Programmation – Erreurs

- Testons la procédure suivante :

```
DELIMITER |
CREATE PROCEDURE test_erreur(IN pi_idclient INT)
BEGIN
  INSERT INTO commande(idcommande,idclient,datecom)
  VALUES(NULL, pi_idclient, NOW());
END|

DELIMITER ;
```

Programmation – Erreurs

Deux cas de figure :

1. Le client existe
2. Il n'existe pas

```
CALL test_erreur(3); -- client existe

Query OK, 1 row affected, 1 warning (0,01 sec)
```

```
CALL test_erreur(100); -- client n'existe pas

ERROR 1452 (23000): Cannot add or update a child row: a
foreign key constraint fails (`clicom`.`commande`,
CONSTRAINT `commande_client_fk_1` FOREIGN KEY (`idclient`)
REFERENCES `client` (`idclient`))
```

Programmation – Erreurs

Lorsqu'une erreur est générée, cela peut être problématique si elle n'est pas traitée car elle pourrait bloquer un processus plus global.

Pour gérer une erreur, il est nécessaire de créer un gestionnaire d'erreur (`HANDLER`), via la requête générique suivante :

```
DECLARE {CONTINUE|EXIT|UNDO} HANDLER
FOR {code_erreur| SQLSTATE [VALUE] etat_sql|
nom_condition| SQLWARNING| NOT FOUND| SQLEXCEPTION}[,…]
instruction(s);
```

Programmation – Erreurs

Conditions d'activation du gestionnaire :

- Selon un numéro d'erreur – nombre entier (`1452`, etc. - voir documentation MySQL)
- Selon un état SQL – chaîne de cinq caractères (`'23000'`, etc.)
- Une condition (voir slide suivant)
- `SQLWARNING` : alias pour les états sql qui commencent par `'01'`
- `NOT FOUND` : alias pour les états sql qui commencent par `'02'`
- `SQLEXCEPTION` : alias pour les états sql qui ne commencent pas par `'00'` (résultat ok), `'01'` (avertissement) ou `'02'` (lié aux curseurs)

Programmation – Erreurs

Une condition est une variable qui rend intelligible un code d'erreur ou un état SQL :

```
DECLARE nom_condition CONDITION FOR code_erreur;
```

Exemple pour l'erreur que nous avons rencontrée :

```
DECLARE cle_primaire_absente CONDITION FOR 1452;
```

Programmation – Erreurs

Comportement dès que le gestionnaire est activé et l'erreur traitée :

- `CONTINUE` : l'exécution de la procédure se poursuit
- `EXIT` : on quitte la procédure
- `UNDO` : pas encore implémenté

Programmation – Erreurs

Exemple complet pour l'erreur rencontrée (pas d'erreur) :

```
DROP PROCEDURE IF EXISTS test_erreur;
DELIMITER |
CREATE PROCEDURE test_erreur(IN pi_idclient INT)
BEGIN
   DECLARE cle_primaire_absente CONDITION FOR 1452;
   DECLARE EXIT HANDLER FOR cle_primaire_absente
   BEGIN
      SELECT 'erreur rencontrée et gérée';
   END;

  INSERT INTO commande(idcommande,idclient,datecom)
  VALUES(NULL, pi_idclient, NOW());

  SELECT 'Fin de la procédure test_erreur';
END|

DELIMITER ;

CALL test_erreur(3);
```

Programmation – Erreurs

Exemple complet pour l'erreur rencontrée (avec erreur et `EXIT`) :

```
DROP PROCEDURE IF EXISTS test_erreur;
DELIMITER |
CREATE PROCEDURE test_erreur(IN pi_idclient INT)
BEGIN
   DECLARE cle_primaire_absente CONDITION FOR 1452;
   DECLARE EXIT HANDLER FOR cle_primaire_absente
   BEGIN
      SELECT 'erreur rencontrée et gérée';
   END;

  INSERT INTO commande(idcommande,idclient,datecom)
  VALUES(NULL, pi_idclient, NOW());

  SELECT 'Fin de la procédure test_erreur';
END|

DELIMITER ;

CALL test_erreur(100);
```

Programmation – Erreurs

Exemple complet pour l'erreur rencontrée (avec erreur et `CONTINUE`) :

```
DROP PROCEDURE IF EXISTS test_erreur;
DELIMITER |
CREATE PROCEDURE test_erreur(IN pi_idclient INT)
BEGIN
   DECLARE cle_primaire_absente CONDITION FOR 1452;
   DECLARE CONTINUE HANDLER FOR cle_primaire_absente
   BEGIN
      SELECT 'erreur rencontrée et gérée';
   END;

  INSERT INTO commande(idcommande,idclient,datecom)
  VALUES(NULL, pi_idclient, NOW());

  SELECT 'Fin de la procédure test_erreur';
END|

DELIMITER ;

CALL test_erreur(100);
```

Programmation – Erreurs

Autre déclaration pour cette erreur (code erreur) :

```
DROP PROCEDURE IF EXISTS test_erreur;
DELIMITER |
CREATE PROCEDURE test_erreur(IN pi_idclient INT)
BEGIN
   DECLARE CONTINUE HANDLER FOR 1452
   BEGIN
      SELECT 'erreur rencontrée et gérée';
   END;

  INSERT INTO commande(idcommande,idclient,datecom)
  VALUES(NULL, pi_idclient, NOW());

  SELECT 'Fin de la procédure test_erreur';
END|

DELIMITER ;

CALL test_erreur(100);
```

Programmation – Erreurs

Autre déclaration pour cette erreur (état sql) :

```
DROP PROCEDURE IF EXISTS test_erreur;
DELIMITER |
CREATE PROCEDURE test_erreur(IN pi_idclient INT)
BEGIN
   DECLARE CONTINUE HANDLER FOR SQLSTATE '23000'
   BEGIN
      SELECT 'erreur rencontrée et gérée';
   END;

  INSERT INTO commande(idcommande,idclient,datecom)
  VALUES(NULL, pi_idclient, NOW());

  SELECT 'Fin de la procédure test_erreur';
END|

DELIMITER ;

CALL test_erreur(100);
```

Programmation – Erreurs

Exercice 1

Créez une procédure stockée d'insertion d'une nouvelle commande dont les paramètres sont les numéros de commande et de client (qui est supposé correct).

Déclarez un gestionnaire pour l'erreur d'unicité de la clé primaire (idcommande).

Programmation – Erreurs

Pour gérer plusieurs erreurs, vous pouvez :

- Mettre plusieurs conditions séparées par une virgule pour un seul gestionnaire d'erreur (plus concis, mais même traitement)

```
CREATE PROCEDURE test_multi_erreurs(IN pi_idcommande INT, IN pi_idclient
INT)
BEGIN
  DECLARE erreur_unicite_cle_primaire CONDITION FOR 1062;
  DECLARE erreur_unicite_cle_etrangere CONDITION FOR 1452;

  DECLARE EXIT HANDLER FOR erreur_unicite_cle_primaire,
erreur_unicite_cle_etrangere

   BEGIN
     SELECT 'Violation d\'une contrainte d\'unicité';
   END;
…
```

Programmation – Erreurs

- Déclarer plusieurs gestionnaires d'erreur (traitements spécifiques)

```
CREATE PROCEDURE test_multi_erreurs(IN pi_idcommande INT, IN pi_idclient
INT)
BEGIN
  DECLARE erreur_unicite_cle_primaire CONDITION FOR 1062;
  DECLARE erreur_unicite_cle_etrangere CONDITION FOR 1452;

  DECLARE EXIT HANDLER FOR erreur_unicite_cle_primaire
    BEGIN
      SELECT 'Violation d\'une contrainte de clé primaire';
    END;

  DECLARE EXIT HANDLER FOR erreur_unicite_cle_etrangere
    BEGIN
      SELECT 'Violation d\'une contrainte de clé étrangère';
    END;
…
```

Programmation – Erreurs

- Déclarer un gestionnaire d'erreur par défaut

```
CREATE PROCEDURE test_multi_erreurs(IN pi_idcommande INT, IN pi_idclient
INT)
BEGIN
  DECLARE EXIT HANDLER FOR SQLWARNING, SQLEXCEPTION
    BEGIN
      SELECT 'Erreur détectée';
    END;
…
```

Programmation – Erreurs

- Si une erreur correspond aux conditions de plusieurs gestionnaires, c'est le plus spécifique qui est déclenché

Programmation – Erreurs

Exercice 2 :

- Testez la déclaration du slide précédent en créant plusieurs gestionnaires pour une même erreur

Programmation – Déclenchements

Il est également possible de déclencher l'envoi d'une erreur via la requête générique suivante :

```
SIGNAL { SQLSTATE valeur_état | condition }
[SET { MESSAGE_TEXT | MYSQL_ERRNO   | SCHEMA_NAME |
TABLE_NAME | … },…] ;
```

NB : la condition ne peut représenter qu'un état SQL

Programmation – Déclenchements

Exemples

```
SIGNAL SQLSTATE '33333';

-- ou
SIGNAL SQLSTATE '33333'
SET MESSAGE_TEXT = 'MON PREMIER DECLENCHEMENT';

-- ou
SIGNAL SQLSTATE '33333'
SET MYSQL_ERRNO = 9999;

- - ou
SIGNAL SQLSTATE '33333'
SET MESSAGE_TEXT = 'MON PREMIER DECLENCHEMENT',
MYSQL_ERRNO = 9999;
```

Programmation – Déclenchements

Exemple avec CONDITION :

```
DROP PROCEDURE IF EXISTS test_declenchement;

DELIMITER |

CREATE PROCEDURE test_declenchement(IN pi_valeur INT)
BEGIN
  DECLARE condition_test CONDITION FOR SQLSTATE '99999';

  IF pi_valeur > 10 THEN
    SIGNAL condition_test;
  END IF;
END |

DELIMITER ;
```

Programmation – Déclenchements

Exemple avec CONDITION et MESSAGE_TEXT :

```
DROP PROCEDURE IF EXISTS test_declenchement;

DELIMITER |

CREATE PROCEDURE test_declenchement(IN pi_valeur INT)
BEGIN
  DECLARE condition_test CONDITION FOR SQLSTATE '99999';

  IF pi_valeur > 10 THEN
    SIGNAL condition_test
    SET MESSAGE_TEXT = 'Test message';
  END IF;
END |

DELIMITER ;
```

Programmation – Déclenchements

Exercice 3 :

- Créez une procédure permettant de déclencher une erreur et l'affichage du message adéquat si le stock du produit qu'un client désire commander n'est pas suffisant (paramètres : idproduit, qcom)

Programmation – Déclenchements

Exercice 4 :

- Créez une procédure permettant de déclencher une erreur et de placer le message adéquat si le compte d'un client est négatif
- Créez une procédure permettant d'ajouter une commande et qui fait appel à la procédure ci-dessus pour vérifier si la commande peut-être ajoutée