SQL : LDD et LMD

SQL ne se résume pas aux requêtes d'interrogation d'une base. Ce langage permet aussi de :
- créer des tables
- de modifier la structure de tables existantes
- de modifier le contenu des tables

et d'autres choses encore que l'on verra plus tard.

Le LDD est le langage de définition de données (DDL en anglais).
Il permet de créer des tables par l'instruction CREATE TABLE et de modifier la STRUCTURE (et non le contenu) des TABLES avec ALTER TABLE ou encore de supprimer une table avec DROP TABLE.

Le LMD est le langage de manipulation de données. Parfois on inclut le LID (langage d'interrogation de données) dans cette partie. Il permet de modifier le CONTENU d'une table en
- ajoutant de nouvelles lignes INSERT INTO nom_table
- modifiant une ou plusieurs lignes UPDATE nom_table SET
- supprimant des lignes DELETE FROM nom_table

1 Introduction au langage de définition de données

1.1 Création d'une table : CREATE TABLE

1.1.1 Syntaxe

Syntaxe simple :
CREATE TABLE nom_table
(
nom_colonne_1 type_colonne_1,
nom_colonne_2 type_colonne_2,
…
nom_colonne_n type_colonne_n
)

1.1.2 Les types de données utilisables

Les principaux types autorisés sont :

• Char(n)	Chaîne de caractères de longueur fixe = n (n=255)
• Varchar2(n)	Chaîne de caractères de longueur variable
• Number	Entier ou réel
• Number (p)	Entier (d'au plus p chiffres)
• Number(p,d)	Décimal sur p positions en tout (séparateur décimal compris), dont d chiffres décimaux
• Date	le format dépend de la configuration d'Oracle. Le plus souvent c'est jj/mm/aaaa
• Long	Chaîne longue ; restriction : par table, pas plus d'une colonne de type long

1.1.3 Exemple :

```
CREATE TABLE employe
(
empno number(8) not null,
nom varchar2(20),
fonction varchar2(10),
sal number(8,2),
code_service char(3),
date_embauche date
) ;
```

1.1.4 Les autres propriétés possibles

NOT NULL
Indique qu'on est obligé de mettre une valeur dans la colonne

UNIQUE
Indique que les valeurs dans la colonne doivent être toutes différentes (comme pour les clés, sauf que ce n'est pas une clé)

DEFAULT
Permet d'indiquer une valeur par défaut à la création d'une nouvelle ligne

Ex :

```
CREATE TABLE compte
(
num_cpte NUMBER(11) NOT NULL,
type_cpt VARCHAR2(5) DEFAULT "chèque",
solde NUMBER(10, 2) DEFAULT 0,
…
);
```

et les contraintes (clé primaire, clé étrangère, …) voir le paragraphe 3 (approfondissements)

1.2 La modification d'une table : ALTER TABLE

Ajouter une ou plusieurs colonne

ALTER TABLE nom_table
ADD (nom_colonne type_colonne);

On peut ajouter plusieurs colonnes en les séparant par des virgules dans les parenthèses du ADD

Exemples:

```
ALTER TABLE service
ADD (budget number(6) ) ;
```

Modifier le type ou tout autre propriété d'une colonne

ALTER TABLE nom_table
MODIFY (nom_colonne nouveau_type_et/ou_nouvelle_propriété);

Exemples:

```
ALTER TABLE employe
MODIFY (sal number(10,2), fonction varchar2(30) ) ;

ALTER TABLE emp
MODIFY(job null) ;
```

1.3 Renommer et supprimer une table

Supprimer une table

```
DROP TABLE nom_table;
```

```
Ex : DROP table employe ;
```

Renommer une table
RENAME TABLE nom_table TO nouveau_nom_table;

```
Ex : RENAME employe to salarie ;
```

2 Le langage de manipulation de données

2.1 Insertion de ligne INSERT INTO

Syntaxe

- Première forme

On indique explicitement le nom des colonnes qu'on veut valoriser, et on indique les valeurs dans le même ordre.

INSERT INTO nom_table (nom_colonne_1, nom_colonne_2, …)
VALUES(valeur_colonne_1, valeur_colonne_2, …)

Exemple
INSERT INTO employe(empno, nom, sal, fonction, code_service)
VALUES(12, 'Dupont', 1800, 'comptable', 2)
Remarquez qu'on est pas obligé de respecter l'ordre initial des colonnes, ni de toutes les indiquer. En effet les colonnes qui ont une valeur par défaut ou qui peuvent avoir une valeur nulle, si elles n'apparaissent pas, sont mises soit à NULL, soit à leur valeur par défaut.

- Deuxième forme : en n'indiquant pas les noms de colonnes. Mais alors il faut absolument respecter l'ordre dans lequel elles ont été créées et les valoriser toutes (ou écrire explicitement NULL)
  INSERT INTO nom_table **VALUES**(valeur_colonne_1, valeur_colonne_2, …)

Exemple :
INSERT INTO employe VALUES(12, 'Dupont', 'comptable', 1800, 2, NULL)

2.2 Suppression de ligne DELETE FROM

DELETE permet de supprimer une ou plusieurs lignes, qui vérifie(nt) une condition.

Syntaxe

```
DELETE FROM nom_table
WHERE condition(s);
```

Exemples :

Suppression de tous les comptables du service 3

```
DELETE FROM employe
WHERE code_service = 3
AND function = 'comptable';
```

Suppression de l'employé numéro 10

```
DELETE FROM employe
WHERE empno = 10;
```

2.3 Modification de ligne UPDATE…SET

Modifier la valeur d'une colonne pour une ou plusieurs lignes en particulier (qui vérifie(nt) un critère)

```
UPDATE nom_table
SET nom_colonne = valeur
WHERE condition(s);
```

Augmentation de 10% de tous les informaticiens

```
UPDATE employe
SET sal = sal * 1.1
WHERE fontion = 'informaticien';
```

Pour modifier la valeur d'une colonne de la même manière pour toutes les lignes, il suffit de ne pas indiquer de condition.
Exemple : augmenter tous les employés de 1%

```
UPDATE employe
SET sal = sal * 1.01;
```

On peut aussi modifier plusieurs colonnes en même temps, en séparant les affectations par une virgule
UPDATE … SET nom_colonneà1 = valeur_1, nom_colonne_2 = valeur_2, …

```
Update employe set deptno=1, sal=2000 where code_service=13;
```

Enfin, on peut utiliser le résultat d'une requête pour affecter de nouvelles valeurs

```
UPDATE nom_table
```

SET (colonne [, colonne]) = (Requête)
[WHERE condition]

Dans cette deuxième forme, les parenthèses (clause SET) sont inutiles si une seule colonne est citée ; le nombre de colonnes retournées par la requête doit être égal au nombre de colonnes mentionnées dans SET.
La requête ne doit renvoyer qu'une seule ligne.

Exemples :

```
Update emp
Set sal = (select 1.1*avg(sal) from emp where job='Analyst')
Where job='Analyst';

Update emp set (job, sal) = (select job, sal from emp where empno=1);
```

Exercice d'application :

Reprendre le **cas compte bancaire**

1. Créer les tables en réfléchissant au type le mieux approprié pour chaque colonne
2. Ajouter une colonne solde à la table compte
3. Affecter la valeur 0 à tous les soldes de tous les comptes existants
4. Modifier la colonne solde de la table compte pour qu'elle prenne la valeur 0 par défaut
5. Ajouter un nouveau compte 698 ouvert le 21/01/04 du client numéro 51 et de type 2
6. M. Christophe Fournier a enfin fournit son adresse : c'est le 3, avenue des peupliers à Cachan
7. La cliente numéro 53 s'est mariée : son nouveau nom est DUDUCHE
8. Ajouter un client 351, nommé Cédric Martin, domicilié 5 rue du Maine à Anthony, géré par Pignon
9. Le client nommé Robert Laroche est décédé. Il faut le supprimer de la base, ainsi que tous les enregistrements qui lui son liés dans les autres tables.
10. Supprimer de la table compte la date d'ouverture
11. Ajouter le prénom du gestionnaire

3 Approfondissement sur le LDD et le LMD

3.1 Les contraintes d'intégrité

Une contrainte d'intégrité est une règle qui permet d'assurer la validité (cohérence) des données stockées dans une base.

Le SGBD contrôle les contraintes d'intégrité à chaque modification dans les tables (saisies, modification ou suppression).

Les différentes contraintes que l'on peut définir sont :

- `non nullité (obligation) : NOT NULL`
  La colonne ne peut pas contenir de valeurs `NULL`.
- `unicité : UNIQUE`
  Toutes les valeurs de la (des) colonnes doivent être différentes ou `NULL`
- `clé primaire : PRIMARY KEY`
  Chaque ligne de la table doit avoir une valeur différente pour la ou les colonnes qui font partie de la clé primaire. Les valeurs `NULL` sont rejetées.
  primary key = unique + not null
- `valeur par défaut : DEFAULT`
  Permet de spécifier une valeur par défaut à la colonne (dans le cas où aucune valeur n'est explicitement donnée)
- `intégrité de domaine : CHECK`
  Permet de spécifier les valeurs acceptables pour une colonne.
- `intégrité référentielle (clé étrangère) : FOREIGN KEY`
  Cette colonne fait référence à une colonne clé d'une autre table.

Nous allons voir comment une contrainte d'intégrité peut se définir au moment de la création d'une table (nous verrons plus tard d'autres techniques).

Les contraintes peuvent soit concerner uniquement une seule colonne, soit concerner une table toute entière. Dans les deux cas, la syntaxe d'expression est un peu différente.

Les contraintes de colonne s'écrivent tout simplement après la colonne à laquelle elles se rapportent. Attention, on ne met pas de virgule entre la définition de la colonne et celle de la contrainte.

3.1.1 Contrainte d'obligation : NOT NULL

Cette contrainte permet de rendre obligatoire la saisie d'une colonne.

Exemple :

```
CREATE TABLE PERSONNE
(NomPers Varchar2(30) NOT NULL,
 PrénomPers Varchar2(30)
);

INSERT INTO PERSONNE VALUES ('Dupont', 'Jean');
INSERT INTO PERSONNE VALUES ('Martin', NULL);                    autorisé
~~INSERT INTO PERSONNE VALUES (NULL, 'toto');~~                  refusé
~~INSERT INTO PERSONNE(PrénomPers) VALUES ('titi');~~            refusé
```

car il faut obligatoirement que le nom ait une valeur.

3.1.2 Contrainte d'unicité : UNIQUE

Cette contrainte permet de contrôler que chaque ligne a une valeur unique pour la colonne ou l'ensemble de colonne unique (pas de doublons)

Sur une seule colonne

```
CREATE TABLE PERSONNE
(…
 Téléphone CHAR(10) UNIQUE
);
```

Sur plusieurs colonnes :

```
CREATE TABLE PERSONNE
( NomPers Varchar2(30) NOT NULL,
  PrénomPers Varchar2(30),
  UNIQUE (NomPers, PrénomPers)
);
```

Une telle contrainte indique qu'on ne pourra pas avoir deux personnes ayant le même nom et le même prénom.

3.1.3 Contrainte de clé primaire

La contrainte de clé primaire permet d'indiquer que la ou les colonnes sont uniques et ne peuvent pas avoir de valeur nulle. On peut dire que :

PRIMARY KEY = UNIQUE + NOT NULL

Si la table possède une clé composée d'une seule colonne, alors il est possible de l'indiquer juste après la colonne clé.

```
CREATE TABLE PERSONNE
(idPers PRIMARY KEY,
… );
```

Si la table a une clé compose de plusieurs colonnes, alors il est impossible d'indiquer PRIMARY KEY au niveau des colonnes.

```
CREATE TABLE PERSONNE
(NomPers Varchar2(30) PRIMARY KEY,
 PrénomPers Varchar2(30) PRIMARY KEY,
…);
```

On doit déclarer la clé à part, après la déclaration des colonnes.

```
CREATE TABLE PERSONNE
(NomPers Varchar2(30),
 PrénomPers Varchar2(30),
 PRIMARY KEY (NomPers, PrénomPe rs) );
```

Autre exemple :

```
CREATE TABLE ligne_commande
(no_commande NUMBER(5),
 code_article VARCHAR2(10),
quantite NOMBER(3),
PRIMARY KEY (no_commande,code_article)
 );
```

3.1.4 Valeur par défaut : DEFAULT

```
CREATE TABLE PERSONNE
(idPers PRIMARY KEY,
 NomPers Varchar2(30) NOT NULL,
 PrénomPers Varchar2(30) DEFAULT 'prénom inconnu',
 DateNaiss Date DEFAULT CURRENT DATE);
```

Ainsi, si on insère la personne toto de la manière suivante :

```
INSERT INTO PERSONNE (idPers, NomPers) VALUES (100, 'toto');
```

Alors les valeurs de ses champs seront :

```
100, 'toto', 'prénom inconnu', '17/03/04'
```

3.1.5 Contrainte de domaine : CHECK

La définition des types de données pour chaque colonne définit déjà un domaine pour les valeurs. On peut encore restreindre les domaines grâce à la clause CHECK.
Attention, cette contrainte est très "coûteuse" en temps.

Syntaxe : après le mot clé CHECK, on indique entre parenthèses le critère à vérifier.

Exemple:

```
CREATE TABLE personne
( idpers  Number PRIMARY KEY,
  NomPers Varchar2(30) NOT NULL,
  PrenomPers Varchar2(30) DEFAULT 'prénom inconnu',
  age Number CHECK (age >= 0  AND age < 130),
  etat_civil Varchar2(20) CHECK (etat_civil IN ('marié(e)', célibataire', 'veuf(ve)', 'divorcé(e)'),
);
```

3.1.6 Exemple récapitulatif

```
CREATE TABLE personne
( idpers  Number PRIMARY KEY,
  nom Varchar2(30) NOT NULL,
  prenom Varchar2(30) DEFAULT 'prénom inconnu',
  age Number CHECK (age >= 0  AND age < 130),
  etat_civil Varchar2(20) CHECK (etat_civil IN ('marié(e)', célibataire', 'veuf(ve)', 'divorcé(e)')
);
```

Il est possible de donner un nom à chaque contrainte (sinon le système en donne un par défaut).
Il suffit de faire précéder la contrainte par le mot-clé CONSTRAINT suivi de son nom

```
CREATE TABLE PERSONNE
( idpers  Number CONSTRAINT clé_primaire PRIMARY KEY,
  nom Varchar2(30) CONSTRAINT nom_existant NOT NULL,
  prenom Varchar2(30) CONSTRAINT prénom_par_défaut DEFAULT 'prénom inconnu',
  age Number CONSTRAINT verify_age CHECK (age >= 0  AND age < 130),
  etat_civil Varchar2(20) CONSTRAINT domaine_état_civil
        CHECK (etat_civil IN ('marié(e)', célibataire', 'veuf(ve)', 'divorcé(e)')
);
```

3.1.7 Intégrité référentielle : REFERENCES

L'intégrité référentielle permet de vérifier la cohérence des liens entre tables, lors de l'ajout, de la suppression et de la modification de lignes. Elle est utilisée lorsqu'on a une clé étrangère. Elle permet d'assurer que toute valeur de clé étrangère correspond bien à une valeur de clé primaire de la table liée.

Voyons tout d'abord la syntaxe avec un exemple:

soit le modèle relationnel suivant :
CLIENT (id_client, Nom_client, …)
FACTURE (code_fact, #id_client, …)

Voilà comment on le traduit en SQL :

```
CREATE TABLE CLIENT
(id_client Number PRIMARY KEY,
 Nom_client Varchar2(30) NOT NULL,
 … );

CREATE TABLE FACTURE
(code_fact Number PRIMARY KEY,
 id_client Number REFERENCES CLIENT(id_client)
…);
```

On indique le mot clé REFERENCES après la clé étrangère, puis le nom de la table qui est en relation suivi entre parenthèse de la clé primaire de cette table référencée.

L'application de l'intégrité référentielle implique plusieurs choses :

- on ne peut pas ajouter une facture avec un id_client qui n'existe pas dans la table client (sinon un message d'erreur apparaît). Il faut d'abord créer le client avant de créer sa facture.
- on ne peut pas supprimer un client s'il existe des factures qui lui correspondent. Il faut d'abord supprimer toutes les factures qui le concernent avant de pouvoir le supprimer
- on ne peut pas modifier l'id_client dans la table client s'il existe des factures qui le concernent.
- On peut modifier l'id_client de facture seulement si on choisit un id_client qui existe dans la table client.

Cela revient à vérifier que toutes les factures correspondent toujours à un client qui existe.

- **Options des contraintes d'intégrité référentielles**

Effacer en cascade ON DELETE CASCADE

```
CREATE TABLE FACTURE
(code_fact Number PRIMARY KEY,
 id_client Number REFERENCES CLIENT(id_client) ON DELETE CASCADE
…);
```

Si on ajoute cette option à la contrainte d'intégrité référentielle, alors si on supprime une ligne de client, toutes les factures de ce client seront supprimées.

Attention : cette contrainte s'applique à la table facture ('enfant') mais elle est vérifiée lorsqu'on modifie la table client ('parent')

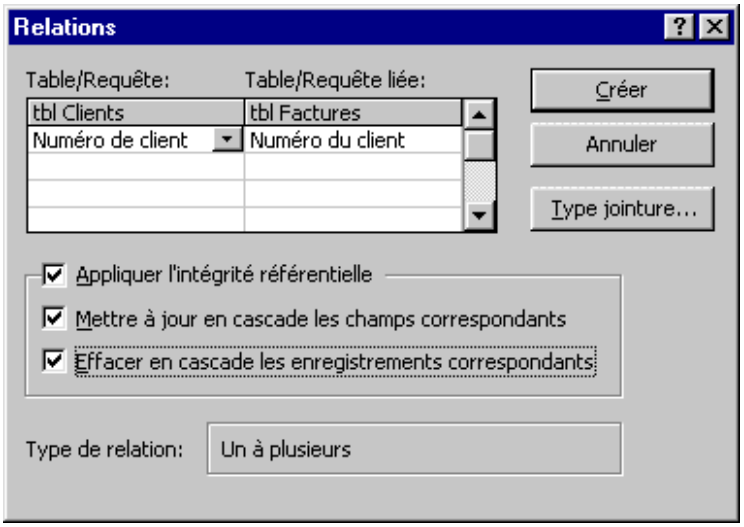
Modifier en cascade

Dans la norme SQL 2, il existe aussi l'option ON UPDATE CASCADE qui permet de répercuter sur toutes les lignes 'enfants' les modifications de valeur de la clé 'parent'.

Cette option n'est pas supportée par Oracle mais Access permet de répercuter les modifications.

Pour appliquer l'intégrité référentielle **en Access**, il faut

- aller dans la fenêtre relation,
- ajouter les tables à relier,
- faire glisser les clés étrangères sur les clés primaires
- cliquer sur le lien créé pour faire apparaître la fenêtre appelée Relations
- cocher Appliquer l'intégrité référentielle
- cocher Mettre à jour en cascade les champs correspondants ne pose aucun problème
- cocher Effacer en cascade les enregistrements correspondant peut être dangereux. A faire seulement si vous êtes certains que vous n'aurez plus besoin des lignes 'enfants' en cas de suppression du 'parent'.



- **Références à des colonnes multiples**

Il arrive qu'une clé primaire, composée de plusieurs colonnes, soit elle-même clé étrangère d'une autre table. On indique alors cette clé étrangère composée après la définition des colonnes, grâce au mot clé FOREIGN KEY suivi des colonnes clés étrangères, puis de REFERENCES, le nom de la table référencée ainsi que le nom des colonnes référencées entre parenthèses.

Exemple :
CHAMBRE(id_hotel, num_ch, tarif)
RESERVATION(id_resa, #id_hotel, #num_ch, date)

```
CREATE TABLE RESERVATION
(id_resa Number PRIMARY KEY,
 id_hotel Number,
 num_ch Number,
 date Date NOT NULL,
 FOREIGN KEY (id_hotel, num_ch) REFERENCES CHAMBRE (id_hotel, num_ch)
);
```

- **Exemple de relation où la clé primaire est composée de plusieurs clés étrangères**

```
CREATE TABLE LIGNE_COMMANDE
(id_commande Number REFERENCES COMMANDE(id_commande),
 ref_produit Varchar2(10) REFERENCES PRODUIT(ref_prod),
 quantité Number DEFAULT 1,
 PRIMARY KEY (id_commande, ref_produit)
);
```

Remarque : dans la table produit, la référence est codée par ref_prod et dans la table COMMANDE, c'est ref_produit

3.1.8 L'importance de l'intégrité référentielle

La gestion de l'intégrité référentielle est un critère primordial pour distinguer les SGBD relationnels et les autres. Un SGBD dépourvu de la gestion de l'intégrité référentielle (tel que MySQL jusqu'à la version 3.23) ne peut pas être qualifié de relationnel.
Lorsqu'on utilise de tels SGBD, il faut alors coder (programmer) les différentes vérifications à effectuer lors de la saisie, la suppression et la modification de données, ce qui est très contraignant et de plus peu sûr.
Une technique alternative aux contraintes d'intégrité, mais prise en charge par le SGBD est celle des triggers. Cette technique sera vue en deuxième année développeur.

3.1.9 Ajout ou modification d'une contrainte

Pour ajouter une contrainte à une table existante, il suffit d'utiliser la commande ALTER TABLE

Syntaxe générale :

```
ALTER TABLE nom_de_la_table
ADD type_contrainte (colonne(s));
```

Exemple1 : on ajoute la contrainte d'intégrité référentielle dans la table COMMANDE.

```
ALTER TABLE COMMANDE
ADD FOREIGN KEY (id_client) REFERENCES CLIENT(id_client);
```

Exemple 2 : on ajoute la contrainte de vérification de l'âge d'une personne

```
ALTER TABLE PERSONNE
ADD CHECK (age <= 130);
```