

# Limiter la complexité du code applicatif grâce au SGBD

par Alain Defrance

Date de publication : 01/03/2008

Dernière mise à jour : 12/08/2008

Dans cet article, il sera démontré comment simplifier le code applicatif en exploitant les possibilités du SGBD grâce aux contraintes.

|   |    |
|---|----|
| I - Introduction.....   | 3  |
| II - Exemple de code sans contrainte.....                         | 3  |
| II-A - Le SQL de création.....                                    | 3  |
| II-B - Code php.....  | 3  |
| II-C - Pourquoi ce programme est-il source d'erreurs ?.....       | 4  |
| III - Les contraintes.....  | 5  |
| III-A - Propriétés des champs (NULL/NOT NULL et/ou DEFAULT).....  | 5  |
| III-A-1 - NULL/NOT NULL.....                                      | 5  |
| III-A-2 - DEFAULT.....  | 5  |
| III-A-3 - Un exemple.....   | 5  |
| III-B - Où écrire la contrainte ?.....                            | 6  |
| III-C - Les contraintes à la création.....                        | 6  |
| III-D - Les ajouts et suppressions de contraintes.....            | 7  |
| IV - Le nouveau code.....   | 8  |
| IV-A - Le SQL de création modifié pour gérer les contraintes..... | 8  |
| IV-B - Code php allégé.....                                       | 9  |
| IV-C - Précautions à prendre avec ce type de développement.....   | 9  |
| V - Remerciements.....  | 10 |

## I - Introduction

Beaucoup de programmeurs pensent que pour développer un programme, il suffit d'enchaîner les lignes de codes qui vont effectuer tous les traitements et tous les contrôles.

Le Système de Gestion de Base de Donnée, ou **SGBD**, est souvent sous estimé alors qu'il se révèle être un excellent moyen de faire ces traitements à la place du code produit par le programmeur.

Le gain de temps est conséquent et évite par ailleurs les erreurs de programmation.

Cet article a pour but de donner des exemples concrets de simplification de code en utilisant des contraintes sur la base de données.

On utilisera Php/MySQL mais ce type d'utilisation est valable avec tous les **SGBDR**.

## II - Exemple de code sans contrainte

### II-A - Le SQL de création

Voici un code **SQL** créant une petite base de données sans aucun contrôle.

```
CREATE TABLE `Clients`
(
  `numClient` INTEGER,
  `nomClient` VARCHAR(20),
  `prenomClient` VARCHAR(20),
  `adresseClient` VARCHAR(100),
  `cpClient` VARCHAR(5),
  `villeClient` VARCHAR(30),
  `dateInscription` TIMESTAMP
);

CREATE TABLE `Produits`
(
  `numProduit` INTEGER,
  `libelleProduit` VARCHAR(50),
  `prixProduit` FLOAT
);

CREATE TABLE `Commandes`
(
  `numCommande` INTEGER,
  `dateCommande` TIMESTAMP,
  `numClient` INTEGER
);

CREATE TABLE `Concerner`
(
  `numProduit` INTEGER,
  `numCommandes` INTEGER,
  `nombreProduit` INTEGER
);
```

### II-B - Code php

Cette base de données créée au dessus est bien sûr exploitable, mais il faudra effectuer tous les contrôles de saisie. Voici l'exemple de deux fonctions :

```
<?php
//Ajoute un nouveau tuple ayant pour information les paramètres.
function addClient($nom, $pnom, $adr, $cp, $ville)
{
  //Saisie obligatoire du nom
```

```

if(!empty($nom))
{
    //Saisie obligatoire du prénom
    if(!empty($pnom))
    {
        //On récupère la date courante.
        $date = time();

        //On récupère le dernier identifiant de client
        $rs = mysql_query("SELECT MAX(numClient) as ancienID FROM `Clients`");
        $data = mysql_fetch_array($rs);
        $newID = $data['ancienID'];

        //On calcule le nouvel identifiant
        if($newID == NULL)
        {
            $newID = 1;
        }
        else
        {
            $newID++;
        }

        //On ajoute en base de données les données venant d'être testées.

mysql_query("INSERT INTO `Clients` VALUES('.$newID.', '.$nom.', '.$pnom.', '.$adr.', '.$cp.', '.$vil
    }
    else
    {
        echo '[Erreur] Le prénom doit être saisi';
    }
}
else
{
    echo '[Erreur] Le nom doit être saisi';
}
}

// Supprime toute trace du client ayant le numéro passé en paramètre.
function delClient($num)
{
    //On supprime le client en question.
    mysql_query("DELETE FROM `Clients` WHERE numClient=".$num);

    //On supprime toutes les associations entre les produits et les commandes du client puisqu'on va supprimer ses
    mysql_query("
        DELETE
        FROM `Concerner`
        WHERE `numCommande` IN
        (SELECT `numCommande` FROM `Commandes` WHERE numClient=".$num.)");

    //On supprime les commandes anciennement passées par le client que l'on vient de supprimer.
    mysql_query("DELETE FROM `Commandes` WHERE numClient=".$num);
}
?>

```

## II-C - Pourquoi ce programme est-il source d'erreurs ?

Lors de la manipulation des données saisies par un utilisateur, plusieurs risques nous menacent. Le premier est que très souvent, nous comptons sur la saisie de l'utilisateur, alors qu'il est fort probable qu'il se trompe. Le deuxième risque concerne la sécurité, en particulier sur le web. Ainsi s'il n'y a pas de contrôle de saisie, la possibilité est laissée à l'utilisateur de saisir volontairement des valeurs erronées afin de provoquer un comportement anormal du programme.

Quant au troisième, ce n'est pas tant un risque qu'une action de vérification supplémentaire. En effet, lorsqu'il y aura action sur les données stockées, même s'il y a cohérence, il y aura une étape de vérification afin d'être sûr que la

modification n'en engendre pas d'autre. Et ce, afin de garder une intégrité entre les données. Par exemple, ici il faut faire trois actions DELETE pour supprimer définitivement un client de la base de données. Tout ceci représente beaucoup de travail. Ce qui vient en plus du but premier de l'application. L'idéal serait de pouvoir se décharger de ces différents contrôles, ce qui représente souvent plus de la moitié du code, pour se concentrer sur l'application elle-même.

### III - Les contraintes

Les contraintes sont des solutions faciles à mettre en place et qui permettent au **SGBD** d'être autonome dans sa gestion de l'**intégrité** des données. Ces contraintes sont une modification de la structure des tables, et se rajoutent en **SQL**. Cependant attention, la diversité des **SGBD** est grande, et en fonction de celui utilisé une contrainte peut ne pas exister. En effet, même si elle est acceptée par le **SGBD** elle ne sera pas forcément appliquée par celui-ci. Il convient donc de vérifier son existence dans la documentation officielle du **SGBD** utilisé. Pour cet article, le **SGBD** utilisé est **MySQL**. Il dispose de plusieurs moteurs proposant différentes fonctionnalités. Celui par défaut s'appelle MyISAM. Afin de mieux gérer l'intégrité, nous allons utiliser un moteur plus adapté : **InnoDB**. Pour demander à **MySQL** son utilisation, la syntaxe suivante doit être utilisée :


```
CREATE TABLE `laTable` ( ... ) ENGINE = `InnoDB`;
```

#### III-A - Propriétés des champs (NULL/NOT NULL et/ou DEFAULT)

Ce ne sont pas véritablement des contraintes, ce sont des propriétés sur les champs de la base de données.


##### III-A-1 - NULL/NOT NULL

L'attribut **NULL** permet d'autoriser la valeur non saisie, le **SGBD** acceptera qu'il n'y ait pas de donnée assignée à ce champ (donnée nulle)  
 En ce qui concerne l'attribut **NOT NULL**, le **SGBD** ne peut accepter un tuple que si une valeur est saisie et donnée pour ce champ.

 *Si on ne précise rien, **NULL** est utilisé par défaut, mais il est conseillé de le spécifier.*

##### III-A-2 - DEFAULT

L'attribut **DEFAULT** permet, comme son nom l'indique, de définir une valeur par défaut. Si une valeur n'est pas précisée lors d'un INSERT INTO ou d'un UPDATE, la valeur définie par défaut sera insérée. On peut utiliser un **DEFAULT** avec un **NULL** ou **NOT NULL**. En effet un DEFAULT définissant une valeur par défaut, il évite un refus de la part du **SGBD** en cas de non saisie d'un champ NOT NULL.  
 On peut demander que le champ ne soit pas vide, et si ce n'est pas le cas, au lieu de refuser l'enregistrement, on insère la valeur par défaut.

 *Un attribut **DEFAULT** particulier est l'**AUTO\_INCREMENT**. Il permet de créer un entier toujours plus grand que celui généré précédemment. Il permet de nous affranchir de la création des identifiants de tables.*

##### III-A-3 - Un exemple


```
CREATE TABLE `laTable`  
(
```

```

`ident` INTEGER NOT NULL
AUTO_INCREMENT PRIMARY KEY, --PRIMARY KEY est indispensable pour mettre AUTO_INCREMENT, nous verrons plus tard so
`uneChaineObligatoire` VARCHAR(20) NOT NULL,
`unPrix` FLOAT NOT NULL DEFAULT 0,
`dateTuple` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
) ENGINE = `InnoDB`;
    
```

### III-B - Où écrire la contrainte ?

Nous allons commencer à parler de véritables contraintes. Les contraintes peuvent être nommées (et donc déclarées) à la fin du champ (comme les attributs **NULL**, **NOT NULL** et **DEFAULT**) grâce au mot clé **CONSTRAINT**. S'il est placé en fin de champ, dans ce cas, la contrainte porte sur le champ à la fin duquel elle est placée. Une contrainte peut être écrite en dehors de toutes définition de champs. Le mot clé **CONSTRAINT** sera alors utilisé après les définitions de champs.

 *Attention il serait faux de penser que le mot clé **CONSTRAINT** permet de créer une contrainte, il permet seulement de lui donner un nom. Si ce n'est pas obligatoire de nommer la contrainte il est conseillé de le faire. Si ce n'est pas fait, le choix est laissé au SGBD et il sera très difficile de s'y retrouver par la suite.*

#### exemple

```

CREATE TABLE `laTable`
(
  `ident` INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY
  CONSTRAINT `nomContrainte` ..., -- contrainte sur ident
  `uneChaineObligatoire` VARCHAR(20) NOT NULL,
  `unPrix` FLOAT NOT NULL DEFAULT 0,
  `dateTuple` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT `nomContrainte2` ... -- contrainte sur nimporte quel champs que l'on précisera
) ENGINE = `InnoDB`;
    
```

Ici les contraintes sont placées à la fin des champs et les exemples suivant seront développés de la même manière.

### III-C - Les contraintes à la création

#### structure de contrainte sur une table

```

CONSTRAINT `nomContrainte`
{
  {UNIQUE | PRIMARY KEY | FOREIGN KEY}
  [ REFERENCES `nomTable` (`nomChamp`) [ON DELETE action] [ON UPDATE action] ]
  | CHECK (operation booléene)
}
    
```

#### Les contraintes :

**UNIQUE** : Demande au **SGBD** de vérifier que ce champ soit unique, on peut être sûr que tous les enregistrements d'un champ unique seront différents (utile pour les login par exemple).

**PRIMARY KEY** : On spécifie au **SGBD** la clé primaire de la table, elle sera par ailleurs unique et une seule contrainte **PRIMARY KEY** sera autorisée par table (ce qui ne veut pas dire qu'elle ne peut pas être composée de plusieurs champs).

**FOREIGN KEY** : On spécifie au **SGBD** une clé étrangère. Elle se rattache donc à un champ d'une autre table, et elle doit faire référence à un champ unique ou une clé primaire.

**REFERENCES** : Définit la référence d'une clé étrangère (**FOREIGN KEY**).

**CHECK** : Effectue une opération booléenne qui doit être vérifiée sur l'ensemble des tuples déjà présents.

#### Les actions :

**ON DELETE action** : Déclenche une action qui sera effectuée à la suppression d'un tuple référence.

**ON UPDATE action** : Déclenche une action qui sera effectué à la modification d'un tuple référence.

**NO ACTION** : Demande par défaut, au **SGBD** de rien faire, il retournera donc une erreur si la cohérence des données n'est plus respectée.

**CASCADE** : Répercute les suppressions/modifications.

**SET NULL** : Quand une référence est supprimée ou modifiée, la valeur est marquée à **NULL**, c'est à dire l'absence de valeur.

**SET DEFAULT** : Quand une référence est supprimée ou modifiée, la valeur est remise par défaut.

 Attention le **CHECK** est accepté par **MySQL**, mais pas appliqué.

#### Exemple

```
CREATE TABLE `laTable1`
(
  `identTable1` INTEGER NOT NULL AUTO_INCREMENT,
  `uneChaineObligatoire` VARCHAR(20) NOT NULL,
  `unPrix` FLOAT NOT NULL DEFAULT 0,
  `dateTuple` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT `PK_laTable1` PRIMARY KEY (`identTable1`)
) ENGINE = `InnoDB`;

CREATE TABLE `laTable2`
(
  `identTable2` INTEGER NOT NULL AUTO_INCREMENT,
  `uneInformation` VARCHAR(30) NULL,
  `idTable1` INTEGER NOT NULL,
  CONSTRAINT `PK_laTable2` PRIMARY KEY (`identTable2`),
  CONSTRAINT `FK_laTable2_laTable1` FOREIGN KEY (`idTable1`) REFERENCES
  laTable1(`identTable1`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE = `InnoDB`;
```

### III-D - Les ajouts et suppressions de contraintes

Il est possible d'ajouter ou de supprimer des contraintes. Voici la syntaxe :

#### Syntaxe

```
ALTER TABLE `laTable`
ADD CONSTRAINT <la contrainte>;

ALTER TABLE `laTable`
DROP CONSTRAINT <nomContrainte>;
```

#### Exemple

```
CREATE TABLE `laTable1`
(
  `identTable1` INTEGER NOT NULL,
  `uneChaineObligatoire` VARCHAR(20) NOT NULL,
  `unPrix` FLOAT NOT NULL DEFAULT 0,
  `dateTuple` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
) ENGINE = `InnoDB`;

CREATE TABLE `laTable2`
(
  `identTable2` INTEGER NOT NULL,
  `uneInformation` VARCHAR(30) NULL,
  `idTable1` INTEGER NOT NULL
) ENGINE = `InnoDB`;

ALTER TABLE `laTable1`
ADD CONSTRAINT `PK_laTable1` PRIMARY KEY (`identTable1`);

ALTER TABLE `laTable2`
```

### Exemple

```

ADD CONSTRAINT `PK_laTable2` PRIMARY KEY (`identTable2`),
ADD CONSTRAINT `FK_laTable2_laTable1` FOREIGN KEY (`identTable2`) REFERENCES `laTable1`
(`identTable1`);

ALTER TABLE `laTable2`
DROP PRIMARY KEY,
DROP FOREIGN KEY `FK_laTable2_laTable1`;

ALTER TABLE `laTable1`
DROP PRIMARY KEY;
    
```



*Souvent beaucoup de générateurs de scripts **SQL**, et de programmeurs commencent par créer leurs tables, puis posent les contraintes juste après. Ceci a pour but d'éviter que le **SGBD** gêne à la création des tables.*

## IV - Le nouveau code

### IV-A - Le SQL de création modifié pour gérer les contraintes

#### Nouvelle base de données

```

CREATE TABLE `Clients`
(
    `numClient` INTEGER AUTO_INCREMENT,
    `nomClient` VARCHAR(20) NOT NULL,
    `prenomClient` VARCHAR(20) NOT NULL,
    `adresseClient` VARCHAR(100) NULL,
    `cpClient` VARCHAR(5) NULL,
    `villeClient` VARCHAR(30) NULL,
    `dateInscription` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT PRIMARY KEY (`numClient`)
) ENGINE = `innnoDB`;

CREATE TABLE `Produits`
(
    `numProduit` INTEGER AUTO_INCREMENT,
    `libelleProduit` VARCHAR(50) NOT NULL,
    `prixProduit` FLOAT NOT NULL,
    CONSTRAINT PRIMARY KEY (`numProduit`)
) ENGINE = `innnoDB`;

CREATE TABLE `Commandes`
(
    `numCommande` INTEGER AUTO_INCREMENT,
    `dateCommande` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    `numClient` INTEGER NOT NULL,
    CONSTRAINT PRIMARY KEY (`numCommande`)
) ENGINE = `innnoDB`;

CREATE TABLE `Concerner`
(
    `numProduit` INTEGER NOT NULL,
    `numCommande` INTEGER NOT NULL,
    `nombreProduit` INTEGER NOT NULL DEFAULT 0,
    CONSTRAINT PRIMARY KEY (`numProduit`,`numCommande`)
) ENGINE = `innnoDB`;

ALTER TABLE `Commandes`
ADD CONSTRAINT `FK_Commandes_Clients` FOREIGN KEY (`numClient`) REFERENCES `Clients` (`numClient`)
ON UPDATE CASCADE
ON DELETE CASCADE;

ALTER TABLE `Concerner`
ADD CONSTRAINT `FK_Concerner_Produits` FOREIGN KEY (`numProduit`) REFERENCES `Produits`
(`numProduit`)
    
```

## Nouvelle base de données

```

ON UPDATE CASCADE
ON DELETE CASCADE,
ADD CONSTRAINT `FK_Concerner_Commandes` FOREIGN KEY ( `numCommande` ) REFERENCES `Commandes`
( `numCommande` )
ON UPDATE CASCADE
ON DELETE CASCADE;
```

## IV-B - Code php allégé

```

<?php
//Ajoute un nouveau tuple ayant pour information les paramètres.
function addClient($nom, $pnom, $adr, $cp, $ville)
{
    $sqlNom = (!empty($nom))?'\''. $nom. '\': 'NULL';
    $sqlPnom = (!empty($pnom))?'\''. $pnom. '\': 'NULL';

    //On ajoute en base de donnée les donnée non vérifiés en ne spécifiant ni l'identifiant, ni la date.

    mysql_query("INSERT INTO `Clients` VALUES('', ".$sqlNom.", ".$sqlPnom.", '".$adr."', '".$cp."', '".$ville."', N
        // Le SGBD controlera lui meme les données

        // On récupère les retours d'erreurs du SGBD
        if(mysql_errno() == 1048) // un champ est vide
        {
            $msg = explode('\'', mysql_error(), 3);
            switch ($msg[1])
            {
                case 'prenomClient':
                    echo '[Erreur] Le prénom doit être saisie';
                    break;
                case 'nomClient':
                    echo '[Erreur] Le nom doit être saisie';
                    break;
            }
        }
    }

    // Supprime toute trace du client ayant le numéro passé en parametre.
    function delClient($num)
    {
        //On supprime le client en question.
        mysql_query("DELETE FROM `Clients` WHERE numClient=".$num);
        /* Plus rien d'autre à faire, le SGBD a vu que les tuples rattachés à ce
        client n'était plus utiles, il les a donc supprimé grâce à la foreing key
        et au ON DELETE CASCADE */
    }
}
?>
```

## IV-C - Précautions à prendre avec ce type de développement

Il est très avantageux de laisser le contrôle de cohérence de données au **SGBD**. Cependant, il faut faire très attention de bien modéliser sa base de données car si elle n'est pas assez précise, il se peut que l'application soit faillible ou bien même boguée.

Les contraintes de base que nous venons de voir ne permettent pas de faire des miracles. Toutefois, il est possible d'aller plus loin en utilisant des **procédures stockées** et des **trigger**.

La philosophie de programmation reste la même, on réduit la ligne de code afin de reporter le travail sur la base de données.

L'utilisation du SQL procédural permet une **gestion poussé des erreurs liées** au domaine de gestion, c'est-à-dire l'aspect métier de notre application.

## V - Remerciements

Merci à **yiannis**, **xave**, **Fleur-Anne.Blain** et **ced** pour les relectures.