Département d'informatique
Faculté des sciences
Université de Sherbrooke
http://info.usherbrooke.ca/llavoie

Thème
Bases de données

Sujet
SQL

Module de cours
BD190
Standard de programmation SQL, niveau 1

par
Luc Lavoie et Christina Khnaisser

dernière modification : 2018-09-01

Historique des révisions

version	date	auteur	description
0.3.0a	2018-01-03	LL	Prise en compte de l'évolution des pratiques, de MS-SQL et MariaDB.
0.2.2b	2016-07-27	CK	Correction de coquilles.
0.2.2a	2015-08-12	CK	Explicitation du format des instructions, ajout d'exemples de commentaires.
0.2.1a	2015-01-18	LL	Ajout d'exemples, assouplissement des règles relatives aux guillemets.
0.2.0a	2014-01-18	LL	Précisions apportées aux règles de dénomination.
0.1.0a	2012-10-14	LL	Prêt pour une première revue, document encore incomplet.
0.0.1a	2012-09-18	LL	Première esquisse.

Sommaire

Le module

Le présent module de cours a été rédigé dans le cadre de l'exploration du thème Bases de données du groupe Ἀκαδήμεια. On y décrit un standard de programmation SQL à l'intention de programmeurs débutants. Le standard ne couvre donc qu'une petite partie du langage et ne comprend que les règles les plus importantes. D'autres standards plus complets sont envisagés.

Le thème Bases de données

Le thème Bases de données comprend un ensemble de modules sur la théorie des bases de données, le modèle relationnel, les langages SQL, Tutorial D et Discipulus.

Le groupe Ἀκαδήμεια

Ἀκαδήμεια, un groupe de réflexion du collectif Μῆτις, s'intéresse à la transmission des connaissances du domaine de l'informatique.

Le collectif Μῆτις

Le collectif Μῆτις s'intéresse l'apport de l'informatique aux divers domaines du savoir et aux sciences en particulier.

Summary

NA

1 Introduction

1.1 Objet et portée du document

Le présent document a pour but de décrire un standard minimal de présentation de programmes SQL destinée aux étudiants d'un premier cours en bases de données (typiquement IFT 187 Éléments de bases de données, à l'Université de Sherbrooke). Un deuxième standard [STD-SQL-02] couvre des éléments plus avancés et complète celui-ci.

1.2 Évolution du document

La première version du document a été établie sur les bases suivantes :

- ◊ des règles de pratique reconnues par certaines communautés d'utilisateurs ;
- ◊ les dialectes DB2, MS-SQL, MariaDB, Oracle et PostgreSQL ;
- ◊ la pratique au sein du collectif Μῆτις ;
- ◊ les expériences réalisées au sein du Groupe de recherche interdisciplinaire en informatique de la santé (GRIIS).

1.3 Problématique

Le standard doit minimalement couvrir les instructions SQL normalement traitées dans un cours d'introduction. Il ne doit pas couvrir beaucoup plus afin de rester le plus court possible et de ne pas interférer avec l'enseignement de concepts plus importants.

1.4 Rappels

NA

2 Règles applicables

2.1 Principes

Principes généraux : voir [STD-PROG-DI] 1.1.

Transportabilité : viser à une transportabilité maximale dans le respect de la lisibilité, de la fonctionnalité et de la modifiabilité.

2.2 Encodage, disposition générale et commentaires

2.2.1 Encodage

Jeu de caractères : ISO 8859-1 (Latin 1).

Encodage : UTF-8.

Fins de ligne : LF (tel que sous Unix en général, Linux et OS X en particulier).

Caractères de commande : aucun sauf LF (en particulier pas de TAB, ni de CR, ni de FF).

2.2.2 Disposition générale

Indentation : décaler de 2 espaces par niveau.

Longueur des lignes : viser un maximum de 80 caractères, ne pas dépasser 100 caractères.

2.2.3 Commentaires

Commentaire général : voir [STD-PROG-DI] à la section 2.2.

Plus spécifiquement, voir l'annexe A du présent document.

2.3 Nomenclature

2.3.1 Identificateurs prescrits par le langage SQL

Identifiants SQL : que ce soit les identificateurs prescrits par la norme ISO (*TABLE, CONSTRAINT, SELECT, etc.*) ou ceux déterminés par le dialecte utilisé (*TRUNCATE* en PostgreSQL), toutes leurs lettres doivent être majuscules.

2.3.2 Identificateurs choisis par le programmeur

Un identificateur peut prendre deux formes en SQL : la forme *limitée* (une lettre suivie de lettres, de chiffres ou de traits[1]) et la forme *libre* (une suite de caractères comprise entre guillemets). Comme la capitalisation n'est pas prise en compte dans la forme libre (pas plus que pour les identificateurs réservés) cela entraine un problème d'équivalence entre les deux formes. Par exemple, quelle est la forme libre correspondante à Ident18 ? Est-ce "ident18", "IDENT18", "Ident18" ou encore une autre forme ?

La norme étant muette à cet égard, les éditeurs ont privilégié différentes solutions, au détriment de la transportabilité des programmes. Par exemple, Oracle « normalise » les identificateurs de forme limitée en majuscules alors que PostgreSQL les « normalise » en minuscules.

La situation se complique du fait que quelques dialectes (Transact-SQL, MySQL, MS-SQL) dérogent du standard ISO en utilisant les crochets (en lieu et place des guillemets) pour délimiter les identificateurs libres, par exemple [Ident18].

Les règles suivantes ont pour but de réduire l'ampleur du problème, du moins au sein des dialectes respectant les formes prescrites par la norme ISO :

- ◊ Dans un même schéma (et préférablement dans une même BD, un même produit ou une même organisation), il convient de n'utiliser qu'une seule forme (limitée ou libre).
- ◊ Dans tous les cas, commencer l'identificateur par une lettre.
- ◊ Lorsque plusieurs mots sont concaténés les uns aux autres pour composer l'identificateur, les mots subséquents débutent par une majuscule (mais ne pas oublier que les identificateurs de forme limitée ne sont pas sensibles à la casse). La casse de la lettre initiale est déterminée par la nature de ce que dénote l'identificateur :

[1] En fait, les règles sont beaucoup complexes et variables, voir annexe 1.

- minuscule : attribut (champ, colonne, *column*), fonction ;
- majuscule : domaine, type, procédure, relation (table), contrainte, tuple (ligne, *row*).

Si la forme limitée est choisie :

◊ Composer les identificateurs à l'aide de lettres et de chiffres, n'utiliser le trait souligné que pour les usages prévus ci-après (corolaire : pas d'émojis, d'accents, de lettres grecques, cyrilliques, arabes, farsis, etc.).

◊ Ne pas utiliser la forme libre, sauf contrainte externe incontournable (importation d'éléments provenant de sources externes faisant usage de la capitalisation ou de caractères non prévus au présent standard).

Les règles suivantes s'appliquent en fonction de ce que désigne l'identificateur, indépendamment de la forme choisie.

Domaine :

◊ Utiliser un nom ou un groupe nominal.

◊ Éviter la marque du pluriel.

◊ Débuter par une majuscule.

Type :

◊ Voir domaine.

Attribut :

◊ Utiliser un nom ou un groupe nominal.

◊ Débuter par une minuscule.

◊ Choisir un identificateur par concept, si un concept apparait plus d'une fois dans une table, le suffixer par une caractéristique distinctive au sein de la relation introduite par le trait souligné, par exemple : adresse, adresse_facturation, adresse_livraison, raisonSociale, raisonSociale_client, raisonSociale_fournisseur. Deux attributs sont relatifs au même concept s'ils sont de même type et peuvent être « sensément » comparer entre eux (ce dernier critère est redondant si la politique de typage est appliquée rigoureusement).

Fonction :

◊ Utiliser un adjectif pour une fonction booléenne, le nom de la caractéristique pour les fonctions d'autres types.

◊ Débuter par une minuscule.

Procédure :

◊ Utiliser un verbe à l'infinitif présent.

◊ Débuter par une majuscule.

Automatisme (déclencheur ou *trigger*) :

◊ Voir procédure.

Relation (table, vue) :

◊ Utiliser un nom ou un groupe nominal.

◊ Éviter la marque du pluriel.

◊ Débuter par une majuscule.

Schéma :

◊ Utiliser un nom ou un groupe nominal.

◊ Éviter la marque du pluriel.

◊ Débuter par une majuscule.

◊ Si plusieurs schémas sont apparentés, les distinguer par suffixation introduction par le trait souligné, par exemple :

Contrainte de table (ou d'attribut) :

◊ Un nom de contrainte est composé d'un préfixe et d'un identifiant. Le préfixe est composé du nom de la table suivi du trait souligné. Les identifiants des contraintes générales (CHECK) sont laissés au choix de du programmeur, mais ceux des contraintes de clé (PRIMARY KEY, UNIQUE et FOREIGN KEY) sont imposées[2] :

- cc0 : clé candidate primaire (PRIMARY KEY) ;
- cc1, ..., cc9 : clés candidates secondaires (UNIQUE) ;
- cr0, ..., cr9 : clés référentielles (FOREIGN KEY).

Assertion (contrainte de schéma) :

◊ Un nom de contrainte est composé d'un préfixe et d'un identifiant. Le préfixe est composé du nom du schéma suivi du trait souligné. Les identifiants sont libres.

2.4 Mise en forme

2.4.1 Consignes générales

Style général retenu : la mise en forme des instructions doit suivre un standard reconnu (Oracle, IBM, INGRES) et être utilisée uniformément partout.

2.4.2 Consignes particulières

...

2.4.2.1 CREATE TABLE

Emplacement des contraintes : une contrainte NOT NULL doit suivre la définition du type, toutes les autres contraintes doivent être regroupées à la fin de la définition par l'entremise d'un énoncé CONSTRAINT. L'identification de chaque contrainte est obligatoire.

Valeurs par défaut : sans être interdites, elles sont fortement déconseillées. Elles se placent à la suite de la contrainte NOT NULL.

2.4.2.2 INSERT

L'instruction INSERT doit comprendre la liste des colonnes (même si elle est facultative en SQL) puis la liste des valeurs. Cette contrainte prend son sens dans un contexte d'évolution, où des colonnes peuvent

[2] Dans un contexte anglophone, les codes « cc » sont remplacés par « ck » et « cr » par « fk ».

ajoutées ou retranchées sans invalider les listes de valeurs ; des permutations peuvent alors induire des erreurs très difficiles à déceler.

2.4.2.3 DELETE

...

2.4.2.4 UPDATE

...

2.4.2.5 SELECT

Chaque élément de l'instruction SELECT doit être défini sur une ligne.

Exemple :

```
SELECT id, nom, prenom
FROM client
WHERE statut > 10
```

3 Exemples

3.1 E1

```
CREATE
  TABLE Sejour
/*
--------------------------------------------------------------------------------
Si confirmée, la date de fin de séjour est effective et définitive, sinon elle
est projetée et maintenue à jour (tout au long du séjour) par le médecin
traitant. Ainsi, elle n'est jamais nulle et peut toujours servir à établir des
planifications et des programmes.
--------------------------------------------------------------------------------
*/
  (
    idPatient  CHAR(6)        NOT NULL,
    dateDebut  DATE           NOT NULL,
    dateFin    DATE           NOT NULL,
    confirmee  BOOLEAN        NOT NULL,
    idUnite    CHAR(4)        NOT NULL,
    noChambre  NUMERIC(4)     NOT NULL,
    noLit      NUMERIC(1)     NOT NULL,
    CONSTRAINT Sejour_cc0     PRIMARY KEY (idPatient, dateDebut),
    CONSTRAINT Sejour_cc1     UNIQUE (idPatient, dateFin),
    CONSTRAINT Sejour_cc2     UNIQUE (noChambre, noLit, dateDebut),
    CONSTRAINT Sejour_cc3     UNIQUE (noChambre, noLit, dateFin),
    CONSTRAINT Sejour_cr0     FOREIGN KEY (idPatient) REFERENCES Patient,
    CONSTRAINT Sejour_cr1     FOREIGN KEY (idUnite) REFERENCES Unite,
    CONSTRAINT Sejour_periode CHECK (dateDebut <= dateFin),
    CONSTRAINT Sejour_chambre CHECK (1000 <= noChambre),
    CONSTRAINT Sejour_lit     CHECK (1 <= noLit AND noLit <= 4)
    );
```

3.2 E2

```
CREATE
  TABLE "ProduitPhysique"
/*
Les longueurs sont en centimètres, le poids en grammes.
*/
  (
    "noProduit" CHAR(6) NOT NULL,
    "longueur" NUMERIC(4) NOT NULL,
    "largeur" NUMERIC(4) NOT NULL,
    "hauteur" NUMERIC(4) NOT NULL,
    "poids" NUMERIC(5) NOT NULL,
    "état" VARCHAR(14) NOT NULL,
    CONSTRAINT "ProduitPhysique_cc0" PRIMARY KEY ("noProduit"),
    CONSTRAINT "ProduitPhysique_cr0" FOREIGN KEY ("noProduit")
      REFERENCES "Produit",
    CONSTRAINT "ProduitPhysique_volume" CHECK
      (0 < "longueur" AND 0 < "largeur" AND 0 < "hauteur"),
    CONSTRAINT "ProduitPhysique_masse" CHECK (0 < "poids")
    CONSTRAINT "ProduitPhysique_état"
      CHECK ("état" IN ('neuf', 'reconditionné', 'usagé'))
  );
```

Annexe 1 – Forme limitée des identificateurs

La norme SQL (ISO/IEC 9075-2:2003) prescrit notamment ce qui suit :

```
      <regular identifier> ::= <identifier body>
      <identifier body> ::= <identifier start> [ <identifier part>... ]
      <identifier part> ::=
          <identifier start>
        | <identifier extend>
      <identifier start> ::= !! See the Syntax Rules
      <identifier extend> ::= !! See the Syntax Rules
   Syntax Rules
   1) An <identifier start> is any character in the Unicode General Category
      classes "Lu", "Ll", "Lt", "Lm", "Lo", or "Nl".
      NOTE 58 – The Unicode General Category classes "Lu", "Ll", "Lt", "Lm",
      "Lo", and "Nl" are assigned to Unicode characters that are, respectively,
      upper-case letters, lower-case letters, title-case letters, modifier
      letters, other letters, and letter numbers.
   2) An <identifier extend> is U+00B7, "Middle Dot", or any character in the
      Unicode General Category classes "Mn", "Mc", "Nd", "Pc", or "Cf".
      NOTE 59 – The Unicode General Category classes "Mn", "Mc", "Nd", "Pc",
      and "Cf" are assigned to Unicode characters that are, respectively,
      nonspacing marks, spacing combining marks, decimal numbers, connector
      punctuations, and formatting codes.
   [...]
   12) In a <regular identifier>, the number of <identifier part>s shall be
       less than 128.
```

Malheureusement, la comparaison de dialectes courants impose des contraintes supplémentaires (a=lettre, n=chiffre) :

```
      ------------------------------------------------------------------
      dialecte    longueur    start           extend
      ------------------------------------------------------------------
      DB2          30*        a, @, #, $      a, n, _, @, #, $
      MS-SQL      116         a, _, @, #      a, n, _, @, #, $
      mySQL        64         "Tout ce qui n'est pas ambigu !"
      Oracle       30         a               a, n, _, $, #
      PostgreSQL   31**       a, $, _         a, n, _, $
      ------------------------------------------------------------------
```

(*) Le cas de DB2 est très problématique, car la longueur maximale distinctive d'un identificateur varie selon ce qu'il désigne et selon le contexte ! Cette longueur peut être aussi faible que huit (8).

Les règles (incomplètes) relatives à DB2 sont les suivantes :

```
      The following conventions apply when naming database manager objects,
      such as databases and tables:
       * Character strings that represent names of database manager objects
         can contain any of the following: a-z, A-Z, 0-9, @, #, and $.
       * Unless otherwise noted, names can be entered in lowercase letters;
         however, the database manager processes them as if they were uppercase.
```

```
* Names can contain the following anywhere but in the first character
  of the string: _
* A database name or database alias is a unique character string
  containing from one to eight letters, numbers, or keyboard
  characters from the set described above.
* Databases are cataloged in the system and local database directories
  by their aliases in one field, and their original name in another.
  For most functions, the database manager uses the name entered in
  the alias field of the database directories. (The exceptions are
  CHANGE DATABASE COMMENT and CREATE DATABASE, where a directory
  path must be specified.)
* The name or the alias name of a table or a view is an SQL identifier
  that is a unique character string 1 to 128 characters in length.
* Column names can be 1 to 30 characters in length.
* A fully qualified table name consists of the schema.tablename.
  The schema is the unique user ID under which the table was created.
  The schema name for a declared temporary table must be SESSION.
* Local aliases for remote nodes that are to be cataloged in the node
  directory cannot exceed eight characters in length.
  The first character in the string must be an alphabetic character,
  @, #, or $; it cannot be a number or the letter sequences
  SYS, DBM, or IBM.
```

(**) Le code source de PostgreSQL étant ouvert, une installation peut être faite en utilisant une valeur supérieure ou égale à 31.

On déduit du tableau que, pour assurer une transportabilité optimale :

- ◊ Les identificateurs engendrés devront être distincts sur les 30 premiers caractères (DB2, Oracle), même s'il est possible que cela pose des problèmes sous DB2.
- ◊ Le premier caractère devra être une lettre (Oracle).
- ◊ Dans la mesure où le symbole $ est traditionnellement réservé aux identificateurs introduits par le SGBD lui-même, le seul caractère légitime autre qu'une lettre ou un chiffre pouvant apparaitre dans un identificateur « régulier » est le soulignement ('_').

Annexe 2 – Commentaires prescrits pour les travaux

Chaque fichier contenant un composant logiciel (script, programme, etc.) doit comporter au moins trois commentaires : deux au début et un à la fin. Le premier commentaire a pour objectif d'identifier uniquement le composant et comporte les champs suivants :

- ◊ Activité : activité (groupe-cours ou projet) pour lequel le composant est soumis ;
- ◊ Trimestre : période (année et trimestre) universitaire au cours de laquelle le composant est soumis ;
- ◊ Encodage : le jeu de caractères et l'encodage utilisés, avec mention explicite du marqueur de fin de ligne ;
- ◊ Plateforme : environnement sous lequel le composant peut être analysé et exécuté ;
- ◊ Responsable : le courriel du contributeur (voir commentaire final) auquel s'adresser ;
- ◊ Version : indicateur progressif mis à jour incrémentalement à chaque modification du composant ;
- ◊ Statut : état du composant (en développement, en tests, en vigueur, obsolète, etc.) ;

Ce premier commentaire peut être complété par des identifiants produits par des logiciels de gestion de versions ; ils sont alors regroupés sous le champ facultatif « Références ».

Le deuxième commentaire a pour objectif de donner deux descriptions du fichier : une courte et une longue.

Le dernier commentaire, placé à la fin du fichier, a pour objectif de rassembler les informations annexes dont la liste des contributeurs (courriel et matricule), l'adresse postale du responsable (facultatif), la propriété intellectuelle, la liste des tâches à faire, l'historique de l'évolution du composant.

Exemple de commentaire initial

```
/*
-- ============================================================================ A
-- Composant Evaluation_cre.sql
-- ----------------------------------------------------------------------------
Activité : IFT 187
Trimestre : 2017-3
Encodage : UTF-8, fin de ligne Unix (LF)
Plateforme : PostgreSQL 9.4.4 à 9.6
Responsable : Luc.Lavoie@USherbrooke.ca
Version : 0.2.0c
Statut : en vigueur
-- ============================================================================ A
*/
```

Exemple de deuxième commentaire

```
/*
-- ============================================================================ B
Création du schéma correspondant au modèle d'Evaluation et consignation des
résultats d'évaluation de l'Université de Samarcande (UdeS).

Le modèle est décrit ci-après, il est présenté dans le module BD012 [mod].
Une seule modification y a été faite, l'introduction d'une clé candidate
supplémentaire dans la relation Etudiant (de façon peu réaliste, les noms des
étudiants sont réputés uniques). Cet ajout permet d'illustre le cas d'une
table ayant deux clés candidates et la syntaxe distinctive de PRIMARY KEY
et de UNIQUE.

  Entêtes
    Activité (sigle : Texte ; titre : Texte)
    TypeÉvaluation (code : Texte ; description : Texte)
    Étudiant (matricule : Texte ; nom : Texte ; adresse : Texte)
    Résultat (
        matricule : Texte ;
        TE : Texte ;
        activité : Texte ;
        trimestre : Entier ;
        note : Entier [0..100]
        )
  Clés
    Activité : {sigle}
    TypeÉvaluation : {code}
    Étudiant : {matricule}
    Résultat : {matricule, TE, activité, trimestre}

Notes de mise en oeuvre
(a) Certains choix quant à la représentation des attributs diffèrent de ceux
    des modules ; ainsi la note passe de INTEGER à SMALLINT.
(b) La présente version est le résultat de trois itérations. Remarquez
    l'introduction du présent commentaire initial et du commentaire
    final, en accord avec le standard BD190. Ces commentaires doivent être
    reproduits et adaptés dans les travaux remis par les étudiants.
-- ============================================================================ B
```

Exemple de commentaire final

```
/*
-- ============================================================================ Z
Contributeurs :
  (CK) Christina.Khnaisser@USherbrooke.ca (matricule 09123456)
  (LL) Luc.Lavoie@USherbrooke.ca (matricule 72123456)

Adresse, droits d'auteur et copyright :
  Groupe Metis
  Département d'informatique
  Faculté des sciences
  Université de Sherbrooke
  Sherbrooke (Québec)  J1K 2R1
  Canada
  http://info.usherbrooke.ca/llavoie/

Tâches projetées :
2013-09-09 (LL) : Compléter le schéma
  * Compléter les contraintes.

Tâches réalisées :
2015-08-28 (CK) : Harmonisation
  * avec les modules BD012 et BD100 (voir [mod]) ;
  * avec les modifications apportées au standard BD190 (voir [mod]).
2013-09-03 (LL) : Création
  * CREATE TABLE Activite, TypeEvaluation, Etudiant, Resultat.

Références :
[mod] http://info.usherbrooke.ca/llavoie/enseignement/Modules/
-- ----------------------------------------------------------------------------
-- Fin de Evaluation_cre.sql
-- ============================================================================ Z
*/
```

Glossaire

BD
: Base de données.

dialecte
: Variante du langage SQL proposé par un éditeur de SGBDR. Parmi les dialectes les plus fréquents ceux d'Oracle (Oracle), d'IBM (DB2), de Microsoft (MS-SQL) et SyBase (Transact-SQL), de SAP (Hanna) et de PostgreSQL (PostgreSQL).

ISO
: *International Organization for Standardization*. Organisation internationale pour la normalisation.

NA
: Non applicable.

PL/SQL
: *Procedural language / Structured Query Language*.
Langage utilisé par certains dialectes SQL pour spécification des PSM.

PSM
: *Persistent Stored Module*.

SGBD
: Système de gestion de bases de données.

SGBDR
: Système de gestion de bases de données relationnelles.

SQL
: *Structured Query Language.*

XML
: *Extensible Markup Language.*

Références

[PG100-STD]
Marc FRAPPIER et coll.
Norme de documentation des programmes.
Version 1.02 ; Groupe Ἀκαδήμεια, Département d'informatique, Faculté des sciences, Université de Sherbrooke, Sherbrooke, Québec, janvier 2014. Source consultée le 2014-01-20 :
http://info.usherbrooke.ca/llavoie/enseignement/Modules/PG100-STD_NDC.pdf.

[STD-PROD-DI]
Marc FRAPPIER et coll.
Norme de documentation des programmes.
Version 1.02 ; Département d'informatique, Faculté des sciences, Université de Sherbrooke, Sherbrooke, Québec, aout 2004. Source consultée le 2012-10-01 :
http://www.usherbrooke.ca/informatique/fileadmin/sites/informatique/documents/Intranet/Documentation_informatique/Normes_de_programmation/normes-de-programmation-1.pdf.