- SELECT A UNE SEULE REPONSE
- SELECT A n REPONSE(S)
- INSTRUCTIONS DE MISE A JOUR

SELECT A UNE REPONSE

- Voici un exemple de programme DB2 COBOL qui utilise une requête SELECT susceptible de renvoyer une seule ligne.
- L'instruction SQL sélectionne un enregistrement unique d'une table en fonction d'une condition précise (par exemple, l'ID d'une pièce).
- Ce type de requête est typiquement utilisé dans des scénarios où vous savez que la condition WHERE est suffisamment restrictive pour retourner un seul résultat.

```
EXEC SQL
        SELECT PNO, PNAME, COLOR, WEIGHT
                INTO    :PARTS-PNO, :PARTS-PNAME ,
                        :PARTS-COLOR , :PARTS-WEIGHT
        FROM PARTS
        WHERE  PNO= 'P1'
END-EXEC
```

- Notons l'apparition pour la première fois de l'instruction INTO qui est propre à un SELECT imbriqué dans un programme. Ici, on aura au maximum une et une seule réponse correspondant à ce critère.
- **Il ne faudra après l'exécution de la requête de vérifier le SQLCODE.**

SELECT A n REPONSE(S)

- Il est rare qu'un ordre SELECT donne exactement une ligne de la table comme dans l'exemple précédent. Très souvent, il donne en réponse un groupe de lignes.
- Lorsque vous devez traiter plusieurs lignes à partir d'une requête SQL, vous ne pouvez pas utiliser un simple SELECT INTO (qui est limité à une seule ligne). Dans ce cas, les **curseurs** permettent de parcourir ligne par ligne les résultats retournés par la requête.
- Un curseur sert à pointer sur ce groupe de lignes de manière à obtenir une ligne après une ligne.
- On utilisera la même philosophie que pour un fichier séquentiel.
- Quatre étapes essentielles :
  - DECLARATION DU CURSEUR
  - OUVERTURE DU CURSEUR
  - PAS A PAS SUR LE CURSEUR
  - FERMETURE DU CURSEUR

DECLARATION DU CURSEUR

- La déclaration du curseur peut être faite soit en WORKING-STORAGE SECTION, soit en PROCEDURE DIVISION.
- Elle doit être faite avant l'usage du curseur.
- La requête n'est pas exécutée à la déclaration du curseur.
- Par conséquent, la valeur de la HOST VARIABLE n'a pas besoin d'être alimentée au moment de la déclaration.
- L'exécution du curseur est différée.

DECLARATION DU CURSEUR

```
EXEC SQL
        DECLARE <NOM DU CURSEUR>  CURSOR
        FOR
        SELECT CHAMP1, CHAMP2
                FROM TABLE
WHERE TABLE.CHAMP3 = VALEUR
END-EXEC
```

```
EXEC SQL
        DECLARE CPARTS CURSOR
        FOR
        SELECT PNO, PNAME, WEIGHT
        FROM PARTS
        WHERE  COLOR = :PARTS-COLOR
END-EXEC
```

- Il n'y a pas de trait d'union dans les noms de curseurs. Etant donné qu'un curseur n'est exécuté qu'au moment de son ouverture, la valeur n'a pas besoin d'être connue au moment de sa déclaration.

OUVERTURE DU CURSEUR

- La requête s'exécute à l'ouverture du curseur. Ce qui implique que la valeur de PARTS-COLOR doit être définie.

```
MOVE 'BLUE' TO PARTS-COLOR

EXEC SQL
        OPEN CPARTS
END-EXEC
```

- L'effet de OPEN est de déclencher l'exécution de l'ordre SELECT et de positionner le curseur au début du tableau résultat.
- Au moment de l'OPEN, DB2 crée une table mémoire qui contiendra les résultats.
- Mais attention, si la table résultante est vide, ce n'est pas à l'OPEN que DB nous l'indiquera mais au premier FETCH.
- Quoiqu'il en soit, on vérifiera la valeur du SQLCODE pour s'assurer de la cohérence de la requête définie lors de sa déclaration.

III. PAS A PAS SUR LE CURSEUR

- Après avoir ouvert le curseur, il faut faire avancer le curseur de manière à lire ligne par ligne comme un simple fichier séquentiel.

- Le curseur :
  - ne peut avancer que d'un pas à la fois
  - ne peut faire de retour arrière

```
EXEC SQL
        FETCH <NOM DU CURSEUR>
        INTO :HOSTVAR1, :HOSTVAR2
END-EXEC

EXEC SQL
        FETCH CPARTS
        INTO    :PARTS-PNO,
                :PARTS-PNAME ,
                :PARTS-WEIGHT
END-EXEC
```

- On vérifiera également le SQLCODE !

- Il s'agit uniquement d'une seule lecture. On mettra ensuite en place une boucle permettant d'itérer sur les différents résultats renvoyés par la requête.

FERMETURE DU CURSEUR

- Il est impératif de fermer le curseur après utilisation.

```
EXEC SQL
        CLOSE CPARTS
END-EXEC
```

- De plus, si on devait changer la valeur de la HOST VARIABLE, c'est à dire avec un WHERE différent. Pour que DB2 puisse prendre cette nouvelle valeur en compte, nous devons fermer et réouvrir le curseur.

Mais comment s'assurer que le jeu d'enregistrements est à sa fin ?

- Pour ce faire, on distingue le SQLCODE. Zone déclarée lors de l'inclusion de la zone SQLCA. Il s'agit d'une zone numérique signée packée.

```
01 SQLCA.
        05 …
        05 SQLCODE PIC S9(9) COMP.
        05 …
```

- Indicateurs de fin d'itération sur un curseur :
  - SQLCODE = 0 :
    - L'instruction FETCH a récupéré une ligne avec succès. Vous pouvez continuer à traiter les données de cette ligne et effectuer une nouvelle itération pour récupérer la suivante.
  - SQLCODE = 100 :
    - Ce code indique la fin de l'ensemble de résultats, c'est-à-dire qu'il n'y a plus de lignes à lire dans le curseur. Lorsque vous recevez ce code, vous avez terminé d'itérer sur le curseur.
  - SQLCODE > 0 et différent de 100 :
    - Il s'agit d'un avertissement. Cela ne nécessite pas forcément un plantage du système
  - SQLCODE < 0
    - Si SQLCODE est inférieur à 0, cela signifie qu'une erreur SQL s'est produite. Dans la plupart des cas, il s'agit d'une anomalie nécessitant un ABEND du système.

```
WORKING-STORAGE SECTION.
        EXEC SQL
                INCLUDE SQLCA
        END-EXEC

        EXEC SQL
                INCLUDE PARTS
        END-EXEC
        EXEC SQL
                DECLARE CPARTS CURSOR
                FOR
                SELECT PNO, PNAME, WEIGHT
                FROM PARTS
                WHERE  COLOR = :PARTS-COLOR
        END-EXEC

PROCEDURE DIVISION.
        MOVE 'BLUE' TO PARTS-COLOR

        EXEC SQL
                OPEN CPARTS
        END-EXEC
        PERFORM TESTSQLCODE

        EXEC SQL
                FETCH CPARTS
                INTO        :PARTS-PNO,
                            :PARTS-PNAME ,
                            :PARTS-WEIGHT

        END-EXEC
        PERFORM TESTSQLCODE
```

```
        PERFORM UNTIL SQLCODE NOT EQUAL  ZERO
                DISPLAY 'PNO : ' PARTS-PNO
                DISPLAY 'PNAME : 'PARTS-PNAME

                EXEC SQL
                        FETCH CPARTS
                        INTO        :PARTS-PNO,
                                    :PARTS-PNAME ,
                                    :PARTS-WEIGHT
                END-EXEC
                PERFORM TESTSQLCODE
        END-PERFORM

        EXEC SQL
                CLOSE CPARTS
        END-EXEC
        PERFORM TESTSQLCODE
        GOBACK.



TESTSQLCODE.
        EVALUATE TRUE
            WHEN SQLCODE EQUAL ZERO
                CONTINUE
            WHEN SQLCODE GREATER ZERO
                DISPLAY 'WARNING : ' SQLCODE
            WHEN SQLCODE LESS ZERO
                DISPLAY 'ABEND DU PROGRAMME '
                PERFORM ABEND-PROG
        END-EVALUATE
```

INSTRUCTIONS DE MISE A JOUR

- Pour mettre à jour les données dans une table DB2 à partir d'un programme COBOL, il est nécessaire de combiner des instructions SQL au sein du code COBOL.

- Ce processus, appelé SQL embarqué (Embedded SQL), permet aux développeurs d'effectuer des opérations telles que la sélection, l'insertion, et la mise à jour des données dans une base DB2 sans quitter l'environnement COBOL.

- Le principe fondamental pour réaliser une mise à jour consiste à :

  - Déclarer les variables nécessaires dans le programme COBOL pour recevoir les données de la base ou contenir les nouvelles valeurs.

  - Écrire et exécuter des instructions SQL telles que UPDATE, INSERT, DELETE pour modifier les enregistrements en fonction des critères spécifiés.

INSTRUCTIONS DE MISE À JOUR

- Cas de la suppression

```
EXEC SQL
        DELETE FROM PARTS
        WHERE P_NO = 'P1'
END-EXEC
```

- Maintenant, lorsque l'on veut utiliser une variable, on est contraint de passer par les HOST VARIABLES préalablement déclarées.

```
MOVE 'P1' TO PARTS-PNO
EXEC SQL
        DELETE FROM PARTS
        WHERE P_NO = :PARTS-PNO
END-EXEC
```

Ici, il s'agit d'une variable cobol qui est utilisée dans une instruction DB2.

Remarque : Quand on met un : devant une variable, cette variable est dite HOST VARIABLE, variable qui appartient au langage HOTE de SQL.

Indicateurs de Variables

- Les indicateurs de variables dans un programme COBOL/DB2 sont utilisés pour gérer les valeurs nulles dans une base de données relationnelle.
- En COBOL, les variables n'ont pas de moyen direct de représenter une valeur NULL. Ces indicateurs permettent donc de résoudre ce problème.
- En DB2, une colonne peut contenir une valeur NULL, qui signifie plutôt l'absence totale de valeur. Ce n'est pas la même chose qu'une valeur vide ou zéro, c'est plutôt l'absence totale de valeur.
- Un indicateur de variable est un champ de données COBOL utilisé pour indiquer si une valeur dans une colonne DB2 est NULL ou non.
- Quand une valeur est récupérée d'une colonne DB2, le système affecte cette valeur à la variable COBOL correspondante. Si la colonne contient un NULL, la variable indicatrice est utilisée pour signaler cela.
- Un indicateur de variable est un entier binaire signé. (PIC S9(4) COMP)

Indicateur de Variables Fonctionnement

- Lorsqu'une ligne de la base de données est lue dans un programme COBOL à l'aide d'une instruction SQL comme FETCH, les indicateurs sont utilisés pour vérifier si une colonne contient une valeur NULL.
- Si la colonne n'est pas NULL, l'indicateur de variable est mis à 0.
- Si la colonne contient NULL, l'indicateur de variable prend une valeur négative, généralement -1, et aucune valeur n'est assignée à la variable COBOL.
- Mais, attention le champ auquel est associé l'indicateur de variable conserve la valeur lue lors du précédent FETCH.
- Evidemment, les indicateurs de variables portent sur des champs qui ont été définis comme potentiellement NULL au moment de la création de la table. Attention, à ne pas oublier de déclarer ces indicateurs afin de gérer le cas le moment venu.

Steeve ASSOUS
pour

INDICATEUR DE VARIABLE EXEMPLE

```
WORKING-STORAGE SECTION.
            ...
            EXEC SQL
                        DECLARE CPARTS CURSOR
                        FOR
                        SELECT PNO, PNAME, WEIGHT
                        FROM PARTS
            END-EXEC

77 INDIC-PNAME          PIC  s9(4) COMP.

PROCEDURE DIVISION.
            EXEC SQL
                        OPEN CPARTS
            END-EXEC
            PERFORM TESTSQLCODE

            EXEC SQL
                        FETCH CPARTS
                        INTO        :PARTS-PNO,
                                    :PARTS-PNAME:INDIC-PNAME ,
                                    :PARTS-WEIGHT

            END-EXEC
            PERFORM TESTSQLCODE
```

```
      PERFORM UNTIL SQLCODE NOT EQUAL  ZERO
            DISPLAY 'PNO : ' PARTS-PNO

            IF INDIC-PNAME IS NEGATIVE
                        MOVE 'NULLE' TO PARTS-PNAME
            END-IF

            DISPLAY 'PNAME : 'PARTS-PNAME
            PERFORM TESTSQLCODE

            EXEC SQL
                        FETCH CPARTS
                        INTO        :PARTS-PNO,
                                    :PARTS-PNAME:INDIC-PNAME ,
                                    :PARTS-WEIGHT
            END-EXEC

            PERFORM TESTSQLCODE
      END-PERFORM

      EXEC SQL
            CLOSE CPARTS
      END-EXEC
      PERFORM TESTSQLCODE
      GOBACK.

TESTSQLCODE.
            ....
```