PROGRAMMATION

DB2

10/12/2020

TABLE DES MATIERES

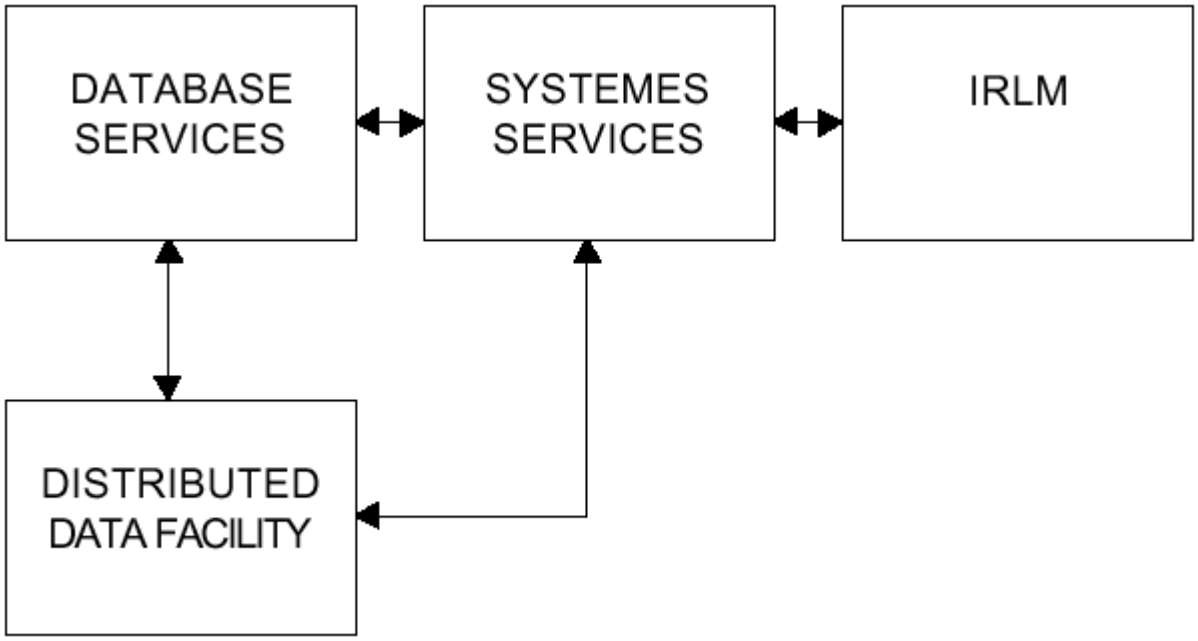
Contenu

TABLE DES MATIERES

Objectifs du chapitre

- Apprendre à mettre en oeuvre des applications dans l'environnement du Système de Gestion de Bases de Données DB2.
- Composants de DB2
- Préparation d'un programme
- Construction d'un plan d'application
- Utilitaires et commandes DB2

Composants de DB2



Chacun des ces composants fonctionne dans son propre espace adresse.

Database Services

- · Assure l'exécution des ordres SQL.
- · Prépare le traitement des programmes d'application etcontrôle leur exécution.

System Services

- Assure la connexion entre DB2 et les autres sous-systèmes (TSO, IMS, CICS).
- Réalise des fonctions de service (journalisation, statistiques, etc.)

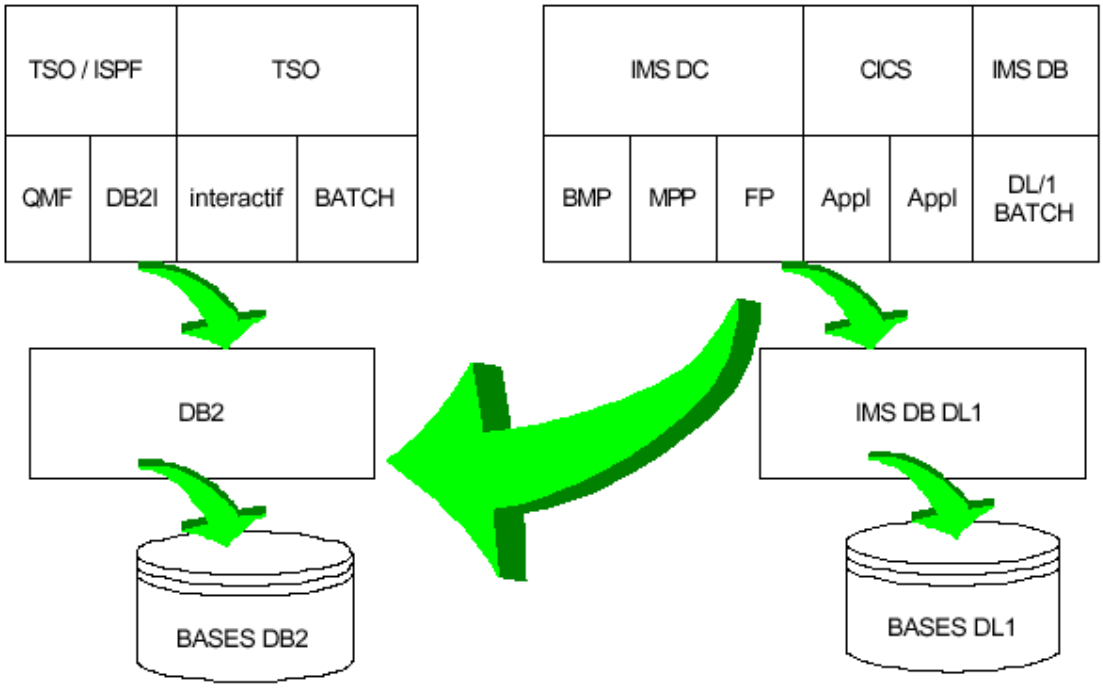
IMS Resource Lock Manager (IRLM)

- Gère le blocage des ressources demandées simultanément par différents utilisateurs.

Distributed Data Facility (DDF) · Composant optionnel.

- Permet l'accès à des données résidant sur un autresystème relationnel.

DB2 communique avec des gestionnaires d'applications.



DB2 et les applications

Il est possible d'accéder aux fonctions de DB2 :

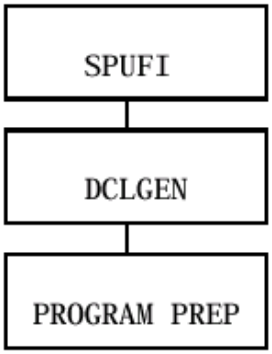
1. A partir de **TSO** de façon interactive directement depuis un terminal, en soumettant un travail en temps différé, · à partir d'applications TSO/ISPF comme QMF ou DB2I.

2. A partir de programmes fonctionnant sous le contrôle d'**IMS/DC** : depuis des BMP, des transactions temps réel (MPP), des régions Fast Path (FP).

3. Depuis des programmes **batch DL/**1.

4. Depuis des programmes fonctionnant sous **CICS**.

Préparation d'un programme

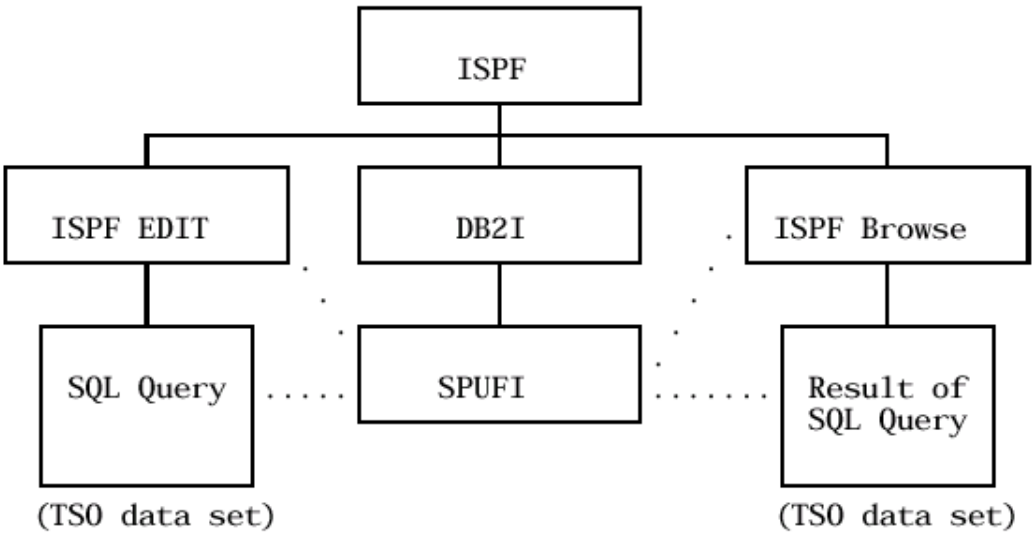


Run SQL interactively
(Objective 2)

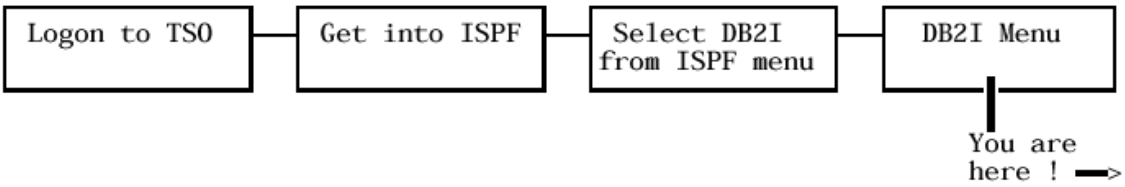
Generate SQL-related code
(Objective 3)

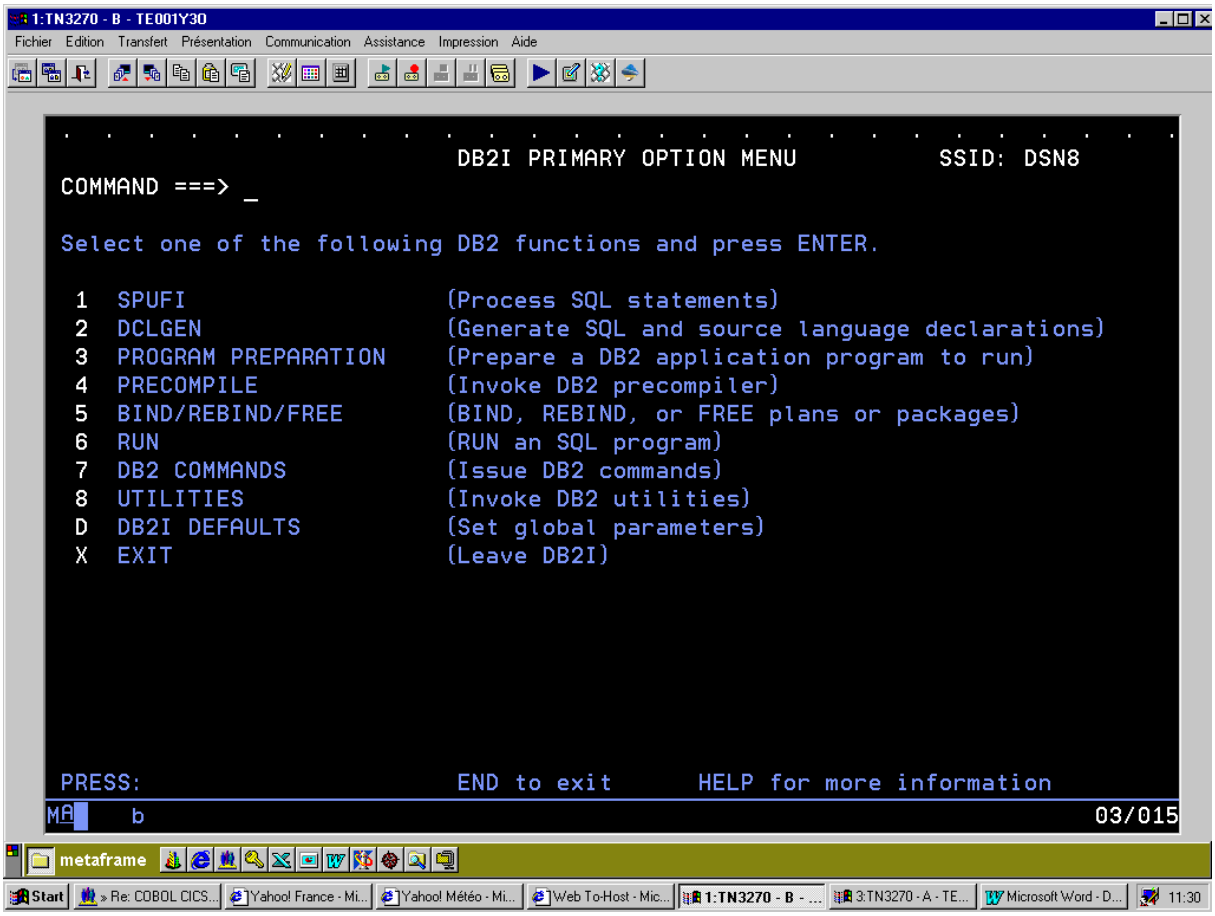
Precompile, bind, compile, link, run
(Objective 4)

TASK	TSO	DB2I	FOREGROUND	BACKGROUND	JCL
SPUFI	No	Yes	Yes	No	No
DCLGEN	Yes	Yes	Yes	Yes	Yes
Program Prep	Yes	Yes	Yes	Yes	Yes

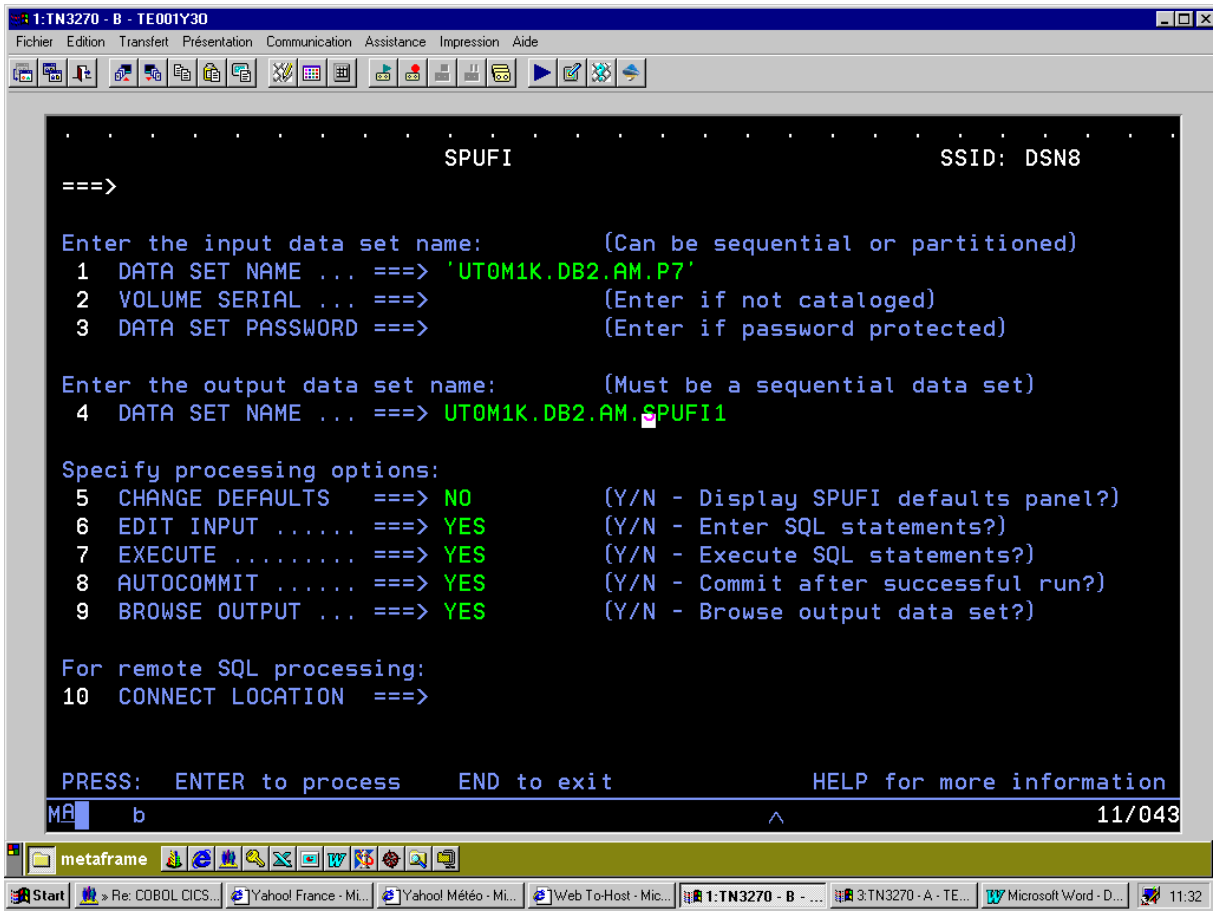


Préparation d'un programme





Préparation d'un programme



Exécution interactive d'ordres SQL

- **SPUFI** (SQL Processor Using File Input) permet d'exécuter directement depuis un terminal des ordres SQL.

- Les ordres à exécuter sont placés dans un fichier avec l'éditeur PDF.
- Ils doivent être séparés par le caractère : " ; ".

- Le résultat sera rangé dans un fichier de sortie et/ou consulté directement à l'écran.

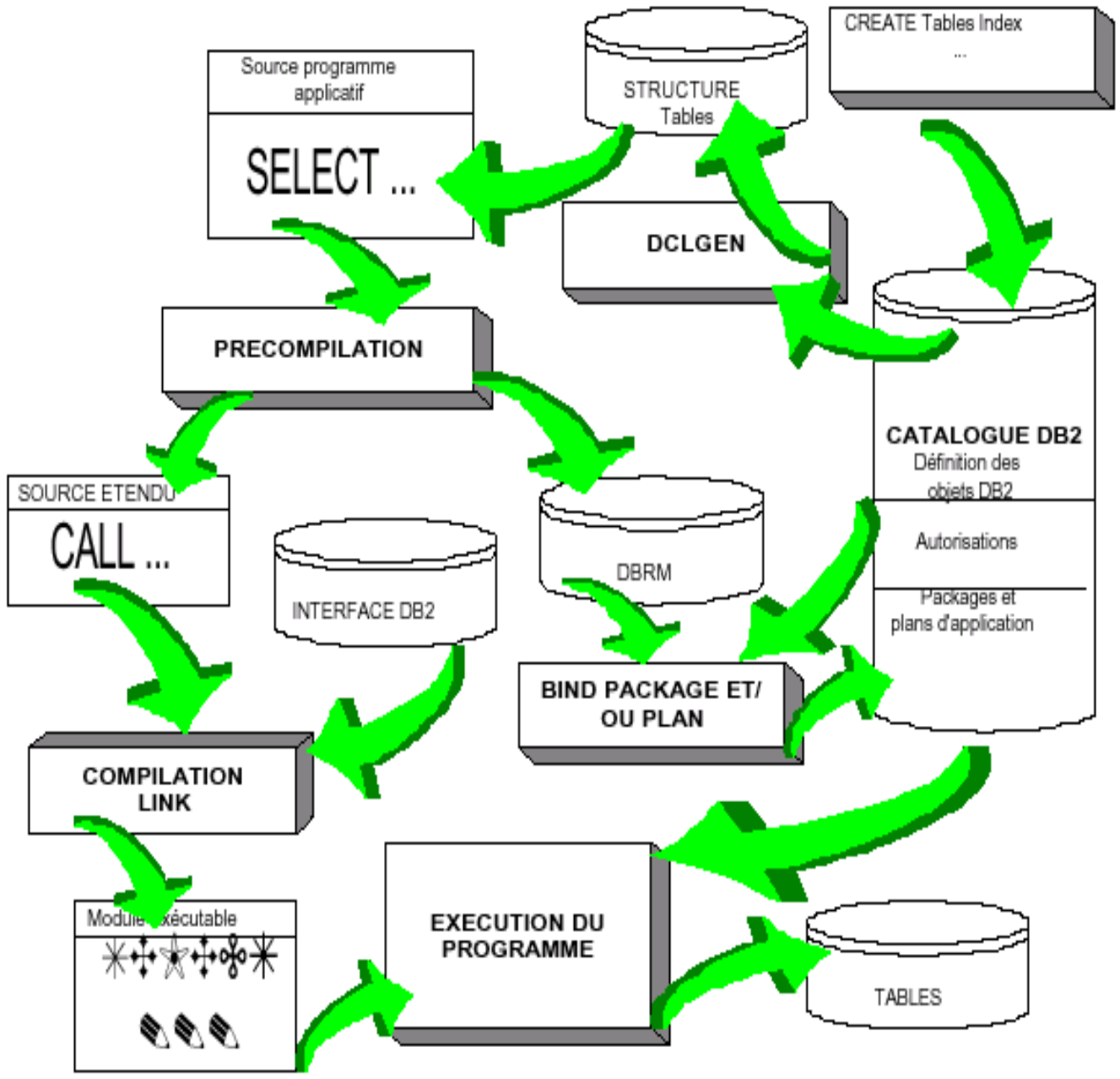
- Utilisation : apprentissage, mise au point.

Exécution d'ordres SQL depuis des programmes

Des ordres SQL peuvent être inclus dans des programmes d'application écrits dans divers langages (PL/I, COBOL, FORTRAN, Assembleur ou C).

Ces programmes (dits *programmes hôtes*) devront être préparés et exécutés de façon particulière pour pouvoir accéder aux ressources de DB2.

Développement d'un programme



Générateur de déclarations

Le Générateur de Déclarations **(DCLGE**N) produit, à partir d'une définition de table déjà exécutée (et rangée dans le catalogue DB2)

- **une définition de structure** COBOL, C ou PL/I qui pourra être appelée depuis un programme source par une instruction INCLUDE

DECLARE TABLE

- Cette opération est facultative mais permet d'assurer la cohérence entre la définition SQL d'une table et sa définition dans les programmes d'application.

- **DCLGEN** peut être exécuté directement sous DB2I ou par l'intermédiaire d'une commande TSO

.

Précompilation

Une étape de précompilation est nécessaire pour traiter les instructions SQL incluses dans le *langage hôte*.

Fonctions du précompilateur :

1. Séparation des instructions SQL et variables associées,des instructions du langage hôte.

2. Vérification de la syntaxe des ordres SQL.

3. Préparation du programme hôte pour qu'il puisse être compilé normalement (instructions SQL remplacées par des CALLs).

4. Production du module de requêtes **(DBR**M) contenant les informations relatives aux instructions SQL du programme. Il est rangé dans un fichier partitionné et sera utilisé pour l'élaboration du package ou du plan.

5. La précompilation peut être effectuée sous DB2I (option "program preparation" ou "precompile"), par une procédure de commandes TSO ou par une soumission en batch

Construction de packages et de plans

Le programme ainsi produit ne peut être exécuté sans que ses liens avec les ordres SQL ne soient établis.

C'est le rôle du **BIND** :
· créer, à partir du DBRM produit par le précompilateur, un **package** ou un **pla**n.

Il peut être exécuté sous DB2I (option "bind/rebind/free"), par des commandes TSO ou une soumission en batch.

Fonctions principales du BIND

1. · Valider les instructions SQL.

2. · Construire les noms complets d'objets.

3. · Vérifier que la personne effectuant la fonction BIND est autorisée à exécuter les ordres SQL du programme.

4. Choix de la meilleure stratégie d'accès par l'optimiseur pour chaque ordre SQL du programme, en considérant :

5. les volumes des tables

6. la présence éventuelle d'index

7. Construire un package ou un plan d'application, contenant, pour chaque ordre SQL, des informations sur le chemin d'accès à suivre.

8. Package : un pour chaque programme.

9. Plan : un pour l'ensemble des programmes et sous-programmes d'une application. Constitué d'une liste de packages ou d'un ensemble de DBRMs (ou des deux).

Packages

Depuis la version 2.3 de DB2, il est possible de construire des PLANS d'application à partir, non plus directement de DBRM, mais de PACKAGES.

Un PACKAGE DB2 est le résultat du BIND d'un seul DBRM. Pour exécuter un PACKAGE, il faut l'inclure dans la liste des PACKAGES d'un PLAN d'application (paramètre PKLIST) en plus, ou de préférence à la place, de la liste des DBRM (paramètre MEMBER).

Un PACKAGE est identifié par les éléments suivants :

LOCATION (identifie le SGBD dans un système distribué, par défaut il s'agit du DB2 local) ;

COLLECTION (regroupement logique de PACKAGES) ;

DBRM à l'origine de la construction du PACKAGE.

Packages

Lorsque les PLANS d'application sont construits directement à partir de DBRM, l'administration et la maintenance constituent un ensemble d'opérations fréquentes et lourdes (et coûteuses en ressources machine).

En effet, la modification d'un sous-programme DB2, et par conséquent de son DBRM, nécessite la reconstruction de tous les PLANS utilisant ce DBRM.

Par ailleurs, le BIND d'un PLAN pour cause de modification de l'un de ses DBRM, entraîne l'analyse de toutes les instructions de tous les DBRM du PLAN.

Par contre, lorsque les PLANS sont construits à partir de PACKAGES uniquement, le traitement des DBRM est effectué lors du BIND PACKAGE.

- La modification d'un DBRM n'implique que la reconstruction du PACKAGE correspondant.
- La prise en compte de cette opération par tous les PLANS utilisant ce PACKAGE est dynamique.

BIND ET REBIND

BIND

- Les packages et les plans d'application sont rangés dans la directory DB2 et leur descriptions (caractéristiques générales, ordres SQL d'origine, ...) dans le catalogue DB2.
- Ils pourront ainsi être réutilisés sans qu'il soit nécessaire de refaire le même travail de préparation à chaque exécution.

REBIND

- Effectue la reconstruction d'un package ou d'un plan d'application (sans avoir à recompiler les programmes concernés).
- Nécessaire et suffisant en particulier quand on crée un nouvel index et que l'on veut que la stratégie d'accès tienne compte de cet index.
- Si le plan est constitué de packages, il est possible de reconstruire un seul package indépendamment de l'ensemble du plan.

Exécution du programme

DB2 effectue les opérations suivantes :

1. Vérification de l'autorisation d'exécution du plan
2. Vérification de la cohérence du plan (index annulés~?)
3. REBIND automatique le cas échéant
4. Vérifications possibles à l'exécution (SQL dynamique)

Utilitaires

Il existe plusieurs utilitaires opérant en temps différé.
Ils peuvent être lancés depuis DB2I (option "utilities"), ou à partir de procédures.

LOAD
Chargement initial performant d'une table et des index associés.

REORG
Réorganisation totale ou partielle d'une table et/ou de ses index.

COPY, MERGECOPY, QUIESCE, REPORT, RECOVER, MODIFY
Sauvegarde et restauration totale ou partielle de tables ou d'index.

CHECK
Vérification de cohérence des index et des données.

RUNSTATS, STOSPACE
Mise à jour du catalogue pour permettre à l'optimiseur de choisir le meilleur chemin d'accès.

Commandes

Les commandes sont destinées à l'exploitation du système.
Elles sont identifiées par un verbe précédé d'un caractère désignant le sous-système DB2 auquel la commande s'adresse.
Elles peuvent être lancées depuis une console MVS, un terminal TSO, un terminal CICS ou IMS autorisé.

Elles permettent en particulier :

- de démarrer DB2 ou une de ses ressources (START),
- d'arrêter DB2 ou une de ses ressources (STOP),
- de contrôler l'état d'une ressource (DISPLAY),
- de terminer un utilitaire (TERM).

SQL dans les programmes éléments de base

Objectif du chapitre

1. Écriture d'un programme communiquant avec SQL
2. Structure du programme
3. Exploitation globale d'une table
4. Exploitation d'une table ligne à ligne
   - Exemple de programme

Syntaxe des ordres SQL inclus dans un programme

En COBOL, chaque ordre SQL inclus dans un programme doit être :

- précédé par **EXEC SQL**
- terminé par **END-EXEC**

```
EXEC SQL
     SELECT .....
END-EXEC
```

- Les mots EXEC et SQL doivent figurer sur la même ligne.

Les ordres « EXEC SQL » et « END-EXEC » doivent commencer en colonne 12 en « Working section » comme en « PROCEDURE Division »

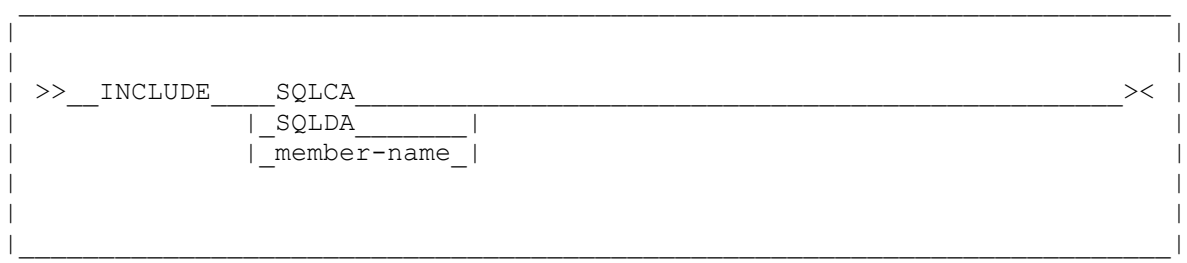
Zone de communication avec SQL

SQLCA

SQL Communication Area

Dans cette zone, DB2 place des informations relatives au résultat de l'exécution de chaque ordre SQL.

Zone en WORKING-STORAGE SECTION du programme COBOL, définie par l'instruction INCLUDE :



WORKING-STORAGE SECTION.

```
EXEC SQL
     INCLUDE SQLCA

END-EXEC
```

Zone de communication avec SQL

```
01 SQLCA.
        05 SQLCAID      PIC X(8).
        05 SQLCABC      PIC S9(9) COMP-4.
        05 SQLCODE      PIC S9(9) COMP-4.
        05 SQLERRM.
           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.
           10 SQLWARN0  PIC X.
           10 SQLWARN1  PIC X.
           10 SQLWARN2  PIC X.
           10 SQLWARN3  PIC X.
           10 SQLWARN4  PIC X.
           10 SQLWARN5  PIC X.
           10 SQLWARN6  PIC X.
           10 SQLWARN7  PIC X.
        05 SQLEXT.
           10 SQLWARN8  PIC X.
           10 SQLWARN9  PIC X.
           10 SQLWARNA  PIC X.
           10 SQLSTATE  PIC X(5).
```

SQLCA

SQLERRP Informations pour diagnostic d'erreur
(ex : nom de module db2)
SQLERRD(3) Contient :
Après un UPDATE, le nombre de lignes mises à Jour
Après un INSERT, le nombre de lignes ajoutées
Après un DELETE, le nombre de lignes supprimées (les lignes supprimées dans les tables dépendantes suite à l'option CASCADE ne sont pas comptabilisées)
-1 après une suppression de masse dans une table d'un tablespace segmenté
SQLERRD(4) Timeron (SQL dynamique)
SQLWARN Groupe de huit indicateurs
SQLWARN0 Blanc si tous les indicateurs sont à blanc
'W' si au moins un des indicateurs suivant a la valeur 'W'
SQLWARN1 ''W' si une valeur de chaîne de caractères a été tronquée en étant transmise au programme
SQLWARN2 'W' si des valeurs nulles ont été éliminées de l'argument d'une fonction sur colonne
SQLWARN3 'W' si le nombre de colonnes de la table résultante est supérieur au nombre de variables hôtes
SQLWARN4 'W' si un ordre préparé, UPDATE ou DELETE, ne contient pas la clause WHERE
SQLWARN5 'W' si l'ordre n'a pas été exécuté car il n'est pas valide
SQLWARN6 'W' si la date résultant d'un calcul : ajout de mois ou d'années, à un jour invalide

SQLSTATE Code retour normalisé :
Premier caractère : '0' si éxécution correcte
Deux premiers caractères : classe du code ('00' si éxécution correcte, '01' si avertissement, …)
Trois derniers caractères : sous classe du code, et le cas échéant, raison précise de l'erreur.

Le programme peut :

- soit tester directement le contenu de la donnée **SQLCODE**
- soit tester le contenu de la donnée **SQLSTATE**
- soit faire appel à un sous-programme généralisé de traitement des erreurs (faisant appel au sous-programme standard DSNTIAR)
- soit demander à DB2 d'effectuer ce contrôle au moyen de la directive **WHENEVER.**

Sous-programme **DSNTIAR**
Il est possible d'utiliser le sous-programme **DSNTIAR** pour obtenir le message associé à un code retour SQL.
Les messages sont stockés dans le module DSNTIAM. Ce module est utilisé par DSNTIAR.

Sous-programme DSNTIAR

Syntaxe d'appel de DSNTIAR

```
CALL 'DSNTIAR' USING sqlca,msg,lrecl
```

SQLCA

Zone de communication SQLCA contenant le code retour SQL à interpréter

Msg

Zone de sortie de longueur variable, où sera placé le message associé au code retour SQL
Les deux premiers caractères contiennent la longueur du message (celle-ci est au minimum de 240 caractères) Le message obtenu est composé de plusieurs lignes de longueur «lrecl» (au maximum 10 lignes)

Lrecl

Longueur de chaque ligne constituant le message
Exprimée sur un mot en binaire, comprise entre 72 et 240

Codes retour de DSNTIAR

0 exécution correcte

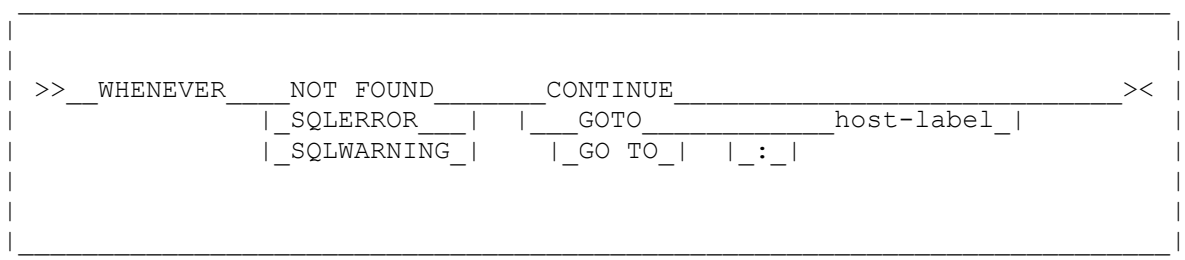
4 plus de données que peut en accueillir la zone réceptrice

8 «lrecl» non comprise entre 72 et 240

12 la zone réceptrice est d'une longueur insuffisante, la longueur du message est supérieure à 240

16 erreur grave

WHENEVER



NOT FOUND correspond à SQLCODE = +100

SQLERROR correspond à SQLCODE < 0

SQLWARNING correspond à SQLCODE > 0 sauf +100
Un ordre **WHENEVER** s'applique à toutes les instructions SQL le suivant, et jusqu'au prochain ordre WHENEVER.
On peut placer à la suite jusqu'à trois ordres WHENEVER, s'appliquant chacun à une condition différente (NOT FOUND, SQLERROR et SQLWARNING).

Exemple :

```
EXEC SQL
     WHENEVER NOT FOUND CONTINUE
END-EXEC
EXEC SQL
     WHENEVER SQLERROR GOTO :PARAG-ERREUR
END-EXEC
```

Résultat :

Exécution de l'instruction suivante si SQLCODE =+100
transfert au paragraphe PARAG-ERREUR si une instruction SQL renvoie un code retour SQLCODE négatif

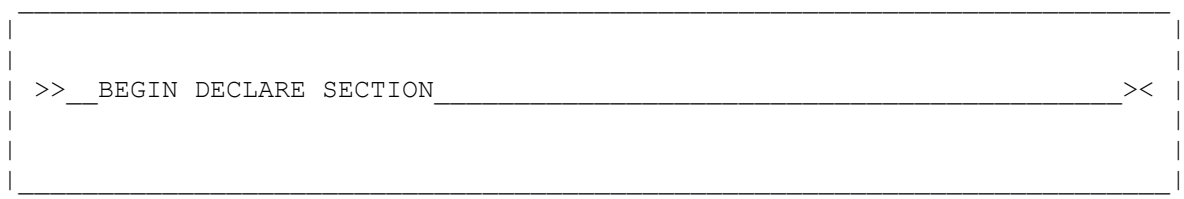
Variable hôte

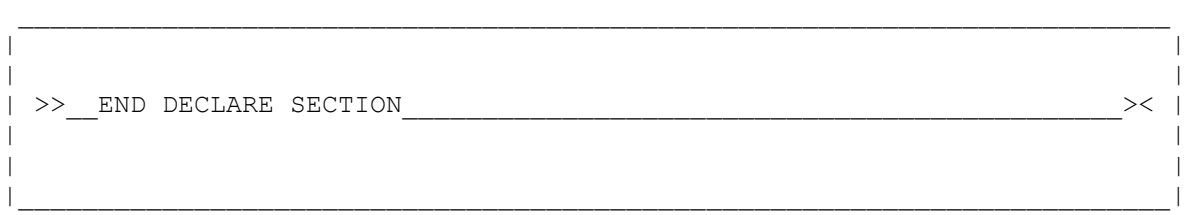
Définition :

On appelle variable hôte une donnée définie dans un programme et utilisée dans une instruction SQL pour recevoir ou transmettre la valeur d'une colonne ou d'une constante.

Format

Il est recommandé de décrire les variables hôtes dans une partie du programme délimitée par les instructions :





Variable hôte

RÈGLES :

- description en niveau 01 ou 77 dans la WORKING-STORAGE SECTION ou dans la LINKAGE SECTION
- Nom obéissant aux règles de formation des noms du langage hôte
- précédée du caractère **':'** dans les instructions SQL
- description compatible avec la définition des données de la colonne qu'elle est destinée à contenir
- ne doit pas contenir les clauses JUSTIFIED, BLANK ZERO ou OCCURS
- une variable hôte destinée à contenir des données de longueur variable (VARCHAR ou VARGRAPHIC) doit être définie sous la forme :

```
01  VARIABLE-HOTE.
    49  LONGUEUR       PIC S9(4) COMP.
    49  VALEUR          PIC X(n).
```

Tableau de correspondance SQLàCOBOL

Cf Annexe (Data-type)

Déclaration SQL	Déclaration COBOL
INTEGER	01 nom PIC S9(9) COMP.
SMALLINT	01 nom PIC s9(4) COMP.
FLOAT(21) ou REAL	01 nom COMP-1.
FLOAT(53) ou double	01 nom COMP-2.
DECIMAL(p, e) ou NUMERIC(p, e)	01 nom PIC s9(p-e) V(e)
CHAR(n)	01 nom PIC x(n).
VARCHAR(n) ou LONG VARCHAR	01 nom1. 49 n1 PIC s9(4) comp. 49 n2 PIC x(n).
DATE	01 nom PIC X(n). n>=10
TIME	01 nom PIC X(n). n>=8
TIMESTAMP	01 nom PIC X(26). N>=19
GRAPHIC(n)	01 nom PIC G(n)
VARGRAPHIC(n) ou LONG VARGRAPHIC	01 nom1. 49 n1 PIC S9(4) comp. 49 n2 PIC G(n).

Variable hôte : Exemple

Création table écrivains

```
CREATE TABLE ECRIVAINS
       (AUTEUR CHARACTER(8) NOT NULL,
        NE_EN SMALLINT NOT NULL,
        LIEU CHARACTER(20) NOT NULL,
        RAYON SMALLINT NOT NULL)
```

Description des variables hôtes correspondantes

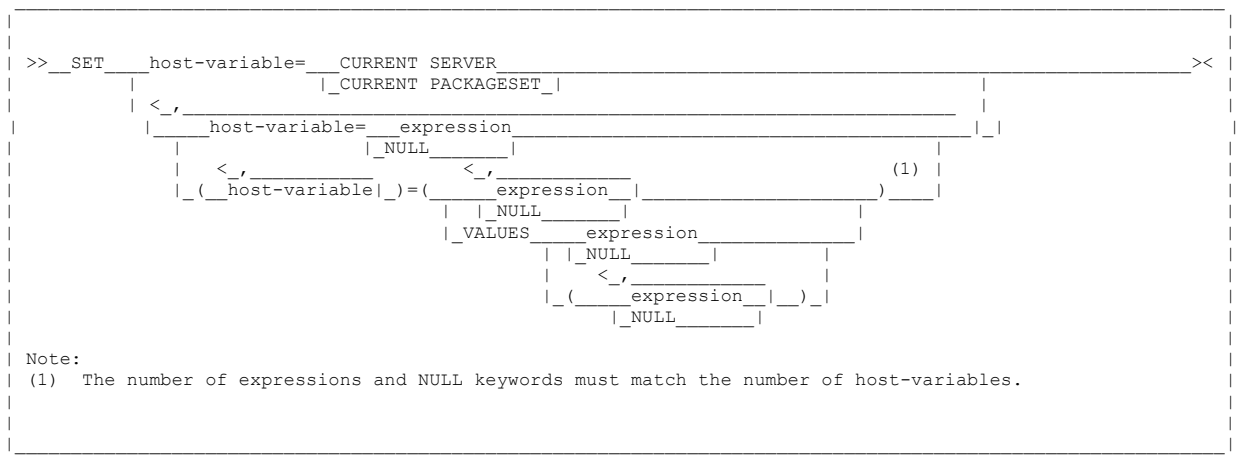
```
01 AUTEUR          PIC X(8).
01 NE-EN           PIC S9(4) COMP.
01 LIEU            PIC X(20).
01 RAYON           PIC S9(4) COMP.
```

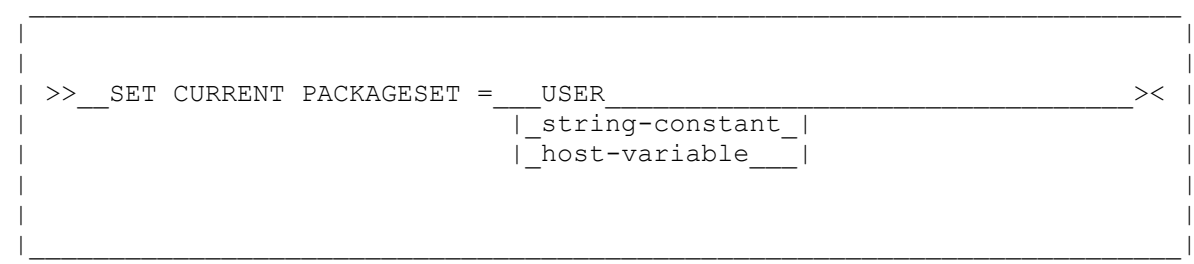
Rechercher les données relatives à l'auteur HUGO.

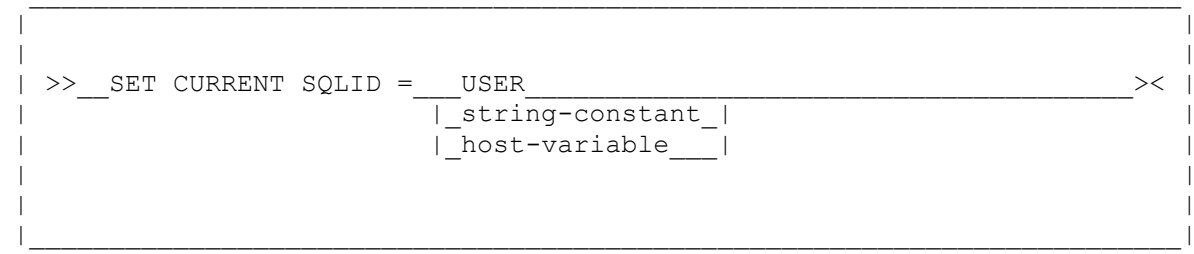
```
MOVE      'HUGO'      TO    AUTEUR

EXEC SQL
      SELECT AUTEUR, NE_EN, LIEU, RAYON
                  INTO :AUTEUR, :NE-EN,
                  :LIEU, :RAYON
            FROM ECRIVAINS
            WHERE AUTEUR = :AUTEUR
END-EXEC
```

Variable hôte et registre spécial

Il est possible d'affecter à une variable hôte le contenu d'un registre spécial au moyen de l'instruction **SET** :







Exemple

Affecter à une variable hôte la date courante.

```
01 DATE-SQL            PIC X(10).
.....
            EXEC SQL
                SET :DATE-SQL = CURRENT DATE
            END-EXEC.
```

LISTE DES REGISTRES SPECIAUX :

- USER
- CURRENT DATE
- CURRENT TIME
- CURRENT TIMESTAMP
- CURRENT TIMEZONE
- CURRENT SQLID
- CURRENT SERVER
- CURRENT DEGREE
- CURRENT PACKAGE SET
- CURRENT RULES

Variable indicateur

Définition Si une colonne de table peut prendre la valeur nulle, il faut associer à la variable hôte correspondante une **variable indicateur** (demi-mot binaire).

Exemple

```
CREATE TABLE RAYONS
        (RAYON SMALLINT,
        SALLE SMALLINT)
```

Création table

RAYONS

Variables hôtes et variables indicateurs correspondantes :

```
01 RAYON      PIC S9(4) COMP.
01 IND-RAYON  PIC S9(4) COMP.
01 SALLE      PIC S9(4) COMP.
01 IND-SALLE  PIC S9(4) COMP.
```

Variable hôte — Variable indicateur — Variable hôte — Variable indicateur

Rechercher les données relatives au rayon 3.

```
EXEC SQL
 SELECT RAYON, SALLE
    INTO  :RAYON:IND-RAYON,
          :SALLE:IND-SALLE
    FROM  RAYONS
   WHERE RAYON  = 3
END-EXEC
```

Variable indicateur

Contenu d'une variable indicateur :

- valeur transmise par DB2 au programme :

- 0 : le contenu de la colonne n'a pas la valeur NULL
  contenu placé dans la variable hôte

- -1 : la colonne contient la valeur NULL
  contenu de la variable hôte inchangé

- -2 : erreur de conversion ou d'expression numérique
  contenu de la variable hôte inchangé

- valeur >0 : la donnée transmise est une chaîne de
  caractères et a été tronquée

la variable indicateur contient la longueur d'origine de la chaîne

- valeur <0 placée par le programme avant mise à jour :
  la donnée correspondante doit être mise à NULL dans la table

Erreur si DB2 renvoie une valeur NULL dans une variable hôte sans variable indicateur associée

STRUCTURE hôte

Définition Une structure hôte est un groupe de données contenant plusieurs variables hôtes définies à n'importe quel niveau entre 01 et 48, mais ne doit comporter que deux niveaux

Seule exception : données de longueur variable qui doivent être déclarées en niveau **49**

Exemple

```
CREATE TABLE EXEMPLE
 (VAR1   CHAR (10)     NOT NULL,
  VAR2   SMALLINT      NOT NULL,
  VAR3   VARCHAR (10)  NOT NULL
```

Création table exemple

```
01 LIG-EX.
   05 VAR1     PIC  X(10).
   05 VAR2     PIC S9(4) COMP.
   05 VAR3.
      49 VAR3L  PIC S9(4) COMP.
      49 VAR3D  PIC  X(10).
```

Structure hôte

Structure hôte

Format On référence la structure hôte soit de manière globale par son nom de groupe, soit en qualifiant chaque variable hôte :

Exemple

```
EXEC SQL
    SELECT VAR1, VAR2, VAR3
    INTO :LIG-EX.VAR1,
         :LIG-EX.VAR2,
         :LIG-EX.VAR3
    FROM  EXEMPLE
    WHERE ............
END-EXEC
MOVE VAR1 OF LIG-EX TO
```

Dans le programme COBOL

```
EXEC SQL
    SELECT VAR1, VAR2, VAR3
       INTO  :LIG-EX
       FROM  EXEMPLE
       WHERE ..........
END-EXEC
```

Les deux ordres SQL sont équivalents

Tableau d'indicateurs

Définition Tableau d'indicateurs associé à une structure hôte. Autant de postes que de données dans la structure hôte. (correspondance positionnelle)

```
CREATE TABLE RAYONS
          (RAYON SMALLINT,
           SALLE SMALLINT)
```

Structure hôte et tableau d'indicateurs

```
01 LIG-RAY.
    05 RAYON                    PIC S9(4) COMP.
    05 SALLE                    PIC S9(4) COMP.
01 IND-RAY.
    05 IND     OCCURS 2         PIC S9(4) COMP.

EXEC SQL
          SELECT RAYON, SALLE
          INTO :LIG-RAY:IND
          FROM RAYONS
   WHERE RAYON = 3
END-EXEC
```

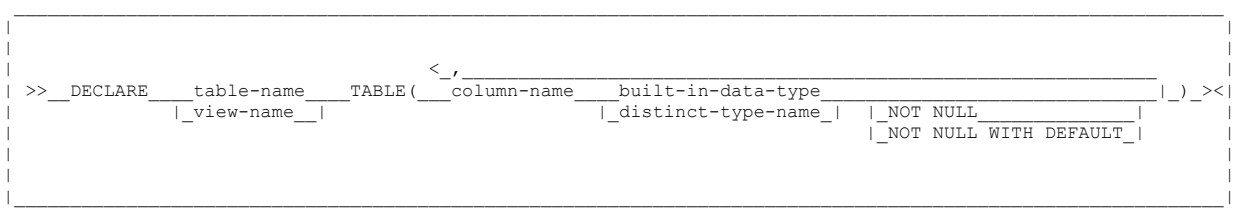
IF IND(2) LESS ZERO ……..

La SALLE du RAYON 3 a-t-elle la valeur NULL ?

DECLARE TABLE

Définition La directive **DECLARE TABLE** est prévue à des fins de documentation et de contrôle.
Elle permet au précompilateur de vérifier la cohérence entre les ordres SQL et les tables auxquelles ils se rapportent.

Format



Directive DECLARE TABLE pour la table ECRIVAINS

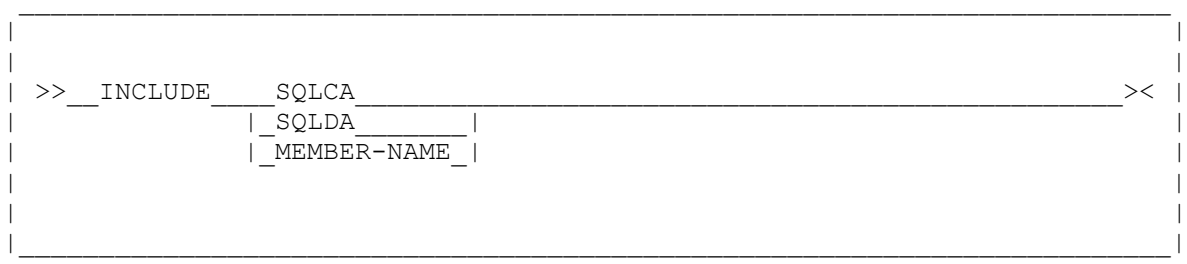
Permet au précompilateur de faire le lien entre les colonnes de la table et celles référencées dans les ordres SQL du programme.

```
EXEC SQL
      DECLARE ECRIVAINS TABLE
            (AUTEUR CHARACTER(8) NOT NULL,
             NE_EN SMALLINT NOT NULL,
             LIEU CHARACTER(20) NOT NULL,
             RAYON SMALLINT NOT NULL)
END-EXEC
```

INCLUDE

Définition La directive **INCLUDE** permet d'utiliser la description des données générée par DCLGEN.

Format



Nom de membre : membre produit par DCLGEN

Permet de placer, dans le programme, l'ordre DECLARE TABLE correspondant à la table considérée et la structure hôte générée

SQLCODE aura la valeur :

- 0 : si une seule ligne résultat
- +100 : si aucune ligne résultat
- -811 : si plus d'une ligne résultat

Exemple

Calculer la somme des prix des ouvrages et le nombre distinct d'auteurs figurant dans la table LIVRES.

```
WORKING-STORAGE SECTION.
01  NOMBRE     PIC S9(9) COMP.
01  SOMME       PIC S9(4)V9(2) COMP-3.

PROCEDURE DIVISION.
...
        EXEC SQL
                SELECT SUM(PRIX),
                        COUNT(DISTINCT AUTEUR)
                        INTO :SOMME, :NOMBRE
                FROM LIVRES
        END-EXEC
```

Exemple

Placer dans la table ECRIVAINS des données figurant dans une structure hôte.

```
WORKING-STORAGE SECTION.
01 ECRIVAIN.
05 AUTEUR    PIC X(8)          VALUE 'GAUTIER'.
05 NE-EN     PIC S9(4) COMP    VALUE 1811.
05 LIEU      PIC X(20)         VALUE 'TARBES'.
05 RAYON     PIC S9(4) COMP    VALUE 3.

PROCEDURE DIVISION.
...
   EXEC SQL
        INSERT INTO ECRIVAINS
            (AUTEUR, NE_EN, LIEU, RAYON)
           VALUES (:ECRIVAIN)
   END-EXEC
```

Il est également possible d'écrire une instruction INSERT comportant un sous-select. (nombre de lignes insérées dans SQLERRD (3) de SQLCA)

Règles

Les instructions **UPDATE** ou **DELETE** peuvent utiliser les variables hôtes :

Dans une condition de recherche WHERE

Dans des expressions SET

(nombre de lignes mises à jour ou annulées dans SQLERRD(3) de SQLCA)

Exemple

Augmenter de 10% (quand on le connaît) le prix des ouvrages de la table LIVRES.

```
WORKING-STORAGE SECTION.
01  TAUX     PIC S9(4)V9(2) COMP-3      VALUE 1.1.

PROCEDURE DIVISION.
...
      EXEC SQL
              UPDATE LIVRES
              SET PRIX = PRIX * :TAUX
         WHERE PRIX IS NOT NULL
      END-EXEC
```

Notion de curseur

Concept

Un curseur est nécessaire pour exploiter le contenu d'une table résultante comportant plusieurs lignes.

Curseur :

- nom symbolique, associé à une opération effectuée sur une table.
- Matérialise la position courante sur une table résultante
- Permet de traiter une table ligne à ligne
- Structure du programme : répétitive

DECLARE : définition de l'opération de sélection constituant la table résultante

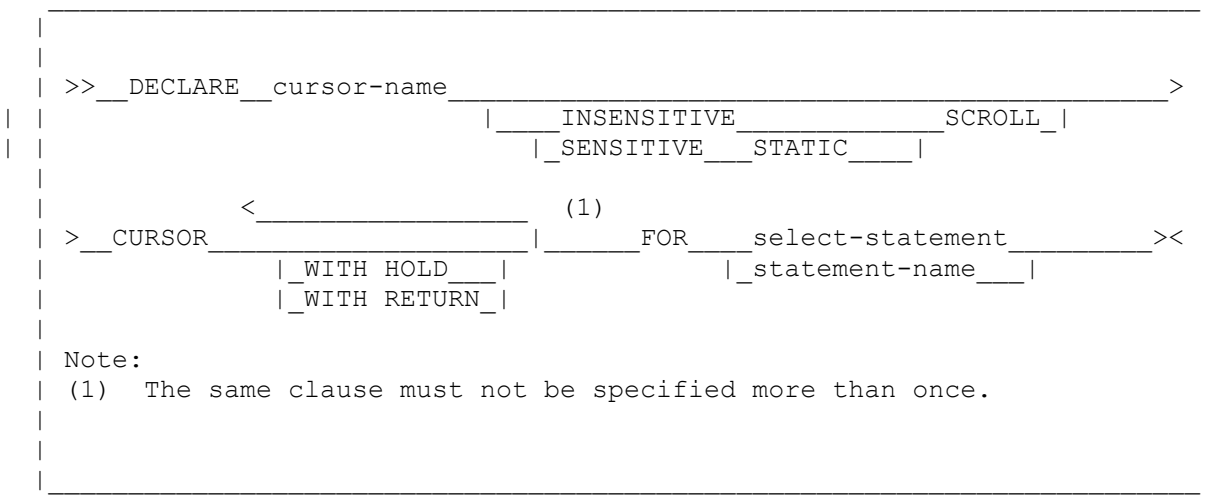
OPEN : début de traitement

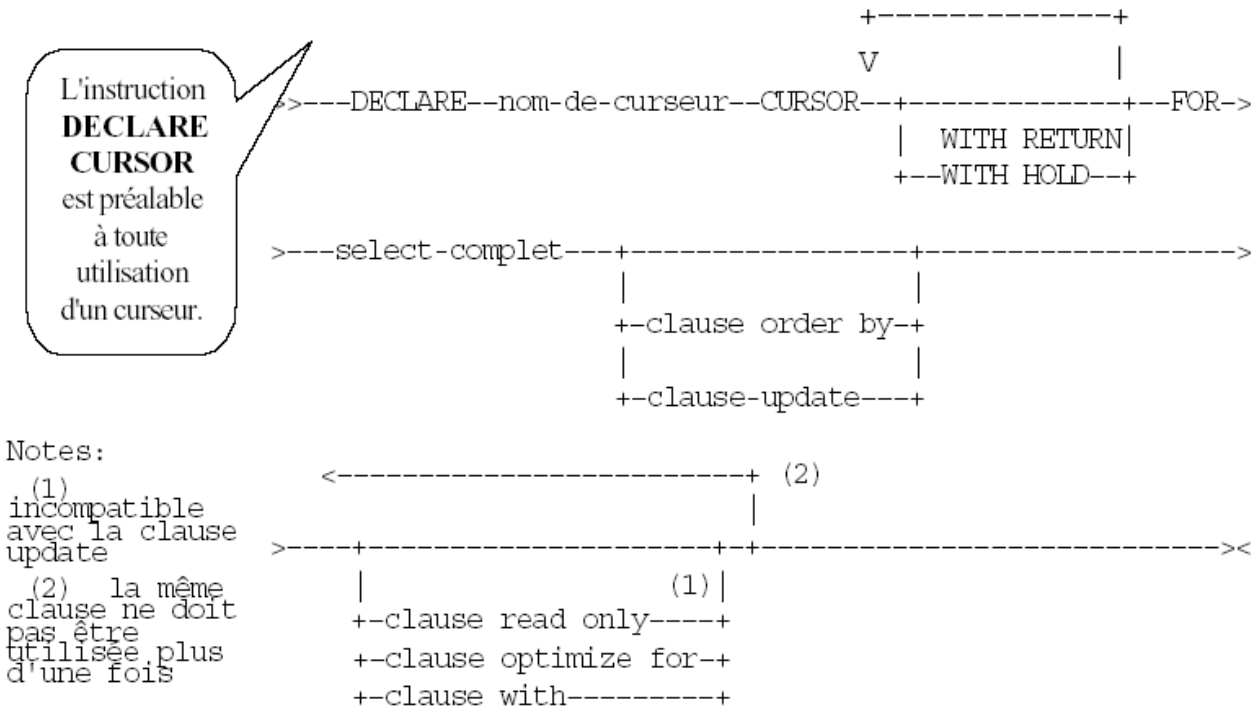
FETCH : recherche d'une nouvelle ligne

CLOSE : fin du traitement après détection de la fin de la table résultante

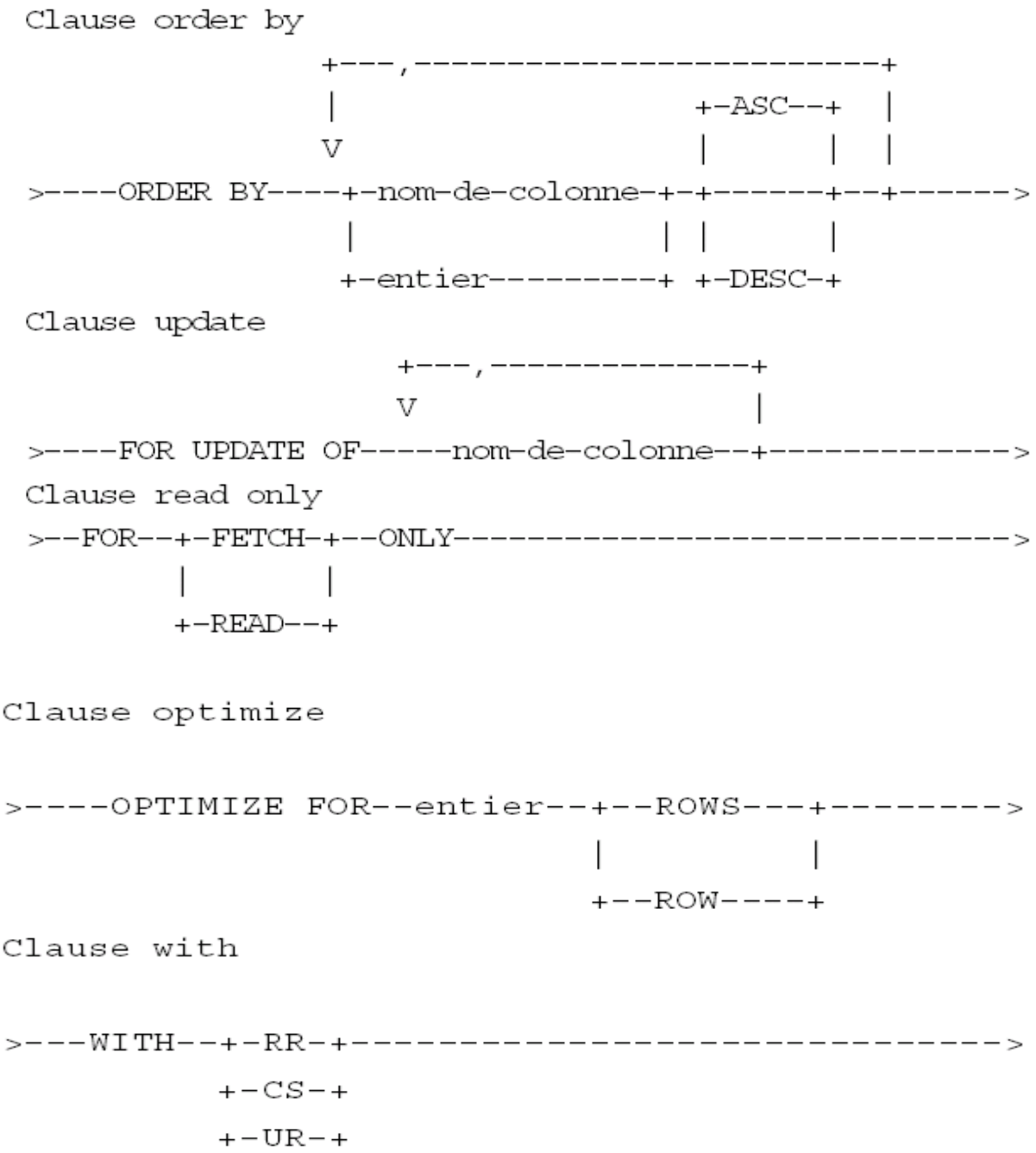
Possibilité d'UPDATE ou de DELETE de la ligne courante

DECLARE CURSOR





DECLARE CURSOR



La clause « optimize » est intéressante en TP où sur un écran vous ne pouvez afficher qu'un certain nombre de lignes.
La clause « With » passe outre le paramètre ISOLATION du BIND (pour l'instruction à laquelle elle se rapporte uniquement).

DECLARE CURSOR

Règles

WITH HOLD : pour conserver le positionnement au moment de la prise d'un point de cohérence

WITH RETURN : Spécifie que le curseur, s'il est utilisé dans une procédure stockée, peut renvoyer un ensemble résultat à l'appellant.

FOR FETCH ONLY : pour déclarer la table en lecture seule

FOR UPDATE : pour indiquer la ou les colonnes que l'on a l'intention de mettre à jour par l'instruction UPDATE option interdite si table en lecture seule

OPTIMIZE FOR : pour améliorer les performances si peu de lignes sont exploitées (ne limite pas le nombre de lignes obtenues)

WITH UR (CS, RR) : précise le niveau d'isolation pour l'instruction.

Une table est en lecture seule quand il y a un :SELECT avec une fonction sur colonne ou les options GROUP BY, DISTINCT, UNION, ORDER BY ou plus d'un nom de table après le mot FROM

DECLARE CURSOR

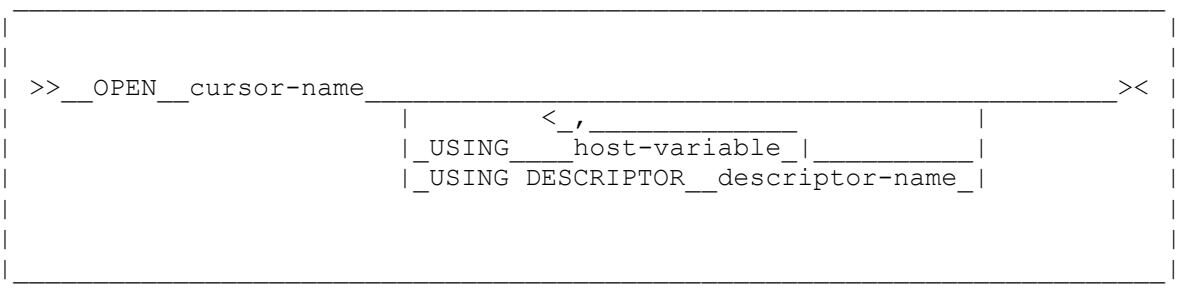
Exemple

Définir un curseur permettant de recevoir, dans un programme, les données AUTEUR et TITRE de la table LIVRES, et **autorisant la mise à jour de la colonne PRIX.**

```
EXEC SQL
        DECLARE CURSEUR_1 CURSOR FOR
                SELECT AUTEUR, TITRE
                FROM LIVRES
                FOR UPDATE OF PRIX
END-EXEC
```

OPEN CURSEUR

Format



Règles

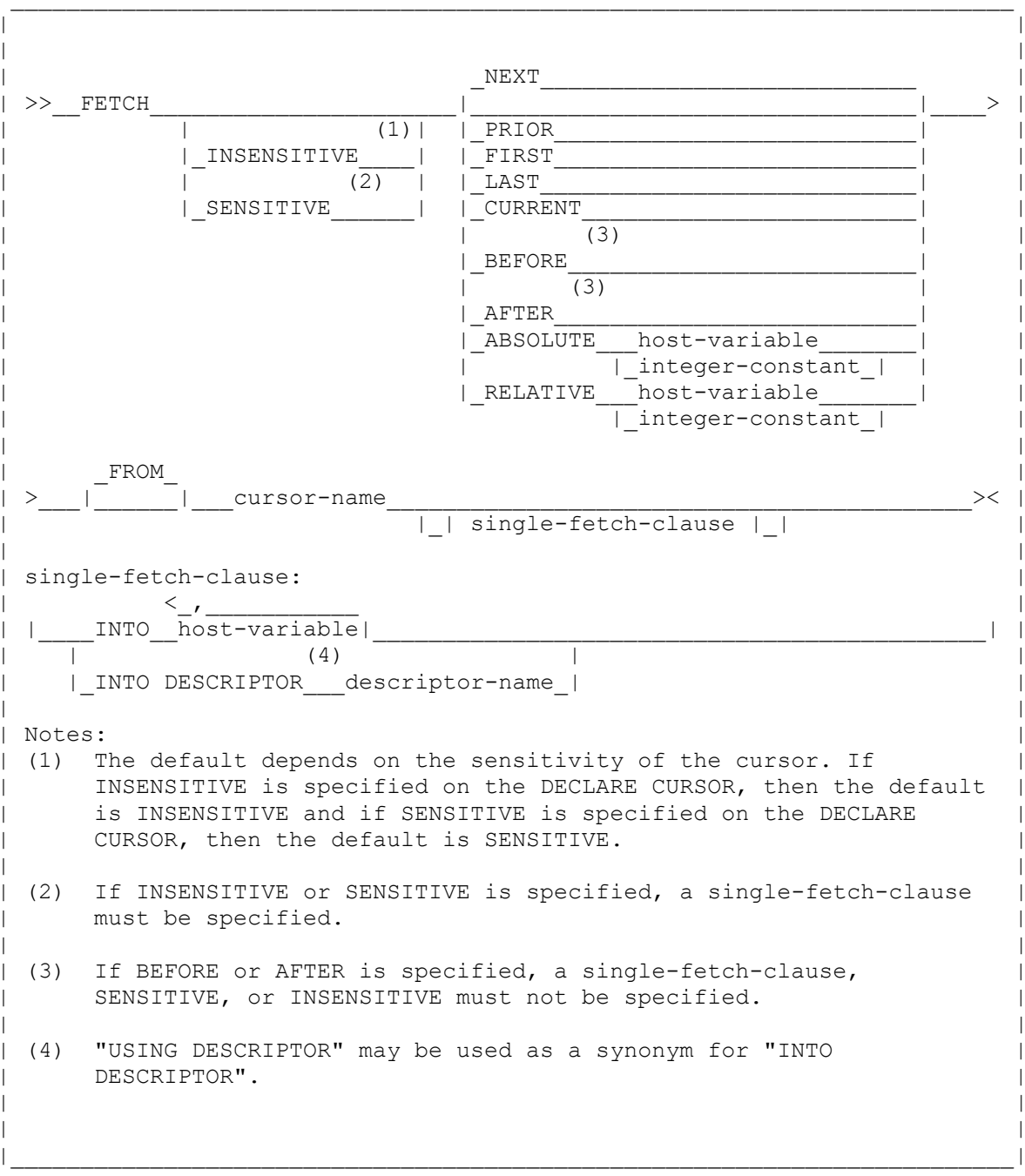
- Après l'OPEN,le curseur est placé devant la première ligne de la table résultante (ou à la fin de la table résultante si elle est vide).
- Si des variables hôtes figurent dans le DECLARE CURSOR, leur valeur est exploitée à l'OPEN pour construire la table résultante.

Exemple

Ouvrir le curseur déclaré dans l'exemple précédent.

```
EXEC SQL
        OPEN CURSEUR_1
END-EXEC
```

Format


>>__FETCH
(1)
INSENSITIVE
(2)
SENSITIVE
NEXT
PRIOR
FIRST
LAST
CURRENT
(3)
BEFORE
(3)
AFTER
ABSOLUTE
host-variable
integer-constant
RELATIVE
host-variable
integer-constant
FROM
cursor-name
single-fetch-clause
single-fetch-clause:
INTO host-variable
(4)
INTO DESCRIPTOR
descriptor-name
Notes:
(1) The default depends on the sensitivity of the cursor. If INSENSITIVE is specified on the DECLARE CURSOR, then the default is INSENSITIVE and if SENSITIVE is specified on the DECLARE CURSOR, then the default is SENSITIVE.
(2) If INSENSITIVE or SENSITIVE is specified, a single-fetch-clause must be specified.
(3) If BEFORE or AFTER is specified, a single-fetch-clause, SENSITIVE, or INSENSITIVE must not be specified.
(4) "USING DESCRIPTOR" may be used as a synonym for "INTO DESCRIPTOR".

FETCH

Règle

L'instruction **FETCH** permet de recevoir, dans la ou les variables hôtes qui y figurent, la ligne courante d'une table résultante associée à un curseur.

- le curseur est déplacé sur la ligne suivante
- SQLCODE = +100 après la dernière ligne

Exemple

Transmettre au programme la ligne de table suivante.

```
EXEC SQL
      FETCH CURSEUR_1
      INTO :AUTEUR, :TITRE
END-EXEC

IF SQLCODE ...
```

Format


searched update:
>>__UPDATE____table-name_____>
|_view-name__|
|_correlation-name_|
>__SET__assignment-clause_____>
|_WHERE__search-condition_|
>_____><
|_WITH____RR___|
|_QUERYNO__integer_|
|_RS_|
|_CS_|
positioned update:
>>__UPDATE____table-name_____>
|_view-name__|
|_correlation-name_|
>__SET__assignment-clause__WHERE CURRENT OF__cursor-name_____><
assignment clause:
<_,_____
>>_____column-name=___expression_____|__><
|_NULL_____|
|_(scalar-fullselect)_|
<_,_____
|_(___column-name_|_)__=__(_____expression___|_____)_|
(1)
|_NULL_____|
(2)
|_row-fullselect_____|
Notes:
(1) The number of expressions and NULL keywords must match the number of column-names.
(2) The number of columns in the select list must match the number of column-names.

UPDATE positionnel

Règles

Quand un curseur a été défini pour une table, l'instruction **UPDATE** permet de mettre à jour la ligne courante de la table.
noms de colonnes = noms de colonnes indiqués dans l'option FOR UPDATE du DECLARE CURSOR.

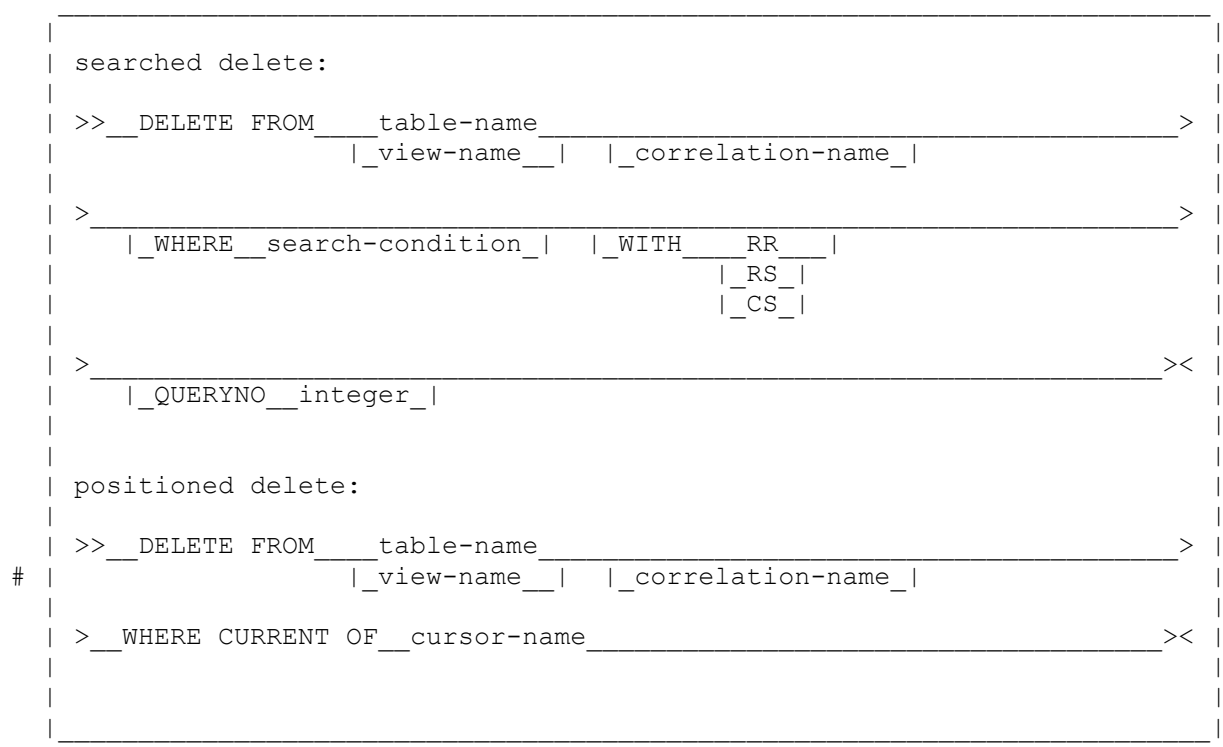
UPDATE met à jour les colonnes de la **ligne courante**.
Le curseur reste positionné sur la ligne courante.

Exemple Augmenter les prix de 10%.

```
EXEC SQL
     UPDATE LIVRES
     SET PRIX = PRIX * 1.1
     WHERE CURRENT OF CURSEUR_1
END-EXEC
```

DELETE positionnel

Format



Règles

- L'instruction **DELETE** permet de supprimer **la ligne courante**.
- La table ne doit pas être en lecture seule.
- Le curseur est positionné devant la prochaine ligne (après la dernière ligne si on vient de supprimer la dernière ligne d'une table).

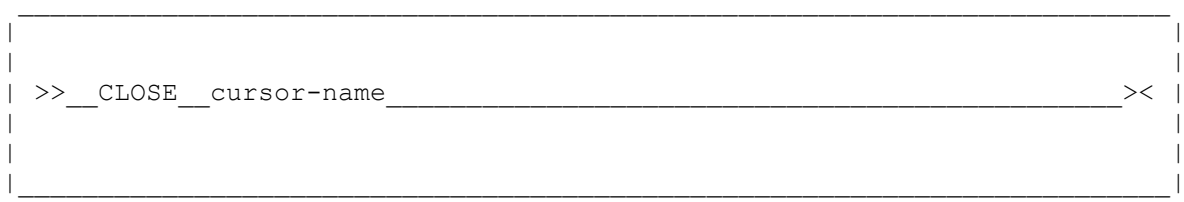
DELETE positionnel

Exemple Annuler le dernier ouvrage transmis au programme.

```
EXEC SQL
     DELETE FROM LIVRES
     WHERE CURRENT OF CURSEUR_1
END-EXEC
```

CLOSE

Format



Règles

L'instruction **CLOSE** permet de fermer un curseur.

Après l'instruction CLOSE :

- curseur fermé
- table résultante inaccessible

Fermetures indirectes de curseur :

- fin du programme
- prise d'un point de cohérence (sauf si with hold)

Exemple Fermer le curseur CURSEUR_1.

```
EXEC SQL
   CLOSE CURSEUR_1
END-EXEC
```

Unité de travail

Pour éviter des incohérences dues à des accès concurrents, le système relationnel opère un blocage des données basé sur le principe suivant :

Aucun programme ne doit pouvoir accéder à des données modifiées par un autre programme tant qu'elles n'ont pas été entérinées.

Une **unité de travail** est l'intervalle situé entre 2 points de cohérence :

- point de cohérence **implicite** (début et fin de programme)
- point de cohérence **explicite** (COMMIT)

PRISE EN COMPTE DES MISES À JOUR

Fin normale :

- une unité de travail est terminée sans erreur
- les mises à jour sont entérinées
- les blocages sur les données sont levés
- on atteint un point de cohérence **COMMIT POINT**

Fin anormale :

- une erreur a été détectée alors que des mises à jour avaient été entreprises
- les mises à jour sont effacées
- les verrous sur les données sont levés
- on effectue un retour arrière **ROLLBACK** au point de cohérence précédent

Prise de points de cohérence

Les points de cohérence peuvent être implicites ou explicites, selon l'environnement d'exécution des programmes.

Unité de travail	TSO	IMS	CICS
Début unité de travail	1er ordre de mise à jour	Lancement du programme	Début de transaction
Fin implicte avec confirmation des mises à jour	Fin normale du programme	- Fin normale du programme - GU IOPCB	- ordre return call term call chkp
Fin implicite avec annulation des mises à jour	Fin anormale du programme	Fin anormale du programme	Fin anormale du programme
Fin explicte avec confirmation des mises à jour	Ordre SQL COMMIT	Call CHKP Call SYNC	Ordre CICS SYNCPOINT
Fin explicite avec annulation des mises à jour	Ordre SQL ROLLBACK	- CALL ROLL - CALL ROLB	Ordre CICS SYNCPOINT

SOUS CICS

Une unité de mise à jour débute avec l'exécution d'une transaction et se termine avec un ordre CICS **RETURN** avec prise d'un point de contrôle par un ordre CICS **SYNCPOINT** (si il comporte l'option ROLLBACK retour au dernier point de cohérence)

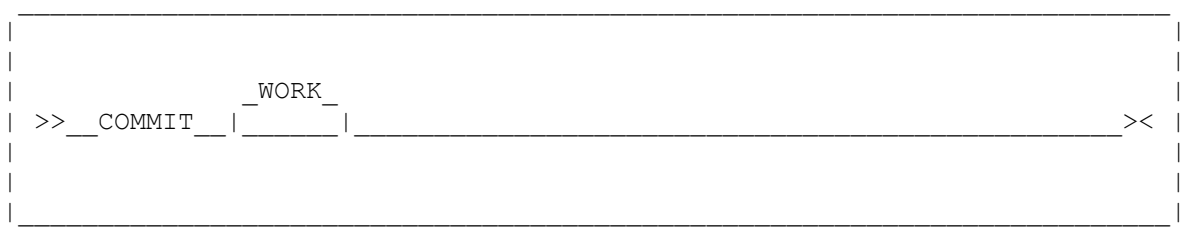
EFFETS de la prise d'un point de cohérence

- Tous les curseurs ouverts sans l'option WITH HOLD sont fermés
- Les tables résultantes éventuellement constituées sont perdues
- Toutes les instructions transmises à SQL sous forme dynamique par l'instruction PREPARE sans l'option WITH HOLD sont perdues
- Il est important de prévoir la prise de points de cohérence dès la conception, et d'adopter la technique de repositionnement la plus performante possible.

COMMIT ET ROLLBACK

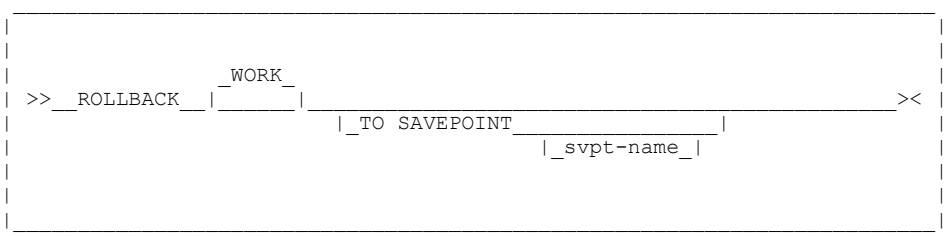
COMMIT

L'instruction **COMMIT** permet la prise explicite d'un point de cohérence.



ROLLBACK

L'instruction **ROLLBACK** provoque un retour au point de cohérence précédent, et efface les mises à jour éventuellement entreprises.



Extraction de compilation cobol

```
 000012                *****EXEC SQL
 000013                *****     INCLUDE SQLCA
 000014                *****END-EXEC.
 000015                  01 SQLCA.
 000016                     05 SQLCAID     PIC X(8).
 000017                     05 SQLCABC     PIC S9(9) COMP-4.
 000018                     05 SQLCODE     PIC S9(9) COMP-4.
 000019                     05 SQLERRM.
 000020                        49 SQLERRML PIC S9(4) COMP-4.
 000021                        49 SQLERRMC PIC X(70).
 000022                     05 SQLERRP     PIC X(8).
 000023                     05 SQLERRD     OCCURS 6 TIMES
 000024                                    PIC S9(9) COMP-4.
 000025                     05 SQLWARN.
 000026                        10 SQLWARN0 PIC X.
 000027                        10 SQLWARN1 PIC X.
 000028                        10 SQLWARN2 PIC X.
 ……..
 ……..
 000276                *****EXEC SQL
 000277                *****     OPEN CURSOR_1
 000278                *****END-EXEC.
 000279                     PERFORM SQL-INITIAL UNTIL SQL-INIT-DONE
 000280                     CALL 'DSNHLI' USING SQL-PLIST5.
 000281                     PERFORM UNTIL SQLCODE = +100
 000282                *****   EXEC SQL
 000283                *****      FETCH CURSOR_1
 000284                *****        INTO :NOM, :SAL, :COM
P 5648-A25 IBM COBOL for OS/390 & VM  2.2.2                    CAIP5B1   Dat
 LineID  PL SL  ----+-*A-1-B--+----2----+----3----+----4----+----5----+----
 000285                *****   END-EXEC
 000286      1              PERFORM SQL-INITIAL UNTIL SQL-INIT-DONE
 000287      1              CALL 'DSNHLI' USING SQL-PLIST6
 000288      1                 IF SQLCODE = +100
 000289                *****      EXEC SQL
 000290                *****           CLOSE CURSOR_1
```

Résultat du BIND

```
READY
  DSN SYSTEM(DSN8)
DSN
  BIND PACKAGE (COLHKG1)          MEMBER(CAIP5B1) OWNER(HKG1) ACT(REP)
      EXPLAIN(NO) VALIDATE (BIND) ISO (CS) CURRENTDATA(NO)
      RELEASE(COMMIT) QUAL(HKG1)
DSNT254I  -DSN8 DSNTBCM2 BIND OPTIONS FOR
           PACKAGE = DSN8.COLHKG1.CAIP5B1.()
           ACTION         ADD
           OWNER          HKG1
           QUALIFIER      HKG1
           VALIDATE       BIND
           EXPLAIN        NO
           ISOLATION      CS
           RELEASE        COMMIT
           COPY
DSNT255I  -DSN8 DSNTBCM2 BIND OPTIONS FOR
           PACKAGE = DSN8.COLHKG1.CAIP5B1.()
```

JCL d'éxecution

```
000001 //UT0M1KA   JOB (UT0M1K),'ALBERT',MSGLEVEL=(1,1),MSGCLASS=A,CLASS=X,
000002 //          NOTIFY=&SYSUID,TIME=(,1)
000003 //S01       EXEC PGM=IKJEFT01,REGION=8M
000004 //SYSREC01 DD DSN=UT0M1K.WNGT.WADOUT.SEQDATA7,
000005 //             DISP=(OLD,CATLG,DELETE),SPACE=(TRK,(80,20),RLSE),
000006 //             DCB=(RECFM=VB,LRECL=32708,BLKSIZE=32716)
000007 //SYSTSPRT DD SYSOUT=*
000008 //SYSPRINT DD SYSOUT=*
000009 //SYSOUT   DD SYSOUT=*
000010 //SYSTSIN  DD *
000011   DSN SYSTEM(DSN8)
000012   RUN PROGRAM(TSTDYN01) PLAN(STK1BAT) LIB('STK7.WIN1.STD.LOADLIB')
000013   END
000014 /*
000015 //SYSIN DD *
000016 SYSIBM.SYSTABLES
000017 /*
```

Résultat de compilation

```
NP    DDNAME   StepName ProcStep DSID Owner    C Dest              Rec-Cnt
S     JESMSGLG JES2                2 UT0M1K   X LOCAL                  27
      JESJCL   JES2                3 UT0M1K   X LOCAL                 168
      JESYSMSG JES2                4 UT0M1K   X LOCAL                 225
      SYSPRINT IOSTDBRM          105 UT0M1K   X LOCAL                 157
      SYSTERM  IOSTDBRM          106 UT0M1K   X LOCAL                   8
      SYSPRINT IOSTCOB2          107 UT0M1K   X LOCAL                 965
      SYSPRINT IOSTLINK          108 UT0M1K   X LOCAL                 269
      SYSTSPRT IOSTBIND          110 UT0M1K   X LOCAL                  66
```

```
15.25.50 JOB00207  IRR010I  USERID UT0M1K   IS ASSIGNED TO THIS JOB.
15.25.50 JOB00207  ICH70001I UT0M1K   LAST ACCESS AT 15:14:35 ON MONDAY,
15.25.50 JOB00207  $HASP373 UT0M1K1  STARTED - INIT 1    - CLASS F - SYS
15.25.50 JOB00207  IEF403I UT0M1K1 - STARTED - TIME=15.25.50
15.25.50 JOB00207  -                                          --TIMINGS
15.25.50 JOB00207  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU    SRB
15.25.50 JOB00207  -UT0M1K1           IOSTCOPY    00     52    .00    .00
15.25.50 JOB00207  -UT0M1K1           IOSTDBRM    00     95    .00    .00
15.25.52 JOB00207  -UT0M1K1           IOSTCOB2    00    393    .00    .00
15.25.52 JOB00207  -UT0M1K1           IOSTLINK    00    249    .00    .00
15.25.54 JOB00207  -UT0M1K1           IOSTBIND    00     54    .00    .00
15.25.54 JOB00207  IEF404I UT0M1K1 - ENDED - TIME=15.25.54
15.25.54 JOB00207  -UT0M1K1  ENDED.  NAME-CAIP5B1-L001-O/Y     TOTAL CPU
```