Chapitre 9 : Bases de données

Table des matières

1 Concepts élémentaires

1.1 Introduction aux bases de données

1.1.1 Introduction

De nombreuses applications informatiques manipulent de grandes quantités de données qui doivent être organisées et stockées de sorte qu'il est possible de les traiter efficacement pour en ajouter, en retirer, ou en extraire de l'information. Ce traitement doit pouvoir se faire de manière concurrente tout en préservant l'intégrité des données.

1.1.2 Systèmes de gestion de bases de données (SGBD) et paradigme logique

Préserver l'intégrité des données est une tâche complexe dans un contexte d'accès concurrents, donc le traitement des données sera confié à un outil appelé SGBD dont le rôle est de recevoir les requêtes des utilisateurs (modification des données ou extraction d'information à partir de ces données) et de les traduire en des opérations effectuées sur la base de données. C'est au SGBD de garantir la cohérence des données au fur et à mesure des opérations réalisées. Les requêtes prennent en général la forme d'une description du résultat attendu sans indication sur la manière de calculer. C'est le SGBD qui implémente la recherche du résultat.

C'est le principe du paradigme logique : un programme est une description des propriétés que doit satisfaire le résultat. Un résultat est un jeu de paramètres qui satisfait les propriétés. Il n'y a aucune indication sur la manière de calculer les résultats.

Pour que cela fonctionne, la plupart des SGBD s'appuient sur un modèle introduit dans les années 1970, appelé modèle relationnel.

1.1.3 Le modèle relationnel

• Le modèle relationnel est une modèle mathématique basé sur la théorie des ensembles et la logique des prédicats, et qui présente les bases de données comme des objets qui définissent des relations entre les blocs d'information. C'est un modèle abstrait qui s'exprime indépendamment des implémentations possibles et qui couvre donc de nombreux SGBD.

• Dans ce modèle, une base de données est vue comme un ensemble de relations. Ces relations sont aussi appelées *tables* car on peut les représenter par des tableaux à double entrée dont les colonnes correspondent à un type d'information particulier. Ces colonnes sont appelées les *attributs* de la relation.

Exemple : dans le système d'information d'une bibliothèque, on peut avoir une table Document dont les attributs sont le titre, l'auteur, le genre, la date de parution, le nombre de pages, ...

• Chaque attribut est associé à un domaine qui correspond à l'ensemble des valeurs possibles par l'attribut. Le domaine permet de choisir un type pour implémenter concrè-

tement la base de données.

Étant donné une relation R dont les attributs sont $A_1, \dots, A_n$ associés aux domaines $D_1, \dots, D_n$, on appelle *schéma relationnel* de R l'association des attributs et des domaines notée

$$R(A_1 : D_1, \dots, A_n : D_n)$$

Exemple : pour la table Document :

- Le titre et l'auteur sont des données textuelles ;
- Le genre est tirée d'une énumération finie (roman, poésie, théâtre, ...) ;
- La date de parution est une date ;
- Le nombre de pages est un entier.

D'où le schéma relationnel :

Document(titre : texte, auteur : texte, genre : enum(roman, ...), date de parution : date, nombre de pages : entier)

Le domaine "texte" peut être par exemple associé au type `string` des chaînes de caractères et le domaine entier au type `int`.

• Les lignes d'une relation de schéma $R(A_1 : D_1, \dots, A_n : D_n)$ correspondent aux éléments de la relation R, qui est un sous ensemble de $\prod_{k=1}^{n} D_k$. On appelle donc ces éléments des tuples ou des enregistrements.

Exemple pour la table Document :

Titre	Auteur	Genre	Date de parution	Nombre de pages
La cousine Bette	Honoré de Balzac	Roman	1846	240
De la guerre	Carl von Clausewitz	Traité	1832	240
Cyrano de Bergerac	Edmond Rostand	Théâtre	1857	280

Remarque : certains enregistrements peuvent coïncider pour certains attributs et on veut une manière efficace de les distinguer.

• On appelle *clé candidate* un ensemble minimal (pour l'inclusion) d'attributs permettant de caractériser de manière unique chaque enregistrement, *i.e* tel qu'il n'existe pas deux enregistrements qui coïncident sur tous les attributs de la clé.

Exemple : pour la table Document, {titre, auteur} devrait convenir.

Il peut y avoir plusieurs clés candidates et on doit en choisir une, appelée *clé primaire*. On souligne dans le schéma relationnel les attributs de la clé primaire pour les repérer efficacement.

Remarque : on choisit souvent d'ajouter un attribut entier pour numéroter les enregistrements, que l'on choisit comme clé primaire.

• Dans une relation R, on appelle *clé étrangère* un ensemble d'attributs qui constitue une clé candidate (souvent primaire) d'une autre relation.

Exemple : dans une table Emprunts décrivant les emprunts de la bibliothèque, on intègre la clé de la table Document pour identifier les documents empruntés.

1.1.4 Algèbre relationnelle

L'algèbre relationnelle est une théorie mathématique qui décrit des opérations que l'on peut réaliser sur une base de données du point de vue du modèle relationnel. Les propriétés qui découlent de cette théorie définissent un fondement rigoureux aux implémentations de SGBD en justifiant les optimisations des requêtes des utilisateurs. Cette théorie des H.P, mais nous l'étudierons *via* le langage de requêtes SQL (Structured Query Language).

1.2 Le langage SQL : requêtes élémentaires

1.2.1 Gestion du contexte

– Choix de la base de données : `USE <nom_base>;`

– Obtention du schéma relationnel d'une table : `DESCRIBE <nom_table>;`

Exemple :

```
USE bibliotheque;
DESCRIBE Document;
```

Les opérations de création / suppression / modification de bases de données / de tables sont H.P.

1.2.2 Opération de projection

Pour visualiser le contenu d'une table, on utilise la requête `SELECT * FROM <nom_table>;`

C'est une cas particulier de l'opération de projection qui permet de construire une table ne contenant que les valeurs des tuples que pour certains attributs.

Attention, la table construite par une requête est éphémère.

Syntaxe : `SELECT <attribut`$_1$`>, ..., <attribut`$_n$`> FROM <nom_table>;`

Exemple :

```
SELECT titre, date_de_parution FROM Document;
```

Remarque : cette requête peut créer une table contenant des doublons qui sont conservés par défaut. On utilise le mot-clé `DISTINCT` pour éliminer les doublons.

Exemple :

```
SELECT DISTINCT titre, date_de_parution FROM Document;
```

Les enregistrements du résultat d'une projection sont a *priori* rangés dans le même ordre que dans la table initiale. On peut choisir de les réordonner en utilisant le mot-clé `ORDER BY` suivi d'une liste d'attributs.

L'ordre associé à cette liste est l'ordre croissant lexicographique.

Exemple :

```sql
SELECT * FROM Document
ORDER BY auteur, date_de_parution;
```

Pour utiliser l'ordre décroissant selon l'un des attributs, on utilise le mot-clé `DESC` après le nom de l'attribut.

Exemple :

```sql
SELECT * FROM Document
ORDER BY date_de_parution DESC;
```

Attention, `DESC` ne porte que sur un seul attribut (le répéter si besoin).

Il est aussi possible de renommer les attributs du résultat d'une projection, ce qui peut être nécessaire dans des opérations plus complexes faisant intervenir des sous-requêtes ou plusieurs tables. On utilise pour cela le mot-clé `AS`.

Exemple :

```sql
SELECT titre, auteur, date_de_parution AS date
FROM Document;
```

On peut combiner tout cela :

```sql
SELECT titre, date_de_parution AS date
FROM Document
ORDER BY date DESC, titre;
```

1.2.3 Opération de sélection

On peut ne conserver que les enregistrements qui satisfont une condition donnée *via* l'opération de sélection, introduite par le mot-clé `WHERE`.

Exemple :

```sql
SELECT * FROM Document
WHERE auteur = 'Edmond Rostand';
```

Les opérateurs de comparaison au programme sont : `=, <>, <, >, <=, >=`

Attention, les enregistrements ne fournissent pas forcément une valeur pour chaque attribut. Les attributs sans valeurs prennent la valeur spéciale `NULL`. On ne peut pas vérifier si un attribut a la valeur `NULL` avec les opérateurs ci-dessus : on utilise les opérateurs `IS NULL` et `IS NOT NULL`

Exemple :

```sql
SELECT * FROM Document
WHERE date_de_parution IS NULL;
```

On peut combiner plusieurs conditions grâce aux connecteurs logiques : `AND, OR, NOT`.

Exemple :

```sql
SELECT * FROM Document
WHERE NOT (date_de_parution >= 2000 AND nombre_de_page < 200);
```

Remarque : On peut combiner sélection et projection

On peut aussi fixer un nombre maximal d'enregistrements pour le résultat : le SGBD choisira les premiers qu'il traite jusqu'à éventuellement atteindre la borne. On utilise le mot clé `LIMIT`

Exemple : 10 ouvrages les + récents

```
SELECT * FROM Document
ORDER BY date_de_parution DESC
LIMIT 10;
```

On peut ajouter un décalage lors de l'usage de `LIMIT` grâce au mot-clé `OFFSET`.

Exemple : dernier tiers du top 15 des auteurs ayant écrit les plus gros livres.

```
SELECT DISTINCT auteur FROM Document
ORDER BY nombre_de_pages DESC
LIMIT 5 OFFSET 10;
```

2 Conception de bases de données

2.1 Modèle entité-association

2.1.1 Introduction

Le modèle *entité-association* est un modèle de la fin des années 1970 qui propose un formalisme pour décrire la structure des données en fonction des liens qu'elles entretiennent.

On lui associe une représentation graphique permettant d'identifier facilement les deux concepts de base du modèle : les *entités* et les *associations*.

2.1.2 Entités

Une entité est un objet donné pour lequel on dispose de données que l'on souhaite traiter.

Exemple : moi, le livre d'Edmond Rostand intitulé *Cyrano de Bergerac*, ...

Les informations associées à une entité sont ses attributs. On réunit un ensemble d'entités ayant des caractéristiques communes dans un type d'entité que l'on appelle aussi par abus entités.

Dans ce cadre, une entité concrète est appelée instance ou occurrence de l'entité.

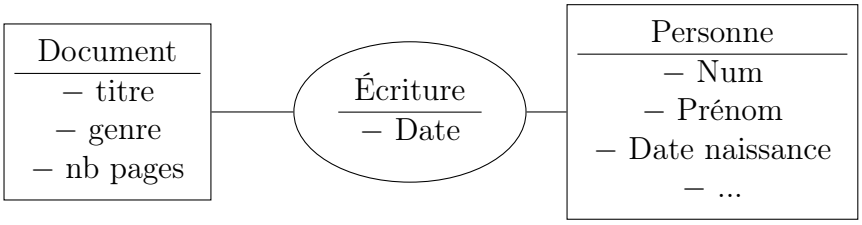
2.1.3 Associations

Une association exprime un lien entre plusieurs entités.

Exemple : Edmond Rostand *a écrit* le livre intitulé *Cyrano de Bergerac*.

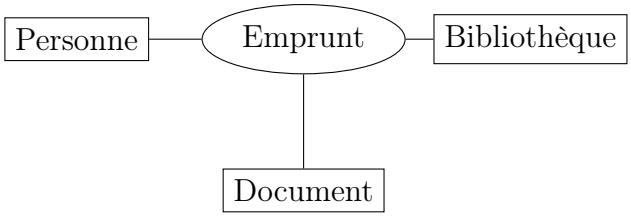
Une association peut aussi avoir des attributs (ex : date d'écriture) et on fait également l'abus de langage qui consiste à appeler association un type d'association.

Représentation graphique : on dessine deux types de bloc (les entités et les associations) et on lie les blocs d'entité *via* les blocs d'association qui expriment ces liens

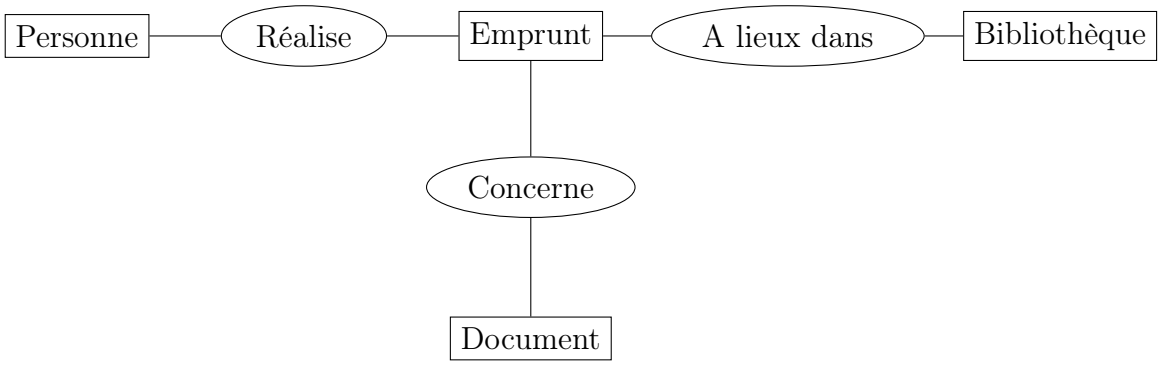


On distingue les associations binaires (*i.e* qui lient 2 entités) des associations n-aires. On peut toujours se limiter aux associations binaires en remplaçant une association n-aire par une nouvelle entité représentant l'association, liée aux n entités par n associations binaires.

Exemple :



donne :



2.1.4 Cardinalité d'une association

Le lien entre une entité et une association peut être étiqueté par un couple (p, q) représentant les nombres minimums et maximum de fois que l'entité peut apparaître dans une association de ce type : $q = *$ s'il n'y a pas de borne supérieure.

Exemple :

On appelle cette association une association $* - *$ car elle peut lier plusieurs entités à plusieurs autres.

D'autres types d'association sont :

– Les associations $1 - *$: elles lient une entité à plusieurs autres.

Exemple : une œuvre peut être tirée à plusieurs exemplaires mais un exemplaire n'est un tirage que d'une seule œuvre

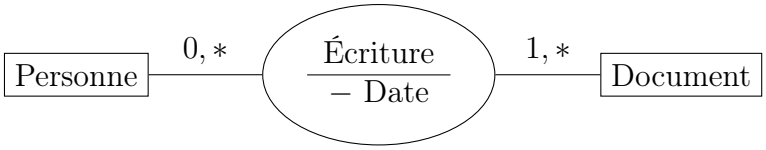
– Les associations $1-1$: elles lient une entité à une seule autre.

Exemple : chaque exemplaire d'une bibliothèque a une référence unique

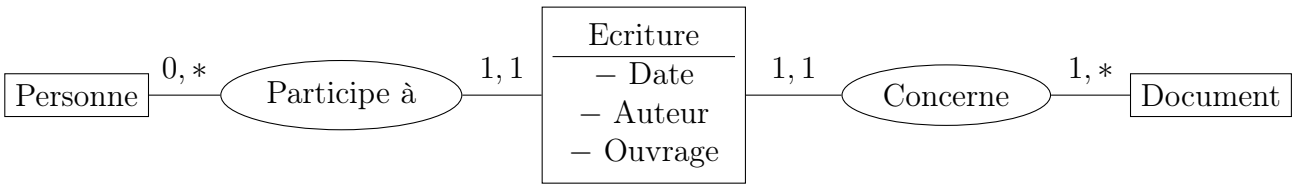
Remarque : on peut souvent fusionner les entités impliquées dans une association $1-1$ sans introduire trop de redondance dans les données.

Une association $*-*$ peut être scindée en deux associations $1-*$ *via* l'introduction d'une nouvelle entité représentant l'association.

Exemple :



donne :



Remarque : les attributs de la nouvelle entité doivent permettre d'identifier les entités impliquées dans l'association : on utilise des clés étrangères dans le modèle relationnel associé à ce modèle.

2.2 Passage du modèle entité-association au modèle relationnel

2.2.1 Idée

On associe une relation à chaque entité, avec les mêmes attributs, les associations expriment des liens entre les tables et introduisent donc des contraintes d'intégrité des données.

• La décomposition d'une association n-aire ou en associations binaires se traduit en la création d'une nouvelle entité donc d'une nouvelle association.

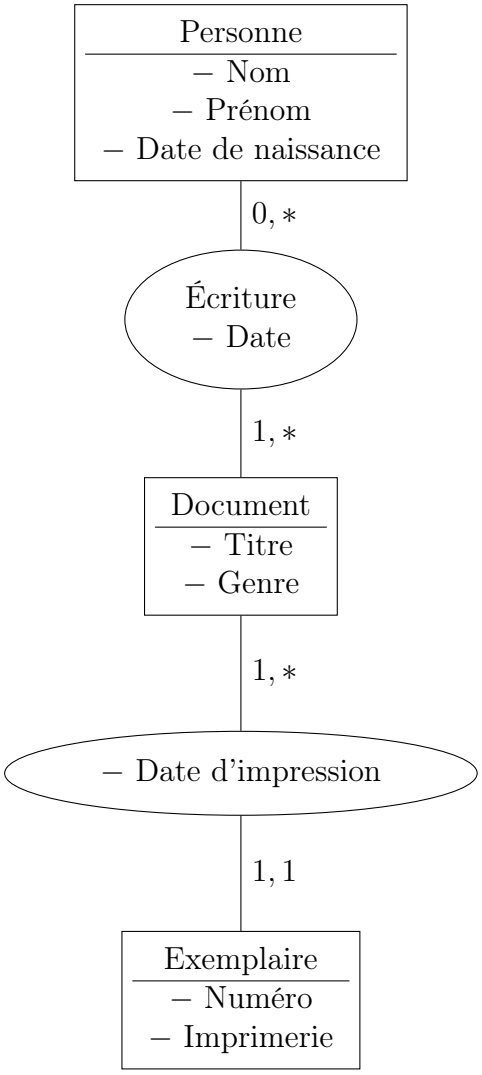
• La décomposition des associations binaires $*-*$ en deux associations $1-*$ se traduit aussi en la création d'une nouvelle table, dite de *jonction*.

• Après cela, il ne reste qu'à traiter les associations $1-1$ et $1-*$.

2.2.2 Cas des associations $1-1$ et $1-*$

Ces types d'association impliquent que pour l'une des deux entités, chaque instance de l'entité ne peut être associée qu'à une unique instance de l'autre entité. Pour exprimer ce lien entre les tables correspondantes on ajoute une clé étrangère à la première table, faisant référence à l'unique enregistrement de la seconde table impliquée dans l'association. Les attributs de l'association seront aussi intégrés à la première table.

Exemple :



Modèle relationnel associé : on utilise des identifiants numériques pour les clés primaires :

Personne(id : entier, nom : texte, prénom : texte, date de naissance : date)

Document(id Doc : entier, titre : texte, genre : enuml(...))

Exemplaire(numéro : entier, imprimerie : texte, id Doc : entier, date d'impression : date)

Écriture(date : date, id Doc : entier, id : entier)

3 Requêtes avancées

3.1 Opérations ensemblistes

3.1.1 Remarque

On s'intéresse à des opérations qui prennent plusieurs tables en argument et qui produisent une nouvelle table.

3.1.2 Produit cartésien

On construit le produit cartésien de deux relations vues comme des ensembles de tuples.

Schéma relationnel : si on a deux relations $R(A_1 : D_1, \ldots, A_n : D_n)$ et $R'(A'_1 : D'_1, \ldots, A'_n : D'_n)$, alors

$$(R \times R')(A_1 : D_1, \ldots, A_n : D_n, A'_1 : D'_1, \ldots, A'_n : D'_n)$$

de clé primaire l'union des clés primaires de R et R'.

Réalisation SQL : on sépare les tables par des virgules dans la clause `FROM`.

Ex :

```
SELECT * FROM R, R';
```

Remarque : on peut faire des produits n-aires :

```
SELECT * FROM R1, R2, ..., Rn;
```

Exemple : auteur, titre et date d'écriture dans la BDD vue en 2.2.2 (page 10).

```
SELECT nom, prenom, titre, date
FROM Personne, Document, Ecriture
WHERE Personne.id = Ecriture.id AND Document.idDoc = Ecriture.
    idDoc;
```

Remarque : il existe une construction appelée jointure qui est plus adaptée à ce type de requête (*cf* 3.2, page 13).

3.1.3 Définition (relations *union-compatibles*)

Deux relations sont *union-compatibles* si elles ont le même nombre d'attributs et si les attributs de même position dans les deux relations ont même domaine.

Intuition : même schéma au renommage des attributs et au choix de la clé primaire près.

3.1.4 Union

L'union de deux relations union-compatibles est une relation de *même schéma que la première* et dont les enregistrements sont ceux qui apparaissent dans au moins l'une des deux relations.

Remarque : cela signifie que si les noms d'attributs sont différents pour deux relations, alors on conserve ceux de la première. De plus, les doublons sont supprimés.

Réalisation SQL : on utilise le mot clé `UNION` entre deux requêtes qui produisent les tables dont on veut faire l'union.

Attention : pas de `UNION` dans une clause `FROM`

Exemple :

```
SELECT ... UNION SELECT ... ;
```

Exemple : dates qui sont des dates d'écriture ou d'impression de documents.

```
SELECT date FROM Ecriture
UNION
SELECT date d impression FROM Exemplaire;
```

3.1.5 Intersection

L'intersection de deux relations union-compatibles est une relation de même schéma que la première dont les enregistrements sont les tuples qui apparaissent dans les deux relations.

Réalisation SQL : on utilise le mot-clé `INTERSECT` de la même manière que `UNION`.

Remarque : certains SGBD (comme MySQL) n'implémentent pas cette opération. Dans ce cas, il faut encoder cette opération avec des requêtes. On peut par exemple utiliser un produit cartésien.

Exemple : Prénom qui sont aussi des titres d'œuvres.

```
SELECT prenom FROM Personne
INTERSECT
SELECT titre FROM Document;
```

```
SELECT DISTINCT prenom FROM Personne, Document
WHERE prenom = titre;
```

Remarque : la clause `WHERE` contient autant de tests d'égalité qu'il y a d'attributs dans le résultat.

3.1.6 Différence

La différence de deux relations union-compatibles est une relation de même schéma que la première relation dont les enregistrements sont les tuples qui apparaissent dans la première relation mais pas dans la seconde.

Réalisation SQL : on utilise le mot-clé `EXCEPT` de la même manière que `UNION`.

Remarque : certains SGBD n'implémentent pas cette opération, d'autres utilisent le mot-clé `MINUS` (qui est H.P).

On peut encoder l'opération à l'aide de requêtes imbriquées (*cf* 3.3, page 15).

Exemple : noms qui ne sont pas des prénoms :

```
SELECT nom FROM Personne
EXCEPT
SELECT prenom FROM Personne;
```

```sql
SELECT nom FROM Personne AS P1
WHERE NOT EXISTS
    (SELECT prenom FROM Personne AS P2
    WHERE P1.nom = P1.prenom);
```

3.2 Jointures

3.2.1 Principe

Il s'agit d'établir un lien entre plusieurs tables sous certaines contraintes.

L'idée est la même que la création d'une table de jonction pour décomposer une association $* - *$: un opérateur de jointure crée une nouvelle relation recollant les enregistrements de deux relations qui se correspondent. La correspondance entre deux enregistrements est exprimée par le satisfaction des contraintes passées en argument de l'opérateur.

Les contraintes sont dans la plupart des cas l'égalité de deux attributs, le plus souvent entre la clé primaire de l'une des tables et une clé étrangère y faisant référence dans la seconde table.

3.2.2 Jointure interne

La jointure interne est une opération prenant deux relations et une condition en argument et qui produit une relation dont le schéma est la concaténation des schémas des deux relations (comme pour le produit cartésien) et dont les enregistrements sont les concaténations des tuples des deux relations qui satisfont la condition.

Réalisation SQL :

```sql
SELECT ... FROM R JOIN R2 ON C ...
```

La condition C s'écrit de la même manière que les conditions de la clause `WHERE`.

Exemple : on dispose des tables suivantes :

Document(idDoc : entier, titre : texte, auteur : texte, genre : enum(...))

Personne(id : entier, nom : texte, prénom : texte)

Emprunt(id : entier, idDoc : entier, dateEmprunt : date, dateRetour : date)

On souhaite récupérer les noms et prénoms des emprunteurs avec les dates de retour associées :

```sql
SELECT nom, prenom, dateRetour
FROM Personne JOIN Emprunt ON Personne.id = Emprunt.id;
```

On peut enchaîner les jointures (en pratique la jointure de deux tables sert d'argument à la jointure suivante). Il faut donc une condition par jointure.

Exemple : on veut les mêmes informations qu'avant et en plus le titre du document, et seulement pour les retour en retard.

```sql
SELECT nom, prenom, dateRetour, titre
FROM Personne
    JOIN Emprunt ON Personne.id = Emprunt.id
    JOIN Document ON Document.idDoc = Emprunt.idDoc
WHERE dateRetour < [date du jour];
```

Pourquoi un opérateur de jointure alors qu'on peut l'implémenter avec un produit cartésien ?

– A *priori*, les SGBD peuvent optimiser les jointures alors qu'il faut construire tous les tuples du produit cartésien avant de faire la sélection.

– Une condition de jointure est une contrainte structurelle exprimant les associations entre plusieurs tables alors qu'une condition de sélection sert plutôt à filtrer les enregistrements pertinents pour la requête : la logique est différente.

– On gagne en lisibilité en séparant les deux types de condition et en séparant les conditions associées à chaque jointure.

Remarque : la *jointure interne* est aussi simplement appelée *jointure* : c'est le type de jointure par défaut.

3.2.3 Jointure externe à gauche

Une *jointure externe à gauche* fonctionne comme une jointure interne, mais les enregistrements de la première relation pour lesquels il n'existe aucun enregistrement dans la seconde relation tel que leur concaténation satisfait la condition de jointure sont conservés. Pour respecter le schéma du résultat (qui est la concaténation des schémas des deux tables), on associe à ces enregistrements la valeur `NULL` pour chaque attribut qui provient de la seconde table.

Réalisation SQL : comme pour la jointure interne en remplaçant `JOIN` par `LEFT JOIN`.

Exemple : noms, prénoms des emprunteurs associés aux dates de retour, s'il y en a :

```sql
SELECT nom, prenom, dateRetour
FROM Personne LEFT JOIN Emprunt ON Personne.id = Emprunt.id;
```

Personne :

idAlice	Alice	Dupond
idBob	Bob	Dupont

Emprunt :

idAlice	2022-05-04	idDoc1
idAlice	2022-05-05	idDoc2

Résultat :

Alice	Dupond	2022-05-04
Alice	Dupond	2022-05-05
Bob	Dupont	NULL

Remarque : il existe d'autres types de jointure (H.P) :

– Externe à droite (`RIGHT JOIN`)

– naturelle (`NATURAL JOIN`)

– totale (`TOTAL JOIN`)

3.3 Agrégation et imbrication

3.3.1 Application de fonctions aux attributs

Il est possible d'appliquer des fonctions aux attributs des tuples sélectionnés par une requête pour calculer de nouvelles valeurs.

Sont au programme : `+, -, *, /`.

Exemple : la durée restante avant le retour pour les documents empruntés :

```
SELECT idDoc, dateRetour - '2022-05-09' AS delai FROM Emprunt;
```

Remarque : ces fonctions permettent d'obtenir un résultat pour chaque tuple sélectionné en fonction des valeurs de ses attributs. On peut vouloir au contraire travailler sur une colonne, *i.e* sur l'ensemble des valeurs prises par un attribut pour tous les enregistrements sélectionnés : c'est le rôle des fonctions agrégatives.

3.3.2 Fonctions d'agrégation

Une *fonction d'agrégation* ou *fonction agrégative* est une fonction qui s'applique à l'ensemble des valeurs d'un attribut donné pour tous les enregistrements sélectionnés. Une telle fonction renvoie en général un unique résultat donc la table créée par la requête ne contient qu'une seule ligne.

Sont au programme : `MIN, MAX, SUM, AVG, COUNT` (cardinal du multi-ensemble des valeurs privé de `NULL`)

Exemple : nombre de documents empruntés :

```
SELECT COUNT(idDoc) FROM Emprunt;
```

Remarque : ici, le choix de l'attribut n'est pas important tant que c'est une clé car on veut calculer le nombre d'enregistrements. On dispose de la syntaxe `COUNT(*)` pour cela.

Pour connaître le nombre d'auteurs dont une œuvre est dans la bibliothèque, on peut avoir des doublons que l'on veut éliminer avant comptage : on place le mot clé `DISTINCT` dans l'argument de la fonction `COUNT` :

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

3.3.3 Regroupement d'enregistrements

On peut réunir dans un groupe les enregistrements d'une table qui coïncident sur un ensemble d'attributs. On obtient dans le résultat de la requête un tuple par groupe.

Réalisation SQL :

```
SELECT ... GROUP BY attribut_1, ... attribut_n;
```

Exemple :

```
SELECT auteur FROM Document
GROUP BY auteur;
```

équivalent à :

```sql
SELECT DISTINCT auteur FROM Document;
```

On peut bien sûr projeter sur plusieurs attributs, mais cela n'a pas de sens si ces attributs n'ont pas la même valeur pour tous les tuples d'un groupe.

Exemple :

```sql
SELECT auteur, titre FROM Document
GROUP BY auteur;
```

Ici, MySQL conserve le titre du premier enregistrement traité de chaque groupe.

En pratique, le regroupement perd tout son intérêt s'il est combiné avec les fonctions agrégatives car elles permettent d'obtenir une valeur unique pour tous les enregistrements d'un groupe.

Exemple : nombre d'ouvrages dans la bibliothèque pour chaque auteur :

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

Nombre d'ouvrages empruntés pour chaque usager :

```sql
SELECT Personne.id, COUNT(idDoc)
FROM Personne LEFT JOIN Emprunt ON Personne.id = Emprunt.id
GROUP BY Personne.id;
```

Nombre d'homonymes parmi les usagers :

```sql
SELECT nom, prenom COUNT(*) FROM Personne
GROUP BY nom, prenom;
```

3.3.4 Filtrage des agrégats

On peut vouloir sélectionner certains enregistrements après regroupement et / ou après application d'une fonction d'agrégation. Par exemple, on veut garder les usagers qui ont empruntés au moins 10 documents.

Réalisation SQL : on utilise le mot clé `HAVING`, qui s'utilise comme `WHERE`, mais qui sélectionne les tuples après regroupement contrairement à `WHERE` qui les sélectionne avant.

```sql
SELECT ... HAVING condition;
```

La condition de la clause `HAVING` ne peut porter que sur des caractéristiques du groupe, *i.e* sur des attributs sur lesquelles tous les tuples du groupe coïncident ou sur le résultat de l'application d'une fonction agrégative sur les tuples du groupe.

Exemple :

```sql
SELECT id, COUNT(*) FROM Emprunt
GROUP BY id
HAVING COUNT(*) >= 10;
```

Usagers ayant au moins deux documents en retard :

```sql
SELECT id FROM Emprunt
WHERE dateRetour < '2022-05-09'
GROUP BY id
HAVING COUNT(*) >= 2;
```

Si on veut connaître les usagers ayant le plus de documents en retard ? Ici, on veut pouvoir réutiliser le résultat d'une requête pour faire une autre requête.

3.3.5 Requêtes imbriquées

Une *sous-requête* est une requête imbriquée dans une autre. On parle aussi de *requête interne*, par opposition à la *requête externe*, ou *englobante* qui la contient. Une sous-requête doit être écrite entre parenthèses et peut être placée à divers endroits.

• Sous-requête dans une clause `FROM` :

Exemple : nombre moyen de documents en retard par usager :

```
SELECT AVG(nbreRetard)
FROM (
    SELECT COUNT(*) AS nbreRetard FROM Emprunt
    WHERE dateRetour < '2022-05-09'
    GROUP BY id
);
```

• Sous-requête dans une clause `WHERE` ou `HAVING` : l'usage du résultat de la sous-requête dépend de son nombre de lignes :

– Cas des sous-requêtes dont le résultat ne contient qu'une seule ligne : on utilise directement ce résultat comme on aurait utilisé les attributs d'un enregistrement donné.

Exemple : usagers ayant le plus de documents en retard :

```
SELECT id FROM (
    SELECT id, COUNT(*) AS nbRetard FROM Emprunt
    WHERE dateRetour < '2022-05-09'
    GROUP BY id
) AS Tretard
WHERE nbRetard = (SELECT MAX(nbRetard) FROM Tretard);
```

– Cas des sous-requêtes à plusieurs résultats : on ne peut pas utiliser les opérateurs habituels car la sous-requête renvoie une table et pas une unique valeur. On dispose d'opérateurs pour vérifier si cette table est vide ou pas (`EXISTS`, `NOT EXISTS`) et pour vérifier si une valeur apparaît dans la table (`IN`, `NOT IN`).

Exemple : usagers sans emprunts en cours :

```
SELECT id FROM Personne
WHERE id NOT IN (
    SELECT id FROM Emprunt
);
```

ou

```
SELECT id FROM Personne
WHERE NOT EXISTS (
    SELECT id FROM Emprunt
    WHERE Personne.id = Emprunt.id
);
```

Usagers ne partageant pas leur nom de famille avec d'autres :

```sql
SELECT id FROM Personne
WHERE NOT EXISTS (
    SELECT id FROM Personne AS P
    WHERE P.id <> Personne =.id AND P.nom = Personne.nom
);
```