Bases de Données
Nouveau programme des CPGE

Dr. Benjamin NGUYEN
benjamin.nguyen@inria.fr

UVSQ & INRIA
Laboratoire PRiSM, CNRS UMR 8144
Equipe-Projet INRIA SMIS « Secured and Mobile Information Systems »

http://cassiopee.prism.uvsq.fr/luminy/

Luminy 2013-05-10

Qui suis-je ?

- Ancien élève de prépa (94-96 au lycée Henri IV)
- Ancien élève de l'ENS Cachan (Dpt Physique, 96-00) (mais je voulais faire info !)
- DEA & Thèse (00-03) à Paris-XI / INRIA sous la direction de Serge Abiteboul, thème « Bases de Données XML »
- Maître de conférences à UVSQ depuis 2004, équipe du Pr. Georges Gardarin « Data Integration and Management » (dont j'emprunte certains transparents)
- Depuis 2010 : Equipe INRIA « Secured and Mobile Information Systems » du Pr. Philippe Pucheral
- Intérêts de recherche actuels : la protection de la vie privée dans les bases de données et le Web

Mais aussi :

- Passionné d'informatique depuis son plus jeune âge !
- « Fertilisation » interdisciplinaire de l'usage des BD (sociologie, économie, droit)
- Promotion de l'informatique en tant que discipline « scientifique » propre, formation des enseignants de Terminale à la spécialité ISN

Programme officiel 1/3

- L'objectif de cette partie de la formation vise à développer les savoir-faire suivants :
  - recourir aux concepts des bases de données relationnelles ;
  - traduire les questions posées dans un langage de requête en respectant sa syntaxe ;
  - prototyper et créer une base de données simple, à l'aide d'un outil interactif ;
  - consulter une base de données à travers des requêtes de type SQL ;
  - comprendre et décrire les rôles des différents éléments d'une architecture trois-tiers.
- La formation doit mettre en évidence la nécessité d'un niveau d'abstraction suffisant dans la conception d'outils permettant la gestion de bases de données de taille importante, là où des algorithmes de recherche simples sur des structures « plates », orientées tableaux, deviennent inopérants : les schémas relationnels sont une réponse à ce problème.

Programme officiel 2/3

Contenus	**Précisions et commentaires**
Vocabulaire des bases de données : relation, attribut, domaine, schéma de relation ; notion de clé primaire.	Ces concepts sont présentés dans une perspective applicative, à partir d'exemples.
Opérateurs usuels sur les ensembles dans un contexte de bases de données : union, intersection, différence. Opérateurs spécifiques de l'algèbre relationnelle : projection, sélection (ou restriction), renommage, jointure, produit et division cartésiennes ; fonctions d'agrégation : min, max, somme, moyenne, comptage.	Ces concepts sont présentés dans une perspective applicative. Les seules jointures présentées seront les jointures symétriques, simples (utilisant JOIN … ON …=...).
Concept de client-serveur. Brève extension au cas de l'architecture trois-tiers.	On se limite à présenter ce concept dans la perspective applicative d'utilisation de bases de données.

Programme officiel 3/3

- La liste suivante énumère un choix non exhaustif d'exercices pratiques. Les bases de données utilisées à des fins d'illustration concerneront de préférence des questions choisies au sein des autres disciplines scientifiques et technologiques.
  - utiliser une application de création et de manipulation de données, offrant une interface graphique, notamment pour créer une base de données simple, ne comportant pas plus de trois tables ayant chacune un nombre limité de colonnes. L'installation et l'exploitation d'un serveur SQL ne fait pas partie des attendus.
  - lancer des requêtes sur une base de données de taille plus importante, comportant plusieurs tables, que les étudiants n'auront pas eu à construire, à l'aide d'une application offrant une interface graphique ;
  - enchaîner une requête sur une base de données et un traitement des réponses enregistrées dans un fichier.
- **Les principales capacités** développées dans cette partie de la formation sont :
  - utiliser une application offrant une interface graphique pour créer une base de données et l'alimenter,
  - utiliser une application offrant une interface graphique pour lancer des requêtes sur une base de données,
  - distinguer les rôles respectifs des machines client, serveur, et éventuellement serveur de données,
  - traduire dans le langage de l'algèbre relationnelle des requêtes écrites en langage courant,
  - concevoir une base constituée de plusieurs tables, et utiliser les jointures symétriques pour effectuer des requêtes croisées.

Ce dont le programme ne parle pas

- Les fondements théoriques des bases de données (implicite / optionnel)
- La modélisation (Modèle E/R) – me parait intéressant (sinon on ne pense que « tuple » et algèbre)

Temps imparti : 15% d'environ 40(?) séances, à raison de 2h par semaine (en moyenne) = 6h (3 semaines).

Découpage proposé :

- 2h de cours
- 2 TDs de 2h : 1 sur « papier » et 1 sur « machine »

À l'université le programme proposé prend environ 4-6 semaines de 4h30.

IL NE FAUT PAS LAISSER CROIRE AUX ÉLÈVES QUE LE TEMPS PASSÉ EST SUFFISANT POUR BIEN CONNAÎTRE LES BASES DE DONNÉES !

IL LEUR MANQUERA UN GRAND NOMBRE DE NOTIONS !

Programme de la journée

- Présentation des bases de données telles qu'elles sont définies dans le « nouveau » programme de prépa
  - Vocabulaire (concepts) des bases de données
  - Algèbre relationnelle
  - SQL
- Quelques développements « hors programme »
  - Modèle Entité/Association
  - Contraintes d'intégrité
  - Calcul relationnel
- Un exemple de TP sur machine

Incitations du programme

- Le programme semble inciter à partir du besoin d'interrogation, de constater la difficulté d'écrire des algorithmes efficaces, et de conclure sur le besoin des bases de données
- Comme il est indiqué dans le programme, je vais toujours prendre des exemples ancrés dans le réel.

De la difficulté d'interroger les grandes masses de données …

- Soit un ensemble de données représentant des élèves, les modules qu'ils suivent, et leur notes.
- On souhaite interroger ces données pour retrouver les notes d'un élève, calculer des moyennes, etc.

1. Il faut modéliser ces données (existe-t-il une méthode générique simple applicable?)
2. Il faut définir pour chaque opération d'interrogation un *programme* qui réalise cette opération. On pourra définir des *sous-programmes* pour des tâches à réaliser fréquemment.
3. On souhaite rendre les données pérennes :
    1. Il faut les sauvegarder sur un média durable
    2. Il faut gérer les pannes à tout moment
4. On souhaite modifier les données
5. On souhaite sécuriser l'accès aux données

Exemple : des élèves et leurs notes

- Définir une structure élève complexe qu'on va mettre dans un tableau. En soi c'est déjà compliqué.

```
struct eleve{
nom : string;
notes : tableau [X] de int ; // ou quelque chose de plus compliqué
}
```

- Calculer la moyenne des notes d'un élève = écrire une fonction

```
float moyenne (eleve e){
        float somme = 0;
        for(int i=0;i<e.notes.length;i++) somme+=e.notes[i];
        return somme/e.notes.length;
}
```

- Stocker les données = définir un format de fichier et les procédures permettant de lire ou écrire des données.
- Modifier les données = écrire un programme
- Sécuriser les donnée = écrire (plusieurs) programmes
- Etc.

DEJA SUR CET EXEMPLE CE N'EST PAS SIMPLE !!

Les problèmes des systèmes à base de fichiers

- Format de fichiers non standards (chacun crée le sien)
- Redondance de données (incohérences possibles ou difficiles à gérer)
- Écriture d'un programme spécifique pour chaque interrogation : coûts de développement élevés, coûts de maintenance élevés
- Gestion des pannes à programmer soi même (l'OS ne suffit pas)
- Partage de données difficile
- Sécurisation des données difficile

BREF TOUT EST A FAIRE SOI-MEME !

La réponse « SGBD » = un « package » comprenant :

- Indépendance physique
  - Possibilité de modifier les structures de stockage sans modifier les programmes
  - Ecriture des applications par des non-spécialistes des fichiers
  - Meilleure portabilité, indépendance vis-à-vis du matériel
- Indépendance logique
  - Possibilité d'ignorer les données d'autres applications
  - Possibilité d'enrichir les applications sans devoir tout réécrire
  - Possibilité de protéger (rendre confidentielles) certaines données
- Manipulation aisée
  - Par le biais d'un langage déclaratif (SQL) équivalent à la logique du 1er ordre
- Vues multiples (virtuelles) des données
- Exécution et optimisation
  - Les requêtes sont traduites en un langage procédural (algèbre relationnelle)
  - … qui peut être optimisé *automatiquement*. (des années de recherche en BD…)
- Intégrité logique (Contraintes d'intégrité)
  - Par le biais d'un langage déclaratif
  - Détection de mises à jour erronées
  - Contrôle sur les données élémentaires et les relations
- Intégrité physique
  - Tolérence aux pannes
    - Transactions
    - Système (panne courrant)
    - Disque (crash)
- Partage de données
  - Isolation (chacun a l'air d'être seul)
  - Concurrence (tout le monde peut agir en même temps)
- Confidentialité
- Standardisation

Plan

- Le Modèle Entité/Association (HP)
- Les Fondements Théoriques
- Les Requêtes par l'Exemple
- Travaux pratiques

LE MODELE ENTITE/ASSOCIATION

The Entity-Relationship Model, Toward a Unified View of Data,
Peter P.-S. Chen
ACM Transactions on Database Systems (TODS) 1:(1), 1976

1. Objectifs et principes
2. Le modèle Entité-Association (E/R)
3. Conclusion

Une Base de Données naît d'un besoin

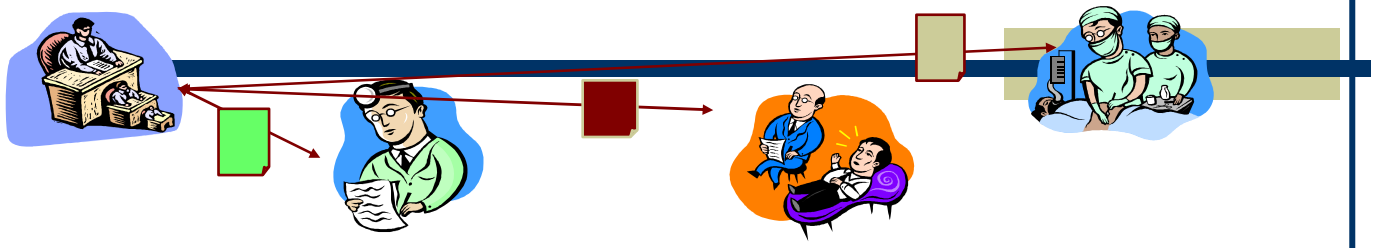
- Stocker des (grandes quantités de) données
  - Quel type de données ?
  - Comment structurer les données conceptuellement ? Et « physiquement » ?
  - → Modélisation Conceptuelle (E/R) puis traduction dans un modèle logique (e.g. Relationnel, XML, Objet, etc.)
  - → Stockage « physique » (on n'en parle pas du tout ici).
- Interroger des (grandes quantités de) données
  - Faut-il écrire un programme pour chaque action spécifique (requête) qu'on souhaite faire sur les données ?
  - → Utilisation d'un langage de requêtes adapté au modèle logique
  - → Exécution (on en parle un peu) et optimisation (on n'en parle pas) de ces requêtes

Introduire le modèle relationnel

- Il me parait plus naturel de venir au modèle relationnel comme *une manière* d'implémenter un modèle conceptuel plutôt que de le poser de manière *ad-hoc*.
- Le modèles conceptuel est une manière de représenter de données à *gérer* :

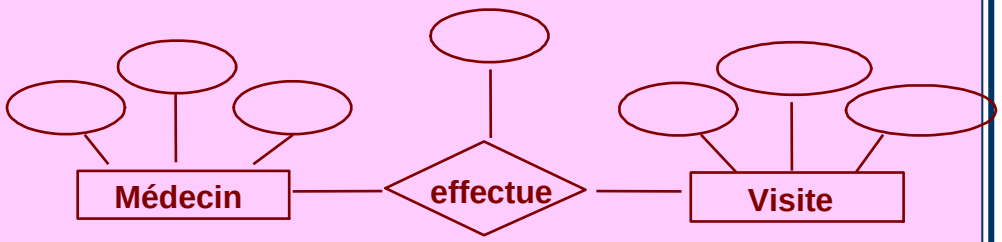
  Les (systèmes de gestion des) bases de données ne résolvent pas des problèmes particuliers, mais permettent de gérer n'importe quelle sorte de données concrètes.
- Le modèle relationnel d'une base de données se déduit simplement d'un modèle entité-association.

Modélisation à plusieurs niveaux



Modèle conceptuel

- Indépendant du modèle de données
- Indépendant du SGBD



Modèle logique

- Dépendant du modèle de données
- Indépendant du SGBD

Relationnel | **Objet** | **XML**

Modèle Physique

- Dépendant du modèle de données
- Dépendant du SGBD

- **Organisation physique des données**
- **Structures de stockage des données**
- **Structures accélératrices (index)**

1. Objectifs de la Modélisation

- Permettre une meilleure compréhension
  - Le monde réel est trop complexes
  - Abstraction des aspects cruciaux du problème
  - Omission des détails
- Permettre une conception progressive
  - Abstractions et raffinements successifs
  - Possibilité de prototypage rapide
  - Découpage en modules ou packages
  - Génération des structures de données (et de traitements)

Élaborer un modèle conceptuel

- Isoler les concepts fondamentaux
  - Que vont représenter les données de la BD ?
  - Découvrir les concepts élémentaires du monde réel
  - Décrire les concepts agrégés et les sous-concepts
- Faciliter la visualisation du système
  - Diagrammes avec notations simple et précise
  - Compréhension visuelle et non seulement intellectuelle

Dériver le schéma de la BD

- Schéma
  - Définition de tous les types de données de la base et de leurs liens
- Agrégation de données
  - Type élémentaire (de base): Entier, Réel, String, ...
  - Type complexe (composé): Collection de types élémentaires
    - Exemple : Type Personne (nom: String, Prenom: String, age: Réel)
- Possibilité d'intégrer des relations entre données (liens)
    - Exemple : Personne → Voitures;
- Le schéma (abstrait) est utilisé pour créer de véritables objets du monde réel (instantiation)
  - Instance ou occurrence : Personne("Dupont", "Jules", 20)
  - Ensemble de Voitures {id:String}: Voitures {"75AB75", "1200VV94"}
  - Création d'une relation entre une personne et une voiture : "Dupont" → "75AB75"

Méthodes

- Méthodes d'analyse et de décomposition hiérarchiques
  - 1e génération basée sur des arbres fonctionnels
  - Diviser pour régner (Problème --> Sous-problème)
  - Warnier, SADT, Jackson, De Marco
- Méthodes d'analyse et de représentation systémiques
  - 2e génération basée sur entité-association
  - Séparation des données et traitements
  - Merise, Axial, SSADM
- Méthodes d'analyse et de conception orientées objets
  - 3e génération basée sur les objets
  - Réconciliation données et traitements
  - Réutilisation de composants

2. Le Modèle Entité – Association (E/R Model)

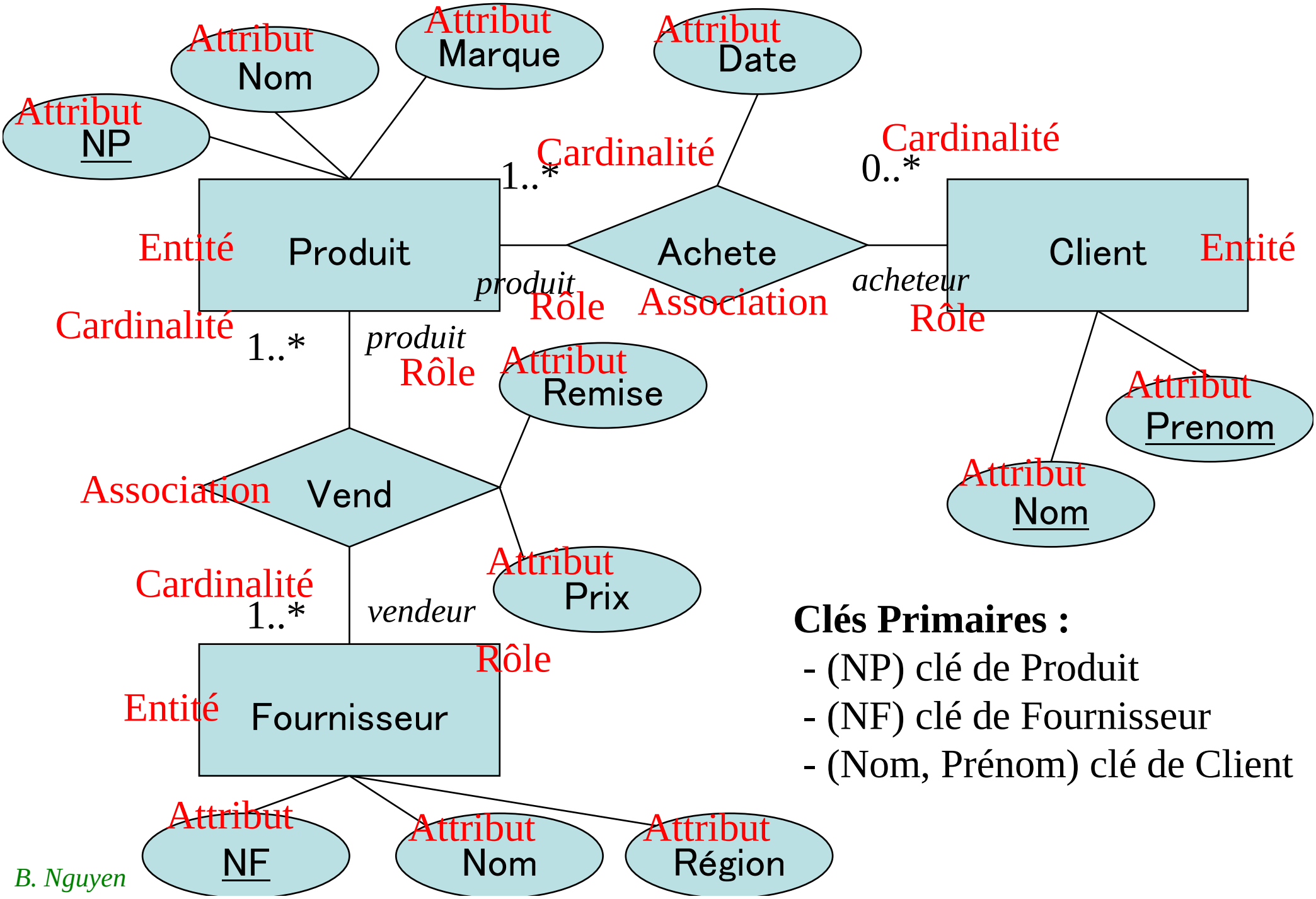
- Ensemble de concepts pour modéliser les données d'une application (d'une entreprise)
- Ensemble de symboles graphiques associés

- Formalisé en 1976 par Peter Chen dans :

  The Entity-Relationship model, towards a unified view of data, in *ACM Transactions on Database Systems*, 1(1), pp 9-36, 1976

- Etendu vers E/R généralisé puis vers l'objet

Un exemple de modèle Entité-Association



Clés Primaires :

- (NP) clé de Produit
- (NF) clé de Fournisseur
- (Nom, Prénom) clé de Client

A- Classe d'Entité

- Un objet du monde réel qui peut être identifié et que l'on souhaite représenter
  - La classe d'entité correspond à une collection d'entités décrites par leur type commun (le format)
  - L'instance d'entité correspond à un élément particulier de la classe d'entité (un objet)
  - Attention: on dit entité pour les deux ! Comprendre selon le contexte.
- Il existe généralement plusieurs *instances* d'entités dans une *classe* d'entité.

Attribut

- Description des propriétés des entités
- Toutes les instances d'une entité ont les mêmes attributs
  - Attribut simple: attribut ayant une valeur d'un type de base
  - Attribut composé: attribut constitué d'un groupe d'attributs
  - Attribut multi-valué: attribut pouvant avoir plus d'une valeur
- Avec le modèle E/R de base tout attribut est simple
- Avec le modèle E/R étendu, les attributs peuvent etre complexes
  - Composés et multi-valués (voir pbs de normalisation, HP)

Identifiant ou Clé

- **Un identifiant aussi appelé clé est un attribut qui permet de retrouver une instance d'entité unique à tout instant parmi celles de la classe.**
  - Exemple: NVeh dans Voitures, NSS dans Personnes
- **Un identifiant peut être constitué de plusieurs attributs (clé composée)**
  - Exemple:
    - [N° , Rue, Ville] pour Maisons
    - [Nom, Prénom] pour Personnes
- **Clés candidates et clés primaires**
  - Une clé candidate est un ensemble d'attributs *permettant* d'identifier de manière unique une instance d'une entité
  - Parmi les clés candidates, on en choisit une, qu'on nomme clé primaire qui *va* identifier l'entité

B- Association

- Les entités sont reliées ensemble par des associations
  - Entre instances: par exemple 1 véhicule est associé à 1 personne
  - Entre classes: abstraction des associations entre instances
- Une association peut avoir des attributs (propriétés)
- Elle peut relier plusieurs entités ensemble
- Il est possible de distinguer le rôle d'une entité (elle peut en avoir plusieurs)

Association: quelques définitions

- Association (Association)
  - Une relation entre des instances de deux (ou plus) classes
- Lien (Link)
  - Une instance d'association
- Rôle (Role)
  - Une extrémité d'une association
- Attribut de lien (Link attribute)
  - Un attribut de l'association instancié pour chaque lien
- Cardinalité (Multiplicity)
  - Le nombre d'instance d'une entité pour chaque instance de l'autre

Cardinalité (max) d'une association

- 1:1

one-to-one

- 1:N

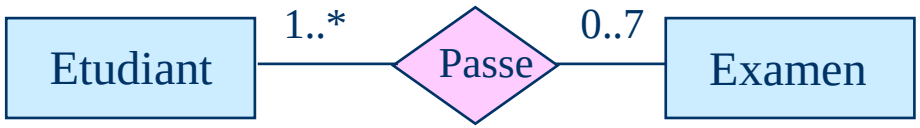
one-to-many

- N:M

many-to-many

Cardinalités min et max (Notations UML)

- Cardinalité maximum
  - Indique le nombre maximum d'instances d'une classe d'entité participant à une association
- Cardinalité minimum
  - Indique le nombre minimum d'instances d'une classe d'entité participant à une association



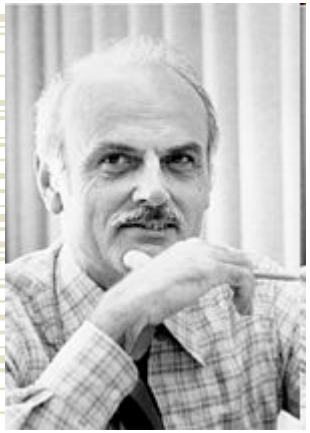
Domaines

- Ensemble nommé de valeurs
  - Un attribut peut prend valeur dans un domaine
  - Généralisation des types élémentaires
- Exemples
  - Liste de valeurs (1,2,3)
  - Type contraint (0< int <100)
- Permettent de préciser les valeurs possibles des attributs
- Réduisent les ambiguïtés

3. Conclusion : La pratique de la conception

- Bien comprendre le problème à résoudre
- Essayer de conserver le modèle simple
- Bien choisir les noms
- Ne pas cacher les associations sous forme d'attributs
  - utiliser les associations
- Faire revoir le modèle par d'autres
  - définir en commun les objets de l'entreprise
- Documenter les significations et conventions
  - élaborer le dictionnaire

FONDEMENTS THEORIQUES : LE CALCUL RELATIONNEL & LE MODELE RELATIONNEL



1923-2003

A relational model of Data For Large Shared Data Banks

E.F. Codd,

Communications of the ACM (CACM) 13(6), 1970

1. Concepts pour la description
2. Concepts pour la manipulation
3. Concepts additionnels

Langages

- Dans la suite nous allons décrire 3 approches/langages « équivalentes »
  - Le calcul relationnel : langage déclaratif (intentionnel) dérivée de la logique du premier ordre
  - L'algèbre relationnelle : langage impératif (procédural)
  - La « syntaxe » Structures Query Language (SQL)
- Les parties « de base » de ces trois langages sont équivalents (au sens du pouvoir expressif) selon le théorème de Codd (1970)
- Les 3 langages utilisent des concepts proches, et des noms parfois différents.

1. CONCEPTS DESCRIPTIFS

- Ensemble de concepts pour formaliser la description d'articles de fichiers plats
- Modèle standardisé mais extensible (depuis 1986)
  - Introduction de types de données variés (SQL2)
  - Introduction de la dynamique (Triggers), des types non scalaires et objets, requêtes récursives… (SQL3)
  - Introduction du XML (SQL:2003)
  - SQL:2006 : amélioration du XML
  - SQL:2008 : triggers instead of
  - Version actuelle : SQL:2011 (fenetres glissantes, support du temporel)

Les concepts

- Domaine *D* (ou type)
  - Les entiers, les chaînes de caractères de longueur 32, les pays, les couleurs aux cartes, etc.
- Attribut : un couple nom/domaine (= Colonne en SQL)
- Variable de relation : un *ensemble* ordonné d'attributs (sert d'entête à une relation)
  - (Nom:String, Age:Int)
- Tuple (n-uplet) : un *ensemble* ordonné de couples attribut/valeur (= Rangée ou *Row* en SQL)
  - (Nom: Toto, Age : 18)
- Relation : un *ensemble* de tuples (= Table en SQL)

Etudiants	Nom	Age
	Toto	18
	Titi	22
	Tata	20

Définitions 1/2

- Soit $D=\{D_i\}$ un ensemble de domaines.
  String, int, etc.
- Soit C un ensemble (fini) d'attributs A_i (colonnes) sur des domaines D_i
  C={NOM:string, PRENOM: string, NC:int, …}
- Soit R un ensemble fini de noms de relations.
  R={RESPONSABLE, COURS, ETUDIANT, INSCRIT}
- Soit h une fonction $R \rightarrow 2^C$ qui à une relation associe un sous ensemble de C.
  h(RESPONSABLE)={NR:int, NOM:string, PRENOM:string}
  …
- On dit que $S=(D, R, h)$ est un *schéma relationnel.*

e.g.

RESPONSABLE (NR:int, NOM:string, PRENOM:string)
COURS (NC:int, CODE_COURS:string, INTITULE:string)
ETUDIANT (NE:int, NOM:string, PRENOM:string)
INSCRIT (NE:int, NC:int,ANNEE:int)

Définitions 2/2

- Un *tuple* (ou *n-uplet*) est défini comme une fonction partielle de $C \rightarrow D_0 \times D_1 \times \ldots$ (i.e. cette fonction donne une valeur à un attribut)
  t_1=(NE: 123456, NOM: «Marie» , PRENOM: « LEGRAND »)
- Le sous ensemble de C sur lequel t est défini est appelé *domaine de t* et noté *dom*(t).
- L'ensemble de tous les tuples possibles (sur D) est noté T_D
- Etant donné un schéma $S=(D, R, h)$, une *base de données relationnelle DB* est définie comme une fonction de $R \rightarrow 2^{TD}$ qui associe à chaque relation r de R un sous-ensemble fini de T_D tel que pour chaque tuple t de ce sous-ensemble $h(r)=dom(t)$

La Clé : un concept fondamental

- GROUPE D'ATTRIBUTS MINIMUM QUI DETERMINE UN TUPLE UNIQUE DANS UNE RELATION = clé « candidate »
- Exemples:
  - Ajouter NUMETU dans ETUDIANTS
  - …ou bien un ensemble d'attributs dont la valeur est unique!
- CONTRAINTE D'ENTITE
  - Toute relation doit posséder au moins une clé
- Dans un SGBD, pour chaque relation une clé (la plus simple possible) est choisie, et est appelée clé primaire de la relation.
  - Si on définit pas de clé, alors on pourrait penser que canoniquement l'ensemble des attributs composant le domaine de la relation sera une clé. (pas tout à fait vrai : SQL gère des multi-ensembles)

Notations

- NOM DE LA RELATION, LISTE DES ATTRIBUTS AVEC DOMAINES, ET LISTE DES CLES D'UNE RELATION
- Exemple:
  - ETUDIANTS(NE: Int, NOM:texte, DATENAISS:entier, VILLE:texte, SECTION:texte)
  - Par convention, seule la clé primaire est soulignée
- INTENTION ET EXTENSION
  - Un schéma de relation définit l'intention de la relation
  - Une instance de table représente une extension de la relation
- SCHEMA D'UNE BD RELATIONNELLE
  - C'est l'ensemble des schémas des relations composantes
- PLUS DES CONTRAINTES (HP)
  - Contraintes d'intégrité
  - Dépendances fonctionnelles

Clé Etrangère (HP)

- GROUPE D'ATTRIBUTS DEVANT APPARAITRE COMME CLE DANS UNE AUTRE RELATION
- Les clés étrangères définissent les contraintes d'intégrité référentielles
  - Lors d'une insertion, la valeur des attributs doit exister dans la relation référencée
  - Lors d'une suppression dans la relation référencée les tuples référençant doivent disparaître
  - Elles correspondent aux liens entité-association obligatoires

Dépendance fonctionnelle (HP)

- Permet d'indiquer que certains attributs dépendent d'autres attributs
  - e.g. La valeur de la prime d'un employé dépend de son poste
  - i.e. SECTION→DEPARTEMENT ou (PRODUIT, QUANTITE) → PRIX

- Formes normales : on divise la relation en attributs clé et attributs non clé
  - 1ère forme normale
    - Tous les attributs sont monovalués (et la relation doit avoir une clé)
  - 2e forme normale
    - Les attributs non clé ne peuvent pas dépendre d'un sous-ensemble d'une clé candidate
  - 3e forme normale
    - Les attributs non clé ne peuvent pas dépendre d'un sous-ensemble d'attributs non clé
  - Boyce-Codd NF
    - Les attributs clé ne peuvent pas dépendre d'un sous-ensemble d'attributs non clé.

« The key, the whole key, nothing but the key, so help me Codd »

Permet de réaliser un « bon » schéma relationnel (éviter les redondances, et donc les problèmes de mise à jour).

Par contre ça demande de faire plus de jointures …

Exemple de Schéma

- EXEMPLE

  ETU(NE, NOM, DATENAISS, VILLE)

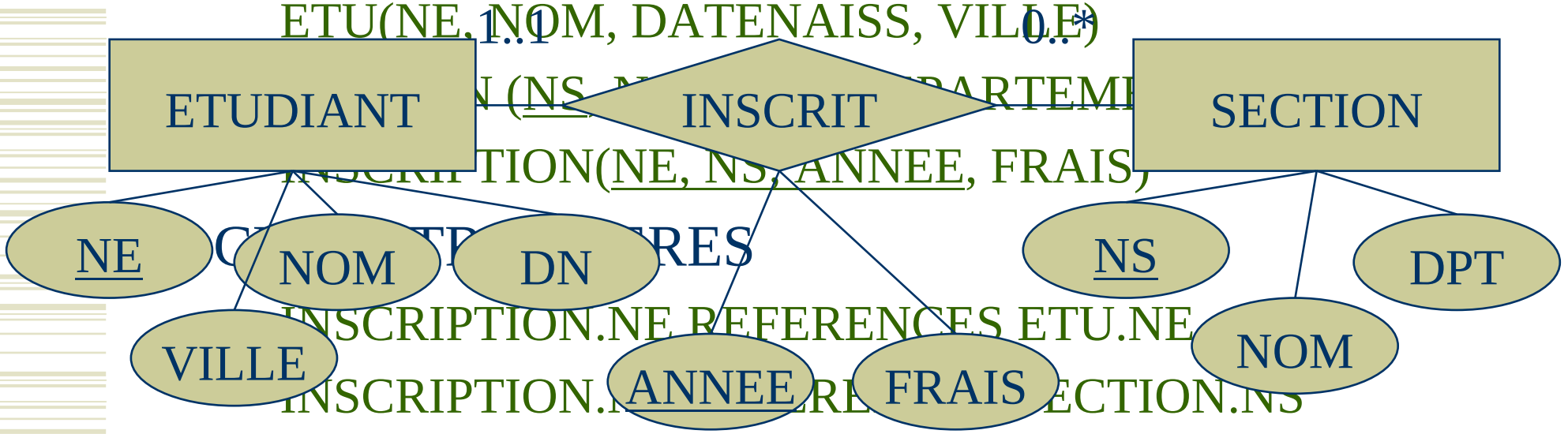
  [illegible]

  INSCRIPTION(NE, NS, ANNEE, FRAIS)

  [illegible]

  INSCRIPTION.NE REFERENCES ETU.NE

  INSCRIPTION.[illegible] ECTION.NS



- CLES CANDIDATES
  - UNIQUE(SECTION.NOMSEC)

Synthèse : Syntaxe SQL

- CREATION DES TABLES EN SQL

  CREATE TABLE <relation name>
  (<attribute definition>+)
  [{PRIMARY KEY | UNIQUE} (<attribute name>+)]

- avec :

  <attribute definition> ::= <attribute name> <data type>
  [NOT NULL [{UNIQUE | PRIMARY KEY}]]

- Exemple :

```
CREATE TABLE ETU
  (  NE   INTEGER  PRIMARY KEY,
     NOM VARCHAR (32),
     DATENAISS  INTEGER NOT NULL,
     VILLE VARCHAR(64) )
```

Les tables sont la plupart du temps créées via une interface.

Le code SQL sert à les créer« automatiquement »

Pourquoi tout ce blabla sur le modèle E/R ??

Méthodologie pour implémenter des entités et associations sous forme de tables

1. Transformer toutes les relations n-aires en relations binaires
2. Chaque entité devient une table
3. Les attributs correspondent aux colonnes des tables
    1. nom attribut ➔ nom colonne
    2. Ensemble de valeurs ➔ domaine
    3. Clé primaire E/A ➔ Clé primaire de la table
    4. Clé candidate E/A ➔ Contrainte UNIQUE de la table

Traduction des associations

- Règle de base
  - Une association est représentée par une table dont le schéma est le nom de l'association et la liste des clés primaires des entités participantes suivie des attributs de l'association. Ces clés primaires deviennent des clés étrangères de cette nouvelle table.
  - Exemples :
    - ACHETE (numproduit, numClient, Date)
    - FOURNIT (NumFournisseur, NumProduit, Prix, Remise)
- Amélioration possible
  - Regrouper les associations 1:N --> 1:1 avec la classe cible
  - Exemple :
    - VOITURE (N°VEH, MARQUE, TYPE, PUISSANCE, COULEUR)
    - POSSEDE (N° SS, N° VEH, DATE , PRIX)
    - regroupées si toute voiture a un et un seul propriétaire

2. CONCEPTS MANIPULATOIRES

- Un ensemble d'expressions formelles
  - Calcul relationnel (déclaratif = impossible à implémenter, facile à énoncer)
  - Algèbre relationnelle (procédural = possible à implémenter, dur à énoncer)
- Ces opérations permettent d'exprimer toutes les requêtes sous forme d'expressions algébriques *qui pourront être exécutées et optimisées*
- Elles sont la base du langage SQL qui est un langage **déclaratif**
  - Paraphrasage en anglais des expressions du calcul relationnel
- Ces opérations sont extensibles

Les requêtes du calcul relationnel

- Le calcul relationnel se décline de deux manières
  - Calcul relationnel de tuples
  - Calcul relationnel de domaines
- Ce sont des fragments de la logique du premier ordre

e.g.

$\exists ne, nometu, date, ville \mid Etudiant(ne, nometu, date, ville) \wedge \exists ns, nomsec \mid Section(ns, nomsec) \wedge \exists annee \mid Inscrit(ne, ns, annee)$

Calcul Relationnel : Atomes

- Les formules atomiques (ou atomes) sont des formules terminales (i.e. elles n'incluent aucune autre proposition)
- Soit *V* un ensemble de variables (à valeurs dans l'ensemble des tuples)
- Les atomes autorisés sont les suivants :
  - si v∈ V, w∈ V, a∈ dom(v), b∈ dom(w) alors v.a = w.b est un atome
    - *e.numetu = i.numetu*
  - si v∈ V, a∈ dom(v), *k*∈ *D* alors v.a = *k* est un atome
    - *e.ville = Versailles*
  - si v∈ V, *r*∈ *R*, dom(*v*)=*h*(*r*) alors *r*(*v*) est un atome
    - Etudiant(e)
- La sémantique formelle est définie étant donnée une base de données et une affectation des variables à des tuples.

Calcul Relationnel : Formules et Requêtes

- Les atomes peuvent être combinées en formules selon la grammaire suivante :
  - Atome = {v.a = v.b | v.a = k | r(v)}
  - Formule = {Atome | $Formule_1 \wedge Formule_2$ | $Formule_1 \vee Formule_2$ | $\neg Formule_1$ | $\exists v{:}H\ (Formule_1)$ | $\forall v{:}H\ (Formule_1)$}
    - $\forall\ t$: {ne, nom, prenom, ville} (Etudiant(t) $\wedge$ t.nom = « Toto » $\wedge$ $\neg$ (t.ville = « Paris »)

    Signifie : *tous les étudiant s'appelant Toto habitent ailleurs qu'à Paris*
- Une requête est de la forme :
  - {v : H | *Formule(v)*}
    - {e : {nom} | $\exists$t : {ne, nom, prenom, ville} (t.nom = e.nom $\wedge$ Etudiant(t) $\wedge$ t.ville = « Versailles »}

    Signifie : Trouver le nom de tous les étudiants habitant Versailles

    /!\ On ne gère pas les agrégats, il faut étendre le formalisme

A quoi ça sert ?

A EXPRIMER FACILEMENT DES REQUETES !

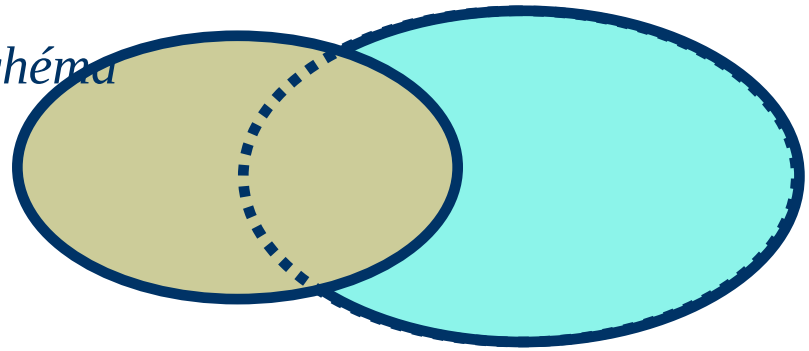
(pour des mathématiciens/logiciens ☺)

Comme c'est pas si simple pour le commun des mortels, on a inventé SQL ! (on va voir ça plus tard)

Les opérations de l'algèbre relationnelle

Opérations Ensemblistes Classiques

- Opérations binaires *pour des relations de même schéma*
  - UNION notée ∪
  - INTERSECTION notée ∩
  - DIFFERENCE notée —
- Opérations binaires *pour des relation de schéma différents*
  - Produit cartésien
- Pas d'opérations unaires
- Extension
  - Union externe pour des relations de schémas différents
  - Ramener au même schéma avec des valeurs nulles



Exemple de Produit Cartésien

R=ETU × INFOVILLE

ETU	NOM	DN	VILLE
	ANNE	1991	VERSAILLES
	BERNARD	1993	PARIS
	CELINE	1993	PARIS
	DAVID	1991	VERSAILLES

×

INFOVILLE	NOMV	DPT
	VERSAILLES	78
	PARIS	75

R	NOM	DN	VILLE	NOMV	DPT
	ANNE	1991	VERSAILLES	VERSAILLES	78
	ANNE	1991	VERSAILLES	PARIS	75
	BERNARD	1993	PARIS	VERSAILLES	78
	BERNARD	1993	PARIS	PARIS	75
	CELINE	1993	PARIS	VERSAILLES	78
	CELINE	1993	PARIS	PARIS	75
	DAVID	1991	VERSAILLES	VERSAILLES	78
	DAVID	1991	VERSAILLES	PARIS	75

Projection

- Elimination des attributs non désirés et suppression des tuples en double
- Relation → Relation notée:

$\pi_{A1,A2,...Ap}(R)$

ETU	NOM	DN	VILLE
	ANNE	1991	VERSAILLES
	BERNARD	1993	PARIS
	CELINE	1993	PARIS
	DAVID	1991	VERSAILLES
	EMILIE	1993	VELIZY

$\pi_{DN,VILLE}(ETU)$

$\pi_{DN,VILLE}(ETU)$	DN	VILLE
	1991	VERSAILLES
	1993	PARIS
	1993	VELIZY

Restriction

- Obtention des tuples de R satisfaisant un critère Q
- Relation → Relation, notée $\sigma_Q(R)$
- Q est le critère de qualification de la forme :
  - $A_i\ \theta$ Valeur
  - $\theta \in$ { =, <, >=, >, <=, !=}

- Il est possible de réaliser des "ou" (union) et des "et" (intersection) de critères simples

Exemple de Restriction

ETU	NOM	DN	VILLE
	ANNE	1991	VERSAILLES
	BERNARD	1993	PARIS
	CELINE	1993	PARIS
	DAVID	1991	VERSAILLES
	EMILIE	1993	VELIZY

$\sigma_{DN>1992}$ (ETU)

ETU	NOM	DN	VILLE
	BERNARD	1993	PARIS
	CELINE	1993	PARIS
	EMILIE	1993	VELIZY

Jointure

- Composition des deux relations sur un domaine commun
- Relation X Relation ->Relation
  - notée $\bowtie$
- Critère de jointure
  - Attributs de même nom égaux :
    - Attribut = Attribut
    - Jointure naturelle
    - **Se fait en principe en utilisant une clé étrangère !!!**
  - Comparaison d'attributs :
    - Attribut1 θ Attribut2
    - Théta-jointure
- La jointure peut se voir comme un produit cartésien, combiné à une restriction sur l'attribut de jointure, puis d'une projection pour éliminer l'attribut doublon.

Exemple de Jointure

ETU	NOM	DN	VILLE
	ANNE	1991	VERSAILLES
	BERNARD	1993	PARIS
	CELINE	1993	PARIS
	DAVID	1991	VERSAILLES

ETU.VILLE = INFOVILLE.NOMV

INFOVILLE	NOMV	DPT
	VERSAILLES	78
	PARIS	75

R=ETU  INFOVILLE

ETU.VILLE = INFOVILLE.NOMV

R	NOM	DN	VILLE	DPT
	ANNE	1991	VERSAILLES	78
	BERNARD	1993	PARIS	75
	CELINE	1993	PARIS	75
	DAVID	1991	VERSAILLES	78

Jointure et Produit Cartésien

$R_1 \underset{A = B}{\bowtie} R_2$ Équivaut à : $\sigma_{A=B}(R_1 \times R_2)$

R	NOM	DN	VILLE	NOMV	DPT
	ANNE	1991	VERSAILLES	VERSAILLES	78
	~~ANNE~~	~~1991~~	~~VERSAILLES~~	~~PARIS~~	~~75~~
	~~BERNARD~~	~~1993~~	~~PARIS~~	~~VERSAILLES~~	~~78~~
	BERNARD	1993	PARIS	PARIS	75
	~~CELINE~~	~~1993~~	~~PARIS~~	~~VERSAILLES~~	~~78~~
	CELINE	1993	PARIS	PARIS	75
	DAVID	1991	VERSAILLES	VERSAILLES	78
	~~DAVID~~	~~1991~~	~~VERSAILLES~~	~~PARIS~~	~~75~~

Division

- L'opération de division permet d'implémenter le « quel que soit » de la logique du premier ordre.
  - L'opérateur est défini pour deux relation R et S telle que dom(S) $\subset$ dom(R) et s'écrit : $Q=R \div S$
  - Q a la propriété suivante : $Q \times S \subset R$
- Sémantique formelle : On note $dom(R) = \{r_1..r_i, s_1..s_j\}$ tel que $dom(S) = \{s_1..s_j\}$
  - $R \div S = \{ t:\{r_1,...,r_n\} \mid t \in R \wedge \forall s \in S ((t[r_1,...,r_n] \times s) \in R) \}$
- L'opération est peu utilisé en pratique
- L'opérateur en lui-même n'existe pas en SQL
- L'opération $Q=R \div S$ se réalise par :
  - $T = \Pi_{r1..ri}(R) \times S$ (tous les tuples possibles)
  - $U = T - R$ (les tuples qui ne sont pas dans R)
  - $V = \Pi_{r1..ri}(U)$
  - $Q = \Pi_{r1..ri}(R) - V$

Completed

Student	Task
Fred	Database1
Fred	Database2
Fred	Compiler1
Eugene	Database1
Eugene	Compiler1
Sarah	Database1
Sarah	Database2

DBProject

Task
Database1
Database2

Completed ÷ *DBProject*

Student
Fred
Sarah

A quoi ça sert ?

A EXECUTER

- Une expression de l'algèbre relationnelle peut se lire de manière fonctionnelle comme un *plan d'exécution de la requête*. i.e. si on a implémenté les opérateurs de l'algèbre relationnelle, et qu'on utilise des structures de type *relation* il me suffit de les appeler en passant en paramètre les relations en question !
- Les plans sont souvent représentés sous forme d'arbre.

A OPTIMISER

- Il est important de noter que certaines opérations *sont commutatives*, c'est la base de l'optimisation des requêtes.

Pourquoi ça marche ? Complétude Relationnelle

- **Théorème de Codd** : l'algèbre relationnelle a un pouvoir expressif équivalent à celui du calcul relationnel. (article *Relational completeness of data base sublanguages*)
  - Les cinq (sept) opérations de base permettent de formaliser sous forme d'expressions toutes les questions que l'on peut poser avec la logique du premier ordre.
- Exemple :
  - Nom et Section des étudiants de Versailles nés en 1991 ?

Algèbre Relationnelle :

$\Pi_{NOM, NOMSEC}\ (\sigma_{DATEN=1991\ ET\ VILLE=«\ VERSAILLES\ »}(ETU \bowtie INSCR \bowtie SECTION)$

Calcul Relationnel :

$\{t:\{nom, nomsec\}\ |\ \exists e:\{ne, nom, ville, datenaiss\}, i:\{ne, ns\}, s:\{ns, nomsec\}\ (t.nom = e.nom \wedge t.nomsec = s.nomsec \wedge i.ne=e.ne \wedge i.ns=s.ns \wedge e.datenaiss = 1991 \wedge e.ville = « Versailles »)$

Et SQL ?

- Une requête SQL (donc une description en langage « naturel ») peut se traduire sous la forme d'une expression de *l'algèbre relationnelle* (en fait SQL va plus loin : en particulier avec les agrégations)
- Requête élémentaire :

  SELECT A_1, A_2, …A_p
  FROM R_1, R_2, …R_k
  WHERE Q [{UNION |INTERSECT | EXCEPT } …]
- Sémantique du bloc select :

  $\Pi_{A1,A2,...Ap}$ (σ_Q ($R_1 \times R_2 \times \ldots \times R_k$)))

  /!\ UN PRODUIT CARTESIEN N'EST PAS UNE JOINTURE !!!

- Et pourtant SQL est déclaratif … ?

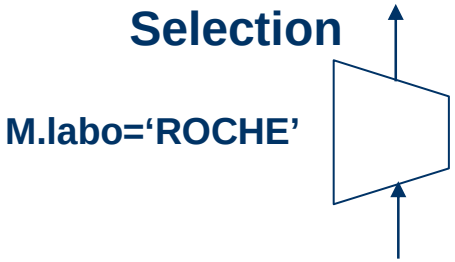
  $\{t:\{A_1, A_2, \ldots A_n\} \mid \exists x_1:\{\ldots\}, x_2:\{\ldots\}, \ldots x_n:\{\ldots\}_k (Q(t, x_1, x_2, \ldots x_n) \wedge$
  LIEN$(t, x_1, x_2, \ldots x_n)\}$

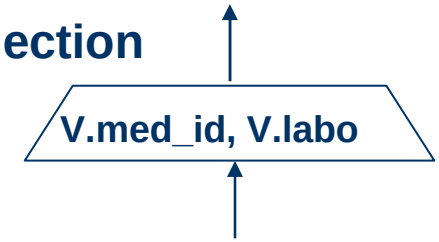
- Tout le problème réside dans la manière de calculer Q !

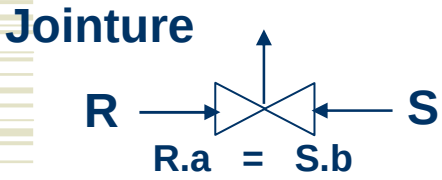
Principe de fonctionnement du moteur d'exécution du SGBD

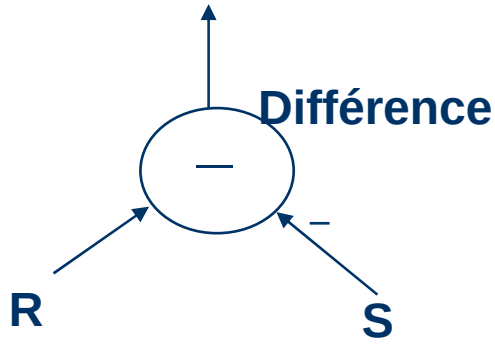
1. On écrit sa requête en SQL
2. Cette requête a une sémantique formelle donnée par le calcul relationnel
3. Cette requête est traduite (et optimisée) en utilisant l'algèbre relationnelle

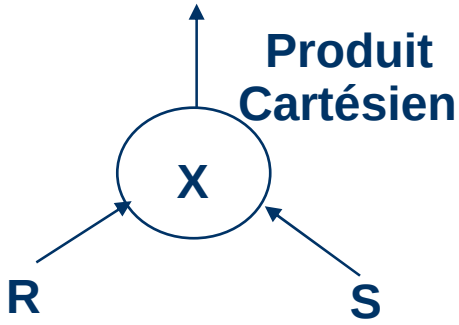
Exemple d'exécution et optimisation (HP)

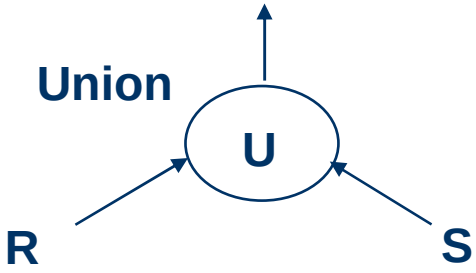








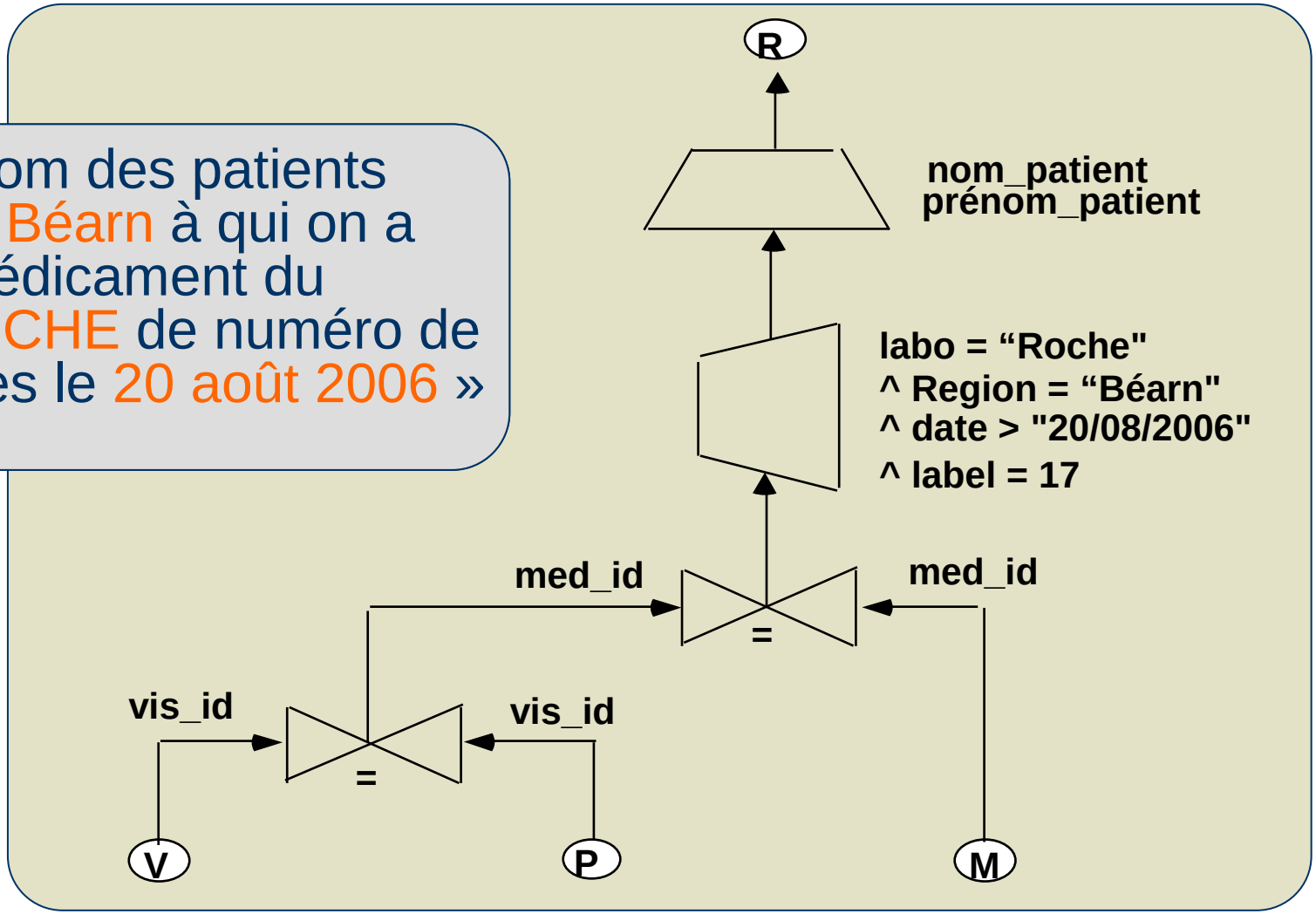




Ex. Plan d'exécution candidat (1)

Requête

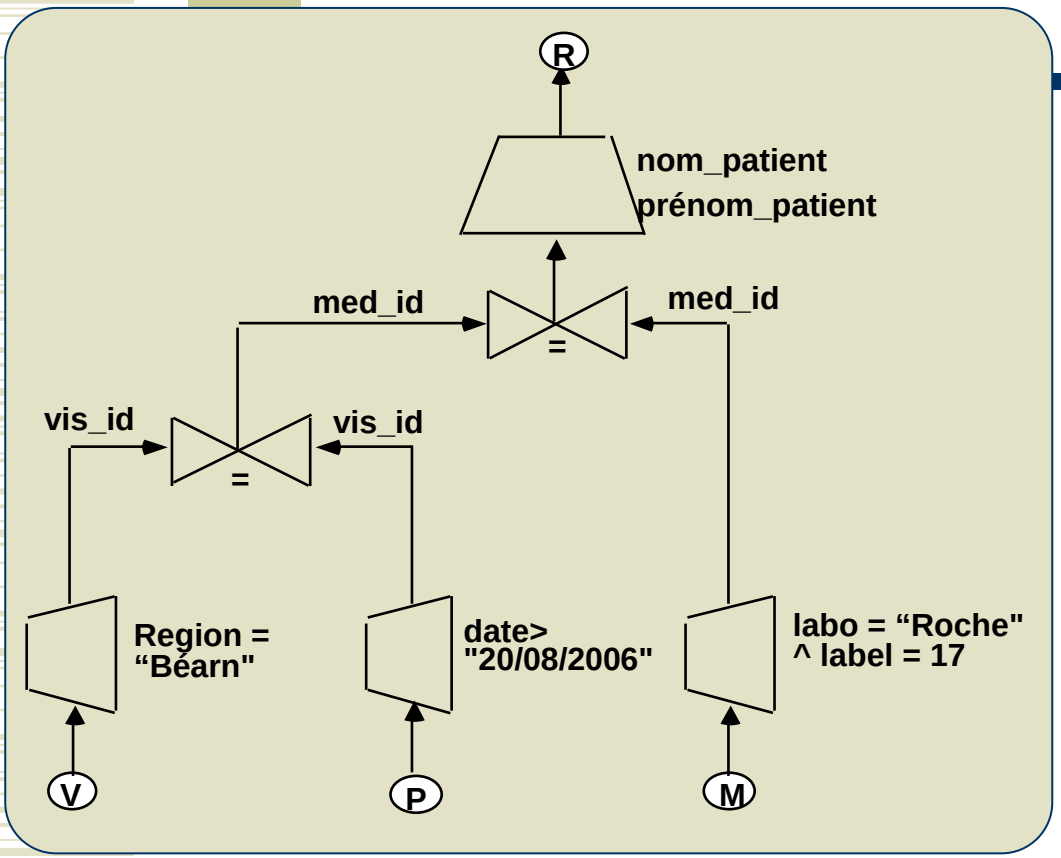
« Nom et prénom des patients visités dans le Béarn à qui on a prescrit des médicament du laboratoire ROCHE de numéro de label = 17 après le 20 août 2006 »



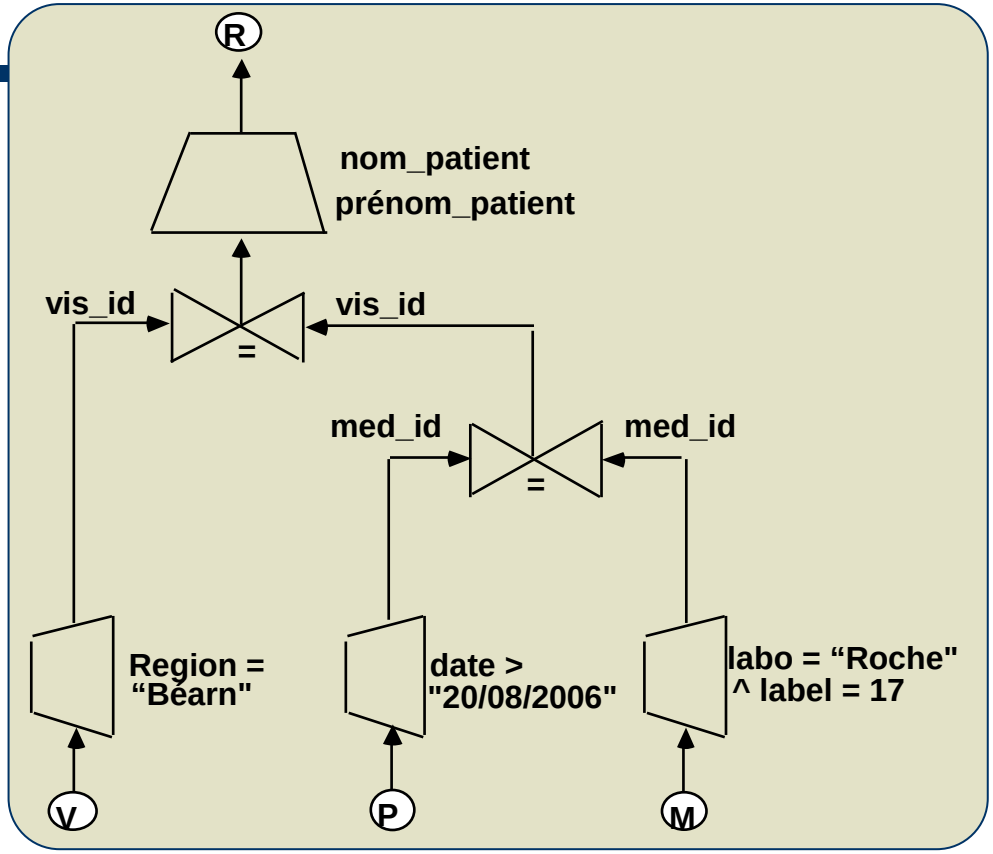
Plan candidat N°1

Ex. Plan d'exécution candidat (2)

Plan candidat N°2



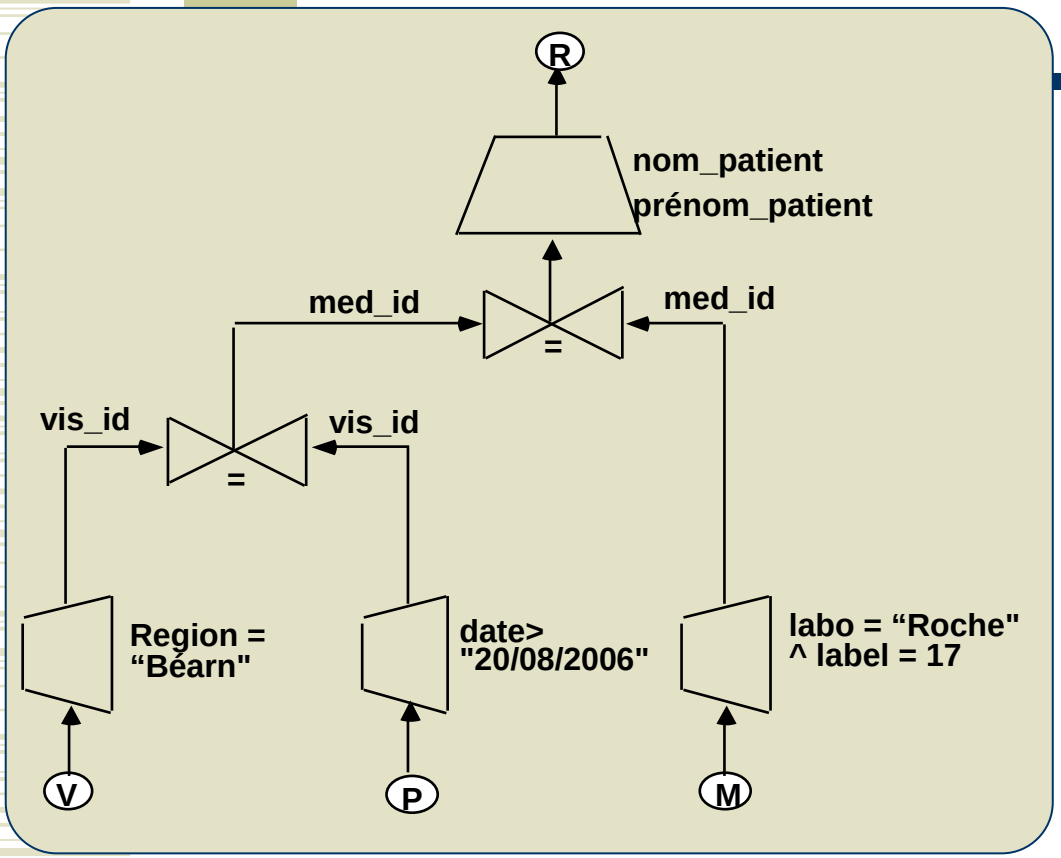
Plan candidat N°3



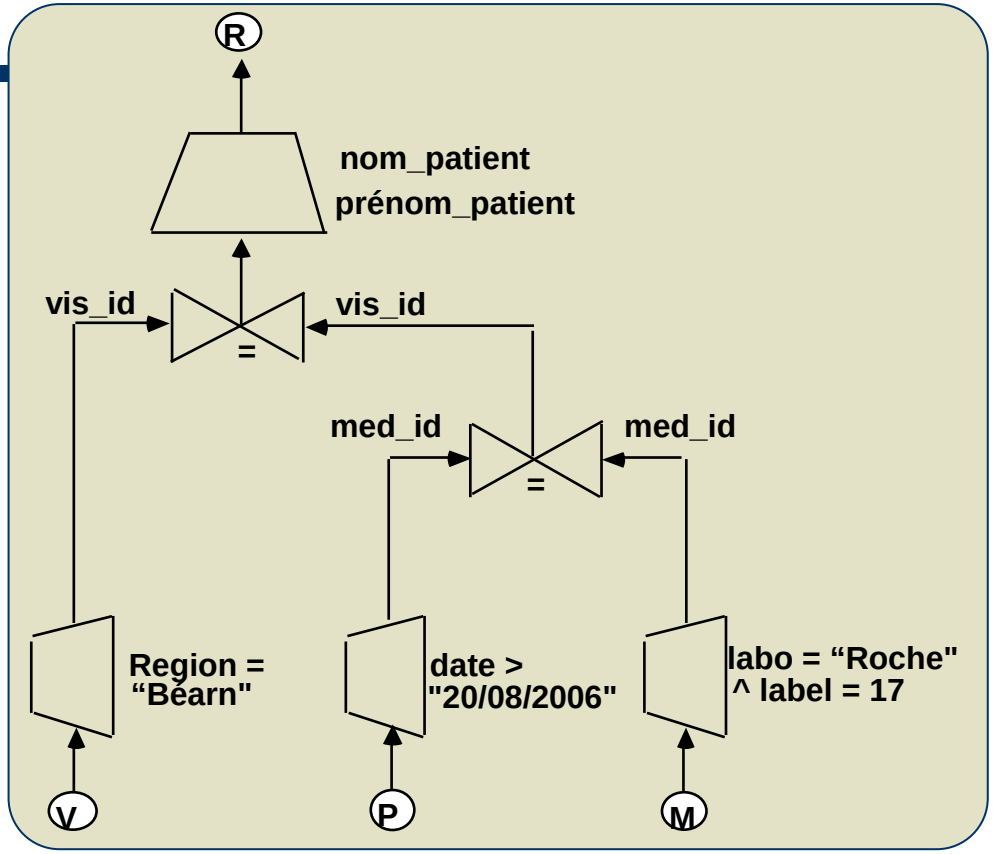
De ces 3 arbres, lequel est le meilleur ?

Ex. Plan d'exécution candidat (2)

Plan candidat N°2



Plan candidat N°3



De ces 3 arbres, lequel est le meilleur ?

Le premier est sûrement moins bon, mais les 2 derniers ?

3. CONCEPTS ADDITIONNELS

- Ensemble de concepts pour :
  - Etendre les fonctionnalités de manipulation
  - Décrire les règles d'évolution des données
  - Supporter des objets complexes (SQL3)
- Introduits progressivement dans le modèle :
  - Complique parfois le modèle
  - Standardisés au niveau de SQL3 (1999)
  - Des extensions multiples …

Renommage

- Pour changer le nom d'une colonne.
- Notation simple en algèbre relationnelle:

$$\rho_{A \rightarrow B}(R)$$

- Exemple : ETU2 = $\rho_{DN \rightarrow DATEN}(ETU1)$

ETU1	NOM	DN	VILLE
	BERNARD	1993	PARIS
	CELINE	1993	PARIS
	EMILIE	1993	VELIZY

ETU2	NOM	DATEN	VILLE
	BERNARD	1993	PARIS
	CELINE	1993	PARIS
	EMILIE	1993	VELIZY

Fonctions et Agrégats

- FONCTION
  - Fonction de calcul en ligne appliquée sur un ou plusieurs attributs
  - Exemple : MUTUELLE = FRAIS*15/100
- AGREGAT

  Partitionnement horizontal d'une relation selon les valeurs d'un groupe d'attributs (B_i), suivi d'un regroupement par une (ou plusieurs) fonction(s) F_i de calcul en colonne (SUM, MIN, MAX, AVG, COUNT, …) sur les attributs C_i respectifs
- NOTATION : gamma minuscule

$$\underbrace{B_1, B_2, \ldots, B_N}_{\text{Ensemble des colonnes}} \gamma_{F_1(C_1), F_2(C_2), \ldots, F_N(C_N)} (R)$$

Exemples d'agrégats

ETU	NOM	AGE	VILLE
	ANNE	21	VERSAILLES
	BERNARD	19	PARIS
	CELINE	19	PARIS
	DAVID	20	VERSAILLES

$\gamma_{AVG(AGE)}(ETU)$

AVG(AGE)
19.75

$VILLE\ \gamma_{MAX(AGE)}(ETU)$

VILLE	MAX(AGE)
VERSAILLES	21
PARIS	19

SQL :

```
SELECT AVG(AGE)
FROM ETU;
```

SQL :

```
SELECT VILLE, MAX(AGE
FROM ETU
GROUP BY VILLE;
```

/!\ le HAVING se fait tout simplement avec un σ

Vue (HP)

- Relation d'un schéma externe déduite des relations de la base par une question
- Exemple : Etudiants Versaillais

```
CREATE VIEW  ETUVERSAILLAIS AS
SELECT NE, NOM, NOMSECTION
FROM ETU E, INSCRIPTION I, SECTION S
WHERE E.NE = I.NE AND I.NS=S.NS
      AND E.VILLE = « VERSAILLES »
```

- Calcul de la vue
  - Une vue est une fenêtre dynamique sur la BD et est recalculée à chaque accès.
  - Une vue peut être matérialisée (vue concrète) pour accélérer les calculs l'utilisant. Dans ce cas, le SGBD doit être capable de savoir quand recalculer la vue.

Déclencheur (Trigger) (HP)

- Action base de données déclenchée suite à l'apparition d'un événement particulier
- Forme :
  - {BEFORE | AFTER} <événement> THEN <action>
  - Un événement peut être :
    - une opération sur une table (début ou fin)
    - un événement externe (heure, appel,etc.)
  - Une action peut être :
    - une requête BD (mise à jour)
    - Une annulation (abort) de transaction
    - l'appel à une procédure cataloguée

Déclencheur avec condition (Règle) (HP)

- Il est possible d'ajouter une condition afin de déclencher l'action seulement quand la condition est vérifiée
  - Une condition est une qualification portant sur la base.
- Exemples :

```
BEFORE          UPDATE  EMPLOYE
IF              SALAIRE > 100.000
THEN    ABORT TRANSACTION
```

4. CONCLUSION

- Un ensemble de concepts bien compris et bien formalisés
- Un modèle unique, riche et standardisé
  - intégration des BD actives
  - intégration des BD objets
  - intégration des BD XML
- Un formalisme qui s'étend plutôt bien
  - algèbre d'objets
- Un langage associé défini à plusieurs niveaux
  - SQL1, 2, 3, 2003, etc.

LES REQUETES PAR L'EXEMPLE
SQL, Calcul Relationnel, Algèbre Relationnelle

- Origines et Evolutions
- SQL1 86: la base
- SQL1 89: l'intégrité

Jim Melton (Oracle)
Editeur de la norme SQL

1. Origines et Evolutions

- SQL est **une manière simple** d'écrire une formule (requête) du **calcul relationnel**. Tout comme le calcul relationnel, une requête SQL peut être traduite en un expression de l'**algèbre relationnelle**.
- Il existe plusieurs versions normalisées, du simple au complexe :
  - SQL-86 version minimale
  - SQL-89 addendum (intégrité)
  - SQL2 (92) langage complet
  - SQL3 (99) aspects objet, triggers
  - SQL:2003 introduction d'aspects XML
  - SQL:2006 intégration du début de XQuery
  - SQL:2008 modifications mineures (instead of, truncate)
  - SQL:2011 améliorations XQuery
- La plupart des systèmes supportent SQL2 ou SQL3

Opérations

- Opérations de base
  - SELECT, INSERT, UPDATE, DELETE
  - /!\ Seul le **SELECT** (ou « SFWGH ») correspond au Calcul Relationnel. SQL est « relationnellement complet » (permet d'exprimer toutes les requêtes du Calcul et de l'Algèbre.
- Opérations additionnelles
  - définition et modification de schémas
  - définition de contraintes d'intégrité
  - définition de vues
  - accord des autorisations
  - gestion de transactions

Organisation du Langage

SQL comprend quatre parties :

1. **Le langage de définition de schéma (Tables, Vues, Droits)**
2. **Le langage de manipulation (Sélection et mises à jour)**
3. *La spécification de modules appelables (Procédures)*
4. *L'intégration aux langages de programmation (Curseurs)*

SQL 86

- LANGAGE DE DEFINITIONS DE DONNEES
  - CREATE TABLE
  - CREATE VIEW
- LANGAGE DE MANIPULATION DE DONNEES
  - SELECT OPEN
  - INSERT FETCH
  - UPDATE CLOSE
  - DELETE
- LANGAGE DE CONTROLE DE DONNEES
  - GRANT et REVOKE
  - BEGIN et END TRANSACTION
  - COMMIT et ROLLBACK
- SQL EST « **COMPLETEMENT UTILISABLE** »

Base de Données

- Collection de tables et de vues dans un schéma

TABLES

RESPONSABLE (NR, NOM, PRENOM, DPT)

COURS (NC, CODE_COURS, INTITULE, ECTS, NR, DPT)

ETUDIANT (NE, NOM, PRENOM, VILLE, AGE)

INSCRIT (NE, NC,ANNEE)

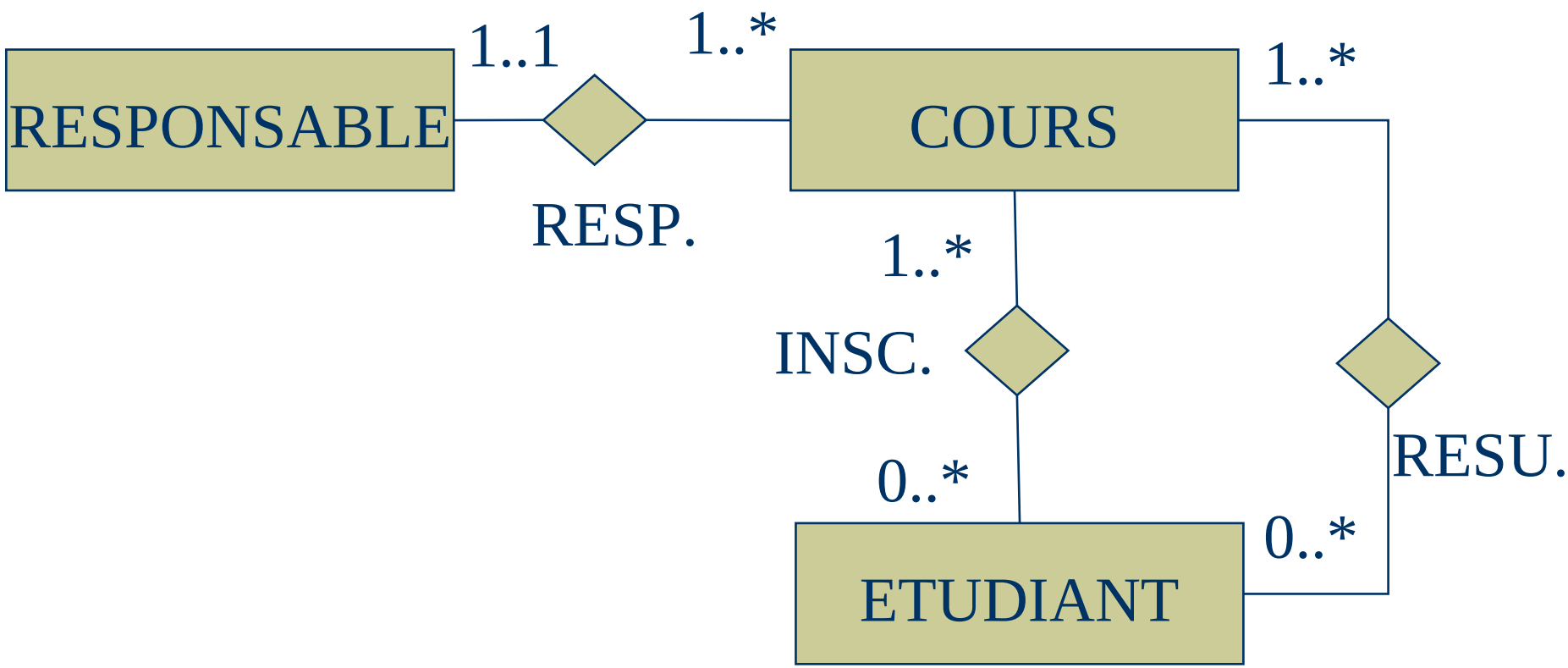
RESULTAT (NE, NC, ANNEE, NOTE)

VUES

ADMIS (NE, NC, ANNEE)

COURS_DPT (DPT, NC, INTITULE)

Schéma E/A “allégé” (sans attributs, rôles)



2. SELECT: Forme Générale

SELECT <liste de projection>
FROM <liste de tables>
[WHERE <critère de jointure> AND <critère de restriction>]
[GROUP BY <attributs de partitionnement>]
[HAVING <citère de restriction>]

- Restriction :
  - arithmétique (=, <, >, <>, >=,<=□)
  - textuelle (LIKE)
  - sur intervalle (BETWEEN)
  - sur liste (IN)
- **Possibilité de blocs imbriqués par :**
  - **IN, EXISTS, NOT EXISTS, ALL, SOME, ANY**

Forme générale de la condition

<**search condition**> ::= **[**NOT**]**

<**nom_colonne**> θ **constante** | <**nom_colonne**>

<**nom_colonne**> LIKE **<modèle_de_chaîne>**

<**nom_colonne**> IN <**liste_de_valeurs**>

<**nom_colonne**> θ **(**ALL | ANY | SOME**)** <**liste_de_valeurs**>

EXISTS <**liste_de_valeurs**>

UNIQUE <**liste_de_valeurs**>

<**tuple**> MATCH **[**UNIQUE**]** <**liste_de_tuples**>

<**nom_colonne**> BETWEEN **constante** AND **constante**

<**search condition**> AND | OR <**search condition**>

avec

θ ::= < | = | > | ≥ |≤ | <>

Remarque: <**liste_de_valeurs**> **peut être dynamiquement déterminée par une requête**

Exemples de Questions (1)

- Q1: Liste des non
  SELECT NOM, PR
  FROM ETUDIANT
  (T:{NOM, PRENOM}|∃E:{NOM, PRENO
  (ETUDIANT(E) ∧E.NOM = T.NOM∧
  E.PRENOM=T.PRENOM)
- Q2: Noms des étudiants inscrits en IN311 en 2007 ou 2008

Π SELECT NOM Projection

× (T:{NOM}|∃E:{NOM, NE}(∃C:{NC, NE, CODE_COURS}(

▷ ∃I:{NC, NE, ANNEE}(ETUDIANT(E) ∧COURS(C)∧INSCR

T.NOM=E.NOM∧

C E.NE=C.NE∧C.NC=I.NC∧E.NE=I.NE ∧

(I.ANNEE=2007∨I.ANNEE=2008) ∧

C.CODE_COURS=« IN311 »)))

Avec la jointure dans le FROM

- Q2: Noms des étudiants inscrits en IN311 en 2011 ou 2012

```
SELECT NOM
⋈  FROM (ETUDIANTS E JOIN INSCRIT I ON E.NE
     =I.NE) JOIN COURS C ON I.NC=C.NC
WHERE
σ  C.CODE_COURS  LIKE '%IN311%'
   AND I.ANNEE IN (2011, 2012)
```

Exemples de Questions (2)

- Q3 : Noms et prénoms des étudiants inscrits à des cours dont le code commence par IN, entre 2009 et 2012.

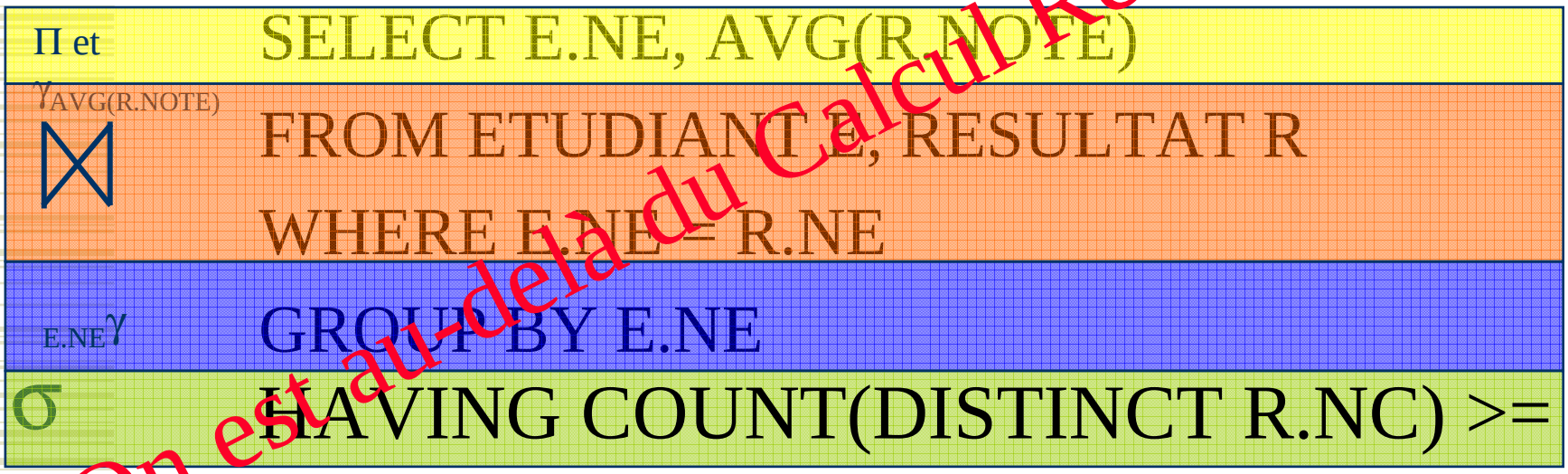
Π	SELECT NOM, PRENOM
$\times$	FROM ETUDIANT E, INSCRIT I, COURS C
$\bowtie$	WHERE E.NE = I.NE AND I.NC = C.NC
σ	AND C.CODE_COURS LIKE "IN%" AND (I.ANNEE BETWEEN 2009 AND 2012)

- Q4 : Code des modules suivis par au moins un etudiant. (requête imbriquée)

$$(T:\{CODE\_COURS\}|\exists E:\{NOM, NE\}(\exists C:\{NC, NE, CODE\_COURS\}(\exists I:\{NC, NE, ANNEE\}(ETUDIANT(E) \wedge COURS(C) \wedge INSCRIT(I) \wedge T.CODE\_COURS=C.CODE\_COURS \wedge E.NE=C.NE \wedge C.NC=I.NC \wedge E.NE=I.NE)$$

Exemples de requêtes agrégat

- Q5 : Calculez la moyenne de chaque étudiant, référencé par son NE



On est au-delà du Calcul Relationnel de Tuples

Pour les étudiants ayant suivi plus de 5 modules …

Exemples de requêtes agrégat

- Q5’ : Calculez la note maximale de chaque module, référencé par son NC

Π et $\gamma_{MAX(R.NOTE)}$ ⋈	SELECT C.NC, MAX(R.NOTE) FROM COURS C, RESULTAT R WHERE C.NC = R.NC
$_{E.NE}\gamma$	GROUP BY C.NC
σ	HAVING COUNT(DISTINCT R.NE) >= 15

Pour les modules de plus de 15 étudiants…

Exemples de Requêtes agrégat

- Q6: Calculer l'age du plus jeune étudiant

Π et $\gamma_{MIN(AGE)}$ SELECT MIN(AGE)
FROM ETUDIANT

- Q7 : Calculer le nombre d'étudiants, ainsi que l'age de l'étudiant le plus vieux reçus par module, pour les modules du dpt INFO dont la moyenne globale est supérieure à 12.

Π et $\gamma_{COUNT(*),\ MAX(AGE)}$ SELECT C.NC, COUNT(*), MAX(E.AGE)
$\times$ FROM ETUDIANT E, RESULTAT R, COURS C
$\bowtie$ WHERE E.NE = R.NE AND R.NC = C.NC
σ AND C.DPT = "INFO"
$_{C.NC}\gamma$ GROUP BY C.NC
HAVING AVG(R.NOTE) > 12

Requêtes agrégat et fonctions

- Q8 : Donnez le nombre d'ECTS obtenus par chaque étudiant, référencé par son NE, dans une colonne appelée CREDITS.

CALCUL DE FONCTION

$\rho_{SUM(R.ECTS) \rightarrow CREDITS}$

```
SELECT E.NE, SUM(R.ECTS)     AS CREDITS
FROM ETUDIANT E, RESULTAT R
σ  WHERE E.NE = R.NE
   AND R.NOTE >= 10
   GROUP BY E.NE
```

/!\ ICI ON NE PREND PAS EN COMPTE LA COMPENSATION !

Requêtes agrégat et fonctions

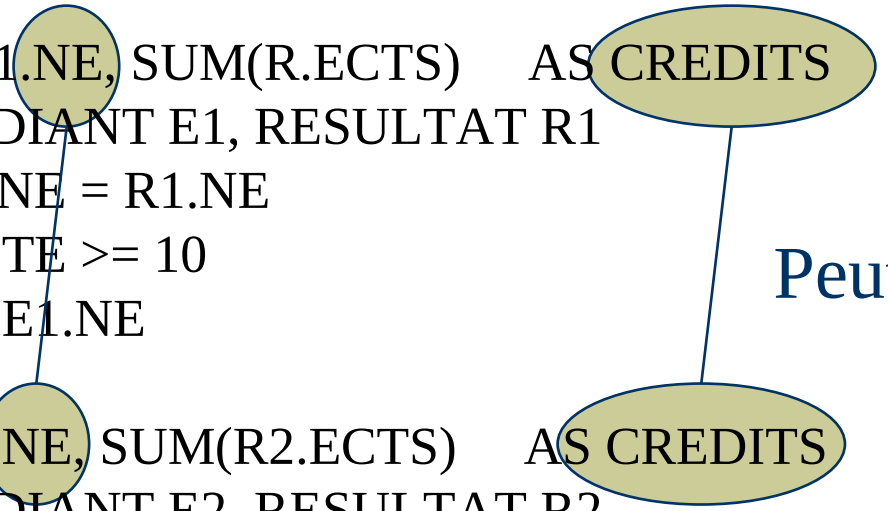
- Q8' : Donnez le nombre d'ECTS obtenus par chaque étudiant, référencé par son NE, dans une colonne appelée CREDITS.

```
SELECT E.NE, SUM(R.ECTS)     AS CREDITS
FROM ETUDIANT E, RESULTAT R
WHERE E.NE = R.NE
GROUP BY E.NE
σ HAVING AVG(R.NOTE) >= 10
```

/!\ COMBIEN D'ECTS A L'ETUDIANT S'IL N'A PAS LA MOYEN
AU SEMESTRE ?

Requêtes agrégat et fonctions

```
Q9 : SELECT CALCUL.NE, MAX(CALCUL.CREDITS)
FROM
( SELECT E1.NE, SUM(R.ECTS)     AS CREDITS
FROM ETUDIANT E1, RESULTAT R1
WHERE E1.NE = R1.NE
AND R1.NOTE >= 10
GROUP BY E1.NE
UNION
SELECT E2.NE, SUM(R2.ECTS)     AS CREDITS
FROM ETUDIANT E2, RESULTAT R2
WHERE E2.NE = R2.NE
GROUP BY E2.NE
HAVING AVG(R2.NOTE) >= 10 ) AS CALCUL
GROUP BY CALCUL.NE
```



Peut on faire sans union

Requêtes agrégat et fonctions

CALCUL DE FONCTION

```
SELECT CALCUL.NE, GREATEST(NORMAL.CREDITS, COMPENSE.CREDITS)
AS CREDITS                                   ρ→CREDITS
FROM
( SELECT E1.NE, SUM(R.ECTS)    AS CREDITS
FROM ETUDIANT E1, RESULTAT R1
WHERE E1.NE = R1.NE                          Table « NORMAL »
AND R1.NOTE >= 10
GROUP BY E1.NE ) AS NORMAL,
SELECT E2.NE, SUM(R2.ECTS)    AS CREDITS
FROM ETUDIANT E2, RESULTAT R2
WHERE E2.NE = R2.NE                          Table « COMPENSE »
GROUP BY E2.NE
HAVING AVG(R2.NOTE) >= 10 ) AS COMPENSE
WHERE NORMAL.NE = COMPENSE.NE                ⋈
GROUP BY CALCUL.NE                           CALCUL.NE γ
```

Requêtes imbriquées (1)

- Q10: Donner les CODE_COURS des cours qui n'ont aucun inscrit

```
SELECT CODE_COURS
FROM COURS C
WHERE C.NC NOT IN
( SELECT I.NC
  FROM INSCRIT I )
```

```
SELECT CODE_COURS
FROM COURS C
WHERE C.NC <> ALL
( SELECT I.NC
  FROM INSCRIT I )
```

On ne se ressert pas forcément de la requête extérieure

Requêtes imbriquées (2)

- Q11 : Donner le NE des étudiants qui ne suivent pas tous les cours

```
SELECT NE
FROM ETUDIANT E
WHERE EXISTS (
        SELECT *
        FROM  COURS C
        WHERE NOT EXISTS (
                SELECT *
                FROM INSCRIT I
                WHERE C.NC = I.NC
                AND I.NE = E.NE) )
```

Requête doublement imbriquée !

Utilisation de SQL depuis un langage de prog.

- Il est très fréquent d'utiliser SQL à partir de programmes
  - Applications C++/Java/etc.
  - Applications Web (PHP, etc.)
- L'utilisation de bibliothèques JDBC est conseillée
- … plus de détails dans le cours sur PHP et Java.

3. Les Mises à Jour (HP)

- INSERT
  - Insertion de lignes dans une table
  - Via formulaire où via requêtes
- UPDATE
  - Modification de lignes dans une table
- DELETE
  - Modification de lignes dans une table

Commande INSERT

- INSERT INTO <relation name>
  [(attribute [,attribute] …)]
  {VALUES <value spec.> [, <value spec.>] …| <query spec.>}
- Exemples

```
INSERT INTO ETUDIANT (NE, NOM, PRENOM, VILLE, AGE)      VALUES
(112, 'MARTIN', 'THOMAS' , 'VERSAILLES', 20)

INSERT INTO RESULTAT (NC,NE, ANNEE, NOTE)
SELECT  C.NC, I.NE, 2013 AS ANNEE, 20 AS NOTE
FROM COURS C, INSCRIT I
WHERE C.CODE_COURS = 'INF311'
AND C.NC = I.NC
```

Commande UPDATE

UPDATE <relation name>
SET <attribute = {value expression | NULL}
[<attribute> = {value expression | NULL}] …
[WHERE <search condition>]

- EXEMPLE

```
UPDATE RESULTAT
SET NOTE = NOTE * 1.2
WHERE RESULTAT.NC IN
(   SELECT NC
    FROM COURS C
    WHERE  C.CODE_COURS = ‘INF311’ )
```

Commande DELETE

DELETE FROM <relation name>
[WHERE <search condition>]

- EXEMPLE

```
DELETE FROM RESULTAT
WHERE NC IN
    SELECT C.NC
    FROM COURS C
    WHERE C.CODE_COURS = 'INF311'
```

4. Contraintes d'intégrité

- Contraintes de domaine
  - Valeurs possibles pour une colonne
- Contraintes de clés primaires
  - Clé et unicité
- Contraintes référentielles(clé étrangères)
  - Définition des liens inter-tables

SQL1 - 89 : INTEGRITE

- VALEURS PAR DEFAUT

```
CREATE TABLE ETUDIANT
( NE INT(5) PRIMARY KEY,
  NOM VARCHAR(128),
  PRENOM VARCHAR(128),
  VILLE VARCHAR(128),
  AGE INT(3) CHECK BETWEEN 10 AND 120)
```

- CONTRAINTES DE DOMAINES

SQL1 - 89 : Contrainte référentielle

- Clé primaire et contrainte référentielle

```
CREATE INSCRIT
( NC INT(5),
  NE INT (5),
  ANNEE INT(4),
PRIMARY KEY (NC, NE, ANNEE),
FOREIGN KEY (NC) REFERENCES COURS(NC),
FOREIGN KEY (NE) REFERENCES ETUDIANT(NE) )
```

- Référence en principe la clé primaire
  - celle de COURS et celle de ETUDIANT

SQL1 – 89 : Création de table

CREATE TABLE <nom_table>
(<def_colonne> *
[<def_contrainte_table>*]) ;

< def_colonne > ::=
<nom_colonne> < type | nom_domaine >
[CONSTRAINT nom_contrainte
< NOT NULL | UNIQUE | PRIMARY KEY |
CHECK (condition) | REFERENCES nom_table (liste_colonnes) >]
< def_contrainte_table > ::= CONSTRAINT nom_contrainte
< UNIQUE (liste_colonnes) | PRIMARY KEY (liste_colonnes) |
CHECK (condition) |
FOREIGN KEY (liste_colonnes) REFERENCES nom_table (liste_colonnes) >

5. CONCLUSION

- SQL1 est un standard minimum
- Les versions étendues:
  - SQL2 = Complétude relationnelle
  - SQL3 = Support de l'objet
  - SQL:2006 = Extension à XQuery
  - Par la suite … pas encore de grande révolution : rajout de fonctions mineures
- Sont aujourd'hui intégrées dans les grands SGBD

Les grandes bases de données (sept 2012)

- « Commerciaux »
  - Oracle (11*g*) version 11.2
  - IBM DB2 version 10
  - Microsoft SQL Server 2012
  - Sybase ASE 15.7 (SAP)
- « Open Source »
  - MySQL v. 5.5 (http://www.mysql.org/)
  - Postgres v. 9.2 (http://www.postgresql.org/)

LA NORMALISATION DE SQL

- Groupe de travail ANSI/X3/H2 et ISO/IEC JTC1/SC2
- Documents ISO :
  - SQL1 - 86 : Database Language SQL X3.135 ISO-9075-1987)
  - SQL1 - 89 : Database Language SQL with Integrity Enhancement X3.168 ISO-9075-1989
  - SQL:1999 (SQL3)
  - SQL2 - 92 : Database Language SQL2 X3.135 ISO-9075-1992
  - SQL:2003 ISO/IEC 9075:2003
  - SQL:2006 ISO/IEC 9075-14:2006
  - SQL:2008 ISO/IEC 9075:2008

Etc..