Système de gestion de base de données

✓ *Utilité*

Une base de données centralise, structure et conserve toute l'information sur un sujet pour :

- Faciliter l'accès à cette information.
- La mettre à la disposition de tous (en respectant des droits d'accès).
- Eviter les redondances anarchiques (mêmes informations présentes dans différents fichiers et gérées de façon indépendantes).
- Permettre une réutilisation ultérieure; les BD sont donc stockées sur mémoires secondaires.
- Faciliter l'évolution indépendante des données et des programmes qui les utilisent. (indispensable car une même BD peut être partagée par des groupes d'utilisateurs différents).
- Permettre l'utilisation des données par plusieurs utilisateurs simultanément avec une gestion fine des transactions.

✓ *SGBD*

Une base de données étant un ensemble cohérent d'informations mémorisées sur support informatique, ces informations doivent être accessibles à l'aide d'applications dédiées. Ces applications sont appelées systèmes de gestion de base de données (SGBD).

Un SGBD est un ensemble de logiciels qui fournit un environnement permettant :

- **Indépendance physique** : Le niveau physique peut être modifié indépendamment du niveau conceptuel. Cela signifie que tous les aspects matériels de la base de données n'apparaissent pas pour l'utilisateur, il s'agit simplement d'une structure transparente de représentation des informations.
- **Indépendance logique** : le niveau conceptuel doit pouvoir être modifié sans remettre en cause le niveau physique, c'est-à-dire que l'administrateur de la base doit pouvoir la faire évoluer sans que cela gêne les utilisateurs.
- **Manipulabilité** : des personnes ne connaissant pas la base de données doivent être capables de décrire leur requête sans faire référence à des éléments techniques de la base de données.
- **Rapidité des accès** : le système doit pouvoir fournir les réponses aux requêtes le plus rapidement possibles, cela implique des algorithmes de recherche rapides.
- **Administration centralisée** : le SGBD doit permettre à l'administrateur de pouvoir manipuler les données, insérer des éléments, vérifier son intégrité de façon centralisée.
- **Limitation de la redondance** : le SGBD doit pouvoir éviter dans la mesure du possible des informations redondantes, afin d'éviter d'une part un gaspillage d'espace mémoire mais aussi des erreurs.
- **Vérification de l'intégrité** : les données doivent être cohérentes entre elles, de plus lorsque des éléments font références à d'autres, ces derniers doivent être présents.
- **Partageabilité des données** : le SGBD doit permettre l'accès simultané à la base de données par plusieurs utilisateurs.
- **Sécurité des données** : Le SGBD doit présenter des mécanismes permettant de gérer les droits d'accès aux données selon les utilisateurs.

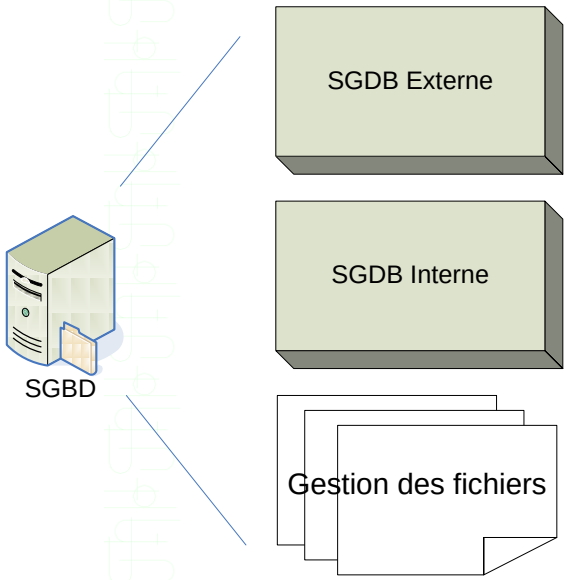
Les principaux systèmes de gestion de bases de données sont les suivants:

Borland Paradox Filemaker IBM DB2 Ingres Interbase Microsoft SQL server Microsoft Access Microsoft FoxPro	Oracle Sybase MySQL PostgreSQL mSQL SQL Server 11

✓ *Structure d'un SGBD*

Le SGBD peut se décomposer en trois sous-systèmes:

- le système de gestion de fichiers: il permet le stockage des informations sur un support physique
- le SGBD interne: il gère l'ordonnancement des informations
- le SGBD externe: il représente l'interface avec l'utilisateur



Les SGBD ne sont pas caractérisés par les applications ou les types d'applications qu'ils permettent de gérer mais par la manière dont ils structurent les données suivant un modèle qui correspond à l'organisation logique des données.

Les principaux modèles que l'on rencontre sont :

- Le modèle relationnel qui est le modèle le plus utilisé actuellement.
- Les modèles réseau et hiérarchique plus anciens mais qui sont néanmoins encore utilisés.
- Le modèle orienté objet.

Base de données relationnelle

✓ *Définition*

Le modèle relationnel a été introduit en 1970 par E.F. Codd, mathématicien au centre de recherche d'IBM à San José (Californie)). Ce modèle découle de la théorie mathématique des ensembles. Les premiers SGBD relationnels sont apparus une dizaine d'années plus tard.

Les points forts du modèle relationnel sont :

- Simplicité de représentation (tables) et de manipulation des données (SQL).
- Précision car il est basée sur les règles de la théorie des ensembles.
- Flexibilité car les liens entre les données sont effectués dynamiquement.

Nous parlons alors de ***SGBDR***.

✓ *Les Tables*

Toutes les données d'un SGBDR sont représentées sous formes de **tables** de valeurs organisées en **lignes** et **colonnes**. Chaque table exprime une **relation** au sens mathématique. A chaque table est associé le nom de la relation.

Cette table est représentée sous forme de tableau :

- Les lignes constituent des enregistrements.
- Les colonnes représentent des champs.

Une table contient une suite de lignes (appelé aussi tuples ou vecteurs) stockées sur un support externe. De façon générale, chaque ligne regroupe les informations concernant une **entité** (un **objet**,

une personne, un événement, etc.) c'est à dire un concept du monde réel (externe à l'informatique). Toutes les lignes d'une même table décrivent des **objets** de même nature.

Chaque colonne représente un **attribut** qui caractérise les entités de la table. Chaque colonne est repérée par un nom significatif. Les valeurs d'une colonne appartiennent à un domaine.

Nota : la notion d'objet et d'attribut se retrouve bien évidemment dans les bases de données orientées objet.

Exemple : la table **livre**

noliv	titre	auteur	genre	prix
1	Les chouans	Balzac	roman	80
2	Germinal	Zola	roman	75
3	L'assommoir	Zola	roman	95
4	La bête humaine	Zola	roman	70
5	Les misérables	Hugo	roman	105
6	La peste	Camus	roman	112
7	Les fleurs du mal	Baudelaire	poésie	130
8	Paroles	Prévert	poésie	120

Nota : L'ordre des lignes dans une table est indifférent. Il est possible d'ajouter, de modifier ou de supprimer des lignes à une table.

✓ Notion de clé

Chaque table doit avoir un identifiant appelé **clé**. Une clé est un attribut particulier (éventuellement une combinaison minimale d'attributs) dont les valeurs permettent **d'identifier sans ambiguïté** une entité de la relation. Cela signifie qu'une certaine valeur de clé ne peut exister qu'une seule fois dans la relation.

Nota : Dans la table précédente, le numéro de livre est une clé.

✓ Les différentes clés

Clé primaire, clé secondaire

Dans le cas où nous disposons dans une table de plusieurs clés, nous en choisirons une, la plus représentative que nous désignerons par **clé primaire** et ainsi toutes les autres seront des **identifiants secondaires** de la table.

Clés étrangères

Dans une table, une **clé étrangère** est une clé primaire pour une autre table. Cela permet de faire référence à une ligne de cette table. C'est un lien logique entre des lignes de tables différentes.

C'est précisément ce concept qui permet d'introduire une relation entre les tables, d'où le nom de base de données relationnelle.

Ex. dans la table **livre**, ***noauteur*** est une clé étrangère et dans la table **auteur** une clé primaire:

noliv	titre	noauteur	genre
1	Les chouans	1	roman
2	Germinal	3	roman
3	L'assommoir	3	roman
4	La bête humaine	3	roman
5	Les misérables	4	roman
6	Les raisins de la colère	5	roman
7	La peste	6	roman
8	Les fleurs du mal	2	poésie
9	Paroles	7	poésie

noauteur	nom	nationalité
1	Balzac	français
2	Baudelaire	français
3	Zola	français
4	Hugo	français
5	Steinbeck	américain
6	Camus	français
7	Prévert	français

Au lieu de scinder l'ensemble des informations sur deux tables, nous aurions pu tout-à-fait prendre une seule table en intégrant tout simplement la nationalité sur la table livre que nous avons déjà mis au point. Nous remarquons toutefois tout de suite que si nous stockons une très grande quantité de livres, le même nom de l'auteur ainsi que sa nationalité apparaissent très fréquemment. C'est une information redondante qu'il faut à tout prix éviter. Le serveur de base de données serait vite saturé.

✓ *Contraintes d'intégrité*

Une **contrainte d'intégrité** est une contrainte que nous imposons de façon à assurer la cohérence de la Base de Données.

Une contrainte d'intégrité est une clause permettant de contraindre la modification de tables, faite par l'intermédiaire de requêtes d'utilisateurs, afin que les données saisies dans la base soient conformes aux données attendues. Ces contraintes doivent être exprimées dès la création de la table elles peuvent être ***: NOT NULL***, ***UNIQUE***, etc.

Conception d'une base de données relationnelle

La conception de la BD doit aboutir à la définition des tables permanentes nécessaires aux besoins de l'application (des utilisateurs). Elle doit prendre en compte toutes les informations nécessaires à la représentation d'un domaine d'application et uniquement que ces données représentatives.

En phase de conception, l'utilisation d'un **modèle conceptuel** des données permet :

- De faciliter grandement l'activité de conception surtout dans le cas de systèmes complexes car il permet de travailler sur un modèle théorique ou idéal des données à prendre en compte dans le système d'information sans s'embarrasser des contraintes d'implantation.
- De maintenir un niveau de raisonnement indépendant des outils ce qui facilite le passage d'un système à un autre (fichiers, relationnel, orienté objet, tableur ou autre). Cela facilitera donc la maintenance ultérieure.
- Une validation par les utilisateurs.

Le but de la modélisation est d'obtenir un schéma conceptuel qui sera traduit ultérieurement en un schéma de base de données sous forme de tables, colonnes et de contraintes d'intégrité.

✓ *Modèle Entité – Association*

Le modèle Entité-Association proposé par Chen (76) est basé sur le fait que le monde réel (la réalité) peut-être représenté à l'aide d'entités qui représentent des objets ayant une existence visible et des associations entre ces objets.

Il permet au moyen d'un formalisme précis, une représentation naturelle et graphique des concepts qu'il décrit sans s'attacher à la manière dont ils seront représentés ultérieurement.

Formalisme du modèle Entité-Association

Le modèle définit les éléments suivants :

- **L'entité** ou individu ou **objet** : plus petit élément se suffisant à lui-même qu'il est nécessaire de prendre en compte dans le système; par ex. un livre, une personne. Chaque entité appartient à un type d'entité (livre, personne). Les types d'entités sont représentés par des rectangles.

- **La propriété** ou **attribut** ou information élémentaire : plus petit élément manipulable du système; par ex. titre, auteur, nom, prénom. Les propriétés décrivent les entités et les associations. Certains attributs peuvent être facultatifs. Ils sont représentés entre parenthèses
- **L'identifiant** : une des propriétés d'une entité qui permet de la discriminer (**clé**); par ex. ***noliv***, ***noauteur***. Les identifiants sont soulignés.
- **L'association** : c'est un lien entre 2 ou plusieurs entités. Par exemple un livre est emprunté par une personne. De même que pour les entités, il existe un type d'association qui relie des types d'entités. Dans l'exemple de gestion de prêt de livre, le type d'association ***emprunte*** lie le type ***livre*** au type ***personne***. Les types d'associations sont souvent représentés par des losanges ou des ellipses.
- **Les cardinalités** : elles indiquent à combien d'associations chaque entité **peut** et **doit** participer. On exprime cette propriété pour chaque type d'entité en l'indiquant au moyen du nombre maximum et du nombre minimum.

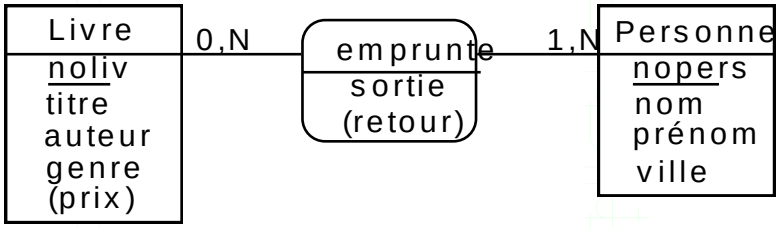
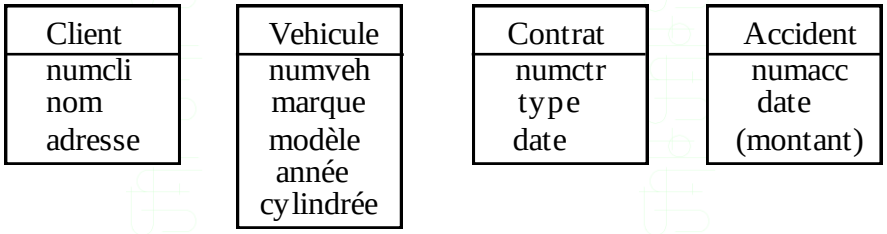


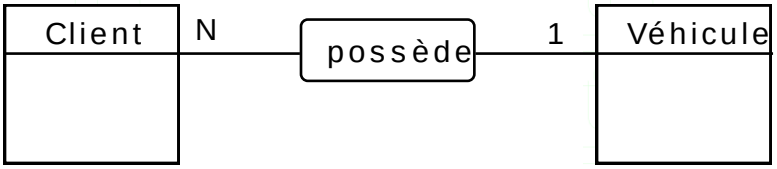
Diagramme Entité-Association de l'exemple de gestion de prêt de livres

Type d'associations

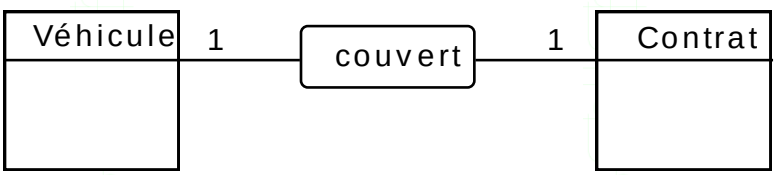
Pour illustrer cela, prenons le domaine de l'assurance automobile dans lequel on repère des entités clients, contrats, véhicules et accidents.



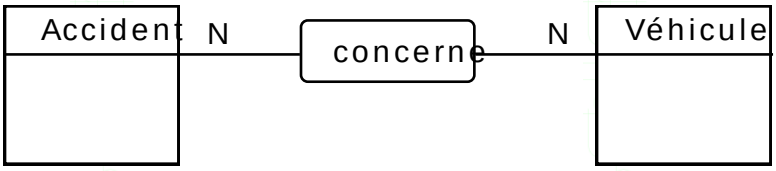
- **Type d'associations un-à-plusieurs** : un client peut posséder plusieurs véhicules mais un véhicule n'appartient qu'à un seul client. On représente ce fait en indiquant sur chacune des branches du type d'associations le nombre maximum d'associations dans laquelle une entité peut apparaître. On distingue 2 valeurs : un (**1**) et plusieurs (**N**).



- **Type d'associations un-à-un** : un véhicule est couvert par un contrat et un contrat ne couvre qu'un véhicule. On indique **1** sur chacune des branches du type d'associations.



- **Type d'associations plusieurs-à-plusieurs** : un accident peut concerner plusieurs véhicules et un véhicule a pu être concerné par plusieurs accidents. On indique **N** sur chacune des branches du type d'associations

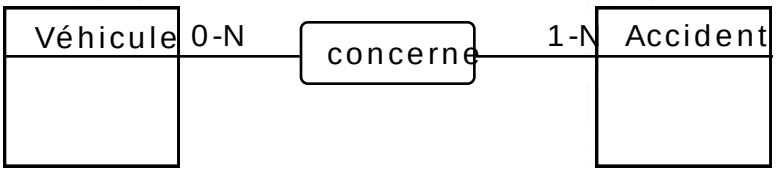


Ainsi, le nombre maximum d'entités associées est indiqué selon le cas par 1 ou N.

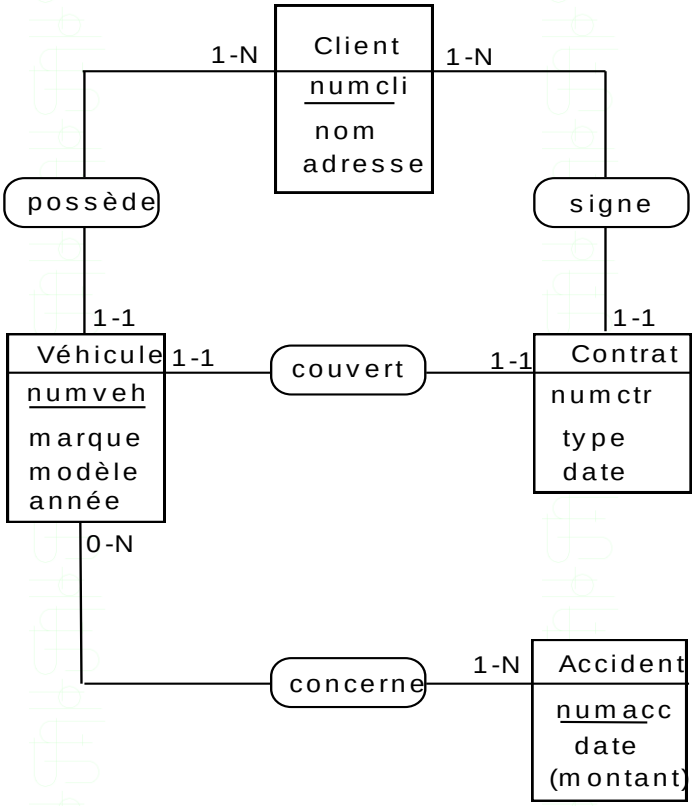
Type d'associations obligatoire ou facultatif

Nous pouvons imposer qu'un type d'association soit **obligatoire** ou **facultatif** pour un type d'entité qui y participe. Ainsi, un véhicule n'est pas forcement impliqué dans un accident; alors qu'un accident met en cause au moins un véhicule.

Nous représentons ce fait en indiquant sur chacune des branches du type d'associations le nombre minimum d'associations (chiffre de gauche) dans laquelle une entité peut apparaître. Ce nombre vaudra selon le cas **0** ou **1**.



Exemple :



✓ *Construction du schéma conceptuel Entité-Association*

Le processus d'élaboration du schéma conceptuel d'une Base de Données est complexe car il représente une activité de modélisation de situations issues de la réalité mais néanmoins déterminant car il conditionne la qualité de celle-ci.

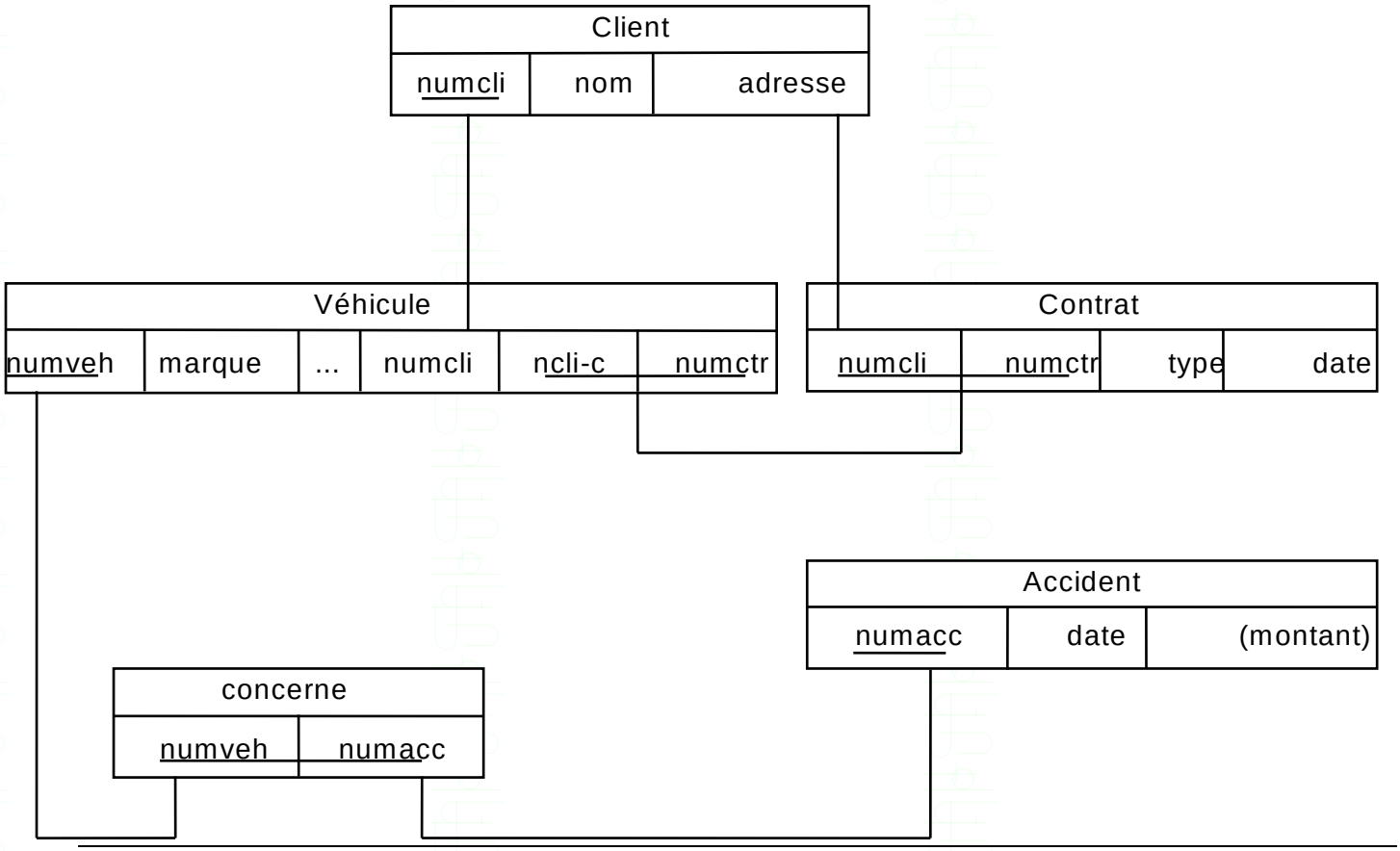
Les techniques proposées dans la bibliographie consistent dans le cas où les sources d'informations sont exprimées en langage courant, à décomposer (méthode descendante) l'énoncé en phrases ou propositions élémentaires décrivant chacune un concept ou un type de faits. Si chaque proposition est pertinente pour le domaine d'application, elle est comparée aux faits déjà incorporées au schéma en cours d'élaboration et si le fait qu'elle exprime en est absent et est non contradictoire, il est traduit en termes du modèle E/A et incorporé dans ce schéma courant.

Généralement dans les propositions simples, les sujets et compléments représentent des concepts qui pourront être pris en tant que propriété (information élémentaire) ou entité (élément se suffisant à lui-même), et les verbes en tant qu'associations possibles entre entités.

Passage du modèle Entité-Association à une Base de Données Relationnelle

Le passage du modèle E-A au schéma relationnel est assez simple; il s'effectue en établissant les correspondances suivantes :

Modèle E-A	Schéma relationnel
Une propriété	Un attribut
Un type d'entité	Une relation (table)
Un identifiant	Une clé primaire
Un type d'associations un-à-plusieurs	Une clé étrangère (identifiant de la relation source) dans la relation cible. Si le type d'association est obligatoire dans la relation cible, la clé étrangère est déclarée obligatoire sinon elle est facultative.
Un type d'associations un-à-un	Si le type d'association est obligatoire seulement dans une des 2 relations alors la clé étrangère est ajoutée à cette relation sinon elle sera ajoutée indifféremment dans l'une ou l'autre relation.
Type d'associations plusieurs-à-plusieurs	Le type d'association devient une relation qui assure le lien entre les 2 relations. Son identifiant multi-colonne est constitué par des clés étrangères.



Le langage SQL

Nous venons de voir comment représenter des données sous forme de relations, nous allons voir maintenant comment décrire et manipuler ces données au travers d'un langage de haut niveau : **SQL**.

Le langage **SQL** (**S**tructured **Q**uery **L**anguage) d'IBM est aujourd'hui le langage normalisé et standard d'accès aux bases de données relationnelles. Cependant, pour faciliter l'écriture de requêtes aux utilisateurs, les éditeurs de logiciels proposent de plus en plus des interfaces graphiques au dessus de SQL. L'exemple le plus significatif est ACCESS.

Une requête **SQL** est la description d'une opération que le **SGBD** doit exécuter. **SQL** est un langage déclaratif et non procédural, c'est à dire qu'il répond ***"au quoi"*** faire et prend à sa charge ***le "comment faire"***.

SQL est un langage complet. Il permet :
- La description des données.
- La manipulation des données.
- La spécification de contraintes d'intégrité (COMMIT/ROLLBACK non développé) (transactions).
- La spécification de droits d'accès.
- L'optimisation des performances.

Au travers d'exemples, nous allons voir les principales possibilités de description et manipulation du langage SQL.

✓ *Définition et description des données*

TYPES DE DONNEES SQL

Les principaux types de données admis par SQL sont :

TINYINT, SMALLINT, INTEGER, LONGINT	Entiers signés (8, 16, 32, 64 bits).
DECIMAL(M,N)	Décimaux de m chiffres dont n après le point décimal.
FLOAT, DOUBLE PRECISION	Réels flottants en simple ou double précision.
CHAR(N), VARCHAR(N)	Chaîne fixe ou variable de n caractères. (optimisations ≠)
TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT	Chaîne de caractères de ≠ longueur 2n (n=8, 16, 32, 64).
DATE, TIME, DATETIME,TIMESTAMP(X)	Date, heure ou les deux.
BLOB	Information binaire non interprété (fichiers).
ENUM	Propose un nombre de valeurs fixes (sous forme de chaîne).
SET	Ensemble de valeurs.
BOOLEAN	Valeurs booléennes.

CREATION ET GESTION D'UNE BASE DE DONNEES

Commandes	*Description*
CREATE DATABASE [IF NOT EXISTS] nom;	**Création :** Lorsque nous devons concevoir une base de données du début jusqu'à la fin, la première chose à réalisée est la création de la base de données (**CREATE DATABASE**) elle-même qui va stocker l'ensemble des informations dans les différentes tables en relation. ***Faites bien une différence entre le serveur de base de données avec la base de données elle-même.***
SHOW DATABASES;	**Consultation des bases** : Il est possible de consulter à tout moment l'ensemble des bases déjà constituées (**SHOW DATABASES**).
USE nom ;	**Base de données par défaut :** Par ailleurs, pour effectuer un ensemble d'opérations sur une base en particulier, vous devez indiquer qu'elle est la base de données par défaut sur laquelle vous allez travailler au moyen de la requête **USE**.
DROP DATABASE [IF EXISTS] nom;	**Suppression d'une base de données :** Enfin, vous pouvez supprimer définitivement le contenu complet d'une base de données.

CREATION, SUPPRESSION ET MANIPULATION DE TABLE

Différents formats de table ?

ISAM	Format standard
MYISAM	Nouvelle version optimisée du format **ISAM** dans la base de données MySQL.
BDB	Format relativement récent, issu des recherches au sein de l'université de Berkeley aux Etats-Unis, il permet de gérer les transactions (Une même base de données pouvant être répartie sur plusieurs serveurs indépendants).
INNODB	Similaire à **BDB** aves une gestion supplémentaire de réplication.

Les différentes contraintes que nous pouvons déclarer sont les suivantes :

NOT NULL	Nous pouvons rendre la saisie d'une colonne obligatoire en apposant le mot clé **NOT NULL**. Dans ce cas, il est impossible que la colonne reste vide. Autrement dit, la colonne devra toujours être renseignée de manière explicite (expression de valeur) ou implicite (valeur par défaut) lors de l'insertion via l'ordre **SELECT**.
UNIQUE	Chaque ligne de la table doit avoir une valeur différente ou **NULL** pour cette (ou ces) colonne.
PRIMARY KEY	Chaque ligne de la table doit avoir une valeur différente pour cette (ou ces) colonne. les valeurs **NULL** sont rejetées.
AUTO_INCREMENT	Attribut que le champ sera automatiquement incrémenté de **1** lors d'une insertion dans la table (souvent utilisée pour la clé primaire).
FOREIGN KEY	Cette colonne fait référence à une colonne **clé primaire** d'une autre table.
CHECK	Permet de spécifier les valeurs acceptables pour une colonne.
DEFAULT	Propose une valeur par défaut si l'utilisateur n'en spécifie pas au moment de l'insertion dans la table.

Valeurs par défaut d'une colonne :

Le mot clé DEFAULT permet de spécifier une valeur qui sera affectée à une colonne si lors de l'insertion de données il n'est pas fait référence de manières explicite ou implicite à cette colonne. La valeur par défaut d'une colonne peut être une expression de valeur, le marqueur NULL ou encore l'une des fonctions suivantes (non utilisées par MySQL) :

CURRENT_DATE	**Date** courante.
CURRENT_TIME(P)	**Heure** courante avec précision **P** de fraction de seconde.
CURRENT_TIMESTAMP(P)	Combiné **date/heure** avec précision **P** de fraction de seconde.
USER	**Utilisateur courant**.

Quelques exemples	Description
CREATE TABLE livre (noliv integer not null auto_increment, titre char (80) not null, auteur char (40) not null, genre char (40) default 'Roman', primary key (noliv));	**Création :** L'attribut **noliv** est de type entier, qui est automatiquement rempli pour chaque ligne de la table. A chaque introduction d'une entité (d'un objet) dans la table (nouvelle ligne), la valeur numérique proposée est la valeur incrémentée par rapport à la dernière introduire. Par la suite, l'attribut **noliv** est désigné comme clé primaire. Une clé primaire est obligatoire et est unique par table. Elle impose également une seule occurrence dans la colonne
CREATE TABLE livre (noliv integer not null primary key, titre char(80) not null, auteur char(40) not null, genre char(40) default 'Roman') ;	**Création :** Dans cet exemple, nous retrouvons pratiquement la même table, avec toutefois quelques petites différences. D'une part, la clé primaire est désignée dès le départ. Par ailleurs, vous êtes dans l'obligation de gérer vous-même le numéro exclusif de la clé primaire, en proposant alors une valeur unique qui ne doit pas se retrouver dans les autres objets. Cela permet d'identifier chaque entité (ligne) de façon sûre et ainsi de les retrouver plus facilement ultérieurement.
CREATE TABLE personne (nopers integer not null primary key, nom char(40) not null, prenom char(40) not null, tel char(14) unique, ville char(40) default 'Aurillac') ;	**Création :** Par défaut, toute colonne est facultative (**null**). Le caractère obligatoire d'une colonne est spécifié par la clause **not null**. La clause **unique** permet de vérifier qu'un champ ne soit pas en doublon (une seule occurrence) comme par exemple un numéro de téléphone.
CREATE TABLE emprunt (nopers integer not null, noliv integer not null, sortie date default current_date, retour date,	**Création :** La clé primaire qui est créée dans cette table possède les mêmes propriétés et contraintes que celles étudiées précédemment, mais qui est composée de plusieurs champs de la table (le nombre maximum est 32). Nous avons déjà rencontrés les champs **nopers** et **noliv**

```
  primary key (noliv, nopers, sortie),
  foreign key (noliv) references livre(noliv),
  foreign key(nopers) references personne(nopers)
);
```

qui sont respectivement les clés primaires des tables **personne** et **livre**. Elles deviennent donc des clés étrangères pour cette table **emprunt** (ces tables sont donc associées – ***base de données relationnelle***).

```
DROP TABLE [IF EXISTS]  livre;
```

Suppression : L'utilisation de **DROP TABLE** supprime toutes les données relatives à cette table (toutes celles introduites par les utilisateurs) et sa définition. Toute suppression est définitive ; il faut faire preuve d'une extrême vigilance lors de l'exécution de cette commande, car vous ne pouvez plus revenir en arrière.

```
DROP TABLE [IF EXISTS]  livre, personne;
```

Vous pouvez supprimer plusieurs tables avec une seule commande, en mettant leurs noms les uns à la suite des autres, séparés par des virgules.

ALTER TABLE nom_table [COMMANDE SPECIFIQUE] – commande permettant d'apporter des modifications à la structure d'une table existante.

```
ALTER TABLE livre ADD prix DECIMAL (5, 2) ;
```

Ajout d'une colonne : Il est possible de rajouter ultérieurement une colonne que nous n'avions pas pensé d'utiliser à priori. Cela se fait au travers par la commande de modification **ADD**. Ainsi, nous pouvons introduire l'attribut **prix** (donnée facultative) à notre table **livre**.

```
ALTER TABLE livre ADD
  prix DECIMAL (5, 2),
  existe BOOLEAN not null ;
```

Ajout de plusieurs colonnes : Ajout possible d'une série de champ à la table, chacun étant séparé par l'opérateur « , ». La définition de type est similaire à celle de la fonction **CREATE TABLE** vue à la rubrique précédente.

```
ALTER TABLE livre ADD
  prix DECIMAL (5, 2) AFTER genre,
  existe BOOLEAN not null  FIRST;
```

Ajout d'une colonne en choisissant la position : Vous avez la possibilité de préciser la position du champ dans la table qui, par défaut, s'ajoute à la fin ; première position via **FIRST**, ou après une autre colonne via **AFTER nom_de_la_colonne**.

```
ALTER TABLE livre ADD
  noliv INTEGER NOT NULL PRIMARY KEY;
```

Ajout d'une clé primaire : Le rajout d'une colonne peut correspondre à une clé primaire. La définition de clé est similaire à celle de la fonction **CREATE TABLE**, vue à la rubrique précédente.

```
ALTER TABLE personne ALTER ville
  SET DEFAULT 'Arpajon' ;
ALTER TABLE personne ALTER ville
  DROP DEFAULT 'Arpajon' ;
```

Modification de la valeur par défaut d'une colonne : Modifie l'attribut **DEFAULT** d'un champ. **SET** permet de fixer une valeur (ou de modifier celle existante, et **DROP** de supprimer celle définie par défaut.

```
ALTER TABLE personne CHANGE
  nopers idPers INTEGER ;
ALTER TABLE personne CHANGE
  nopers idPers INTEGER UNSIGNED ;
ALTER TABLE personne CHANGE
  idPers idPers INT(10) ;
```

Changer le nom du champ d'une colonne : Cette fonction permet le nom et/ou le type d'une colonne. Pour changer le type sans le nom, il vous suffit de mettre deux fois le nom de votre champ avant la nouvelle définition de type.

```
ALTER TABLE personne MODIFY
   idPers INTEGER UNSIGNED ;
ALTER TABLE personne MODIFY
   idPers INTEGER PRIMARY KEY ;
```

Changer le type du champ uniquement : Cette fonction est plus simple que la précédente dans le cas où nous désirons modifier uniquement le type du champ sélectionné. C'est également une solution pour intégrer une clé primaire

```
ALTER TABLE livre DROP genre ;
```

Suppression d'une colonne : De façon similaire, il est possible de retirer de la table une colonne que nous n'estimons plus importante.

```
ALTER TABLE livre DROP PRIMARY KEY ;
```

Suppression de la clé primaire : Cette fonction supprime la clé primaire tout en conservant le nom du champ.

```
ALTER TABLE personne RENAME auteur;
```

Renommer une table : Cette fonction permet de proposer un nouveau nom à la table spécifiée.

```
ALTER TABLE personne ORDER BY nom;
```

Ordre automatique d'un des champs : Cette fonction permet de trier les lignes de la table selon l'ordre croissant du champ sélectionné.

DESCRIBE personne;	**Constitution de la table :** Cette fonction permet de lister le contenu structurel de la table spécifiée.

✓ *Manipulation des données (mise à jour des données)*

Insertion, ajout, suppression et modification des données

INSERT INTO personne (nom, ville) VALUES ('REMY', 'AURILLAC') ; INSERT INTO personne (nom, ville) VALUES (('REMY', 'AURILLAC') ('DUPONT', 'ARPAJON') ;	**Insertion d'une donnée dans la table :** Cette commande permet d'ajouter une ou plusieurs lignes dans une table. Vous spécifiez alors les colonnes qui vous intéressent, et vous proposez alors les valeurs correspondantes à la suite de la commande **VALUES**. Dans notre exemple, nous avons une clé primaire qui est auto incrémentée. Il n'est donc pas nécessaire de s'en préoccuper. Si les valeurs sont des chaînes de caractères, vous devez les écrire entre guillemets ou bien entre apostrophes.
INSERT INTO personne (ville) VALUES ('AURILLAC') ;	**Insertion d'une donnée dans la table :** Sans spécifier toutes les colonnes. Dans ce cas, les valeurs par défaut des champs sont prises en compte. Ainsi, si la valeur proposée vous convient, vous pouvez omettre de renseigner ce champ en particulier.
INSERT INTO personne VALUES (7, 'REMY', 'AURILLAC') ;	**Insertion d'une donnée dans la table :** Vous pouvez omettre le nom des colonnes dans votre requête. Dans ce cas, il est impératif de spécifier toutes les valeurs des champs, car l'insertion se fait dans l'ordre des champs de la table (clé primaire comprise, ce qui d'ailleurs peut être gênant).
INSERT INTO personne SET nom='DURANT', ville='VIC/CERE' ;	**Insertion d'une donnée dans la table :** La commande **SET** permet de proposée la valeur directement après le nom du champ.
DELETE FROM personne WHERE nom='DURANT'; DELETE FROM emprunt WHERE retour IS NOT NULL;	**Suppression d'une donnée dans la table :** Cette fonction supprime toutes les lignes qui correspondent aux critères de sélection de la clause **WHERE**. Si vous ne spécifiez pas de clause **WHERE**, la commande **DELETE** vide entièrement la table.
REPLACE INTO personne (nopers, nom, ville) VALUES (7, 'REMY', 'ARPAJON') ;	**Modification d'une donnée dans la table :** Il existe deux fonctions pour apporter des modifications aux enregistrements dans une table. La première est la fonction **REPLACE**. Cette fonction agit exactement comme **INSERT**. La principale différence réside dans le fait que cette fonction permet de remplacer un enregistrement identifié par une clé primaire, déjà existant dans une table. Dans le cas où la commande **REPLACE** est utilisée sur un enregistrement qui n'est plus ou pas présent dans la table, la commande se comporte exactement comme **INSERT**. Si **REPLACE** permet de remplacer un enregistrement déjà présent dans une table, il faut pour cela ajouter dans la commande le champ de la clé primaire et sa valeur.
UPDATE personne SET nom='Rémy' WHERE nom='REMY'; UPDATE emprunt SET retour='02/05/09' WHERE noliv=14 AND retour IS NOT NULL; UPDATE personne SET ville='Aurillac' WHERE nom LIKE 'R%';	**Modification d'une donnée dans la table :** La deuxième commande de modification est la commande **UPDATE**. Cette commande est indispensable à tous vos projets, puisque c'est grâce à elle que vous allez pouvoir faire évoluer et mettre à jour vos informations dans votre base de données. Le premier exemple nous montre comment changer un nom de personne sur plusieurs lignes partout où l'occurrence se trouve. Le deuxième nous montre comment effectuer un changement sur une ligne unique en prenant en compte la clé primaire. L'intérêt des clés primaires est de pouvoir mettre à jour un seul et unique enregistrement de la table de façon sûre, sans risquer d'interférences sur une autre ligne. Dans le troisième exemple, nous nommons la ville **'Aurillac'** pour tous les noms qui commencent par la lettre **'R'**.

✓ *Sélection des données (recherche)*

Les fonctions de sélection des données font la puissance des bases de données. Elles sont la pierre angulaire des SGBD, la fonctionnalité indispensable qui transforme les bases de données d'outils de stockage en outils de croisement et de synthèse des informations.
La requête de sélection est la plus complexe du langage SQL. Elle permet d'interroger la BD en composant les opérations de l'algèbre relationnelle : **projections, restrictions, unions, jointures**.

Le résultat d'une sélection en mode interactif est représenté sous la forme d'un tableau d'une ou de plusieurs lignes et colonnes affichées à l'écran. En mode programme, elle est accessible dans un fichier appelé curseur.

SELECT

```
SELECT <clause d'unicité> <liste de colonnes>
FROM   <liste de tables>
WHERE <critère de sélection des lignes>
GROUP BY <liste de colonnes>
HAVING <critère de sélection des groupes>
ORDER BY <critère d'ordre>
LIMIT <critère de limitation du résultat>
```

Sélection de données dans la table : Commande de sélection de données à partir d'une ou plusieurs tables, ou directement à partir de la ligne de commande. Cette commande peut être pilotée par des clauses de sélection **WHERE**.

ATTENTION l'ordre des commandes est important. Vous ne pouvez pas, par exemple inverser **ORDER BY** et **LIMIT**.

DINSTINCT ou ALL

Clause d'unicité : Attribut optionnel permettant d'appliquer un filtre à la sélection. **DISTINCT** signifie que la commande ne renverra que des lignes différentes les unes des autres. **ALL** signifie que la commande renverra tous les résultats, c'est l'attribut sélectionné par défaut.

*

Liste de colonnes : Il est possible, avec une requête de sélection, de choisir certains champs ou tous les champs. Pour cela, il suffit de préciser le nom des champs, séparés par des virgules ou de mettre une étoile *****, qui signifie tous les champs.

GROUPE BY

Groupement des résultats : Commande optionnelle permettant de grouper les résultats selon les valeurs identique d'une des colonnes. Cette fonction est particulièrement pratique pour tous les traitements sur des ensembles comme les sommes ou les moyennes.

HAVING

Filtre et condition : Commande optionnelle permettant d'ajouter une condition sur un champ filtrant ainsi les lignes de résultats. Elle doit être utilisée pour des champs dont la valeur est retournée par la commande. S'il est possible d'ajouter la même condition dans la clause **WHERE**, c'est encore mieux, car **HAVING** ne subit aucune optimisation lors de son filtrage.

ORDER BY

Tri dans l'ordre : Commande optionnelle permettant d'ordonnancer les lignes selon une ou plusieurs colonnes, de manière croissante **ASC** ou décroissante **DESC**.

LIMIT

Limitation du nombre de résultat : Commande optionnelle permettant de limiter le nombre de résultats retournés. Elle peut également retourner une plage de lignes (de la 25ème à la 30ème, par exemple).

PROJECTION

Faire une projection consiste à définir un sous-ensemble des colonnes de la liste de tables.

```
SELECT ALL titre, genre FROM livre;
```

Lister tous les titres avec leur genre des livres présents :

titre	genre
Les chouans	roman
Germinal	roman
L'assommoir	roman
La bête humaine	roman
Les misérables	roman
La peste	roman
Les lettres persanes	roman
Bel ami	roman
Les lettres de mon moulin	roman
César	roman
Marius	roman
Fanny	roman
Les fleurs du mal	poésie
Paroles	poésie
Les raisins de la colère	roman

```
SELECT DISTINCT auteur FROM livre;
```

Lister tous les auteurs des livres présents :

auteur
Balzac
Zola
Hugo
Camus
Maupassant
Daudet
Pagnol
Baudelaire
Prévert
Steinbeck

RESTRICTION

Une restriction, exprimé par la clause WHERE, permet de fixer des conditions, de poser des filtres sur le contenu afin d'avoir un résultat, c'est-à-dire un ensemble de données organisées en ligne (comme une table), collant à vos besoins.

SQL permet l'utilisation dans la clause WHERE des opérateurs :

- ***De comparaison : =, <>, >, <, >=, <=.***
- ***Arithmétiques : +, -, *, /; ces opérateurs ne sont utilisés que sur les attributs numériques. Ils peuvent être utilisés pour effectuer un calcul sur les colonnes sélectionnées ou sur les colonnes du critère de sélection.***
- ***Logiques : AND, OR, NOT permettent de combiner des critères de sélection.***
- ***D'intervalle : BETWEEN.***
- ***D'ensemble : IN.***
- ***De ressemblance : LIKE : Cet opérateur utilise quelques caractères spéciaux pour étendre les possibilités de recherche d'une chaîne dans une autre. Cet opérateur fonctionne parfaitement avec des champs de type numérique.***
  - ***Le caractère % est utilisé pour indiquer la position que peut avoir le mot recherché par rapport à la chaîne dans laquelle se fait la recherche. Ainsi, '%mot%' correspond à une recherche dans toute la chaîne, quelle que soit la position de mot. L'expression, 'mot%' signifie que mot doit être en début de chaîne pour que la condition soit vraie, et, à l'inverse, l'expression '%mot' signifie que la chaîne doit se finir par mot pour que la condition soit vraie.***
  - ***Le caractère _ est utilisé pour indiquer qu'il peut y avoir un caractère d'une valeur quelconque à cette position. Ainsi, '_mot' signifie que la condition est vraie si la chaîne de caractères finit par mot et possède un seul caractère devant celui-ci (la valeur 'cmot' donne vrai, alors que 'ctmot' ou 'cmotc' donne faux).***
  - ***La combinaison [ ] est utilisée pour faire correspondre un ensemble de valeurs à un caractère. Ainsi '[cty]mot' signifie que la condition est vraie si la chaîne de caractères est 'cmot' ou 'tmot' ou 'ymot'.***
- ***Le test de valeur nulle : IS NULL (ou IS NOT NULL)***

```
SELECT *
FROM livre
WHERE auteur ='Zola';
```

Recherche tous les livres écrits par Zola :

noliv	titre	auteur	genre	prix
2	Germinal	Zola	roman	7,5
3	L'assommoir	Zola	roman	9,5
4	La bête humaine	Zola	roman	7

```
SELECT *
FROM livre
WHERE auteur ='Zola'
AND prix < 75;
```

Recherche tous les livres écrits par Zola et dont le prix est inférieur à 75 francs :

titre	genre	prix
La bête humaine	roman	7

```
SELECT *
FROM livre
WHERE NOT
(auteur ='Balzac' OR auteur='Zola');
```

Rechercher tous les livres écrit ni par Balzac et ni par Zola .

```
SELECT titre
FROM livre
WHERE auteur > 'Camus';
```

Rechercher tous les titres dont les noms des auteurs se trouvent après Camus dans l'ordre alphabétique.

```
SELECT *
FROM emprunt
WHERE retour IS NULL;
```

Rechercher tous les livres en cours d'emprunt :

nopers	noliv	sortie	retour
4	14	01/01/94	
2	11	18/03/94	
3	4	30/03/94	
8	1	02/04/94	

```
SELECT DISTINCT nom, prénom
FROM personne
WHERE ville
IN ('Aurillac', 'Arpajon', 'Murat');
```

Rechercher tous les emprunteurs habitant Aurillac, Arpajon ou Murat :

```
SELECT *
FROM livre
WHERE titre
LIKE 'Les lettres %';
```

Rechercher tous les titres commençant par "Les lettres ":

noliv	titre	auteur	genre	prix
7	Les lettres persanes	Maupassant	roman	14
9	Les lettres de mon moulin	Daudet	roman	10

```
SELECT *
FROM personne
WHERE nom
LIKE '_____';
```

Rechercher tous les noms de personnes de 6 lettres.

```
SELECT *
FROM pages
WHERE nopages
LIKE '1%';
```

L'opérateur LIKE fonctionne parfaitement avec des champs de type numérique. Ici, la commande retourne toutes les lignes des pages dont le numéro commence par 1, par exemple 1, 10, 11, 17, 103…

```
SELECT *
FROM personne
WHERE nopers = LAST_INSERT_ID();
```

LAST_INSERT_ID() : Cette fonction est très pratique. Elle permet de récupérer l'identifiant, c'est-à-dire la valeur de la clé primaire, du dernier élément inséré dans la table.

```
SELECT *
FROM emprunts
WHERE sortie >= '2009/08/20'
AND sortie <= '2009/09/01';
```

Dates : Les recherches sur les dates peuvent se faire de différentes façons :
- Sur un intervalle de temps.

ATTENTION : dans la syntaxe de la date, vous devez préciser d'abord l'année suivie du mois et ensuite du jour du mois.

```
SELECT *
FROM emprunts
WHERE sortie
BETWEEN '2009/08/20'
AND '2009/09/01';
```

```
SELECT *
FROM emprunts
WHERE sortie >= '2009/08/20';
```

Dates : Les recherches sur les dates peuvent se faire de différentes façons :
- Depuis un certain temps.

```
SELECT *
FROM emprunts
WHERE sortie <= CURRENT_DATE();
```

GROUP BY

Cette clause est très pratique et très utilisée pour les synthèses d'informations. Elle sert principalement pour des statistiques ou des calculs groupés sur une ou plusieurs tables. Certaines fonctions, dites de regroupement, permettent de partitionner une relation en groupant des lignes selon une ou plusieurs colonnes. SQL dispose des fonctions de calcul de base que l'on retrouve dans les tableurs :

- ***COUNT : retourne le nombre d'occurrences correspondant au critère.***
- ***SUM, AVG : calcule la somme, la moyenne d'une colonne.***
- ***MIN, MAX : recherche la valeur minimum, maximum d'une colonne.***

```
SELECT COUNT (*) FROM livre;
```

Requête pour compter le nombre d'enregistrement d'une table :
(obtenir le nombre de livres stockés) : 15

```
SELECT COUNT (DISTINCT auteur)
FROM livre;
```

Obtenir le nombre d'auteurs : 10

```
SELECT auteur, COUNT (*)
FROM livre
GROUP BY auteur;
```

Obtenir la liste des auteurs avec le nombre de livres de chacun :

auteur	COUNT (*)
Balzac	1
Zola	3
Hugo	1
Camus	1
Maupassant	2
Daudet	1
Pagnol	3
Baudelaire	1
Prévert	1
Steinbeck	1

```
SELECT genre, COUNT (*), MIN(prix)
FROM livre
GROUP BY genre;
```

Obtenir la liste des genres avec le nombre de genres de chacun et le prix minimum pour chaque genre :

genre	COUNT (*)	MIN(prix)
poésie	2	12
roman	13	6.5

```
SELECT auteur, MAX(prix)
FROM livre
GROUP BY genre;
```

Connaître quel auteur dont le prix du livre est le plus cher par genre :

auteur	MAX(prix)
Baudelaire	13
Balzac	14

```
SELECT COUNT (*),
AVG (prix), MAX (prix), MIN (prix)
FROM livre;
```

Obtenir le nombre de livres, le prix moyen, l'écart de prix maximum

15	9.8333333333	7.5

HAVING

L'utilisation de la clause GROUP BY peut être combinée avec HAVING. La clause HAVING sélectionne un sous-ensemble de groupes en fonction d'un critère de la même manière que la clause WHERE sélectionne un nombre d'occurrences.

```
SELECT auteur, MIN(prix), MAX(prix)
FROM livre
GROUP BY auteur
HAVING COUNT(*) > 1;
```

Obtenir le nom de l'auteur, le prix du livre le moins cher et le prix du livre le plus cher de cet auteur en ne retenant que les auteurs ayant écrit plus d'un livre :

auteur	MIN (prix)	MAX (prix)
Zola	70	95
Maupassant	76	140
Pagnol	65	100

ORDER BY

Cette commande permet d'organiser les lignes de résultats renvoyées par une requête de sélection Le fonctionnement est relativement simple, puisqu'il se fait par colonnes, les unes après les autres en partant de la gauche, en ordre croissant ou décroissant.

Par défaut, les colonnes sont triées par ordre croissant. Il suffit d'utiliser le mot-clé DESC pour signaler que l'ordre est inversé, et qu'il est donc décroissant.

```
SELECT titre, genre, prix
FROM livre
WHERE prix BETWEEN 5 AND 18
ORDER BY prix;
```

Rechercher tous les titres dont le prix est compris entre 80 et 100 francs et les ordonner par prix croissant.

```
SELECT titre, auteur
FROM livre
WHERE genre ='roman'
ORDER BY auteur, titre DESC;
```

Rechercher tous les romans et les ranger par titre et par auteur ordonnés de façon décroissante :

titre	auteur
La bête humaine	Zola
L'assommoir	Zola
Germinal	Zola
Les raisins de la colère	Steinbeck
Marius	Pagnol
Fanny	Pagnol
César	Pagnol
Les lettres persanes	Maupassant
Bel ami	Maupassant
Les misérables	Hugo
Les lettres de mon moulin	Daudet
La peste	Camus
Les chouans	Balzac

LIMIT

Cette commande LIMIT sert à limiter le nombre de résultats retournés par la base de données. Cette commande est prise en compte dès le début de l'exécution des requêtes, ce qui implique qu'une limitation du nombre de résultat à 10, a comme conséquence que, dans ses recherches, la base de données s'arrête dès que les 10 résultats sont atteints.

Ce type de fonctionnement est très pratique et optimum, puisque ce n'est pas, comme la clause HAVING, un filtre sur le nombre de résultats qui est appliqué à la fin du traitement. La commande LIMIT peut prendre deux arguments :

LIMIT premièreLigne, nombreDeLigne ;

```
SELECT DISTINCT auteur
FROM livre
ORDER BY auteur
LIMIT 3;
```

Obtenir les trois premiers noms de l'auteur par ordre alphabétiques :

auteur
Balzac
Baudelaire
Camus

```
SELECT DISTINCT auteur
FROM livre
ORDER BY auteur
LIMIT 3, 5;
```

Obtenir les cinq suivants :

auteur
Daudet
Hugo
Maupassant
Pagnol
Prévert

JOINTURE

Les jointures sont l'expression aboutie du système relationnel des bases de données. Les jointures permettent, par l'intermédiaire des clés de chaque table d'exprimer des relations qui les lient entre elles, et ainsi, lors des sélections, d'utiliser ces relations pour restituer des données cohérentes et en relations les unes avec les autres.

Nous avons vu que les informations devaient être stockées de manière atomique dans la base de données. Cela implique que les tables fassent en permanence référence les unes avec les autres via des clés étrangères. Lors des sélections, il est souvent intéressant de récupérer toutes les informations nécessaires à un traitement en une seule fois.

La jointure permet de regrouper des informations provenant de plusieurs relations en fonction d'un critère de comparaison d'attributs (le type d'attribut doit être commun aux 2 relations) et de les présenter sous forme d'une table.

L'équijointure = est la jointure la plus fréquemment utilisée, mais les autres opérateurs de comparaisons peuvent être également utilisés : $<, \leq, >, \geq, \neq$.

La jointure naturelle est généralement basée sur l'égalité d'une clé étrangère (à la table S) et d'une clé primaire (de la table R). Dans ce cas, l'évaluation de l'expression contient autant de lignes qu'il y en a dans la table S. Le résultat est donc une version complétée (et éventuellement réduite par des conditions de sélection additionnelles) de la population S.

```
SELECT titre, auteur
FROM livre, emprunt
WHERE livre.noliv = emprunt.noliv
AND retour IS NULL;
```

Ancienne écriture : Recherche les livres (titre + auteur) en cours d'emprunt :

titre	auteur
Les chouans	Balzac
La bête humaine	Zola
Fanny	Pagnol
Les raisins de la colère	Steinbeck

*Remarquez que nous sommes obligés de préciser le nom des tables **livre** et **emprunt** devant **noliv** à l'aise de l'opérateur de séparation « **.** » pour confronter la clé primaire de l'un avec la clé étrangère de l'autre.*

```
SELECT titre, auteur
FROM livre JOIN emprunt
ON livre.noliv = emprunt.noliv
AND retour IS NULL;

Ou

SELECT titre, auteur
FROM livre INNER JOIN emprunt
ON livre.noliv = emprunt.noliv
AND retour IS NULL;
```

Nouvelle écriture (préférable) : Recherche les livres (titre + auteur) en cours d'emprunt :

titre	auteur
Les chouans	Balzac
La bête humaine	Zola
Fanny	Pagnol
Les raisins de la colère	Steinbeck

Par défaut, la jointure de deux tables est dite interne, car elle fait correspondre des lignes de deux tables en fonction d'un prédicat portant sur les colonnes des tables en jeu.

*Elle s'écrit dans la clause **FROM** de l'ordre **SELECT** à l'aide de l'opérateur **JOIN** et introduit le prédicat de jointure **ON**. Quelquefois, nous rajoutons **INNER** (facultatif) devant **JOIN** pour bien stipuler qu'il s'agit d'une jointure*

interne.

```
SELECT titre, auteur
FROM livre AS l
INNER JOIN emprunt AS e
ON l.noliv = e.noliv
AND retour IS NULL;
```

Alias : *Même si ce n'est pas le cas ici, il peut être pratique d'utiliser la notion d'alias introduit par le mot clé* ***AS*** *pour donner un surnom aux tables. Ainsi, nous pourrons préfixer les noms des colonnes avec des alias de tables, plus courts, afin de lever toute ambiguïté dans les références aux colonnes.*

```
SELECT titre, auteur
FROM livre l
JOIN emprunt e
ON l.noliv = e.noliv
AND retour IS NULL;
```

Alias : *le mot clé* ***AS*** *est optionnel.*

```
SELECT titre, nom
FROM livre l
JOIN emprunt e
ON l.noliv = e.noliv
JOIN personne p
ON e.nopers = p.nopers
AND retour IS NULL;
```

Les jointures imbriquées : ***Recherche les titres de livres en cours d'emprunt associés aux noms des emprunteurs :***

titre	nom
Les chouans	REMY
La bête humaine	DURANT
Fanny	TEZIGUE
Les lettres persanes	REMY

```
SELECT titre, auteur
FROM livre l
JOIN emprunt e
ON l.noliv = e.noliv
AND retour IS NOT NULL;
JOIN personne p
ON e.nopers = p.nopers
AND nom = 'REMY';

Ou bien :

SELECT titre, auteur
FROM livre l
JOIN emprunt e
ON l.noliv = e.noliv
JOIN personne p
ON e.nopers = p.nopers
AND nom = 'REMY';
AND retour IS NOT NULL;
```

Recherche l'ensemble des titres lus et retournés par REMY

titre	auteur
Les chouans	Balzac
Les lettres persanes	Maupassant

```
SELECT nom, titre, auteur
FROM livre
JOIN emprunt
USING (noliv)
JOIN personne
USING (nopers);
```

L'équijointure : *Nous appelons l'équijointure une jointure dont le prédicat met en relation les colonnes par équivalence de valeurs. La comparaison doit se faire au moyen de l'égalité.*

L'équijointure est le type de jointure la plus utilisée du fait du modèle relationnel.

Une syntaxe plus synthétique de l'équijointure consiste à utiliser le mot clé ***USING*** *et de spécifier la liste des colonnes jointes deux à deux. Cela permet également d'éviter les alias.*

Exemple : Recherche tous les titres de livres empruntés avec le nom de l'emprunteur.

(INTERDIT)

```
SELECT titre, auteur
FROM livre
JOIN emprunt
USING (noliv)
AND retour IS NOT NULL
JOIN personne
USING (nopers)
AND nom = 'REMY';
```

Par opposition, n'est pas une équijointure toute jointure dans laquelle le prédicat utilise une comparaison au moyen de l'inégalité, la différence ou la correspondance partielle.

Je n'ai donc pas la possibilité d'utiliser les recherches précédentes avec le mot clé USING.

L'écriture proposée à gauche n'est donc pas tolérée.

Requêtes imbriquées – Sous-requêtes

Les requêtes imbriquées sont utilisées dans une requête pour effectuer une comparaison avec des valeurs provenant d'une autre requête (sous-requête). La sous-requête est une requête de sélection entre parenthèse. Le résultat de la sous-requête doit avoir une seule colonne.

Une sous-requête est donc une requête SQL (généralement un SELECT) apparaissant entre parenthèses à l'intérieur d'une requête dite requête externe. Par opposition, la sous-requête peut être qualifiée de requête interne.

Les sous-requêtes sont utilisées dans la comparaison :

- ***Avec les opérateurs de comparaison. = , <, ≤, >, ≥, ≠.***
- ***Avec l'opérateur d'ensemble IN (appartient à l'ensemble).***
- ***Avec un test d'existence EXISTS, NOT EXISTS d'au moins une ligne dans le résultat d'une sous requête.***
- ***Avec les contrôles ANY et ALL :***
  - ***ANY qui signifie qu'au moins un élément de l'ensemble satisfait à la comparaison.***
  - ***ALL qui signifie que tous les éléments la satisfont.***

```
SELECT * FROM livre
WHERE prix >
   (SELECT AVG(prix) FROM livre);
```

Sélection des livres dont le prix est supérieur au prix moyen.

```
SELECT *
FROM   livre WHERE titre = ANY
(SELECT nom FROM     personne);
```

Sélection des livres dont le titre est égal à au moins un nom de personne.

```
SELECT titre
FROM   livre WHERE noliv  = ANY
(
  SELECT noliv
  FROM emprunt
  WHERE retour IS NULL
);
```

Recherche de tous les titres en cours d'emprunt :

```
SELECT nom, prenom
FROM   personne
WHERE ville  IN
(
  SELECT ville
  FROM personne
  WHERE nom = 'REMY'
);
```

Recherche toutes les personnes qui habitent dans la même ville que 'REMY'.

```
SELECT DISTINCT titre
FROM   livre
WHERE noliv  IN
(
  SELECT noliv
  FROM emprunt
  WHERE nopers IN
  (
    SELECT nopers
    FROM personne
    WHERE ville = 'Aurillac'
  )
);
```

Recherche les titres empruntés au moins une fois par un Aurillacois.

```
SELECT titre
FROM   livre
```

Recherche tous les romans dont le prix est inférieur à tous les livres de 'Zola'.

```
WHERE prix  < ALL
(
  SELECT prix
  FROM livre
  WHERE nopers = 'Zola'
)
AND genre = 'roman' ;
```

```
SELECT titre FROM livre
WHERE noliv  NOT IN
(SELECT noliv FROM emprunt);
```

Recherche les titres qui n'ont pas encore été empruntés.

✓ *Spécification de droits d'accès*

Privilèges

Les privilèges sont des autorisations accordées à des utilisateurs d'exécuter telle ou telle famille d'ordre SQL sur tel ou tel objet de la base. L'octroi et le retrait de privilèges reposent sur les ordres GRANT (littéralement « gratification ») et REVOKE (littéralement « révoquer »).

La syntaxe générale est la suivante :

GRANT <liste des privilèges>
ON <objet privilégié>
TO { <liste des identifiants autorisés> | PUBLIC }

Liste des privilèges : SELECT, INSERT, UPDATE, DELETE, ALL PRIVILEGES
Extension des droits a tous : PUBLIC

WITH GRANT OPTION signifie que les utilisateurs recevant les privilèges pourront les rétrocéder.

```
GRANT SELECT
ON livre
TO Pierre;
```

Autorise Pierre à lancer des ordres SQL SELECT sur le table livre.

```
GRANT INSERT, UPDATE, DELETE
ON livre, personne ;
TO Pierre, Michel ;
```

Autorise Pierre et Michel à modifier les données avec tous les ordres SQL de mise à jour (INSERT, UPDATE, DELETE), mais pas à les lire.

```
GRANT SELECT
ON livre
TO Artur WITH GRANT OPTION ;
```

Autorise Pierre à lancer des ordres SQL SELECT sur le table livre mais aussi à transmettre à tout autre utilisateur les privilèges qu'il a acquis par cet ordre.

```
GRANT SELECT, UPDATE
ON livre
TO PUBLIC ;
```

La commande suivante autorise tous les utilisateurs, présents et à venir, à lancer des ordres SQL SELECT et UPDATE sur la table livre.

```
GRANT ALL PRIVILEGES
ON livre
TO Marcel;
```

Pierre lance l'ordre suivant. Cela autorise Marcel à lancer sur la table livre, les mêmes ordres que ceux dont dispose Pierre (INSERT, UPDATE, DELETE).

✓ *Optimisation des performances : Création d'index*

Un **index** est une table de correspondance associée à une table de données qui permet d'accélérer la localisation de la ou les lignes qui possèdent des valeurs déterminées de certaines colonnes. Dans cette table de correspondance, les valeurs des colonnes sont rangées par ordre croissant ou décroissant. Un index est défini par son nom, la table à laquelle il est associé, les colonnes qui le constituent, le sens de l'ordre pour chaque colonne. L'utilisateur ignore l'existence des index. Un index est déclaré ainsi :

```
create index xnoliv on livre (noliv asc)
```

Annexe

La table **livre** :

noliv	titre	auteur	genre	prix
1	Les chouans	Balzac	roman	80
2	Germinal	Zola	roman	75
3	L'assommoir	Zola	roman	95
4	La bête humaine	Zola	roman	70
5	Les misérables	Hugo	roman	105
6	La peste	Camus	roman	112
7	Les lettres persanes	Maupassant	roman	140
8	Bel ami	Maupassant	roman	76
9	Les lettres de mon	Daudet	roman	100
10	moulin	Pagnol	roman	100
11	César	Pagnol	roman	65
12	Marius	Pagnol	roman	72
13	Fanny	Baudelaire	poésie	130
14	Les fleurs du mal	Prévert	poésie	120
15	Paroles	Steinbeck	roman	135
	Les raisins de la colère			

La table **personne** :

nopers	nom	prenom	ville
1	Durand	Jean-Pierre	Toulouse
2	Brieusel	Chantal	Colomiers
3	Riols	Jacques	Toulouse
4	Denayville	Hélène	Toulouse
5	Planchon	André	Muret
6	Pène	Gérôme	Albi
7	Bert	Jean-Pierre	St Orens
8	Gonzales	Alain	Toulouse
9	Martin	François	Balma
10	Jourda	Véronique	Colomiers

La table **emprunt** :

nopers	noliv	sortie	retour
4	14	01/01/94	
1	3	03/03/94	30/03/94
7	9	05/03/94	21/03/94
2	11	18/03/94	
3	3	30/03/94	15/04/94
3	4	30/03/94	
8	7	31/03/94	18/04/94
8	1	02/04/94	