2019

SQL

PARTIE 3

JOACHIM ZADI

TABLE DES MATIERES

LE DDL

LES TYPES DE DONNEES

Nous abordons dans cette section les types de données les plus utilisés pour la description des colonnes d'une table. Il existe trois grandes familles de données : numérique, caractère (ou alphanumérique) et temporelle (dates et heures).

Chaque SGBDR a décliné des types spécifiques pour un besoin précis comme les types géographiques, ou pour des problématiques de stockage. Le type choisi dépendra donc de la précision recherchée, tout en tenant compte de la taille nécessaire pour stocker la donnée. Un bon compromis permet d'accéder rapidement à la valeur.

1. NUMERIQUES

Les types numériques permettent de définir si l'on souhaite un entier, un décimal ou un nombre à virgule flottante.

Selon le type de base de données, il existe des propriétés qui complètent le type de données comme :

- IDENTITY(s, i) dans SQL Server, où s représente la valeur de la première ligne insérée et i le pas entre les lignes suivantes, IDENTITY dans Oracle, AUTO_INCREMENT dans MySQL pour incrémenter automatiquement un entier.
- UNSIGNED dans MySQL pour préciser si l'on accepte ou non les nombres négatifs.
- ZEROFILL dans MySQL permet de remplacer les espaces par des 0. L'attribut UNSIGNED est automatiquement ajouté avec ZEROFILL.

Exemple Oracle

`PRIX NUMBER (8,2)` ou `PRIX DECIMAL (8,2)` sont identiques.

`IDCHAMBRE NUMBER (5)` ou `IDCHAMBRE NUMBER (5,0)` sont identiques.

Dans le premier exemple, on décrit un type décimal de 8 chiffres de longueur dont deux décimales, comme 123456,12.

Dans le deuxième exemple, on ne précise pas de longueur après la virgule donc il s'agit d'un entier.

MySQL, de son côté, a également fusionné les deux types numériques dans le type DECIMAL.

Exemple MySQL

```
PRIX    DECIMAL (8,2)
IDCHAMBRE DECIMAL (5)
NUMCHAMBRE SMALLINT
```

2. CARACTERES

Le type alphanumérique classique se note CHAR pour les données de taille fixe et VARCHAR pour les chaînes de caractères de longueur variable.

La majorité du temps, il est conseillé d'utiliser VARCHAR. En effet, une donnée déclarée sur 50 par exemple et qui ne contient que deux caractères sera effectivement stockée sur deux caractères avec VARCHAR alors qu'elle sera stockée sur 50 caractères avec CHAR. CHAR complète avec des espaces jusqu'à la longueur maximum. NCHAR ou NVARCHAR sont utilisés pour stocker des chaînes de caractères comprenant des caractères spéciaux comme le chinois, l'hébreu, le russe… Le nombre d'octets est doublé.

Dans le cas de VARCHAR, la longueur effective de la chaîne est stockée sur un octet, permettant ainsi au SGBDR de restituer correctement la donnée.

Il est possible de trouver le type TEXT, mais ce type est amené à disparaître. Ce n'est pas un standard SQL.

Dernière remarque, Oracle préconise d'utiliser le type VARCHAR2 à la place du VARCHAR. A priori, ces deux types sont identiques, mais Oracle pourrait utiliser le VARCHAR différemment dans ses versions futures.

Selon le type de base de données, il existe des propriétés qui complètent le type de données.

Par exemple, dans MySQL, sur le type CHAR, il est possible d'ajouter des options :

- BINARY est utilisé pour tenir compte de la casse.
- NATIONAL spécifie que le jeu de caractères par défaut de MySQL doit être utilisé.
- ASCII spécifie le jeu de caractères latin1 à utiliser.
- UNICODE spécifie le jeu de caractères à utiliser.

Exemple

```
TYPE        CHAR(2),
COULEUR     VARCHAR(25));
```

3. DATES ET HEURES

Les types de format temporels sont principalement DATE, TIME et TIMESTAMP.

Les formats par défaut des dates ou des heures varient d'une base de données à l'autre. Pour connaître le format par défaut il faut utiliser :

- Pour SQL Server :

```
SELECT GETDATE();
```

- Pour PostgreSQL :

```
SELECT current_date;
```

- Pour Oracle :

```
SELECT SYSDATE FROM DUAL;
```

- Pour MySQL :

```
SELECT NOW();
```

qui donnent la date du jour au format utilisé par la base.

Il est préférable de ne pas s'appuyer sur un format défini pour une base de données. En effet, ce format peut évoluer ou la séquence de code s'appliquer sur une autre base qui n'aura pas le même format de date par défaut.

Nous verrons dans la suite de cet ouvrage que la manipulation des dates et des heures est très variable d'un SGBDR à l'autre, les fonctions de conversion notamment sont différentes (cf. chapitre Les fonctions - Les différents formats d'affichage des dates).

Lors de la création de vos tables, il n'est pas utile de connaître le format de stockage, il faut déclarer la colonne DATE, TIME ou DATETIME selon la précision attendue.

- DATE : format date, norme ISO AAAA-MM-JJ
- TIME : format heure, norme ISO HH:MM:ss.nnnnnnn
- TIMESTAMP : format date et heure, norme ISO AAAA-MM-JJ HH:MM:ss .nnn

Attention : Oracle ne connaît pas le type TIME, il faut utiliser le type TIMESTAMP, ou stocker l'heure dans un format CHAR(10) puis utiliser les formats de conversion ou stocker les heures en les convertissant en secondes dans un champ de type NUMBER.

Type	Précision	Stockage (octets)	BDD
DATETIME	Milliseconde (du 01/01/1753 au 31/12/9999)	8 octects	SQL Server
DATETIME	Nanoseconde (du 01/01/1000 au 31/12/9999)	8 octets	MySQL
SMALLDATETIME	Minute (du 01/01/1900 au 06/06/2079)	4 octects	SQL Server
DATETIME2	Nanoseconde (du 01/01/0001 au 31/12/9999)	6 à 8 octets	SQL Server
DATE	Jour (du 01/01/0001 au 31/12/9999)	3 octects	SQL Server, Oracle, MySQL
TIME	Nanoseconde	3 à 5 octects	SQL Server, MySQL
TIMESTAMP	Date et heure courantes	8 octets	MySQL
DATETIMEOFFSET	Nanoseconde (du 01/01/0001 au 31/12/9999 - affiche le fuseau horaire)	8 à 10 octets	SQL Server

Exemple

```
DATE_ACHAT    DATE,
DATE_MAJ      TIMESTAMP
```

4. LES TYPES BINAIRES

Oracle utilise les types suivants :

Types numériques	Valeur min	Valeur max	Remarque
RAW	1	32 767 octets	Entrée en hexadécimal
LONG RAW	1	2 Go	Entrée en hexadécimal
BLOB	1	4 Go	Gros objet binaire
BFILE	1	4 Go	Pointeur vers un fichier binaire externe

Cette liste n'est pas exhaustive, consultez la documentation de la base de données pour détailler tous les types possibles.

Données de type BLOB :

Type	Précision	Stockage (octets)	BDD
TINYBLOB	1 à 255 caractères (sensible à la casse)	Longueur chaîne + 1 octet	MySQL
BLOB	1 à 65 535 caractères (sensible à la casse)	Longueur chaîne + 1 octet	MySQL

MEDIUMBLOB	1 à 16 777 215 caractères (sensible à la casse)	Longueur chaîne + 3 octets	MySQL
LONGBLOB	1 à 4 294 967 295 caractères (sensible à la casse)	Longueur chaîne + 4 octets	MySQL

5. AUTRES TYPES DE DONNEES

Selon les bases de données, il existe d'autres types de données comme :

- géographiques,
- spaciales,
- spécifiques comme URL.

LA CREATION DE TABLES

Dans cette section, nous allons voir comment créer une table, ajouter ou supprimer des colonnes, mettre des commentaires sur les colonnes et les tables, mais également la méthode pour copier une table dans une autre puis comment attribuer un synonyme à un nom de table.

1. L'ORDRE CREATE

CREATE est l'ordre de base en langage SQL pour créer un élément. Celui-ci sert à créer une table, un index, une vue ou un synonyme. En fonction du besoin, la syntaxe est différente.

Dans cette section, nous allons traiter de la création de tables. Une table se définit principalement par les colonnes qui la composent et les règles qui s'appliquent à ces colonnes.

Nous n'aborderons pas les aspects de stockage physique de la table. En effet, chaque SGBDR a sa propre syntaxe dans ce domaine. Dans la majorité des cas, ce sont les DBA (*Database Administrator*, administrateur de base de données) qui spécifient les normes de stockage et les options à appliquer sur les tables. Le stockage des tables est un élément déterminant en termes de performance et de sécurité de la base de données. Il est donc fortement conseillé de se rapprocher d'un administrateur avant de se lancer dans la création d'une table.

Pour supprimer une table, on utilise l'ordre DROP TABLE.

Avant de créer une table, il est préférable de se poser quelques questions et de respecter certaines règles.

Essayez de donner des noms parlants aux tables et aux colonnes afin de retrouver ensuite facilement une information. Les DBA définissent la plupart du temps les règles de nommage des tables et des colonnes. Ils fournissent quelquefois des scripts standards pour créer une table. Il est donc impératif de consulter les normes en vigueur dans l'entreprise avant toute création.

Les règles minimums à respecter dans le nommage d'une table ne sont pas très nombreuses et se résument ainsi : un nom de table commence par une lettre, il est unique et il ne doit pas dépasser 256 caractères.

Quand une colonne a la même signification dans plusieurs tables, il est conseillé de garder le même nom. En effet, si la colonne se retrouve dans plusieurs tables et que le modèle relationnel a été respecté, cela signifie que la colonne est la clé d'une table. Donc, en gardant le même nom, on simplifie ensuite les jointures entre ces deux tables, les colonnes de clé seront facilement identifiables. Cela permet à certains outils, comme Power BI, de proposer les relations lorsqu'elles n'existent pas.

Nous verrons, dans le chapitre La manipulation des données (LMD) - section L'utilisation des alias, comment distinguer deux colonnes de même nom dans une même sélection.

Syntaxe

```
CREATE TABLE nom_de_table(
       Nom_colonne Type_colonne
       Nom_colonne Type_colonne,
       Nom_colonne Type_colonne
       ...
);
```

Voici trois structures de tables.

Table TARIFS

Tarifs
IdTarif
Hotel
TypeChambre
DateDebut
DateFin
Prix

Table HOTELS

Hotels
idHotel
Libelle
Etoile

Table TypesChambre

TypesChambre
idTypeChambre
NombreLit
TypeLit
Description

Les scripts nécessaires pour créer les tables ci-dessus sont :

Syntaxe SQL Server et PostgreSQL

```
CREATE TABLE TARIFS (
       idTarif       INTEGER,
       hotel         INTEGER,
       typeChambre   INTEGER,
       DateDebut     DATE,
       DateFin       DATE,
       Prix          MONEY
);
```

Syntaxe MySQL (remplacement du type MONEY par DECIMAL)

```
CREATE TABLE TARIFS (
       idTarif       INTEGER
       hotel         INTEGER
       typeChambre   INTEGER,
       DateDebut     DATE,
       DateFin       DATE,
       Prix          DECIMAL(7,3)
);
```

Syntaxe Oracle

```
CREATE TABLE TARIFS (
       idTarif       NUMBER(38,0),
       hotel         NUMBER(38,0),
       typeChambre   NUMBER(38,0),
       DateDebut     DATE,
       DateFin       DATE,
       Prix          DECIMAL(7,3)
);
```

Syntaxe SQL Server, PostgreSQL et MySQL

```
CREATE TABLE Hotels(
       idHotel     INTEGER,
       Libelle     VARCHAR(50,
       Etoile      VARCHAR(5)
);
```

```
CREATE TABLE TypesChambre (
       idTypeChambre   INTEGER
       NombreLit       INTEGER,
       TypeLit         VARCHAR(20),
       Description     VARCHAR(255)
);
```

Il s'agit d'exemples simples. Aucune colonne n'a de restriction ou de valeur par défaut. Aucune clé n'a été définie.

Nous pouvons constater que les colonnes idHotel et idTypeChambre sont les clés des tables Hotels et TypesChambre et que l'on retrouve ces colonnes sous le nom hotel et typeChambre dans la table Tarifs. Ainsi, si nous avons des jointures à réaliser entre ces tables, nous utiliserons ces colonnes.

2. TABLES TEMPORAIRES

Il est possible de créer des tables temporaires. Comme leur nom l'indique, ces tables ont une durée de vie limitée qui correspond à la session dans laquelle elles sont créées. Elles sont supprimées automatiquement à la fermeture de la session. Elles sont utilisées dans les fonctions ou procédures stockées, comme une table. La syntaxe est quasi identique à celle de la création de tables. Les tables temporaires globales peuvent être utilisées par plusieurs sessions.

Syntaxe SQL Server

Il faut juste ajouter un # devant le libellé de la table locale et deux ## pour les globales.

```
CREATE TABLE #TARIFS2019 (idTarif       INTEGER,
                     hotel          INTEGER,
                     typeChambre    INTEGER,
                     DateDebut      DATE,
                     DateFin        DATE,
                     Prix           MONEY);
```

Syntaxe MySQL

```
CREATE TEMPORARY TABLE TMP_TARIFS2019 (idTarif       INTEGER,
                     hotel          INTEGER,
                     typeChambre    INTEGER,
                     DateDebut      DATE,
                     DateFin        DATE,
                     Prix           DECIMAL(7,3));
```

Syntaxe Oracle

```
CREATE LOCAL TEMPORARY TABLE TMP_TARIFS2019 (idTarif
NUMBER(38,0),
                      hotel          NUMBER(38,0),
                      typeChambre    NUMBER(38,0),
                      DateDebut      DATE,
                      DateFin        DATE,
                      Prix           DECIMAL(7,3))
ON COMMIT PRESERVE ROWS;
```

Pour créer une table temporaire globale, il suffit de remplacer LOCAL par GLOBAL dans la syntaxe ci-dessus. Pour que les lignes soient supprimées à chaque validation, remplacer l'ordre PRESERVE par DELETE.

Syntaxe PostgreSQL

```
CREATE TEMPORARY TABLE TMP_TARIFS2019 (idTarif       INTEGER,
                      hotel          INTEGER,
                      typeChambre    INTEGER,
                      DateDebut      DATE,
                      DateFin        DATE,
                      Prix           MONEY);
```

3. LES COMMENTAIRES (COMMENT)

La clause COMMENT permet d'ajouter des commentaires sur les colonnes et les tables. Cette option est très utile pour les développeurs futurs qui pourront ainsi connaître la fonction d'une table et le contenu de chaque colonne en affichant sa description.

Cela ne dispense pas d'avoir une documentation papier décrivant chaque table dans la base de données.

Néanmoins, en prenant l'habitude de documenter toutes les tables et les colonnes directement dans la base au moment de la création de celles-ci, automatiquement la maintenance et l'évolution de la base s'en trouvent simplifiées.

Une base de données a une durée de vie conséquente et va donc être utilisée par une population importante et hétéroclite.

Les commentaires seront d'autant plus utiles que la base est ancienne et que la documentation papier n'est plus toujours disponible, notamment en cas de migration de base de données ou de migration d'un système à un autre.

La syntaxe de cette clause diffère d'un SGBDR à un autre.

Dans Oracle et PostgreSQL, la syntaxe est la suivante :

```
COMMENT ON <'COLUMN' ou 'TABLE'> <Nom colonne ou nom de table> IS
'libellé libre';
```

- Ajouter un commentaire sur une table :

```
COMMENT ON TABLE Tarifs IS 'Tarif par hotel et type de chambre';
```

- Ajouter un commentaire sur une colonne :

```
COMMENT ON COLUMN Hotels.idHotel IS 'Numéro 'Numéro de l''hotel';
```

Les commentaires sont stockés dans deux tables Oracle :

- `user_tab_comments` : les commentaires sur les tables.
- `user_col_comments` : les commentaires sur les colonnes.

Dans MySQL, les commentaires seront mis au moment de la création de la table en ajoutant l'ordre COMMENT ainsi :

```
CREATE TABLE  Hotels(
    idHotel int NOT NULL PRIMARY KEY COMMENT 'Numéro de l''hotel',
    Libelle varchar(50) NULL COMMENT 'Dénomination de l''hotel' ,
    Etoile varchar(5) NULL COMMENT 'Nombre d''étoiles obtenues par
l''hotel'
);
```

Pour afficher les colonnes et les commentaires, il faut utiliser cette commande :

```
SHOW FULL COLUMNS FROM Hotels;
```

Dans SQL Server, il n'existe pas de propriété spécifique pour ajouter un commentaire. Il est possible de contourner ce manque en spécifiant une propriété étendue avec la syntaxe suivante :

```
EXEC sys.sp_addextendedproperty @name=N'Description', @value=N'Liste des
chambres', @level0type=N'SCHEMA',@level0name=N'dbo',
@level1type=N'TABLE',@level1name=N'Chambres'
```

Pour les colonnes, la propriété étendue 'Description' existe et peut être incrémentée en utilisant cette commande :

```
EXEC sys.sp_addextendedproperty @name=N'MS_Description',
@value=N'N° identifiant de chambre indépendant de l''hôtel',
@level0type=N'SCHEMA',@level0name=N'dbo',
@level1type=N'TABLE',@level1name=N'Chambres',
@level2type=N'COLUMN',@level2name=N'idChambre' ;
```

Dans PostgreSQL, il existe une commande 'COMMENT' pour ajouter un commentaire sur une table ou tout autre objet de la base comme la colonne.

```
COMMENT ON TABLE matable IS 'Ceci est ma table.';
COMMENT ON COLUMN matable.macolonne IS 'Ceci est ma colonne';
```

4. CREER UNE TABLE A PARTIR D'UNE SELECTION DE DONNEES

Une autre méthode pour créer une table consiste à dupliquer ou à s'inspirer de tables existantes.

Cette méthode est souvent utilisée par les développeurs pour créer des tables de test et des jeux d'essais. Il n'est pas souhaitable d'utiliser cette méthode pour la définition initiale de la base de données.

La création de table associée à un ordre de sélection de colonne d'une autre table ou de plusieurs tables permet de générer rapidement la structure d'une table spécifique et, dans le même ordre, de remplir cette table.

Avec cette syntaxe, les possibilités sont très étendues, étant donné que l'on peut utiliser toutes les possibilités de l'ordre SELECT.

Il est possible de récupérer certaines colonnes, de filtrer les données, mais également de copier uniquement la structure sans les données en ajoutant une restriction qui n'est jamais vérifiée

Syntaxe

```
CREATE TABLE <Nouvelle table> AS SELECT <nom colonne1>,<nom
colonne2>... ou <*> FROM <Table à copier> WHERE ... ... ;
```

Exemple : copie de structure et de données

Avec Oracle et PostgreSQL, il faut utiliser la syntaxe suivante :

```
CREATE TABLE SAV_Chambres AS SELECT * FROM Chambres;
```

Avec cet ordre, on copie toute la structure de la table Chambres ainsi que tout son contenu dans la table SAV_Chambres.

Attention : avec les très grosses tables, il existe un risque de rencontrer des problèmes d'espace disque ou de temps de réponse importants. Il est préférable de tester la requête SELECT avant de lancer la création proprement dite.

Par exemple, il est possible de compter le nombre de lignes de la table Chambres par un SELECT COUNT(*) FROM Chambres qui permet de connaître le volume à transférer.

Avec MySQL, la syntaxe est la suivante (le AS disparaît) :

```
CREATE TABLE SAV_Chambres SELECT * FROM Chambres;
```

Avec SQL Server, la syntaxe est la suivante :

```
SELECT * INTO Chambres_SAV
FROM Chambres;
```

Pour récupérer uniquement la structure de la table d'origine, il faut ajouter une clause qui n'est jamais vérifiée. Nous verrons, dans le chapitre La manipulation des données (LMD) - section La sélection de données, la description détaillée de la clause SELECT.

Exemple : copie de structure

```
CREATE TABLE SAV_Chambres AS SELECT * FROM Chambres WHERE 1=0;
```

La clause WHERE n'étant jamais vraie, le SELECT ramène zéro ligne.

Il est également possible de ne sélectionner que quelques colonnes de la table d'origine.

Exemple : copie d'une partie de la structure

```
CREATE TABLE SAV_Chambres AS SELECT idChambre, Hotel FROM
Chambres WHERE 1=0;
```

La table SAV_Chambres contient seulement deux colonnes et sera vide.

SAV_Chambres
idChambre Hotel

Avec MySQL, il faut copier la structure d'une table sur une autre avec l'utilisation d'un LIKE.

```
CREATE TABLE SAV_Chambres LIKE Chambres;
```

Il est également possible de ne sélectionner que quelques lignes de la table d'origine.

Exemple : copie d'une partie de la structure et sélection de lignes (syntaxe Oracle)

```
CREATE TABLE SAV_Chambres AS SELECT idChambre, Hotel,
TypeChambre FROM Chambres WHERE TypeChambre = 3;
```

Ainsi, si la table Chambres contient les données suivantes :

idChambre	Hotel	TypeChambre	NumChambre	Commentaire
23	4	2	2	NULL
16	3	2	2	NULL
9	2	2	2	NULL
2	1	2	2	NULL
3	1	3	3	NULL
10	2	3	3	NULL
17	3	3	3	NULL
24	4	3	3	NULL
25	4	4	4	NULL

La table SAV_Chambres contient les éléments suivants :

idChambre	Hotel	TypeChambre
3	1	3
10	2	3
17	3	3
24	4	3

Seules les quatre lignes concernées de la table Chambres ont été sélectionnées. Elles correspondent au critère demandé TypeChambre = 3 et la table SAV_Chambres contient seulement trois colonnes.

Avec ce type de sélection, il est possible de créer très facilement un jeu d'essais pour son propre compte ou à destination d'un utilisateur. Nous verrons, dans la section Les vues de ce chapitre, le mécanisme des vues, qui est assez semblable à cette méthode.

5. UTILISATION DES SYNONYMES

Pour donner un nom différent à une table, il faut utiliser la commande CREATE SYNONYM.

On peut utiliser également un synonyme pour simplifier un nom de table qui est normalisé, ou pour pointer sur une archive dont le nom n'a pas à être connu par les utilisateurs.

Cette fonction n'est pas implémentée avec MySQL ni dans PostgreSQL.

Syntaxe (Oracle, SQL Server)

```
CREATE SYNONYM <Nom synonyme> FOR <Nom table>;
```

Exemple

```
CREATE SYNONYM PRIX FOR Tarifs;
```

Ainsi, lorsqu'un développeur utilise la table, il peut indiquer PRIX comme ceci :

```
SELECT * FROM PRIX;
```

6. LES SEQUENCES

Dans une table, il y a souvent une colonne qui fait office d'identifiant unique et qui est la plupart du temps de type numérique.

Pour laisser le SGBDR gérer l'incrémentation de cette colonne, on peut utiliser un objet de type séquence et l'associer à une colonne.

Lors de chaque ajout de ligne dans la table, il n'est pas nécessaire d'attribuer une valeur à cette colonne, le système s'en occupe. Ainsi, on est certain de ne jamais avoir de doublons sur cette colonne.

Une séquence peut être créée indépendamment d'une colonne de table. Dans le fonctionnement d'une application, un compteur peut servir à de multiples endroits.

Syntaxe de base

```
CREATE SEQUENCE <nom séquence>;
```

Le système va créer un objet qui prend par défaut la valeur 1 et s'incrémente de 1 en 1.

Exemple

```
CREATE SEQUENCE S_NUMERO;
```

Si maintenant nous voulons attribuer une valeur de départ, un pas précis ou une valeur maximum, il faut utiliser les options proposées.

Exemple

```
CREATE SEQUENCE S_NUMERO START WITH 5 INCREMENT BY 1
MINVALUE 2 MAXVALUE 999999 CYCLE;
```

Dans cet exemple, la séquence va démarrer à la valeur 5, sera incrémentée de 1, et ne pourra jamais être inférieure à 2 et supérieure à 999999. Lorsque la séquence aura atteint la valeur maximum 999999, elle recommencera à 2.

Si l'on ne veut pas que la séquence boucle après avoir atteint la valeur maximum, il faut enlever l'option CYCLE. Ainsi, le SGBDR retournera une erreur lorsque l'on demandera la valeur suivante de 999999. Il est assez risqué d'autoriser une séquence à boucler sur elle-même, cela peut générer des doublons dans la base ou provoquer une erreur si la colonne est déclarée UNIQUE.

Pour connaître la valeur actuelle d'une séquence ou faire avancer une séquence, il faut utiliser les fonctions suivantes.

Pour Oracle :

- CURRVAL : récupère la valeur actuelle.
- NEXTVAL : incrémente la séquence du pas prévu (1 par défaut).

Exemple

```
SELECT S_NUMERO.NEXTVAL FROM DUAL;
SELECT S_NUMERO.CURRVAL FROM DUAL;
```

La table DUAL est utilisée pour travailler sur des fonctions sans lien avec une table réelle. C'est une table virtuelle.

SQL Server n'utilise pas DUAL mais la syntaxe suivante :

```
SELECT current_value
FROM sys.sequences
WHERE name = 'S_NUMERO' ;
SELECT NEXT VALUE FOR S_NUMERO;
```

Pour lier une séquence à une colonne d'une table, il faut créer pour Oracle un trigger. Nous verrons dans un autre chapitre l'utilisation des triggers, mais, pour information, voici comment créer un trigger sur la colonne idChambre de la table Chambres qui utilise la séquence S_NUMERO.

Exemple de colonne auto-incrémentée avec Oracle

```
CREATE OR REPLACE TRIGGER TR_NUMERO
```

```
BEFORE INSERT ON Chambres
FOR EACH ROW
DECLARE

BEGIN

   SELECT S_NUMERO.NEXTVAL
   INTO :NEW.NUMERO FROM DUAL;
END;
/
```

Ainsi, à chaque insertion dans la table Chambres, il n'est pas nécessaire d'indiquer de valeur pour la colonne idChambre.

Il existe une autre façon d'utiliser la séquence lors de l'insertion, il faut la coder dans l'ordre INSERT :

```
INSERT INTO Chambres VALUES (S_NUMERO.NEXTVAL,2,1,2,'2',NULL);
```

Attention : il peut y avoir des « trous » dans une série de chiffres issus d'une séquence. En effet, si après un INSERT l'utilisateur ne valide pas la transaction par un COMMIT ou qu'il lance un ROLLBACK, la dernière valeur de la séquence n'est pas modifiée et continue de s'incrémenter normalement.

Oracle implémente l'objet séquence, comme indiqué ci-dessus.

MySQL n'implémente pas les séquences. Il existe néanmoins la fonction AUTO_INCREMENT sur une colonne lors de la création de la table. Cette colonne doit être la clé primaire.

Exemple de colonne auto-incrémentée avec MySQL

```
CREATE TABLE Chambres (idChambre       INTEGER AUTO_INCREMENT,
                       Hotel           INTEGER,
                       TypeChambre     INTEGER,
                       NumChambre      VARCHAR(6),
                       Commentaire     VARCHAR(400),
CONSTRAINT PK_Chambres PRIMARY KEY (idChambre));
```

Cette fonction existe aussi dans SQL Server et se nomme IDENTITY. Dans PostgreSQL, pour obtenir une colonne qui s'auto-incrémente, on choisira le type SERIAL à la place du type INTEGER.

Exemple de colonne auto-incrémentée avec SQL Server

```
CREATE TABLE Chambres (idChambre       INTEGER IDENTITY(1,1) NOT NULL,
```

```
                        Hotel           INTEGER,
                        TypeChambre     INTEGER,
                        NumChambre      VARCHAR(6),
                        Commentaire     VARCHAR(400),
CONSTRAINT PK_Chambres PRIMARY KEY (idChambre));
```

Pour supprimer une séquence, il faut utiliser DROP SEQUENCE.

Exemple

```
DROP SEQUENCE S_NUMERO;
```

Pour modifier une séquence, il faut utiliser ALTER SEQUENCE.

Exemples

Changer la valeur maximum de la séquence :

```
ALTER SEQUENCE S_NUMERO MAXVALUE 888888;
```

Modifier le pas de 1 à 5 :

```
ALTER SEQUENCE S_NUMERO INCREMENT BY 5;
```

LA SUPPRESSION DE TABLES

La suppression de tables est une opération simple mais qu'il faut manier avec prudence. La suppression est le plus souvent définitive. Il n'y aura aucune possibilité de récupérer les données de la table une fois l'ordre lancé.

1. L'ORDRE DROP

L'ordre DROP permet de supprimer définitivement une table. La table et son contenu sont supprimés. L'order DROP est aussi utilisé sur d'autres objets de base de données comme les vues ou une base.

On utilise souvent l'ordre DROP juste avant la création d'une table. Ainsi, on évite les erreurs sur la table existante.

La commande détruit automatiquement les index et contraintes posés sur cette table ainsi que les commentaires. En revanche, l'ordre ne détruit pas les synonymes.

S'il s'agit d'une table sensible, il est préférable de la sauvegarder au préalable par un CREATE ... AS SELECT … par exemple.

Attention : la table ne doit pas être en cours d'utilisation par une autre personne.

Dans le cas d'une fausse manipulation, il ne sera pas possible de récupérer la table (ROLLBACK). Certaines versions de SGBDR autorisent la récupération après un DROP (Oracle à partir de la version 10g, par exemple).

Dans MySQL et SQL Server, les ordres de manipulation de tables entraînent une validation automatique (COMMIT). Il est donc impossible de récupérer la table après un DROP.

Syntaxe

```
DROP TABLE nom_de_table;
```

Exemple

```
DROP TABLE Chambres;
```

LA MODIFICATION DE TABLES

Une fois les tables créées, elles vont évoluer dans le temps et il faudra ajouter ou supprimer une colonne ou encore intervenir sur les contraintes des colonnes.

Dans certains cas, il faudra renommer une table. C'est ce que nous allons détailler dans les sections suivantes.

1. L'ORDRE ALTER

L'ordre ALTER est utilisé pour réaliser plusieurs actions. Il peut être utilisé pour supprimer ou ajouter une colonne d'une table, mais également pour ajouter ou supprimer une contrainte ou ajouter une valeur par défaut à une colonne.

Attention : il est interdit de changer le nom d'une colonne ainsi que son type et ses attributs NULL ou NOT NULL.

Certains SGBDR acceptent l'ordre MODIFY et permettent ainsi de modifier le type d'une colonne.

Attention néanmoins au contenu de cette colonne. Lors du passage d'un type VARCHAR à un type INTEGER, la conversion automatique réalisée par le SGBDR va modifier le contenu des données.

Il existe un risque de perdre de l'information ou de rendre certaines données incompatibles avec leur utilisation. D'une manière générale, il est déconseillé de modifier le type d'une colonne. Pour prévenir tout problème, il faut vider la table en amont avant de changer les types.

Syntaxe

```
ALTER TABLE nom_de_table [ADD nom_de_colonne type_colonne]
                         [,DROP COLUMN nom_de_colonne]
                         [,ADD CONSTRAINT nom_contrainte]
                         [,DROP CONSTRAINT nom_contrainte];
```

Exemples

Ajouter une colonne :

```
ALTER TABLE Chambres ADD Vue VARCHAR(20);
```

Table Chambres

Chambres
idChambre
Hotel
TypeChambre
NumChambre
Commentaire
Vue

La colonne ajoutée se place toujours en fin de définition de la table.

Supprimer une colonne :

```
ALTER TABLE Chambres DROP COLUMN Vue;
```

2. RENOMMER UNE TABLE (RENAME)

L'ordre RENAME permet de renommer une table. Cette commande peut être utilisée dans le cas où la table doit être recréée tout en conservant la version actuelle en archive. Il faut donc renommer la table actuelle puis refaire le CREATE initial.

Cela est plus simple et plus rapide que de faire un CREATE … AS SELECT … de la table concernée.

Syntaxe

Oracle :

```
RENAME nom_de_table_ancien TO nom_de_table_nouveau;
```

MySQL :

```
RENAME TABLE Chambres TO SAV_Chambres;
```

SQL Server :

```
EXEC sp_rename 'NomTable', 'NouveauNomTable'
```

Exemple

```
EXEC sp_rename 'Chambres', 'SAV_Chambres';
```

PostgreSQL :

```
ALTER TABLE Chambres RENAME TO SAV_Chambres;
```

Attention : certains SGBDR ne connaissent pas cet ordre. Dans ce cas, il faut créer une autre table avec la même structure comme ceci :

```
CREATE TABLE SAV_Chambres AS SELECT * FROM Chambres;
```

puis supprimer la table initiale :

```
DROP TABLE Chambres;
```

VIDER UNE TABLE

1. L'ORDRE TRUNCATE

L'ordre TRUNCATE est utilisé pour supprimer toutes les occurrences d'une table, sans restriction. Cet ordre occupera une seule ligne dans le journal. En cas de retour arrière, toutes les occurrences seront récupérées. Il ne sera pas possible de reprendre une partie des enregistrements. L'intérêt de l'ordre TRUNCATE est qu'il libère l'espace disponible par la suppression des lignes, dans le fichier physique de la base. Un autre avantage de cet ordre est qu'il réinitialise l'auto-incrémentation si cette option est appliquée sur une colonne.

Les déclencheurs ne sont pas exécutés.

Cet ordre est généralement employé par les administrateurs de base de données.

Syntaxe

```
TRUNCATE TABLE <nom table>
```

Exemple

```
TRUNCATE TABLE Chambres;
```

LES VUES

Dans cette section, nous allons voir comment créer ou supprimer des vues. Les vues sont des éléments très utilisés dans la programmation SQL. Elles permettent principalement de créer des tables « virtuelles » spécifiques pour un domaine ou pour une classe d'utilisateurs.

1. POURQUOI UTILISER DES VUES ?

Dans une base de données, il y a les tables permanentes qui ont été définies après une analyse des besoins et une modélisation sous forme de tables.

Si l'on respecte le modèle relationnel, il n'y a pas de données redondantes dans les tables, à l'exception des clés utilisables pour les jointures. En revanche, les utilisateurs ou les développeurs ont des besoins d'extractions spécifiques de la base. Ces extractions se matérialisent sous forme de requêtes lancées manuellement ou incluses dans les programmes.

Si ces demandes sont répétitives ou communes à plusieurs utilisateurs, il peut être nécessaire de créer une vue. La vue est une représentation logique de la base qui résulte d'une requête pour un besoin spécifique et répétitif. Contrairement à une table, elle n'est pas stockée sur disque (sauf demande spécifique) mais en mémoire.

La vue peut également permettre de simplifier pour un utilisateur la base de données. Il n'a pas besoin de connaître l'ensemble du schéma mais simplement quelques éléments spécifiques utiles dans son métier.

Si vos tables contiennent des informations confidentielles, la vue permet de masquer les colonnes en cause. Ainsi, l'utilisateur ne voit que ce que l'on veut bien lui montrer.

La création d'une vue suit le même mécanisme que le CREATE TABLE … AS SELECT … expliqué dans les sections précédentes. En effet, la vue est une agrégation de colonnes issues d'une ou plusieurs tables.

L'avantage majeur d'une vue est qu'elle est en permanence mise à jour. En effet, une vue est mise à jour automatiquement lorsque les tables qu'elle utilise sont modifiées. En revanche, une vue n'étant pas un objet proprement dit mais un résultat de requête, on ne peut pas faire de mise à jour de données sur une vue. Les données appartiennent aux tables du SELECT.

Une vue représente à un instant t l'image des tables qu'elle utilise.

2. LA CREATION DE VUES

La création s'effectue par l'ordre CREATE VIEW puis un ordre SELECT qui récupère les colonnes que l'on veut extraire des tables concernées.

Comme pour le CREATE TABLE ... AS SELECT ... toutes les possibilités offertes par l'ordre SELECT sont utilisables. On peut avoir des requêtes assez complexes pour le remplissage d'une vue.

Attention néanmoins aux temps d'exécution de la requête, et également au nombre de lignes ramenées. Une vue n'a généralement pas d'index, donc un SELECT sur une vue parcourt l'ensemble de la vue. Une vue ne doit pas se substituer à une table. Si les données contenues dans les vues sont nombreuses et ont besoin d'être archivées ou utilisées de manière récurrente, peut-être faut-il revoir le schéma des données et créer une nouvelle table.

Pour un utilisateur, la vue se comporte comme une table. Il peut réaliser ses requêtes sur la vue comme sur une table. Il est ainsi possible de créer un ensemble de vues par type d'utilisateur ou métier de l'entreprise. Ainsi, chacun aura les données dont il a besoin pour une utilisation précise.

Syntaxe

```
CREATE VIEW <Nom_Vue> AS SELECT ...
```

Si on veut la liste des chambres avec le n° de l'hôtel, son numéro, le type de lit, le nombre de lits et sa description, on va faire un SELECT sur les trois tables et récupérer les colonnes nécessaires avec cet ordre :

Exemple

```
CREATE VIEW V_Chambre_desc AS
SELECT Chambres.Hotel, Chambres.NumChambre,
       TypesChambre.TypeLit,
       TypesChambre.NombreLit, TypesChambre.Description
FROM Chambres, TypesChambre
WHERE Chambres.TypeChambre = TypesChambre.idTypeChambre;
```

Ce qui va avoir pour effet de créer la vue V_Chambre_desc qui aura cette structure.

Table V_Chambre_desc

V_Chambre_desc
Hotel
NumChambre
TypeLit
NombreLit
Description

À chaque fois que les données des tables Chambres et TypesChambre seront modifiées, la vue V_Chambre_desc sera mise à jour automatiquement.

L'accès à la vue se fera comme pour une table classique avec l'ordre SELECT.

Exemple

Table Chambres

idChambre	Hotel	TypeChambre	NumChambre	Commentaire
23	4	2	2	NULL
16	3	2	2	NULL
9	2	2	2	NULL
2	1	2	2	NULL
3	1	3	3	NULL
10	2	3	3	NULL
17	3	3	3	NULL
24	4	3	3	NULL
25	4	4	4	NULL

Table TypesChambre

idTypeChambre	NombreLit	TypeLit	Description
1	1	lit simple	1 lit simple avec douche
2	2	lit simple	2 lits simples avec douche
3	3	lit simple	3 lits simples avec douche et WC séparés
4	1	lit double	1 lit double avec douche
5	1	lit double	1 lit double avec douche et WC séparés
6	1	lit double	1 lit double avec bain et WC séparés
7	1	lit XL	1 lit double large avec bain et WC séparés

Avec le contenu des tables ci-dessus, la vue V_Chambres_desc contient :

```
SELECT * FROM V_Chambres_desc;
```

Hotel	NumChambre	TypeLit	NombreLit	Description
1	2	lit simple	2	2 lits simples avec douche
2	2	lit simple	2	2 lits simples avec douche
3	2	lit simple	2	2 lits simples avec douche
4	2	lit simple	2	2 lits simples avec douche
4	3	lit simple	3	3 lits simples avec douche et WC séparés
3	3	lit simple	3	3 lits simples avec douche et WC séparés

2	3	lit simple	3	3 lits simples avec douche et WC séparés
1	3	lit simple	3	3 lits simples avec douche et WC séparés

STOCKAGE DES VUES

La vue n'est pas stockée physiquement sur disque, elle est montée en mémoire lors de sa première utilisation. La récupération des données d'une vue est donc très rapide à partir de la deuxième demande. En termes de performance, les vues sont une solution à analyser à condition d'utiliser plusieurs fois le même SELECT dans une session.

Si on accède souvent à ces données, l'accès à une vue est plus rapide que l'exécution du SELECT sur plusieurs tables.

Néanmoins, il faut se rapprocher des DBA, qui donneront les normes en vigueur dans l'entreprise et les bonnes pratiques concernant l'utilisation des vues.

Attention : certains SGBDR proposent de stocker les vues sur disque. Elles se comportent ainsi exactement comme des tables et donc nécessitent une autre analyse en termes de performance.

3. LA SUPPRESSION DE VUES

L'ordre DROP VIEW permet de supprimer définitivement une vue. La vue et son contenu sont supprimés.

L'ordre DROP est définitif, il est impossible de récupérer la vue lors d'une mauvaise manipulation.

Cet ordre n'a aucune influence sur les tables qui sont utilisées dans le SELECT de création de la vue.

Syntaxe

```
DROP VIEW nom_vue;
```

Exemple

```
DROP VIEW V_Chambres_desc;
```

LES INDEX

Dans cette section, nous abordons une notion importante : les index. Toutes les bases de données utilisent des index. L'implémentation physique de ceux-ci diffère d'un SGBDR à un autre.

Il existe plusieurs types d'index et plusieurs méthodes pour traiter ces index. Nous verrons comment créer et supprimer ces index et pourquoi utiliser tel type d'index en fonction des besoins que l'on a.

1. LES INDEX ET LA NORME SQL

Tout d'abord, il faut préciser que les index ne font pas partie de la norme SQL. En effet, l'index est utilisé pour accélérer une recherche dans une table et s'appuie sur des fichiers physiques qui sont créés lors de la création d'un index.

Il s'agit donc d'une implémentation physique et, dans la norme SQL, à l'instar du stockage des tables, l'aspect physique n'est pas traité. Chaque SGBDR l'implémente à sa façon.

En revanche, les index sont quasiment indispensables dans une base de données relationnelle. Le temps d'accès aux données étant un paramètre très important pour tous les utilisateurs et les développeurs, l'utilisation ou non d'un index peut augmenter les temps de réponse de manière exponentielle.

Dans le cas de tables avec plusieurs millions de lignes, l'accès à une même donnée peut mettre plusieurs heures sans index et quelques secondes avec un index.

Sans index, l'ensemble de la table sera parcouru séquentiellement jusqu'à trouver l'enregistrement demandé.

C'est le SGBDR qui gère les fichiers d'index, l'utilisateur ne peut pas intervenir sur la technique de stockage.

Attention néanmoins à ne pas créer des index sur toutes les colonnes. La création d'un index doit découler d'une réflexion sur l'utilisation de la table par les programmes et les utilisateurs. Les index ralentissent les traitements lors des mises à jour, car le SGBDR doit recalculer les clés après chaque ajout, suppression ou modification de lignes.

Il faut cibler les colonnes, analyser les taux de mise à jour d'une table et se baser sur une analyse fonctionnelle des données, puis consulter les DBA sur les normes en place dans l'entreprise et la méthode utilisée par défaut par le SGBDR.

2. LES DIFFERENTES METHODES D'ORGANISATION DES INDEX

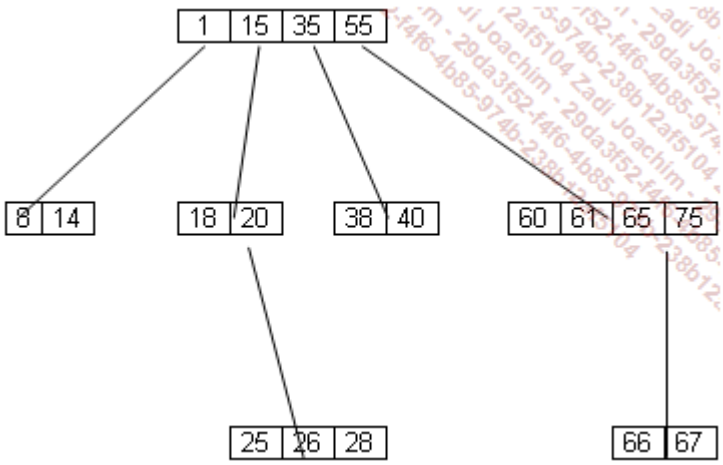
Il existe cinq méthodes principales de gestion des index : hachage, séquentiel, bitmap, arbre et cluster.

En tant que développeur, il est peu probable que le choix de la méthode d'indexation des tables soit de votre ressort. C'est un domaine souvent réservé au DBA. Mais il est bon néanmoins de connaître leur existence afin de répondre à des problématiques de performance.

Nous n'allons pas rentrer dans le détail de chacune de ces méthodes. Les plus utilisées sont les index en arbre (B*Tree) et les index bitmap.

Par défaut, Oracle va créer des index de type B*Tree si rien n'est précisé. Ce type d'index convient aux tables volumineuses avec des clés uniques ou avec très peu de doublons (moins de 5 %).

*Exemple d'index B*Tree*



Si l'on demande par exemple :

```
SELECT * FROM <Table> WHERE Identifiant = 18;
```

il passera directement par 15 puis accède au 18.

Les index de type bitmap sont à utiliser pour les tables volumineuses qui ont beaucoup de clés en commun avec un taux de mise à jour très faible. Le principe est de créer pour chaque valeur une chaîne de bits et ensuite de ne travailler que sur les bits.

Exemple d'index bitmap

idTypeChambre	NombreLit	TypeLit	Lit simple	Lit double	Lit XL
1	1	Lit simple	1	0	0
2	2	Lit simple	0	1	0
3	2	Lit simple	0	1	0
4	1	Lit double	0	0	1
5	1	Lit double	0	1	0
7	1	Lit XL	1	0	0

Dans la colonne Lit double, on trouve la chaîne 011010. Ensuite, le SGBDR accède aux enregistrements très rapidement.

N'oubliez pas de consulter vos DBA avant toute création d'index.

3. LA CREATION D'INDEX

L'index est le principal élément permettant d'améliorer les performances d'accès aux données. A contrario, la mise à jour des données sera pénalisée étant donné que le SGBDR devra maintenir les fichiers d'index à chaque mise à jour de la table.

Dans le cas d'une base qui est très peu évolutive, il peut être intéressant de multiplier les index afin d'optimiser les temps de réponse. Les index doivent être posés sur des colonnes qui ont des valeurs distinctives. L'index le plus performant sera sur une clé unique, par exemple.

Il ne faut pas confondre PRIMARY KEY et CREATE INDEX UNIQUE, le premier étant une contrainte d'intégrité qui vérifie l'unicité d'une colonne (voir la section L'intégrité des données), le deuxième étant une création de fichier d'index utilisé pour les performances.

La confusion vient du fait que certains SGBDR comme Oracle, SQL Server et MySQL créent automatiquement un index lorsque l'on crée une clé primaire.

La création d'index peut se réaliser sur une ou plusieurs colonnes. Il faut bien choisir les colonnes nécessitant un index. Il faut analyser les requêtes les plus souvent utilisées, les plus coûteuses en temps, les clés primaires et étrangères posées…

Il n'y a pas de règles tout établies pour la création des index. Il est conseillé de créer des index sur les colonnes qui sont déclarées en PRIMARY KEY (si le SGBDR ne l'a pas déjà fait), sur les colonnes en FOREIGN KEY, sur les colonnes les plus accédées, les colonnes qui servent de jointure entre les tables, les colonnes les plus discriminantes, etc.

Si des problèmes de performance apparaissent après la création d'un index, il faut le supprimer et vérifier l'utilisation qui est faite de la table, notamment en mise à jour.

Syntaxe

```
CREATE [UNIQUE] INDEX <nom index> ON <nom table>
<nom colonne 1> [ASC|DESC], <nom colonne 2> [ASC|DESC], ... ...
```

Les options possibles sont :

- UNIQUE, qui indique au SGBDR que cette colonne sera unique, donc il contrôle également lors d'une insertion que cette unicité est vérifiée. Il est possible de créer autant de UNIQUE INDEX que nécessaire, contrairement à la PRIMARY KEY, qui est unique.

- ASC ou DESC, qui indique au SGBDR comment l'index doit être trié. Par défaut, il sera en ordre ascendant. Cette option peut être utile si les requêtes demandent une restitution de cette colonne (avec ORDER BY) dans un ordre précis. Si l'index est créé avec l'option DESC et que l'ordre contient un ORDER BY <colonne> ASC, l'index ne sera pas efficace.

Exemples

```
CREATE INDEX I2_TypeChambre ON Chambres (TypeChambre);
CREATE INDEX I3_TypHotel ON Chambres (TypeChambre, Hotel);
CREATE INDEX I4_NumChambre ON Chambres (NumChambre DESC);
```

Une colonne peut être utilisée dans plusieurs index. Il est intéressant de donner un numéro aux index. Ainsi, on peut connaître automatiquement le nombre d'index posés sur une table.

Lorsque plusieurs colonnes composent l'index, il est préférable de mettre la colonne la plus discriminante en premier. Ainsi, par exemple, si l'index est sur trois colonnes et qu'une requête ne teste que la première ou les deux premières colonnes, l'index sera utilisé partiellement quand même par le SGBDR.

Il n'est pas nécessaire en règle générale d'indexer les petites tables. L'analyse et la maintenance de l'index seront plus coûteuses que la lecture de la table complète.

La création d'un index peut être très longue en termes de temps de réponse si la création s'effectue sur une table importante. L'espace disque est également à surveiller car le système va créer des fichiers pour stocker ces index.

4. LA SUPPRESSION D'INDEX

L'ordre DROP INDEX permet de supprimer un index.

L'ordre DROP est définitif, il sera impossible de récupérer l'index dans le cas d'une mauvaise manipulation. Un utilisateur ne peut supprimer que les index qu'il a lui-même créés et les index que l'administrateur de la base de données a autorisés.

Cet ordre ne peut pas être utilisé pour supprimer un index créé par le système après une création de clé primaire.

Syntaxe

```
DROP INDEX <nom_index>
```

Exemple

```
DROP INDEX I2_TypeChambre;
DROP INDEX I4_NumChambre;
```

L'INTEGRITE DES DONNEES

Les contraintes d'intégrité permettent de maintenir la base cohérente. On confie au SGBDR les tâches de contrôle de la validité des données qui sont insérées.

Les contraintes se substituent aux contrôles réalisés par programme.

Il existe plusieurs types de contrôles. Il est possible d'indiquer au SGBDR :

- Quelle est la valeur par défaut à affecter à une colonne (DEFAULT),
- Qu'une colonne ne peut pas être null (NOT NULL),
- Qu'une colonne doit être unique (UNIQUE),
- Ou de coder un contrôle sur une colonne (CHECK).

Il existe également deux contraintes particulières qui sont la clé primaire et la clé étrangère. Nous allons détailler leurs fonctions.

1. LA CLE PRIMAIRE (PRIMARY KEY)

Par définition, la clé primaire est la clé principale d'une table. Le SGBDR va contrôler systématiquement à chaque insertion ou modification que la clé est unique dans la table. Dans le cas contraire, il rejette la demande de modification avec un message d'erreur de ce type : « Violation constraint ... ».

La clé primaire est toujours une clé unique. Elle est composée d'une ou de plusieurs colonnes en fonction de la méthode de création, le plus important étant qu'il ne peut y avoir deux lignes de la table avec cette même clé.

Il s'agit souvent d'un numéro que l'on incrémente de 1 à chaque création d'une ligne dans la table.

On peut utiliser aussi des données métier, comme des numéros de sécurité sociale ou des numéros de permis de conduire, mais il faut être certain que toutes les lignes de la table ont une valeur pour cette ou ces colonnes. En effet, une clé primaire ne peut pas prendre la valeur NULL.

La création d'une clé primaire génère dans la plupart des SGBDR la création automatique d'un index sur cette colonne.

Il y a deux méthodes pour déclarer une clé primaire. Si la clé correspond à une seule colonne, il faut utiliser cette syntaxe :

Exemple : déclaration d'une clé primaire sur une colonne

```
CREATE TABLE Chambres (
       idChambre       INTEGER PRIMARY KEY,
       Hotel           INTEGER,
       TypeChambre     INTEGER
```

```
        NumChambre      VARCHAR(6),
        Commentaire     VARCHAR(400)
);
```

Automatiquement, la colonne idChambre sera NOT NULL et UNIQUE.

S'il y a plusieurs colonnes qui composent la clé, il faudra utiliser la clause CONSTRAINT qui permet de déclarer une contrainte d'intégrité.

Exemple : déclaration d'une clé primaire sur plusieurs colonnes

```
CREATE TABLE Chambres (
        idChambre         INTEGER,
        Hotel             INTEGER,
        TypeChambre       INTEGER,
        NumChambre      VARCHAR(6),
        Commentaire     VARCHAR(400),
        CONSTRAINT PK_Chambres PRIMARY KEY (Hotel, NumChambre)
);
```

Les colonnes Hotel et NumChambre composent la clé. Elles seront NOT NULL et l'association des deux est UNIQUE.

Nous pouvons remarquer également que l'on donne un nom à la contrainte, ici PK_Chambres. De cette façon, lorsque le SGBDR nous signalera une insertion incorrecte dans la table, il citera la contrainte explicitement.

Les contraintes de clé primaire sont souvent notées PK_<nom table> (PK pour Primary Key). Ainsi, le développeur ou l'utilisateur peut indiquer simplement en lisant le nom de la contrainte qu'il s'agit de la clé primaire et sur quelle table le problème a été rencontré.

Dans le cas où la table est déjà créée et qu'il faut ajouter une clé primaire, il faudra utiliser ALTER TABLE.

Exemple : création d'une clé primaire sur une table existante

```
ALTER TABLE Chambres ADD
CONSTRAINT PK_Chambres PRIMARY KEY (Hotel, NumChambre);
```

2. LA CLE ETRANGERE (FOREIGN KEY)

La clé étrangère s'appuie sur une autre table pour indiquer comment contrôler les colonnes de notre table principale.

Il faut que la table étrangère contienne une clé primaire pour que le SGBDR puisse faire le lien.

Le principe est simple : à chaque mise à jour de la table principale, le SGBDR vérifie que la valeur de la colonne en question existe bien dans l'autre table.

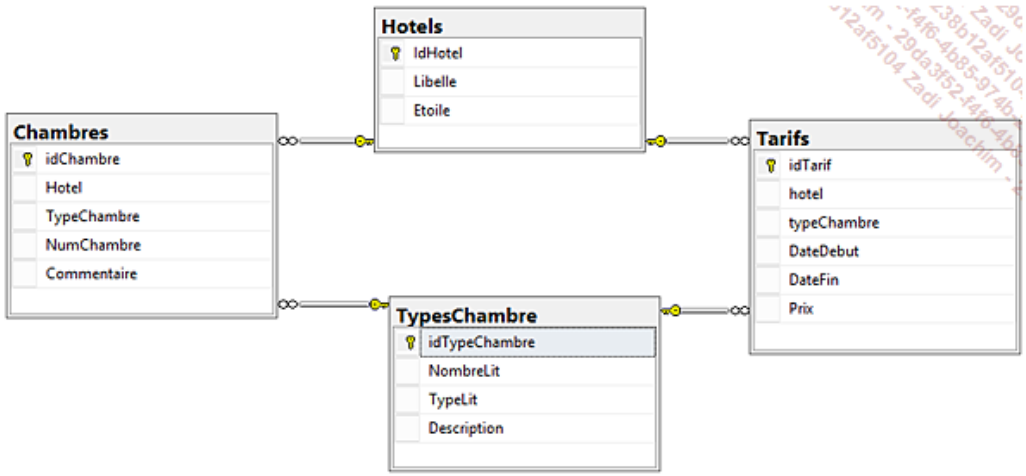
Il est préférable que le nom de la colonne soit identique entre les deux tables concernées. Dans le cas contraire, le SGBDR prendra la clé primaire de la table secondaire.

Il est possible d'avoir plusieurs contraintes de clés étrangères sur une même table.

L'utilisation ou non d'une clé étrangère dépend des normes de développement de l'entreprise. On peut préférer contrôler l'intégrité par programme et non laisser la base de données le faire.

Dans le cas d'utilisation des clés étrangères, il faut que vos programmes contrôlent les codes retour à chaque mise à jour de la base, afin de vérifier qu'aucune contrainte n'a été violée.

Exemple



Syntaxe

```
CONSTRAINT <nom contrainte> FOREIGN KEY (colonne 1, colonne 2 ...)
REFERENCES <table secondaire> );
```

Dans l'exemple, il faut tout d'abord créer une clé primaire sur la table TypesChambre et ensuite créer la clé étrangère dans la table Chambres.

Il est possible d'ajouter les instructions ON DELETE CASCADE ou ON UPDATE CASCADE pour que la suppression ou la mise à jour de la table parent impacte la table enfant. Il existe aussi les instructions ON DELETE SET NULL ou ON UPDATE SET NULL pour mettre la valeur NULL dans la table enfant lors de la suppression ou la modification de la donnée dans la table parent. Enfin, il est possible d'utiliser les instructions ON DELETE SET DEFAULT ou ON UPDATE SET DEFAULT pour que la valeur par défaut impacte également la table enfant.

Exemple de création d'une clé étrangère

```
ALTER TABLE TypeChambres ADD
CONSTRAINT PK_TypeChambres PRIMARY KEY (idTypeChambre);


CREATE TABLE Chambres (idChambre       INTEGER PRIMARY KEY,
                       Hotel           INTEGER,
                       TypeChambre     INTEGER,
                       NumChambre      VARCHAR(6),
                       Commentaire     VARCHAR(400),
CONSTRAINT FK_TypeChambre FOREIGN KEY (TypeChambre) REFERENCES
TypesChambres (idTypeChambre)) ON DELETE CASCADE ON UPDATE SET
DEFAULT;


CREATE TABLE Chambres (idChambre       INTEGER PRIMARY KEY,
                       Hotel           INTEGER,
                       TypeChambre     INTEGER,
                       NumChambre      VARCHAR(6),
                       Commentaire     VARCHAR(400),
CONSTRAINT FK_TypeChambre FOREIGN KEY (TypeChambre) REFERENCES
TypesChambres (idTypeChambre)) ON DELETE SET NULL;
```

3. LES VALEURS PAR DEFAUT (DEFAULT)

Certaines colonnes ne sont pas forcément renseignées par l'utilisateur au moment de la saisie. Le développeur peut alors décider d'attribuer une valeur par défaut à cette colonne afin de ne pas avoir de valeur NULL dans la base.

Dans la section suivante de ce même chapitre, nous aborderons la gestion des valeurs NULL et nous verrons que celle-ci est assez particulière.

Il est préférable d'avoir des valeurs par défaut bien définies afin d'éviter d'obtenir des résultats de requêtes non conformes.

L'attribution d'une valeur par défaut s'effectue lors de la création de la table ou par un ALTER TABLE.

Il est possible d'indiquer différentes valeurs par défaut :

- Une valeur numérique ou caractère,
- Des résultats de fonction comme la date du jour (CURRENT_DATE ou SYSDATE) ou la date et l'heure du jour (CURRENT_TIMESTAMP), utiles notamment pour les colonnes qui tracent les mises à jour réalisées sur une table,
- Pour attribuer le nom de l'utilisateur qui a réalisé la mise à jour avec la fonction USER.

Pour qu'une valeur par défaut soit prise en compte par le SGBDR, il faut qu'aucune valeur ne soit affectée à la colonne lors de l'ordre INSERT. Il faut que la colonne n'apparaisse pas dans l'ordre. Si on indique une valeur blanc ou null, elle se substitue à la valeur par défaut.

Exemple de valeurs par défaut

Lors de la création de la table :

```
CREATE TABLE Chambres (
      idChambre       INTEGER PRIMARY KEY
      Hotel           INTEGER
      TypeChambre     INTEGER DEFAULT 2
      NumChambre      VARCHAR(6),
      Commentaire     VARCHAR(400)
);
```

Une fois la table créée, on peut modifier une valeur par défaut ainsi (ne fonctionne pas dans MySQL) :

```
ALTER TABLE Chambres MODIFY TypeChambre DEFAULT 3;
```

Ajouter une colonne et lui attribuer une valeur par défaut :

```
ALTER TABLE Chambres ADD (Vue VARCHAR(20) DEFAULT 'Mer');
```

4. LA VALEUR NULL

Lors d'une insertion ou d'une modification, si l'on ne précise pas de valeur pour une colonne, celle-ci est vide et prend la valeur NULL.

NULL ne correspond ni à un espace ni à la valeur zéro. Pour utiliser une colonne contenant NULL, il faut dans le SELECT préciser IS NULL afin de tester le contenu.

Par exemple, si les lignes de la table Chambres ont ce contenu :

IdChambre	Hotel	TypeChambre	NumChambre	Commentaire
1	1	1	1	Vue sur mer
2	1	2	2	
3	1	3	3	NULL
4	1	4	4	NULL
5	1	5	5	Vue sur mer
6	1	6	6	NULL
7	1	7	7	Vue sur me

Les lignes 2, 3, 4 et 6 ont été entrées sans préciser de valeur pour le commentaire, donc la colonne est considérée comme NULL mais le commentaire de la ligne 2 n'est pas NULL.

Si on réalise ensuite une sélection sur la colonne Commentaire comme :

```
SELECT COUNT(*) FROM Chambres WHERE commentaire NOT IN
('Vue sur mer');
```

Le résultat est 1. En effet, les trois lignes qui ont des valeurs NULL ne sont pas prises en compte, même si le commentaire est bien différent des valeurs recherchées. Par contre, la ligne 2 est comptée car le commentaire n'est pas NULL même s'il paraît vide.

Il faut donc manier avec prudence les colonnes qui acceptent les NULL. Par précaution, il est préférable de mettre toutes les colonnes qui sont utilisées dans une clause WHERE en NOT NULL et leur affecter des valeurs par défaut lors des insertions et modifications.

Toutes les colonnes contenues dans une clé doivent être en NOT NULL afin d'éviter tout problème d'indexation.

5. LA CONTRAINTE D'UNICITE UNIQUE

Comme indiqué dans la section qui traite des clés primaires, la clause UNIQUE permet de préciser au SGBDR que la valeur de cette colonne ne doit pas être en doublon dans la table. Il est conseillé d'utiliser cette propriété pour les colonnes de type Libellé.

Attention, une colonne déclarée en UNIQUE peut quand même contenir du NULL si on ne précise pas la clause NOT NULL au moment de sa création. Dans ce cas, il peut y avoir plusieurs lignes avec la colonne à NULL, et l'unicité ne s'applique alors pas.

Exemple : ajout de l'unicité lors de la création de la table

```
CREATE TABLE Hotels (
      idHotel       INTEGER PRIMARY KEY,
      Libelle       VARCHAR(50) UNIQUE,
      Etoile        VARCHAR(5)
);
```

Modifier une colonne (ne fonctionne pas dans MySQL)

```
ALTER TABLE Hotels MODIFY (Libelle UNIQUE)
```

6. LA CONTRAINTE DE VERIFICATION CHECK

Cette clause permet de réaliser toutes sortes de contrôles sur les colonnes d'une table. Il faut l'utiliser avec précaution car elle peut être source de problèmes de performances significatifs. En effet, à chaque modification de la colonne, l'ordre est exécuté et peut ralentir sensiblement les mises à jour.

On peut contrôler la colonne avec des valeurs alphanumériques ou numériques, mais également par l'appel à une fonction ou encore en spécifiant un SELECT spécifique.

Attention : la clause CHECK est implémentée dans MySQL mais n'a aucun effet. Celui-ci ne tient absolument pas compte de ces contraintes. Il faut passer par des contrôles par programme ou utiliser un trigger. MySQL implémente la clause pour des raisons de compatibilité avec la norme uniquement. Oracle l'implémente et la gère, mais pas complètement, on ne peut pas faire appel à d'autres tables sous forme de sous-SELECT, par exemple. Il se contente de gérer les contrôles sur des constantes, sur des colonnes de la table et sur des fonctions simples.

Exemple de contrôles possibles selon la norme SQL92

```
CREATE TABLE Chambres (
       idChambre      INTEGER PRIMARY KEY,
       Hotel          INTEGER CHECK (VALUE BETWEEN 1 AND 999),
       TypeChambre    CHECK (VALUE IN (SELECT
       idTypeChambre FROM TypesChambre)),
       NumChambre     VARCHAR(6),
       Commentaire    VARCHAR(400)
);
```

La colonne Hotel doit prendre une valeur entre 1 et 999.

Le contenu de la colonne TypeChambre doit appartenir aux valeurs de la table TypesChambre. Généralement, il est préférable d'utiliser une clé étrangère plutôt que ce type de vérification.

Avec Oracle, on peut créer la table avec les contrôles sur Hotel mais pas sur TypeChambre.