BASES DE DONNÉES MP*

Nicolas CHIREUX

Nous nous proposons dans ce cours une approche des bases de données. Il a été vu en Sup la nécessité de l'utilisation des bases de données.

Imaginons que nous voulions enregistrer les classes, l'état civil, les options, ainsi que les notes de tous les élèves d'un lycée d'un part, l'état civil des professeurs ainsi que leurs classes d'intervention d'autre part.

Nous pourrions créer un énorme tableau avec de nombreuses colonnes. Outre l'énormité du tableau rendant toute recherche interne délicate, nous stockerions dans ce tableau des données redondantes : pour tous les élèves d'une classe ayant le même professeur, l'état civil de ce dernier figurerait à chaque ligne.

Par ailleurs

— qu'arriverait-il si nous décidions de stocker des attributs supplémentaires comme les examens déjà passés par les élèves ?

— où enregistrer ce fichier ? sur un ordinateur personnel ? Mais alors un seul utilisateur pourrait-y avoir accès ? Et si on le duplique pour permettre un accès multi-utilisateurs, comment faire les mises à jour ?

— et si on venait à changer le logiciel avec lequel on a créé le tableau, qu'arriverait-il ? et si l'ordinateur servant de sauvegarde venait à tomber en panne ?

— comment contrôler l'accès à ce fichier ? comment permettre l'accès à certaines informations et pas à d'autres ? Comment permettre à des utilisateurs de lire les informations et à d'autres de pouvoir les modifier ?

Nous allons voir que les BD répondent à ces diverses question.

1 Introduction aux bases de données relationnelles

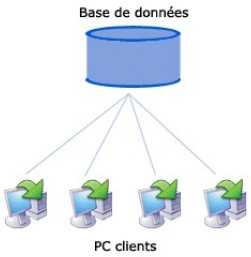
1.1 Définitions

- une **base de données** (BD) est une collection de données inter-reliées. C'est une entité cohérente logiquement et véhiculant une certaine sémantique.
- un **Système de Gestion de Bases de Données** (SGBD) est un ensemble de programmes qui permettent à des utilisateurs de créer et maintenir une base de données. Les activités supportées sont la définition d'une base de données (spécification des types de données à stocker), la construction d'une base de données (stockage des données proprement dites) et la manipulation des données (principalement ajouter, supprimer, retrouver des données). Les SGBD commerciaux les plus connus sont Oracle, Sybase, Ingres, Informix et DB2, Il existe plusieurs initiatives dans le monde du logiciel libre : MySQL, PostgreSQL.
- un SGBD sépare la partie description des données, des données elles mêmes. Cette description est stockée dans un dictionnaire de données (également géré dans le SGBD) et peut être consultée par les utilisateurs
- un **modèle de données** (MD) est un ensemble de concepts permettant de décrire la structure d'une base de données. La plupart des modèles de données incluent des opérations permettant de mettre à jour et questionner la base. **Le modèle de données le plus utilisé est le modèle relationnel**
- un **schéma de base de données (ou compréhension ou intension)** est la description des données à gérer. Il est conçu dans la phase de spécification et est peu évolutif (pratiquement statique)
- une **extension d'une base de données** correspond aux données de la base à un instant donné. Par définition cet état est dynamique.

1.2 Fonctionnalités

Les caractéristiques souhaitables des SGBD sont les suivantes :

1. **Contrôler la redondance d'informations.** La redondance d'informations pose différents problèmes (coût en temps, coût en volume et risque d'incohérence entre les différentes copies). Un des objectifs des bases de données est de contrôler cette redondance, voire de la supprimer, en offrant une gestion unifiée des informations complétée par différentes vues pour des classes d'utilisateurs différents.
2. **Partage des données.**



Une base de données doit permettre d'accéder la même information par plusieurs utilisateurs en même temps. Le SGBD doit inclure un mécanisme de contrôle de la concurrence basé sur des techniques de verrouillage des données (pour éviter par exemple qu'on puisse lire une information qu'on est en train de mettre à jour). Le partage des données se fait également par la notion de vue utilisateur, qui permet de définir pour chaque classe d'utilisateurs la portion de la base de données qui l'intéresse (et dans la forme qui l'intéresse).

3. **Gérer les autorisations d'accès.**

Une base de données étant multi-utilisateurs, se pose le problème de la confidentialité des données. Des droits doivent être gérés sur les données, droits de lecture, mise à jour, création, ... qui permettent d'affiner la notion de vue utilisateur.

4. **Offrir des interfaces d'accès multiples.** Un SGBD doit offrir plusieurs interfaces d'accès, correspondant aux différents types d'utilisateurs pouvant s'adresser à lui. On trouve des interfaces orientées utilisateur final (langages de requêtes déclaratifs comme SQL avec mise en œuvre graphique, interface de type formulaire, ...) ou bien orientées programmeurs d'applications (interface avec des langages de programmation classiques comme par exemple l'approche SQL immergé ou "embedded SQL").
5. **Représenter des relations complexes entre les données.** Un SGBD doit permettre de représenter des données inter-reliées de manière complexe. Cette facilité s'exprime à travers le modèle de données sous-jacent au SGBD. Chaque modèle de données offre ses propres concepts pour représenter les relations. Nous nous intéresserons cette année au modèle relationnel.
6. **Vérifier les contraintes d'intégrité.** Un schéma de base de données se compose d'une description des données et de leurs relations ainsi que d'un ensemble de contraintes d'intégrité. Une **contrainte d'intégrité** est une propriété de l'application à modéliser qui renforce la connaissance que l'on en a.
   On peut classifier les contraintes d'intégrité, en contraintes structurelles (un employé a un chef et un seul par exemple) et contraintes dynamiques (un salaire ne peut diminuer dans la fonction publique).
7. **Assurer la sécurité et la reprise après panne.** Une base de données est souvent vitale dans le fonctionnement d'une organisation, et il n'est pas tolérable qu'une panne puisse remettre en cause son fonctionnement de manière durable. Les SGBD fournissent des mécanismes pour assurer cette sécurité.
   Le premier mécanisme est celui de transaction qui permet d'assurer un comportement atomique à une séquence d'actions (elle s'effectue complètement avec succès ou elle est annulée). Une transaction est une séquence d'opérations qui fait passer la base de données d'un état cohérent à un nouvel état cohérent. L'exemple typique est celui du débit-crédit pour la gestion d'une carte bancaire. Ce mécanisme permet de s'affranchir des petites pannes (style coupure de courant).
   En ce qui concerne les risques liés aux pannes disques, les SGBD s'appuient sur un mécanisme de journalisation qui permet de régénérer une base de données automatiquement à partir d'une version de sauvegarde et du journal des mouvements.

1.3 Architecture logique d'un SGBD

La plupart des SGBD suivent l'architecture standard Ansi/Sparc qui permet d'isoler les différents niveaux d'abstraction nécessaires pour un SGBD.

1.3.1 Description

Elle est définie sur trois niveaux :

- **niveau interne ou physique :** décrit le modèle de stockage des données et les fonctions d'accès, les détails de l'organisation sur disque et de structures comme des index dont le rôle est d'accélérer les calculs.
- **modèle conceptuel ou logique :** décrit la structure de la base de données globalement à tous les utilisateurs (limite la redondance). Le schéma conceptuel est produit à partir d'une analyse du problème à modéliser et des besoins des utilisateurs (intégration des différentes vues utilisateurs). Ce schéma décrit la structure de la base indépendamment de son implantation. On ne se préoccupe pas ici de savoir comment le schéma créé va être implanté physiquement.

- **niveau externe :** correspond aux différentes vues des utilisateurs. Chaque schéma externe donne une vue sur le schéma conceptuel à une classe d'utilisateurs.

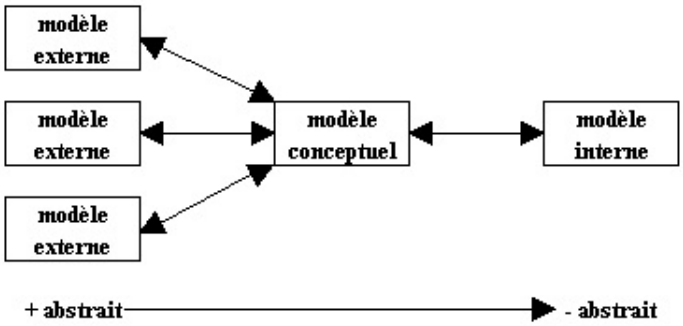


FIGURE 1 – Architecture SGBD

Le SGBD doit être capable de faire des transformations entre chaque niveau, de manière à transformer une requête exprimée en terme du niveau externe en requête du niveau conceptuel puis du niveau physique.

1.3.2 Indépendance données - programmes

L'architecture à trois niveaux permet de supporter le concept d'indépendance données - programmes, c'est à dire la capacité de modifier le schéma de la base de données à un niveau donné, sans remettre en cause le schéma aux niveaux supérieurs :

- **indépendance logique :** on peut changer le niveau conceptuel sans remettre en cause les schémas externes ou les programmes d'application. L'ajout ou le retrait de nouveaux concepts ne doit pas modifier des éléments qui n'y font pas explicitement référence,
- **indépendance physique :** on peut changer le schéma physique sans remettre en cause le schéma conceptuel (et les schémas externes). On peut modifier l'organisation physique des fichiers, rajouter ou supprimer des méthodes d'accès.

Le modèle relationnel, contrairement à ces prédécesseurs, permet un certain niveau d'indépendance sans aller jusqu'à une indépendance complète.

2 Modèle relationnel de données

Le modèle relationnel de données a été défini en 1970 par Codd et les premiers systèmes commerciaux sont apparus au début des années 80. Le modèle relationnel est simple (3 concepts), facile à appréhender, même pour un non spécialiste et repose sur de solides bases théoriques qui permettent notamment de définir de façon formelle les langages de manipulation associés.

Le modèle relationnel représente l'information dans une collection de relations. Chaque relation est un tableau à deux dimensions. Chaque ligne de la table (appelée n-uplet ou **tuple**) peut être vue comme un fait décrivant une entité du monde. Une colonne de la table est appelée un **attribut**.

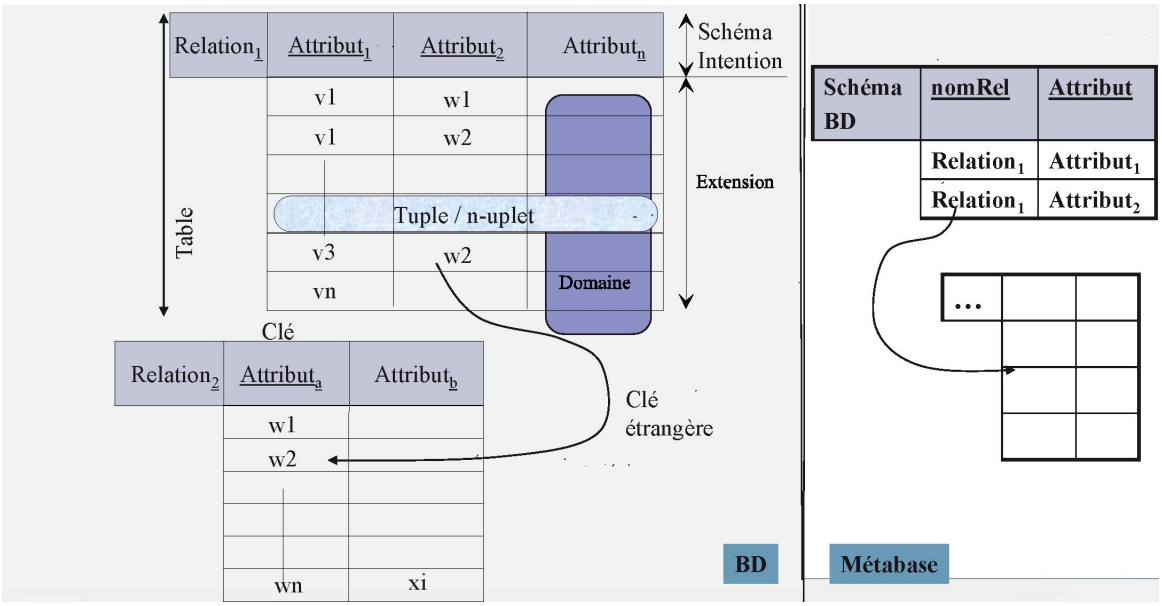


FIGURE 2 – Synthèse des concepts en BDD relationnelle

2.1 Définition formelle

- un **domaine** $\mathcal{D}$ est un ensemble de valeurs atomiques. Le terme atomique signifie que ces valeurs sont considérées comme insécables au niveau du modèle.
  Par exemple :
  — Entier, réel, chaîne de caractères, booléen
  — Jour = 1...31
  — Couleur = 'rosé', 'blanc', 'rouge'
- un **schéma de relation** R, dénoté par $R(A_1, A_2, ..., A_n)$, est un ensemble d'attributs $R = A_1, A_2, ..., A_n$. A chaque attribut A_i est associé un domaine $\mathcal{D}_i$, A_i indiquant le rôle joué par le domaine $\mathcal{D}_i$ dans la relation R : les valeurs de l'attribut seront obligatoirement prises dans le domaine associé. Un schéma de relation décrit une relation et représente l'**intension** de celle-ci. Le degré de la relation est le nombre d'attributs de celle-ci.
  Par exemple :
  — ETUDIANT(Nom, No-ss, adresse, age, diplôme)
  — FILMS (id, titre, annee, score, nbvotant, idrealisateur)
  — VINS (Num, Cru, Annee, Degre)
- une **relation** r sur le schéma de relation $R(A_1, A_2, ..., A_n)$ est un ensemble de n-uplets $r = t_1, t_2, ..., t_n$. r est souvent appelée **extension** du schéma R.
  On peut également définir une relation à partir du produit cartésien des domaines de son schéma R :

$$r(R) \subseteq \mathcal{D}_1 \times \mathcal{D}_2 \times ... \times \mathcal{D}_n \quad (1)$$

  Il est important de noter qu'un schéma de relation est quasi invariant dans le temps, alors que l'extension représente les données présentes à un instant donné dans la base.
  Caractéristiques des relations
  — une relation est un ensemble de n-uplets, il n'y a donc pas de notion d'ordre sur les n-uplets
  — par contre un n-uplet est un séquence ordonnée d'attributs
  — une valeur d'attribut est atomique mais peut être éventuellement nulle (valeur particulière qui indique que la valeur est manquante)
- une **clé** est un groupe minimum d'attributs qui détermine un n-uplet unique dans une relation (à tout instant).
- une **clé étrangère** est un groupe d'attributs qui apparaît comme clé dans une autre relation

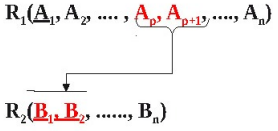


FIGURE 3 – Clé étrangère

- une **métabase** est une base de données contenant l'ensemble des schémas et des règles de correspondances associées à une base de données. C'est en gros une base décrivant les autres bases, c'est-à-dire :
  — les relations
  — les attributs
  — les domaines
  — les clés
  — ...

2.2 Contraintes d'intégrité

Un schéma de base de données est un ensemble de schémas de relation $S = R_1, R_2, ..., R_n$ et un ensemble de contraintes d'intégrité CI.

Une **contrainte d'intégrité** est une propriété du schéma, invariante dans le temps. On peut distinguer plusieurs catégories de contraintes :

— les **contraintes structurelles** qui définissent plus précisément la structure des associations entre les données (le modèle de données les supporte en partie)

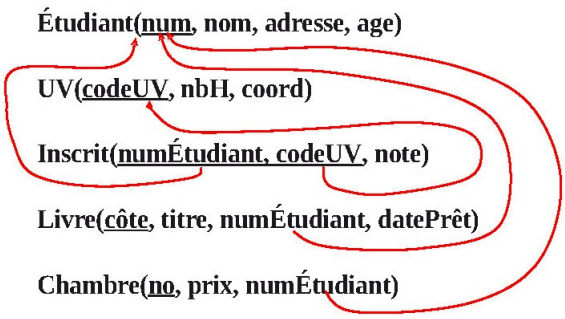
— les **contraintes sur les valeurs** qui donnent des relations entre les données (un chef gagne plus que ses subordonnés...en théorie). La plupart des contraintes ne sont pas supportées pas le modèle de données et doivent donc être codées par les programmeurs dans des programmes d'application.

— les **contraintes d'intégrité sur les clés**. Par définition, tous les n-uplets d'une relation sont distincts deux à deux (puisqu'il s'agit d'un ensemble !). Le système doit donc garantir l'unicité des valeurs de clé (un seul n-uplet FILM peut avoir une valeur de id donnée).

  Il peut y avoir plusieurs clés pour une même relation (elles sont alors appelées clés candidates) et on choisit le plus souvent une clé particulière dite **clé primaire** qui servira d'identification privilégiée des n-uplets. Cette clé sera indiquée de manière graphique en soulignant par exemple les attributs la composant.

— les **contraintes d'intégrité sur les entités**. Une clé primaire ne peut contenir de valeur nulle.

— les **contraintes d'intégrité référentielle**. C'est une contrainte exprimée entre deux relations. Ce sont les clés étrangères qui les définissent. Intuitivement cela consiste à vérifier que l'information utilisée dans un n-uplet pour désigner un autre n-uplet est valide, notamment si le n-uplet désigné existe bien.
  Plus précisément à l'insertion, il faut que la valeur des attributs existe dans la relation référencée. Lors de la suppression dans la relation référencée, les n-uplets référençant doivent aussi disparaître



Par exemple, si on considère une relation **Etudiant**(**num**, nom, adress, age)) et une relation **UV**(**codeUV**, nbH, coord), un nuplet de **Inscrit** référence un nuplet de **UV** via l'attribut **codeUV** et un nuplet de **Etudiant** via l'attribut **num**. Il est important de s'assurer que les n-uplets référencés existent bien. Cette contrainte implique un ordre dans la création ou la destruction des entités, puisqu'on ne peut créer un **Inscrit** si les n-uplets correspondant à **Etudiant** et à **UV** n'ont pas été créés et on ne peut détruire un **Etudiant** si **Inscrit** existe toujours.

FIGURE 4 – Contraintes d'intégrité référentielle à l'aide des clés étrangères

3 L'algèbre relationnelle

L'algèbre relationnelle se compose d'un ensemble d'opérateurs opérant sur des relations et produisant de nouvelles relations. Il est donc possible de construire de nouvelles informations à partir des relations de départ et d'une composition séquentielle d'opérateurs.

3.1 Principaux opérateurs de l'algèbre relationnelle

De nombreux opérateurs relationnels ont été proposés, on peut cependant présenter ici les plus courants. On peut classifier les opérateurs relationnels en trois catégories :

- les opérateurs unaires : sélection et projection
- les opérateurs binaires travaillant sur des relations de même schéma : union, intersection, différence

- les opérateurs binaires travaillant sur des relations de schémas différents : jointure, produit cartésien, division

Opérateur	Sémantique	Notation textuelle	Schéma
Restriction	Sélectionner des tuples	$T \leftarrow \sigma_{cond}(R)$	Schéma(T)=Schéma(R)
Projection	Sélectionner des attributs	$T \leftarrow \Pi_{attributs}(R)$	Schéma(T)⊆Schéma(R)
Union	Fusionner les extensions de deux relations	$T \leftarrow (R) \cup (S)$	Schéma(T)=Schéma(R)=Schéma(S)
Intersection	Obtenir l'ensemble des tuples communs à deux relations	$T \leftarrow (R) \cap (S)$	Schéma(T)=Schéma(R)=Schéma(S)
Différence	Concaténer chaque tuple de R avec chaque tuple de S	$T \leftarrow (R) - (S)$	Schéma(T)=Schéma(R)=Schéma(S)
Produit cartésien	Obtenir l'ensemble des tuples d'une relation qui ne figurent pas dans l'autre	$T \leftarrow (R) \times (S)$	Schéma(T)=Schéma(R)∪Schéma(S)
Jointure	Etablir le lien sémantique entre les relations	$T \leftarrow (R) \underbrace{\bowtie}_{cond} (S)$	Schéma(T)=Schéma(R)∪Schéma(S)
Division	Répondre aux requêtes de tuple "tous les"	$T \leftarrow (R) \div (S)$	Schéma(T)=Schéma(R)∪Schéma(S)

3.2 Détail des divers opérateurs illustrés sur un exemple

La présentation des opérateurs algébriques sera ici combinée avec un exemple de gestion d'une base d'invitations. Cette base de données décrit les personnes invitées et les plats qui ont été servis. Elle est composée de trois relations :

- REPAS(**date**,**invité**) donne la liste des invités qui ont été reçus et à quelle date
- MENU(**date**,**plat**) donne le menu servi à chaque date
- PREFERENCE(**personne**,**plat**) donne pour chaque personne ses plats préférés

Rem : les attributs "personne" et "invité" ont même domaine et les clés sont en gras.

Rem : les requêtes sont aussi données en SQL à titre d'information. Il sera bon d'y revenir après lecture du paragraphe concernant le lanquage SQL.

3.2.1 Restriction

La restriction sert à sélectionner des tuples. L'expression de restriction est une condition booléenne élémentaire ou combinée telle que vue précédemment.

Exemple 1 : Quels sont les invités du repas du 010597 ?
- en algèbre : $T \leftarrow \sigma_{date=010597}(REPAS)$
- en SQL : SELECT * FROM REPAS WHERE date='010597'

Exemple 2 : Faire la liste des plats préférés d'Alice ?
- en algèbre : $T \leftarrow \sigma_{personne=Alice}(PREFERENCES)$
- en SQL : SELECT * FROM PREFERENCES WHERE personne='Alice'

3.2.2 Projection

La projection est souvent combinée avec la restriction. Elle sert à ne sélectionner que des attributs précis lors d'une requête. Il faut noter que la cardinalité de la nouvelle relation est inférieure ou égale à celle de la relation de départ, puisque des doublons ont pu être produits par la projection et sont donc supprimés (une relation est toujours un ensemble)

Exemple 1 : Quels sont les invités du repas du 010597 ?
- en algèbre : $T \leftarrow \Pi_{invites}\,(\sigma_{date=010597}(REPAS))$
- en SQL : SELECT DISTINCT invité FROM REPAS WHERE date='010597'

Exemple 2 : Faire la liste des plats préférés d'Alice ?
- en algèbre : $T \leftarrow \Pi_{plat}\,(\sigma_{personne=Alice}(PREFERENCES))$
- en SQL : SELECT plat FROM PREFERENCES WHERE personne='Alice'

Ici on n'affiche qu'un attribut précis pour chaque tuple.

3.2.3 Jointure

Il faut que les schémas de R et de S aient une intersection non vide. La relation produite a un schéma qui est l'union des schémas de R et de S et dont les n-uplets sont la concaténation des n-uplets de R avec ceux de S s'ils ont même valeur pour tous les attributs communs.

En pratique une jointure sert à chercher des informations dans deux relations distinctes en utilisant une clé étrangère.

Exemple 3 : Quels sont les plats qui ont été servis à Alice ?
Il faut ici appliquer une jointure pour créer une relation intégrant à la fois les noms des invités et les plats consommés. La jointure se fait sur la date qui est une clé étrangère de MENU. Une restriction sur la relation obtenue permet de trouver les tuples correspondant à Alice. Enfin une projection sera réalisée pour n'afficher que l'attribut plat.

- en algèbre : $T \leftarrow \Pi_{plat}\left(\sigma_{invite=Alice}(REPAS \underbrace{\bowtie}_{REPAS.date=MENU.date} MENU)\right)$
- en SQL : SELECT DISTINCT plat FROM REPAS R, MENU M WHERE R.date=M.date AND invité='Alice'
  Rem : Notez comment en SQL, on peut renommer les relations afin d'alléger l'écriture. Notez aussi que comme les deux clés date ont le même nom, il faut les rapporter à leur relation d'origine par R.date ou M.date

3.2.4 Union - Intersection - Différence

Les trois opérateurs ensemblistes opèrent sur des relations R et S **de même schéma**. **Il faudra donc penser à faire des projections avant d'utiliser ces opérateurs afin de rendre les schémas égaux.**

- union : $T \leftarrow (R) \cup (S)$ produit une nouvelle relation ayant les n-uplets de R et ceux de S (les doublons sont supprimés).
- intersection : $T \leftarrow (R) \cap (S)$ produit une nouvelle relation ayant les n-uplets présents dans R et dans S.
- différence : $T \leftarrow (R) - (S)$ produit une nouvelle relation ayant les n-uplets de R qui ne sont pas dans S. **Attention, la différence n'est pas commutative**.

Exemple 4 : Quelles sont les personnes qui n'ont jamais été invités ?
Il faut ici commencer par aligner les schémas de la relation PREFERENCES et REPAS avant de faire la soustraction.

- en algèbre : $T \leftarrow \Pi_{personne}(PREFERENCES) - \Pi_{invite}(REPAS)$
- en SQL : SELECT personne FROM PREFERENCES EXCEPT SELECT invité FROM REPAS
  Rem : Notez le caractère non commutatif de la soustraction.

3.2.5 Division

Le but de cet opérateur est de répondre aux requêtes de type "tous les".

$T \leftarrow (R) \div (S)$ un tuple t sera dans T si pour tout tuple s de S, le tuple $< t, s >$ est dans R.

La division peut se définir à l'aide du produit cartésien, de la différence et de la projection : soit R la relation $R(A_1, .., A_p, A_{p+1}, ..., A_n)$ et soit S la relation $S(A_{p+1}, ..., A_n)$ alors :

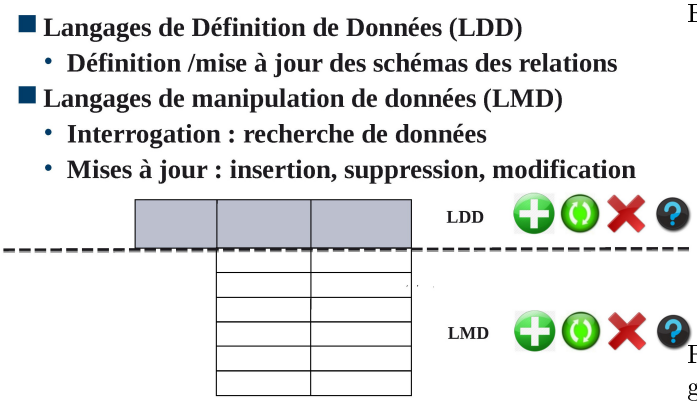
$R \div S = R1 - R2$ avec $R1 = \Pi_{A_1,...,A_p}(R)$ et $R2 = \Pi_{A_1,...,A_p}((R1 \times S) - R)$

Exemple 5 : Quels sont les invités qui sont venus à tous les repas ?
Il faut ici commencer par aligner les schémas de la relation PREFERENCES et REPAS avant de faire la soustraction.

- en algèbre : $T \leftarrow REPAS \div \Pi_{date}(REPAS)$
- en SQL :
  SELECT invité FROM REPAS
  GROUP BY invité
  HAVING count(*) = (SELECT count(distinct date) FROM REPAS)

4 Langage SQL - Interrogation d'une base de données

Nous allons maintenant présenter les bases d'un des langages les plus utilisés pour exprimer les diverses requêtes d'algèbre relationnelle. Nous verrons que ce langage va bien au delà.



En fait , les SGBD doivent implanter deux langages :

1. le **Langage de Définition de Données** (LDD) qui permet de créer le schéma conceptuel de la base de données, les schémas externe, le schéma physique ...
2. le **Langage de Manipulation de Données** (LMD) qui permet d'interroger la base de données bien sûr, mais aussi de la modifier (crée, supprimer ou modifier des tuples)

FIGURE 5 – Langage de Définition de Données - Langage de Manipulation de Données

SQL est un langage déclaratif qui va répondre aux divers besoins énoncés précédemment. L'utilisateur y énonce les propriétés du résultat souhaité mais ne décrit pas les différentes étapes à suivre pour construire ce résultat.

SQL est une implémentation de langages formels : il est emprunté à l'algèbre relationnelle et au calcul relationnel de tuples. Ceci garantit à ce langage des fondements théoriques solides.

C'est un langage très puissant car il allie à l'algèbre relationnelle des fonctions-agrégats ainsi que des tris. Une requête SQL sans fonctions et tri est une simple suite d'opérations de l'algèbre relationnelle.

4.1 Syntaxe générale

Afin de présenter les divers aspect de SQL nous nous aiderons d'un exemple. Il s'agit d'une base de données d'une coopérative vinicole. Son schéma relationnel est le suivant :

- Vins (**Num**, Cru, Annee, Degre) dont la clé est Num
- Producteurs (**Num**, Nom, Prenom, Region) dont la clé est Num
- Recoltes (**Nprod**, **Nvin**, Quantite) dont les clé étrangères sont Nprod et Nvin
- Clients(**Num**, Nom, Prenom, Adresse,Ncommande)dont la clé est Num et Ncommande est une clé étrangère
- Commande(**Num**,Nclient, Nvin, Quantite) dont la clé est Num, Nclient et Nvin sont des clés étrangères
- Livraison(**Ncom**,quantite_livree) qui fonctionne avec une seule clé étrangère Ncom

Une requête SQL classique se présente sous la forme SELECT ...(3) FROM ...(1) WHERE ...(2)

Le champ (1) du FROM se complète en premier en répondant aux questions suivantes : **dans quelles relations sont les attributs dont j'ai besoin ? ai-je plusieurs relations dans ma clause FROM ? Si oui, quelles sont les conditions de jointure(s) ?** Si plusieurs relations sont nécessaires, on les ajoutera grâce à divers JOIN.

Le champ (2) du WHERE se complète ensuite en répondant aux questions : **ai-je des conditions sur les valeurs d'attributs exprimées dans ma requête ?**

Si toutes les réponses sont négatives la clause WHERE n'apparaitra pas.

Enfin le champ (3) du SELECT se complète en dernier en répondant à la question : **quel résultat souhaite voir l'utilisateur (schéma du résultat) ?**

Exemple 1 : Donner les vins de degré compris entre 8 et 12.

```
SELECT * FROM Vins WHERE degre >=8 AND degre <=12 ;
SELECT * FROM Vins WHERE degre BETWEEN 8 AND 12 ;
SELECT * FROM Vins WHERE degre IN (8, 9, 10, 11, 12) ;
```

Quel souci peut-on avoir avoir la dernière proposition ci-dessus ?

Exemple 2 : Donner les vins dont le cru commence par p ou P.
SELECT * FROM Vins WHERE cru LIKE 'p%' OR cru LIKE 'P%';

Donner les crus des vins de millésime 1995 et de degré 12, triés par ordre croissant.
SELECT cru FROM Vins WHERE annee=1995 AND degre=12 ORDER BY cru;

Donner les crus des vins de millésime 1995 et de degré 12, triés par ordre décroissant.
SELECT cru FROM Vins WHERE annee=1995 AND degre=12 ORDER BY cru DESC;

4.2 Jointure

Les conditions de jointure seront données dans la clause WHERE. Il faut faire attention aux clés dont les noms sont identiques alors qu'elles caractérisent des relations différentes. Pour cela, il sera souvent utile pour alléger l'écriture d'utiliser des alias (noms abrégés pour les relations).

Exemple 1 : Donner les noms des producteurs de Pommard.
```
SELECT nom
FROM Vins V
JOIN Recoltes R ON V.Num=R.Nvin
JOIN Producteurs P ON P.Num=R.Nprod
WHERE cru='Pommard'
```

On peut aussi réaliser des jointures d'une relation avec elle-même. On parle d'autojointure. Le rôle des alias prend ici tout son sens car il faut définir des synonymes pour les relations.

Exemple 2 : Donner les couples de producteurs produisant le même vin.
```
SELECT P1.Num, P2.Num
FROM Producteurs P1, Producteurs P2
JOIN Recoltes R1 ON P1.Num = R1.Nprod
JOIN Recoltes R2 ON P2.Num = R2.Nprod
WHERE R1.Nvin = R2.Nvin AND P1.Num > P2.Num
```

Quel est l'intérêt de la dernière condition de la clause WHERE ?

4.3 Opérateurs ensemblistes

Nous retrouvons ici les trois opérateurs UNION, INTERSECT et EXCEPT.

Exemple 1 : Donner l'ensemble des noms des producteurs et des clients.
```
SELECT Nom FROM Producteurs UNION SELECT Nom FROM Clients
```

Exemple 2 : Donner l'ensemble des noms des producteurs qui sont aussi clients.
```
SELECT Nom FROM Producteurs INTERSECT SELECT Nom FROM Clients
```

Exemple 3 : Donner l'ensemble des noms des producteurs qui ne sont pas clients.
```
SELECT Nom FROM Producteurs EXCEPT SELECT Nom FROM Clients
```

4.4 Fonctions agrégats

Il existe 5 fonctions prédéfinies : COUNT, SUM, MIN, MAX, AVG. (AVG est la moyenne). Elles s'appliquent à l'ensemble des valeurs d'un attribut d'une relation. Elles produisent une valeur unique.

Pour une requête sans partionnement (nous verrons ce concept ultérieurement), ces fonctions se placent uniquement dans le SELECT, jamais dans le WHERE. Par ailleurs, on veillera à ne pas mélanger dans le SELECT les fonctions et les attributs simples afin d'éviter les conflits au cas où l'attribut serait multivalué.

Exemple 1 : Donner la moyenne des degrés de tous les vins.
SELECT AVG(degre) FROM Vins;

Donner la quantité totale commandée par le client de nom Poivrot.
SELECT SUM(Quantité)
FROM Commandes
JOIN Clients ON Clients.Num=Commandes.Nclient
WHERE Clients.Nom= 'Poivrot'

Donner le nombre de crus différents.
SELECT COUNT (DISTINCT cru) FROM Vins;

Donner le nombre de vins.
SELECT COUNT(*) FROM Vins;

Exemple 2 : Donner les vins dont le degré est supérieur à la moyenne des degrés des vins.
SELECT *
FROM Vins
WHERE degre > (SELECT AVG(degre) FROM Vins);

Exemple 3 : Donner les numéros de commande où la quantité commandée a été totalement livrée
SELECT Num
FROM Commandes C
WHERE Quantite = (SELECT SUM(L.quantite_livree)
FROM Livraisons L
WHERE L.Ncom = C.Num)

4.5 Partitionnement

Le partitionnement horizontal d'une relation, selon les valeurs d'un attribut ou d'un groupe d'attributs qui est spécifié dans la clause GROUP BY va permettre de fragmenter une relation en groupes de tuples, où tous les tuples de chaque groupe ont la même valeur pour l'attribut (ou le groupe d'attributs) de partitionnement (qui est dans le GROUP BY).

Nous pourrons ensuite appliquer des fonctions sur les groupes ou appliquer des restrictions sur les groupes à l'aide de la clause HAVING

Exemple 1 : Donner, pour chaque cru, la moyenne des degrés des vins de ce cru.
SELECT cru, AVG(degre) FROM Vins GROUP BY cru
Rem : on peut ici faire coexister dans le SELECT un attribut et une fonction puisque le fait d'avoir créé des groupes m'assure que l'attribut est monovalué comme la fonction et donc que je n'aurai pas de conflit.

Donner, pour chaque cru, la moyenne des degrés des vins de ce cru avec un tri par degré décroissant.
SELECT cru, AVG(degre) FROM Vins GROUP BY cru ORDER BY AVG(degre) DESC;

Donner, pour chaque cru, la moyenne des degrés des vins de ce cru uniquement si ce cru concerne plus de 3 vins.
SELECT cru, AVG(degre) FROM Vins GROUP BY cru HAVING COUNT(*)>=3;

Pour obtenir le résultat de la requête, SQL va procéder ainsi :

1. Trier la relation selon les attributs de groupement
2. Créer une sous-relation pour chaque paquet ayant même valeur sur l'ensemble des attributs de groupement, ici « cru »
3. Calculer les fonctions (clause SELECT ou HAVING ou ORDER BY) sur chaque partition (dans notre exemple la valeur de cru et la moyenne des degrés sur la partition)
4. Appliquer la restriction du HAVING

5. Unifier les résultats

   Rem : pour être assuré d'avoir toujours des attributs monovalué quand le SELECT comprend une fonction agrégat, on s'imposera d'avoir dans le GROUP BY au moins les mêmes attributs que dans le SELECT, les fonctions-agrégats ne comptant pas ici.

4.6 Prédicats

Nous disposons de 4 prédicats : ALL, ANY, EXISTS et NOT EXISTS.

ALL teste si la valeur d'un attribut satisfait un critère de comparaison avec tous les résultats d'une sous-requête

Exemple : Quels sont les numéros et noms des clients ayant passé la plus grosse commande en quantité ?

```
SELECT Cl.Num, Cl.Nom
FROM Clients Cl
JOIN Commandes C ON Cl.Num = C.Nclient
WHERE C.Quantite >= ALL(SELECT Quantite FROM Commandes) ;
```

ANY teste si la valeur d'un attribut satisfait un critère de comparaison avec au moins un résultat d'une sous-requête

Exemple : Quels sont les numéros et noms des clients ayant passé une commande au moins supérieure en quantité à une autre ?

```
SELECT Cl.Num, Cl.Nom
FROM Clients Cl
JOIN Commandes C ON Cl.Num = C.Nclient
WHERE C.Quantite >= ANY(SELECT Quantite FROM Commandes) ;
```

EXISTS teste si la réponse à une sous-requête est non vide. **NOT EXISTS** teste si la réponse à une sous-requête est vide.

Exemple : Quels sont les producteurs ayant produit au moins un vin ?

```
SELECT P.*
FROM Producteurs P
WHERE EXISTS (SELECT R.*
              FROM Recoltes R
              WHERE P.Num = R.Nprod) ;
```

4.7 Division

La division est délicate à traduire de l'algèbre relationnelle en SQL. Nous allons voir sur un exemple deux façons de faire.

4.7.1 Division avec prédicat EXISTS

L'étape préalable dans ce cas consistera à reformuler la requête à l'aide d'une double négation.

Exemple : Quels sont les producteurs ayant produit tous les vins ?

Reformulation : Un producteur est sélectionné s'il n'existe aucun vin pour lequel il n'y a aucune récolte (sous entendu de ce producteur et de ce vin)

```
SELECT P.*
FROM Producteurs P
WHERE NOT EXISTS (SELECT V.*
                  FROM Vins V
                  F WHERE NOT EXISTS (SELECT R.*
                                      FROM Recoltes R
                                      WHERE R.Nprod=P.Num AND R.Nvin=V.Num))
```

4.7.2 Division sans prédicat EXISTS

Exemple : Quels sont les producteurs ayant produit tous les vins ?

```
SELECT P.*
FROM Producteurs P
WHERE P.Num IN ( SELECT Nprod
                 FROM Recoltes
                 GROUP BY Nprod
                 HAVING COUNT(*)=(SELECT COUNT(*)
                                  FROM Vins))
```

4.8 Synthèse

N	Instruction	Argument	Résultat
6	SELECT	liste et/ou expressions attributs A_j et/ou fonctions sur attributs A_p	Projection de l'ensemble obtenu en (5) sur les A_j, calcul des expressions, calcul des fonctions (appliquées aux groupes s'il y en a) sur A_p
1	FROM	liste de relations R_i	Produit cartésien des relations R_i
2	WHERE	conditions sur les tuples C_i	Sélection de tuples de (1) respectant la condition C_i
3	GROUP BY	liste d'attributs $A_k \supseteq A_j$	Partitionnement de l'ensemble obtenu en (2) suivant les valeurs A_k
4	HAVING	conditions sur groupes-fonctions C_2	Sélection de groupes de (3) vérifiant C_2
5	ORDER BY	liste d'attributs A_1 ou numéro d'ordre dans le SELECT	Tri des tuples obtenus en suivant les valeurs A_1

Les conditions de recherche dans le WHERE (sélection de tuples) ou le HAVING (sélection de groupes) peuvent être élémentaires ou composées par les opérateurs **AND**, **OR** et **NOT**. Elles doivent obligatoirement être VRAIES ou FAUSSES.

Les conditions élémentaires sont :

- une comparaison Attribut/Valeur ou Attribut/Attribut du type $=, <, <=, >, <=, <>$
- l'appartenance à un intervalle : **BETWEEN**
- l'identité à une chaîne de caractères : **LIKE**. Les "jokers" sont % qui remplace un nombre variable de caractères et _ qui remplace un seul caractère.
- la nullité (pas d'entrée) : **IS NULL**
- l'appartenance à un domaine : **IN**
- la quantification : **EXISTS, ANY, ALL**

5 Langage SQL - Définition et contrôle des données

5.1 Domaines

Les domaines utilisés dans les bases de données sont conformes aux types classiquement utilisés dans les langages de programmation, à savoir :

- Entier : INTEGER
- Décimal : DECIMAL (m,n)où m est le nombre de chiffres et n le nombre de chiffres après la virgule. Ainsi DECIMAL(4,2) peut représenter des chiffres comme 99,99
- Réel flottant : FLOAT
- Chaîne de caractères : CHAR (n) et VARCHAR(n). Les CHAR font systématiquement n caractères (les chaînes plus courtes sont complétées par des espaces, les VARCHAR sont taillés au plus juste dans la limite des n caractères
- Date : DATE (dans la norme SQL2 !). Ce type normalisé très tard possède une manipulation sensible. Il faut bien connaître le format d'expression de la date et sa langue.Le 1er janvier 2010 peut par exemple s'exprimer en '2010-01-01' ou '01-JAN-2010' ou '1/1/10' ...
- il n'y a pas de type booléen en SQL2

Rem : en SQL3, il y a de nombreux autres domaines mais les utiliser vous expose à des erreurs si la norme SQL3 n'est pas supportée. On peut citer ENUM pour les énumérations, IMAGE, MONEY, ARRAY...

5.2 Création d'un schéma de relation

L'ordre de création de relation minimal suit la syntaxe de l'exemple ci-après :

Exemple :

```
CREATE TABLE vin
( nv INTEGER,
cru CHAR(20),
mil INTEGER) ;
```

5.3 Ajout d'un attribut (norme SQL2)

L'ajout d'un attribut suit la syntaxe de l'exemple ci-après :

Exemple :

```
ALTER TABLE vin
ADD COLUMN deg INTEGER ;
```

5.4 Suppression d'un schéma de relation (norme SQL2)

La suppression d'un schéma de relation suit la syntaxe de l'exemple ci-après :

Exemple :

```
DROP TABLE vin ;
```

5.5 Contraintes d'intégrité

Une contrainte d'intégrité est une règle qui définit la cohérence d'une donnée ou d'un ensemble de données de la BD

Très peu de contraintes sont considérées dans la norme SQL 1 :

- non nullité des valeurs d'un attribut
- unicité de la valeur d'un attribut ou d'un groupe d'attributs
- valeur par défaut pour un attribut
- contrainte de domaine
- intégrité référentielle "minimale"

Exemple :

```
CREATE TABLE vin (
        nv INTEGER UNIQUE NOT NULL,
        cru CHAR(20),
        mil INTEGER,
        deg INTEGER BETWEEN 5 AND 15 ) ;
```

La norme SQL2 prend en charge de nouvelles contraintes d'intégrités, plus complexes. Les plus remarquables sont la notion de clé primaire et les contraintes d'intégrité référentielles avec gestion des suppressions, des mises à jours en cascade

6 Langage SQL - Mise à jour des données

Le langage de mise à jour couvre trois fonctionnalités :

- l'insertion de tuples (un seul tuple ou un ensemble de tuples)
- la suppression d'un ensemble de tuples
- la modification d'un ensemble de tuples

6.1 Insertion de tuples

6.1.1 Insertion d'un seul tuple

Si tous les attributs sont renseignés, il est possible de ne pas spécifier le noms des attributs. Dans ce cas, il faut renseigner les attributs dans l'ordre de leur création (cf. ordre utilisé au CREATE TABLE).

Exemple :
INSERT INTO vins VALUES(100, 'Jurançon', 1979, 12) ;

Si quelques attributs seulement sont renseignés, il faut préciser leur nom. Les attributs non renseignés prennent alors la valeur NULL.

Exemple :
INSERT INTO vins (nv, cru) VALUES (200, 'Gamay') ;

6.1.2 Insertion d'un ensemble de tuples

Il est possible d'insérer tous les tuples résultat d'une requête au moyen d'une seule requête :

Exemple :
CREATE TABLE bordeaux
(nv INTEGER, mil INTEGER, deg INTEGER) ;
INSERT INTO bordeaux
SELECT nv, mil, deg
FROM vins
WHERE cru= 'Bordeaux' ;

6.2 Suppression de tuples

L'instruction utilisée est DELETE. Elle peut être assortie d'une clause WHERE afin de préciser les tuples à supprimer.

Exemple :
Supprimer tous les tuples de VINS
DELETE FROM vins ;
Supprimer le vin de numéro 150
DELETE FROM vins WHERE nv = 150 ;
Supprimer les vins de degré <9 ou >12
DELETE FROM vins WHERE deg < 9 OR deg > 12 ;
Supprimer les commandes passées par Dupond
DELETE FROM commandes
WHERE nb IN (SELECT nb FROM buveurs WHERE nom= 'Dupond') ;

6.3 Modification des valeurs de tuples

Pour modifier les valeurs d'un attribut d'un tuple donné, on utilisera la commande UPDATE.

Exemple :
Mettre la ville du viticulteur 150 à Bordeaux
UPDATE VITICULTEURS SET VILLE = 'Bordeaux' WHERE NVT = 150 ;
Augmenter de 10% le degré des Gamay
UPDATE VINS SET DEG = DEG * 1.1 WHERE CRU = 'Gamay' ;