Programmes stockés

Programmes stockés

- Blocs nommés qui sont compilés et qui résident dans la base de données.
- Ce sont des **fonctions** ou **procédures stockées** (*stored routines* ou *stored modules*) capables d'inclure des paramètres en entrée.
- Comme dans tous les langages de programmation:
  - les **fonction**s retournent un unique résultat,
  - Les **procédures** réalisent des actions sans en donner de résultat (sauf éventuellement en paramètre de sortie).
- Le cycle de vie d'un programme est le suivant :
  1. création de la procédure ou de la fonction,
  2. compilation et stockage dans la base,
  3. appel et éventuellement suppression de la base.

Structure d'un programme

```
CREATE { PROCEDURE | FUNCTION }
     nomSousProgramme [(...)] [RETURNS typeMSQL]
BEGIN
[DECLARE déclaration]; ...
instructions MySQL;
     BEGIN
      [DECLARE déclaration];...
                 ...
     ...
       instructions MySQL;
     END;
   ...
END
$
```


Procédures stockées

```
CREATE PROCEDURE [nomBase.]nomProcédure(
                [ [ IN | OUT | INOUT ] param typeMySQL
               [,[ IN | OUT | INOUT ] param2 typeMySQL ] ] ...)
   [ LANGUAGE SQL
     | [NOT] DETERMINISTIC
     | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL D
ATA }
     | SQL SECURITY { DEFINER | INVOKER }
     | COMMENT 'commentaire'
    ]
BEGIN
 [DECLARE… ;]
 bloc d'instructions SQL et MySQL ... ;
END
délimiteur
```

Procédures stockées

- Par défaut, la procédure est créée dans la base de données courante (sélectionnée). Si un nom est spécifié (***nomBase***), la procédure appartiendra à cette base de données.
- **IN** désigne un paramètre d'entrée (par défaut), **OUT** un paramètre de sortie et **INOUT** un paramètre d'entrée et de sortie.
- **LANGUAGE SQL** (par défaut) détermine le langage de programmation de la procédure. MySQL n'est pas encore compatible avec d'autres langages que le sien.
- **DETERMINISTIC:** Une procédure est considérée comme " DETERMINISTIC" si elle produit toujours le même résultat pour les mêmes paramètres d'entrée, et "NOT DETERMINISTIC " sinon. Si ni DETERMINISTIC ni NOT DETERMINISTIC n'est donné dans la définition de la procédure, la valeur par défaut est NOT DETERMINISTIC.

Procédures stockées

- **CONTAINS SQL** renseigne sur le fait que la procédure interagit avec la base. **NO SQL** indique l'inverse. **READS SQL DATA** précise que les interactions sont en lecture seulement. **MODIFIES SQL DATA** signifie que des mises à jour de la base sont possibles.
- **SQL SECURITY** détermine si la procédure s'exécute avec les privilèges du créateur (option par défaut : *definer-rights procedure*) ou ceux de l'utilisateur qui appelle la procédure (*invoker-rights procedure*).
- **COMMENT** permet de commenter la procédure au niveau du dictionnaire des données.
- ***bloc d'instructions SQL et MySQL*** contient les déclarations et les instructions de la procédure écrite dans le langage SQL/PSM.
- ***délimiteur*** : délimiteur de commandes différent de « ; » (symbole utilisé obligatoirement en fin de chaque déclaration et instruction du langage procédural de SQL/PSM).

Fonctions stockées

- Une fonction est une procédure nommée retournant une valeur.

```
CREATE FUNCTION [nomBase.]nomFonction(
                        [ param typeMySQL
                       [,param2 typeMySQL ] ] ...)
    RETURNS typeMySQL
    [ LANGUAGE SQL
      | [NOT] DETERMINISTIC
      | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL
DATA }
      | SQL SECURITY { DEFINER | INVOKER }
      | COMMENT 'commentaire'
    ]
BEGIN
 [DECLARE… ;]
 bloc d'instructions SQL et MySQL ... ;
 contenant un « RETURN variable ; »
END
délimiteur
```


Compilation

- Pour compiler ces programmes à partir de la console, il faut ajouter **un délimiteur après chaque dernier END** comme suit :

```
DELIMITER $
CREATE PROCEDURE GetAllProducts()
  BEGIN
    SELECT *  FROM products;
  END $
DELIMITER ;
```

  - La première commande est **DELIMITER $**, modifie le délimiteur standard qui est un **point-virgule (;)** à $. Cela permet le ; utilisé dans le corps de la procédure à être transmis au serveur plutôt que d'être interprété par MySQL.
  - Après le mot-clé END, nous utilisons le délimiteur **$** pour indiquer la fin du programme stocké.
  - La dernière commande (**DELIMITER;**) modifie le délimiteur de nouveau au point-virgule (;).

Exemple

Préfixer les variables	Commentaire
`delimiter $` `SET @vs_nom = 'Placide Fresnais'$`	Déclaration du délimiteur et d'une variable de session.
`DROP PROCEDURE sp1$`	Suppression de la procédure.
`CREATE PROCEDURE sp1()`	Création de la procédure.

`BEGIN`
`DECLARE v_nbHVol DECIMAL(7,2);`
`SELECT nbHVol INTO v_nbHVol`
`FROM Pilote WHERE nom = @vs_nom;`
`SELECT v_nbHVol;`
`END`
`$`
	Bloc d'instructions.
	Trace du résultat.
	Fin du bloc

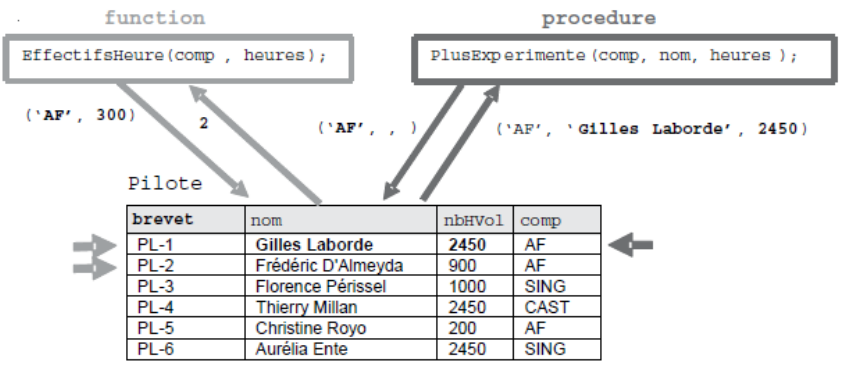
`CALL sp1()$`	Appel de la procédure.

Appel

- Un programme stocké peut être invoquée par des triggers, d'autres procédures stockées et des applications telles que Java, Python, PHP, etc.
- **Appel d'une procédure**
  - **CALL** <procedure_name> [(parameters)];
- **Appel d'une fonction**
  - SELECT <function_name> ([parameters])
  - SET <variable> := <function_name> ([parameters])

Exemple

- Considérons la table Pilote. Nous allons écrire (dans la base bdsoutou) une fonction et une procédure :



brevet	nom	nbHVol	comp
PL-1	Gilles Laborde	2450	AF
PL-2	Frédéric D'Almeyda	900	AF
PL-3	Florence Périssel	1000	SING
PL-4	Thierry Millan	2450	CAST
PL-5	Christine Royo	200	AF
PL-6	Aurélia Ente	2450	SING

Exemple: Fonction

- La fonction **EffectifsHeure(comp,heures)** devra renvoyer le nombre de pilotes d'une compagnie donnée (premier paramètre) qui ont plus d'heures de vol que la valeur du deuxième paramètre.
  - Si aucun pilote, retourne 0.
  - Si aucune compagnie n'est passée en paramètre (mettre NULL), le calcul inclut toutes les compagnies.

brevet	nom	nbHVol	comp
PL-1	Gilles Laborde	2450	AF
PL-2	Frédéric D'Almeyda	900	AF
PL-3	Florence Périssel	1000	SING
PL-4	Thierry Millan	2450	CAST
PL-5	Christine Royo	200	AF
PL-6	Aurélia Ente	2450	SING

Exemple: Fonction

```
CREATE FUNCTION bdsoutou.EffectifsHeure(pcomp VARCHAR(4),
                        pheuresVol DECIMAL(7,2)) RETURNS SMALLINT
BEGIN
 DECLARE resultat SMALLINT;
 IF (pcomp IS NULL) THEN
    SELECT COUNT(*) INTO resultat FROM Pilote WHERE nbHVol > pheuresVol;
 ELSE
    SELECT COUNT(*) INTO resultat FROM Pilote WHERE nbHVol > pheuresVol
                    AND comp = pcomp;
 END IF;
 RETURN resultat;
END$
```


Exercice

- Récrivez la fonction précédente sous forme d'une procédure.

Exemple: Procédure

- La procédure **PlusExperimente(comp,nom,heures)** doit retourner le nom et le nombre d'heures de vol du pilote (par l'intermédiaire des deuxième et troisième paramètres) le plus expérimenté d'une compagnie donnée (premier paramètre).
  - Si plusieurs pilotes ont la même expérience, un message d'erreur est affiché.
  - Si aucune compagnie n'est passée en paramètre (mettre NULL), la procédure retourne le nom du plus expérimenté et le code de sa compagnie (par l'intermédiaire du premier paramètre).

brevet	nom	nbHVol	comp
PL-1	Gilles Laborde	2450	AF
PL-2	Frédéric D'Almeyda	900	AF
PL-3	Florence Périssel	1000	SING
PL-4	Thierry Millan	2450	CAST
PL-5	Christine Royo	200	AF
PL-6	Aurélia Ente	2450	SING

Exemple: Procédure

```
CREATE PROCEDURE bdsoutou.PlusExperimente
 (INOUT pcomp VARCHAR(4), OUT pnomPil VARCHAR(20), OUT pheuresVol DECIMAL(7,2))
BEGIN
 DECLARE p1 SMALLINT;
 IF (pcomp IS NULL) THEN
   SELECT COUNT(*) INTO p1 FROM Pilote
          WHERE nbHVol=(SELECT MAX(nbHVol) FROM Pilote);
 ELSE
   SELECT COUNT(*) INTO p1 FROM Pilote
          WHERE nbHVol=(SELECT MAX(nbHVol) FROM Pilote WHERE comp=pcomp)
          AND comp = pcomp;
 END IF;
 IF (p1 = 0) THEN
   SELECT ('Aucun pilote n''est le plus expérimenté') AS resultat;
 ELSEIF p1 > 1 THEN
  SELECT('Plusieurs pilotes sont les plus expérimentés') AS resultat;
 ELSE
   IF (pcomp IS NULL) THEN
       SELECT nom, nbHVol, comp INTO pnomPil, pheuresVol, pcomp
        FROM Pilote WHERE nbHVol=(SELECT MAX(nbHVol) FROM Pilote);
   ELSE
       SELECT nom, nbHVol INTO pnomPil, pheuresVol FROM Pilote
            WHERE nbHVol=(SELECT MAX(nbHVol) FROM Pilote WHERE comp=pcomp)
            AND comp = pcomp;
   END IF;
 END IF;
END $
```

Exemple: Appel

Appel d'un sous-programme	Résultat

```
delimiter ;
SELECT bdsoutou.EffectifsHeure
     ('AF',300);
```

```
+-----------------------------------+
| bdsoutou.EffectifsHeure('AF',300) |
+-----------------------------------+
|                                 2 |
+-----------------------------------+
```

```
delimiter ;
SET @vs_compa = 'AF';
SET @vs_nompil = '';
SET @vs_heures = '';

CALL bdsoutou.PlusExperimente
 (@vs_compa, @vs_nompil,
  @vs_heures);
```

```
SELECT @vs_compa,@vs_nompil,@vs_heures;
+-----------+----------------+------------+
| @vs_compa | @vs_nompil     | @vs_heures |
+-----------+----------------+------------+
| AF        | Gilles Laborde | 2450.00    |
+-----------+----------------+------------+
```


Destruction d'un programme

- La syntaxe de suppression d'un sous-programme est la suivante :
  - **DROP** {**PROCEDURE** | **FUNCTION**} [IF EXISTS] [*nomBase.*]*nomSousProg*
  - IF EXISTS évite un message de *warning* si le sous-programme n'existe pas.

- Pour supprimer un programme, si vous n'êtes pas son créateur, le privilège **ALTER ROUTINE** est requis sur la base de données (ou au niveau global).

Exceptions

Erreurs [codes et messages]

- Un client MySQL se voit retourner un certain nombre d'informations dès qu'il contrarie le serveur par une instruction SQL illégale:

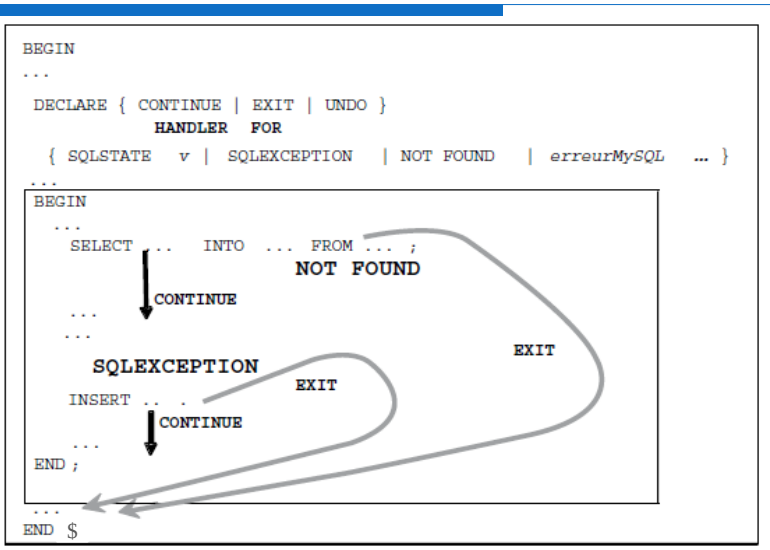
Instructions	Résultats
`SELECT nom,nbHVol FROM Pilote_;`	`ERROR 1146 (42S02): Table 'bdsoutou.Pilote_' doesn't exist`
`SELECT 3nom,nbHVol FROM Pilote;`	`ERROR 1054 (42S22): Unknown column '3nom' in 'field list'`
`SELECT nom,nbHVol FROME Pilote;`	`ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'Pilote' at line 1`

  - Un code numérique signalé par le titre ERROR (1146, 1054, etc.).
  - Une chaîne de 5 caractères signalée par le titre SQLSTATE ('42S02', '42S22' et '42000').
  - Une chaîne de caractères fournissant la description textuelle de l'erreur.
- **Il faut consulter la liste des erreurs dans la documentation officielle de manière à connaître le numéro d'erreur MySQL.**

Généralités

- Le mécanisme des exceptions (handling errors) permet d'éviter qu'un programme ne s'arrête dès la première erreur suite à une instruction SQL:
  - SELECT ne retournant aucune ligne, INSERT ou UPDATE d'une valeur incorrecte, DELETE d'un enregistrement ayant des enregistrements associés, etc.
- Une exception (**handler**) MySQL correspond à une condition d'erreur et peut être associée à un identificateur (exception nommée).
- Une exception est détectée (levée) si elle est prévue dans un handler au cours de l'exécution d'un bloc (entre BEGIN et END).
- Une fois levée, elle fait continuer (ou sortir du bloc) le programme après avoir réalisé une ou plusieurs instructions que le programmeur aura explicitement spécifiées.

Principe général des exceptions



Syntaxe générale d'une exception

DECLARE { CONTINUE | EXIT | UNDO }

HANDLER FOR

{SQLSTATE [VALUE] '*valeur_sqlstate*' | *nomException* | SQLWARNING | NOT FOUND | SQLEXCEPTION | *code_erreur_mysql* }

instructions MySQL;

[, { SQLSTATE...} ...]

- La directive **CONTINUE** force à poursuivre l'exécution de programme lorsqu'il se passe un événement prévu dans la clause FOR.
- La directive **EXIT** fait sortir l'exécution du bloc courant (entre BEGIN et END).

Syntaxe générale d'une exception

- **SQLSTATE** permet de couvrir toutes les erreurs d'un état donné.
- ***nomException*** s'applique à la gestion des exceptions nommées (étudiées plus loin).
- **SQLWARNING** permet de couvrir toutes les erreurs d'état SQLSTATE débutant par 01.
- **NOT FOUND** permet de couvrir toutes les erreurs d'état SQLSTATE débutant par 02.
- **SQLEXCEPTION** gère toutes les erreurs qui ne sont ni gérées par SQLWARNING ni par NOT FOUND.
- ***instructions MySQL*** : une ou plusieurs instructions du langage de MySQL (bloc, appel possibles par CALL d'une fonction ou d'une procédure stockée).

Exceptions avec EXIT

- L'exemple suivant décrit une procédure qui gère une erreur :
  - Aucun pilote n'est associé à la compagnie de code passé en paramètre (**NOT FOUND**).
  - La procédure ne se termine pas correctement si plusieurs lignes sont retournées (**ERROR 1172 (42000): Result consisted of more than one row**).

Code MySQL	Commentaires
CREATE PROCEDURE bdsoutou.procException1 (IN p_comp VARCHAR(4)) BEGIN DECLARE flagPlusDun BOOLEAN DEFAULT 0; DECLARE flagNOTFOUND BOOLEAN DEFAULT 0; DECLARE var1 VARCHAR(20);	Déclaration de la procédure et des variables.
BEGIN DECLARE **EXIT** HANDLER FOR **1172** SET flagPlusDun :=1; DECLARE EXIT HANDLER FOR **NOT FOUND** SET flagNOTFOUND :=1;	Bloc qui déclare les deux exceptions.
SELECT nom INTO var1 FROM bdsoutou.Pilote WHERE comp=p_comp;	Requête pouvant déclencher l'exception prévue.
SELECT CONCAT('Le seul pilote de la compagnie ', p_comp,' est ',var1) AS 'Resultat procException1';	Affichage du résultat.
END;	Fin du bloc.

IF flagNOTFOUND THEN
SELECT CONCAT('Il n''y a pas de pilote pour la
compagnie ',p_comp)
AS 'Resultat procException1';
END IF;
IF flagPlusDun THEN
SELECT CONCAT('Il y a plusieurs pilotes pour la
compagnie ',p_comp)
AS 'Resultat procException1';
END IF;
END $
	Gestion des erreurs.
	
	Fin de la procédure.

Exception NOT FOUND traitée avec EXIT et CALL

Code MySQL	Commentaires

`CREATE PROCEDURE bdsoutou.pasTrouve`
`(IN p_comp VARCHAR(4))`
`BEGIN`
`SELECT CONCAT('Il n''y a pas de pilote pour`
`la compagnie ',p_comp)`
`AS 'Resultat procpasTrouve';`
`END;`
	Codage du sous-programme appelé lors de l'exception.

`CREATE PROCEDURE bdsoutou.procException1`
`(IN p_comp VARCHAR(4))`
`BEGIN`
`DECLARE var1 VARCHAR(20);`
`DECLARE EXIT HANDLER FOR NOT FOUND`
`CALL bdsoutou.pasTrouve(p_comp);`
	Procédure qui déclare l'exception.

`SELECT nom INTO var1 FROM bdsoutou.Pilote`
`WHERE comp = p_comp;`
 	
`SELECT CONCAT('Le seul pilote de la compagnie`
`',p_comp,' est ',var1)`
`AS 'Resultat procException1';`
	Requête pouvant déclencher l'exception prévue.
	Affichage du résultat.

`END $`	Fin de la procédure principale.

Exception NOT FOUND traitée avec CONTINUE

Code MySQL	Commentaires

`CREATE PROCEDURE bdsoutou.procException1`
`(IN p_comp VARCHAR(4))`
`BEGIN`
`DECLARE flagNOTFOUND BOOLEAN DEFAULT 0;`
`DECLARE var1 VARCHAR(20);`
	Déclaration de la procédure et des variables.

`DECLARE CONTINUE HANDLER FOR NOT FOUND` `SET flagNOTFOUND :=1;`	Bloc qui déclare l'exception.
`SELECT nom INTO var1 FROM bdsoutou.Pilote` `WHERE comp=p_comp;`	Requête pouvant déclencher l'exception prévue.

`IF flagNOTFOUND THEN`
`SELECT CONCAT('Il n''y a pas de pilote pour la`
`compagnie ',p_comp)`
`AS 'Resultat procException1';`
`ELSE`
`SELECT CONCAT('Le seul pilote de la compagnie ',`
`p_comp,' est ',var1)`
`AS 'Resultat procException1';`
`END IF;`
	Test de gestion de l'erreur.
	Affichage du résultat.

`END $`	Fin de la procédure.

Exercice

- Réécrivez la même procédure qui gère (avec CONTINUE) en plus du NOT FOUNT l'erreur "Result consisted of more than one row".

Gestion des autres erreurs (SQLEXCEPTION)

- Si une erreur non prévue en tant qu'exception (dans les clauses DECLARE HANDLER) se produisait, le programme se terminerait anormalement en renvoyant l'erreur en question.
- La directive **SQLEXCEPTION** couvre toutes les erreurs qui ne sont administrées ni par SQLWARNING ni par NOT FOUND.
- Il faudra soit :
  - interrompre brusquement la procédure avec différents cas d'erreurs, pour lister les codes erreur générés en sortie (pas encore possible avec MySQL);
  - gérer globalement les autres exceptions, tout en ne sachant pas de quelles erreurs il s'agit.

Exceptions toutes traitées avec EXIT et SQLEXCEPTION

Code MySQL	Commentaires

`CREATE PROCEDURE bdsoutou.tropdeLignes`
`(IN p_comp VARCHAR(4))`
`BEGIN`
`SELECT CONCAT('Il y a plusieurs pilotes pour`
`la compagnie ',p_comp)`
`AS 'Resultat tropdeLignes';`
`END $`
	Codage des sous-programmes appelés lors des exceptions.

`CREATE PROCEDURE bdsoutou.pasTrouve`
`(IN p_comp VARCHAR(4))`
`BEGIN`
`SELECT CONCAT('Il n''y a pas de pilote pour`
`la compagnie ',p_comp)`
`AS 'Resultat pasTrouve';`
`END $`	

Exceptions toutes traitées avec EXIT et SQLEXCEPTION

```
CREATE PROCEDURE bdsoutou.autreErreur()
BEGIN
  SELECT 'Erreur mais laquelle?'
         AS 'Resultat autreErreur';
END;
```

3 cas

```
CREATE PROCEDURE bdsoutou.procException1
               (IN p_comp VARCHAR(4))
BEGIN
 DECLARE var1 VARCHAR(20);
 DECLARE EXIT HANDLER FOR 1172
         CALL bdsoutou.tropdeLignes(p_comp);
 DECLARE EXIT HANDLER FOR NOT FOUND
         CALL bdsoutou.pasTrouve(p_comp);
 DECLARE EXIT HANDLER FOR SQLEXCEPTION
         CALL bdsoutou.autreErreur();
 SELECT nom INTO var1 FROM bdsoutou.Pilote
        WHERE comp = p_comp;
 SELECT CONCAT('Le seul pilote de la compagnie',
               p_comp,' est ',var1)
        AS 'Resultat procException1';
END;
```

Procédure principale qui déclare les deux exceptions et toutes les autres.

Requête pouvant déclencher l'exception prévue. Affichage du résultat.

Fin de la procédure.

Même erreur sur différentes instructions
Gestion d'une seule exception

Code MySQL	Commentaires

```
CREATE PROCEDURE bdsoutou.procException2
      (IN p_brevet VARCHAR(6),IN p_heures DECIMAL(7,2))
BEGIN
 DECLARE v_nom         VARCHAR(20);
 DECLARE flagNOTFOUND BOOLEAN DEFAULT 0;
 DECLARE v_requete     TINYINT;
 BEGIN
 DECLARE EXIT HANDLER FOR NOT FOUND SET flagNOTFOUND :=1;
  SET v_requete := 1;
   SELECT nom INTO v_nom FROM bdsoutou.Pilote
              WHERE brevet = p_brevet;
   SELECT CONCAT('Le pilote de  code ',p_brevet,
                 ' est ',v_nom);
   SET v_requete := 2;
   SELECT nom INTO v_nom FROM bdsoutou.Pilote
              WHERE nbHVol = p_heures;
   SELECT CONCAT('Le pilote ayant ',p_heures,
                 ' heures de vol est ',v_nom);
 END;
IF flagNOTFOUND THEN
   IF v_requete = 1 THEN
    SELECT CONCAT('Pas de pilote de brevet : ',p_brevet);
   ELSEIF v_requete = 2 THEN
    SELECT CONCAT('Pas de pilote ayant ce nombre
           d''heures de vol : ',p_heures);
   END IF;
 END IF;
END;
```

Bloc avec les requêtes déclenchant potentiellement une exception prévue.

Traitement pour savoir quelle requête a provoqué l'exception.

Même erreur sur différentes instructions
Gestion de plusieurs exceptions

Code MySQL	Commentaires

```
CREATE PROCEDURE bdsoutou.procException3
      (IN p_brevet VARCHAR(6),IN p_heures DECIMAL(7,2))
BEGIN
  DECLARE v_nom         VARCHAR(20);
  DECLARE flagNOTFOUND BOOLEAN DEFAULT 0;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
          SET flagNOTFOUND := 1;
  SELECT nom INTO v_nom FROM bdsoutou.Pilote
             WHERE brevet = p_brevet;
  IF flagNOTFOUND THEN
     SELECT CONCAT('Pas de pilote de brevet : ',p_brevet);
     SET flagNOTFOUND := 0;
  ELSE
     SELECT CONCAT('Le pilote de  code ',p_brevet,
                   ' est ',v_nom);
  END IF;
  SELECT nm INTO v_nom FROM bdsoutou.Pilote
             WHERE nbHVol = p_heures;
  IF flagNOTFOUND THEN
     SELECT CONCAT('Pas de pilote ayant ce nombre d'
                   'heures de vol : ',p_heures);
  ELSE
     SELECT CONCAT('Le pilote ayant ',p_heures,
                   ' heures de vol est ',v_nom);
  END IF;
END;
```

Gestion de l'exception de la première requête.

Gestion de l'exception de la deuxième requête.

Exceptions nommées

- Pour intercepter une erreur MySQL et lui attribuer au passage un identificateur, il faut utiliser la clause DECLARE CONDITION. La syntaxe est la suivante :

  DECLARE *nomException* **CONDITION FOR**

  {SQLSTATE [VALUE] '*valeur_sqlstate*' | *code_erreur_mysql*}

- Il est ainsi possible de regrouper plusieurs types d'erreurs (avec SQLSTATE ou cibler une erreur en particulier en indiquant le code erreur de MySQL).

- Une fois l'exception nommée, il est possible de l'utiliser dans la déclaration de l'événement associé via la directive DECLARE HANDLER.

Exceptions nommées: exemple

Compagnie

comp	ville	nomComp
AF	Paris	Air France
SING	Singapour	Singapore AL
CAST	Blagnac	Castanet AL
EJET	**Dublin**	**Easy Jet**

à détruire

Pilote

prevet	nom	nbHVol	comp
PL-1	Gilles Laborde	2450	AF
PL-2	Frédéric D'Almeyda	900	AF
PL-3	Florence Périssel	1000	SING
PL-4	Thierry Millan	2450	CAST
PL-5	Christine Royo	200	AF
PL-6	Aurélia Ente	2450	SING

- Le tableau suivant décrit la procédure **procExceptionNommee** qui intercepte une erreur référentielle (SQLSTATE à 23 000). Il s'agit de contrôler le programme si la compagnie à détruire est encore rattachée à un enregistrement référencé dans la table Pilote.

Exceptions nommées: exemple

Code MySQL	Commentaires

```
CREATE PROCEDURE bdsoutou.procExceptionNommee
      (IN p_comp VARCHAR(4))
BEGIN
 DECLARE flagerr BOOLEAN DEFAULT 0;
 BEGIN
   DECLARE erreur_ilResteUnPilote CONDITION
           FOR SQLSTATE '23000';
   DECLARE EXIT HANDLER
           FOR erreur_ilResteUnPilote SET flagerr :=1;
   SET AUTOCOMMIT=0;
    DELETE FROM Compagnie WHERE comp = p_comp;
    SELECT CONCAT('Compagnie ',p_comp, ' détruite')
           AS 'Resultat procExceptionNommee';
 END;
 IF flagerr THEN
   ROLLBACK;
     SELECT CONCAT('Désolé, il reste encore un pilote à
                   la compagnie ',p_comp)
          AS 'Resultat procExceptionNommee';
   ELSE
   COMMIT;
 END IF;
END;
```

Déclaration de l'exception nommée.

Corps du traitement (validation).

Gestion de l'exception.