Introduction aux Bases de données

Plan du cours

- 1. Introduction aux bases de données
  - Caractéristiques des bases de données (BD)
  - Les Systèmes de Gestion de BD (SGBD)
- 2. Le modèle entité-association
- 3. Le modèle relationnel
  - Structure
  - Un langage de requêtes : l'algèbre relationnel
  - Contraintes
- 4. Implémentation du modèle relationnel
  - SQL : un langage de requêtes
  - Optimisation de requêtes SQL

Objectifs de la formation

- Comprendre le modèle relationnel
- Savoir faire des requêtes SQL
- Comprendre les étapes du traitement d'une requête SQL
- Savoir créer une base depuis un énoncé

Partie I
Introduction aux bases de données

Définitions

- **Base de données (BDD)** :
  - collection d'informations ou de données qui existent sur une longue période de temps et qui décrivent les activités d'une ou plusieurs organisations
  - ensemble de données modélisant les objets d'une partie du monde réel et servant de support à une application informatique
- **SGBD** : Système de Gestion de Bases de Données (***DataBase Management System - DBMS***)
  - **Ensemble de logiciels systèmes** permettant aux utilisateurs de gérer (insérer, modifier, supprimer et rechercher efficacement des données spécifiques) **une grande masse d'informations** partagée par de multiples utilisateurs

Définitions

- Connectés à de nombreux domaines, des plus théoriques (Mathématiques discrètes, Logiques du premier ordre) aux plus pratiques (systèmes d'exploitation, réseaux, aspects méthodologiques)
- Présentes dans la plupart des domaines d'application de l'informatique : informatique de gestion, systèmes industriels, applications scientifiques . . .

Définitions

- Une base de données est un gros ensemble d'informations structurées mémorisées sur un support permanent et qui peut être partagée par plusieurs applications et qui est interrrogeable par le contenu.
- vs. Ensemble de fichiers
  - Lourdeur d'accès aux données (connaître le détail de l'implantation physique des fichiers)
  - Manque de sécurité (tout le monde peut modifier le fichier)
  - Pas de contrôle de concurrence entre utilisateurs

- Base de données Database
  - Grande masse de données
  - Mémoire secondaire (disque dur, disque optique)
  - Liens abstraits/thématiques entre données
  - Longévité des données (des dizaines d'années)
- SGBD Database Management System (DBMS)
  - Gestion du stockage des données
  - Traitement des données
  - Modification
  - Interrogation, extraction
  - Programmation d'applications

Caractéristiques

- Volume des données à manipuler
  - il n'est pas rare d'avoir des relations de plusieurs millions de lignes (enregistrements)
  - les accès concurrents aux données, par exemple quand des milliers d'utilisateurs accèdent en même temps aux mêmes données
  - la complexité de la structure des données, de quelques dizaines à plusieurs centaines de relations (tables)
- Facilité d'interrogation des données
  - Pas besoin d'écrire un programme, il existe des langages de requêtes déclaratifs

Caractéristiques

- Une BDD est composée de:
  - De Données
    - Structure intégrée
      - Toute donnée n'est définie qu'une fois
        - Pas de copies inutiles
        - Données hétérogènes
      - Ex. le prix est défini une fois HT et en €
    - Liens entre les tables
    - Contraintes d'intégrité
    - Contraintes de sécurité
  - De vues de la base

Caractéristiques

- Une base de données supporte:
  - De la recherche de données
    - Interactive ou imbriquée
    - Performante
    - Sécurisée
  - De l'insertion, de la mise à jour, de la suppression de données
    - Cohérente
    - Partagée
    - Fiable
    - sûre

Pourquoi une base de données

- Intégration de données
  - Moins de duplications
- Partage de données
- Fiabilité de données
  - Transactions, Reprises sur pannes, Tolérance de pannes
- Sécurité de données
- Langages assertionnels de requêtes
  - SQL par exemple
- Interfaces conviviales

Qu'est ce qu'un SGBD?

- Un SGBD est un logiciel qui fournit un ensemble de services :
  - Langage de définition de données (LDD)
  - Langage de manipulation de données (LMD)
  - Réponse "rapide" aux requêtes
  - Gestion économe de l'espace de stockage mémoire
  - Cohérence et intégrité des données
  - Contrôle de la concurrence, du partage des données
- Qui utilise les SGBD?
  - Les concepteurs de base de données
  - Les utilisateurs via des programmes dédiés ou via SQL
  - Les programmeurs d'applications
  - Les administrateurs

Objectifs

- Indépendance physique des programmes et des données :
  - Pouvoir modifier les schémas internes sans modifier les schémas conceptuels et externes
- Indépendance logique des programmes et des données :
  - Pouvoir modifier les schémas externes sans modifier les schémas conceptuels
  - Indépendance entre les différents utilisateurs
- Manipulation des données par des langages non procéduraux
  - Données facilement manipulables par les utilisateurs (interactifs ou programmeurs)
- Administration facile des données
  - Outils pour définir et modifier les définition de données

Objectifs

- Efficacité d'accès aux données :
  - Optimisation : temps de réponse, débit, ...
  - Optimisation des opérations d'E/S
- Redondance contrôlée des données :
  - Dans les BD réparties : redondance nécessaire, mais contrôlée
- Cohérence des données
  - Satisfaction de contraintes d'intégrité
- Partage des données
  - Permettre les accès concurrents
- Sécurité des données
  - Outils pour définir et modifier les définition de données
  - Protection en cas de panne (du SGBD, de la machine, ...)
  - Assurer l'atomicité des transactions et l'intégrité des données

Fonctionnalités

1. Persistance
   - Données stockées sur disque
2. Gestion du disque
   - Techniques spécifiques pour de bonnes performances
     - Index, hash-coding
     - Regroupement des données sur disque
     - Optimisation des requêtes
     - Cache-mémoire
3. Partage des données
   - Chaque utilisateur doit avoir l'illusion d'être seul
     - Notion de transaction (begin, abort, commit)
   - Cohérence des mises à jour effectuées par un utilisateur
   - Cohérence collective : sérialisabilité

Fonctionnalités

4. Fiabilité des données
   - Vérification de contraintes d'intégrité
   - Atomiticité des transactions : transaction complètement effectuée ou pas du tout
   - Résistance aux pannes
     - Si panne mémoire : restauration automatique de la base intégrant les dernières transactions validées avant la panne
     - Si panne disque : restauration d'une sauvegarde et déroulement du journal archivé
     - Mise en place d'un mécanisme de réplication synchrone de la base dans une base miroir : mirroring
5. Sécurité des données
   - Tous les utilisateurs ne peuvent pas tout faire sur toutes les données
     - Notion de groupes d'utilisateurs
     - Notion d'autorisation (lecture, écriture, exécution)
     - Granularité des autorisations : base de données, table, colonne, n-uplet, ...
     - Possibilité d'accorder ou de supprimer des droits

Fonctionnalités

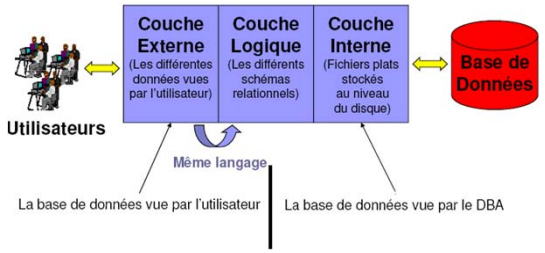
6. Indépendance logique / physique
   - Organisation physique de la BD transparente pour le développeur d'application
   - On doit pouvoir changer la répartition des données sur le disque (ex : un regroupement ou la pose d'un index) sans changer le code de l'application manipulant les données
7. Langage de requête
   - Les requêtes doivent être :
     - Simples
     - Déclaratives
     - Optimisées avant leur exécution

   → SQL

Architecture des SGBD

- **Couche interne ou physique** :
  - plus bas niveau
  - indique **comment** (avec quelles structures de données) sont stockées physiquement les données (sous forme de fichiers propriétaires)
- **Couche logique ou conceptuel** :
  - décrite par un schéma conceptuel
  - indique quelles sont les données stockées et quelles sont leurs relations **indépendamment de l'implantation physique**
- **Couche externe ou vue** :
  - **propre à chaque utilisateur**
  - décrit par un ou plusieurs schémas externes

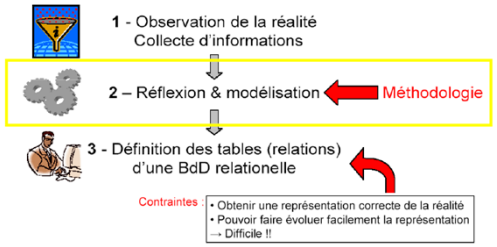
Architecture des SGBD



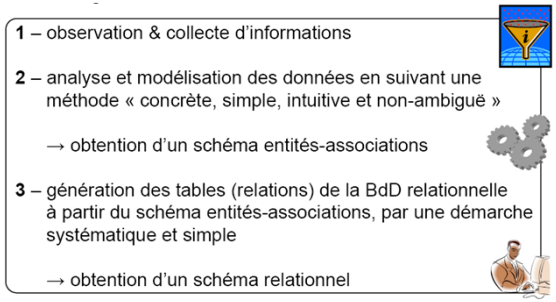
Modèles de données

- Ensemble de concepts permettant de décrire les données, leurs liens,leurs sémantiques, et les contraintes d'intégrité.
- **Modèle de type conceptuel** : sous forme de graphique,utilisés pour l'analyse d'applications.
  - Standard : le modèle entité-association
  - Concepts : entités, associations, spécialisation, attributs,identificateurs, etc.
- **Modèle de type logique** : traduction du modèle conceptuel,il est associé au SGBD.
  - Standard : le modèle relationnel
  - Concepts : relations, attributs, domaines, clés, n-uplets, etc.

Modélisation d'une BD



Objectifs et démarches



Objectifs et démarche

- **Pourquoi pas une seule table ?**
- **Exemple** : bibliothèque Bibli(Titre, Auteur, Nom, Prénom, Adresse, Date)
- **Problèmes** :
  - Un livre peut avoir plusieurs auteurs …. Que faire ?
  - Duplication de données, ex : adresse d'une personne empruntant plusieurs livres.
  - Comment conserver un client qui rend son dernier livre ?
  - …
- **Bilan** : Sémantique des données très mal représentée ! (emprunteurs et livres non distincts)
  ➔ il faut définir plusieurs tables/relations

Partie II
Le modèle Entité/Association (E/A)

Définitions

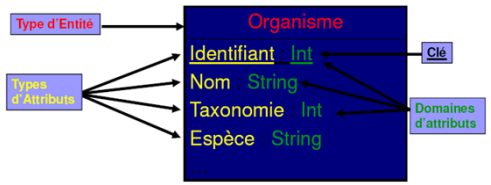
- **Entités** : « objets concrets ou abstraits » provenant de l'observation du monde réel. Possède un nom : Organisme, Gène, …
  → objet que l'on peut distinguer
- **Occurrence** : instanciation d'une entité : Bactérie
  → exemple d'une entité
- **Attribut** : propriété d'une entité (nom, espèce, chromosome, ...), il possède un domaine de valeurs.
  → information qui décrit l'entité

Définitions

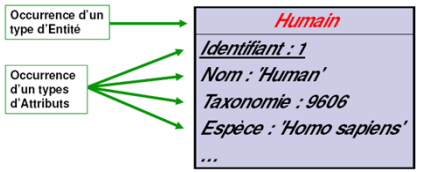
- **Clé/Identifiant** : ensemble minimum d'attributs dont les valeurs identifient de façon unique chaque occurrence de l'entité
  → Une entité ne peut pas avoir deux occurrences ayant la même clé
- **Association** : permet de connecter 2 entités ou plus

Exemple d'entité

- Un modèle conceptuel

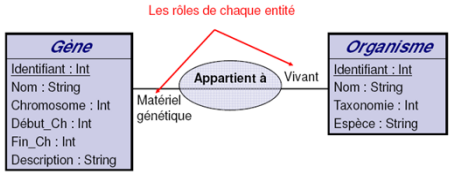


Exemple d'occurence



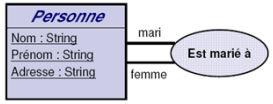
Définitions

- **Association** : relation entre ***plusieurs*** entités.
  - 1, 2, ou plus de 2 entités concernées,
  - 2 ou plus de 2 occurrences concernées.
  - Possède un nom : « appartient », « possède », …
  - Peut posséder des attributs.



Définitions

- **Association Cyclique (ou réflexive)**
  - Lorsque 2 rôles relient le même type d'entité
  - Comment savoir qui est le mari et qui est la femme ?
    - (Dominique Petit, Dominique Petit)
    - Spécifier les rôles pour éviter les ambiguïtés

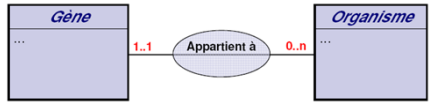


Définitions

- **Cardinalité :** Compte le nombre de fois (min et max) où l'entité peut se retrouver engagée dans l'association
- Cardinalité minimum :
  - 0 : peut ne pas être engagée
  - 1 : doit être engagée au moins une fois
- Cardinalité maximum
  - 1 : ne peut pas être engagée plus d'une fois
  - n : peut être engagée plus d'une fois
- Représentation : Min..Max ou Min:Max

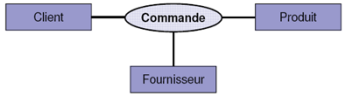
Définitions

- **Cardinalités** : Exemple
  - Un gène appartient à un et un seul organisme
  - Les paralogues ?
  - Un organisme possède 0 ou plusieurs gènes ?



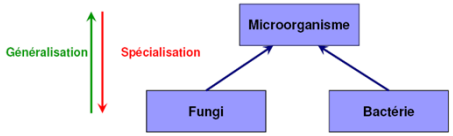
Définitions

- **Association Ternaire**
  - Lorsque une association se fait entre 3 entités
  - 3 rôles sont obligatoires



Définitions

- **Spécialisation** : consiste à représenter une nouvelle entité plus spécifique
- **Généralisation** : consiste à représenter une nouvelle entité qui factorise les propriétés communes aux différentes entités



Définitions

Attributs

- **simple (atomique)**: non décomposable
  - Exemples: jour, prénom
    - Le domaine de valeurs est constitué de valeurs atomiques
  - Ex.: jour - domaine de valeurs: {1, 2, …., 31}
    - Domaines prédéfinis standard, intervalles, énumérés
- **Complexe**: décomposé en d'autres attributs
  - Exemples: date (jour, mois, année), adresse (rue, ville, code postal)
  - Un attribut complexe ne porte pas de valeur propre (pas de domaine directement associé)
  - La valeur d'un attribut complexe est la composition des valeurs de ses attributs composants.
  - Un composant d'attribut complexe peut être lui-même un attribut complexe.

Définitions

Attributs

- **Monovalué**: une seule valeur par occurrence (cardinalité max=1)
  - Exemples: date de naissance, numéro Sécu Sociale
- **Multivalué**: plusieurs valeurs par occurrence (cardinalité max>1).
  - Exemples: prénoms, téléphones
    - Une valeur d'attribut multivalué est un ensemble (ou liste ou multi ensemble) de valeurs, prises chacune dans le domaine de valeurs associé à l'attribut.

Définitions

Attributs

- **Obligatoire**: une valeur au moins par occurrence (cardinalité min>=1).
  - Exemples: nom, prénom
- **Facultatif**: peut ne pas prendre de valeur (cardinalité min=0).
  - Exemples: salaire, téléphone

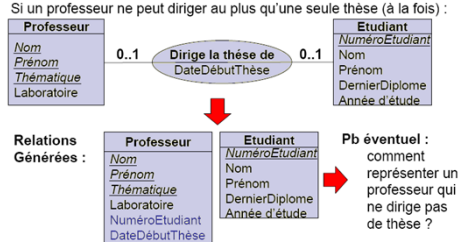
Identifiant (clé)

- ensemble d'attributs qui donne à l'entité son caractère unique
- obligatoire pour éviter la redondance des données ou l'erreur de saisie

Passage aux relations

• E1 0..1 - 0..1 E2 :
ajout à une entité (E1 ou E2) de la clé de l'autre et des attributs de l'association.

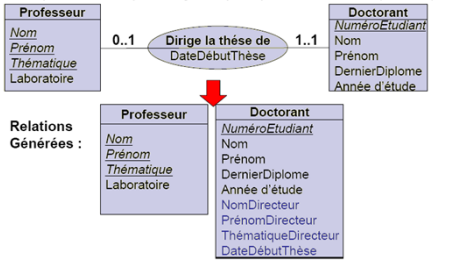
Si un professeur ne peut diriger au plus qu'une seule thèse (à la fois) :



Passage aux relations

• E1 0..1 - 1..1 E2 :
ajout à E2 de l'identifiant de E1 et des attributs de l'association.

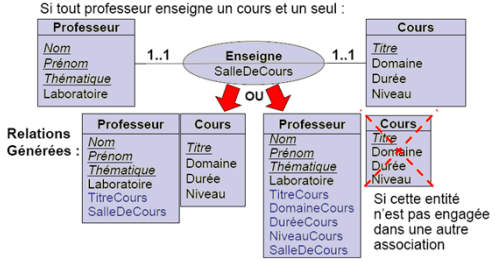
Si un professeur ne peut diriger au plus qu'une seule thèse (à la fois) :



Passage aux relations

• E1 1..1 - 1..1 E2 :
ajout à une entité (E1 ou E2) de l'identifiant de l'autre et des attributs de l'association. Possibilité de ne faire qu'une seule relation.

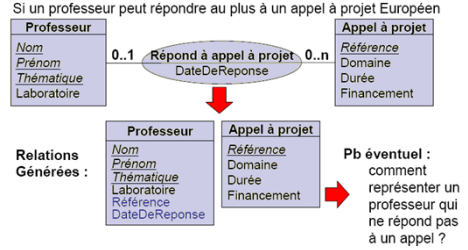
Si tout professeur enseigne un cours et un seul :



Passage aux relations

• E1 x..1 - y..n E2 :
ajout à E1 de l'identifiant de E2 ainsi que des attributs de l'association.

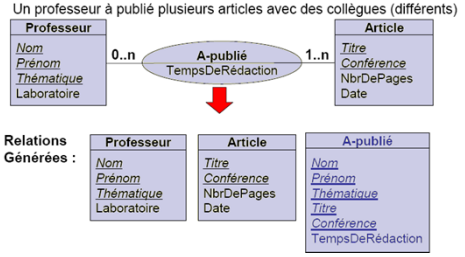
Si un professeur peut répondre au plus à un appel à projet Européen



Passage aux relations

• E1 x..n - y..n E2 :
l'association devient une nouvelle relation (3 relations au final).

Un professeur à publié plusieurs articles avec des collègues (différents)



Passage aux relations

- Dans certains modèles entités-associations on définit l'association particulière « est_un » (*isa*).
  - Ex : Professeur « est_un » Personne
  - Etudiants sont des « est_un » Personne
- C'est une sorte d'héritage
- Dans ce cas les entités Professeur et Etudiants n'ont pas de clé dans le modèle entités-associations, et vont hériter d'une clé.
- « E1 est_un E2 » : se traduit en modèle relationnel par l'ajout de la clé de E2 dans la relation E1

Introduction de clés numériques

- Quand la clé est constituée de nombreux attributs il peut être efficace d'introduire une clé (artificielle) numérique.
- Ex : on veut identifier les employés d'une société
  - Idée 1 : clé = numéro de sécurité sociale

  →Identifiant unique pour une personne : parfait

  → Mais la CNIL l'interdit !!!
  - Idée 2 : clé = {nom, prénom, adresse, …}

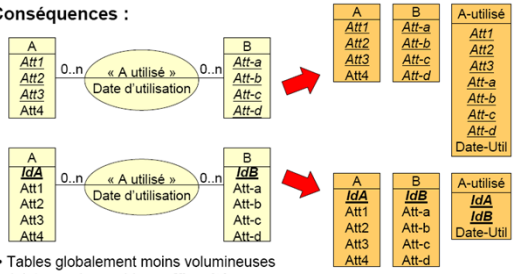
  →Assez d'attribut pour éviter les confusions

  →Mais les associations vont générer des tables avec bcp d'attributs (on reprend les clés des entités)
  - Idée 3 : **clé = numéro d'employé (créé spécialement)**

  →n° gérés par la base (incrément automatique)

Introduction de clés numériques

Conséquences :



- Tables globalement moins volumineuses
- Jointures plus rapides : **efficacité accrue**

Partie III
Le modèle relationnel

Le modèle relationnel

- Le modèle relationnel est une manière de modéliser les informations contenues dans une base de données
- Structure
  - Donne l'organisation des données du modèle, c'est-à-dire sous forme de tableaux à deux dimensions (LDD)

Le modèle relationnel

- Langages
  - Il existe trois principaux langages de requêtes relationnels :
    - Algèbre relationnel
    - Calcul relationnel
    - Datalog Non-recursif : langage à base de règles
  - Ce sont des langages déclaratifs qui ont donné lieu au langage SQL utilisé en pratique.
- Contraintes : basées sur différents types de dépendances
  - Restreindre les relations autorisées dans une base de données pour satisfaire certaines conditions logiques, appelées contraintes d'intégrité.

Structure du modèle

- Les données sont structurées en relation (ou table ou tableau à deux dimensions)
  - les colonnes donnent les caractéristiques de ce que la relation représente
  - les lignes donnent les "individus" qui peuplent la relation
- La première ligne donne les noms donnés aux différentes caractéristiques, appelés attributs.
- Ex: gènes

Nom	Chromosome	Début	Fin	Descr
TNFRSF18	1	1044947	1048147	Tumor ...
CAB45	1	1058370	1073469	45 kDa ...
FGFA_RAT	2	50438931	50445674	null

Attributs et domaines

- Soit U un ensemble infini dénombrable de noms d'attributs ou attributs, nommé univers.
  - Exemple : U = {nom, chrosome,debut,fin,descr}
- Soit D un ensemble infini dénombrable de valeurs constantes.
- D représente l'ensemble des valeurs qui peuvent être prises dans une base de données.
- Pour A Є U, on note dom(A) le domaine de A, càd le sous ensemble de D dans lequel A peut prendre des valeurs (dom(A) $\subseteq$ D).
  - Exemple : dom(chromosome) = {1,2}

Schéma de relation

- Un schéma de relation est un symbole de relation R auquel on associe un entier naturel type(R), qui donne le nombre d'attributs de R.
- A chaque schéma, on associe une fonction att définie de {1, . . . , type(R)} dans U qui permet d'associer des attributs de U à R dans un certain ordre.
- On note schema(R) = {att(1), . . . , att(type(R))} l'ensemble des attributs de R.

Schéma de relation

Exemple:

- Soit Gene un schéma de relation avec
  - type(Gene) = 5 et schema(Gene) = {att(1), att(2), att(3), att(4), att(5)} où att(1) = nom, att(2) = chromosome, att(3) = debut, att(4) = fin, att(5) = descr
- Un schéma de relation R est en première forme normale (1FN) si tous les domaines des attributs de R ont des valeurs constantes et atomiques.

Tuples et relations

- Soit R un schéma de relation avec
  - schema(R) = $\{A_1, \ldots, A_m\}$, att(i) = A_i, i = 1,m
- Un tuple sur R est un élément du produit cartésien $dom(A_1) \times \ldots \times dom(A_m)$
- Une relation sur R est un ensemble fini de tuples sur R.
  - Exemple : La relation Genes est une relation sur le schéma de relation Gene.
    <CAB45, 1,1058370,1073469,45kDa> Є Gene

Tuples et relation

- Le domaine actif d'un attribut A Є schema(R) dans r, noté ADOM(A, r), est l'ensemble des valeurs constantes prises par A dans r, c'est-à-dire:
  ADOM(A, r) = $\pi_A(r)$.
- Le domaine actif d'une relation r, noté ADOM(r), est défini par:
  ADOM(r) =$U_{A \in schema(R)}$ ADOM(A, r).

Manipulation sur un tuple

- Soient R un schéma de relation avec schema(R) = {A_1, . . . ,A_m}, att(i) = A_i(i = 1,m), r une relation sur R, t un tuple de r et A_i Є schema(R).
- La projection d'un tuple t sur A_i, notée t[A_i], est la $i^{ème}$ coordonnée de t, c'est-à-dire t(i).
  → se généralise à plusieurs attributs
- Soit X = {att(i_1), . . . , att(i_n)} ⊆ schema(R). La projection de t sur X, notée t[X], est définie par t[X] =< t(i_1), . . . , t(i_n) >

Manipulations sur un tuple

- Exemple:
- Soit t =< CAB45, 1,1058370,1073469,45kDa >.
- On a :
  - t[nom] = t[att(1)] = t(1) =< CAB45 >
  - t[nom, chromosome]= t[att(1), att(2)]
    =< t(1), t(2) >
    =< CAB45 , 1 >

Schéma de base de données

- Un schéma de base de données R est un ensemble de schémas de relation.
- Soit R = {R_1, . . . ,R_n}. On a :
  - schema(R) = $U_{R_i \in R}$ schema(R_i)
  - Un schéma de BD est en 1FN si ses schémas de relation sont en 1FN.
- Importance des hypothèses de nommage, par exemple :
- Universal Relation Schema Assumption (URSA) : Si un attribut apparaît dans plusieurs schémas de relation, alors cet attribut représente les mêmes données

Schéma de base de données

- Exemple:
- Soit R = { Personne, Departement, Travaille } avec
  - schema(Personne)={ nss, nom, prenom, age },
  - schema(Departement)={dep, adresse },
  - schema(Travaille)={ nss, dep, activite }

Bases de données

- Une base de données d est définie sur un schéma de base de données et représente un ensemble de relations.
- Soit R = {R_1, . . . ,R_n}. d = {r_1, . . . , r_n} est une base de données sur R, r_i définie sur R_i (i=1,n)
- Le domaine actif d'une base de données d , noté ADOM(d), est l'union des domaines actifs de ses relations.

Bases de données

- Soit d = { Personnes, Departements, Travaillent } avec *Personnes, Departements, Travaillent* définies sur *Personne, Departement, Travaille* respectivement

Personnes

nss	nom	prenom	age
12	Reichstadt	Matthieu	29
45	Claude	Alexandre	32

departements

dep	adresse
Math	Carnot
info	cezeaux

travaillent

nss	dep	Activite
12	Math	Prof
45	Math	MdC
45	info	MdC

L'algèbre relationnel

- collection d'opérateurs algébriques
  - Entrée : une ou deux relations
  - Sortie : une relation
- Possibilité de composition des opérateurs (propriété de fermeture de l'algèbre relationnel)
- Requête relationnelle : composition d'un nombre fini d'opérateurs algébriques
- L'ordre d'évaluation des opérateurs est spécifié dans la requête
  →à la base de l'optimisation de requêtes

Opérateurs de l'algèbre relationnel

- Opérations ensemblistes
  - Union (U) : sélection des tuples d'une relation r_1 et de ceux d'une autre relation r_2
  - Intersection (∩)
  - Différence (−) : sélection des tuples d'une relation r_1 qui n'appartiennent pas à une relation r_2
- Autres opérations
  - Projection (π) : sélection des attributs d'une relation
  - Selection (σ) : sélection d'un sous ensemble de tuples d'une relation.
  - Jointure (⋈) : combine deux relations
  - Renommage ($\sigma_{A \to B}$) : renomme le nom d'un attribut
  - Produit cartésien (×)
  - Division (÷)

Opérateurs de l'algèbre relationnel

- Certains opérateurs sont indispensables, d'autres pas, càd ils peuvent s'exprimer à partir des autres.
- Par exemple, on peut exprimer
  - une jointure ⋈ avec ×, ρ, π, σ,
  - un produit cartésien × avec ⋈, ρ
  - l'intersection ∩ avec −

La projection

- Soient r une relation sur R et Y ⊆ schema(R).
- La projection de r sur Y, notée $\pi_Y(r)$, est définie par :
  - $\pi_Y(r) = \{t[Y] \mid t \in r\}$
- Soit S le schéma de relation associé à $\pi_Y(r)$. On a schema(S) = Y
- Exercice:
  - π_{nom}(Personnes) =?
  - $\pi_{\{nom,prenom\}}$ (Personnes) =?
  - π_{dep}(Travaillent) =?

  → correspond à une coupe verticale de la relation

La sélection

→ va correspondre à une coupe horizontale de la relation
Nécessite tout d'abord de définir la notion de formule de sélection puis la satisfaction d'une formule de sélection par un tuple.

- Une formule de sélection simple sur R est une expression de la forme :
- A = a ou A = B, où A,B ∈ schema(R) et a ∈ dom(A)
- Une formule de sélection est une expression composée de formules de sélection simples connectées à l'aide des connecteurs logiques ∨,∧,¬ et des parenthèses.

La sélection

- Soient r une relation sur R, t Є r et F une formule de sélection. t satisfait F, noté t ⊨ F, est défini récursivement par :
  - 1. t ⊨ A = a si t[A] = a
  - 2. t ⊨ A = B si t[A] = t[B]
  - 3. t ⊨ F1 ∧ F2 si t ⊨ F1 et t ⊨ F2
  - 4. t ⊨ F1 ∨ F2 si t ⊨ F1 ou t ⊨ F2
  - 5. t ⊨ ¬F si t ⊭ F
- Soient r une relation sur R et F une formule de sélection.
- La sélection des tuples de r par rapport à F, notée F(r), est définie par : $\sigma_{F(r)} = \{t \in r \mid t \models F\}$
- Exercice: $\sigma_{\text{Activite='Prof'}}$ (travaillent) =?

Opérations ensemblistes

- Soient r_1 et r_2 deux relations sur le même schéma de relation R
  - Union : $r_1 \cup r_2 = \{t \mid t \in r_1 \text{ ou } t \in r_2\}$
  - Différence : $r_1 - r_2 = \{t \mid t \in r_1 \text{ et } t \notin r_2\}$
  - Intersection : $r_1 \cap r_2 = \{t \mid t \in r_1 \text{ et } t \in r_2\}$
- Exercice: Donner $r_1 \cup r_2$, $r_1 \cap r_2$ et $r_1 - r_2$.

r_1

A	B	C
1	2	3
1	1	1
1	2	2

r_2

A	B	C
2	2	2
1	1	1

Quelques propriétés

- $r_1 \cap r_2 = r_1 - (r_1 - r_2)$
- $\sigma_{\neg F}(r) = r - \sigma_F(r)$
- $\sigma_{F1 \vee F2}(r) = \sigma_{F1}(r) \cup \sigma_{F2}(r)$
- $\sigma_{F1 \wedge F2}(r) = \sigma_{F1}(r) \cap \sigma_{F2}(r)$

Jointure naturelle

- Soient r_1 et r_2 deux relations sur R_1 et R_2 respectivement.
- La jointure naturelle de r_1 et r_2, notée $r_1 \bowtie r_2$, est une relation sur un schéma de relation R, avec schema(R) = schema(R_1) U schema(R_2), définie par :
- $r_1 \bowtie r_2 = \{t \mid \exists t_1 \in r_1 \; \exists t_2 \in r_2 \text{ tq } t[\text{schema}(R_1)] = t_1 \text{ et } t[\text{schema}(R_2)] = t_2\}$

Jointure naturelle

- Exercice: Donner s1 ⋈ s2, s1 ⋈ s3, s1 ⋈ s4.

s1

A	B	C
2	2	2
1	1	1

s2

A	D	E
1	2	3
1	1	2
2	3	1
3	1	2

s3

A	B	C
1	1	1
1	2	3

s4

D	E	F
1	1	1
2	2	2

- Si schema(R_1) = schema(R_2), que vaut $r_1 \bowtie r_2$?
- Si schema(R_1) ∩ schema(R_2) = 0, que vaut $r_1 \bowtie r_2$?

Un algorithme de jointure

- Soient r_1 et r_2 deux relations sur R_1 et R_2 respectivement et X = schema(R_1) ∩ schema(R_2)

Jointure(r_1, r_2)

```
1: Resultat = 0
2: for all t1 Є r1 do
3:   for all t2 Є r2 do
4:     if t1[X] = t2[X] then
5:       Construire un tuple t tq t[schema(R1)] = t1 et
         t[schema(R2)] = t2
6:       Resultat = Resultat U { t }
7: retourner Resultat
```

Le renommage

→ permet de forcer ou d'éviter des jointures naturelles

- Soit r une relation sur R, A Є schema(R) et B ∉ schema(R)
- Le renommage de A en B dans r, noté $\rho_{A \to B}(r)$, est une relation sur S avec schema(S) = (schema(R) − {A}) U {B} définie par :
- $\rho_{A \to B}(r) = \{t \mid \exists u \in r, t[schema(S) - \{B\}] = u[schema(R) - \{A\}]$ et $t[B] = u[A]\}$
- Exercice:
  - $\rho_{dep \to dep1}$(Departements) =?
  - Travaillent ⋈ $\rho_{dep \to dep1}$(Departements) =?

Les contraintes d'intégrité

- L'idée de base est de fournir des moyens qui permettent de restreindre les données "valides" d'une base de données
  - → on ne peut pas insérer tout et n'importe quoi
  - → La mise en oeuvre de cette notion se fait via des contraintes d'intégrité : "déclarations logiques qui permettent de restreindre le domaine actif d'une base de données"
- Exemple:
  - Un numéro de sécurité sociale ne peut pas être le même pour deux personnes différentes. → notion de clé primaire
  - Un employé ne peut pas être affecté à un numéro de projet qui n'existe pas → notion de clé étrangère

Dépendances fonctionnelles

- Concept très important
  → à la base de la théorie de la conception des bases de données.
- Intérêts pratiques :
  - permet de définir formellement la notion de bons schémas, càd de schémas de BD pour lesquels il n'existe pas d'anomalies lors de mises à jour (MAJ), càd lors d'insertions, de modifications, ou de suppressions de tuples.
  - généralise la notion de clé des SGBD (càd clé primaire, contrainte d'unicité).

Exemple

- Exemple: Soit U = {id, nom, adresse, cnum, desc, note} un univers décrivant des étudiants et des cours.
- Soit le schéma de BD suivant :
  - R_1 = {Donnees} avec schema(Donnees) = U
- Comment peut-on évaluer ce schéma de BD?
  Est-il bon?
  Pourquoi ?

Exemple

Données

id	nom	adresse	cnum	desc	note
124	Jean	Paris	F234	Info I	A
456	Matt	Lyon	F234	Info I	B
789	Ana	Clermont	M321	Analyse I	C
124	Jean	Paris	M321	Analyse I	A
789	ana	Clermont	CS24	BD I	B

Quels sont les problèmes ?

- l'information est redondante.
  - Ex : '124, Jean, Paris' apparaît dans deux lignes différentes
    → Anomalie de modification
  - Une modification sur une ligne peut nécessiter des modifications sur d'autres lignes

Exemple

- Certaines informations dépendent de l'existence d'autres informations.
  - Ex : Le cours 'CS24' de BD dépend de l'existence d'Ana.
    → Anomalie de suppression
  - Ex : Soit '145, Bob, Theix' un nouvel étudiant. On ne peut l'insérer que si l'on connait un de ses cours et sa note dans ce cours, à moins de permettre les valeurs nulles.
    → Anomalie d'insertion
- Le moyen simple qui permet d'éviter ces problèmes est l'étude de contraintes particulières, les dépendances fonctionnelles.

Syntaxe et sémantique des DF

- Définition (ou syntaxe)
  - Une DF sur un schéma R est une déclaration de la forme :
    R : X → Y , où X, Y Є schema(R)
- Satisfaction d'une DF (ou sémantique)
  - Une DF R : X → Y est satisfaite par une relation r sur R, noté r $\models$ X → Y , ssi $\forall t_1, t_2 \in r$ si $t_1[X] = t_2[X]$ alors $t_1[Y] = t_2[Y]$
- Exemple: Que pensez-vous de ces deux assertions ?
  Donnees $\models$ id → nom, adresse
  Donnees $\not\models$ id → note
- Comment construire une BD sur R_2 à partir de R_1?

Retour sur la notion de clé

→ Les clés sont des cas particuliers de dépendances fonctionnelles

- Soit F un ensemble de DF sur R.
- Superclé
  - Un ensemble d'attributs X $\subseteq$ schema(R) est une superclé de R par rapport à F si X → schema(R) peut se déduire de F
- Clé
  - Un ensemble d'attributs X Є schema(R) est une clé (ou clé minimale) pour F ssi X est une superclé de R et $\nexists$ Y $\subset$ X, Y superclé de R
- Autrement dit, si X est une clé de r, il ne peut pas exister deux tuples de r ayant la même valeur sur X.

Les dépendances d'inclusion (DI)

- Concept clé pour la conception de schémas de BDs
  → comparable aux DF, càd permet de dissocier les bons des mauvais schémas
- Les DF permettent de spécifier des contraintes intra-relations
  → les DI permettent de spécifier des contraintes inter-relations
- Les DF généralisent les clés
  →les DI généralisent les clés étrangères, aussi appelés contraintes d'intégrité référentielle

Retour sur la BD sur R_2

- En terme de mises à jour de la BD, on peut encore insérer des données non valides.
- Exemple : sans violer aucune DF, on peut donner une note à un étudiant qui n'existe pas et ce, dans un cours qui n'existe pas non plus ! ! !

  →l'idée est à nouveau de restreindre les insertions permises en spécifiant deux nouvelles contraintes : " une note ne peut être saisie que pour un étudiant qui existe dans la relation Etudiants et un cours qui existe dans la relation Cours"

Syntaxe des DI

- Definition: Soient R un schéma de base de données R_1 et R_2 deux schémas de relation de R.
- Une dépendance d'inclusion sur R est une déclaration de la forme $R_1[X] \subseteq R_2[Y]$, avec X et Y des séquences d'attributs distincts appartenant à schema(R_1) et schema(R_2) respectivement.
- Exemple:
  - Affectation[id] $\subseteq$ Etudiant[id]
  - Affectation[cnum] $\subseteq$ Cours[cnum]

Sémantique des DI

- Définition: Soient d une base de données sur R et r_1, r_2 Є d définies sur R_1, R_2 Є R respectivement.
- Une dépendance d'inclusion $R_1[X] \subseteq R_2[Y]$ est satisfaite par d, noté d $\models R_1[X] \subseteq R_2[Y]$, ssi $\forall t_1 \in r_1, \exists t_2 \in r_2$ tq $t_1[X] = t_2[Y]$
- Exemple: Que pensez vous des assertions suivantes ?
  - { Affectations, Etudiants } $\models$ Affectation[id] $\subseteq$ Etudiant[id]
  - { Affectations, Cours } $\models$ Affectation[cnum] $\subseteq$ Cours[cnum]

  →Les clés étrangères sont des cas particuliers de dépendances d'inclusion, càd une clé étrangère est une partie gauche d'une DI dont la partie droite est une clé.

Partie IV
Implémentation du modèle relationnel

SQL – modèle relationnel

→ on se focalise sur l'implantation de l'algèbre relationnel dans un langage pratique nommé SQL.

- Plus de distinction entre schéma de relation et relation : notion de table
- Une table SQL n'est pas un ensemble de tuples, mais un multi-ensemble de tuples, principalement pour des raisons d'efficacité (coût engendré par les tris pour supprimer les doublons)

SQL (Structured Query Language)

- Langage d'interrogation des données le plus populaire
- Peut être vu comme une surcouche syntaxique de l'algèbre relationnel avec des différences notoires comme par exemple la gestion de multi-ensembles au lieu des ensembles comme en algèbre
- Comporte aussi de nombreuses extensions propres aux SGBD
- SQL2: standard adopté en 1992

Définition d'une base de données

- Table = schéma de relation + relation
  - Création / Mise à jour / Suppression
    - CREATE TABLE ma_table…;
    - ALTER TABLE ma_table…;
    - DROP TABLE ma_table…;
  - Insertion/modifications/suppression de tuples
    - INSERT / UPDATE / DELETE

Types de données

- Domaine = type de données
  - Les valeurs d'une colonne
  - Type choisi lors de la création de la table
  - Spécifie l'espace de stockage
- Les nombres
  - Entiers ou flottants
  - Ex: 1000 2000.40 20.5E-9
- Chaînes de caractères
  - Données alphanumériques
  - Représentées entre quotes
  - Ex: 'Bonjour''une personne'
- Dates
  - Manipulation non standardisée

Types de données

- Entiers

nom	borne inférieure	borne supérieure
TINYINT	-128	127
TINYINT UNSIGNED	0	255
SMALLINT	-32768	32767
SMALLINT UNSIGNED	0	65535
MEDIUMINT	-8388608	8388607
MEDIUMINT UNSIGNED	0	16777215
INT ou INTEGER	-2147483648	2147483647
INT UNSIGNED	0	4294967295
BIGINT	-9,22337E+18	9,22337E+18
BIGINT UNSIGNED	0	1,84467E+19

- Flottants

nom	domaine négatif : borne inférieure / borne supérieure	Domaine positif : borne inférieure / borne supérieure
FLOAT	-3.402823466E+38 / -1.175494351E-38	1.175494351E-38 / 3.402823466E+38
DOUBLE	-1.7976931348623157E+308 / -2.2250738585072014E-308	2.2250738585072014E-308 / 1.7976931348623157E+308

Types de données

- Les chaînes de caractères

nom	longueur
CHAR(M)	Chaîne de taille fixée à M, où 1<M<255, complétée avec des espaces si nécessaire.
CHAR(M) BINARY	Idem, mais insensible à la casse lors des tris et recherches.
VARCHAR(M)	Chaîne de taille variable, de taille maximum M, où 1<M<255, complété avec des espaces si nécessaire.
VARCHAR(M) BINARY	Idem, mais insensible à la casse lors des tris et recherches.
TINYTEXT	Longueur maximale de 255 caractères.
TEXT	Longueur maximale de 65535 caractères.
MEDIUMTEXT	Longueur maximale de 16777215 caractères.
LONGTEXT	Longueur maximale de 4294967295 caractères.
DECIMAL(M,D)	Simule un nombre flottant de D chiffres après la virgule et de M chiffres au maximum. Chaque chiffre ainsi que la virgule et le signe moins (pas le plus) occupe un caractère.

Types de données

- Date et heure

nom	description
DATE	Date au format anglophone AAAA-MM-JJ.
DATETIME	Date et heure au format anglophone AAAA-MM-JJ HH:MM:SS.
TIMESTAMP	Affiche la date et l'heure sans séparateur : AAAAMMJJHHMMSS.
TIMESTAMP(M)	Idem mais M vaut un entier pair entre 2 et 14. Affiche les M premiers caractères de **TIMESTAMP**.
TIME	Heure au format HH:MM:SS.
YEAR	Année au format AAAA.

nom	description
TIMESTAMP(2)	AA
TIMESTAMP(4)	AAMM
TIMESTAMP(6)	AAMMJJ
TIMESTAMP(8)	AAAAMMJJ
TIMESTAMP(10)	AAMMJJHHMM
TIMESTAMP(12)	AAMMJJHHMMSS
TIMESTAMP(14)	AAAAMMJJHHMMSS

Création et suppression des tables

- Création de table:
  CREATE TABLE ma_table(
  attribut1 type,
  attribut2 type,
  …);
  - Ex:CREATE TABLE Personne (
    id int,
    nom varchar(20));
- Suppression de table
  DROP TABLE ma_table;
  - DROP TABLE Personne;

Modifications des tables

- ALTER TABLE table
  { RENAME TO nom_table
  | ADD (col typecol [contrainte])
  | MODIFY (col [typecol] [contrainte])
  | DROP COLUMN col [cascade constraints]
  | RENAME COLUMN nom TO nouveau_nom
  | ADD contraint
  | DROP {PRIMARY KEY | UNIQUE(col) | CONSTRAINT contrainte}
  [cascade constraints]
  | RENAME CONSTRAINT nom TO nouveau_nom
  | MODIFY [CONSTRAINT] contrainte statut_contrainte;

Modifications des tables

- Ajout de colonne:
  ALTER TABLE ma_table
  ADD (nom_colonne[type]);
  - Ex:ALTER TABLE ajout
    ADD (secteur char(6));
- Agrandir la taille d'une colonne
  ALTER TABLE ma_table
  MODIFY (nom_colonne type(taille));
  - Ex:ALTER TABLE ajout
    MODIFY (lib char(6));

Modifications des tables

- Renommage de table:
  ALTER TABLE ma_table
  RENAME TO ma_table2;
  - Ex:ALTER TABLE chocapic
    RENAME TO kellogs;
- Ajouter une contrainte
  ALTER TABLE ma_table
  ADD CONSTRAINT nom_contrainte
  - Ex:ALTER TABLE ajout
    ADD CONSTRAINT Cnombre (Nb>0)

Contraintes d'intégrité

- PRIMARY KEY:
  - clé primaire, identifiant de la relation (UNIQUE+NOT NULL)
- FOREIGN KEY listecolonne1 REFERENCES table(listecolonne2) [options]
  - clé étrangère (contrainte d'intégrité référentielle)
  - référence un attribut clé primaire ou unique
  - Options:
    - on delete {set null | set default | cascade}
      - set default: valeur par défaut si elle existe, sinon NULL
      - cascade: on met à jour la table secondaire

Les contraintes d'intégrité

- NOT NULL : force la saisie de la colonne
- DEFAULT : précise une valeur par défaut
- UNIQUE : vérifie que toutes les valeurs sont différentes (sauf NULL)
- CHECK : vérifie la condition précisée
- CONSTRAINT : permet de nommer une contrainte

Contraintes d'intégrité

- Exemples :

```
create table Animal (
num_exploitant varchar(20) primary key,
nom varchar(20) not null,
poids int constraint Cpoids (poids > 0),
pds_Sq int,
exploitation varchar(20),
constraint CPds_Sq CHECK (poids_Sq>0),
constraint CExploi CHECK (exploitation in
   ('Theix','Marcenat','Laqueuille'))
);
```

Mises à jour dans une base de données

- 3 ordres de base
  - INSERT → Ajoute une ligne
  - UPDATE → Modification de lignes existantes
  - DELETE → Suppression de lignes
- Ajout d'une ligne:
  - Syntaxe: INSERT INTO ma_table VALUES (…)

```
Ex: INSERT INTO tab1(nom, âge) VALUES ('PROFBD',152);
INSERT INTO tab1(âge, nom)     VALUES (152,'PROFBD');
INSERT INTO tab1               VALUES ('PROFBD',152);
```

Mises à jour dans une base de données

- Insertion « en masse »
  - Exemple:
    - INSERT INTO tab2(nom,age)
      SELECT nom,age from TEST.tab1;
  - Si erreur pendant la création aucune ligne n'est créée.
  - Création d'une table et insertion
  - Ce format permet la création de la table et l'insertion de lignes dans cette table
    - CREATE TABLE tab2 AS
      SELECT nom,age from TEST.tab1;

Mises à jour dans une base de données

- Modifications de tuples
  - Syntaxe:
  - UPDATE ma_table SET mon_attribut=…
    WHERE …
  - Si WHERE est omis alors toutes les lignes seront modifiées
    - UPDATE tab1
      SET attr1=val1
      WHERE attr2='val2';

Mises à jour dans une base de données

- Suppression de tuples
  - DELETE from ma_table
  - WHERE …
  - Si WHERE est omis, alors tous les tuples seront supprimés
    - DELETE from tab1
      WHERE nom='PROFBD';

Mises à jour dans une base de données

- Ordre TRUNCATE

  TRUNCATE TABLE ma_table;
  - Supprime toutes les lignes de la table
  - Plus rapide que le DELETE
  - Les lignes supprimées ne sont pas journalisées
  - Il est donc impossible de revenir en arrière pour les obtenir à nouveau par ROLLBACK

Définitions

- Lorsque toutes les opérations constituant une transaction (ensemble d'opérations) sont exécutées, et deviennent effectives, on dit qu'elle est validée:
  COMMIT;
- Dans le cas contraire, on dit qu'elle est annulée, on revient à l'état d'avant la transaction:
  ROLLBACK;

Notions de clés

- Clés primaire
  CREATE TABLE ma_table (
  attribut1 type PRIMARY KEY,
  attribut 2 type,
  …);

  OU

  CREATE TABLE ma_table (
  attribut1 type,
  attribut2 type,
  attribut3 type,
  PRIMARY KEY (attribut1,attribut2));

Notions de clés

- Clés étrangères
  CREATE TABLE ma_table2 (
  attribut1 type PRIMARY KEY,
  attribut 2 type,
  …,
  FOREIGN KEY (attribut2) REFERENCES ma_table(attribut3));
- Connaître la structure des tables
  DESCRIBE ma_table;
- Voir toutes les tables
  SHOW TABLES;

Syntaxe d'une requête SQL

- Une requête SQL comporte trois clauses :
  SELECT <liste d'attributs>
  FROM <listes relations>
  [WHERE <conditions sur les lignes>]
  →requête SFW
- La clause SELECT permet de coder la projection
- La clause WHERE permet de coder la sélection. Elle est optionnelle.
- Le résultat est une relation sur le schéma défini par la clause SELECT.
- Une requête SQL peut être nommée → on parle de vue.

Requêtes SFW

- Exemple : Q : "Nom et description des gènes du chromosome 1?"
- En algèbre : $\pi_{nom,descr}(\sigma_{chromosome=1}(Gene))$
- En SQL :

```
SELECT nom, descr
FROM gene
WHERE chromosome = 1;
```

Le renommage en SQL

- Possible sur les attributs dans la clause 'SELECT' et sur les relations dans la clause 'FROM'
- Mot clé réservé AS

```
SELECT nom [AS] "num_acc"
FROM Gene [AS] P
WHERE [P.]chromosome = 1;
```

- AS est optionnel !
- Résolution d'ambiguités :
- On prefixe l'attribut par le nom de sa relation (relation.nom)

Conditions complexes

- Mots clés AND, OR, NOT et parenthèses pour former des expressions complexes
- Permettent de former des formules de sélection complexes

```
SELECT nom, descr
FROM Gene
WHERE (chromosome=1) AND (debut > 1000000);
```

Utilisations de caractères jokers

→ ne s'exprime pas en algèbre relationnel

- Dans la clause 'SELECT' :
  - On peut lister tous les attributs avec le caractère '*'
- Dans la clause 'WHERE' :
  - Mot clé LIKE avec les caractères jokers : %,_

```
SELECT id
FROM Gene
WHERE nom LIKE 'B%';
```

- Expressions arithmétiques usuelles sur les valeurs retournées

```
SELECT nom, debut-50
FROM Gene
```

Jointure

- Permet de joindre 2 tables
  - **jointure naturelle** sur tous les attributs communs
    select * from R1, R2 where R1.A1=R2.A1 and R1.A2=R2.A2…..
  - **jointure simple**
    select * from R1,R2 where R1.A1=R2.A1;
    select * from R1,R2 where R1.A1<R2.A1;
  - **auto-jointure**: jointure sur la même table
    select * from R R1, R R2 where R1.A1=R2.A2

Jointure

- **jointure externe**: définition d'une table prioritaire

En ORACLE

```
select * from R1,R2 where R1.A1=R2.A1(+);
```

- + valeurs nulles de R1

EN MySQL

```
select R2.* from R2
left join R1 on R1.A1=R2.A1
where R1.A1 is null
```

Jointure

- Il faut expliciter les attributs de jointures, alors qu'ils sont implicites en algèbre
- Exemple:

  "Nom, prénom des personnes et les départements dans lesquels ils travaillent ?"
  - En algèbre : $\pi_{nom,\ prenom,\ dep}$ (Personnes ⋈ Travaillent)
  - En SQL :

    SELECT nom, prenom, dep
    FROM Personnes p, Travaillent t
    WHERE p.nss= t.nss;

Jointure

- 2 tables: prof et cours

id_p	nom_p
1	matthieu
2	alex
3	pierre
4	ana
5	emilie
6	vincent

id_c	nom_c	id_p
1	PHP	1
2	Bdd	1
3	Perl	2
4	Java	3
5	ModMol	4
6	Phylo	NULL

Jointure

- Jointure simple

```
select *                          OU      select *
from prof,cours                           from prof
where prof.id_p=cours.id_p                inner join cours on prof.id_p=cours.id_p
```

id_p	nom_p
1	matthieu
2	alex
3	pierre
4	ana
5	emilie
6	vincent

id_c	nom_c	id_p
1	PHP	1
2	Bdd	1
3	Perl	2
4	Java	3
5	ModMol	4
6	Phylo	NULL

id_p	nom_p	id_c	nom_c	id_p
1	matthieu	1	PHP	1
1	matthieu	2	Bdd	1
2	alex	3	Perl	2
3	pierre	4	Java	3
4	ana	5	ModMol	4

Jointure

- Jointure complexe: LEFT (outer) JOIN
  - S'il y a une ligne dans A qui répond à la clause WHERE, mais qu'il n'y a aucune ligne dans B qui répond à la condition du LEFT JOIN, alors une ligne supplémentaire de B est générée avec toutes les colonnes mises à NULL

```
select * from prof
left join cours on prof.id_p=cours.id_p
```

id_p	nom_p
1	matthieu
2	alex
3	pierre
4	ana
5	emilie
6	vincent

id_c	nom_c	id_p
1	PHP	1
2	Bdd	1
3	Perl	2
4	Java	3
5	ModMol	4
6	Phylo	NULL

id_p	nom_p	id_c	nom_c	id_p
1	matthieu	1	PHP	1
1	matthieu	2	Bdd	1
2	alex	3	Perl	2
3	pierre	4	Java	3
4	ana	5	ModMol	4
5	emilie	NULL	NULL	NULL
6	vincent	NULL	NULL	NULL

Jointure

- Jointure complexe: RIGHT (outer) JOIN
  - S'il y a une ligne dans B qui répond à la clause WHERE, mais qu'il n'y a aucune ligne dans A qui répond à la condition du RIGHT JOIN, alors une ligne supplémentaire de A est générée avec toutes les colonnes mises à NULL

```
select * from prof
right join cours on prof.id_p=cours.id_p
```

id_p	nom_p
1	matthieu
2	alex
3	pierre
4	ana
5	emilie
6	vincent

id_c	nom_c	id_p
1	PHP	1
2	Bdd	1
3	Perl	2
4	Java	3
5	ModMol	4
6	Phylo	NULL

id_p	nom_p	id_c	nom_c	id_p
1	matthieu	1	PHP	1
1	matthieu	2	Bdd	1
2	alex	3	Perl	2
3	pierre	4	Java	3
4	ana	5	ModMol	4
NULL	NULL	6	Phylo	NULL

Jointure

- Jointure complexe: FULL (outer) JOIN
  - C'est une combinaison des 2 précédentes, non nativement supporté par mysql

```
select * from prof left join cours on prof.id_p=cours.id_p
UNION
select * from prof right join cours on prof.id_p=cours.id_p
```

id_p	nom_p
1	matthieu
2	alex
3	pierre
4	ana
5	emilie
6	vincent

id_c	nom_c	id_p
1	PHP	1
2	Bdd	1
3	Perl	2
4	Java	3
5	ModMol	4
6	Phylo	NULL

id_p	nom_p	id_c	nom_c	id_p
1	matthieu	1	PHP	1
1	matthieu	2	Bdd	1
2	alex	3	Perl	2
3	pierre	4	Java	3
4	ana	5	ModMol	4
5	emilie	NULL	NULL	NULL
6	vincent	NULL	NULL	NULL
NULL	NULL	6	Phylo	NULL

Produit cartésien

- Il suffit de ne pas expliciter les attributs de jointure

```
SELECT P.*, T.*
FROM Personnes P, Travaillent T;
```

Les opérateurs ensemblistes

- L'union, l'intersection et la différence sont représentés respectivement par les mots clés UNION, INTERSECT et MINUS.
- Les deux opérandes doivent avoir le même schéma.
- Sur l'exemple, schema(R) = schema(S) = { A, B }.

```
SELECT A, B
FROM R
UNION
SELECT A, B
FROM S
```

Compositions de requêtes

- Le résultat d'une requête SFW peut être utilisé dans la clause FROM ou WHERE d'une autre requête
- En cas d'ambiguité, utiliser des parenthèses
- Les sous-requêtes de la clause 'WHERE' sont introduites par les mots

  → clés IN, EXISTS, ANY, ALL

Opérateur IN

```
SELECT ...
FROM R
WHERE X IN      (SELECT Y
                 FROM S
                 WHERE ...)
```

- Soit t Є R. t est sélectionnée si ∃t' Є S | t[X] = t' [Y]
  - SELECT nom, prenom
    FROM Personnes
    WHERE nss IN (SELECT nss
    FROM Travaillent)
- Possibilité d'utiliser NOT IN

Opérateur EXISTS

```
SELECT ...
FROM R
WHERE EXISTS (SELECT *
                FROM S
                WHERE C)
```

- La condition C doit comporter au moins un attribut de R
- Soit t Є R. t est sélectionnée si ∃t' Є S | t et t' satisfont C
  - SELECT nom, prenom
    FROM Personnes P
    WHERE EXISTS (SELECT *
    FROM Travaillent T
    WHERE P.nss = T.nss)

Opérateur BETWEEN

```
SELECT ...
FROM R
WHERE attr BETWEEN val1 AND val2
```

- L'attribut attr doit être de type numérique
- Exemple
  - SELECT nom, prenom
    FROM Personnes P
    WHERE age BETWEEN 20 AND 30

Le quantificateur existentiel ANY

→ généralise IN

- Permet d'exprimer les requêtes du type "il existe ...«
  - SELECT ...
    FROM R
    WHERE X bop ANY (SELECT Y
    FROM S
    WHERE ...)
- 'bop' est un opérateur binaire de comparaison classique, càd =, <,>, . . . qui retourne vrai ou faux
- Soit t Є R. t est sélectionnée si ∃t' Є S tq t[X] bop t'[Y]
- Remarque : IN ≡ = ANY

Le quantificateur universel ALL

- Permet d'exprimer les requêtes du type "pour tous les ..."
  SELECT ...
  FROM R
  WHERE X bop ALL (SELECT Y
  FROM S
  WHERE ...)
- 'bop' est un opérateur binaire de comparaison classique, càd =, <,>, . . . qui retourne vrai ou faux
- Soit t Є R. t est sélectionnée si ∀t' Є S tq t[X] bop t'[Y]

Exemples

id_e	nom_e
1	matthieu
2	alex
3	pierre
4	ana
5	emilie
6	vincent

id_c	nom_c
1	PHP
2	Bdd
3	Perl
4	Java
5	ModMol
6	Phylo

id_e	id_c	note
1	1	17
1	2	15
2	3	9
3	3	14
4	5	12

- IN et EXISTS
  - Les étudiants qui ont une note en perl

```
select id_e from aff where id_c in
(select id_c from cours where nom_c='perl')

OU

select id_e from aff where exists
(select cours.id_c from cours where cours.id_c=prof.id_c and nom_c='perl')
```

Exemples

id_e	id_c	note
1	1	17
1	2	15
2	3	9
3	3	14
4	5	12

- ANY et ALL
  - Liste des étudiants de la classe 5 qui ont plus qu'au moins 1 étudiant de la classe 3

```
select id_e from aff where id_c=5
and note > ANY (select note from aff where id_c=3)
```

  - Liste des étudiants de la classe 5 qui ont plus que tous les étudiants de la classe 3

```
select id_e from aff where id_c=5
and note > ALL (select note from aff where id_c=3)
```

Opérations ensemblistes

- INTERSECT Et MINUS ne sont pas implémentés en SQL donc les requêtes se font en utilisant le mot-clé EXISTS

```
select a,b from table1
intersect                 →   select a,b from table1 where exists
select c,d from table 2       (select c,d from table2 where a=c and b=d)

select a,b from table1
minus                     →   select a,b from table1 where not exists
select c,d from table 2       (select c,d from table2 where a=c and b=d)
```

Opérations ensemblistes

- Exemples:

id_e	nom_e
1	matthieu
2	alex
3	pierre
4	ana
5	emilie
6	vincent

id_c	nom_c
1	PHP
2	Bdd
3	Perl
4	Java
5	ModMol
6	Phylo

id_e	id_c	note
1	1	17
1	2	15
2	2	9
3	3	14
4	5	12

  - Liste des étudiants ayant des notes en Bdd ou en Perl

```
select id_e FROM aff where id_c=1
union select id_e from aff where id_c=2
```

  - Liste des étudiants ayant des notes en PHP et en Bdd

```
select id_e FROM aff where id_c=1
and exists (select id_e from aff where id_c=2)
```

  - Liste des étudiants ayant des notes en Bdd mais pas en PHP

```
select id_p FROM aff a1 where id_c=2
and not exists (select id_p from aff a2 where id_c=1 and a1.id_p != a2.id_p)
```

Passerelles entre ensembles et multi-ensembles

- Possible grâce aux mots clés DISTINCT et ALL
  - multi ensemble vers ensemble : DISTINCT
  - ensemble vers multi ensemble : ALL
- Les requêtes SFW sont par défaut définies sur des multi ensembles
  → pour forcer la sémantique ensembliste, utiliser le mot clé 'DISTINCT' après 'SELECT'
- Les opérateurs ensemblistes (UNION, INTERSECT, MINUS) forcent par défaut la sémantique ensembliste ! ! !
  → pour forcer la sémantique multi ensemble, utiliser le mot clé 'ALL' après 'UNION', ...
  → Etre prudent sur les conversions, et se méfier du coût (opération de TRI) engendré par DISTINCT

Les valeurs nulles

- Très nombreuses en pratique
- Rend la conception de requêtes toujours TRES compliquées
- Sémantique des valeurs nulles : une condition logique avec un NULL
- retourne toujours FAUX
- Mot clé IS [NOT] NULL
- Expression de sélection :
  - SELECT A, B, C
    FROM R
    WHERE A IS NOT NULL OR (B IS NULL AND C IS NULL);

Fonctions d'agrégation

- Les fonctions d'agrégation permettent de répondre aux requêtes du type :
  - "Quel est le nombre d'étudiants inscrits dans chaque département"
  - "Quel est l'âge moyen des étudiants"
- Possibilités de calculer des fonctions sur une colonne du résultat
  → la requête retourne une colonne et une ligne ! ! !
- Fonctions arithmétiques classiques : SUM, AVG, MIN, AVG, COUNT ... (dépend du SGBD utilisé)
  - SELECT count(*) "Nombre de lignes "
    FROM Gene;
  - SELECT AVG(debut)
    FROM Gene;

Fonctions d'agrégation

- AVG(expr): moyenne de toutes les valeurs de expr
- COUNT(*): nombre de tuples renvoyés par la sélection
- COUNT(expr): nombre de tuples renvoyés par la sélection, pour lesquels expr n'a pas une valeur nulle
- MAX(expr): valeur maximale de toutes les valeurs renvoyées par expr
- MIN(expr): valeur minimale de toutes les valeurs renvoyées par expr
- STDDEV(expr): ecart-type de toutes les valeurs de expr
- SUM(expr): somme de toutes les valeurs de expr
- VARIANCE(expr): variance de toutes les valeurs de expr

Regroupement des données: la clause GROUP BY

- Permet de généraliser les fonctions précédentes : le calcul des agrégations est possible sur des éléments d'une partition
- Permet d'utiliser les fonctions d'agrégation sur des sous ensembles de valeurs.
- Syntaxe :
  - SELECT <liste d'attributs>
    FROM <listes relations>
    WHERE <conditions sur les lignes>
    GROUP BY <liste d'attributs>
    HAVING <conditions sur les éléments de la partition>
- La partition de la table résultat se fait sur les attributs spécifiés après le GROUP BY.
- Le résultat de la requête comporte autant de lignes que d'éléments dans la partition.

Exemple

- Exemple : Achat(id, client, date, prix)
- "Prix moyen des achats effectués par chaque client"
  SELECT avg(prix), client
  FROM Achat
  GROUP BY client
- L'utilisation de la clause GROUP BY impose des restrictions sur la liste des attributs de la clause SELECT :
  - soit un attribut agrégé, càd avec SUM, AVG ...
  - soit un attribut qui apparaît déjà dans la clause GROUP BY

Conditions de regroupement sur les regroupements

- La clause WHERE permettait de définir des sélections sur les lignes du résultat d'une requête
  → la clause HAVING va permettre de définir des sélections sur les éléments d'une partition
- Exemple: Achat(id, client, date, prix)
- "Prix moyen des achats effectués par chaque client en quantité au moins égale à 2"
  - SELECT avg(prix), client
    FROM Achat
    GROUP BY client
    HAVING count(*)>=2

Tri des résultats

- ORDER BY
  - Permet de trier les résultats de votre requête
    - Ex: select nom, taille from Gene
    order by nom
  - Possibilités de gérer le tri:
    - ascendant (ASC) par défaut
    - descendant (DESC)
  - NULLS FIRST: mettre les NULL en premier
  - NULLS LAST: mettre les NULL en dernier (par défaut)

Fonctions de date

- date +/- n: date obtenue en ajoutant/soustrayant n jours
- round(date[,format]): arrondi de la date au format spécifié
  - round('30/06/10','Y')='01/01/10'
  - round('01/07/10','Y')='01/01/11'
  - round('19/07/10','12:00')='20/07/10'
- trunc(date[,format]): date tronquée au format spécifié
  - trunc('30/06/10','Y')='01/01/10'
  - trunc('01/07/10','Y')='01/01/10'
  - trunc('19/07/10','12:00')='19/07/10'
- sysdate: renvoie la date et l'heure courante dans le système
  - select sysdate() from DUAL; → utilisation de la pseudo table DUAL

Fonctions de date

- to_days(date)
  - Where to_days(now()) – to_days(date) <30
- last_day(date): renvoie le dernier jour du mois de date
  - last_day('19/07/10')='31/07/10'
- datediff: renvoie le nombre de jours entre 2 dates
- adddate(date, INTERVAL valeur): ajoute une valeur à la date donnée
  - SELECT date_modif, ADDDATE(date_modif, INTERVAL 1 MONTH)
  - SELECT date_modif, ADDDATE(date_modif, INTERVAL 30 DAY)
- Date2 – Date1: nombre de jours séparant 2 dates
  - '20/07/10' – '18/07/10' = 2

Les vues

- Une vue est une table virtuelle, les données ne sont pas stockées dans une table de la base de données
- On peut y rassembler des informations provenant de plusieurs tables
- vue = représentation des données dans le but d'une exploitation visuelle
- données présentes définies grâce à une clause SELECT
- Avantages:
  - informations dynamiques
  - restriction d'accès à la table pas l'utilisateur
  - regroupement d'informations au sein d'une seule entité

Les vues

- Syntaxe:
  - CREATE VIEW Nom_vue (attributs) AS SELECT ...
  - CREATE VIEW age_abattage AS select
    a.NUMOFFICIEL AS NUMOFFICIEL,
    (to_days(e.DATEVT)-to_days(a.DATNAI)) AS AGE_JOUR
    from d_evts e,d_animal a
    where d.NUMOFFICIEL=a.NUMOFFICIEL and (e.CODEVT=1001);

La pseudo table DUAL

- Permet d'afficher une expression dont la valeur ne dépend d'aucune table de votre base:
- Exemples:
  - Select sysdate() from DUAL;
    → affichera la date du jour au format aaaa-mm-jj hh:mm:ss
  - Select 10*4 from DUAL;
    → affichera 40

Partie V
Utilisation de BD dans un langage de programmation

Perl et bases de données

- Package perl DBI (DataBase Interface)
- Permet d'effectuer des requêtes SQL au sein d'un programme
- Fonctionne avec tous les SGBD
- Principe:
  - En début de programme

```
use DBI();
```

Perl et bases de données

- Connexion à la base de données

```
my $dbh =DBI->connect("DBI:<SGBD>:database=<nombd>"),
("<nom_user>","<mdp>") or die "Echec de la connexion: ".$DBI.errstr;
my $sth;
```

  - Exemple:

```
my $dbh =DBI->connect("DBI:mysql:database=biotools"), ("root","")
or die "Echec de la connexion: ".$DBI.errstr;
my $sth;
```

- Déconnexion

```
$dbh->disconnect;
```

Perl et bases de données

- Requêtes de création/insertion:

```
my $sql= "insert into tab1 values ('matthieu',30) ";
$dbh->do($sql);
```

- Requêtes de sélection:
  - création de la requête

```
my $sql= "select * from tab1";
```

  - exécution de la séquence

```
$sth = $dbh->prepare($sql);
$sth->execute;
```

Perl et bases de données

  - récupération des informations

```
while (my ($nom , $age) = $sth->fetchrow_array())
{
      print "nom: $nom,age:$age\n";
}
$sth->finish;
```

→ affichera nom:matthieu,age:30

  - valider vos changements:

```
$dbh ->commit;
```

  - annuler:

```
$dbh ->rollback;
```