Formation CAPES option Info
Le modèle relationnel

Romuald THION

Maître de conférences – Université Lyon 1
romuald.thion@univ-lyon1.fr
http://liris.cnrs.fr/romuald.thion/

31 octobre 2016

Plan

Jeu de données *Stanford*

Student

sID	sName	GPA	sizeHS
123	Amy	3,9	1.000
234	Bob	3,6	1.500
345	Craig	3,5	500
456	Doris	3,9	1.000
567	Edward	2,9	2.000
678	Fay	3,8	200
789	Gary	3,4	800
987	Helen	3,7	800
876	Irene	3,9	400
765	Jay	2,9	1.500
654	Amy	3,9	1.000
543	Craig	3,4	2.000

College

cName	state	enrollment
Stanford	CA	15.000
Berkeley	CA	36.000
MIT	MA	10.000
Cornell	NY	21.000

Jeu de données *Stanford*

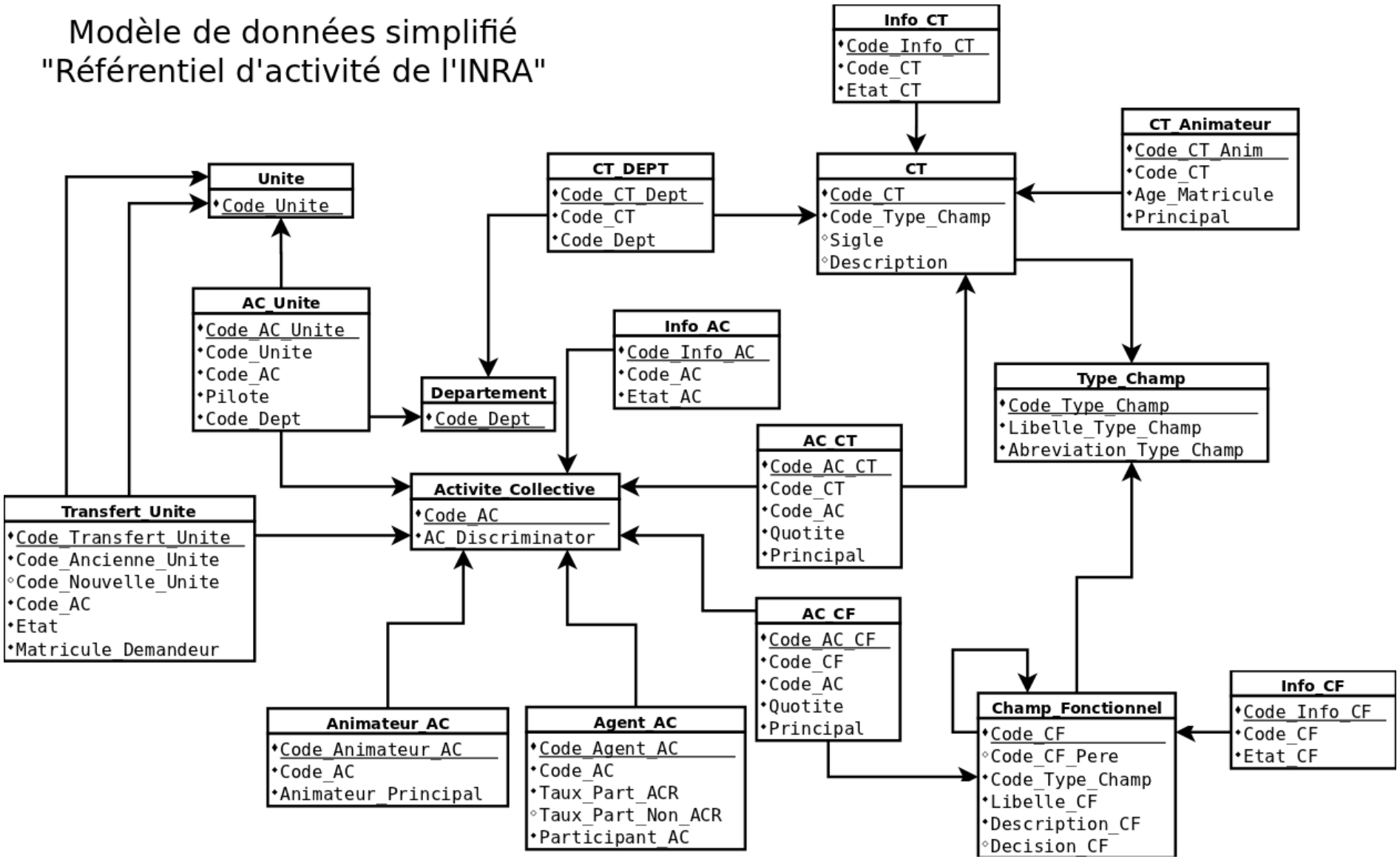
Apply

sID	cName	major	dec.
123	Stanford	CS	Y
123	Stanford	EE	N
123	Berkeley	CS	Y
123	Cornell	EE	Y
234	Berkeley	biology	N
345	MIT	bioeng.	Y
345	Cornell	bioeng.	N
345	Cornell	CS	Y
345	Cornell	EE	N
678	Stanford	history	Y

Apply

sID	cName	major	dec.
987	Stanford	CS	Y
987	Berkeley	CS	Y
876	Stanford	CS	N
876	MIT	biology	Y
876	MIT	marine bio.	N
765	Stanford	history	Y
765	Cornell	history	N
765	Cornell	psych.	Y
543	MIT	CS	N

Exemple *référentiel d'activités*



Gestion des données

Conditions nécessaires d'utilisation

- Pérennes (sauvegarde, archivage, reprise après panne. . .)
- Cohérentes(mises à jour, accès concurrentiels, réplications. . .)
- Accessibles de manière juste et efficace

On enregistre les données d'étudiants dans des fichiers

- Chaque application doit gérer un ensemble de fichiers de données et les maintenir
- Variation des formats des fichiers
- Mises à jour redondantes (erreurs et incohérences)
- mais aussi : efficacité, concurrence, sécurité, dépendance vis-à-vis du programme . . .

Système de Gestion de Bases de Données

Objectif : pallier aux insuffisances de la gestion de données directe via des fichiers

Bases de données (relationnelles)

- Données enregistrées
- Cohérentes
- De redondance minimale
- Dont la structure est indépendante des applications
- Accessible de manière concurrente

Système de Gestion de Bases de Données

Le concepteur gère

- La structuration
- Non redondance
- Mise en commun et la distribution éventuelle des données

Le Système de Gestion de Bases de Données gère

- le stockage
- la disponibilité des données
- l'accès
- la concurrence
- et propose les fonctionnalités et outils connexes

Système de Gestion de Bases de Données

SGBD : ensemble d'outils logiciels permettant la création et l'utilisation de bases de données (relationnelles).

Fonctions essentielles

- Définition d'une base de donnée :
  - spécification des type des données
  - structuration des données
  - contraintes d'intégrité
- Interrogations des données
- Mise à jour des données
- Garantie de l'intégrité des données
- Gestion de la concurrence
- Gestion de la sécurité et de la confidentialité

Système de Gestion de Bases de Données

Données structurées : le schéma

Langage de Description de Données (LDD) :

- organisation des données
- type des données
- contraintes d'intégrité

Approche orientée données

Description unique des données, commune aux différentes applications : ce n'est pas l'application qui guide la structuration mais les données à représenter

Système de Gestion de Bases de Données

Langage déclaratif : manipulation de données

Langage de Manipulation de Données (LMD) :

- Recherche, création, modification et suppression
- On spécifie ce que l'on veut faire . . .
- . . . et pas comment le faire

Indépendance logique/physique

Séparation entre le quoi et le comment : principe fondateur du modèle relationnel (et des approches déclaratives en général)

Système de Gestion de Bases de Données

Interaction avec le SGBD

- Interpréteur de commandes
- Interface de développement (SQLDeveloper)
- Dans une langage de programmation
  - C, C++, Java, Python, PHP, Haskell ...
  - via les bibliothèques adéquates
- Via des environnements *ad hoc* : aide à la formulation de requêtes, formulaires web ...

La boîte noire SGBD

Toutes les interactions passent par LDD et LMD via la couche externe du SGBD

Modèle relationnel

Modèle de donnée

Définition (formelle) d'un mode de représentation de l'information

Indépendance de la représentation physique : simplifier

- l'administration
- l'optimisation
- l'utilisation

Expressivité & performance

Modèle relationnel

Modèle ensembliste

- Les objets sont simples, atomiques :
  - entier, flottants, chaînes de caractères, dates...
- Pas d'objets complexes :
  - Pas de listes, pas de tableaux, pas de structures...
- On se dotte des opérations ensemblistes usuelles :
  - Union ($\cup$), intersection ($\cap$), différence ($-$), produit ($\times$)
- On utilise les relations (au sens de la théorie des ensembles) définir les données :
  - Une relation portant entre n ensembles $E_1, \ldots, E_n$ est un sous-ensemble du produit cartésien $E_1 \times \cdots \times E_n$.

Schéma

Ensemble d'attributs

- Décrit les données atomiques que l'on veut manipuler, par exemple `titre`, `annee`, `genre`
- Les attributs sont (souvent) typés, par exemple `titre:String`, `annee:Int`

Ensemble de relations entre les attributs

- Représente les liens entre les données atomiques, par exemple `Film(titre, annee, genre)`
- Arité d'une relation : nombre d'attributs de cette relation
- une table est une relation d'un schéma

Conception de schéma

Modélisation relationnelle

- La création est simple . . . une fois toutes les relations déterminées. . .
- Le choix des relations est difficile :
  - Il détermine les caractéristiques de qualité de la base : performances, exactitude, exhaustivité, disponibilité des informations
- Méthodes et outils de conception :
  - Schéma entités-associations
  - Merise
  - UML

Instances

Instances de relations et de bases de données

Une instance d'une relation $R(A_1, \ldots, A_n)$ est un sous-ensemble fini du produit cartésien des domaines de ses attributs : un ensemble fini de n-uplets (tuples en anglais).
Une instance d'une base de données est un ensemble d'instances de relations (une par relation)

Instance de la relation `Film(titre, annee, genre)`

{(`Alien`, `1979`, `Science-fiction`)
(`Vertigo`, `1958`, `Suspense`)
(`Volte-face`, `1997`, `Thriller`)
(`Pulp fiction`, `1995`, `Policier`)}

Instances

Ce sont les schémas (méta-données) et les instances (données) de relations qui sont stockées.

Représentation tabulaire

$Film$

titre	annee	genre
Alien	1979	Science-fiction
Vertigo	1958	Suspense
Volte-face	1997	Thriller
Pulp fiction	1995	Policier

Théorie VS pratique

En théorie (avec les ensembles)

- $\{x, x\} = \{x\}$
- $\{x, y\} = \{y, x\}$

Conséquences

- Toutes les valeurs des attributs sont définies
- Absence de doublons
- L'ordre est tuples n'est pas important

En pratique, les choses sont différentes

Présentation générale

L'algèbre relationnelle

- LE langage de manipulation de données
- Base pour l'étude théorique :
  décidabilité, expressivité, complétude, complexité . . .
- Fondement pour les langages concrets (SQL)
- Inclus tous les concepts fondamentaux des BD

Concepts clefs

- Ensemble d'opérateurs algébrique
- Tout est relation
- Pas de détails de l'implémentation

Opérateurs de base (SPJRU)

- Sélection : $\sigma_C(R)$
  filtrer les tuples d'une relation
- Projection : $\pi_{A_1,\ldots,A_n}(R)$
  sélectionner les attributs
- Jointure : $R \bowtie S$
  combiner deux relations ensemble
- Renommage : $\rho_{A/A',B/B',\ldots}(R)$
  opérateur purement syntaxique, utilisé pour lever les ambiguïtés
- Union : $R \cup S$
  réunis les tuples dans une même relation

Présentation générale

Opérateurs dérivés (sucre syntaxique)

- Intersection : $R \cap S$
- Différence : $R \setminus S$ (ou $R - S$)
- Produit : $R \times S$
- Division : $R \div S$

Classification

- Opérateurs relationnels : sélection (σ), projection (π), jointure ($\bowtie$), renommage (ρ)
- Opérateurs ensemblistes : union ($\cup$), intersection ($\cap$), différence ($\setminus$), produit ($\times$)

Sélection (σ)

Sémantique

- $\sigma_C(R)$ sélection les n-uplets de R qui satisfont la condition C (filtre)
- Condition : expression utilisant
  - des opérateurs booléens ($\wedge, \vee, \neg$)
  - des prédicats ($=, <, >, \leq, \geq \ldots$)
  - des fonctions ($\times, +, - \ldots$)
  - entre attributs de R et constantes

Exemple

$$\sigma_{(NoDept=5)}(Employe)$$

Sélection (σ)

$Employe$

prenom	nom	noDept
John	Smith	5
Ricardo	Brown	3
Susan	Yao	5
Daniel	Johnson	2
Francis	Johnson	2
Ramesh	Shah	4
Ramesh	Shah	2

$\sigma_{(NoDept=5)}(Employe)$

prenom	nom	noDept
John	Smith	5
Susan	Yao	5

Quels sont les résultats de $\sigma_{(1=1)}(Employe)$, de $\sigma_{(0=1)}(Employe)$?

Projection (π)

Sémantique

- $\pi_{A_1,\ldots,A_k}(R)$ ne garde que les attributs $A_1, \ldots, A_k$ de la relation R.
- On ne supprime pas de ligne mais des colonnes
- Élimination des doublons (sens des ensembles)

Exemple

$$\pi_{nom,noDept}(Employe)$$

Projection (π)

$Employe$

prenom	nom	noDept
John	Smith	5
Ricardo	Brown	3
Susan	Yao	5
Daniel	Johnson	2
Francis	Johnson	2
Ramesh	Shah	4
Ramesh	Shah	2

$\pi_{nom,noDept}(Employe)$

nom	noDept
Smith	5
Brown	3
Yao	5
Johnson	2
Shah	4
Shah	2

Question : quel doublon a été éliminé ?

Jointure ($\bowtie$)

Sémantique

- R et S ont les attributs $A_1, \ldots, A_k$ en commun
- On obtient l'ensemble des n-uplets constitués à partir de n-uplets n_1 de R et de n-uplets n_2 de R ayant les mêmes valeur pour les attributs $A_1, \ldots, A_k$.
- Les n-uplets obtenus sont construits comme suit : on ajoute à n_1 la valeur des attributs de n_2 qui ne sont pas dans R

Exemple

$$Employe \bowtie Emplacement$$

Employe

prenom	nom	noDept
John	Smith	5
Ricardo	Brown	3
Susan	Yao	5
Francis	Johnson	2
Ramesh	Shah	4

Emplacement

noDept	building
1	centre
3	sud
4	est
5	ouest

$Employe \bowtie Emplacement$

prenom	nom	noDept	building
John	Smith	5	ouest
Ricardo	Brown	3	sud
Susan	Yao	5	ouest
Ramesh	Shah	4	est

Question : quel employé a disparu, pourquoi ?

Renommage (ρ)

Sémantique

- $\rho_{A_1/A'_1,...,A_k/A'_k}(R)$
- Changement du nom d'un ou plusieurs attributs d'une relation R : A_1 devient A'_1, ..., A_k devient A'_k
- Utile en cas de problème d'homonymie ou avant certaines opérations ensemblistes.

Exemple

$$\rho_{Prenom/First,Nom/Last}(Employe)$$

Renommage (ρ)

$Employe$

prenom	nom	noDept
John	Smith	5
Ricardo	Brown	3
Susan	Yao	5
Daniel	Johnson	2
Francis	Johnson	2
Ramesh	Shah	4
Ramesh	Shah	2

$\rho_{Prenom/First,Nom/Last}(Employe)$

first	last	noDept
John	Smith	5
Ricardo	Brown	3
Susan	Yao	5
Daniel	Johnson	2
Francis	Johnson	2
Ramesh	Shah	4
Ramesh	Shah	2

Quel est l'effet de cet opérateur sur les données ?

Union ($\cup$)

Sémantique

- $R \cup S$ crée une relation comprenant tous les n-uplets existants dans l'une ou l'autre des relations R et S.
- Les deux relations doivent avoir le même nombre d'attributs, et les mêmes types (*i.e.* même domaine)
- Éliminations des doublons.

Exemple

$$Etudiants \cup Profs$$

$Etudiants$

prenom	nom
Susan	Yao
Ramesh	Shah
Barbara	Jones
Amy	Ford
Jimmy	Wang

$Profs$

prenomP	nomP
John	Smith
Ricardo	Brown
Susan	Yao
Francis	Johnson
Ramesh	Shah

$Etudiants \cup Profs$

prenom	nom
Susan	Yao
Ramesh	Shah
Barbara	Jones
Amy	Ford
Jimmy	Wang
John	Smith
Ricardo	Brown
Francis	Johnson

Intersection ($\cap$)

Sémantique

- $R \cap S$ crée une nouvelle relation de même schéma et dont les tuples sont ceux qui apparaissent dans R et dans S
- Les deux relations doivent avoir le même nombre d'attributs, et les mêmes types (*i.e.* même domaine)

Exemple

$$Etudiants \cap Profs$$

$Etudiants$

prenom	nom
Susan	Yao
Ramesh	Shah
Barbara	Jones
Amy	Ford
Jimmy	Wang

$Profs$

prenomP	nomP
John	Smith
Ricardo	Brown
Susan	Yao
Francis	Johnson
Ramesh	Shah

$Etudiants \cap Profs$

prenom	nom
Susan	Yao
Ramesh	Shah

Différence (\)

Sémantique

- $R \setminus S$ crée une relation de même schéma et de population égale à l'ensemble des n-uplets de R privée de ceux de S, c'est-à-dire les n-uplets qui se trouvent dans R mais pas dans S.
- Les deux relations doivent avoir le même nombre d'attributs, et les mêmes types (*i.e.* même domaine)

Exemple

$$Etudiants \cap Profs$$

$Etudiants$

prenom	nom
Susan	Yao
Ramesh	Shah
Barbara	Jones
Amy	Ford
Jimmy	Wang

$Profs$

prenomP	nomP
John	Smith
Ricardo	Brown
Susan	Yao
Francis	Johnson
Ramesh	Shah

$Etudiants \setminus Profs$

prenom	nom
Barbara	Jones
Amy	Ford
Jimmy	Wang

Produit cartésien ($\times$)

Sémantique

- $R \times S$ crée une nouvelle relation où chaque n-uplet de R est associé à chaque n-uplet de S
- Le nombre de lignes est $\mid R \mid \times \mid S \mid$ (où $\mid R \mid$) est le nombre de lignes dans la relation R).
- La relation de sortie contient tous les attributs de S et ceux de R (renommage !)

Exemple

$$Etudiants \times Profs$$

$Etudiants$

prenom	nom
Susan	Yao
Ramesh	Shah

$Profs$

prenomP	nomP
John	Smith
Ricardo	Brown
Susan	Yao

$Etudiants \times Profs$

prenom	nom	prenomP	nomP
Susan	Yao	John	Smith
Susan	Yao	Ricardo	Brown
Susan	Yao	Susan	Yao
Ramesh	Shah	John	Smith
Ramesh	Shah	Ricardo	Brown
Ramesh	Shah	Susan	Yao

Comment exprimer $\bowtie$ à partir de $\times$?

Egalités algébriques

Egalités algébriques

- Propriétés remarquables, prouvables (langage formel)
- Montre que l'algèbre « se comporte bien »
- Utilisées pour l'optimisation

Sélection

- $\sigma_A(R) = \sigma_A\sigma_A(R)$
- $\sigma_{A \wedge B}(R) = \sigma_B\sigma_A(R) = \sigma_A\sigma_B(R)$
- $\sigma_{A \vee B}(R) = \sigma_A(R) \cup \sigma_B(R)$

Egalités algébriques

Opérations ensemblistes

- $\sigma_A(R \setminus P) = \sigma_A(R) \setminus \sigma_A(P) = \sigma_A(R) \setminus P$
- $\sigma_A(R \cup P) = \sigma_A(R) \cup \sigma_A(P)$
- $\sigma_A(R \cap P) = \sigma_A(R) \cap \sigma_A(P) = \sigma_A(R) \cap P = R \cap \sigma_A(P)$

Produit

- si $\sigma_A(R \times S) = \sigma_{B \wedge C \wedge D}(R \times S)$ avec B qui porte uniquement sur les attributs de R et C sur ceux de S,
- alors $\sigma_A(R \times S) = \sigma_D(\sigma_B(R) \times \sigma_C(S))$

Un des principes des optimiseurs de requêtes dans les SGBDs :
« pousser les sélections à l'intérieur »

Introduction à SQL

Structured Query Language : un langage concret

- Un langage de manipulation de données
- Un langage de description de données
- Un langage pour administrer la base, gérer les contrôles d'accès

Standards

- SQL-87 : 1987 (ISO)
- SQL-2 : 1992 (ANSI)
- SQL-3 : 1999
- SQL-2003
- SQL-2006

Introduction à SQL

Cœur standardisé, avec des variations

Source	Nom	Commentaire
ANSI/ISO	SQL/PSM	SQL/Persistent Stored Modules
MySQL	SQL/PSM	N'implémente pas tous les opérateurs
IBM DB2	SQL PL	SQL Procedural Language
Microsoft	T-SQL	Transact-SQL
Oracle	PL/SQL	Procedural Language/SQL
PostgreSQL	PL/pgSQL	PostgreSQL SQL
SQLite	SQL/PSM	N'implémente pas tous les opérateurs

La famille des SQLs

- Déclarer des requêtes de haut-niveau (c.f. algèbre)
- Des variations sur les aspects techniques (e.g., typage)
- De nombreuses extensions $+$ ou $-$ standards

Introduction à SQL

Différences avec la théorie

- Possibilités de doublons ;
- Possibilité d'ordonner le résultat des requêtes ;
- Notion de valeur non définie (logique tri-valuée).

Déclarer $\neq$ calculer

- Facile/difficile à déclarer en SQL
- Facile/difficile à calculer par le SGBD

Sont des notions complètement différentes !

SELECT

Prototype

```
SELECT att1 , att2 ...
FROM MaTable ;
```

SELECT = Projection

- Récupérer les valeurs contenusde `MaTable` en ne gardant que les attributs `att1,att2...`
- En algèbre relationnelle : $\pi_{att_1,att_2,...}(MaTable)$
- On peut remplacer `att1,att2...` par `*` pour utiliser tous les attributs
- Le point virgule `;` est utilisé pour marquer la fin des commandes SQL (†)

SELECT

Noms et moyennes des étudiants

```
SELECT sName, GPA
FROM Student;
```

$$\pi_{sName,GPA}(Student)$$

L'intégralité des étudiants

```
SELECT *
FROM Student;
```

$\pi_{sID,sName,GPA,sizeHS}(Student)$ ou simplement $Student$

WHERE

Prototype

```
SELECT att1 , att2 ...
FROM MaTable
WHERE condition ;
```

WHERE = Sélection

- La clause `WHERE` spécifie les lignes à sélectionner grâce à la condition
- En algèbre relationnelle : $\pi_{att_1,att_2...}(\sigma_{condition}(MaTable))$

WHERE

Expression des conditions

- Comparaisons (`=`, `!=`, `<`, `<=`, `>`, `>=`)
- Entre un attribut et une constante ou un autre attribut
- Différents types de données pour les constantes :
  - nombres : `1`, `1980`, `1.5`
  - chaînes de caractères : `'Martin'`,`'directeur'`
  - dates : `'1980-06-18'`
- Le formattage des types (dates en particulier) peut varier !

Combinaison d'expressions

- Le « et » logique ($\wedge$) : `AND`
- le « ou » logique ($\vee$) : `OR`
- le « non »logique ($\neg$) : `NOT`

WHERE

Exemple

```
SELECT sName
FROM Student
WHERE GPA > 3.6;
```

$$\pi_{sName,GPA}(\sigma_{GPA>3.6}(Student))$$

Exemple

```
SELECT sName, GPA
FROM Student
WHERE GPA > 3.6 or GPA < 2.9;
```

$$\pi_{sName,GPA}(\sigma_{(GPA>3.6 \vee GPA<2.9)}(Student))$$

WHERE

Opérateur `IN`

- Syntaxe : *attribut* `IN` *liste de valeurs*
- Permet d'éviter une répétition de `OR`

Exemple

```
SELECT cName, state
FROM College
WHERE state IN ('CA', 'NY', 'WY');

SELECT cName, state
FROM College
WHERE state = 'CA' OR state = 'NY' OR state = 'WY';
```

WHERE

Opérateur `BETWEEN`

- Syntaxe : *attribut* `BETWEEN` *mininum* `AND` *maximum*
- Sucre pour les conditions de la forme $l \leq x \leq u$

Exemple

```
SELECT sName, GPA
FROM Student
WHERE GPA BETWEEN 2.0 AND 3.0;
```

```
SELECT sName, GPA
FROM Student
WHERE GPA >= 2.0 AND GPA <= 3.0;
```

FROM

Prototype

```
SELECT att1 , att2 ...
FROM MaTable1 , MaTable2 , MaTable3 ...
WHERE condition ;
```

FROM = Produit

- Il est possible d'utiliser plusieurs tables dans une requête
- Cela correspond à effectuer un produit cartésien entre les différentes tables
- Si un attribut est présent dans plusieurs tables, on doit l'écrire `nom_table.att`

FROM

Exemple

```
SELECT sName, cName
FROM Student, College
WHERE Student.sizeHS >= 2000;
```

$$\pi_{sName,cName}(\sigma_{sizeHS>2000}(Student \times College))$$

sName	cName
Edward	Stanford
Edward	Berkeley
Edward	MIT
Edward	Cornell

sName	cName
Craig	Stanford
Craig	Berkeley
Craig	MIT
Craig	Cornell

$$\{Edward, Craig\} \times \{Stanford, Berkeley, MIT, Cornell\}$$

FROM

Traduction de la jointure (naturelle)

- Soit on utilise la clause `WHERE` pour la condition de jointure
- Soit on utilise la clause `NATURAL JOIN` qui traduit $\bowtie$

Exemple

```
SELECT sName, cName
FROM Student, Apply
WHERE Student.sID = Apply.sID;

SELECT sName, cName
FROM Student NATURAL JOIN Apply;
```

Noter la désambiguisation des attributs !

FROM

Renommage

- Indication des renommage dans le `FROM`
- Indispensable lorsque l'on veut effectuer des jointures ou des produits cartésiens d'une table avec elle-même

Exemple

```
SELECT Student.sID, sName, GPA, Apply.cName, enrollment
FROM Student, College, Apply
WHERE Apply.sID = Student.sID and Apply.cName = College.cName;

SELECT S.sID, S.sName, S.GPA, A.cName, C.enrollment
FROM Student S, College C, Apply A
WHERE A.sID = S.sID and A.cName = C.cName;
```

FROM

Exemple

```
SELECT S1.sID, S1.sName, S1.GPA, S2.sID, S2.sName, S2.GPA
FROM Student S1, Student S2
WHERE S1.GPA = S2.GPA;
```

S1.sID	S1.sName	S1.GPA	S2.sID	S2.sName	S2.GPA
654	Amy	3,9	123	Amy	3,9
876	Irene	3,9	123	Amy	3,9
456	Doris	3,9	123	Amy	3,9
123	Amy	3,9	123	Amy	3,9
234	Bob	3,6	234	Bob	3,6
		...			

Opérateurs ensemblistes

Traduction SQL

- $\cup$: `UNION`
- $\cap$: `INTERSECT`
- $\setminus$: `MINUS`

Effets

- Permettent de combiner les résultats de plusieurs `SELECT`
- Pas de doublons : utiliser `UNION ALL` pour les obtenir
- Les `SELECT` doivent contenir le même nombre d'attributs, de types compatibles
- Les noms des attributs sont ceux du premier `SELECT`

Opérateurs ensemblistes

Exemple

```
SELECT sID FROM Apply WHERE major = 'CS'
UNION
SELECT sID FROM Apply WHERE major = 'EE';

SELECT sID FROM Apply WHERE major = 'CS'
MINUS
SELECT sID FROM Apply WHERE major = 'EE';

SELECT sID FROM Apply WHERE major = 'CS'
INTERSECT
SELECT sID FROM Apply WHERE major = 'EE';
```

▸ Jeu de données

DISTINCT

DISTINCT = suppression des doublons

- Permet d'*éliminer les doublons* dans le résultat
- Permet de se rapprocher du comportement théorique
- Très (trop ?) utilisé en pratique :
- les doublons peuvent avoir du sens !

Exemple

```
SELECT DISTINCT College.state
FROM College
WHERE enrollment > 15000;
```

DISTINCT

Exemple

Donner (seulement) les noms des étudiants qui ont entre 3,4 et 3,5 de moyenne

```
SELECT Student.sName
FROM Student
WHERE GPA IN (3.4,3.5);

SELECT DISTINCT Student.sName
FROM Student
WHERE GPA IN (3.4,3.5);
```

les noms distincts des étudiants

$\neq$

les noms des étudiants distincts . . .

ORDER BY

Prototype

```
SELECT att1 , att2 ...
FROM MaTable1 , MaTable2 ...
WHERE condition
ORDER BY att1 , att2 ... ASC
```

ORDER BY = tri selon

- Trie du résultat selon l'ordre lexicographique (†)
- Dans un `ORDER BY` on peut préciser :
  - `ASC` (par défaut) pour indiquer un ordre croissant
  - `DESC` pour indiquer un ordre décroissant
- Lors de l'utilisation d'opérateurs ensemblistes, `ORDER BY` doit être sur la deuxième requête

IS (NOT) NULL

Prototype

```
SELECT att1 , att2 ...
FROM MaTable1 , MaTable2 ...
WHERE IS (NOT) NULL att2
ORDER BY att1 , att2 ... ASC
```

IS (NOT) NULL = tester si (non) défini

- Les valeurs non définies sont représentées par `NULL`.
- On peut tester si une valeur n'est pas définie grâce à la condition `IS NULL` (ou au contraire `IS NOT NULL`)
- Très (trop ?) utilisé en pratique ...
- ... Attention aux choix concernants `NULL` et à l'intuition

IS (NOT) NULL

```
INSERT INTO Student VALUES (432, 'Kevin', null, 1500);
INSERT INTO Student VALUES (321, 'Lori', null, 2500);
```

Quel est le résultat de cette requête ?

```
SELECT sID, sName, GPA
SELECT Student
WHERE GPA > 3.5 or GPA <= 3.5;
```

Et de celle-ci ?

```
SELECT sID, sName
FROM Student;
```

Attention à l'évaluation des attributs en présence de NULL !

Concepts importants

- Fonctions standards (catalogue de base)
- Requêtes imbriquées
- Requêtes hiérarchiques
- Agrégation de données

Ecriture de requêtes avançées

- Dépendance au SGBD, ici Oracle (11g)
- Concepts et réalisations concrètes non triviales

Requêtes imbriquées

Utiliser le résultat d'une requête dans une autre requête

- Augmentation de la puissance d'expression du langage.
- Les requêtes imbriquées sont utilisables dans les parties
  - `FROM` (à condition de renommer le résultat)
  - `SELECT` (à condition que pour chaque ligne de la requête principale, on n'ait qu'un seul résultat de la sous-requête).
  - `WHERE` et `HAVING` (opérateurs spécifiques)
- En cas de conflit sur les noms, c'est la déclaration *la plus proche* qui est utilisée.

Bonne pratique

Indenter les requêtes à chaque imbrication !

Requêtes imbriquées FROM

Principe

- Utiliser un résultat calculé dans une clause `FROM`
- Correspond à définir une table en intension

Exemple

▸ Requête originale

```
SELECT *
FROM (SELECT sID, sName, GPA, Round(GPA*(sizeHS/1000.0),1) AS
      scaledGPA
        FROM Student
        ORDER BY GPA*(sizeHS/1000.0) DESC) G
WHERE abs(scaledGPA - GPA) > 1.0;
```

Requêtes imbriquées WHERE

Principe

- Quand une requête renvoie :
  1. un unique attribut
  2. un unique tuple
- . . . alors on peut considérer ce résultat comme une valeur
- Puissant et très pratique

Erreurs classiques

1. `00913.00000 - "too many values"`
2. `01427.00000 - "single-row subquery returns more than one row"`

Requêtes imbriquées WHERE

Exemple

```
SELECT cName
FROM College
WHERE College.State = (SELECT State
                       FROM College
                       WHERE cName='Berkeley');
```

Que fait cette requête ?

Opérateurs disponibles

- `EXISTS sub-query`
  - vrai si le résultat de *sub-query* n'est pas vide
- `a IN sub-query`
  - vrai si `a` appartient au résultat de `sub-query`
- `a` □ `ANY sub-query`
  - vrai si il existe un `b` parmi les lignes de `sub-query` tel que `a` □ `b` soit vrai.
- `a` □ `ALL sub-query`
  - vrai si pour toutes les lignes de `sub-query`, `a` □ `b` est vrai.

Avec □ un prédicat dans {=, <, >, <=, >=,...}

IN, EXISTS, ANY, ALL

Exemple avec/sans `IN`

```
SELECT sID, sName
FROM Student
WHERE sID IN (SELECT sID FROM Apply WHERE major = 'CS');

SELECT DISTINCT Student.sID, sName
FROM Student, Apply
WHERE Student.sID = Apply.sID and major = 'CS';
```

IN, EXISTS, ANY, ALL

Exemple avec/sans `MINUS`

```
SELECT sID, sName
FROM Apply NATURAL JOIN Student WHERE major = 'CS'
MINUS
SELECT sID, sName
FROM Apply NATURAL JOIN Student WHERE major = 'EE';

SELECT sID, sName
FROM Student
WHERE sID IN (SELECT sID FROM Apply WHERE major = 'CS')
  and sID not IN (SELECT sID FROM Apply WHERE major = 'EE');
```

Liaisons de requêtes imbriquées

Liaisons de requêtes (références corrélées)

- Possibilité de faire référence dans les requêtes imbriquées...
- ...aux attributs de la sur-requête
- Mécanisme très (trop ?) puissant

Exemple

```
SELECT cName, state
FROM College C1
WHERE EXISTS (SELECT * FROM College C2
              WHERE C2.state = C1.state and C2.cName <> C1.
                  cName);
```

Liaisons de requêtes imbriquées

Exemple

```
SELECT cName
FROM College C1
WHERE not EXISTS (SELECT * FROM College C2
                  WHERE C2.enrollment > C1.enrollment);
```

Que fait cette requête ?

Liaisons de requêtes imbriquées

Exemple avec `IN`

▸ Jeu de données

```
SELECT sName
FROM Student
WHERE sID IN (SELECT sID FROM Apply WHERE major = 'CS');
```

Solutions incorrectes sans `IN` (pourquoi ?)

```
SELECT sName
FROM Student, Apply
WHERE Student.sID = Apply.sID and major = 'CS';
```

```
SELECT DISTINCT sName
FROM Student, Apply
WHERE Student.sID = Apply.sID and major = 'CS';
```

Liaisons de requêtes imbriquées

Déterminer le meilleur candidat

```
SELECT  S1.sName, S1.GPA
FROM Student S1, Student S2
WHERE S1.GPA > S2.GPA;

SELECT DISTINCT S1.sName, S1.GPA
FROM Student S1, Student S2
WHERE S1.GPA > S2.GPA;
```

Que font ces requêtes ?

Liaisons de requêtes imbriquées

Solution avec ALL

```
SELECT sName, GPA
FROM Student
WHERE GPA >= ALL (SELECT GPA FROM Student);
```

Attention au comportement de ALL en présence de NULL

Exemple

```
INSERT INTO Student VALUES (432, 'Kevin', null, 1500);
INSERT INTO Student VALUES (321, 'Lori', null, 2500);
```

Quel est le nouveau résultat de la requête ?

GROUP BY

Prototype

```
SELECT att1, att2...
FROM MaTable1, MaTable2
WHERE conditions
GROUP BY attk, attl...
ORDER BY atti, attj... ASC
```

`GROUP BY` = regroupement de tuples

- Indique de procéder à une répartition du résultat en groupes de n-uplets ;
- Deux n-uplets sont dans un groupe s'il ont mêmes valeurs sur les attributs `attk,attl.....`

GROUP BY

La requête renvoie un seul n-uplet par groupe

Restrictions sur `SELECT` et `ORDER BY`

Le `SELECT` et le `ORDER BY` ne peuvent utiliser que des attributs présents dans le `GROUP BY` :

- Dans un groupe, la valeur pour les attributs du `GROUP BY` est fixe, on peut donc l'utiliser.
- En revanche, la valeur pour les autres attributs peut varier, ce qui rend leur utilisation impossible
- `00979. 00000 - "not a GROUP BY expression"`

GROUP BY cname, major

Apply

sID	cName	major	dec.
123	Stanford	CS	Y
987	Stanford	CS	Y
876	Stanford	CS	N
123	Stanford	EE	N
765	Stanford	history	Y
678	Stanford	history	Y
234	Berkeley	biology	N
123	Berkeley	CS	Y
987	Berkeley	CS	Y
345	MIT	bioeng.	Y

Apply

sID	cName	major	dec.
876	MIT	biology	Y
543	MIT	CS	N
876	MIT	marine bio.	N
345	Cornell	bioeng.	N
345	Cornell	CS	Y
345	Cornell	EE	N
123	Cornell	EE	Y
765	Cornell	history	N
765	Cornell	psych.	Y

Fonctions d'agrégation

Principe

- Utilisées dans le `SELECT` et dans le `ORDER BY`
- Utilisables en conjonction avec un `GROUP BY`
- Combine les attributs qui ne font pas partie du `GROUP BY`
- Par exemple, `AVG(e)` donne la moyenne de l'expression `e` pour le groupe considéré

Attention

- On ne peut pas les utiliser dans le `WHERE`, car le `WHERE` a lieu avant regroupement (cf. clause `HAVING`)
- `00934. 00000 - "group function is not allowed here"`

Fonctions d'agrégation

Exemple

```
SELECT cName, Round(avg(GPA),1) AS Avg
FROM Student NATURAL JOIN Apply
GROUP BY cName;
```

cName	Avg
Stanford	3,7
Cornell	3,4
Berkeley	3,7
MIT	3,7

Fonctions d'agrégation

Catalogue des fonctions

- `COUNT(e)` : le nombre d'occurrences de `e`
  - les n-uplets où `e` vaut `NULL` ne sont pas comptés
  - `*` compte le nombre de n-uplets du groupe.
- `MAX(e)` : La valeur maximale de `e`
- `MIN(e)` : La valeur minimale de `e`
- `SUM(e)` : La somme des valeurs de `e`
- `AVG(e)` : La moyenne de l'évaluation de `e`
- `STDDEV(e)` : L'écart-type de `e`
- `VARIANCE(e)` : La variance de `e`

Fonctions d'agrégation

Mot clef `DISTINCT`

- L'expression `e` peut être précédée du mot clé `DISTINCT` pour éliminer les doublons
- Important pour `COUNT`, `SUM`, `AVG`, `STDDEV` et `VARIANCE`.

Exemple : quelle différence de sens ?

```
SELECT cName, COUNT (Major)
FROM Apply
GROUP BY cName;
```

```
SELECT cName, COUNT (DISTINCT Major)
FROM Apply
GROUP BY cName;
```

Fonctions d'agrégation

Le meilleur candidat de chaque établissement

```
SELECT DISTINCT cName, sName, GPA
FROM Apply NATURAL JOIN Student
WHERE (cName, GPA) IN (SELECT cName, MAX(GPA*(sizeHS/1000.0))
                       FROM Apply NATURAL JOIN Student
                       GROUP BY cName);
```

Utilisation d'une requête imbriquée pour accéder à `sName`

Fonctions d'agrégation

Fonction d'agrégation sans `GROUP BY`

- Provoque la création d'un groupe englobant tous les n-uplets sélectionnés.
- Le `SELECT` ne peut alors contenir que des fonctions d'agrégation.
- Utile pour obtenir des informations sur l'ensemble des lignes sélectionnées.

Exemple

```
SELECT AVG(GPA)
FROM Apply NATURAL JOIN Student
WHERE Major = 'CS';
```

Fonctions d'agrégation

Double regroupement

Utilisation d'une fonction d'agrégation au résultat d'une fonction d'agrégation dans un `SELECT` :

- Possible uniquement avec un `GROUP BY`.
- Cette utilisation provoque deux regroupements :
  - Premier regroupement classique par le `GROUP BY`
  - Deuxième regroupement implicite

Exemple

```
SELECT MAX(COUNT(*))
FROM Apply
GROUP BY cName;
```

HAVING

Prototype

```
SELECT att1 , att2 ...
FROM MaTable1 , MaTable2
WHERE conditions
GROUP BY attk , attl ...
HAVING condition_groupe
ORDER BY atti , attj ... ASC
```

Principe

- Le `WHERE` ne pour que sur les n-uplets individuels, avant regroupement
- La condition du `HAVING` porte sur les groupes et pas sur les n-uplets individuels

HAVING

Exemple

```
SELECT cName, COUNT(*)
FROM Apply
GROUP BY cName
HAVING COUNT(*) >=4;

HAVING cName LIKE '%rnel%' AND COUNT(DISTINCT Major) >=2;
```

Utilisation

- Usage direct des attributs du `GROUP BY` possible
- Usage des autres attributs via les fonctions d'agrégation
- Exécuté entre le `GROUP BY` et le `ORDER BY`

HAVING

Le nombre d'inscriptions et le nom des facs qui ont eu le plus gd nombre d'inscriptions

```
SELECT cName, COUNT(*) as NB
FROM Apply
GROUP BY cName
HAVING COUNT(*) = (SELECT MAX(COUNT(*))
                   FROM Apply
                   GROUP BY cName);
```

Quelle modification pour avoir la statistique par discipline ?

Langage SQL – avancé

Attention !

Partie dépendante du système

- Variations entre SGBDs
- Variations selon les versions d'un SGBD

Langage SQL – avancé

Expressions plus complexes que les simples attributs

- Fonctions et expressions numériques
- Fonctions sur les chaînes de caractères
- Fonctions sur les dates
- Fonctions de conversion

Expression utilisables

- Dans le `SELECT` : calcul sur les valeurs, définition d'attributs (l'expression sert de nom pour l'attribut) (†)
- Dans le `WHERE` : permet d'exprimer des conditions
- Dans le `ORDER BY` : possible de trier les lignes selon des valeur complexes

Catalogue de fonctions numériques

- `+` : unaire et binaire ;
- `-` : unaire et binaire ;
- `*` : multiplication et `/` : division ;
- `ABS(e)` : valeur absolue de e ;
- `COS(e)` : cosinus de e avec e en radians ;
- `SQRT(e)` : racine carrée de e ;
- `MOD(m,n)` : reste de la division entière de m par n ;
- `ROUND(e,n)` : valeur arrondie de e à n chiffres après la virgule, n optionnel et vaut 0 par défaut ;
- `TRUNC(e,n)` : valeur tronquée.

Fonctions

Renommage d'attribut

Opérateur `AS` utilisé dans `SELECT` (très utile)

Que fait cette requête ?

Exemple

```
SELECT sID, sName, GPA, Round(GPA*(sizeHS/1000.0),1) AS
    scaledGPA
FROM Student
WHERE Abs(GPA*(sizeHS/1000.0) - GPA) > 1.0
ORDER BY GPA*(sizeHS/1000.0) DESC;
```

Fonctions

Catalogue de fonctions sur les chaînes

- `LENGTH(e)` : longueur
- `TRIM(pat,e)` : nettoyage de la chaîne `e` selon expression `pat` (voir syntaxe détaillée)
- `UPPER(e), LOWER(e), INITCAP(e)` : conversion de casse majuscule, minuscule, première lettre
- `e1 || e2` : concaténation
- `REPLACE(e,old,new)` : remplace `old` par `new` dans `e`
- `INSTR(e,s)` : première occurrence de `s` dans `e`
- `SUBSTR(e,n,l)` : renvoie la sous-chaîne de e commençant au caractère n (négatif = à l'envers) et de longueur l

Fonctions

Opérateur `chaine LIKE pattern`

- Opérateur de *pattern-matching*, où le motif utilise :
  - Un caractère arbitraire (_)
  - Des caractères arbitraires (%)

Exemple

```
SELECT sID, sName
FROM Student
WHERE sName LIKE '%ar%';
```

Quel est le résultat de la requête ? ▸ Jeu de données

Comment rendre la requête insensible à la casse ?

Fonctions

Catalogue de fonctions sur les dates

- `d+n, d-n, d-d'` : arithmétique de dates
- `ADD_MONTHS(d,n)` : ajoute `n` mois à `d`
- `SYSDATE` : date courante
- `NEXT_DAY(e,d)` : le premier `d` après `e`
- `LAST_DAY(d)` : dernier jour du mois de la date `d`

Exemple (30/12/1991)

```
SELECT NEXT_DAY(TO_DATE('26121991','ddmmyyyy'),'LUNDI') AS
    uneDate
FROM dual;
```

Fonctions

Catalogue de fonctions

- `CHR(e)` renvoie le caractère dont le code ASCII est `e`
- `TO_NUMBER(e)` convertit la chaîne `e` en nombre.
- `TO_CHAR(e,format)` convertit `e` en chaîne :
  - `e` peut être un nombre ou une date ;
  - `format` indique la forme que doit avoir le résultat.
- `TO_DATE(e,format)` convertit une chaîne de caractères en date.
  - `format` choix de représentation de la date
  - `TO_DATE('12122003','ddmmyyyy')` donne la date `'2003-12-12'`

Conclusion

Le modèle relationnel

- Séparation physique VS logique
- Aspect déclaratif (le quoi, pas de comment)
- Paradigme « tout est relation »
- Le SGBD assure les fontions essentielles

L'algèbre relationnelle

- Opérations fondamentales sur les données
- Permet de comprendre le pourquoi et le comment de SQL
  - Les opérateurs de base de SQL $\cong$ ceux de l'algèbre
  - Calculer une requête SQL $\cong$ traduire en algèbre
- Tout se passe dans le meilleur des mondes (théoriques)

Conclusion

Prototype complet

```
SELECT (DISTINCT) att1 , att2 ...
FROM MaTable1 , MaTable2 ...
WHERE (c1 AND c2) OR ...
ORDER BY att1 , att2 ... ASC
(UNION, MINUS, INTERSECT)
SELECT ...
```

Requêtes simples : en résumé

- Structure de base `SELECT... FROM... WHERE...`
- Opérateurs ensemblistes `UNION`, `MINUS`, `INTERSECT`
- Opérateurs et options non-algébriques `ORDER BY`, `DISTINCT` et `IS (NOT) NULL`

Conclusion

Toute l'expressivité de SQL

- Opérateurs de l'algèbre relationnelle
  `SELECT, FROM, WHERE, UNION, MINUS ...`
- Fonctionnalités non algébriques
  `DISTINCT, ORDER BY, NULL`
- Fonctions
  `LIKE, UPPER, TO_DATE`, arithmétique de dates...
- Requêtes imbriquées
  `IN, ALL, ANY, EXISTS`
- Agrégation de données
  `GROUP BY, HAVING, COUNT ...`

Conclusion

Quelques pièges à éviter

- Usage de `DISTINCT`
- Evaluation de `NULL` dans les conditions, les agrégats
- Accès aux attributs du `GROUP BY`
- Conditions `HAVING` versus `WHERE` pour les agrégats

Attention à la mauvaise intuition sur un jeu de donnée spécifique (exemple : attributs uniques) !