Aide-Mémoire SQL v1.0.4

[ ] = optionnel
{ .. | .. | .. } = l'un des choix est obligatoire
ALL = ne sert à rien (est par défaut)
table = valeur à remplacer

COMMENTAIRES

-- commentaires
Tout SGBD
commentaires
/!\ MySQL uniquement
/* commentaires */
/!\ MySQL/Oracle

CREATION

CREATE DATABASE *base*
Créé une nouvelle base de donnée de nom *base*
CREATE TABLE *table* (*ch1* int(2), *ch2* char(30))
CREATE TABLE *table* (**SELECT** ...)
Créé une nouvelle table selon une liste de colonne ou selon le résultat du SELECT (peut copier entièrement une table)

AJOUT ET MODIFICATION

INSERT INTO *table* [*(champ1, champ2)*]
VALUES (*val_1*, *val_2*) [, *(val_1b, val_2b)*, ...]
Permet d'insérer une ou plusieurs lignes dans une table
/!\ Access : une seul expression après VALUES
INSERT INTO *table* [*(champ, ...)*] **SELECT** ...
Permet d'insérer le résultat d'une sélection
RENAME TABLE *ancien_nom* **TO** *nouveau_nom*
Renomme une table
ALTER TABLE *table*
{**ADD** | **DROP** | **MODIFY**} [**COLUMN** | **VIEW** | **CONSTRAINT**] (*nom&Type* | *contrainte* [, ..])
Ajoute/Supprime/Modifie une colonne ou une contrainte
UPDATE *table* **SET** *champ=val*2* [, *champ2=val2* ...]
WHERE *condition*
Modifie les champs des lignes qui vérifient la condition

SUPPRESSION

DROP TABLE [**IF EXISTS**] *table* [, *table2*, ...]
[**CASCADE CONSTRAINTS**]
Supprime la ou les tables sélectionner
/!\ Access : pas de IF EXISTS
DELETE FROM *table* **WHERE** *condition*
Supprime les lignes qui vérifient la condition
TRUNCATE TABLE *table*
Vide la table et libère la place (delete ne libère pas)

REQUETE DE SELECTION

SELECT [ALL | **DISTINCT**] *champ, ...*
Précise les champs à sélectionner
FROM *table, ...*
Précise les tables à utiliser
WHERE *condition*
Précise des conditions pour limiter la recherche
GROUP BY *champ* [, ...]
/!\ Avec fonction dans SELECT
/!\ Tous les champs utilisés par SELECT (hormis fonctions) doivent apparaître !

La fonction agira différemment selon le champ. Ex :
SELECT nom, COUNT(*)
FROM table
GROUP BY nom
ne renverra que le nombre de lignes par nom, et pas le nombre de ligne total.
HAVING *condition* [{**AND** | **OR**} *condition2* ...]
/!\ Après GROUP BY

Ajoute des conditions. Peut utiliser des fonctions sur la table au complet (pas la sélection). Ex :
HAVING *COUNT(*) >2*
ORDER BY *champ* [ASC | **DESC**] [, *champ2* ...]
Ordonne de manière ascendante ou descendante le(s) champ(s)

RENOMMAGE

/!\ *Access : AS obligatoire et [ ] pour nom avec espaces*

SELECT *champ* [AS] *Nom* [, *(cond1 > cond2)*]
Renomme la colonne *champ* en *Nom*. On peut également afficher un booléen résultant d'un test.
FROM *table* [AS] *nomTable* [*(nomChamp1, ...)*]
Renomme la table avec ses champs (si précisé)

PRODUIT ET JOINTURES

FROM *table1 t1*, *table2 t2*
Produit : Chaque tuple de *t1* est associé aux n tuples de *t2*. Nb de ligne = l1 * l2. Nb de colonne : c1 + c2.
FROM *t1* **INNER JOIN** *t2* **ON** *t1.num = t2.num*
Jointure : Retourne le produit de *t1* et *t2* et supprime les lignes qui ne vérifient pas la condition. (ici : jointure naturelle)
FROM *table1 t1*, *table2 t2*
WHERE *t1.num = t2.num*
Jointure naturelle : sur le champs *num*
FROM *table1* **NATURAL JOIN** *table2*
Jointure naturelle : sur un champs de même nom/type (SQL3).
FROM *t1* {**LEFT** | **RIGHT**} **OUTER JOIN** *t2* **ON** *t1.num = t2.num*
Jointure externe gauche/droite : C'est une jointure naturelle auquel on ajoute les infos de la table de gauche ou droite qui sont non affichées et rempli les vides par des NULL.

UNION / INTERSECTION / SOUSTRACTION

SELECT ...
{**UNION** | **INTERSECT** | **EXCEPT**} [**ALL**]
SELECT ...
Fait une union ou autre entre les résultats des 2 select.
/!\ Parfois MINUS à la place d'EXCEPT (Oracle)
/!\ MySQL5 : seulement UNION, essayer de transformer le reste avec des NOT IN etc...

CONDITIONS

=, <> (ou **!=**)**, <, >, <=, >=**
Opérateurs de comparaison
attr **IS** [**NOT**] **NULL**
Teste si la valeur est NULL ou non
attr [**NOT**] **EXIST** (**SELECT** ...)
Teste si la sélection renvoi quelque chose ou non
attr [**NOT**] **BETWEEN** *val_1* **AND** *val_2*
Teste si la valeur est entre val_1 et val_2 (inclus)
attr [**NOT**] **LIKE** '*%chaîne_%*'
Teste si la variable ressemble ou non à la chaîne
'%' remplace une chaîne
'_' remplace un caractère
/!\ Access, Oracle : '%' -> '*' et '_' -> '?'
attr [**NOT**] **IN** (*var_1*, *var_2*, ...)
attr [**NOT**] **IN** (**SELECT** ...)
Teste si la variable est dans la liste/sous-requête
SELECT *

FROM *table*
WHERE *attr* [**NOT**] **EXISTS** (**SELECT** * ...)
Sélectionne le tuple de *table* quand la sous-requête renvoie quelque chose (utiliser des attributs ou table de la première requête)
attr operateur **ALL** (*var_1*, *var_2*, ...)
attr operateur **ALL** (**SELECT** ...)
Teste si la valeur est =, !=, >, ... à toutes les valeurs de la liste/sous-requête
attr operateur **ANY** (*var_1*, *var_2*, ...)
attr operateur **ANY** (**SELECT** ...)
Teste si la valeur est =, <>, >, ... *à* au moins un tuple de la sous sélection

LES CONTRAINTES

Dans un CREATE TABLE, ALTER TABLE...

attr type [**NOT**] **NULL**
Permet de préciser qu'une valeur ne peut pas être NULL
attr type **UNIQUE**
Permet de préciser que chaque valeur doit être unique
attr type **AUTOINCREMENT**
Permet de préciser que l'entier va s'autoincrémenter
UNIQUE (*nom_colonne*, *[nom_colonne2, ...]*)
Permet de déclarer un composé unique
attr type **PRIMARY KEY**
Permet de préciser que la valeur est une clé primaire
PRIMARY KEY (*nom_colonne*, *[nom_colonne2, ...]*)
Permet de préciser que la(les) valeur est une clé primaire
attr type **REFERENCES** *table* (*nom_col_distante*)
Permet de préciser que la valeur est une clé externe
/!\ MySQL5 : uniquement avec InnoDB
FOREIGN KEY (*nom_colonne*)
REFERENCES *table* (*nom_col_distante*)
Permet de préciser que la valeur est une clé externe
/!\ MySQL5 : uniquement avec InnoDB
CHECK (*condition*)
Permet de fixer des conditions. Ex : nom_colonne > 10
/!\ Non supporté par MySQL5

AJOUT D'UNE CONTRAINTE

ALTER TABLE *table* **ADD CONSTRAINT**
pk_nom **PRIMARY KEY** (*nom_colonne*)

ALTER TABLE *table* **ADD CONSTRAINT**
fk_nom **FOREIGN KEY** (*nom_colonne*)
REFERENCES *table* (*nom_col_distante*)

ALTER TABLE *table* **ADD CONSTRAINT**
chk_nom **CHECK** (*condition*)

CREATE TABLE *table* (
champ1 type,
CONSTRAINT *ct_nom* [**PRIMARY KEY** | ...] ...)

MODIFICATION D'UNE CONTRAINTE

ALTER TABLE *table*
DROP {**PRIMARY** | **FOREIGN**} **KEY** [**CASCADE**]
ALTER TABLE *table*
DROP {**UNIQUE** | **NULL**} (*nom_colonne*) [**CASCADE**]
ALTER TABLE *table*
DROP CONSTRAINT *nom_contrainte* [**CASCADE**]
Suppression d'une contrainte

ALTER TABLE *table*
{**DISABLE** | **ENABLE**} **CONSTRAINT** *nom_contrainte*
Activation ou désactivation d'une contrainte
/!\ *Oracle uniquement*

VUES

CREATE VIEW *nom_vue* **AS SELECT** ...
Permet de créer une vue affichant la sélection faite
DROP VIEW *nom_vue*
Permet de supprimer une vue (sans toucher aux données)

FONCTIONS

COUNT([**DISTINCT**] *champ*)
Renvoie le nombre de ligne trouvé pour ce champ. Sans distinct, renvoie le nombre de toutes les occurrences, même des doublons.
/!\ Access : pas de DISTINCT
SUM(*champ*)
Renvoie la somme des valeurs du champ
AVG(*champ*)
Renvoie la moyenne des valeurs du champ
MIN(*champ*)
Renvoie le minimum du champ
MAX(*champ*)
Renvoie le maximum du champ

DROITS UTILISATEURS (LCD)

GRANT {**SELECT** | **INSERT** | **DELETE** | **UPDATE** | **ALTER** | **ALL PRIVILEGES**} **ON** *table* **TO** '*user*' [**WITH GRANT OPTION**]
Donne un/des droits à l'utilisateur *user* sur la table *table.*
GRANT OPTION permet de permettre à l'utilisateur de transférer ses propres droits.
/!\ Oracle : TO PUBLIC est possible
REVOKE *droit* **ON** *table* **FROM** '*user*'
Donne tous les droits à l'utilisateur *user* sur la table *table*

TRANSACTIONS

Avec PostgreSQL, Oracle...

[BEGIN TRANSACTION];
...
{**ROLLBACK** | **COMMIT**} [TRANSACTION];
Commence une transaction et la termine en l'annulant ou l'acceptant

TRIGGERS

Avec PostgreSQL, Oracle...

CREATE [**OR REPLACE**] **TRIGGER** *trigger*
{**BEFORE** | **AFTER**} {**INSERT OR UPDATE** [**OF** *champ*] **OR DELETE**} **ON** *table*
[**FOR EACH ROW** [**WHEN** (*condition*)]]
BEGIN
  ...
  UPDATE table
  SET **new.***champ* = **old.***champ + 1***;**
  ...
END
Créé un trigger sur la table. Ici, exemple de UPDATE dans le trigger.
/!\ *Généralement, trigger before*
/!\ *Pas de modif sur les PK, FK, UNIQUE KEY*
/!\ *Toujours utiliser old|new pour la table en cours de modification, pas pour les autres.*

SQL ORACLE

...
/
SHOW ERRORS;
Affiche les erreurs produites avec les lignes ci-dessus (PL/SQL uniquement)
SELECT * **FROM** cat;
Permet d'afficher toute les tables de la base (catalogue)
SELECT * **FROM** user_constraints;
Permet d'afficher toute les contraintes de la base
SELECT 125 * 8987 **FROM** dual;
Permet de faire divers calculs qui ne nécessite pas de

table
SELECT *, **CASE WHEN** *ch op value* **THEN** '*X1*' [**WHEN** *ch op value2* **THEN** '*X2*'] **ELSE** '*X3*' **END** [*Renom*]
SELECT CASE *champ* **WHEN** *value* **THEN** '*X1*' [**WHEN** *value2* **THEN** '*X2*'] **ELSE** '*X3*' **END** [*Renom*]
Permet de rajouter une colonne avec les valeurs prédéfinies selon certaines conditions

ORACLE PL/SQL

/!\ Toujours finir avec un '/'

nomVar [**CONSTANT**] *type* [**NOT NULL**] [**:=** *defVal*]**;**
*nomVar table***.***champ***%TYPE;**
CURSOR *c_nom* **IS SELECT** *...*
Entre IS-BEGIN ou DECLARE-BEGIN. Déclare une variable ou un cursor.

SELECT *champ1* [, *champ2*, ...] **INTO** *var1* [, *var2*, ...]
Affecte la variable avec le résultat du SELECT

var **:=** *val***;**
Affecte la variable avec la valeur

CREATE [**OR REPLACE**]
PROCEDURE *nom* [(param {**IN | OUT | INOUT**} type [, ...])]
IS
  [*...déclaration de vars...*]
BEGIN
  ...code...
[**EXCEPTION**
  ...exceptions...]
END [*nom*]**;**
Créé une procédure stockée.

CREATE [**OR REPLACE**]
FONCTION *nom* [(param {**IN | OUT | INOUT**} type [, ...])]
    RETURN *type*
IS
  [*...déclaration de vars...*]
BEGIN
  ...code...
  RETURN *val***;**
[**EXCEPTION**
  ...exceptions...]
END [*nom*]**;**
Créé une fonction stockée.

[**DECLARE**
  ...déclaration vars...]
BEGIN
  ...code...
END;
Créé une procédure anonyme et l'exécute immédiatement

IF *cond* **THEN**
  ...
ELSIF *cond* **THEN**
  ...
ELSE
  ...
END IF;
Structure conditionnelle

WHILE *cond* **LOOP**
  ...
END LOOP;
Boucle WHILE

LOOP
  ...
  EXIT [**WHEN** *cond*];
END LOOP;
Boucle REPEAT

FOR *compteur* **IN** [**REVERSE**] *bInf***..***bSup* **LOOP**
  ...
END LOOP;
Boucle FOR

FOR *c_tuple* **IN** *cursor* **LOOP**
  ... c_tuple.champ ...
END LOOP;
Permet de manipuler un cursor

OPEN *cursor***;**
LOOP
  FETCH *cursor* **INTO** *var1* [, *var2*, ...];
  EXIT WHEN *cursor***%NOTFOUND;**
  ...
END LOOP;
Permet de manipuler un cursor

DBMS_OUTPUT.PUT_LINE(*varOuVal***)**
Affiche une valeur ou variable

RELATIONNEL-OBJET ORACLE

/!\ Toujours finir avec un '/'

CREATE TYPE *nomType* **AS OBJECT(**
*champ type***,**
[... ,]
champRef **REF** *nomTableObjets_ou_nomType***,**
MEMBER FUNCTION *nomFunc* [(*...*)] **RETURN** *type***,**
MEMBER PROCEDURE *nomProc*
) [**NOT FINAL**] **;**
Exemple de création d'un nouveau type.
/!\ Le NOT FINAL est important pour que le type puisse être hérité.
/!\ Le corps des fonctions est créé ultérieurement.

CREATE TYPE *nomType* **UNDER** *nomTypeHerite***(**
*champ type***,**
[... ,]
[**OVERRIDING**] **MEMBER** ...
) [**NOT FINAL**] **;**
Exemple de création d'un nouveau type qui hérite d'un autre. Nous pouvons écraser (overriding) les fonctions à condition que la signature soit la même.

CREATE TYPE *nomType*
AS TABLE OF *nomType2***;**
Création d'un nouveau type qui est une liste d'élément d'un autre type.

CREATE TABLE *table* **OF** *type* **(**
PRIMARY KEY(*champ*)**,**
CONSTRAINT *...* **,**
[**SCOPE FOR** (*typeRef*) **IS** *tableDeLimitation*]
) [**NESTED TABLE** *tableImbriquee* **STORE AS** *nomPourLaSousTableCree*] **;**
Création d'une table basée sur un type. On peut rajouter les contraintes habituelles, ainsi qu'un SCOPE FOR permettant de restreindre les références à une table donnée (la référence ne peut donc référencer les données d'une autre tables).
/!\ Le NESTED TABLE est obligatoire pour les attributs de type « TABLE OF » de la table.

CREATE [**OR REPLACE**] **TYPE BODY** *table* **AS**
  MEMBER FUNCTION *nomFunc* [(*...*)] **RETURN** *type* **IS**
  BEGIN
    ...
    RETURN self.*attr***;**
  END *nomFunc* **;**
END ;
Exemple d'implémentation d'une fonction membre d'une

table.
/!\ *vous pouvez utiliser self pour indiquer l'objet en cours*
/!\ *« self.attr := val; » change la valeur en mémoire uniquement, pas de persistance (utiliser UPDATE)*

```
SELECT i.val Renom
FROM table t, [ ... ,] TABLE(t.tableImbriquee) i
WHERE t.num=val ;
```

Simple sélection d'une valeur d'une table imbriquée.
/!\ *Jointure non nécessaire*
/!\ *Renommage obligatoire*

```
DECLARE
  ref_table REF table;
BEGIN
  SELECT REF(t) INTO ref_table
  FROM table t
  WHERE t.champ=val;
  INSERT INTO tableAvecRef(champ, champRef,
                           champTypeCompose)
         VALUES ( ref_table.func(), ref_table,
                  typeImbrique(val2, val3) )
  );
END;
```

Exemple d'insertion de données dans une table nécessitant une référence (en PL/SQL). On voit également qu'il est possible d'utiliser les fonctions d'une référence ou de créer des objets à l'intérieur du INSERT.

ORACLE SQL*PLUS

```
COLUMN champ FORMAT format
SELECT ...
```

Modifie le format d'affichage du champ *champ* de la requête selon le format *format* (peut-être A32 pour avoir une colonne de 32 caractères, $99,990.00 etc...)

```
show variable
```

Affiche la valeur de la variable (peut être *pagesize*, *linesize* ...)

```
set variable value
```

Modifie la valeur de la variable

SPECIFICITES PARTICULIERES

• Oracle et MySQL possède un mot clé pour donner le nom des champs et infos d'une table :

```
DESCRIBE table
```

• SQLite permet généralement de se passer d'id AUTOINCREMENT en utilisant les rowid (de 1 à X).

```
SELECT rowid FROM table
```

• PostgreSQL/Oracle peut avoir un mode « strict » ou les champs sont entre guillemets et les valeur AlphaNumérique entre simple quote. Ex :

```
SELECT "champ" FROM "table" WHERE "champ" = 'valeurAN'
```

• Un UPDATE sur plusieurs tables est non standard et ne peut se faire qu'avec SQLServer et Oracle (2 syntaxes différentes)

• Limiter le nombre de tuples sélectionnés : LIMIT n (MySQL/PostgreSQL, à la fin), TOP n (Access, dans select)
Standard (SQL 2003) :

```
SELECT * FROM (
  SELECT
    ROW_NUMBER() OVER (ORDER BY key ASC) AS
      rownumber, columns
  FROM table
)
WHERE rownumber <= N
```

a faire :
séquence
RECORD ORACLE
oracle RO : is [not] of type, Treat() ?
Create index