MySQL / SQL EXEMPLES

Typologie du langage

Il est possible d'inclure des requêtes SQL dans un programme écrit dans un autre langage (en langage C par exemple), ainsi que d'envoyer directement les requêtes SQL telles quelles au SGBD.

Il est possible d'ajouter des commentaires grâce:

- au caractère %. Tous les caractères situés après celui-ci sur la même ligne ne seront pas interprétés
- aux délimiteurs /* et */. Tous les caractères compris entre les délimiteurs sont considérés comme des commentaires


Les commentaires ne peuvent pas être imbriqués!

Le langage SQL n'est pas sensible à la casse (en anglais *case sensitive*), cela signifie que l'on peut aussi bien écrire les instructions en minuscules qu'en majuscule. Toutefois, cette insensibilité à la casse n'est que partielle dans la mesure où la différenciation entre minuscules et majuscules existe au niveau des identificateurs d'objets.

Syntaxe de la commande SELECT

La commande *SELECT* est basée sur l'algèbre relationnelle, en effectuant des opérations de sélection de données sur plusieurs tables relationnelles par projection. Sa syntaxe est la suivante:

```
SELECT [ALL] | [DISTINCT] <liste des noms de colonnes> | *

FROM <Liste des tables>

[WHERE <condition logique>]
```

Il existe d'autres options pour la commande *SELECT*:

```
GROUP BY

HAVING

ORDER BY
```

- L'option **ALL** est, par opposition à l'option *DISTINCT*, l'option par défaut. Elle permet de sélectionner l'ensemble des lignes satisfaisant à la condition logique
- L'option **DISTINCT** permet de ne conserver que des lignes distinctes, en éliminant les doublons
- La **liste des noms de colonnes** indique la liste des colonnes choisies, séparées par des virgules. Lorsque l'on désire sélectionner l'ensemble des colonnes d'une table il n'est pas nécessaire de saisir la liste de ses colonnes, l'option ***** permet de réaliser cette tâche
- La **liste des tables** indique l'ensemble des tables (séparées par des virgules) sur lesquelles on opère
- La **condition logique** permet d'exprimer des qualifications complexes à l'aide d'opérateurs logiques et de comparateurs arithmétiques

La création de tables

Le création de tables se fait à l'aide du couple de mots-clés *CREATE TABLE*. La syntaxe de définition simplifiée d'une table est la suivante:

```
CREATE TABLE Nom_de_la_table (Nom_de_colonne1 Type_de_donnée,
                              Nom_de_colonne2 Type_de_donnée,
                              ...);
```

Le nom donné à la table doit généralement (sur la plupart des SGBD) commencer par une lettre, et le nombre de colonnes maximum par table est de 254.

Les types de données

Pour chaque colonne que l'on crée, il faut préciser le type de données que le champ va contenir. Celui-ci peut être un des types suivants:

Type de donnée	Syntaxe	Description
Type alphanumérique	CHAR(n)	Chaîne de caractères de longueur fixe *n* (n<16383)
Type alphanumérique	VARCHAR(n)	Chaîne de caractères de n caractères maximum (n<16383)
Type numérique	NUMBER(n,[d])	Nombre de n chiffres [optionnellement *d* après la virgule]
Type numérique	SMALLINT	Entier signé de 16 bits (-32768 à 32757)
Type numérique	INTEGER	Entier signé de 32 bits (-2E31 à 2E31-1)
Type numérique	FLOAT	Nombre à virgule flottante
Type horaire	DATE	Date sous la forme 16/07/99
Type horaire	TIME	Heure sous la forme 12:54:24.85
Type horaire	TIMESTAMP	Date et Heure

L'option *NOT NULL*, placée immédiatement après le type de donnée permet de préciser au système que la saisie de ce champ est obligatoire

Qu'est-ce qu'une vue?

Une vue est une table virtuelle, c'est-à-dire dont les données ne sont pas stockées dans une table de la base de données, et dans laquelle il est possible de rassembler des informations provenant de plusieurs tables. On parle de "vue" car il s'agit simplement d'une représentation des données dans le but d'une exploitation visuelle. Les données présentes dans une vue sont définies grâce à une clause *SELECT*

Création d'une vue en SQL

La création d'une vue se fait grâce à la clause *CREATE VIEW* suivie du nom que l'on donne à la vue, puis du nom des colonnes dont on désire agrémenter cette vue (il faut autant de redéfinitions de colonne qu'il y en aura en sortie), puis enfin d'une clause *AS* précédant la sélection. La syntaxe d'une vue ressemble donc à ceci:

```
CREATE VIEW Nom_de_la_Vue
(colonnes)
AS SELECT ...
```

Voici ce que cela pourrait donner:

```
CREATE VIEW Vue
(colonneA,colonneB,colonneC,colonneD)
AS SELECT colonne1,colonne2,colonneI,colonneII
FROM Nom_table1 Alias1,Nom_tableII AliasII
WHERE Alias1.colonne1 = AliasII.colonneI
```

```
AND Alias1.colonne2 = AliasII.colonneII
```
Les vues ainsi créées peuvent être l'objet de nouvelles requêtes en précisant le nom de la vue au lieu d'un nom de table dans un ordre SELECT...

Intérêts des vues

La vue représente de cette façon une sorte d'intermédiaire entre la base de données et l'utilisateur. Celà a de nombreuses conséquences:

- une sélection des données à afficher
- une restriction d'accès à la table pour l'utilisateur, c'est-à-dire une sécurité des données accrue
- un regroupement d'informations au sein d'une entité

Qu'est-ce qu'un index?

Un index est un objet complémentaire (mais non indispensable) à la base de données permettant d'"indexer" certaines colonnes dans le but d'améliorer l'accès aux données par le SGBDR, au même titre qu'un index dans un livre ne vous est pas indispensable mais vous permet souvent d'économiser du temps lorsque vous recherchez une partie spécifique de ce dernier...

Toutefois la création d'index utilise de l'espace mémoire dans la base de données, et, étant donné qu'il est mis à jour à chaque modification de la table à laquelle il est rattaché, peut alourdir le temps de traitement du SGBDR lors de la saisie de données. Par conséquent il faut que la création d'index soit justifiée et que les colonnes sur lesquelles il porte soient judicieusement choisies (de telle façon à minimiser les doublons). De cette façon certains SGBDR créent automatiquement un index lorsqu'une clé primaire est définie.

La création d'un index

La création d'index en SQL se fait grâce à la clause *INDEX* précédée de la clause *CREATE*. Elle permet de définir un index désigné par son nom, portant sur certains champs d'une table. La syntaxe est la suivante:

```
CREATE [UNIQUE] INDEX Nom_de_l_index

ON Nom_de_la_table

(Nom_de_champ [ASC/DESC], ...)
```

- L'option UNIQUE permet de définir la présence ou non de doublons pour les valeurs de la colonne
- Les options ASC/DESC permettent de définir un ordre de classement des valeurs présentes dans la colonne

Intérêt des permissions

Plusieurs personnes peuvent travailler simultanément sur une base de données, toutefois ces personnes n'ont pas forcément les mêmes besoins: certaines peuvent par exemple nécessiter de modifier des données dans la table, tandis que les autres ne l'utiliseront que pour la consulter. Ainsi, il est possible de définir des permissions pour chaque personne en leur octroyant un mot de passe. Cette tâche incombe à l'administrateur de la base de données (en anglais *DBA*, DataBase Administrator). Il doit dans un premier temps définir les besoins de chacuns, puis les appliquer à la base de donnée sous forme de permissions. Le langage SQL permet d'effectuer ces opérations grâce à deux clauses:

- *GRANT* permet d'accorder des droits à un (parfois plusieurs sur certains SGBD) utilisateur
- *REVOKE* permet de retirer des droits à un (ou plusieurs sur certains SGBD) utilisateur

Les permissions (appelées aussi *droits* ou *privilèges*) peuvent être définies pour chaque (un grand nombre) clause.

D'autre part il est aussi possible de définir des rôles c'est-à-dire de permettre à d'autre utilisateurs d'accorder des permissions.

Les privilèges

Les privilèges sont les clauses qui peuvent être autorisées/retirées à un utilisateur. Les principales sont:

- *DELETE*: privilège de supprimer les données d'une table
- *INSERT*: privilège d'ajouter des données à une table
- *SELECT*: privilège d'accèder aux données d'une table
- *UPDATE*: privilège de mettre à jour les données d'une table

Qui peut accorder/retirer des permissions?

L'unique personne pouvant accorder ou retirer des droits sur un élément (table, vue ou index) est la personne qui l'a créé. Toutefois, il lui est possible de transmettre ce droit d'accorder/retirer des droits, auquel cas la personne recevant cet "honneur" aura le droit de transmettre ce "pouvoir" sur ces éléments

Expression des jointures

Une jointure (ou θ-jointure) est un produit cartésien de deux tables. On appelle équijointure une θ-jointure dont la qualification est une égalité entre deux colonnes.
En SQL, l'expression d'une jointure se fait en précisant le nom des colonnes des tables sur lesquelles on fait la jointure, en désignant les colonnes des différentes tables en écrivant le nom de la table, suivie d'un point puis du nom de la colonne. La clause *WHERE* permet de préciser la qualification de la jointure.

Soit les deux tables suivantes:

Table Occaz

Marque	Modele	Serie	Numero	Compteur
Renault	18	RL	4698 SJ 45	123450
Renault	Kangoo	RL	4568 HD 16	56000
Renault	Kangoo	RL	6576 VE 38	12000
Peugeot	106	KID	7845 ZS 83	75600
Peugeot	309	chorus	7647 ABY 82	189500
Ford	Escort	Match	8562 EV 23	
Fiat	Punto	GTI	8941 UD 61	
Audi	A4	Quattro	7846 AZS 75	21350

Table Societe

Nom	Pays
Renault	France
Fiat	Italie
Peugeot	France
Volkswagen	Allemagne
Ford	Etats-Unis

- L'affichage des pays d'origine des voitures par marque/modèle se fait par l'instruction:
- ```
  SELECT Occaz.Marque, Occaz.Modele, Societe.Pays FROM OCCAZ,SOCIETE
  WHERE Occaz.Marque = Societe.Nom
  ```

Marque	Modele	Pays
  ```

Renault	18	France
Renault	Kangoo	France
Renault	Kangoo	France
Peugeot	106	France
Peugeot	309	France
Ford	Escort	Etats-Unis
Fiat	Punto	Italie

Il est possible de donner des alias aux noms des tables pour diminuer la taille des requêtes.

```
SELECT O.Marque, O.Modele, S.Pays FROM OCCAZ O,SOCIETE S
WHERE O.Marque = S.Nom
```

Il est possible de supprimer une table grâce à la clause *DROP*, il existe aussi des commandes moins extrêmes permettant

- L'ajout de colonnes
- La modification de colonnes
- La suppression de colonnes

Enfin, il est possible d'ajouter des commentaires à une table grâce à la clause *COMMENT*.

La suppression d'éléments

La clause *DROP* permet d'éliminer des vues, des index et même des tables. Cette clause est toutefois à utiliser avec parcimonie dans la mesure où elle est irréversible.

La suppression d'une vue se fait avec la syntaxe suivante:

```
DROP VIEW Nom_de_la_vue
```

La suppression d'un index se fait avec la syntaxe suivante:

```
DROP INDEX Nom_de_l_index
```

La suppression d'une table se fait avec la syntaxe suivante:

```
DROP TABLE Nom_de_la_table
```

La suppression des données uniquement

La clause *DROP* lorsqu'elle est utilisée sur une table élimine les données ainsi que la structure de la table. Il est possible de supprimer uniquement les données en conservant la structure de la table grâce à la clause *TRUNCATE*.

La suppression des données d'une table se fait avec la syntaxe suivante:

```
TRUNCATE TABLE Nom_de_la_table
```

Renommer une table

Il peut parfois être intéressant de renommer une table, c'est la clause *RENAME* qui permet cette opération. La syntaxe de cette clause est:

```
RENAME Ancien_Nom TO Nouveau_Nom
```

La clause *RENAME* n'est pas implémentée dans tous les SGBDR, consultez la documentation de votre SGBD!

Suppression de colonnes

La clause *ALTER* permet la modification des colonnes d'un table. Associée avec la clause *DROP COLUMN*, elle permet de supprimer des colonnes. La syntaxe est la suivante:

```
ALTER TABLE Nom_de_la_table
DROP COLUMN Nom_de_la_colonne
```

Il faut noter que la suppression de colonnes n'est possible que dans le cas où:

- La colonne ne fait pas partie d'une vue
- La colonne ne fait pas partie d'un index
- La colonne n'est pas l'objet d'une contrainte d'intégrité

Ajout de colonnes

Associée avec la clause *ADD*, la clause *ALTER* permet l'ajout de colonnes à une table. La syntaxe est la suivante:

```
ALTER TABLE Nom_de_la_table
ADD Nom_de_la_colonne Type_de_donnees
```

Modification de colonnes

Associée avec la clause *MODIFY*, la clause *ALTER* permet la modification du type de données d'une colonne. La syntaxe est la suivante:

```
ALTER TABLE Nom_de_la_table
MODIFY Nom_de_la_colonne Type_de_donnees
```

Sous Access 2000, la syntaxe est la suivante :

```
ALTER TABLE Nom_de_la_table
ALTER COLUMN Nom_de_la_colonne Type_de_donnees
```

Les opérations ensemblistes

Les opérations ensemblistes en SQL, sont celles définies dans l'algèbre relationnelle. Elles sont réalisées grâce aux opérateurs:

- UNION
- INTERSECT (ne fait pas partie de la norme SQL et n'est donc pas implémenté dans tous les SGBD)
- EXCEPT (ne fait pas partie de la norme SQL et n'est donc pas implémenté dans tous les SGBD)

Ces opérateurs s'utilisent entre deux clauses *SELECT*.

L'opérateur *UNION*

Cet opérateur permet d'effectuer une UNION des tuples sélectionnés par deux clauses *SELECT* (les deux tables sur lesquelles on travaille devant avoir le même schéma).

```
SELECT ---- FROM ---- WHERE ------
UNION
SELECT ---- FROM ---- WHERE ------
```

Par défaut les doublons sont automatiquement éliminés. Pour conserver les doublons, il est possible d'utiliser une clause *UNION ALL*.

L'opérateur *INTERSECT*

Cet opérateur permet d'effectuer une INTERSECTION des tuples sélectionnés par deux clauses *SELECT* (les deux tables sur lesquelles on travaille devant avoir le même schéma).

```
SELECT ---- FROM ---- WHERE ------

INTERSECT

SELECT ---- FROM ---- WHERE ------
```

L'opérateur *INTERSECT* n'étant pas implémenté dans tous les SGBD, il est possible de le remplacer par des commandes usuelles:

```
SELECT a,b FROM table1

WHERE EXISTS (   SELECT c,d FROM table2

                 WHERE a=c AND b=d )
```

L'opérateur *EXCEPT*

Cet opérateur permet d'effectuer une DIFFERENCE entre les tuples sélectionnés par deux clauses *SELECT*, c'est-à-dire sélectionner les tuples de la première table n'appartenant pas à la seconde (les deux tables devant avoir le même schéma).

```
SELECT a,b FROM table1 WHERE ------

EXCEPT

SELECT c,d FROM table2 WHERE ------
```

L'opérateur *EXCEPT* n'étant pas implémenté dans tous les SGBD, il est possible de le remplacer par des commandes usuelles:

```
SELECT a,b FROM table1

WHERE NOT EXISTS (       SELECT c,d FROM table2

                 WHERE a=c AND b=d )
```

Expression d'une projection

Une projection est une instruction permettant de sélectionner un ensemble de colonnes dans une table. Soit la table *VOITURE* suivante:

Voiture

Marque	Modele	Serie	Numero
Renault	18	RL	4698 SJ 45
Renault	Kangoo	RL	4568 HD 16
Renault	Kangoo	RL	6576 VE 38
Peugeot	106	KID	7845 ZS 83
Peugeot	309	chorus	7647 ABY 82
Ford	Escort	Match	8562 EV 23

- La sélection de toutes les colonnes de la table se fait par l'instruction:

```
SELECT * FROM VOITURE
```

Résultat

Marque	Modele	Serie	Numero
Renault	18	RL	4698 SJ 45
Renault	Kangoo	RL	4568 HD 16
Renault	Kangoo	RL	6576 VE 38
Peugeot	106	KID	7845 ZS 83
Peugeot	309	chorus	7647 ABY 82
Ford	Escort	Match	8562 EV 23

- La sélection des colonnes *Modèle* et *Série* de la table se fait par l'instruction:

```
SELECT Modele, Serie FROM VOITURE
```

Résultat

Modele	Serie
18	RL
Kangoo	RL
Kangoo	RL
106	KID
309	chorus
Escort	Match

- La sélection des colonnes *Modèle* et *Série* en éliminant les doublons se fait par l'instruction:

```
SELECT DISTINCT Modele, Serie FROM VOITURE
```

Résultat

Modele	Serie
18	RL
Kangoo	RL
106	KID
309	chorus
Escort	Match

Expression des restrictions

Une restriction consiste à sélectionner les lignes satisfaisant à une condition logique effectuée sur leurs attributs.
En SQL, les restrictions s'expriment à l'aide de la clause *WHERE* suivie d'une condition logique exprimée à l'aide d'opérateurs logiques

- AND
- OR
- NOT

de comparateurs de chaîne:

- IN
- BETWEEN
- LIKE

d'opérateurs arithmétiques:

- +
- -
- *
- /
- %
- &
- |
- ^
- ~

et de comparateurs arithmétiques:

- =
- !=
- >
- <
- >=
- <=
- <>
- !>
- !<

Restrictions simples

Soit la table suivante, présentant des voitures d'occasion:

Occaz

Marque	Modele	Serie	Numero	Compteur
Renault	18	RL	4698 SJ 45	123450
Renault	Kangoo	RL	4568 HD 16	56000
Renault	Kangoo	RL	6576 VE 38	12000
Peugeot	106	KID	7845 ZS 83	75600
Peugeot	309	chorus	7647 ABY 82	189500
Ford	Escort	Match	8562 EV 23	

Le champ présentant la valeur du kilométrage au compteur de la *Ford Escort* est délibérément non renseigné.

- La sélection de toutes les voitures d'occasion ayant un kilométrage inférieur à 100 000 Km se fait par l'instruction:
- `SELECT * FROM OCCAZ`
- `WHERE (Compteur < 100000)`

  Résultat

Marque	Modele	Serie	Numero	Compteur
Renault	Kangoo	RL	4568 HD 16	56000
Renault	Kangoo	RL	6576 VE 38	12000
Peugeot	106	KID	7845 ZS 83	75600

- La sélection des colonnes *Marque* et *Compteur* des voitures ayant un kilométrage inférieur à 100 000 Km se fait par l'instruction:
- `SELECT Marque,Compteur FROM OCCAZ`
- `WHERE (Compteur < 100000)`

Résultat

Marque	Compteur
Renault	56000
Renault	12000
Peugeot	75600

- La sélection de toutes les voitures d'occasion ayant un kilométrage inférieur ou égal à 100 000 Km, et supérieur ou égal à 30000Km, se fait par l'instruction:
- `SELECT * FROM OCCAZ`
- `WHERE (Compteur <= 100000) AND (Compteur >= 30000)`

Résultat

Marque	Modele	Serie	Numero	Compteur
Renault	Kangoo	RL	4568 HD 16	56000
Peugeot	106	KID	7845 ZS 83	75600

Restriction sur une comparaison de chaîne

Le prédicat *LIKE* permet de faire des comparaisons sur des chaines grâce à des caractères, appelés caractères *jokers*:

- Le caractère **%** permet de remplacer une séquence de caractères (éventuellement nulle)
- La caractère **_** permet de remplacer un caractère (l'équivalent du "blanc" au scrabble...)
- Les caractères **[-]** permettent de définir un intervalle de caractères (par exemple [J-M])

- La sélection des voitures dont la marque a un E en deuxième position se fait par l'instruction:
- `SELECT * FROM OCCAZ`
- `WHERE Marque LIKE "_E%"`

Résultat

Marque	Modele	Serie	Numero	Compteur
Renault	18	RL	4698 SJ 45	123450
Renault	Kangoo	RL	4568 HD 16	56000
Renault	Kangoo	RL	6576 VE 38	12000
Peugeot	106	KID	7845 ZS 83	75600
Peugeot	309	chorus	7647 ABY 82	189500

Suivant l'environnement ou vous utilisez ce prédicat, il sera peut-être nécessaire d'"échapper" les guillemets avec un caractère d'échappement (généralement la barre oblique inverse "\").

Restriction sur un ensemble

Les prédicats *BETWEEN* et *IN* permettent de vérifier respectivement qu'une valeur se trouve dans un intervalle ou qu'une valeur appartient à une liste de valeurs:

- La sélection de toutes les voitures d'occasion ayant un kilométrage inférieur ou égal à 100 000 Km, mais supérieur ou égal à 30000Km, (effectuée plus haut avec des comparateurs arithmétiques) peut se faire par l'instruction:
- `SELECT * FROM OCCAZ`
- `WHERE Compteur BETWEEN 100000 AND 30000`

Résultat

Marque	Modele	Serie	Numero	Compteur
Renault	Kangoo	RL	4568 HD 16	56000
Peugeot	106	KID	7845 ZS 83	75600

- La sélection des voitures d'occasion dont la marque est Peugeot ou Ford se fait grâce à l'instruction:
- `SELECT * FROM OCCAZ`
- `WHERE Marque IN ("Peugeot", "Ford")`

Résultat

Marque	Modele	Serie	Numero	Compteur
Peugeot	106	KID	7845 ZS 83	75600
Peugeot	309	chorus	7647 ABY 82	189500
Ford	Escort	Match	8562 EV 23	

Restriction sur les valeurs manquantes

Lorsqu'un champ n'est pas renseigné, le SGBD lui attribue une valeur spéciale que l'on note *NULL*. La recherche de cette valeur ne peut pas se faire à l'aide des opérateurs standards, il faut utiliser les prédicats *IS NULL* ou bien *IS NOT NULL*.

- La sélection de toutes les voitures d'occasion dont le kilométrage n'est pas renseigné se fait par l'instruction:
- `SELECT * FROM OCCAZ`
- `WHERE Compteur IS NULL`

Résultat

Marque	Modele	Serie	Numero	Compteur
Ford	Escort	Match	8562 EV 23	

Tri des résultats

Il est possible en SQL d'organiser les tuples fournis en résultat grâce à la clause *ORDER BY*. La clause *ORDER BY* est suivie des mots clés *ASC* ou *DESC*, qui précisent respectivement si le tri se fait de manière croissante (par défaut) ou décroissante. Le classement se fait sur des nombres ou des chaines de caractères.
Prenons l'exemple de la table **voiture** :

Marque	Modele	Serie	Numero	Compteur
Renault	18	RL	4698 SJ 45	123450
Renault	Kangoo	RL	4568 HD 16	56000
Renault	Kangoo	RL	6576 VE 38	12000
Peugeot	106	KID	7845 ZS 83	75600
Peugeot	309	chorus	7647 ABY 82	189500
Ford	Escort	Match	8562 EV 23	

- La sélection de toutes les colonnes de la table triées par ordre croissant de l'attribut *Marque* se fait par l'instruction:
- `SELECT * FROM VOITURE`
- `ORDER BY Marque ASC`

Marque	Modele	Serie	Numero	Compteur
Ford	Escort	Match	8562 EV 23	
Peugeot	106	KID	7845 ZS 83	75600
Peugeot	309	chorus	7647 ABY 82	189500
Renault	18	RL	4698 SJ 45	123450
Renault	Kangoo	RL	4568 HD 16	56000
Renault	Kangoo	RL	6576 VE 38	12000

- La sélection de toutes les colonnes de la table triées par ordre croissant de l'attribut *Marque*, puis par ordre décroissant du compteur, se fait par l'instruction:
- `SELECT * FROM VOITURE`
- `ORDER BY Marque ASC, Compteur DESC`

Marque	Modele	Serie	Numero	Compteur
Ford	Escort	Match	8562 EV 23	
Peugeot	309	chorus	7647 ABY 82	189500
Peugeot	106	KID	7845 ZS 83	75600
Renault	18	RL	4698 SJ 45	123450
Renault	Kangoo	RL	4568 HD 16	56000
Renault	Kangoo	RL	6576 VE 38	12000

Regroupement de résultats

Il peut être intéressant de regrouper des résultats afin de faire des opérations par groupe (opérations statistiques par exemple). Cette opération se réalise à l'aide de la clause *GROUP BY*, suivie du nom de chaque colonne sur laquelle on veut effectuer des regroupements.

Les principales fonctions pouvant être effectuées par groupe sont:

- **AVG**: Calcule la moyenne d'une colonne (ou de chaque regroupement si elle est couplée à la clause *GROUP BY*)

- **COUNT**: Calcule le nombre de lignes d'une table (ou de chaque regroupement ...)
- **MAX**: Calcule la valeur maximale d'une colonne (ou de chaque regroupement ...)
- **MIN**: Calcule la valeur minimale colonne (ou de chaque regroupement ...)
- **SUM**: Effectue la somme des valeurs d'une colonne (ou de chaque regroupement ...)

Soit la table VOITURE ci-dessus:

- L'affichage des moyennes des compteurs par marque se fait par l'instruction:
- `SELECT Marque, AVG(Compteur) AS Moyenne FROM VOITURE`
- `GROUP BY Marque`

Marque	Moyenne
Renault	63816.6
Peugeot	132550
Ford	

La clause *HAVING* va de pair avec la clause *GROUP BY*, elle permet d'appliquer une restriction sur les groupes créés grâce à la clause *GROUP BY*.

- L'affichage des moyennes des compteurs non nulles regroupées par marque se fait par l'instruction:
- `SELECT Marque, AVG(Compteur) AS Moyenne FROM VOITURE`
-
- `GROUP BY Marque`
- `HAVING Moyenne IS NOT NULL`

Marque	Moyenne
Renault	63816.6
Peugeot	132550

Remarquez l'utilisation de *AS* pour donner un nom à la colonne créée à l'aide de la fonction *AVG*.

Expression de contraintes d'intégrité

Une contrainte d'intégrité est une clause permettant de contraindre la modification de tables, faite par l'intermédiaire de requêtes d'utilisateurs, afin que les données saisies dans la base soient conformes aux données attendues. Ces contraintes doivent être exprimées dès la création de la table grâce aux mots clés suivants:

- CONSTRAINT
- DEFAULT
- NOT NULL
- UNIQUE
- CHECK

Définir une valeur par défaut

Le langage SQL permet de définir une valeur par défaut lorsqu'un champ de la base n'est pas renseigné grâce à la clause *DEFAULT*. Cela permet notamment de faciliter la création de tables, ainsi que de garantir qu'un champ ne sera pas vide.

La clause *DEFAULT* doit être suivie par la valeur à affecter. Cette valeur peut être un des types suivants:

- constante numérique
- constante alphanumérique (chaîne de caractères)
- le mot clé **USER** (nom de l'utilisateur)
- le mot clé **NULL**
- le mot clé **CURRENT_DATE** (date de saisie)
- le mot clé **CURRENT_TIME** (heure de saisie)
- le mot clé **CURRENT_TIMESTAMP** (date et heure de saisie)

Forcer la saisie d'un champ

Le mot clé *NOT NULL* permet de spécifier qu'un champ doit être saisi, c'est-à-dire que le SGBD refusera d'insérer des tuples dont un champ comportant la clause *NOT NULL* n'est pas renseigné.

Emettre une condition sur un champ

Il est possible de faire un test sur un champ grâce à la clause *CHECK()* comportant une condition logique portant sur une valeur entre les parenthèses. Si la valeur saisie est différente de *NULL*, le SGBD va effectuer un test grâce à la condition logique. Celui-ci peut évenutellement être une condition avec des ordres *SELECT*...

Tester l'unicité d'une valeur

La clause *UNIQUE* permet de vérifier que la valeur saisie pour un champ n'existe pas déjà dans la table. Cela permet de garantir que toutes les valeurs d'une colonne d'une table seront différentes.

Nommer une contrainte

Il est possible de donner un nom à une contrainte grâce au mot clé *CONSTRAINT* suivi du nom que l'on donne à la contrainte, de telle manière à ce que le nom donné s'affiche en cas de non respect de l'intégrité, c'est-à-dire lorsque la clause que l'on a spécifiée n'est pas validée.

Si la clause *CONSTRAINT* n'est pas spécifiée, un nom sera donné arbitrairement par le SGBD. Toutefois, le nom donné par le SGBD risque fortement de ne pas être compréhensible, et ne sera vraisemblablement pas compris lorsqu'il y aura une erreur d'intégrité. La stipulation de cette clause est donc fortement conseillée.

Exemple de création de table avec contrainte

Voici un exemple permettant de voir la syntaxe d'une instruction de création de table avec contraintes:

```
CREATE TABLE clients(
Nom    char(30) NOT NULL,
Prenom char(30) NOT NULL,
Age    integer, check (age < 100),
Email  char(50) NOT NULL, check (Email LIKE "%@%")
)
```

Definition de clés

Grâce à SQL, il est possible de définir des clés, c'est-à-dire spécifier la (ou les) colonne(s) dont la connaissance permet de désigner précisément un et un seul tuple (une ligne).

- L'ensemble des colonnes faisant partie de la table en cours permettant de désigner de façon unique un tuple est appelé **clé primaire** et se définit grâce à la clause *PRIMARY KEY* suivie de la liste de colonnes, séparées par des virgules, entre parenthèses. Ces colonnes ne peuvent alors plus prendre la valeur *NULL* et doivent être telles que deux lignes ne puissent avoir simultanément la même combinaison de valeurs pour ces colonnes.

  ```
  PRIMARY KEY (colonne1, colonne2, ...)
  ```

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

Trigger (gâchette): garantie de l'intégrité référentielle

Les clés étrangères permettent de définir les colonnes d'une table garantissant la validité d'une autre table. Ainsi, il existe des éléments (appelés *triggers*, ou en français *gâchettes* ou *déclencheurs*) permettant de garantir l'ensemble de ces contraintes que l'on désigne par le terme d'**intégrité référentielle**, c'est-à-dire notamment de s'assurer qu'un tuple utilisé à partir d'une autre table existe réellement.
Ces triggers sont *ON DELETE* et *ON UPDATE*:

- *ON DELETE* est suivi d'arguments entre accolades permettant de spécifier l'action à réaliser en cas d'effacement d'une ligne de la table faisant partie de la clé étrangère:
  - *CASCADE* indique la suppression en cascade des lignes de la table étrangère dont les clés étrangères correspondent aux clés primaires des lignes effacées
  - *RESTRICT* indique une erreur en cas d'effacement d'une valeur correspondant à la clé
  - *SET NULL* place la valeur NULL dans la ligne de la table étrangère en cas d'effacement d'une valeur correspondant à la clé
  - *SET DEFAULT* place la valeur par défaut (qui suit ce paramètre) dans la ligne de la table étrangère en cas d'effacement d'une valeur correspondant à la clé
- *ON UPDATE* est suivi d'arguments entre accolades permettant de spécifier l'action à réaliser en cas de modification d'une ligne de la table faisant partie de la clé étrangère:
  - *CASCADE* indique la modification en cascade des lignes de la table étrangères dont les clé primaires correspondent aux clés étrangères des lignes modifiées
  - *RESTRICT* indique une erreur en cas de modification d'une valeur correspondant à la clé
  - *SET NULL* place la valeur NULL dans la ligne de la table étrangère en cas de modification d'une valeur correspondant à la clé

- *SET DEFAULT* place la valeur par défaut (qui suit ce paramètre) dans la ligne de la table étrangère en cas de modification d'une valeur correspondant à la clé

Création d'assertions

Les assertions sont des expressions devant être satisfaites lors de la modifications de données pour que celles-ci puissent être réalisées. Ainsi, elles permettent de garantir l'intégrité des données. Leur syntaxe est la suivante:

```
CREATE ASSERTION Nom_de_la_contrainte CHECK (expression_conditionnelle)
```

La condition à remplir peut (et est généralement) être effectuée grâce à une clause *SELECT*.

Les assertions ne sont pas implémentées dans l'ensemble des SGBDR...