Base de données

S. Faisan

Cours 1A

Plan

Base de données

définitions

- Une base de données (BD) est une collection de données reliées entre elles nécessaires au fonctionnement d'une entreprise
- L'ensemble des données est
  - structuré
  - cohérent (signification)
  - partagé (utilisé par plusieurs utilisateurs)
  - défini pour les besoins d'une application
- Intérêt d'une base de donnée si :
  - natures diverses des données
  - différents liens entre les données

Exemple : gestion d'une chaîne de vidéothèques

définitions

- natures diverses des données : client, acteur, magasin, film...
- Différentes associations entre les données :
  - un acteur joue dans un film,
  - un client loue un film dans un magasin,
  - un magasin met à disposition des films
- Différentes opérations (question) :
  - service location : est-ce que le solde du client est suffisant pour louer le film ?
  - espace client : dans quels films a joué cet acteur ?
- Différentes opérations (mise à jour) :
  - service location : ajouter une location (client X loue film Y)
  - espace client : augmenter le solde d'un client

Système de Gestion de Bases de Données (SGBD)

définitions

UN SGBD est un ensemble de programmes (software) permettant de :

- définir la BD (spécification des types de données, de la structure...)
- construire la BD (stockage des données sur le disque)
- manipuler les données : récupérer les données et les mettre à jour
- maintenir la BD dans un état cohérent : gérer les données (fiabilité, concurrence, ...) et les utilisateurs (droit)

Les 7 caractéristiques d'un SGBD

1. La persistance
   - Une donnée est persistante si elle survit à la fin du programme qui l'a créée (écriture sur disque)
2. Gestion du disque et de la mémoire
   - Les données sont trop volumineuses pour être stockées en mémoire centrale. Il faut mettre en place des techniques pour minimiser les accès disques (optimisation des requêtes, création d'index,...)
3. Partage des données
   - Chacun doit avoir l'impression qu'il est seul mais les données doivent rester cohérentes.

Les 7 caractéristiques d'un SGBD

4. Fiabilité
   - Capacité à restaurer la base de données dans un état connu comme correct :
     - solution matérielle : algorithmes RAID
     - solution logicielle : journalisation, procédure de reprise après panne
5. Sécurité
   - Identification et droit des utilisateurs (autorisations : lire, écrire, ...)
6. Indépendance Logique / Physique
   - Organisation physique de la BD doit (peut) être inconnue des programmeurs d'application
   - Organisation des données doit être modifiable sans reprendre les applications qui les utilisent

Les 7 caractéristiques d'un SGBD

7. Langage de requêtes. Les requêtes doivent être :
   - simples à écrire
   - déclaratives (dire ce que l'on veut sans préciser la façon dont s'est obtenu)
   - optimisées automatiquement par le processeur de requêtes

Cycle de vie d'une base de données

Phase 1 : conception

- Détermination d'une base de connaissance suite à l'analyse de l'entreprise.
  - une information peu structurée qui contient des informations sur les données, les contraintes, les règles de gestion (une personne peut louer au plus 2 films) et de transformation des données (on enlève du solde au client suite à une location).
- Détermination du schéma conceptuel : abstraction des informations de la base de connaissance s'appuyant sur un modèle conceptuel indépendant de la BD choisie (on utilisera le modèle Entité-Association)

Avantages de la modélisation conceptuelle

- Attention portée sur les applications
- Indépendante des technologies (portabilité, longévité)
- Orientée utilisateur
  - Support d'interfaces visuelles
  - Compréhensibilité
  - Permet la collaboration et la validation par les utilisateurs
- La qualité de la conception de la BD est un facteur critique de réussite

Cycle de vie d'une base de données

Phase 2 : Implantation

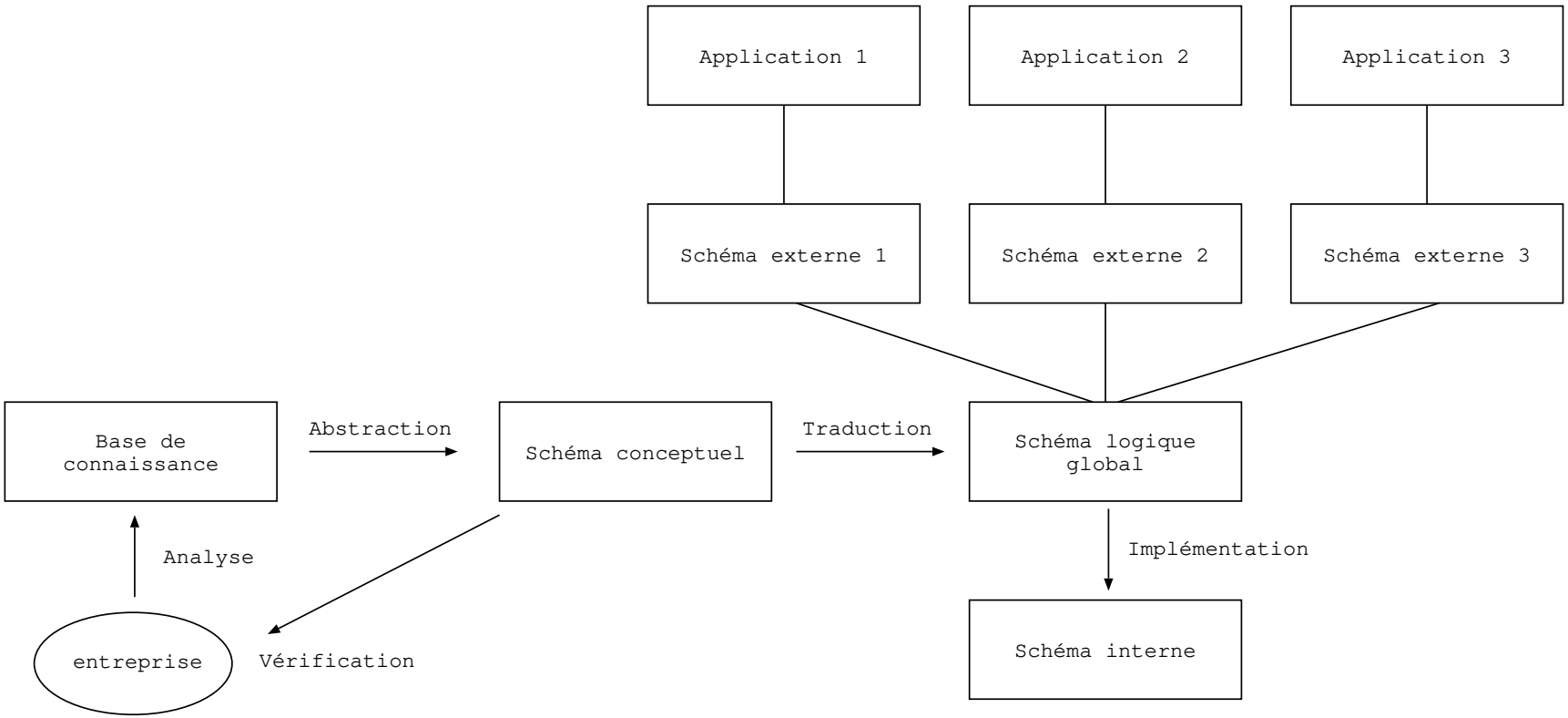
- Transmission de la description des données au SGBD choisi :
  - Traduction du schéma conceptuel selon le modèle logique utilisé par une BD choisie (modèle hiérarchique, réseau, relationnel...).
  - Utilisation un langage de description des données (LDD) spécifique au SGBD choisi.
  - Traduction du schéma logique en schéma interne. Il comprend les informations nécessaires pour accéder aux données (index, chemins, codage...)

Cycle de vie d'une base de données

Phase 3 : Utilisation et phase 4 : Maintenance

- Utilisation
  - Définition des schémas externes (les schémas externes sont des adaptations du schéma logique global aux différents points de vue de l'entreprise)
  - Utilisation d'un langage de manipulation des données (LMD)
    - Requêtes d'interrogation
    - Requêtes de mise à jour (ajout, suppression, modification)
    - Insertion des données
- Maintenance :
  - Corrective (supprimer les données erronées, redondantes)
  - Evolutive (évolution de la structure de données)

Le processus complet



Plan

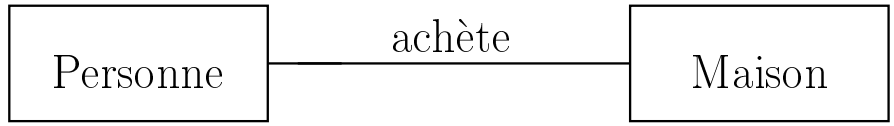
Le modèle Entité-Association

- Représentation à l'aide de la trilogie : objet, lien propriété :
  - objet – entité (l'entité est la représentation d'un objet du monde réel ayant une existence propre)
  - lien – association (une association représente un lien non orienté entre plusieurs entités)
  - propriété – attribut, ou caractéristique
- Abstraction des particularités permet de passer des entités aux classes d'entité (CE), et des associations aux classes d'association (CA).
  - Une CE représente un ensemble d'entités ayant la même sémantique et décrites par les mêmes caractéristiques
  - Une CA représente un ensemble d'associations ayant la même sémantique et décrites par les mêmes caratéristiques.

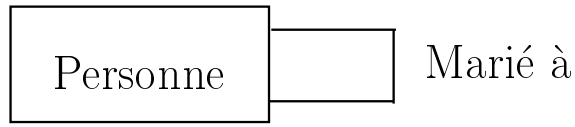
Association et Entité

Dans une association, chaque entité joue un rôle

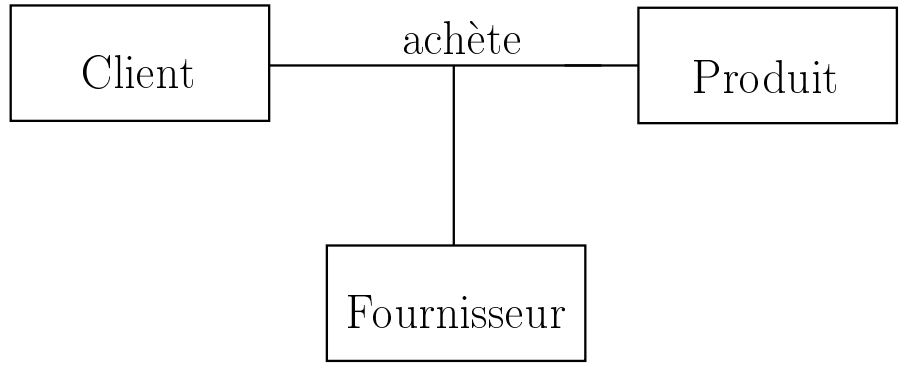
association binaire : deux rôles



association cyclique : un ou deux rôles

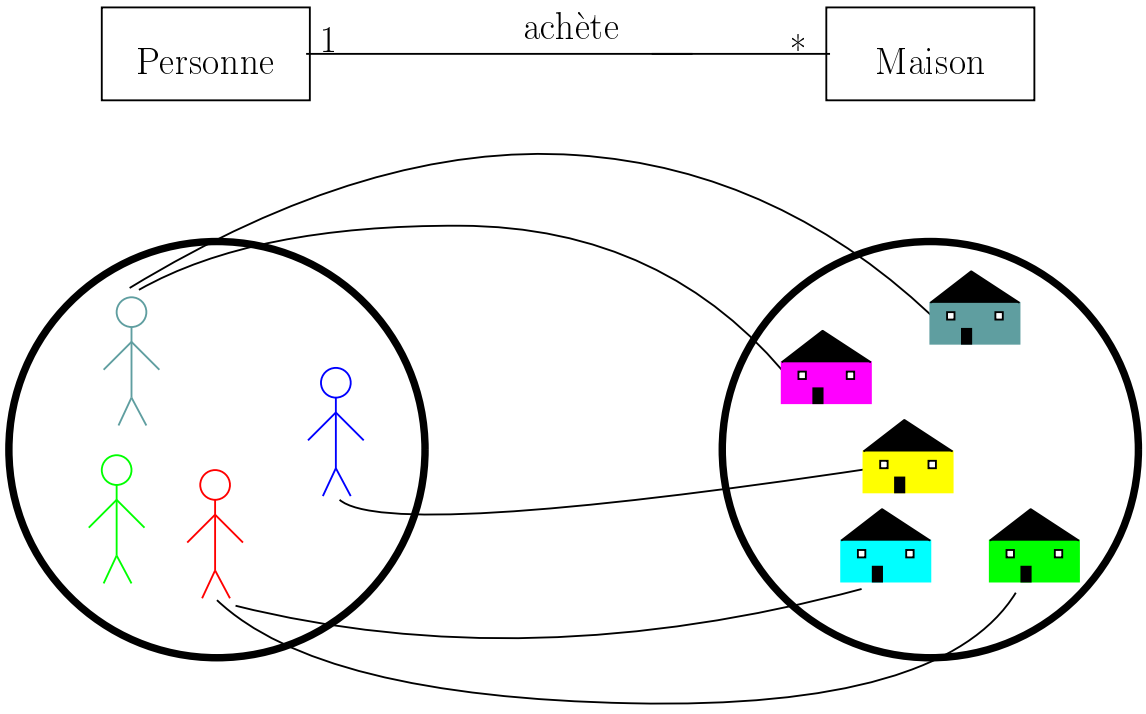


association ternaire : trois rôles



Cardinalité des rôles

1	un seul
*	de 0 à N
0..1	de 0 à 1
1..*	de 1 à N
n..m	de n à m



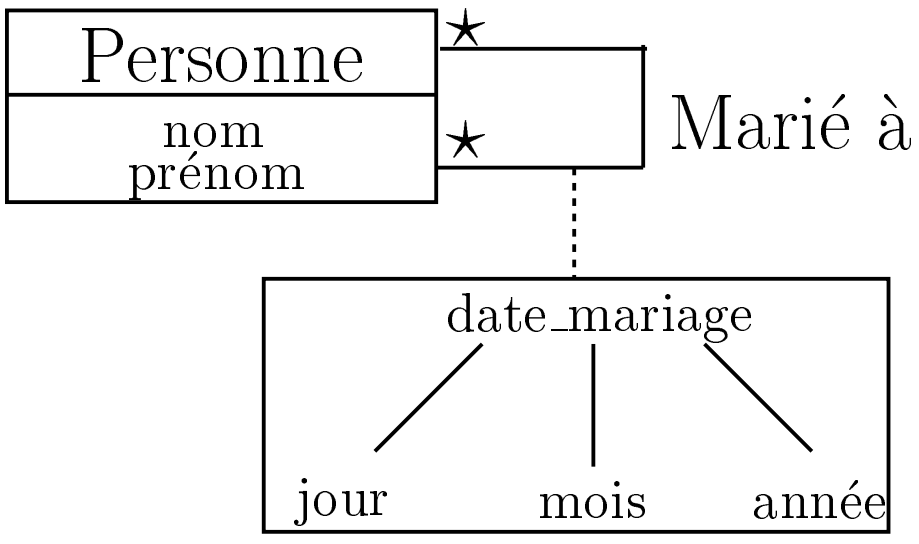
Attributs

Les attributs décrivent

- une classe d'entité
- une classe d'association
- un autre attribut

Les attributs peuvent être

- simples ou complexes
- mono ou multi-valués
- obligatoires ou facultatifs



Exercice 1

Opérateur téléphonique

Chaque client est enregistré dans la BD avec son nom, son prénom et son adresse. Un client peut posséder plusieurs numéros de téléphone. Un historique des appels est stocké en enregistrant pour chaque appel le numéro appelant, le numéro appelé, la date, l'heure ainsi que la durée.
Dans le cadre du programme numéro préféré, un client peut associer un numéro de téléphone à un autre numéro afin d'obtenir un tarif plus avantageux. L'historique de l'ensemble des associations est conservé dans la base.

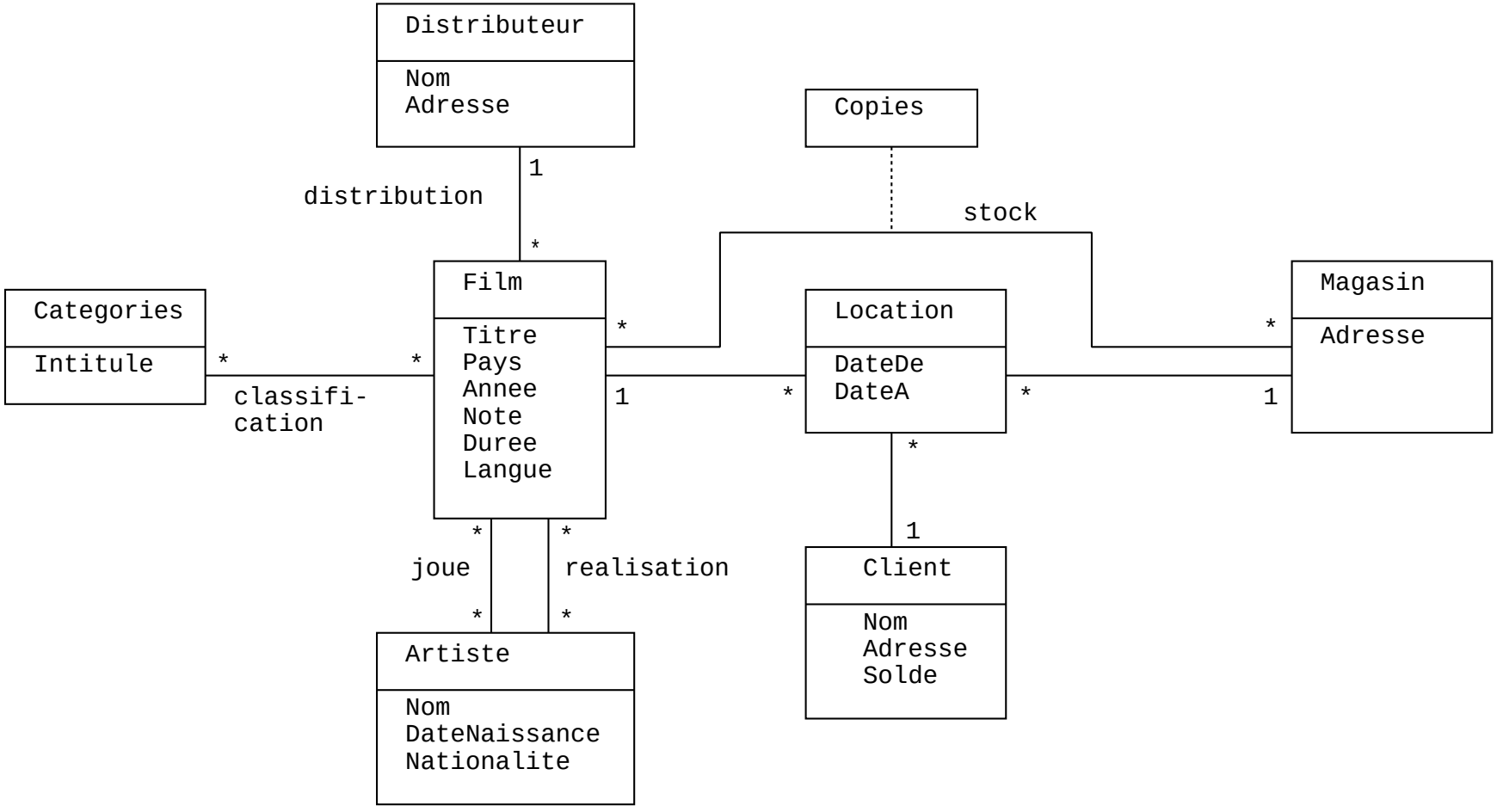
Exercice 2

Traduction d'une association ternaire en associations binaires

- « Fausse » association ternaire : un client loue un film dans un magasin (on veut l'historique, on suppose que le client est obligé de toujours aller dans le même magasin).
- « Vrai » association ternaire : Un client loue un film dans un magasin (on veut l'historique, le client peut aller dans le magasin qu'il désire).

Exercice 3 : Comprendre un schéma

Quid des contraintes et des règles de transformation des données ?

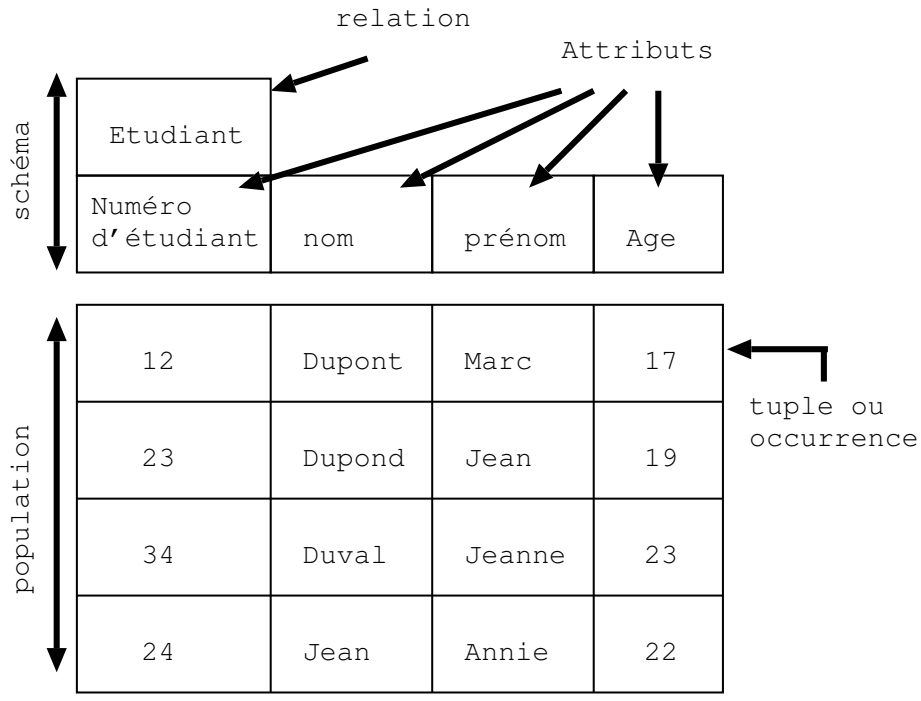


Plan

Relation et attribut

- Une relation est une structure plate régulière (matrice)
- Une BD relationnelle est un ensemble de relations

- Le schéma d'une relation est un ensemble d'attributs simples et monovalués
- Un attribut peut ne pas être valué pour un tuple. Il a la valeur `NULL`.



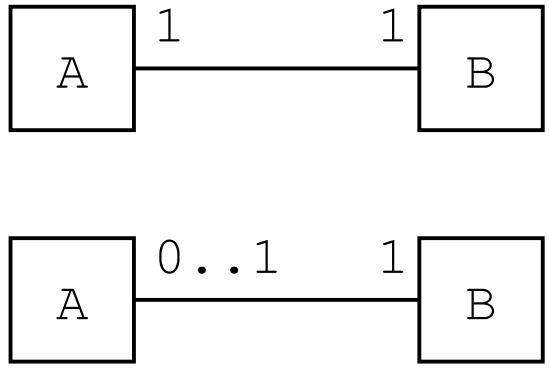
Règles d'identification

- Toute relation possède un identifiant unique (clé primaire).
  - L'identifiant n'admet pas de valeurs `NULL`.
  - L'identifiant peut porter sur plusieurs attributs
  - On souligne la clé primaire dans le schéma de la relation
- Une clé peut être ajoutée si aucun attribut ne peut servir (surrogate key)
- Exemples :
  - Film(<u>Titre</u>, Pays, Année)
  - Film(<u>Titre</u>, Pays, <u>Année</u>)
  - Film(<u>numFilm</u>, Titre, Pays, Année)

Identifiant externe (clé étrangère)

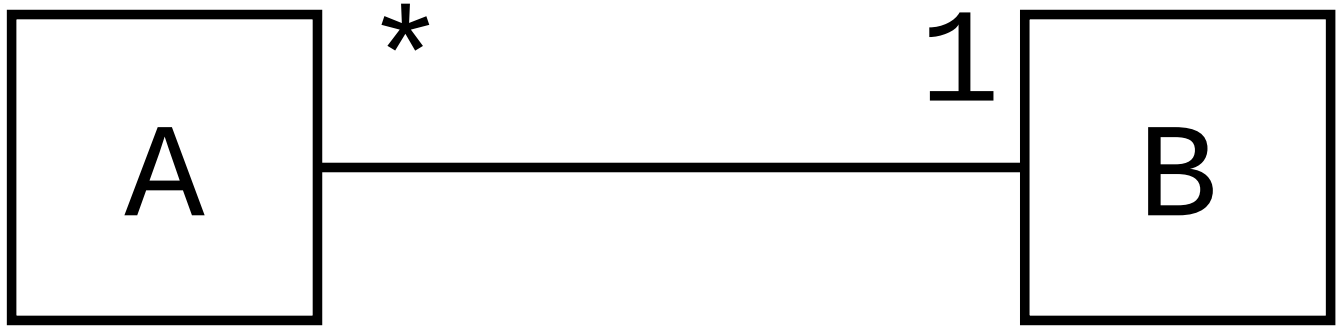
- Permet de décrire des liens entre les différentes relations
- Exemple :
  - Film(Titre, Pays, Année, numFournisseur$_{FK}$)
  - Fournisseur(numFournisseur,Nom,Adresse)
- Intégrité référentielle : les identifiants externes désignent nécessairement des tuples existants.
  - Un film fait obligatoirement référence à un fournisseur qui existe (mais un fournisseur peut n'être lié à aucun film)
- Vérification de l'intégrité référentielle assurée par le SGBD (si on le désire).

Passage du modèle E/A au relationnel



Ajout d'attributs de la relation B à la relation A

Passage du modèle E/A au relationnel



- Ajout de la clé primaire de la relation B et des attributs de l'association comme attribut(s) à la relation A. Les attributs de la relation B deviennent clés étrangères dans la relation A (foreign key).

Passage du modèle E/A au relationnel

- Ajout d'une nouvelle relation correspondant au lien maillé. Les clés primaires des deux relations liées deviennent clés étrangères dans la nouvelle relation. Les propriétés des associations deviennent des attributs de la nouvelle relation.

Exemple de schéma relationnel

Film(numFilm, Titre, Pays, Annee, Note, Duree, Langue, numDistributeur$_{FK}$)
Artiste(numArtiste, Nom, DateNaissance, Nationalite)
Distributeur(numDistributeur, Nom,Adresse)
Client(numClient, Nom, Adresse, Solde)
Magasin(numMagasin, Adresse)
Joue(numFilm$_{FK}$, numArtiste$_{FK}$)
Realisation(numFilm$_{FK}$, numArtiste$_{FK}$)
Location(numLocation,numFilm$_{FK}$, numClient$_{FK}$, DateDe, DateA, numMagasin$_{FK}$)
Stock(numFilm$_{FK}$, numMagasin$_{FK}$, Copies)
Categorie(numCategorie)
Classification(numCategorie$_{FK}$,numFilm$_{FK}$)

Opérateurs ensemblistes

- **Union** : l'union de deux relations R1 et R2 est la relation contenant les tuples de R1 et de R2 (les schémas doivent être égaux).
- **Différence** : la différence de deux relations R1 et R2 est la relation contenant les tuples de R1 et pas ceux de R2 (les schémas doivent être égaux).
- **Intersection** : l'intersection de deux relations R1 et R2 est la relation contenant les tuples appartenant à R1 et à R2 (les schémas doivent être égaux).

Opérateurs spécifiques

- **Projection** : la projection de la relation R1 sur les attributs (A1, A2, ..., An) est une "relation" ayant comme schéma seulement les attributs mentionnés et contenant tous les tuples de la relation R1.
- **Restriction** : la restriction d'une relation R1 par une condition est une relation contenant les tuples de R1 satisfaisant la condition.
- **Produit cartésien** : le produit cartésien de deux relations R1 et R2 est la relation ayant comme schéma l'union des schémas et contenant comme tuple toutes les combinaisons des tuples de R1 et de R2.

Opérateurs spécifiques

- **Jointure interne** : la jointure de deux relations R1 et R2 sous une condition est une relation ayant comme schéma l'union des schémas et contenant comme tuples toutes les combinaisons des tuples de R1 et de R2 satisfaisant la condition.
- **Jointure naturelle** : : omission de la condition s'il existe deux attributs ayant le même nom. L'opérateur de comparaison est alors l'opérateur "=="

Opérateurs spécifiques

- **Jointure externe** : la jointure externe des deux relations R1 et R2 sous une condition est une relation ayant comme schéma l'union des schémas et contenant comme tuples les tuples produites par une jointure interne et les tuples des relations R1 et R2 (qui n'ont pas participé à la jointure) avec des valeurs NULL pour les attributs de l'autre relation.
- Identique pour jointure externe gauche et droite sauf qu'on rajoute uniquement les tuples des relations R1 ou de R2 (qui n'ont pas participé à la jointure)

Exercice

- Donner les titres des films distribués par "Constance Film"
- Donner les noms des clients ayant loué des films dans lesquels apparaît l'acteur "Michael Biehn".
- Donner les adresses des magasins qui stockent au moins un film d'horreur, ainsi que les titres de ces films
- Donner les noms des films réalisés par un artiste qui a aussi de l'expérience professionnelle comme acteur (dans le même film ou dans d'autres).
- Donner les noms des films réalisés par un artiste qui a aussi joué dans le film.

Plan

Structured Query Language

- Langage de définition et de manipulation des données
  - Définition : `CREATE` (bases de données, tables, index)
  - Modification : `INSERT`,`SELECT`, `UPDATE`
- Standard industriel (normalisé ISO)
- Utilisation de mySQL dans le cadre de ce cours (version interactive et intégrée).
- Guide de lecture :
  - `[...]` : argument optionnel
  - `<nom>` : représente le nom d'une variable (supprimer `< >`, exemple `client`)
  - `<type>` : représente le type d'une variable (supprimer `< >`, exemple `int`)
  - On peut indifféremment utiliser les majuscules ou les minuscules pour les mots clés (SELECT)
  - Ce cours est loin d'être exhausif : aller sur l'aide en ligne de MySQL

Gestion des bases de données et des tables

- Création : `CREATE DATABASE <nom-bd>;`
- Suppression : `DROP DATABASE [if exists] <nom-bd>;`
- Utilisation : `USE DATABASE <nom-bd>;`
- Référence une table hors de la base de données : `<nom-bd>.<nom-table>`
- Liste de tables de la base de données : `SHOW TABLES;`
- Afficher la définition d'une table :
  - `SHOW CREATE TABLE <nom-table>;` affichage sous la forme de commandes SQL
  - `DESCRIBE <nom-table>;` (avec mySQL uniquement) affichage sous la forme d'un tableau.

Création d'une table

```
CREATE TABLE <nom-table>
(
 <nom-attr-1> <type> [<options>],
 <nom-attr-1> <type> [<options>],
 [PRIMARY KEY (<nom-attr-i>,<nom-attr-j>,...)],
 [FOREIGN KEY (<nom-attr-a>,<nom-attr-b>,...)] references
<nom-table2>(<nom-attr-c>,<nom-attr-d>,...) <options>],
 [FOREIGN KEY (<nom-attr-r>,<nom-attr-s>,...)] references
<nom-table3>(<nom-attr-t>,<nom-attr-u>,...) <options>]
);
```

- On peut modifier la structure d'une table (ajouter/supprimer des colonnes) avec la commande `ALTER`
- Quelques options associées à un attribut :
  - `NOT NULL`, `DEFAULT <valeur>` et `AUTO_INCREMENT`
  - `PRIMARY KEY` et `UNIQUE` (valeur `NULL` permise pour `UNIQUE`) : peut également porter sur plusieurs attributs

Quelques types

- Différentes options peuvent exister suivant les types (façon d'afficher, signé/non signé pour les valeurs entières).
- Valeurs entières : `TINYINT`, `SMALLINT`, `INT`, `BIGINT`
- Valeurs réelles : `FLOAT`, `DOUBLE`
- Valeurs décimales : `DECIMAL`
- Chaîne de caractères :`CHAR(<longeur>)`, `VARCHAR(<longeur>)`, `TEXT(<longeur>)`, `MEDIUMTEXT(<longeur>)` et `LONGTEXT(<longeur>)`

Quelques types

- Date :
  - `DATE` au format `aaaa-mm-jj`
  - `TIME` au format `hh:mm:ss`
  - `DATETIME` au format `aaaa-mm-jj hh:mm:ss`
- Types particuliers :
  - `ENUM(<val1,val2,...>)` : une valeur parmi les valeurs de la liste (taille max. 64)
  - `SET(<val1,val2,...>)` : zéro ou plusieurs valeurs parmi les valeurs de la liste (taille max. 64)

Insertion de données

- Insertion à partir de valeurs constantes :
  - Pour chaque tuple, on donne la valeur de chaque attribut :

    ```
    INSERT INTO <nom-table> values(<v1a>,<v2a>,...),
    (<v1b>,<v2b>,...);
    ```
  - Pour chaque tuple, on ne donne que certains attributs (si les attributs non donnés n'ont pas de valeur par défaut, et ne s'auto-incrémentent pas, alors, il auront la valeur `NULL`).

    ```
    INSERT INTO <nom-table> (<nom-attr-1>, <nom-attr-2>,
    ...) values(<v1a>,<v2a>,...), (<v1b>,<v2b>,...);
    ```
- On peut insérer les résultats d'une requête :
  - `INSERT INTO <nom-table> SELECT ...;`
  - `INSERT INTO <nom-table> (<nom-attr-1>, <nom-attr-2>, ...) SELECT ...;`

Suppression et modification du contenu

- Suppression de lignes

```
DELETE FROM <nom-table> WHERE <condition>;
```

- Modification de lignes

```
UPDATE <nom-table>
SET
<nom-attr-1>=<expression1>
<nom-attr-2>=<expression2>
WHERE <condition>
ORDER BY <exp-a>, <exp-b>
LIMIT ...
```

- Il est possible de faire des jointures dans la clause `UPDATE`.

Requêtes à partir d'une table

```
SELECT [DISTINCT|ALL] <exp-i>,<exp-j>,...
FROM <nom-table>
WHERE <condition>
ORDER BY <exp-k> [ASC|DESC], <exp-l>
[ASC|DESC], ...
LIMIT <entier>
```

- Pour chercher tous les attributs (`SELECT * ...`)
- Toutes les clauses sont optionnelles (sauf le `SELECT`)
- Les expressions `<exp-i>` : `<nom-table>.<nom-attribut>` ou `<nom-attribut>` si aucune ambiguité. On verra plus tard que les expressions peuvent être plus « complexes ».

Requêtes sur plusieurs tables : la clause from

- Produit cartésien
  - `FROM <nom-table1> CROSS JOIN <nom-table2>`
  - `FROM <nom-table1>,<nom-table2>`
- Jointures
  - jointure interne : `FROM <nom-table1> INNER JOIN <nom-table2> on <cond-jointure>`
  - jointure externe à gauche : remplacer `INNER` par `LEFT`
  - jointure externe à droite : remplacer `INNER` par `RIGHT`
  - jointure externe : remplacer `INNER` par `OUTER` (non disponible avec mySQL)
  - jointure naturelle : remplacer `INNER` par `NATURAL` et ne pas mettre de condition de jointure !
- Remarques :
  - On peut réaliser autant de jointures et de produits cartésiens que l'on veut dans la clause `FROM`.
  - On peut réaliser une jointure interne en réalisant un produit cartésien dans la clause `FROM` et en insérant la condition de jointure dans la clause `WHERE`

Les alias

- Les alias d'un attribut
  - `SELECT <expr-1> as <alias1>, <expr-2> as <alias2> FROM`
  - En général, ils ne peuvent pas être utilisés dans les clauses suivantes.
  - Ils servent notamment à définir les sous-requêtes
- Les alias de table
  - `FROM <nom-table1> <aliastable1> INNER ...`
  - Les alias de table doivent être utilisées dans toutes les clauses !
  - Les alias de tables permettent d'utiliser deux fois la même table dans la clause `FROM` et de définir des sous-requêtes

Les fonctions d'agrégation

Les fonctions d'agrégation permettent de combiner plusieurs lignes.

- `avg <exp>` : moyenne des valeurs
- `sum <exp>` : somme des valeurs
- `min <exp>` : minimum des valeurs
- `max <exp>` : maximum des valeurs
- `count(*)` : nombre de lignes
- `count(<exp>)` : nombre de valeurs non nulles
- `count(distinct <exp>)` : nombre de valeurs différentes non nulles
- `group_concat <exp>` : concaténer les chaînes de caractères

Agrégation

```
SELECT <expr-1>,<expr-2>, ..., <expr-comb-1>,<expr-comb-2>,...
FROM ...
WHERE ...
GROUP BY <expr-1>,<expr-2>, ...
WITH ROLLUP
HAVING <expr-comb-condition>
ORDER BY ...
```

- Il y a une ligne par groupe : les seuls attributs permis dans la clause `SELECT` sont ceux de la clause `GROUP BY`.
- La clause `HAVING` impose une condition sur les groupes
- La clause `ROLLUP` crée des méta-groupes (ne peut pas être utilisée avec `ORDER BY ...`).
- Il est possible de ne pas utiliser la clause `GROUP BY` : le résultat rendra alors au plus une ligne. Dans ce cas, on ne peut sélectionner que des fonctions d'agrégation dans la clause `SELECT`.

Les opérateurs ensemblistes

```
SELECT ...
FROM ...
UNION | MINUS | INTERSECT ...
SELECT ...
FROM ...
```

- `UNION` : union des lignes et suppression des doublons
- `MINUS` : ne garde dans le résultat de la première requête que les lignes qui ne sont pas dans le résultat de la seconde requête, puis supprime les doublons. Non disponible avec mySQL
- `INTERSECT` : ne prend que les lignes communes aux résultats deux requêtes et supprime les doublons. Non disponible avec mySQL.
- Le comportement des opérateurs `MINUS` et `INTERSECT` pourra être réalisé avec des sous-requêtes.

Sous-requête dans la clause `FROM`

```
SELECT <expr-1a>,<expr-2a>, ...
FROM ... INNER JOIN
(SELECT <expr-1b> as <alias-1b> ,<expr-1c> as <alias-1c>,...
FROM ... WHERE ... [et autres clauses possibles] )<alias-table>
ON <expr-condition-jointure>

...
```

- Tout se passe comme si : la sous-requête est exécutée en premier et le résultat est mis dans une table temporaire de nom `<alias-table>` avec les attributs `<alias-1b>`, `<alias-1c>`.
- En pratique, cela ne se passe pas comme ça (optimisation des requêtes de manière à optimiser les performances)

Sous-requête dans la clause WHERE : version simple

```
SELECT <expr-1a>,<expr-2a>, ...
FROM ...
WHERE <expr-i>, <expr-j>,... <comparaison-avec-tables>
( SELECT <expr-r>, <expr-s>, ... FROM WHERE ...)
GROUP BY ... HAVING...
```

- `<comparaison-avec-tables>` permet de comparer une ligne avec une « table ». Il est défini comme ceci : `<operateur-comparaison> ALL`, `<operateur-comparaison> ANY`, `IN`, `NOT IN`, et uniquement `<operateur-comparaison>` si la sous-requête rend une ligne
- Tout se passe comme si : la sous-requête est exécutée en premier et on met le résultat dans un tableau temporaire. Ensuite, on exécute la requête principale et on utilise la sous-requête pour tester la condition de restriction (clause `WHERE`).
- Remarque : on peut également utiliser une sous-requête dans la clause `HAVING`

Sous-requête dans la clause `WHERE` : version corrélée

```
SELECT <expr-1a>,<expr-2a>, ...
FROM <tab1> ....
WHERE <expr-i>, <expr-j>,... <comparaison-avec-tables>
( SELECT <expr-r>, <expr-s>, ... FROM WHERE tab1.i=....)

GROUP BY ... HAVING...
```

- La sous-requête utilise la valeur d'une colonne de la requête principale (dans l'exemple ci-dessus, on l'a mis dans la clause `WHERE` de la sous-requête).
- La sous-requête corrélée est exécutée pour chaque ligne de la requête principale (de manière à tester si la condition de restriction de la requête principale est vérifiée)

Sous-requête dans la clause `WHERE` : version corrélée

```
SELECT <expr-1a>,<expr-2a>, ...
FROM <tab1> ....
WHERE [NOT] EXISTS
( SELECT ... FROM ... WHERE tab1.i=....)

GROUP BY ... HAVING...
```

- Pour chaque ligne de la requête extérieure, la sous requête est exécutée.
- `EXISTS` : la condition de restriction est vérifiée si la requête rend au moins une ligne.
- On peut remplacer `EXISTS` par `UNIQUE` : la condition de restriction est vérifiée si la sous-requête rend exactement une ligne.
- On peut utiliser la négation `NOT` pour obtenir le comportement inverse.

Les expressions en pratique

- `<nom-table>.<nom-attribut>` ou `<nom-attribut>` si aucune ambiguité.
- On peut utiliser les opérateurs mathématiques traditionnels pour la plupart des types(l'ordre pouvant être forcé par des parenthèses), ainsi que des fonctions :
  - pour des valeurs numériques : `ABS`,`SIN`,`COS`,...
  - pour des chaînes de caractères : `CONCAT`,`INSERT`,...
  - pour les dates : `ADDDATE`, `SUBDATE`, extraire des champs (`YEAR`, `DAYOFYEAR`...) ,...
  - pour gérer les valeures nulles : `IFNULL(<e1>,<e2>)` (renvoie `<e1>` si `<e1>` est non `NULL`, `<e2>` sinon), `COALESCE` (retourne la première valeure non `NULL`).
  - Les opérateurs mathématiques, les fonctions (et les fonctions d'aggrégation) sont adaptés au cas des valeurs `NULL`

Gestion du flux

- `IF(<expr_test,exp_vrai,expr_sinon>)`
  Exemple : `SELECT IF(1,2,3),IF(0,2,3)`
  renvoie 2,3.
- `CASE`
  Exemple :

```
SELECT CASE(1)
WHEN 1 THEN 'un'
WHEN 2 THEN 'deux'
ELSE 'ni un, ni deux'
END
```

  renvoie un.

Les expressions logiques (1/2)

- 3 états (vrai, faux, indéterminé)
- une valeur numérique est considérée comme vraie si différente de 0 (et `NOT NULL`), fausse si égale à 0, et indéterminée si `NULL`
- les opérateurs logiques (`NOT`, `AND`, `OR`, `XOR`) sont étendues aux 3 états
- Obtenir des expressions logiques : `=, <>, >, >=, <, <=, <=>, BETWEEN, IN`
  - `select 4=4;` rend 1 (vrai)
  - `select NULL=NULL;` rend NULL (indéterminée)
  - `select NULL<=>NULL;` rend 1 (vrai)
  - `select 2 BETWEEN 1 AND 4;` rend 1 (vrai)
  - `select 2 IN (1,4);` rend 0 (faux)

Les expressions logiques (2/2)

- Pour les chaînes de caractère, on peut aussi utiliser les expressions régulières ou la fonction `LIKE` pour regarder si une chaîne de caractère se conforme à un modèle.
- Tester si une variable est `NULL`
  - `ISNULL(<expr>)` : renvoie 1 si `expr` est `NULL`, 0 sinon.
  - `<expr> IS NULL` : renvoie 1 si `expr` est `NULL`, 0 sinon.
  - `<expr> IS NOT NULL` : renvoie 0 si `expr` est `NULL`, 1 sinon.

Plan

Normalisation

- Il faut que la base de données contienne le moins de données redondantes possibles.
- Des redondances peuvent conduire à des mises à jours (insertion, modification, suppression) plus complexes
- Les relations pour qu'une base de données soit « correcte » peuvent être définies formellement : ce sont les règles de normalisation.
- Après la traduction du schéma conceptuel en schéma logique relationnel, on peut vérifier que le schéma logique est « correct », et le modifier le cas échéant.

Les contraintes

- Contraintes de clés primaires, étrangères, d'unicité et de non-nullité.
- Il existe d'autres types de contraintes :
  - contraintes temporelles : basées sur la comparaison entre l'ancienne valeur d'un attribut et la nouvelle
  - contraintes équationnelles : comparer deux expressions arithmétiques calculées à partir des données et forcer l'égalité ou l'inégalité (le nombre de locations d'un livre en cours ne peut pas dépasser le nombre d'exemplaires).
  - il existe plein d'autres contraintes : contrainte de positivité, dépendances fonctionnelles...
- Garder la base de données cohérente (les données doivent vérifier les contraintes).
  - Vérifier les données lors des mises à jour (insertions, modifications, suppressions).
  - Eventuellement répercuter certaines mises à jour entre tables.

Contraintes avec SQL

- Contraintes simple définies avec la table
  - Contrainte sur la valeur d'une colonne : CHECK (A between 1 and 10)
  - Contrainte sur la valeur de plusieurs colonnes : CHECK(A-B=C)
- Non disponible avec MySQL
- Les contrainte simples avec MySQL peuvent être réalisées avec des vues, des déclencheurs et des procédures stockées.
- Les contraintes « complexes » peuvent être réalisées avec des déclencheurs et des procédures stockées.

« MySQL intégré »

- Nécessité d'avoir un langage procédural. Il est possible par exemple avec MySQL de
  - Déclarer des variables typées
  - Effectuer des traitements itératifs
  - Effectuer des traitements conditionnels
  - Gérer des exceptions
- Utilisé pour définir les déclencheurs, les procédures stockées.

Procédures et fonctions embarquées

- Une procédure stockée est un jeu de commandes SQL (MySQL intégré) qui réside sur le serveur.
  - gain de performances car moins d'informations sont échangées entre le serveur et le client
  - augmente la charge du serveur
- Utilité :
  - Plusieurs applications clientes sur différentes plate-formes et langages.
  - Sécurité : les applications et les utilisateurs n'ont aucun accès direct aux tables, mais passent par des procédures stockées pre-définies
  - Ré-utilisabilité
  - Ne pas dissocier les données et les requêtes
  - Peut permettre de gérer des données redondantes et les contraintes

Les déclencheurs : SGBD actifs

- Un déclencheur (trigger en anglais) est activé par un événement
  - création d'une table, insertion de tuples, modification de tuples, suppression de tuples...
  - éléments d'administration : démarrage de la BD.
- Le déclencheur est déclenché avant, après, ou à la place de l'événement déclencheur
- Le déclencheur peut
  - interdir l'opération (si les contraintes ne sont pas vérifiées)
  - annuler la transaction ou la mise à jour (ou même la modifier)
  - déclencher d'autres opérations (utilisation de mySQL intégré)
- utilité : SGBD capable de réagir afin de contrôler l'intégrité, de gérer les redondances, d'autoriser ou d'interdire...

Les vues

- Vue : Table dont le schéma et le contenu sont dérivés de la base réelle par un ensemble de questions : n'a pas d'existence propre, aucune données ne lui est associée, sa description est stockée sous la forme d'une requête.
- Utilité :
  - Indépendance logique des applications par rapport à la base
  - Confidentialité : les utilisateurs n'ont le droit d'accéder qu'aux vues
  - Cacher la complexité des données aux utilisateurs (vue simplifiée des données, simplifier les requêtes des non spécialistes)
  - Présenter différentes perspectives sur les données aux utilisateurs (renommage des colonnes)
- Problèmes :
  - Interrogation efficace
  - Mise à jour au travers des vues (pas toujours possible)

Vue concrète

- Vue concrète : table calculée à partir des tables de la base par une question mais qui est matérialisée sur disques par le SGBD. La vue contient donc des données redondantes !
- Vue auto-maintenable (Self-maintenable view) : vue contenant suffisamment d'information pour être mise à jour à partir des mises à jour des relations de base et de la vue, sans accès aux autres tuples des relations de la base.
- Intérêt d'être matérialisé sur le disque : interrogation beaucoup plus efficace si la vue est « complexe » à calculer...
- Comment tenir à jour une vue concrète lors des mises à jour (modification,suppression,insertion)
  - Modification immédiate : utilisation de triggers (qui met à jour la vue concrète) ou de procédures stockées (qui s'occupe de la mise à jour des tables de la base de donnée et de la vue concrète).
  - Modification différée : scripts de mise à jours (ensemble de requêtes qui se basent sur les dates d'enregistrement par exemple).

Les transactions

- Une transaction est une unité de travail (la location d'un film nécessite de mettre à jour la table Client (diminuer le solde) et d'insérer une ligne à la table Location).
- Propiété des transactions : ACID
  - Atomicité : elle est exécutée entièrement ou pas du tout, même dans le cas d'une panne.
  - Cohérence : elle fait passer la base de données d'un état cohérent à un autre état cohérent, même dans le cas d'une panne.
  - Isolation : si plusieurs transactions sont executées simultanément, les effets de chaque transaction sont invisibles aux autres transactions.
  - Durabilité : si une transaction aboutit, ses résultats ne peuvent pas être annulés par une autre transaction.

Les transactions : en pratique (table innodb sous mySQL)

- Atomicité :
  - Journalisation (sur support non volatile) de l'activité des transactions : fichier de journalisation LOG
  - Procédure de reprise après panne : reconstruction de la BD à partir du LOG
  - Possbilité de réduire la taille du LOG (checkpointing)
- Isolation :
  - Utilisation de méthodes pessimistes (verrouillage)
  - Il existe plusieurs niveaux d'isolation
  - Le niveau de verrouillage le plus important permet d'obtenir des exécutions sérialisables (il existe un ordre des transactions qui donnerait le même résultat final que celui obtenu en exécutant les transactions en parallèle)

Fiabilité

- La base de données doit toujours être cohérente.
- Fiabilité : capacité à restaurer la base de données dans un état connu comme correct
- Une base de données peut être incohérente à cause de
  - Erreurs dans l'entrée des données $\rightarrow$ contraintes, déclencheurs, procédures stockées ;
  - Erreurs disques, catastrophes $\rightarrow$ redondance, archivage et fichiers LOG de journalisation ;
  - Défaillance système $\rightarrow$ fichiers LOG de journalisation

Organisation des tables

- Une table est sauvegardée dans un fichier
- Différents types d'organisation du fichier : organisation des enregistrements les uns par rapport aux autres (organisation séquentielle, organisation aléatoire ou indexée)
- Différentes méthode d'accès suivant l'organisation du fichier : façon d'accéder à un enregistrement dans un fichier (accès séquentiel et accès sélectif selon la valeur d'un attribut de l'enregistrement).

Traitement des requêtes

- SQL : langage assertionnel (expression de l'ensemble des données que l'on souhaite obtenir sans indiquer la façon de les obtenir)
- 2 techniques d'optimisation (qui sont, pour le plus souvent combinées) pour améliorer la performance :
  - Réorganisation des expressions algébriques
  - Choix de la meilleure stratégie pour chaque opération ou groupe d'opération (jonction, restriction, ...)
  - Le choix dépend de l'organisation des différentes tables ainsi que de leur statistique (lancer la fonction `ANALYZE` sur les tables régulièrement)

En cas de problèmes

- L'outils `EXPLAIN` permet de savoir comment la requête est « traitée ».
  - Pour chaque table accédée, il donne le mode d'accès utilisé, la liste des index utilisables, l'index utilisé, la longueur du préfixe de la clé pour parcourir l'index, le critère d'accès à l'index, l'estimation du nombre d'enregistrements auquel il faut accéder.
  - modifier le type d'organisation du fichier associé à une table (alétoire vs indexé)
  - Ajouter/Supprimer des indexes, modifier la fonction de hachage
  - Les requêtes peuvent être écrites de manière à forcer les modes d'accès aux fichiers.