Initiation aux bases de données relationnelles

ATTENTION : Support de cours
Version (très) dégradée des diapositives projetées en cours (exemples supprimés)

Bruno PINAUD

bruno.pinaud@u-bordeaux.fr

Bibliographie

Georges Gardarin, Base de données, Ed. Eyrolles, 2003
A First Course in Database Systems (3rd edition) J. D. Ullman, J. Wildom, Ed. Pearson, 2008

PREMIÈRE PARTIE

INTRODUCTION ET DÉFINITIONS

Définition : base de données

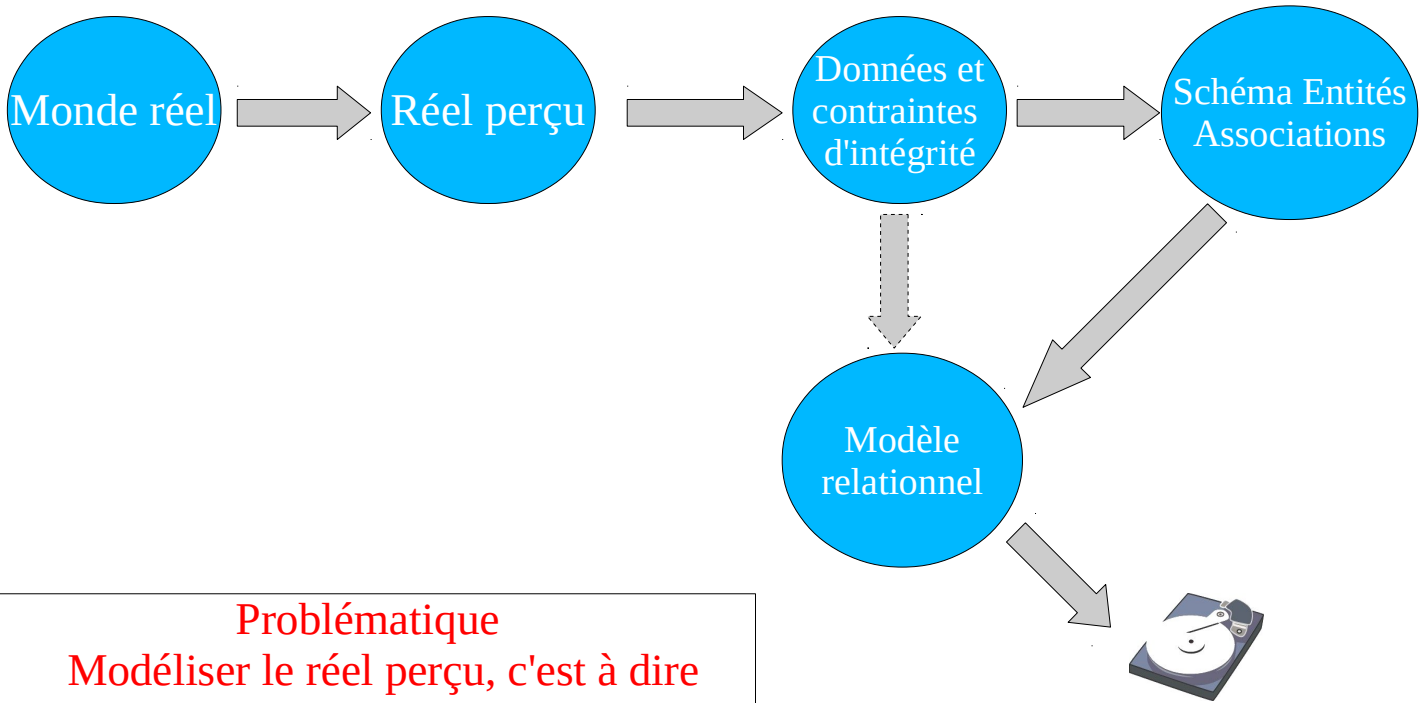
- Plus qu'un simple ensemble de données non-indépendantes

« […] Ensemble de données modélisant les objets d'une partie du monde réel et servant de support à une application informatique » G. Gardarin

Propriété importante
interrogeable par le contenu selon n'importe quel critère

Exemple : quels sont les numéros de Radioactive Man à moins de 10$?

Introduction
Processus de construction d'une base de données



Problématique
Modéliser le réel perçu, c'est à dire traduire les contraintes d'intégrité

Définition : SGBD Relationnel

- **SGBD** : Système de Gestion de Bases de Données (parfois SGBDR)
- **DBMS** : DataBase Management System (parfois RDBMS)

- Outil informatique pour gérer des BD.

- **Doit absolument posséder plusieurs fonctionnalités**
  - Sauvegarde (persistance) des données
  - Interrogation (SQL) des données
  - Recherche et mise en forme des données stockées
  - Partage des données entre les différents utilisateurs
  - Gestion de la concurrence d'accès
  - Sécurité des données (gestion des incidents)
  - Optimisation des opérations dans un souci constant de performance

BD : Vocabulaires et définitions

Concepts de base : domaine

Définition

- Ensemble de valeurs atomiques (non décomposable)
- Équivalent au typage en programmation

Concepts de base : produit cartésien

Définition

Le produit cartésien d'un ensemble de domaine D_1, D_2, ..., D_n noté $D_1 \times D_2 \times ... \times D_n$ est l'ensemble des n-uplets (tuples) $<v_1, v_2, ..., v_n>$ tels que $v_i \in D_i$.

Concepts de base : relation

Définition

- Sous ensemble *r* du produit cartésien d'un ensemble de domaines
- Caractérisée par un nom

Concepts de base : attribut

Définition

- Nom donné à une colonne
- Composé d'un identifiant et d'un domaine

- Nombre d'attributs d'une relation = degré de la relation (ou arité)

Concepts de base : schéma de relation

Définition

- **Un schéma de relation R** est défini par un ensemble d'attributs **U** et un ensemble de contraintes.
- On le note couramment **R(U)**.

- Le schéma décrit l'**intention** de la relation.
- La relation (tableau) définit une **extension**.

- Une relation ***r*** est une instance finie d'un schéma de relation, notée ***r*:R(U)**.

Concepts de base : base de données

Définition

- Un **schéma de base de données** est un ensemble de schémas de relations **liés** par des **dépendances référentielles** : attributs communs ou plus généralement des dépendances d'inclusion.

- Une **base de données** est alors un ensemble de relations (extensions) associé au schéma de base de données et vérifiant ses **contraintes d'intégrité.**

Concepts de base : base de données

Définition

- Un **schéma de base de données** est un ensemble de schémas de relations **liés** par des **dépendances référentielles** : attributs communs ou plus généralement des dépendances d'inclusion.

- Une **base de données** est alors un ensemble de relations (extensions) associé au schéma de base de données et vérifiant ses **contraintes d'intégrité.**

Les opérations sur les données (LDD/LMD SQL)

• Création d'une relation (*CREATE TABLE* …)
• Suppression d'une relation (*DROP TABLE* …)
• Modification d'une relation (*ALTER TABLE* …)

• Interrogation d'une relation (*SELECT* …)
• Insertion de données dans une relation (*INSERT INTO* …)
• Mise à jour des données d'une relation (*UPDATE* …)
• Suppression de données dans une relation (DELETE FROM…*)*

DEUXIEME PARTIE

DÉFINITION D'UN MODÈLE DE DONNÉES

LE MODÈLE RELATIONNEL

Un mauvais exemple

Objectif : Modélisation d'une compagnie aérienne.
On veut savoir quel est l'avion utilisé pour chaque vol et sa capacité.

Point de départ : création d'une **relation universelle** avec tous les attributs

Transports

Vol	Avion	Capacité
IT5033	Airbus A330	335
AF2401	Airbus A330	335
AF2409	Boeing 727	150
IT5133	Airbus A330	335
IT5035	Canadair	90
AF2802	Airbus A330	335

Un mauvais exemple

Transports

Vol	Avion	Capacité
IT5033	Airbus A330	335
AF2401	Airbus A330	335
AF2409	Boeing 727	150
IT5133	Airbus A330	335
IT5035	Canadair	90
AF2802	Airbus A330	335

Anomalies de la relation **Transports**

Le modèle relationnel

• Défini par E. F. Codd (IBM Research) dans « A Relational Model of Data for Large Shared Data Banks », CACM 13, No. 6, June 1970)

• Indépendant de la représentation physique des données

• Assise mathématique forte (algèbre relationnelle, formes normales)

Objectifs généraux

- Éliminer les comportements anormaux des relations lors des mises à jours
- Éliminer les données redondantes
- Meilleure compréhension des relations sémantiques entre les données

Conception d'un schéma relationnel

But : Éliminer les anomalies de la relation universelle pour faciliter la manipulation des relations ⇒ **Normaliser** les relations.

Méthode : **Décomposer** la relation universelle en sous-relations qui ne souffrent pas des anomalies précédentes.

Problématique : obtenir un nouveau schéma de base de données qui :
- conserve toutes les données,
- conserve un minimum de contraintes d'intégrité mais respecte le réel perçu,
- élimine toutes les anomalies.

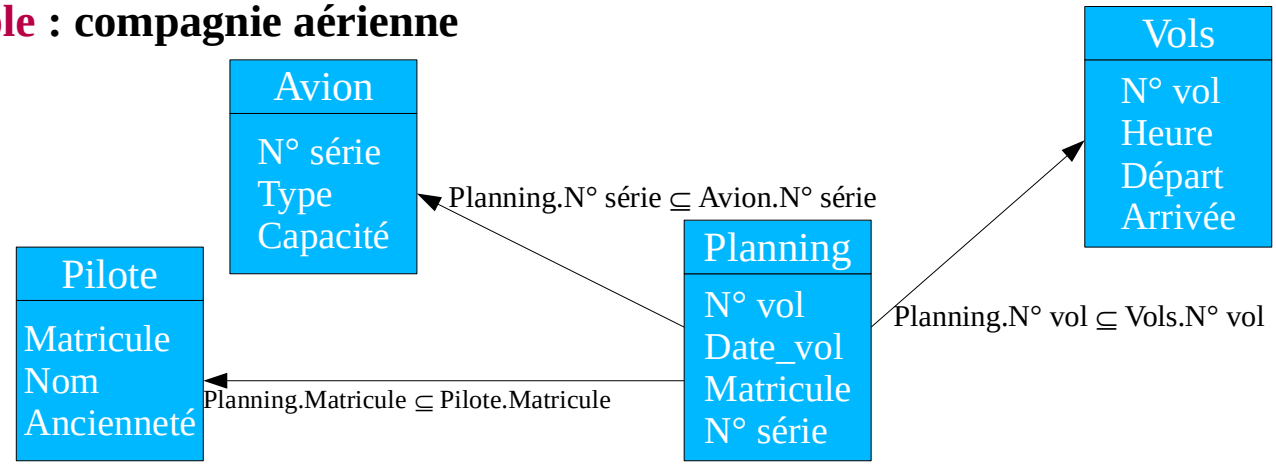
Concepts de base : base de données

Définition (rappel)

- Un **schéma de base de données** est un ensemble de schémas de relations **liés** par des **dépendances référentielles (un type de contraintes)** : attributs communs ou plus généralement des dépendances d'inclusion.

- Une **base de données** est alors un ensemble de relations (extensions) associé au schéma de base de données et vérifiant toutes ses **contraintes.**

Exemple : compagnie aérienne



Concepts de base : base de données

Avion

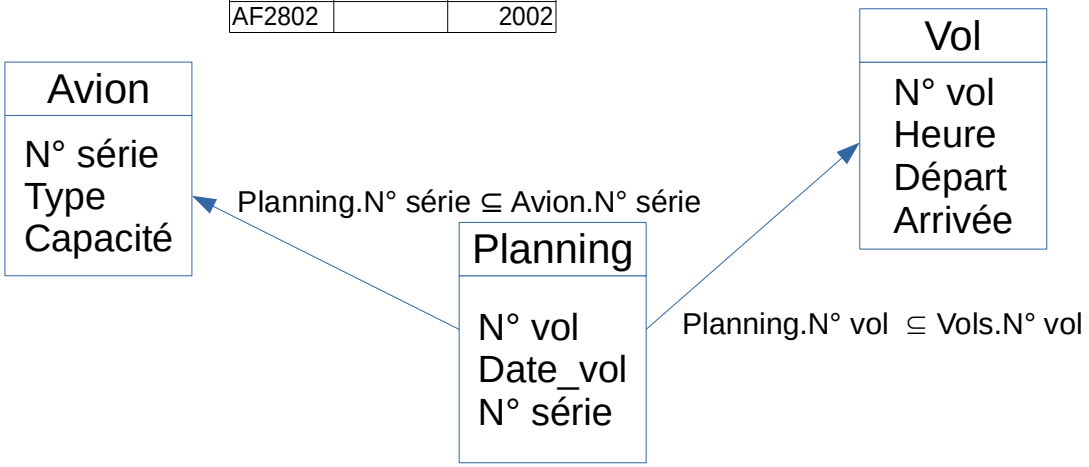
No Série	Type	Capacité
2001	Airbus A330	335
2002	Airbus A330	335
301	Boeing 727	150
94	Canadair	90

Planning

No Vol	Date_vol	No série
IT5033	…	2001
AF2401		2002
AF2409		2002
IT5133		301
IT5035		301
AF2802		2002

Vol

No Vol	Heure Départ	Heure Arrivée
IT5033	…	
AF2401		
AF2409		
IT5133		
IT5035		
AF2802		



LES CONTRAINTES D'INTÉGRITÉ

Contraintes d'intégrité

Définition

- Tout schéma de base de données doit être conçu pour imposer le respect d'un maximum de contraintes d'intégrité du réel perçu.

- Les **contraintes d'intégrité** sont des expressions logiques qui **doivent être satisfaites à tout instant** par une instance de base de données.

Contraintes d'intégrité

Définition

- Tout schéma de base de données doit être conçu pour imposer le respect d'un maximum de contraintes d'intégrité du réel perçu.

- Les **contraintes d'intégrité** sont des expressions logiques qui **doivent être satisfaites à tout instant** par une instance de base de données.

- Plusieurs types
  - Contraintes sur les attributs : domaines, valeurs nulles, ... ;
  - Contraintes sur les *n*-uplets : la valeur d'un attribut peut dépendre d'un ou plusieurs autres attributs du même *n*-uplet (dates d'emprunt et de retour pour une bibliothèque), ... ;
  - Contraintes sur les relations : clés, cardinalité, ... ;
  - Contraintes sur la base de données : clés étrangères, ... ;
  - Contraintes temporelles : évolution chronologique (diplômes, état-civil), ...

Contraintes d'intégrité

Définition

- Tout schéma de base de données doit être conçu pour imposer le respect d'un maximum de contraintes d'intégrité du réel perçu.

- Les **contraintes d'intégrité** sont des expressions logiques qui **doivent être satisfaites à tout instant** par une instance de base de données.

- Plusieurs types
  - Contraintes sur les attributs : domaines, valeurs nulles, ... ;
  - Contraintes sur les *n*-uplets : la valeur d'un attribut peut dépendre d'un ou plusieurs autres attributs du même *n*-uplet (dates d'emprunt et de retour pour une bibliothèque), ... ;
  - Contraintes sur les relations : clés, cardinalité, ... ;
  - Contraintes sur la base de données : clés étrangères, ... ;
  - Contraintes temporelles : évolution chronologique (diplômes, état-civil), ...

Problématique : minimiser le nombre de contraintes tout en restant équivalent à l'ensemble des contraintes d'origine

Contraintes d'intégrité : notion de clé

• Une relation est un ensemble de *n*-uplets. Par définition, un ensemble n'a pas d'élément en double, donc chaque *n*-uplet d'une relation est unique.
• Pour identifier les *n*-uplets de façon unique sans en donner toutes les valeurs et respecter leur unicité une clé est nécessaire.

Définition

- Groupe d'attributs minimum qui détermine un *n*-uplet de façon unique.

- Plus formellement : $X \text{ clé de } R(U) \text{ avec } X \subseteq U \text{ ssi } \forall r : R(U), \forall t_1, t_2 \in r,$

$$t_1[X] = t_2[X] \Rightarrow t_1 = t_2$$

$$et$$

$$\nexists Y \subset X \Rightarrow t_1[Y] = t_2[Y] \Rightarrow t_1 = t_2$$

Contraintes d'intégrité : notion de clé

Propriétés

- Toute relation possède au moins 1 clé : l'ensemble de ses attributs
- Si une relation possède plusieurs **clés candidates**, on choisit une clé qui sera privilégiée : **la clé primaire**. Aucun des attributs d'une clé primaire n'admet de valeur nulle (vide).

Conventions d'écriture

- On souligne la clé primaire.
- Les clés candidates sont soulignées en pointillés (souvent omis).

Contraintes d'intégrité : notion de clé

Exemple

Soit la relation R(A,B,C,D,E) :

A	B	C	D	E
A1	B1	C1	D1	E1
A1	B1	C2	D1	E1
A1	B1	C5	D1	E1
A1	B2	C5	D4	E1
A2	B1	C2	D1	E1
A2	B1	C5	D1	E1
A2	B1	C5	D3	E1
A2	B2	C5	D4	E1
A3	B3	C5	D1	E2

Est-ce que (A, C, D) est une clé candidate de R ?

CONTRAINTES D'INTÉGRITÉ :

LES DÉPENDANCES FONCTIONNELLES

Dépendances fonctionnelles : définition

• Principales contraintes d'intégrité formelles dans une BD.

• Une dépendance fonctionnelle (ou DF) indique une implication vérifiée universellement entre deux groupes d'attributs A et B : à une valeur pour A correspond toujours la même valeur de B.

• On la note A → B (A détermine B).

• A est parfois appelé « la source » et B « le but ».

Dépendances fonctionnelles : définition

• Soit R(A, B, C). L'attribut B est dit fonctionnellement dépendant de l'attribut A si étant donné 2 n-uplets $<a_1, b_1, c_1>$ et $<a_2, b_2, c_2>$

$$a_1 = a_2 \Rightarrow b_1 = b_2$$

• Ou encore, A détermine B si étant donné une valeur de A, il lui correspond une valeur unique de B.

Dépendances fonctionnelles

Remarques importantes

• Une DF est une assertion sur toutes les valeurs possibles (domaine des attributs) et non pas sur les valeurs actuelles (extension courante de la relation). Elle caractérise une intention et non pas une extension d'une relation : invariante au cours du temps.

• Les DF doivent être définies à partir d'un schéma de relation. Par défaut, elles sont définies sur la relation universelle.

Dépendances fonctionnelles

Utilités des DF

• Vérification que les extensions r d'un schéma R sont conformes au réel perçu.
• Spécifier (modéliser) les contraintes que devront vérifier toutes les relations d'un schéma relationnel.
• Outil de conception automatique d'un bon schéma relationnel.

DF : exemple

Soit une relation R exprimant l'emploi du temps d'un lycée construite sur les attributs suivants :
P (professeur), **H** (heure du cours), **S** (Salle), **C** (classe) et **M** (matière).
La signification d'un *n*-uplet de cette relation est :
Le professeur **P** enseigne la matière **M** à l'heure **H** dans la salle **S** à la classe **C**.
Donnez la liste des dépendances fonctionnelles.

DF : propriétés

- Système de règle d'inférences définit par Armstrong en 1974 :
  - Réflexivité : $Y \subseteq X \Rightarrow X \rightarrow Y$ (ex. $A \rightarrow A$; $AB \rightarrow A$)
  - Augmentation : $X \rightarrow Y \Rightarrow \forall Z, XZ \rightarrow YZ$ ou $XZ \rightarrow Y$
  - Transitivité : $X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$

DF : propriétés

- Autres règles déduites des 3 premières
  - Pseudo-transitivité : $X \rightarrow Y, WY \rightarrow Z \Rightarrow WX \rightarrow Z$

    Démonstration

    Augmentation à $X \rightarrow Y \Rightarrow WX \rightarrow WY$

    Transitivité : $WX \rightarrow WY, WY \rightarrow Z \Rightarrow WX \rightarrow Z$
  - Union (ou composition) : $X \rightarrow Y, X \rightarrow Z \Rightarrow X \rightarrow YZ$

    Démonstration

    Augmentation à $X \rightarrow Y \Rightarrow X \rightarrow YX$

    Augmentation à $X \rightarrow Z \Rightarrow XY \rightarrow ZY$

    Transitivité : $X \rightarrow YX, XY \rightarrow ZY \Rightarrow X \rightarrow ZY$

DF : propriétés

- Autres règles déduites des 3 premières
  - Décomposition : $X \rightarrow Y$ et $Z \subseteq Y \Rightarrow X \rightarrow Z$

    Démonstration

    Réfléxivité à $Z \subseteq Y \Rightarrow Y \rightarrow Z$

    Transitivité : $X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$

Formes des DF

- **DF triviale**

  Dépendance fonctionnelle obtenue par réflexivité.

- **DF simple/composée**

  Dépendance fonctionnelle qui possède un seul/plusieurs élément(s) en partie gauche.

- **DF directe**
  - Dépendance fonctionnelle non obtenue par transitivité.
  - $X \rightarrow A$ directe ssi $\neg\exists Z$ / $X \rightarrow Z$ et $Z \rightarrow A$.

- **DF élémentaire**
  - Dépendance fonctionnelle non décomposable (DF simple ou partie gauche sans attribut superflu).
  - $X \rightarrow A$ élémentaire ssi $A \notin X$ et $\neg\exists X' \subset X, X' \rightarrow A$.

Fermeture transitive d'un ensemble de DF

• Plus grand ensemble stationnaire F^+ de DF valides obtenu à partir d'un ensemble F initial en appliquant les propriétés des DF (axiomes d'Armstrong).

• La fermeture transitive d'un ensemble de DF est unique. Elle ne dépend pas de l'ordre d'utilisation des propriétés.

• Deux ensembles des DF sont équivalents ssi ils ont la même fermeture transitive $F_1 \equiv F_2 \Leftrightarrow F_1^+ = F_2^+$.

Fermeture transitive d'un ensemble de DF

• Plus grand ensemble stationnaire F^+ de DF valides obtenu à partir d'un ensemble F initial en appliquant les propriétés des DF (axiomes d'Armstrong).

• La fermeture transitive d'un ensemble de DF est unique. Elle ne dépend pas de l'ordre d'utilisation des propriétés.

• Deux ensembles des DF sont équivalents ssi ils ont la même fermeture transitive $F_1 \equiv F_2 \Leftrightarrow F_1^+ = F_2^+$.

Exemple

$N \rightarrow T$

$T \rightarrow M$

$T \rightarrow P$

$N \rightarrow C$

Fermeture transitive d'un ensemble d'attributs

• La fermeture transitive d'un ensemble d'attributs X par rapport à un ensemble de DF *F* est l'ensemble des attributs qui peuvent être inférés à partir de X en utilisant les DF contenues dans *F*.

Exemple

$N \rightarrow T$

$T \rightarrow M$

$T \rightarrow P$

$N \rightarrow C$

Fermeture transitive de l'attribut N ?

Couverture minimale

• Sous ensemble F_{min} minimum (sans redondance) de DF qui permet de générer toutes les DF de F^+ :

$$F^+(F_{min}) = F^+(F)$$
$$\neg\exists\ F' / F' \subset F_{min} \text{ et } F^+(F') = F^+(F_{min})$$

• Contrairement à F^+, la couverture minimale n'est pas unique.

• Ensemble essentiel pour effectuer des décompositions sans perte.

• Algorithme en 3 étapes (ordre à respecter) à partir d'un ensemble de DF :
1. Décomposer les DF (un seul attribut en partie droite) ;
2. Rendre les DF élémentaires (membre gauche non décomposable) ;
3. Supprimer les DF redondantes (obtenues par transitivité) ;

Couverture minimale

Soit l'ensemble *F* :

$A \rightarrow BC$

$ABC \rightarrow D$

$C \rightarrow D$

$D \rightarrow E$

$E \rightarrow F$

$DEF \rightarrow G$

Calculer une couverture minimale.

Couverture minimale

Étape 1 : Décomposer les DF (un seul élément en partie droite).

F

$A \rightarrow BC$

$AB \rightarrow C$

$ABC \rightarrow D$

$C \rightarrow D$

$D \rightarrow E$

$E \rightarrow F$

$DEF \rightarrow G$

Couverture minimale

Étape 2 : rendre les DF élémentaires (membre gauche non décomposable, c'est à dire nombre minimum d'attributs).

F_1 :

- $A \rightarrow C$
- $A \rightarrow B$
- $AB \rightarrow C$
- $ABC \rightarrow D$
- $C \rightarrow D$
- $D \rightarrow E$
- $E \rightarrow F$
- $DEF \rightarrow G$

Couverture minimale

Étape 2 : rendre les DF élémentaires (membre gauche non décomposable, c'est à dire nombre minimum d'attributs).

F_1 :

- $A \rightarrow C$
- $A \rightarrow B$
- $AB \rightarrow C$
- $ABC \rightarrow D$
- $C \rightarrow D$
- $D \rightarrow E$
- $E \rightarrow F$
- $DEF \rightarrow G$

$AB \rightarrow C$

B superflu ?

Couverture minimale

Étape 3 : Élimination des DF redondantes (obtenues par transitivité, c'est à dire non directes).

$$F_4 \left\{ \begin{array}{l} A \rightarrow C \\ A \rightarrow B \\ C \rightarrow D \\ D \rightarrow E \\ E \rightarrow F \\ D \rightarrow G \end{array} \right.$$

Retour sur la notion de clé

Définition

Un sous ensemble X des attributs d'un schéma relationnel R<U,F> avec U l'ensemble des attributs et F une couverture minimale de DF est une clé candidate ssi :

1. $\{X \rightarrow U\} \in F^+$
2. $\neg\exists\, X' \subset X \,/\, \{X' \rightarrow U\} \in F^+$ (minimalité d'une clé)

   et
3. $X^+_F = U$

Une clé candidate est un ensemble minimum d'attributs qui détermine tous les autres.

Retour sur la notion de clé

$F_4 = \{ A \rightarrow C,\ A \rightarrow B,\ C \rightarrow D,\ D \rightarrow E,\ E \rightarrow F,\ D \rightarrow G \}$

Quelle est la clé de la relation R(A, B, C, D, E, F, G) dont F_4 est la couverture minimale ?

LES FORMES NORMALES

Les formes normales

Objectifs

Différentes formes de normalisation de la relation universelle, le plus souvent obtenues par décomposition, afin d'obtenir un schéma de base de données qui soit (pour la 3FN) :

- sans redondance,
- sans anomalies de mise à jour,
- sans perte de données,
- sans perte de DF.

Dans ce cours : 1re, 2e, 3e FN (NF). Il existe d'autres formes normales de plus haut-niveau (intérêt plus théorique que pratique).

La première forme normale (1FN)

Définition

Une relation est en 1FN si chacun des attributs ne prend ses valeurs que dans un domaine constitué de valeurs élémentaires (*i.e.* atomiques).

La 1FN consiste à éviter les domaines composés de plusieurs valeurs (pas de données composées, pas de listes).

La première forme normale (1FN)

Définition

Une relation est en 1FN si chacun des attributs ne prend ses valeurs que dans un domaine constitué de valeurs élémentaires (*i.e.* atomiques).

La 1FN consiste à éviter les domaines composés de plusieurs valeurs (pas de données composées, pas de listes).

La deuxième forme normale (2FN)

Intérêt historique car supprime peu d'anomalies.

Définition 1

Un schéma de relation est en 2FN ssi :

- il est en 1FN,
- il n'admet pas de **dépendance de clé partielle,** c'est à dire une DF d'une partie stricte d'une clé vers des attributs non clés.

Définition 2

- 1FN
- Toutes les DF des clés vers les attributs non clés sont élémentaires.

La troisième forme normale (3FN)

Définition

Un schéma de relation est en 3FN ssi :

- il est en 2FN,
- chaque attribut non clé est pleinement et directement dépendant des clés.

ou

- tout attribut n'appartenant pas à une clé ne dépend pas d'un attribut non clé.

ou encore

- il n'admet pas de DF transitive, c'est-à-dire d'un ensemble d'attributs non inclus dans une clé vers un autre ensemble d'attributs non (sur-)clé. Toutes les DF des clés vers les attributs non clés sont élémentaires et directes.

La troisième forme normale (3FN)

Exemple

Avec le nouveau système d'immatriculation des véhicules, on a maintenant :
VOITURE(<u>immatriculation</u>, marque, type, puissance, couleur)
DF supplémentaires : type → marque et type → puissance

Est-ce que la relation VOITURE est en 3FN ?

Algorithmes de normalisation

Objectif

Obtenir un schéma de base de données qui minimise le nombre de schémas de relation et qui maximise leur forme normale et soit sans perte (pas de perte de données, pas de perte de DF). Cela revient à décomposer la relation universelle en 3FN.

Deux classes d'algorithmes :

• **par décomposition**

⇒ Décompositions successives des relations obtenues à partir de la relation universelle afin de maximiser la forme normale des relations obtenues. Risque important de perdre des DF.

• **algorithme de synthèse**

⇒ À partir de la couverture minimale des DF, utilisation des propriétés des DF pour obtenir « directement » une bonne décomposition.

Algorithme de synthèse

Définition

A partir de l'ensemble des DF fourni et en utilisant leurs propriétés, on peut directement obtenir une décomposition sans perte.

Principe

1. Calculer la couverture minimale ;
2. Condenser les DF ayant même parties gauche (union) ;
   Chaque DF obtenues est potentiellement une relation en 3FN
3. Détecter les équivalences entre DF (regrouper des relations) ;

Algorithme de synthèse

Exemple

Soit R={A, B, C, D, E} et F={AB → C ; C → B ; A → D}.
Décomposer R en 3FN.

Algorithme de synthèse

Exemple

Soit R={A, B, C, D, E} et F={AB → C ; C → B ; A → D}.
Décomposer R en 3FN.

1. Calcul de la couverture minimale

2. Condenser les DF ayant même parties gauches

Algorithme de synthèse

Exemple

3. Détecter les équivalences entre DF

Regroupement des relations qui se référencent mutuellement *via* des clés étrangères.

R_1(<u>A, B</u>, C) R_2(<u>C</u>, B) R_3(<u>A</u>, D)

R_1 et R_2 se référencent mutuellement. On les regroupe en R_{12}(<u>A, B</u>, C).

La relation R à deux clés candidates {A, B, E} et {A, C, E}. Pour ne pas perdre l'attribut E, il faut ajouter une relation composée de la clé de R, par exemple R_4(<u>A, B, E</u>).

La décomposition est donc : R_{12}(<u>A, B</u>, C) R_3(<u>A</u>, D) R_4(<u>A, B, E</u>).

Décomposition sans perte

Définition

La décomposition d'une relation R<U,F> en un ensemble de schémas de relations {R_1, ..., R_n} obtenus par projection est dite sans perte si et seulement si quelque soit la réalisation r de R :

- La jointure naturelle des r_i donne exactement r (pas de perte de données) ;
- $(F_1 \cup \ldots \cup F_n)^+ = F^+$ (pas de perte de dépendances).

Théorème de Heath

Soit R(X, Y, Z) et la DF X→Y qui est vérifiée. R peut alors être décomposée en R_1(X, Y) et R_2(X, Z). R est égale à la jointure de ses projections sur R_1 et R_2.

Lemme de Rissanen

Soit R(X, Y, Z) et les DF X→Y et Y→Z qui sont vérifiées. R peut alors être décomposée en R_1(X, Y) et R_2(Y, Z).

Modélisation relationnelle : exercices

Soit la relation R(A, B, C, D, E, F, G) et son ensemble de DF :

$$F=\left\{\begin{array}{l} A \rightarrow D \\ A, B, C \rightarrow E, F \\ D \rightarrow E \\ C \rightarrow B \\ E,F \rightarrow G \\ G \rightarrow D \end{array}\right.$$

- Quelle est la clé de R ?
- Quelle est la forme normale de R ?
- Proposer une décomposition sans perte en 3FN.

Modélisation relationnelle : exercices

Intérêt de la normalisation

Soit la relation R(<u>A</u>, B, C, D, E, F) et les DF supplémentaires B $\rightarrow$ CDEF. La clé primaire de R est A.

On sait aussi que card(π_A(R))=10^6 et card(π_B(R))=10^3. De plus, pour chaque attribut, il faut 100 octets pour stocker une valeur.

- Quelle est la forme normale de R ?
- Trouver un schéma équivalent formé de relations en 3FN qui préserve les données et les dépendances.
- Montrer l'intérêt de la normalisation en 3FN.

Modélisation relationnelle : exercices

Chiens, propriétaires et vétérinaires
On souhaite stocker les identités d'un ensemble de personnes : nom, prénom, numéro de sécurité sociale et adresse. Ces personnes ont des chiens à qui elles ont donné un nom. Un chien n'a qu'un seul propriétaire et plusieurs personnes peuvent donner le même nom à leur chien. On considère également une relation « a été vu par », entre chien et vétérinaires (qui sont des personnes qui peuvent être propriétaire de chiens mais elles ont en plus un numéro d'agrément), un chien peut avoir été vu par plusieurs vétérinaires et tous les chiens d'un même propriétaire n'ont pas forcément vu les mêmes vétérinaires (par exemple à cause d'une visite urgente le week-end).

- Donner la liste des DF élémentaires et directes issue de ce réel perçu.
- Proposer un schéma entité-association.
- Donner un schéma de relations en 3FN.

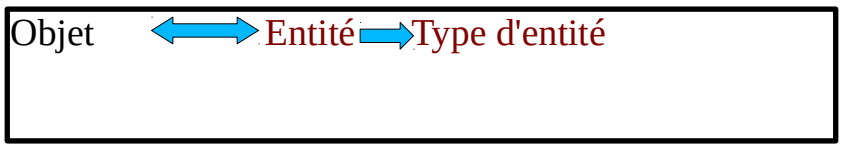
LE MODÈLE ENTITÉS-ASSOCIATIONS

Le modèle entités-associations : concepts de base

- Modèle conceptuel simple et graphique des données
- Sémantiquement riche
- Modèle de haut-niveau indépendant de l'implémentation.

Le modèle entités-associations : Les types d'entités

- Un type d'entité représente un objet du réel perçu.
- Possède forcément une clé.



Le modèle entités-associations : Les types d'associations

- Associe au moins deux types d'entités.

Objet	⟺ Entité	⇨ Type d'entité
Lien	⟺ Association	⇨ Type d'association

Le modèle entités-associations : Les attributs

Objet	⟺ Entité	⇨ Type d'entité
Lien	⟺ Association	⇨ Type d'association
Propriété	⟺ Attribut	

Le modèle entités-associations : concepts de base

• Un type d'association peut-être relié plusieurs fois à un même type d'entité. Nécessité de définir le rôle de chaque association.

Le modèle entités-associations : contraintes de cardinalités

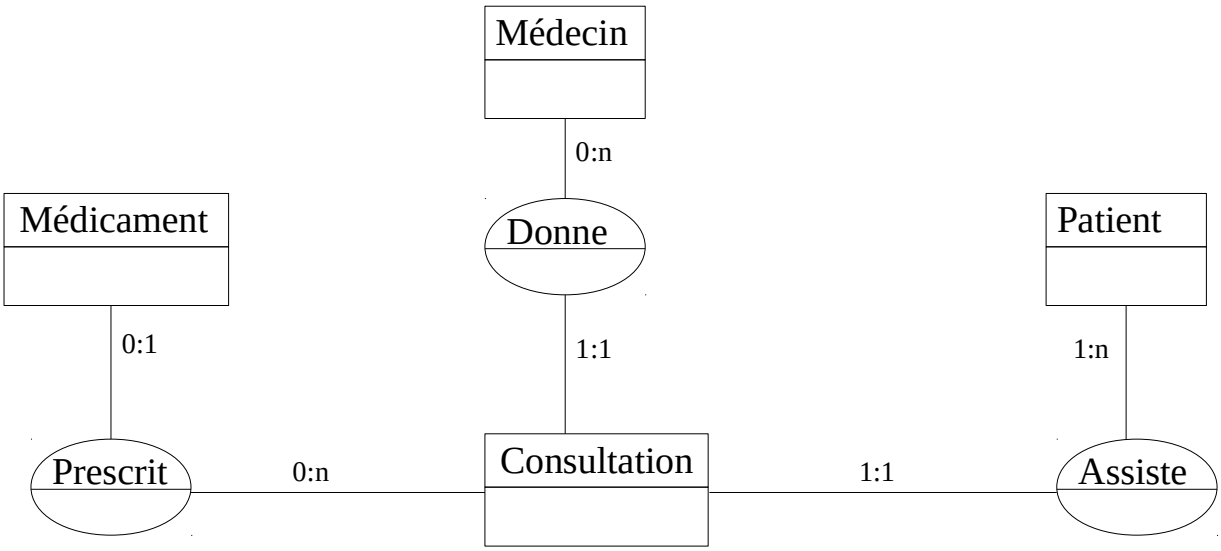
• Pour les types d'entités A et B, à combien d'associations R chaque entité peut et doit participer ?

Le modèle entité-association : spécialisation/généralisation

•La **spécialisation** permet de différencier plusieurs entités en fonction de leurs caractéristiques spécifiques.
• A l'inverse, la **généralisation** est la réunion en une entité unique de plusieurs entités ayant des attributs communs.

Relation de type « is-a »

Le modèle entités-associations : exemple



Les identifiants des types d'entités ont été omis pour simplifier.

Le modèle entités-associations : Gestion d'un supermarché

Des articles en quantité variable sont vendus dans les rayons d'un supermarché. Certains employés sont chargés de mettre ces articles dans des rayons. Ces employés sont affectés à un ou plusieurs rayons particuliers. Les articles sont livrées par des fournisseurs selon des commandes passées par le chef de rayon (lui même est un employé). Il est aussi responsable des employés chargés de remplir les rayons.

TROISIÈME PARTIE

PROGRAMMATION D'UN SYSTÈME DE GESTION DE BASE DE DONNÉES

ALGÈBRE RELATIONNELLE

Algèbre relationnelle

- Manipulations des relations par interrogation.
- Base théorique du langage SQL.
- Combinaisons de relations avec différents opérateurs pour obtenir de nouvelles relations.

Algèbre relationnelle

•Manipulations des relations par interrogation.
•Base théorique du langage SQL.
•Combinaisons de relations avec différents opérateurs pour obtenir de nouvelles relations.

•Deux opérations fondamentales pour la normalisation et la décomposition :
- **la projection**
- **la jointure**

L'opérateur de projection

Définition

L'opérateur de **projection**, noté π, permet de ne retenir que les *n*-uplets des attributs indiqués par l'opérateur en supprimant les éventuels doublons (partition verticale).

Exemple

Soit R(A,B,C,D).

$\pi_{C, D}(R)$ =

R

A	B	C	D
2	A	1	1
2	B	5	2
3	C	2	2
4	D	7	6
3	E	5	2
6	G	7	6

L'opérateur de sélection

Définition

L'opérateur de **sélection**, noté σ, permet de ne retenir que les *n*-uplets vérifiant une propriété particulière donnée sous la forme d'un prédicat (partition horizontale).

Exemples d'opérateurs : >, <, =, ≤, ≥, ≠, ⊂, ⊆, ⊄

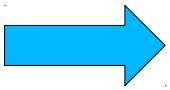
Exemple

Soit R(<u>A</u>,<u>B</u>,C,D).

$\sigma_{C>2}(R)$ =

R

A	B	C	D
2	A	1	1
2	B	5	2
3	C	2	2
4	D	7	6
3	E	5	2
6	G	7	6



Exemple

Avion
N° série Type Capacité

Quel est le numéro de série des avions dont la capacité est supérieure à 150 passagers ?

L'opérateur de jointure

Définition

- Opération notée ⋈.
- La jointure naturelle ou simplement jointure est le rapprochement de deux relations liées *via* des attributs communs dans leurs schémas correspondants.
- Les *n*-uplets du résultat sont obtenus par concaténation des attributs des deux relations lorsque les attributs communs ont des valeurs identiques (on simplifie en n'écrivant ces attributs qu'une seule fois).
- Équivalent à une sélection des valeurs de l'attribut commun sur le résultat d'un produit cartésien des relations concernées.

Quelques autres opérateurs de jointure

- Θ-jointure : utilise d'autres opérateurs que l'égalité
- Jointure externe : récupération des n-uplets qui ne joignent pas. Notation : ⟕
- Semi-jointure droite ou gauche : jointure suivie d'une projection sur les attributs de la relation à droite ou à gauche de l'opérateur de jointure.

L'opérateur de jointure

Exemple

Nom de la personne qui consomme des crêpes ?

Consommation

N°P	NomP	Id
1	Crêpes	45
2	Cidre	89
5	Galettes	3

Clients

Id	NomC	Adresse
3	Parker	New-York
45	Simpson	Springfield
89	Kent	Metropolis

L'opérateur de jointure : l'auto-jointure

•Auto-jointure : jointure d'une relation avec elle-même.

N°Emp	Nom	N°Sup
1	Simpson	2
2	Kent	7
5	Kenobi	
7	Luthor	42

- Par rapport à l'exemple, à utiliser pour répondre à des questions du type « Qui est le supérieur hiérarchique de mon supérieur hiérarchique » ?
- Nécessité de renommer la relation pour identifier de façon unique chaque relation qui compose la jointure : opérateur ρ.

- $\rho_{S(A, B, C)}$(Employés) ⇒ Renomme la relation Employés(N°Emp, Nom, N°Sup) en S(A, B, C) le temps de la requête

L'opérateur de jointure : l'auto-jointure

- Jointure d'une relation avec elle-même.

N°Emp	Nom	N°Sup
1	Simpson	2
2	Kent	7
5	Kenobi	
7	Luthor	42

- Quel est le supérieur du supérieur de Simpson ?

Opérations binaires

Soit deux relations **A(A_1, A_2, ..., A_n)** et **B(B_1, B_2, ..., B_n)**. **A** et **B** sont **compatibles** ssi elles ont le **même degré** et si **dom(A_i)=dom(B_i)** pour $1 \leq i \leq n$.

Définition

Ce sont les opérations mathématiques standards de la théorie des ensembles. Elles ne peuvent être appliquées que sur des relations compatibles.

Union : $A \cup B$. Relation qui inclut tous les *n*-uplets appartenant à A, à B ou au deux. Les doublons sont éliminés.

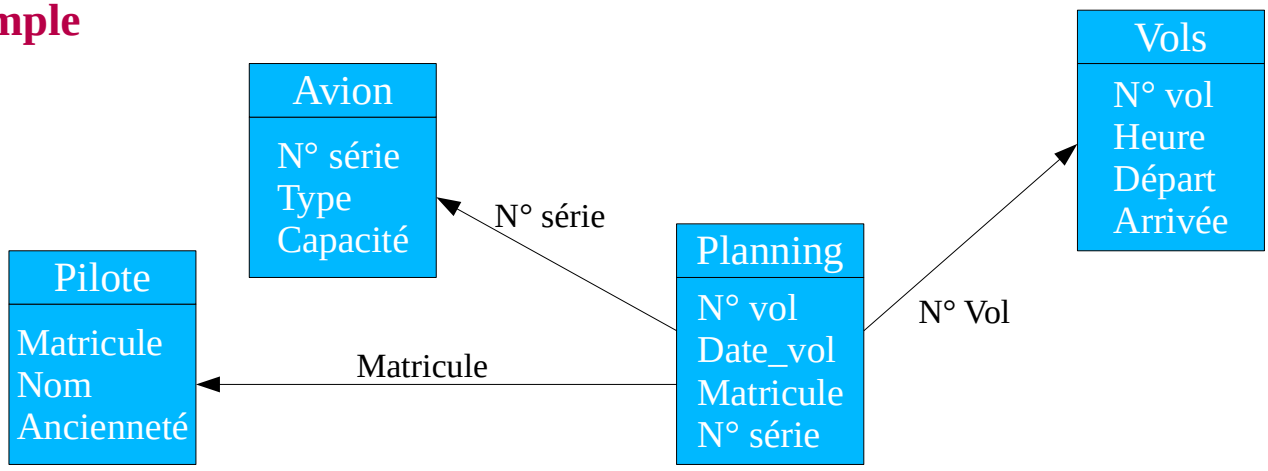
Intersection : $A \cap B$. Relation qui inclut tous les *n*-uplets appartenant à la fois à A et B.

Différence : A-B. Relation qui inclut tous les *n*-uplets appartenant à A mais pas à B.

La différence

Différence : A-B. Relation qui inclut tous les *n*-uplets appartenant à A mais pas à B.

Exemple



Quelles sont les lignes qui ne sont jamais parcourues par des "Airbus" ?

La division

Notation : R÷S

Définition

Permet de conserver un sous-ensemble des n-uplets partiels de R qui sont tous présents dans S. Réponse à des requêtes de la forme « quel que soit x, trouver y », « quel est le x qui à tous les y ? »

Principe

R(X,Y)	S(Y)
	Q(X)

Schéma de Q : tous les attributs de R n'appartenant pas à S

n-uplets de Q : *n*-uplets q_j de Q / ∀ *n*-uplets S_i de S, le *n*-uplet (q_j, s_j) est un *n*-uplet de R ⇒ Q×S ⊆ R

La division

Schéma de Q : tous les attributs de R n'appartenant pas à S

n-uplets de Q : *n*-uplets q_j de Q / ∀ *n*-uplets S_i de S, le *n*-uplet (q_j, s_j) est un *n*-uplet de R ⇒ Q×S ⊆ R

Exemple

R

A	B
a1	b1
a2	b1
a1	b2

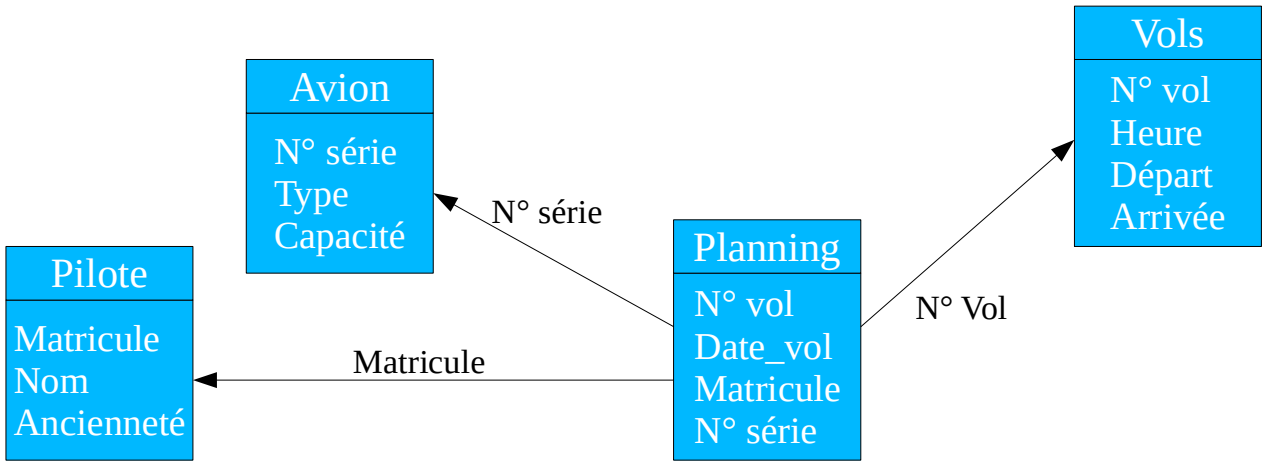
÷

S

B
b1
b2

=

La division



Quels sont les commandants qui volent sur tous les types d'avions ?

LE LANGAGE SQL

Bibliographie
Georges Gardarin, Base de données, Ed. Eyrolles, 2003

Le Langage SQL

Définition

SQL = **S**tructured **Q**uery **L**anguage
Relations ⟹ Tables

Regroupement de plusieurs langages :

- Langage de définition de données (LDD)
  - ➔ Création des tables, modifications, suppressions
- Langage de manipulation de données (LMD)
  - ➔ Requêtes (interrogations), insertions, suppressions et mises à jours des données contenues dans les tables
- Langage de contrôle de données (LCD)
  - ➔ Permissions, contrôle d'intégrité (triggers), *etc*.

Le Langage SQL

Pour la suite, utilisation d'une base de données d'acheteurs de vins (Cf livre de Gardarin) dont le schéma relationnel est :

Acheteurs(NoA, NomA, VilA)
Vins(NoV, NomV, MilV)
Achats(NoA, NoV, Qu)

NoA et NoV sont des clés étrangères respectivement sur les tables Acheteurs et Vins.

Tous les exemples fonctionnent sous Oracle. Avec d'autres SGBD (PostgreSQL, MySQL), certains opérateurs peuvent ne pas exister ou s'exprimer d'une autre façon.

Langage de Manipulation de Données

Interrogation des tables : la projection

select <liste d'attributs> from <liste de tables>;

Exemples

Quelle est la liste des numéros, noms et villes des acheteurs ?

select NoA, NomA, VilA from acheteurs;

Donner tous les renseignements sur les vins.

select * from vins;

** est équivalent à la liste de tous les attributs de la table*

Langage de Manipulation de Données

Liste des villes des acheteurs sans doublon

select distinct vilA from Acheteurs

Contrairement à l'algèbre relationnelle, SQL ne supprime pas automatiquement les doublons dans les n-uplets résultats d'une requête. Le mot clé distinct *sert à cela.*

Langage de Manipulation de Données

Interrogation des tables : la sélection

select <liste d'attributs> from <liste de tables> where <prédicat>;

Prédicat se compose des comparateurs =, <, >, >=, <= et <> ou != et des connecteurs *not*, *and* et *or*.

Pour des intervalles, il est aussi possible d'utiliser :

select <liste d'attributs> from <liste de tables> where <attribut> [not] between <val1> and <val2>;

Pour des recherches sur des chaînes de caractères, on peut utiliser :

- le caractère '%' qui représente une chaîne de longueur quelconque,
- le caractère '_' qui remplace un caractère quelconque.

Langage de Manipulation de Données

Quelle est le nom de l'acheteur dont le numéro est 40 ?

select NomA from acheteurs where NoA=40;

Quels sont les acheteurs qui habitent Nantes ou Paris ?

select * from acheteurs where VilA='Nantes' or VilA='Paris';

Obtenir la liste des vins dont le numéro est inférieur ou égal à 5 et dont le millésime est différent de 2000.

select * from vins where NoV<=5 and MilV!=2000;

Obtenir le nom des vins contenant un i en avant-dernière position.

select * from vins where NomV like '%i_';

Interrogation des tables : opérations binaires

Rappel : les relations doivent être compatibles.

Union : union
Intersection : intersect
Soustraction : minus (except sous PostgreSQL, non supporté sous MySQL)

Exemples

Noms des acheteurs qui ont acheté le vin n°8 ou le n°1 ou les deux.

```
select NoA from Achats where NoV=8
union
select NoA from Achats where NoV=1;
```


Noms des acheteurs qui ont acheté le vin n°8 et le n°1

```
select NoA from Achats where NoV=8
intersect
select NoA from Achats where NoV=1;
```

Noms des acheteurs qui ont acheté le vin n°8 et pas le n°1

```
select NoA from Achats where NoV=8
minus
select NoA from Achats where NoV=1;
```

Interrogation des tables : jointure

Préalable

Soit deux relations R(A,B,C) et S(B,D,E).
Pour accéder aux attributs communs, il faut utiliser la notation pointée : R.B et S.B

Jointure naturelle (ancienne syntaxe)

Le produit cartésien R×S s'écrit en SQL : select * from R,S
select <liste attributs> from table1, table2 where table1.attribut = table2.attribut;

Jointure naturelle (syntaxe moderne)

select <liste attributs> from table1 join table2 on table1.attribut = table2.attribut;

Exemples

Noms des acheteurs qui ont acheté le vin n°1.

select NomA from Acheteurs, Achats where Acheteurs.NoA = Achats.NoA and NoV=1;
select NomA from Acheteurs join Achats on Acheteurs.NoA = Achats.NoA and NoV=1;

Langage de Manipulation de Données

Interrogation des tables : jointure

Notion d'alias

Pour simplifier l'écriture des clauses where, il est possible d'utiliser un alias sur les noms des tables. Une fois l'alias défini (en gras), la table ne peut plus être accédée qu'en utilisant l'alias.

Obtenir les noms des vins achetés par Marie ainsi que leur quantité.

SELECT NomV, Qu FROM Acheteurs **A1**, Achats **A2**, Vins **V** WHERE NomA='Marie' AND **A1**.NoA=**A2**.NoA AND **A2**.NoV=**V**.NoV;

SELECT NomV, Qu FROM Acheteurs **A1** JOIN Achats **A2** JOIN Vins **V** ON **A1**.NoA=**A2**.NoA AND **A2**.NoV=**V**.NoV WHERE NomA='Marie';

Interrogation des tables : opérateurs d'appartenance

select ... from … where <attribut> [not] in (select … from... [where ...]);
in/not in : permet de tester si un élément donné appartient ou pas à une relation.

Interrogation des tables : Quantificateur

select ... from … where <attribut> <quantificateur> (select … from... [where ...]);
Quantificateur : comparer la valeur d'un attribut à un ensemble de valeurs d'autres attributs.
Liste : >all, <all, >=all, <=all, =all, <>all, >any, <any, >=any, <=any, =any, <>any

A noter : <>all ⇔ not in et =any ⇔ in

Interrogation des tables : Existence

*select ... from … where [not] exists (select * from … [where ...]);*
La requête peut s'exécuter si la requête imbriquée dans le *exists* retourne au moins un n-uplet.

Exemples

Quels sont les acheteurs qui n'ont pas acheté le vin n°3 ?

select NoA, NomA from acheteurs where NoA not in (select NoA from Achats where NoV=3);

Liste des vins achetés en plus grande quantité que le vin n°8.

select distinct NoV from Achats where Qu>all(select distinct Qu from Achats where NoV=8);

Trouver les acheteurs qui n'ont pas acheté de vin.

select AE.NoA from Acheteurs AE --sélection des acheteurs répertoriés, un par un
where not exists
(select * from Achats AC where AE.NoA=AC.NoA); /*sélection des achats répertoriés pour un acheteur donné*/

Interrogation des tables : opérations mathématiques et ensemblistes

De nombreux opérateurs et fonctions arithmétiques et ensemblistes sont définis. Ils peuvent être utilisés dans la clause select ou dans la clause where.

Opérateurs : +, -, * et /

Principales fonctions arithmétiques : abs, mod, ...
Principales fonctions ensemblistes : max, min, count, sum, avg, stddev, variance, ...

Exemples

Nombre d'acheteurs répertoriés

select count(*) from Acheteurs;

Quantité moyenne de vins achetée par l'ensemble des acheteurs

select avg(Qu) from Achats;

Interrogation des tables : l'opérateur de groupe

select <liste1 d'attributs>, [<liste2 d'attributs>] from … [where …] group by <liste d'attributs incluse dans liste1 d'attributs> [having ...];

La clause *having* … pour un *group by* est équivalente à une clause *where*.

Partitionner les occurrences d'un ensemble sur un critère donné de façon à appliquer une fonction d'agrégation.

Exemples

Quel est le nombre d'achats effectué par acheteur ?

select NoA, count(*) from Achats group by NoA;

Quels sont les acheteurs qui ont effectué plus de trois achats ?

select NoA from achats group by NoA having count(*)>3;

Présentation des résultats : classement des résultats

select <attribut1>, <attribut2>, … from … [where …] order by <attribut1|1> [ASC|DESC], <attribut2|2> [ASC|DESC], …

Permet de classer les résultats d'une requête. Par défaut l'ordre est ascendant (ASC).

Exemples

Classer les acheteurs par ordre décroissant des villes, puis par ordre croissant des noms.

select NoA, NomA, VilA from acheteurs order by VilA desc, nomA asc;

Ou

select NoA, NomA, VilA from acheteurs order by 3 desc, 2 asc;

Interrogation des tables : la division - rappel

R(A,B)

A	B
a1	b1
a2	b1
a2	b2

S(B,C)

B	C
b1	c1
b2	c1

Requête

Quels sont les valeurs de A liées à toutes les valeurs de B ou à au moins chaque valeur de B ?

R ÷ S =

A
a2

Opérateur inexistant en SQL contrairement à l'algèbre relationnelle.
Quelques exemples de solutions : avec un group by, avec un not exists et un minus (solution lente) ou encore 2 not exists imbriqués (ça devient compliqué)

Langage de Manipulation de Données

Interrogation des tables : la division – en utilisant un group by

SELECT A FROM R JOIN S ON R.B = S.B group by A having count(R.B)=(select count(distinct B) from S);

- Groupement des n-uplets de R par valeur de A
- Calcul du nombre de valeurs distinctes de B dans chaque groupe
- Egalité avec le nombre de valeurs distinctes de B dans S

Exemples

Obtenir la liste des acheteurs qui ont acheté tous les vins de 2000.

SELECT NoA FROM Achats JOIN Vins ON Achats.NoV = Vins.NoV
WHERE Milv = 2000 GROUP BY NoA
HAVING COUNT(achats.NoV) = (SELECT COUNT(*) FROM Vins WHERE milv = 2000);

Langage de Manipulation de Données

Interrogation des tables : la division – en utilisant un not exists et un minus

select distinct A from R R1 where not exists (
(select B from S) minus (select B from R R2 where R1.A=R2.A));

On regarde si l'ensemble suivant est vide :

- L'ensemble de toutes les valeurs possibles de B dans S
- Moins
- L'ensemble des toutes les valeurs de B pour une valeur de A considérée

Exemples

Obtenir la liste des acheteurs qui ont acheté tous les vins de 2000.

select distinct NoA from Achats A1 where not exists (
(select NoV from Vins where Milv=2000)
minus
(select A2.NoV from Achats A2, Vins where milv=2000 and A2.NoV=Vins.NoV and A1.NoA = A2.NoA));

Interrogation des tables : la division – en utilisant deux not exists

Obtenir la liste des acheteurs qui ont acheté tous les vins de 2000.

```
select distinct NoA
from Achats A1
where not exists (
( select * from vins where milv=2000
and not exists
(select * from Achats A2 where A1.NoA = A2.NoA and A2.NoV = Vins.NoV));
```


Présentation des résultats : renommage de colonne

Pour améliorer la présentation des résultats et rendre le résultat plus lisible, on peut décider de renommer une colonne :

select <attribut1> as "alias", <attribut2> as "alias" from … where …

Exemples

Quelle est la quantité moyenne de vins achetée par les acheteurs ?

```
select avg(Qu) as "Quantité Moyenne" from achats;
```

Langage de Manipulation de Données

Présentation des résultats : ajout de texte

Pour améliorer la présentation des résultats, il est possible de faire afficher un texte au lieu d'un tableau :

select 'la valeur de l'attribut1 est:' || <attribut1>, … from … [where ...];

Exemples

Afficher les acheteurs et pour chacun d'eux, la quantité totale de vins achetée si celle-ci est supérieure à 50.

select 'L\'acheteur numéro ' || NoA || 'a acheté une quantité de vins égale à ' || sum(Qu)
from achats group by NoA having sum(Qu)>50;

Langage de Définition de Données

Création des tables

create table <nom table> (
<nom attribut1> <type> constraint <nom> <contrainte1 attribut1>,
<nom attribut2> <type> constraint <nom> <contrainte1 attribut2>,
...,
constraint <nom> <contraintes associées à la table>
);

Le mot clé facultatif « constraint » suivi d'un nom permet de nommer la contrainte déclarée juste après. Il est vivement conseillé de l'utiliser le plus souvent possible car les messages d'erreurs seront plus explicites lors de l'importation ou la mise à jours de données .

Les types les plus courants sous **Oracle** sont :

- **number(x,y)** où x et y représentent respectivement le nombre de chiffres avant et après la virgule ;
- **char(n)** : chaine de n octets (max 2000 octets) ;
- **varchar2(n)** : chaine de taille variable avec au maximum n octets (max 4000 octets) ;
- **clob** ou **nclob** : chaine de grande taille variable (environ 4G octets maxi) ;
- **Date** : pour stocker des dates ;
- **Timestamp** : pour stocker des dates avec une heure.

Langage de Définition de Données

Création des tables

Les contraintes les plus courantes sont :
•**null** / **not null** : valeur nulle autorisée ou non ;
•**primary key** : l'attribut est clé primaire (la clé primaire n'est composée que d'un seul attribut) ;
•**unique key** : l'attribut est une clé secondaire ;
•**references <nomtable>(<attribut>)** : clé étrangère vers un attribut d'une autre table ;
• **on delete cascade (ou null)** : valable uniquement pour les clés étrangères après le mot clé references. Le n-uplet associé sera supprimé (ou mis à null) automatiquement si la source de la clé étrangère est supprimée ;
•**check (<expression booléenne>)** : autre contrainte sur l'attribut. L'expression booléenne est construite comme dans le cas d'une sélection dans une clause *where* d'une requête.

La contrainte de table la plus courante est :
• **primary key(att$_1$, att$_2$, ..., att$_n$)** : clé primaire composée de plusieurs attributs.

Langage de Définition de Données

Suppression des tables

drop table <nom table>;

Ajout d'un attribut

alter table <nom table> add <nom attribut> <type attribut> constraint <name> <contrainte>, … ;

Augmenter la taille d'un attribut

alter table <nom table> modify <nom attribut> <type attribut>;

Supprimer un attribut (quand cela est possible)

alter table <nom table> drop column <nom attribut>;

La commande *alter table* est très riche. De nombreuses autres opérations sont disponibles.

Insertions des données dans une table

insert into <table name>(att_1, att_2, ..., att_n) values(<val_1>, <val_2>, ..., <val_n>);

Si une valeur est donnée pour l'ensemble des attributs, il n'est pas utile de préciser la liste des attributs après le nom de la table. Dans ce cas, il faudra donner les valeurs dans le même ordre que la liste des attributs lors de la création de la table.

Suppressions de n-uplets

delete from <table name> [where ...];

La clause *where* fonctionne de la même manière que dans les requêtes.

Modifications de n-uplets

update <nom table> set <nom attribut>=<expression> [where ...];

<expression> doit être compatible avec le type de l'attribut. La clause *where* est facultative si la modification porte sur l'ensemble de la table.

QUATRIÈME PARTIE

LES AUTRES PARADIGMES DE BASES DE DONNÉES : LES BASES NOSQL

Bibliographie

A First Course in DATABASE SYSTEMS
3rd edition
J. D. Ullman, J. Widom
Pearson, Prentice Hall, 2008

NoSQL Distilled
A Brief Guide to the Emerging World of Polyglot Persistence
P. J. Sadalage, M. Fowler
Addison-Wesley, 2013

Documentation en ligne PostgreSQL
http://www.postgresql.org/docs/

Origine du mot

- **NoSQL** : reconnu maintenant comme *Not Only SQL*.

Mais, que signifie vraiment « Not Only » ??? Pas grand chose... Pas formellement défini

- **NoSQL** → Open-source, distributed, non-relational databases

Historique

Le terme « NoSQL » dans sa signification actuelle vient du nom d'un séminaire (pendant une conférence sur Hadoop) qui a eu lieu en 2009 sur les nouveaux types de bases de données qui n'utilisent pas SQL. Il fallait un nom qui fasse un bon hashtag sur Twitter : court, facile à mémoriser, et non populaire sur Google pour espérer pouvoir remonter dans les premiers résultats.

Les systèmes du type NoSQL sont en fait utilisés depuis longtemps :
LDAP, stockage des cookies des navigateurs, ...

Rappels sur les bases de données relationnelles

Les principales opérations (LDD et LMD, SQL–92) :

- Interrogation (*select*)
- Insertion (*insert*)
- Mise à jour (*update*)
- Suppression (*delete)*

sont regroupées dans des **transactions**.

Une **transaction** est un regroupement atomique d'un ensemble d'opérations.

Un **Système de Gestion de Bases de Données** (SGBD) se charge de la mise en œuvre des transactions.

Régies par le **Modèle relationnel** (Codd, 70) : tous les SGBD fonctionnent à peu près de la même façon (au moins pour le LDD et LMD) : « One Size Fits All ».

Rappels sur les bases de données relationnelles

Propriétés ACID des transactions :

- **A**tomicité
- **C**ohérence (Consistency)
- **I**solation
- **D**urabilité

Rappels sur les bases de données relationnelles

- **Atomicité**

  Toute transaction doit s'exécuter entièrement sans erreur ou pas du tout (ne doit alors laisser aucune trace de son existence).

- **Cohérence (Consistency)**

  La base de données doit toujours être dans un état cohérent entre deux transactions (même si une transaction ne se termine pas correctement).

Rappels sur les bases de données relationnelles

- **Isolation**

  Toute transaction doit s'exécuter comme si elle était la seule sur le système. Aucune dépendance possible entre les transactions.

- **Durabilité**

  Quand la transaction est terminée, la base de données doit être dans un état stable et durable dans le temps (persistance).

Rappels sur les bases de données relationnelles

Problèmes à éviter lors du déroulement des transactions

- *Lectures impropres*
  Une transaction lit des données écrites par une autre transaction non validée.

- *Lectures non reproductibles*
  Une transaction relit des données (donc déjà lues précédemment) mais les données ont été modifiées (par une autre transaction validée entre les deux lectures).

- *Lectures fantômes*
  Une même requête exécutée dans deux transactions ne renvoie pas le même résultat.

Limites des bases de données relationnelles

Relation
Contient un ensemble de *n*-uplets (les lignes) qui contiennent des attributs (les colonnes). Le domaine de chaque attribut est fixé et doit être composé de valeurs atomiques simples, *i.e.* non décomposables (1FN).
Une requête SQL de type *select* retourne toujours une relation.

Comment stocker des structures plus complexes ?
Possible dans la plupart des langages de programmation (notamment à objets) mais pas avec le modèle relationnel afin de respecter la normalisation et la définition d'une relation.
Un changement de modèle et une traduction des données sont donc nécessaires.

Impedance mismatch

Pour la petite histoire : tentative de solutions avec les BD objets dans les années 1990.

Limites des bases de données relationnelles

- Une BD relationnelle est une couche spécifique pour accéder aux données (architecture à trois couches (niveaux, tiers) ; 3-tier en anglais).
- De nos jours, intégration de la base de données dans l'application pour simplifier l'architecture du système et accéder plus rapidement aux données (pas de serveur dédié).
- Les BD relationnelles sont conçues pour fonctionner sur un serveur unique avec des volumes de données modérés (maintenance des index, maintien de la cohérence).
- Contraintes ACID impossibles à respecter sur un cluster (et *a fortiori* sur Internet) pour conserver un fonctionnement correct (haute disponibilité, rapidité d'exécution et cohérence permanente de la base, partition accidentelle du cluster).

Les systèmes NoSQL

- *Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable* (http://nosql-database.org/).
- Pas un Système de Gestion de Bases de Données mais plus un système de stockage de données.
- Pas de modèle pré-défini, pas de contrainte stricte comme en relationnel, tout doit rester simple pour aller vite. L'ajout d'un serveur doit améliorer les performances.
- Maintien de la cohérence simple. Exemple du QUORUM : Soit un nombre de serveurs *n*. Dès que *w* serveurs sur les *n* ont confirmé la bonne exécution d'un ordre d'écriture, l'écriture est considérée comme valide sur l'ensemble du cluster. Les autres serveurs finiront par se mettre à jour. En relationnel, il faudrait attendre que tous les serveurs aient confirmé.

Les systèmes NoSQL : CAP

Théorème CAP (proposé par Brewer, 2000 et démontré par Gilbert et Lynch 2002)

- Dans un environnement distribué, il n'est pas possible de respecter simultanément :
  - –

Les systèmes NoSQL

À l'opposé de ACID, les systèmes NoSQL sont parfois présentés comme BASE :

Basically Available, Soft state, Eventually consistent

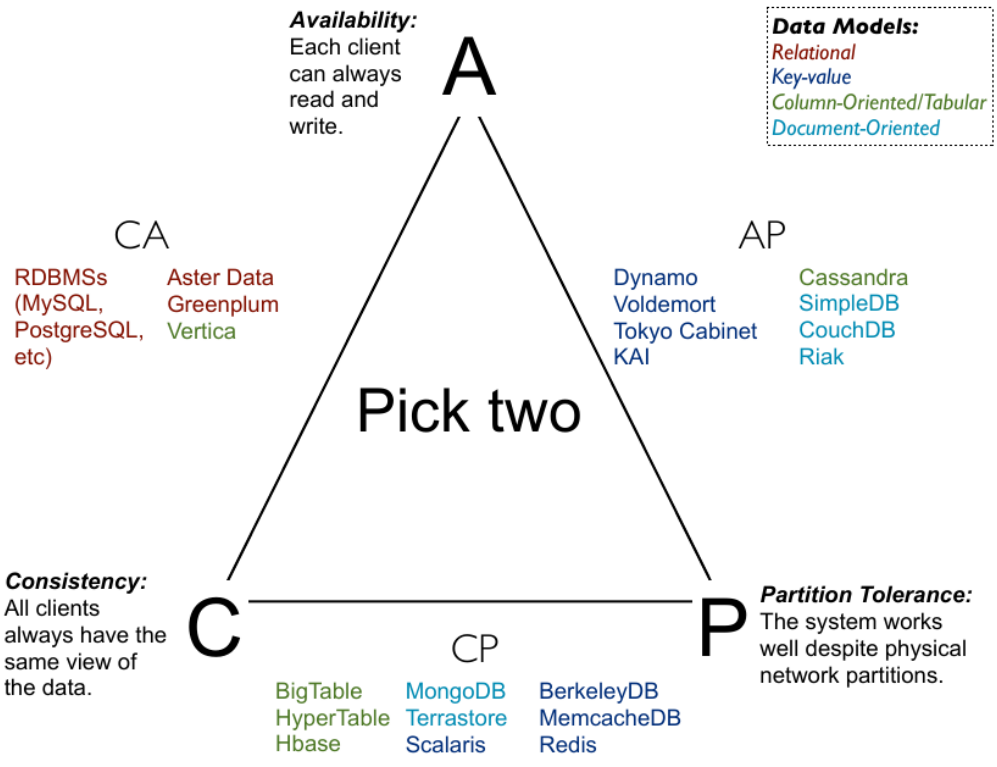
- **Eventual consistency :** la cohérence du système est atteinte au bout d'un certain temps.
- **Soft state :** l'état du système peut donc changer même sans action de l'utilisateur pour atteindre un état global cohérent. Il faut donc que le système soit capable de fonctionner dans des états transitoires.
- **Basically available** : le système reste donc globalement opérationnel

Les systèmes NoSQL : choix

4 grands types de systèmes

- Orienté clé-valeurs
- Orienté documents
- Orienté colonnes
- Orienté graphes

Les systèmes NoSQL : choix



Les systèmes NoSQL : clé-valeurs

- Rapidement : stockage simple et direct d'un tableau associatif (hash-table). Possible en relationnel avec une table à deux attributs dont les accès se font uniquement par la clé primaire.
- Système le plus simple : lecture, écriture et suppression. Ne fait que stocker des données en rapport avec une clé.
- Implique pas de relation entre les données. Le système ne connaît pas les données ou ce qu'elles représentent (sémantique associée).
- Interrogation uniquement par la clé. On récupère une donnée (blob notamment) associée à une clé.
- **Exemples de systèmes** : Riak, Project Voldemort, DynamoDB, Redis, Berkeley DB
- **Exemples d'usages** : stockage d'informations de session, profil utilisateurs, préférences, paniers d'achats

Les systèmes NoSQL : Documents

- Basé sur un système clé-valeur
- Mais la valeur est une structure que le système connaît et peut donc parcourir.
- Pour la plupart des systèmes pas d'opérations croisées entres plusieurs documents.
- **Exemples de systèmes** : MongoDB, CouchDB, RavenDB
- **Exemples d'usages** : stockage d’événements (event logging), CMS et dérivés, applications e-commerce.

Les systèmes NoSQL : colonnes

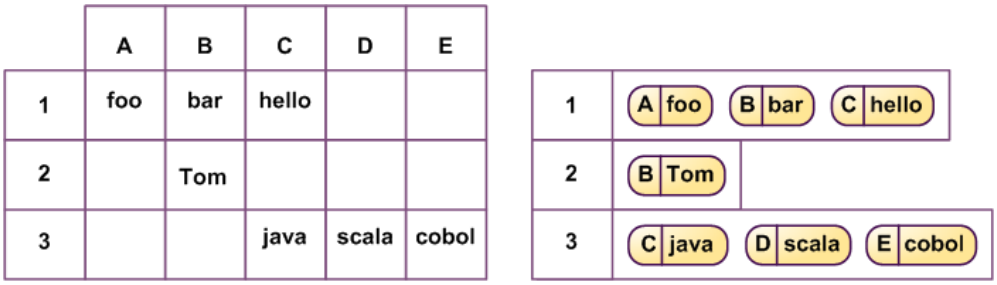
- « Opposé » d'une BD relationnelle : en relationnel, le modèle est orienté *n*-uplets dans des tables, le nombre de colonnes est fixé. Avec un modèle orienté colonnes, on ne stocke que les colonnes qui ont des valeurs non nulles regroupées dans des familles de colonnes.

- Les colonnes sont stockées sous une forme clé-valeur. On peut ajouter une colonne dans un enregistrement aussi facilement que l'ajout d'un n-uplet en relationnel.

Les systèmes NoSQL : colonnes

- « Opposé » d'une BD relationnelle : en relationnel, le modèle est orienté *n*-uplets dans des tables, le nombre de colonnes est fixé. Avec un modèle orienté colonnes, on ne stocke que les colonnes qui ont des valeurs non nulles regroupées dans des familles de colonnes.

- Les colonnes sont stockées sous une forme clé-valeur. On peut ajouter une colonne dans un enregistrement aussi facilement que l'ajout d'un n-uplet en relationnel.



	A	B	C	D	E
1	foo	bar	hello		
2		Tom			
3			java	scala	cobol

Organisation d'une table dans une BDD relationnelle

1	A: foo	B: bar	C: hello
2	B: Tom		
3	C: java	D: scala	E: cobol

Organisation d'une table dans une BDD orientée colonnes

http://blog.xebia.fr/2010/05/04/nosql-europe-bases-de-donnees-orientees-colonnes-et-cassandra/

Les systèmes NoSQL : colonnes

- Coût de stockage d'une valeur nulle : 0
- Stockage de plusieurs millions de colonnes

- **Exemples de systèmes** : Cassandra, HBase, Hypertable
- **Exemples d'usages** : stockage d’événements (event logging), CRM (Customer Relationship Management), CMS (Content Management System), compteurs (comptage et trie des visiteurs passant sur une page d'un site)

Les systèmes NoSQL : Graphes

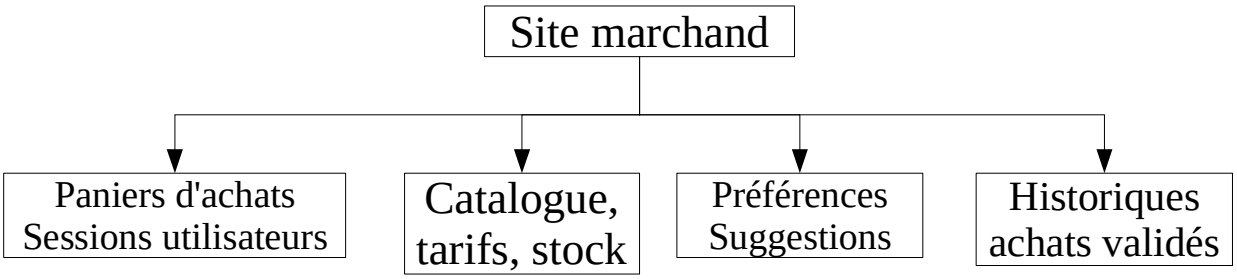
- On ne stocke plus un simple ensemble de données mais des relations.
- Stockage d'entités et de relations entre les entités. On peut ajouter des propriétés aux relations et aux entités.
- Cohérence forte des données. Pas d'arête pendante.
- Difficile de monter en charge (ajout de serveurs). Le stockage d'un graphe sur plusieurs serveurs est un problème difficile (problème de performance même pour un simple calcul de chemin si le graphe est réparti sur plusieurs serveurs).

- **Exemples de systèmes** : Neo4J, Infinite Graph, FlockDB (Twitter)
- **Exemples d'usages** : réseaux sociaux, système de recommandations, services géolocalisés

Conclusion

• Les SGBDR sont maintenant une option parmi un panel de plus en plus large.

•Il faut apprendre à utiliser la base de données la plus adaptée aux données à gérer quitte à en utiliser plusieurs.



Conclusion

• Les systèmes NoSQL ne remplaceront jamais les bases de données relationnelles. Il faut les voir comme une alternative après un quasi monopole du relationnel de plusieurs décennies (*One size fits all*).

• Les systèmes de gestion de données relationnelles sont maintenant une option parmi d'autres. Leur généricité leur permet d'être utilisés dans de nombreux cas mais les rends peu performants et complexes sur certains types de données (masse de données, attributs trop nombreux, …).

• Il faut savoir utiliser la solution la plus adaptée et pourquoi pas en utiliser plusieurs.

• Pas de schéma figé comme en relationnel, moins de contraintes. On déporte sur l'application la gestion du modèle de données et sa connaissance.

• Changement drastique des méthodes de développements notamment par l'absence de cohérence immédiate pour la tolérance au partitionnement et les évolutions du schéma.