionalités de SQL*Plus (cf. TP)
  1. Lancer, Quitter
  2. Connexion, Déconnexion
  3. Exécuter des commandes SQL ou des blocs PL/SQL
  4. Editer des commandes
  5. Formater les résultats
  6. etc.

Le langage SQL (SGBD ORACLE)

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

(L'essentiel de) SQL

- CREATE, ALTER, DROP TABLE : Opérations sur le schéma de relation



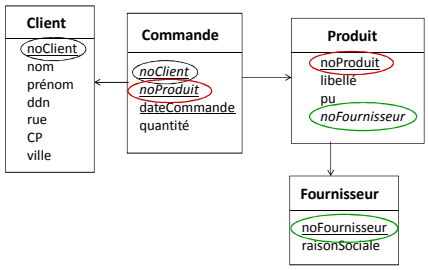
- SELECT, INSERT, UPDATE, DELETE : Opérations sur l'instance de relation

Base de données exemple (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
Clés étrangères

Base de données exemple (2/2)



Oracle 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
  - noms d'attributs, de tables, de contraintes…
- Seul cas où la casse est prise en compte : la comparaison de chaînes de caractères

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

- Mais Dupont ≠DUPONT

I- Interrogation de données

- *Traduction* des opérateurs de l'algèbre en SQL (projection, restriction, jointure, …)

  SELECT … FROM … [WHERE …]

- Opérateurs supplémentaires pour
  - Trier les résultats
  - Agréger des données (somme, moyenne, cardinal, …)
  - etc.

SQL : Interrogation des données

I. Traduction des opérateurs algébriques
II. Expressions et fonctions prédéfinies
III. Fonctions d'agrégation
   - Cardinal, minimum, maximum, moyenne, …
IV. Groupement (Classes d'équivalence)
V. Ordre sur les résultats
VI. Appartenance et quantificateur existentiel

La commande SELECT

- **Détails de la syntaxe : voir le manuel de référence**

```
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]…]
```

- **Requête composée**: CdeSelect = commande SELECT :
  CdeSelect {UNION | INTERSECT | EXCEPT} CdeSelect

Notations pour la syntaxe

Notation	Signification
[*élément*]	*élément* est **optionnel**
élément$_1$ | *élément*$_2$	on a **le choix** entre *élément*$_1$ et *élément*$_2$
{ *élément*$_1$ | *élément*$_2$ }	idem
{ *élément1* } | { *élément2* }	idem

I.1- Traduction de la projection (1/3)

1. Si *R* est un nom de relation alors *R* est une expression algébrique

- Expression algébrique: R
- Expression SQL : SELECT * FROM R
- Exemple : Valeurs de tous les attributs de toutes les pièces
  - En Algèbre : pièce
  - En SQL : SELECT * FROM pièces
  - Résultat : Instance de la relation pièces

Traduction de la projection (2/3)

2. Si *R* est une *ea* et que *L* est une liste d'attributs alors $\prod_L(R)$ est une *ea*

- En SQL : SELECT L FROM R
- *Trouver le nom et la ville de tous les fournisseurs:*
  $\prod_{nomf, ville}$ (fournisseur)

```
SELECT     nomf, ville
FROM       fournisseur
```
ou
```
SELECT     ALL nomf, ville
FROM       fournisseur
```

nomf	ville
Girard	Lyon
Blanc	Paris
Merlin	Nancy

- **Attention : les doublons ne sont pas supprimés !**

Traduction de la projection (3/3)

- Trouver les numéros des fournisseurs vendant au moins une pièce (sans les doublons): clause **DISTINCT**

En Algèbre: $\prod_{nof}$ (vente)

En SQL : SELECT **DISTINCT** nof FROM vente

nof
1
2
3

Traduction de la sélection/restriction

3. Si R est une ea et que C est une expression logique alors $\sigma_C(R)$ est une ea

- En algèbre : $\sigma_C(R)$
- En SQL : SELECT * FROM R WHERE C

Restriction sur une condition

- *Quels sont les produits dont le prix est inférieur à 20€ et le numéro est supérieur à 30 ?*

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

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

Exp. SQL équivalente

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


Conditions de Restriction

- SELECT * FROM .....
  WHERE condition atomique ou composée
- **Atomique**: <Attribut> <opérateur de comparaison> <expression de valeur>
  - <, > >=, >=, !=, < >
  - [**NOT**] **BETWEEN** expression **AND** expression
  - [**NOT**] **IN** listeConstantes ou [**NOT**] **IN** (requête SELECT)
  - Cas des valeurs NULL: <attribut> **IS** [**NOT**] **NULL**
  - Notion de « patron » : expression [**NOT**] **LIKE**
- **Composition** à l'aide de connecteurs (AND, OR)

Restriction : Exemples

- *Produits dont le prix coûtant entre 50 et 100€*
  - avec **AND**:
    ... where prix >= 50 and prix <= 100
  - Avec **BETWEEN**:
    ... where prix 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 selon un *patron* avec jockers éventuels, où :
  - **%** : « jocker » pour **zéro à n** caractères
  - **_** : « jocker » pour **un et un seul** caractère

NULL et valeurs de vérité

A	B	A ET B	A OU B	NON(A)
VRAI	VRAI	VRAI	VRAI	FAUX
VRAI	FAUX	FAUX	VRAI	FAUX
FAUX	VRAI	FAUX	VRAI	VRAI
FAUX	FAUX	FAUX	FAUX	VRAI
VRAI	NULL	NULL	VRAI	FAUX
FAUX	NULL	FAUX	NULL	VRAI
NULL	VRAI	NULL	VRAI	NULL
NULL	FAUX	FAUX	NULL	NULL
NULL	NULL	NULL	NULL	NULL

Composition : Restriction et projection

- noClient et dateCommande des Commandes dont la date est postérieure au 01/01/2004 ?

```
SELECT   noClient, dateCommande
FROM     Commande
WHERE    dateCommande > '01/01/2004'
```

noClient	dateCommande
100	5/1/2004
300	25/2/2004
400	30/1/2004

Exp. Algébr. équivalente

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

Restriction : Opérateur IN

- *Prénom des clients dont le nom est Dupont, Durant ou Martin :*
  … where nom **IN** ('Dupont', 'Durant', 'Martin')
- *Prénom des clients dont le nom n'est pas dans l'ensemble {Dupont, Durant, Martin}*
  … where nom **NOT IN** ('Dupont', 'Durant', 'Martin')

Traduction de l'union et de la différence

4. Si R et S sont des ea
   alors $(R \cup S)$, $(R \setminus S)$ sont des ea

- En SQL:

```
        select * from R
UNION (Select * from S)

        select *from R
EXCEPT (select * from S)
```


Opérations ensemblistes (UNION, MINUS, INTERSECT)

- 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 Client)  INTERSECT
(SELECT nomPass as Nom, prénomPass as Prénom
 FROM Passager)
```

Nom	prénom
Conrad	James
Leconte	Jean

- **Note:** Renommage des colonnes en sortie (**as** Nouveau Nom)

Produit cartésien et jointure

5. Si R et S sont des ea alors (R x S) est une ea
   En SQL: select * from R, S

- Cas de la jointure: $R \bowtie_{\theta\text{-Expression}} S$
  En SQL: select * from R, S where θ-Expression
- Exemple :
  select * from personne, voiture
  where **personne**.NoPers = **voiture**.NoPers

Composition : Restriction et projection

- noClient et dateCommande des Commandes dont la date est postérieure au 01/01/2004 ?

```
SELECT   noClient, dateCommande
FROM     Commande
WHERE    dateCommande > '01/01/2004'
```

noClient	dateCommande
100	5/1/2004
300	25/2/2004
400	30/1/2004

Exp. Algébr. équivalente

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

Jointure : suite

- Forme algébrique: $\prod_L (R \bowtie_{\theta\text{-Expression}} S)$
- *Nom des clients et liste de leurs commandes*

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

- Nommage des attributs : en cas d'ambiguïté préfixer le nom d'un attribut par sa relation ou par un **alias** de relation

Jointure : suite

- Forme algébrique: $\prod_L (\sigma_C(R) \bowtie_{\theta\text{-Expression}} S)$
- *Produits commandés en quantité supérieure à 100 et dont le prix dépasse 1000 €. Afficher les numéros de produit, leur libellé , leur prix unitaire ainsi que la date de la commande.*

```
SELECT   Produit.noProduit, libellé, prixUnitaire, date
FROM     Produit, Commande
WHERE    quantité > 100 AND prixUnitaire >1000
AND      Produit.noProduit = Commande.noProduit
```


Remarque: SQL et arbre algébrique

```
SELECT  L  FROM R1, …, Rn
WHERE  Condition
```

- Arbre « canonique »:
  1. Produit cartésien des relations
  2. Restriction sur la clause where
  3. Projection

Remarques: Nommage des attributs

1. Nommage des attributs dans une requête :
   - Attributs **UTILISABLES** après SELECT et dans la clause WHERE : **UNIQUEMENT** ceux des relations de la clause FROM
   - Cas où des attributs ont le **même nom**, mais appartiennent à des relations différentes: les **désigner de façon non ambigüe**
2. Nommage des attributs dans le résultat d'une requête

Rmq1: Nommage des Attributs dans les requêtes

1. Nom de l'attribut
2. *Nom de Relation* .Nom de l'attribut
3. *Alias de Relation* .Nom de l'attribut

- Forme 2 ou 3 OBLIGATOIRE en cas d'ambiguïté entre les noms des attributs de TOUTES les relations de la clause FROM

Rmq1: Nommage des Attributs dans les requêtes

- **Alias** de relations pour alléger l'écriture

Liste des ventes et fournisseurs associés

```
SELECT  V.nop, V.nof, F.nomf
FROM    vente V, fournisseur F
WHERE   V.nof = F.nof
```

- **V, F** *alias* respectifs de vente et fournisseur
- **V**.nop ≡ **vente**.nop et **F**.nof ≡ **fournisseur**.nof

Rmq2: Renommage des colonnes du résultat

- Par défaut : noms dans les schémas de relations
- *Nom et prix des pièces dont le numéro est supérieur à 2*

```
SELECT   nomp as nom-produit ,
         prix as prix-produit
FROM     pièce   WHERE  nop > 2
```

nom-produit	prix-produit
boulon	2.5

Au lieu de :

nomp	prix
boulon	2.5

SQL : Interrogation des données

1) Traduction des opérateurs algébriques
2) Expressions et fonctions prédéfinies
3) Fonctions d'agrégation
   - Cardinal, minimum, maximum, moyenne, …
4) Groupement (Classes d'équivalence)
5) Ordre sur les résultats
6) Appartenance et quantificateur existentiel

Expressions et fonctions prédéfinies

- Utilisables dans :
  - La liste de projection
  - La clause where

Expression de calcul 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 dans la clause WHERE

- *Liste des noArticle dont le prix toutes taxes comprises (TTC) dépasse 40€*

```
SELECT  noArticle FROM  Article
WHERE   prixUnitaire*1.15 > 40
```

- Une expression peut aussi faire appel à des **fonctions**
- Liste des commandes de la journée:

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


Principales fonctions prédéfinies (1/3)

- **ABS(n)** :Valeur absolue
- **CEIL(n)** : plus petit entier ≥ n
- **FLOOR(n)** : plus grand entier ≤ 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

- **RTRIM**(chaîne[,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

Principales fonctions prédéfinies (2/3)

- **LTRIM**(chaîne[,ens_car]): Suppression de caractères par la gauche
- **LENGTH**(ch) : Longueur de chaîne
- **COUNT**(-) : Cardinal
- **AVG(-)** : Moyenne
- **SUM(-)** : Somme
- **MAX(-)** : Maximum

- **INSTR**(ch,sous_ch(,position(, n))) : Recherche la sous-chaîne dans la chaîne
- **GREATEST**(n1,n2,...) : Le plus grand
- **LEAST**(n1,n2,...) : Le plus petit de la liste

Principales fonctions prédéfinies (3/3)

- **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
- **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_DAT**E(ch[,'format']) : Conversion d'une chaîne en date
- **NVL**(expr,valeur) : Protège de la manipulation d'une valeur NULL
- **UID** : Identifiant système de l'utilisateur
- **USER** : Nom de l'utilisateur

SQL : Interrogation des données

1) Traduction des opérateurs algébriques
2) Expressions et fonctions prédéfinies
3) Fonctions d'agrégation
   - Cardinal, minimum, maximum, moyenne, …
4) Groupement (Classes d'équivalence)
5) Ordre sur les résultats
6) Appartenance et quantificateur existentiel

Fonctions d'agrégation (ou de groupe)

- Opèrent sur un groupe de valeurs d'attributs et produisent une valeur résultat (= extension de l'algèbre relationelle)

- ***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 nbr2NonNull
FROM Article [WHERE prixUnitaire IS NOT NULL]
```

- ***Nombre de prixUnitaires non null différents***

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

nbPrix
8

Fonctions d'agrégation et valeur NULL

- **AVG**(n) : valeur moyenne de n, **ignore** les valeurs NULL
- **COUNT**(expr) : nombre de lignes où **expr est non** NULL
- **MAX**(expr) : valeur maximum de l'expression
- **MIN**(expr) : valeur minimum de l'expression
- **SUM**(n) : somme des valeurs de n
- **STDDEV**(n) : écart type de n, **ignore** les valeurs NULL
- **VARIANCE**(n) : variance de n, **ignore** les valeurs NULL
  - n : attribut de type numérique
  - expr : attribut ou expression sur attribut(s)

- **Plus de détails**: cf. manuel de référence

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 **invalide** parce que 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)
```

SQL : Interrogation des données

1) Traduction des opérateurs algébriques
2) Expressions et fonctions prédéfinies
3) Fonctions d'agrégation
   - Cardinal, minimum, maximum, moyenne, …
4) Groupement (Classes d'équivalence)
5) Ordre sur les résultats
6) Appartenance et quantificateur existentiel

Partition de relations (GROUP BY)

- Partitionner une relation en groupes de tuples tels que chaque groupe ait la même valeur pour une (des) colonne(s) de groupement

  SELECT … GROUP BY nom(s) de colonne(s)

- ≅ Construire des classes d'équivalence (« groupes ») telles que :

  t1 ≡ t2 si t1 et t2 ont les mêmes valeurs pour les attributs de groupement (*les tuples t1 et t2 seront dans le même « groupe »*)

Partition de relations (GROUP BY)

- *Nombre de produits commandés par chacun des clients*

1) Commandes groupées par numéro de client
2) Pour chaque groupe, afficher le numéro du client concerné par le groupe et le nombre de commandes.

- **Note** : chaque expression du SELECT doit avoir une valeur **unique** par **groupe**.

Partition de relations (GROUP BY)

Nombre de produits commandés par client :

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

noProduit	dateCommande	noClient
4	5/1/2003	10
5	5/1/2003	10
20	14/5/2003	12
28	15/8/2003	12
68	15/8/2003	12
59	20/9/2003	15

noClient	totalProduits
10	2
12	3
15	1

Partition de relations (GROUP BY)

- *Quantité totale de produits commandés par client en excluant le produit F565*

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

1. **Exclusion** des tuples ne vérifiant pas la clause WHERE
2. **Groupement** des commandes restantes par noClient
3. **Pour chaque groupe**, afficher le numéro du client concerné par le groupe et la somme des quantités

Partition de relations (GROUP BY)

- Clause **WHERE** : Sélection parmi les tuples
- Clause **HAVING** : Sélection parmi les groupes
- *Quantité moyenne commandée par produit pour les produits commandés plus de 3 commandes. Ignorer les commandes du client C47.*

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

- Après évaluation de la condition du WHERE et groupement par numéro de client:
  1. Pour chaque groupe, compter le nombre d'éléments
  2. Eliminer les groupes à moins de 3 éléments.
  3. Afficher le résultat

Partition de relations (GROUP BY)

Partitionnement sur plusieurs colonnes

- *Nombre de produits commandés par client et par date*

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

noProduit	dateCommande	noClient
4	5/1/2003	10
5	5/1/2003	10
20	14/5/2003	12
28	15/8/2003	12
68	15/8/2003	12
59	**20/9/2003**	**15**

noClient	dateCommande	nbProduits
10	5/1/2003	2
12	14/5/2003	1
12	15/8/2003	2
15	**20/9/2003**	**1**

SQL : Interrogation des données

1) Traduction des opérateurs algébriques
2) Expressions et fonctions prédéfinies
3) Fonctions d'agrégation
   - Cardinal, minimum, maximum, moyenne, …
4) Groupement (Classes d'équivalence)
5) Ordre sur les résultats
6) Appartenance et quantificateur existentiel

Ordre sur résultat (ORDER BY)

- Trier les résultats sur une ou plusieurs colonnes

```
SELECT … FROM … [WHERE …]
ORDER BY colonne(s) [ASC|DESC]
```

  - **ASC** : ordre ascendant (par défaut)
  - **DESC** : ordre descendant

- *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**

Appartenance et Requêtes imbriquées

- Utiliser le résultat d'une requête dans une condition de la clause WHERE

```
SELECT colonne(s) FROM relation(s)
WHERE
  expression [NOT]IN (sous-requête)   |
  expression opérateurComparaison
  [ANY|ALL](sous-requête)|
  {EXISTS | NOT EXISTS} (sous-requête)
```


Appartenance et Requêtes imbriquées ([NOT] IN)

- Nom des clients ayant passé commande le 24/10/2009
- *{c.nom | Client(c) ∧ Commande(c1)*
  ∧ $\Pi_{c1.noClient}(c) \in \Pi_{c1.noClient}(\sigma_{dateCommande = '...'}(c1))$}

```
SELECT c.nom FROM Client c
WHERE c.noClient IN
(SELECT c1.noClient FROM Commande c1
 WHERE dateCommande ='24/10/2009')
```

- Autre formulation (cf. CP à variables domaines) :
  {nom | Client(noClient, nom, …) ∧ ∃noCommande
  Commande(noClient, noCommande…, '24/10/2009')}

Quantificateur existentiel : [NOT] EXISTS

- Clients ayant passé au moins une commande

{C1 | Client(C1) ∧ ∃C2 Client(C2) ∧ C1.noClient = C2.noClient}

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

- Clients n'ayant passé aucune commande

{C1 | Client(C1) ∧ ¬∃C2 Client(C2) ∧ C1.noClient = C2.noClient}

- Remarque: $(p \rightarrow q) \equiv (\neg p \vee q)$
- $(a > 5) \rightarrow (a > 0) \equiv (\neg(a > 5) \vee (a > 0))$

Requêtes imbriquées : opérateurs ALL/ANY (SOME)

- Produits plus chers que tous les produits commandés aujourd'hui.

```
SELECT * FROM Produit p1 WHERE prixUnitaire >
ALL(SELECT prixUnitaire FROM Produit p2, Commande C
    WHERE P2.noProduit=C.noProduit
    AND C.dateCommande=CURRENT_DATE)
```

- Produits plus chers qu'un produit commandé aujourd'hui.

```
SELECT * FROM Produit p1 WHERE prixUnitaire >
SOME(SELECT prixUnitaire FROM Produit p2, Commande C
     WHERE P2.noProduit = C.noProduit
     AND C.dateCommande = CURRENT_DATE)
```


Quelques règles de nommage des objets

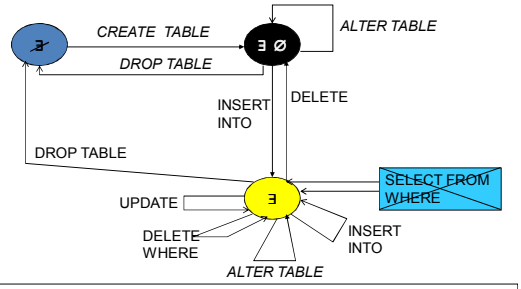
- Dénomination d'un objet : choix d'un nom (ou identificateur) pour désigner cet objet
- Un nom d'objet doit respecter certaines règles :
  - 1 à 30 caractères
  - pas de distinction majuscule/minuscule
  - le premier caractère est une lettre
  - caractères possibles :
    - caractères alphanumériques
    - 3 caractères spéciaux : _ $ #
  - ne doit pas être un mot réservé du langage SQL
  - doit être unique dans le schéma d'un utilisateur

Quelques règles de nommage des objets

- **Propriétaire** d'un objet = utilisateur l'ayant créé
- Nommage d'une relation
  - Si elle m'appartient : ***Nom-de-la-relation***
  - Sinon: ***Nom-du-propriétaire.Nom-de-relation***
  - Possibilité de créer (puis utiliser) des synonymes
    CREATE SYNONYM LesProduits FOR toto.produit
- Nommage d'un attribut (rappel)
  - Si pas d'ambiguïté : ***nom-attribut***
  - Sinon : ***nom-de-relation.nom-attribut***
  - ou : ***alias-de-relation.nom-attribut***

(L'essentiel de) SQL

- CREATE, ALTER, DROP TABLE : Opérations sur le schéma de relation



- SELECT, INSERT, UPDATE, DELETE : Opérations sur l'instance de relation

Le langage SQL (SGBD ORACLE)

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

III. Définition de données (schémas)

1. Création de schéma de relation :
   CREATE TABLE
2. Création de schéma + instanciation :
   CREATE TABLE … AS…
3. Modification de schéma: **ALTER TABLE**
4. Suppression de schéma: **DROP TABLE**
5. Index: **CREATE/DROP INDEX**
6. Déclencheurs : **CREATE/DROP TRIGGER**
7. Procédures, fonctions:
   CREATE/DROP PROCEDURE/FUNCTION

Rappel: Notion de Schéma de relation

Schéma de relation : **`NomDeRelation(U, P)`**

- **`U` : liste de couples (attribut, domaine)**
  - Domaine : type de données
- **`P` : liste des contraintes d'intégrité**
  - Ce qu'on peut exprimer dépend du SGBD
  - Les autres : programmer leur vérification

III.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 définition de la table
    - Schéma stocké dans le dictionnaire de données
    - Allocation des structures physiques

Schéma et dictionnaire Oracle

```
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                    DATA_TYPE
------------------------------ ------------------
------------
NOCLIENT                       NUMBER
NOMCLIENT                      VARCHAR2
NOTÉLÉPHONE                    VARCHAR2
```


Définition de schémas de relations

- Définition « minimale » :

  CREATE TABLE MaRelation (
  Attribut1 Type de l'attribut1,
  Attribut2 Type de l'attribut2,

  Attribut-n Type de l'attribut-n)

- Adjonction éventuelle des définitions des contraintes

Syntaxe de CREATE TABLE (1/2)

```
CREATE TABLE nomTable
       (définitionDattribut
       [,définitionDattribut]…
       [,définitionDeContrainte]…));
```

- *définitionDattribut*

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

NULL Autorisé ou pas

Typage (domaine)

Contraintes

Syntaxe de CREATE TABLE (2/2)

Clé Primaire

Nommer une contrainte

Inclusion/Référence

```
[CONSTRAINT nomContrainte]
{PRIMARY KEY 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)
}
```

Que faire si…

Autres contraintes

III.1- Définition de schémas

1. Typage des attributs
2. Contrainte d'intégrité
   1. Intra-relation
   2. Inter-relations (CI Référentielle)
3. Conduite à tenir si violation

III.1- Définition de schémas

1. Typage des attributs
   a) Numérique
   b) Chaînes de caractères
   c) Date et temps
   d) Objets de grande taille

III.1.1- Types de données (1/5)

- *Numérique*
  1. **NUMBER** ou **NUMBER (p) :** Nombre entier avec au plus ***p*** chiffres ($p \leq 38$)
  2. **NUMBER(p, c):** décimal avec au plus ***p*** chiffres (excluant la virgule) dont ***c*** chiffres après la virgule (si ***c*** > 0)
- Exemples:
  - NUMBER (3): entier avec au plus 3 chiffres (–999 à +999)
  - NUMBER (5, 2) : réel à 5 chiffres dont 2 après la virgule;

Types de données (2/5)

- *Numérique*
  - **INT(EGER)**, **SMALLINT**, **FLOAT**, **REAL**…: Dans la norme ANSI
  - « Convertis " dans le type NUMBER

Type SQL ANSI	Type Oracle
NUMERIC(p,c) DECIMAL (p,c)	NUMBER (p,c)
INT(EGER), SMALLINT	NUMBER (38)
FLOAT (b), REAL	NUMBER

Types de données (3/5)

Chaîne de caractères

- CHARACTER(n) ou **CHAR(n)**
  - De taille fixe (n) *; avec n <=2000*
  - Exemples : 'SGBD', 'Le Rois est nu '
- CHARACTER VARYING (n) ou **VARCHAR2(n)**
  - De taille variable (*n* caractères maximum)
  - *n* <= 4000
- cf. Fonctions associées

Types de données (4/5)

Date et temps

- **DATE**
  - année (quatre chiffres), mois (2 chiffres) et jour (2 chiffres)
  - Exemple : DATE '1998-08-25'
- **TIME[(p)]**
  - Heure (2 chiffres), minutes (2 chiffres), secondes: 2 +*p* chiffres ($p_{ième}$ de seconde)
  - Exemple : TIME '14:04:32.25'
- **TIMESTAMP[(p)]**
  - DATE + TIME
  - Exemple : TIMESTAMP '1998-08-25 14:04:32.25'

Types de données (5/5)

Objets Longs (Grande taille)

- Binaires
  - **BLOB(n)** : BINARY LARGE OBJECT
  - *n* : taille en octets (ex: 1024, 5K, 3M, 2G)
- Chaînes de caractères
  - **CLOB(n)** : CHARACTER LARGE OBJECT
  - **NCLOB(n)** : NATIONAL CHARACTER LARGE OBJECT (n)

III.1- Définition de schémas

1. Typage des attributs
2. Contrainte d'intégrité
   1. Intra-relation
   2. Inter-relations (CI Référentielle)
3. Conduite à tenir si violation

III.1.2- Contraintes d'intégrité (CI)

- Plusieurs types de contraintes statiques déclarables avec `CREATE TABLE`
  - Contraintes sur le domaine d'un attribut
  - Contraintes de clés (primaire, candidate)
  - Contraintes d'intégrité référentielle (contrainte d'inclusion et clé étrangère)
- 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 (1/5)

- Type de données (number, char...)
- Contrainte NOT NULL
  - Valeur obligatoire pour 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
  - Insertion de la valeur NULL si absence de valeur et de default

CI sur le domaine d'un attribut (2/5)

- Contrainte CHECK sur un attribut
- CHECK (Expression logique)
- Exemple: numéro de client ∈ [0..1000]

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


CI sur le domaine d'un attribut (3/5)

- Contrainte CHECK sur plusieurs attributs du même tuple (Contrainte intra-relation)
- *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 (4/5)

- Création d'un domaine spécifique (`CREATE DOMAIN`)
- *Un employé/client est un homme ou une femme.*

```
CREATE DOMAIN domaineSexe AS
                CHAR(1) CHECK (VALUE IN ('M', 'F'))

CREATE TABLE employé
 (numEmp        INTEGER,
  genre   domaineSexe, … );
CREATE TABLE client
 (noClient      INTEGER,
  genre   domaineSexe, … )
```

N.B. **Les domaines ne sont pas supportés dans Oracle.**

CI sur le domaine d'un attribut (5/5)

- Valeur d'un attribut par défaut (`DEFAULT`)
  - Si pas de valeur lors d'une insertion (NULL)
  - Ou attribut mis à NULL lors d'une modification
- Exemple: *numéro de téléphone : présence obligatoire et valeur "confidentiel" par défaut*

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


III.1.2- Contraintes d'intégrité (CI)

- Plusieurs types de contraintes statiques déclarables avec `CREATE TABLE`
  - Contraintes sur le domaine d'un attribut
  - Contraintes de clé
    - Primaire
    - Candidate
  - Contraintes d'intégrité référentielle (contrainte d'inclusion et clé étrangère)

Contrainte de clé primaire

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

- *noClient , noProduit et dateCde = clé primaire de la table Commande*

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

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

Clé Candidate et Unicité

- Une seule clé primaire par table mais d'autres attributs peuvent avoir des valeurs uniques pour chaque tuple (exemple: clés candidates)
- Clause UNIQUE

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

Contrainte d'intégrité référentielle (FOREIGN KEY / REFERENCES)

Contrainte inter-relations

- Émetteur d'une Commande doit être un Client connu dans la relation Client

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

- Tables ***Client*** et ***Produit*** déjà définies : pas de référence en avant
- Cible d'une référence : PRIMARY KEY ou UNIQUE
- Ajout de commandes : si le client et le produit existent

III.1- Définition de schémas

1. Typage des attributs
2. Contrainte d'intégrité
   1. Intra-relation
   2. Inter-relations (CI Référentielle)
3. Conduite à tenir si violation de CI référentielle

III.1.3- CI 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 ?**
- 4 options :

NO ACTION	- SET NULL
CASCADE	- SET DEFAULT

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

Suppression du client 45 : que faire de ses éventuelles commandes ?

- NO ACTION : suppression du client 45 refusée
- CASCADE :
  1. Supprimer ses commandes
  2. Supprimer le client.
- SET NULL :
  1. noClient = NULL dans les commandes du client 45
  2. Supprimer le client
- SET DEFAULT :
  1. noClient = valeur par défaut si celle-ci est définie
  2. Supprimer le client

CI référentielle : conduite à tenir en 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 ?***

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

- NO ACTION : Modification refusée
- CASCADE :
  1. Modifier noClient dans ses commandes
  2. Modifier noClient dans Client.
- SET NULL :
  1. noClient = NULL dans les commandes du client
  2. Modifier noClient dans Client
- SET DEFAULT :
  1. noClient = valeur par défaut si celle-ci est définie
  2. Modifier 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 )
```

- Sous Oracle
  - par défaut :
    - ON DELETE NO ACTION
    - ON UPDATE NO ACTION
  - seules possibilités dans CREATE TABLE
    - ON DELETE CASCADE
    - ON DELETE SET NULL

III. 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 **(CREATE/DROP Index)**
6. [Procédures, fonctions: **CREATE/DROP PROCEDURE/FUNCTION**
7. Déclencheurs : **CREATE/DROP TRIGGER]**
8. **etc.**

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

1. Définition d'un schéma → relation vide
   - Instanciation de la relation par une suite d'insertions de tuples (cf. commande INSERT)
2. Définir et instancier par le résultat d'un SELECT :

CREATE TABLE … AS (SELECT …)

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

1. Créer un schéma de relation et instancier
2. Créer et instancier une relation par les tuples résultant d'une requête

```
CREATE TABLE Client
 (noClient integer not null,
  nom varchar(50)        not null,
  CP      number(5) check CP>=1000 and CP<=99999)
```

En supposant que Client a été instanciée :

```
CREATE TABLE Client54 AS
        (SELECT noClient, nom FROM Client
         WHERE CP BETWEEN 54000 AND 54999)
```


III.3. Modification de schémas (1/3)

- Ajouter un attribut (nom, type, défaut, NOT NULL uniquement) ou une contrainte
- Modifier ou supprimer le type ou la valeur par défaut d'un attribut
- Supprimer un attribut ou une contrainte
- Supprimer une contrainte: nommée, clé primaire, unicité
- Impact possible sur les programmes !!!!

III.3. Modification de schémas (1/3)

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} (attribut) | CONSTRAINT nom}

- Renommage de table

```
RENAME ancienNomDeTable TO nouveauNomDeTable
```


III.3. Modification de schémas (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
```


III.3. Modification de schémas (3/3)

- ALTER TABLE et désactivation d'une contrainte nommée

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

- ALTER TABLE et activation d'une contrainte nommée

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


III.4. Suppression d'un schéma (DROP TABLE)

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

- Suppression
  1. des **tuples** ET du **schéma** de la relation
  2. des **index**
  3. désactivation des **vues** liées
  4. des contraintes référentielles éventuelles (dans les relations où la clé primaire de la relation à supprimer est référencée)

III. 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 **(CREATE/DROP Index)**
6. [Procédures, fonctions: **CREATE/DROP PROCEDURE/FUNCTION**
7. Déclencheurs : **CREATE/DROP TRIGGER]**
8. **etc.**

III.5. Les index

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

- UNIQUE : pas de doubles sur les clés de l'index (ni pour les valeurs de l'attribut)
- ASC|DESC : ordre des clés dans l'index.

```
CREATE INDEX idx_prod_lib ON Produit (libellé)
CREATE INDEX idx_stock ON Stock(prod#, dep#)
```

- Index sur 1 à 32 attributs
- **Suppression :** `DROP INDEX nom_index`
- ***N.B.*** : un index est automatiquement créé pour la clé primaire

Le langage SQL (SGBD ORACLE)

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

IV. Les Vues (1/3)

- Vue : relation calculée
  - ses tuples ne sont pas stockés
  - sa requête de définition est stockée
  - ses tuples sont générés à chaque appel de la vue

```
CREATE [OR REPLACE] VIEW nom-vue[(attribut(s)]
  AS  (commande SELECT de définition)
```

- *Définir, comme une vue, la liste des produits dont le prix dépasse 500€. Mettre dans la vue les colonnes nomProduit et prixUnitaire*

Les Vues (2/3)

- Une fois créée, une vue est utilisable comme toute autre table
- *Liste des produits chers dont le libellé comporte 'de luxe'*

```
SELECT *    FROM          ProduitsChers
WHERE  nomProduit like '%de luxe%'
SELECT *  FROM  ProduitsChers pc, Stock s
Where ....
```

- Suppression d'une vue

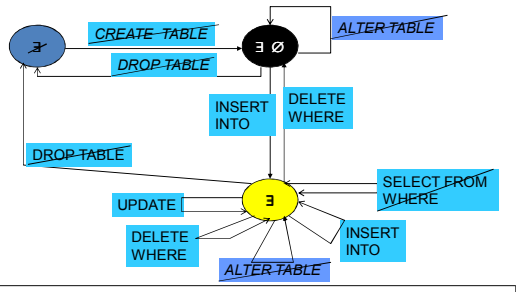
```
DROP VIEW nom-vue
```


Les Vues (3/3): intérêt

- 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

(L'essentiel de) SQL

- CREATE, ALTER, DROP TABLE : Opérations sur le schéma de relation



- SELECT, INSERT, UPDATE, DELETE : Opérations sur l'instance de relation

Le langage SQL (SGBD ORACLE)

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

V. SQL: Mises à jour

1. Ajout de tuple(s) :
   INSERT INTO …
   [SELECT …]
2. Suppression de tuple(s) :
   DELETE FROM …
   [WHERE …]
3. Modification de tuple(s) :
   UPDATE … SET nouvelles valeurs
   [FROM … WHERE]

V.1- Insertion de tuple(s)

- *Ajout d'une « ligne » dans produit*

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

- *Ajout d'une « ligne incomplète» dans produit*

```
INSERT INTO Commande (noClient, noProduit)
VALUES (1234, 430)
```

- *Ajout d'un ensemble de lignes résultat d'un SELECT*

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

- Les attributs non cités sont positionnés à NULL

V.2- Modification de tuple(s)

- Modification d'une « ligne »: *Positionner à 'Durand' le nom du client n° 3 :*

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

- Modification d'un ensemble de lignes: *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')
```


V.2- Modification de tuple(s)

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

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

- *Positionner le libellé du produit 99 à 'produiTest' et son prix au prix moyen des produits:*

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


V.3- Suppression de tuple(s)

- *Supprimer les clients de Metz :*

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

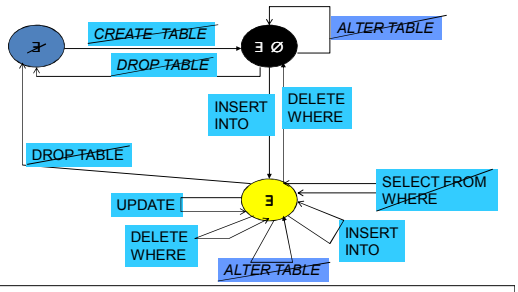
- *Supprimer tous les tuples de la table Client :*

```
DELETE Client
```

- ATTENTION :
  - DELETE concerne l'extension de la relation !
  - DROP concerne son schéma !
- DELETE :
  - suppression de **tous** les tuples de la table
  - le **schéma** de la table existe toujours

(L'essentiel de) SQL

- CREATE, ALTER, DROP TABLE : Opérations sur le schéma de relation



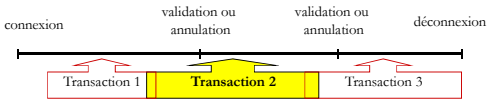
- SELECT, INSERT, UPDATE, DELETE : Opérations sur l'instance de relation

Le langage SQL (SGBD ORACLE)

I. Introduction
II. Interrogation des données
III. Définition des données
IV. Les vues
V. Mise à jour des données
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
- TOUT ou RIEN
- Possible chevauchement des exécutions (concurrence)



IV. Notion de transaction (2/2)

- Validation et fin d'une transaction (TOUTES ses actions sont valides)
  `Commande SQL : COMMIT`
- Annulation (une ou des actions sont invalides → annuler les mises à jour)
  Commande SQL: **`ROLLBACK`**
- Commandes du LDD (create table, view…): exécutées comme des transactions
- En savoir plus : LMI-3

VII. Notion de privilèges (1/2)

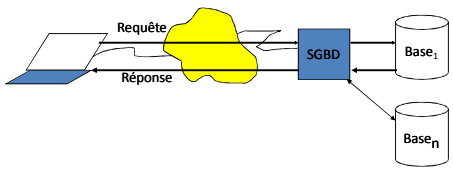
- Privilège : opération autorisée sur un objet (table, vue, procédure, fonction, …)
- Exemples de privilèges sur les objets
  - SELECT : lecture
  - INSERT : ajout de tuples
  - UPDATE : modification de tuples
  - DELETE : suppression de tuples
  - INDEX : construction
  - ALL : tous les privilèges

VII. Notion de privilèges (2/2)

- Transmission de privilèges
  `Commande SQL : GRANT`
  - *Donner le droit de lecture sur la table Commande à tout utilisateur*
    `GRANT SELECT ON Commande TO PUBLIC`
- Suppression de privilèges
  `Commande SQL : REVOKE`
  - *Retirer le droit de lecture sur Commande à public*
    `REVOKE SELECT ON Commande FROM PUBLIC`

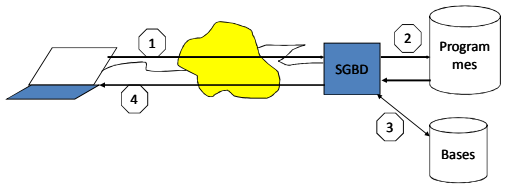
SQL : Conclusion

- SQL : « Syntaxe au-dessus » de l'algèbre
- Déclaratif, orienté ensembles (résultat)
- Adapté *peu ou prou* pour :



SQL : Limites

- Puissance de calcul et types de données limités
- Pas de facilités de « présentation » des résultats
- Insuffisant pour les 2 autres types de communication (cf. chapitre d'introduction) dont :



SQL : Pallier les limites

- Extensions procédurales : Ajout de structures de contrôle (de programmes):
  - Conditionnelles : *si … alors … sinon …;* cas …
  - Itérations : *pour* itérateur *dans* intervalle *faire …*
    tant que condition faire …
- Créer et invoquer des « programmes SQL-pur »
- Bénéficier de la puissance de calcul d'un langage de programmation : Immersion dans un langage « hôte » (C, Java, Cobol, Fortran, assembleur, …)
- Supporter des types de données non prédéfinis
- etc.

SQL : Conclusion

- Langage commun aux SGBDR
- Fondement : Algèbre + Calcul relationnel
- Définition & Manipulation :
  - Schémas : CREATE, ALTER, DROP
  - Instances : SELECT, INSERT, UPDATE, DELETE
- Déclaratif, Orienté ensembles
- Extensions de SQL :
  - Fonctions d'agrégation, tri (vu)
  - Procédurales (si alors sinon, tant que, etc.): à voir