SQL DB2 chez AJC

Les ordres SQL

Les mises à jour des tables

Les programmes Cobol-DB2

# Les ordres SQL

...

# Mises à jour des tables

Il est conseillé de d’écrire les ordres SQL dans un membre TSO, en ne gardant dans chaque membre TSO un seul ordre SQL et une seule table.

Le travail en sera facilité.

Deux tirets consécutifs indiquent que le reste de la ligne est en commentaire.

Pour la création d’une table, il est conseillé de prévoir une ligne DROP suivie d’un COMMIT.

La première fois, ces deux lignes sont en commentaire.

## CREATE TABLE

Quand une table fait référence à une autre table, on dit que c’est une table fille qui fait référence à une table mère.

Il faut déjà créer la table mère puis la table fille, puis sa table fille et ainsi de suite.

Les tables indépendantes peuvent être créées n’importe quand.

ACT07.SOURCE.SQL(**CENTR**)

--DROP TABLE **ENTREPOT**;

--COMMIT;

-- CREATION DE LA TABLE ENTREPOT

CREATE **TABLE** **ENTREPOT**(

ID\_ENTREPOT CHAR(02) NOT NULL,

DESIGNATION CHAR(20) NOT NULL,

PRIMARY KEY (ID\_ENTREPOT)

) IN APIETUDE.API7;

CREATE UNIQUE **INDEX** **PK\_ENTREPOT**

ON ENTREPOT (ID\_ENTREPOT) ;

--

COMMIT;

Entre parenthèses on indique les colonnes. Pour chacune on écrit son format (CHAR, VARCHAR, DEC, INT, SMALLINT).

Si on indique pas NOT NULL, la colonne pourra être NULL.

Si la table possède une clé primaire, ce qui est le cas le plus fréquent, on indique PRIMARY KEY(nom\_de\_la\_colonne).

La création de l’index primaire et des autres index peuvent être créés en même temps ou plus tard.

Un entrepôt regroupe les marchandises d’une ou de deux familles d’articles, et une famille n’est hébergée que par un seul entrepôt.

La table famille est donc fille de la table entrepôt.

ACT07.SOURCE.SQL(**CFAM**)

DROP TABLE FAMILLE;

COMMIT;

-- CREATION DE LA TABLE FAMILLE

CREATE TABLE FAMILLE(

ID\_FAMILLE CHAR(02) NOT NULL,

DESIGNATION CHAR(20) NOT NULL ,

ID\_ENTREPOT CHAR(02) NOT NULL,

PRIMARY KEY (ID\_FAMILLE),

FOREIGN KEY (ID\_ENTREPOT) REFERENCES ENTREPOT (ID\_ENTREPOT)

ON DELETE RESTRICT

) IN APIETUDE.API7;

CREATE UNIQUE INDEX PK\_FAMILLE

ON FAMILLE (ID\_FAMILLE) ;

--

COMMIT;

Le lien entre deux tables est indiqué lors de la création de la table fille par « FOREIGN KEY nom\_de\_la\_colonne dans la table fille) REFERENCES NOM\_DE\_LA\_TABLE\_MERE(Nom\_de\_la\_colonne qui est clé primaire)

La table marque n’a pas de table mère.

ACT07.SOURCE.SQL(**CMAR**)

--DROP TABLE MARQUE;

--COMMIT;

-- CREATION DE LA TABLE MARQUE

CREATE TABLE MARQUE(

ID\_MARQUE CHAR(02) NOT NULL,

DESIGNATION CHAR(20) NOT NULL,

PRIMARY KEY (ID\_MARQUE)

) IN APIETUDE.API7;

CREATE UNIQUE INDEX PK\_MARQUE

ON MARQUE (ID\_MARQUE) ;

--

COMMIT;

La table article a deux tables mères : marque et famille

ACT07.SOURCE.SQL(**CART**)

--DROP TABLE ARTICLE;

--COMMIT ;

-- CREATION DE LA TABLE ARTICLE

CREATE TABLE ARTICLE(

ID\_ARTICLE INTEGER NOT NULL,

DESIGNATION CHAR(20) NOT NULL,

PRIX DECIMAL(5,2) NOT NULL,

ID\_MARQUE CHAR(2),

ID\_FAMILLE CHAR(2) NOT NULL,

PRIMARY KEY (ID\_ARTICLE),

FOREIGN KEY (ID\_FAMILLE) REFERENCES FAMILLE (ID\_FAMILLE)

ON DELETE RESTRICT,

FOREIGN KEY (ID\_MARQUE) REFERENCES MARQUE (ID\_MARQUE)

ON DELETE SET NULL

) IN APIETUDE.API7;

CREATE UNIQUE INDEX PK\_ARTICLE

ON ARTICLE (ID\_ARTICLE) ;

--

COMMIT;

La table déclinaison a une table mère : article

ACT07.SOURCE.SQL(**CDECLI**)

--DROP TABLE DECLINAISON;

--COMMIT;

-- CREATION DE LA TABLE DECLINAISON

CREATE TABLE DECLINAISON(

ID\_ARTICL INTEGER NOT NULL,

ID\_DECLINAISON CHAR(02) NOT NULL,

DESIGNATION CHAR(20) NOT NULL,

STOCK INTEGER NOT NULL WITH DEFAULT 0,

STOCK\_ALERTE INTEGER NOT NULL WITH DEFAULT 0,

TAILLE CHAR(05) NOT NULL,

COLORIS CHAR(10) NOT NULL,

QUOTITE INTEGER NOT NULL WITH DEFAULT 1,

STOCK\_COMMANDEE INTEGER NOT NULL WITH DEFAULT 0,

PRIMARY KEY (ID\_ARTICL,ID\_DECLINAISON),

FOREIGN KEY (ID\_ARTICL) REFERENCES ARTICLE (ID\_ARTICLE)

ON DELETE CASCADE

) IN APIETUDE.API7;

CREATE UNIQUE INDEX PK\_DECLINAISON

ON DECLINAISON (ID\_ARTICL,ID\_DECLINAISON) ;

--

COMMIT;

## INSERT

Il faut insérer d’abord les lignes des tables mères.

ACT07.SOURCE.SQL(**INSENTR**)

INSERT INTO ENTREPOT VALUES ('01' , 'ENTREPOT1' ) ;

INSERT INTO ENTREPOT VALUES ('02' , 'ENTREPOT2' ) ;

INSERT INTO ENTREPOT VALUES ('03' , 'ENTREPOT3' ) ;

INSERT INTO ENTREPOT VALUES ('E1' , 'ENTREPOT1' ) ;

INSERT INTO ENTREPOT VALUES ('E2' , 'ENTREPOT2' ) ;

INSERT INTO ENTREPOT VALUES ('E3' , 'ENTREPOT3' ) ;

ACT07.SOURCE.SQL(**INSMAR**)

INSERT INTO MARQUE VALUES ('01' , 'MARQUE1' ) ;

INSERT INTO MARQUE VALUES ('02' , 'MARQUE2' ) ;

INSERT INTO MARQUE VALUES ('03' , 'MARQUE3' ) ;

INSERT INTO MARQUE VALUES ('M1' , 'MARQUE1' ) ;

INSERT INTO MARQUE VALUES ('M2' , 'MARQUE2' ) ;

INSERT INTO MARQUE VALUES ('M3' , 'MARQUE3' ) ;

ACT07.SOURCE.SQL(**INSFAM**)

INSERT INTO FAMILLE VALUES ('01' , 'FAMILLE1', '01');

INSERT INTO FAMILLE VALUES ('02' , 'FAMILLE2', '01') ;

INSERT INTO FAMILLE VALUES ('03' , 'FAMILLE3', '02') ;

INSERT INTO FAMILLE VALUES ('01' , 'FAMILLE1', 'E1');

INSERT INTO FAMILLE VALUES ('02' , 'FAMILLE2', 'E1') ;

INSERT INTO FAMILLE VALUES ('03' , 'FAMILLE3', 'E2') ;

INSERT INTO FAMILLE VALUES ('F1' , 'FAMILLE1', 'E1');

INSERT INTO FAMILLE VALUES ('F2' , 'FAMILLE2', 'E1') ;

INSERT INTO FAMILLE VALUES ('F3' , 'FAMILLE3', 'E2') ;

ACT07.SOURCE.SQL(**INSART**)

INSERT INTO ARTICLE VALUES (1 , 'ARTICLE1', 10.50,'M1','F1')

INSERT INTO ARTICLE VALUES (2 , 'ARTICLE2', 100.50,'M2','F1')

INSERT INTO ARTICLE VALUES (3 , 'ARTICLE3', 64.00,'M3','F3')

ACT07.SOURCE.SQL(**INSDECLI**)

INSERT INTO DECLINAISON VALUES (1 ,'01','1-01', 50 , 0,'XL'

,'VIOLET', 3, 0) ;

INSERT INTO DECLINAISON VALUES (1 ,'02','1-02', 50 , 0,'L'

,'VIOLET', 3, 0) ;

INSERT INTO DECLINAISON VALUES (2 ,'01','2-01', 50 , 0,'M'

,'ROUGE', 2, 0) ;

INSERT INTO DECLINAISON VALUES (2 ,'02','2-02', 50 , 0,'L'

,'ROUGE', 2, 0) ;

INSERT INTO DECLINAISON VALUES (2 ,'03','2-03', 50 , 0,'L'

,'BLEU', 2, 0) ;

## DELETE

Pour supprimer toutes les lignes d’une table :

**DELETE**  **FROM** nom\_de\_la\_table

Pour supprimer certaines lignes d’une table :

**DELETE**  **FROM** nom\_de\_la\_table

WHERE …

**UPDATE**

Pour mettre à jour une ou plusieurs colonnes :

**UODATE** nom\_de\_la\_table

**SET** colonne\_1 = valeur\_1

, colonne\_2 = valeur\_2

etc

**WHERE** …

Si le WHERE n’est pas précisé, c’est toute la page qui est mise à jour.

Il est possible de modifier le contenu d’une colonne à partir de sa valeur.

**UODATE** nom\_de\_la\_table

**SET** colonne\_1 = colonne\_1 \* valeur\_2 + 100c

etc

DCLGEN

Pour les description des tables et des vues, le plus simple est d’utiliser l’outil

DB2I PRIMARY OPTION MENU

Par exemple, pour la table TEST :

2 DCLGEN (Generate SQL and source language declarations)

Avec

* 1 SOURCE TABLE NAME ===> test
* 4 DATA SET NAME ... ===> 'MATA01.DCLGEN.COBOL(TEST)'

Dans le programme cobol, il faut écrire les lignes

EXEC SQL

INCLUDE TEST

END-EXEC.

La compilation les transformera en :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* COBOL DECLARATION FOR TABLE TEST\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

01 DCLTEST.

10 TESNO PIC S9(9) USAGE COMP.

10 TESLIBPIC X(20).

10 TESCODPIC X(02).

10 TESDATPIC X(10).

10 TESNB PIC S9(5)V99 COMP-3.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 5 \*

# SPUFI

Il est conseillé d’écrire chaque requête dans un membre TSO dédié.

Par exemple, on veut connaître la valeur du stock suivant tous les critères possibles : déclinaison, article, marque, entrepôt, etc. Et aussi la valeur du stock sans marque.

--- VALEUR DU STOCK PAR DECLINAISON

--

SELECT D.ID\_ARTICL , ID\_DECLINAISON

, STOCK , PRIX, STOCK \* PRIX AS VALEUR

FROM **ARTICLE** A

, **DECLINAISON** D

WHERE A.ID\_ARTICLE = D.ID\_ARTICL

;

---------+---------+---------+---------+---------+---------+---------+

ID\_ARTICL ID\_DECLINAISON STOCK PRIX VALEUR

---------+---------+---------+---------+---------+---------+---------+

1 01 50 10.50 525.00

1 02 50 10.50 525.00

2 01 50 100.50 5025.00

2 02 50 100.50 5025.00

2 03 50 100.50 5025.00

DSNE610I NUMBER OF ROWS DISPLAYED IS 5

DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100

On veut connaître la valeur du stock par article.

SELECT ID\_ARTICL

, **SUM**(STOCK \* PRIX) AS VALEUR

FROM ARTICLE A

, DECLINAISON D

WHERE A.ID\_ARTICLE = D.ID\_ARTICL

**GROUP BY** ID\_ARTICL

;

---------+---------+---------+---------+--------

ID\_ARTICL VALEUR

---------+---------+---------+---------+--------

1 1050.00

2 15075.00

Si on veut récupérer d’autres colonnes de la table article, il faut les spécifier dans le SELECt Et dans le GROUP BY. L’ordre peut être différent.

SELECT **ID\_ARTICL , DESIGNATION**

, **SUM**(STOCK \* PRIX) AS VALEUR

FROM ARTICLE A

, DECLINAISON D

WHERE A.ID\_ARTICLE = D.ID\_ARTICL

**GROUP BY ID\_ARTICL , DESIGNATION**

## DCLGEN

DCLGEN SSID: **DSN1**

===>

Enter table name for which declarations are required:

1 SOURCE TABLE NAME ===> **api12.employe** (Unqualified)

2 TABLE OWNER ..... ===> (Optional)

3 AT LOCATION ..... ===> (Optional)

Enter destination data set: (Can be sequential or partitioned)

4 DATA SET NAME ... ===> '**ACT12.SOURCE.DCLGEN(EMPLOYE)'**

5 DATA SET PASSWORD ===> (If password protected)

Enter options as desired:

6 ACTION .......... ===> **REPLACE** (ADD new or REPLACE old declaration)

7 COLUMN LABEL .... ===> YES (Enter YES for column label)

8 STRUCTURE NAME .. ===> (Optional)

9 FIELD NAME PREFIX ===> **w-**  (Optional)

10 DELIMIT DBCS .... ===> **YES** (Enter YES to delimit DBCS identifiers)

11 COLUMN SUFFIX ... ===> **YES** (Enter YES to append column name)

12 INDICATOR VARS .. ===> **NO** (Enter YES for indicator variables)

PRESS: ENTER to process END to exit HELP for more information

\*\*\*

DSNE110E DSN NOT VALID SUBSYSTEM ID, COMMAND TERMINATED

\*\*\*

# Les programmes Cobol DB2

1. En cobol, chaque ordre doit encapsulé par EXEC et END-EXEC :
2. EXEC SQL
3. ordre SQL
4. ;
5. END-EXEC

## SQLCA

SQLCA : SQL Communication Area, zone renseignée par DB2 après chaque ordre SQL

Il faut commencer par appeler la définition la zone commune de l’application DB2,SQLCA : SQL Common Area.

EXEC SQL

INCLUDE SQLCA

END-EXEC.

La compilation traduit en :

\*\*\*\*\*EXEC SQL

\*\*\*\*\* INCLUDE SQLCA

\*\*\*\*\*END-EXEC.

01 SQLCA.

05 SQLCAID PIC X(8).

\*\*\* constante valant SQLCA, utilisée comme repère par ceux qui fouillent dans les Dumps

05 SQLCABC PIC S9(9) COMP-4.

\*\*\* longueur de la zone qui vaut X'88’

05 SQLCODE PIC S9(9) COMP-4.

\*\*\*\*\* en binaire. Il faut passer le code SQLCODE souvent négatif dans une zone éditable –(9)9

\*\*\* zéro  : OK

\*\*\* négatif : erreur

\*\*\* positif : warning

\*\*\* +100 : non trouvé

05 SQLERRM.

\*\*\* longueur et valeur du message explicative

49 SQLERRML PIC S9(4) COMP-4.

49 SQLERRMC PIC X(70).

05 SQLERRP PIC X(8).

05 SQLERRD OCCURS 6 TIMES

PIC S9(9) COMP-4.

05 SQLWARN.

\*\*\* indicateur du warning

10 SQLWARN0 PIC X.

\*\*\* W => au moins 1 warning

10 SQLWARN1 PIC X.

\*\*\* valeur tronquée

10 SQLWARN2 PIC X.

\*\*\* elimination de NULL pour function

10 SQLWARN3 PIC X.

\*\*\* nombre de colonnes > nombre de host-variables

10 SQLWARN4 PIC X.

\*\*\* concerne les SQL dynamiques

10 SQLWARN5 PIC X.

\*\*\* ordre SQL de SQL/DS non valide en DB2

10 SQLWARN6 PIC X.

\*\*\* reserve pour le future

10 SQLWARN7 PIC X.

\*\*\* reserve pour le futur

05 SQLEXT.

\*\*\* reserve pour le futur

10 SQLWARN8 PIC X.

10 SQLWARN9 PIC X.

10 SQLWARNA PIC X.

10 SQLSTATE PIC X(5).

## Les variables hôtes (host variables)

Une variable hôte est une variable définie dans le programme et utilisée dans les ordres SQL

1. Dans le résultat d’une sélection SELECT … INTO  :VARIABLE
2. Dans la mise à jour INSERT …VALUE( :VARIABLE)
3. Dans les critères de sélection … WHERE COLONNE = :VARIABLE
4. Dans des expressions arithmétiques

Elle est précédée de « : » sans blanc à l’intérieur deEXEC SQL … END-EXEC.

Elle est de type compatible avec la colonne

Elle est de dernier niveau (de 01 à 48 ou 77) ou c’est une zone de groupe à 2 niveaux.

Elle peut être préfixée par une variable de niveau supérieur

Exemple :

MOVE '2015-01-02' TO TESDAT.

EXEC SQL

UPDATE TEST

SET TESDAT = :TESDAT

END-EXEC.

Exemple :

EXEC SQL

SELECT TESTDAT

INTO = :TESDAT

FROM TEST

WHERE TESTNO = :WS-TESTNO

END-EXEC.

## Les colonnes de longueur variable

Pour des colonnes de longueur variable, la variable doit regrouper deux variables de **niveau 49**, la première pour la longueur, la deuxième pour la valeur avec la longueur maximum

05 **FOURNI**.

49 FOURNI-LEN PIC S9(4) COMP.

49 FOURNI-TEXT PIC X(25).

EXEC SQL

SELECT **FOURNI**

INTO = :FOURNI

FROM FLAQ

WHERE FLAQNO = :WS-FLQNO

END-EXEC.

Règle d’écriture : il faut écrire SELECT le nom de la colonne, mais le contenu se trouve dans « COLONNE-TEXT »

01 WS-FOURNI-LIGNE.

05 WS-FLQNO PIC S9(3) COMP-3.

05 WS-LIBELLE PIC X(20).

Les 2 ordres suivants sont équivalents :

EXEC SQL

SELECT \* FROM FLAQ

INTO :WS-FOURNI-LIGNE

WHERE FLAQNO = :WS-FLQNO

END-EXEC.

EXEC SQL

SELECT WS-FLQNO , WS-LIBELLE

FROM FLAQ

INTO :WS-FLQNO , :WS-LIBELLE

WHERE FLAQNO = :WS-FLQNO

END-EXEC.

## Les variables indicateurs

Pour les colonnes qui peuvent être NULL , il faut associer à la variable hôte une variable indicateur, d’un demi-mot binaire, qui a :

* une valeur négative si la colonne est NULL
* une valeur à zéro si la conne est renseignée
* une valeur positive si la colonne est renseignée mais que le résultat est tronqué

Les variables indicateurs ne peuvent être utilisées dans une clause WHERE. On utilisera alors la COLONNE is NULL

Exemple :

05 WS-CODERESP PIC X(4).

…

05 WS-CODERESP-I PIC S9(4) COMP.

EXEC SQL

SELECT CODERESP

INTO : WS-CODERESP:WS-CODERESP-I

FROM MAISON

WHERE SIREN = :WS-SIREN

END-EXEC.

IF WS-CODERESP-I < ZERO

DISPLAY “PAS DE RESPONSABLE POUR SIREN “ WS-SIREN.

## Mise à NULL d’une colonne

Une colonne est mise à NULL en positionnant son indicateur à une valeur négative.

EXEC SQL

UPDATE MAISON

SET WS-CODERESP-I = -1

WHERE SIREN = :WS-SIREN

END-EXEC.

## Lire une ligne

Si la requête ne ramène qu’une seule ligne, on peut utiliser l’ordre SELECT

EXEC SQL

**SELECT**

TESNO, TESLIB , TESCOD, TESDAT, TESNB

**INTO**

: WS-TESNO, : WS-TESLIB , : WS-TESCOD, : WS-TESDAT

, : WS-TESNB

WHERE TESTNO = :WS-TESTNO

**FROM** TEST

END-EXEC.

SQLCODE = ZERO OK, la ligne a été trouvée, les host variables sont alimentées

SQLCODE = +100 aucune ligne n’a été trouvée

SQLCODE = -811 KO, plusieurs lignes ont été trouvées

## L’ordre DECLARE TABLE

La déclaration des tables en working permet au précompilateur de vérifier la cohérence entre l’ordre SQL et les tables de cet ordre SQL.

EXEC SQL

INCLUDE TEST

END-EXEC.

La compilation traduit ce lignes en

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DCLGEN TABLE(TEST)\*

\* LIBRARY(MATA01.DCLGEN.COBOL(TEST)) \*

\* ACTION(REPLACE) \*

\* LANGUAGE(COBOL) \*

\* QUOTE\*

\* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS\*

\* ( TESNO INTEGER NOT NULL ,

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

EXEC SQL DECLARE TEST TABLE

( TESNO INTEGER ,

TESLIB CHAR(20),

TESCOD CHAR(02),

TESDAT DATE ,

TESNB DECIMAL (5,2)

) END-EXEC.

\* PRIMAREY KEY (TESNO)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* COBOL DECLARATION FOR TABLE TEST \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

01 DCLTEST.

10 TESNO PIC S9(9) USAGE COMP.

10 TESLIBPIC X(20).

10 TESCODPIC X(02).

10 TESDATPIC X(10).

10 TESNB PIC S9(5)V99 COMP-3 .

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 5 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Lire plusieurs lignes

En cobol pour lire plusieurs lignes, on utilise un curseur, qu’on déclare, qu’on ouvre, qu’on lit (FETCH), et qu’on ferme.

Le nom du curseur est unique pour un programme donné.

La clause INTO est absente du SELECT de la déclaration du curseur, mais est présente dans le FETCH.

La déclaration doit être faite avant toute utilisation. Il est conseillé de le faire en WORKING.

Comme pour les fichier en lecture séquentielle, il faut ouvrir le curseur, faire une boucle de FETCH, et fermer le curseur.

**Déclaration** :

EXEC SQL

**DECLARE** CURSEUR-TEST **CURSOR FOR**

**SELECT** TESNO, TESLIB

, TESCOD, TESDAT, TESNB

**FROM** TEST

END-EXEC.

**Ouverture** :

EXEC SQL

**OPEN** CURSEUR-TEST

END-EXEC.

**Fetch :**

EXEC SQL

**FETCH** CURSEUR-TEST

**INTO**  :TESNO, :TESLIB , :TESCOD

, :TESDAT, :TESNB

END-EXEC.

**Fermeture :**

EXEC SQL

**CLOSE** CURSEUR-TEST

END-EXEC.

Dans le Fetch, il faut indiquer les variables définies en Working, précédées de “:”.

Le FETCH est séquentiel, sans retour arrière, sans positionnement sur une nième ligne.

Attention : tous les curseurs sont fermés lors d’un COMMIT.

## UPDATE ou DELETE avec un curseur

Si on doit modifier ou supprimer une ligne lue avec un curseur, on ajoute FOR UPDATE à la déclaration du curseur.

En cas de mise à jour, le SELECT du DECLARE CURSOR ne peut pas :

* Référencer plus d’une table
* Référencer une fonction colonne ( MIN, MAX, …)
* Utiliser les clauses DISTINCT, UNION, GROUP BY, ORDER BY

EXEC SQL

**DECLAR**E CURSEUR-TEST **CURSOR FOR**

**SELECT** TESNO, TESLIB , TESCOD, TESDAT, TESNB

**FROM** TEST

**FOR UPDATE OF** col1, col2, etc

END-EXEC.

**Le Fetch :**

EXEC SQL

**FETCH** CURSEUR-TEST

**INTO** :TESNO, :TESLIB , :TESCOD, :TESDAT, :TESNB

END-EXEC.

**L’Update :**

EXEC SQL

**UPDATE** nom\_de\_la table

**SET** col1 = …, col2 = …, etc

**WHERE CURRENT OF** CURSEUR-TEST

END-EXEC.

**Le Delete :**

EXEC SQL

**DELETE**

**FROM** TEST

**WHERE CURRENT OF** CURSEUR-TEST

END-EXEC.

# JCL

## Le JCL de compilation

API16.COB.JCL(JCOMPDB2)

//**API15**DBC JOB (ACCT#),'COMPDB2',MSGCLASS=H,REGION=4M,

// CLASS=A,MSGLEVEL=(1,1),NOTIFY=&SYSUID,

// COND=(4,LT)

//\*

//\*------------------------------------------------------\*

//\* ===> CHANGER XX PAR N¢ DU GROUPE (XX 01 @ 15) \*

//\* CHANGER API15DB$ PAR LE NOM DU PROGRAMME \*

//\*------------------------------------------------------\*

//\*

//\*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=-=-=-=-=-=-=-\*

//\* CETTE PROCEDURE CONTIENT 5 STEPS :

//\* ======> SI RE-EXECUTION FAIRE RESTART AU "STEPRUN"

//\*

//\* 1/ PRECOMPILE DB2

//\* 2/ COMPILE COBOL II

//\* 3/ LINKEDIT (DANS FORM.CICS.LOAD)

//\* 4/ BIND PLAN PARTIR DE API15.SOURCE.DBRMLIB

//\* 5/ EXECUTE DU PROGRAMME

//\* LES PROCEDURES SE TROUVENT DANS SDJ.FORM.PROCLIB

//\*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=-=-=-=-=-=-=-\*

//PROCLIB JCLLIB ORDER=SDJ.FORM.PROCLIB

//\*

// SET SYSUID=**API15**,

// NOMPGM=**AJCD113**

//\*

//APPROC EXEC COMPDB2

//STEPDB2.SYSLIB DD DSN=&SYSUID..COB.CPY,DISP=SHR

//STEPDB2.SYSIN DD DSN=&SYSUID..COB.SRC(&NOMPGM),DISP=SHR

//STEPDB2.DBRMLIB DD DSN=&SYSUID..COB.DBRM(&NOMPGM),DISP=SHR

//STEPCOB.SYSLIB DD DSN=&SYSUID..COB.CPY,DISP=SHR

// DD DSN=**API15**.COB.CPY,DISP=SHR

//STEPLNK.SYSLMOD DD DSN=&SYSUID..COB.LOAD(&NOMPGM),DISP=SHR

//\*

//\*--- ETAPE DE BIND --------------------------------------

//\*

//\*DBRMLIB DD DSN=&SYSUID..DB2.DBRMLIB,DISP=SHR

//DBRMLIB DD DSN=&SYSUID..COB.DBRM,DISP=SHR

//SYSTSPRT DD SYSOUT=\*,OUTLIM=25000

//SYSTSIN DD \*

DSN SYSTEM (DSN1)

BIND PLAN (**AJCD113**) -

QUALIFIER (**API15**) -

ACTION (REPLACE) -

MEMBER (**AJCD113**) -

VALIDATE (BIND) -

ISOLATION (CS) -

ACQUIRE (USE) -

RELEASE (COMMIT) -

EXPLAIN (NO)

/\*

//STEPRUN EXEC PGM=IKJEFT01,COND=(4,LT)

//STEPLIB DD DSN=&SYSUID..COB.LOAD,DISP=SHR

//SYSOUT DD SYSOUT=\*,OUTLIM=1000

//SYSTSPRT DD SYSOUT=\*,OUTLIM=2500

//SYSTSIN DD \*

DSN SYSTEM (DSN1)

RUN PROGRAM(**AJCD113**) PLAN (**AJCD113**)

//

La compilation d’un programme DB2 comporte plusieurs phases :

* la précompilation
* la génération du DBRM
* le bind
* la compilation

## La précompilation

La précompilation transforme le source en source étendu en ajoutant les spécificités DB2 et enregistre en working un TIMESTAMP

Dans le cas de CICS, il y a une étape supplémentaire.

## Le BIND

Le BIND construit le plan :

* Il valide les ordres SQL contenus dans le DBRM (= module de requête de base de donnée qui encapsule les instructions SQL et leurs métadonnées) via le catalogue DB2
* Il vérifie les autorisations
* Il choisit les chemins optimaux, suivant les index, la taille des table, ...
* Il met à jour le catalogue (= tables système DB2 commençant par SYSIBM.SYS) avec les caractéristiques du plan et de ses DBRM
* Il crée une structure de contrôle, le plan d’application « load module » stocké dans le directory de DB2

Le BIND peut être exécuté

* Via un panel de DB2I (le panel de SPUFI, DCLGEN, etc)
* via la sous-commande BIND du processeur de commande DSN , en batch

Paramètre ISOLATION :

* c’est le niveau de verrouillage des pages en lecture pour le programme associé
* - CS, Cursor Stability, libère la page verrouillée au changement de page
* - RR, Repeatable Read, libère la page verrouillée au point de synchronisation

Paramètres CREATOR et OWNER :

BIND PLAN(ce\_plan) … OWNER (proprio)

Proprio est le propriétaire du plan

* C’est le CREATOR dans le catalogue, enregistré comme BOUNDBY
* Il doit avoir les privilège de BINDAA (si ajout) ou BIND sur ce\_plan (si REPLACE)
* Il doit avoir les autorisations pour exécuter les ordres SQL du plan
* Il devient le qualifieur des tables et des vues référencées

Dans le cas d’un programme principal qui appelle des **sous-programmes,** il faut spécifier tous ceux qui ont des ordres SQL :

Si le programme\_principal a des ordres SQL :

BIND PLAN(**programme\_principal**)

MEMBER (**programme\_principal, sous\_programme1, sous\_programme2**)

Si le programme\_principal n’a pas d’ordre SQL :

BIND PLAN(**programme\_principal**)

MEMBER (**sous\_programme1, sous\_programme2**)

Le contrôle des timestamps des DBRM et du timestamps du load module signale la désynchronisation par SQLCODE = -818

Si la structure de la base de données a été modifiée (ajout d’un colonne, modification d’un index, etc), la stratégie de DB2 devient différente et il faut refaire le plan. On peut « rebinder » le plan sans être obligé de recompiler le programme Cobol qui n’a pas été modifié.

## REBIND, BIND REPLACE et FREE

La modification d’un plan est exécutée par

* REBIND en cas de
  + changement d’autorisation
  + création et de suppression d’index
  + mise à jour des statistiques utilisées par l’optimiseur (RUNSTATS, …)
  + mise à jour des paramètres du plan
* BIND REPLACE dans tous les cas,
  + Mais obligatoire en cas de recompilation de DBRM

REBIND PLAN(le-plan)

FREE détruit le plan

FREE PLAN(le\_plan)

FREE PLAN(\*) : pour détruire tous les plans dont l’utilisateur a le privilège de BIND, y compris ceux dont l’autorisation est « PUBLIC »

Mise à jour des plans du catalogue :

* 1 ligne par plan dans SYSIBM.SYSPLAN
* N lignes par plan dans SYSIBM.SYSPLANDEP
* 1 ligne par DBRM d’un plan dans SYSIBM.SYSDBRM
* N lignes par ordres SQL de chaque DBRM dans SYSIBM.SYSSTMT

# Exercice

Ecrire les requêtes pour les cas suivants.

Fournir la description des tables et des objets DB2 (index, etc).

Fournir les requêtes écrites.

Fournir les résultats obtenus.

## Le contexte

Un centre de distribution gère des articles et veut connaître le prix du stock suivant certains critères.

- l’article. L’identifiant est un nombre séquentiel. Il a une famille, une désignation et un prix , et éventuellement une marque.

- la famille a un identifiant alphanumérique et une désignation

- la marque a un identifiant alphanumérique et une désignation. Il ne faut pas prendre de nom qui existe dans la réalité.

- la déclinaison est une variante de l’article, qui a une taille (parfois taille unique) et un coloris (parfois l’article n’a qu’un seul coloris). L’identifiant est la concaténation de l’identifiant de l’article et un nombre. Elle a aussi un stock, un stock d’alerte (en dessous de ce stock, il faut passer commande), une quotité (multiple de quantité à commander, par exemple les œufs se vendent par 6, on commande donc 6, 12, 18 etc)

- l’entrepôt. Il a un identifiant et une désignation. Chaque entrepôt stocke les articles d’une ou deux familles.

## MLD

Marque

Famille

Article

Entrepot

Déclinaison

Identifiant déclinaison

Code article

Stock

Stock alerte

Taille

Coloris

Quotité

Quantité commandée

## La demande

On veut connaître la valeur du stock suivant tous les critères possibles : déclinaison, article, marque, entrepôt, etc. Et aussi la valeur du stock sans marque.

Pour les déclinaisons dont le stock a atteint le stock d’alerte, il faut passer une commande avec un multiple de sa quotité, sans commander de stock trop important.

Il faut écrire une requête pour mémoriser le stock commandé.

Le stock d’alerte est comparé avec le cumul du stock réel et du stock commandé pour éviter trop de commandes le temps de la livraison.

Choisissez la manière dont vous allez gérer le stock commandé (quelle table?).

Gérer les livraisons :

- écrire les requêtes qui passent le stock commandé dans le stock réel.

- écrire les requêtes qui simulent des diminutions de stock (ventes) et diminuent donc les stocks réels.

- refaire le calcul valorisé du stock selon divers critères, comme plus haut.

- recommencer le calcul pour d’éventuelles commandes automatiques.

Refaire plusieurs fois le processus en contrôlant les valeurs.

# Normes et Contrôles spécifiques

## Nommage

Pour la plupart des systèmes de gestion de bases de données relationnelles utilisant SQL, comme MySQL, PostgreSQL, et SQL Server, voici quelques règles courantes concernant les noms des objets de base de données (tables, colonnes, etc.) :

1. **Caractères valides** : Les noms peuvent contenir des lettres (majuscules ou minuscules), des chiffres et le caractère de soulignement (\_).
2. **Premier caractère** : Le premier caractère doit être une lettre. Cela signifie qu'il ne peut pas commencer par un chiffre ou un caractère de soulignement.
3. **Longueur** : La longueur maximale des noms peut varier selon le système de gestion de base de données, mais elle est généralement suffisamment grande pour couvrir la plupart des besoins pratiques.
4. **Sensibilité à la casse** : Certains systèmes de gestion de base de données sont sensibles à la casse pour les noms d'objets, tandis que d'autres ne le sont pas. Par exemple, MySQL est sensible à la casse sous Unix mais pas sous Windows, tandis que PostgreSQL convertit les noms non cités en minuscules.
5. **Mots réservés** : Les noms ne doivent pas être des mots réservés par SQL, comme SELECT, INSERT, TABLE, etc., à moins qu'ils ne soient correctement échappés.

Il est toujours bon de consulter la documentation spécifique de votre système de gestion de base de données pour connaître les règles exactes et les bonnes pratiques.

## Cas particulier de l’email

Dans la création d'une base de données, il n'existe pas de propriété native ou de type de données spécifique qui garantit qu'un champ contient une adresse e-mail valide, c'est-à-dire avec exactement un caractère arobase (@). Cependant, on peut mettre en place des validations pour s’assurer que les données insérées dans un champ respectent le format d'une adresse e-mail.

Voici quelques approches courantes pour valider les adresses e-mail dans une base de données :

1. **Contraintes de validation** : Vous pouvez utiliser des contraintes CHECK dans SQL pour valider le format des données. Par exemple, en SQL, vous pouvez créer une contrainte qui vérifie la présence d'un seul caractère @.
2. **Validation au niveau de l'application** : Avant d'insérer des données dans la base de données, vous pouvez utiliser des expressions régulières dans votre code applicatif pour valider le format de l'adresse e-mail.
3. **Types de données personnalisés** : Certains systèmes de gestion de base de données permettent de créer des types de données personnalisés avec des règles de validation spécifiques.
4. **Triggers** : Vous pouvez utiliser des triggers pour valider les données avant qu'elles ne soient insérées ou mises à jour dans la base de données.

Voici un exemple de contrainte CHECK en SQL pour s'assurer qu'un champ contient exactement un caractère @ :

CREATE TABLE utilisateurs (

id SERIAL PRIMARY KEY,

email VARCHAR(255) CHECK (email LIKE '%\_@\_%.\_%' AND email NOT LIKE '%@%@%' ) );

Cette contrainte vérifie que l'adresse e-mail contient un seul caractère @ et qu'il est entouré de caractères valides. Cependant, cette méthode a ses limites et ne couvre pas tous les cas de validation possibles pour une adresse e-mail. Une validation plus robuste est généralement effectuée au niveau de l'application.

## Mots réservés

La liste de ces mots réservés ne doivent pas être affectés à des noms de bases de données, de tables, de colonnes ou de tout autre objet de base de données. Certains SGBD réservent d'autres mots.

**A**BORT ADMIN AGGREGATE ALIGN ALL ALLOCATE ANALYSE ANALYZE AND ANY AS ASC

**B**ETWEEN BINARY BIT BOTH

**C**ASE CAST CHAR CHARACTER CHECK CLUSTER COALESCE COLLATE COLLATION COLUMN CONSTRAINT COPY CROSS CURRENT CURRENT\_CATALOG CURRENT\_DATE CURRENT\_DB CURRENT\_SCHEMA CURRENT\_SID

CURRENT\_TIME CURRENT\_TIMESTAMP CURRENT\_USER CURRENT\_USERID CURRENT\_USEROID

**D**EALLOCATE DEC DECIMAL DECODE DEFAULT DEFERRABLE DESC DISTINCT DISTRIBUTE DO

**E**LSE END EXCEPT EXCLUDE EXISTS EXPLAIN EXPRESS EXTEND EXTERNAL EXTRACT

**F**ALSE FIRST FLOAT FOLLOWING FOR FOREIGN FROM FULL FUNCTION

**H**AVING

**G**ENSTATS GLOBAL GROUP

**I**DENTIFIER\_CASE ILIKE IN INDEX INITIALLY INNER INOUT INTERSECT INTERVAL INTO

**L**EADING LEFT LIKE LIMIT LISTEN LIMIT LOAD LOCAL LOCK

**M**ATERIALIZED MINUS MOVE

**N**ATURAL NCHAR NEW NOT NOTNULL NULL NULLIF NULLS NUMERIC NVL NVL2

**O**FF OFFSET OLD ON ONLINE ONLY OR ORDER OTHERS OUT OUTER OVER OVERLAPS

**P**ARTITION POSITION PRECEDING PRECISION PRESERVE PRIMARY

**R**ESET REUSE RIGHT ROWS ROWSET RULE REUSE

**S**EARCH SELECT SEQUENCE SESSION\_USER SETOF SHOW SOME SUBSTRING SYSTEM

**T**ABLE THEN TIES TIME TIMESTAMP TO TRAILING TRANSACTION TRIGGER TRIM TRUE

**U**NBOUNDED UNION UNIQUE USER USING

**V**ACUUM VARCHAR VERBOSE VERSION VIEW

**W**HEN WHERE WITH WRITE

Table des matières

[Les ordres SQL 2](#__RefHeading___Toc11921_1874922832)

[Mises à jour des tables 3](#__RefHeading___Toc7480_1874922832)

[CREATE TABLE 3](#__RefHeading___Toc429581227)

[INSERT 5](#__RefHeading___Toc21754_1874922832)

[DELETE 6](#__RefHeading___Toc21754_1874922832_Copie)

[SPUFI 9](#__RefHeading___Toc8560_1874922832)

[DCLGEN 10](#__RefHeading___Toc4963_1874922832)

[Les programmes Cobol DB2 11](#__RefHeading___Toc19508_1757787122)

[SQLCA 11](#__RefHeading___Toc429581202)

[Les variables hôtes (host variables) 12](#__RefHeading___Toc429581204)

[Les colonnes de longueur variable 13](#__RefHeading___Toc429581205)

[Les variables indicateurs 14](#__RefHeading___Toc429581206)

[Mise à NULL d’une colonne 15](#__RefHeading___Toc429581207)

[Lire une ligne 16](#__RefHeading___Toc429581209)

[L’ordre DECLARE TABLE 17](#__RefHeading___Toc429581208)

[Lire plusieurs lignes 18](#__RefHeading___Toc429581210)

[UPDATE ou DELETE avec un curseur 19](#__RefHeading___Toc429581211)

[JCL 20](#__RefHeading___Toc9261_1874922832)

[Le JCL de compilation 20](#__RefHeading___Toc18788_1874922832)

[La précompilation 21](#__RefHeading___Toc429581214)

[Le BIND 21](#__RefHeading___Toc429581215)

[REBIND, BIND REPLACE et FREE 23](#__RefHeading___Toc429581216)

[Exercice 24](#__RefHeading___Toc4840_1874922832)

[Le contexte 24](#__RefHeading___Toc4842_1874922832)

[MLD 25](#__RefHeading___Toc4844_1874922832)

[La demande 26](#__RefHeading___Toc4846_1874922832)

[Normes et Contrôles spécifiques 27](#__RefHeading___Toc21380_145764793)

[Nommage 27](#__RefHeading___Toc20870_1074248952)

[Cas particulier de l’email 27](#__RefHeading___Toc5253_1874922832)

[Mots réservés 29](#__RefHeading___Toc51547_3889704470)