SUPPORT DE TRAVAUX PRATIQUES (SQL)

SQL est un langage informatique standard pour la communication avec les SGBDR. Les commandes SQL se divisent en deux catégories : Langage de Définition de Données et Langage de Manipulation de Données.

Langage de Définition de Données

SQL offre des commandes de définition et de modification des structures d'une base de données. Ces commandes permettent de créer, de modifier et de supprimer des tables, des colonnes, et des contraintes. Parmi ces commandes : CREATE, DROP, ALTER, MODIFY, CHANGE, etc.

CREATE DATABASE

CREATE DATABASE [IF NOT EXISTS] nomBD ;

L'option « IF NOT EXISTS » permet d'éviter d'avoir une erreur si la base existe déjà.

DROP DATABASE

DROP DATABASE [IF EXISTS] nomBD ;

L'option « IF EXISTS » permet d'éviter d'avoir une erreur si la base n'existe pas.

CREATE TABLE

CREATE TABLE nomTable (col1 type1 [Contraintes1], col2 type2 [Contraintes2]…);

nomTable est le nom donné à la table créée.
col1, col2,… représentent les noms des colonnes.
type1, type2,… représentent les types des données contenues dans les colonnes *col1, col2,…* respectivement.
[Contraintesi] représente les contraintes d'intégrité pour la colonne *coli* (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL, et DEFAULT).

DROP TABLE

DROP TABLE nomTable [{CASCADE | RESTRICT}];

Supprime la structure d'une table ainsi que tous les tuples qu'elle contient.
CASCADE : les objets dépendant de la table supprimée (contraintes, etc.) sont automatiquement supprimés.
RESTRICT : c'est l'option par défaut ; la table n'est pas supprimée si un objet en dépend.

ALTER TABLE

ALTER TABLE est utilisé pour modifier la structure (les colonnes, les types de colonnes, les contraintes, etc.) d'une table existante.

Ajout d'une colonne
ALTER TABLE nomTable ADD [COLUMN] col type [Contraintes] ;

Col et ***type*** représentent respectivement le nom et le type de la colonne à ajouter.

Ajout d'une contrainte pour la colonne *col*
ALTER TABLE nomTable ADD [CONSTRAINT nomConst] Contrainte col [autresOptions] ;

nomConst représente le nom attribué explicitement à la contrainte par la clause *CONSTRAINT*.

Contrainte est le type de la contrainte (PRIMARY KEY, FOREIGN KEY, UNIQUE, etc.) définie pour la colonne *col*.
autresOptions concerne la contrainte d'intégrité référentielle.

Modification d'une caractéristique d'une colonne
ALTER TABLE nomTable MODIFY [COLUMN] col nouveauType [Contraintes];

Modification du nom d'une colonne
ALTER TABLE nomTable CHANGE ancienNomCol nouveauNomCol type [Contraintes] ;

Modification du nom (renommer) d'une table
ALTER TABLE nomTable RENAME TO nouveauNomTable ;

Suppression d'une colonne
ALTER TABLE nomTable DROP [COLUMN] col ;

Suppression d'une contrainte existante nommée
ALTER TABLE nomTable DROP CONSTRAINT nomConst ;

CONSTRAINT

Cette clause permet de nommer explicitement des contraintes d'intégrité définies pour les colonnes d'une table. Ces contraintes imposent des conditions qui doivent être satisfaites par les valeurs stockées dans ces colonnes. Par exemple, lors de la création d'une table *module*, il est en toute logique de préciser que la colonne *coefficient* soit un entier positif.

Les différentes contraintes d'intégrité sont : PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL, et DEFAULT.

PRIMARY KEY

… PRIMARY KEY (col1, col2,…)

indique la clé primaire d'une table. Aucune des colonnes de cette clé ne doit avoir une valeur NULL.

FOREIGN KEY

… FOREIGN KEY (col1, col2 …)
REFERENCES tableReference (colRef1, colRef2…)
[ON DELETE {CASCADE | SET NULL | RESTRICT}
ON UPDATE {CASCADE | SET NULL | RESTRICT}]

Contrainte d'intégrité référentielle pour les colonnes *col1, col2…* de la table en cours de définition. Les valeurs prises par ces colonnes doivent exister dans les valeurs des colonnes de la table *tableReference* possédant une contrainte PRIMARY KEY ou UNIQUE.
La clause *FOREIGN KEY* indique que la concaténation de *col1, col2,…* est une clé étrangère faisant référence à la concaténation de *colRef1, colRef2,…* de *tableReference*.

[ON DELETE {CASCADE | SET NUL | RESTRICT}
ON UPDATE {CASCADE | SET NUL | RESTRICT}] indiquent au SGBD l'opération à réaliser lorsque les valeurs de clé étrangère (tuples dans la table *tableReference*) sont supprimées ou modifiées.

UNIQUE

… UNIQUE (col1, col2,…)

La contrainte d'unicité pour les colonnes *col1, col2,…* d'une table, permet de garantir que deux lignes de cette table ne peuvent en aucun cas avoir les mêmes valeurs pour ces colonnes.

CHECK

… CHECK (condition) ;

La contrainte de vérification permet de spécifier que la valeur d'une colonne particulière *col* doit satisfaire une condition booléenne.

NOT NULL

… col type NOT NULL ;

La contrainte de non nullité impose que la valeur d'une colonne soit renseignée (c.à.d. ne soit pas NULL). Par exemple, dans le cas d'un *module*, il est cohérent d'imposer que l'intitulé soit toujours renseigné.

Par défaut (c.à.d. si on ne spécifie aucune valeur), toute colonne est facultative. Le caractère obligatoire d'une colonne se déclarera par la clause *NOT NULL*.

DEFAULT

… col type DEFAULT val ;

val est une valeur par défaut que le SGBD assignera automatiquement à *col* lorsque l'utilisateur omettra d'en fournir une lors de l'insertion d'une ligne.

Langage de Manipulation de Données

SQL offre des commandes permettant l'ajout, la modification, la suppression et l'interrogation de données d'une base. Ces commandes forment le *Langage de Manipulation de Données*. Parmi ces commandes : INSERT INTO, UPDATE, DELETE, SELECT …FROM, etc.

INSERT INTO

INSERT INTO nomTable [(col1, col2,…)]
VALUES (val1, val2,…) ;

permet d'insérer un tuple dans la table *nomTable* en spécifiant les valeurs à insérer. Dans le cas où la liste des noms de colonnes *col1, col2,…* est omise, l'ordre utilisé pour l'insertion est celui spécifié lors de la création de la table. En revanche, dans le cas où une liste est donnée, les colonnes n'y figurant pas auront la valeur NULL ou, le cas échéant, la valeur par défaut.

UPDATE

UPDATE nomTable
SET col1 = expr1, col2 = expr2…
[WHERE predicat] ;

permet de modifier les tuples de *nomTable* vérifiant le *prédicat* (expression logique). En l'omission de la clause *WHERE*, tous les tuplets de la table seront mis à jour.

DELETE

DELETE FROM nomTable
[WHERE predicat];

permet de supprime les tuples de la table *nomTable* vérifiant le *prédicat*. Dans le cas où la clause *WHERE* est omise, tous les tuples seront supprimés.

SELECT … FROM

SELECT [DISTINCT | ALL] col1, col2, …
FROM nomTable, [nomTable2, …]
[WHERE conditions]

permet d'extraire d'une base les données répondant à des questions particulières. Le résultat de cette requête est une table fictive qui sera affiché à l'écran. Les trois clauses de cette requête sont :

- *SELECT* permet de spécifier les colonnes à afficher dans le résultat de la requête. Le caractère *étoile (*)* indique toutes les colonnes de la table indiquée dans la clause FROM de la requête.
- FROM spécifie les tables sur lesquelles porte la requête. Cette clause réalise le produit cartésien des tables spécifiées.

- WHERE (facultative) définie les conditions que les tuples du résultat de la requête doivent satisfaire. Cette clause réalise l'opération de *sélection*.

Ex. Considérons le schéma relationnel suivant :
ETUDIANTS (Matricule, nomEtu, preEtu, dateNaiEtu, villeEtu) ;
MODULES (codeModule, intitule, coef) ;
SUIVRE (#idEtu, #idModule, note) ;

Affiche le nom et le prénom de tous les étudiants :
SELECT nomEtu, preEtu FROM ETUDIANTS ;

Affiche toutes les informations des étudiants qui habitent à Béjaia :
SELECT * FROM ETUDIANTS WHERE villeEtu = 'Béjaia' ;

Remarques
Les *cotes* (') sont obligatoires pour entourer les chaînes de caractères et les dates.

La clause WHERE permet aussi de réaliser des opérations de jointure, le plus souvent par égalité de deux colonnes ayant un sens identique dans deux tables.

DISTINCT

La clause DISTINCT permet d'éliminer les doublons dans le résultat d'une requête.

Ex. Afficher toutes les villes où habite au moins un étudiant.
SELECT DISTINCT villeEtu
FROM ETUDIANTS;

ORDER BY

La clause *ORDER BY* permet de trier les lignes du résultat d'une requête par ordre croissant (ASC) ou décroissant (DESC). L'ordre par défaut est ASC. Lorsque plusieurs colonnes sont mentionnées, le tri se fait d'abord selon les premières, puis selon les suivantes en cas d'égalité des premières.

Ex. Afficher les étudiants par ordre alphabétique du nom puis du prénom.
SELECT * FROM ETUDIANTS ORDER BY nomEtu, preEtu ;

AS

Dans la clause SELECT, l'opérateur *AS* permet de renommer une colonne lors de l'affichage du résultat d'une requête.

Ex. Affiche le matricule et la date de naissance de tous les étudiants.
SELECT Matricule, DateNaiEtu AS 'Date de naissance'
FROM ETUDIANTS ;

Dans la clause FROM, l'opérateur *AS* permet de renommer une table afin de pouvoir ensuite y faire référence. L'une des deux syntaxes est utilisée :

*SELECT * FROM nomTable AS nouveauNomTable*
*SELECT * FROM nomTable nouveauNomTable;*

L'objectif de renommer une table est de simplifier les noms trop longs :
*SELECT * FROM nomDeLaTable AS table ;*

IN

L'opérateur *IN* indique si la valeur d'une colonne est égale à l'une des valeurs se trouvant dans une liste de choix.

Ex. Liste des étudiants habitant Béjaia, Alger ou Oran.
SELECT *
FROM ETUDIANTS
WHERE villeEtu IN ('Béjaia', 'Alger', 'Oran');

NULL

NULL indique une valeur non définie (non renseignée). Il est à noter que 0 n'est pas une valeur NULL. L'opérateur de comparaison avec la valeur NULL est « IS » ou « IS NOT ».

Ex. Afficher le matricule et le nom des étudiants dont la date de naissance est inconnue.

```
SELECT Matricule, NomEtu
FROM ETUDIANTS
WHERE DateNaiEtu IS NULL ;
```

LIKE

L'opérateur *LIKE* permet de comparer la valeur d'une colonne avec une chaîne non complète. La caractère % désigne plusieurs caractères alors que le caractère _ désigne un seul caractère quelconque.

Ex. Afficher les étudiants dont le nom commence par DE.

```
SELECT *
FROM ETUDIANTS
WHERE nomEtu LIKE 'DE%' ;
```

BETWEEN

Permet de sélectionner les valeurs dans un intervalle donné. Ces valeurs peuvent être des nombres, texte ou dates. Les bornes de l'intervalle sont incluses.

Ex. Liste des étudiants nés durant les années 80.

```
SELECT *
FROM ETUDIANTS
WHERE DateNaiEtu BETWEEN '1/1/1980' AND '31/12/1989' ;
```

EXISTS

Permet de tester l'existence de valeurs dans une sous-requête. Dans le cas où la sous-requête renvoie au moins une ligne, même remplie de NULL, le prédicat retourne *vrai*. Dans le cas contraire, le prédicat retourne *faux*.

Le prédicat NOT EXISTS, quant à lui, est utilisé pour tester l'absence de valeurs dans la sous-requête.

```
SELECT col1
FROM nomTable1
WHERE EXISTS (SELECT *
              FROM nomTable2
              WHERE condition) ;
```

```
SELECT col1
FROM nomTable1
WHERE NOT EXISTS (SELECT *
                  FROM nomTable2
                  WHERE condition) ;
```

JOINTURE

La jointure entre deux tables « *table1* » et « *table1* » représente un produit cartésien suivi d'une sélection selon un critère dit *de jointure*, c.à.d. n'importe quelle opération de comparaison entre un attribut de *table1* avec un attribut de *table2*.

```
SELECT attr1, attr2, …
FROM table1, table2
WHERE table1.attr1 = table2.attr2;
```

Ex. Matricule et nom des étudiants ayant obtenu une note > 10 dans le module *BDL2*.

```
SELECT  E.Matricule,  nomEtud
FROM  ETUDIANTS  E,  SUIVRE  S
WHERE  E.Matricule = S.idEtu AND
       note > 10 AND codeModule = 'BDL2' ;
```

Les Types de Données SQL

Le langage ***SQL*** offre divers types de données, dits *types de bases*, à utiliser lors de la déclaration des colonnes d'une table. Ces types peuvent être regroupés dans

trois catégories : les types numériques, les types date et heure, et les types chaînes de caractères. Dans ce qui suit, nous présentons les différents types disponibles et leurs tailles respectives.

- M indique la taille maximale d'affichage. La taille maximale autorisée par défaut est 255.
- D s'applique aux types à virgule flottante, et précise le nombre de chiffre après la virgule.
- Crochets (« [» et «] ») indique que cet argument est optionnel.

TINYINT [(M)] [UNSIGNED] [ZEROFILL] Un très petit entier. Signé, il couvre l'intervalle −128 à 127 ; non signé, il couvre 0 à 255.

SMALLINT [(M)] [UNSIGNED] [ZEROFILL] Un entire court. Signé, il couvre l'intervalle −32768 à 32767; non signé, il couvre 0 à 65535.

MEDIUMINT [(M)] [UNSIGNED] [ZEROFILL] Un entier de taille intermédiaire. Signé, il couvre l'intervalle −8388608 à 8388607; non signé, il couvre 0 à 16777215.
INTEGER ou **INT [(M)] [UNSIGNED] [ZEROFILL]** Un entier de taille normale. Signé, il couvre l'intervalle −2147483648 à 2147483647; non signé, il couvre 0 à 4294967295.

BIGINT [(M)] [UNSIGNED] [ZEROFILL] Un entier de grande taille. Signé, il couvre l'intervalle −9223372036854775808 à 9223372036854775807; non signé, il couvre 0 à 18446744073709551615.

FLOAT (precision) [ZEROFILL] Un nombre à virgule flottante. Il est obligatoirement signé. « precision » peut prendre les valeurs de 4 ou 8.
FLOAT(8) est un nombre à précision double. Ces types sont identiques aux types FLOAT et DOUBLE décrit ci−dessous, mais leur précision peut être paramétrée.

FLOAT [(M, D)] [ZEROFILL] Un nombre à virgule flottante, en précision simple. Il est toujours signé. Les valeurs sont comprises entre −3.402823466E+38 et −1.175494351E−38.

DOUBLE [(M, D)] [ZEROFILL] Un nombre à virgule flottante, en précision double. Il est toujours signé. Les valeurs sont comprises entre −1.7976931348623157E+308 et 2.2250738585072014E−308.

REAL [(M, D)] [ZEROFILL] Des synonymes pour DOUBLE.

DECIMAL (M, D) [ZEROFILL] Un nombre à virgule flottante. Il est toujours signé. Il se comporte comme une colonne CHAR. Le nombre est stocké comme une chaîne de chiffre. Chaque chiffre, le signe moins, la virgule occupe un caractère.

NUMERIC (M, D) [ZEROFILL] Un synonyme pour DECIMAL.

DATE Une date. L'intervalle valide de date va de '1000−01−01' à '9999−12−31'. *MySQL* affiche les DATE avec le format 'YYYY−MM−DD', mais il est possible d'affecter des DATE en utilisant indifféremment des chaînes ou des nombres.

DATETIME Une combinaison de date et d'heure. L'intervalle valide va de '1000−01−01 00:00:00' to '9999−12−31 23:59:59'. *MySQL* affiche DATETIME avec le format 'YYYY−MM−DD HH:MM:SS', mais il est possible d'affecter des DATETIME en utilisant indifféremment des chaînes ou des nombres.

TIMESTAMP [(M)] Un timestamp : la date et l'heure, exprimée en secondes, depuis le 1er janviers 1970. Il permet de couvrir un intervalle allant de '1970−01−01 00:00:00' à l'année 2037. *MySQL* affiche les TIMESTAMP avec les formats YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD ou YYMMDD, suivant que M vaut 14 (ou absent), 12, 8 ou 6, mais il est possible d'affecter des TIMESTAMP en utilisant indifféremment des chaînes ou des nombres.

TIME Une mesure de l'heure. L'intervalle valide est '−838:59:59' à '838:59:59'. *MySQL* affiche TIME au format 'HH:MM:SS', mais il est possible d'affecter des TIME en utilisant indifféremment des chaînes ou des nombres.

YEAR Un an. L'intervalle valide est 1901 à 2155, et 0000. *MySQL* affiche YEAR au format YYYY, mais il est possible d'affecter des YEAR en utilisant indifféremment des chaînes ou des nombres.

CHAR (M) [BINARY] Une chaîne de caractère de taille fixe, et toujours complétée à droite par des espaces. « M » va de 1 à 255 caractères. Les espaces supplémentaires sont supprimés lorsque la valeur est retournée dans une requête. Les tris et comparaisons effectués sur des valeurs de type CHAR sont insensibles à la casse, à moins que le mot clé BINARY soit précisé.

VARCHAR(M) [BINARY] Une chaîne de caractère de longueur variable. Les espaces en fin de chaîne sont supprimés lorsque la chaîne est stockée. « M » va de 1 à 255 caractères. Les tris et comparaisons effectués sur des valeurs de type VARCHAR sont insensibles à la casse, à moins que le mot clé BINARY soit précisé.

TINYTEXT Un objet TEXT avec une longueur maximale de 255.

TEXT Un objet TEXT avec une longueur maximale de 65535.

MEDIUMTEXT Un objet TEXT avec une longueur maximale de 16777215.

LONGTEXT Un objet TEXT avec une longueur maximale de 4294967295.

ENUM ('value1', 'value2', ...) Une énumération. Un objet chaîne peut prendre une des valeurs contenue dans une liste de valeur 'value1', 'value2', ..., ou NULL. Une ENUM peut avoir un maximum de 65535 valeurs distinctes.

SET ('value1', 'value2',...) Un ensemble. Un objet chaîne peut prendre une ou plusieurs valeurs, chacun de ces valeur devant être contenue dans une liste de valeurs 'value1', 'value2', ... Un SET peut prendre jusqu'à 64 éléments.