Programmation DB2

Historique

- Le traitement des données a évolué de manière significative au fil des décennies, en passant par des étapes clés qui ont conduit au développement de bases de données relationnelles comme DB2.

- Dans les années 1950, les premiers ordinateurs étaient utilisés principalement pour des traitements par lots. Les données étaient stockées sur des supports physiques comme des bandes magnétiques ou des cartes perforées. Il n'y avait pas de systèmes de gestion de bases de données (SGBD) dans le sens moderne.

- Les données étaient stockées dans des fichiers séquentiels, et chaque application gérait ses propres fichiers. Ce processus était laborieux et peu flexible.

- À mesure que les besoins en traitement de données augmentaient, des approches plus structurées furent mises en place. Cela a conduit à la création des bases de données hiérarchiques et en réseau, comme **IMS** (Information Management System), développé par IBM en 1966.

Historique

- L'étape décisive dans l'évolution des bases de données est venue avec Edgar F. Codd, chercheur chez IBM, qui a proposé le modèle relationnel en 1970. Codd suggéra que les données pouvaient être stockées sous forme de tableaux (relations), permettant des manipulations plus simples et plus logiques via des langages déclaratifs comme le SQL (Structured Query Language).

- Le modèle relationnel a révolutionné la gestion des données en séparant les aspects logiques de l'organisation des données de leur représentation physique, offrant ainsi une flexibilité accrue pour manipuler les données.

- Dans les années 1970, plusieurs projets de recherche basés sur le modèle relationnel ont vu le jour, notamment **System R** d'IBM et **Ingres** de l'université de Californie à Berkeley.

- IBM a ensuite commercialisé ces idées sous la forme du logiciel **DB2** en 1983, marquant le début de l'ère des bases de données relationnelles commerciales.

Historique

- Le succès de DB2, ainsi que d'autres systèmes comme **Oracle** (1979) et **Microsoft SQL Server** (1989), a popularisé les bases de données relationnelles. Ces systèmes se sont imposés comme des outils clés dans les entreprises pour la gestion des grandes quantités de données structurées.

- DB2, notamment, a d'abord été conçu pour les mainframes (grands systèmes), mais a évolué pour supporter des environnements plus diversifiés comme UNIX, Linux et Windows.

- Aujourd'hui, DB2 continue d'évoluer pour répondre aux besoins des entreprises en matière de cloud computing, de big data et d'analyse prédictive. IBM a intégré des fonctionnalités pour le traitement en mémoire, l'analyse en temps réel et la compatibilité avec des bases de données non-relationnelles comme le **NoSQL**.

- De plus, DB2 est disponible en tant que service cloud, ce qui permet une gestion flexible et scalable des données.

Présentation

- DB2 (ou IBM Db2) est un système de gestion de bases de données relationnelles (SGBDR) développé par IBM, utilisé pour stocker, organiser et gérer de grandes quantités de données.
- Ce système est largement adopté dans les environnements critiques en raison de sa robustesse, de ses fonctionnalités avancées et surtout de ses capacités de sécurité renforcées.
- DB2 est un SGBDR qui organise les données sous forme de tables interconnectées.
- DB2 prend en charge le langage SQL, un standard pour la gestion des bases de données relationnelles. Il offre un ensemble complet d'instructions SQL permettant de réaliser toutes les opérations classiques de l'algèbre relationnelle (sélection, projection, jointures, etc.).
- L'un des points forts de DB2 est son mécanisme de sécurité et d'autorisation, qui se distingue des autres SGBD.

- Voici quelques-unes des fonctionnalités qui renforcent la sécurité dans DB2 :
  - Contrôle d'accès granulaire
  - Chiffrement des données
  - Authentification renforcée
  - Audit et traçabilité
  - Isolation des transactions
  - Mécanismes de haute disponibilité
- Ses capacités d'audit et ses options d'authentification renforcée en font une solution adaptée aux environnements sensibles, tels que les secteurs bancaires, médicaux ou gouvernementaux, où la protection des données est cruciale.
- Comparé à d'autres SGBDR, DB2 se distingue par sa flexibilité et ses capacités de sécurisation à plusieurs niveaux, garantissant à la fois la confidentialité, l'intégrité et la disponibilité des données.

Syntaxe

- L'ordre SQL lui-même ne change pas mais il faut qu'il soit reconnu par le compilateur du langage de programmation.

- Pour cela :
  - il doit être précédé par EXEC SQL
  - il doit être terminé par END-EXEC en COBOL

```
EXEC SQL
        ....
        ORDRE SQL
        ....
END-EXEC
```

Zone SQLCA

STRUCTURED QUERY LANGUAGE COMMUNICATION AREA

- Pour travailler avec SQL dans un programme, il est impératif de déclarer dans le programme une zone de communication avec SQL.

- C'est la zone SQLCA. Elle est définit en WORKING-STORAGE SECTION.

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

Il s'agit d'une zone qui joue un rôle crucial dans la gestion des erreurs et des diagnostics SQL en retour après l'exécution d'une commande SQL, qu'elle soit réussie ou échouée.

Zone SQLCA Structure

- La structure de la **SQLCA** est composée de plusieurs champs, chacun ayant un rôle précis pour renvoyer des informations spécifiques après une exécution SQL.

- Voici les principaux champs :
  - SQLCODE (int) :
    - Ce champ renvoie un code de retour qui indique le résultat de l'exécution de l'instruction SQL
  - SQLERRM (array)
    - Ce champ contient un message explicatif plus détaillé lorsque SQLCODE contient une valeur négative. Il peut s'agir d'une description ou d'un diagnostic de l'erreur rencontrée.
  - SQLERRD (array of int) :Tableau contenant six éléments utilisés pour stocker des informations supplémentaires :
    - SQLERRD(1) : Nombre de lignes affectées par l'instruction SQL.
    - SQLERRD(2) : Code SQL interne, utilisé pour des informations spécifiques à DB2.
    - SQLERRD(3) : Nombre de lignes ou d'objets accédés par la requête.

Zone SQLCA SQLCODE

- La zone SQLCA est intéressante dans la mesure où elle nous permet, entre autres, de récupérer un code retour, après l'exécution d'un ordre SQL.
- Ce code retour porte le nom de SQLCODE .
- Il appartient au programmeur de tester le code retour de SQL pour déterminer l'action à entreprendre.
- Quelques codes retour :
-
  - SQLCODE = ZERO ➔ PAS D'ERREUR
  - SQLCODE > ZERO ➔ WARNING
    - EXEMPLES : SQLCODE = 100
      - DELETE, UPDATE, SELECT : NON TROUVE
      - -
  - SQLCODE < ZERO ➔ ERREUR GRAVE ABEND DU PROGRAMME

DECLARATION DES TABLES

- Déclarer les tables dans un programme COBOL/DB2 présente de nombreux avantages qui contribuent à la fiabilité, la maintenabilité, et la performance du code.

- Cette pratique améliore la lisibilité du code, facilite la détection des erreurs potentielles dès la compilation, et assure une meilleure correspondance entre la structure de la base de données et le programme COBOL.

- Intérêts de cette déclaration :
  - S'assure de la cohérence entre les types de données
  - Réduit les erreurs d'exécution
  - Préparation des accès SQL associés
  - Réduit les temps de compilation et d'exécution
  - Contrôle les accès
  - Validé par la précompilation DB2

DECLARATION DES TABLES
Exemple

- Les tables SQL auxquels on accède dans un programme doivent être déclarées.

```
EXEC SQL
        DECLARE <NOM DE LA TABLE> TABLE
                (CHAMP1 TYPE(LONGUEUR),
                 CHAMP2 TYPE(LONGUEUR),
                 CHAMPn TYPE(LONGUEUR)
                )
END-EXEC
```

Par exemple :

```
EXEC SQL
        DECLARE PARTS TABLE
                (PNO      CHAR(3) NOT NULL,
                PNAME              CHAR(5),
                COLOR              CHAR(5) NOT NULL,
                WEIGHT             SMALLINT NOT NULL,
                CITY               CHAR(7) NOT NULL)
END-EXEC
```

Cette déclaration n'est pas obligatoire, mais fortement recommandée car le compilateur peut s'en servir pour contrôler la description des tables.

Host Variables

- Les **host variables** sont des variables COBOL qui sont utilisées dans les instructions SQL pour stocker des valeurs transmises à la base de données ou pour recevoir des résultats de requêtes SQL.

- Les **host variables** sont le **pont** entre le langage de programmation COBOL et le système de gestion de base de données DB2. Elles permettent d'échanger des informations entre les deux environnements et de contrôler la compatibilité des formats.

- De plus, déclarer explicitement les **host variables** garantit que les types de données COBOL utilisés dans les requêtes SQL sont **compatibles avec les types de données** des colonnes de la base DB2. Cela aide à prévenir les erreurs liées à la conversion de types ou aux incohérences de format.

- Les Host variables sont des variables du programme destinées à être utilisées dans des ordres SQL.

- **ATTENTION** : les host variables doivent être précédées de deux points (:) pour les distinguer des noms de colonnes des tables de DB2 qui sont des variables SQL.

Steeve ASSOUS
pour

AJC FORMATION

Tableau de correspondance SQL DB2 / COBOL

Type DB2	Type COBOL	Description
CHAR(n)	PIC X(n)	Chaîne de caractères fixe de longueur n.
VARCHAR(n)	PIC X(n)	Chaîne de caractères variable de longueur maximale n.
BLOB	PIC X(n)	Objet binaire de grande taille (Binary Large Object).
SMALLINT	PIC S9(4) COMP	Entier court (2 octets, plage de -32 768 à 32 767).
INTEGER	PIC S9(9) COMP	Entier long (4 octets, plage de -2 147 483 648 à 2 147 483 647).
BIGINT	PIC S9(18) COMP	Entier très long (8 octets).
DECIMAL(p,s)	PIC S9(p-s)V9(s) COMP-3	Nombre décimal avec précision p et échelle s, format zoné (packed decimal).
NUMERIC(p,s)	PIC S9(p-s)V9(s) COMP-3	Identique à DECIMAL.
FLOAT	COMP-1 ou COMP-2	Nombre à virgule flottante (simple précision ou double précision).
DOUBLE	COMP-2	Nombre à virgule flottante double précision (8 octets).
REAL	COMP-1	Nombre à virgule flottante simple précision (4 octets).
DATE	PIC X(10)	Date au format YYYY-MM-DD.
TIME	PIC X(8)	Heure au format HH:MM:SS.
TIMESTAMP	PIC X(26)	Horodatage avec année, mois, jour, heure, minute, seconde, et fractions de seconde au format YYYY-MM-DD-HH.MM.SS.FFFFFF.

- La déclaration de ces variables dites HOST VARIABLES se fait en WORKING STORAGE SECTION.

```
WORKING-STORAGE SECTION.

01 PARTS-PNO      PIC   X(3).
01 PARTS-PNAME    PIC   X(5).
01 PARTS-COLOR    PIC   X(5).
01 PARTS-WEIGHT   PIC   S9(4) COMP.
01 PARTS-CITY     PIC   X(7).
```

DCLGEN
Declaration Generator

- Cette étape de correspondance est assez fastidieuse dans la mesure où cette correspondance doit être réalisée pour chaque table utilisée dans le programme. Certes, il existe des moyens de l'écrire une fois et de réutiliser dans différents programmes.

- Cependant, cela serait susceptible de provoquer des problèmes de maintenance en cas d'évolution des tables.

- Un outil de DB2I (le DECLARATOR GENERATOR) aide à créer un membre dans une bibliothèque défini à destination de l'ordre INCLUDE.

- Il offre un gage de cohérence, de justesse, gain de performance, et de temps une fois configuré selon nos souhaits.

- Ce membre est inséré par le biais de l'ordre INCLUDE, indiqué dans la WSS d'un programme COBOL comme suit :

```
EXEC SQL
        INCLUDE <NOM DE LA TABLE>
END-EXEC
```

Par exemple :

```
EXEC SQL
        INCLUDE PARTS
END-EXEC
```

DCLGEN

```
                           DCLGEN                                SSID: DSN1
===>

Enter table name for which declarations are required:
 1  SOURCE TABLE NAME ===>                                  (Unqualified)
 2  TABLE OWNER ..... ===>                                     (Optional)
 3  AT LOCATION ..... ===>                                     (Optional)

Enter destination data set:          (Can be sequential or partitioned)
 4  DATA SET NAME ... ===>
 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 ===>                                     (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)
```