Création, modification et suppression

1. Les tables

La voila la grosse partie du DDL qui vous passionne. Alors otons nous tout de suite une épine du pied en définissant la syntaxe de la création de table :

CREATE [{ GLOBAL | LOCAL } TEMPORARY] TABLE nom_table (colonne | contrainte_de_table [{ , colonne | contrainte_de_table }...])
colonne :: nom_colonne { type | domaine } [DEFAULT valeur_default] [contrainte_de_colonne...] [COLLATE collation]
contrainte_de_colonne :: [CONSTRAINT nom_contrainte] [NOT] NULL | UNIQUE | PRIMARY KEY | CHECK (prédicat_de_colonne) | FOREIGN KEY [colone] REFERENCES table (colonne) spécification_référence
contrainte_de_table :: CONSTRAINT nom_contrainte { UNIQUE | PRIMARY KEY (liste_colonne) | CHECK (prédicat_de_table) | FOREIGN KEY liste colonne REFERENCES nom_table (liste_colonne) spécification_référence }

C'est une des syntaxes les plus simples que j'ai pu trouver pour vous montrer l'ensemble des possibilités qu'offre la norme SQL pour créer une table.

En gros, disons que :

- une table peut être créée de manière durable (par défaut) ou temporaire et dans ce dernier cas uniquement pour l'utilisateur et la connexion qui l'a créé ou bien pour l'ensemble des utilisateurs de la base
- une table comporte des colonnes et des contraintes de table
- une colonne peut être spécifiée d'après un type SQL ou un domaine créé par l'utilisateur
- une colonne définie peut être dotée de contraintes de colonnes telles que : obligatoire, clef, unicité, intégrité référentielle et validation
- une contrainte de table porte sur une ou plusieurs colonnes et permet l'unicité, la validation et l'intégrité référentielle

RAPPEL : un nom de colonne doit être unique au sein de la table

Voici quelques exemple de création de table.

```
CREATE TABLE T_CLIENT
(CLI_NOM    CHAR(32),
 CLI_PRENOM VARCHAR(32))
```
Une table de clients dotée de deux colonnes avec les nom et prénom des clients.

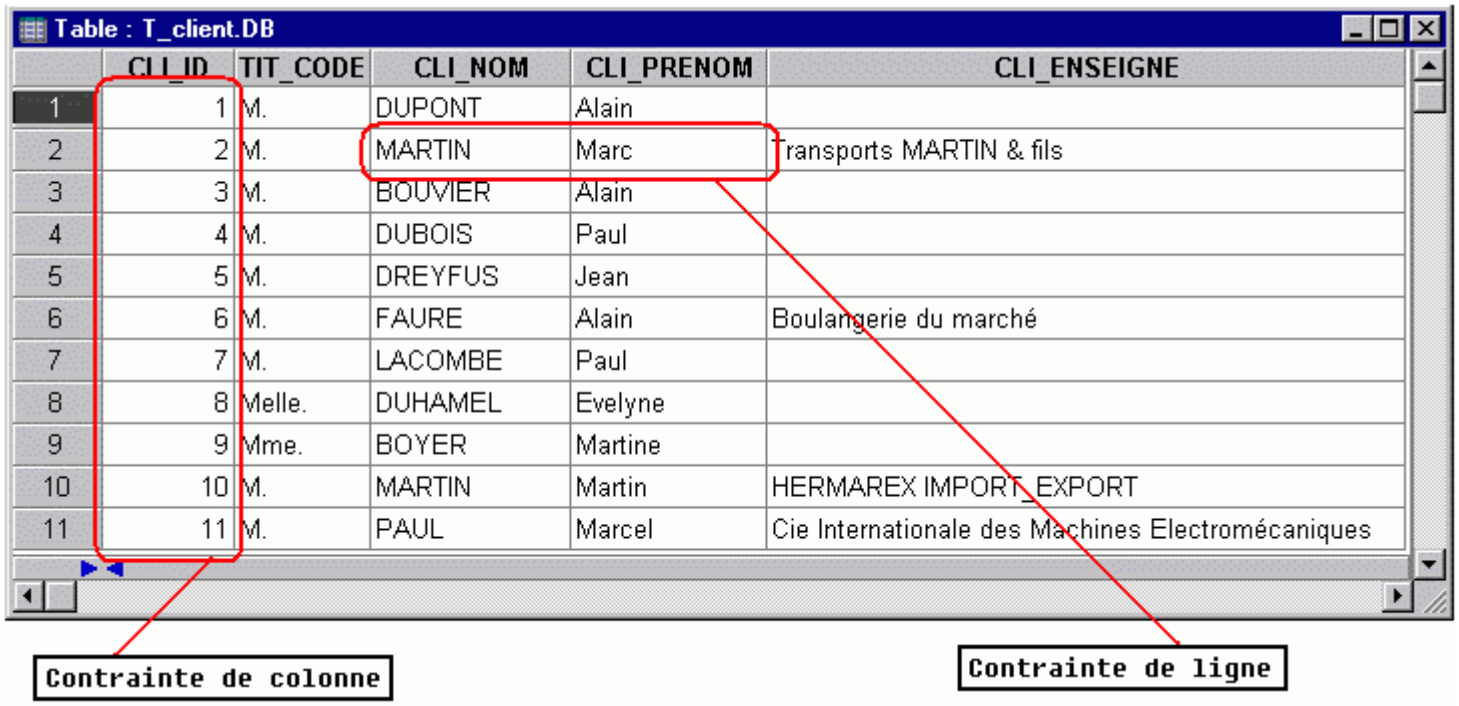
```
CREATE TABLE T_CLIENT
(CLI_ID     INTEGER      NOT NULL PRIMARY KEY,
 CLI_NOM    CHAR(32)     NOT NULL,
 CLI_PRENOM VARCHAR(32))
```
Une table de clients dotée de trois colonnes avec la clef (numéro du client) les nom et prénom des clients.

```
CREATE TABLE T_CLIENT
(CLI_ID     INTEGER      NOT NULL PRIMARY KEY,
 CLI_NOM    CHAR(32)     NOT NULL CHECK (SUBSTRING(VALUE, 1, 1) <> ' ' AND UPPER(VALUE) = VALUE),
 CLI_PRENOM VARCHAR(32)  REFERENCES TR_PRENOM (PRN_PRENOM))
```
Une table de clients dont le nom ne peut commencer par un blanc, doit être en majuscule et dont le prénom doit figurer dans la table de référence TR_PRENOM à la colonne PRN_PRENOM.

```
CREATE TABLE T_VOITURE
(VTR_ID             INTEGER      NOT NULL PRIMARY KEY,
 VTR_MARQUE         CHAR(32)     NOT NULL,
 VTR_MODELE         VARCHAR(16),
 VTR_IMMATRICULATION CHAR(10)    NOT NULL UNIQUE,
 VTR_COULEUR        CHAR(16)     CHECK (VALUE IN ('BLANC', 'NOIR', 'ROUGE', 'VERT', 'BLEU')))
```
Une table de voiture avec immatriculation unique et couleur limitée à 'BLANC', 'NOIR', 'ROUGE', 'VERT', 'BLEU'.

```
CREATE TABLE T_CLIENT
(CLI_NOM    CHAR(32)     NOT NULL,
 CLI_PRENOM VARCHAR(32)  NOT NULL,
 CONSTRAINT PK_CLIENT PRIMARY KEY (CLI_NOM, CLI_PRENOM))
```
Une table de clients dont la clef est le couple de colonne NOM/PRENOM.

Nous allons maintenant détailler les différentes contraintes dites verticales ou horizontales suivant qu'il s'agit de contrainte de colonne ou de contrainte de ligne. Cette notion de vertical et horizontal fait référence à la visualisation des données de la table :



Une contrainte de colonne est dite verticale parce qu'elle porte sur une seule colonne. Dans la figure ci dessus la contrainte de colonne est une clef (PRIMARY KEY).
Une contrainte de ligne est dite horizontale parce qu'elle porte sur plusieurs colonne et se valide pour chaque ligne insérée.

La différence entre contrainte de ligne (horizontale) et contrainte de colonne (verticale) est purement terminologique puisque certaines contraintes peuvent être définies horizontalement comme verticalement :

contrainte	[NOT] NULL	DEFAULT	COLLATE	PRIMARY KEY	UNIQUE	CHECK	FOREIGN KEY
colonne (verticale)	OUI	OUI	OUI	OUI	OUI	OUI	OUI
ligne (horizontale)	NON	NON	NON	OUI	OUI	OUI	OUI

REMARQUE : notons que toute contrainte peut être déférrée, c'est à dire que ses effets peuvent être suspendue pour ne jouer qu'à la fin d'une transaction plutôt qu'à chaque ordre SQL sensée la solliciter.

1.1. Les contraintes de colonnes (verticales)

Une colonne peut donc recevoir les contraintes suivantes :

- **NULL / NOT NULL** : précise si une valeur doit obligatoirement être saisie dans la colonne ou non
- **DEFAULT** : valeur par défaut qui est placée dans la colonne lors des insertions et de certaines opération particulières, lorsque l'on a pas donné de valeur explicite à la colonne
- **COLLATE** : précise la séquence de collation, c'est à dire l'ordre des caractères pour le tri et les évnetuelles confusions possible (minuscules/majuscules, caractères diacritiques distinct ou non). Voir paragraphe 4 à ce sujet
- **PRIMARY KEY** : précise si la colonne est la clef de la table. ATTENTION : nécessite que la colonne soit NOT NULL
- **UNIQUE** : les valeurs de la colonne doivent être unique ou NULL, c'est à dire qu'à l'exception du marqueur NULL, il ne doit jamais y avoir plus d'une fois la même valeur (pas de doublon)
- **CHECK** : permet de préciser un prédicat qui acceptera la valeur s'il est évalué à vrai
- **FOREIGN KEY** : permet, pour les valeurs de la colonne, de faire référence à des valeurs préexitantes dans une colonne d'une autre table. Ce mécanisme s'apelle intégrité référentielle

NOTA : toutes ces contraintes peuvent être placées dans plusieurs colonnes, à l'exception de la contrainte de clef PRIMARY KEY qui ne peut être placée que sur une seule colonne. Pour faire de plusieurs colonnes une clef, il faut utiliser une contrainte de ligne (horizontale).

Lorsqu'au cours d'un ordre SQL d'insertion, de modification ou de suppression, une contrainte n'est pas vérifiée on dit qu'il y a "violation" de la contrainte et les effets de l'ordre SQL sont totalement annulé (ROLLBACK).

REMARQUE : le mot clef CONSTRAINT comme le nom de la contrainte n'est pas obligatoire dans le cas de contraintes de colonnes.

1.1.1. Obligatoire ([NOT] NULL)

On peut rendre la saisie d'une colonne obligatoire en apposant le mot clef NOT NULL. Dans ce cas, il ne sera jamais possible de faire en sorte que la colonne soit vide. Autrement dit, la colonne devra toujours être renseigné lors des ordres d'insertion INSERT et de modification UPDATE.
Si l'on désire que la colonne puisse ne pas être renseignée (donc accepter les marqueurs NULL), il n'est pas nécessaire de préciser le mot clef NULL, mais il est courrant qu'on le fasse par facilité de lecture.

```
CREATE TABLE T_PERSONNE1
(PRS_ID            INTEGER     NOT NULL
 PRS_NOM           VARCHAR(32) NOT NULL,
 PRS_PRENOM        VARCHAR(32) NULL,
 PRS_DATE_NAISSANCE DATE)
```

Crée une table dont les colonnes PRS_ID et PRS_NOM doivent obligatoirement être renseignés.

insertion et modification acceptées :

```
INSERT INTO T_PERSONNE1 VALUES (1, 'DUPONT', NULL, NULL)
INSERT INTO T_PERSONNE1 (PRS_ID, PRS_NOM) VALUES (2, 'DURAND')
```

insertion et modification refusées :

```
INSERT INTO T_PERSONNE1 VALUES (3, NULL, 'Marcel', NULL)
INSERT INTO T_PERSONNE1 (PRS_ID, PRS_PRENOM) VALUES (4, 'Jean')
```

NOTA : les colonnes concourrant à la définition d'une clef de table doivent impérativement possèder une contrainte NOT NULL.

1.1.2. Valeur par défaut (DEFAULT)

La contrainte DEFAULT permet de préciser une valeur qui sera automatiquement insérée en l'absence de précision d'une valeur explicite dans un ordre d'insertion. Certains autres ordres SQL, comme la gestion de l'intégrité référentielle peuvent faire référence à cette valeur par défaut. Seule une valeur explicite, un marqueur NULL ou la valeur retournée par les fonctions suivantes sont acceptées : CURRENT_DATE, CURRENT_TIME[(p)], CURRENT_TIMESTAMP[(p)], LOCALTIME[(p)], LOCALTIMESTAMP[(p)], USER, CURRENT_USER, SESSION_USER, SYSTEM_USER.

```
CREATE TABLE T_PERSONNE2
(PRS_ID           INTEGER,
 PRS_NOM          VARCHAR(32),
 PRS_PRENOM       VARCHAR(32),
 PRS_SEXE         CHAR(1)    DEFAULT 'M',
 PRS_DATE_NAISSANCE DATE       DEFAULT CURRENT_DATE)
```

NOTA : il n'est pas possible de préciser une valeur par défaut qui soit le résultat d'une expression de requête.

```
CREATE TABLE T_PERSONNE3
(PRS_ID           INTEGER DEFAULT (SELECT MAX(PRS_ID) + 1
                          FROM T_PERSONNE3),
 PRS_NOM          VARCHAR(32),
 PRS_PRENOM       VARCHAR(32))
```

Qui pourrait s'avérer bien utile pour générer de nouvelles valeurs de clefs auto incrémentées !

1.1.4. Clef (PRIMARY KEY)

Selon le docteur Codd, toute table doit être munie d'une clef (souvent apellé à tort clef primaire en opposition à clef étrangère...). Et toujours selon le docteur Codd et la théorie des bases de données, une clef doit impérativement toujours être pourvue d'une valeur ! (sinon à quoi servirait une clef en l'absence de serrure ?).
Lorsque la clef porte sur une seule colonne il est possible de donner à cette colonne la contrainte PRIMARY KEY.

Nous avons vu que la contrainte PRIMARY KEY peut être posée sur une colonne (contrainte verticale) ou sur plusieurs colonnes en contrainte de ligne (horizontale). Si nous choisissons de la poser en contrainte de colonne, alors une seule colonne de la table peut en bénéficier.

```
CREATE TABLE T_PERSONNE5
(PRS_ID           INTEGER    NOT NULL PRIMARY KEY,
 PRS_NOM          VARCHAR(32),
 PRS_PRENOM       VARCHAR(32))
```

La contrainte PRIMARY KEY assure qu'il n'y aura aucune valeur redondante (doublon) dans la colonne. La contrainte complémentaire NOT NULL assure qu'il y aura toujours une valeur. Toute tentative d'insérer une valeur préexistante de la colonne se soldera par une violation de contrainte de clef. Voici par exemple le message généré par SQL Server dans ce cas :

```
Violation de la contrainte PRIMARY KEY 'PK__T_PERSONNE5__45F365D3'.
Impossible d'insérer une clé en double dans l'objet 'T_PERSONNE5'.
L'instruction a été arrêtée.
```

NOTA : il est d'usage de placer la colonne clef en tête de la description de la table pour des fins de lisibilité.

1.1.5. Unicité (UNIQUE)

La contrainte d'unicité exige que toutes les valeurs explicites contenues dans la colonne soient uniques au sein de la table. En revanche, la colonne peut ne pas être renseignée. En effet, souvenez vous que les marqueurs NULL se propagent dans les calculs et donc comparaison d'un marqueur NULL à un ensemble de valeurs est impossible et se solde par le renvoi d'un marqueur UNKNOW à la place des valeurs TRUE ou FALSE attendue.

```
CREATE TABLE T_PERSONNE7
(PRS_NOM           VARCHAR(32),
 PRS_PRENOM        VARCHAR(32),
 PRS_TELEPHONE     CHAR(14)    UNIQUE)
```

```
INSERT INTO T_PERSONNE7 VALUES ('Dupont', 'Marcel',  '01 44 21 57 18')
INSERT INTO T_PERSONNE7 VALUES ('Duval',  'André',    NULL)
INSERT INTO T_PERSONNE7 VALUES ('Durand', 'Jean',    '06 11 86 46 69')
INSERT INTO T_PERSONNE7 (PRS_NOM, PRS_PRENOM) VALUES ('Dubois',
'Claude')
INSERT INTO T_PERSONNE7 VALUES ('Dugland', 'Alfred', '06 11 86 46 69')
```

```
Violation de la contrainte UNIQUE KEY 'UQ__T_PERSONNE7__47DBAE45'.
Impossible d'insérer une clé en double dans l'objet 'T_PERSONNE7'.
L'instruction a été arrêtée.
```

```
SELECT *
FROM T_PERSONNE7
```

```
PRS_NOM                    PRS_PRENOM                 PRS_TELEPHONE
-------------------------------- -------------------------------- --------------
Dupont                      Marcel                      01 44 21 57 18
Duval                      André                     NULL
Durand                      Jean                     06 11 86 46 69
Dubois                      Claude                      NULL
```

Dans cet exemple Dugland n'a pas été inséré car son numéro de téléphone est identique à Durand.

REMARQUE : certains SGBD comme MS SQL Server refuse de voir la présence de plusieurs marqueurs NULL dans la cas d'une contrainte d'unicité. D'autres comme InterBase refusent une contrainte d'unicité dépourvue d'une contrainte NOT NULL...

ATTENTION : vous ne pouvez pas définir une contrainte d'unicité sur des colonnes de type BLOB

1.1.6. Validation (CHECK)

La contrainte CHECK de validation est celle qui offre le plus de possibilité. En contre partie son exécution est très coûteuse. Elle permet de définir un prédicat complexe, basé sur une comparaison pouvant contenir une requête de type SELECT. Pour valider la contrainte, le prédicat doit être évalué à TRUE ou UNKNOWN (présence de NULL).

Sa syntaxe est :
CHECK (prédicat)
où prédicat peut contenir le mot clef VALUE pour faire référence à la colonne pour laquelle la contrainte est définie.

```
CREATE TABLE T_PERSONNE8
(PRS_ID            INTEGER     CHECK (VALUE > 0),
 PRS_NOM           VARCHAR(32) CHECK (CHARACTER_LENGTH(VALUE) > 2),
 PRS_PRENOM        VARCHAR(32) CHECK (COALESCE(SUBSTRING(VALUE, 1, 1), 'X') BETWEEN 'A' AND 'Z'),
 PRS_SEXE          CHAR(1)     CHECK (VALUE IN ('M', 'F')),
 PRS_TELEPHONE     CHAR(14)    CHECK (SUBSTRING(VALUE, 1, 3) IN (SELECT PREFIXE FROM T_NUM_TEL) OR
IS NULL))
```

La colonne PRS_ID ne peut avoir de valeurs inférieures à 0.
La colonne PRS_NOM doit avoir des valeurs contenant au moins 2 caractères.
Le premier caractère de la colonne PRS_PRENOM, si elle est renseigné, doit être compris entre A et Z.
La colonne PRS_SEXE peut avoir exclusivement les valeurs M ou F.
Les trois premiers caractères de la colonne PRS_TELEPHONE si elle est renseignée doit correspondre à une valeur se trouvant dans la colonne PREFIXE de la table T_NUM_TEL.

ATTENTION : la longueur du prédicat d'une contrainte CHECK (en nombre de caractères) peut être limité. Il faut en effet pouvoir stocker cette contrainte dans le dictionnaire des informations de la base et ce dernier n'est pas illimité.

1.1.7. Intégrité référentielle (FOREIGN KEY / REFERENCES)

La contrainte de type FOREIGN KEY permet de mettre en place une intégrité référentielle entre une (ou plusieurs) colonnes d'une table et la (ou les) colonne composant la clef d'une autre table afin d'assurer les relations existantes et joindre les tables dans le requête selon le modèle relationnel que l'on a défini.
Le but de l'intégrité référentielle est de maintenir les liens entre les tables quelque soit les modiifications engendrées sur les données dans l'une ou l'autre table.

Cette contrainte dans sa syntaxe complète est assez complexe et c'est pourquoi nous allons dans ce paragraphe donner une syntaxe très simplifié à des fins didactiques :

FOREIGN KEY REFERENCES table (colonne)
La syntaxe complète de la clause FOREIGN KEY sera vue au paragraphe 7.3.

ATTENTION : la colonne spécifiée comme référence doit être une colonne clef.

```
CREATE TABLE T_FACTURE1
(FTC_ID          INTEGER,
 PRS_ID          INTEGER     FOREIGN KEY REFERENCES T_PERSONNE5 (PRS_ID) ,
 FCT_DATE        DATE,
 FCT_MONTANT     DECIMAL(16,2))
```

La table T_FACTURE1 est liée à la table T_PERSONNE5 et ce lien se fait entre la clef étrangère PRS_ID de la table T_FACTURE1 et la clef de la table T_PERSONNE5 qui s'intitule aussi PRS_ID.

NOTA : il est très important que les noms des colonnes de jointure soit les mêmes dans les différentes tables (notamment à cause du NATURAL JOIN), mais cela n'est pas obligatoire.

Dès lors toute tentative d'insertion d'une facture dont la référence de client est inexistante se soldera par un échec. De même toute tentative de supprimer un client pour lequel les données d'une ou de plusieurs factures sont présente se soldera par un arrêt sans effet de l'ordre SQL.

Examinons maintenant comment le SGBD réagit pour assurer la cohérence de la base lors d'opérations tenant de briser les liens d'intégrité référentielle :

```
INSERT INTO T_PERSONNE5 VALUES (1, 'Dupont', 'Marcel')
INSERT INTO T_PERSONNE5 VALUES (2, 'Duval', 'André')
```

```
INSERT INTO T_FACTURE1 VALUES (1, 1, '2002-03-15', 1256.45)
INSERT INTO T_FACTURE1 VALUES (1, 2, '2002-04-22', 7452.89)
```

Tentative d'insertion d'une facture dont la personne n'est pas référencé dans la table T_PERSONNE5 : `INSERT INTO T_FACTURE1 VALUES (1, 3, '2002-03-15', 1256.45)`	Conflit entre l'instruction INSERT et la contrainte COLUMN FOREIGN KEY 'FK__T_FACTURE__PRS_I__5165187F'. Le conflit est survenu dans la base de données 'DB_HOTEL', table 'T_PERSONNE5', column 'PRS_ID'. L'instruction a été arrêtée.
Tentative de suppression d'une personnet possédant encore des factures `DELETE FROM T_PERSONNE5 WHERE PRS_NOM = 'Dupont'`	Conflit entre l'instruction DELETE et la contrainte COLUMN REFERENCE 'FK__T_FACTURE__PRS_I__5165187F'. Le conflit est survenu dans la base de données 'DB_HOTEL', table 'T_FACTURE1', column 'PRS_ID'. L'instruction a été arrêtée.

REMARQUE : Comme on le voit, le mécanisme d'intégrité référentielle est un élément indispensable au maintient des relations entre tables. Un SGBD qui en est dépourvu ne peut pas prétendre à gérer le relationnel. En particulier MySQL ne peut en aucun cas prétendre être une base de données relationnelle !

1.2. Les contraintes de table

Une table peut être pourvue des contraintes de ligne suivante :

- **PRIMARY KEY** : précise que la ou les colonnes composent la clef de la table. ATTENTION : nécessite que chaque colonne concourrant à la clef soit NOT NULL.
- **UNIQUE** : les valeurs de la ou les colonnes doivent être unique ou NULL, c'est à dire qu'à l'exception du marqueur NULL, il ne doit jamais y avoir plus d'une fois la même valeur (pas de doublon) au sein de l'ensemble de données formé par les valeurs des différentes colonnes composant la contrainte.
- **CHECK** : permet de préciser un prédicat validant différentes colonnes de la table et qui accepterons les valeurs s'il est évalué à vrai.
- **FOREIGN KEY** : permet, pour les valeurs de la ou les colonnes, de faire référence à des valeurs préexitantes dans une ou plusieurs colonnes d'une autre table. Ce mécanisme s'apelle intégrité référentielle.

Comme dans le cas des contraintes de colonne, lorsqu'au cours d'un ordre SQL d'insertion, de modification ou de suppression, une contrainte n'est pas vérifiée on dit qu'il y a "violation" de la contrainte et les effets de l'ordre SQL sont totalement annulé (ROLLBACK).

1.2.1. Clef multicolonne (PRIMARY KEY)

La clef d'une table peut être composée de plusieurs colonnes. Dans ce cas la syntaxe est :

CONSTRAINT nom_contrainte PRIMARY KEY (liste_colonne)
Exemple 76 - clef primaire sur PRS_NOM / PRS_PRENOM

```
CREATE TABLE T_PERSONNE9
(PRS_NOM           VARCHAR(32) NOT NULL,
 PRS_PRENOM        VARCHAR(32) NOT NULL,
 PRS_TELEPHONE     CHAR(14),
 CONSTRAINT PK_PRS PRIMARY KEY (PRS_NOM, PRS_PRENOM))
```

1.2.2. Unicité globale (UNIQUE)

Un contrainte d'unicité peut être portée sur plusieurs colonnes. Dans ce cas chaque n-uplets de valeurs explicite doit être différents. Dans ce cas la syntaxe est :

CONSTRAINT nom_contrainte UNIQUE (liste_colonne)
définition d'une clef unique sur PRS_NOM / PRS_PRENOM

```
CREATE TABLE T_PERSONNE10
(PRS_ID            INTEGER,
 PRS_NOM           VARCHAR(32),
 PRS_PRENOM        VARCHAR(32),
 CONSTRAINT UNI_NOM_PRENOM UNIQUE (PRS_NOM,
PRS_PRENOM))
```

```
INSERT INTO T_PERSONNE10 VALUES (1, 'Dupont', 'Marcel')
INSERT INTO T_PERSONNE10 VALUES (2, 'Duval',  'André')
INSERT INTO T_PERSONNE10 VALUES (3, 'Dupond',  NULL)
INSERT INTO T_PERSONNE10 VALUES (4, NULL, NULL)
INSERT INTO T_PERSONNE10 VALUES (5, NULL, 'Alfred')
INSERT INTO T_PERSONNE10 VALUES (6, 'Duval',  'André')
```

```
Violation de la contrainte UNIQUE KEY 'UNI_NOM_PRENOM'.
Impossible d'insérer une clé en double dans l'objet 'T_PERSONNE10'.
L'instruction a été arrêtée.
```

REMARQUE : certains SGBD comme MS SQL Server refuse de voir la présence de plusieurs marqueurs NULL dans la cas d'une contrainte d'unicité. D'autres comme InterBase refusent une contrainte d'unicité dépourvue d'une contrainte NOT NULL...

1.2.3. Validation de ligne (CHECK)

La contrainte CHECK permet d'effectuer un contrôle de validation multicolonne au sein de la table.

Sa syntaxe est :

CONSTRAINT nom_contrainte CHECK (prédicat)
vérification de présence d'information dans au moins une colonne crédit ou débit de la table compte :

```
CREATE TABLE T_COMPTE
(CPT_ID            INTEGER,
 CPT_DATE          DATE,
 CPT_CREDIT        DECIMAL (16,2),
 CPT_DEBIT         DECIMAL (16,2),
 CLI_ID            INTEGER,
 CONSTRAINT CHK_OPERATION CHECK((CPT_CREDIT >= 0 AND CPT_DEBIT IS NULL) OR (CPT_DEBIT >= 0 AND
CPT_CREDITIS NULL)))
```

Toute tentative d'insérer une ligne avec des valeurs non renseignées pour les colonnes debit et credit, ou bien avec des valeurs négative se soldera par un refus.

1.2.4. Integrité référentielle de table (FOREIGN KEY / REFERENCES)

Comme dans la cas d'une contrainte référentielle de colonne, il est possible de placer une contrainte d'intégrité portant sur plusieurs colonne. Ceci est d'autant plus important qu'il n'est pas rare de trouver des tables dont la clef est composée de plusieurs colonnes. La syntaxe est la suivante :

CONSTRAINT nom_contrainte FOREIGN KEY (liste_colonne) REFERENCES nom_table_ref (liste_colonne_ref)

```
CREATE TABLE T_FACTURE2
(FTC_ID            INTEGER,
 PRS_NOM           VARCHAR(32),
 PRS_PRENOM        VARCHAR(32),
 FCT_DATE          DATE,
 FCT_MONTANT       DECIMAL(16,2),
 CONSTRAINT FK_FCT_PRS FOREIGN KEY (PRS_NOM, PRS_PRENOM) REFERENCES T_PERSONNE9 (PRS_NOM,
PRS_PRENOM))
```

La table T_FACTURE2 est liée à la table T_PERSONNE9 et ce lien se fait entre la clef étrangère composite PRS_NOM / PRS_PRENOM de la table T_FACTURE2 et la clef de la table T_PERSONNE9 elle même composée des colonnes PRS_NOM / PRS_PRENOM.

Examinons maintenant comment le SGBD réagit pour assurer la cohérence de la base lors d'opérations tenant de briser les liens d'intégrité référentielle :

```
INSERT INTO T_PERSONNE9 VALUES ('Dupont', 'Marcel', '01 45 78 74 25')
INSERT INTO T_PERSONNE9 VALUES ('Duval', 'André', NULL)
```

```
INSERT INTO T_FACTURE2 VALUES (1, 'Dupont', 'Marcel', '2002-03-15', 1256.45)
INSERT INTO T_FACTURE2 VALUES (1, 'Duval', 'André', '2002-04-22', 7452.89)
```

Tentative d'insertion d'une facture dont la personne n'est pas référencé dans la table T_PERSONNE5 :

```
INSERT INTO T_FACTURE1
VALUES (1, 'Dubois', 'Maurice', '2002-03-15', 1256.45)
```

```
Conflit entre l'instruction INSERT et la contrainte
TABLE FOREIGN KEY 'FK_FCT_PRS'. Le conflit est survenu
dans la base de données 'DB_HOTEL', table 'T_PERSONNE9'.
L'instruction a été arrêtée.
```

Tentative de suppression d'une personnet possédant
encore des factures

```
DELETE FROM T_PERSONNE5
WHERE PRS_NOM = 'Dupont' AND PRS_PRENOM = 'Marcel'
```

```
Conflit entre l'instruction DELETE et la contrainte
COLUMN REFERENCE 'FK__T_FACTURE__PRS_I__5165187F'.
Le conflit est survenu dans la base de données
'DB_HOTEL', table 'T_FACTURE1', column 'PRS_ID'.
```

L'instruction a été arrêtée.

1.3. La gestion de l'intégrité référentielle

Comme nous l'avions annoncé, la syntaxe de la pose de contraintes d'intégrité est plus complexe que ce qui vient d'être évoqué. Voici la syntaxe complète de cette structure :

CONSTRAINT nom_contrainte FOREIGN KEY (liste_colonne_table) REFERENCES table_référencée (liste_colonne_référencées) [MATCH { FULL | PARTIAL | SIMPLE }] [{ ON UPDATE { NO ACTION | CASCADE | RESTRICT | SET NULL | SET DEFAULT }] [{ ON DELETE { NO ACTION | CASCADE | RESTRICT | SET NULL | SET DEFAULT }] [{ INITIALLY DEFERRED | INITIALLY IMMEDIATE } [[NOT] DEFERRABLE] | [NOT] DEFERRABLE [{ INITIALLY DEFERRED | INITIALLY IMMEDIATE }]]	*Clause MATCH de gestion de la référence* *Clause ON UPDATE de mise à jour* *Clause ON DELETE de suppression* *Clause de déférabilité*

1.3.1. Mode de gestion de la la référence, clause MATCH

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

CREATE TABLE T_FOURNISSEUR (FRN_NOM CHAR(16) NOT NULL, FRN_PRENOM CHAR(16) NOT NULL, CONSTRAINT PK_FRN PRIMARY KEY (FRN_NOM, FRN_PRENOM))
INSERT INTO T_FOURNISSEUR VALUES ('DUBOIS', 'Alain') INSERT INTO T_FOURNISSEUR VALUES ('DURAND', 'Paula')

MATCH SIMPLE implique que :

- 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

```
CREATE TABLE T_COMMANDE1
(CMD_ID      INTEGER NOT NULL PRIMARY KEY,
 FRN_NOM     CHAR(16),
 FRN_PRENOM  CHAR(16),
 CONSTRAINT  FK_CMD_FRN_MATCH_SIMPLE
          FOREIGN KEY (FRN_NOM, FRN_PRENOM)
          REFERENCES T_FOURNISSEUR (FRN_NOM, FRN_PRENOM)
          MATCH SIMPLE)
```

INSERT INTO T_COMMANDE1 VALUES (1, 'DUBOIS', 'Alain') INSERT INTO T_COMMANDE1 VALUES (2, 'DUBOIS', NULL) INSERT INTO T_COMMANDE1 VALUES (3, 'DUHAMEL', NULL) INSERT INTO T_COMMANDE1 VALUES (4, NULL, 'Paula') INSERT INTO T_COMMANDE1 VALUES (5, NULL, NULL)	Insertion réussie
INSERT INTO T_COMMANDE1 VALUES (6, 'DUHAMEL', 'Marcel')	Conflit entre l'instruction INSERT et la contrainte TABLE FOREIGN KEY 'FK_CMD_FRN_MATCH_SIMPLE'. Le conflit est survenu dans la base de données 'DB_HOTEL', table 'T_FOURNISSEUR'. L'instruction a été arrêtée.

MATCH FULL implique que :

- la contrainte s'aplique toujours sauf si toutes les colonnes sont pourvues d'un marqueur NULL

Par conséquent, il ne peut y avoir une colonne renseigné et l'autre pas.

```
CREATE TABLE T_COMMANDE2
(CMD_ID      INTEGER NOT NULL PRIMARY KEY,
 FRN_NOM     CHAR(16),
 FRN_PRENOM  CHAR(16),
 CONSTRAINT  FK_CMD_FRN_MATCH_FULL
        FOREIGN KEY (FRN_NOM, FRN_PRENOM)
        REFERENCES T_FOURNISSEUR (FRN_NOM, FRN_PRENOM)
        MATCH FULL)
```

INSERT INTO T_COMMANDE1 VALUES (1, 'DUBOIS', 'Alain') INSERT INTO T_COMMANDE1 VALUES (2, NULL, NULL)	Insertion réussie
INSERT INTO T_COMMANDE1 VALUES (3, 'DUHAMEL', NULL) INSERT INTO T_COMMANDE1 VALUES (4, NULL, 'Paula') INSERT INTO T_COMMANDE1 VALUES (5, 'DUBOIS', NULL) INSERT INTO T_COMMANDE1 VALUES (6, 'DUHAMEL', 'Marcel')	Conflit entre l'instruction INSERT et la contrainte TABLE FOREIGN KEY 'FK_CMD_FRN_MATCH_FULL'. Le conflit est survenu dans la base de données 'DB_HOTEL', table 'T_FOURNISSEUR'. L'instruction a été arrêtée.

MATCH PARTIAL implique que :

- La contrainte s'applique pour toutes les colonnes renseignées

```
CREATE TABLE T_COMMANDE2
(CMD_ID      INTEGER NOT NULL PRIMARY KEY,
 FRN_NOM     CHAR(16),
 FRN_PRENOM  CHAR(16),
 CONSTRAINT  FK_CMD_FRN_MATCH_PARTIAL
        FOREIGN KEY (FRN_NOM, FRN_PRENOM)
        REFERENCES T_FOURNISSEUR (FRN_NOM, FRN_PRENOM)
        MATCH PARTIAL)
```

INSERT INTO T_COMMANDE1 VALUES (1, 'DUBOIS', 'Alain') INSERT INTO T_COMMANDE1 VALUES (2, NULL, NULL) INSERT INTO T_COMMANDE1 VALUES (4, NULL, 'Paula') INSERT INTO T_COMMANDE1 VALUES (5, 'DUBOIS', NULL)	Insertion réussie
INSERT INTO T_COMMANDE1 VALUES (3, 'DUHAMEL', NULL) INSERT INTO T_COMMANDE1 VALUES (6, 'DUHAMEL', 'Marcel')	Conflit entre l'instruction INSERT et la contrainte TABLE FOREIGN KEY 'FK_CMD_FRN_MATCH_PARTIAL'. Le conflit est survenu dans la base de données 'DB_HOTEL', table 'T_FOURNISSEUR'. L'instruction a été arrêtée.

NOTA : certains SGBD n'ont pas implémenté le mode de gestion de la référence. C'est le cas en particulier de MS SQL Server et d'InterBase.

1.3.2. Mode de gestion de l'intégrité clauses ON UPDATE / ON DELETE

Le mode de gestion de l'intégrité consiste à se poser la question de ce que la machine doit faire dans le cas ou l'on tente de briser une intégrité référentielle. Nous avons vu que par défaut il n'est pas possible de supprimer une personne ayant encore desdonnées dans la table des factures et qu'il n'est pas possible d'insérer une facture pour une personne nonréférencée. Se mode est dit en SQL : ON UPDATE NO ACTION, ON DELETE NO ACTION ce qui signifie qu'aucune action particulière n'est entreprise en cas de mise à jour ou suppression.
Nous allons maintenant voir quels sont les autres modes de gestion de l'intégrité référentielle

ATTENTION : ce mode n'a aucun effet sur le comportement de la contrainte qui s'exerce de toute façon en fonction de la clause MATCH

ON DELETE NO ACTION / ON UPDATE NO ACTION : aucun traitement particulier n'est entrepris en cas de mise à jour ou suppression d'informations référencées. Autrement dit, il y a blocage du traitement car le lien d'intégrité ne doit pas être brisé. Même effets que RESTRICT, mais post opératoire.

ON DELETE CASCADE / ON UPDATE CASCADE : en cas de suppression d'un élément, les éléments qui le référence sont aux aussi supprimés. En cas de modification de la valeur de la clef, les valeurs des clefs étrangères qui le référence sont elles aussi modifiées afin de maintenir l'intégrité. Par exemple en cas de suppression d'un client les factures et commandes sont elles aussi supprimées.

NOTA : ce mode est très tentant, mais son coût de traitement est très élevé et les performances peuvent très rapidement se dégrader fortement.

ON DELETE SET NULL / ON UPDATE SET NULL : en cas de suppression d'un élément, les éléments qui le référence voit leur clef étrangère posséder le marqueur NULL . De même en cas de modification de la valeur de la clef. Le lien d'intégrité est alors brisé.
L'intérêt d'une telle manoeuvre est de permettre la suppression des lignes devenues orphelines de manière différé, par exemple dans un batch de nuit.

ON DELETE SET DEFAULT / ON UPDATE SET DEFAULT : en cas de suppression comme en cas de mise à jour de la clef référencée, la référence passe à la valeur par défaut définie lors de la création de la table. Ce mode permet l'insertion d'un client générique, possédant un identifiant particulier (par exemple 0 ou -1) afin de ne jamais briser le lient d'intégrité référentielle. Bien entendu on veillera ensuite à rectifier la vrai valeur du lien au moment opportun si besoin est.

ON DELETE RESTRICT / ON UPDATE RESTRICT : mêmes effets que NO ACTION, mais pré opératoire.

NOTA : certains SGBD n'ont pas implémenté le mode de gestion de l'intégrité. C'est le cas en particulier de MS SQL Server. En revanche, il est courant de trouver dans les SGBD des options plus limitées que celles fournies par la norme.

1.5. Contraintes horizontales ou verticales ?

Comme nous l'avons vu, les contraintes peuvent être définie PRIMARY KEY, UNIQUE, CHECK et FOREIGN KEY peuvent être définies indifférement en contraintes de colonnes comme en contrainte de ligne. Ainsi une clef portant sur une seule colonne peut parfaitement être définie en tant que contrainte de table.
Ainsi les deux ordres suivants :

```
CREATE TABLE T_CLIENT
(CLI_ID      INTEGER NOT NULL PRIMARY
KEY,
 CLI_NOM     CHAR(32))
```

```
CREATE TABLE T_CLIENT
(CLI_ID      INTEGER NOT NULL,
 CLI_NOM     CHAR(32),
 CONSTRAINT  PK_CLI PRIMARY KEY
(CLI_ID))
```

Sont strictement équivalent, même si l'un est plus verbeux.
Mais il y a un net avantage à utiliser systématiquemen des contraintes horizontales.
Simplement parce que :

- elle sont plus lisible
- elle sont nommées
- elle peuvent facilement être supprimées et réinsérées

1.6. Alter et Drop

Les ordres ALTER et DROP sont les ordres de modification (ALTER pour altération) et suppression (DROP).

L'ordre ALTER peut porter sur un domaine, une assertion, une table, une vue, etc...

L'ordre **ALTER** sur une table permet de :

- supprimer une colonne
- supprimer une contrainte
- ajouter une colonne
- ajouter une contrainte
- ajouter une contrainte de ligne DEFAULT

Il ne permet pas de :

- changer le nom d'une colonne
- changer le type d'une colonne
- ajouter une contrainte de ligne NULL / NOT NULL

Syntaxe de l'ordre ALTER sur table :

```
ALTER TABLE nom_table
{  ADD   definition_colonne
 | ALTER nom_colonne { SET DEFAULT valeur_défaut | DROP DEFAULT }
 | DROP  nom_colonne [ CASCADE | RESTRICT ]
 | ADD   définition_contrainte_ligne
 | DROP  CONSTRAINT nom_contrainte [ CASCADE | RESTRICT ] }
```

L'option CASCADE / RESTRICT permet de gérer l'intégrité de référence de la colonne ou la contrainte.
Si RESTRICT, alors tout objet dépendant de cette colonne ou de cette contrainte provoquera l'annulation de l'opération de suppression.
Si CASCADE, alors tous les objets dépendant de cette colonne ou contrainte seront supprimés.

```
ALTER TABLE T_CLIENT
ADD CLI_PRENOM VARCHAR(25)
```

```
ALTER TABLE T_CLIENT
ADD CLI_DATE_NAISSANCE DATE,
```

```
ALTER TABLE T_CLIENT
ADD CONSTRAINT CHK_DATE_NAISSANCE CHECK (CLI_DATE_NAISSANCE BETWEEN '1880-01-01' AND '2020-01-01')
```

ATTENTION : ne pas tenter de rajouter une colonne avec l'attribut NOT NULL lorsque la table contient déjà des lignes. Pour cette opération, veuillez procéder en plusieurs étapes dans un script transactionné.

DROP est l'ordre de suppression. Sa syntaxe est onne peut plus simple :

```
DROP {TABLE | DOMAIN | ASSERTION | VIEW } nom_objet
```

1.6.1. Changer le nom ou le type d'une colonne

Ce cas n'est pas géré par un ordre simple de SQL. En effet cette modification est trop risquée pour être standardisée. Quid des données contenue dans la colonne au passage de CHAR en FLOAT ? Quid des références de cette colonne dans des vues, des contraintes, des triggers si l'on en change le nom ?

Mais il est possible de contourner le problème en réalisant un script transactionné. Certains SGBD proposent un ordre ALTER étendu ou une procédure stockée (par exemple sp_rename de MS SQL Server).

Avant de lancer un tel script il convient de s'assurer que le colonne ne fait l'objet d'aucune référence interne (contraintes de table par exemple) ou externe (vue, triggers...). Si c'est le cas, il faut impérativement modifier, désactiver ou supprimer ces éléments avant la modification de la colonne.

Voici les différentes étapes du script à mettre en oeuvre :

- créer une colonne temporaire de même nom et même type (ALTER TABLE ADD...
- alimenter la colonne temporaire avec les valeurs de l'actuelle (UPDATE ...
- supprimer l'actuelle colonne (ALTER TABLE DROP...
- créer une nouvelle colonne avec le nouveau nom et/ou le nouveau type (ALTER TABLE ADD...
- alimenter la nouvelle colonne avec les données de la colonne temporaire (UPDATE ...
- supprimer la colonne temporaire (ALTER TABLE DROP ...

modification d'une colonne CHAR(6) contenant une date courte format FR en date SQL :

```
-- condition de départ

CREATE TABLE T_IMPORT
(IMP_ID   INTEGER,
 IMP_NOM  VARCHAR(16),
 IMP_DATE CHAR(6))

INSERT INTO T_IMPORT VALUES (254, 'Dupont', '251159')
INSERT INTO T_IMPORT VALUES (321, 'Durand', '130278')
INSERT INTO T_IMPORT VALUES (187, 'Dubois', '110401')

-- le script de modification

ALTER TABLE T_IMPORT
ADD TMP_IMP_DATE CHAR(6)

UPDATE T_IMPORT
SET TMP_IMP_DATE = IMP_DATE

ALTER TABLE
DROP IMP_DATE

ALTER TABLE
ADD IMP_DATE DATE

UPDATE T_IMPORT
SET IMP_DATE = CAST(CASE SUBSTRING(TMP_IMP_DATE, 5, 2)
                      WHEN < '03' THEN  '20'
                      ELSE '19'
                    END || SUBSTRING(TMP_IMP_DATE, 5, 2)
                    || '-' || SUBSTRING(TMP_IMP_DATE, 3, 2)
                    || '-' || SUBSTRING(TMP_IMP_DATE, 1, 2) AS DATE)

ALTER TABLE
DROP TMP_IMP_DATE

COMMIT

-- on aura noté que le pivot de date pour changement de siècle aura été géré dans le dernier update...
```

1.6.2. Ajouter ou supprimer la contrainte NULL ou NOT NULL

Les étapes du script différent très peu. Voici un exemple :

modification d'une colonne IMP_NOM en plaçant la contrainte NOT NULL :

```
-- condition de départ

CREATE TABLE T_IMPORT
(IMP_ID   INTEGER,
 IMP_NOM  VARCHAR(16),
 IMP_DATE CHAR(6))

INSERT INTO T_IMPORT VALUES (254, 'Dupont', '251159')
INSERT INTO T_IMPORT VALUES (321,  NULL,    '130278')
INSERT INTO T_IMPORT VALUES (187, 'Dubois', '110401')

-- le script de modification

ALTER TABLE T_IMPORT
ADD TMP_IMP_NOM VARCHAR(16)

UPDATE T_IMPORT
SET TMP_IMP_NOM = COALESCE(IMP_NOM, '')

ALTER TABLE
DROP IMP_NOM

ALTER TABLE
ADD IMP_NOM VARCHAR(16) NOT NULL

UPDATE T_IMPORT
SET IMP_NOM = TMP_IMP_NOM

ALTER TABLE
DROP TMP_IMP_NOM

COMMIT

-- on aura noté qu'afin d'éviter un rejet massif de notre script,
-- on place un nom constitué d'une chaîne vide grace à l'opérateur
-- coalesce, dans le premier update
```

2. Les vues

Les vues de la norme SQL 2 ne sont autre que des requêtes instanciées.
Elles sont nécessaires pour gérer finement les privilèges. Elles sont utiles pour masquer la complexité de certains modèles relationnel.

Voici la syntaxe SQL 2 pour définir une vue :

```
CREATE VIEW nom_vue [ ( nom_col1, [, nom_col2 ... ] ) ]
AS
  requête_select
[WITH CHECK OPTIONS]
```

Exemple 94 - vue simplifiant un modèle

```
-- la table suivante :
CREATE TABLE T_TARIF
(TRF_ID     INTEGER PRIMARY KEY,
 TRF_DATE   DATE,
 PRD_ID     INTEGER,
 TRF_VALEUR FLOAT)

-- permet de stocker l'évolution d'un tarif, sachant que celui-ci n'est applicable
-- pour un produit donné (PRD_ID) qu'à partir de la date TRF_DATE
INSERT INTO T_TARIF VALUES (1, '1996-01-01', 53, 123.45)
INSERT INTO T_TARIF VALUES (2, '1998-09-15', 53, 128.52)
INSERT INTO T_TARIF VALUES (3, '1999-12-31', 53, 147.28)
INSERT INTO T_TARIF VALUES (4, '1997-01-01', 89, 254.89)
INSERT INTO T_TARIF VALUES (5, '1999-12-31', 89, 259.99)
INSERT INTO T_TARIF VALUES (6, '1996-01-01', 97, 589.52)

-- pour des raisons de commodité d'interrogation des données, on voudrait
-- faire apparaître l'intervalle de validité du tarif plutôt que la date d'application

-- la vue suivante répond à cette attente
CREATE VIEW V_TARIF
AS
SELECT TRF_ID,  PRD_ID, TRF_DATE AS TRF_DATE_DEBUT,
      (SELECT COALESCE(MIN(TRF_DATE) - INTERVAL 1 DAY, CURRENT_DATE)
       FROM   T_TARIF T2
       WHERE  T2.PRD_ID = T1.PRD_ID
         AND T2.TRF_DATE > T1.TRF_DATE) AS TRF_DATE_FIN,
      TRF_VALEUR
FROM   T_TARIF T1
```

```
SELECT *
FROM V_TARIF
```

TRF_ID	PRD_ID	TRF_DATE_DEBUT	TRF_DATE_FIN	TRF_VALEUR
1	53	1996-01-01	1998-09-14	123.45
2	53	1998-09-15	1999-12-30	128.52
3	53	1999-12-31	2002-09-03	147.28
4	89	1997-01-01	1999-12-30	254.89
5	89	1999-12-31	2002-09-03	259.99
6	97	1996-01-01	2002-09-03	589.52

Une vue peut être utilisée comme une table dans toute requête de type SELECT. Mais à la différence des tables, une vue peut être mise à jour (INSERT, UPDATE, DELETE) que si elle obéit à un certain nombre de conditions :

- ne porter que sur une table (pas de jointure)
- ne pas contenir de dédoublonnage (pas de mot clef DISTINCT) si la table n'a pas de clef
- contenir la clef de la table si la table en a une
- ne pas transformer les données (pas de concaténation, addition de colonne, calcul d'agrégat...)
- ne pas contenir de clause GROUP BY ou HAVING
- ne pas contenir de sous requête
- répondre au filtre WHERE si la clause WITH CHECK OPTIONS est spécifié lors de la création de la vue

Bien évidemment une vue peut porter sur une autre vue et pour que la nouvelle vue construite à partir d'une autre vue puisse être modifié, il faut que les deux vues répondent aussi à ces critères.

En fait c'est plus simple qu'il n'y parait : il suffit que le SGBD puisse retrouver trace de la ligne dans la table et de chaque valeur de chaque colonne.

vue restreignant l'accès aux colonnes

```
-- soit la table suivante :
CREATE TABLE T_EMPLOYE
(EMP_ID        INTEGER PRIMARY KEY,
 EMP_MATRICULE CHAR(8),
 EMP_TITRE     VARCHAR(4),
 EMP_NOM       VARCHAR(32),
 EMP_PRENOM    VARCHAR(32),
 EMP_DATE_NAIS DATE,
 EMP_SALAIRE   FLOAT,
 EMP_STATUT    CHAR(8),
 EMP_MAIL      VARCHAR(128),
 EMP_TEL       CHAR(16))
-- permet de stocker les employés de l'entreprise

-- pour le syndicat, on pourra définir la vue suivante :
CREATE VIEW V_EMP_SYNDICAT
AS
  SELECT EMP_MATRICULE, EMP_TITRE, EMP_NOM, EMP_PRENOM, EMP_DATE_NAIS, EMP_MAIL, EMP_TEL
  FROM   T_EMPLOYE
-- elle ne peut être mise à jour car la clef ne s'y trouve pas

-- pour le carnet d'adresse on pourra définir la vue suivante
CREATE VIEW V_EMP_SYNDICAT
AS
  SELECT EMP_ID, EMP_TITRE || ' ' || EMP_PRENOM || ' ' || EMP_NOM AS EMP_NOM_COMPLET, EMP_MAIL,
EMP_TEL
  FROM   T_EMPLOYE
-- elle ne peut être mise à jour à cause des transformation de données (concaténation au niveau du nom)

-- pour le service comptable, on pourra définir la vue suivante :
CREATE VIEW V_EMP_SYNDICAT
AS
  SELECT EMP_ID, EMP_PRENOM, EMP_NOM, EMP_SALAIRE
  FROM   T_EMPLOYE
  WHERE  STATUT = 'ETAM'
WITH CHECK OPTIONS
-- elle pourra être mis à jour uniquement pour les salariés de type 'ETAM'
```

La clause WITH CHECK OPTION implique que si la vue peut être mise à jour, alors les valeurs modifiées insérées ou supprimées doivent répondre à la validation de la clause WHERE comme s'il s'agissait d'une contrainte.

Par exemple dans le cadre de la vue pour le service comptable, il n'est pas possible de faire :

```
UPDATE T_EMPLOYE
SET EMP_SALAIRE = EMP_SALAIRE + 100
WHERE STATUT = 'CADRE'
```

3. Les index

Contrairement à une idée reçue, les index ne font nullement partie du SQL. Ce sont en revanche des éléments indispensables à une exploitation performante de base de données. En effet un index permet de spécifier au SGBD qu'il convient de créer une structure de données adéquate afin de stocker les données dans un ordre précis. Par conséquent les recherches et en particuliers les comparaisons, notamment pour les jointures, sont notablement accélérées. Dans le principe le gain de temps espéré est quadratique. Par exemple si une recherche sur une colonne dépourvue d'index met 144 secondes, avec un index cette même recherche sera supposée mettre 12 seconde (racine carré de 144) !

Différents types d'index sont généralement proposés. Voici quelques exemples de techniques d'indexation :

- index en cluster : l'ordre des données répond à un ordre physique d'insertion, convient particulièrement pour les clefs numériques auto incrémentées (dans ce cas une table ne peut recevoir qu'un seul index de ce type)
- index en arbre équilibré : convient pour la plupart des types de données
- index en clef de hachage : convient pour des colonnes dont la dispersion est très importante. Un algorithme de hachage est mis en place (il s'agit en général d'une transformation injective)
- index bitmap : convient pour des colonnes à faible dispersion (ideal pour des colonnes booléennes)

En règle général les fabriquants de SGBD proposent un mécanisme de création d'index dont la syntaxe est proche des ordres basiques du SQL. C'est en général l'ordre CREATE INDEX.

Voici la syntaxe d'un tel ordre pour MS SQL Server :

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX nom_index
ON nom_table (col1 [, col2 ...] )
[WITH
  [PAD_INDEX]
  [[,] FILLFACTOR = facteur_de_remplissage]
  [[,] IGNORE_DUP_KEY]
  [[,] DROP_EXISTING]
  [[,] STATISTICS_NORECOMPUTE]
]
[ON groupe_de_fichiers]
```

La plupart du temps lorsque vous créez une contrainte de clef primaire, étrangère ou une contrainte d'unicité, le SGBD implante automatiquement un index pour assurer la mécanisme de contrainte avec des performances correctes. En effet une contrainte d'unicité est facilité si un tri sur les données de la colonne peut être activé très rapidement.

CONSEIL : pour une table donnée, il convient d'indexer dans l'ordre :

- les colonnes composant la clef
- les colonnes composant les clefs étrangères
- les colonnes composant les contraintes d'unicité
- les colonnes dotées de contraintes de validité
- les colonnes fréquemment mises en relation, indépendemenent des jointures naturelles
- les colonnes les plus sollicitées par les recherches

Dans la mesure du possible on placera des index à ordre descendant pour les colonnes de type DATE, TIME et DATETIME.