Généralités (1)

- Nous allons voir dans cette partie les concepts et la syntaxe du langage SQL
- Le langage SQL permet d'exprimer 3 grandes familles d'opérations :
  - l'interrogation et la recherche dans les tables
  - la gestion des tables et des vues (avec les contraintes)
  - la manipulation de données
- Nous ferons le lien avec l'algèbre relationnelle dans la partie interrogation et recherche dans les tables
- La gestion des tables concerne la table et sa structure : partie ***LDD*** de *SQL*
- La manipulation de données concerne les données contenues dans la table : partie ***LMD*** de *SQL*

Généralités (2)

- Evolution du langage *SEQUEL*, lui-même dérivé du langage de recherche *SQUARE*
- Langage normalisé par l'ISO depuis 1987
- *SQL2* a été adopté en 1992 → base de travail
- *SQL3* a été adopté en 1999 : fonctionnalités liées à l'approche objet
- Le langage SQL permet de manipuler des relations
- Un ensemble d'instructions SQL se nomme une **requête**
- Pour les traitements complexe, on utilise un **langage hôte** (C, PHP, Java, …) :
  - les instructions SQL sont intégrées au langage via une interface spécifique
  - les résultats des requêtes sont stockées dans des structures de données propres

Généralités (3)

Exemple : *SQL* et *PHP*

```
<html>
 <head>
  <title>Agenda</title>
 </head>
 <body>
  <form action="agenda.php" METHOD="POST">
   <select name="tri">
    <option value="Nom">Classement par noms</option>
    <option value="Age">Moyenne d'age</option>
   </select>
   <input type="submit" value="Exécuter">
  </form>
  <?php
   mysql_connect(localhost','root')
   mysql_select_db($bdd);
   $choix=$_POST['tri'];
   if ($choix == "Nom") {
    print "<br>Classement par Noms";
    $query = "SELECT * FROM $nomtable ORDER BY nom ";}
   $result= mysql_query($query);
   mysql_close();
  ?>
 </body>
</html>
```

Interrogations en SQL

Expression d'une interrogation

- La forme générale d'une interrogation est la suivante :

  SELECT *Liste d'attributs* **FROM** *Noms de relations* **WHERE** *Conditions* **;**

- La clause *WHERE* est optionnelle
- La condition de la clause *WHERE* peut-être multiple : *OR*, *AND*, *NOT*
- Le résultat de la requête est retourné sous la forme d'une relation
- Une interrogation SQL n'entraîne aucune modification dans la base
- Le résultat d'une interrogation n'est pas mémorisé

Opérations relationnelles avec SQL (1)

Projection (1)

- Extraction de certains champs (colonnes)
- On spécifie la liste des colonnes à inclure derrière l'instruction *SELECT*, en les séparant par des virgules :

```
SELECT <col1>, … FROM <table1> ;
```

- Si l'on désire afficher toutes les colonnes, on utilise le caractère *****
- On peut renommer les colonnes de la table "résultat" par le mot clé **AS**

Voiture

Code	Marque	Modèle	Cat
1	Peugeot	107	A
2	Renault	Clio	B
3	Fiat	Panda	A
4	Peugeot	207	B
5	Citroën	C4	B
6	Fiat	500	Z

SELECT Marque, Modèle **FROM** Voiture **;**

Marque	Modèle
Peugeot	107
Renault	Clio
Fiat	Panda
Peugeot	207
Citroën	C4
Fiat	500

SELECT Modèle **AS** Type **FROM** Voiture **;**

Type
107
Clio
Panda
207
C4
500

Opérations relationnelles avec SQL (2)

Projection (2)

- Il est possible d'éliminer les éventuels doublons d'une colonne : il faut faire précéder le nom de la colonne par le mot clé **DISTINCT**
- Il est également possible d'utiliser une expression algébrique (*+ - * / %*) pour créer une colonne, grâce au mot clé **AS**
- Des fonctions statistiques sont disponibles (COUNT, MAX, MIN, AVG, SUM, …) pour évaluer une colonne (ou la table entière pour COUNT)

Voiture

Code	Marque	Modèle	Prix	Nb
1	Peugeot	107	8000	3
2	Renault	Clio	13000	10
3	Fiat	Panda	7500	5
4	Peugeot	207	13000	15
5	Citroën	C4	16000	8
6	Fiat	500	12000	1

SELECT DISTINCT Marque **FROM** Voiture **;**

Marque
Peugeot
Renault
Fiat
Citroën

SELECT (Prix * Nb) **AS** Total **FROM** Voiture **;**

Total
24000
130000
37500
195000
128000
12000

SELECT COUNT(*) **AS** Nb_ref **FROM** Voiture **;**

Nb_ref
6

Opérations relationnelles avec SQL (3)

Sélection

- Extraction de certains tuples (ligne) suivant certains critères
- Le critère de sélection est indiqué à la suite du mot-clé **WHERE**
- Les opérateurs de comparaison sont les suivants : **= != > < >= <=**

```
SELECT * FROM <table1> WHERE <condition> ;
```

- Les critères peuvent être composés avec des opérateurs logiques : **OR**, **NOT**, **AND**
- Test d'appartenance à un intervalle **BETWEEN … AND …** , à un ensemble de valeurs **IN <liste de valeurs>**, comparaison de chaînes de caractères **LIKE**

Voiture

Code	Marque	Modèle	Cat
1	Peugeot	107	A
2	Renault	Clio	B
3	Fiat	Panda	A
4	Peugeot	207	B

SELECT * **FROM** Voiture **WHERE** Marque **=** FIAT **;**

SELECT * **FROM** Voiture **WHERE** Marque **IN** ("Renault", "Fiat") **;**

SELECT * **FROM** Voiture **WHERE** Marque **=** "Fiat" **OR** Marque **=** "Audi" **;**

Code	Marque	Modèle	Cat
3	Fiat	Panda	A

SELECT * **FROM** Voiture **WHERE** Modèle **BETWEEN** 105 **AND** 108

Code	Marque	Modèle	Cat
1	Peugeot	107	A

Opérations relationnelles avec SQL (4)

Agrégation ou *groupage* (1)

- Regroupement de n-uplets par valeur contenues dans une colonne
- On utilise le mot-clé **GROUP BY** suivi du nom de la colonne sur laquelle s'effectue l'agrégat

SELECT Marque, **AVG(**Prix**) AS** Prix_Moyen **FROM** Voiture **GROUP BY** Marque **;**

Marque	Prix_moyen
Peugeot	10500
Renault	13000
Fiat	9750
Citroën	16000

Voiture

Code	Marque	Modèle	Prix	Nb
1	Peugeot	107	8000	3
2	Renault	Clio	13000	10
3	Fiat	Panda	7500	5
4	Peugeot	207	13000	15
5	Citroën	C4	16000	8
6	Fiat	500	12000	1

SELECT Marque, **COUNT(*) AS** Nombre **FROM** Voiture **GROUP BY** Marque **;**

Marque	Nombre
Peugeot	2
Renault	1
Fiat	2
Citroën	1

Opérations relationnelles avec SQL (5)

Agrégation ou *groupage* (2)

- Le résultat de l'opération de groupage peut être filtré
- En pratique, cette opération est utile pour filtrer sur le résultat des opérations statistiques appliquées aux sous-ensembles définis par groupage
- Le mot-clé **HAVING** permet d'effectuer une sélection sur le résultat de l'opération de groupage : ne pas utiliser WHERE !!!!

Voiture

Code	Marque	Modèle	Prix	Nb
1	Peugeot	107	8000	3
2	Renault	Clio	13000	10
3	Fiat	Panda	7500	5
4	Peugeot	207	13000	15
5	Citroën	C4	16000	8
6	Fiat	500	12000	1

Marque	Nombre
Peugeot	2
Fiat	2

SELECT Marque, **COUNT(*) AS** Nombre **FROM** Voiture **GROUP BY** Marque HAVING Nombre > 2 **;**

Opérations relationnelles avec SQL (6)

Intersection

- On utilise le mot-clé **INTERSECT** :

```
<requête1> INTERSECT <requête2> ;
```

T1

Code	Nom	Prix	Stock
DT4300	Dell Precision 4300	749	17
DL4200	Dell Latitude 4200	900	3
SV2540	Sony Vaio 2540	1529	2

T2

Code	Nom	Prix	Stock
DT4300	Dell Precision 4300	749	17
ITP42	IBM Thinkpad T42	1849	9
SV2540	Sony Vaio 2540	1529	2

```
SELECT * FROM T1
INTERSECT
SELECT * FROM T2 ;
```

Code	Nom	Prix	Stock
DT4300	Dell Precision 4300	749	17
SV2540	Sony Vaio 2540	1529	2

Opérations relationnelles avec SQL (7)

Union

- On utilise le mot-clé **UNION**

```
<requête1> UNION <requête2> ;
```

T1

Code	Nom	Prix	Stock
DT4300	Dell Precision 4300	749	17
DL4200	Dell Latitude 4200	900	3

T2

Code	Nom	Prix	Stock
DT4300	Dell Precision 4300	749	17
SV2540	Sony Vaio 2540	1529	2

```
SELECT * FROM T1
UNION
SELECT * FROM T2 ;
```

Code	Nom	Prix	Stock
DT4300	Dell Precision 4300	749	17
DL4200	Dell Latitude 4200	900	3
SV2540	Sony Vaio 2540	1529	2

Opérations relationnelles avec SQL (7)

Différence

- Pour obtenir les enregistrements d'une première relation qui n'appartiennent pas à une seconde
- On utilise le mot-clé **MINUS**

```
<requête1> MINUS <requête2> ;
```

T1

Code	Nom	Prix	Stock
DT4300	Dell Precision 4300	749	17
DL4200	Dell Latitude 4200	900	3
SV2540	Sony Vaio 2540	1529	2

T2

Code	Nom	Prix	Stock
DT4300	Dell Precision 4300	749	17
ITP42	IBM Thinkpad T42	1849	9
SV2540	Sony Vaio 2540	1529	2

```
SELECT * FROM T1
MINUS
SELECT * FROM T2 ;
```

Code	Nom	Prix	Stock
DL4200	Dell Latitude 4200	900	3

Opérations relationnelles avec SQL (8)

Produit cartésien

- Combinaison de toutes les lignes d'une table avec toutes les lignes d'une autre table
- Il s'écrit très simplement en SQL :

```
SELECT * FROM <table1>, <table2>
```

T1

Appareil	Couleur
Ordinateur	Noir
Ecran	Blanc

T2

Marque	Type
Volkswagen	Polo
Peugeot	206

SELECT * **FROM** T1, T2 **;**

Appareil	Couleur	Marque	Type
Ordinateur	Noir	Volkswagen	Polo
Ecran	Blanc	Volkswagen	Polo
Ordinateur	Noir	Peugeot	206
Ecran	Blanc	Peugeot	206

Opérations relationnelles avec SQL (9)

Jointure (1)

- Combinaison de toutes les lignes d'une table avec toutes les lignes d'une autre table, sans tenir compte du *sens* associé aux données
- En SQL, elle peut s'écrire simplement via le mot-clé **WHERE** (SQL 1)

T1

Code	Nom	Prix	Stock
DT4300	Dell Precision 4300	900	3
DL4200	Dell Latitude 4200	900	3
SV2540	Sony Vaio 2540	1529	2

T2

Ref	Processeur	HDD
DT4300	Core 2 Duo	250
IBMT42	Core 2 Quad	500
SV2540	Pentium	160

SELECT T1.Nom, T2.Processeur **FROM** T1, T2 **WHERE** T1.Code **=** T2.Ref **;**

Nom	Processeur
Dell Precision 4300	Core 2 Duo
Sony Vaio 2540	Pentium

Opérations relationnelles avec SQL (10)

Jointure (2)

- Il est préférable d'utiliser l'opérateur de jointure interne, qui se traduit par le mot-clé **INNER JOIN** associé à **FROM** et **ON** :

```
SELECT <table1.col1>, <table2.col1> FROM <table1> INNER
JOIN <table2> ON <table1.coli> = <table2.colj> ;
```

T1

Code	Nom	Prix	Stock
DT4300	Dell Precision 4300	900	3
DL4200	Dell Latitude 4200	900	3
SV2540	Sony Vaio 2540	1529	2

T2

Ref	Processeur	HDD
DT4300	Core 2 Duo	250
IBMT42	Core 2 Quad	500
SV2540	Pentium	160

SELECT T1.Nom, T2.Processeur **FROM** T1 **INNER JOIN** T2 **ON** T1.Code **=** T2.Ref **;**

Nom	Processeur
Dell Precision 4300	Core 2 Duo
Sony Vaio 2540	Pentium

Opérations relationnelles avec SQL (11)

Jointure externe (1)

- Elle permet d'inclure dans le résultat les lignes d'une table qui n'ont pas de correspondance avec la seconde table
- Les champs qui ne peuvent être mis en correspondance ont la valeur **NULL**
- La jointure externe se fait grâce au mot-clé **OUTER JOIN**
- Cette opération n'est pas symétrique : soit on inclut toutes les lignes d'une table, soit toutes les lignes de l'autre. On utilise les mots-clés **LEFT** et **RIGHT** :

```
SELECT ... FROM table1 LEFT OUTER JOIN table2 ON
<Conditions de jointure> ;
```

Opérations relationnelles avec SQL (12)

Jointure externe (2)

Voiture

Code	Marque	Modèle	Prix	Nb
1	Peugeot	107	8000	3
2	Renault	Clio	13000	10
3	Fiat	Panda	7500	5
4	Peugeot	207	13000	15
5	Citroën	C4	16000	8
6	Fiat	500	12000	1

Vente

Code	Acheteur
1	Dupont
2	Durant
4	Martin
5	Malin

SELECT Voiture.Marque, Voiture.Modèle, Voiture.Prix, Vente.Acheteur **FROM** Voiture
LEFT OUTER JOIN Vente **ON** Voiture.Code **=** Vente.Code **;**

Marque	Modèle	Prix	Acheteur
Peugeot	107	8000	Dupont
Renault	Clio	13000	Durand
Fiat	Panda	7500	NULL
Peugeot	207	13000	Martin
Citroën	C4	16000	Malin
Fiat	500	12000	NULL

Opérations relationnelles avec SQL (13)

Opérateurs spécifiques

Nom	Action
IN	Appartenance à une liste de valeurs
NOT IN	Non-appartenance à une liste de valeurs
ANY, SOME	La condition est vraie si elle est satisfaite pour une ou plusieurs valeurs de la liste
ALL	La condition est vraie si elle est satisfaite pour toutes les valeurs de la liste
[NOT] BETWEEN … AND …	[Non] Appartenance à un intervalle (bornes comprises)
[NOT] EXISTS	La condition est vraie si la sous-requête [ne] retourne [pas] au moins un tuple
[NOT] LIKE	La condition est vraie si le premier opérande [ne] contient [pas] un ensemble de caractères défini avec les jokers – et %
IS [NOT] NULL	La condition est nulle si la valeur de l'attribut [n'] est [pas] nulle

Opérations relationnelles avec SQL (14)

Requêtes imbriquées

- Les requêtes sont dites imbriquées quand une des conditions du SELECT utilise le résultat d'une sous-requête
- On distingue 2 cas :
  - requêtes non synchronisées : la sous-requête est exécutée en premier (*correspond à des boucles juxtaposées*)
  - requêtes synchronisées : les sous requêtes sont exécutées pour chaque valeur de la requête principale (*correspond à des boucles imbriquées*)

Requêtes non synchronisées

- Elle s'écrivent de manière très simple : la sous-requête est mise entre parenthèses

T1

Code	Nom	Prix	Stock
DT4300	Dell Precision 4300	900	3
DL4200	Dell Latitude 4200	900	3
SV2540	Sony Vaio 2540	1529	2

SELECT Nom **FROM** T1 **WHERE** Prix = (**SELECT MAX(**Prix**) FROM** T1) **;**

Nom
Sony Vaio 2540

Opérations relationnelles avec SQL (15)

Requêtes synchronisées

T1

Code	Nom	Prix	Stock
DT4300	Dell Precision 4300	900	3
DL4200	Dell Latitude 4200	900	3
SV2540	Sony Vaio 2540	1529	2

```
SELECT Nom FROM T1 WHERE Prix >= ALL (SELECT Prix FROM T1) ;
```

Nom
Sony Vaio 2540

```
SELECT NomClient FROM Client WHERE EXISTS (SELECT * FROM Facture WHERE
CodeClient = Client.CodeClient AND NumeroFact = 3) ;
```

Opérations relationnelles avec SQL (16)

Tri du résultat d'une requête

- On utilise le mot-clé **ORDER BY** pour spécifier la (les) colonne(s) sur laquelle (lesquelles) on souhaite trier le résultat
- Il est possible de préciser l'ordre du tri par les mots-clés :
  - **ASC** : tri par ordre croissant (par défaut)
  - **DESC** : tri par ordre décroissant

Voiture

Code	Marque	Modèle	Prix	Nb
1	Peugeot	107	8000	3
2	Renault	Clio	13000	10
3	Fiat	Panda	7500	5
4	Peugeot	207	13000	15
5	Citroën	C4	16000	8
6	Fiat	500	12000	1

SELECT Modèle, Prix **FROM** Voiture **ORDER BY** Prix **DESC** ;

Modèle	Prix
Panda	7500
107	8000
500	12000
Clio	13000
207	13000
C4	16000

Opérations relationnelles avec SQL (17)

Fonction numériques (1)

Nom	Action
ABS(<n>)	Valeur absolue de *n*
CEIL(<n>)	Entier supérieur ou égal à *n*
FLOOR(<n>)	Troncature à valeur entière
MOD(<m>, <n>)	Reste de la division de *m* par *n*
POWER(<m>, <n>)	*m* élevé à la puissance *n*
ROUND(<m>, <n>)	*m* arrondi à *n* décimales
SIGN(<n>)	Signe (-1 si <0, 0 si =0, 1 si >0)
SQRT(<n>)	Racine carrée de *n* (0 si n<0)
TRUNC(<m>, <n>)	*m* tronqué à *n* décimales

Opérations relationnelles avec SQL (18)

Fonction numériques (2)

Nom	Action
AVG(<n>)	Moyenne des valeurs de *n* (valeurs nulles non comptées)
COUNT(*)	Nombre de tuples renvoyées par la requête
COUNT(<n>)	Nombre de valeurs non nulles
SUM(<n>)	Somme des valeurs de *n*
MAX(<n>)	Valeur maximum de *n*
MIN(<n>)	Valeur minimum de *n*
STDDEV(<n>)	Ecart-type de *n* (valeurs nulles non comptées)
VARIANCE(<n>)	Variance de *n* (valeurs nulles non comptées)

Opérations relationnelles avec SQL (19)

Fonction sur les chaînes

Nom	Action
INITCAP(<c>)	Première lettre de chaque mot en majuscule
LOWER(<c>)	Conversion en minuscules
UPPER(<c>)	Conversion en majuscules
LTRIM(<c>)	Suppression des espaces à gauche
RTRIM(<c>)	Suppression des espaces à droite
REPLACE(<c>, <c1>, <c2>)	Remplacement dans *c* de *c1* par *c2*
SOUNDEX(<c>)	Donne la représentation phonétique de *c*
SUBSTR(<c>, <d>, <l>)	Sous-chaîne extraite commençant au caractère de rang d et de longueur *l*
LENGTH(<c>)	Longueur de la chaîne

Opérations relationnelles avec SQL (20)

Fonction sur les dates

Nom	Action
ADD_MONTH(<d>, <n>)	Ajouter *n* mois à la date *d*
LAST_DAY(<d>)	Dernier jour du mois *d*
MONTH_BETWEEN(<d1>, <d2>)	Nombre de mois entre les dates *d1* et *d2*
NEW_TIME(…)	Date et heure dans un autre méridien
SYSDATE	Date et heure système
TO_CHAR(…)	Convertit une date en chaîne de caractères
TO_DATE(…)	Convertit une chaîne de caractères en date

Exercice 8 : un peu de SQL (1)

Soit la base de données composée des tables suivantes :

Voiture

Code	Marque	Modèle
1	Peugeot	107
2	Renault	Clio
3	Fiat	Panda
4	Peugeot	207
5	Citroën	C4
6	Fiat	500

Personne

Ach	Nom	Age	Ville	Sexe
1	Nestor	96	Paris	M
2	Irma	20	Lille	F
3	Henri	45	Paris	M
4	Josette	34	Lyon	F
5	Jacques	50	Bordeaux	M

Vente

Date	Prix	Code	Ach
05-01-2009	8000	1	1
14-01-2009	13000	2	4
25-02-2009	13000	4	1
19-04-2009	16000	5	2

Exercice 8 : un peu de SQL (2)

Question 1

Trouvez les villes dans lesquelles habitent les personnes, et ordonnez le résultat de manière décroissante

→ SELECT DISTINCT *Ville* FROM *Personne* ORDER BY *Ville* DESC ;

Question 2

Affichez le chiffre d'affaires, comprenant les taxes (20%). Renommez la colonne « résultat » CA_TTC

→ SELECT SUM(*Prix**1.2) AS *CA_TTC* FROM *Vente*

Question 3

Trouvez le nombre de voitures par marque

→ SELECT *Marque*, COUNT(*) FROM *Voiture* GROUP BY *Marque*

Question 4

Trouvez l'âge des personnes qui n'habitent pas à Paris

→ SELECT *Age* FROM *Personne* WHERE NOT (*Ville* = `*Paris*`)

Exercice 8 : un peu de SQL (3)

Question 5

Donnez le nom des personnes et la marque des voitures qu'elles ont achetées

→ SELECT *Voiture.Marque*, *Personne.Nom* FROM *Voiture* INNER JOIN *Vente* INNER JOIN *Personne* ON (*Voiture.Code* = *Vente.Code*) AND (*Personne.Ach* = *Vente.Ach*) ;

Question 6

Quelles sont les villes où habitent les personnes qui n'ont pas acheté de voiture ?

→ jointure externe sur *Vente* et *Personne* : on va conserver les champs de la table *Personne* qui ne sont pas liés à des champs de la table *Vente*

SELECT DISTINCT *Personne.Ville* FROM *Personne* LEFT OUTER JOIN *Vente* ON *Personne.Ach* = *Vente.Ach* WHERE *Vente.Ach* IS NULL ;

Question 7

Calculez la moyenne des prix de vente par marque, en ne considérant que les marques dont cette moyenne est supérieure à 12000

→ SELECT Voiture.*Marque*, AVG(Vente.Prix) AS Moyenne
FROM Voiture INNER JOIN Vente ON Vente.Code = Voiture.Code
GROUP BY Voiture.Marque HAVING Moyenne >= 12000

Gestion de tables et de vues (1)

Gestion des tables : généralités

- Fait appel à la partie LMD du langage SQL
- Les opérations de création, de suppression et de modification des tables mettent à jour le **dictionnaire de données** du SGBD
- Dictionnaire de données : structure propre au SGBD qui contient la description des objets (bases de données, tables, colonnes, droits, …)

Gestion de tables et de vues (2)

Création (1)

- L'instruction de création, **CREATE**, permet d'intégrer la description des relations dans le dictionnaire de données de la base
- Elle permet donc de définir le type de données, la clé et les éventuelles contraintes qui devront être respectées
- La syntaxe est la suivante :

```
CREATE TABLE <nom_table>
  (<colonne1>          <format1> [<option1>],
   <colonne2>          <format2> [<option2>], ...
);
```

avec :

- `<nom_table>` : *le nom de la table (relation)*
- `<colonne>` : *le nom de la colonne*
- `<format>` : *le type et la longueur de la colonne*
- `<option>` : *la contrainte d'intégrité sur la colonne*

- Il est également possible de définir des contraintes sur l'ensemble de la table

Gestion de tables et de vues (3)

Création (2)

- Voici une liste (non exhaustive) des types de données SQL

INT	Entier standard (32 bits)
SMALLINT	Entier (16 bits)
REAL	Réel
FLOAT(n)	Réel représenté sur *n* bits

CHAR(n)	Chaîne de caractères de longueur *n*
VARCHAR(n)	Chaîne de caractères de longueur max *n*

DATE	Date
TIME[(n)]	Heure, n(optionnel) est le nombre de décimales représentant la fraction de secondes

BOOLEAN	Booléen
BLOB	*Binary Large Object* : pour stocker tout type de fichier binaire

Gestion de tables et de vues (4)

Création (3)

- Exemple de création de table :

```
CREATE TABLE Voiture (
   numVoit INT,
   marque CHAR(40),
   type CHAR(30),
   couleur CHAR(20)
   );
```

- Il est possible de créer une table temporaire, qui sera effacée à la fin de la session de l'utilisateur, par le mot clé **TEMPORARY**
- Une table peut être directement issue du résultat d'une requête

```
CREATE TEMPORARY TABLE Temp (
   identifiant INT,
   jour DATE,
   valide BOOLEAN
   );
```

```
CREATE TEMPORARY TABLE Resultat (
AS
   ( SELECT Marque, Couleur FROM Voiture );
```

Gestion de tables et de vues (5)

Contraintes d'intégrité (1)

- Les données de la base respectent un ensemble de conditions, appelées **contraintes d'intégrité**
- Il existe 3 types de contraintes d'intégrité :
  - les règles générales : type de la donnée, largeur de la donnée
  - les règles d'intégrité au niveau colonne (PRIMARY KEY, NOT NULL, UNIQUE, CHECK) et au niveau table
  - l'intégrité référentielle
- Les contraintes d'intégrité de colonne sont déclarées de la façon suivante :

```
[CONSTRAINT <nom_contrainte>] type_contrainte
```

  → Le mot clé **CONSTRAINT** ainsi que le nom sont facultatifs
- Les contraintes d'intégrité de tables sont déclarées ainsi :

```
[CONSTRAINT <nom_contrainte>] (colonne1 [,colonne2,...])
```

- Les contraintes de colonnes sont définies après la déclaration de la colonne sur laquelle elles portent
- Les contraintes de tables sont définies après la déclaration de toutes les colonnes

Gestion de tables et de vues (6)

Contraintes d'intégrité (2)

- Les types de contraintes sont les suivants :
  - **NOT NULL** : impose de donner une valeur à une colonne
  - **UNIQUE** : interdit 2 valeurs identiques pour 1 ou plusieurs colonnes
  - **PRIMARY KEY** : indique la clé primaire de la table (NOT NULL + UNIQUE). La contrainte peut concerner une ou plusieurs colonnes
  - **DEFAULT** : permet de donner une valeur par défaut à la colonne si aucune valeur n'est spécifiée lors de l'insertion (contrainte non nommée)
  - **CHECK (condition)** : spécifie une restriction sur les valeurs admises dans une colonne. Condition est une expression conditionnelle. La contrainte peut concerner une ou plusieurs colonnes et peut comparer des colonnes. Elle ne contient pas de sous-requête
  - **FOREIGN KEY… REFERENCES nom_table(nom_colonne)** : relation de clé étrangère avec une ou plusieurs colonnes référencées comme clés primaires
  - **ON DELETE [CASCADE]** : sans cette option, un tuple de la table "mère" ne peut être mis à jour ou supprimé s'il est référencé par une clé étrangère

Gestion de tables et de vues (7)

Contraintes d'intégrité (3)

- Exemple 1

```
CREATE TABLE Personne (
   ine INT PRIMARY KEY,
   nom CHAR(20) NOT NULL,
   age INT
   );
```

```
CREATE TABLE Personne (
   ine INT CONSTRAINT num_ine PRIMARY KEY,
   nom CHAR(20) CONSTRAINT pers_nom NOT NULL,
   age INT
   );
```

- Exemple 2 : domaines → CHECK

```
CREATE TABLE Voiture (
   immat INT PRIMARY KEY,
   couleur CHAR(40),
   CHECK couleur IN ('Gris', 'Noir')
   );
```

```
CREATE TABLE Personne (
   ine INT PRIMARY KEY,
   age CHAR(40),
   CHECK (age BETWEEN 30 AND 40)
   );
```

- Exemple 3 : contraintes sur plusieurs colonnes

```
CREATE TABLE Vente (
   date_vente DATE,
   num_ach INT,
   num_voit INT,
   PRIMARY KEY (num-ach, num_voit) );
```

Gestion de tables et de vues (8)

Modification de la structure (1)

- L'instruction **ALTER** permet de modifier la structure d'une table existante en :
  - rajoutant des colonnes
  - modifiant les caractéristiques des colonnes existantes
- La syntaxe est la suivante :

```
ALTER TABLE <nom_table>
ADD(<colonne1>        <format1> [<option1>],
    <colonne2>        <format2> [<option2>], ...
);
```

ou

```
ALTER TABLE <nom_table>
MODIFY(<colonne1>     <format1> [<option1>],
       <colonne2>     <format2> [<option2>], ...
);
```

- L'instruction **DROP** permet de supprimer entièrement une table
- La syntaxe est la suivante :

```
DROP TABLE <nom_table> ;
```

Gestion de tables et de vues (9)

Modification de la structure (2)

- Si une table référence une autre (clé étrangère), il faut la supprimer en premier
- Pour éviter de gérer les dépendances lors de la suppression, on peut utiliser la clause **CASCADE CONSTRAINTS**
- Exemples

```
ALTER TABLE Personne (
   ADD (Nbfact INT NOT NULL)
   );
```

```
DROP TABLE Personne CASCADE CONSTRAINTS
```

```
ALTER TABLE Fournisseur
   MODIFY nom VARCHAR(100) NOT NULL;
```

- Il est également possible de supprimer une colonne via **DROP COLUMN** :

```
ALTER TABLE Personne (
   DROP COMUN Nom
   );
```

Gestion de tables et de vues (10)

Modification de la structure (3)

- Pour renommer une table, il faut utiliser l'instruction **RENAME**
- La syntaxe est la suivante :

```
RENAME <ancien_nom> TO [nouveau_nom]
```

- Exemple :

```
CREATE TABLE Travail
   AS SELECT Code_cli, Nom_Cli FROM Client ;
DROP TABLE Client ;
RENAME Travail TO Client
```

- L'instruction ALTER permet aussi d'ajouter, de supprimer ou d'activer des contraintes :

```
ALTER TABLE Client
ADD CONSTRAINT ville_ct CHECK (Ville IN ('Nancy', 'Nantes')) ;
```

```
ALTER TABLE Client
DISABLE CONSTRAINT ville_ct ;
```

Gestion de tables et de vues (11)

Vue (1)

- Une vue est une table virtuelle calculée lors de son utilisation. Elle n'a pas d'existence propre
- A l'appel de la vue, le SGBD reconstruit son contenu
- Il existe 2 sortes de vues :
  - les vues mono-table : elles peuvent êtres utilisées en mise à jour → une modification dans la vue entraîne une modification dans la table
  - les vues multi-table : elles ne peuvent pas être utilisées en mise à jour
- Les vues permettent d'assure la confidentialité de la BDD en construisant des sous-schémas adaptés à chaque utilisateur
- Le mot-clé est **VIEW** ; la syntaxe est la suivante :

```
CREATE [OR REPLACE] VIEW <nom_vue> [(colonne1, ...)]
AS SELECT ...
[WITH CHECK OPTION [CONSTRAINT nom_cont]];
```

- La suppression d'une vue se fait simplement avec la commande **DROP** :

```
DROP VIEW <nom_vue>
```

Gestion de tables et de vues (12)

Vue (2)

- Exemples

```
CREATE VIEW Bon_clients AS
   SELECT * FROM Client
   WHERE code IN (SELECT code FROM Facture WHERE Montant > 500) ;
```

```
CREATE VIEW Produit_Prix (Code, Libelle, Prix_HT, Prix_TTC) AS
   SELECT Code, Libelle, Pu, Pu*1.196 FROM Produit ;
```

```
CREATE VIEW Fact_client (Nom_Client, Montant) AS
   SELECT Client.NomCli, Facture.Montant FROM Client, Facture
   WHERE Client.CodeCli = Facture.CodeCli ;
```

Gestion de données (1)

Gestion des données : généralités

- Gestion du contenu des tables : insertion, suppression, mise à jour
- L'insertion se fait enregistrement par enregistrement
- ll est également possible de faire une lecture d'un fichier externe pour constituer le contenu d'une table
- Les opérations de suppression et de modifications de données se font à partir de critères de sélection des enregistrements
- Ces critères s'expriment de la même manière que pour les opérations de sélection
- Il est possible d'utiliser le résultat d'une requête pour déterminer un critère de sélection

Gestion de données (2)

Insertion de données

- L'insertion se fait via l'instruction **INSERT INTO**
- Elle peut se faire de 2 manières : par saisie ou à partir d'une autre table
- Pour être insérées, les valeurs des colonnes doivent respecter les contraintes d'intégrité associées à la table
- La syntaxe est la suivante :

```
INSERT INTO <nom_table>
      [(<colonne1>, <colonne2>, ...)]
      VALUES (<valeur1>, <valeur2>, ...) ;
```

ou

```
INSERT INTO <nom_table>
      [(<colonne1>, <colonne2>, ...)]
      SELECT ... FROM ... ;
```

- Si certaines colonnes sont omises, elles prennent la valeur **NULL**

```
INSERT INTO Client
   VALUES ('DF13', 'Jean-Jacques', 'Nancy') ;
```

```
INSERT INTO NouveauProduit
   SELECT * FROM Produit WHERE PU > 100 ;
```

Gestion de données (3)

Modification (1)

- La modification se fait via l'instruction **UPDATE**
- Pour cette opération, il faut spécifier :
  - la (les) colonne(s) concernée(s)
  - la (les) nouvelle(s) valeur(s)
  - les enregistrements pour lesquels on modifiera ces valeurs
- La syntaxe est la suivante :

```
UPDATE <nom_table>
SET (<col1> = <expression1> [, <col2> = <expression2>, ...]
[WHERE <condition>]
```

ou

```
UPDATE <nom_table>
      SET (<colonne1>, <colonne2>, ...) = <sous-requête>
      [WHERE <condition>]
```

- Les valeurs fournies doivent être compatibles en nombre et en type avec les colonnes à mettre à jour

Gestion de données (4)

Modification (2)

- Exemples

```
UPDATE Personne
   SET Ville = 'Paris-Centre'
   WHERE Ville = 'Paris' ;
```

```
UPDATE Produit
   SET PU = PU * 1.1 ;
```

```
UPDATE Produit
   SET PU = (SELECT PU FROM PRODUIT WHERE Code = 'CO12')
   WHERE Code = 'C011' ;
```

Gestion de données (5)

Suppression

- La suppression d'enregistrements se fait via l'instruction **DELETE FROM**
- La syntaxe est la suivante :

```
DELETE FROM <nom_table>
       [WHERE <condition>]
```

- Exemples :

```
DELETE FROM Voiture
   WHERE couleur = 'Rouge' ;
```

```
DELETE FROM Personne ;
```

Exercice 9 : retour sur l'algèbre relationnelle (1)

- On considère la base de données suivante :
  - **Chercheur** [NC, NOMC]
  - **Projet** [NP, NOMP, NE, BUDGET]
  - **Equipe** [NE, NOME]
  - **Aff** [#NC, #NP]
- Les tables chercheur, Projet et Equipe décrivent les chercheurs, les projets et les équipes. La table **Aff** associe les chercheurs aux projets sur lesquels ils travaillent
- Les règles sont les suivantes :
  - une équipe comprend plusieurs chercheurs
  - un chercheur n'appartient qu'à une équipe
  - une équipe développe plusieurs projets
  - un projet n'est développé que par une équipe
  - les chercheurs d'un projet appartiennent à une même équipe

Exercice 9 : retour sur l'algèbre relationnelle (2)

Equipe

NE	NOME
e1	Merlin
e2	Parole

Chercheur

NC	NOMC
ch1	Jean
ch2	Jacques
ch3	Paul
ch4	Pierre
ch5	François
ch6	Mathieu
ch7	Jean
ch8	Florent
ch9	Michel

Aff

NC	NP
ch3	p5
ch2	p5
ch2	p4
ch7	p4
ch6	p4
ch2	p6
ch3	p6
ch6	p6
ch2	p3
ch1	p1
ch4	p1
ch1	p2
ch4	p2
ch1	p8
ch2	p9
ch5	p8
ch4	p8

ch5	p2
ch8	p7
ch8	p1
ch8	p2
ch8	p8

Projet

NP	NOMP	NE	BUDGET
p7	VACBI	e2	120000
p5	SRI	e1	160000
p1	HYPERMEDIA	e2	130000
p4	BIG	e1	110000
p6	DIABETO	e1	200000
p8	QUAERO	e2	500000
p9	VORTEX	e1	250000
p2	IMAGE	e2	140000
p3	VIDEO	e1	120000

Exercice 9 : retour sur l'algèbre relationnelle (3)

Question 1

Restituer le nom du chercheur « ch1 »

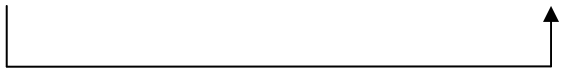
→ **RQ1 = $\prod_{NOMC}$ ($\sigma_{NC = 'ch1'}$ Chercheur)**

→ **SELECT** NOMC **FROM** Chercheur **WHERE** NC **=** 'ch1' **;**

Question 2

Restituer le nom des chercheurs travaillant sur le projet « p5 »

→ **Aff** [NP, NC] **Chercheur** [NC, NOMC]



R1 = $\sigma_{NP = 'p5'}$ Aff

R2 = R1 $\triangleleft_{(R1.NC = Chercheur.NC)}$ Chercheur

RQ2 = $\prod_{NOMC}$ R2

→ **SELECT** NOMC **FROM** Aff **INNER JOIN** Chercheur **ON** Aff.NC **=** Chercheur.NC
WHERE NP **=** 'p5' **;**

→ **SELECT** NOMC **FROM** Chercheur **WHERE** NC **IN**
(SELECT NC **FROM** Aff **WHERE** NP **=** 'p5'**) ;**

Exercice 9 : retour sur l'algèbre relationnelle (4)

Question 3

Restituer par chercheur le nombre de projets sur lequel ils travaillent

→ Agrégation avec :

- AGG = COUNT
- critère de regroupement = chercheur (NC)
- attribut = projet (NP)

RQ3 = COUNT(Aff ; NC ; NP)

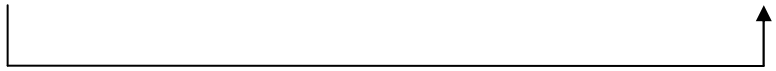
→ **SELECT** NC**, COUNT(**NP**) FROM** Aff **GROUP BY** NC **;**

Exercice 9 : retour sur l'algèbre relationnelle (5)

Question 4

Restituer par chercheur le nombre de projets sur lequel ils travaillent (nom du chercheur ↔ nombre de projets)

→ **RQ3** [NC, COUNT_NP] **Chercheur** [NC, NOMC]



R1 = RQ3 $\triangleright\triangleleft_{NC}$ **Chercheur**

RQ4 = $\prod_{NOMC,\ COUNT\_NP}$ **R1**

→ **SELECT** NOMC**, COUNT(**NP**)**
FROM Aff **INNER JOIN** Chercheur **ON** Aff.NC **=** Chercheur.NC
GROUP BY NOMC

Exercice 9 : retour sur l'algèbre relationnelle (6)

Question 5

Restituer les noms des chercheurs qui ne travaillent pas sur le projet « IMAGE »

→ **R1 = $\sigma_{NOMP = 'IMAGE'}$ Projet**

R2 = Aff $\triangleright_{NP}$ R1

R3 = $\prod_{NC}$ R2

R4 = $\prod_{NC}$ Chercheur

R5 = R4 – R3

R6 = Chercheur $\triangleright_{NC}$ R5

RQ7 = $\prod_{NOMC}$ R6

→ **SELECT** NOMC **FROM** Chercheur
WHERE NC **NOT IN**
(SELECT NC **FROM** Aff **WHERE** NP **=**
(SELECT NP **FROM** Projet **WHERE** NOMP = 'IMAGE'**)) ;**