Thème
Bases de données

Sujet
SQL

Note technique
MAD_STD-SQL-01_NT
Standard de programmation SQL, niveau 1

proposée par
Samuel Dussault, Marc Dupuis, Christina Khnaisser et Luc Lavoie

Version
042a

Statut
document de travail

dernière modification : 2022-08-04

Sommaire

La présente note technique décrit la première partie du standard de programmation SQL à l'intention de programmeurs du GRIIS, du MAD, de Μῆτις et d'Ἀκαδήμεια.

Groupe Metis (Μῆτις)
Faculté des sciences
`http://info.usherbrooke.ca/llavoie/`

Groupe de recherche interdisciplinaire en informatique de la santé (GRIIS)
Faculté des sciences et Faculté de médecine et sciences de la santé
`https://griis.ca`

Université de Sherbrooke
Sherbrooke (Québec) J1K 2R1
Canada

Historique des révisions

version	date	auteur	description
0.4.2a	2022-07-31	LL	Corrections diverses en vue de la revue du 2022-08-03 (sujets à débattre surlignés en jaune).
0.4.1d	2022-06-10	LL	Droits, licences ; suite de documents ; cartouche initial.
0.4.1c	2022-06-10	LL+CK	Échange de commentaires
0.4.1b	2022-05-20	LL	Clarifications diverses, corrections syntaxiques, etc.
0.4.1a	2022-03-31	LL	Acceptation des multiples modifications, suggestions mineures, mise en forme GRIIS.
0.4.0a	2022-01-13	SD	Adaptation initiale pour le MAD.
0.3.0a	2018-01-03	LL	Prise en compte de l'évolution des pratiques, de MS-SQL et de 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.

1 Introduction

1.1 Objet et portée du document

Le présent document fait partie d'une suite de documents ayant pour but de décrire un standard minimal de présentation et d'organisation de programmes SQL. Il est d'abord destiné aux personnes développant des bases de données au sein des groupes Μῆτις (comprenant Ἀκαδήμεια) et GRIIS (comprenant MAD), mais peut être diffusé sans contraintes.

Le présent document est le premier de la série et s'intéresse principalement à la présentation.

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 des éditeurs IBM, MS-SQL, MariaDB, Oracle et PostgreSQL ;
- ◊ la pratique des groupes GRIIS, MAD, Μῆτις et Ἀκαδήμεια.

1.3 Problématique

La mise en forme des programmes conformément au présent standard devrait pouvoir être facilitée et validée à l'aide d'outils normalement fournis par les éditeurs spécialisés et les environnements de développement logiciel. Cette problématique est ignorée dans la version courante du document.

1.4 Rappels

S. O.

2 Règles applicables

2.1 Principes

La documentation s'adresse d'abord aux lecteurs d'un programme (réviseurs, valideurs, modificateurs), puis éventuellement à son auteur (6 mois après l'avoir écrit ; l'auteur pourrait ne pas très bien se souvenir de ce qu'il a fait ni des motivations qui l'ont guidé).

Elle aide le lecteur à comprendre le code source, à présenter les motivations des choix sous-jacents (par exemple les choix de conception) et à l'utiliser ; elle ne duplique pas les informations déjà contenues de manière explicite dans le code source.

Elle doit être claire, concise, précise, complète, uniforme et simple à maintenir. Elle doit être faite au fur et à mesure du développement du programme, facilitant le travail du rédacteur lui-même (l'écriture contribuant à la clarification des idées et à leur expression rigoureuse) et permettant de mieux s'assurer de sa complétude et de sa mise à disposition dès la diffusion du programme.

Elle facilite la maintenance des programmes dans une organisation en uniformisant le style de programmation, ce qui permet à une personne de reprendre plus facilement le travail d'une autre.

Finalement, elle facilite la diffusion, l'appropriation, l'utilisation et l'évolutivité du programme par la présentation claire et lisible des fonctionnalités.

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 macOS 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

Chaque fichier source comprend trois cartouches de commentaires :

- ◊ Le cartouche initial, servant à décrire sommairement le rôle du fichier et son encodage.
- ◊ Le cartouche principal, servant à décrire le contenu du fichier.
- ◊ Le cartouche final, comportant la mention des contributeurs, des droits, des licences, des adresses, des références et des tâches.

Le contenu de chaque cartouche est décrit à l'annexe C du présent document.

2.3 Nomenclature

2.3.1 Identificateurs prescrits par SQL

Identificateurs propres à SQL : que ce soit les identificateurs prescrits par la norme ISO ou ceux réservés par le dialecte utilisé, toutes leurs lettres doivent être minuscules. Pour les identificateurs prédéfinis par SQL, les règles applicables aux identificateurs choisis par le programmeur seront utilisées.

2.3.2 Identificateurs choisis par le programmeur

Un identificateur peut prendre deux formes en SQL : la forme non délimitée (une lettre suivie d'une suite de signes [lettres, chiffres ou ligatures][1]) et la forme délimitée (une suite de caractères comprise entre guillemets[2]). Comme la capitalisation n'est pas prise en compte dans la forme non délimitée (pas plus que pour les identificateurs prescrits), cela entraine un problème d'équivalence entre les deux formes. La norme ayant d'abord été muette à cet égard, les éditeurs ont privilégié différentes solutions, au détriment de la transportabilité des programmes.

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

[2] Cetains dialectes (dont Access, MS-SQL et mySQL) ont eu l'idée saugrenue d'utiliser d'autres délimiteurs que ceux prescrits par la norme (typiquement «[» et «]») tout en ne traitant pas adéquatement deux prescrits pas la norme. Si on projette de porter le code vers de tels dialectes, il n'y a guère d'autrer choix que de trouver une outil de conversion fiable (tâche pour le moins ardue et jusqu'à présent nous n'en avons identifier aucun) ou d'imposer les identificateurs non délimités.

Par exemple, la forme délimitée correspondant à la forme non délimitée Ident18 varie selon le dialecte : "ident18" (PostgreSQL[3]), "IDENT18" (ISO, Oracle), "Ident18" (MariaDB), etc. L'utilisation systématique des guillemets permet de s'affranchir de ce risque tout en permettant une meilleure documentation des programmes[4]. Elle est toutefois plus lourde à l'usage et souvent mal prise en charge par les outils et les ateliers de développement.

Les règles suivantes ont pour but de réduire l'ampleur de ce problème :

- Dans un même produit, il convient de n'utiliser qu'une seule forme (délimitée ou non). Dans un environnement où les produits se croisent, il est préférable qu'ils utilisent la même convention.
- Dans les projets à responsabilité partagée, une entente explicite à cet égard doit être convenue.

⚠ Le MAD poursuit ses recherches et son étude des outils et ateliers de développement.

Il pourrait en découler qu'une des solutions sera prescrite à l'avenir.

Règles générales quant à la forme lexicale de l'identificateur

Les règles lexicales sont les suivantes :

- les caractères permis sont les lettres, les chiffres et le tiret bas ;
- un mot est une suite de lettres ou de chiffres ;
- l'identificateur est composé d'une suite de mots séparés par un tiret bas ;
- le premier mot d'un identificateur (et donc de l'identificateur lui-même) doit débuter par une lettre.

Règles spécifiques applicables aux identificateurs

Les règles de composition des identificateurs sont les suivantes :

- les mots qui composent l'identificateur sont ordonnés du général au particulier, par exemple, pour désigner le statut de l'étape d'un processus, on utilisera `processus_etape_statut` ;
- un mot est écrit en minuscules, sauf :
  - les noms propres et les acronymes, qui conservent leur mise en forme usuelle ;
  - les cas prévus ci-après en fonction de la catégorie de l'identificateur .

La catégorie d'un identificateur est celle, au sein du langage SQL, de l'entité qu'il désigne. Les règles suivantes sont regroupées et applicables par catégories d'entités SQL.

---

[3] Voir https://www.postgresql.org/docs/15/sql-syntax-lexical.html#SQL-SYNTAX-IDENTIFIERS, en particulier « Quoting an identifier also makes it case-sensitive, whereas unquoted names are always folded to lower case. For example, the identifiers FOO, foo, and "foo" are considered the same by PostgreSQL, but "Foo" and "FOO" are different from these three and each other. (The folding of unquoted names to lower case in PostgreSQL is incompatible with the SQL standard, which says that unquoted names should be folded to upper case. Thus, foo should be equivalent to "FOO" not "foo" according to the standard. If you want to write portable applications you are advised to always quote a particular name or never quote it.) ».

[4] On remarque toutefois que la question de l'usage des guillemets est très débattue, notamment parce que certains dialectes, en contradiction avec la norme ISO, utilisent d'autres délimiteurs que les guillemets (e.a. : mySQL et Transact-SQL et MS-SQL).

Attribut :

- ◊ Débuter par une minuscule.
- ◊ Utiliser un nom (ou un groupe nominal) qui désigne la propriété de l'entité (du type) de la façon la plus spécifique possible.
- ◊ Dans le cas d'un attribut booléen, un adjectif peut être utilisé.

Type :

- ◊ Débuter par une majuscule.
- ◊ Utiliser un nom au singulier (ou un groupe nominal) qui désigne de la façon la plus spécifique possible l'ensemble des valeurs.

Domaine :

- ◊ Comme pour Type.

Fonction :

- ◊ Comme pour Attribut.

Procédure :

- ◊ Débuter par une majuscule.
- ◊ Utiliser un verbe (ou un groupe verbal) à l'infinitif présent qui désigne de la façon la plus spécifique possible le traitement opérer par la procédure.

Automatisme (déclencheur ou trigger) :

- ◊ Comme pour Procédure.

Relation (table, vue) :

- ◊ Comme pour Type.

Schéma :

- ◊ Comme pour Type.

Base de données :

- ◊ Comme pour Type.

Variable (au sein d'une routine) :

- ◊ Comme pour Attribut.
- ◊ Dans le cas des paramètres formels, il est fortement recommandé de les préfixer par « i_ » afin de les distinguer facilement des attributs et fonctions utilisés dans le corps de la routine.

Contrainte :

- ◊ Un nom de contrainte est composé d'une racine et d'un complément spécifique, la racine et le complément étant joints par un tiret bas. La racine est l'identificateur de l'entité principale relativement à laquelle la contrainte est définie (type, domaine, attribut, table, schéma, base de

données). Les compléments spécifiques des contraintes générales (`check`) sont libres, mais ceux des contraintes de clé sont prescrits[5] :

- `cc0` : clé candidate primaire (`primary key`) ;
- `cc1`, ..., `cc9` : clés candidates secondaires (`unique`) ;
- `cr0`, ..., `cr9` : clés référentielles (`foreign key`).

Longueur des identificateurs

Ainsi formulées, les règles permettent de composer commodément des identificateurs significatifs. Le problème qui demeure cependant est celui de la longueur significative maximale des identificateurs (si le dialecte se conforme à la norme, la longueur des identificateurs n'est pas bornée a priori, seule la longueur significative peut l'être sans être moindre que 31). Ce qui normalement ne devrait poser un problème effectif que pour les identificateurs engendrés automatiquement à partir d'autres identificateurs (par un post-processeur, par exemple). Je suggère que cette contrainte soit traitée en suggérant des limites aux classes d'identificateurs pouvant servir de racines aux identificateurs générés automatiquement soit, typiquement, les identificateurs de types et de relvars (tables et vues). Si on compte 12 caractères de suffixe, cela en laisse 51 pour la racine en PostgreSQL et 18 pour les dialectes plus chiches, voir annexe B.

2.4 Mise en forme

2.4.1 Consignes générales

Idéalement, la mise en forme des instructions devrait suivre un standard reconnu, applicable automatiquement et mettant de l'avant la lisibilité. Si les éditeurs des principaux dialectes en proposent chacun au moins un (parfois plusieurs), d'autres proviennent des communautés de pratique[6]. Ils sont toutefois rarement adéquatement pris en charge à la fois par les ateliers (Eclipse, DataGrip, Navicat...) et par les outils d'automatisation. Par ailleurs, les applications censées les mettre en oeuvre, parfois appelées «formateur», sont plutôt défaillantes (si on les compare aux outils analogues disponibles pour des langages comme C++, Java, C#, Swift ou Go).

À défaut d'avoir trouvé jusqu'à présent un référentiel reconnu, automatisable, j'ai regroupé quelques consignes particulières ci-après.

2.4.2 Consignes particulières

En général, l'indentation doit refléter la structure syntaxique des énoncés. Il convient dès lors de n'augmenter le niveau que d'une unité à fois. En particulier, on évitera l'alignement par mots réservés et la création de colonnes, sauf dans le contexte d'une instruction `insert` lorsque les données seront présentées, à titre documentaire sous forme de tableau.

2.4.2.1 Contraintes

Emplacement des contraintes :

- ◊ Dans une table, une contrainte `not null` doit suivre la spécification du type de chacun des attributs.

[5] Dans un contexte anglophone, les codes « cc » sont remplacés par « ck » et « cr » par « fk » et, le plus souvent, «cc0» par «pk».

[6] https://www.sqlstyle.guide, https://docs.sqlfluff.com/en/stable/ et d'autres.

- ◊ Toutes les autres contraintes doivent être décrites par une instruction `constraint` lui associant un identificateur propre.
- ◊ Les contraintes doivent être regroupées à la fin de la définition de l'entité à laquelle elles se rapportent. Dans certains contextes, par exemple les contraintes référentielles en engendrées automatiquement, les contraintes peuvent être regroupées à la fin du module (du schéma ou du fichier) et introduite par une instruction `alter`.

2.4.2.2 Commentaires

Dans la mesure du possible, tous les commentaires relatifs à un objet dont la définition est conservée dans le catalogue de la base de données doivent être consignés à l'aide de l'instruction `comment on` afin d'y être conservés également. Par ailleurs, on évitera de mettre les commentaires en double, les commentaires lexicaux étant réservés aux informations et annotations ne devant pas être conservées dans le catalogue.

⚠ Le choix des délimiteurs de textes et la présentation des commentaires doivent être examinés à la lumière du format sous lequel ils sont conservés dans le catalogue et utilisés par les outils. Le MAD poursuit ses recherches et son étude des outils et ateliers de développement afin de déterminer la mise en forme à privilégier et corolairement le choix de la structure du commentaire lui-même (texte unique ou concaténation de textes, choix des délimiteurs, etc.).

Il pourrait en découler que des règles complémentaires seront prescrites à l'avenir.

2.4.2.3 CREATE TABLE

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

Exemple (à corriger)

```
create table "Allen_op_doc" (
  "sym" Text not null,
  "fig_g" Text not null,
  "fig_d" Text not null,
  "nom_fr" Text not null,
  "nom_en" Text not null,
  constraint "Allen_op_doc_cp" primary key ("sym"),
  constraint "Allen_op_doc_cs0" unique ("nom_fr"),
  constraint "Allen_op_doc_cs1" unique ("nom_en"),
  constraint "Allen_op_doc_cr0" foreign key ("sym") references "Allen_op_def"
  );
comment on table "Allen_op_doc" is $$
Documentation de l'opérateur "sym" comprenant la figure gauche ("fig_g") et
la droite ("fig_d"), le nom français ("nom_fr") et l'anglais ("nom_en").
$$;
```

2.4.2.4 INSERT

L'instruction `insert` doit comprendre la liste des attributs (même si elle est facultative en SQL) puis la liste des valeurs. Cette contrainte prend son sens dans un contexte d'évolution, où des attributs peuvent être ajoutés, retranchés ou permutés.

L'instruction `insert` a souvent une grande valeur documentaire et en présenter les données sous forme de tableau en facilite la lecture. Pour éviter qu'un éventuel formateur cochonne votre travail, il est suggéré d'utiliser une commande de suspension de mise en forme comme dans l'exemple qui suit.

Exemple (à corriger)

```
-- @formatter:off
insert into "Allen_op_doc"
  ("sym", "fig_g",      "fig_d",       "nom_fr",                          "nom_en"    )
values
  ('<',  '|---|oooooo', 'oooooo|---|', 'antériorité stricte',             'before'    ),
  ('m',  'o|---|ooooo', 'ooooo|---|', 'adjacence antérieure',             'meets'     ),
  ('o',  'oo|---|oooo', 'oooo|---|o', 'chevauchement antérieur strict',   'overlaps'  ),
  ('s',  'ooo|--|oooo', 'ooo|---|ooo', 'commencement strict',             'starts'    ),
  ('d',  'oooo|-|oooo', 'ooo|---|ooo', 'inclusion bi-stricte',            'during'    ),
  ('f',  'oooo|--|ooo', 'ooo|---|ooo', 'achèvement strict',               'finishes'  ),
  ('=',  'ooo|---|ooo', 'ooo|---|ooo', 'égalité',                         'equals'    ),
  ('fi', 'ooo|---|ooo', 'oooo|--|ooo', 'achèvement strict-1',             'finishes-1'),
  ('di', 'ooo|---|ooo', 'oooo|-|oooo', 'inclusion bi-stricte-1',          'during-1'  ),
  ('si', 'ooo|---|ooo', 'ooo|--|oooo', 'commencement strict-1',           'starts-1'  ),
  ('oi', 'ooooo|---|o', 'oo|---|oooo', 'chevauchement antérieur strict-1', 'overlaps-1'),
  ('mi', 'oooooo|---|', 'o|---|ooooo', 'adjacence antérieure-1',          'meets-1'   ),
  ('>',  'oooooo|---|', '|---|oooooo', 'antériorité stricte-1',           'before-1'  );
-- @formatter:on
```

2.4.2.5 *SELECT, DELETE, UPDATE…*

Respecter la structure syntaxique des instructions et modéliser les emboitements par l'indentation.

Éviter les décalages abusifs.

Exemples (à corriger)

```
WITH
  nbPoste AS ( -- nombre de postes occupés par un artisan par film auquel il a participé
    SELECT artisan_id, film_id, count(poste_id) AS nbPostes
    FROM Participation
    GROUP BY artisan_id, film_id),
  X AS ( -- les artisans ayant occupé plus d'un poste dans au moins un film
    SELECT DISTINCT N1.artisan_id
    FROM nbPoste AS N1
    WHERE N1.nbPostes > 1
      AND NOT EXISTS (
        SELECT 1
        FROM NbPoste AS N2
        WHERE N1.artisan_id = N2.artisan_id AND N2.nbPostes = 1))
SELECT nom, prenom, artisan_id
FROM Artisan NATURAL JOIN X
ORDER BY nom ASC, prenom ASC, artisan_id DESC ;
```

```
CREATE VIEW NbFilmsParDecennie (decennie, fin, nbFilms) AS
  -- La décennie est calculée de (xxx0 à xxx9), ce qui est l'usage dans l'industrie
  -- cinématographique et non selon la règle des calendriers juliens, grégoriens ou ISO
  -- (xxx1 à xxx0).
  WITH
    F (film_id, decennie) AS (
      SELECT film_id, (parution/10)*10 AS decennie
      FROM Film
      )
  SELECT decennie, decennie+9 AS fin, COUNT(film_id) AS nbFilms
  FROM F
  GROUP BY decennie ;


create or replace function CoProdInter (_f Film_id) returns boolean
  stable
return
  exists (
    select 1
    from Production natural join Studio
    where film_id = _f
    group by localisation
    having count(*) > 1
  );
```

2.4.2.6 Dénomination des routines du CRUD

Lors du développement d'une base de données, plusieurs entités (relvars, tables) sont accompagnées d'un ensemble de routines en permettant la gestion, communément appelées collectivement CRUD (create, retrieve, update, delete) ou ÉMIR[7] (évaluation, modification, insertion, retrait). Attendu leur grand nombre, mais aussi l'uniformité de leur comportement, fin de faciliter le repérage et la systématisation de la sémantique, il est souvent convenu de catégoriser les routines d'un ÉMIR et de leur donner des règles de dénomination. Voici les nôtres, attendu une entité <e>.

Système ÉMIR

La dénomination des routines suit les règles suivantes :

- Évaluation, <e>_eva<car>
  - une évaluation est toujours une fonction puisque son objet est de retourner une valeur ;
  - la distinction entre stricte et non stricte ne peut être applicable.
- Modification, <e>_mod<car><statut><strict> :
  - ne peut être une fonction, puisque l'état est susceptible d'être modifié, donc procédure ou procédure avec retour de statut ;
  - stricte ou non stricte.

[7] Les deux systèmes ne sont pas équivalents, le système CRUD est un cas particulier du système EMIR : (a) l'évaluation d'une expression relationnelle est plus large que la « recherche » d'un tuple dans une table ; (b) l'insertion est une dénomination plus exacte que « crreation » puisque le tuple est non seulement créé, mais ajouté à la relation (en fait, l'opération aurait dû s'appeler «ajouter » plutôt que « insérer »).

- Insertion, <e>_ins<car><statut><strict> :
  - ne peut être une fonction, puisque l'état est susceptible d'être modifié, donc procédure ou procédure avec retour de statut ;
  - stricte ou non stricte ; .
- Retrait, <e>_ret<car><statut>
  - ne peut être une fonction, puisque l'état est susceptible d'être modifié, donc procédure ou procédure avec retour de statut ;
  - la distinction entre stricte et non stricte ne peut être applicable.

Où les suffixes complémentaires sont définis comme suit :

Le suffixe <car> est la caractéristique de l'opération permettant de différencier le cas général des cas particuliers. Le choix du mot désignant la caractéristique doit suivre les règles préalablement établies. Le cas général doit-il être désigné par l'absence de caractéristique pour un mot prescrit ? Voici les deux options :

- <car> ::= « _gen » | (« _ » mot)
- <car> ::= (« _ » mot) ?

Le suffixe <statut> est la caractéristique de l'opération permettant de différencier le cas général des cas particuliers. Le choix du mot désignant la caractéristique doit suivre les règles préalablement établies. Le cas général doit-il être désigné par l'absence de caractéristique pour un mot prescrit ? Voici les deux options :

- <statut> ::= « _sst » | « _ast » -- sans statut ou avec statut
- <statut> ::= (« _ast ») ?

Le suffixe <strict> est la caractéristique de l'opération permettant de différencier le cas général des cas particuliers. Le choix du mot désignant la caractéristique doit suivre les règles préalablement établies. Le cas général doit-il être désigné par l'absence de caractéristique pour un mot prescrit ? Voici les deux options :

- <strict> ::= « _exs » | « _exn » -- exigences strictes ou exigences non strictes
- <strict> ::= (« _exn ») ?

Système CRUD

Si un système CRUD est privilégié (donc sans évaluation généralement, mais seulement avec des «recherches»), les changements suivants sont de mise (à noter, pour retrieve, l'utilisation de « sel » pour « select », attendu l'instruction correspondante en SQL) :

Tableau 1 - Équivalences ÉMIR - CRUD

cas ÉMIR	suffixe ÉMIR	cas CRUD	suffixe CRUD
creation	ins	creation	cre
retrieve	eva	retrieve	sel
update	mod	update	upd
delete	ret	delete	del
cas général	gen	cas général	gen
exigences strictes	exs	strict requirements	sre
exigences non strictes	exn	non strict requirements	nre
sans statut	sst	no status	nst
avec statut	ast	with status	wst

⚠ Le MAD poursuit ses recherches et son étude des besoins issus du développement et de l'évolution des bases de données afin de déterminer de meilleures règles.

En conséquence, un changement aux présentes règles pourrait intervenir lors d'une prochaine version du présent standard.

A Autres exemples

A.1 E1 (à corriger)

```
CREATE TABLE Sejour
  (
    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_ce0     FOREIGN KEY (idPatient)
      REFERENCES Patient,
    CONSTRAINT Sejour_ce1     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)
  );
```

A.2 E2 (à corriger)

```
CREATE TABLE "ProduitPhysique"
  (
    "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_ce0" 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'))
  );
```

B Forme non délimitée (régulière) des identificateurs

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

```
Grammar Rules
    <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	63**	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 catalogued 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 catalogued 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.
```

(**) La longueur en PostgreSQL est exprimée en octets, plutôt qu'en caractères. De plus, le code source de PostgreSQL étant ouvert, une installation peut être faite en utilisant une valeur différente :

> « SQL identifiers and key words must begin with a letter (a-z, *but also letters with diacritical marks and non-Latin letters*) or an underscore (_). Subsequent characters in an identifier or key word can be letters, underscores, digits (0-9), or dollar signs ($). Note that dollar signs are not allowed in identifiers according to the letter of the SQL standard, so their use might render applications less portable. The SQL standard will not define a key word that contains digits or starts or ends with an underscore, so identifiers of this form are safe against possible conflict with future extensions of the standard. The system uses no more than NAMEDATALEN-1 bytes of an identifier; longer names can be written in commands, but they will be truncated. By default, NAMEDATALEN is 64 so the maximum identifier length is 63 bytes. If this limit is problematic, it can be raised by changing the NAMEDATALEN constant in src/include/pg_config_manual.h. »[8]

Recommandation

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 (ISO, Oracle).

[8] Voir https://www.postgresql.org/docs/15/sql-syntax-lexical.html#SQL-SYNTAX-IDENTIFIERS, section 4.1.1

◊ 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 est le tiret bas ('_').

C Cartouches

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

- ◊ Fichier : le nom[9] du fichier qui contient le composant (au début du cartouche initial puis répété à la fin du cartouche final) ;
- ◊ Produit :...
- ◊ Résumé :...
- ◊ Projet :...
- ◊ Responsable : le courriel de la personne responsable du composant (c'est-à-dire celle à laquelle il faut s'adresser en premier lieu afin d'obtenir des informations complémentaires) ;
- ◊ Version : indicateur progressif mis à jour incrémentalement à chaque modification du composant, on peut utiliser la date de la dernière modification, suffixée d'une lettre (a-z) afin de permettre plusieurs versions un même jour ; à défaut d'utiliser une notation incluant la date, il faudra ajouter un champ donnant cette information.
- ◊ Statut :...
- ◊ 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 ;

Ce premier commentaire peut être complété par des identifiants produits par des logiciels de gestion de versions afin d'identifier uniquement la version[10] ; ils sont alors regroupés sous le champ « Référence unique ».

Le deuxième cartouche, appelé *cartouche principal*, a pour objectif de donner deux descriptions du fichier : une courte et une longue.

D'autres cartouches peuvent être ajoutés par la suite afin notamment de présenter des aspects importants de la conception *in situ*.

Le *cartouche final*, placé à la fin du fichier, a pour objectif de rassembler les informations annexes dont l'emplacement relatif du fichier (en accord avec celui du cartouche initial), la liste des contributeurs (adresse de courriel), la propriété intellectuelle, la liste des tâches projetées et l'historique de l'évolution du composant.

[9] L'emplacement est déterminé en fonction de la politique des gestions des artéfacts.

[10] Une variété consiérable de moyens différents sont utilisés à cet égard : UID, GUID, UUID, notation pointée, notation dérivée de la date (et l'heure) de la dernière modification contributive...

Exemple de cartouche initial (à corriger)

```
/*
================================================================================ A
Evaluation_def.sql
-------------------------------------------------------------------------------- A
Produit : Evaluation
Résumé : Exemple didactique référant à l'évaluation d'activités universitaires
Projet : IFT187_2022-1
Responsable : Luc.Lavoie@USherbrooke.ca
Version : 2021-09-10a
Statut : en vigueur
Encodage : UTF-8 sans BOM, fin de ligne Unix (LF)
Plateforme : PostgreSQL 9.4.4 à 10.4
================================================================================ A
*/
```

La forme et la signification du numéro de version dépendent de l'environnement de gestion des sources.

La forme et la signification du statut dépendent de l'environnement de développement. Par exemple : en vigueur, en test, en revue, en développement.

Exemple de cartouche final (à corriger)

```
/*
================================================================================ Z
Evaluation_fin.sql
-------------------------------------------------------------------------------- Z
.Contributeurs
  (CK01) Christina.Khnaisser@USherbrooke.ca
  (LL01) Luc.Lavoie@USherbrooke.ca

.Droits, licences et adresses
  Copyright 2016-2022, GRIIS
  Le code est sous licence
    LILIQ-R 1.1 (https://forge.gouv.qc.ca/licence/liliq-v1-1/.
  La documentation est sous licence
    CC-BY 4.0 (https://creativecommons.org/licenses/by/4.0/.

  GRIIS (Groupe de recherche interdisciplinaire en informatique de la santé)
  Faculté des sciences et Faculté de médecine et sciences de la santé
  Université de Sherbrooke (Québec) J1K 2R1 CANADA
  http://griis.ca

.Tâches projetées
  2013-09-09 (LL01) : Compléter le schéma
  * Compléter les contraintes.

.Tâches réalisées
  2013-09-03 (LL01) : Création
  * CREATE TABLE Activite, TypeEvaluation, Etudiant, Resultat.
  2015-08-28 (CK01) : Harmonisation
  * avec les modules BD012 et BD100 (voir [mod]);
  * avec les modifications apportées au standard BD190 (voir [mod]).

.Références
  [mod] http://info.usherbrooke.ca/llavoie/enseignement/Modules/
================================================================================ Z
*/
```

Les licences varient selon la situation.

Ἀκαδήμεια et Μῆτις

Le code sous licence LILIQ-R 1.1 (https://forge.gouv.qc.ca/licence/liliq-v1-1/).

La documentation est sous licence CC-BY 4.0 (https://creativecommons.org/licenses/by/4.0/) .

GRIIS et MAD

Le code sous licence LILIQ-R 1.1 (https://forge.gouv.qc.ca/licence/liliq-v1-1/).

La documentation est sous licence CC-BY 4.0 (https://creativecommons.org/licenses/by/4.0/) .

CENSASS

À déterminer avec le partenaire.

D Textes divers à récupérer

D.1 Clés, codes, noms et annotations (CCNA)

Les entités pertinentes découlant du domaine d'application sont nécessairement représentées dans le modèle conceptuel et corolairement dans le modèle logique. Le modèle logique en comprend souvent plusieurs autres, invisibles aux utilisateurs du modèle conceptuel ; en conséquence, nous ne les prendrons pas en considération dans la présente discussion.

Définitions relatives aux clés, codes, noms et annotations

Une clé est un ensemble de valeurs permettant d'identifier uniquement (c'est un pléonasme volontaire) une entité relativement à un ensemble d'entités (appelé ensemble de référence) : une classe[11], une relation, une base de données, la Terre, voire l'Univers.

Sauf mention explicite, pour la suite de la présente section, le mot « clé » sera entendu au sens strict[12], à savoir qu'aucun sous-ensemble d'une clé n'est lui-même une clé.

Le choix des attributs formant une clé d'une entité est primordial pour assurer une bonne adéquation au réel et une bonne communication avec les utilisateurs. Il faut également prendre en compte dans ce choix les besoins applicatifs et les contraintes de mise en oeuvre. Ces besoins et ces contraintes conduisent donc parfois à définir plus d'une clé pour une même entité, même si, en théorie, une seule est toujours suffisante.

Par ailleurs, en pratique, les clés ne sont pas toujours suffisantes lorsqu'on prend en compte les interactions avec les utilisateurs : l'identification d'une entité par les seules clés définies est parfois réductrice, certains noms (qui ne sont pas des clés) ont souvent une meilleure valeur de représentation, etc. Il est alors utile, sinon nécessaire, d'ajouter d'autres attributs permettant de donner un éclairage particulier ou complémentaire. Il en découle fréquemment une confusion en ce qui est vraiment une clé de ce qui ne l'est pas. Pour aider à réduire cette confusion, une nomenclature, basée sur les identificateurs associés aux attributs, est proposée après l'interprétation relationnelle suivante.

Interprétation relationnelle

Dans la théorie relationnelle, une entité est représentée par un tuple et un ensemble de référence par une relation. La relation[13] est définie par l'ensemble des attributs dont se composent ses tuples et par une clé qui détermine (identifie) ses tuples. La clé de relation est formée d'un sous-ensemble non vide d'attributs de la relation. La clé d'un tuple est déterminée par les valeurs des attributs de la clé de la relation correspondante.

Une relation peut définir plusieurs clés, mais elle en a au moins une. Il est d'usage d'en désigner une comme étant primaire et les autres comme étant secondaires.

Attributs caractéristiques

Aux fins de la discussion, voici les classes d'attributs pris en considération.

---

[11] Le mot classe est ici entendu en son sens général (ensemble d'entités ayant des attributs communs) et non au sens spécifique que lui donne les modèles orientés objets.

[12] Certains auteurs désignent la clé stricte sous le nom de « clé candidate ».

[13] Plus exactement, le type de la reation.

- ◊ *identifiant* : attribut constituant à lui seul une clé, exclusivement à l'usage interne du logiciel; la valeur de la clé (celle associée à l'identifiant) est donc artificielle et engendrée par le SGBD ; l'automatisation de la génération des identifiants simplifie la gestion et minimise les risques de collision.
- ◊ *code* : attribut constituant à lui seul une clé, à usage interne ou externe – les plus souvent les deux ; l'utilisation d'un code facilite la référence à des entités spécifiques au sein des requêtes et des routines ; les valeurs peuvent être choisies pour leur caractère mnémonique présumé ou reprises d'un codage terminologique établi par ailleurs (norme ou standard, par exemple ISO 639 pour les codes de pays).
- ◊ *nom normatif* : attribut dont la valeur est un nom unique établi en regard d'une autorité ou d'une norme établie ; en conséquence, les règles le régissant échappent aux applications et au modèle.
- ◊ *nom usuel* : nom consacré par l'usage et suffisant pour identifier l'entité relativement à un contexte convenu ; en conséquence, les règles le régissant échappent aux applications et au modèle.
- ◊ *nom abrégé* : abréviation du nom usuel, consacrée par l'usage et suffisante pour identifier l'entité relativement à un contexte convenu (possiblement plus restreint que celui du nom usuel lui-même) ; des règles d'abréviation peuvent être convenues au niveau du modèle (mais ne devrait pas relever d'une application).
- ◊ *nom relatif* : nom unique relativement à celui d'autres entités partageant la même entité de référence (souvent un « parent » dans une hiérarchie de référence) ; des règles de relativisation peuvent être convenues au niveau du modèle (mais ne devrait pas relever d'une application).
- ◊ *annotation* : information relative à l'entité ; elle peut être formelle ou non.
  - o L'*annotation formelle* est exprimée à l'aide d'un langage (formel) précis, soutenu par un modèle sémantique (normalement dérivé de la logique afin de permettre l'inférence). Une telle annotation est réputée utilisable à l'aide d'outils informatiques appropriés, notamment pour soutenir des raisonnements logiques.
  - o L'*annotation non formelle* (appelée également *annotation textuelle*) est généralement exprimée dans une langue ou un dialecte humain, et prioritairement destiné à l'affichage. Nous les considèrerons ici hors de portée d'un traitement informatique permettant de soutenir des raisonnements logiques.

Propriétés

Quelques propriétés importantes de ces attributs en sont synthétisées dans le tableau ci-après.

Tableau 2 - Propriétés des attributs génériques

terme	clé**	portée	sujet	identificateur	
			à traduction	explicite	implicite
identifiant	oui	int	non	<e>_id	<e>
code	oui	int et ext	non	<e>_code	<e>++
nom normatif	oui	ext	non*	nom_normatif	
nom usuel	?	ext	oui	nom_usuel	
nom abrégé	?	ext	oui	nom_abrege	
nom relatif	non+	ext	oui	nom_relatif	
annotation formelle+++	non	int et ext	oui	annotation	
annotation textuelle+++	non	ext	oui	description	

Légende

<e> le nom du type d'entités considérées.

? selon le domaine d'application et le contexte.

+ e nom relatif est toutefois unique relativement aux autres noms relatifs ayant le même parent dans la hiérarchie de référence.

++ s'il y a aussi un identifiant, ce dernier conserve la forme «<e>» et le code est alors désigné par «<e>_code».

+++ en général, une convention ou un attribut complémentaire permet de déterminer la langue, le dialecte ou le langage utilisé.

* l'objectif du nom normatifs est de garantir l'uncité de l'identification dans le contexte de référence, en ce sens, il ne devrait pas être traduit – toutefois, eu égard aux législations, cela pourrait être requis, il faut alors révoir plusieurs noms normatifs *in situ* ou préférablement dans une table auxiliaire.

** interne (int) destiné à être utilisé par le logiciel (contrôle d'intégrité, programmation, etc.) et externe (ext) destiné à être utilisé par des utilisateurs du logiciel (affichage, saisie, etc.).

⚠ Les discussions entreprises jusqu'à présent n'on pas permis des bine départager lea avantages et les inconvénienrs des des formes explicites et implicites. Pour le moment les deux formes sont donc autérisées, dans la mesure où un même produit se contonne à une seul des deux formes.

En conséquence, un changement aux présentes règles pourrait intervenir lors d'une prochaine version du présent standard.

D.2 Communauté de portée des identificateurs de types (domaines) et de tables (vues)

Quels sont les problèmes pratiques occasionnés par la restriction suivante (conforme à ISO 9075 :2016 et prise en charge par PostgreSQL) ?

> *A relation (table, view) has an associated type of the same name, so you must use a name that doesn't conflict with any existing type.*

D.3 Communauté de portée des identificateurs de vues, de tables et de contraintes

Les identificateurs de contraintes sont dans la portée des identificateurs de tables et de vues, même si ces contraintes ne sont pas relatives à une table ni une vue (par exemple, relativement à un domaine). L'usage qui consiste à utiliser l'identificateur de l'objet sur lequel porte la contrainte comme préfixe du nom de la contrainte afin de le localiser n'est conséquemment pas toujours applicable s'il y a un type (ou un domaine) de même nom qu'une table ou une vue.

D.4 Mise en oeuvre des assertions

Absence de create assertion => create trigger.

Problème d'identification, car il se pourrait qu'une même assertion engendre plus d'un trigger (typiquement un par table contribuant à l'expression de l'assertion, voir plusieurs dans les cas complexes).

D.5 Synthèses des suffixes utilisés pour la composition des identificateurs

...

D.6 Annotations de code

Voici les annotations que j'utilise le plus couramment :

- la note : NOTE ;
- la question : QUESTION et sa réponse ANSWER ;
- les demandes de modification (en ordre croissant d'urgence) : TODEPRECATE, TOIMPROVE, TORESTORE, TOPROMOTE, TODO, TOFIX ;
- les statuts : DEPRECATED, IMPROVED, RESTORED, PROMOTED, DONE, PATCHED, FIXED.

La note désigne une explication qui a pour vocation de demeurer dans le code, suite à la résolution d'une question ou d'une demande de modification.

La question est une demande d'éclaircissement ou d'avis, adressée aux développeurs (programmeurs, concepteurs, réviseurs, spécificateurs, responsable des essais...) susceptibles de prendre connaissance du code. La réponse qui suit (ANSWER) a pour vocation de demeurer telle jusqu'à son approbation. Ensuite, les informations qui y sont consignées peuvent faire l'objet d'une note.

Les demandes appellent une modification du code à plus ou moins brève échéance. Lorsque la demande est traitée, le temps de la revue, son texte est conservé (ou augmenté) et le statut de la demande est modifié en conséquence :

- ◎ —> TODEPRECATE —> DEPRECATED —> ◉ ;
  |
  —> TORESTORE —> RESTORED —> ◉ ;
- ◎ —> TORESTORE —> RESTORED —> ◉ ;
- ◎ —> TOIMPROVE —> IMPROVED —> ◉ ;
- ◎ —> TOPROMOTE —> PROMOTED —> ◉ ;
- ◎ —> TODO —> DONE —> ◉ ;

- ◊ ◎ —> TOFIX —> PATCHED —> FIXED —> ◉.

Suite à des essais prescrits concluants suivie de la période de *latence* prescrite (établie dans la fiche du produit), tout statut doit (sauf dérogation) :

- ◊ soit disparaitre de la source ;
- ◊ soit y être remplacé par une NOTE.

Une source contributive à un logiciel en cours d'essai de livraison ne doit comporter (sauf dérogation) aucun TOFIX.

Une source contributive à un logiciel en cours d'exploitation ne doit comporter (sauf dérogation) aucun TODO ni aucun TOFIX.

Encodage et présentation des annotations

Ces annotations peuvent être définies ainsi dans DataGrip :

```
\b(NOTE)\b.* [vert]
\b(QUESTION)\b.* [jaune]
\b(ANSWERED)\b.*. [jaune]
\b(TODEPRECATE)\b.* [bleu]
\b(DEPRECATED)\b.* [bleu]
\b(TOIMPROVE)\b.* [bleu]
\b(IMPROVED)\b.* [bleu]
\b(TORESTORE)\b.* [bleu]
\b(RESTORED)\b.* [bleu]
\b(TOPROMOTE)\b.* [bleu]
\b(PROMOTED)\b.* [bleu]
\b(DONE)\b.* [bleu]
\b(TOFIX)\b.* [rouge]
\b(PATCHED)\b.* [rouge]
\b(FIXED)\b.* [rouge]
```

💡 Il est bien sûr possible de programmer un atelier afin qu'il reconnaisse les anciennes et les nouvelles formes !

Pour la mise en forme des annotations, il suggéré de procéder comme suit :

```
-- TODO 2021-10-31 (LL01) Court titre.
--  * élément 1
--  …
--  * élément n
```

Lorsqu'une annotation franchit une étape, la forme suggérée est la suivante :

```
-- DONE 2022-02-03 (CK01), 2021-10-31 (LL01) Court titre.
--  * élément 1
--  …
--  * élément n
--  Correction
--  * élément 1
--  …
--  * élément n
```

Glossaire

BD
: Base de données.

dialecte
: [sens général] variante d'un langage formel ;
[sens particulier] variante du langage SQL (défini par la norme ISO 9075 :2016) proposé par un éditeur de SGBDR. Parmi les dialectes les plus répandus, notons ceux d'Oracle, d'IBM (DB2), de Microsoft et SyBase (Transact-SQL), de SAP (Hanna) et de PostgreSQL.

identifiant
: une valeur associée à une entité permettant de la distinguer de toute autre entité de même type ; les valeurs utilisées aux fins d'identification d'un type d'entités sont elles-mêmes généralement contraintes par un type choisi pour cet usage et explicitement associé au type des entités ciblées ; par extension, la classe, le type ou un attribut associé à de telles valeurs.

identificateur
: au sein d'un texte conforme aux règles d'un langage formel (par exemple, un langage de programmation), un identificateur est un identifiant associé à une entité définie par une partie dudit texte. L'unicité de l'entité référée par l'identifiant requière parfois l'utilisation d'un contexte (par exemple, en SQL, la préfixation d'un identificateur d'attribut par un identificateur de table, lui-même préfixable par un identificateur du schéma, lui-même préfixable par un identificateur de la base de données :

```
identificateur_de_base_de_données ::= identificateur
identificateur_de_schéma ::= (identificateur_base_de_données '.') ? identificateur
identificateur_de_table ::= (identificateur_de-schéma '.') ? identificateur
identificateur_d_attribut ::= (identificateur_de_table '.') ? identificateur
```

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 la définition 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*.