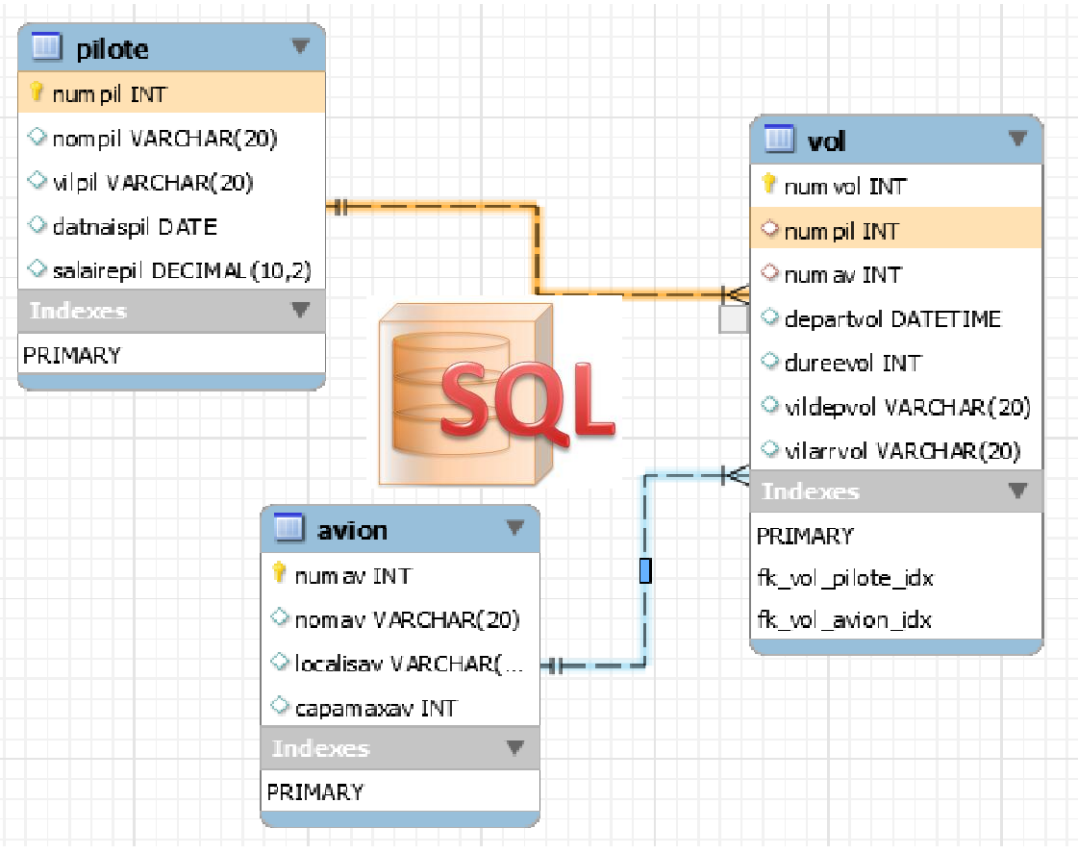
Eléments de cours - Année 1

Introduction aux bases de données et au langage SQL



1er 2018 - 2019

Table des matières

Table des figures

Liste des tableaux

Première partie

Introduction

1 Introduction

2 Pré-requis

Ce support considère comme acquises les notions fondamentales

- de modèle relationnel
- et d'algèbre relationnelle.

3 Bases de données et SGBD

3.1 Bases de données relationnelles

Une base de données (en anglais : *database*) est un ensemble de données structurées, généralement stocké sur un support de persistance (disque dur).

Les données d'une base de données relationnelle (en anglais : *relational database*) sont organisées sous forme de tables mises en relation.

Elles sont stockées sous forme de fichiers organisés spécifiquement selon le SGBD.

3.2 Système de Gestion de Bases de Données

Un SGBD (en anglais : *DBMS, Database Management System*) est un ensemble de progammes qui gère le contenu et la structure d'une base de données de manière :

- *sécurisée* : l'accès aux données est contrôlé ; des privilèges d'accès spécifiques sont attribués aux utilisateurs, ou groupes d'utilisateurs ;
- *partagée* : l'accès simultané aux données par plusieurs utilisateurs est permis : des mécanismes de contrôle sont mis en oeuvre afin de garantir la cohérence des données accédées ;
- *cohérente* : les données sont contrôlées en utilisant divers mécanismes comme le contrôle des doublons de valeur dans une colonne (intégrité d'entité), le contrôle qu'une donnée d'une table fait bien référence à une données identique dans une autre table (intégrité référentielle), etc. ;
- *fiable* : la garantie de stabilité des données est assurée, même après incident ; des systèmes de journalisation de toutes les modifications permettent le retour de la base de données à un état antérieur cohérent ;

- *performante* : l'accès aux informations est réalisé en un temps optimal grâce à des structures de données performantes (sous forme d'arbres) aussi bien pour stocker les données que pour les interroger ;
- *indépendante* des programmes : assure une indépendance des données au regard des applications et des utilisateurs qui y accède (indépendance logique) et des systèmes qui les hébergent (indépendance physique vis-à-vis des systèmes de gestion de fichiers)

Finalement, grâce à un langage standard, SQL, il est possible de décrire tous les objets d'une base de données et définir les liens existant entre eux, d'ajouter, modifier et supprimer des données, d'accéder aux données déjà mémorisées, etc.

Les SGBDR (R pour Relationnels) sont les SGBD des bases de données relationnelles (en anglais : *RDBMS, Relational Database Management System*)

3.3 SGBDR fichier

Un SGBDR fichier, comme Microsoft Access, SQLite, Open Office Base s'appuie sur un logiciel qui assure l'accès et la gestion d'une base de données présentée sous forme d'un fichier unique.

Il est souvent associé à une utilisation restreinte à un seul utilisateur (ou bien à un nombre très peu élevé). Il n'assure pas tous les services attendus d'un SGBD, notamment la fiabilité grâce à des mécanismes de journalisation.

3.4 SGBDR serveur

Un SGBDR serveur, comme MySQL, Oracle, IBM DB2, PostgreSQL, Microsoft SQL Server s'appuie sur un processus (logiciel qui s'exécute en tâche de fond et qui attend les requêtes) qui assure la gestion des fichiers d'une base de données en garantissant toutes les caractéristiques attendues.

Les interactions entre un logiciel applicatif (client, demandeur) et le SGBD (fournisseur, processus serveur) fonctionnent selon un dialogue définit par un protocole de type client-serveur (indépendamment des protocoles réseaux sous-jacents qui transportent les requêtes et les réponses).

3.5 Intégrité et contraintes

Afin d'assurer que les données seront fiables, différents mécanismes d'intégrité peuvent être mis en oeuvre :

- *intégrité de domaine* : assure qu'une colonne contient une donnée d'un certain type ;
- *intégrité d'entité* : assure qu'il n'y a pas de doublons de lignes dans une table ;
- *intégrité référentielle* : assure qu'un ligne ne peut être supprimée si elle est référencée par une autre ligne ;
- autres formes d'intégrité personnalisables (comme les déclencheurs (en anglais : *triggers*)).

Les contraintes sont les règles que le SGBDR devra contrôler afin de garantir l'intégrité des données d'une base de données (voir section 7.1 en page 9).

3.6 SGBD et norme ANSI/SPARC

La norme ANSI/SPARC définit l'architecture fondamentale sur laquelle reposent les SGBD, en 3 couches (ou niveaux), ce qui permet de rendre indépendant les applications développées des bases de données qu'elles utilisent :

- niveau *externe* : offrir une vue utilisateurs d'un sous-ensemble du schéma de la BD (vues)
- niveau *conceptuel* (ou logique) : organisation logiques de données et des liens (schémas de bases de données : tables, contraintes)
- niveau *interne* (ou physique) : implantation des BD sur des supports physiques (fichiers)

Chacun des niveaux est indépendant des autres. Ainsi l'administrateur de la base de données peut-il modifier

- le stockage des données (sur plusieurs disques par exemple) sans que cela n'influe sur la représentation conceptuelle des données
- le schéma conceptuel (ajout de tables, par exemple) sans que cela n'influe sur les vues qu'ont les utilisateurs sur la base de données.

4 Du modèle relationnel au modèle physique de base de données

Le modèle de Codd est le socle théorique des bases de données relationnelles, qui représentent la majorité des bases de données de production en 2016.

4.1 Règles de passage du Modèle Relationnel au modèle physique

1. chaque relation devient une table
2. les attributs deviennent des colonnes, chacune se voyant attribuer un type de donnée le plus proche possible du domaine de l'attribut correspondant : l'intégrité de domaine de valeur
3. un attribut clef primaire devient une clef primaire (en anglais : *primary key*) ; le SGBDR est responsable de la vérification de cette contrainte d'intégrité, l'intégrité d'entité
4. un attribut clef étrangère devient une clef étrangère (en anglais : *foreign key*) ; le SGBDR est responsable de la vérification de cette contrainte d'intégrité, l'intégrité référentielle
5. un attribut clef candidate devient un index qui peut autoriser les doublons ou pas ; le SGBDR assure également cette vérification

4.2 Choix du SGBD et types de données

Le choix d'un SGBD va définir les types de données à utiliser en lieu et place des domaines des attributs. En effet, bien qu'un standard soit défini, chaque éditeur de SGBD propose souvent des extensions aux types de données classiques.

La notion de domaine telle qu'on l'entend dans le modèle relationnel fait partie du standard SQL(92) mais elle est peu implantée dans les SGBDR.

La notion de domaine est alors remplacée par celle de type de données le plus proche du contenu.

Attention

Il faut néanmoins garder à l'esprit les domaines initiaux afin de ne pas effectuer d'opération non compatibles dans les requêtes d'interrogation.

TABLE 1 – Types de données du SGBDR MySQL (extrait)

type de donnée	signification
`char`(n)	chaine de caractères de longueur fixe (*n* caractères)
`varchar`(n)	chaine de caractères de longueur variable (au plus *n* caractères)
`int`	nombre entier
`numeric`(n,d)	nombre de *n* chiffres dont *d* décimales
`real`	nombre réel
`date`	date
`time`	heure
`boolean`	valeur booléenne (0=false, 1=true)

4.3 Convention de nommage et d'écriture

Les caractères acceptés pour les noms de tables et colonnes sont les caractères alphanumériques (lettres et chiffres), et le séparateur '_' (tiret bas).

D'autres caractères peuvent être utilisés mais nécessitent d'être encadrés par un couple '[]' ou '" "'. Le nom devrait commencer par une lettre.

Conseil

utilisez uniquement les caractères : [a .. z] ou [A .. Z], [0 .. 9], [_] pour coder les noms de tables et colonnes. Vous limiterez ainsi les problèmes de portage des requêtes d'une base de données vers une autre .

Deux approches coexistent dans l'écriture des requêtes SQL et concernent la casse des caractères :

- mots-clefs en lettres minuscules : les éditeurs mettent maintenant en évidence la syntaxe
- mots-clefs en lettres majuscules : pratique visant à disparaitre...

Nous nous en tiendrons à la 1ère.

4.4 Représentation du Modèle Physique de Données

La représentation graphique du MPD est très utile pour avoir une vue de l'ensemble des tables d'une base de données et des liens, exprimés sous forme de clefs étrangères, qui existent entre ces tables. Les types de données compléteront le diagramme.

Ce schéma est très utile lorsqu'il s'agit de construire des requêtes d'interrogation.

Exemple de la table R La relation R de schéma : R(A : lettre, B : lettre, C : lettre).

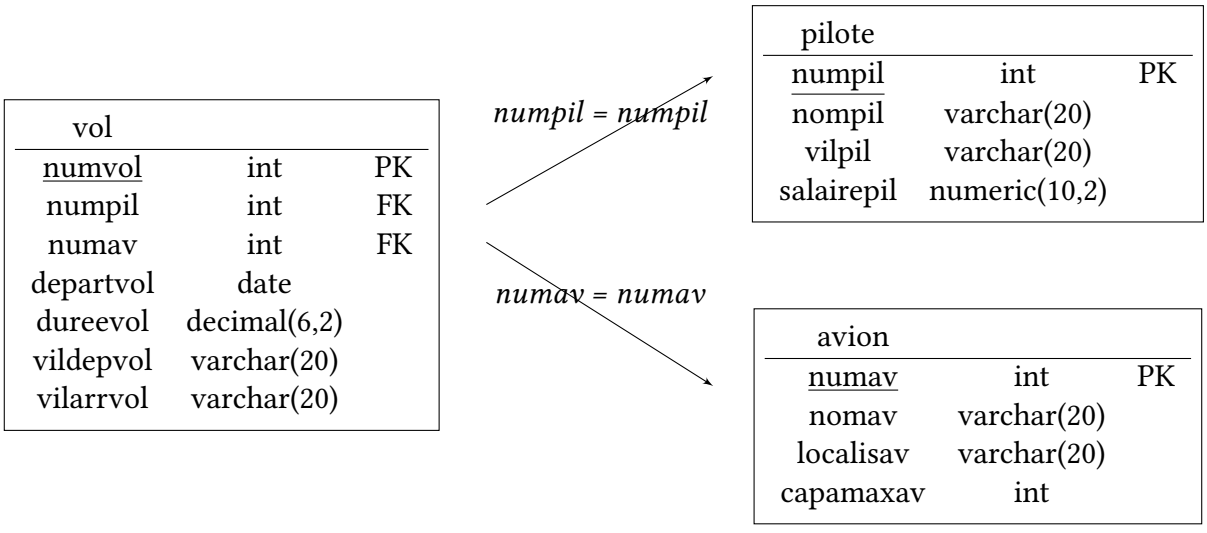
FIGURE 1 – Table R

R	
A	char(1)
B	char(1)
C	char(1)

Exemple de la base de données de suivi des vols

- **`pilote`** (<u>numpil</u>, nompil, vilpil : ville, datnaispil : date, salairepil)
  - \+ numpil : clef primaire
- **`vol`** (<u>numvol</u>, numpil#, numav#, departvol : date, dureevol, vildepvol : ville, vilarrvol : ville)
  - \+ numvol : clef primaire
  - \+ numpil : clef étrangère vers pilote(numpil)
  - \+ numav : clef étrangère vers avion(numav)
- **`avion`** (<u>numav</u>, nomav, localisav : ville, capamaxav : capacite)
  - \+ numav : clef primaire

FIGURE 2 – Modèle Physique de la base de données des vols



5 Le langage SQL

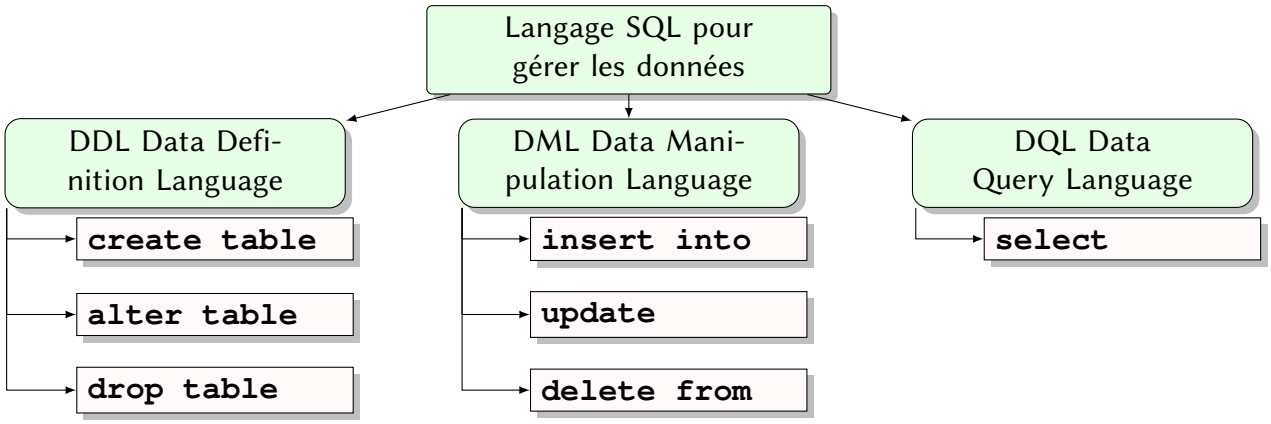
SQL (en anglais : *Structured Query Language*) est LE langage permettant de donner des ordres aux SGBDR afin d'accéder aux services qu'il offre : créer les tables, gérer leur contenu et les interroger mais aussi configurer la sécurité d'accès aux tables.

Le langage SQL s'appuie sur les opérateurs de l'algèbre relationnelle définis en 1970 par Ted Codd, mathématicien, chercheur chez IBM (à l'époque : Sequel, Structured English as a QUery Language).

SQL est un langage qui évolue :

- SQL1 (SQL89) : première version avec quelques défauts corrigés dans la suivante
- → *SQL2 (SQL92)* : la version la plus implantée actuellement (norme ANSI X3.135-1992, www.ansi.org) ; les versions suivantes vont étendre les possibilités
- SQL3 (SQL99) : Hybride objet-relationnel
- SQL4 (2003) : SQL Object Langage Binding, OLAP
- SQL5 (2006) : SQL/XML, XQuery
- SQL6 (2008) : SQL/JRT pour fonctions Java
- SQL7 (2011) : ISO/IEC 9075-2011

FIGURE 3 – Les ordres DDL, DML et DQL



Requête

Une *requête* est un ordre transmis au SGBD(R) afin de modifier la structure ou le contenu de la base de données, ou d'interroger les données qui y sont stockées.

Une requête retourne un résultat différent selon le type d'ordre : soit une information signalant que l'ordre a bien été exécuté, soit les informations qui ont été demandées.

Le langage SQL permet de couvrir tous les besoins d'accès aux bases de données relationnelles grâce à 5 groupes d'ordres SQL :

- *Data Definition Language* (DDL), le Langage de Définition des Données (LDD), permet la gestion des tables de la base de données (création, jouter des colonnes, supprimer des colonnes, etc.)
- *Data Manipulation Language* (DML), le Langage de Manipulation des Données (LMD) donne accès au contenu des tables : ajouter des lignes, modifier des valeurs de colonnes, supprimer des lignes.
- *Data Query Language* (DQL), le Langage d'Interrogation des Données (LID) permet de construire des requêtes d'interrogation des données à partir des tables.
- *Data Control Language* (DCL), le Langage de Contrôle des Données (LCD), permet la gestion des privilèges d'accès aux tables par utilisateur (voir section 36 en page 107)
- *Transaction Control Language* (TCL), le Langage de Contrôle des Transactions, permet la gestion de l'intégrité des données dans la base de données (voir section 35 en page 105)

D'autres ordres spécifiques peuvent compléter cet ensemble pour réaliser des tâches d'administration de la base de données.

Les requêtes SQL d'interrogation sont soumises au SGBDR qui effectue les traitement suivants :

1. *analyse lexicale* du texte de la requête : en cas d'erreur de syntaxe, une erreur est retournée
2. *traduction* en opérateurs d'algèbre relationnelle
3. *optimisation* à partir des statistiques de volumétrie des tables utilisées et construction du plan d'exécution
4. *exécution* et retour du résultat

6 Terminologie

En passant de la théorie de Codd aux bases de données relationnelles, de nouveaux termes s'appliquent :

Table 2 – Terminologie relationnelle/bases de données

algèbre relationnelle	*bases de données relationnelles et SQL*
relation	table (en anglais : *table*)
attribut	colonne (en anglais : *column*) ou champ (en anglais : *field*)
n-uplet ou tuple	ligne (en anglais : *row*) ou enregistrement (en anglais : *record*)
domaine de valeur	type de donnée (en anglais : *data type*)
expression d'algèbre relationnelle	requête (en anglais : *query*)

Les langage SQL prend quelques libertés avec la théorie de l'algèbre relationnelle. Cela sera précisé.

Deuxième partie

Définir la structure des tables

7 DDL - gérer les tables et l'intégrité

Les ordres DDL (en anglais : *Data Definition Language*), en français LDD, langage de Définition des Données, permettent la gestion des objets d'une base de données (tables, index, vues, fonctions, etc.). Nous nous intéresserons essentiellement aux tables ici.

Les requêtes basées sur des ordres SQL DDL relatifs aux tables ne touchent qu'à la *structure d'accueil* des données, pas au *contenu.*[1]

7.1 Contraintes d'intégrité

La définition des contraintes assigne au SGBDR la tâche de vérification des données lors de la modification du contenu et de la structure des tables.

Pour assurer ces contrôles, les règles suivantes peuvent être définies à la création d'une table :

- **`not null`** : assure qu'une colonne ne peut avoir une valeur nulle
- **`default`** : assure une valeur par défaut au cas où la valeur n'est pas renseignée pour une colonne
- **`unique`** : assure que toutes les valeurs d'une colonne seront différentes (chacune sera unique...)
- **`primary key`** : identifie chaque ligne avec une valeur unique
- **`foreign key`** : identifie une ligne d'une autre table (généralement) avec une valeur unique
- **`check`** : détermine les conditions de validité d'une valeur, les valeurs autorisées pour une colonne

7.2 Séparer la construction des tables et l'expression des contraintes

Cela permet de séparer 2 phases dans la construction d'une table

- la définition du 'conteneur', les colonnes et leur type de données
- la définition des 'contraintes' que le SGBD aura à effectuer lors du stockage de valeurs dans ces colonnes (les contraintes appliquées aux colonnes)

Le nommage explicite des contraintes permet également d'identifier clairement les problèmes liés à celles-ci (messages d'erreurs), de les supprimer ou de les modifier.

[1] Attention quand même : quand on supprime une table, son contenu disparait également !

8 DDL - `create database` et `drop database`

L'ordre SQL **`create database`** crée une nouvelle base de données, l'ordre SQL **`drop database`** supprime une base de données et tous les objets qu'elle contient.

Syntaxe SQL

Créer une base de données

```
1 create database nomBase
2   character set  nomJeuDeCaracteres
3 ;
```

où :

- *nomBase* : nom donné à la base de donnée
- *nomJeuDeCaracteres* : nom du jeu de caractères, c'est à dire un ensemble nommé de symboles encodés[2] qui sera utilisé par défaut pour les tables de la base de données
  - *latin1* : jeu de caractères d'Europe occidentale
  - *utf8* : jeu de caractères universel

  Par défaut, un jeu de caractères est associé à une 'collation' qui est la manière dont certains caractères sont considérés lors des classements, par exemple ; on trouvera des collations 'CI' pour 'case insensitive', soit insensibles à la casse, d'autres 'CS' pour 'case sensitive', sensibles à la casse.

Syntaxe SQL

Supprimer une base de données

```
1 drop database nomBase
2 ;
```

Attention

Attention à l'utilisation de l'ordre **`drop database`** qui supprime tous les objets de la base de données et ses fichiers du disque dur.

[2] la commande 'show character set' liste tous les noms des jeux de caractères disponibles

9 DDL - `create table`

L'ordre SQL **`create table`** permet la création d'une table.

Créer une table, c'est définir :

- la *structure d'accueil de données* : les colonnes associées à des types de donnée
- les *règles de vérifications* de ces données : les contraintes

9.1 Syntaxe

Syntaxe SQL

Créer une table

```
create table nomTable (
  nomColonne type [contrainte][auto_increment][defaut],
  [nomColonne type [contrainte][defaut],etc.]
);
```

où :

- *nomColonne* : nom d'une colonne de la table
- *type* : type de donnée associé à la colonne
- *contrainte* : contrainte concernant une seule colonne
- *defaut* : valeur par défaut lors de l'ajout d'une ligne
- **`auto_increment`** : mot-clef précisant que la valeur de cette colonne sera, par défaut, incrémentée automatiquement

Syntaxe SQL

Créer une table à partir de la description d'une autre

```
create table nomTable like autreTable
;
```

Lorsqu'on crée une table sous MySql, plusieurs fichiers sont créés sur le disque dur :

- nomTable.frm : définition de la table (la structure)
- nomTable.myd : contenu de la table (les données)
- nomTable.myi : fichier d'index

9.2 Exemple

Listing 1 – Ordre SQL de création de la table 'personnel' (conteneur des données)

```
create table personnel    (
  numero      int           not null,
  nom         varchar(16)   not null,
  prenom      varchar(16)   not null,
  ville       varchar(16)   null,
  salaire     decimal(8,2)  not null default 100,
  dateentree  date          not null,
  sexe        char(1)       not null default 'h'
);
```

Listing 2 – Ordre SQL de création de la table 'produit'

```
create table produit (
  refProduit    int           not null,
  designation   varchar(20)   not null,
  stock         int           null default 0,
  codeFamille   char(4)       not null
);
```

Listing 3 – Ordre SQL de création de la table 'famille'

```
create table famille (
  codeFamille   char(4)       not null,
  nomFamille    varchar(20)   not null
);
```

Ici, on a volontairement ignoré les contraintes d'intégrité d'entité et référentielle. Elle pourront être définies en utilisant l'ordre **`alter table`**.

Conseil

Séparer la définition du conteneur de celle des contraintes est une bonne pratique.

L'ordre **`show columns`** permet l'affichage de la structure d'une table :

Listing 4 – Afficher de la description de la table 'personnel'

```
show columns from personnel;
```

personnel	Field	Type	Null	Key	Default	Extra
	numero	int(11)	NO		null	
	nom	varchar(16)	NO		null	
	prenom	varchar(16)	NO		null	
	ville	varchar(16)	YES		null	
	salaire	decimal(8,2)	NO		100.00	
	dateentree	date	NO		null	
	sexe	char(1)	NO		h	

nombres de lignes : 7

L'ordre **`show create table`** permet l'affichage d'ordre SQL de création de la table :

Listing 5 – Afficher de la description de la table 'personnel'

```
1 show create table personnel;
```

personnel	Table	Create Table
	personnel	CREATE TABLE 'personnel' ('numero' int(11) NOT NULL, 'nom' varchar(16) NOT NULL, 'prenom' varchar(16) NOT NULL, 'ville' varchar(16) DEFAULT NULL, 'salaire' decimal(8,2) NOT NULL DEFAULT '100.00', 'dateentree' date NOT NULL, 'sexe' char(1) NOT NULL DEFAULT 'h') ENGINE=InnoDB DEFAULT CHARSET=latin1

nombres de lignes : 1

10 DDL - `alter table`

L'ordre SQL **`alter table`** permet la modification des caractéristiques d'une table : ajouter ou supprimer des colonnes, modifier le type de donnée d'une colonne, ajouter ou supprimer des contraintes, etc.

10.1 Syntaxe

Syntaxe SQL

Modifier la structure ou des contraintes associées à une table

```
1 alter table  nomTable
2   add constraint defContrainte
3 | add [column] defColonne
4 | alter [column] {set default valeur | drop default }
5 | modify [column] nomColonne defColonne
6 | change [column] nomColonne nouvColonne defColonne
7 | rename [to|as] nouvNomTable
8 | drop [constraint] nomContrainte
9 | drop foreign key nomContrainte
10 | drop primary key
11 | drop [column] nomColonne
12 ;
```

où :

- *nomTable* : nom de la table concernée par la modification de structure
- *defContrainte* : contrainte concernant une colonne ou une table : type, nom, etc.
- *defColonne* : définition d'une colonne : nom, type, contrainte
- *nomColonne* : nom d'une colonne de la table
- *nouvColonne* : nouveau nom de colonne de la table

- *nomContrainte* : nom d'une contrainte associée à la table
- *type* : type de donnée associé à la colonne
- *defaut* : valeur par défaut d'une colonne
- *nouvNomTable* : nouveau nom de la table

10.2 Définition des contraintes

10.2.1 Clef primaire (en anglais : *primary key*) : intégrité d'entité

Le maintien de l'*intégrité d'entité* consiste à s'assurer que chaque ligne d'une table possédera une *clef primaire* différente.

La clef primaire peut être composée de plusieurs colonnes.

Listing 6 – Contrainte d'intégrité d'entité en SQL (MySQL)

```
1 constraint nomContrainte
2   primary key (colonnesClef)
```

où :

- **constraint** : mot-clef introduisant une contrainte
- *nomContrainte* : nom que l'on donne à la contrainte (par exemple 'PK_x' pour une contrainte d'intégrité d'entité sur la table 'x')
- **primary key** : mot-clef introduisant le type de la contrainte, ici de clé primaire
- *colonnesClef* : colonne(s) composant la clef primaire

10.2.2 Clef étrangère (en anglais : *foreign key*)

Le maintien de l'*intégrité référentielle* consiste à s'assurer que l'ensemble de colonnes formant une *clef étrangère* existe bien comme *clé primaire* dans la table cible, généralement une autre table.

La clef étrangère peut être composée de plusieurs colonnes.

Listing 7 – Contrainte d'intégrité référentielle avec MySQL

```
1 constraint nomContrainte
2   foreign key (colonnesClef)
3     references nomTableRef (colonnesClef)
4       [on update {cascade | set null} ]
5       [on delete {cascade | set null} ]
6       [ [not] deferrable]
```

avec

- **constraint** : mot-clef introduisant une contrainte

- *nomContrainte* : nom que l'on donne à la contrainte (par exemple 'FK_x_y' pour une contrainte de la table source 'x' vers la table cible 'y')
- **foreign key** : mot-clef introduisant le type de la contrainte
- *colonnesClef* : colonne(s) composant la clef étrangère
- **references** : mot-clef introduisant la cible de la contrainte
- *nomTableRef* et *colonnesRef* : table à laquelle la contrainte fait référence et clef primaire de cette table
- **on update** : action à prendre en cas de modification de la valeur dans la colonne référencée : modification de la valeur de la ligne ou valeur positionnée à la valeur **null** (sans cette clause, la modification est refusée)
- **on delete** : action à prendre en cas de suppression de la valeur dans la colonne reference : suppression de la ligne, valeur positionnée à la valeur **null** (sans cette clause, la suppression est refusée)
- **deferrable** : permet de différer la vérification de l'intégrité référentielle à la fin d'une transaction[3] (et non pas immédiatement), et ainsi permettre les dépendances mutuelles [4] pour le temps d'une transaction.

10.3 Exemples

Listing 8 – Ordre SQL d'ajout d'une colonne 'stockMini'

```
1 alter table  produit
2   add column stockMini real null default 0
3 ;
```

Listing 9 – Ordre SQL d'ajout d'une contrainte de vérification de valeur

```
1 alter table  produit
2   add constraint Ck_Verif_Stock
3     check (stock >= stockMini)
4 ;
```

la contrainte 'check' n'est pas reconnu par MySQL

[3] Sur les transactions, Cf section 35 en page ??.
[4] cf. "set constraint nomContrainte deferred"

Listing 10 – Ordre SQL de suppression de la colonne 'stockMini' : les données sont perdues

```
alter table  produit
  drop column stockMini
;
```

Listing 11 – Ordre SQL d'ajout de clef primaire (intégrité d'entité)

```
alter table  produit
  add constraint PK_Produit
    primary key (refProduit)
;
alter table  famille
  add constraint PK_Famille
    primary key (codeFamille)
;
```

Listing 12 – Ordre SQL d'ajout de clef étrangère (contrainte d'intégrité référentielle)

```
alter table produit
  add constraint FK_Produit_Famille
    foreign key (codeFamille)
      references famille (codeFamille)
;
```

Listing 13 – Ordre SQL de suppression de clef étrangère

```
alter table produit
  drop foreign key FK_Famille_Produit
;
```

L'ordre **`show columns`** permet l'affichage de la structure d'une table avec des informations sur les index :

Listing 14 – Afficher de la description de la table 'personne'

```
show columns from personne;
```

personnel	Field	Type	Null	Key	Default	Extra
	numero	int(11)	NO	PRI	0	
	nom	varchar(16)	YES	MUL	null	
	prenom	varchar(16)	YES		null	
	ville	varchar(16)	YES		null	
	salaire	decimal(8,2)	YES		null	
	dateEntree	date	YES		null	
	sexe	char(1)	YES		h	

nombres de lignes : 7

où :

- la colonne 'key' contient : 'pri' pour **`primary key`**, 'mul' pour un index avec doublons (une clef étrangère ou un index),

- la colonne 'extra' contient des informations complémentaires comme **`auto_increment`**.

11 DDL - `drop table`

L'ordre SQL drop table supprime une table de la base de données : les données sont donc perdues définitivement.

Suppression de table
Les données sont perdues !

11.1 Syntaxe

Syntaxe SQL
Supprimer une table

```
1 drop table [if exists] nomTable;
```

où :

- nomTable : nom de la table à supprimer

11.2 Exemple

Soient les tables 'produit' et 'famille', soit la clef étrangère Fk_Produit_Famille :

Listing 15 – Ordre SQL de suppression de la table 'famille', cible d'une contrainte d'intégrité référentielle

```
1 drop table famille
2 ;
```

L'ordre de suppression de la table 'famille' produit une erreur :

ERROR 1217 (23000) :
Cannot delete or update a parent row : a foreign key constraint fails

Pour pouvoir supprimer la table, il faut d'abord supprimer les contraintes dont elle est la cible, celle-ci se trouvant généralement dans d'autres tables.

12 DDL - `create index` et `drop index`

Un index sert à accélérer les recherches portant sur une ou plusieurs colonnes d'une table, autres que la clef primaire, pour laquelle un index est automatiquement créé. En général, l'index porte sur des clefs candidates des relations d'origine.

Un index est une table système référençant toutes les valeurs d'une colonne (une clef) et l'adresse de la ligne où elles se trouvent dans la table (cf. Section 34 en page **??**.)

Par exemple, pour une table 'client' où chaque ligne possède un numéro de client et un nom, le numéro de client sera clef primaire mais il pourra être utile de créer un index sur le nom du client si des recherches portent souvent sur cette colonne.

12.1 Syntaxe

Syntaxe SQL

Créer un index

```
1 create [unique] index nomIndex
2   on nomTable (nomColonne, ,nomColonne1,...)
3 ;
```

Syntaxe SQL

Supprimer un index

```
1 drop index nomIndex on nomTable
2 ;
```

où :

- *nomIndex* : nom de l'index à supprimer (parfois préfixé par 'IDX' comme repère)
- *nomTable* : nom de la table concernée par l'index
- *nomColonne*, *nomColonne1* : nom d'une (ou plusieurs) colonne de la table sur laquelle porte l'index
- l'option **`unique`** précise qu'on n'autorisera pas de doublons dans cette colonne : cela peut être le cas d'un code INSEE, d'une adresse mail, etc. ; par contre dans le cas d'un index sur le nom et le prénom, on pourra autoriser les doublons.

12.2 Exemple

Listing 16 – Ordre SQL de création d'un index sur le nom d'une personne

```
1 create index Idx_Nom_Personne
2   on personne (nom)
3 ;
```

Troisième partie

Gérer le contenu des tables

13 DML - gérer le contenu des tables

Les ordres d'ajout de lignes, de mise à jour des valeurs de colonnes et de suppression de lignes interviennent au niveau du contenu des tables.

Ces ordres déclenchent

- l'*application des contraintes d'intégrité* spécifiées lors de la création des tables
- les *mécanismes de gestion de transaction* afin de garantir l'intégrité des données lors d'accès concurrent aux données
- les *mécanismes de journalisation* permettant de garantir l'intégrité des données en cas d'erreur en cours de transaction ou en cas de nécessité de restauration d'une base à un état antérieur avec ré-application des transactions .

14 DML - `insert into`

L'ordre SQL **`insert into`** permet l'ajout de lignes dans une table.

14.1 Syntaxe

Syntaxe SQL

Ajouter une ligne complète avec des valeurs fixes - *non recommandé*

```
1 insert into nomTable
2   values ( val1[,val2, val3, ...])[,
3          ( val1[,val2, val3, ...]),...]
4 ;
```

où :

- *nomTable* : nom de la table concernée par l'ajout
- *val1, val2, val3, ...* : liste des valeurs associées chacune à une colonne de la table ; toutes les colonnes doivent avoir une valeur qui leur correspond (dans l'ordre défini dans la table) ; on peut ainsi ajouter plusieurs lignes

Attention

L'ordre des valeurs doir être identique aà celui des colonnes dans la table. En cas d'ajout de nouvelles colonnes, la requête doit être modifiée.

Syntaxe SQL

Ajouter une ligne partielle avec des valeurs fixes - *recommandé*

```
insert into nomTable
          (col1[, col2, col3, ...])
  values ( val1[,val2, val3, ...])[,
          ( val1[,val2, val3, ...]),...]
;
```

où :

- *nomTable* : nom de la table concernée par l'ajout
- *col1, col2, col3, ...* : liste des noms de colonnes dont les valeurs vont être précisées dessous
- *val1, val2, val3, ...* : liste des valeurs associées chacune à une colonne ; toutes les colonnes doivent avoir une valeur qui leur correspond (dans l'ordre défini juste au dessus)

Syntaxe à privilégier

Cette syntaxe est la plus sûre :

- elle est indépendante l'ordre des colonnes dans la table
- elle reste correcte même si des colonnes (non obligatoires) ont été ajoutées

Les colonnes pour lesquelles aucune valeur n'aura été précisée auront une valeur nulle (**`null`**) ; l'insertion pourra être réalisée seulement si aucune contrainte **`not null`** n'est spécifiée pour ces dernières.

Syntaxe SQL

Ajouter une ligne partielle ou complète à partir d'une requête d'interrogation (select)

```
insert into nomTable  [(col1[, col2, col3, ...])]
  requete
;
```

où :

- *nomTable* : nom de la table concernée
- *col1, col2, col3, ...* : liste des noms de colonnes dont les valeurs vont être précisées dessous
- *requête* : requête SQL d'interrogation (**select**)

14.2 Exemple

Listing 17 – Ordre SQL d'ajout d'une personne, toutes les colonnes

```
1 insert into personne
2   values (1,'Dupont','Max', 'arras' , 1000, '2007-01-01')
3 ;
```

Listing 18 – Ordre SQL d'ajout d'une personne, certaines colonnes

```
1 insert into personne
2           (numero, nom, prenom, salaire, dateEntree)
3   values (5,'Rigole','Jean', 1300, '2007-09-28')
4 ;
```

La colonne 'ville' n'a pas été renseignée : elle aura la valeur nulle autorisée lors de la création de la table.

15 DML - update

L'ordre SQL **update** modifie les valeurs de colonnes dans une table, en tenant éventuellement compte d'une condition de sélection des lignes à modifier grâce à une clause **where** (voir section 19 en page 28 pour l'expression des conditions).

Attention
un ordre de mise a jour sans clause **where** affecte toute les lignes de la table !

15.1 Syntaxe

Syntaxe SQL
Modifier les valeurs d'une ou plusieurs colonnes

```
1 update nomTable
2   set col1 = valeur1 [,col2 = valeur2, col3 = valeur3
      ,...]
3   [where condition]
4 ;
```

où :

- *nomTable* : table mise à jour
- *col1, col2, col3, ...* : colonnes dont le contenu est modifié
- *valeur1, valeur2, valeur3, ...* : nouvelles valeurs
- *condition* : condition spécifiée peut être exprimée :
  - par rapport aux colonnes de la table mise à jour
  - par rapport à une sous-requête indépendante ou corrélée

15.2 Exemple

Listing 19 – Ordre SQL de modification de la colonne 'prenom' d'une personne sélectionné sur son numéro

```
1 update personne
2   set prenom = 'john'
3   where numero = 4
4 ;
```

Listing 20 – Ordre SQL de modification de la colonne 'salaire', augmentation de 10% pour tous

```
1 update personne
2   set salaire = salaire * 1.1
3 ;
```

Listing 21 – Ordre SQL de modification de la colonne 'salpil', augmentation de 10% pour tous les pilotes qui ont assuré un vol

```
1 update pilote
2   set salpil = salpil * 1.1
3   where numpil in (select numpil from vol)
4 ;
```

16 DML - `delete from`

L'ordre SQL **`delete from`** supprime une ou plusieurs lignes d'une table en tenant éventuellement compte d'une condition de sélection des lignes à supprimer grâce à une clause **`where`** (voir section 19 en page 28 pour l'expression des conditions).

Attention

un ordre de suppression sans clause **where** supprime toute les lignes de la table[a]

[a] à moins que des contraintes d'intégrité référentielles l'en empêche...

16.1 Syntaxe

Syntaxe SQL

Supprimer des lignes d'une table

```
1 delete from nomTable
2   [where condition]
3 ;
```

où :

- *nomTable* : nom de la table dont on veut supprimer des lignes
- *condition* : la condition de sélection des lignes à supprimer, exprimée :
  - en lien avec les colonnes de la table
  - en utilisant une sous-requête (indépendante ou corrélée)

16.2 Exemple

Listing 22 – Ordre SQL de suppression des lignes de pilote dont le numero est 10

```
1 delete from pilote
2   where numpil = 10
3 ;
```

Listing 23 – Ordre SQL de suppression des lignes de pilote qui n'ont pas volé

```
1 delete from pilote
2   where numpil not in (
3     select distinct numpil
4       from vol
5   )
6 ;
```

Quatrième partie

Effectuer des requêtes d'interrogation

17 DQL - `select from`

Le langage SQL pour interroger les tables propose une phrase complète pour réunir l'ensemble des opérateurs algébriques d'une requête.

Cette phrase est constituée de mots-clefs introduisant la succession des étapes réalisées :

1. **`select`** : sélectionner les colonnes du résultat final (projection)
2. **`from`** : rassembler toutes les tables utiles à l'exécution de la requête (jointures, produit cartésien) pour constituer les lignes brutes
3. **`where`** : sélectionner les lignes qui nous seront effectivement utiles (sélection)
4. **`group by`** : définir les niveaux de regroupement pour le calcul d'agrégats
5. **`having`** : sélectionner les lignes après que les calculs d'agrégats ont été réalisés (sélection)
6. **`order by`** : classer le résultat

La phrase est terminée par un ';' (point-virgule).

Syntaxe SQL

Interroger les lignes et colonnes des tables

```
select  [distinct | all] {* | listeDeColonnes}
  from table(s)
  [where criteresDeSelection]
  [group by listeDeColonnesDeRegroupement]
  [having criteresDeSelectionApresRegroupement]
  [order by criteresDeClassement]
  ;
```

où :

- *listeDeColonnes* : colonnes retournées par la requête, séparées par une virgule (sauf la dernière)
- *table(s)* : la ou les tables (jointes de manière cohérente)
- *criteresDeSelection* : expression logique de sélection des lignes extraites
- *listeDeColonnesDeRegroupement* : noms des colonnes sur lesquelles appliquer le regroupement, séparées par une virgule, sauf la dernière

- *criteresDeSelectionApresRegroupement* : nouvelle sélection appliquée généralement à des calculs agrégés
- *critèresDeClassement* : noms des colonnes et type de classement, séparée par une virgule, sauf le dernier

18 Projection

18.1 Syntaxe

Projection SQL

$\pi_{xa,...,xz}(R)$

```
select distinct xa,...,xz
  from R
  ;
```

où :

- **distinct** : mot-clef SQL de suppression des doublons
- *xa,...,xz* : la liste des colonnes conservées
- *R* : table d'origine

Projection avec conservation des doublons en SQL

```
select  [all] xa,...,xz
  from R
  ;
```

où :

- **all** : mot-clef SQL de non-suppression des doublons (action par défaut)
- *xa,...,xz* : la liste des colonnes conservées
- *R* : table d'origine

18.2 Exemples

18.2.1 Exemple1

personne	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h
	5	Rigole	Jean	null	1300.00	2007-09-28	h

nombres de lignes : 5

Listing 24 – lister les nom et prénom des personnes

```
1 select nom, prenom
2   from personne;
```

resultat	nom	prenom
	Dupont	Max
	Durand	Tim
	Lambert	Betty
	Bradford	Jean
	Rigole	Jean

nombres de lignes : 5

18.2.2 Exemple2

personne	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h
	5	Rigole	Jean	null	1300.00	2007-09-28	h

nombres de lignes : 5

Listing 25 – lister les villes

```
1 select all ville
2   from personne;
```

resultat	ville
	arras
	Aix
	Pau
	Arras
	null

nombres de lignes : 5

On peut constater que le doublon n'a pas été supprimé : par défaut, SQL ne respecte pas le socle théorique de l'algèbre relationnelle.

Listing 26 – lister les villes (sans doublons) ($\pi_{ville}(personne)$)

```
1 select distinct ville
2   from personne
3 ;
```

resultat	ville
	arras
	Aix
	Pau
	null

nombres de lignes : 4

19 Sélection : `where`

19.1 Syntaxe

Sélection SQL

$\sigma_Q(R)$

```
1 select   *
2   from R
3   where Q
4 ;
```

où :

- `*` : facilité d'écriture = 'toutes les colonnes'
- R : table d'origine
- Q : condition de sélection des lignes

19.2 Opérateurs de comparaison et connecteurs logiques

Table 3 – Opérateur logiques

égal	=
différent	!= ou <>
inférieur	<
inférieur ou égal	<=
supérieur	>
supérieur ou égal	>=

Table 4 – Connecteurs logiques

et	**and**
ou	**or**
non	**not**

19.3 Opérateur étendus SQL

Table 5 – Opérateurs étendus SQL

est égal à une valeur au moins d'une liste	**in** (liste)
est différent de toutes les valeurs d'une liste	**not in** (liste)
est dans l'intervalle $[a, b]$	**between** a **and** b
n'est pas dans l'intervalle $[a, b]$	**not between** a **and** b
ressemble à un modèle	**like** "modele"
ne ressemble pas à un modèle	**not like** "modele"
a une valeur nulle	**is null**
n'a pas une valeur nulle	**is not null**

La liste de l'opérateur `in` est une suite de valeurs séparées par une virgule, par exemple : **`(1,3,8,9,2)`**, ou **`('calais', 'paris', 'brest')`**.

Le modèle de l'opérateur `like` L'opérateur **`like`** permet de comparer la valeur d'une colonne de type chaîne de caractères avec un modèle. Ce modèle est constitué d'un texte dans lequel peuvent être positionnés des caractères de substitution (ou caractères jokers ou caractères omnibus)[5] :

- % (ou *) : remplace plusieurs caractères
- _ (ou ?) : remplace un seul caractère

On peut ainsi effectuer les types de comparaison suivants, et tester que la valeur d'une colonne...

- *commence par* 'xx' : colonne **`like`** 'xx%'
- *contient* 'xx' : colonne **`like`** '%xx%'
- *se termine par* 'xx' : colonne **`like`** '%xx'

[5] Ces caractères de substitution ne sont pas les mêmes pour tout les SGBDR...

- *contient* un 'x' au caractère 2 : colonne **like** '_x%'
- *contient* un 'x' en avant dernier caractère : colonne **like** '%x_'
- a une longueur exacte de n caractères , par exemple 4 : colonne **like** '____'

Les opérateurs suivants sont particulièrement utilisés combinés à des requêtes imbriquées (voir Section 32 en page 75)

TABLE 6 – Opérateurs étendus SQL (2)

teste si l'opérateur s'applique à toutes les valeurs de la liste	operateur **all** (liste)
teste si l'opérateur s'applique à au moins une des valeurs de la liste	operateur **any** (liste)
teste si la liste n'est pas vide	**exists** (liste)

L'opérateur associé à **all** et **any** est l'un des opérateur de comparaison.

19.4 Exemples : opérateurs de comparaison

19.4.1 Exemple1

R	A	B	C
	a	b	c
	d	a	f
	c	b	d

nombres de lignes : 3

Listing 27 – Sélection SQL ($\sigma_{B<>'a'}(R)$)

```
1 select  *
2   from R
3   where B <>  'a'
4 ;
```

resultat	A	B	C
	a	b	c
	c	b	d

nombres de lignes : 2

19.4.2 Exemple2

personne	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h
	5	Rigole	Jean	null	1300.00	2007-09-28	h

nombres de lignes : 5

Listing 28 – les personnes dont le numéro est inférieur à 3

```
1 select  *
2   from personne
3   where numero < 3
4 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	2	Durand	Tim	Aix	1500.00	2007-03-15	h

nombres de lignes : 2

Listing 29 – les personnes entrés avant le 1er juin 2007

```
1 select  *
2   from personne
3   where dateEntree < '2007-06-01'
4 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f

nombres de lignes : 3

Listing 30 – les personnes habitant à Arras

```
1 select  *
2   from personne
3   where ville = 'Arras'
4 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h

nombres de lignes : 2

19.5 Exemples : opérateurs SQL étendus

19.5.1 Exemple 1 : `in` et `not in`

Listing 31 – les personnes habitant Arras ou Aix

```
1 select  *
2   from personne
3   where ville = 'Arras' or ville = 'Aix'
4 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h

nombres de lignes : 3

Listing 32 – les personnes habitant Arras ou Aix : opérateur **in**

```
1 select  *
2   from personne
3   where ville in ('Arras' , 'Aix')
4 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h

nombres de lignes : 3

Listing 33 – les personnes dont le numéro est 1 3 ou 5

```
1 select  *
2   from personne
3   where numero = 1 or numero = 3 or numero = 5
4 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f
	5	Rigole	Jean	null	1300.00	2007-09-28	h

nombres de lignes : 3

Listing 34 – les personnes dont le numéro est 1 3 ou 5 : opérateur **in**

```
1 select  *
2   from personne
3   where numero in ( 1, 3, 5)
4 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f
	5	Rigole	Jean	null	1300.00	2007-09-28	h

nombres de lignes : 3

Listing 35 – les personnes dont le numéro n'est pas 1 3 ou 5 : opérateur **not in**

```
1 select  *
2   from personne
3   where numero not in ( 1, 3, 5)
4 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h

nombres de lignes : 2

19.5.2 exemple 2 : `like` et `not like`

Listing 36 – les personnes dont le nom commence par 'L' : opérateur **`like`**

```
1 select  *
2   from personne
3   where nom like 'L%'
4 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f

nombres de lignes : 1

Listing 37 – les personnes dont le nom ne contient pas 'am' : opérateur **`like`**

```
1 select  *
2   from personne
3   where nom not like '%am%'
4 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h
	5	Rigole	Jean	null	1300.00	2007-09-28	h

nombres de lignes : 4

19.5.3 Exemple 3 : `between` et `not between`

Listing 38 – les personnes dont le numéro est compris entre 2 et 5

```
1 select  *
2   from personne
3   where numero >=2 and numero <= 5
4 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h
	5	Rigole	Jean	null	1300.00	2007-09-28	h

nombres de lignes : 4

Listing 39 – les personnes dont le numéro est compris entre 2 et 5 : opérateur **`between`**

```
1 select  *
```

```
2   from personne
3   where numero between 2 and 5
4 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h
	5	Rigole	Jean	null	1300.00	2007-09-28	h

nombres de lignes : 4

Listing 40 – les personnes dont le numéro n'est pas compris entre 2 et 5

```
1 select   *
2   from personne
3   where numero < 2 or numero > 5
4 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h

nombres de lignes : 1

Listing 41 – les personnes dont le numéro n'est pas compris entre 2 et 5 : opérateur **not between**

```
1 select   *
2   from personne
3   where numero not between 2 and 5
4 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h

nombres de lignes : 1

Listing 42 – les personnes entrées en avril 2007 : : opérateur **between**

```
1 select   *
2   from personne
3   where dateEntree between '2007-04-01'
4                        and '2007-04-30'
5 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f

nombres de lignes : 1

19.5.4 Exemple 4 : is null, is not null

Listing 43 – les personnes dont la ville n'est pas renseignée : opérateur **is null**

```
1 select   *
```

```
2   from personne
3   where ville is null
4 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	5	Rigole	Jean	null	1300.00	2007-09-28	h

nombres de lignes : 1

Listing 44 – les personnes dont la ville est renseignée : opérateur **`is not null`**

```
1 select  *
2   from personne
3   where ville is not null
4 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h

nombres de lignes : 4

20 Renommage

L'opération de renommage est utilisée afin de rebaptiser des tables ou des colonnes afin de lever une ambiguïté lorsque, dans une requête, un même identificateur de colonne est utilisé plusieurs fois, ou un même nom de table apparait plusieurs fois (cf. le cas de l'auto-jointure : section ?? en page 56).

20.1 Syntaxe : alias de table

L'alias de table est un nouveau nom donné à une table au sein d'une requête, et valable seulement dans la requête.

Renommage de table en SQL - alias de table

$\rho_S(R)$

```
1 select  *
2   from R S
3   ;
```

où :

- R : table d'origine
- S : alias de la table

Après qu'un alias a été défini dans une requête, on ne peut plus faire référence au nom originel de la table dans la requête.

L'alias de table est utilisé essentiellement dans les jointures où des colonnes de noms identiques proviennent de table différentes : un alias court, d'une seule lettre par exemple, permet de rendre la requête plus lisible.

20.2 Syntaxe : alias de colonne

L'alias de colonne permet de donner un nom plus lisible à une colonne dans le résultat.

Renommage de colonne en SQL - alias de colonne

$\rho_{x1->nx1,x2->nx2,...,xn->nxn}(R)$

```
select x1 AS nx1 ,x2 AS nx2,...,xn AS nxn
  from R
  ;
```

où :

- *x1,x2,...,xn* : noms des attributs
- *nx1,nx2,...,nxn* : nouveaux noms donnés aux attributs
- *R* : table d'origine

L'alias de colonne est donnée dans la clause **`select`** et ne peut être utilisé qu'après la clause **`where`**.

20.3 Exemples

20.3.1 Exemple1 : alias de colonne

personne	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h
	5	Rigole	Jean	null	1300.00	2007-09-28	h

nombres de lignes : 5

Listing 45 – Exemple SQL d'alias de colonne dans une projection

```
select nom, prenom, salaire as salaireMensuel
  from personne;
```

resultat	nom	prenom	salaireMensuel
	Dupont	Max	1000.00
	Durand	Tim	1500.00
	Lambert	Betty	1350.00
	Bradford	Jean	1250.00
	Rigole	Jean	1300.00

nombres de lignes : 5

Listing 46 – les nom et prenom des personnes qui habite Arras

```
1 select A.nom, A.prenom, A.dateEntree as entreLe
2   from personne A
3   where A.ville = 'Arras'
4 ;
```

resultat	nom	prenom	entreLe
	Dupont	Max	2007-01-01
	Bradford	Jean	2007-09-07

nombres de lignes : 2

21 Union : `union`

L'opération d'union en SQL regroupe les lignes de 2 requêtes (ou plus) qui doivent être union-compatibles (même nombre de colonnes et même type de donnée pour chaque colonne).

21.1 Syntaxe

Union en SQL - sans doublons

$(R \cup S)$

```
requete1
union
requete2
;
```

Union en SQL - les doublons sont conservés

```
requete1
union all
requete2
;
```

où :

- *requete1* et *requete2* : toute requete SQL **`select`** valide, les 2 requêtes devant être union-compatibles
- **`union`** : mot-clef de l'union de 2 requêtes
- **`all`** : optionnel, permet de conserver les doublons de lignes

21.2 Exemples

21.2.1 Exemple 1

R	A	B	C
	a	b	c
	d	a	f
	c	b	d

nombres de lignes : 3 -

S	A	B	C
	b	g	a
	d	a	f

nombres de lignes : 2

R et S sont union-compatibles.

Listing 47 – Union distinct ($(R \cup S)$)

```
1 select A,B,C
2   from R
3 union
4 select A,B,C
5   from S
6 ;
```

resultat	A	B	C
	a	b	c
	d	a	f
	c	b	d
	b	g	a

nombres de lignes : 4

Listing 48 – **union all**

```
1 select A,B,C
2   from R
3 union all
4 select A,B,C
5   from S
6 ;
```

resultat	A	B	C
	a	b	c
	d	a	f
	c	b	d
	b	g	a
	d	a	f

nombres de lignes : 5

21.2.2 Exemple 2

personne	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h
	5	Rigole	Jean	null	1300.00	2007-09-28	h

nombres de lignes : 5 -

Listing 49 – les personnes entrées en janvier et celles habitant Arras

```
1 select  *
2   from personne
3   where dateEntree between '2007-01-01'
```

```
4                               and '2007-01-31'
5 union
6 select  *
7   from personne
8   where ville = 'arras'
9 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h

nombres de lignes : 2

Listing 50 – les personnes entrées en janvier et celles habitant Arras

```
1 select  *
2   from personne
3   where dateEntree between '2007-01-01'
4                               and '2007-01-31'
5 union all
6 select  *
7   from personne
8   where ville = 'arras'
9 ;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h

nombres de lignes : 3

22 Différence

L'opération de différence en SQL reprend les lignes d'une 1ère requête qui ne se trouvent pas dans une 2ème requête.

22.1 Syntaxe

Différence en SQL

$(R - S)$

```
1 requete1
2 except
3 requete2
4 ;
```

où :

- *requete1* et *requete2* : toute requete SQL **`select`** valide, les 2 requêtes devant être union-compatibles
- **`except`** : mot-clef de la différence (on trouve parfois **`minus`**)

Le mot-clef **`except`** n'est pas implanté dans tous les SGBDR : il est alors nécessaire d'utiliser une sous-requête pour mettre en oeuvre cette notion (voir section 32.6 en page 85).

23 Produit cartésien : `from` + `cross join`

Le produit cartésien permet la construction de toutes les combinaisons possibles des lignes de 2 tables.

Attention

Le résultat du produit cartésien n'a pas de sens : on y associe en effet des colonnes de tables différentes sans conserver de lien sémantique entre elles.

23.1 Syntaxe

Produit cartésien en SQL - SQL92

$(R \times S)$

```
1 select *
2   from R
3     cross join S
4 ;
```

où :

- R et S : tables objets du produit cartésien
- **`cross join`** : le mot-clef SQL92 du produit cartésien

Produit cartésien en SQL - SQL89

$(R \times S)$

```
1 select  *
2   from R,S
3 ;
```

où :

- R et S : tables objets du produit cartésien

Danger

Éviter d'utiliser cette dernière notation qui ne met pas en évidence le choix d'un produit cartésien dans la clause **`from`**, et qui, sans clause de sélection, peut être un simple oubli aux conséquences non négligeables...

23.2 Exemples

23.2.1 Exemple 1

R	A	B	C
	a	b	c
	d	a	f
	c	b	d

nombres de lignes : 3 -

V	A	D
	b	g
	d	a

nombres de lignes : 2

Listing 51 – Exemple de produit cartésien $(R \times V)$ (SQL92)

```
1 select *
2   from R
3        cross join V
4 ;
```

résultat	A	B	C	A	D
	a	b	c	b	g
	a	b	c	d	a
	d	a	f	b	g
	d	a	f	d	a
	c	b	d	b	g
	c	b	d	d	a

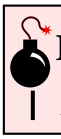
nombres de lignes : 6

Listing 52 – Exemple de produit cartésien $(R \times S)$ (SQL89)

```
1 select  *
2   from R, V
3 ;
```

résultat	A	B	C	A	D
	a	b	c	b	g
	a	b	c	d	a
	d	a	f	b	g
	d	a	f	d	a
	c	b	d	b	g
	c	b	d	d	a

nombres de lignes : 6



Danger

Attention au volume de données produit par cette opération...

23.2.2 Exemple 2

etudiant	numero	nom	prenom	dateNaissance
	1	Dupont	Jacques	1990-01-15
	9	Durand	Pierre	1999-07-20
	23	Lambert	Paul	1994-12-03
	34	Durand	Jacques	1994-12-15
	105	Lambert	Paul	1995-10-20

nombres de lignes : 5

inscrire	code	numero
	INFO	1
	INFO	9
	INFO	34
	MATH	1
	MATH	23
	MATH	34
	PHCH	1

nombres de lignes : 7

Listing 53 – Produit cartésien : $(etudiant \times inscrire)$ (SQL92)

```
1 select  *
2   from etudiant
3        cross join inscrire
4 ;
```

formation	numero	nom	prenom	dateNaissance	code	numero
	1	Dupont	Jacques	1990-01-15	INFO	1
	9	Durand	Pierre	1999-07-20	INFO	1
	23	Lambert	Paul	1994-12-03	INFO	1
	34	Durand	Jacques	1994-12-15	INFO	1
	105	Lambert	Paul	1995-10-20	INFO	1
	1	Dupont	Jacques	1990-01-15	INFO	9
	9	Durand	Pierre	1999-07-20	INFO	9
	23	Lambert	Paul	1994-12-03	INFO	9
	34	Durand	Jacques	1994-12-15	INFO	9
	105	Lambert	Paul	1995-10-20	INFO	9
	1	Dupont	Jacques	1990-01-15	INFO	34
	9	Durand	Pierre	1999-07-20	INFO	34
	23	Lambert	Paul	1994-12-03	INFO	34
	34	Durand	Jacques	1994-12-15	INFO	34
	105	Lambert	Paul	1995-10-20	INFO	34
	1	Dupont	Jacques	1990-01-15	MATH	1
	9	Durand	Pierre	1999-07-20	MATH	1
	23	Lambert	Paul	1994-12-03	MATH	1
	34	Durand	Jacques	1994-12-15	MATH	1
	105	Lambert	Paul	1995-10-20	MATH	1
	1	Dupont	Jacques	1990-01-15	MATH	23
	9	Durand	Pierre	1999-07-20	MATH	23
	23	Lambert	Paul	1994-12-03	MATH	23
	34	Durand	Jacques	1994-12-15	MATH	23
	105	Lambert	Paul	1995-10-20	MATH	23
	1	Dupont	Jacques	1990-01-15	MATH	34
	9	Durand	Pierre	1999-07-20	MATH	34
	23	Lambert	Paul	1994-12-03	MATH	34
	34	Durand	Jacques	1994-12-15	MATH	34
	105	Lambert	Paul	1995-10-20	MATH	34
	1	Dupont	Jacques	1990-01-15	PHCH	1
	9	Durand	Pierre	1999-07-20	PHCH	1
	23	Lambert	Paul	1994-12-03	PHCH	1
	34	Durand	Jacques	1994-12-15	PHCH	1
	105	Lambert	Paul	1995-10-20	PHCH	1

nombres de lignes : 35

Attention

Ce résultat n'a pas de sens : il fournit toutes les possibilités de combinaison de chaque étudiant avec toutes les inscriptions...

24 Intersection

L'opération d'intersection en SQL reprend les lignes d'une 1ère requête qui se trouvent aussi dans une 2ème requête.

24.1 Syntaxe SQL

Intersection en SQL

$(R \cap S)$

```
1 requete1
2 intersect
3 requete2
4   ;
```

où :

- *requete1* et *requete2* : toute requete SQL **`select`** valide, les 2 requêtes devant être union-compatibles

Le mot-clef **`intersect`** n'est pas implanté dans tous les SGBDR : il est alors nécessaire d'utiliser une sous-requête pour mettre en oeuvre cette notion (voir section 32.7 en page 87).

25 Jointure interne : `from` + `inner join`

25.1 Théta-jointure

25.1.1 Syntaxe

Jointure interne (thêta jointure) en SQL - SQL92

$(R \underset{Q}{\bowtie} S)$

```
1 select  *
2   from R
3      inner join S
4         on Q
5 ;
```

Jointure interne (thêta jointure) en SQL - SQL89

$(R \underset{Q}{\bowtie} S)$

```
select  *
  from R, S
  where Q
  ;
```

où :

- R et S : tables objets de la jointure
- **inner join** ou **join** : mot-clé de la jointure interne (SQL92)S
- Q : critère de jointure utilisant des opérateurs de comparaison

Lorsque la(ou les) colonne(s) de jointure porte(nt) le même nom dans les 2 tables jointures, il est possible d'utiliser une notation plus simple en SQL92, et qui supprime le doublon de colonne(s) de jointure :

Jointure interne (equi-jointure) en SQL - SQL92

$(R \underset{(x1,...,xn)}{\bowtie} S)$

```
select  *
  from R
     inner join S
       using(x1,...,xn)
  ;
```

Cette syntaxe est se rapproche de la jointure naturelle mais elle est plus sûre.

Jointure interne (equi-jointure) en SQL - SQL89

$\sigma_{R.x1=S.x1 and R.x2=S.x2 ... and R.xn=S.xn}(R \times S)$

```
select  *
  from R, S
  where (R.x1 = S.x1 and R.x2 = S.x2 ... and R.xn = S.xn
     )
  ;
```

où :

- R et S : tables objets de la jointure
- *x1,...,xn* : la ou les colonnes constituant le critère de jointure ; l'opérateur d'égalité est utilisé

25.1.2 Exemples

Exemple : 1

R	A	B	C
	a	b	c
	d	a	f
	c	b	d

nombres de lignes : 3 -

S	A	B	C
	b	g	a
	d	a	f

nombres de lignes : 2

Listing 54 – Exemple de jointure interne d'égalité ($(R \underset{\text{R.B = S.A}}{\bowtie} S)$)

```
1 select  *
2   from R
3        inner join S
4                on R.B = S.A
5 ;
```

resultat	A	B	C	A	B	C
	a	b	c	b	g	a
	c	b	d	b	g	a

nombres de lignes : 2

Exemple : 2

pilote	numpil	nompil	vilpil	datnaispil	salairepil
	1	William	Nice	1980-06-29	16000.00
	2	Peter	Nice	1970-01-10	18000.00
	3	Max	Paris	1975-04-03	12000.00
	4	Scott	Marseille	1981-08-29	12000.00
	5	Mandy	Marseille	1985-05-16	16000.00
	6	John	Nice	1982-06-11	20000.00
	7	Bill	Marseille	1980-06-29	18000.00
	8	Camille	Paris	1989-10-06	16000.00
	9	boule	Paris	1980-06-29	17000.00

nombres de lignes : 9

vol	numvol	numpil	numav	departvol	dureevol	vildepvol	vilarrvol
	1002	3	104	2015-09-01 09 :00 :00.0	20	Marseille	Nice
	1003	4	105	2015-09-01 15 :30 :00.0	65	Paris	Nice
	1004	2	101	2015-09-01 16 :10 :00.0	60	Paris	Marseille
	1005	3	101	2015-09-02 10 :50 :00.0	45	Marseille	Lyon
	1006	4	107	2015-09-03 11 :10 :00.0	20	Nice	Marseille
	1007	5	101	2015-09-03 15 :00 :00.0	45	Paris	Lyon
	1008	6	101	2015-09-04 09 :30 :00.0	45	Lyon	Marseille
	1009	7	101	2015-09-05 08 :00 :00.0	60	Paris	Marseille
	1010	2	108	2015-09-05 17 :00 :00.0	60	Nice	Paris

nombres de lignes : 9

Listing 55 – les pilotes et leurs vols ($(pilote \underset{\text{pilote.numpil = vol.numpil}}{\bowtie} vol)$)

```
1 select  *
2   from pilote
3           inner join vol
4              on pilote.numpil = vol.numpil
5 ;
```

resultat	numpil	nompil	vilpil	datnaispil	salairepil	numvol	numpil	numav	departvol	dureevol	vildepvol	vilarrvol
	3	Max	Paris	1975-04-03	12000.00	1002	3	104	2015-09-01 09 :00 :00.0	20	Marseille	Nice
	4	Scott	Marseille	1981-08-29	12000.00	1003	4	105	2015-09-01 15 :30 :00.0	65	Paris	Nice
	2	Peter	Nice	1970-01-10	18000.00	1004	2	101	2015-09-01 16 :10 :00.0	60	Paris	Marseille
	3	Max	Paris	1975-04-03	12000.00	1005	3	101	2015-09-02 10 :50 :00.0	45	Marseille	Lyon
	4	Scott	Marseille	1981-08-29	12000.00	1006	4	107	2015-09-03 11 :10 :00.0	20	Nice	Marseille
	5	Mandy	Marseille	1985-05-16	16000.00	1007	5	101	2015-09-03 15 :00 :00.0	45	Paris	Lyon
	6	John	Nice	1982-06-11	20000.00	1008	6	101	2015-09-04 09 :30 :00.0	45	Lyon	Marseille
	7	Bill	Marseille	1980-06-29	18000.00	1009	7	101	2015-09-05 08 :00 :00.0	60	Paris	Marseille
	2	Peter	Nice	1970-01-10	18000.00	1010	2	108	2015-09-05 17 :00 :00.0	60	Nice	Paris

nombres de lignes : 9

Listing 56 – les pilotes et leurs vols ($(pilote \underset{\text{pilote.numpil = vol.numpil}}{\bowtie} vol)$)

```
1 select  *
2   from pilote
3           inner join vol
4              using (numpil)
5 ;
```

resultat	numpil	nompil	vilpil	datnaispil	salairepil	numvol	numav	departvol	dureevol	vildepvol	vilarrvol
	3	Max	Paris	1975-04-03	12000.00	1002	104	2015-09-01 09 :00 :00.0	20	Marseille	Nice
	4	Scott	Marseille	1981-08-29	12000.00	1003	105	2015-09-01 15 :30 :00.0	65	Paris	Nice
	2	Peter	Nice	1970-01-10	18000.00	1004	101	2015-09-01 16 :10 :00.0	60	Paris	Marseille
	3	Max	Paris	1975-04-03	12000.00	1005	101	2015-09-02 10 :50 :00.0	45	Marseille	Lyon
	4	Scott	Marseille	1981-08-29	12000.00	1006	107	2015-09-03 11 :10 :00.0	20	Nice	Marseille
	5	Mandy	Marseille	1985-05-16	16000.00	1007	101	2015-09-03 15 :00 :00.0	45	Paris	Lyon
	6	John	Nice	1982-06-11	20000.00	1008	101	2015-09-04 09 :30 :00.0	45	Lyon	Marseille
	7	Bill	Marseille	1980-06-29	18000.00	1009	101	2015-09-05 08 :00 :00.0	60	Paris	Marseille
	2	Peter	Nice	1970-01-10	18000.00	1010	108	2015-09-05 17 :00 :00.0	60	Nice	Paris

nombres de lignes : 9

Le doublon de colonne de jointure a été supprimé.

Exemple : 3

etudiant	numero	nom	prenom	dateNaissance
	1	Dupont	Jacques	1990-01-15
	9	Durand	Pierre	1999-07-20
	23	Lambert	Paul	1994-12-03
	34	Durand	Jacques	1994-12-15
	105	Lambert	Paul	1995-10-20

nombres de lignes : 5 -

inscrire	code	numero
	INFO	1
	INFO	9
	INFO	34
	MATH	1
	MATH	23
	MATH	34
	PHCH	1

nombres de lignes : 7

formation	code	libelle
	INFO	Informatique
	MATH	Math,matiques
	PHCH	Physique-Chimie

nombres de lignes : 3

Listing 57 – Les inscriptions des étudiants aux formation

```
select nom, prenom, libelle
  from (etudiant
          inner join inscrire using (numero))
          inner join formation using (code)
;
```

resultat	nom	prenom	libelle
	Dupont	Jacques	Informatique
	Durand	Pierre	Informatique
	Durand	Jacques	Informatique
	Dupont	Jacques	Math,matiques
	Lambert	Paul	Math,matiques
	Durand	Jacques	Math,matiques
	Dupont	Jacques	Physique-Chimie

nombres de lignes : 7

25.2 Jointure naturelle

La jointure naturelle utilise, de manière transparente, comme critère de jointure l'égalité appliquée aux colonnes qui portent le même nom dans les 2 tables.

25.2.1 Syntaxe

Jointure naturelle en SQL - SQL92

$(R \bowtie S)$

```
select  *
  from R
     natural join S
  ;
```

où :

- R et S : tables objets de la jointure
- **`natural join`** : mot-clé de la jointure naturelle

Attention à l'utilisation de cette forme de jointure

- si aucun nom d'attribut n'est commun aux 2 relations, on obtient un produit cartésien
- si des attributs portent le même nom sans toutefois avoir le même sens, on obtient une jointure incohérente...

25.2.2 Exemple

Listing 58 – les inscription des étudiants (natural join)

```
select  *
  from etudiant
          natural join inscrire
;
```

resultat	numero	nom	prenom	dateNaissance	code
	1	Dupont	Jacques	1990-01-15	INFO
	9	Durand	Pierre	1999-07-20	INFO
	34	Durand	Jacques	1994-12-15	INFO
	1	Dupont	Jacques	1990-01-15	MATH
	23	Lambert	Paul	1994-12-03	MATH
	34	Durand	Jacques	1994-12-15	MATH
	1	Dupont	Jacques	1990-01-15	PHCH

nombres de lignes : 7

Exemple Jointure naturelle

Exemple : 1

resultat	A	B	C
	a	b	c
	d	a	f
	c	b	d

nombres de lignes : 3

resultat	A	B	C
	a	b	c
	d	a	f
	c	b	d

nombres de lignes : 3

Listing 59 – jointure naturelle

```
1 select  *
2   from R
3        natural join S
4 ;
```

resultat	A	B	C
	d	a	f

nombres de lignes : 1

Exemple : 2

etudiant	numero	nom	prenom	dateNaissance
	1	Dupont	Jacques	1990-01-15
	9	Durand	Pierre	1999-07-20
	23	Lambert	Paul	1994-12-03
	34	Durand	Jacques	1994-12-15
	105	Lambert	Paul	1995-10-20

nombres de lignes : 5

inscrire	code	numero
	INFO	1
	INFO	9
	INFO	34
	MATH	1
	MATH	23
	MATH	34
	PHCH	1

nombres de lignes : 7

formation	code	libelle
	INFO	Informatique
	MATH	Math,matiques
	PHCH	Physique-Chimie

nombres de lignes : 3

Listing 60 – Les inscriptions des étudiants aux formation

```
select nom, prenom, libelle
  from (etudiant
          natural join inscrire)
          natural join formation
;
```

resultat	nom	prenom	libelle
	Dupont	Jacques	Informatique
	Durand	Pierre	Informatique
	Durand	Jacques	Informatique
	Dupont	Jacques	Math,matiques
	Lambert	Paul	Math,matiques
	Durand	Jacques	Math,matiques
	Dupont	Jacques	Physique-Chimie

nombres de lignes : 7

25.3 Exemple Non-equi-jointure

La non équi-jointure est une thêta-jointure donc le critère de jointure comporte un opérateur autre que l'égalité.

25.3.1 Exemple 1

pilote	numpil	nompil	vilpil	datnaispil	salairepil
	1	William	Nice	1980-06-29	16000.00
	2	Peter	Nice	1970-01-10	18000.00
	3	Max	Paris	1975-04-03	12000.00
	4	Scott	Marseille	1981-08-29	12000.00
	5	Mandy	Marseille	1985-05-16	16000.00
	6	John	Nice	1982-06-11	20000.00
	7	Bill	Marseille	1980-06-29	18000.00
	8	Camille	Paris	1989-10-06	16000.00
	9	boule	Paris	1980-06-29	17000.00

nombres de lignes : 9

vol	numvol	numpil	numav	departvol	dureevol	vildepvol	vilarrvol
	1002	3	104	2015-09-01 09 :00 :00.0	20	Marseille	Nice
	1003	4	105	2015-09-01 15 :30 :00.0	65	Paris	Nice
	1004	2	101	2015-09-01 16 :10 :00.0	60	Paris	Marseille
	1005	3	101	2015-09-02 10 :50 :00.0	45	Marseille	Lyon
	1006	4	107	2015-09-03 11 :10 :00.0	20	Nice	Marseille
	1007	5	101	2015-09-03 15 :00 :00.0	45	Paris	Lyon
	1008	6	101	2015-09-04 09 :30 :00.0	45	Lyon	Marseille
	1009	7	101	2015-09-05 08 :00 :00.0	60	Paris	Marseille
	1010	2	108	2015-09-05 17 :00 :00.0	60	Nice	Paris

nombres de lignes : 9

25.3.2 Exemple 2

Listing 61 – les pilotes et les vols avec vilpil différent de vildepvol (projection pour limiter le nombre de colonnes)

```
1 select  pilote.numpil, nompil, vilpil, numvol, vol.numpil,
    vildepvol
2   from pilote
3        inner join vol
4                on pilote.numpil = vol.numpil
5                    and vilpil <> vildepvol
6 ;
```

resultat	numpil	nompil	vilpil	numvol	numpil	vildepvol
	3	Max	Paris	1002	3	Marseille
	4	Scott	Marseille	1003	4	Paris
	2	Peter	Nice	1004	2	Paris
	3	Max	Paris	1005	3	Marseille
	4	Scott	Marseille	1006	4	Nice
	5	Mandy	Marseille	1007	5	Paris
	6	John	Nice	1008	6	Lyon
	7	Bill	Marseille	1009	7	Paris

nombres de lignes : 8

26 Jointure externe : `from` + `outer join`

26.0.1 Syntaxe

Jointure externe en SQL - SQL92

$(R \underset{Q}{\bowtie\!\!\!|} S)$ $(R \underset{Q}{|\!\!\!\bowtie} S)$ $(R \underset{Q}{\bowtie\!\!\!|\!\!\!\bowtie} S)$

```
1 select   *
2   from R
```

```
3         [left | right | full] outer join S
4           on Q
5   ;
```

Jointure externe en SQL - SQL92

$(R \overset{\text{(x1,...,xn)}}{\bowtie\!\!\!|} S)$ $(R \overset{\text{(x1,...,xn)}}{|\!\!\!\bowtie} S)$ $(R \overset{\text{(x1,...,xn)}}{\bowtie\!\!\!|\!\!\!\bowtie} S)$

```
1 select  *
2   from R
3         [left | right | full] outer join S
4           using (x1, x2, ..., xn)
5 ;
```

où :

- R et S : tables objets de la jointure
- **left**, **right** et **full** : indique si on conserve les lignes de la table de gauche, de droite ou des 2
- **outer join** : mot-clé de la jointure externe
- Q : critère de jointure utilisant des opérateurs de comparaison
- *x1, x2, xn* : colonne(s) de jointure (égalité)

La valeur des attributs des n-uplets de la relation ne satisfaisant pas critère de jointure ont pour valeur **null**.

26.0.2 Exemples

Exemple : 1

R	A	B	C
	a	b	c
	d	a	f
	c	b	d

nombres de lignes : 3 -

V	A	D
	b	g
	d	a

nombres de lignes : 2

Listing 62 – R left outer join S

```
1 select  *
2   from R
3        left outer join S
4                   on R.B = S.A
5 ;
```

resultat	A	B	C	A	B	C
	a	b	c	b	g	a
	d	a	f	null	null	null
	c	b	d	b	g	a

nombres de lignes : 3

Exemple : 2

etudiant	numero	nom	prenom	dateNaissance
	1	Dupont	Jacques	1990-01-15
	9	Durand	Pierre	1999-07-20
	23	Lambert	Paul	1994-12-03
	34	Durand	Jacques	1994-12-15
	105	Lambert	Paul	1995-10-20

nombres de lignes : 5

inscrire	code	numero
	INFO	1
	INFO	9
	INFO	34
	MATH	1
	MATH	23
	MATH	34
	PHCH	1

nombres de lignes : 7

Listing 63 – tous les etudiants et éventuellement les inscriptions

```
1 select  *
2   from etudiant
3          left outer join inscrire
4             on etudiant.numero = inscrire.numero
5 ;
```

resultat	numero	nom	prenom	dateNaissance	code	numero
	1	Dupont	Jacques	1990-01-15	INFO	1
	1	Dupont	Jacques	1990-01-15	MATH	1
	1	Dupont	Jacques	1990-01-15	PHCH	1
	9	Durand	Pierre	1999-07-20	INFO	9
	23	Lambert	Paul	1994-12-03	MATH	23
	34	Durand	Jacques	1994-12-15	INFO	34
	34	Durand	Jacques	1994-12-15	MATH	34
	105	Lambert	Paul	1995-10-20	null	null

nombres de lignes : 8

Listing 64 – tous les etudiants et éventuellement les inscriptions (using)

```
select  *
  from etudiant
          left outer join inscrire
             using (numero)
;
```

resultat	numero	nom	prenom	dateNaissance	code
	1	Dupont	Jacques	1990-01-15	INFO
	1	Dupont	Jacques	1990-01-15	MATH
	1	Dupont	Jacques	1990-01-15	PHCH
	9	Durand	Pierre	1999-07-20	INFO
	23	Lambert	Paul	1994-12-03	MATH
	34	Durand	Jacques	1994-12-15	INFO
	34	Durand	Jacques	1994-12-15	MATH
	105	Lambert	Paul	1995-10-20	null

nombres de lignes : 8

27 Cas particuliers de jointures

27.1 Auto-jointure

L'auto-jointure est un cas de jointure d'une relation avec elle-même, quelle soit interne ou externe.

Les noms des attributs doivent être préfixés du nom de relation afin d'éviter toute ambiguïté ou bien être renommés.

Listing 65 – autojointure en SQL

```
select  *
  from R
     inner join R A
           on Q
  ;
```

où :

- *R* : table objet de l'auto- jointure
- *A* : un alias sur R
- *Q* : critère de jointure

Exemple :

à partir de de la table 'Personne', produire un résultat comportant tous les binômes possibles de personnes d'une même ville :

Listing 66 – les binomes de personnes de la même ville)

```
1 select P.nom, P.prenom, B.nom as nomBinome, B.prenom as
    prenomBinome
2   from personne P
3        inner join  personne B
4           on P.ville = B.ville
5 ;
```

resultat	nom	prenom	nomBinome	prenomBinome
	Dupont	Max	Dupont	Max
	Bradford	Jean	Dupont	Max
	Durand	Tim	Durand	Tim
	Lambert	Betty	Lambert	Betty
	Dupont	Max	Bradford	Jean
	Bradford	Jean	Bradford	Jean

nombres de lignes : 6

Une amélioration sera à apporter : en effet, chaque personne formera un binôme avec lui-même...

Listing 67 – les binomes de personne

```
1 select P.nom, P.prenom, B.nom as nomBinome, B.prenom as
    prenomBinome
2   from personne P
3        inner join  personne B
4           on P.ville = B.ville
5              and P.numero <> B.numero
6 ;
```

resultat	nom	prenom	nomBinome	prenomBinome
	Bradford	Jean	Dupont	Max
	Dupont	Max	Bradford	Jean

nombres de lignes : 2

Une amélioration encore : dupont-durant est le même binôme que durant-dupont...

Listing 68 – les binomes de personne

```sql
select P.nom, P.prenom, B.nom as nomBinome, B.prenom as
    prenomBinome
  from personne P
       inner join  personne B
          on P.ville = B.ville
             and P.numero <> B.numero
             and P.nom < B.nom
;
```

resultat	nom	prenom	nomBinome	prenomBinome
	Bradford	Jean	Dupont	Max

nombres de lignes : 1

27.2 Semi-jointure

La semi-jointure est une jointure à laquelle est appliquée une projection pour ne conserver que les colonnes d'une des 2 tables.

Il n'existe pas d'opérateur spécifique pour cette forme de jointure : une simple projection permet de conserver les colonnes d'une des 2 tables.

Listing 69 – Semi-jointure gauche en SQL ($(R \ltimes S)$)

```sql
select R.*
  from R
     inner join S
             on Q
  ;
```

où :

- R et S : tables objets de la jointure
- Q : critère de jointure

Listing 70 – semi-jointure gauche (avec suppression des doublons : plusieurs inscription par étudiant...)

```sql
select distinct etudiant.*
  from etudiant
       inner join inscrire
            using (numero)
;
```

resultat	numero	nom	prenom	dateNaissance
	1	Dupont	Jacques	1990-01-15
	9	Durand	Pierre	1999-07-20
	34	Durand	Jacques	1994-12-15
	23	Lambert	Paul	1994-12-03

nombres de lignes : 4

27.3 Anti-jointure

L'anti-jointure est similaire à une semi-jointure, mais elle ne conserve que les lignes qui n'ont pas été jointes.

Il n'existe pas d'opérateur spécifique pour cette forme de jointure : une simple projection permet de conserver les colonnes d'une des 2 tables et une sélection permet de ne conserver que les lignes qui n'ont pas été jointes.

Listing 71 – Antijointure gauche en SQL ($(R \triangleright S)$)

```
select R.*
  from R
     left join S
             on Q
  where S.col1 is null [and S.col2 is null ...]
;
```

où :

- *R* et *S* : tables objets de la jointure
- *Q* : critère de jointure
- *col1*, *col2* : attributs de S

Listing 72 – anti-jointure gauche ($(R \triangleleft S)$)

```
select pilote.*
  from pilote
     left join vol
             using (numpil)
  where vol.numvol is null
;
```

resultat	numpil	nompil	vilpil	datnaispil	salairepil
	1	William	Nice	1980-06-29	16000.00
	8	Camille	Paris	1989-10-06	16000.00
	9	boule	Paris	1980-06-29	17000.00

nombres de lignes : 3

La colonne 'numvol' étant la clé primaire de 'vol', cette seule comparaison suffit à déduire qu'il n'y avait pas de vol pour ce pilote et que cette ligne n'avais pas été jointe.

28 Division

La division n'est pas implanté dans les SGBDR : il est nécessaire d'utiliser une sous-requête pour mettre en oeuvre cette notion (voir section 32.8 en page 89).

29 Agrégats

L'agrégat consiste à appliquer une fonction statistique à une ou plusieurs colonnes :

- de manière globale pour tous les lignes d'une table,
- ou en définissant une liste de colonnes de regroupement des lignes (calcul de sous-totaux par valeurs différentes des colonnes de regroupement).

Table 7 – Fonctions statistiques de l'agrégat SQL

count	compter les valeurs d'une colonne
sum	calculer la somme des valeurs d'une colonne
avg	calculer la moyenne (en anglais : *average*) des valeurs d'une colonne
min	déterminer la plus petite des valeurs d'une colonne
max	déterminer la plus grande des valeurs d'une colonne

Les fonctions s'appliquent à *toutes les valeurs non nulles* d'une colonne :

```
1 fonction(colonne)
```

Elles peuvent également s'appliquer aux *différentes valeurs* d'une colonne :

```
1 fonction(distinct colonne)
```

où :

- *fonction* : une des fonctions d'agrégat
- *colonne* : colonne disponible

La fonction **count** peut utiliser la valeur * pour signifier 'les lignes' de la table :

```
1 count(*)
```

29.1 Syntaxe générale

agrégats en SQL

```
1 select  [a1, a2, ..., an]
2         fn1(x1), fn2(x2), ..., fnN(xn)
3   from table(s)
4   [where Q1]
5   [group by a1, a2, ..., an]
6   [having Q2]
7 ;
```

où :

- *a1, a2, ..., an* sont les colonnes qui seront renvoyées
- *fn1, fn2, ...*, fnN sont des fonctions d'agrégation appliquées aux colonnes *x1, x2, ..., xn*
- *Q1* est une condition de sélection s'appliquant aux lignes lues
- **`group by`** *a1, a2, ..., an* : mot-clef SQL précisant les colonnes de regroupement
- **`having`** *Q2* : mot-clef SQL précisant la condition de sélection s'appliquant généralement à des valeurs agrégées

29.2 Agrégat global

Agrégat global en SQL

$\mathcal{G}_{fonction(x1)}(R)$

```
select fn1(x1)[, fn2(x21), ..., fnN(xn)]
  from table(s)
  [where selection]
  ;
```

où :

- *fn1, fn2, ...*, fnN sont des fonctions d'agrégation appliquées à des colonnes
- *x1, x2, ...* : colonnes auxquelles s'appliquent les fonctions
- *table(s)* : table(s)
- *selection* : critère de sélection des lignes à agréger

29.2.1 Exemples

Exemple : 1

W	X	Y	N
	a	b	10
	a	c	30
	b	b	20
	c	b	20

nombres de lignes : 4

Listing 73 – Compter les valeurs ($\mathcal{G}_{count(X)}(R)$)

```
select count(X)
  from W
;
```

resultat	count(X)
	4

nombres de lignes : 1

Listing 74 – Compter les valeurs distinctes

```
1 select count(distinct X)
2   from W
3 ;
```

resultat	count(distinct X)
	3

nombres de lignes : 1

Listing 75 – Somme ($\mathcal{G}_{Sum(N)->total}(R)$)

```
1 select sum(N) AS total
2   from W
3 ;
```

resultat	total
	80

nombres de lignes : 1

personne	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h
	5	Rigole	Jean	null	1300.00	2007-09-28	h

nombres de lignes : 5

Listing 76 – Somme des salaires

```
1 select sum(salaire) as totalDesSalaires,
2        count(*) as nombrePersonnes
3   from personne
4 ;
```

resultat	totalDesSalaires	nombrePersonnes
	6400.00	5

nombres de lignes : 1

Listing 77 – Salaire le plus élevé et le plus bas

```
1 select max(salaire) as plusEleve,
2        min(salaire) as plusBas
3   from personne
4 ;
```

resultat	plusEleve	plusBas
	1500.00	1000.00

nombres de lignes : 1

Listing 78 – Somme des salaires des personnes d'Arras

```
1 select sum(salaire) as totalDesSalaires,
2        count(*) as nombrePersonnes
3   from personne
4   where ville = 'arras'
5 ;
```

resultat	totalDesSalaires	nombrePersonnes
	2250.00	2

nombres de lignes : 1

29.3 Agrégat par valeur de regroupement : group by

Agrégat par regroupement en SQL

$_{a1,a2,...,an}\mathcal{G}_{fonction(x1)}(R)$

```
1 select  [a1, a2, ..., an]
2         fn1(x1), fn2(x2), ..., fnN(xn)
3   from table(s)
4   [where selection]
5   group by a1, a2, ..., an
6 ;
```

où :

- *a1, a2, ..., an* sont les colonnes qui seront renvoyées (la liste correspond généralement à celle du GROUP BY, mais peut-être vide dans certains cas où on souhaite obtenir simplement les valeurs agrégées)
- *fn1, fn2, ..., fnN* sont des fonctions d'agrégation appliquées aux colonnes *x1, x2, ..., xn*
- *R* : table(s)
- **group by** *a1, a2, ..., an* : mot-clef SQL précisant les colonnes de regroupement
- *selection* : critère de sélection des lignes à agréger

29.3.1 Exemples

Exemple : 1

Listing 79 – W

```
select  *
  from W
;
```

resultat	X	Y	N
	a	b	10
	a	c	30
	b	b	20
	c	b	20

nombres de lignes : 4

Listing 80 – Compter le nombre de valeurs de la colonne X (${}_X\mathcal{G}_{count(X)}(R)$)

```
select X , count(X)
  from W
  group by X
;
```

resultat	X	count(X)
	a	2
	b	1
	c	1

nombres de lignes : 3

Listing 81 – Compter le nombre de valeurs de la colonne X avec un alias

```
select X , count(X) as Nombre
  from W
  group by X
;
```

resultat	X	Nombre
	a	2
	b	1
	c	1

nombres de lignes : 3

Listing 82 – Somme des valeurs de la colonne N par regroupement de X ${}_X\mathcal{G}_{Sum(N)->Total}(R)$

```
select X, sum(N) as Total
  from W
  group by X
;
```

resultat	X	Total
	a	40
	b	20
	c	20

nombres de lignes : 3

Listing 83 – Somme des salaires et nombre de lignes par ville et nombre de villes et nombre de villes différentes

```
select ville, sum(salaire) as totalDesSalaires,
       count(*) as nombrePersonnes,
       count(ville) as nombreVilles,
       count(distinct ville) as villesDiff
  from personne
  group by ville
;
```

resultat	ville	totalDesSalaires	nombrePersonnes	nombreVilles	villesDiff
	null	1300.00	1	0	0
	Aix	1500.00	1	1	1
	arras	2250.00	2	2	1
	Pau	1350.00	1	1	1

nombres de lignes : 4

On peut voir ci-dessus que les comptages sont différents : en effet une ligne a sa colonne vide à la valeur **null** (non renseignée).

Listing 84 – Somme des salaires par ville

```
select sum(salaire) as totalDesSalaires
  from personne
  group by ville
;
```

resultat	totalDesSalaires
	1300.00
	1500.00
	2250.00
	1350.00

nombres de lignes : 4

On peut voir ci-dessus un agrégat par ville, sans mention des villes : pas très intéressant dans un premier abord, mais ce résultat peut fournir une liste de valeurs utilisable dans une sélection avec les opérateurs **in**, **all** ou **any** associée à une sous-requête (voir section 32 en page 75).

29.4 Sélection après agrégat : having

La clause **having** est une sélection réalisée après que les agrégats sont réalisés. Elle va essentiellement utiliser des opérateurs relationnels et comparer la valeur d'une colonne agrégée à une valeur fixe (ou calculée).

Sélection après agrégat en SQL

```
select  [a1, a2, ..., an]
        fn1(x1), fn2(x2), ..., fnN(xn)
  from table(s)
  [where Q1]
  group by a1, a2, ..., an
  having Q2
;
```

où :

- *Q2* est une condition de sélection s'appliquant généralement à des valeurs agrégées

Listing 85 – Somme des salaires par ville qui ont plus de 1 personne

```
select ville, sum(salaire) as totalDesSalaires,
       count(*) as nombrePersonnes
  from personne
  group by ville
  having  (nombrePersonnes > 1)
;
```

resultat	ville	totalDesSalaires	nombrePersonnes
	arras	2250.00	2

nombres de lignes : 1

Listing 86 – Somme des salaires par ville qui ont plus de 1 personne

```
select ville, sum(salaire) as totalDesSalaires,
       count(*) as nombrePersonnes
  from personne
  group by ville
  having  count(*) > 1
;
```

resultat	ville	totalDesSalaires	nombrePersonnes
	arras	2250.00	2

nombres de lignes : 1

30 Calculs et fonctions

Les valeur retournées par une requête correspondent généralement à celles des colonnes des tables. Elles peuvent être également provenir

- de calculs arithmétiques classiques

- ou de fonctions spécifiques

pour construire des réponses adaptées aux requêtes complexes.

Les valeurs ainsi calculées peuvent être utilisées

- comme valeurs renvoyées dans la liste des colonnes retournées (**`select`**)
- ou dans des expressions logiques pour limiter le nombre de lignes retournées (**`where`**)

30.1 Calculs de valeurs d'attributs

Les opérateurs arithmétiques peuvent être utilisés pour calculer de nouvelles valeurs de colonnes. Ces dernières sont souvent renommées (alias utilisables après la clause **`where`**)

TABLE 8 – Opérateurs arithmétiques

opérateur	*signification*
+	somme
-	différence
*	produit
/	rapport
%	modulo (reste de la division euclidienne, ou entière)

Les opérateurs ont une priorités différente ; l'utilisation des parenthèses est fortement recommandé dans les calculs complexes, ou pour simplement mettre en évidence la séquence des calculs.

Listing 87 – liste des commandes avec calcul du montant et de l'écart de valeur

```
1 select idComm, (qteComm * prixComm) as montant,
2          (qteComm * (prixComm - prix)) as ecart
3   from commande
4           inner join produit
5              using (refProduit)
6 ;
```

resultat	idComm	montant	ecart
	1	15.00	5.00
	2	19.00	6.50
	3	105.60	93.60
	4	100.00	80.00

nombres de lignes : 4

30.2 Calculs à l'aide de fonctions intégrées

Attention

Les fonctions sont mises en oeuvre dans la plupart des SGBD ; il arrive cependant que des différences existent au niveau du nom d'une fonction, ou bien ses paramètres, ou encore de leur mode de calcul.

Il est indispensable de consulter la documentation du sgbd sur lequel vous allez travailler avant l'utilisation des fonctions

Voir une synthèse comparative des fonctions par SGBDcomparatif sur developpez.com

Les fonctions présentées ici sont implantées sur le SGBDR MySql (Oracle).

TABLE 9 – Fonctions mathématiques

fonction	*résultat renvoyé*
abs(expr)	Valeur absolue de l'expression
acos(expr) **asin**(expr) **atan**(expr)	Angle exprimé en radians dont le cosinus, le sinus ou la tangente est fourni
cos(expr), **sin**(expr), **tan**(expr)	cosinus, sinus ou tangente d'un angle exprimé en radians
ceiling(expr)	Plus petit entier supérieur ou égal à la valeur spécifiée.
degrees(expr)	Conversion de radians en degrés
exp(expr)	Valeur exponentielle de la valeur spécifiée (e
floor(expr)	Le plus grand entier inférieur ou égal à la valeur spécifiée
log(expr)	Logarithme naturel de la valeur spécifiée
log10(expr)	Logarithme en base 10 de la valeur spécifiée
pi()	Valeur 3.1415927....
power (x, y) **pow**(x,y)	Valeur de x élevée à la puissance y
radians(expr)	Conversion de degrés en radians
rand()	Nombre pseudo-aléatoire entre 0 et 1
round(x, y)	Arrondi de x avec y chiffres
sign(expr)	Signe de la valeur spécifiée
	(1 si positif, 0 si 0, -1 si négatif)
sqrt(expr)	Racine carrée de la valeur spécifiée

point2d	x	y
	0.0	0.0
	1.0	1.0
	0.0	2.0
	0.0	3.0

nombres de lignes : 4

Listing 88 – liste des distances entre chaque point

```
1 select A.x, A.y, B.x, B.y,
```

```
2          sqrt(pow(A.x - B.x,2)+pow(A.y - B.y,2)) as distance
3   from point2d A
4        inner join point2d B
5          on (A.x <> B.x or A.y <> B.y)
6              and abs(A.x)+abs(A.y) < abs(B.x)+abs(B.y)
7   order by A.x, A.y, B.x, B.y
8 ;
```

resultat	x	y	x	y	distance
	0.0	0.0	0.0	2.0	2.0
	0.0	0.0	0.0	3.0	3.0
	0.0	0.0	1.0	1.0	1.4142135623730951
	0.0	2.0	0.0	3.0	1.0
	1.0	1.0	0.0	3.0	2.23606797749979

nombres de lignes : 5

TABLE 10 – Fonctions de date

fonction	*résultat renvoyé*
extract(periode from uneDate)	Extrait une période à partir d'une date periode : **day**, **month**, **year**, etc. uneDate : date provenant d'une table ou calculée
datediff(date1, date2)	nombre de jours entre 2 dates
year(une date)	extrait l'année d'une date
month(une date)	extrait le mois d'une date
day(une date)	extrait le jour d'une date
date_add(uneDate, **interval** n periode)	retourne une nouvelle date à partir d'une date auquel a été ajouté un intervalle de n periodes parmi kwday, kwmonth, kwyear, etc.
date_format(date, chaine de format)	formate la date en fonction du format comportant :
	%d ou %D (jour), %m ou %M (mois), %y ou %Y (année),
	%a et %b (nom du jour et du mois abrégés), etc.

Listing 89 – liste des commandes avec calcul de la date de livraison

```
1 select idComm, dateComm, day(dateComm) as jour,
2          month(dateComm) as mois, year(dateComm) as annee,
3          date_add(dateComm, interval 15 day) as
             previsionLivraison
4   from commande
5 ;
```

resultat	idComm	dateComm	jour	mois	annee	previsionLivraison
	1	2016-05-02	2	5	2016	2016-05-17
	2	2016-05-06	6	5	2016	2016-05-21
	3	2016-06-15	15	6	2016	2016-06-30
	4	2016-06-22	22	6	2016	2016-07-07

nombres de lignes : 4

TABLE 11 – Fonctions de chaines de caractères

fonction	*résultat renvoyé*
ch1 + ch2 ch1 || ch2 **concat**(ch1,ch2,...)	Concaténation de chaines (dépend du SGBD) (MySql)
left(x,y)	Renvoie y caractères de x à partir de la gauche
lower(chaine)	Convertit en minuscule
ltrim(chaine)	Supprime des espaces devant une colonne chaîne de caractères (à gauche : L=left)
reverse(chaine)	Renvoie l'inverse **reverse**("trottinette") renvoie ettenittort
right(x,y)	Renvoie y caractères de x à partir de la droite
rtrim(chaine)	Supprime des espaces derrière une colonne chaîne de caractères (à droite : R=right)
soundex(chaine)	Renvoie un code à 4 chiffres permettant d'évaluer la similitude, en terme de sonorité, entre 2 chaînes de caractères **soundex**("stiller") renvoie S346 **soundex**("stylo") renvoie S340
space(n)	Renvoie une chaîne composée de n espaces
str(chaine)	Conversion d'une donnée numérique en chaîne de caractères
char_length(chaine)	donne la longueur d'une chaine en nombre de caractères (les espaces comptent...)
substring(x, y, z)	Renvoie z caractères de x à partir du caractère à la position y
trim(chaine)	supprime les espaces devant et derrière une chaine
upper(chaine)	Convertit en majuscule
convert(ch , type) **convert**(ch **using** encodage)	convertit une expression dans un certain type ou change une valeur d'encodage (pour passer d'un encodage à un autre)

membre	nom_memb	prenom_memb
	dupont	pierrE
	Lajoie	caroline
	durant	JaCques
	durand	FRED
	Lambert	annie
	durant	PAul

nombres de lignes : 6

Listing 90 – les membres en mieux bien présentés...

```
1 select upper(nom_memb) as NOM,
2         concat(upper(left(prenom_memb,1)),lower(right(
            prenom_memb,(length(prenom_memb)-1)))) as Prenom
            ,
3        upper(concat(left(nom_memb,1), left(prenom_memb,1)))
            as initiales
4   from membre
5 ;
```

resultat	NOM	Prenom	initiales
	DUPONT	Pierre	DP
	LAJOIE	Caroline	LC
	DURANT	Jacques	DJ
	DURAND	Fred	DF
	LAMBERT	Annie	LA
	DURANT	Paul	DP

nombres de lignes : 6

TABLE 12 – Autres fonctions ou variables système

fonction	*valeur renvoyée*
current_date **now**()	date courante date heure courante
current_time	heure courante (du serveur...)
cast (colonne **as** type)	Transtypage (changement de type) de colonne vers un nouveau type
coalesce(colonne, valeur si null)	Définir une valeur pour remplacer la valeur nulle d'une colonne

personne	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h
	5	Rigole	Jean	null	1300.00	2007-09-28	h

nombres de lignes : 5

Listing 91 – les personne et leur ville

```
1 select nom, prenom, coalesce(ville,"** absent **") as ville
2   from personne
3 ;
```

resultat	nom	prenom	ville
	Dupont	Max	arras
	Durand	Tim	Aix
	Lambert	Betty	Pau
	Bradford	Jean	Arras
	Rigole	Jean	** absent **

nombres de lignes : 5

Listing 92 – date et heure du jour

```
select now(), current_date, current_time
  from dual
;
```

resultat	now()	current_date	current_time
	2024-02-07 17 :06 :53.0	2024-02-07	17 :06 :53

nombres de lignes : 1

'dual' est une pseudo-table utilisable pour effectuer des calculs sans pour autant avoir de données :

Listing 93 – calcul de l'aire et de la circonférence d'un disque

```
select 5 as rayon, (pi()*pow(5,2)) as aireDisque,
        (2*pi()*5) as circonference
  from dual
;
```

resultat	rayon	aireDisque	circonference
	5	78.53981633974483	31.415927

nombres de lignes : 1

30.3 Choix d'une valeur selon une condition...

Listing 94 – Syntaxe **case** : choix de valeur

```
case valeur
    when valeurComparaison THEN valeurResultat
    [when valeurComparaison THEN valeurResultat] ...
    [else valeurResultat]
end
```

ou

Listing 95 – Syntaxe **case** : choix de valeur

```
case
    when condition THEN valeurResultat
    [when condition THEN valeurResultat] ...
    [else valeurResultat]
end
```

Listing 96 – date et heure du jour

```
select nom, prenom,
        case sexe
          when 'f' then 'femme'
          when 'h' then 'homme'
          else 'inconnu'
        end as sexe
  from personne
;
```

resultat	nom	prenom	sexe
	Dupont	Max	homme
	Durand	Tim	homme
	Lambert	Betty	femme
	Bradford	Jean	homme
	Rigole	Jean	homme

nombres de lignes : 5

31 Classement des lignes du résultat : `order by`

Cette clause permet le classement du résultat final avant d'être renvoyé.

31.1 Syntaxe

Classement des lignes en SQL

```
select *
  from table(s)
  [where ...]
  [group by ...]
  [having ...]
  order by critere1[, criteres2, ...]
;
```

où :

- *critere1*, *critere2*, etc. sont des critères de classement comportant chacun un nom de colonne suivi de :
  - soit **`asc`**, ordre croissant (en anglais : *ascending*), par défaut
  - soit **`desc`**, ordre décroissant (en anglais : *descending*)

31.2 Exemples

Listing 97 – Les avions classés par ville et par numéro

```
select  *
  from avion
  order by localisav, numav
;
```

Resultat	numav	nomav	localisav	capamaxav
	107	A320	Lille	178
	101	A300	Marseille	300
	104	A330	Marseille	208
	105	A330	Marseille	208
	102	A330	Nice	208
	106	A320	Nice	178
	103	A340	Paris	275
	108	A330	Paris	208
	109	A380	Paris	516

nombres de lignes : 9

Listing 98 – Les pilotes classés par ville et par salaire decroissant

```
select  *
  from pilote
  order by vilpil, salairepil desc
;
```

Resultat	numpil	nompil	vilpil	datnaispil	salairepil
	7	Bill	Marseille	1980-06-29	18000.00
	5	Mandy	Marseille	1985-05-16	16000.00
	4	Scott	Marseille	1981-08-29	12000.00
	6	John	Nice	1982-06-11	20000.00
	2	Peter	Nice	1970-01-10	18000.00
	1	William	Nice	1980-06-29	16000.00
	9	boule	Paris	1980-06-29	17000.00
	8	Camille	Paris	1989-10-06	16000.00
	3	Max	Paris	1975-04-03	12000.00

nombres de lignes : 9

Listing 99 – Le total des salaires par ville classé par total décroissant

```
select  vilpil, sum(salairepil) as total
  from pilote
  group by vilpil
  order by total desc, vilpil asc
;
```

Resultat	vilpil	total
	Nice	54000.00
	Marseille	46000.00
	Paris	45000.00

nombres de lignes : 3

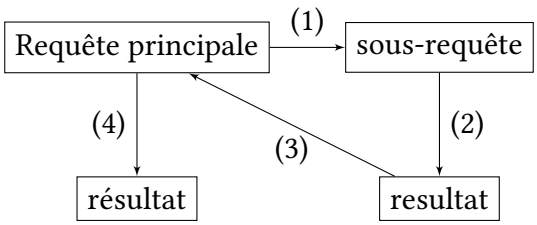
32 Sous-requêtes

Une *sous-requête* (en anglais : *subquery*), ou *requête interne* (en anglais : *inner subquery*), ou encore requête imbriquée (en anglais : *nested subquery*), est une requête utilisée pour constituer un résultat qui va être utilisé dans une *requête principale*, ou *requête externe* (en anglais : *outer query*), de la manière suivante :

- en général comme élément de comparaison dans les clauses **`where`** ou **`having`** (dans les ordres DML et DQL),
- parfois comme valeur d'une colonne dans la clause **`select`**.
- exceptionnellement en remplacement d'une table dans la clause **`from`** (SQL92).

Attention
La sous-requête n'est pas terminée par un point-virgule (' ;').

FIGURE 4 – Requête principale et sous-requête



32.1 Sous-requêtes indépendantes ou corrélées

Deux formes de sous-requêtes sont définies, en fonction de leur lien avec la requête principale :

- sous-requêtes *indépendantes*, sans aucun lien avec la requête appelante
  - elle peut être lancée seule, elle est totalement indépendante de la requête appelante ; son évaluation est effectuée une seule fois et le jeu de résultat qu'elle renvoie va servir à la requête appelante comme valeur unique, ou liste de valeurs .
    1. la sous-requête est appelée
    2. elle produit son jeu de donnée
    3. ce dernier est récupéré par la requête principale qui peut poursuivre son exécution
    4. la requête principale retourne le jeu de donnée résultat
- sous-requêtes *corrélées*, ou dépendantes ou liées, qui ont un lien avec la requête appelante
  - son exécution va dépendre, cette fois, de valeurs de colonnes de la requête principale, elle est liée à la requête principale, elle en est dépendante, corrélée (en relation avec) ; elle est exécutée pour chaque ligne de la requête appelante (coût très élevé !).

1. pour chaque ligne de la requête principale
   (a) la sous-requête est appelée
   (b) elle produit son jeu de donnée
   (c) ce dernier est récupéré par la requête principale qui l'utilise
2. la requête principale retourne le jeu de donnée résultat

FIGURE 5 – Sous-requête indépendante

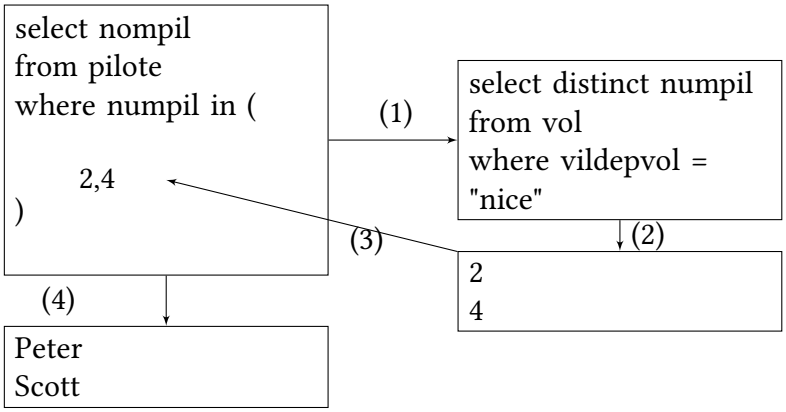
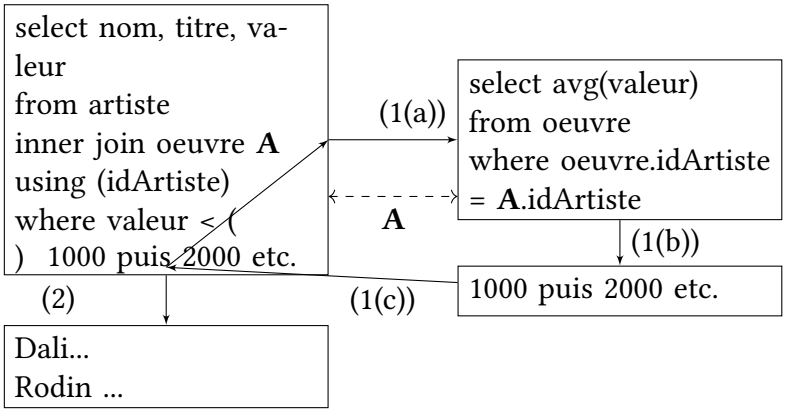


FIGURE 6 – Sous-requête corrélée



32.2 Formes de résultat d'une sous-requête

Une sous-requête doit produire un résultat (ensemble de lignes et de colonnes) conforme à ce qu'attend la requête principale et de l'opérateur utilisé dans le cas d'une sélection.

Attention

Il faut absolument s'assurer que le résultat produit par la sous-requête sera toujours cohérent par rapport à ce qu'attend la requête principale.

Dans le cas contraire, une erreur d'exécution serait générée.

Ce résultat peut-être sous forme :

- d'une *valeur unique* (1 ligne et 1 colonne) : il pourra être utilisé
  - dans un critère de sélection dans les clauses **`where`** et **`having`**, combinés avec un opérateur de comparaison =, <, >, <=,>=, <>, ou l'opérateur étendu **`between`**
  - pour constituer la valeur d'une colonne dans la clause **`select`**
- de *plusieurs lignes mais une colonne unique* (1 à N lignes – une seule colonne) : il pourra être utilisé
  - pour constituer un critère de sélection dans les clauses **`where`** et **`having`** : opérateurs **`in`**, **`not in`**, opérateurs **`all`**, **`any`** combinés avec un opérateur de comparaison =, <, >, <=,>=, <>,
  - pour constituer un jeu de lignes utilisé dans la clause **`from`** en guise de table
- d'un *jeu de données quelconque* (0 à N lignes – 1 à N colonnes) :
  - pour constituer un critère de sélection dans les clauses **`where`** et **`having`** : opérateur **`exists`**
  - pour constituer un jeu de lignes dans la clause **`from`**

32.3 Utilisation de sous-requêtes à résultat unique

32.3.1 Requêtes produisant une valeur unique

Listing 100 – lister le nombre de personnes à Berlin

```
1 select  count(*)
2   from personne
3   where ville = 'berlin'
4 ;
```

personne	count(*)
	0

nombres de lignes : 1

Le résultat est toujours garanti : si aucune ligne est trouvée, le résultat vaudra 0.

Listing 101 – lister la dernière date d'entrée de personne

```
1 select max(dateEntree)
2   from personne
3 ;
```

personne	max(dateEntree)
	2007-09-28

nombres de lignes : 1

On peut garantir la cohérence du résultat à condition qu'il y ait au moins une personne dans la table. Sinon la valeur vaudra **null** est sera utilisée comme telle.

Listing 102 – la date d'entrée d'une personne

```
1 select  dateEntree
2   from personne
3   where numero = 3
4 ;
```

personne	dateEntree
	2007-04-20

nombres de lignes : 1

On peut garantir la cohérence du résultat à condition que la personne de numéro 3 (clef primaire) existe bien. Sinon la valeur vaudra **null** est sera utilisée comme telle.

32.3.2 Exemples : constituer la valeur d'une colonne

Le titre des peintures, leur largeur et la largeur moyenne de toutes les peintures)

1. Sous-requête indépendante : la largeur moyenne des peintures

```
1 select  avg(largeur_peinture)
2   from peinture;
```

resultat	avg(largeur_peinture)
	2.575000

nombres de lignes : 1

2. Requête complète :

```
1 select titre_oeuvre, largeur_peinture,
2        (select avg(largeur_peinture) from peinture)
3           as largeurMoyenne
4   from peinture
5        inner join oeuvre
6           on id_oeuvre_peinture = id_oeuvre
7 ;
```

resultat	titre_oeuvre	largeur_peinture	largeurMoyenne
	La Joconde	0.40	2.575000
	La cŠne	6.00	2.575000
	Le sommeil	1.20	2.575000
	M,tamorphose de Narcisse	2.00	2.575000
	Guernica	6.00	2.575000
	Le champs des coquelicots	1.80	2.575000
	La louve	1.20	2.575000
	White Light	2.00	2.575000

nombres de lignes : 8

32.3.3 Exemple : effectuer une sélection

Le titre et largeur des peintures dont la largeur est supérieure à la moyenne

1. Sous-requête indépendante : la largeur moyenne des peintures

```
1 select  avg(largeur_peinture)
2   from peinture;
```

resultat	avg(largeur_peinture)
	2.575000

nombres de lignes : 1

2. Requête complète :

```
1 select titre_oeuvre, largeur_peinture
2   from peinture
3        inner join oeuvre
4           on id_oeuvre_peinture = id_oeuvre
5   where largeur_peinture >
6        (select avg(largeur_peinture) from peinture)
7 ;
```

resultat	titre_oeuvre	largeur_peinture
	La cŠne	6.00
	Guernica	6.00

nombres de lignes : 2

Le titre et largeur des peintures, dont la largeur est autour de la moyenne plus ou moins 30 %

1. Sous-requête indépendante : la largeur moyenne des peintures

```
1 select  avg(largeur_peinture)
2   from peinture;
```

resultat	avg(largeur_peinture)
	2.575000

nombres de lignes : 1

2. Requête complète :

```
1 select titre_oeuvre, largeur_peinture
2   from peinture
3        inner join oeuvre
4           on id_oeuvre_peinture = id_oeuvre
5   where largeur_peinture between
6        (select avg(largeur_peinture) from peinture)*0.70
7           and
```

```
8          (select avg(largeur_peinture) from peinture)*1.30
9 ;
```

resultat	titre_oeuvre	largeur_peinture
	M,tamorphose de Narcisse	2.00
	White Light	2.00

nombres de lignes : 2

32.3.4 Exemple : constituer un jeu de données

Alternatives à l'utilisation de sous-requêtes dans les clauses **`select`** ou **`where`** : constituer un jeu de données comme une table dans la clause **`from`**

```
1 select titre_oeuvre, largeur_peinture, M.largeurMoyenne
2   from (peinture
3         inner join oeuvre
4            on id_oeuvre_peinture = id_oeuvre)
5         cross join
6              (select avg(largeur_peinture) as largeurMoyenne
7                  from peinture
8              ) M
9 ;
```

resultat	titre_oeuvre	largeur_peinture	largeurMoyenne
	La Joconde	0.40	2.575000
	La cŠne	6.00	2.575000
	Le sommeil	1.20	2.575000
	M,tamorphose de Narcisse	2.00	2.575000
	Guernica	6.00	2.575000
	Le champs des coquelicots	1.80	2.575000
	La louve	1.20	2.575000
	White Light	2.00	2.575000

nombres de lignes : 8

```
1 select titre_oeuvre, largeur_peinture
2   from (peinture
3            inner join oeuvre
4              on id_oeuvre_peinture = id_oeuvre
5         )
6         cross join
7            (select avg(largeur_peinture) as largeurMoyenne
8               from peinture
9            ) M
```

```
10   where largeur_peinture > M.largeurMoyenne
11 ;
```

resultat	titre_oeuvre	largeur_peinture
	La cŠne	6.00
	Guernica	6.00

nombres de lignes : 2

```
1 select titre_oeuvre, largeur_peinture,
2           MA.largeurMoyenne
3   from peinture
4        inner join oeuvre
5           on id_oeuvre_peinture = id_oeuvre
6        inner join
7         (select  id_artiste, avg(largeur_peinture) as
             largeurMoyenne
8           from peinture P
9             inner join oeuvre O
10             on (P.id_oeuvre_peinture = O.id_oeuvre)
11          group by id_artiste
12        ) MA on oeuvre.id_artiste = MA.id_artiste
13 ;
```

resultat	titre_oeuvre	largeur_peinture	largeurMoyenne
	La Joconde	0.40	3.200000
	La cŠne	6.00	3.200000
	Le sommeil	1.20	1.600000
	M,tamorphose de Narcisse	2.00	1.600000
	Guernica	6.00	6.000000
	Le champs des coquelicots	1.80	1.800000
	La louve	1.20	1.600000
	White Light	2.00	1.600000

nombres de lignes : 8

32.3.5 Exemples : sous-requête corrélée, constituer la valeur d'une colonne

Le titre des peintures, leur largeur et la largeur moyenne des peintures du peintre

1. Sous-requête corrélée : la largeur moyenne des peintures du peintre

```
1 select  avg(largeur_peinture)
2   from peinture P
```

```
3           inner join oeuvre O
4              on (P.id_oeuvre_peinture = O.id_oeuvre)
5  -- where (O.id_artiste = OEUVRE.id_artiste)
6 ;
```

resultat	avg(largeur_peinture)
	2.575000

nombres de lignes : 1

Cette requête est dépendante d'une requête principale, elle ne peut fonctionner seule, elle doit avoir accès à l'information 'OEUVRE.id_artiste' qui est disponible dans la requête principale qui suit ('–' correspond à un commentaire à supprimer pour la véritable requête);

2. Requête complète :

```
1 select titre_oeuvre, largeur_peinture,
2         (select  avg(largeur_peinture)
3           from peinture P
4             inner join oeuvre O
5              on (P.id_oeuvre_peinture = O.id_oeuvre)
6           where (O.id_artiste =
7                       oeuvre.id_artiste)
8         )
9           as largeurMoyenne
10   from peinture
11        inner join oeuvre
12           on id_oeuvre_peinture = id_oeuvre
13 ;
```

resultat	titre_oeuvre	largeur_peinture	largeurMoyenne
	La Joconde	0.40	3.200000
	La cŠne	6.00	3.200000
	Le sommeil	1.20	1.600000
	M,tamorphose de Narcisse	2.00	1.600000
	Guernica	6.00	6.000000
	Le champs des coquelicots	1.80	1.800000
	La louve	1.20	1.600000
	White Light	2.00	1.600000

nombres de lignes : 8

32.4 Utilisation de sous-requêtes à lignes multiples et colonne unique

Les valeurs de la colonne unique retournées par une sous-requête vont servir d'élément de comparaison avec un opérateur traitant d'un ensemble de valeurs : **in** et **not in**, **all**, **any**.

32.4.1 Requêtes produisant des lignes multiples et une colonne unique

Listing 103 – les numéros des membres qui habitent Cachan

```
1 select  id_memb
2   from membre
3   where ville_memb = 'cachan'
4 ;
```

resultat	id_memb
	1
	4

nombres de lignes : 2

Listing 104 – les villes des membres dont le nom commence par D

```
1 select  distinct ville_memb
2   from membre
3   where nom_memb like 'd%'
4 ;
```

resultat	ville_memb
	cachan
	creteil
	melun

nombres de lignes : 3

32.4.2 Utiliser dans une sélection

le nom des membres qui ne se sont jamais inscrits à une activité

1. Sous-requête : les numéros des membres inscrits

```
1 select distinct id_memb
2   from inscrire
3 ;
```

resultat	id_memb
	1
	2
	3
	5
	6

nombres de lignes : 5

2. Requête principale : avec **not in**

```
1 select nom_memb, prenom_memb
2   from membre
3   where id_memb not in
4           (select id_memb from inscrire)
5 ;
```

resultat	nom_memb	prenom_memb
	durand	FRED

nombres de lignes : 1

soit : les membres dont le numéro ne se trouve pas dans la liste des numéros des membres inscrits (= opération *différence*),

3. Requête principale : avec **all**

```
1 select nom_memb, prenom_memb
2   from membre
3   where id_memb <> all
4             (select id_memb from inscrire)
5 ;
```

resultat	nom_memb	prenom_memb
	durand	FRED

nombres de lignes : 1

soit : les membres dont le numéro est différent de tous les numéros de membres inscrits (= opération *différence*),

- L'opérateur **all** est associé à un opérateur de comparaison et permet de comparer une valeur de colonne de la requête principale à toutes les valeurs retournées par la sous-requête. Avec l'opérateur **all**, toutes les valeurs de la sous-requête doivent répondre au critère de comparaison.

32.5 Utilisation de sous-requêtes à jeu de données quelconque : opérateur exists

Les requêtes exploitent le retour de ce type de sous-requêtes pour vérifier que le résultat est vide ou non.

```
1 select *
2   from membre
3   where ville_memb = 'cachan'
4 ;
```

resultat	id_memb	nom_memb	prenom_memb	adresse_memb	ville_memb	dateNaissance_memb	id_memb_parrain
	1	dupont	pierrE	bld Gambetta	cachan	1975-10-20 00 :00 :00.0	null
	4	durand	FRED	bld Lafayette	cachan	1989-07-14 00 :00 :00.0	5

nombres de lignes : 2

Listing 105 – les membres qui habitent moscou (résultat vide)

```
1 select *
2   from membre
3   where ville_memb = 'moscou'
4 ;
```

resultat	id_memb	nom_memb	prenom_memb	adresse_memb	ville_memb	dateNaissance_memb	id_memb_parrain

nombres de lignes : 0

32.5.1 Utiliser dans une sélection

Lister tous les membres si certains ont leur anniversaire ce mois-ci

1. Sous-requête : les membres dont l'anniversaire est dans le mois courant

Listing 106 – tous les membres s'il existe des membres qui sont nés ce mois-ci

```
1 select 1
2   from membre
3   where month(dateNaissance_memb) = month(now())
4 ;
```

resultat	1

nombres de lignes : 0

2. Requête principale :

```
1 select *
2   from membre
3   where exists (
4         select 1
5           from membre
6           where month(dateNaissance_memb) = month(now())
7         )
8 ;
```

resultat	id_memb	nom_memb	prenom_memb	adresse_memb	ville_memb	dateNaissance_memb	id_memb_parrain

nombres de lignes : 0

32.6 Application à la différence

La différence utilisant une sous-requête

$(R - S)$

```
1 select  col1, col2, col3, ...colN
2   from R
3   where (col1, col2, col3, ...colN) not in
4   (
5     select colS1, colS2, colS3, ...colSN
6       from S
7   )
8   ;
```

où :

- *R* et *S* : tables objets de la différence
- *col1, col2, col3, ...colN* : liste des attributs format une ligne de R qu'on ne doit pas trouver dans S (ligne de S formée par *colS1, colS2, colS3, ...colSN*)

On pourra souvent simplifier en utilisant seulement des valeurs de clef primaire au lieu de tous les attributs.

32.6.1 Exemples

Exemple 1

R	A	B	C
	a	b	c
	d	a	f
	c	b	d

nombres de lignes : 3 -

S	A	B	C
	b	g	a
	d	a	f

nombres de lignes : 2

Listing 107 – Exemple de différence à partir d'une sous-requête ($(R - S)$)

```
select  A,B,C
  from R
  where (A,B,C) not in
    (select  A,B,C
       from S
    )
;
```

resultat	A	B	C
	a	b	c
	c	b	d

nombres de lignes : 2

Exemple 2 : les pilotes qui n'ont pas volé

1. Sous-requête : les numéros des pilotes qui ont volé

```
select distinct numpil
  from vol
;
```

resultat	numpil
	3
	4
	2
	5
	6
	7

nombres de lignes : 6

2. Requête complète

```
1 select  numpil
2   from pilote
3   where numpil not in
4           (select  numpil
5              from vol)
6 ;
```

resultat	numpil
	1
	8
	9

nombres de lignes : 3

soit : les pilotes dont le numéro n'est pas dans la liste de ceux qui ont volé

32.7 Application à l'intersection

L'intersection utilisant une sous-requête

$(R \cup S)$

```
1 select  col1, col2, col3, ...colN
2   from R
3   where (col1, col2, col3, ...colN) in
4   (
5     select colS1, colS2, colS3, ...colSN
6       from S
7   )
8   ;
```

où :

- *R* et *S* : tables objets de l'intersection
- *col1, col2, col3, ...colN* : liste des attributs format une ligne de R qu'on doit trouver dans S (ligne de S formée par *colS1, colS2, colS3, ...colSN*)

On pourra souvent simplifier en utilisant seulement des valeurs de clef primaire au lieu de tous les attributs.

32.7.1 Exemples

Exemple 1

R	A	B	C
	a	b	c
	d	a	f
	c	b	d

nombres de lignes : 3 -

S	A	B	C
	b	g	a
	d	a	f

nombres de lignes : 2

Listing 108 – Exemple d'intersection SQL utilisant une sous-requête ($(R \cap S)$)

```
select A,B,C
  from R
  where (A,B,C) IN
     (select  A,B,C
        from S)
;
```

resultat	A	B	C
	d	a	f

nombres de lignes : 1

Exemple 2 : les pilotes qui ont volé

1. Sous-requête : les numéros des pilotes qui ont volé

```
select distinct numpil
  from vol
;
```

resultat	numpil
	3
	4
	2
	5
	6
	7

nombres de lignes : 6

2. Requête complète

```
1 select  numpil
2   from pilote
3   where numpil in
4           (select  numpil
5              from vol)
6 ;
```

resultat	numpil
	2
	3
	4
	5
	6
	7

nombres de lignes : 6

soit : les pilotes sont le numéro est aussi dans la liste de ceux qui ont volé.

Remarque : une jointure répond également à cette question.

32.8 Application à la division

La requête de division relationnelle peut être traitée de plusieurs manières. L'une des méthodes est l'utilisation d'une double négation avec test d'existence et l'autre en utilisant un comptage de lignes.

membre	id_memb	nom_memb	prenom_memb	adresse_memb	ville_memb	dateNaissance_memb	id_memb_parrain
	1	dupont	pierrE	bld Gambetta	cachan	1975-10-20 00 :00 :00.0	null
	2	Lajoie	caroline	jardin fleuri	bagneux	1990-06-30 00 :00 :00.0	3
	3	durant	JaCques	Les ulysses	creteil	1980-01-01 00 :00 :00.0	1
	4	durand	FRED	bld Lafayette	cachan	1989-07-14 00 :00 :00.0	5
	5	Lambert	annie	rue Ionesco	creteil	1979-05-21 00 :00 :00.0	null
	6	durant	PAul	rue des soupirants	melun	1984-12-12 00 :00 :00.0	null

nombres de lignes : 6

inscrire	id_memb	id_activ	date_inscrire
	1	1	2005-02-10 00 :00 :00.0
	1	2	2005-02-01 00 :00 :00.0
	1	3	2005-04-10 00 :00 :00.0
	1	4	2005-02-10 00 :00 :00.0
	2	1	2005-02-15 00 :00 :00.0
	2	3	2005-05-12 00 :00 :00.0
	2	4	2005-02-10 00 :00 :00.0
	3	1	2005-02-22 00 :00 :00.0
	3	2	2005-02-12 00 :00 :00.0
	5	2	2005-02-28 00 :00 :00.0
	6	1	2005-03-15 00 :00 :00.0
	6	2	2005-02-26 00 :00 :00.0
	6	3	2005-05-16 00 :00 :00.0

nombres de lignes : 13

activite	id_activ	intitule_activ	date_activ	minPart_activ	tarif1_activ	tarif2_activ
	1	Lille expo.	2005-04-01 00 :00 :00.0	2	20.00	30.00
	2	La baie de Somme	2005-03-15 00 :00 :00.0	3	8.00	16.00
	3	Paris Expo.	2005-06-15 00 :00 :00.0	3	30.00	45.00
	4	repas de fin d'annee	2005-03-15 00 :00 :00.0	5	8.00	15.00

nombres de lignes : 4

Les membres qui se sont inscrits à toutes les activités

32.8.1 En utilisant une double négation

Quels sont les membres tels qu'il n'existe pas d'activités pour lesquelles ces membres ne s'y soient pas inscrits ?

Listing 109 – Division avec double négation

```
select  id_memb, nom_memb, prenom_memb
  from membre
  where not exists
    (select * from activite
       where not exists
        (select *
           from inscrire
           where id_memb = membre.id_memb
               and id_activ = activite.id_activ
        )
    )
;
```

resultat	id_memb	nom_memb	prenom_memb
	1	dupont	pierrE

nombres de lignes : 1

32.8.2 En utilisant des comptages

Quels sont les membres pour lesquelles le nombre d'activités différentes auxquels ils ont participé est égal au nombre total d'activités différentes ?

Listing 110 – Division avec un agrégat

```
select m.id_memb, nom_memb, prenom_memb
  from membre m
         inner join inscrire p
            on m.id_memb = p.id_memb
  group by id_memb, nom_memb, prenom_memb
  having count(distinct id_activ) =
           (select count(distinct id_activ)
               from activite)
;
```

resultat	id_memb	nom_memb	prenom_memb
	1	dupont	pierrE

nombres de lignes : 1

32.8.3 En utilisant des comptages

Quels sont les membres pour lesquelles le nombre d'activités différentes auxquels ils ont participé est égal au nombre total d'activités différentes ?

Listing 111 – Division avec plusieurs sous-requêtes

```
1 select m.id_memb, nom_memb, prenom_memb
2   from membre m
3   where (select count(distinct id_activ)
4             from inscrire
5             where inscrire.id_memb = m.id_memb
6           )
7           =  (select count(distinct id_activ)
8                       from activite)
9 ;
```

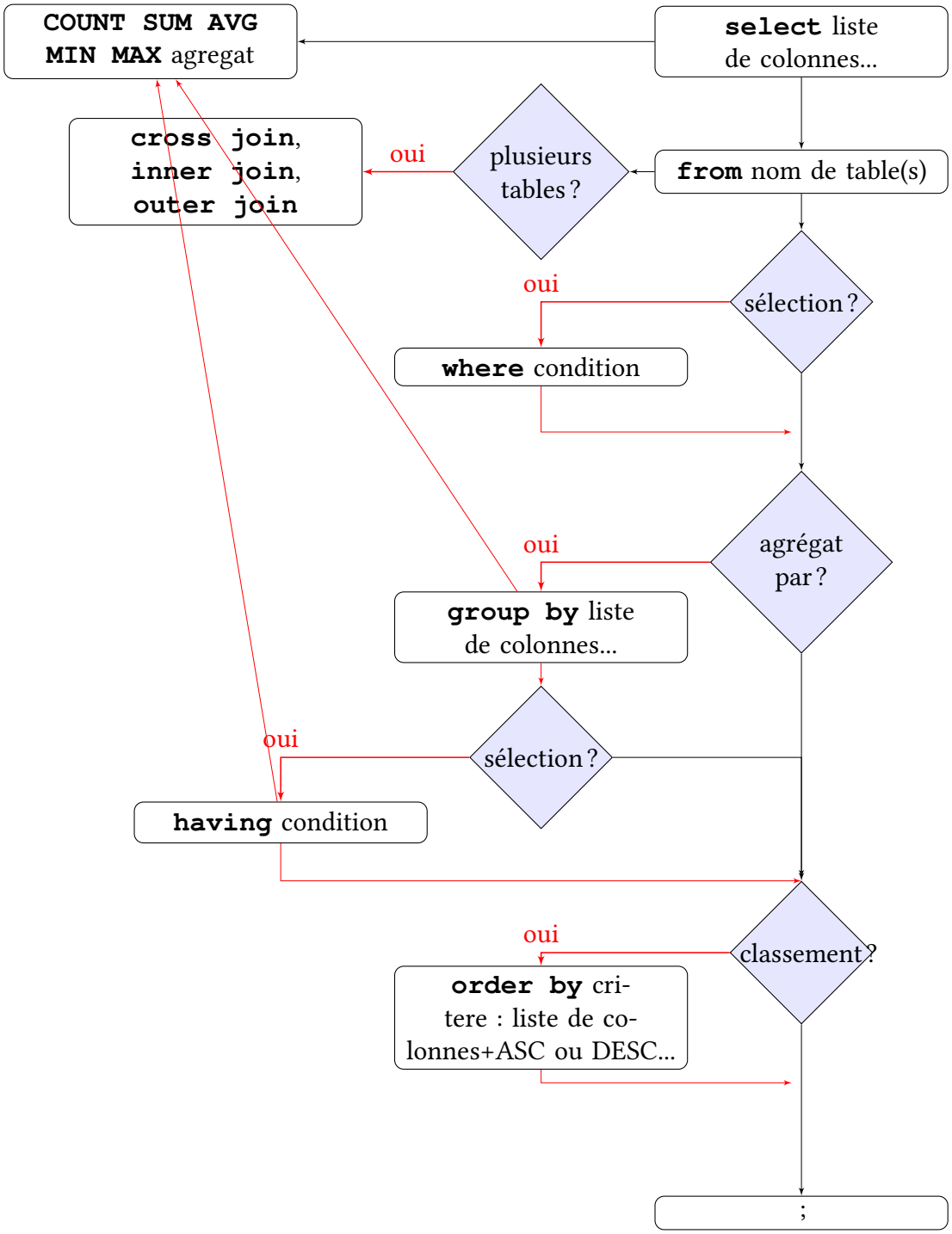
resultat	id_memb	nom_memb	prenom_memb
	1	dupont	pierrE

nombres de lignes : 1

Attention : ces requêtes fonctionnent correctement dans ce cas précis car les comptages sont réalisés sur des colonnes clé primaires et clé étrangères avec contraintes d'intégrité référentielle.

Pièges pouvant fournir des résultats erronés sur ce type de requêtes : cf. site Web exemples division relationnelle

33 Résumé `select`



Cinquième partie

Performance et sécurité

34 Organisation des index

Les *index* sont des tables "système" gérées par le SGBDR pour ses besoins propres : intégrité, performance.

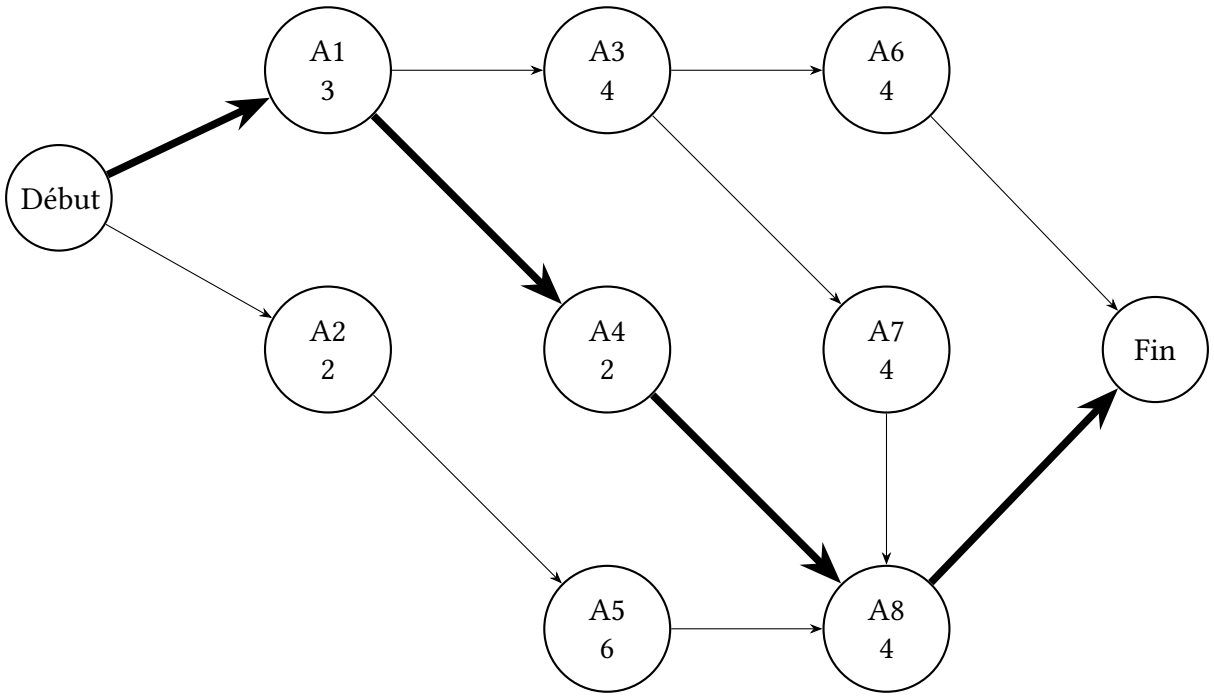
Ils sont implantés sous forme d'arbres, graphes de structure hiérarchique.

34.1 Graphes

Un graphe est un ensemble de points dont certains sont reliés 2 à 2. Les liaisons entre ces points peuvent être orientées ou non. Les points sont appelés sommets (en anglais : *vertice*) ou noeuds (en anglais : *nodes*), les liens sont appelés arêtes (en anglais : *edge*) ou arcs (orientés). Les graphes peuvent être étiquetés : aux sommets ou arêtes sont associés des valeurs d'un ensemble (nombres, couleurs, etc.)

Les graphes permettent la représentation de données complexes comme : les hyperliens du Web, le réseau Internet, les réseaux sociaux, la succession des états d'un système, la planification (exemple ici d'un graphe PERT), etc.

Figure 7 – Graphe



34.2 Arbres binaires

Un *arbre binaire* (en anglais : *BT, Binary Tree*) est une structure de données de type graphe représentée sous forme hiérarchique, arborescente, dont chaque élément est appelé noeud (en anglais : *node*). Un noeud permet le stockage de données de manière efficace (pour la recherche), extensible (la limite est la mémoire) et cohérente (représentation de hiérarchies diverses, nomenclatures, familles, etc.).

Un noeud peut posséder au plus 2 noeuds fils : un noeud fils gauche et un noeud fils droit.

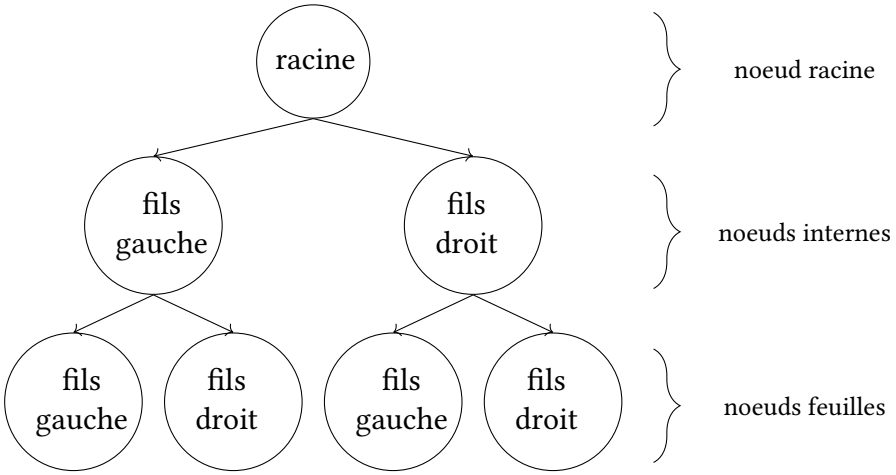
Le noeud initial d'un arbre binaire est appelé *racine*.

Les noeuds terminaux (ceux qui ne possèdent aucun noeud) sont appelés *feuilles*. Les autres noeuds sont appelés noeuds *internes*.

Les noeuds internes peuvent être considérés comme racines de sous-arbres. Le niveau d'un noeud dans un arbre est appelé profondeur.

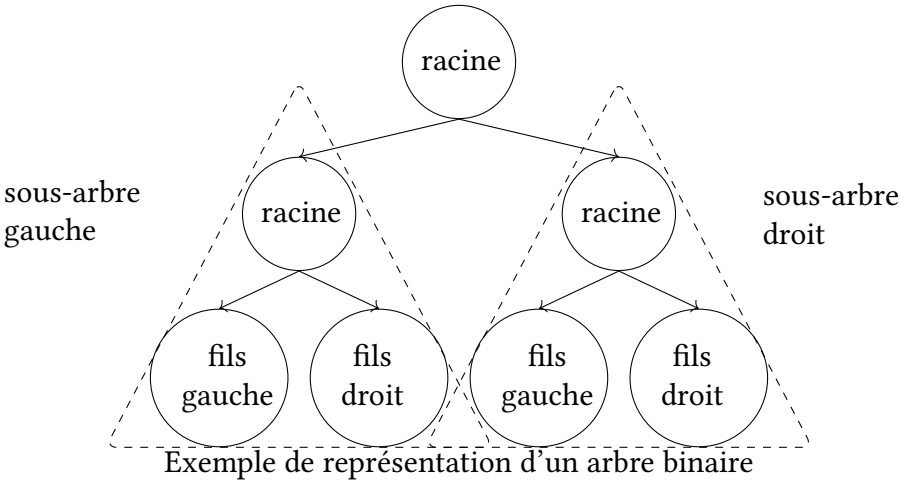
La *hauteur d'un arbre* binaire correspond à la distance entre la racine et la feuille la plus éloignée.

Figure 8 – Arbre



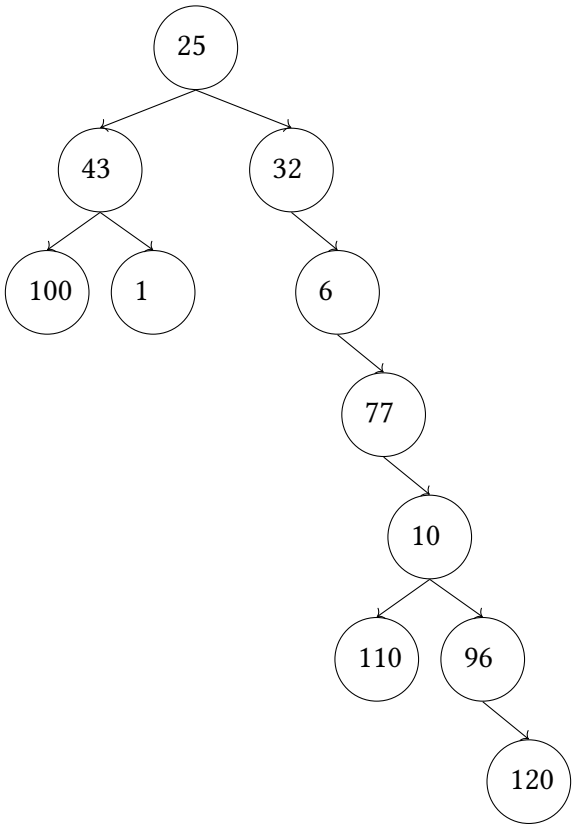
Chaque noeud peut être à son tour racine d'un sous-arbre :

Figure 9 – Arbres et sous-arbres



Exemple de représentation d'un arbre binaire

FIGURE 10 – Arbre binaire



34.2.1 Parcours en profondeur des arbres binaires

Trois formes de parcours dits « en profondeur » sont proposées pour le parcours des noeuds d'un arbre binaire (algorithmes récursifs) :

- le parcours *préfixe*

Algorithme 1 parcours préfixe

```
1: Fonction parcourirPrefixe(Noeud n)
2:     afficher n
3:     si nonVide(gauche(n)) alors
4:         parcourirPrefixe (gauche(n))
5:     fin si
6:     si nonVide(droite(n)) alors
7:         parcourirPrefixe (droite(n))
8:     fin si
9: fin Fonction
```

affiche : 25, 43, 100, 1, 32, 6, 77, 10, 110, 96, 120

- le parcours *suffixe* ou *postfixe*

Algorithme 2 parcours suffixe

```
1: Fonction parcourirSuffixe(Noeud n)
2:     si nonVide(gauche(n)) alors
3:         parcourirSuffixe (gauche(n))
4:     fin si
5:     si nonVide(droite(n)) alors
6:         parcourirSuffixe (droite(n))
7:     fin si
8:     afficher n
9: fin Fonction
```

affiche : 100, 1, 43, 110, 120, 96, 10, 77, 6, 32, 25

- le parcours *infixe*

Algorithme 3 parcours infixe

```
1: Fonction parcourirInfixe(Noeud n)
2:     si nonVide(gauche(n)) alors
3:         parcourirInfixe (gauche(n))
4:     fin si
5:     afficher n
6:     si nonVide(droite(n)) alors
7:         parcourirInfixe (droite(n))
8:     fin si
9: fin Fonction
```

affiche : 100, 43, 1, 25, 32, 6, 77, 110, 10, 96, 120

34.3 Arbres binaires de recherche

Un *ABR*, *Arbre Binaire de Recherche* (en anglais : *BST, Binary Search Tree*), est un arbre binaire dans lequel chaque noeud est porteur d'une valeur de clef unique, et donc les clefs des noeuds fils doivent respecter une relation d'ordre telle que

- toutes les clefs des noeuds du sous-arbre gauche lui soient inférieures
- toutes les clefs des noeuds du sous-arbre droit lui soient supérieures.

La localisation d'une valeur de clef dans un arbre binaire de recherche procède d'une manière similaire à la recherche dichotomique.

FIGURE 11 – Arbre binaire de recherche : fils gauche et droit ordonnés

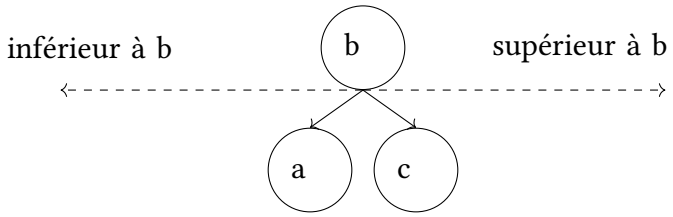
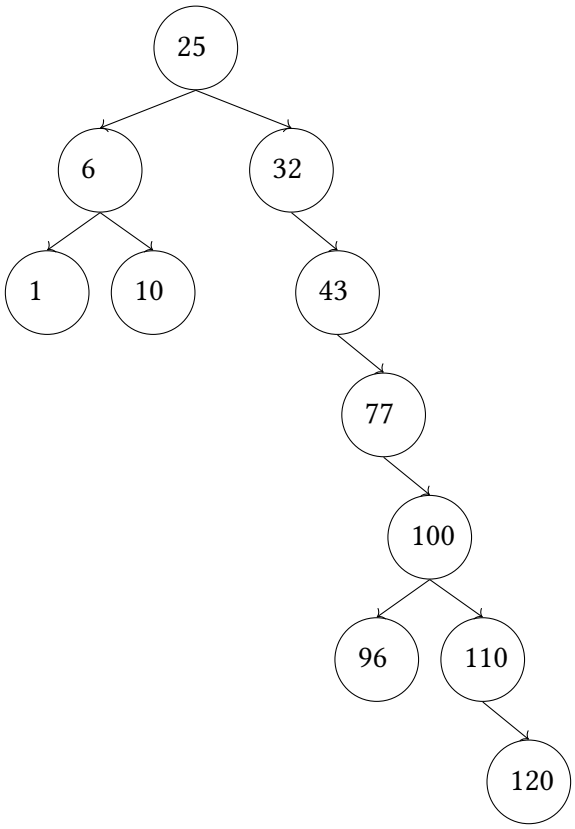


FIGURE 12 – Arbre binaire de recherche

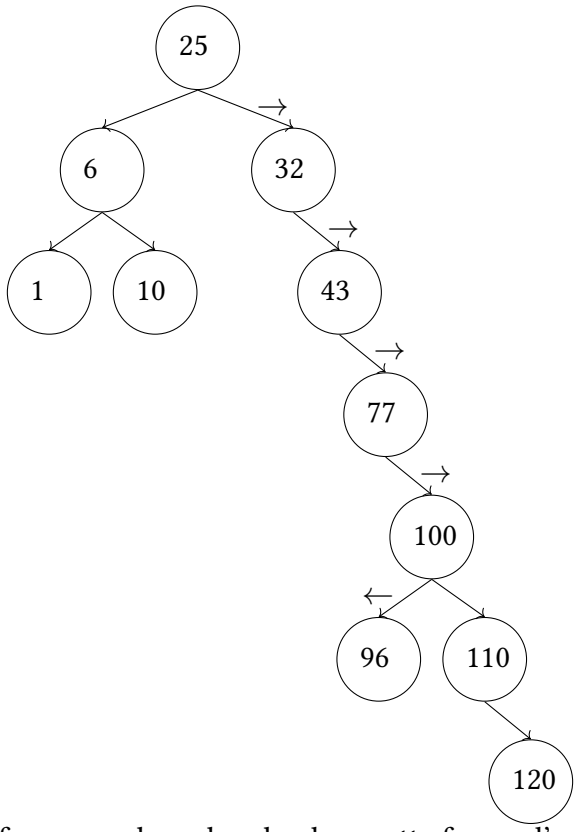


34.3.1 Parcours

La recherche de la valeur de clef 96 va emprunter les différentes branches indiquées, par comparaison de la clef avec la valeur de chaque nœud :

- vers le fils gauche si la clef est inférieure
- ou vers le fils droit si elle est supérieure.

FIGURE 13 – Parcours d'un arbre binaire de recherche (clé 96)



On voit donc que la performance de recherche dans cette forme d'arbre n'est pas garantie et dépend essentiellement de sa hauteur.

34.4 Arbres binaires de recherche équilibrés ou arbres AVL

Un arbre *AVL*[6] (en anglais : *AVL tree*) est arbre binaire de recherche équilibré dans lequel la différence de hauteur des 2 sous-arbres d'un noeud, appelé facteur d'équilibrage (facteur d'équilibrage = hauteur sous-arbre gauche − hauteur sous-arbre droit), ne soit jamais supérieure à 1, c'est-à-dire qu'elle ne peut être que l'une des 3 valeurs suivantes :

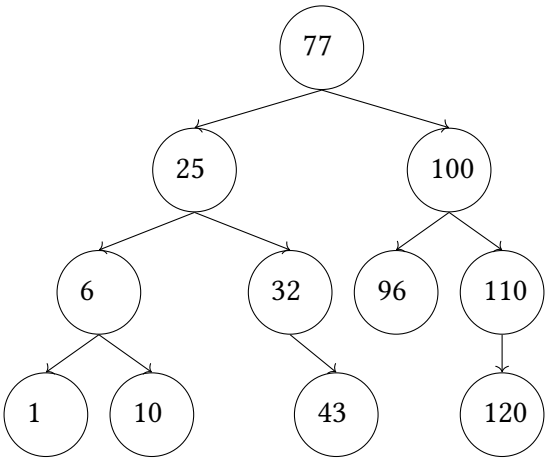
- -1 : sous-arbre droit plus haut de 1 niveau
- 0 : arbre parfaitement équilibré
- +1 : sous-arbre gauche plus haut de 1 niveau

La hauteur d'un arbre est le plus long chemin de la racine aux feuilles.

Ainsi l'arbre précédent peut être représenté sous forme équilibrée

[6] vient du nom des 2 inventeurs : Adelson-Velsky et Landis

FIGURE 14 – Arbre binaire de recherche équilibré



Le parcours infixe d'un arbre binaire de recherche fournit la séquence des clefs classées dans l'ordre croissant : 1 - 6 - 10 - 25 - 32 - 43 - 77 - 96 - 100 - 110 - 120 .

34.4.1 Équilibrage des noeuds

L'équilibrage des noeuds d'un arbre est réalisé à chaque opération d'insertion ou de suppression par un mécanisme de rotation (permutation) de 2 noeuds afin de maintenir la relation d'ordre des sous-arbres et le facteur d'équilibrage.

L'insertion d'un noeud doit s'opérer en 2 ou 3 étapes :

- placer le nouveau noeud au bon endroit, pour maintenir la relation d'ordre entre les clefs
- remonter vers les arbres supérieurs afin de contrôler si l'insertion a été la cause d'un déséquilibre
- si c'est le cas, effectuer un mouvement de rotation afin de restaurer l'équilibre de l'arbre

34.4.2 Exemple

FIGURE 15 – Insertion de la clé 2

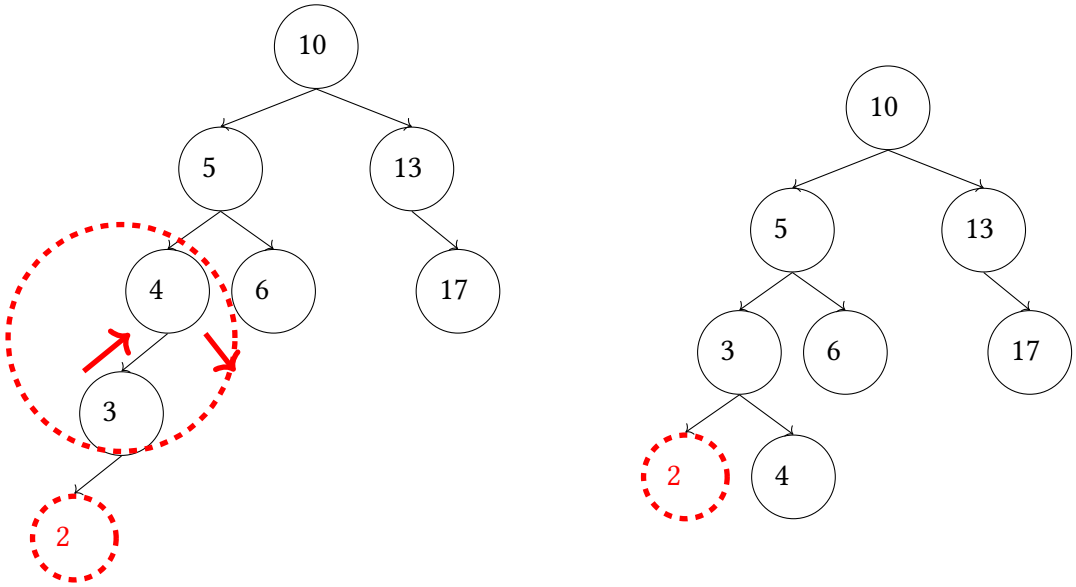


FIGURE 16 – Insertion de la clé 8

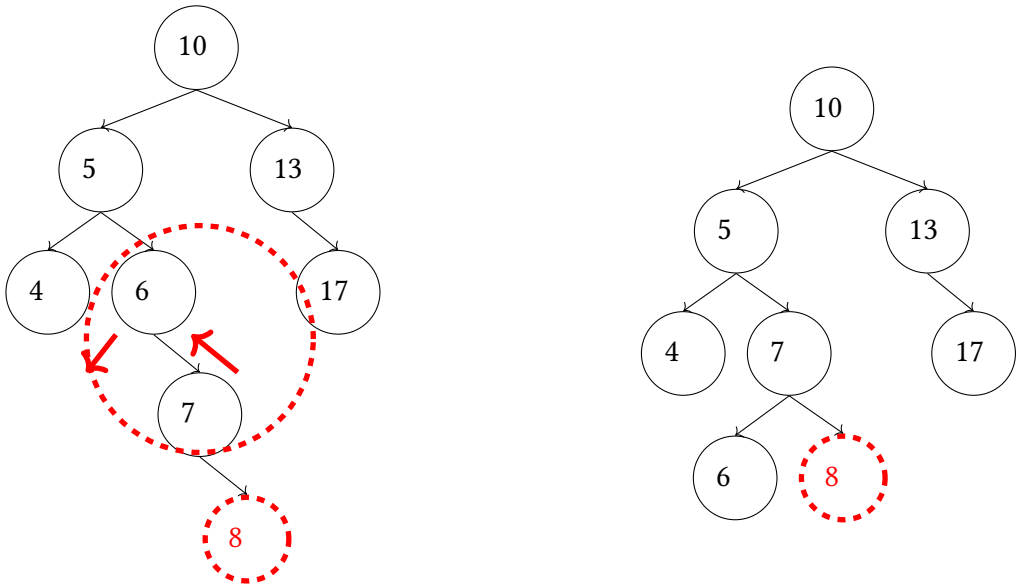
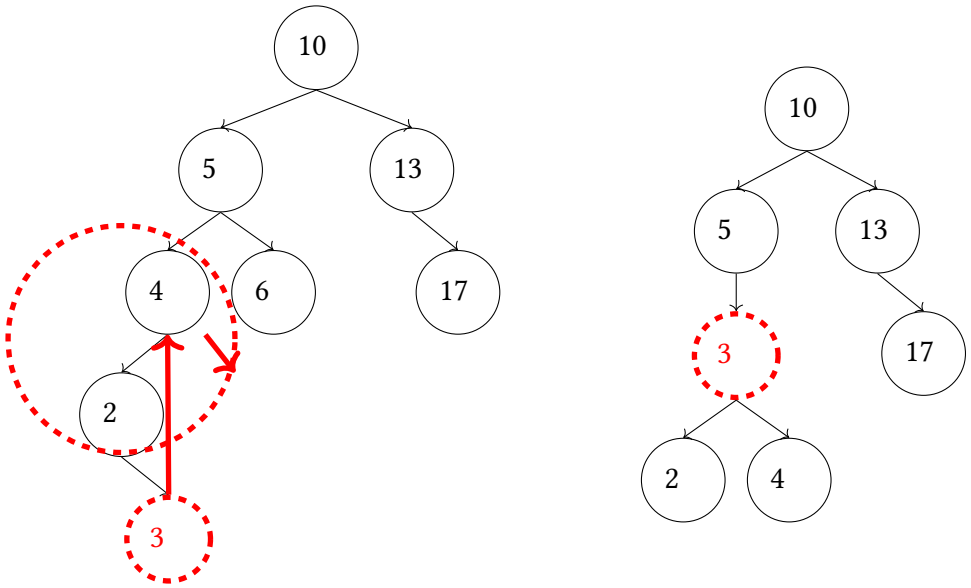


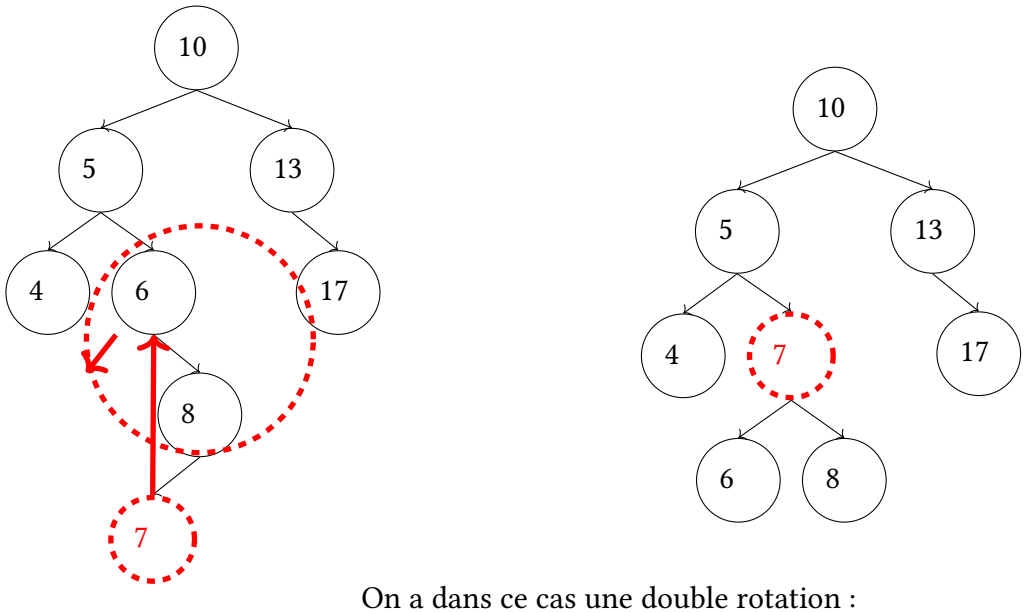
FIGURE 17 – Insertion de la clé 3



On a dans ce cas une double rotation :

- de 3 avec 2 : on revient alors dans la situation de l'exemple 1
- puis de 3 avec 4

FIGURE 18 – Insertion de la clé 7



On a dans ce cas une double rotation :

- de 7 avec 8 : on revient alors dans la situation de l'exemple 2
- puis de 7 avec 6

34.5 Application aux bases de données

34.5.1 B-arbres

Un *B-arbre* (en anglais : *B tree, Balanced Tree*) est une structure de données mise en oeuvre dans les bases de données pour la construction des index.

Le B-arbre est un arbre de recherche toujours parfaitement équilibré.

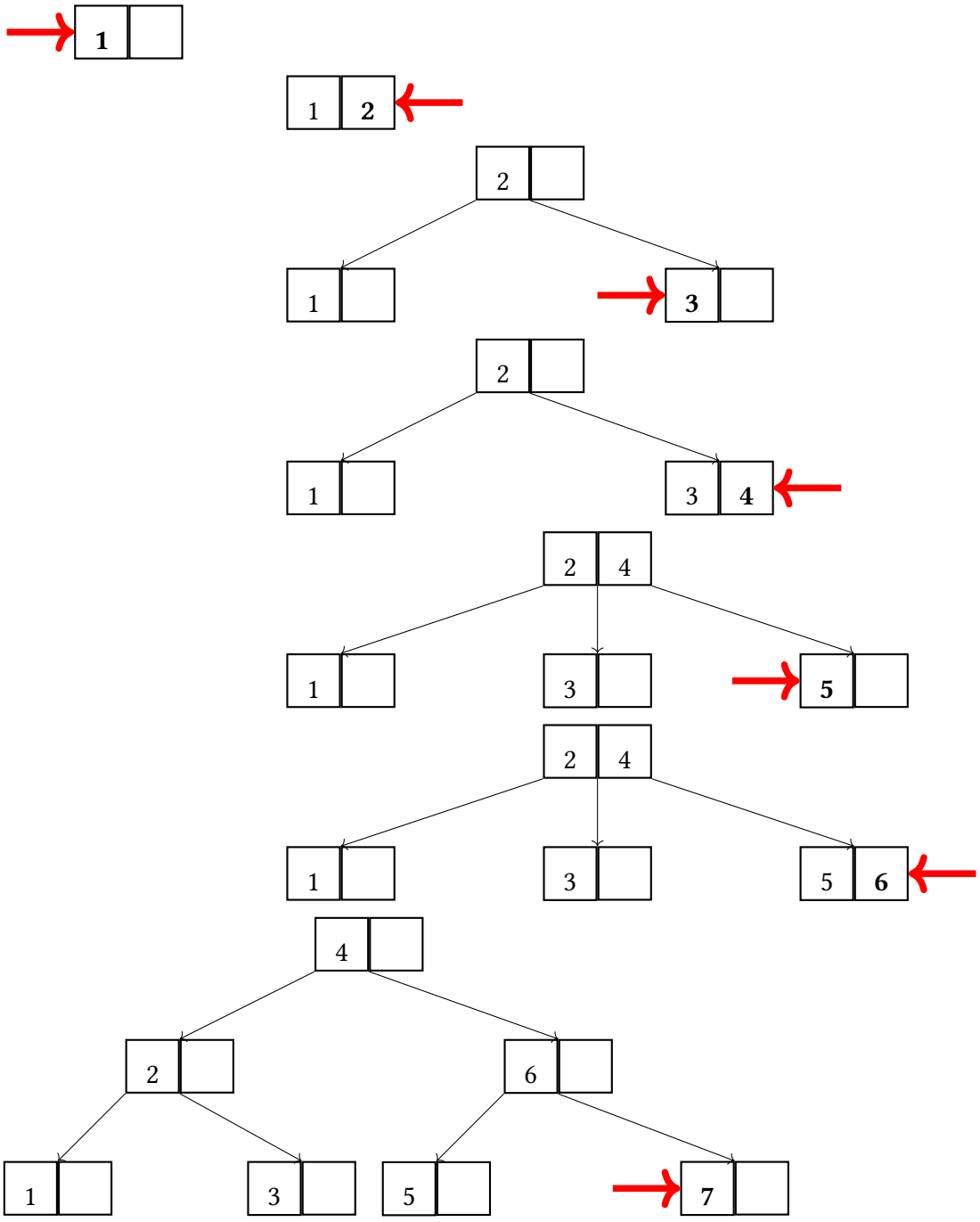
Un B-arbre peut comporter plusieurs clefs par noeuds afin d'optimiser les accès aux données liées aux clés.

Chaque noeud occupe une *page*[7] de base de données.

L'évolution de la construction du B-arbre (ici chaque noeud possède 2 clés qui définissent 3 intervalles de valeurs) montre les différentes opérations réalisées afin de maintenir l'équilibre.

[7] la page est l'unité de base de stockage des données d'une base de données ; sa capacité est de l'ordre de plusieurs kio : 16 kio est une valeur courante ; une page peut contenir les noeuds d'un index ou des données

FIGURE 19 – Évolution du B-arbre après chaque insertion de clé



Le principe d'insertion est le suivant :

- rechercher la feuille de placement, puis, récursivement :
  - si la feuille n'est pas pleine, insérer la clef à sa position normale
  - sinon :
    - laisser les clefs les plus petites dans le noeud (1ère moitié des clés)
    - créer un nouveau noeud et y placer les clefs les plus grandes (2ème moitié des clés)
    - remonter la clé médiane dans le noeud père

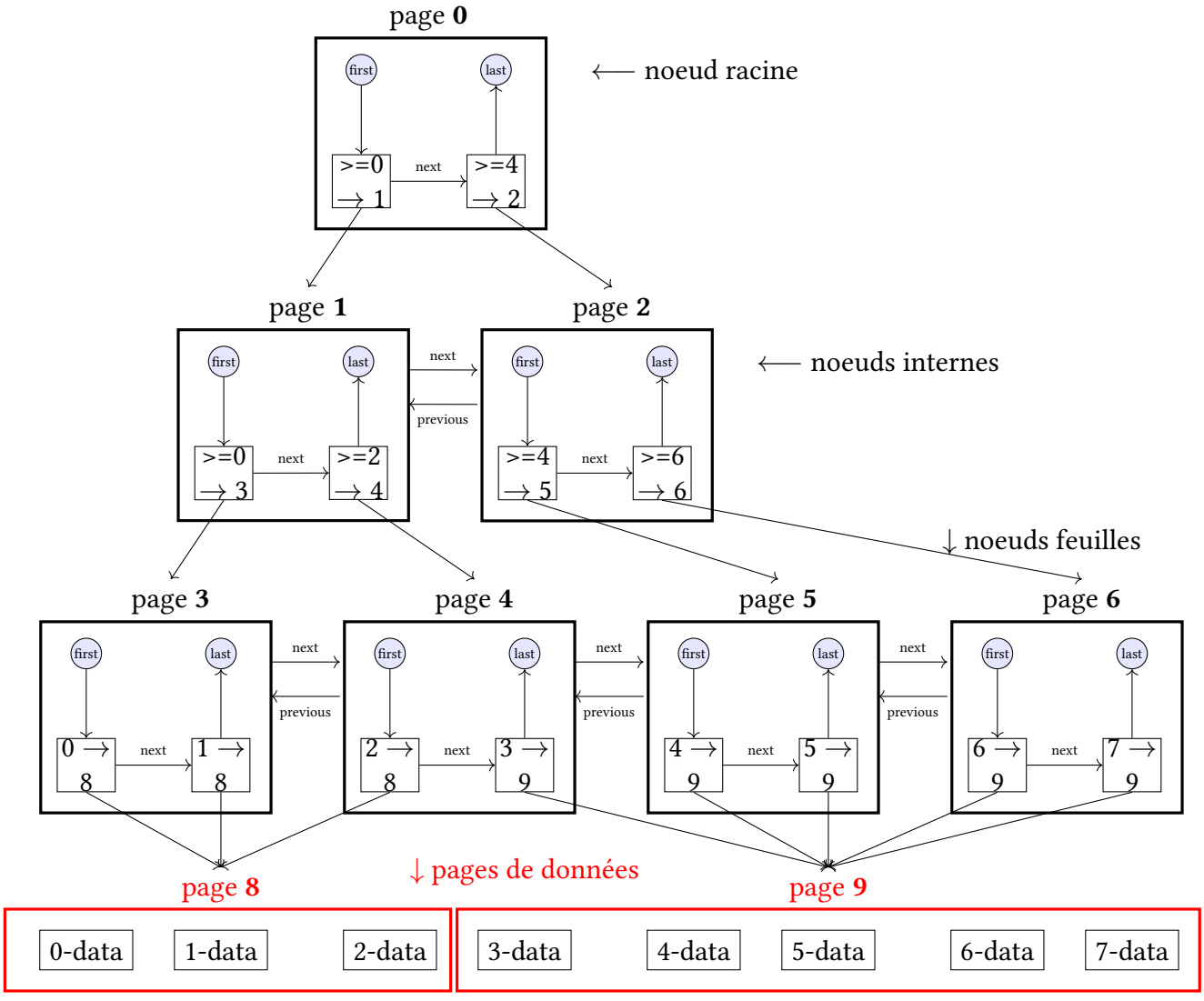
Dans un B-arbre, les noeuds internes et feuilles contiennent des clefs et les pointeurs vers les données associées.

34.5.2 Arbres B^+

Un arbre B^+ (en anglais : B^+ *tree*) est un B-arbre dans lequel

- les noeuds internes contiennent les clefs permettant la recherche
- les feuilles stockent toutes les clefs et pointent vers les données dans les pages
- les noeuds internes et feuilles sont liées par une liste doublement chainée (chainage avant et chainage arrière)
- les valeurs des clefs sont liées par une liste chainée

FIGURE 20 – Exemple de B^+



Exemple simpliste d'un index primaire :

- taille des pages : 16 kio, soit : 16*1024 octets
- taille d'un index sur une valeur de clef primaire de type **`int`** : 2*4 octets (un octet pour une valeur de clef et 1 octet pour l'adresse de la page à accéder)
- nombre de clefs d'un noeud : 16*1024 / 8 octets par clef = 2048 clefs
- en considérant un index à 2 niveaux :
  - chaque noeud peut avoir 2048 fils, chacun ayant 2048 positions de clef, soit 2048 * 2048 = 4.194.304 valeurs de clefs.
- On voit donc, qu'en 2 accès disques, on peut accéder à plus de 4 millions de valeurs de clefs.

Ainsi pour localiser une ligne de données ayant une clef K, le serveur interroge le noeud racine jusqu'à ce qu'il trouve une clef plus grande ou égale à K, puis il suit le pointeur vers le noeud fils et ainsi de suite jusqu'à arriver à un noeud feuille. Ce noeud feuille contient les pointeurs vers les pages de données où la ligne avec la clef K peut être trouvée.

35 Transactions

Une transaction est une unité de traitement qui doit être vue comme un tout : elle doit être totalement validée ou totalement annulée si un des composants échoue.

Exemple d'une transaction bancaire :

1. vérifier si le compte à débiter A existe (**`select`**),
2. débiter le compte A (**`update`**),
3. vérifier si le compte à créditer B existe (**`select`**),
4. créditer le compte B (**`update`**)

Si le 3ème partie échoue, la base de donnée est incohérente.

Les mises à jour demandées au SGBD ne devraient être effectivement enregistrées dans la base de données que si la transaction complète est validée.

Mais cette validation est parfois implicite et chaque ordre SQL est validé séparément (on parle d'*auto-commit*).

Une transaction doit avoir les propriétés suivantes

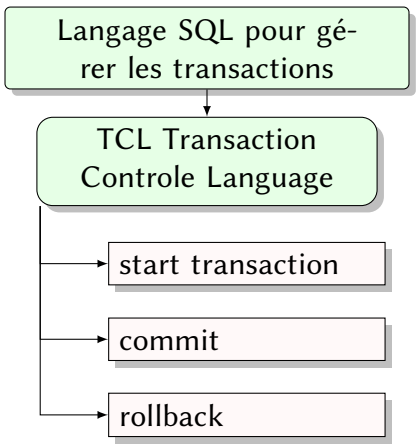
- **A**tomicité (en anglais : *atomicity*) : former un tout
- **C**ohérence (en anglais : *consistency*) : à la fin de la transaction, la base de donnée doit être à nouveau dans un état cohérent

- **I**solation (en anglais : *isolation*) : aucune autre transaction ne doit interférer (plusieurs niveaux d'isolation (en anglais : *isolation level*) sont configurables, du plus souple au plus strict)
- **D**urabilité (en anglais : *durability*) : une fois la transaction validée, celle-ci est enregistrée de manière permanente dans la base de donnée

Cet ensemble de propriétés est résumé dans l'acronyme ACID.

Le groupe d'ordre TCL permet de contrôler le déroulement des transaction comportant plus d'un ordre SQL.

FIGURE 21 – Ordres SQL associés aux transaction



Par exemple, lors de l'ajout d'une commande à un client, on souhaite insérer une ligne de commande (insert into commande ...) puis mettre à jour le client (update client set ... where ...)

35.1 Démarrer une transaction : `start transaction`

Listing 112 – Syntaxe SQL ordre **`start transaction`**

```
1 start transaction;
```

L'ordre **`set transaction`** permet de préciser le niveau d'isolation souhaité avant de démarrer une transaction.

35.2 Valider les mises à jour effectuées : `commit`

Permet de confirmer la mise à jour des données dans la base de données (depuis le dernier ordre **`commit`**)

Listing 113 – Syntaxe SQL ordre **`commit`**

```
1 commit [transaction];
```

35.3 Annuler les mises à jour effectuées : `rollback`

Permet d'annuler la mise à jour des données dans la base de données (depuis le dernier ordre **`commit`**).

Listing 114 – Syntaxe SQL ordre **rollback**

```
1 rollback [transaction];
```

35.4 Exemple

Ainsi pour assurer la validité de la transaction sur les comptes, il faut absolument gérer une transaction.

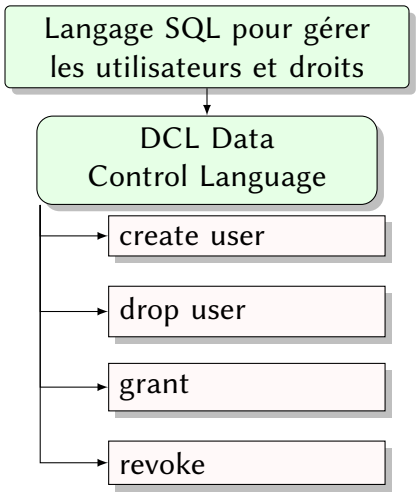
Listing 115 – exemple d'algorithme

```
1  -- demarrer une transaction
2  start transaction;
3  -- lire le compte A
4  select * from compte where idCompte = 'A';
5  --
6  si le compte existe :
7     update compte set solde = solde - montant where
         id_compte = 'A';
8     -- lire le compte B
9     select * from compte where idCompte = 'B';
10    si le compte existe :
11        update compte set solde = solde + montant where
             id_compte = 'B';
12        -- valider la transaction
13        commit;
14    sinon
15        -- invalider la transaction
16        rollback;
17    finsi
18 sinon
19    -- invalider la transaction
20    rollback;
21 finsi
```

36 Utilisateurs et privilèges

La sécurité d'accès à une base de données est liée à des comptes d'utilisateurs auxquels des privilèges d'accès peuvent être attribués.

FIGURE 22 – Ordres SQL associés aux utilisateurs et privilèges



36.1 Créer un utilisateur : `create user`

Listing 116 – Syntaxe SQL ordre **create user**

```
1 create user nom
2         identified by motPasse;
```

où :

- nom : compte de l'utilisateur sous la forme 'x'@'y' :
  - x : nom de l'utilisateur
  - y : adresse IP à partir de laquelle cet utilisateur peut se connecter (localhot ou 127.0.0.1, %, etc.)

Listing 117 – Exemple de création de l'utilisateur, lié à une connexion locale, moi avec le mot de passe (non crypté) toto

```
1 create user 'moi'@'localhost'
2         identified by 'toto';
```

36.2 Supprimer un utilisateur : `drop user`

L'ordre SQL **drop user** supprime un compte d'utilisateur et ses droits.

Listing 118 – Syntaxe SQL ordre **create user**

```
1 drop user nom;
```

Listing 119 – Exemple de suppression de l'utilisateur moi

```
1 drop user 'moi'@'localhost';
```

36.3 Attribuer des privilèges à un utilisateur : grant

Les privilèges peuvent être accordés sur une base de données, des tables ou même des colonnes.

Listing 120 – Syntaxe SQL ordre **grant**

```
1 grant privilege
2    on objets
3    to utilisateur;
```

où :

- privilege : priviléges accordés, parmi **all**, **select**, **update**, etc.
- objets : objets cibles du privilège parmi *, maBase.*, maBase.uneTable, etc.
- utilisateur : compte d'utilisateur concerné par le privilège

Listing 121 – Exemple d'attribution des droits de lecture, ajout et mise à jour des données de toutes les tables de la base mabase

```
1 grant select, insert, update
2       on maBase.*;
3       to 'moi'@'localhost';
```

36.4 Enlever des privilèges à un utilisateur : revoke

Listing 122 – Syntaxe SQL ordre **revoke**

```
1 revoke droits
2    on objets
3    from user;
```

Listing 123 – Exemple de suppression des droits de lecture, ajout et mise à jour des données de tous les tables de la base mabase

```
1 revoke insert, update
2       on mabase.*
3       from 'moi'@'localhost';
```

37 Vues utilisateurs

Une vue (en anglais : *view*) est une requête pré-enregistrée dont le résultat peut être utilisé comme une table dans une requête. Le contenu de la vue est actualisé à chaque utilisation dans une requête.

L'utilisation d'une vue présente 2 avantages :

- elle est plus performante que la même requête soumise au SGBD car elle a déjà été vérifiée et optimisée
- elle permet la gestion plus fine des droits d'accès aux données en donnant des droits sur une vue (certaines colonnes) plutôt que sur une table complète (toutes ses colonnes)

Pour créer une vue (ou la remplacer si elle existe déjà) :

```
1 CREATE OR REPLACE VIEW nomVue
2   AS
3   requeteSQL;
```

où :

- *nomVue* : le nom donné à la vue
- *requeSQL* : toute requête SQL valide (il existe certaines restrictions dépendant des versions du SGBD)

Exemple :

```
1 CREATE OR REPLACE VIEW lesSalaries
2   AS
3   SELECT matricule, nom, prenom
4     FROM salarie;
```

Utilisation de la vue 'lesSalaries' :

```
1 SELECT *
2     FROM lesSalaries
3     WHERE matricule < 1000
4     ORDER BY nom, prenom;
```

En MySQL, il est possible d'utiliser les ordres DML sur des vues à condition que le résultat soit en concordance avec le contenu de la table

38 Procédures et fonctions stockées

Les SGBD offrent des possibilités d'extension au déjà puissant langage SQL. Ils mettent à disposition un langage de programmation et différents objets qui permettent de l'utiliser.

Les procédures stockées sont des jeux de commandes stockés en tant qu'objets dans une base de données :

- Les procédures et fonctions : de la même manière que tout langage programmation, on peut écrire des sous-programmes pour effectuer des actions trop complexes pour être réalisées en SQL,
- Les déclencheurs : mécanisme qui permet l'exécution d'un bloc d'instructions lorsqu'un évènement de modification du contenu d'une table se produit.

Le code de ces procédures est pré-compilé.

38.1 Langage procédural

Comme tout langage procédural, le langage intégré au SGBD utilise :

- des déclarations de variables,
- des types de données : types de base du SGBD, types spécifiques pour manipuler les lignes des tables, etc.,
- des opérateurs d'affectation de valeurs,
- des opérateurs de calcul d'expressions ,
- des instructions de parcours du résultat d'une requête (curseur)
- des structures de contrôles conditionnelles et répétitives,
- une gestion d'exception avec des instructions permettant de gérer les erreurs.

La structure d'un bloc d'instructions est la suivante :

Syntaxe : bloc d'instructions

```
1 [etiquette:] BEGIN
2     [declarations]
3     [instructions]
4 END [etiquette]
```

où :

- etiquette : permet de nommer un bloc d'instructions
- déclarations : déclaration des variables utilisées dans le bloc (ou les blocs imbriqués),
- instructions : instructions du bloc ; un bloc peut comporter des blocs imbriqués).

38.1.1 DECLARE variable

La commande **DECLARE** permet la déclaration de variables utilisées dans le corps d'un bloc d'instructions.

Syntaxe : DECLARE variable

```
1 DECLARE nomVariable[,...] type [DEFAULT valeurDefaut]
```

où :

- nomVariable : nom donné à la variable,
- type : un type de données SQL valide,
- valeurDefaut : valeur par défaut de la variable.

Exemple : DECLARE variable

```
1 DECLARE nom VARCHAR(20);
2 DECLARE prenom VARCHAR(20);
3 DECLARE nombre INT DEFAULT 0;
```

38.1.2 DECLARE ...CURSOR

La commande **DECLARE ...CURSOR** déclare un curseur [8] qui permet le parcours d'une liste de lignes renvoyées par une requête SQL.

Les déclarations de curseurs doivent se trouver après les déclarations des variables.

Syntaxe : DECLARE ...CURSOR

```
1 DECLARE nomCurseur CURSOR FOR requeteSQLSelect;
```

où :

- nomCurseur : nom donné au curseur,
- requeteSQLSelect : requete SQL SELECT valide.

Exemple : un curseur de parcours de la liste des membres

```
1 DECLARE curs1 CURSOR
2     FOR SELECT id_memb, nom_memb, prenom_memb
3     FROM assocArt.membre;
```

[8] un curseur est une variable pointant vers une des lignes d'un jeu de données ((en anglais : *recordset*))

38.1.3 DECLARE ...HANDLER

La commande **DECLARE ...HANDLER** déclare une variable de gestion des exceptions engendrées durant l'exécution d'une procédure.

Ces déclarations doivent se trouver au dessous des déclarations de curseurs.

38.1.4 SET

La commande **SET** permet l'affectation d'une valeur à une variable.

Syntaxe : DECLARE

```
1 SET variable = expression [,var2 = expr2,...]
```

où :

- variable, var2 : nom de la (ou des) variable concernée,
- expression, expr2 : expression affectée à la variable.

Exemple : affecter 0 à l'entier 'nombre'

```
1 SET nombre = 0;
```

38.1.5 SELECT...INTO

La commande **SELECT...INTO** permet d'affecter à une variable le résultat de l'exécution d'une requête SELECT.

La requête ne doit retourner qu'une valeur élémentaire (1 colonne, 1 ligne). En cas de retour d'un résultat vide, le contenu des variables reste inchangé.

Syntaxe : SELECT...INTO

```
1 SELECT nomColonne[,...]
2     INTO variable[,...]
3     tableExpression
4     [LIMIT 1]
```

où :

- nomColonne : nom de la (ou des) colonne source,
- variable : nom de la (ou des) variable receptrice,
- tableExpression : origine des valeurs, soit la définition d'une requête SELECT à partir de la clause FROM,
- LIMIT 1 : limite le nombre de lignes retournées à 1.

Exemple : affecter une valeur provenant de 'membre' à 'nom' et 'prenom'

```
SELECT nom_memb, prenom_memb
    INTO nom,prenom
    FROM assocArt.membre
    WHERE id_memb < 10
    LIMIT 1;
```

38.1.6 Structure conditionnelle IF

La structure de contrôle **IF** permet d'exécuter un bloc d'instructions si une condition est remplie.

Syntaxe : DECLARE

```
IF condition THEN instructions
    [ELSEIF condition THEN instructions] ...
    [ELSE instructions]
END IF;
```

où :

- condition : expression booléenne,
- instructions : groupe d'instructions.

Exemple : tester si une i est égal à j

```
DECLARE i INT DEFAULT 1;
DECLARE j INT DEFAULT 2;
DECLARE rep VARCHAR(20);
IF i = j THEN   SET rep = "egal";
    ELSEIF i < j THEN SET rep = "inf";
    ELSE SET rep = "sup";
END IF;
```

38.1.7 Structure répétitive LOOP et LEAVE

La structure de contrôle **LOOP . . . END LOOP** permet de répéter un bloc d'instructions et de l'interrompre par **LEAVE**.

Syntaxe : DECLARE

```
[etiquette:] LOOP
    instructions (avec LEAVE)
END LOOP [etiquette]
```

Exemple : boucler 10 fois

```
DECLARE i INT DEFAULT 1;
boucle: LOOP
    -- instructions
    SET i = i + 1;
    IF i > 10 THEN LEAVE boucle; END IF;
END LOOP boucle;
```

38.1.8 Structure répétitive REPEAT...UNTIL

La structure de contrôle **REPEAT...UNTIL** permet de répéter un groupe d'instructions jusqu'à ce qu'une condition soit vraie.

Syntaxe : REPEAT

```
[etiquette:] REPEAT
    instructions
UNTIL conditionArret
END REPEAT [etiquette]
```

Exemple : executer 10 fois

```
DECLARE i INT DEFAULT 10;
REPEAT
    -- instructions
    SET i = i - 1;
    UNTIL i <=0
END WHILE;
```

Exemple : executer 10 fois

```
DECLARE i INT DEFAULT 0;
REPEAT SET i = i + 1;
  UNTIL i > 10
END REPEAT;
```

38.1.9 Structure répétitive WHILE...DO

La structure de contrôle **WHILE DO...** permet de répéter un groupe d'instructions tant qu'une condition est vraie.

Syntaxe : WHILE...DO

```
[etiquette:] WHILE condition DO
    instructions
END WHILE [etiquette]
```

Exemple : executer 10 fois

```
1 DECLARE i INT DEFAULT 10;
2 WHILE i > 0 DO
3     -- instructions
4     SET i = i - 1;
5 END WHILE;
```

38.1.10 OPEN, FETCH, CLOSE

Ces 3 commandes permettent l'accès à un jeu de données en utilisant un curseur :

- **OPEN** ouvre un jeu de données et positionne le curseur au début.
- **FETCH** permet de positionner un curseur sur la ligne suivante du jeu de données.
- **CLOSE** ferme un jeu de données.

Exemple : parcourir 'membre' et recopier les numéros dans 'listeMembre' : LOOP

```
1  DELIMITER $$
2  CREATE PROCEDURE recopier()
3  CONTAINS SQL
4  BEGIN
5    DECLARE done INT DEFAULT 0;
6    DECLARE n INT;
7    DECLARE curs1 CURSOR FOR SELECT id_memb FROM assocArt.
       membre;
8    -- gestion de la fin du jeu de lignes
9    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
10
11   OPEN curs1;
12
13   lecture: LOOP
14     FETCH curs1 INTO n;
15     IF done THEN
16       LEAVE lecture; -- sortie en fin de 'set'
17     END IF;
18     INSERT INTO assocArt.listeMembre VALUES (n);
19   END LOOP lecture;
20
21   CLOSE curs1;
22 END;
23 $$
24 DELIMITER ;
```

Remarque : **DELIMITER** permet de préciser le marqueur de fin d'instruction SQL pour éviter la prise en compte des ';' ausein de la création de la procédure.

Exécution par :

```
call recopier;
```

Pour lister les procédures :

```
select name, db from mysql.proc;
```

Exemple : parcourir 'membre' et recopier les numéros dans 'listeMembre' : REPEAT

```
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE n INT;
  DECLARE curs1 CURSOR FOR SELECT id_memb FROM assocArt.
     membre;
  -- gestion de la fin du jeu de lignes
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    OPEN curs1;
    REPEAT
        FETCH curs1 INTO n;
        IF NOT done THEN
            INSERT INTO assocArt.listeMembre VALUES (n);
        END IF;
    UNTIL done END REPEAT;

    CLOSE curs1;
END;
```

38.2 Procédures et fonctions stockées

Les procédures et fonctions stockées sont des objets de bases de données permettant d'enregistrer des blocs d'instructions.

Une fonction est une forme d'expression : elle peut être utilisée partout où est utilisée une expression dans les requêtes SQL. Une fonction comporte un bloc d'instructions stocké dans la base de données. Cette fonction renvoie une valeur en utilisant l'instruction **RETURN**.

38.3 Créer une procédure : CREATE PROCEDURE

Syntaxe : CREATE PROCEDURE

```
CREATE PROCEDURE nomProc
    ([[ IN | OUT | INOUT ] nomParam1 typeParam1[,...]])
    [options...]
    corpsProc
```

où :

- nomProc : nom de la procédure
- IN ou OUT ou INOUT : mode de passage du paramètre 1 : entrée, sortie(résultat), entrée-sortie
- nomParam1 : nom du paramètre 1
- typeParam1 : type de données du paramètre 1, tout type de données valide pour MySQL ;
- options : précisions concernant la requête (cf. ci-dessous) ;
- corpsProc : corps de la procédure.

Syntaxe : options de la procedure

```
options:
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL
      DATA }
  | SQL SECURITY { DEFINER | INVOKER }
```

où :

- Deterministic = fonction pure, qui retourne toujours le même résultat en fonction du parametre passé.

Exemple : Créer une procédure

```
CREATE OR REPLACE PROCEDURE suppActivite (numActiv IN
    INTEGER)
    AS
    BEGIN
        DELETE FROM visiter WHERE id_activ = numActiv ;
        DELETE FROM inscrire WHERE id_activ = numActiv ;
        DELETE FROM activite WHERE id_activ = numActiv ;
    END ;
```

38.4 Utiliser une procedure

Exemple : Exécuter dans un autre sous-programme procedure ou fonction

```
CREATE OR REPLACE PROCEDURE suppActivite1 ()
    AS
    DECLARE
        uneActivite INTEGER;
    BEGIN
        uneActivite := 1 ;
        suppActivite (uneActivite) ;
    END;
```

Syntaxe : EXECUTE nomProc

```
EXECUTE nomProc param1, param2,..., paramN;
```

Exemple : Exécuter dans un autre langage

```
EXECUTE suppActivite 1 ;
```

38.5 Supprimer une procedure : DROP PROCEDURE

Syntaxe : DROP PROCEDURE

```
DROP PROCEDURE nomProc ;
```

où :

- nomProc : nom de la procédure

38.6 Créer une fonction : CREATE FUNCTION

Syntaxe : CREATE FUNCTION

```
CREATE FUNCTION nomFonc
    ([[ IN | OUT | INOUT ] nomParam1 typeParam1[,...]])
    RETURNS typeRetour
    [options ...] corpsFonc
```

où :

- nomFonc : nom de la fonction
- IN ou OUT ou INOUT : mode de passage du paramètre 1 : entrée, sortie(résultat), entrée-sortie
- nomParam1 : nom du paramètre 1
- typeParam1 : type de donnée du paramètre 1, tout type de données valide pour MySQL ;
- typeRetour : type de la donnée renvoyée par la fonction, tout type de données valide pour MySQL ;
- options : précisions concernant la requête (cf. procedure) ;
- corpsFonc : corps de la fonction, tout type d'instruction SQL valide pour MySQL

Exemple : fonction 'bonjour'

```
CREATE FUNCTION bonjour (s CHAR(20))
    RETURNS CHAR(50) DETERMINISTIC
    RETURN CONCAT('Bonjour ',s,'!');
```

38.7 Utiliser une fonction

Exemple : utiliser la fonction 'bonjour'

```
1 SELECT bonjour(prenom)
2     FROM membre;
```

Exemple : fonction 'calculTTC'

```
1 CREATE FUNCTION calculTTC (montantHT IN NUMBER)
2     RETURNS NUMBER DETERMINISTIC
3     RETURN (montantHT * 1.196);
```

Exemple : tester la fonction 'calculTTC' avec DUAL

```
1 SELECT calculTTC(100)
2     FROM DUAL ;
```

Le nom DUAL représente unetable fictive permettant l'appel de fonctions système tout en conservant la syntaxe de base du SELECT.

Exemple : utiliser la fonction 'calculTTC'

```
1 SELECT id_activ, intitule_activ, tarif1_activ, COUNT(*) as
    Nombre,
2  calculTTC(tarif1_activ * COUNT(*)) AS MontantTTC
3     FROM activite NATURAL JOIN inscrire
4     GROUP BY id_activ, intitule_activ, tarif1_activ;
```

Les fonctions peuvent être utilisées pour simuler (de manière très complexe) les vues paramétrées (non disponibles dans MySQL). Exemple : une fonction paramétre (le premier paramètre)'p1'

```
1 CREATE FUNCTION p1()
2   RETURNS INTEGER
3   DETERMINISTIC
4   NO SQL
5   RETURN @p1;
```

Cette fonction retourne la valeur d'une variable p1.

Exemple : une vue paramétrée utilisant 'p1'

```
1 CREATE VIEW lesMembres
2   AS
3   SELECT * FROM acteur
4   WHERE num_act = p1();
```

La vue utilise la fonction.

Exemple : L'utilisation de la vue paramétrée

```
1 SELECT M.*
2     FROM  lesMembres M,
3       (SELECT @p1:=1 param) P;
```

On affecte à une variable p1 la valeur souhaitée.

Ainsi, la fonction utilisée dans la vue aura accès à la valeur du paramètre.

38.8 Supprimer une fonction : DROP FUNCTION

Syntaxe : DROP FUNCTION

```
1 DROP FUNCTION nomFonc ;
```

où :

- nomFonc : nom de la fonction

39 Déclencheurs, ou triggers - contrôles d'intégrité

L'intégrité des données d'une base de données est assurée par les contraintes d'intégrité d'entité (clef primaire, (en anglais : ***primary key***)) et d'intégrité référentielle (clef étrangère, (en anglais : ***foreign key***)). La contrainte d'unicité complète ces vérifications en s'assurant d'une colonne pourra ou non comporter des valeurs en double (unique index).

Les domaines de valeurs associés au modèle relationnel n'ont pu être traduits complètement : les types de données associés parfois à la contrainte **check** ne peuvent les représenter aussi précisément.

Cependant certaines contraintes d'intégrité présentes dans la modélisation des données (MCD Merise, par exemple), ou certaines règles de gestion (en anglais : *business rules*), n'ont pu être traduites dans le modèle relationnel :

- des domaines de valeurs comme une liste de valeurs comprises entre des bornes qui nécessitent un accès à d'autres tables
- des valeurs de clefs à vérifier (exemple clef RIB)
- une cardinalité maximale n sous forme d'une valeur littérale précise : 1 étudiants peut s'inscrire à un nombre d'options de 1 à 3
- mettre en oeuvre les contraintes sur associations du modèle conceptuel (un étudiant qui passe un examen doit d'abord avoir validé son inscription)

Les déclencheurs (en anglais : *triggers*), complètent les possibilités de maintien de l'intégrité des données d'une base de données.

Ils permettent également de répondre à d'autres besoins :

- contrôler des droits d'accès spécifiques aux données
- mettre en oeuvre la réplication[9] de données vers d'autres bases de données
- tracer les mises à jour effectuées sur les tables (audit des modifications de données)

Un déclencheur est un bloc de code associé à une table et déclenché ((en anglais : *fired*)) lorsqu'une instruction DML (**INSERT**, **UPDATE**, **DELETE**) est exécutée sur la table.

Il donne accès au contenu en cours de modification juste avant qu'il soit modifié et juste après qu'il l'ait été.

Il offre un moyen :

- de définir des contraintes d'intégrité complexes, ou spécifiques, que les contrainte de colonnes ou de tables n'ont pu prendre en compte,
- d'effectuer des traitements complémentaires lors de l'exécution d'un ordre DML : mettre à jour des valeurs cumulées, journaliser les mises à jour d'informations sensibles, etc.

39.1 Créer un déclencheur : CREATE TRIGGER

La commande SQL **CREATE TRIGGER** (ou **CREATE OR REPLACE TRIGGER**) permet la définition d'un déclencheur.

Syntaxe : CREATE TRIGGER

```
1 CREATE TRIGGER nomDeclencheur
2     {BEFORE|AFTER}    {INSERT|UPDATE|DELETE}
3         ON nomTable
4         FOR EACH ROW
5             instructions
```

Le corps de la procédure peut faire référence aux anciennes et nouvelles valeurs des lignes mises à jour en utilisant les préfixes **OLD** et **NEW**.

39.1.1 Exemple : audit de table

On souhaite conserver une trace des actions sur la table des membres. Chaque insertion, mise à jour ou suppression de ligne dans la table 'membre' va ajouter une ligne dans la table 'auditmembre'.

Exemple : Création de la table d'audit des membres

[9] la réplication consiste à dupliquer des informations vers d'autres bases de données/d'autres serveurs afin, par exemple, de répartir les accès sur plusieurs serveurs (moins de problèmes de concurrence d'accès), et ainsi d'accroitre la performance d'accès

```sql
CREATE TABLE auditMembre (
    utilisateur     VARCHAR(40),
    date_maj    DATETIME    NOT NULL,
    type_maj    CHAR(4) NOT NULL,
    id_memb     INTEGER     NOT NULL,
    nom_memb_av     VARCHAR(20),
    prenom_memb_av  VARCHAR(20),
    nom_memb_ap     VARCHAR(20),
    prenom_memb_ap  VARCHAR(20)
);
```

Cette table ne comporte par de contrainte d'intégrité, elle sert simplement de trace des modifications.

Exemple : Triggers sur la table 'membre'

```sql
--
-- auditer les mises  jour sur les membres
--
-- ajouter un membre
--
DROP TRIGGER IF EXISTS membre_insert;

CREATE TRIGGER membre_insert
  AFTER INSERT
       ON membre
       FOR EACH ROW
  INSERT INTO auditMembre (utilisateur,date_maj,type_maj,
     id_memb,
       nom_memb_av,prenom_memb_av,nom_memb_ap,prenom_memb_ap
          )
      VALUES (USER(), now(), "crea", NEW.id_memb ,
       NULL, NULL, NEW.nom_memb, NEW.prenom_memb) ;
--
-- modifier un membre
--
DROP TRIGGER IF EXISTS membre_update;

CREATE TRIGGER membre_update
  AFTER UPDATE
       ON membre
       FOR EACH ROW
  INSERT INTO auditMembre (utilisateur,date_maj,type_maj,
     id_memb,
       nom_memb_av,prenom_memb_av,nom_memb_ap,prenom_memb_ap
          )
      VALUES (USER(), now(), "majo", NEW.id_memb ,
       OLD.nom_memb,OLD.prenom_memb, NEW.nom_memb, NEW.
```

```
            prenom_memb) ;
29 --
30 -- supprimer un membre
31 --
32 DROP TRIGGER IF EXISTS membre_delete;
33
34 CREATE TRIGGER  membre_delete
35   AFTER DELETE
36       ON membre
37       FOR EACH ROW
38   INSERT INTO auditMembre (utilisateur,date_maj,type_maj,
        id_memb,
39       nom_memb_av,prenom_memb_av,nom_memb_ap,prenom_memb_ap
            )
40      VALUES (USER(), now(), "supp", OLD.id_memb ,
41     OLD.nom_memb,OLD.prenom_memb, NULL, NULL) ;
```

39.2 Lister les déclencheurs : SHOW TRIGGERS

Exemple : lister les triggers

```
1 SHOW TRIGGERS;
```

39.3 Supprimer un déclencheur : DROP TRIGGER

Syntaxe : DROP TRIGGER

```
1 DROP TRIGGER [IF EXISTS] [nomBD.]nomDeclencheur;
```

40 Organisation physique des données : page

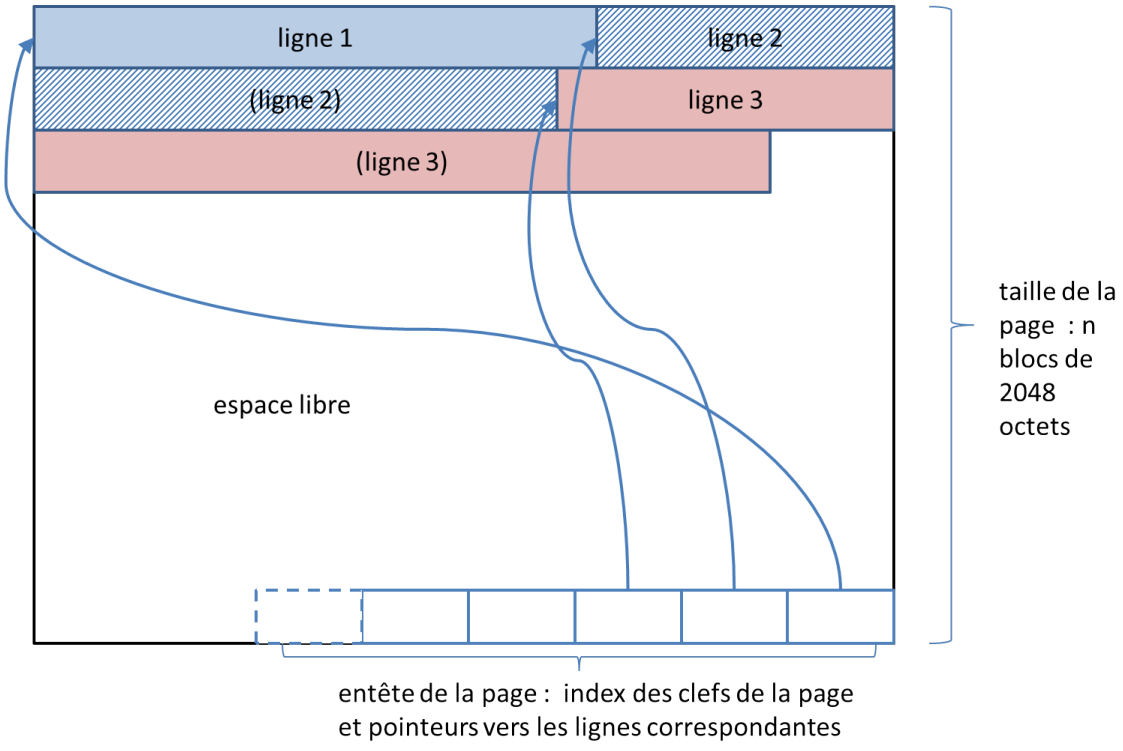
L'organisation des données sur les supports de persistance comme le disque dur :

- des pistes concentriques forment des cylindres (formé par une piste sur les plateaux d'un disque dur)
- des secteurs forment l'unité d'accès aux données stockées : un secteur permet le stockage de 512 octets.
- des blocs regroupent 4 à 8 secteurs, soit 2048 à 4096 octets, afin d'optimiser l'accès disque (la taille d'un fichier sera un multiple de 2048)

Afin d'améliorer les performances d'accès aux données, les SGBDs ont une structure logique de page comportant plusieurs blocs. Lors d'un accès physique, plusieurs blocs consécutifs formant une page seront récupérés.

Les pages relatives aux index comportent une organisation sous forme d'arbres dont les noeuds comportent des clefs et les pages où se trouvent les lignes correspondantes.

Les pages relatives aux données comportent un index des clefs et la localisation de la ligne dans la page.



Sixième partie

Bases de données et législation

La production de bases de données destinées à devenir un service doivent être protégées. Mais celles-ci peuvent contenir des informations à caractère personnel qui doivent être protégées.

40.1 Droit des producteurs de bases de données

Les bases de données comme support d'informations stratégiques sont protégées par le droit pour les producteurs de ces bases (droit d'auteur). Ces droits leur permettent d'amortir les investissements nécessaires à leur réalisation.

Informations complémentaires :
http://www.les-infostrateges.com/article/0705296/le-droit-des-bases-de-donnees

40.2 Informations à caractère personnel

Les bases de données peuvent contenir des informations susceptibles d'être protégées, par exemple, les données personnelles. Elles sont donc soumises, en autres, à la loi *informatique, fichiers et libertés de 1978*. En France, la CNIL, Commission Nationale Informatique et Liberté, est l'organisme chargé de la protection des données personnelle.

40.3 Le droit : un élément à prendre en compte

Lors de la création d'une base de données, il est donc nécessaire d'avoir une réflexion qui permettra de connaitre les implications au niveau législatif.

Septième partie

Annexes

41 Compléments à la création des tables

41.1 Installer un SGBDR sur son ordinateur

Vous pouvez installer un SGBDR sur votre ordinateur (sous Windows XP ou une distribution de Linux), par exemple :

- *PostgreSQL* (libre) et l'outil d'administration *pgAdmin* (qui offre une aide SQL pour PostgreSQL très complète) : solution complète, riche en possibilités, aide SQL, etc.
- *SQL Server*(Microsoft) Express Edition (seulement sous Microsoft Windows...)
- *Oracle10i* (Oracle) : téléchargeable
- IBM *DB2* (IBM) : téléchargeable
- *MySQL* (Oracle) : intégré dans un pack ? EasyPHP - séparément avec des outils d'administration

Vous pourrez construire des bases de données puis des applications qui pourront y accéder.

41.2 Consulter des ressources complémentaires

Une ressource Web très complète pour les développeurs : http://sqlpro.developpez.com/.

Un forum en anglais où vous pourrez trouver toutes les réponses à vos questions (il faudra les exprimer de manière correcte en anglais) : http://dba.stackexchange.com/

42 Métiers liés aux bases de données

Par exemple :

- administra-teur(-trice) de bases de données (en anglais : *database administrator, DBA*) : responsable de la performance, de l'intégrité et de la sécurité des bases de données (moyennes et grandes entreprises), https://www.prospects.ac.uk/job-profiles/database-administrator
- gestionnaire de bases de données (en anglais : *database administrator*) (petites et moyennes entreprises)

43 Exemples d'intégration du SQL en programmation

43.1 Exemple en Python avec MySQL

Dans un premier temps, télécharger puis installer le connecteur Python édité par MySQL (Choisir le 'Platform Independant' en cas de problème)

Puis écrire et tester votre code source.

```
1  import mysql.connector
2
3  def lister(c):
4      # parcourir les lignes (rows) du jeu de donnees renvoye
5      for row in cur.fetchall() :
6          # afficher les lignes (ou les colonnes separement)
7          print row # row[0], row[1]
8
9
10 # ouvrir une connexion avec le SGBD
11 cnx = mysql.connector.connect(user='root', password='xxxx',
12                                host='127.0.0.1', # SGBD
                                      local
13                                database='mabase')
14
15 # creation d'un variable qui reference les donnees echange
       es avec la connexion
16 cur = cnx.cursor()
17
18 # definir la chaine de requete
19 req = "select * from acteur where num_act between 1 and 10;
       "
20 # soumettre une requete SQL au SGBD et recuperer le
       resultat
21 cur.execute(req)
22 # appeler la fonction de liste
23 lister(cur)
24
25 # demander un nom a l'utilisateur
26 unNom = raw_input("Entrez une partie du nom cherche :");
27 # definir la chaine de requete
28 req = "".join(["select nom, prenom from acteur where
       nom_act like ",'"%',unNom,'%"',";"])
29 print req
30 # soumettre une requete SQL au SGBD et recuperer le
       resultat
31 cur.execute(req)
32 # appeler la fonction de liste
```

```
33 lister(cur)
34
35
36 # fermer la connexion
37 cnx.close()
```

43.2 Exemple en PHP avec MySQL

Les pilotes MySQL sont généralement intégrés aux distribution PHP-MySQL.

Listing 124 – page Web simpliste

```
1 <html><head><title>test</title></head><body>
2 <?php
3 // ****** DEBUT DE PARTIE A PROTEGER (mot de passe visible)
4 //on effectue la connexion
5   $cnx = mysqli_connect("127.0.0.1","root","xxxx", "mabase"
      );
6 // ****** FIN DE PARTIE A PROTEGER
7
8 // declaration et initialisation de la requete
9 $req="select nom, prenom from acteur where num_act between
    1 and 10;";
10 // soumpission de la requete au serveur et recuperation du
    resultat
11 $result=mysqli_query($cnx, $req) or die(mysqli_error($cnx))
    ;
12 // recuperation du nombre de lignes du resultat
13 $nb=mysqli_num_rows($result);
14 echo "$nb ligne(s) renvoyees";
15 // exploitation du resultat
16 while($row = mysqli_fetch_row($result))
17 {
18     $nom = $row[0];
19     $prenom = $row[1];
20     echo "<br>$nom $prenom";
21 }
22 // liberation des resultats
23 mysqli_free_result($result);
24 // fermeture de la connexion
25 mysqli_close($cnx);
26 ?>
27 </body></html>
```

43.3 Exemple en C avec MySQL

Dans un premier temps, télécharger puis installer le connecteur C édité par MySQL.

Dans le répertoire du projet C, sont présents :

- le répertoire des fichiers d'entêtes des fonctions d'accès à MySQL
- le fichier libmysql.dll
- le fichier libmysqlclient.a

```c
#include <stdio.h>
#include <stdlib.h>
#include <winsock.h>
#include "MYSQL/mysql.h"

int main(void)
{
    // Déclaration d'un enregistrement de type MYSQL
    MYSQL conn;
    // Initialisation des données de la variable précédente
    mysql_init(&conn);
    //Options de connexion
    mysql_options(&conn,MYSQL_READ_DEFAULT_GROUP,"option");

    // si la connection s'est bien passée
    if (mysql_real_connect(&conn,"localhost","root","xxx","
       mabase",0,NULL,0))
    {
        // soumission de la requête au SGBD
        mysql_query(&conn, "select nom, prenom from acteur
           where num_act between 1 and 10;");

        // Déclaration des variables résultat
        MYSQL_RES *result = NULL;
        MYSQL_ROW row;

        unsigned int i = 0;
        unsigned int num_champs = 0;

        // affecter à result le jeu de données renvoyé par
           le SGBD
        result = mysql_use_result(&conn);

        // récupérer le nombre de colonnes du jeu de ré
           sultat
        num_champs = mysql_num_fields(result);
```

```
33
34          // tant qu'il y a des lignes dans le résultat
35              // affecter à la variable row la prochaine ligne
                    du jeu de résultat
36          while ((row = mysql_fetch_row(result))) {
37              //On déclare un pointeur long non signé pour y
                    stocker la taille des valeurs
38              unsigned long *lengths;
39
40              //On stocke ces tailles dans le pointeur
41              lengths = mysql_fetch_lengths(result);
42
43              //On fait une boucle pour accéder à  la valeur
                    de chaque champs
44              for(i = 0; i < num_champs; i++)
45              {
46                  //On ecrit toutes les valeurs
47
48                  //printf("[%.*s] ", (int) lengths[i], row[i]
                        ? row[i] : "NULL");
49                  printf("[%.*s] ", (int) lengths[i], row[i] ?
                        row[i] : "NULL");
50
51              }
52              printf("\n");
53          }
54
55          // libération du jeu de résultat
56          mysql_free_result(result);
57
58          // libération de la connection au SGBD
59          mysql_close(NULL);
60          //mysql_close(&conn);
61      } else {
62          printf("Une erreur s'est produite lors de la
                connexion au SGBD");
63      }
64
65      return 0;
66
67  }
```

44 Grammaire BNF

Lien vers la grammaire BNF (forme de Backus-Naur) pour SQL92.

https://github.com/ronsavage/SQL/blob/master/sql-92.bnf

Index