

Laboratoires de bases de données

Laboratoire n°1

Table et normalisation

par Danièle BAYERS et Louis SWINNEN
Révision 2016 : Vincent Reip

Ce document est disponible sous licence Creative Commons indiquant qu'il peut être reproduit, distribué et communiqué pour autant que le nom des auteurs reste présent, qu'aucune utilisation commerciale ne soit faite à partir de celui-ci et que le document ne soit ni modifié, ni transformé, ni adapté.



<http://creativecommons.org/licenses/by-nc-nd/2.0/be/>

La Haute Ecole Libre Mosane (HELMo) attache une grande importance au respect des droits d'auteur. C'est la raison pour laquelle nous invitons les auteurs dont une oeuvre aurait été, malgré tous nos efforts, reproduite sans autorisation suffisante, à contacter immédiatement le service juridique de la Haute Ecole afin de pouvoir régulariser la situation au mieux.

Octobre 2016

1. Introduction

L'objectif de ce laboratoire est de montrer, par quelques exemples pratiques, la **création**, la **modification**, la **suppression** de la structure d'une table. Ensuite, nous aborderons les contraintes qui peuvent être définies sur une table. Nous continuerons en détaillant la **modification d'un enregistrement**.

Finalement, un bref aperçu concernant **la normalisation** sera également présenté.

Nous allons commencer par quelques définitions simples afin de (re-)préciser certaines notions :

- **Base de données** : définitions
 - Grande collection de données exploitées par un ensemble d'applications
 - « *Lot d'informations stockées dans un dispositif informatique. Les technologies existantes permettent d'organiser et de structurer la base de données de manière à pouvoir facilement manipuler le contenu et stocker efficacement de très grandes quantités d'informations.* ». Source : Wikipedia
 - Système d'organisation de l'information, conçu pour une localisation et une mise à jour rapide et facile des données. Une base de données organise l'information qu'elle contient en tables, en champs (les colonnes) et en enregistrements (les lignes).
 - « *Chaque enregistrement correspond à un item stocké dans la base de données.* » Source : dicodunet
- **SGBD** (Système de gestion de base de données) : définitions
 - Intermédiaire entre les applications et la base de données offrant (entre autres) les fonctionnalités d'insertion, de modification et de suppression de données.
 - « *Un système de gestion de base de données est un ensemble de logiciels qui sert à la manipulation des bases de données. Il sert à effectuer des opérations ordinaires telles que consulter, modifier, construire, organiser, transformer, copier, sauvegarder ou restaurer des bases de données. Il est souvent utilisé par d'autres logiciels ainsi que les administrateurs ou les développeurs.* » Source : Wikipedia

AVERTISSEMENT ! Ce laboratoire aborde les éléments importants des bases de données de manière empirique. Il convient de se référer au cours théorique pour trouver des définitions plus précises des notions présentées ici.

2. La base de données

Supposons que nous souhaitons créer la table correspondant à la relation suivante :

Client
<u>NCLI</u>
Nom
Adresse
Localite
cat
compte
id: NCLI

Il faut commencer par définir **la structure** des enregistrements avant de pouvoir remplir cette table. Pour créer la structure, nous utiliserons l'instruction CREATE TABLE. Il est possible de modifier une structure créée grâce à ALTER TABLE. Enfin, supprimer une table peut se faire à l'aide de DROP.

2.1 CREATE TABLE

Format général de cette commande :

```
CREATE TABLE nom_table (  
    nom_attribut_1      type  [contraintes],  
    nom_attribut_2      type  [contraintes],  
    ...  
    [contrainte1]  
    [contrainte2]  
    ...  
);
```

Ce qu'il faut respecter :

- Pas de nom d'attributs répétés
- Le type de l'attribut est obligatoire
- Des contraintes peuvent être définies sur un attribut. Elles sont alors placées le plus souvent en regard de cet attribut. Si des contraintes concernent plusieurs attributs, elles sont regroupées à la fin de la commande CREATE TABLE.

Les différents SGBD imposent des limites quant aux nombres de caractères pour les noms de tables et d'attributs. Il convient de se reporter à la documentation du SGBD pour vérifier si ces contraintes sont bien respectées.

2.1.1 Quelques types standard définis en SQL-2

La norme SQL-2 (ou SQL-92) définit plusieurs types de données. Nous allons voir ici quelques types standards que nous utiliserons dans le cadre de ces laboratoires.

Type	Explication
INTEGER	Représente un entier. La taille de cet entier dépend du type de SGBD utilisé
NUMERIC(p,s) DECIMAL(p,s)	Définit une donnée numérique avec fraction décimale fixe. <i>p</i> définit la précision alors que <i>s</i> définit l' échelle . Sous Oracle, la précision est <i>par défaut</i> fixée à 39 ou 40 et l'échelle à 0. Sous SQL Server, la précision est fixée par défaut à 18 et l'échelle à 0.
REAL	Représente un réel. La précision de ce dernier dépend de l'implémentation.
FLOAT(p)	Représente un réel. Il est possible de définir la précision dans ce type de données.
CHAR(t)	Définit une zone de <i>t</i> caractères. Les caractères non-utilisés sont placés à blanc.
VARCHAR(t)	Définit une zone de <i>t</i> caractères maximum. Seul les caractères présents occupent de l'espace. Il n'y a donc pas de « remplissage ».
DATE	Champ pouvant contenir une date. Il faut remarquer que malheureusement, les dates en SQL-2 posent beaucoup de problèmes car les fournisseurs de SGBD proposent très souvent des implémentations diverses et parfois incompatible. Sous SQL Server, nous utiliserons le plus souvent le type

	<p>DATE (\geq v.2005) qui stocke la date sous la forme AAAA-MM-JJ ou SMALLDATETIME (\geq v. 2000) qui stocke la date et l'heure sous la forme AAAA-MM-JJ hh:mm:ss.</p> <p>Sous Oracle, le type DATE est défini et il permet de stocker la date et l'heure.</p>
--	--

Il existe encore bien d'autres types (TIME, TIMESTAMP, ...) utilisés en fonction des usages particuliers. Nous aborderons lors d'une prochaine séance les différentes fonctions qui permettent de gérer ces types de données, de réaliser des conversions, ...

Certains SGBD autorisent également les utilisateurs (i.e. les programmeurs) à spécifier des types particuliers (USER DOMAIN).

2.1.2 Les contraintes

Lors de la définition d'un attribut, en plus de spécifier un type (qui est déjà une contrainte sur le domaine des valeurs permises), il est possible de définir d'autres contraintes.

Quelques contraintes standard :

NOT NULL	<p>Indiquée directement à la suite de l'attribut, elle rend l'attribut obligatoire. Ainsi, lors d'une insertion ou modification d'un enregistrement, une valeur doit absolument être indiquée pour cet attribut.</p> <p>Sans cette contrainte, l'attribut peut, en plus de l'ensemble des valeurs autorisées par son type (i. e. contrainte de domaine), prendre la valeur particulière <i>NULL</i> qui signale l'absence de valeur pour cet attribut.</p>
PRIMARY KEY	<p>Cette contrainte permet de spécifier la clé primaire (identifiante) d'une table. En conséquence, 2 enregistrements ayant une même valeur de clé ne peuvent exister. Si la clé est constituée d'un seul attribut, la contrainte peut alors être mentionnée directement à la suite de cet attribut. Par contre, si la clé primaire est composée de plusieurs attributs, il est nécessaire de définir celle-ci de manière séparée.</p> <p>Les attributs composant la clé primaire sont obligatoires (contrainte NOT NULL).</p> <p>Il ne peut y avoir qu'une seule clé primaire par table.</p>
UNIQUE	<p>Cette contrainte permet de définir une clé secondaire ou candidate. Lorsque cette contrainte est positionnée pour un ou un ensemble d'attributs, elle exprime qu'il ne peut exister dans la table 2 enregistrements ayant même valeur pour ce(s) attribut(s).</p> <p>Si la clé candidate n'est pas composée de plusieurs attributs, elle peut être exprimée directement après la définition de celui-ci. Dans le cas contraire, la clé candidate doit faire l'objet d'une définition séparée.</p>

CHECK(expression)	Cette contrainte oblige le SGBD à vérifier que tout ajout ou modification dans une table respecte bien l'expression indiquée.
DEFAULT	Cette contrainte particulière permet d'indiquer une valeur par défaut pour un attribut. Cette contrainte permet très souvent d'éviter des valeurs <i>NULL</i> qui sont souvent particulières à gérer.
REFERENCES table (attribut)	<p>Cette contrainte indique la présence d'une clé étrangère. Une clé étrangère est un <i>lien</i> entre 2 tables (une sorte de pointeur). Les tables sont liées entre-elles simplement en important une clé (le plus souvent la clé primaire) comme attribut d'une autre table. Ainsi, comme on peut le lire dans [1] : « <i>une clé étrangère est un attribut ou un groupe d'attributs dans une table T1, dont les valeurs doivent exister comme valeurs de la clé candidate dans une table T2. T1 et T2 ne sont pas nécessairement distinctes.</i> »</p> <p>En résumant, on peut dire qu'une clé étrangère est un attribut ou un groupe d'attributs qui est (sont) clé candidate dans une autre table (la clé candidate peut être la clé primaire).</p> <p>Il faut également remarquer que lors de l'insertion d'un enregistrement, il est primordial que la valeur mentionnée dans cet attribut existe dans la table qui lui est liée.</p> <p>Si la clé étrangère est constituée d'un seul attribut, la contrainte peut être mentionnée juste après celui-ci. Dans le cas contraire, il est nécessaire de l'exprimer séparément.</p> <p>Cette contrainte très importante est nommée contrainte d'intégrité référentielle.</p>

2.1.3 Exemples

```
CREATE TABLE client (
  ncli CHAR(4) PRIMARY KEY,
  nom VARCHAR(20) NOT NULL,
  adresse VARCHAR(30),
  localite VARCHAR(20),
  cat CHAR(2),
  compte DECIMAL(9,2) ) ;
```

Dans cet exemple, on crée une table *client* dont l'attribut *ncli* est la clé primaire. Son type est une suite de 4 caractères. Cet attribut est obligatoire. Le *nom* est le second attribut, il est également obligatoire et son type est une suite variable de caractères de taille maximale égale à 20. L'*adresse*, la *localité*, la *catégorie* et le *compte* sont des attributs facultatifs. En particulier, le *compte* est une donnée numérique de 9 chiffres au total dont 2 chiffres se trouvent après la virgule.

Variante :

```
CREATE TABLE client (  
    ncli CHAR(4),  
    nom VARCHAR(20) NOT NULL,  
    adresse VARCHAR(30),  
    localite VARCHAR(20),  
    cat CHAR(2),  
    compte DECIMAL(9,2)  
    PRIMARY KEY(ncli)  
) ;
```

Cet exemple montre que la contrainte *primary key* peut être placée à la fin de la création de la table. Il faut noter que si la clé primaire est composée de plusieurs attributs, cette forme est obligatoire.

On peut également exprimer quelques contraintes particulières :

```
CREATE TABLE client (  
    ncli CHAR(4),  
    nom VARCHAR(20) NOT NULL,  
    adresse VARCHAR(30),  
    localite VARCHAR(20),  
    cat CHAR(2),  
    compte DECIMAL(9,2),  
    PRIMARY KEY(ncli),  
    CHECK(cat IN('C1','C2','C3','C4')),  
    CONSTRAINT compte_positif CHECK(compte >= 0)  
) ;
```

La première contrainte exprime que l'attribut *catégorie* doit être compris parmi les valeurs *C1*, *C2*, *C3* ou *C4*. Le mot clé *IN* sera détaillé ultérieurement lors de l'étude des requêtes. La seconde contrainte exprime le fait que la valeur du compte d'un client ne peut être négative. Cette seconde contrainte sera associée à un nom (*compte_positif*) qui apparaîtra dans le message d'erreur généré par le SGBD lorsque la contrainte est violée.

2.2 ALTER TABLE

Cette commande permet de modifier la structure d'une table. Modifier la structure d'une table n'est pas nécessairement une opération anodine. Par exemple, lors de l'ajout d'un attribut dans une table, que faire avec les enregistrements existants ?

Très souvent, la modification de la structure concerne soit une table vide, soit introduit des astuces pour les enregistrements existants. Par exemple, l'ajout d'un champ avec mention d'une clause *DEFAULT* ou encore l'ajout d'un champ pouvant prendre la valeur *NULL*.

Il faut, par contre, faire très attention à la suppression d'un attribut dans une table car cela peut conduire à des problèmes d'intégrité (suppression d'une clé candidate utilisée comme clé étrangère, ...).

Opérations possibles :

- **ADD** qui permet d'ajouter un attribut, une contrainte, ...
- **MODIFY** qui permet de modifier le type d'un attribut ou changer sa précision
- **DROP** qui permet de supprimer un attribut, une contrainte, ...

Format :

```
ALTER TABLE nom_table  
  [ADD|MODIFY] attribut type [contraintes]  
  [DROP COLUMN|CONSTRAINT] nom ;
```

2.2.1 Exemples

```
ALTER TABLE client  
  ADD reg_nat DECIMAL(11) NOT NULL UNIQUE ;
```

Dans cet exemple, nous ajoutons à la table *client* l'attribut *reg_nat* devant contenir le numéro d'inscription au registre national. Cet attribut est une clé candidate et doit être définie comme unique. De par ce fait, il serait impossible que 2 clients différents disposent du même numéro d'inscription au registre national.

2.3 DROP TABLE

Cette commande permet de modifier sensiblement la structure de la base de données en supprimant une table. Elle doit toujours être utilisée avec la plus grande précaution car les éventuelles données contenues dans la table seront perdues.

```
DROP TABLE nom_table ;
```

2.3.1 Exemple

```
DROP TABLE client ;
```

Dans cet exemple, la table *client* est supprimée par le SGBD.

2.4 INSERT INTO

La commande INSERT INTO permet d'ordonner l'insertion d'un enregistrement dans la table. Il est nécessaire de fournir une valeur pour tous les attributs obligatoires (déclaré NOT NULL) ne disposant pas de valeur par défaut (option DEFAULT).

La forme de la commande est la suivante :

```
INSERT INTO nom_table (attr1, attr2, ..., attrn)  
  VALUES (val1, val2, ..., valn)
```

Cette commande provoque la création d'un enregistrement dans la table en fixant respectivement, pour les attributs attr₁, attr₂, ..., attr_n les valeurs val₁, val₂, ..., val_n.

Il faut remarquer que la liste d'attributs n'est obligatoire que si la commande ne fixe pas une valeur pour tous les attributs de la table. En effet, si des attributs sont facultatifs ou acceptent une valeur par défaut, il faut lister les attributs pour lesquels une valeur est fournie.

2.4.1 Exemples

```
INSERT INTO client VALUES ('C001', 'Tintin', 'Rue du château 3',  
'Moulinsart', 'C1',0) ;
```

Dans cet exemple, nous ajoutons dans la table client l'enregistrement identifié comme C001 reprenant l'ensemble des valeurs. Remarquons que, comme nous donnons une valeur à chaque attribut, il n'est pas nécessaire de lister les champs de la table.

```
INSERT INTO client(nclic,nom) VALUES ('C002', 'Dupont');
```

Voici un exemple dans lequel une valeur n'est pas donnée pour chaque champ. Il est donc nécessaire de lister les attributs qui seront initialisés. Dans cet exemple et au vu de la définition de la table, les autres champs ont, comme valeur, *NULL*.

2.5 UPDATE

La commande UPDATE permet de modifier des enregistrements. Grâce à cette commande, la valeur d'un attribut ou plusieurs attributs peut être mise à jour. Il faut toujours être vigilant à bien identifier le ou les enregistrements à mettre à jour.

Format de la commande :

```
UPDATE nom_table
  SET nom_attribut1 = valeur1[, ..., nom_attributn = valeurn]
WHERE attribut = valeur
```

Cette commande mettra à jour la table mentionnée en remplaçant les valeurs des attributs listés par les nouvelles valeurs mentionnées en paramètres pour tous les enregistrements répondant à la condition énoncée dans le WHERE.

Nous verrons plus loin dans les séances de laboratoires qu'il est possible de construire des requêtes retournant des valeurs. Il faut savoir que de telles requêtes peuvent être utilisées comme « valeur » dans l'argument SET. Ce champ peut également mentionner une expression (arithmétique par exemple si la donnée le permet, ...).

2.5.1 Exemples

```
UPDATE client
  SET cat = 'C2', adresse='rue de la police, 4'
WHERE nom = 'Dupont';
```

Dans cet exemple, tous les enregistrements pour lesquels le nom est *Dupont* sont modifiés. Dans cette modification, la *catégorie* est fixée à la valeur *C2* et l'attribut *adresse* est initialisé.

2.6 DELETE

La commande DELETE permet d'ordonner la suppression d'un, de plusieurs ou même de la totalité des enregistrements présents dans une table. Il faut toujours identifier très clairement les enregistrements à supprimer et utiliser cette commande avec une extrême précaution.

Format de la commande :

```
DELETE FROM nom_table
WHERE attribut = valeur
```

Cette commande ordonne dans la table mentionnée la suppression de tous les enregistrements répondant à la condition exprimée dans le WHERE.

Comme mentionné ci-dessus, il est également possible d'utiliser dans le champ « valeur » (énoncé dans le WHERE de la commande) une requête plutôt qu'une valeur précise.

Attention ! Ecrire une requête DELETE FROM sans la clause WHERE revient à vider la table de son contenu.

3. Normalisation

Le modèle entité-association est un modèle riche et conceptuel. Cela signifie qu'il est principalement utilisé lors de la phase d'analyse pour l'élaboration et la réflexion sur la modélisation des données.

Une fois le modèle conceptuel terminé, il faut choisir une technologie adaptée en fonction de l'environnement technologique et de programmation. Ainsi, il est possible de se diriger vers :

- une base de données (relationnelle, objet ou autre)
- des fichiers (séquentiels, indexés ou à accès calculé)
- ...

Le choix de la technologie est important car des contraintes particulières propres à cette technologie doivent être vérifiées. Ainsi, lorsqu'on se dirige vers les bases de données relationnelles, il faut respecter les contraintes du modèle relationnel des données.

3.1 Le modèle relationnel des données

Le modèle relationnel des données est une implémentation du modèle conceptuel (entité-association par exemple) qui respecte les contraintes propre à la technologie des systèmes de gestion de base de données relationnels (ou encore SGBDR).

3.1.1 Vocabulaire

Une **table** ou **relation** est un ensemble de données relatives à un même sujet (définition tirée de [1]).

Un **déterminant** est un attribut ou un ensemble d'attributs qui détermine la valeur d'un (groupe d') attribut(s).

Une **dépendance fonctionnelle** existe entre les attributs *déterminants* et *le(s) attribut(s) déterminé(s)*. En effet, la valeur de(s) l'attribut(s) déterminé(s) dépend(ent) fonctionnellement de la valeur de(s) l'attribut(s) déterminant(s). On peut également dire que la valeur des attributs déterminés sont fonctions de la valeur du (des) déterminants.

L'**identifiant** d'une relation est un attribut ou un groupe d'attributs déterminant la valeur de tous les autres attributs de cette relation. On peut donc en déduire qu'un identifiant est le déterminant de tous les attributs de cette relation. L'identifiant d'une relation est également appelé **clé primaire** de la table ou relation.

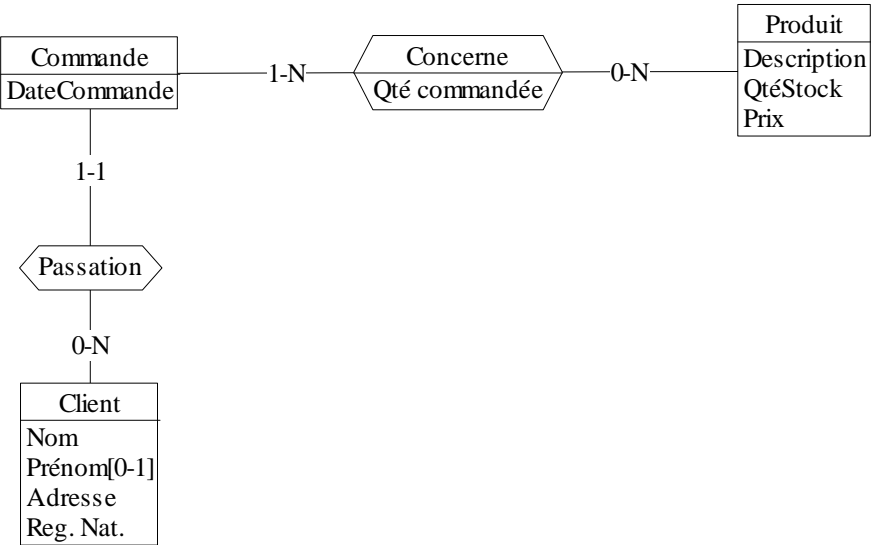
Une **clé candidate** dans une relation est un identifiant possible pour cette relation. Par définition, les clés candidates d'une relation déterminent tous les attributs de cette relation.

3.1.2 Passage du modèle entité-association vers le modèle relationnel

Pour les besoins de ce laboratoire, nous pourrions résumer les règles de transformation comme suit (le cours d'analyse illustrera ces concepts de manière plus précise) :

- Les *entités* dans le modèle conceptuel deviennent des *relations* ou *tables* dans le modèle relationnel.
- Les *attributs* présents dans les entités deviennent des *propriétés* ou *colonnes* dans les tables du modèle relationnel.
- Les *associations* présentes dans le modèle relationnel peuvent, **suivant les cardinalités** se transformer en *table* ou disparaître.

Afin d’illustrer ces différents points, voici un extrait d’un schéma conceptuel de données :



EXEMPLE 1 : Schéma conceptuel commande-produit

À partir de ce schéma conceptuel, nous allons le traduire en schéma relationnel. Pour ce faire, nous allons commencer par transformer toutes les entités en table. Les attributs sont simplement recopiés dans le schéma relationnel (comme mentionné dans le tableau 1).

	<table><tr><td>A1</td></tr><tr><td>AT1</td></tr><tr><td>AT2</td></tr><tr><td>AT3[0-1]</td></tr></table>	A1	AT1	AT2	AT3[0-1]	<table><tr><td>A1</td></tr><tr><td>AT1</td></tr><tr><td>AT2</td></tr><tr><td>AT3[0-1]</td></tr></table>	A1	AT1	AT2	AT3[0-1]
A1										
AT1										
AT2										
AT3[0-1]										
A1										
AT1										
AT2										
AT3[0-1]										

TABEAU 1 : Types d’entités et attributs (extrait de [3])

En ce qui concerne les associations, il faut distinguer les cas suivants qui prennent en compte les cardinalités maximales de chaque côtés de l’association :

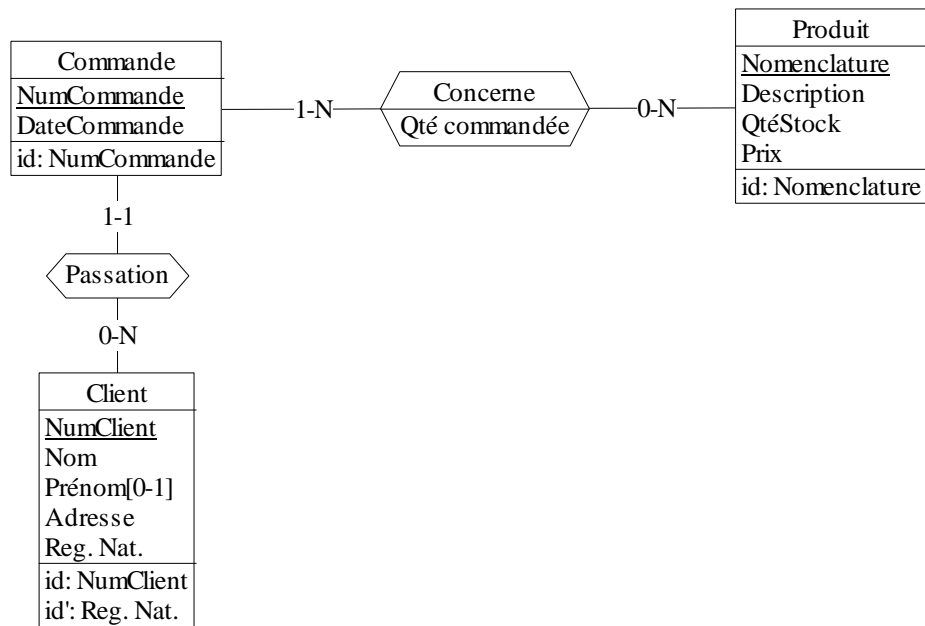
1:N	<table><tr><td>A2</td></tr><tr><td><u>IA</u></td></tr><tr><td>id: IA</td></tr></table> <div>0-N</div> <table><tr><td>R2</td></tr></table> <div>1-1</div> <table><tr><td>B2</td></tr></table>	A2	<u>IA</u>	id: IA	R2	B2	<table><tr><td>A2</td></tr><tr><td><u>IA</u></td></tr><tr><td>id: IA</td></tr><tr><td>acc</td></tr></table> <div>←</div> <table><tr><td>B2</td></tr><tr><td>IA</td></tr><tr><td>ref: IA</td></tr><tr><td>acc</td></tr></table>	A2	<u>IA</u>	id: IA	acc	B2	IA	ref: IA	acc										
A2																									
<u>IA</u>																									
id: IA																									
R2																									
B2																									
A2																									
<u>IA</u>																									
id: IA																									
acc																									
B2																									
IA																									
ref: IA																									
acc																									
1:1	<table><tr><td>A3</td></tr><tr><td><u>IA</u></td></tr><tr><td>id: IA</td></tr></table> <div>0-1</div> <table><tr><td>R3</td></tr></table> <div>1-1</div> <table><tr><td>B3</td></tr></table>	A3	<u>IA</u>	id: IA	R3	B3	<table><tr><td>A3</td></tr><tr><td><u>IA</u></td></tr><tr><td>id: IA</td></tr><tr><td>acc</td></tr></table> <div>←</div> <table><tr><td>B3</td></tr><tr><td><u>IA</u></td></tr><tr><td>id: IA</td></tr><tr><td>ref acc</td></tr></table>	A3	<u>IA</u>	id: IA	acc	B3	<u>IA</u>	id: IA	ref acc										
A3																									
<u>IA</u>																									
id: IA																									
R3																									
B3																									
A3																									
<u>IA</u>																									
id: IA																									
acc																									
B3																									
<u>IA</u>																									
id: IA																									
ref acc																									
N:N	<table><tr><td>A4</td></tr><tr><td><u>IA</u></td></tr><tr><td>id: IA</td></tr></table> <div>0-N</div> <table><tr><td>R4</td></tr></table> <div>0-N</div> <table><tr><td>B4</td></tr><tr><td><u>IB</u></td></tr><tr><td>id: IB</td></tr></table>	A4	<u>IA</u>	id: IA	R4	B4	<u>IB</u>	id: IB	<table><tr><td>A4</td></tr><tr><td><u>IA</u></td></tr><tr><td>id: IA</td></tr><tr><td>acc</td></tr></table> <div>←</div> <table><tr><td>R4</td></tr><tr><td><u>IA</u></td></tr><tr><td><u>IB</u></td></tr><tr><td>id: IB</td></tr><tr><td>acc</td></tr><tr><td>ref: IB</td></tr><tr><td>ref: IA</td></tr><tr><td>acc</td></tr></table> <div>→</div> <table><tr><td>B4</td></tr><tr><td><u>IB</u></td></tr><tr><td>id: IB</td></tr><tr><td>acc</td></tr></table>	A4	<u>IA</u>	id: IA	acc	R4	<u>IA</u>	<u>IB</u>	id: IB	acc	ref: IB	ref: IA	acc	B4	<u>IB</u>	id: IB	acc
A4																									
<u>IA</u>																									
id: IA																									
R4																									
B4																									
<u>IB</u>																									
id: IB																									
A4																									
<u>IA</u>																									
id: IA																									
acc																									
R4																									
<u>IA</u>																									
<u>IB</u>																									
id: IB																									
acc																									
ref: IB																									
ref: IA																									
acc																									
B4																									
<u>IB</u>																									
id: IB																									
acc																									

Identifiant composé	<table><tr><td>A5</td></tr><tr><td><u>IA1</u></td></tr><tr><td><u>IA2</u></td></tr><tr><td>id: IA1</td></tr><tr><td>IA2</td></tr></table> <div>0-N</div> <div>R5</div> <div>1-1</div> <table><tr><td>B5</td></tr></table>	A5	<u>IA1</u>	<u>IA2</u>	id: IA1	IA2	B5	<table><tr><td>A5</td></tr><tr><td><u>IA1</u></td></tr><tr><td><u>IA2</u></td></tr><tr><td>id: IA1</td></tr><tr><td>IA2</td></tr><tr><td>acc</td></tr></table> <div>←</div> <table><tr><td>B5</td></tr><tr><td>IA1</td></tr><tr><td>IA2</td></tr><tr><td>ref: IA1</td></tr><tr><td>IA2</td></tr><tr><td>acc</td></tr></table>	A5	<u>IA1</u>	<u>IA2</u>	id: IA1	IA2	acc	B5	IA1	IA2	ref: IA1	IA2	acc
A5																				
<u>IA1</u>																				
<u>IA2</u>																				
id: IA1																				
IA2																				
B5																				
A5																				
<u>IA1</u>																				
<u>IA2</u>																				
id: IA1																				
IA2																				
acc																				
B5																				
IA1																				
IA2																				
ref: IA1																				
IA2																				
acc																				

TABLEAU 2 : Types d'associations (extrait de [3])

Etape 1

Nous allons commencer par ajouter (ou calculer) les identifiants pour toutes les entités. Nous obtenons, dès lors, une nouvelle version de notre schéma :

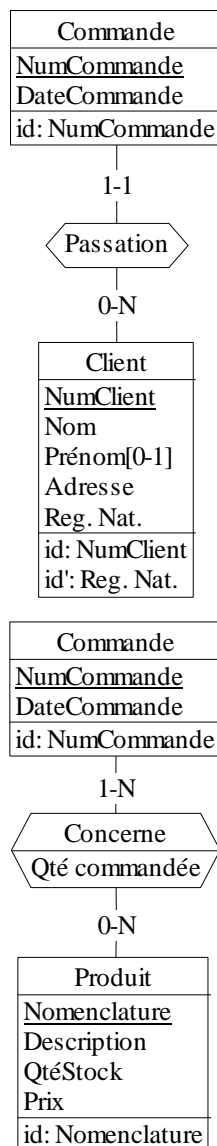


ETAPE 1 : ajout des contraintes d'identifiant

Toutes les entités disposent au moins d'un identifiant. Dans la plupart des cas, l'identifiant est créé et maintenu par le programmeur. On utilise rarement des identifiants extérieurs dans l'exploitation des données. Cependant, ces identifiants extérieurs (comme le numéro d'inscription au registre national dans la table Client) sont néanmoins mémorisés et peuvent servir de clé candidate (mention id').

Etape 2

Nous allons maintenant transformer toutes les entités en table, recopier tous les attributs et transformer toutes les associations.



Le client passe une commande. Il faut se souvenir de quel client passe la commande. Etant donné qu'une commande *est passée par un et un seul client* (en fonction des cardinalités), nous savons en regardant le tableau 2 que :

- L'association va disparaître
- L'identifiant du client (NumClient) va être *importé* dans la commande afin de retenir quel client a passé la commande. L'attribut importé est appelé *clé étrangère*.

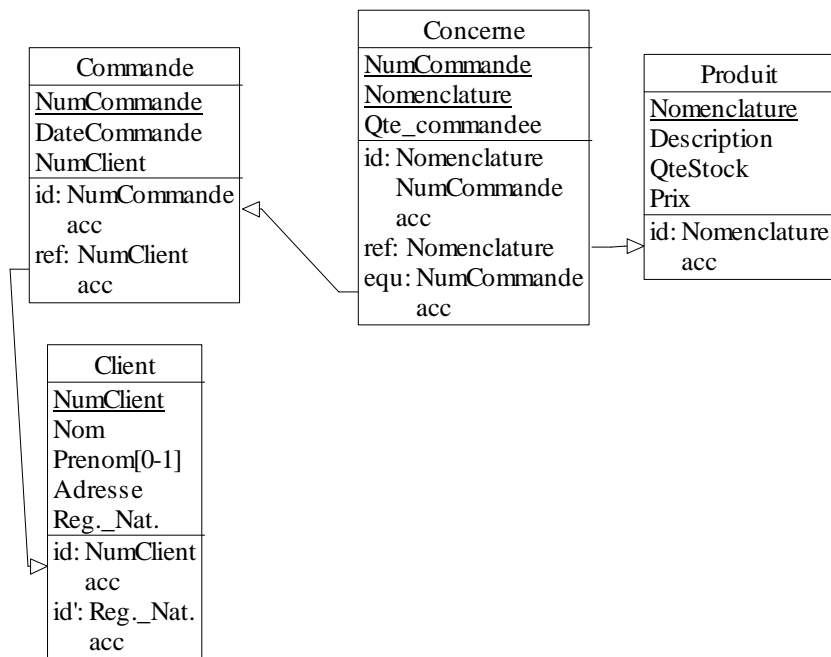
Après réflexion, ces modifications sont logiques puisque, pour une commande donnée, elle ne concerne qu'un client, il est naturel d'importer le client dans la commande.

Une commande concerne plusieurs produits (au moins 1). Chaque produit peut être référencé (0-N) dans plusieurs commandes. En observant le tableau 2, nous pouvons déduire que :

- L'association *Concerne* doit être transformée en table
- L'identifiant de cette nouvelle table sera composé des identifiants des deux tables référencées. Ainsi, chaque ligne sera identifiée par *la paire* (NumCommande, Nomenclature). Les deux attributs *importés* sont des clés étrangères vers les tables concernées.

Ici aussi, les modifications sont assez logiques car il faut pouvoir retenir que dans la commande X, le produit Y a été commandé en quantité Z.

Nous obtenons ainsi le schéma relationnel suivant :



ETAPE 2 : Le schéma relationnel

3.1.3 Règles de normalisation du schéma relationnel

Dans le schéma relationnel, il est nécessaire de vérifier au moins 4 règles fondamentales appelées *formes normales*. Ces règles ont pour but de transformer le schéma de sorte à minimiser les redondances et faciliter l'exploitation.

En effet, les redondances constituent un véritable problème. Dès qu'une information se trouve à plusieurs endroits, il est difficile de faire face à toutes ces différentes versions de la même information. Il convient, lors d'une mise à jour, de bien s'assurer que ces différentes versions sont modifiées. Or, si ces synchronisations ne sont pas automatiques, le risque d'avoir des incohérences est important.

Une relation est dite en **1^{ère} forme normale** si elle possède au moins une clé candidate qui joue le rôle d'identifiant (appelé également clé primaire) et que tous les attributs sont atomiques : c'est à dire ni multivalués (pas de « tableaux »), ni composés (pas de « structures »).

Illustration théorique :

Non normalisé

1FN
ATT1
ATT2[1-3]
ATT3
ATT3.1
ATT3.2

Normalisé

1FN
<u>ID</u>
ATT1
ATT2.1
ATT2.2[0-1]
ATT2.3[0-1]
ATT3.1
ATT3.2
id: ID
acc

Atteindre la 1^{ère} forme normale est assez simple. Dans l'illustration théorique précédente, nous avons :

- ajouté un identifiant ID (clé primaire),
- *ajouté* le tableau, comme celui-ci doit au-moins contenir un élément, l'attribut *ATT2.1* est obligatoire tandis que les deux suivants ATT2.2 et ATT2.3 sont facultatifs (ce qui correspond au schéma initial).
- *supprimé* la structure ATT3 qui contenait les deux éléments ATT3.1 et ATT3.2.

Nous pourrions appliquer cette règle à la relation suivante :

Non normalisé

Etudiant
Nom
Prénom[1-3]
Parents
Père
Mère

Normalisé

Etudiant
<u>Matricule</u>
Nom
Prénom1
Prénom2[0-1]
Prénom3[0-1]
Pere
Mere
id: Matricule
acc

La relation de droite est en 1^{ère} forme normale.

Une relation est dite en **2^{ème} forme normale** si elle est en 1^{ère} forme normale et que tous les attributs dépendent fonctionnellement de l'ensemble de la clé primaire. En corrolaire, une relation en 1^{ère} forme normale dont la clé primaire est élémentaire est en 2^{ème} forme normale également.

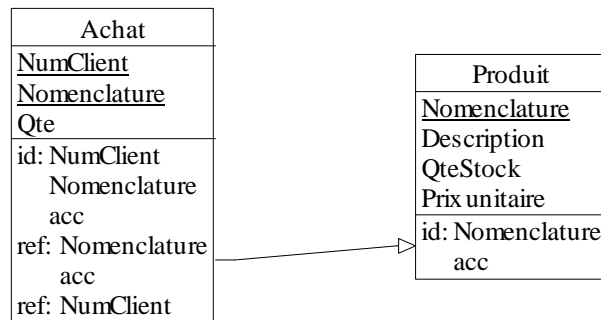
Illustrons cette forme normale sur base de l'exemple suivant (tiré de [3]) :

Achat
<u>NumClient</u>
<u>Nomenclature</u>
Qté
Prix unitaire
id: NumClient
Nomenclature

Nomenclature -> Prix unitaire

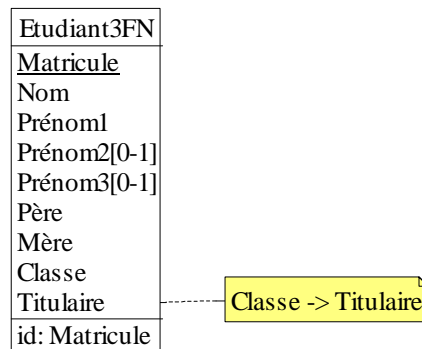
Dans cet exemple, la relation *Achat* n'est pas en 2^{ème} forme normale car la donnée *Prix unitaire* dépend uniquement de la nomenclature du produit (dit autrement, dès que j'ai l'identifiant du produit, je connais son prix unitaire).

Pour corriger ce problème, il convient de déplacer le prix unitaire dans la table Produit et de garder le lien vers cette table grâce à la nomenclature, qui fait également office de clé étrangère (lien vers une autre table) :

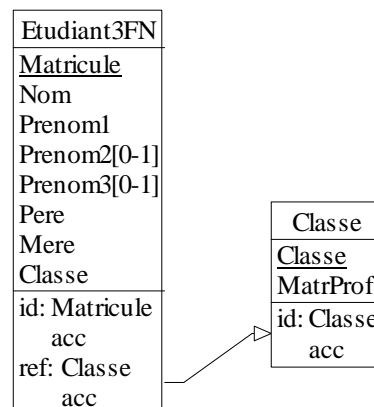


Une relation est dite en **3^{ème} forme normale** si elle est en 2^{ème} forme normale et qu'il n'existe pas de dépendance transitive vers la clé primaire (c'est-à-dire qu'un attribut ne dépend pas fonctionnellement d'un autre, qui lui-même dépend fonctionnellement de la clé primaire).

Illustrons cette forme normale sur base de l'exemple suivant :

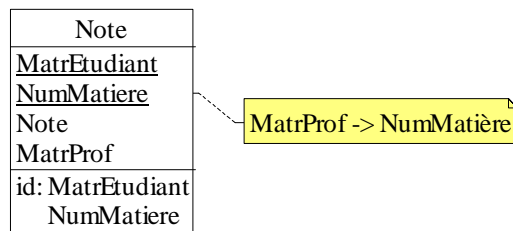


Dans cette relation, l'attribut *Titulaire* dépend fonctionnellement de l'attribut *Classe* (dit autrement, dès que l'on connaît la classe d'un étudiant, on connaît le titulaire). Pour corriger ce problème, il faut déplacer le titulaire dans une autre relation, la *Classe* qui reprendra cette dépendance.



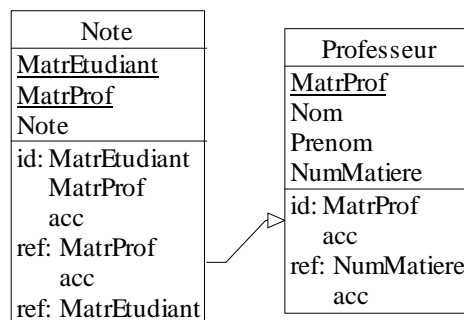
Une relation est dite en **forme normale de Boyce-Codd** si chaque déterminant est clé candidate (définition reprise de [2]).

Illustrons cette forme normale sur base de l'exemple suivant (tiré de [3]) :



Supposons que les notations d'un cours soient illustrées comme suit : pour *un étudiant* dans *une matière* donnée par *un professeur*, la *note* est identifiée (si je connais l'étudiant et la matière, alors je trouve la note et le professeur qui l'a évalué). Supposons également qu'un professeur donne cours dans une seule matière (hypothèse fortement restrictive et nécessaire pour l'illustration).

En conséquence, nous avons, en plus, la dépendance suivante du *professeur* vers *la matière* (si je connais le *professeur*, je trouve *la matière* qu'il enseigne), nous avons la relation *Note* qui **n'est pas en forme normale de Boyce-Codd** parce que le déterminant *MatrProf* n'est pas clé candidate de la relation (si je connais le professeur, je ne peux pas trouver l'étudiant, la matière et la note).



4. Exemple

4.1 Normalisation

En partant du schéma relationnel obtenu à l'étape 2 ci-avant, nous pouvons remarquer que celui-ci est :

- en 1^{ère} forme normale,
- en 2^{ème} forme normale,
- en 3^{ème} forme normale
- en forme normale de Boyce-Codd

Pour la 1^{ère}, la 2^{ème} et la 3^{ème} forme normale, c'est assez évident.

En effet, toutes les relations ont une clé primaire et les attributs sont tous atomiques (→ 1FN).

Les tables *Commande*, *Client* et *Produit* ont une clé élémentaire, elles sont de facto en 2FN. La relation *Concerne* a effectivement une clé composée mais les attributs dépendent de l'ensemble de la clé primaire (→ 2FN).

Enfin, il n'y a pas de dépendance transitive vers la clé primaire (tous les attributs dépendent directement de la clé primaire) → 3FN.

Pour la forme normale de Boyce-Codd, il y a un point à expliciter. En effet, dans la table client, il y a l'attribut RegNat. Il y a des dépendances fonctionnelles particulières :

- RegNat → Adresse
- RegNat → Prenom
- RegNat → Nom
- RegNat → NumClient

Ceci dit, RegNat est donc une clé candidate de la relation client. Dès lors, nous sommes bien également en forme normale de Boyce-Codd puisque tous les déterminants (NumClient et RegNat) sont clés candidates.

4.2 Code de création des tables

Voici le script de création de la base de données client-commande en guise d'exemple.

```
CREATE TABLE client (  
    NumClient CHAR(5) PRIMARY KEY,  
    Nom VARCHAR(50) NOT NULL,  
    Prenom VARCHAR(50),  
    Adresse VARCHAR(200) NOT NULL,  
    RegNat VARCHAR(15) NOT NULL  
);  
  
CREATE TABLE produit (  
    Nomenclature CHAR(5) PRIMARY KEY,  
    Description VARCHAR(50) NOT NULL,  
    QteStock INTEGER NOT NULL,  
    Prix DECIMAL(7,2)  
);  
  
CREATE TABLE commande (  
    NumCommande CHAR(5) PRIMARY KEY,  
    DateCommande DATE NOT NULL,  
    NumClient CHAR(5) NOT NULL REFERENCES Client(NumClient)  
);  
  
CREATE TABLE concerne (  
    NumCommande CHAR(5) REFERENCES commande,  
    Nomenclature CHAR(5) REFERENCES produit,  
    QteCommandee INTEGER NOT NULL,  
    PRIMARY KEY(NumCommande, Nomenclature)  
);
```


5. Accès à Oracle

Dans le cadre de ce laboratoire, nous utiliserons 2 SGBD : *Microsoft SQL Server et Oracle*. *Oracle* est installé sur le serveur *Oracle11* dont l'adresse IP est 192.168.128.17. Il s'agit d'un serveur Linux qui est accessible aux étudiants pour réaliser leurs laboratoires.

La version d'Oracle installée est la version gratuite Oracle Express Edition 11g. Cette version est disponible pour les postes Windows et Linux. L'interaction avec Oracle peut se faire au moyen du client Oracle SQL Developer. Ce logiciel est installé sur les machines des laboratoires et vous pouvez le télécharger gratuitement sur le site de Oracle si vous désirez l'installer sur votre ordinateur personnel.

5.1 Etablir la connexion et exécuter des requêtes

Pour accéder à la base de données, vous devrez tout d'abord configurer la connexion. La capture d'écran ci-dessous vous montre la marche à suivre :

- Cliquez sur le sigle  en haut à gauche
- Complétez le formulaire avec les données de connexion et cliquez sur le bouton Enregistrer

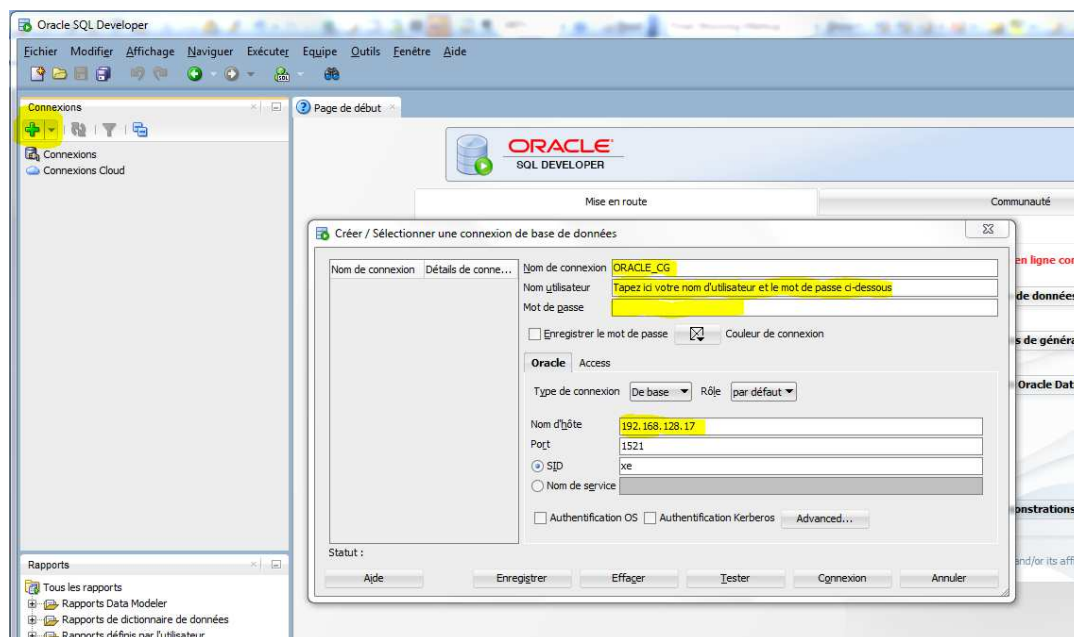



Figure 1 : configuration de la connexion à la BD

Une fois cette manipulation effectuée, vous verrez apparaître votre connexion dans la colonne de gauche. Un 'click-droit' sur celle-ci vous permet de vous connecter (en précisant votre mot de passe).

Vous verrez alors apparaître toute une série d'item dans la colonne de gauche et une nouvelle 'Feuille de calcul SQL' apparaîtra.

Vous pouvez alors entrer des commandes SQL de manière interactive et les exécuter en cliquant sur le bouton 'Exécuter l'instruction'  (raccourci clavier : CTRL – Enter).

Le résultat de votre requête apparaîtra dans la partie basse de votre fenêtre.

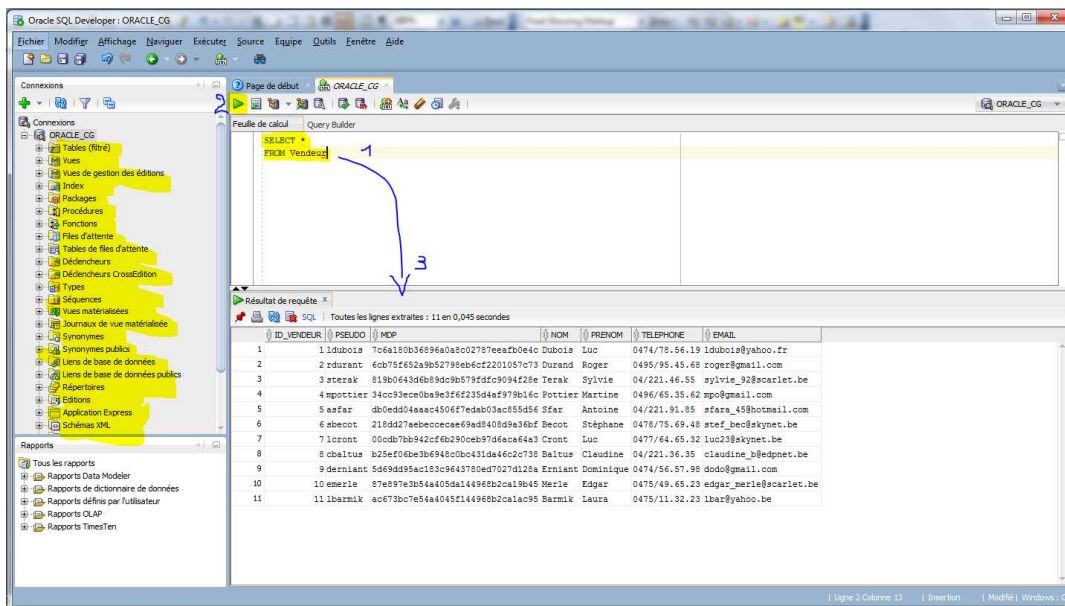


Figure 2 : Exécution d'une requête SQL

5.2 Les scripts

Il est aussi possible de d'exécuter des scripts SQL (un script est un ensemble de requêtes SQL séparées par un point-virgule). Cette option est très utile pour les scripts de création des bases de données qui consistent en une liste d'instructions CREATE TABLE... Pour exécuter un tel script, il faut écrire le script dans la 'Feuille de calcul SQL' (ou faire un copier/coller à partir d'un fichier, ou encore ouvrir le fichier contenant le script via le menu Fichier>Ouvrir) et ensuite cliquer sur le bouton 'Exécuter un script' (raccourci clavier : F5).

Après exécution, il faut impérativement vérifier le résultat qui est visible dans la partie basse de la fenêtre ('Sortie de script'). Si des erreurs se sont produites, il faut analyser le message d'erreur et, le cas échéant, faire des corrections au niveau du script.

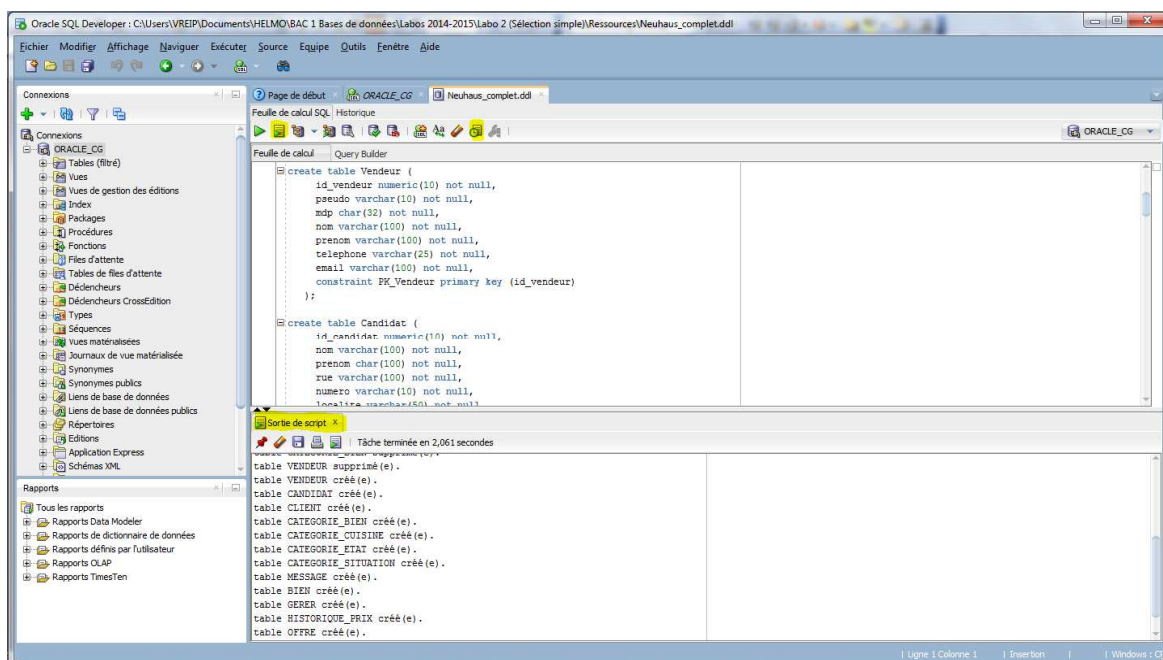



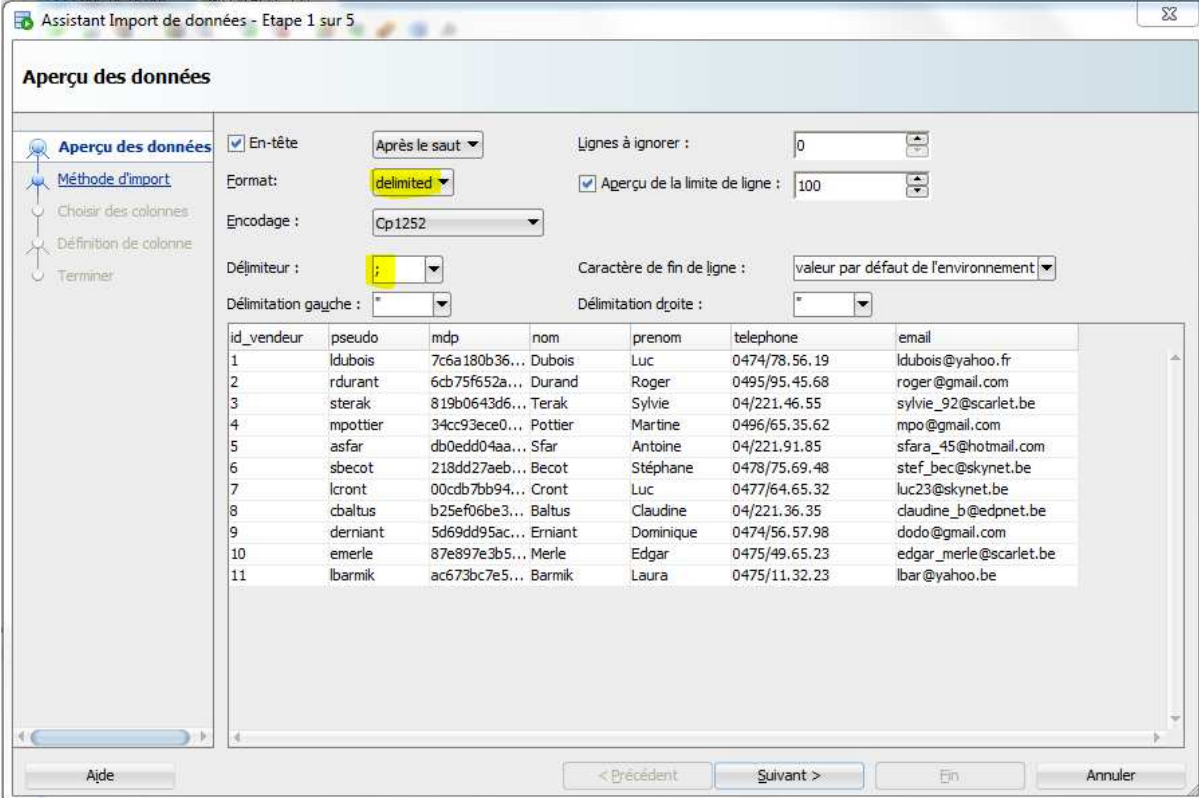
Figure 3 : Exécution d'un script

Par ailleurs, l'historique des requêtes et scripts exécutés est disponible en cliquant sur le bouton 'Historique SQL'  (raccourci clavier : F8).

5.3 Importer des données

Il est fréquent de devoir importer des données dans des tables à partir de fichiers (CSV, XML, ...). Pour importer un fichier de données dans une table, il faut faire un 'clic-droit' sur le nom de cette table (colonne de gauche) ; un menu contextuel apparaît alors et propose une option « Importer des données... ». Vous devez ensuite sélectionner le fichier de données sur votre disque dur. Il faudra alors préciser toute une série de paramètres indiquant la nature du fichier à importer. Les captures d'écran ci-dessous vous montrent la marche à suivre pour des fichiers dont les données sont séparées par de point-virgule (;) comme ceux qui vous seront fournis pour les laboratoires BD.

ETAPE 1 : sélectionnez le format 'Delimited' et précisez que les données sont séparées par des points-virgules. Les données doivent alors être présentées en colonnes.



Aperçu des données

☒ En-tête Après le saut Lignes à ignorer : 0

Format: delimited ☒ Aperçu de la limite de ligne : 100

Encodage : Cp1252

Délimiteur : ; Caractère de fin de ligne : valeur par défaut de l'environnement

Délimitation gauche : " Délimitation droite : "

	id_vendeur	pseudo	mdp	nom	prenom	telephone	email
1	ldubois	7c6a180b36...	Dubois	Luc	0474/78.56.19	ldubois@yahoo.fr	
2	rdurant	6cb75f652a...	Durand	Roger	0495/95.45.68	roger@gmail.com	
3	sterak	819b0643d6...	Terak	Sylvie	04/221.46.55	sylvie_92@scarlet.be	
4	mpottier	34cc93ece0...	Pottier	Martine	0496/65.35.62	mpo@gmail.com	
5	asfar	db0edd04aa...	Sfar	Antoine	04/221.91.85	sfara_45@hotmail.com	
6	sbecot	218dd27aeb...	Becot	Stéphane	0478/75.69.48	stef_bec@skynet.be	
7	lcront	00cdb7bb94...	Cront	Luc	0477/64.65.32	luc23@skynet.be	
8	cbaltus	b25ef06be3...	Baltus	Claudine	04/221.36.35	claudine_b@edpnet.be	
9	derniant	5d69dd95ac...	Erniant	Dominique	0474/56.57.98	dodo@gmail.com	
10	emerle	87e897e3b5...	Merle	Edgar	0475/49.65.23	edgar_merle@scarlet.be	
11	lbarmik	ac673bc7e5...	Barmik	Laura	0475/11.32.23	lbar@yahoo.be	

Aide < Précédent Suivant > Fin Annuler

Figure 4 : Import (1) : aperçu des données

ETAPE 2 : la méthode d'import est 'Insérer' (sélectionnée par défaut)

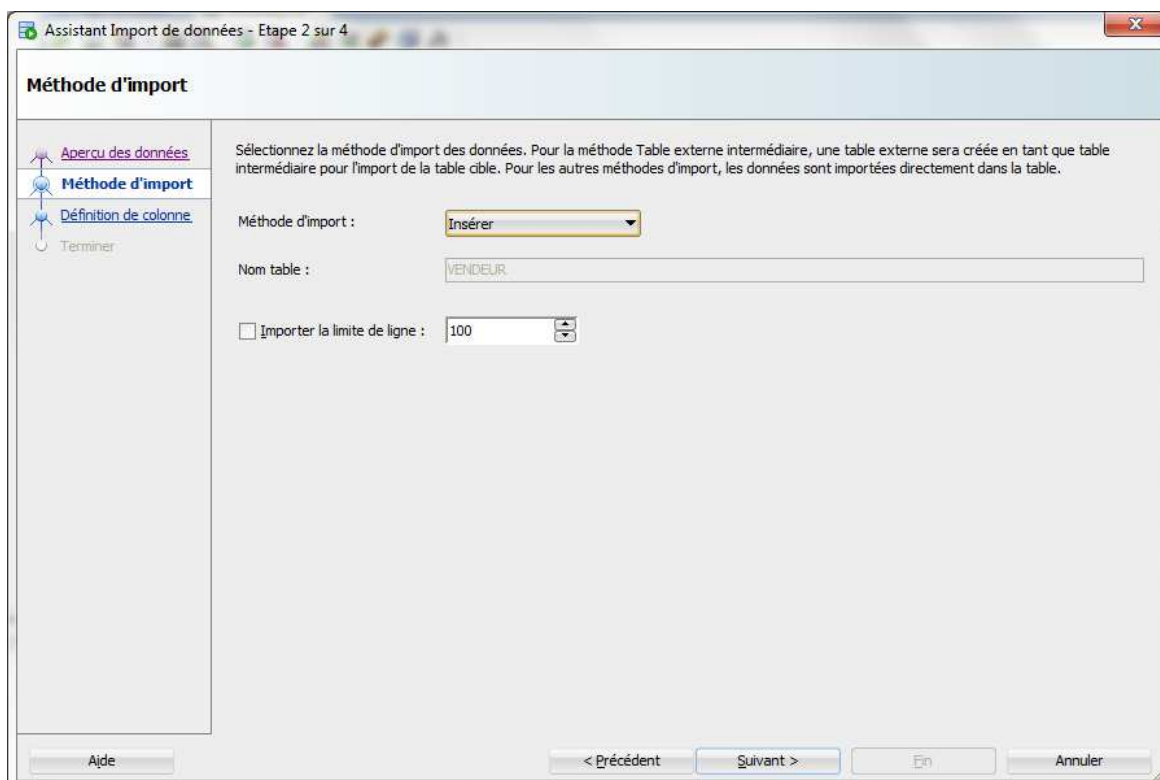


Figure 5 : Import (2) Méthode d'import

ETAPE 3 : on laisse le choix par défaut qui est de sélectionner toutes les colonnes présentes dans le fichier.

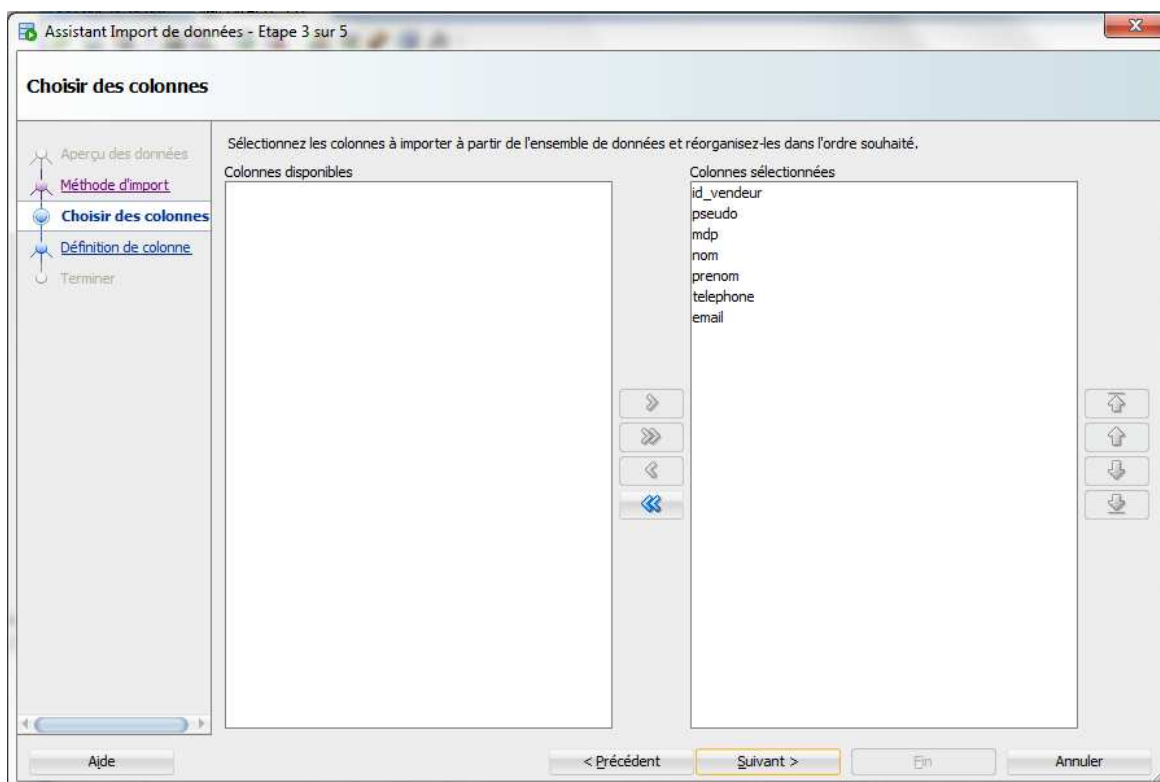


Figure 6 : Import (3) Choix des colonnes à importer

ETAPE 4 : on met en correspondance les données présentes dans le fichier et les différentes colonnes de la table. La correspondance pourra s'opérer sur le nom (car nos fichiers de données ont des entêtes en première ligne qui correspondent aux noms des colonnes. Certaines tables contiennent des données au format 'Date'. Il faut alors explicitement préciser le format qui est 'mm/dd/yy' dans nos fichiers).

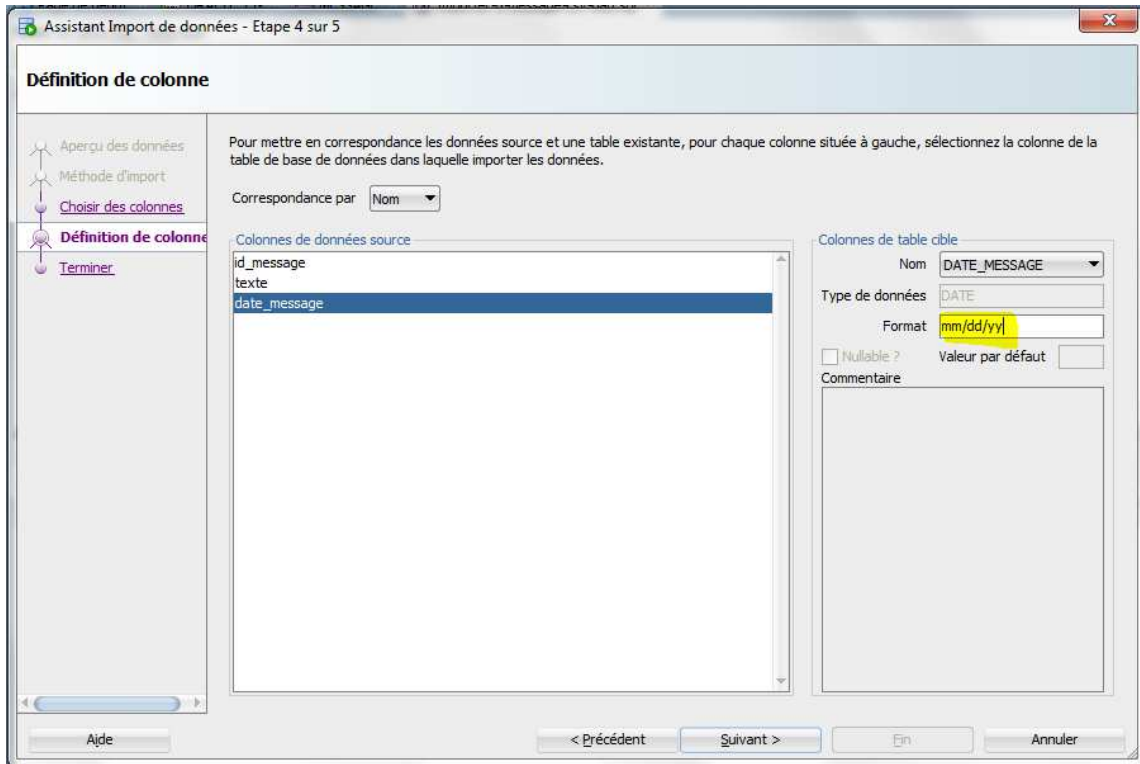


Figure 7 : Import (4) Correspondance données/colonnes

ETAPE 5 : on peut procéder à une vérification (optionnelle) et terminer l'import en cliquant sur le bouton 'Fin'

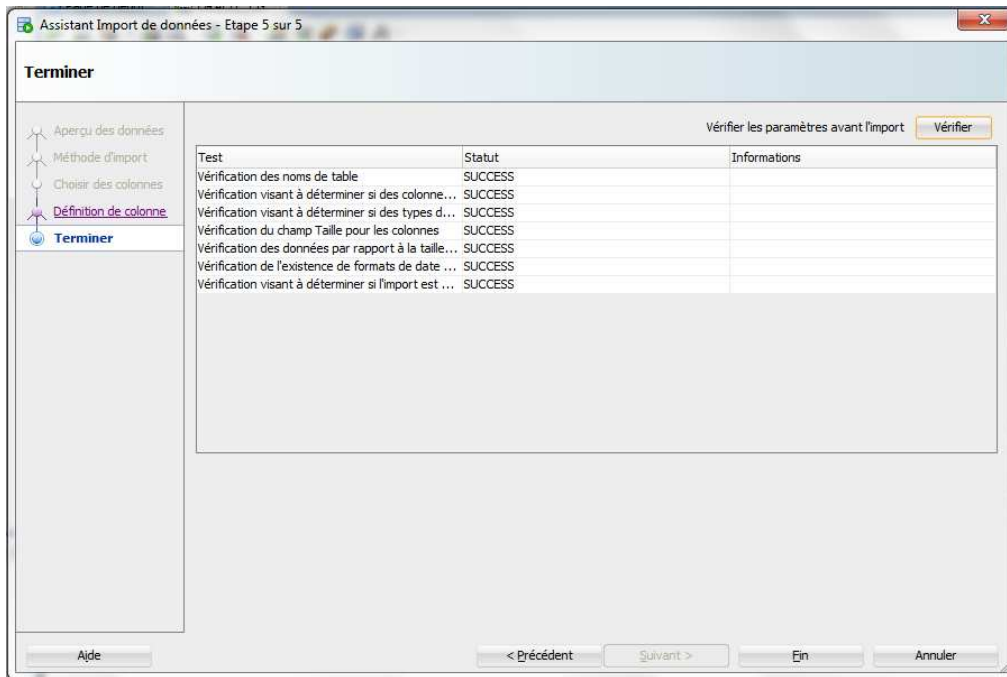


Figure 8 : Import (5) Vérification (optionnelle)

Bibliographie

- [1] C. MAREE et G. LEDANT, *SQL-2 : Initiation, Programmation*, 2^{ème} édition, Armand Colin, 1994, Paris
- [2] P. DELMAL, *SQL2 – SQL3 : application à Oracle*, 3^{ème} édition, De Boeck Université, 2001, Bruxelles
- [3] JL. HAINAUT, *Bases de données : concepts, utilisation et développement*, Dunod, 2009, Paris