Initiation aux Base de Données

(BD01)

BASE DE DONNEES RELATIONNELLES

Base de données en général (BD)

Une base de données est une **collection de données**, composée d'un ou de plusieurs fichiers, concernant un sujet, **enregistré de <u>manière permanente</u>** sur un ou plusieurs supports accessibles par des programmes.

On accède à cette collection de données via des applications (des programmes informatiques).

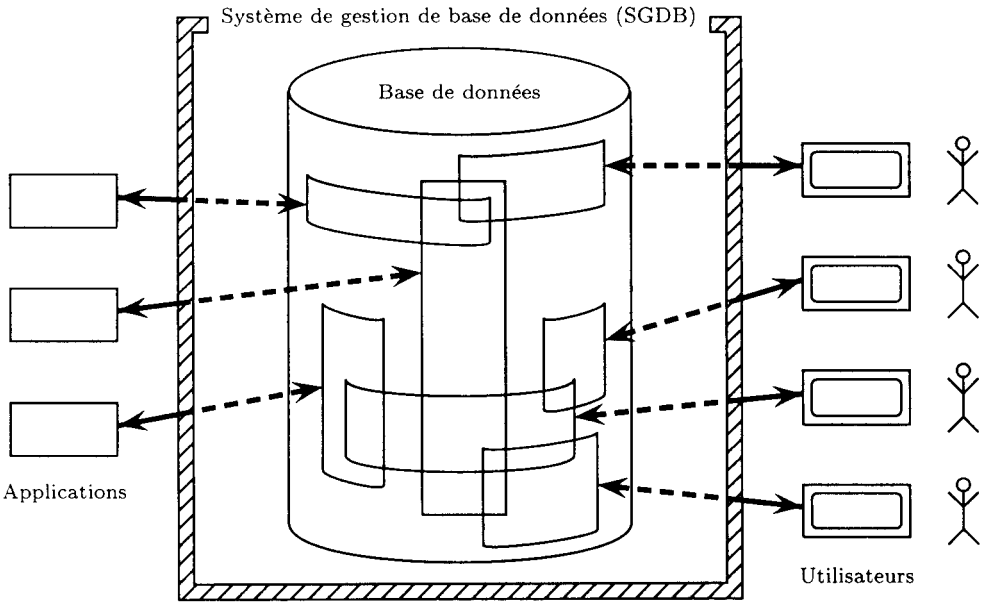


Figure issue de [Date])

Soit la base de données de la médiathèque :
La collection de données concerne les membres, les films, les disques, les employées, les prêts, les prestations du personnel, les jours de congé.
Les opérations qu'on doit être à même de faire sur cette base de données sont en autre

- La consultation sans critères (exemple : voir l'ensemble des films de la médiathèque)
- La consultation avec critères (exemple : voir la liste des films de David Lynch)
- L'ajout de données (exemple : encoder un nouveau membre)
- La suppression de données (exemple : supprimer un film dans la base de données)
- Modifier une donnée (exemple : changer les dates de congé d'un employé)
- Donner des autorisations aux utilisateurs (exemple : seul le chef du personnel peut encoder un congé)

Actuellement, la création de la base de données ainsi que les opérations que l'on y fait se font à l'aide d'un langage standardisé appelé **SQL**

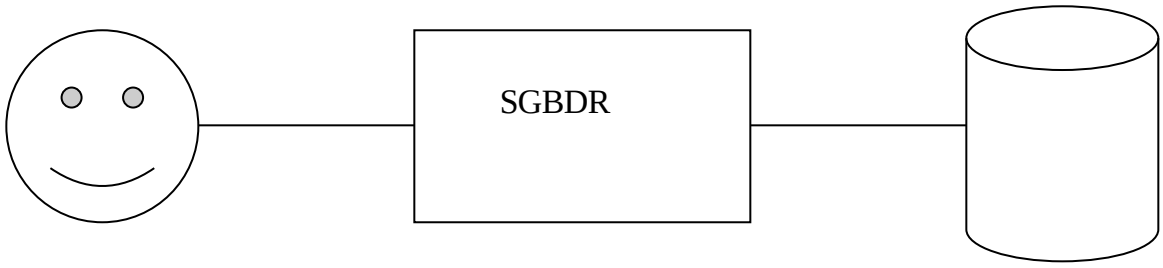
Cycle de vie d'une base de données

étape 1
Avant d'utiliser une base de données, il faut la créer.
Mais avant de la créer, il est nécessaire de définir ce qu'elle doit contenir.
Ce travail est fait par une équipe d'analystes qui doit récolter ces informations.

étape 2
Lorsque cette équipe a déterminé les données que doit contenir la base, elle doit faire un travail de réflexion pour les organiser en les regroupant dans des tables suivant un modèle choisi.
Actuellement le modèle standard est le modèle relationnel.
Exemple d'une table contenant les noms d'auteurs avec leur nationalité

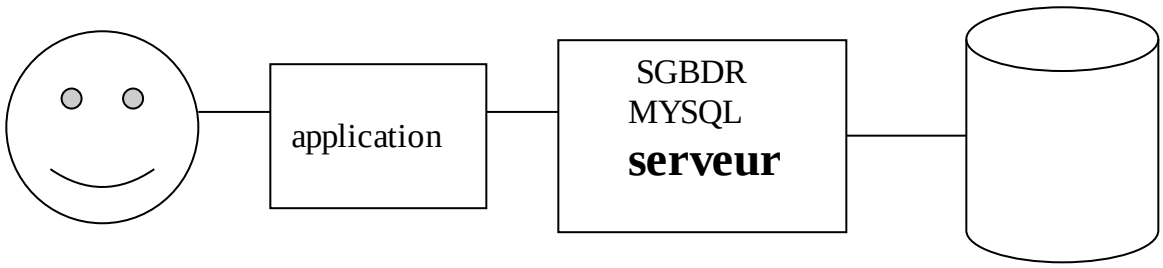
ID_AUTEUR	NOM	NATIONALITE
1	BALZAC	Français
2	MARX	Allemand
3	POUCHKINE	Russe
4	DELMAL	Belge

étape3
Lorsque le modèle de la base de données est établi, il faut créer sa structure informatique, c'est à dire une collection de tables vides.
La création de cette structure se fait par l'intermédiaire de programmes spécialisés que l'on appelle SGBDR.
Un SGBDR est un ensemble de programmes qui sert d'intermédiaire entre un utilisateur et une base de donnée. Dans ce cours, nous utiliserons MYSQL.



Le SGBDR met à notre disposition un ensemble de programmes qui permettent entre autre de créer les tables. Cette création se fait à l'aide d'une sous partie du langage SQL qui s'appelle le **DDL.**

étape4
Il s'agit maintenant de remplir les tables de leur contenu. Ce remplissage peut se faire directement à l'aide de programmes inclus dans le SGBDR ou à l'aide de programmes écrits par nous ou d'autres programmeurs qui s'adressent au SGBDR. Le SGBDR joue le rôle de serveur entre les données enregistrées sur le disque et nos applications.
Ce remplissage se fait à l'aide d'une sous partie du langage SQL qui s'appelle le **DML**



Utilisateur base de données physique

étape 5

Il faut maintenant définir des groupes d'utilisateurs qui vont pouvoir utiliser la base de données.
Cela se fait en donnant à chaque utilisateur des droits.
Ces droits définissent les tables que l'utilisateur peut voir, ainsi que les opérations qu'il peut y faire.
La partie visible par un utilisateur de base de données est appelé schéma.
Une base de données va comprendre plusieurs schémas et chaque utilisateur sera connecté à son schéma.
Des schémas ou des parties de schéma peuvent être commun à plusieurs utilisateurs.
Ces droits se définissent à l'aide d'une sous partie du langage SQL qui s'appelle le **DCL**

Personnes concernées par les bases de données.

L'administrateur des données (A.D)

Il s'agit d'une personne qui connaît parfaitement l'entreprise à informatiser.
Elle doit décider quelles sont les données que l'entreprise doit conserver dans sa base de données pour gérer correctement son activité. Elle doit décider également quelle politique de sécurité on doit appliquer à ces données (qui peut faire quoi).
Il décrit donc les schémas des utilisateurs.
Il n'est pas nécessaire que ce poste incombe à un informaticien.

L'administrateur de la base de données(D.B.A).

Il s'agit du responsable technique de la base de données.

Il doit décrire dans le langage du SGBD les schémas externes nécessaires aux différentes applications
Il exprime les droits d'accès qui sont accordés aux différents utilisateurs.
Il doit assurer une bonne performance globale des temps de réponse pendant l'exploitation de la base de données.
Il met en œuvres les procédures qui assurent un niveau de sécurité de fonctionnement suffisant (procédure de sauvegarde, de sécurité) décidées de commun accord avec le AD .
Ce personnage doit être un informaticien.

Les programmeurs d'applications(P.A)

Il s'agit des personnes qui écrivent des programmes destinés aux utilisateurs de la base de données.
Ces programmeurs n'ont pas les mêmes droits et donc pas les mêmes pouvoirs que le DBA.
Ils programment sur des schémas de la base de données… même s'ils ne s'en aperçoivent pas.
Ces gens sont souvent en contact avec les utilisateurs finaux pour prendre connaissance de leurs souhaits et pour les former à l'utilisation des programmes qu'ils écrivent.
IL est donc souhaitable que ces informaticiens aient des qualités d'écoute et des facilités de communication pour mener à bien leurs tâches.
Les programmeurs d'applications sont en contact avec les utilisateurs finaux et le D.B.A

Les utilisateurs finaux(U.F).
Les utilisateurs finaux sont les personnes qui utilisent la base de données dans le cadre de leurs activités.
Ce sont rarement des informaticiens. Ils manipulent la base de données grâce aux programmes écrits par les programmeurs d'applications.
S'ils connaissent le schéma externe qui leur est dévolu et qu'il ont des connaissances en SQL, ils peuvent alors interroger la base de données en mode commande.

Les programmeurs du gestionnaire de base de données.
Ce sont des programmeurs de haut niveau (souvent des ingénieurs) qui écrivent en équipe le SGBD.

LE MODELE RELATIONNEL

Les deux piliers du modèle relationnel
Le modèle relationnel est un modèle mathématique qui est basé sur la théorie des relations, construite au départ de la théorie des ensembles.
Le modèle relationnel est actuellement le modèle le plus utilisé pour l'organisation d'une base de données.
Son succès provient de plusieurs facteurs:

1) c'est un modèle rigoureux, facile à comprendre, qui repose sur des concepts simples et puissants.
2) ce modèle peut être décrit et manipulé avec le langage SQL qui s'est imposé comme un standard dans le monde informatique.

Ce modèle sert à décrire les schémas des utilisateurs finaux ou des programmeurs d'applications

Le modèle relationnel est associé au nom de CODD.
Codd est un mathématicien chercheur engagé par IBM qui a établi les bases du modèle en 1970.

Il repose sur deux piliers:

1) les **objets** et **contraintes** sur ces objets
   table (relation)
   enregistrement composé de champs (tuple composé d'attributs)
   clé primaire (primary key)
   clé étrangère (foreign key)
2) les **opérateurs** de manipulation (algèbre relationnelle)
   sélection , union, intersection, différence etc …

Les concepts du modèle relationnel.

Table, clé primaire, clé étrangère

Exemple de tables avec clé primaire

Ouvrage (clé : id_ouvrage)

ID_OUVRAGE	TITRE
1	LE CAPITAL
2	EUGENIE GRANDET
3	PERE GORIOT
4	DAME DE PIQUE

Tuple
(enregistrement)

Champ
(attribut)

Clé primaire
(sert à identifier un enregistrement)

La table (terme correct : relation) est un ensemble d'enregistrements (terme correct : tuples).
Chaque ligne est composée de données élémentaires (terme correct : attribut).

Un des attributs de la table doit obligatoirement permettre d'identifier un enregistrement de la table.
Cet attribut est appelé la **clé primaire** de la table.
Comme la clé primaire doit permettre d'identifier une ligne parmi l'ensemble des lignes, son contenu doit être différent pour chaque ligne.

Création en SQL de la table ouvrage:

```
CREATE TABLE ouvrage(
id_ouvrage int( 2 ),
titre char( 20 ),
PRIMARY KEY ( id_ouvrage )
)
```

Insertion en SQL d'un enregistrement dans la table ouvrage

```
insert into ouvrage values (2,'EUGENIE GRANDET')
```

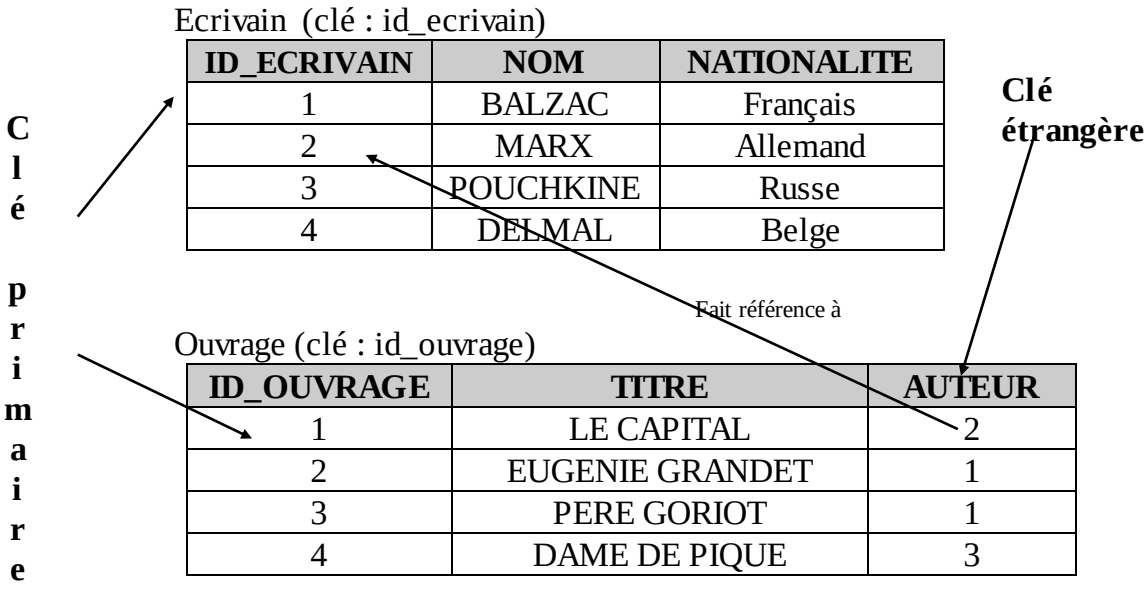
affichage du contenu de de la table ouvrage

```
select * from ouvrage
```

suppression de la table ouvrage

```
drop table ouvrage
```

Exemple de tables avec clé primaire et clé étrangère



Une **clé étrangère** est un attribut de la table dont le contenu fait référence à un enregistrement d'une autre table.

Remarques

1) l'attribut clé étrangère de la table ouvrage ne doit pas obligatoirement avoir le même nom que la clé primaire dans la table auteur; ils doivent cependant pouvoir contenir le même genre de valeur (on dit qu'ils sont de même domaine) .
2) Une clé étrangère ne joue pas un rôle d'identifiant, elle sert à lier des enregistrements situés dans des tables différentes.

Création en SQL de ces tables:

```
CREATE TABLE ecrivain(
id_ecrivain int( 2 ),
nom char( 20 ),
nationalite char( 20 ),
PRIMARY KEY ( id_ecrivain )
            )
```

```
CREATE TABLE ouvrage(
id_ouvrage int( 2 ) ,
titre char( 20 ) ,
auteur int( 2 ) ,
PRIMARY KEY ( id_ouvrage ) ,
FOREIGN KEY ( auteur ) REFERENCES ecrivain( id_ecrivain ) )
```

Une clé primaire multi attributs.

Les deux tables illustrées à la page 7 ne permettent pas de contenir l'information suivante: l'ouvrage intitulé SQL2-SQL3 à été écrit par le belge DELMAL et le français LAMLED.

En effet, la clé étrangère de la table ouvrage ne permet de référencer qu'une seule ligne de la table ECRIVAIN et non deux.

Si une table doit référencer plusieurs lignes d'une autre table, il faut alors avoir une table intermédiaire qui fait la liaison. Dans l'exemple qui suit la table intermédiaire est la table aEcrit

Base de données pouvant avoir plusieurs auteurs par livre.

Ecrivain

ID_ECRIVAIN	NOM	NATIONALITE
1	BALZAC	Français
2	MARX	Allemand
3	POUCHKINE	Russe
4	DELMAL	Belge
5	LAMLED	Français

aEcrit

AUTEUR	OUVRAGE
1	2
2	1
4	5
5	5

Ouvrage

ID_OUVRAGE	TITRE
1	LE CAPITAL
2	EUGENIE GRANDET
3	PERE GORIOT
4	DAME DE PIQUE
5	SQL2-SQL3

Création en SQL de la table aEcrit.

```
CREATE TABLE aEcrit(
auteur int( 2 ) ,
ouvrage int( 2 ) ,
PRIMARY KEY ( auteur, ouvrage ) ,
FOREIGN KEY ( auteur ) REFERENCES ecrivain( id_ecrivain ) ,
FOREIGN KEY ( ouvrage ) REFERENCES ouvrage( id_ouvrage ) )
```

Exercice : TP1

PILIER 2 les **opérateurs** de manipulation (algèbre relationnelle)

Un utilisateur peut n'être intéressé que par une partie du contenu d'une table, ou par de l'information se trouvant dispersées dans plusieurs tables qu'il faudra reconstituer.
Cette extraction ou reconstitution d'information se fait à l'aide d'opérateurs ensemblistes.

Les opérateurs ensemblistes.

Les opérateurs ensemblistes et relationnels forment l'algèbre relationnelle (algèbre de Codd) .
Un SGBDR via le langage SQL supporte les opérateurs définis dans cette algèbre.
On interroge donc les tables de la base de données en raisonnant en termes de cette algèbre.

Les 5 opérateurs de base

- l'union
- la différence
- le produit cartésien
- la sélection (restriction)
- la projection

Les 3 opérateurs supplémentaires

- l'intersection
- la jointure
- la division

Remarque importante :
Une opération ensembliste sur une table **ne modifie pas le contenu de la table**. Elle fabrique une table temporaire pendant le temps de l'opération.

Les opérations en algèbre relationnelle que nous allons découvrir sont internes.
Cela signifie que si on fait une opération sur une table ou entre plusieurs tables, le résultat sera toujours une table, même si ce résultat ne possède qu'un seul champ.
Cette propriété permet de remplacer une table par **une opération entre tables** et donc d'enchaîner ainsi plusieurs opérations dans une même expression.
On peut donc écrire des **expressions emboîtées**, c'est à dire des expressions dans lesquelles les opérandes sont elles même des expressions (ceci sera vu dans le chapitre sous requêtes).

Description des 7 opérateurs de l'algèbre relationnelle.

Table clientele (clé primaire : id_client)

ID_CLIENT	NOM	VILLE
CL01	PAUL	BRUXELLES
CL02	PIERRE	BRUXELLES
CL03	JACQUES	LOUVAIN
CL04	HENRY	ANVERS
CL05	JACQUES	GAND

Table pret_hypothecaire (clé primaire :id_prêt_hypothecaire)

ID_PRET_HYPOTHECAIRE	CLIENT	AGENCE	MONTANT
HY01	CL01	AG01	15000
HY02	CL01	AG03	20000
HY03	CL02	AG01	40000
HY04	CL03	AG01	20000
HY05	CL03	AG03	20000
HY06	CL05	AG01	12000
HY07	CL01	AG02	15000

Table agence (clé primaire : id_agence)

ID_AGENCE	VILLE
AG01	BRUXELLES
AG02	LIEGE
AG03	ANVERS

Table pret_voiture (clé primaire : id_prêt_voiture)

ID_PRET_VOITURE	CLIENT	AGENCE	MONTANT
PV01	CL04	AG01	8500
PV02	CL03	AG03	7800
PV03	CL04	AG03	9000
PV04	CL02	AG03	8500
PV05	CL03	AG03	7000

La selection ou restriction

La sélection est une opération qui s'applique à une seule table.
La restriction consiste à ne retenir que les lignes qui répondent au critère exprimé.

Exemple : On souhaite connaître les clients qui habitent BRUXELLES

- Notation logique clientele **where** ville='BRUXELES'
- Notation SQL select * from clientele where ville='BRUXELLES'

La projection

La projection est une opération qui s'applique à une seule table.
La table sur laquelle la projection s'applique peut être le résultat d'opérations entre tables.
Elle consiste à ne retenir de la table que les colonnes désignées par c1,c2,c3,etc...
Les lignes doubles ainsi créés n'apparaissent qu'une seule fois dans la table solution

Exemple: On souhaite connaître les identifiants des clients qui ont un prêt hypothécaire.

- Notation logique pret_hypothecaire[client]
- Notation SQL select distinct client from pret_hypothecaire

Union

Cette opération ne s'applique qu'à des tables qui sont compatibles.
Les tables doivent avoir le même nombre de colonnes, (on dit même degré), et les colonnes sont définies sur des domaines compatibles.
L'union des 2 tables T1 et T2 donnera l'ensemble des lignes de T1 et l'ensemble des lignes de T2. Les lignes identiques ne sont conservées qu'une seule fois.

Exemple: *Si on veut connaître l'ensemble des prêts hypothécaires et des prêts voitures*:

- Notation logique

```
pret_hypothecaire  Union  pret_voiture
```

- Notation SQL

```
select *  from pret_hypothecaire
union
select * from pret_voiture
```

ID_PRET_VOITURE	CLIENT	AGENCE	MONTANT
HY01	CL01	AG01	15000
HY02	CL01	AG03	20000
HY03	CL02	AG01	40000
HY04	CL03	AG01	20000
HY05	CL03	AG03	20000
HY06	CL05	AG01	12000
HY07	CL01	AG02	15000
PV01	CL04	AG01	8500
PV02	CL03	AG03	7800
PV03	CL04	AG03	9000
PV04	CL02	AG03	8500
PV05	CL03	AG03	7000

Remarque : on peut rendre des morceaux de tables compatibles à l'aide de projection.
Exemple : on souhaite connaître toutes les villes, client et agence confondus.

- Notation logique

```
Clientele[ville] union agence[ville]
```

- Notation SQL

```
select ville   from clientele
 union
select ville  from agence
```

VILLE
ANVERS
BRUXELLES
GAND
LIEGE
LOUVAIN

La différence

Cette opération ne s'applique qu'aux tables compatibles.
La différence des tables T1 et T2 donnera l'ensemble des lignes de T1 non présentes dans T2.
Exemple: On souhaite connaître les identifiants de clients qui ont un prêt hypothécaire sans avoir de prêt de voiture.

- Notation logique

```
pret_hypothecaire[client]   minus pret_voiture[client]
```

- Notation SQL

opérateur non implémenté en MYSQL (voir sous requêtes plus loin dans le cours)

L'intersection

Les tables ou morceaux de tables doivent être compatibles
L'intersection de deux tables consiste en une table ne contenant que les lignes se trouvant à la fois dans les deux tables.

Exemple: on souhaite connaître les identifiants de clients qui ont un prêt hypothécaire et un prêt voiture.

- Notation logique | pret_hypothecaire[client] intersect pret_voiture[client] |
- Notation SQL
  opérateur non implémenté en MYSQL (voir sous requêtes plus loin dans le cours)

CLIENT
CL02
CL03

Le produit cartésien:

Le produit cartésien fait intervenir au moins deux tables (éventuellement deux fois la même) . Les tables ne doivent pas être compatibles.
Le produit cartésien de deux tables consiste en une table constituées de **toutes les concaténations possibles** des lignes de la première table avec les lignes de la deuxième table.
Le produit cartésien est rarement utilisé tout seul, mais très souvent comme étape indispensable lorsque l'on doit rassembler des informations dispersées dans des tables différentes.

Exemple : le produit cartésien de clientele et pret_hypothecaire

- Notation logique | Clientele **times** pret_hypothecaire |
- Notation SQL select * from clientele, pret_hypothecaire

ID_CLIENT	NOM	VILLE	ID_PRET_HYPOTHECAIRE	CLIENT	AGENCE	MONTANT
CL01	PAUL	BRUXELLES	HY01	CL01	AG01	15000
CL01	PAUL	BRUXELLES	HY02	CL01	AG03	20000
CL01	PAUL	BRUXELLES	HY03	CL02	AG01	40000
CL01	PAUL	BRUXELLES	HY04	CL03	AG01	20000
CL01	PAUL	BRUXELLES	HY05	CL03	AG03	20000
CL01	PAUL	BRUXELLES	HY06	CL05	AG01	12000
CL01	PAUL	BRUXELLES	HY07	CL01	AG02	15000
CL02	PIERRE	BRUXELLES	HY01	CL01	AG01	15000
CL02	PIERRE	BRUXELLES	HY02	CL01	AG03	20000
CL02	PIERRE	BRUXELLES	HY03	CL02	AG01	40000
CL02	PIERRE	BRUXELLES	HY04	CL03	AG01	20000
CL02	PIERRE	BRUXELLES	HY05	CL03	AG03	20000
CL02	PIERRE	BRUXELLES	HY06	CL05	AG01	12000
CL02	PIERRE	BRUXELLES	HY07	CL01	AG02	15000
CL03	JACQUES	LOUVAIN	HY01	CL01	AG01	15000
CL03	JACQUES	LOUVAIN	HY02	CL01	AG03	20000
CL03	JACQUES	LOUVAIN	HY03	CL02	AG01	40000
CL03	JACQUES	LOUVAIN	HY04	CL03	AG01	20000
CL03	JACQUES	LOUVAIN	HY05	CL03	AG03	20000
CL03	JACQUES	LOUVAIN	HY06	CL05	AG01	12000
CL03	JACQUES	LOUVAIN	HY07	CL01	AG02	15000
CL04	HENRY	ANVERS	HY01	CL01	AG01	15000
CL04	HENRY	ANVERS	HY02	CL01	AG03	20000
CL04	HENRY	ANVERS	HY03	CL02	AG01	40000
CL04	HENRY	ANVERS	HY04	CL03	AG01	20000
CL04	HENRY	ANVERS	HY05	CL03	AG03	20000
CL04	HENRY	ANVERS	HY06	CL05	AG01	12000
CL04	HENRY	ANVERS	HY07	CL01	AG02	15000
CL05	JACQUES	GAND	HY01	CL01	AG01	15000
CL05	JACQUES	GAND	HY02	CL01	AG03	20000
CL05	JACQUES	GAND	HY03	CL02	AG01	40000
CL05	JACQUES	GAND	HY04	CL03	AG01	20000
CL05	JACQUES	GAND	HY05	CL03	AG03	20000
CL05	JACQUES	GAND	HY06	CL05	AG01	12000
CL05	JACQUES	GAND	HY07	CL01	AG02	15000

La table résultante d'un produit cartésien est toujours composée d'un grand nombre de lignes:
Si T1 a 200 lignes et que T2 a 300 lignes, T1 x T2 aura 60000 lignes.

Cette opérations appliquée seule ne présente que rarement d'intérêt.
Elle est cependant une étape **indispensable** pour regrouper des informations qui ont étés éparpillées dans des tables différentes.

La Jointure

La jointure consiste à regrouper dans une même table des informations situées dans des tables ayant une information commune.

Il existe en fait différentes natures de jointures. Retenez que la plupart des jointures entre tables s'effectuent en imposant l'égalité des valeurs d'une colonne d'une table à une colonne d'une autre table. On parle alors d' équi-jointure. Mais on trouve aussi des jointures d'une table sur elle-même. On parle alors d'auto-jointure.

a) Equi-jointure

Exemple : On souhaite connaître les numéros de prêts hypothécaires associés aux noms des clients concernés.

ID_PRET_HYPOTHECAIRE	NOM
HY01	PAUL
HY02	PAUL
HY07	PAUL
HY03	PIERRE
HY04	JACQUES
HY05	JACQUES
HY06	JACQUES

Ces informations se trouvent dans deux tables séparées qui ont une information commune (le champ client de la table prêt_hypothecaire a la même information que le champ id_client de la table clientele).
La colonne commune est clé primaire dans une table (le champ id_client dans la table Clientele) et clé étrangère faisant référence à cette clé primaire dans l'autre table (le champ client dans la table prêt_hypothecaire).

Remarque: c'est l'information qui doit être commune et non pas le nom du champ.

L'idée consiste à

1) appliquer un produit cartésien entre les tables clientele et produit hypothecaire pour fabriquer une seule table
2) on ne conserver de cette table que les lignes ayant la même information commune.

Explication
Le produit cartésien « fabrique » un grand nombre de lignes n'ayant pas de sens « **lignes Ko** ».

CL01	PAUL	BRUXELLES	HY01	**CL01**	AG01	15000 → ligne ok
CL01	PAUL	BRUXELLES	HY02	**CL01**	AG03	20000 → ligne ok
CL01	PAUL	BRUXELLES	HY03	**CL02**	AG01	40000 → **ligne KO**
CL01	PAUL	BRUXELLES	HY04	**CL03**	AG01	20000 → **ligne KO**
CL01	PAUL	BRUXELLES	HY05	**CL03**	AG03	20000 → **ligne KO**

Etc …
On applique ensuite une sélection sur base du champ commun (on ne garde que les lignes ok)

- Notation logique
  pret_hypothecaire **times** clientele

where pret_hypothecaire.client =clientele.id_client

- Notation SQL (sans l'opérateur join)

  Select * from pret_hypothecaire , clientele
  where client = id_client

- Notation SQL (avec opérateur join)

  Select * from pret_hypothecaire join clientele
  on client = id_client

id_pret_hypothecaire	client	agence	montant	id_client	nom	ville
HY07	CL01	AG02	15000	CL01	PAUL	BRUXELLES
HY06	CL05	AG01	12000	CL05	JACQUES	GAND
HY05	CL03	AG03	20000	CL03	JACQUES	LOUVAIN
HY04	CL03	AG01	20000	CL03	JACQUES	LOUVAIN
HY03	CL02	AG01	40000	CL02	PIERRE	BRUXELLES
HY02	CL01	AG03	20000	CL01	PAUL	BRUXELLES
HY01	CL01	AG01	15000	CL01	PAUL	BRUXELLES

Remarque On peut combiner l'expression avec d'autres critères de sélection en plus du critère de jointure.

Exemple: On souhaite connaître le numéro de prêts hypothécaire et le nom du client **lorsque le prêt est supérieur à 16000.**

ID_PRET_HYPOTHECAIRE	NOM
HY02	PAUL
HY03	PIERRE
HY04	JACQUES
HY05	JACQUES

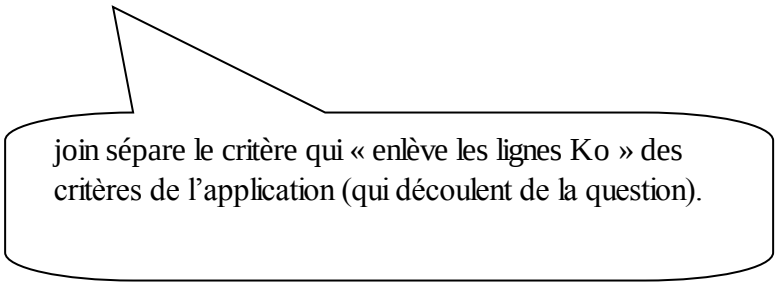
- Notation logique

  (pret_hypothecaire times clientele
  where pret_hypothecaire.client =clientele.id_client
  and montant >16000)[id_pret_hypothecaire, nom]

  - Notation(s) SQL

    Select id_pret_hypothecaire,nom from pret_hypothecaire , clientele
    where client = id_client **and** montant > 16000

```
Select id_pret_hypothecaire,nom from
pret_hypothecaire join  clientele on   client = id_client where  montant >16000
```



Remarque :

1) On peut toujours qualifier le nom d'un champ en le faisant précéder du nom de sa table et d'un point : clientele.id_client
   Cette manière de procéder est absolument nécessaire lorsque des colonnes issues de tables différentes portent le même nom.

2) Il arrive que l'on doive procéder à des jointures externes, c'est-à-dire joindre une table à une autre, même si la valeur de liaison est absente dans une table ou l'autre. Enfin, dans quelques cas, on peut procéder à des jointures hétérogènes, c'est-à-dire que l'on remplace le critère d'égalité par un critère d'inégalité ou de différence.

b) Auto-jointure:

Il s'agit de la jointure d'une table avec elle –même.
Comme les tables ont les mêmes noms, on doit les **renommer** pour les distinguer.

Exemple : on veut connaître les paires de clients qui habitent dans la même ville.

L'idée consiste à fabriquer toutes les combinaisons possibles de deux personnes
et à ne garder parmi ces combinaisons que les lignes ou les noms sont différents et les adresses les mêmes.

- Notation logique

```
(clientele T1 join clientele T2
 on T1.ville=T2.ville
 where T1.id_client  > T2.id_client
 )[T1.nom,T2.nom]
```

remarque: la condition T1.id_client >T2.id_client à la place de T1id_client <> t2.id_client évite d'avoir des doublons dans la solution (paul, pierre et pierre,paul)

- Notation SQL

```
Select T1.nom,T2.nom from
clientele T1 join clientele T2
on  T1.ville = T2.ville
where t1.id_client > t2.id_client
```

Exercices

Donner en pseudo-code la succession d'opérateurs à appliquer pour résoudre ces requêtes :
Ecrivez ensuite le SQL correspondant sauf pour les opérateurs non implémentés en MYSQL.
(matière vue dans les TP ultérieurs)

1) On souhaite afficher les informations de la table clientele des clients qui habitent Bruxelles

2) On souhaite connaître les identifiants des clients qui ont souscrit un prêt hypothécaire

3) On souhaite les enregistrements concernant les prêts hypothécaires souscrit à l'agence AG01

4) On souhaite les **identifiants des clients** qui ont souscrit un prêt voiture d'un montant >8000

5) On souhaite **les identifiants et les noms** des clients qui ont souscrit un prêt voiture d'un montant >8000

6) On souhaite connaître les identifiants des clients ayant des prêts hypothécaires ou des prêts voiture

7) On souhaite connaître toutes les villes des tables clientele et agence.

8) On souhaite connaître les identifiants des clients qui ont un prêt hypothécaire sans avoir un prêt voiture

9) On souhaite les identifiants des clients qui ont un prêt hypothécaire et un prêt voiture

10) On souhaite connaître les identifiants des clients qui ont souscrit un prêt hypothécaire dont le montant est > 15000

11) On souhaite connaître les numéros de prêts hypothécaires associés aux noms des clients concernés

12) On souhaite les identifiants des clients qui ont un prêt hypothécaire ou un prêt voiture, mais pas les deux

13) On souhaite afficher les paires de clients qui ont conclu un prêt hypothécaire pour le même montant.

Exercices (repris de date[98]).

SPJ (clé primaire : id_s,id_p,id_j)

ID_S	ID_P	ID_J	QTY	DATE_DERNIERE_LIVRAISON
S1	P1	J1	200	5/10/2001
S1	P1	J4	700	10/05/2001
S2	P3	J1	400	20/05/2001
S2	P3	J2	200	30/07/2000
S2	P3	J3	200	10/05/2001
S2	P3	J4	500	3/10/2001
S2	P3	J5	600	20/09/2001
S2	P3	J6	400	12/05/2000
S2	P3	J7	800	23/08/2001
S2	P5	J2	100	23/06/2000
S3	P3	J1	200	7/07/2001
S3	P4	J2	500	18/05/2001
S4	P6	J3	300	10/05/2001
S4	P6	J7	300	16/09/2001
S5	P1	J4	100	18/03/2000
S5	P2	J2	200	10/11/2001
S5	P2	J4	100	17/04/2001
S5	P3	J4	200	19/05/2001
S5	P4	J4	800	10/05/2001
S5	P5	J4	400	16/12/2001
S5	P5	J5	500	8/02/2001
S5	P5	J7	100	25/06/2001
S5	P6	J2	200	9/02/2001
S5	P6	J4	500	10/10/2001

S (clé primaire : id_s)

ID_S	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P (clé primaire : id_p)

ID_P	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

J (clé primaire id_j)

ID_J	JNAME	CITY
J1	Sorter	Paris
J2	Display	Rome
J3	OCR	Athens
J4	Console	Athens
J5	RAID	London
J6	EDS	Oslo
J7	Tape	London

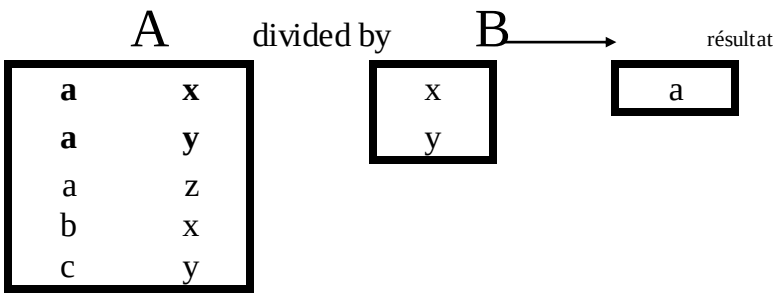
Exercice 1.
Cet exercice se base sur la description des tables ci dessus
Expliquez à l'aide de la notation logique comment :

1. Obtenir les noms des fournisseurs de la pièce P2
2. Obtenir les noms des fournisseurs qui fournissent au moins une pièce rouge

Exercice 2

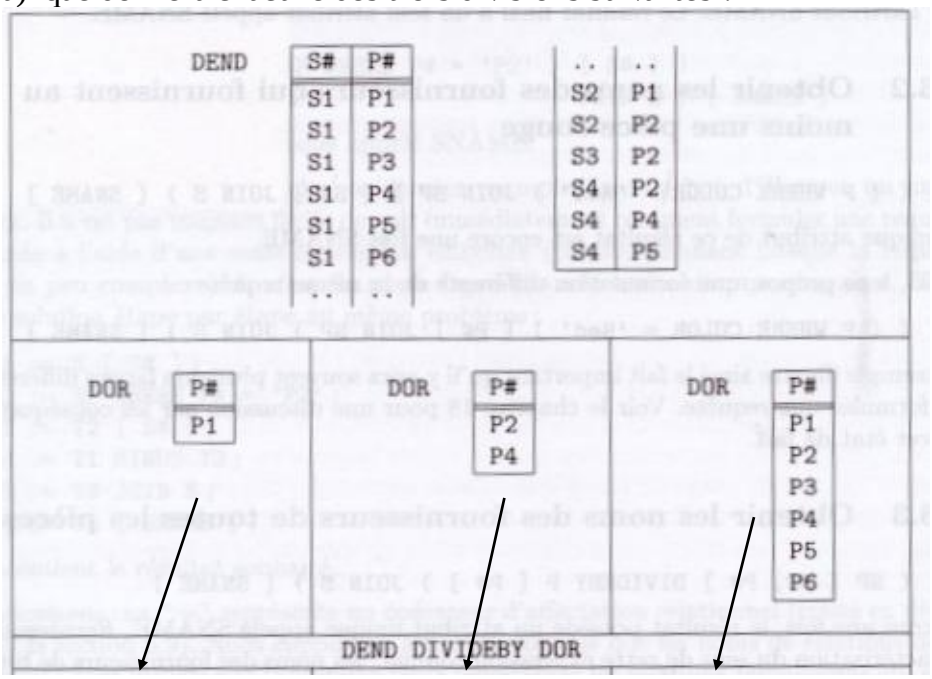
On vous demande d'écrire en notation logique la manière d'obtenir tous les couples de numéros de fournisseurs tels que les deux fournisseurs concernés soient situés dans la même ville.

Exercice 3
La norme SQL2 inclu un opérateur ensembliste appelé DIVISION (DIVIDED BY).
Voici ce que fait cet opérateur :



La condition d'utilisation est :
Toutes les colonnes de la table B doivent être dans la table A
Le résultats est une table comprenant les attributs de A ne se trouvant pas dans B .
Les tuples de cette table resultat sont les lignes issues de A qui sont associées à **toutes les valeurs** comprises dans B.

a) que donnera chacune des trois divisions suivantes :



résultat résultat résultat

Exercice 4

Exprimez les requêtes en notation logique pour :

1. Obtenir les triplets id_s,id_p,id_j tel que les fournisseurs, les pièces et les projets aient la même ville.
   a) les triplets ne doivent pas spécialement appartenir à une livraison existante.
   b) les triplets doivent appartenir à une livraison existante.

2. Obtenir les numéros de pièces fournies par un fournisseur de Londres à un projet de Londres.
3. Obtenir les numéros de projets exclusivement fournis par le fournisseur S1.
4. Obtenir les numéros de fournisseurs qui ont fournit au moins une même pièce à tous les projets (on ne doit pas connaître la ou les pièces concernées).
5. Obtenir toutes les villes concernant un fournisseur, une pièce ou un projet est situé.
6. Obtenir les numéros de pièce qui sont fournies soit par un fournisseur de Londres, soit à un projet de Londres.
7. Obtenir les couples de numéros fournisseur/numéro de pièces tels que le fournisseur ne fournit pas la pièce correspondante.
8. Obtenir les numéros de projets utilisant au moins une pièce disponible chez le fournisseur S1. (considérons arbitrairement qu'une pièce disponible chez un fournisseur est une pièce que le fournisseur a déjà livrée)
9. Obtenir le nom des fournisseurs de toutes les pièces (penser à diviser)
10. Obtenir les numéros de fournisseurs d'au moins toutes les pièces fournies par S2. (ils peuvent donc fournir plus de pièces)

SPJ (clé primaire id_s,id_p,id_j)

ID_S	ID_P	ID_J	QTY	DATE_DERNIERE_LIVRAISON
S1	P1	J1	200	5/10/2001
S1	P1	J4	700	10/05/2001
S2	P3	J1	400	20/05/2001
S2	P3	J2	200	30/07/2000
S2	P3	J3	200	10/05/2001
S2	P3	J4	500	3/10/2001
S2	P3	J5	600	20/09/2001
S2	P3	J6	400	12/05/2000
S2	P3	J7	800	23/08/2001
S2	P5	J2	100	23/06/2000
S3	P3	J1	200	7/07/2001
S3	P4	J2	500	18/05/2001
S4	P6	J3	300	10/05/2001
S4	P6	J7	300	16/09/2001
S5	P1	J4	100	18/03/2000
S5	P2	J2	200	10/11/2001
S5	P2	J4	100	17/04/2001
S5	P3	J4	200	19/05/2001
S5	P4	J4	800	10/05/2001
S5	P5	J4	400	16/12/2001
S5	P5	J5	500	8/02/2001
S5	P5	J7	100	25/06/2001
S5	P6	J2	200	9/02/2001
S5	P6	J4	500	10/10/2001

S (clé primaire : id_s)

ID_S	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P (clé primaire : id_p)

ID_P	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

J (clé primaire id_j)

ID_J	JNAME	CITY
J1	Sorter	Paris
J2	Display	Rome
J3	OCR	Athens
J4	Console	Athens
J5	RAID	London
J6	EDS	Oslo
J7	Tape	London

NOTIONS ABORDEES DANS LES TP

- Tp1 création de tables et insertion de données
- Tp2 exécution d'un script + requêtes simples
- Tp3 requêtes simples
- Tp4 requêtes simples
- Tp5 fonctions d'agrégations , group by
- Tp6 clause having
- Tp7 les dates et exercices récapitulatifs group by, having
- Tp8 les sous requêtes avec exists
- Tp9 les sous requêtes corrélées et non corrélées liaison naturelles, par in, par all, par any
- Tp10 variation de raisonnements pour une même requête
- Tp11 sous requêtes mélangées
- Tp12 les opérateurs non implémentés différence, intersection

BIBLIOGRAPHIE

- CRIS J.DATE Introduction aux bases de données « 8ème édition ». Vuibert . Décembre 2004
- JEFFREY D.ULLMAN A first course in database systems. International edition. 1997
- JOE CELKO SQL avancé. International Thomson publication. Janvier 1997
- PIERRE DELMAL SQL2-SQL3 3ème édition De Boeck Université. 2001

Table des matières