Bases de données relationnelles et contraintes SQL

par Frédéric Brouard, alias SQLpro
Expert langage SQL, SGBDR, modélisation de données
Auteur de :

- SQLpro http://sqlpro.developpez.com/
- "SQL", coll. Synthex, avec C. Soutou, Pearson Education 2005
- "SQL" coll. Développement, Campus Press 2001

Enseignant aux Arts & Métiers et à l'ISEN Toulon

L'une des idées force de la conception des bases de données relationnelles repose sur la notion de contrainte. Une contrainte n'est autre qu'une règle impérative ne devant en aucun cas être violée. Certaines contraintes sont le reflet du modèle de données et permettent d'assurer la cohérence fonctionnelle des relations entre les tables. D'autres assurent que les données saisies correspondent bien aux limites de l'univers que l'on modélise. Enfin, les règles "métiers", c'est à dire le fonctionnel applicatif, nécessite la mise en oeuvre de contraintes complémentaires souvent complexes. Mais, si toutes les contraintes s'avèrent nécessaires, elle sont souvent mal comprise, très souvent mal gérées dans les processus de développement, et se voient donc souvent reléguées, voire abandonnées.
Cet article à pour but de vous montrer l'intérêt des contraintes SQL avec des exemples concret de leur utilité.

Cet article est paru pour la première fois dans le magazine "Programmation sous Linux" en septembre 2006

Mes audits m'ont souvent montrés que les contraintes étaient souvent peu utilisées. Je crois avoir une explication... Le mot contrainte fait peur. A l'origine, dans le droit, on trouve la *contrainte par corps*, une mesure d'exécution de peine qui permet l'incarcération de la personne qui ne s'acquitte pas d'une condamnation pécuniaire. Abolie en France depuis quelques décennies, elle reste en vigueur dans certains pays comme la Tunisie : *la contrainte par corps est exécutée à raison d'un jour d'emprisonnement par trois dinars ou fraction de trois dinars sans que sa durée puisse excéder deux ans* (art. 344 de la Loi n°99-90 du 2 août 1999).
Dans d'autres domaines, la notion de contrainte est quand même plus réjouissante. Il y a un peu plus d'un an, devant donner une formation sur SQL, je voyais une jeune fille, arrivée fort tôt au cours, patienter en remplissant une série de grille que je pris pour un carré magique. "tenez lui dis-je, pourquoi ne pas résoudre votre problème en utilisant une simple requête SQL ?" Je lui promis de terminer la formation par une telle démonstration. La résolution des sudokus reposent sur l'utilisation de contraintes.
Il existe toute une branche des mathématiques et plus précisément dans l'algorithmique et dans l'informatique qui utilise massivement les contraintes. Je veut parler de la programmation par contraintes, notamment à travers un langage comme PROLOG ou encore un framework Java spécialisé et non commercial comme CHOCO.
C'est d'ailleurs à l'aide de la programmation par contrainte qu'en 1976 Appel et Haken, à l'université de l'Illinois, firent la démonstration finale du théorème de la carte à 4 couleurs, en explorant systématiquement les cas particuliers. C'était aussi la première fois que la démonstration d'un théorème se faisait à l'aide d'un ordinateur.

Ce que je vous propose de visiter c'est comment SQL implémente les contraintes dans les bases de données relationnelles.

1 – La portée des contraintes.

Dans une application qui utilise SQL, on trouve les éléments suivants : des bases de données relationnelle dans lesquelles se trouvent des tables et des vues. Tables et vues sont dotées de colonnes et les données sont écrites lignes par lignes. Finalement l'élément le plus petit de cet ensemble est la donnée. Les contraintes se trouvent à chacun des niveaux de cet édifice.
Les contraintes dites "de domaine" concernent les données. Les contraintes de tables peuvent porter sur une colonne, sur une ligne ou sur une table. Les assertions peuvent porter sur plusieurs tables, voire toutes les tables de la base.

1.1 portée des contraintes de table

Les contraintes de table sont les plus connues. La plus classique ne concerne qu'une colonne à la fois. C'est la contrainte d'**obligation de valeur**

(NOT NULL) qui exige, pour la colonne qui en est pourvue, qu'à toute ligne de la table une valeur soit exprimée.
On trouve ensuite la contrainte de **clef primaire** (PRIMARY KEY) qui assure l'unicité de la référence à une ligne d'une table. C'est le moyen par lequel on repère une ligne et une seule dans la table. Toutes les colonnes concourrant à la clef se doivent d'être valuées (NOT NULL).
La contrainte d'**unicité** (UNIQUE) permet de s'assurer qu'une autre clef pourrait remplacer la clef primaire. Mais à la différence de la clef primaire, la contrainte d'unicité n'oblige pas à ce que les données participant à la formation de la contrainte soient valuées. Il peut même y avoir plusieurs lignes de la table dont les données formant la contrainte d'unicité sont vide de toutes valeurs. En d'autres termes, la contrainte d'unicité exige que toute donnée *valuée* soit distincte... Par essence une donnée non valuée ne peut jamais être comparée à une autre données non valuée, pas même à elle même. C'est à dire que le prédicat " NULL = NULL " ne sera ni vrai ni faux mais tout simplement inévaluable !
La contrainte la plus draconienne est la contrainte de **validation** (CHECK). Elle permet de restreindre les valeurs de la ou les colonnes qui la compose afin de respecter des règles même les plus complexes qui soient. Nous lui consacrerons un long paragraphe.
Enfin, la contrainte la plus redoutée par les développeurs, parce que mal appréhendée, est la contrainte de **clef étrangère** (FOREIGN KEY) destinée à assurer l'intégrité de référence. Nous la détaillerons.

Voici quelques exemples de ces contraintes résumé dans une table :

```
CREATE TABLE T_PATIENT_PTN
(PTN_ID           INT NOT NULL PRIMARY KEY,
 PTN_NUM_SECU     CHAR(13) UNIQUE,
 PTN_CLEF_SECU    CHAR(2)
                  CHECK (PTN_CLEF_SECU IS NULL
                     OR (SUBSTRING(PTN_CLEF_SECU FROM 1 FOR 1)
                                 BETWEEN 0 AND 9) AND
                         SUBSTRING(PTN_CLEF_SECU FROM 2 FOR 1)
                                 BETWEEN 0 AND 9)),
 PTN_NOM          CHAR(32) NOT NULL,
 PTN_PRENOM       VARCHAR(25),
 PTN_DATE_NAIS    DATE,
 PTN_CIVILITE     INT FOREIGN KEY REFERENCES T_CIVILITE_CVT ( CVT_ID))
```

Notez que le numéro de sécurité sociale est unique mais n'a pas l'obligation d'être valué. C'est pratique si vous êtes médecin et que votre patient à oublié sa carte vitale ! Pour la clef de contrôle du numéro de sécurité sociale, nous avons exigée qu'elle soit composée de deux caractères dont les valeurs peuvent être saisies entre '0' et '9'. Enfin, le code civilité doit être choisit parmi les valeurs de la clef de la table `T_CIVILITE_CVT`.

Les contraintes PRIMARY KEY, UNIQUE, FOREIGN KEY et CHECK peuvent être spécifiées directement dans la définition de la ligne de la table si elles ne portent que sur une seule colonne, sinon et tant qu'attribut de la table. Ce dernier mode est à préférer, même pour des contraintes mono colonnes. En

effet il présente plusieurs avantages, car dans ce cas la contrainte doit être nommée : son nom fera partie du message d'erreur et il sera plus facile de supprimer ou désactiver cette contrainte.

Exemple :

```
CREATE TABLE T_PATIENT_PTN
(PTN_ID           INT NOT NULL,
 PTN_NUM_SECU     CHAR(13),
 PTN_CLEF_SECU    CHAR(2),
 PTN_NOM          CHAR(32) NOT NULL,
 PTN_PRENOM       VARCHAR(25),
 PTN_DATE_NAIS    DATE,
 PTN_CIVILITE     INT,
 CONSTRAINT       PK_PTN PRIMARY KEY (PTN_ID),
 CONSTRAINT       UK_PTN_NUM_CLEF_SECU UNIQUE (PTN_NUM_SECU, PTN_CLEF_SECU),

 CONSTRAINT       CK_PTN_CLEF_SECU CHECK
                  (PTN_CLEF_SECU IS NULL
                     OR (SUBSTRING(PTN_CLEF_SECU FROM 1 FOR 1)
                                   BETWEEN 0 AND 9) AND
                         SUBSTRING(PTN_CLEF_SECU FROM 2 FOR 1)
                                   BETWEEN 0 AND 9)),
 CONSTRAINT       FK_PTN_CVT_ID FOREIGN KEY (CVT_ID) REFERENCES T_CIVILITE_CVT (CVT_ID))
```

Remarquez dans cet exemple que les contraintes ont toutes été exprimées en tant qu'attribut de la table et introduites à l'aide du mot clef CONSTRAINT, et possèdent toutes un nom bien codifié.

1.2 Portée des contraintes de domaine

Les contraintes de domaine sont des contraintes qui restreignent les valeurs qu'une donnée peut prendre. Pour mieux en mesurer le sens, il nous faut comprendre ce qu'est un domaine au sens SQL. Par exemple, un domaine POURCENT se verra doté d'une contrainte qui ne lui permet que des valeurs comprises entre 0 et 100. Nous découvrirons les domaines et leur utilisation un peu plus tard dans cet article.

1.3 Portée des assertions

Au sens commun du terme, une assertion n'est autre qu'une proposition devant toujours être vérifiée. Il en va de même en SQL. Une assertion est donc une contrainte. Une assertion porte sur différents objets de la base (en général des tables ou des vues) et on l'exprime sous la forme d'un prédicat[1] devant toujours se vérifier. La portée d'une assertion est donc la base de données dans son ensemble. Par exemple notre base pourrait être composée d'une table des prospects et d'une table des clients, chacune des tables ayant pour clef primaire un nombre entier. Nous aimerions qu'une clef attribuée à un client ne puisse être reprise pour un prospect et vice versa. Seule une assertion permet de résoudre ce cas de figure.

[1] Prédicat : expression logique dont l'évalusation conduit à une valeur booléenne (vrai ou faux)

1.4 But des contraintes

Mais enfin, quel est l'intérêt des contraintes ? Assurer l'intégrité et la cohérence des données. Sans contrainte une base de données est rapidement incohérence et faiblement intègre. La redondance y est commune et la qualité des données piètre. Toute chose qui condamne la base de données à une évolution incertaine, une mort lente et surtout à un coût exorbitant du traitement des données. Car enfin si les contraintes existent, ce n'est pas pour fustiger le développeur ni admoneste l'utilisateur, c'est pour rendre aisé, rapide et efficace le traitement des données, comme nous le verrons à travers différents exemples.
Si je devais mesurer la qualité d'une base de données au nombre des contraintes qui la compose, je dirais que 95% des tables devraient avoir une clef primaire, la plupart du temps sous forme d'entier autoincrémenté, 80 % une contrainte d'unicité (code, référence...). Dans chaque table il devrait y avoir environ 30% de colonnes pourvues de contraintes de validité. Enfin, toutes les relations devraient être pourvues d'un mécanisme d'intégrité référentielle.
Il n'est par exemple pas logique que l'on puisse saisir un caractère espace au début d'une valeur dans la plupart des colonnes de type alphanumérique. Il n'est pas non plus logique qu'une date de facture ou de commande puisse être saisie en dehors d'une plage considérée comme normale. Il est étrange de trouver des dates de paiement de facture avec une facture dont la date d'émission est située *après* le paiement ! Dans ce dernier cas, une contrainte CHECK telle que celle-ci aurait éviter une telle erreur que la justice peut considérer comme fausse facture !

```
CREATE TABLE T_FACTURE_FCT
(FCT_ID               INT NOT NULL PRIMARY KEY,
 FCT_DATE_EMMISSION   DATE NOT NULL DEFAULT CURRENT_TIMESTAMP,
 FCT_DATE_PAIEMENT    DATE CHECK (FCT_DATE_PAIEMENT IS NULL
                                  OR FCT_DATE_PAIEMENT >= FCT_DATE_EMMISSION),
 ...)
```

2 - Les domaines SQL

Le domaine SQL est en fait un remplaçant au type de base de SQL. Dans la norme SQL 2 on compte notamment les types de données suivants : char, varchar, nchar, nvarchar, date, time, timestamp, int, smallint, float, real, decimal, numeric... Un domaine SQL se construit à partir de l'un des types de base de SQL auquel on rajoute :

- une valeur par défaut si besoin est,
- zéro, une ou plusieurs contraintes CHECK,
- une éventuelle collation[2] si le type sous jacent est alphanumérique.

[2] Une collation est un attribut de la définition d'un domaine, d'une colonne ou d'une expression alphanumérique en vue de piloter son comportement (sensible ou non à la casse, aux caractères diacritiques, à la largeur d'encodage, etc...)

Exemple :

```
CREATE DOMAINE D_POURCENT
AS
   FLOAT
   DEFAULT 0.0
   CHECK (VALUE BETWEEN 0.0 AND 100.0)
```

Dès lors un domaine s'utilise comme n'importe quel type SQL pour la définition d'une table ou d'une vue.

Exemple :

```
CREATE TABLE T_CLIENT_CLI
(CLI_ID           INT NOT NULL PRIMARY KEY,
 CLI_NOM          VARCHAR(64),
 CLI_REMISE_MAX   D_POURCENT NOT NULL DEFAULT 0.0)
```

L'intérêt des domaines SQL est immense. D'abord les contraintes de validation associées, empêchent des saisies erronées. Mais plus important, dans une base de données comportant de nombreuses tables, l'absence de domaine conduit vite à associer des colonnes dont les types sont divergents. Dans ce cas, les requêtes mettant en jeu de telles colonnes sont particulièrement contre performantes car cela oblige à une conversion implicite des types de données ce qui empêche toute utilisation d'index.
Une bonne habitude consiste donc à se créer sa bibliothèque de domaine et à n'utiliser que ces derniers pour la création des tables. Pour ma part j'utilise une bibliothèque de domaine comportant environ 50 domaines SQL différents.

Certaines bases de données implémentent directement les domaines SQL. C'est le cas en particulier de PosGreSQL. D'autre indirectement comme Sybase Adaptive via l'utilisation des objets *RULE* et *DEFAULT* et des procédures comme *sp_addtype*, *sp_bindrule*, *sp_bindefault*. Enfin, si cela n'est pas le cas on peut utiliser un outil de modélisation qui les gèrent en amont (au niveau du modèle conceptuel de données), comme c'est le cas de Power Designer (ex AMC Designor) de Sybase / PowerSoft.

3 - Les assertions

Comme nous l'avons déjà dit, une assertion est une expression logique (prédicat) devant toujours être évaluée à vrai. Une assertion peut concerner plusieurs tables et c'est son intérêt. Cela se fait avec un ordre SQL commençant par CREATE ASSERTION et contenant une contrainte CHECK.

Exemple :

```
CREATE ASSERTION A_UNIQUE_CLIPRO
CHECK (NOT EXISTS (SELECT CLI_ID
                   FROM   T_CLIENT_CLI AS C
                          INNER JOIN T_PROSPECT_PSP AS P
                                ON C.CLI_ID = P.PSP_ID))
```

Cette assertion propose qu'aucune valeur de la clef de la table client définie par la colonne CLI_ID ne soit utilisée comme clef de la table des prospects et vice versa.

A ma connaissance aucune base de données commerciale ou gratuite n'a encore implémenté directement les assertions. Il est en effet difficile de traduire ce genre de contraintes dans un mécanisme codé de façon optimisé. Mais toute assertion peut être traduite en déclencheurs (triggers) à raison d'un trigger par tables concernées par l'assertion.

Ainsi l'équivalence de l'assertion de l'exemple précédent en triggers se résume à :

```
CREATE TRIGGER A_UNIQUE_CLIPRO_CLI
   ON T_CLIENT_CLI FOR INSERT, UPDATE
AS
BEGIN
IF EXISTS (SELECT CLI_ID
           FROM   T_CLIENT_CLI AS C
                  INNER JOIN NEW N
                        ON C.CLI_ID = N.CLI_ID
                  INNER JOIN T_PROSPECT_PSP AS P
                        ON C.CLI_ID = P.PSP_ID))
BEGIN
   ROLLBACK;
END;
END;
```

```
CREATE TRIGGER A_UNIQUE_CLIPRO_PSP
   ON T_PROSPECT_PSP FOR INSERT, UPDATE
AS
BEGIN
IF EXISTS (SELECT CLI_ID
           FROM   T_PROSPECT_PSP AS P
                  INNER JOIN NEW N
                        ON P.PSP_ID = N.PSP_ID
                  INNER JOIN T_CLIENT_CLI AS C
                        ON P.PSP_ID = C.CLI_ID))
BEGIN
   ROLLBACK TRANSACTION;
END;
END;
```

On notera que les triggers sont d'autres formes de contraintes permettant des traitements plus avancés.

4 - La contrainte CHECK

La contrainte de validation de données CHECK permet de restreindre les valeurs acceptables pour la colonne en appliquant un prédicat. Lorsque le prédicat est évalué à Vrai, l'occurrence est acceptée et la ligne validable.

Dans le cas contraire il y a violation de contrainte et les lignes visées par l'ordre SQL sont rejetées.
Si la contrainte est exprimée en tant qu'attribut de la colonne (contrainte monocolonne) il est possible d'utiliser le mot clef VALUE en lieu et place de la colonne. Sinon, il faut faire référence au nom des colonnes en jeu directement.
Le prédicat doit contenir les références aux colonnes en jeu, dans une expression booléenne évaluable à Vrai ou Faux.
Ce prédicat peut contenir toute expression portant sur les données de la table, telle que :

- des valeurs explicites;
- le marqueur NULL;
- des valeurs sous forme de fonctions SQL ou d'UDF (User Define Function);
- des opérateurs algébriques (+, -, *, /)
- l'opérateur de concaténation de chaîne (||)
- des opérateurs de comparaison (>, <, >=, <=, <>);
- des connecteurs logiques (AND, OR);
- l'opérateur de négation (NOT);
- la hiérarchisation des opérateurs à l'aide de parenthèses;
- des expressions SQL spécifiques telles que : BETWEEN, LIKE, IN, CASE, OVERLAPS, SIMILAR...;
- des agrégats et des sous requêtes.

Exemple

```
CREATE TABLE dbo.T_EMPLOYEE_EMP
(EMP_MATRICULE      CHAR(6) NOT NULL PRIMARY KEY,
 EMP_NOM            CHAR(32) NOT NULL
                       CHECK (SUBSTRING(VALUE FROM 1 FOR 1)
                              BETWEEN 'A' AND 'Z'
                              COLLATE French_CS_AI),
 EMP_PRENOM         VARCHAR(25)
                       CHECK (VALUE IS NULL OR
                              SUBSTRING(VALUE FROM 1 FOR 1)
                              BETWEEN 'A' AND 'Z'
                              COLLATE French_CS_AI),
 EMP_DATE_NAIS      DATE
                       CHECK (VALUE IS NULL OR
                              VALUE BETWEEN '1900-01-01'
                                        AND CURRENT_TIMESTAMP),
 EMP_NNI_SEXE       CHAR(1) CHECK (VALUE IS NULL OR
                                   VALUE IN ('1', '2')),
 EMP_NNI_AN         CHAR(2) CHECK (VALUE IS NULL OR
                                   CAST(VALUE AS INTEGER) BETWEEN '00' AND '99'),
 EMP_NNI_MOIS       CHAR(2) CHECK (VALUE IS NULL OR
                                   VALUE BETWEEN '01' AND '09'
                                   OR VALUE BETWEEN '10' AND '12',
 EMP_NNI_DPT        CHAR(2) CHECK (VALUE IS NULL OR
                                   VALUE IN (SELECT DPT_CODE
                                             FROM T_DEPARTEMENT_DPT),
 EMP_NNI_CMN        CHAR(3) CHECK (VALUE IS NULL OR
                                   CAST(VALUE AS INT) BETWEEN 0 AND 999),
 EMP_NNI_RANG       CHAR(3) CHECK (VALUE IS NULL OR
```

```
                                CAST(VALUE AS INT) BETWEEN 0 AND 999),
EMP_NNI_CLEF       CHAR(2) CHECK (VALUE IS NULL OR
                                CAST(VALUE AS INT) BETWEEN 0 AND 97),
CONSTRAINT CK_NNI_NULL CHECK ((EMP_NNI_SEXE IS NULL AND
                               EMP_NNI_AN   IS NULL AND
                               EMP_NNI_MOIS IS NULL AND
                               EMP_NNI_DPT  IS NULL AND
                               EMP_NNI_CMN  IS NULL AND
                               EMP_NNI_RANG IS NULL AND
                               EMP_NNI_CLEF IS NULL) OR
                              (EMP_NNI_SEXE IS NOT NULL AND
                               EMP_NNI_AN   IS NOT NULL AND
                               EMP_NNI_MOIS IS NOT NULL AND
                               EMP_NNI_DPT  IS NOT NULL AND
                               EMP_NNI_CMN  IS NOT NULL AND
                               EMP_NNI_RANG IS NOT NULL AND
                               EMP_NNI_CLEF IS NOT NULL)),
CONSTRAINT CK_NNI CHECK (EMP_NNI_CLEF IS NULL OR
                         EMP_NNI_CLEF = F_NNI_CLEF(EMP_NNI_SEXE,
                                                   EMP_NNI_AN  ,
                                                    EMP_NNI_MOIS,
                                                    EMP_NNI_DPT ,
                                                    EMP_NNI_CMN ,
                                                    EMP_NNI_RANG)),
EMP_SERVICE        VARCHAR(16) CHECK (EMP_SERVICE IN ('Production',
                                                      'Commercial',
                                                      'Administration')),
CONSTRAINT UK_NNI UNIQUE (EMP_NNI_SEXE, EMP_NNI_AN, EMP_NNI_MOIS, EMP_NNI_DPT,
                          EMP_NNI_CMN, EMP_NNI_RANG, EMP_NNI_CLEF))
```

Quelques remarques d'importance à propos de cet exemple :

- Chaque fois qu'une contrainte CHECK est introduite sur une colonne "nullable" le prédicat commence par `VALUE IS NULL OR...` Sans cela il ne serait pas possible de laisser cette colonne vide.

- Remarquez que pour le mois du code de sécurité sociale, une fausse bonne idée aurait été de proposer un prédicat du genre :
  `BETWEEN '01' AND '12'`
  En effet un tel prédicat laisse passer des saisies telles que :
  `'0A', '1-'`
  en fonction du jeux de caractères et de la collation.

- La contrainte `CK_NNI_NULL` s'occupe de la cohérence dans l'absence des valeurs de la clef NNI (Numéro National d'Identité). Elle considère que soit toutes les colonnes participant au numéro de sécurité sociale sont valuées, soit aucune. En effet il serait illogique de n'avoir remplit que partiellement cette clef car il pourrait alors y avoir de grande chance qu'un viol de la contrainte d'unicité ait lieu !

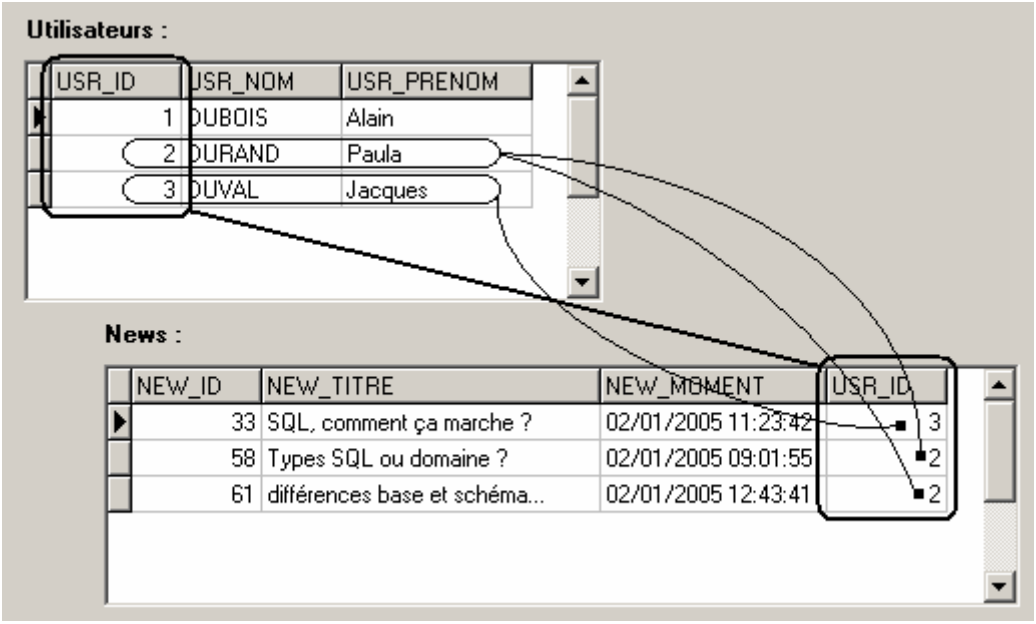
- Notez aussi l'emploi d'une UDF (fonction utilisateur) pour le contrôle de cohérence du numéro de sécurité sociale à l'aide de sa clef, fonction qui peut avoir été codée par exemple comme ceci :

```
CREATE FUNCTION F_NNI_CLEF (SEXE CHAR(1),
                            MOIS CHAR(2),
                            AN   CHAR(2),
                            DPT  CHAR(2),
                            CMN  CHAR(3),
                            RANG CHAR(3))
RETURNS CHAR(2)
LANGUAGE SQL
BEGIN
   DECLARE OUT CHAR(2)

   DECLARE N BIGINT
   SET N = CAST(SEXE + MOIS + DPT + CMN + RANG AS DECIMAL(38))
   SET N = 97 - (N  MOD 97)
   SET OUT = CASE
               WHEN N < 10 THEN '0' + CAST(N AS CHAR(1))
               ELSE CAST(N AS CHAR(2))
             END
   RETURN OUT

END
```

5 - L'intégrité référentielle

L'intégrité référentielle sert à empêcher qu'une ligne d'une table qui référence une ligne d'une autre table voit le lien logique entre les deux lignes brisée. Elle assure donc la cohérence de la relation qui a été héritée du modèle relationnel dans sa phase de conception.

Que serait une facture si le client venait à être effacé de la table des clients ?



Intégrité référentielle entre la table mère des utilisateurs et la table fille des "news" dans un modèle de données de gestion d'un forum de news

Le mécanisme d'intégrité référentielle doit permettre d'assurer que :

- toute ligne référencée par une autre table ne soit pas supprimé, ou alors que l'on supprime aussi toutes les lignes filles des tables qui référence la ligne supprimé de la table mère;
- la référence de la ligne mère, si elle est modifié, soit répercutée dans toutes les lignes des tables filles qui la référence, ou alors que toute modification de cette référence soit interdite si des lignes de tables filles l'utilise.

Il existe d'autres mode de gestion de la référence que nous allons détailler ci après.

La gestion de l'intégrité référentielle est de loin le point le plus important pour décider si un SGBD est relationnel ou non. Dépourvu de ce mécanisme il agit comme au bon vieux temps des fichiers COBOL. Muni de ce mécanisme il agit en responsable de l'intégrité des données. Autrement dit un SGBD qui n'est pas doté d'une mécanisme de gestion des intégrités référentielles ne mérite tout simplement pas le nom de système de gestion de bases de données "relationnelles".
Ce fut longtemps le cas de MySQL, qui avant la version 4.1 n'était autre qu'un gestionnaire de fichier dotée d'une couche d'accès aux données à travers un langage proche de SQL.

Mais l'implémentation de l'intégrité référentielle ne relève que de la responsabilité du développeur et en particulier de celui qui modélise les données. Souvent hélas on rencontre des bases de données faiblement dotées de ce mécanisme. En effet, pour certains développeurs, il parait difficile de vider des tables ou d'insérer des lignes pour les tests lorsque toutes les relations sont "matérialisées" par des contraintes de clefs étrangères.

5.1 - FOREIGN KEY et REFERENCES

La contrainte de type FOREIGN KEY permet donc de mettre en place une intégrité référentielle entre une ou plusieurs colonnes d'une table et les colonnes en regard composant la clef ou la contrainte d'unicité d'une autre table afin d'assurer les relations existantes à l'origine dans le modèle relationnel. Cela permet aussi de joindre les tables dans la requête selon le modèle relationnel que l'on a défini. On parle alors de relation maître/esclave, parent/enfant ou mère/fille.

Voici la syntaxe complète de la contrainte de clef étrangère avec les clauses de validation et de gestion de l'intégrité référentielle :

```
CONSTRAINT nom_contrainte
FOREIGN KEY ( <liste_colonne_table> )
REFERENCES table_référencée ( <liste_colonne_référencées> )
[ MATCH { FULL | PARTIAL | SIMPLE } ]
[ ON UPDATE { NO ACTION | RESTRICT | CASCADE | SET NULL | SET DEFAULT } ]
[ ON DELETE { NO ACTION | RESTRICT | CASCADE | SET NULL | SET DEFAULT } ]
```

Les colonnes composant la clef étrangère en relation avec la clef primaire ou la contrainte d'unicité de la table mère ne doivent pas forcément posséder le même nom, mais doivent avoir impérativement le même type.

Notez bien qu'une contrainte de clef étrangère peut porter soit sur une contrainte de clef primaire ou une contrainte d'unicité. En conclusion la contrainte FOREIGN KEY doit reposer sur une autre contrainte PRIMARY KEY ou UNIQUE. Elle peut aussi porter sur la propre clef primaire de sa table en vue d'établir une auto relation

5.2 - Gestion de l'intégrité référentielle

Le but de l'intégrité référentielle est de maintenir les liens entre les tables quelque soient les modifications engendrées sur les données dans l'une ou l'autre table.

Il existe en fait deux mécanismes bien distincts pour la gestion de cette contrainte. L'un concerne la mise en relation des valeurs, c'est la clause MATCH et elle agit au moment de l'insertion (INSERT). L'autre concerne l'évolution des données lors de opérations de modification ou de suppression (UPDATE, DELETE). C'est la clause ON DELETE, ON UPDATE.

5.2.1 Mode de gestion de l'intégrité, clauses ON UPDATE / ON DELETE

Au cours de la vie de la base de données, il est possible que l'on soit amené à supprimer la ligne qui sert de référence, comme à en modifier la valeur de la clef. Une telle manipulation des données de la table mère n'est pas sans conséquence dans le maintient de l'intégrité référentielle en regard des lignes filialisées.
Le mode de gestion de l'intégrité consiste à se poser la question de la règle qui doit être mise en oeuvre dans le cas ou l'on tente de modifier les données sous jacentes à une intégrité référentielle déjà établie.
En particulier SQL permet d'implémenter différents modes de gestion lorsque l'on tente de modifier des valeurs clefs de la table mère alors que des lignes de la table fille y font référence. En fait si la vérification de l'intégrité référentielle est concernée par une table fille qui tente d'insérer ou de modifier des données en référence à une table mère, la règle de gestion, elle, suppose que c'est depuis la table mère que l'on manipule les données en modification ou suppression (UPDATE, DELETE) qui conduiront à une éventuelle violation ou à une évolution des données dans la table fille.

En fait tout l''intérêt de la gestion de l'intégrité référentielle consiste à se poser la question de l'acceptation de lignes orphelines, donc sans référence à la table mère, ou veuve, c'est à dire sans lien avec la table fille.

Les différents modes de gestion de l'intégrité référentielle que propose SQL sont les suivants : NO ACTION, CASCADE, SET DEFAULT, SET NULL,

RESTRICT et ne peuvent s'appliquer qu'aux ordres SQL UPDATE (modification de données) et DELETE (suppression de lignes).

Mode	Règle	Explication
ON DELETE NO ACTION ON UPDATE NO ACTION	Toute action de modification de clef ou de suppression de ligne dans la table mère échoue pour les lignes en relation d'intégrité référentielle entre table mère et fille.	Dans le cas du NO ACTION, il y a blocage de l'ordre SQL UPDATE ou DELETE dans le but de maintenir le lien d'intégrité tel quel. Ce blocage intervient en fin de transaction.
ON DELETE RESTRICT ON UPDATE RESTRICT	Similaire à NO ACTION	le blocage intervient immédiatement sur l'ordre SQL qui a engendré le viol de la contrainte
ON DELETE CASCADE ON UPDATE CASCADE	Toute modification de clef ou suppression de ligne dans la table mère est répercutée dans la table fille.	Dans le cas du CASCADE, il y répercussion de l'ordre SQL avec modification de valeur ou suppression de lignes afin de maintenir ou d'évacuer le lien d'intégrité.
ON DELETE SET NULL ON UPDATE SET NULL	Toute action de modification ou de suppression dans la table mère est répercutée dans la table fille par la suppression des valeurs des clefs étrangères, si les colonnes en jeu sont "nullables"	Dans le cas du SET NULL, il y suppression des valeurs des clefs étrangères pour les lignes concernées, ce qui conduit à un déréférencement du lien d'intégrité.
ON DELETE SET DEFAULT ON UPDATE SET DEFAULT	Toute action de modification ou de suppression dans la table mère est répercutée dans la table fille par la mise en place des valeurs spécifiées par défaut pour les colonnes des clefs étrangères, si des valeurs par défaut ont bien été spécifiées pour toutes les colonnes en jeu.	Dans le cas du SET DEFAULT, il y mise en place des valeurs par défaut des clefs étrangères pour les lignes concernées, ce qui conduit à un glissement des valeurs de références du lien d'intégrité

Quelques remarques :

- Autant le mode NO ACTION / RESTRICT est toujours possible et s'avère être celui qui opère par défaut, autant les autres modes sont dépendants de facteurs complémentaires tel que d'autres contraintes de clef étrangères, l'absence de spécification de valeurs par défaut ou encore l'obligation de spécification de valeur (NOT NULL).
- Le mode cascade est très tentant, mais son coût de traitement peut s'avérer élevé. Mal maîtrisé il peut se révéler bloquant et devenir totalement contre performant, notamment dans le cas de cascades multiples ou de combinaisons de cascades et NO ACTION / RESTRICT.
- L'intérêt du SET NULL est de permettre la suppression des lignes devenues orphelines de manière différé, par exemple dans un traitement par lot intervenant aux heures creuses.
- L'intérêt du SET DEFAULT réside dans la possibilité de définir une référence particulière qui concentre les efforts de gestion des références obsolètes dans un traitement spécifique, par exemple le client 0 ou moi même…
- NO ACTION est le mode par défaut en l'absence de spécification.

Les modes SET NULL et SET DEFAULT sont spécialement adaptées aux VLDB dans lesquels on tente de déporter notamment les opérations de suppressions aux heures creuses. Mais cela nécessite de rajouter dans le code des requêtes d'extraction des filtres particulier dans la clause WHERE afin d'empêcher de voir ces lignes parasites.

5.2.2 Mode de validation de la référence, clause MATCH

La clause MATCH précise la manière dont la contrainte valide ou refuse l'opération d'insertion ou de mise à jour en fonction des colonnes en jeu lorsque les valeurs de certaines colonnes sont manquantes (marqueurs NULL). La clause MATCH n'a donc d'intérêt que pour des clefs étrangères multicolonnes.

Mode	Règle	Explication
MATCH SIMPLE	si toutes les colonnes contraintes sont renseignées, la contrainte s'applique si une colonne au moins possède un marqueur NULL, la contrainte ne s'applique pas	Il y a violation de contrainte si les valeurs divergent et qu'elles sont toutes renseignées
MATCH PARTIAL	La contrainte s'applique pour toutes les colonnes renseignées	Il y a violation de la contrainte si au moins une valeur renseignée diverge.
MATCH FULL	la contrainte s'applique toujours sauf si toutes les colonnes sont pourvues d'un marqueur NULL	Il y a violation de la contrainte si les valeurs divergent ou si une colonne est renseignée et l'autre pas

A défaut de la spécifier c'est l'option MATCH SIMPLE qui s'impose par défaut.

Pour mieux comprendre le fonctionnement de cette clause, voici le modèle utilisé comme exemple :

```
CREATE TABLE T_UTILISATEUR_USR
(USR_NOM            CHAR(32)     NOT NULL,
 USR_PRENOM         VARCHAR(16)  NOT NULL,
 CONSTRAINT PK_USR  PRIMARY KEY (USR_NOM, USR_PRENOM))
```

Supposons que la table T_UTILISATEUR_USR soit remplie avec les lignes suivantes :

```
USR_NOM             USR_PRENOM
------------------- ------------------
DUBOIS              Alain
DURAND              Paula
```

Exemple avec MATCH SIMPLE :

```
CREATE TABLE T_NEWS_NEW
(NEW_ID      INTEGER NOT NULL PRIMARY KEY,
 USR_NOM     CHAR(32),
 USR_PRENOM  VARCHAR(16),
 CONSTRAINT  FK_NEW_NOMPRE_MATCHSIMPLE
             FOREIGN KEY (USR_NOM, USR_PRENOM)
             REFERENCES T_UTILISATEUR_USR (USR_NOM, USR_PRENOM)
             MATCH SIMPLE)
```

Analysons maintenant les conséquences du scénario d'insertion des données suivant :

```
INSERT INTO T_NEWS_NEW (NEW_ID, USR_NOM, USR_PRENOM) VALUES (1, 'DUBOIS', 'Alain')
INSERT INTO T_NEWS_NEW (NEW_ID, USR_NOM, USR_PRENOM) VALUES (2, NULL, NULL)
INSERT INTO T_NEWS_NEW (NEW_ID, USR_NOM, USR_PRENOM) VALUES (3, 'DUBOIS', NULL)
INSERT INTO T_NEWS_NEW (NEW_ID, USR_NOM, USR_PRENOM) VALUES (4, 'DUHAMEL', NULL)
INSERT INTO T_NEWS_NEW (NEW_ID, USR_NOM, USR_PRENOM) VALUES (5, NULL, 'Paula')
-- Ces cinq lignes sont validées.
```

```
INSERT INTO T_NEWS_NEW (NEW_ID, USR_NOM, USR_PRENOM) VALUES (6, 'DUVAL', 'Marcel')
-- Violation de la contrainte FK_NEW_NOMPRE_MATCHSIMPLE lors de la tentative d'insertion
de données.
```

En fait, dans le cas du MATCH SIMPLE, la seule ligne entrant en violation est une ligne dont toutes les valeurs de clef étrangère sont à la fois renseignées et divergentes.

Exemple avec MATCH PARTIAL :

```
CREATE TABLE T_NEWS_NEW
(NEW_ID      INTEGER NOT NULL PRIMARY KEY,
 USR_NOM     CHAR(32),
 USR_PRENOM  VARCHAR(16),
 CONSTRAINT  FK_NEW_NOMPRE_MATCHPARTIAL
             FOREIGN KEY (USR_NOM, USR_PRENOM)
             REFERENCES T_UTILISATEUR_USR (USR_NOM, USR_PRENOM)
             MATCH PARTIAL)
```

Analysons maintenant les conséquences du scénario d'insertion des données suivant :

```
INSERT INTO T_NEWS_NEW (NEW_ID, USR_NOM, USR_PRENOM) VALUES (1, 'DUBOIS', 'Alain')
INSERT INTO T_NEWS_NEW (NEW_ID, USR_NOM, USR_PRENOM) VALUES (2, NULL, NULL)
INSERT INTO T_NEWS_NEW (NEW_ID, USR_NOM, USR_PRENOM) VALUES (3, 'DUBOIS', NULL)
-- Ces trois lignes sont validées.
INSERT INTO T_NEWS_NEW (NEW_ID, USR_NOM, USR_PRENOM) VALUES (4, 'DUHAMEL', NULL)
-- Violation de la contrainte FK_NEW_NOMPRE_MATCHPARTIAL lors de la tentative d'insertion
de données.
```

Dans le cas du MATCH PARTIAL, une seule divergence de données dans une seule colonne provoque la violation de la contrainte.

Exemple avec MATCH FULL :

```
CREATE TABLE T_NEWS_NEW
(NEW_ID      INTEGER NOT NULL PRIMARY KEY,
 USR_NOM     CHAR(32),
 USR_PRENOM  VARCHAR(16),
 CONSTRAINT  FK_NEW_NOMPRE_MATCHFULL
             FOREIGN KEY (USR_NOM, USR_PRENOM)
             REFERENCES T_UTILISATEUR_USR (USR_NOM, USR_PRENOM)
             MATCH FULL)
```

Analysons maintenant les conséquences du scénario d'insertion des données suivant :

```
INSERT INTO T_NEWS_NEW (NEW_ID, USR_NOM, USR_PRENOM) VALUES (1, 'DUBOIS', 'Alain')
INSERT INTO T_NEWS_NEW (NEW_ID, USR_NOM, USR_PRENOM) VALUES (2, NULL, NULL)
-- Ces deux lignes sont validées.
INSERT INTO T_NEWS_NEW (NEW_ID, USR_NOM, USR_PRENOM) VALUES (3, 'DUBOIS', NULL)
-- Violation de la contrainte FK_NEW_NOMPRE_MATCHFULL lors de la tentative d'insertion de
données.
```

Dans le cas du MATCH FULL il ne peut y avoir une colonne renseignée tandis qu'une autre ne l'est pas.

6 - Déférabilité des contraintes et exécution

L'ordre d'exécution des contraintes d'un même objet n'est pas explicitement définit par la norme. Cela laisse toute latitude aux éditeurs pour concevoir l'enchaînement des contraintes dans un ordre qui permet d'en optimiser l'exécution, notamment en commençant pas les contraintes les plus sévères et les plus restrictives, afin, si la contrainte est violée, de ne pas avoir à lancer la suivante. L'optimisation consistera donc pour le SGBDR à choisir d'exécuter en premier les contraintes les plus "filtrantes" et les moins complexes en terme de code.
Mais qu'il s'agisse de contraintes de domaine, de table ou encore d'assertions, toutes les contraintes peuvent inclure une clause de déférabilité, c'est à dire que leur application peut être différée à un moment plus opportun.
Jusqu'ici, nous avons supposé implicitement que la contrainte opérait dès que le fait générateur la déclenchait (insertion, modification ou suppression dans une table par exemple). Or SQL admet une grande souplesse en ce sens qu'il propose différents outils pour différer la validation de la contrainte. Ces outils sont la clause de déférabilité propre à chaque contrainte, et le paramétrage de la déférabilité des contraintes au sein de la base de données.

La déférabilité d'une contrainte est une opération nécessaire dès que différentes contraintes interagissent créant ainsi ce que l'on appelle une référence croisée...
Pour comprendre ce mécanisme et les modes d'utilisation qu'il offre, il est nécessaire de prendre en compte la notion de transaction, c'est à dire une combinaison de différents ordres SQL que l'on veut exécuter en séquence et d'un seul bloc, c'est à dire en tout ou rien.
Ainsi lorsqu'une table T1 fait référence à une table T2 par une intégrité référentielle, se pose le problème de la mise en place d'une intégrité référentielle inverse de T2 vers T1 (intégrité référentielle circulaire)... Nous voici confronté au problème de l'oeuf et de la poule... C'est pour trancher ce dilemme que les outils de déférabilité de contrainte ont été définis par SQL.
Par exemple, dans ce cas de figure, on ne peut insérer dans T1 que si l'on a inséré dans T2. Les opérations doivent donc être successives et combinées et si l'une d'entre elle échoue, il faut défaire les autres, ce qui se fait au travers d'une transaction SQL.

La syntaxe de la clause de déférabilité de la contrainte est la suivante :

```
[ NOT ] DEFERRABLE |
[ NOT ] DEFERRABLE { INITIALLY DEFERRED | INITIALY IMMEDIATE }
```

Option	Explication
NOT DEFERRABLE	La contrainte opère dès que l'ordre SQL qui la met en cause est appliqué. Elle ne pourra jamais être déférée. C'est l'option par défaut en l'absence de spécification.
DEFERRABLE	La contrainte opère après que l'ordre SQL qui la met en en cause se soit appliqué au moment de la validation de la transaction.
INITIALLY DEFERRED	La contrainte sera déférée à l'initiation de chaque transaction, sauf si elle a été créée en NOT DEFERRABLE. En d'autres termes elle est vérifiée en fin de transaction.
INITIALLY IMMEDIATE	La contrainte sera appliquée immédiatement à l'initiation de la transaction. En d'autre terme elle est vérifié au cours de la transaction pour chaque ordre SQL qui l'appelle.

En sus de la l'attribut de déférabilité de la contrainte, il existe un paramètre de session pour indiquer le comportement par défaut des contraintes. C'est le paramètre CONSTRAINTS, qui permet de définir le fonctionnement global des contraintes au sein du schéma. Il se spécifie à l'aide de la syntaxe suivante :

```
SET CONSTRAINTS { <liste_de_contraintes> | ALL }
[ DEFERRED | IMMEDIATE ]
```

Il n'opère que pour les contraintes qui n'ont pas été définies en tant que NOT DEFFERABLE.

Imaginons que nous voulons modéliser un client et ses commandes et placer dans la table du client la dernière commande servie, cela afin d'éviter de placer des clients sans commande dans la table des clients (les clients sans commandes étant placé naturellement dans la table des prospects...). Le problème se pose ainsi : comment insérer un nouveau client qui, par définition, n'a pas encore de commande, alors que l'on exige dans la table client de faire référence à la dernière commande ?

Nous pouvons concevoir cela à l'aide des descriptions de table suivante :

```
CREATE TABLE T_CLIENT_CLI
(CLI_ID      INTEGER NOT NULL PRIMARY KEY,
 CLI_NOM     CHAR(32),
 CDE_ID      INTEGER NOT NULL,
 CONSTRAINT  FK_CLI_CDE FOREIGN KEY (CDE_ID)
             REFERENCES T_COMMANDE (CDE_ID));

CREATE TABLE T_COMMANDE_CDE
(CDE_ID      INTEGER NOT NULL PRIMARY KEY,
 CDE_DATE    DATE,
 CLI_ID      INTEGER NOT NULL,
 CONSTRAINT  FK_CDE_CLI FOREIGN KEY (CLI_ID)
             REFERENCES T_CLIENT (CLIK_ID));
```

Or, si le lancement de ce script SQL de création des tables au sein d'un schéma permet bien de d'instancier les objets, il s'avère en revanche impossible d'y rentrer la moindre donnée.

Dès lors, deux moyens sont possibles pour résoudre ce dilemme : modifier l'une des contraintes en la déférant en fin de transaction, ou bien paramétrer le SGBDR pour qu'il valide toutes ou partie des contraintes à la fin de la transaction.

Solution n°1

A la création des objets du schéma :

```
CREATE TABLE T_CLIENT_CLI
(CLI_ID      INTEGER NOT NULL PRIMARY KEY,
 CLI_NOM     CHAR(32),
 CDE_ID      INTEGER NOT NULL,
 CONSTRAINT  FK_CLI_CDE FOREIGN KEY (CDE_ID)
             REFERENCES T_COMMANDE (CDE_ID));

CREATE TABLE T_COMMANDE_CDE
(CDE_ID      INTEGER NOT NULL PRIMARY KEY,
 CDE_DATE    DATE,
 CLI_ID      INTEGER NOT NULL,
 CONSTRAINT  FK_CDE_CLI FOREIGN KEY (CLI_ID)
             REFERENCES T_CLIENT (CLIK_ID)
             DEFERRABLE INITIALLY DEFFERED);
```

Dans la transaction d'insertion combinée :

```
INSERT INTO T_CLIENT_CLI (CLI_ID, CLI_NOM )
                  VALUES (101,    'DUPONT');

INSERT INTO T_COMMANDE_CDE (CDE_ID, CDE_DATE,     CLI_ID)
                     VALUE (4589,   '2005-09-01', 101);

UPDATE T_CLIENT_CLI
SET    CDE_ID = 4589
WHERE  CLI_ID = 101;

COMMIT;
```

Solution n°2

A la création des objets du schéma :

```
CREATE TABLE T_CLIENT_CLI
(CLI_ID      INTEGER NOT NULL PRIMARY KEY,
 CLI_NOM     CHAR(32),
 CDE_ID      INTEGER NOT NULL,
 CONSTRAINT  FK_CLI_CDE FOREIGN KEY (CDE_ID)
             REFERENCES T_COMMANDE (CDE_ID));

CREATE TABLE T_COMMANDE_CDE
(CDE_ID      INTEGER NOT NULL PRIMARY KEY,
 CDE_DATE    DATE,
 CLI_ID      INTEGER NOT NULL,
 CONSTRAINT  FK_CDE_CLI FOREIGN KEY (CLI_ID)
             REFERENCES T_CLIENT (CLIK_ID)
             DEFERRABLE);
```

Au démarrage de la session (par exemple) :

```
SET CONSTRAINTS FK_CDE_CLI DEFERRED
```

Dans la transaction d'insertion combinée :

```
INSERT INTO T_CLIENT_CLI (CLI_ID, CLI_NOM )
                  VALUES (101,    'DUPONT');

INSERT INTO T_COMMANDE_CDE (CDE_ID, CDE_DATE,     CLI_ID)
                     VALUE (4589,   '2005-09-01', 101);

UPDATE T_CLIENT_CLI
SET    CDE_ID = 4589
WHERE  CLI_ID = 101;

COMMIT;
```

La déférabilité d'une contrainte est le seul élément du langage capable de créer un "auto rollback", c'est à dire une dévalidation automatique des ordres SQL passés.
Attention : certains SGBDR valident les contraintes pour chaque ligne ce qui n'est pas conforme aux spécification du langage SQL pour laquelle toute requête est une transaction.

7 - Implémentation des contraintes chez les différents éditeurs

Afin de vous aider à choisir le bon SGBDR devant un développement nécessitant l'implémentation de contraintes complexes, voici les points à vérifier :

Nature de contrainte	Type de contrainte	Particularité	Nota
DOMAINE	CHECK		Ou mécanisme similaire dont les UDT (User Data Type introduite avec le relationnel objet, norme SQL:1999)
ASSERTION	CHECK		Ou triggers before et after, sinon triggers after avec posibilité de rollback. Voir aussi si les contraintes CHECH de table peuvent comporter l'appel à d'autre table que la table cible.
TABLE	NOT NULL		
	PRIMARY KEY		
	FOREIGN KEY	Clauses MATCH et ON UPDATE / ON DELETE	Peut éventuellement être opéré par trigger
	UNIQUE		Vérifier la mutivaluation du NULL
	CHECK		Vérifiez si la portée de la contrainte de validation est uniquement la ligne ou va jusquà la table entière, voir la base...
Vérifiez enfin la déferrabilité des contraintes			

8 Conclusion

Comme vous avez pu le constater, le monde des contraintes SQL est vaste et peut rapidement devenir complexe. Mais ce qu'il y a de pire c'est l'absence de contraintes. Les études entreprises aux états unis sur l'implémentation de data warehouse et de solutions de décisionnels (data mart) montre que le coût de nettoyage des données lorsque les données de la base sont de piètre

qualité – du fait de la quasi absence de contraintes – peut aller jusqu'à 50% du coût global de la solution...
De quoi faire réfléchir n'importe quel décideur informatique !

Une colle pour terminer

Pour vous exercer, voici un problème à résoudre sous forme de contrainte SQL. Cet exercice est inspiré des fameux puzzles SQL de Joe Celko...
Une compétition de schnorkelzig[3] ne comporte jamais que quatre compétiteurs. Dans la présente compétition, les athlètes sont Paul, Marc, Luc et Jean.
Chaque membre du jury, et ils sont très nombreux, doit voter de manière à indiquer quel serait l'ordre dans lequel il voudrait placer ses champions.
Par exemple, le jury DUPONT aimerait que le podium soit : 1er Jean, 2nd Marc, 3e Luc, 4e Paul. Mais les membres du jury peuvent ne voter que pour certaines personnes laissant ainsi le vote incomplet. Ainsi le jury MARTIN, aimerait le podium suivant : 1er Luc, 2nd Paul, 3e Marc. Dans ce cas Jean aurait un marqueur NULL (absence de valeur).

La table destinée à recevoir les votes est ainsi constituée :

```
CREATE TABLE schnorkelzig
(JURY  VARCHAR(16) NOT NULL PRIMARY KEY,
 Jean  INT,
 Marc  INT,
 Paul  INT,
 Luc   INT)
```

[3] Ne me demander pas ce qu'est le schnorkelzig. J'en suis incapabale. J'ai inventé ce mot pour les besoins de cet exemple afin de ne faire aucun tort aux afficionados de tel ou tel sport !

Votre mission, si vous l'acceptez, sera d'écrire les contraintes nécessaires à valider la saisie et empêcher notamment des erreurs telles que :

JURY	Jean	Marc	Paul	Luc	
MOULIN	1	3	5	11	*--> restreindre le vote à 1,2,3,4,5*
DUVAL	1	NULL	3	4	*--> séquence incorrecte, devrait être 1, NULL, 2, 3*
SCHMIDT	1	1	2	3	*--> doublon (présence du 1 deux fois)*
FOSTER	NULL	NULL	NULL	NULL	*--> il doit y avoir au moins un vote*

Bien entendu les contraintes à poser doivent être écrites de la manière la plus élégante possible.

Envoyez moi vos réponses à : sqlpro@club-internet.fr

Références :

- SQL, synthèses de cours et exercice – F. Brouard et C. Soutou – Pearson Education 2006 (2e édition,)
- SQL, développement – F. Brouard – Campus Press 2001 (épuisé en version papier, téléchargeable en version électronique)
- SQL:1999 Understanding Relational Language Components - Jim Melton, Alan R. Simon - Morgan Kaufmann 2002
- http://sqlpro.developpez.com