1. Introduction sur le langage

Le langage de programmation ajoute au langage SQL les principales notions d'un langage de programmation (variable, structure alternative, structure itérative, procédure, fonction)

Déclaration d'une variable **locale**	**declare** nomClient char(30) [default valeur] Le type correspondent à un type MySQL : int, char, real … La portée d'une variable locale est le bloc BEGIN ... END dans qui est déclaré.
Utilisation d'une variable **utilisateur**	Variable non déclarée, non typée, préfixée par @ La portée d'une variable utilisateur est la session. Si on teste une variable qui n'existe pas on récupère la valeur null
Affectation d'une valeur	**set** nomClient = 'Legrand' set @nom = 'Legrand'
Récupération d'une valeur en utilisant une variable locale	declare nomClient char(30) select nom **into** nomClient from client where id = 2
Récupération de plusieurs valeurs en utilisant des variables utilisateur	select nom **into** nomClient from client where id = 2 select nom, prenom into @nom, @prenom from client where id = 2

Structure alternative if else	**if** condition **then**
	instruction(s)
	[**else**
	instruction(s)]
	end if;

Structure alternative if elseif	**if** condition **then**
	instruction(s)
	elseif condition **then**
	instruction(s)
	…
	[**else**
	instruction(s)]
	end if;

Structure selon valeur	**case** variable **when** valeur then instruction; [**when** valeur then instruction] ... [**else** instruction] **end case**
Structure selon condition	case when condition then instruction; [when condition then instruction] ... [else instruction] end case

Structure itération	**while** condition **do**
	instruction(s)
	end while;

Ces éléments ne peuvent être mis en œuvre qu'à l'intérieur d'une procédure ou d'un déclencheur.
Si tous les cas ne sont pas traités dans la structure case (absence de else) le système retourne une erreur : Error Code: 1339. Case not found for CASE statement

MySQL dispose aussi de variables 'système' préfixées par @@.
Pour obtenir la liste : SHOW VARIABLES;

2. Définition d'une procédure stockée

Une procédure stockée est une procédure enregistrée sur le serveur et utilisable par un client.

Création

```
create procedure nomProcedure (parametre, ... )
    begin
       instruction(s)
    end
```

Un paramètre est défini par trois éléments : sens nom type

sens	in | out | inout (en entrée, en sortie, en entrée/sortie). In est la valeur par défaut.
nom	nom du paramètre
type	type du paramètre : int, varchar(30), …

Suppression

```
drop procedure if exists nomProcedure
```

Exécution

```
call nomProcedure(nomParametre, …)
```

Pour créer une procédure ou une fonction il faut modifier le délimiteur d'instruction (par défaut ;) pour pouvoir utiliser le ; à l'intérieur de la procédure ou de la fonction.

```
delimiter  $$
Create ……
   instruction(s)
$$
```

Exemple : écriture d'une procédure retournant le nom d'un client dont l'identifiant est passé en paramètre.

```
drop procedure if exists getNom;
```

```
delimiter $$
create procedure getNom(idClient int, out nomClient char(30))
    if exists (select nom from Client where id = idClient) then
        select nom into nomClient from Client where id = idClient;
    else
        set nomClient = 'inexistant';
    end if;
$$
```

Les mots clés 'begin' et 'end' ne sont nécessaires que si le bloc d'instructions comporte plusieurs instructions.

Appel de la procédure

```
call getNom(1, @nom);
select @nom;
```

Pour lister l'ensemble des procédures contenues dans une base de données

```
SHOW PROCEDURE STATUS where db = 'nomBase';
```

Exemple : écriture d'une procédure pour mettre à jour le solde d'un compte. En paramètre on dispose du numéro de compte, de sens de l'écriture (d pour débit et c pour crédit) et du montant

```
create procedure majSolde(idCompte integer, sens varchar(1), montant decimal(6,2))
    begin
        if sens = 'c' then
            update compte set solde = solde + montant where id = idCompte;
        else
            update compte set solde = solde - montant where id = idCompte;
        end if;
    end
```

Appel de la procédure :

```
set @numCompte = 1;
set @sens = 'c';
set @montant = 2000 ;
call majSolde(@numCompte, @sens, @montant);
```

3. Définition d'une fonction stockée

Comme dans un langage de programmation, une fonction est une procédure qui retourne un résultat. Une fonction peut être utilisée dans une requête SQL mais elle ne peut pas exécuter des actions qui modifient l'état de la base de données.

Création	**create function** nomFonction (NomParametre type, …) returns type DETERMINISTIC begin instruction SQL; return valeur end
Suppression	**drop function** if exists nomFonction
Exécution	select nomFonction(parametre, …)

Pour éviter l'emploi du mot DETERMINISTIC : set global log_bin_trust_function_creators = 1;
Contrairement à une procédure, un paramètre ne peut être qu'en entrée donc le sens ne s'applique pas.

Exemple

```
create function getNom(idClient int) returns char(30)  DETERMINISTIC
    begin
        declare nomClient char(30) default 'inexistant';
        if exists (select nom from Client where id = idClient) then
            select nom into nomClient from Client where id = idClient;
        else
            set nomClient = 'inexistant';
        end if;
        return nomClient;
    end
```

Pour exécuter une fonction : select getNom(1);

4. La gestion des erreurs par l'utilisation d'un gestionnaire d'erreur

Il n'est pas rare qu'une instruction SQL génère une erreur.
Par exemple : un ajout qui ne respecte pas une contrainte d'intégrité, la tentative de suppression d'une table qui est lié à une autre table par une contrainte d'intégrité, etc.

On obtient alors un message d'erreur

Error Code: 1062. Duplicata du champ '1' pour la clef 'fichefrais.PRIMARY'
Error Code: 1452. Cannot add or update a child row: a foreign key constraint
Error Code: 1828. Cannot drop column '…': needed in a foreign key constraint '…'
Error Code: 1091. Ne peut effacer (DROP) 'login'. Vérifiez s'il existe…

Certaines erreurs système peuvent être évitées en effectuant un contrôle préalable. On peut ainsi vérifier l'unicité de la clé primaire et l'existence de la clé étrangère.

Dans le cas de la suppression d'un champ sur une table, on peut vérifier l'existence de la table et du champ en utilisant la table COLUMNS, table système MySQL présent dans la base INFORMATION_SCHEMA.

```
if not EXISTS(SELECT 1 FROM INFORMATION_SCHEMA.COLUMNS
                    WHERE TABLE_NAME = nomTable
                     AND COLUMN_NAME = nomColonne) then
```

Mais d'autres ne peuvent pas être éviter. C'est le cas par exemple pour supprimer un champ d'une table si ce dernier est lié à une contrainte d'intégrité.

Dans ce cas on peut traiter l'erreur à l'aide d'un gestionnaire d'erreur. C'est un peu l'équivalent de l'instruction try … catch mais le principe est assez différent.

Un gestionnaire d'erreur définit une instruction ou un bloc d'instructions, qui va être exécuté en cas d'erreur correspondant au gestionnaire.
Le gestionnaire est déclaré après la déclaration des variables locales, mais avant les instructions de la procédure.
Un gestionnaire peut, soit provoquer l'arrêt de la procédure (EXIT), soit faire reprendre la procédure après avoir géré l'erreur (CONTINUE).
On peut associer le gestionnaire à une erreur à partir du numéro d'erreur

Pour créer un gestionnaire d'erreur

```
DECLARE { EXIT | CONTINUE } HANDLER FOR numero_erreur
   instruction ou bloc d'instructions
```

Exemple : Procédure permettant de supprimer une colonne d'une table

```
delimiter $$

create procedure suppressionColonne(nomTable varchar(30), nomColonne varchar(30), out resultat
varchar(80))
begin

   DECLARE exit HANDLER FOR 1828 set resultat = 'Le champ ne peut être supprimé, il est lié à
une contrainte d''intégrité ';

   if not EXISTS(SELECT 1 FROM INFORMATION_SCHEMA.COLUMNS
                     WHERE TABLE_NAME = nomTable) then
      set resultat = "La table n'existe pas";
   elseif not EXISTS(SELECT 1 FROM INFORMATION_SCHEMA.COLUMNS
                           WHERE TABLE_NAME = nomTable
                            AND COLUMN_NAME = nomColonne) then
      set resultat = "la colonne n'existe pas";
   else
      # pour passer en paramètre le nom d'une table ou d'une colonne dans une requête, il faut utiliser
une requête préparée
      set @sql = concat('alter table ', nomtable, ' drop ' , nomColonne);
      prepare maRequete from @sql;
      execute maRequete;
      deallocate prepare maRequete;
      set resultat = "La colonne a été supprimée";
   end if;
end;

$$
```

Complément:
https://zestedesavoir.com/tutoriels/730/administrez-vos-bases-de-donnees-avec-mysql/952_securiser-et-automatiser-ses-actions/3957_gestionnaires-derreurs-curseurs-et-utilisation-avancee

https://www.grafikart.fr/tutoriels/procedures-triggers-fonctions-593