Part 5

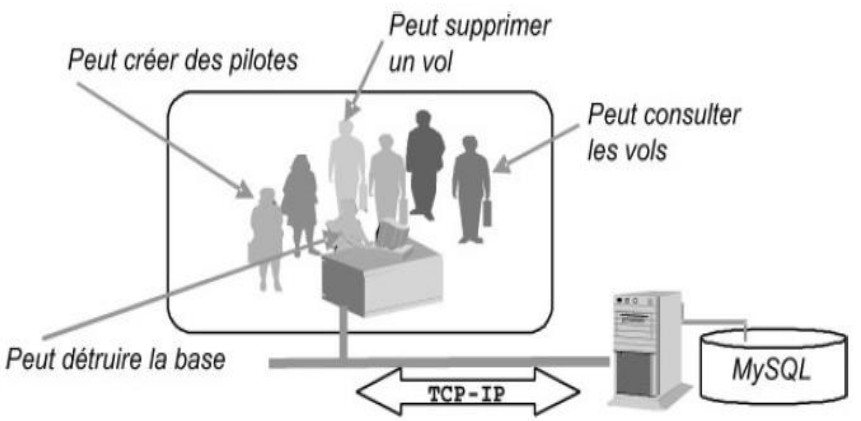
NOTIONS AVANCÉES

Contrôle d'accès

Sécurité des bases de données

- **Confidentialité**
  - Seules les personnes autorisées ont accès aux ressources de la BD.
  - Ressources BD: données stockées dans la base ainsi que traitements activables sur ces données.
- **Intégrité**
  - Les ressources de la BD ne doivent pas être corrompues.
  - Les données stockées et échangées doivent être protégées de toutes modifications illicites (destruction, altération, substitution, …).
- **Disponibilité**
  - L'accès aux ressources de la BD est garanti de façon permanente.

L'aspect multi-utilisateur des BD



Les types d'utilisateurs

- **Le DBA (*DataBase Administrator*)**: Il en existe au moins un. Une base importante peut en regrouper plusieurs qui se partagent les tâches suivantes:
  - installation et mises à jour de la base et des outils éventuels ;
  - gestion de l'espace disque et des espaces pour les données ;
  - gestion des utilisateurs et de leurs objets (s'ils ne les gèrent pas eux-mêmes) ;
  - optimisation des performances ;
  - sauvegardes, restaurations et archivages ;
  - contact avec le support technique.
- **L'administrateur réseau** (qui peut être le DBA) se charge de la configuration des couches client pour les accès distants.

Les types d'utilisateurs

- **Les développeurs** qui conçoivent et mettent à jour la base. Ils peuvent aussi agir sur leurs objets (création et modification des tables, index, séquences, etc.). Ils transmettent au DBA leurs demandes spécifiques (stockage, optimisation, sécurité).
- **Les administrateurs d'application** qui gèrent les données manipulées par la ou les applications. Pour les petites et les moyennes bases, le DBA joue ce rôle.
- **Les utilisateurs** qui se connectent et interagissent avec la base à travers les applications ou à l'aide d'outils (interrogations pour la génération de rapports, ajouts, modifications ou suppressions d'enregistrements).

Contrôle d'accès

- Un utilisateur (user) est identifié par MySQL par **son nom** et celui de **la machine** à partir de laquelle il se connecte.
- Il ne faut autoriser à un utilisateur à effectuer une **opération** sur un **objet** que s'il en a obtenu le **droit** (ou le privilège).
- Dans un SGBD relationnel :
  - les objets à protéger sont les tables, les vues, les procédure stockés, …
  - les opérations sont SELECT, INSERT, UPDATE ou DELETE.
- L'attribution des droits peut être :
  - **centralisé** : l'administrateur a tous les droits sur tous les objets de la BD et peut transmettre certains de ces droits à d'autres utilisateurs.
  - **décentralisée** : l'utilisateur qui crée un objet a tous les droits sur cet objet et peut les transmettre en totalité ou en partie à d'autres utilisateurs.

Création d'un utilisateur

- Pour pouvoir créer un utilisateur, vous devez posséder le privilège **CREATE USER** ou **INSERT** sur la base système mysql
- La syntaxe de création d'un utilisateur est la suivante :
  CREATE USER *utilisateur*
  [IDENTIFIED BY [PASSWORD] '*motdePasse*']
  [,*utilisateur2* [IDENTIFIED BY [PASSWORD] '*motdePasse2*'] ...];
- Exemple:
  CREATE USER soutou@localhost
  IDENTIFIED BY 'ensaj';

  soutou est déclaré « utilisateur à accès local », il devra se connecter à l'aide de son mot de passe.
- Par défaut, les utilisateurs, une fois créés, n'ont aucun droit sur aucune base de données (à part en lecture écriture sur la **base test** et en lecture seule sur la base **information_schema**).

L'utilisateur root

- L'utilisateur **root** (mot de passe saisi à l'installation) est le DBA que MySQL offre.
- Il permet d'effectuer les tâches administratives en ligne de commande ou par une console graphique (créer des utilisateurs par exemple).

Liste des utilisateurs

- Pour retrouver les informations relatives à la connexion pour l'utilisateur root et les autres il faut interroger la table user de la base mysql (**mysql.user**).

  SELECT User,Host FROM **mysql.user**;

  - Le résultat de la requête précédente sera en fonction de la version, des options d'installation et des utilisateurs créés.
  - La machine désignée par « localhost » spécifie que la connexion est autorisée en local. La même adresse peut être indiquée en IP V4 (127.0.0.1) et IP V6 (:: 1).
  - L'absence de nom d'utilisateur signifie une connexion anonyme.
  - La machine désignée par « % » indique que la connexion est autorisée à partir de tout site, en supposant qu'un client MySQL est installé et qu'il est relié au serveur par TCP-IP (voir la section « Table mysql.db »).

Modification du mot de passe d'un utilisateur

- Le mot de passe d'un utilisateur peut être modifié indépendamment des privilèges dont il dispose.
- Pour changer un mot de passe jusqu'en version 5.7.5:

  SET PASSWORD FOR *'utilisateur'@'serveur'* = **PASSWORD**(*'mot_de_passe'*);
- Pour changer un mot de passe depuis la version 5.7.6 :

  ALTER USER 'utilisateur'@'serveur' **IDENTIFIED BY** *'mot_de_passe'*;

Renommer un utilisateur

- Pour pouvoir renommer un utilisateur, il faut posséder le privilège **CREATE USER** (ou le privilège UPDATE sur la base de données mysql). La syntaxe SQL est la suivante:

  RENAME USER *utilisateur* TO *nouveauNom*;
- Exemples:

Instruction SQL	Commentaire
`RENAME USER soutou@localhost TO christiansoutou@localhost;`	L'accès *soutou* en local est renommé *christiansoutou* en local.
`RENAME USER christiansoutou@localhost TO christiansoutou@194.53.227.12;`	L'accès *christiansoutou* en local est renommé *christiansoutou* en accès distant.
`RENAME USER christiansoutou@194.53.227.12 TO soutou@localhost;`	L'accès est renommé complètement.

Suppression d'un utilisateur

- Pour pouvoir supprimer un utilisateur, il faut posséder le privilège CREATE USER (ou le privilège DELETE sur la base de données mysql). La syntaxe SQL est la suivante :

  DROP USER utilisateur [,utilisateur2 ...];

  - Il faut spécifier l'accès à éliminer (user@machine). Tous les privilèges relatifs à cet accès sont détruits.
  - Si l'utilisateur est connecté dans le même temps, sa suppression ne sera effective qu'à la fin de sa (dernière) session.

- Exemple:

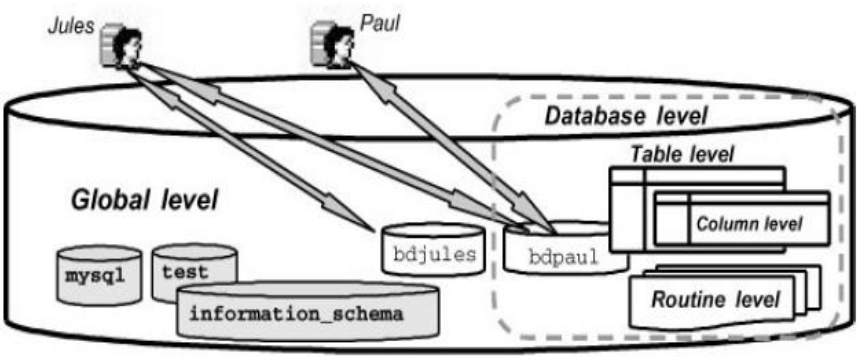
  DROP USER soutou@localhost;

Privilèges

- Un privilège (sous-entendu utilisateur) est un droit d'exécuter une certaine instruction SQL (on parle de **privilège système**), ou un droit relatif aux données des tables situées dans différentes bases (on parle de **privilège objet**).

- Les privilèges système diffèrent sensiblement d'un SGBD à un autre. Chez Oracle, il y en a plus d'une centaine, MySQL est plus modeste en n'en proposant qu'une vingtaine.

Niveaux de privilèges



Niveaux de privilèges

- **Global level**: privilèges s'appliquant à toutes les bases du serveur. Ces privilèges sont stockés dans la table **mysql.user** (exemple d'attribution d'un privilège global: **GRANT CREATE ON *.***…).
- **Database level**: privilèges s'appliquant à tous les objets d'une base de données en particulier. Ces privilèges sont stockés dans les tables **mysql.db** et **mysql.host** (exemple d'attribution d'un privilège database : **GRANT SELECT ON bdpaul.* …**).
- **Table level**: privilèges s'appliquant à la globalité d'une table d'une base de données en particulier. Ces privilèges sont stockés dans la table **mysql.tables_priv** (exemple d'attribution d'un privilège table: **GRANT INSERT ON bdpaul.Avion …**).

Niveaux de privilèges

- **Column level**: privilèges s'appliquant à une des colonnes d'une table d'une base de données en particulier. Ces privilèges sont stockés dans la table **mysql.columns_priv** (exemple d'attribution d'un privilège column: **GRANT UPDATE (nomComp) ON bdpaul.Compagnie …**).
- **Routine level**: privilèges globaux ou au niveau d'une base s'appliquant aux procédures stockées (étudiées à la section suivante). Ces privilèges sont stockés dans la table **mysql.procs_priv** de la base mysql (exemple d'attribution d'un privilège routine: **GRANT EXECUTE ON PROCEDURE bdpaul.sp1 …**).

Attribution de privilèges

- L'instruction **GRANT** permet d'attribuer un (ou plusieurs) privilège(s) à propos d'un objet à un (ou plusieurs) bénéficiaire(s).
- L'utilisateur qui exécute cette commande doit avoir reçu lui-même le droit de transmettre ces privilèges (reçu avec la directive **GRANT OPTION**).
- Dans le cas de root, aucun problème, car il a implicitement tous les droits.

Attribution de privilèges

GRANT *privilège* [(*col1* [, *col2*...])] [,*privilège2* ...]
ON [{TABLE | FUNCTION | PROCEDURE}]
{*nomTable* | * | *.* | *nomBase*.*}
TO *utilisateur* [IDENTIFIED BY [PASSWORD] '*password*']
[,*utilisateur2* ...]
[WITH [GRANT OPTION]
[MAX_QUERIES_PER_HOUR *nb*]
[MAX_UPDATES_PER_HOUR *nb2*]
[MAX_CONNECTIONS_PER_HOUR *nb3*]
[MAX_USER_CONNECTIONS *nb4*]];

- **privilège**: description du privilège (ex : SELECT, DELETE, etc.)
- **col**: précise la ou les colonnes sur lesquelles se portent les privilèges SELECT, INSERT ou UPDATE.
- **GRANT OPTION**: permet de donner le droit de retransmettre les privilèges reçus à une tierce personne.

Exemples

Instruction faite par `root`	Explication
`GRANT CREATE, DROP ON bdpaul.* TO 'Paul'@'localhost';`	Privilège système *database level* : `Paul` (en accès local) peut créer ou supprimer des tables dans la base `bdpaul`.
`GRANT INSERT, SELECT, DELETE, UPDATE(ISBN) ON bdpaul.Livre TO 'Paul'@'localhost';`	Privilège objet *table level* : `Paul` peut insérer, extraire, supprimer et modifier la colonne `ISBN` de la table `Livre` contenue dans la base `bdpaul`.
`GRANT ALTER ON bdpaul.Livre TO 'Paul'@'localhost';`	Privilège système *table level* : `Paul` peut modifier la structure ou les contraintes de la table `Livre` contenue dans la base `bdpaul`.
`GRANT SELECT(titre) ON bdpaul.Livre TO 'Jules'@'localhost' WITH GRANT OPTION;`	Privilège objet *column level* : `Jules` peut extraire seulement la colonne titre de la table `Livre` contenue dans la base `bdpaul`. Il pourra par la suite retransmettre éventuellement ce droit.
`GRANT CREATE ON *.* TO 'Jules'@'localhost';`	Privilège système *global level* : `Jules` peut créer des bases de données.
`GRANT USAGE ON bdpaul.* TO 'Jules'@'localhost' WITH MAX_QUERIES_PER_HOUR 50 MAX_UPDATES_PER_HOUR 20 MAX_CONNECTIONS_PER_HOUR 6 MAX_USER_CONNECTIONS 3;`	Privilège système *database level* : `Jules` ne peut lancer, chaque heure, que 50 `SELECT`, 20 `UPDATE`, se connecter 6 fois (dont 3 connexions simultanées) sur la base de données `bdpaul`.

Exemples

Instruction faite par `root`	Explication
`GRANT CREATE ROUTINE ON bdpaul.* TO 'Paul'@'localhost';`	Privilège système *database level* : `Paul` (en accès local) peut créer ou supprimer des sous-programmes catalogués dans la base `bdpaul`.
`GRANT ALTER ROUTINE ON PROCEDURE bdpaul.sp1 TO 'Paul'@'localhost';`	Privilège système *routine level* : `Paul` peut modifier la procédure `sp1` contenue dans la base `bdpaul`.
`GRANT EXECUTE ON PROCEDURE bdpaul.sp1 TO 'Jules'@'localhost';` `GRANT EXECUTE ON FUNCTION bdpaul.sp2 TO 'Jules'@'localhost';`	Privilèges système *routine level* : `Jules` peut exécuter la procédure `sp1` et la fonction `sp2` contenues dans la base `bdpaul`.

Voir les privilèges

- La commande **SHOW GRANTS FOR** liste les différentes instructions GRANT équivalentes à toutes les prérogatives d'un utilisateur donné:

  SHOW GRANTS FOR *utilisateur*;

- Exemple:

  SHOW GRANTS FOR 'Jules'@'localhost';

```
+-----------------------------------------------------------------------------+
| Grants for Jules@localhost                                                  |
+-----------------------------------------------------------------------------+
| GRANT CREATE ON *.* TO 'Jules'@'localhost' IDENTIFIED BY PASSWORD
 '*6AE163FB9EE8BB011EB2E87316AA5BE563A6CDB7' WITH MAX_QUERIES_PER_HOUR 50
  MAX_UPDATES_PER_HOUR 20 MAX_CONNECTIONS_PER_HOUR 6 MAX_USER_CONNECTIONS 3 |
| GRANT SELECT (titre) ON `bdpaul`.`Livre` TO 'Jules'@'localhost'
      WITH GRANT OPTION                                                     |
+-----------------------------------------------------------------------------+
```

Révocation de privilèges

- La révocation d'un ou de plusieurs privilèges est réalisée par l'instruction **REVOKE**.
- Pour pouvoir révoquer un privilège, vous devez détenir (avoir reçu) au préalable ce même privilège avec l'option **WITH GRANT OPTION**.
- Dans la syntaxe suivante, les options sont les mêmes que pour la commande **GRANT**.

REVOKE *privilège* [(*col1* [, *col2*...])] [,*privilège2* ...]
ON [{TABLE | FUNCTION | PROCEDURE}]
{*nomTable* | * | *.* | *nomBase*.*}
FROM *utilisateur* [,*utilisateur2* ...];

Exemples

Instruction faite par `root`	Explication
`REVOKE CREATE ON bdpaul.* FROM 'Paul'@'localhost';`	Privilège système *database level* : `Paul` (en accès local) ne peut plus créer de tables dans la base `bdpaul`.
`REVOKE ALTER,INSERT,UPDATE(ISBN) ON bdpaul.Livre FROM 'Paul'@'localhost';`	Privilège objet *table level* : `Paul` ne peut plus modifier la structure (ou les contraintes), insérer et modifier la colonne `ISBN` de la table `Livre` contenue dans la base `bdpaul`.
`GRANT USAGE ON bdpaul.* TO 'Jules'@'localhost' WITH MAX_QUERIES_PER_HOUR 0 MAX_UPDATES_PER_HOUR 0;`	Privilège système *database level* : `Jules` n'est plus limité en requêtes `SELECT` et `UPDATE` sur la base de données `bdpaul`. Ici c'est un `GRANT` qu'il faut faire, car il s'agit plus d'une restriction de connexion que d'une instruction SQL.

Révocation de tous les privilèges

- Il existe une instruction qui révoque tous les droits en une fois:
  REVOKE ALL PRIVILEGES, GRANT OPTION FROM *utilisateur* [, *utilisateur2…*] ;

- Exemple:
  REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'Jules'@'localhost';

Le Langage Procédural

Motivation

- SQL : langage ensembliste
  - Ensemble de requêtes distinctes
- Le développement d'application autour d'une BDR nécessite d'utiliser :
  - des variables
  - des structures de contrôle de la programmation (boucles et alternatives)

➡ Besoin d'un **langage procédural** pour lier plusieurs requêtes SQL avec des variables et dans les structures de contrôle habituelles.

Normalisation du langage

- **Oracle** propose un langage propriétaire **PL/SQL** (Acronyme : **P**rocedura**L SQL**). La plupart des SGBDR propose des L4G spécifiques, semblables à PL/SQL,
- **PostgreSQL** propose **PL/pgSQL** très proche de PL/SQL et **PL/pgPSM**,
- **MySQL** et **Mimer SQL** proposent un langage analogue dans le principe mais plus limité : **SQL/PSM** (de la norme SQL2003),
- **IBM DB2** propose un dérivé de PL/SQL : **SQL-PL**,
- **Microsoft/SQL server** et **Sybase** propose **Transact-SQL** (**T-SQL**) développé par à l'origine par Sybase,
- …

Dans le cadre de ce cours nous allons voir le langage SQL/PSM de MySQL

SQL/PSM

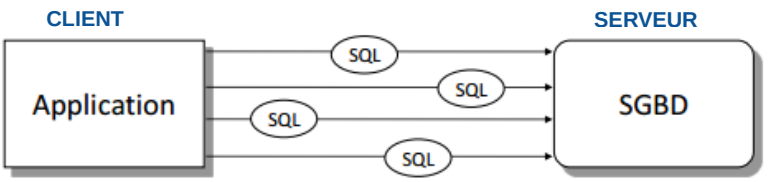
- Extension de SQL
- Langage de programmation procédural (**Persistent Stored Modules: PSM**)
- Clauses SQL intégrées dans le code procédural
- SQL/PSM est donc un langage de programmation, propre à mySQL:
  - Intègre directement les clauses SQL d'interrogation, de manipulation (généralement pas de définition des données)
  - Permet l'encapsulation des données dans du code
  - Gestion des exceptions

Environnement client-serveur

- Dans un environnement client-serveur, chaque instruction SQL donne lieu à l'envoi d'un message du client vers le serveur suivi de la réponse du serveur vers le client.
- Il est préférable de travailler avec **un sous-programme** (qui sera stocké, en fait, côté serveur) plutôt qu'avec **une suite d'instructions SQL** susceptibles d'encombrer le trafic réseau!

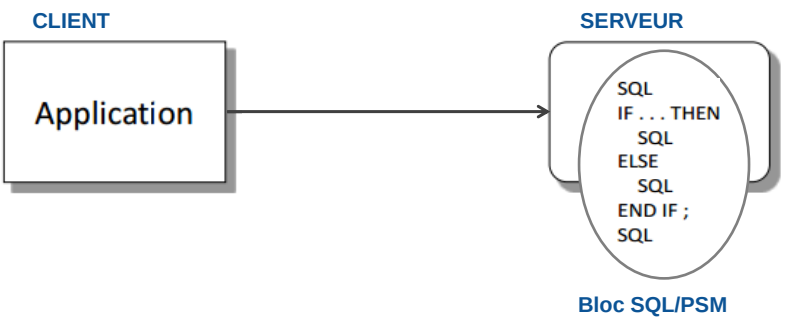
Requêtes SQL

- Chaque requête 'client' est transmise au serveur de données pour être exécutée avec retour de résultats



Bloc SQL/PSM

- Le bloc de requêtes est envoyé sur le serveur. Celui-ci exécute le bloc et renvoie 1 résultat final.

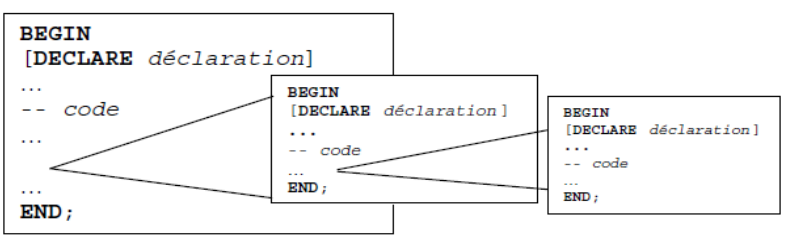


Bénéfices de SQL/PSM

- SQL/PSM regroupe les requêtes SQL en un seul bloc qui est envoyé au serveur en un seul appel
  - Amélioration des performances (moins de communications sur le réseau)
- Le bloc SQL/PSM peut être stocké sur le serveur. Seul l'appel de l'exécution du bloc et le résultat transiteront sur le réseau
- Permet de créer des bibliothèques de code réutilisable
  - Programmation de **fonctions**, **procédures**, packages

Structure d'un bloc

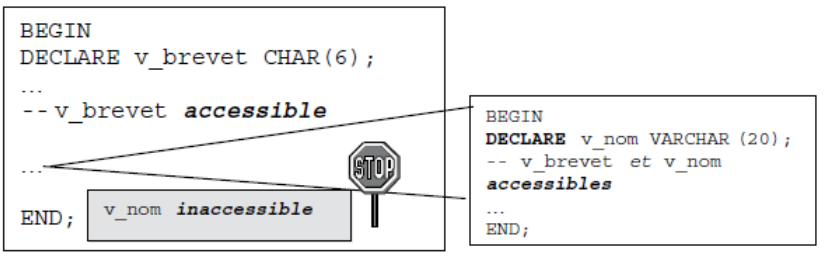
- Un bloc d'instructions est composé de :
  - **BEGIN** (section obligatoire) contient le code incluant ou non des directives SQL se terminant par le symbole « ; »
  - **DECLARE** (directive optionnelle) déclare une variable, un curseur, une exception, etc.
  - **END** ferme le bloc.



- Un bloc peut être imbriqué dans un autre bloc.

Portée des objets

- La portée d'un objet (variable, curseur ou exception) est la zone du programme qui peut y accéder.
  - Un objet déclaré dans un bloc est accessible dans les sous-blocs.
  - En revanche, un objet déclaré dans un sous-bloc n'est pas visible du bloc supérieur (principe des accolades des langages C et Java).



Casse et lisibilité

- Aucun objet manipulé par programme n'est sensible à la casse (not case sensitive).
  - Ainsi numeroBrevet et NumeroBREVET désignent le même identificateur (tout est traduit en minuscules au niveau du dictionnaire des données).

- Les règles d'écriture classiques concernant l'indentation et les espaces entre variables, mots-clés et instructions doivent être respectées dans un souci de lisibilité.

Peu lisible	C'est mieux

`IF x>y THEN SET max:=x;ELSE SET max:=y;END IF;`	`IF x>y THEN`
	`  SET max:=x;`
	`ELSE`
	`  SET max:=y;`
	`END IF;`

Identificateurs

- Un identificateur n'est pas limité en nombre de caractères.
- On ne peut utiliser ce que l'on veut :
  - Interdiction d'utiliser les mots-clés
  - Interdiction d'utiliser des caractères spéciaux (autre que « _ »)
- Commencent par :
  - Une lettre
  - Un chiffre
  - Un « _ » (underscore)

Autorisés	Interdits

`t2`
`code_brevet`
`2nombresMysql`
`_t`
	`moi&toi` (symbole « & »)
	`debit-credit` (symbole « - »)
	`on/off` (symbole « / »)
	`code brevet` (symbole espace)

Commentaires

- MySQL prend en charge deux types de commentaires :
  - **Monolignes**: commençant au symbole « **--** » et finissant à la fin de la ligne (besoin d'un espace après les tirets) ;
  - **Monolignes**: commençant au symbole «**#**» et finissant à la fin de la ligne;
  - **Multilignes**: commençant par « **/*** » et finissant par « ***/** ».

Variables

- Deux types de variables sont disponibles sous MySQL :
  - **scalaires** : recevant une seule valeur d'un type SQL (ex : colonne d'une table) ;
  - **externes** : définies dans la session et qui peuvent servir de paramètres d'entrée ou de sortie.

- **Notes:**
  - Le type tableau (array) n'est pas encore présent dans le langage de MySQL !
  - Il est impossible d'utiliser un identificateur dans une expression, s'il n'est pas déclaré au préalable.
  - Les déclarations multiples sont permises: DECLARE **i, j, k** INT;

Variables scalaires

- **Syntaxe:**
  - **DECLARE** *nomVariable1*[,*nomVariable2*...] *typeMySQL* [DEFAULT *expression*];
  - **DEFAULT** permet d'initialiser la (ou les) variable(s) – pas forcément à l'aide d'une constante.

Déclarations	Commentaires
`DECLARE v_dateNaissance DATE;`	Déclare la variable sans l'initialiser. Équivalent à `SET v_dateNaissance := NULL;`
`DECLARE v_capacite SMALLINT(4) DEFAULT 999;`	Initialise la variable à 999.
`DECLARE v_trouve BOOLEAN DEFAULT TRUE;`	Initialise la variable à vrai (1).
`DECLARE v_Dans2jours DATE DEFAULT ADDDATE(SYSDATE(),2);`	Initialise la variable à dans 2 jours.

Affectations

- Il existe plusieurs possibilités pour affecter une valeur à une variable :
  - l'affectation comme on la connaît dans les langages de programmation (**SET variable := expression**). Vous pouvez aussi utiliser le symbole « = », mais il est plus prudent de le réserver à la programmation de conditions ;
  - la directive **DEFAULT** ;
  - la directive **INTO** d'une requête (SELECT … INTO variable FROM …).

Résolution de noms

- Pour lever d'éventuelles ambiguïtés, il faut nommer toutes les variables différemment des colonnes:
  - en utilisant un **préfixe**,
  - ou en utilisant une **étiquette de bloc** (*block label*).

Préfixer les variables

```
DECLARE v_nom VARCHAR(16)
        DEFAULT 'Placide Fresnais';
…
DELETE FROM Pilote WHERE nom = v_nom ;
--ou
DELETE FROM Pilote WHERE v_nom = nom ;
```

Étiquette de bloc (préfixe pas opérationnel)

```
principal: BEGIN
 DECLARE nom VARCHAR(16)
         DEFAULT 'Placide Fresnais';
 DELETE  FROM Pilote
         WHERE principal.nom = nom;
END principal ;
```

Opérateurs

- Les opérateurs SQL vu précédemment (logiques, arithmétiques, de concaténation…) sont disponibles au sein SQL/PSM.
- Les règles de priorité sont les mêmes que dans le cas de SQL.

Code MySQL	Commentaires

`DECLARE v_compteur INT(3) DEFAULT 0;`
`DECLARE v_boolean BOOLEAN;`
`DECLARE v_nombre INT(3);`
	Trois déclarations dont une avec initialisation.

`SET v_compteur := v_compteur+1;`	Incrémentation de `v_compteur` (opérateur +)
`SET v_boolean := (v_compteur=v_nombre);`	`v_boolean` reçoit **NULL**, car la condition est fausse.
`SET v_boolean := (v_nombre IS NULL);`	`v_boolean` reçoit **TRUE** (1 en fait), car la condition est vraie.

Variables externes

- **Syntaxe**:
  SET *@var1 = expression1* [, *@var2 = expression2*] ...
- Ces variables sont dites **de session** (variables **utilisateurs** ou **globales**). Elles n'existent que durant la session.

Code MySQL

```
SET @vs_num = 'PL-4'$
SET @vs_hvol = 15$
…
BEGIN
  DECLARE v_nom    CHAR(16);
  DECLARE v_nbHVol DECIMAL(7,2);
  SELECT nom,nbHVol
       INTO v_nom, v_nbHVol FROM Pilote
       WHERE brevet = @vs_num ;
  SET v_nbHVol := v_nbHVol + @vs_hvol ;
  SELECT v_nom, v_nbHVol;
END;
```

Résultat

```
+------------------+----------+
| v_nom            | v_nbHVol |
+------------------+----------+
| Placide Fresnais |  2465.00 |
+------------------+----------+
```

Conventions recommandées

- Adoptez les conventions d'écriture suivantes pour que vos programmes MySQL soient plus facilement lisibles et maintenables :

Objet	Convention	Exemple
Variable	v_*nomVariable*	v_compteur
Constante	c_nomConstante	c_pi
Variable de session (globale)	vs_*nomVariable*	vs_brevet

Structures conditionnelles

- **Trois formes de IF**

IF-THEN	IF-THEN-ELSE	IF-THEN-ELSEIF
IF *condition* **THEN** *instructions;* **END IF;**	**IF** *condition* **THEN** *instructions;* ELSE *instructions;* **END IF;**	**IF** *condition1* **THEN** *instructions;* **ELSEIF** *condition2* **THEN** *instructions2;* ELSE *instructions3;* **END IF;**

```
BEGIN
  DECLARE v_telephone CHAR(14)
        DEFAULT '06-76-85-14-89';
  IF SUBSTR(v_telephone,1,2)='06' THEN
     SELECT "C'est un portable";
  ELSE
     SELECT "C'est un fixe...";
  END IF;
END;
```

Structure CASE

- Deux écritures:
  - Choix selon la valeur d'une variable
  - Plusieurs choix possibles (conditions)

CASE	*searched* CASE
CASE *variable* WHEN *expr1* THEN *instructions1;* WHEN *expr2* THEN *instructions2;* … WHEN *exprN* THEN *instructionsN;* [ELSE *instructionsN+1;*] **END CASE;**	CASE WHEN *condition1* THEN *instructions1;* WHEN *condition2* THEN *instructions2;* … WHEN *conditionN* THEN *instructionsN;* [ELSE *instructionsN+1;*] **END CASE;**

Structure itérative: WHILE

- Syntaxe: **WHILE** condition **DO**
  instructions;
  END WHILE

Condition simple

```
DECLARE v_somme INT DEFAULT 0;
DECLARE v_entier SMALLINT DEFAULT 1;

WHILE (v_entier <= 100) DO
 SET v_somme  := v_somme+v_entier;
 SET v_entier := v_entier+1;
END WHILE ;
SELECT v_somme;
```

```
+---------+
| v_somme |
+---------+
|    5050 |
+---------+
```

Condition composée

```
DECLARE v_telephone CHAR(14)
       DEFAULT '06-76-85-14-89';
DECLARE v_trouve BOOLEAN
       DEFAULT  FALSE;
DECLARE v_indice SMALLINT DEFAULT  1;
WHILE (v_indice <= 14 AND NOT v_trouve) DO
  IF SUBSTR(v_telephone,v_indice,1) = '4'
   THEN
     SET v_trouve := TRUE;
   ELSE
     SET v_indice := v_indice + 1;
 END IF;
END WHILE ;
IF v_trouve THEN
  SELECT CONCAT('Trouvé 4 à l''indice :
',v_indice);
END IF;
```

Trouvé 4 à l'indice : 11

Structure itérative: REPEAT

- Syntaxe: **REPEAT**
  instructions;
  UNTIL condition **END REPEAT**

Condition simple	Condition composée

`REPEAT`
`  SET v_somme  := v_somme + v_entier;`
`  SET v_entier := v_entier + 1;`
`UNTIL v_entier > 100 END REPEAT;`
	`REPEAT`
	`  IF SUBSTR(v_telephone,v_indice,1) = '4'`
	`  THEN`
	`  SET v_trouve := TRUE;`
	`  ELSE`
	`  SET v_indice := v_indice + 1;`
	`  END IF;`
	`UNTIL (v_indice > 14 OR v_trouve)`
	`  END REPEAT;`
	`IF v_trouve THEN`
	`  SELECT CONCAT('Trouvé 4 à l''indice :`
	`',v_indice);`
	`END IF;`

Structure itérative: LOOP

- Syntaxe: [etiquette:] **LOOP**
  instructions;
  END LOOP [etiquette];

Avec LEAVE	Avec ITERATE

`boucle1: LOOP`
`  SET v_somme  := v_somme + v_entier;`
`  SET v_entier := v_entier + 1;`
`  IF v_entier > 100 THEN`
`  LEAVE boucle1;`
`  END IF;`
`END LOOP boucle1;`
	`boucle1: LOOP`
	`  SET v_somme  := v_somme + v_entier;`
	`  SET v_entier := v_entier + 1;`
	`  IF v_entier <= 100 THEN`
	`  ITERATE boucle1;`
	`  END IF;`
	`  LEAVE boucle1;`
	`END LOOP boucle1;`

- Boucle infinie sans l'instruction **LEAVE**.
- **ITERATE** force à reprendre l'exécution au début de la boucle.

Structure itérative: FOR

- La structure de contrôle FOR n'est pas encore implémentée par MySQL !
- Vous devrez la programmer par un *répéter* (REPEAT…), un *tant que* (WHILE…) ou encore par une *boucle sans fin* (LOOP…).

Inclure une requête SELECT dans SQL/PSM

- Syntaxe: **SELECT** col1 [,col2…] **INTO** variable1 [,variable2…] **FROM** nomTable ...;

Code MYSQL	Commentaires
BEGIN DECLARE v_comp VARCHAR(15); SELECT compa **INTO** v_comp FROM Pilote WHERE brevet='PL-2'; ... END;	Chargement d'une variable locale à un bloc. Nécessité d'appeler par la suite cette procédure (CALL).

- **ATTENTION**
  - La **requête doit retourner un et un seul enregistrement**. Sinon, les exceptions **NO_DATA_FOUND** ou **TOO_MANY_ROWS** sont levées.
  - Pour traiter des requêtes renvoyant plusieurs enregistrements, il faudra utiliser des **curseurs** !