Optimisation des Bases de Données

Sarah Cohen-Boulakia
http://www.lri.fr/~cohen/BD.html
cohen@lri.fr
01 69 15 32 16
Laboratoire de Recherche en Informatique

Objectifs du module

- S'assurer des bases (du cahier des charges à l'implementation SQL)
- Performance en base de données
- Comment est implementée une base de données (couche physique) ?
- Comment s'évalue une requête ?
- Comment optimiser une requête (réécriture)

Contenu du module et MCC

- Consolidation des acquis en Bases de données
  - Des connaissances en bases de données
    - Cahier des charges vers schema UML, Passage au relationnel
  - Programmation SQL et Algèbre
- Structure physique d'une BD
  - Organisation de la base sur les disques
- Indexation
  - Structures, parcours
- Optimisation des opérateurs et des requêtes
- MCC: Contrôle continu (40%, petites interros, TP noté) Exam (60%)

Rappels
(et quelques compléments !)

Sarah Cohen-Boulakia
http://www.lri.fr/~cohen
cohen@lri.fr

Plans des 2 premières séances

(1) Modélisation d'une base de données en UML

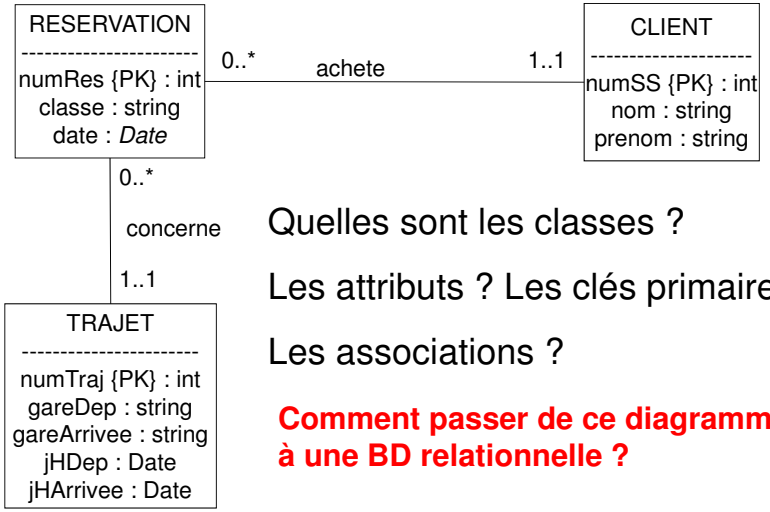
(2) Passage d'un diagramme des classes UML à un schéma de base de données

(3) SQL de base : concevoir une base, la remplir, faire des requêtes simples

(4) À la base d'SQL : l'algèbre relationnelle

(5) SQL avancé

Exemple de modèle des classes UML



Quelles sont les classes ?

Les attributs ? Les clés primaires ?

Les associations ?

Comment passer de ce diagramme à une BD relationnelle ?

Base de données relationnelle

- Un BD relationnelle = un ensemble de ***relations***
- Une relation est une **table** qui comporte
  - Des **colonnes** : *attributs*
  - Des **lignes** : *n-uplets, tuples*

Dans chaque table, un attribut (ou un ensemble d'attributs) forme une **clé primaire** qui identifie la ligne de façon unique

Relation CLIENT

numSS	nom	prenom
17902567	Payen	Olivier
2780289	Payen	Judith
29005579	Quoti	Mathilde
14504573	Renzi	Paul
17702562	Salout	Anne
24902567	Thibo	Caroline

Schéma, Instance

- ***Schéma de relation :*** nom de relation + nom et type (*domaine*) de chaque colonne
- ***Instance de relation :*** un ensemble fini de valeurs (i.e. tuples, n-uplets) respectant le schéma de relation

Schéma de la relation CLIENT

CLIENT(numSS : int, nom : string, prenom : string)

Instance : ensemble de toutes les lignes

Relation CLIENT

numSS	nom	prenom
17902567	Payen	Olivier
2780289	Payen	Judith
29005579	Quoti	Mathilde
14504573	Renzi	Bastien
17702562	Salout	Anne
24902567	Thibo	Caroline

Passage du diag. de classes UML au modèle relationnel

- Passage d'un modèle disposant de **deux structures (classes - associations)** à un modèle ne disposant que d'**une seule structure (la relation i.e. la table)**

- Application d'un **ensemble de règles** qui garantissent la **cohérence sémantique** entre le modèle UML et le modèle relationnel

Préliminaire 1 : Les valeurs *NULL*

Une valeur NULL représente une valeur **indéterminée pour un attribut**

- Par définition une clé n'est jamais à NULL
- On évite au maximum les valeurs NULL

NumSS	Nom	Prénom	Age
17902567	Payen	Olivier	32
2780289	Payen	Judith	14
29005579	Quoti	Mathilde	
14504573	Renzi	Bastien	27

NULL

Problème : Réponse à la requête "Les personnes de plus de 20 ans ": 29005579 (Mathilde Quoti) ?

Préliminaire 2 : Clé étrangère (1/2)

- **Intuition** : la valeur d'un attribut d'une table appartient toujours à l'ensemble des valeurs d'un attribut d'une autre table

- **Exemple**
  - CLIENT(**numClient : int**, adresse : string, …)
  - RESERVATION(numRes : int, **clientRes : int**)
  - **clientRes** clé étrangère
  - clientRes de RESERVATION prend ses valeurs dans l'attribut numClient de la table CLIENT : *seuls les clients recensés dans la table CLIENT peuvent faire des reservations*

Préliminaire 2 : Clé étrangère (2/2)

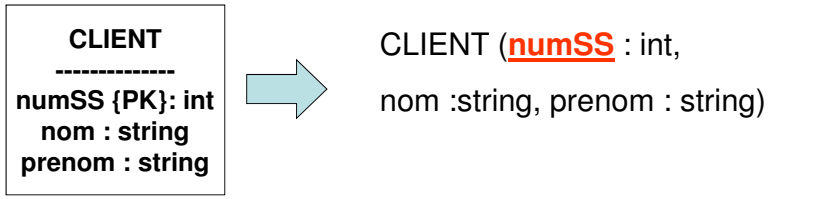
- Soient R et S deux relations. L'ensemble d'attributs X de R forme une **clé étrangère** de R ssi il existe Y clé de S tel que **R[X] ⊆ S[Y]**
  - CLIENT(**numClient : int**, adresse : string, …)
  - RESERVATION(numRes : int, **clientRes : int**)

➔ **RESERVATION[ClientRes] ⊆ CLIENT[numClient]**

- NB : *Un SGBD n'interdit pas d'avoir des valeurs NULL dans les clés étrangères*

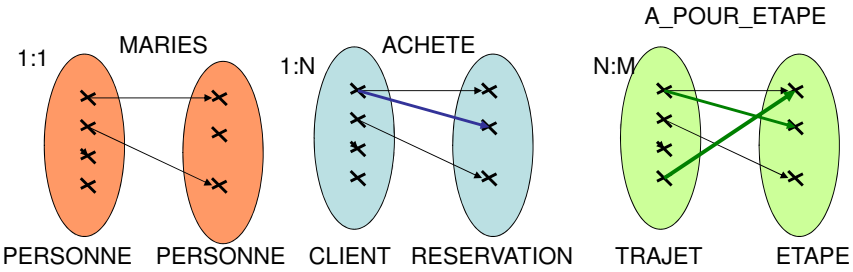
Passage UML relationnel : Classe

- Règles de passage pour une classe
  - **Classe ➔ Relation** de même nom, avec les mêmes attributs que la classe + on précise les types (domaines)
  - **L'identifiant** de la classe devient la **clé** de la relation



Associations 1:1, 1:N, N:M

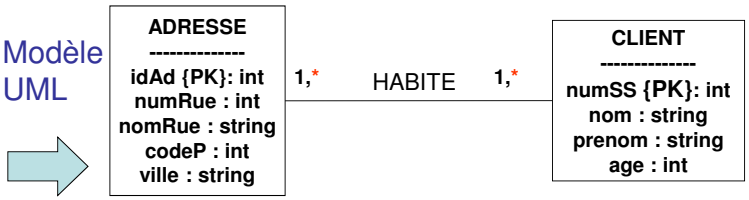
- On regarde les valeurs **maximum** des cardinalités
- 1:1 ➔ MARIES : *Un homme n'est marié qu'à une femme et une femme n'a qu'un mari (on ne gère pas les re-mariages)*
- 1:N ➔ ACHETE ((1,1),(1,*)) et CONCERNE ((1,1), (0,*))
- N:M ➔A_POUR_ETAPE : un trajet est constitué de plusieurs étapes (1,*) et une étape peut être visitée par plusieurs trains (1,*)



Passage association de type N:M (définition)

- **Association N:M ➔ relation** de même nom, avec les attributs de l'association
- La **clé** est donnée par l'ensemble des clés de chaque relation qui sont aussi des clés étrangères

Modèle UML



Modèle relationnel

ADRESSE(idAd: int, numRue: int, nomRue: string, codeP: int, ville : string)

CLIENT(numSS: int, nom : string, prenom : string, age : int)

HABITE(numSS : int, idAd : int)

Passage association de type N:M (illustration)

ADRESSE(idAd : int, numRue : int, nomRue : string, codeP: int, ville: string)

CLIENT(numSS : int, nom : string, prenom : string, age : int)

HABITE(numSS : int, idAd : int)

ADRESSE

idAd	numRue	nomRue	codeP	ville
001	18	Boileau	91400	Orsay
002	45	Gde Rue	91400	Saclay
003	1bis	Pl d'Italie	75013	Paris
004	32	Fleurs	49000	Angers

CLIENT

numSS	nom	prenom	age
17902567	Payen	Olivier	32
2780289	Payen	Judith	14

HABITE

numSS	idAd
17902567	001
17902567	004
2780289	001
2780289	002

Olivier habite à Orsay et à Anger

Judith habite à Orsay et à Saclay

Passage association de type 1:N (définition)

- La clé côté 1 devient la clé étrangère côté * (nouvel attribut dans la relation coté *)

Modèle UML

CLIENT(numSS: int, nom : string, prenom : string)

RESERVATION(numRes: int, classe : int, dateRes :Date, **numSS** : int)

Modèle relationnel

Passage association de type 1:N (clés)

CLIENT(numSS: int, nom : string, prenom : string)

RESERVATION(numRes: int, Classe : int, dateRes : Date, **numSS** : int)

- Pour chaque réservation il n'y a qu'un seul Client (1:N)
- Le numéro du Client est inclus dans la table RESERVATION
- Le client de la table RESERVATION devra apparaître dans la table Client (clé étrangère)

On applique ces connaissances

TD 1

Partie 1 (modélisation) et Partie 2 (passage au relationnel)

Plans des 2 premières séances

(1) Modélisation d'une base de données en UML

(2) Passage d'un diagramme des classes UML à un schéma de base de données

(3) SQL de base : concevoir une base, la remplir, faire des requêtes simples

(4) À la base d'SQL : l'algèbre relationnelle

(5) SQL avancé

Introduction à SQL (1/2)

- SQL **Structured Query Language**
  - SEQUEL = Structured English as a QUEry Language
  - standard ISO depuis 87
- **Avantages**
  - implanté (+ ou -) complètement sur principaux SGBDs
  - **portabilité** des applications
  - **interopérabilité**
- **Inconvénients**
  - évolution du langage par **"extensions"** (SQL2, SQL3 ...)
  - suivre l'évolution des systèmes, des architectures
  - frein à l'émergence d'un nouveau langage

Introduction à SQL (2/2)

- SQL est bien **plus qu'un langage de requêtes**
  - langage de **définition de données** (CREATE, ALTER, DROP)
  - langage de **manipulation** de données
  - **interrogations** (SELECT)
  - **mises à jour** (UPDATE, INSERT, DELETE)
  - **contrôle d'accès** aux données (GRANT, REVOKE)
- SQL est un langage **utilisable**
  - en mode **interactif**
  - associé à une **interface graphique**
  - associé à des **langages de programmation** : C, JAVA ... PHP

Contenu de ce chapitre (SQL)

- Création, suppression des **schémas** avec expression de **contraintes** et **droits** d'utilisations

Créer les tables Client et Réservation avec leurs clés primaires et étrangères
Seuls les employés peuvent consulter la base de clients

Insérer le Client numéro 23456789, dont le nom est Karime Amrane

- Création, suppression des **instances**
  - **Petits** volumes de données
  - **Gros** volumes de données

Insérer tout le fichier clients de la société Expadia

- **Interrogation simple** des données

Quelles réservations ont été faites le 12 Novembre 2015 ?

Objectifs de ce chapitre

- Donner un **aperçu des commandes SQL** (disponibles sous Postgres) pour créer/modifier/supprimer des tables et des lignes et interroger ces tables
- Comprendre **les effets de ces commandes** (erreurs potentielles etc.)
- NB : Il n'y a aucun **intérêt** à être exhaustif sur les commandes disponibles !

SCHEMAS : TABLES ET CONTRAINTES

Création de schéma (Table)

- Commande Create Table
- Spécification des attributs et de leur types
- Spécification de contraintes
  - Sur les colonnes de la tables
    - CHECK, NOT NULL
- Clé primaire
  - La ou les colonnes dont la valeur sera toujours unique dans la table
- Clé étrangère
  - La ou les colonnes qui prendront leurs valeurs dans d'autres tables (préexistantes)

Création de schéma

```
CREATE TABLE Client(
NumSS INTEGER,
Nom VARCHAR(20),
Prenom VARCHAR(20),
PRIMARY KEY(NumSS))

CREATE TABLE Reservation(
NumRes  INTEGER,
Classe  INTEGER,
DateRes  DATE,
NumClient  INTEGER,
PRIMARY KEY(NumRes),
FOREIGN KEY (NumClient) REFERENCES Client (NumSS))
```

Attribut domaine

Clé primaire (identifiant unique)

NOTICE : Primary key will create implicit index « client_pkey » for table client

Un index accélère l'accès aux données

Cle étangère : l'attribut NumClient de la table Reservation prend des valeurs issue des valeurs de l'attribut NumSS de la table Client

Clé Primaire

- Il doit toujours y avoir une clé primaire dans une table
- Une clé primaire est constituée d'un ou plusieurs attributs
- CREATE TABLE Notation

```
(idArticle integer not null,
email  varchar (40) not null,
note  integer not null,
primary key (idArticle, email));
```

Colonnes *auto incrémentales* : Intro

- !! Syntaxe non normalisée !!
- Création d'une colonne dont les valeurs s'auto incrémentent à chaque insertion
- Mot clé : Auto increment, serial…
- Pas de type à déclarer (entier)

```
CREATE TABLE Client(
    id SERIAL,
    NumSS INTEGER UNIQUE,
      …
    primary key (id))
```

Ne pas en abuser : rend la gestion des clés étrangères complexes !

Colonnes *auto incrémentales* : UNIQUE

- Attention : les redondances ne sont plus vérifiées
  - deux lignes insérées auront toujours leur valeur de colonne *serialisée différentes*

  → *Utilisation de la contrainte UNIQUE*
- CREATE TABLE Notation

```
(idNotation SERIAL,
 idArticle INTEGER,
email  VARCAHR(40),
note  INTEGER,
PRIMARY KEY (idNotation)
UNIQUE (idArticle, email));
```

NB: (idArticle, email) est une *clé secondaire*

CHECK

- CHECK assure une contrainte sur une ou plusieurs colonnes

- CREATE TABLE Products (
  Product_no SERIAL,
  Name TEXT,
  Price INTEGER CHECK (price > 0),
  Disc_price INTEGER CHECK (Disc_price > 0),
  CHECK (Price > Disc_price),
  …);

Clé étrangère

- Contrainte d'intégrité référentielle
  - restreint les valeurs prises par une colonne qui doivent apparaitre dans une autre colonne (d'une autre table)
- Attention : une clé étrangère fait toujours référence à une clé primaire

```
CREATE TABLE Reservation(
NumRes  INTEGER,
NumClient  INTEGER,
PRIMARY KEY(NumRes),
FOREIGN KEY (NumClient) REFERENCES Client (NumSS))
```

→ NumSS doit être la clé primaire de la table Client

- Rappel : une clé étrangère peut prendre des valeurs nulles

Destruction de schema

- **DROP TABLE Reservation;**

➔Détruit la table Reservation

Attention : Lorsqu'on a défini des clés étrangères, les tables doivent être détruites dans un ordre précis !

- **DROP TABLE Client;**

Si la table reservation n'est pas détruite on a une erreur :

➔***ERROR :*** *Cannot drop Client because other objects depend on it*

- On peut utiliser le mode CASCADE pour supprimer un ensemble de tables

Modification de schéma

- **ALTER TABLE Client**
  ADD COLUMN Age : Integer;

➔**Ajoute une colonne** Age à la table Client : toutes les données déjà présentes dans la table auront alors par défaut une valeur NULL pour Age

- D'autres formes de modifications sont possibles: ajout d'une clé, d'une contrainte UNIQUE/CHECK, suppression d'une colonne, modification d'un type etc.

Droits (1/2)

- On peut choisir de donner des **droits** sur des tables à certains utilisateurs (et pas à d'autres)
- On utilise les commandes **GRANT** et **REVOKE**
- **GRANT INSERT,DELETE** ON TABLE Client to ManagerUser
  - Donne **le droit d'insertion et de suppression** sur la table Client à l'utilisateur ManagerUser

Droits (2/2)

- **GRANT SELECT** ON TABLE Client TO groupWeb
  - Donne les **droits de consultation** sur la table Client au groupe d'utilisateurs groupWeb
- **REVOKE DELETE** ON TABLE Client FROM ManagerUser
  - **Retire les droits de suppression** des lignes de la table Client à l'utilisateur ManagerUser

➔ Beaucoup d'options existent (voir la documentation pour en savoir plus !)

INSTANCES : LIGNES (CONTENU)

Insertion d'instances (peu de données a la fois)

INSERT INTO Client (numSS, nom, prenom)
VALUES (1180280, 'Nidam', 'Boukhris')
➔Insert une ligne dans la table Client
Attention aux types : pour les chaînes de caractères on met des ' '

Optionnel si on respecte l'ordre des attributs donné lors de la création de la table

- On peut préciser les attributs et leur ordre
  INSERT INTO Client (nom, prenom, numSS)
  VALUES ('Nidam', 'Boukhris', 1180280)
- *Default* pour les colonnes auto-incrémentées

Mise à jour et suppression d'instances

UPDATE CLIENTS **SET** Nom = 'Ranzi'
WHERE NumSS= 14504573;

- On peut faire des mises à jour sur un **ensemble de lignes** (WHERE Nom='Payen')
- Ou des mises à jours plus **complexes**

UPDATE CLIENTS **SET Age = Age + 1**
WHERE ….

- Suppression d'instances **DELETE FROM…**
  - **NB** : Impossible de supprimer une seule occurrence d'un doublon !

Insertion de données (en grande quantité)

- Lorsque les données à charger sont dans des **fichiers** (e.g. fichier Client), les SGBD permettent de charger des données dans la base
- Sous Postgres : commande **COPY**
  - **COPY** Client FROM '/projects/projdb/FichierClient.txt'
- Sous MySQL : commande **LOAD DATA INFILE**
  - **LOAD DATA INFILE** '/projects/projdb/FichierClient.txt' INTO TABLE Client
- D'autres solutions existent

Insertion et clé primaire

Considérons l'insertion du n-uplet (17902567, 'Martin', 'Paul') dans la table Client

➔ **Impossible**

➔ un n-uplet dont la valeur de NumSS (clé primaire) est 17902567 est déjà dans la base

Relation Client

NumSS	Nom	Prénom
17902567	Payen	Olivier
2780289	Payen	Judith
29005579	Quoti	Mathilde
14504573	Renzi	Bastien
17702562	Salout	Anne
24902567	Thibo	Caroline

➔ ***ERROR:*** *duplicate key violates unique constraint « client_pkey »*

Insertion et clé étrangère

Considérons l'insertion de (5, 1, '2007-01-12', **789**) dans la table Reservation

➔ **Impossible**

➔ D'après la clé étrangère, 789 doit être l'identifiant d'un client dans la relation Client

➔ ***ERROR:*** *insert or update on table "reservation" violates foreign key constraint "reservation_numclient_fkey"*

DETAIL: *key (numClient)=(789) is not present in table "client'*

Client

NumSS	Nom	Prénom
17902567	Payen	Olivier
2780289	Payen	Judith
29005579	Quoti	Mathilde

Reservation

NumRes	Classe	DateRes	NumClient
001	1	18/11/2007	17902567
002	2	19/11/2007	2780289

Problèmes similaires lors de suppressions

Transactions simples : Objectif

- On veut pouvoir garantir la cohérence des données en cas de problème (panne…)
- On définit des **transactions** : on traite les données par **paquets**
  - Si une panne a lieu (coupure d'électricité, problème réseau etc.) la transaction en cours ne sera pas traitée mais toutes les précédentes le seront

Transactions simples

- Principe
  - Au lieu de traiter une à une les instructions, on les traite par groupes, chaque groupe est considéré comme **unitaire**
    - soit tout le **groupe est traité**
    - soit **aucun des membres** du groupe n'est traité
- Les transactions sont souvent générées

Transactions simples : Instructions SQL

```
BEGIN TRANSACTION;
    ….;
    ….;
    ….;
  COMMIT;
END TRANSACTION;
BEGIN TRANSACTION;
    ….;
    ….;
ROLLBACK;
END TRANSACTION;
```

Ensemble d'instructions SQL

Commande qui indique que l'on doit valider les changements de données depuis le dernier COMMIT ou le début de la transaction en cours

Commande qui indique que l'on doit annuler les changements de données depuis le dernier COMMIT ou le début de la transaction en cours

Transactions simples : Exemple

```
BEGIN TRANSACTION;
  Insert into Clients values
    (1,'Jue','Noe');
COMMIT;
END TRANSACTION;
BEGIN TRANSACTION;
Insert into Clients values
    (2,'Ah','Lea');
ROLLBACK;
END TRANSACTION;
BEGIN TRANSACTION;
Insert into Clients values
    (3,'Bou','Lila');
COMMIT;
END TRANSACTION;
```

Relation Client

NumSS	Nom	Prenom
1	Jue	Noe
3	Bou	Lila

- On peut faire des tests dans les transactions et choisir d'insérer certains n-uplets
- **Attention :** Lorsqu'une clé est violée ou qu'une panne mécanique a lieu : le SGBD fait **automatiquement un ROLLBACK**

Exercice

La base de données ci-dessous décrit des UEs et des enseignants : chaque UE a un enseignant responsable et plusieurs enseignants intervenants

UE

NumUE	NomUE	Mots clés	Responsable
1	Base de Données	SQL, tables, modelisation	2
2	Algorithmique	Boucles, tableaux, pointeurs	3

Enseignant

NumEns	Nom	Prenom
1	Voisin	Jean
2	Benzaken	Claudine
3	Forest	Jean

Enseigne

Prof	UE
2	1
3	1
1	2
3	2

Exercice 1

- Donnez le contenu de la table Enseignant après les instructions suivantes

```
BEGIN TRANSACTION;
  Insert into Enseignant values
(20,'Key', 'David');
COMMIT;
END TRANSACTION;
BEGIN TRANSACTION;
  Insert into Enseignant values
(21,'Pilou', 'Julie');
ROLLBACK;
END TRANSACTION;
BEGIN TRANSACTION;
  Insert into Enseignant values (21,'Bea', 'Balle');
COMMIT;
END TRANSACTION;
BEGIN TRANSACTION;
Insert into Enseignant values (22, 'Li','Lou');
Insert into Enseignant values (20,'Karl', 'Dave');
COMMIT;
END TRANSACTION;
```

NumEns	Nom	Prenom
1	Voisin	Jean
2	Benzaken	Claudine
3	Forest	Jean

Correction

Interrogation simple en SQL

- Une **requête** (question) simple SQL a la forme
  SELECT *liste de colonnes*
  FROM *liste de tables*
  WHERE *liste de conditions*
- Clause FROM : les **relations utiles** à la requête
- Clause SELECT : les attributs constituant le **schéma** du résultat (* = tous les attributs)
- Clause WHERE : les **conditions** d'extraction des n-uplets

NB : la clause WHERE est optionnelle

Exemple d'interrogation simple en SQL

- Une **requête** (question) simple SQL a la forme
  SELECT *liste de colonnes*
  FROM *liste de tables*
  WHERE *liste de conditions*

SELECT *Prenom*
FROM *Client*
WHERE *Nom='Payen';*

Client

NumSS	Nom	Prenom
17902567	Payen	Olivier
2780289	Payen	Judith
29005579	Quoti	Mathilde

Quels sont les prénoms des clients dont le nom de famille est Payen ?

Prenom
Olivier
Judith

Autre exemple d'interrogation simple

SELECT *NumClient, NumRes*
FROM *reservation*
WHERE *DateRes > '2015-11-19'*

Quels sont les numéros de clients et de réservation dont la réservation a été faite strictement après le 19 Novembre 2007 ?

NumClient	NumRes
29005579	003
17902567	004

Reservation

NumRes	Classe	DateRes	NumClient
001	1	18/11/2015	17902567
002	2	19/11/2015	2780289
003	2	22/11/2015	29005579
004	2	22/11/2015	17902567

Utilisation de plusieurs tables

Reservation

NumRes	Classe	DateRes	NumClient
001	1	18/11/2015	17902567
002	2	19/11/2015	2780289

Client

NumSS	Nom	Prénom
17902567	Payen	Olivier
2780289	Payen	Judith
29005579	Quoti	Mathilde

SELECT *Nom, Prenom, NumRes, DateRes*
FROM *Reservation, Client*
WHERE *DateRes < '2015-11-19'*
AND Reservation. NumClient= Client.NumSS

Quels sont les noms, prénoms des clients, numéros de réservation et date de réservation des réservations qui ont été faites strictement avant le 19 Nov. 2015 ?

Une seule réponse

Nom	Prénom	NumRes	DateRes
Payen	Olivier	001	18/11/2015

Exercice 2

On considère à nouveau la base de données qui décrit des UEs et des enseignants (chaque UE a un enseignant responsable et plusieurs enseignants intervenants)

- **Exprimez les requêtes suivantes en SQL**
  1. *Quel est le nom de l'UE dont le numéro de l'enseignant responsable est 2 ?*
  2. *Quel est le nom et le prénom du responsable de l'UE 'Algorithmique' ?*
  3. *Quels sont les noms des enseignants qui enseignent (et ne sont pas forcément responsables) de l'UE n° 1 ?*

Exercice

La base de données ci-dessous décrit des UEs et des enseignants : chaque UE a un enseignant responsable et plusieurs enseignants intervenants

UE

NumUE	NomUE	Mots_cles	Responsable
1	Base de Données	SQL, tables, modelisation	2
2	Algorithmique	Boucle, tableau, pointeur	3

Enseignant

NumEns	Nom	Prenom
1	Voisin	Jean
2	Benzaken	Claudine
3	Forest	Jean

Enseigne

Prof	UE
2	1
3	1
1	2
3	2

Correction Exercice 2

1. *Quel est le nom de l'UE dont le numéro de l'enseignant responsable est 2 ?*

2. *Quel est le nom et le prénom du responsable de l'UE 'Algorithmique' ?*

3. *Quels sont les noms des enseignants qui enseignent l'UE n°1 ?*

Quelle réponse obtient-on ?

Reservation

NumRes	Classe	DateRes	NumClient
1	1	18/11/2015	17902567
2	2	19/11/2015	2780289

Client

NumSS	Nom	Prenom
17902567	Payen	Olivier
2780289	Payen	Judith
29005579	Quoti	Mathilde

SELECT *Nom, Prenom, NumRes, DateRes*
FROM *Reservation, Client;*

1- Quel schéma ?

Reservation

NumRes	Classe	DateRes	NumClient
1	1	18/11/2015	17902567
2	2	19/11/2015	2780289

Client

NumSS	Nom	Prenom
17902567	Payen	Olivier
2780289	Payen	Judith
29005579	Quoti	Mathilde

SELECT *Nom, Prenom, NumRes, DateRes*
FROM *Reservation, Client;*

Nom	Prenom	NumRes	DateRes

2- Quel Contenu ?

Reservation

NumRes	Classe	DateRes	NumClient
1	1	18/11/2015	17902567
2	2	19/11/2015	2780289

Client

NumSS	Nom	Prénom
17902567	Payen	Olivier
2780289	Payen	Judith
29005579	Quoti	Mathilde

SELECT Nom, Prenom, NumRes, DateRes
FROM Reservation, Client;

Nom	Prenom	NumRes	DateRes
Payen	Olivier	1	18/11/2015
Payen	Judith	1	18/11/2015
Quoti	Mathilde	1	18/11/2015
Payen	Olivier	2	19/11/2015
Payen	Judith	2	19/11/2015
Quoti	Mathilde	2	19/11/2015

2- Quel Contenu ?

Reservation

NumRes	Classe	DateRes	NumClient
1	1	18/11/2015	17902567
2	2	19/11/2015	2780289

Client

NumSS	Nom	Prénom
17902567	Payen	Olivier
2780289	Payen	Judith
29005579	Quoti	Mathilde

SELECT Nom, Prenom, NumRes, DateRes
FROM Reservation, Client;

Nom	Prenom	NumRes	DateRes
Payen	Olivier	1	18/11/2015
Payen	Judith	1	18/11/2015
Quoti	Mathilde	1	18/11/2015
Payen	Olivier	2	19/11/2015
Payen	Judith	2	19/11/2015
Quoti	Mathilde	2	19/11/2015

Comprendre SQL…

- SQL fonde ses requêtes sur des **opérateurs de *l'algèbre relationnelle*** : sélection, projection, jointure, produit cartésien…

- Ce sont aussi ces opérateurs qui sont utilisé par le système pour optimiser les requêtes !

Plans des 2 premières séances

(1) Modélisation d'une base de données en UML

(2) Passage d'un diagramme des classes UML à un schéma de base de données

(3) SQL de base : concevoir une base, la remplir, faire des requêtes simples

(4) À la base d'SQL : l'algèbre relationnelle

(5) SQL avancé

Algèbre relationnelle

- **Opérateurs de base**
  - Sélection (σ), Projection (π), Produit cartésien ($\mathbf{X}$),
  - Différence ($\setminus$ ou $-$) et Union ($\mathbf{U}$)
- **Autres opérateurs**
  - Intersection, jointure, division, renomage
- Puisque chaque opérateur appliqué à une relation renvoie une relation, **les opérateurs peuvent être imbriqués** (*clôture* de l'Algèbre relationnelle)

Projection : introduction

- **Notation** : $\pi_{A_1,\ldots,A_p}(R)$ avec r relation et $A_1,\ldots,A_p$ attributs de R
- **Supprime les attributs** qui sont différents des attributs de la liste de projection $(A_1,\ldots,A_p)$
- Le schéma du résultat contient **exactement les champs** de la liste de projection $(A_1,\ldots,A_p)$
- NB : L'opérateur de projection élimine les redondances (ce n'est pas le cas en SQL)

Projection : exemple

$Q1 = \pi_{NumClient, DateRes}(Reservation)$

Reservation

NumRes	Classe	DateRes	NumClient
001	1	18/11/2007	17902567
002	2	19/11/2007	2780289

Q1

NumClient	DateRes
17902567	18/11/2007
2780289	19/11/2007

*NB : la projection est un opérateur qui **n'agit pas sur les lignes du résultat** mais sur les colonnes (c'est-à-dire sur le **schéma**)*

Projection : Définition formelle

Input

- r une relation de schéma R
- X un ensemble d'attributs tels que $X \subseteq R$

Output

$$\pi_X(r) = \{t(X) / t \in r\}$$

Où t(X) est la restriction de t sur les attributs de X

➔ *Ensemble des n-uplets de r restreints aux attributs de X*

Le schéma de $\pi_X(r)$ est X

Sélection : introduction

- Sélection des n-uplets (lignes) qui satisfont la **condition de sélection**
- Le **schéma** du résultat est **identique** au schéma pris en entrée

Sélection : exemple

$$Q2 = \sigma_{(DateRes<20\text{-}11\text{-}2007 \wedge Classe=2)}(Reservation)$$

ET

Condition de sélection

Réservation

NumRes	Classe	DateRes	NumClient
001	1	18/11/2007	17902567
002	2	19/11/2007	2780289
003	2	20/11/2007	2780289

Q2

NumRes	Classe	DateRes	NumClient
002	2	19/11/2007	2780289

Sélection : Définition formelle

Input

- r une relation de schéma R
- φ une condition de sélection (définie ci-après)

Output

$$\sigma_{\varphi}(r)=\{t / t \in r \text{ et } \varphi(t)\}$$

➔ L'ensemble des n-uplets de la relation r pour lesquels la condition φ est vraie

Le schéma de $\sigma_{\varphi}(r)$ est R

Définition formelle de la condition φ

Définition récursive

- **Base** : Soient 2 attributs A et B (non nécesairement différents), et a ∈ Dom(A)

φ peut prendre les formes suivantes

A =B, A < B, A=a, A < a (on utilisera aussi >, ≤, ≥ et ≠)

- **Si φ_1 et φ_2** sont deux conditions alors

(φ_1), $\neg\varphi_1$ (négation), $\varphi_1 \wedge \varphi_2$ (et) et $\varphi_1 \vee \varphi_2$ (ou) sont des conditions de sélection

Exemples de conditions

- DateRes ≥ 20-11-2007
- Classe = 2
- Nom = 'Payen'
- (DateRes = 20-11-2007 ∨ DateRes = 21-11-2007) ∧ Classe = 2
- NumClient = NumSS

Produit cartésien : Introduction

- Opérateur **binaire** (prend deux relations entre entrée : r1 et r2)
- **Chacun** des n-uplets de r1 est **combiné** avec **chacun** des n-uplets de r2
- Le schéma du résultat *a l'**union des attributs des relations***
- NB : *Si les deux relations ont un attribut de même nom, on renomme cet attribut*

Produit cartésien : Exemple

R1

A	B	C
a1	b1	c1
a2	b2	c2

R2

A	D
a2	d2
a2	d3
a3	d4

R1 x R2 — Renommage

A	B	C	A'	D
a1	b1	c1	a2	d2
a1	b1	c1	a2	d3
a1	b1	c1	a3	d4
a2	b2	c2	a2	d2
a2	b2	c2	a2	d3
a2	b2	c2	a3	d4

Produit cartésien : Définition formelle

Input

– r, s deux relations de schéma R et S

Output

$$r \times s = \{t / t(R) \in r \text{ et } t(S) \in s\}$$

La restriction du produit aux attribut de R est r et celle aux attributs de S est s

r x s a pour schéma R+S

NB : R+S est l'union disjointe de R et S

{R.A / A∈ R } U {S.B / B ∈ S}

SQL *par défaut* fait un produit cartésien !

Reservation

NumRes	Classe	DateRes	NumClient
1	1	**18/11/2007**	17902567
~~2~~	~~2~~	~~19/11/2007~~	~~2780289~~

Client

NumSS	Nom	Prénom
17902567	**Payen**	**Olivier**
2780289	**Payen**	**Judith**
29005579	**Quoti**	**Mathilde**

SELECT Nom, Prenom, Reservation.NumRes, DateRes
FROM Reservation, Client
WHERE DateRes < '2007-11-19'

Nom	Prenom	NumRes	DateRes
Payen	Olivier	1	18/11/2007
Payen	Judith	1	18/11/2007
Quoti	Mathilde	1	18/11/2007

Ces deux lignes mélangent des informations qui n'ont rien à voir !!

Jointure naturelle : Introduction

- Opération binaire **fondamentale (optimisation)** !
- Utilise R1 et R2 qui ont des **attributs communs** (appelons-les X)
- Le **schéma** du résultat est
  - similaire au schéma du produit cartésien
  - modulo que les attributs X n'apparaissent qu'une fois
- Au niveau des lignes : on **combine** les lignes de R1 avec les lignes de R2 qui ont **même valeur pour les attributs X**

Jointure naturelle : Exemple

R1

A	B	C
a1	b1	c1
a2	b2	c2

Équivaut à

$R1 \bowtie R2$

$R1 \bowtie_{R1.A = R2.A} R2$

R2

A	D
a2	d2
a2	d3
a3	d4

A	B	C	D
a2	b2	c2	d2
a2	b2	c2	d3

Jointure naturelle : Définition formelle

- **Input**
  - r et s de schéma R et S
- **Output**

$r \bowtie s = \{t / t(R) \in r \text{ et } t(S) \in s\}$

$r \bowtie s = \pi_{R \text{ union } S} \sigma_{\varphi} (r \times s)$

Avec $R \cap S = \{B1, \ldots, Bp\}$

$\varphi : t(R.B1) = t(S.B1) \ldots t(R.Bp) = t(S.Bp)$

Phi (ou théta) Jointure

- Cas particulier de jointure avec condition

$R1 \bowtie_{\varphi} R2$

φ est une condition de sélection

$R1 \bowtie_{R2.A > R1.C} R2$

R2

A	D
3	2
2	3
1	4

R1

E	B	C
1	7	1
9	2	2

A	D	E	B	C
3	2	1	7	1
3	2	9	2	2
2	3	1	7	1

Union, Intersection, Différence : Introduction

- Tous ces opérateurs prennent en **entrée 2 relations** qui doivent avoir le **même schéma**
- **Union** (R1 ∪ R2) : ensemble de n-uplets qui sont **dans R1 ou dans R2**
- **Intersection** (R1 ∩ R2) ensemble de n-uplets qui sont **dans R1 et dans R2**
- **Différence** (R1 \ R2 ou R1 – R2) : ensemble de n-uplets qui sont **dans R1 mais pas dans R2**

Union, Intersection, Différence : Exemples

R1

A	B
a1	b1
a2	b2
a3	b3

R2

A	B
a1	b1
a2	b3
a3	b4

R1 ∪ R2

A	B
a1	b1
a2	b2
a3	b3
a2	b3
a3	b4

Les n-uplets de R1 union ceux de R2

R1 ∩ R2

A	B
a1	b1

Les n-uplets à la fois dans R1 et dans R2

R1 \ R2

A	B
a2	b2
a3	b3

Les n-uplets de R1 mais qui ne sont pas dans R2

➔ **Pas de doublon**

Union, Intersection, Différence : Définitions formelles

Input

– r, s deux relations de **même schéma R**

Output

$r \cup s = \{ t / t \in r \text{ ou } t \in s \}$

$r \cap s = \{ t / t \in r \text{ et } t \in s \}$

$r \setminus s = \{ t / t \in r \text{ et } t \notin s \}$

de schéma R

Propriétés algébriques

- L'**union** ∪ et l'**intersection** ∩ sont **commutatifs** et **associatifs**
  - r ∪ s = s ∪ r, r ∪ (s1 ∪ s2) = (r ∪ s1) ∪ s2
  - De même pour ∩
- Le **produit cartésien** est **associatif et commutatif***
- La **jointure** est **associative et commutative***
- D'autres propriétés peuvent être énoncées : monotonie de certains opérateurs, fermeture etc.

*si on considère un ordre sur les colonnes

Exemple d'équivalences

- Les relations d'équivalence de l'algèbres sont des égalités entre des formules
- Elles sont à la base des optimisations de requêtes !
- **Exercice**
  - Sachant que les schémas de R et S sont les mêmes et si X ⊆ R, **trouvez un contre exemple aux égalités qui sont fausses et démontrez celles qui sont justes**

- $\pi_X (r \cap s) = \pi_X (r) \cap \pi_X (s)$
- $\pi_X(r - s) = \pi_X (r) - \pi_X (s)$
- $\pi_X(r \cup s) = \pi_X (r) \cup \pi_X (s)$

Exemple d'équivalences (correction)

Exemple d'équivalences (correction)

Exemple d'équivalences (correction)

Exemple d'équivalences (correction)

Suite du TD1

Quelques requêtes en algèbre (Partie 3)

Plans des 2 premières séances

(1) Modélisation d'une base de données en UML

(2) Passage d'un diagramme des classes UML à un schéma de base de données

(3) SQL de base : concevoir une base, la remplir, faire des requêtes simples

(4) À la base d'SQL : l'algèbre relationnelle

(5) SQL avancé

Requêtes simples

- SELECT *
  FROM FILM
  WHERE Acteur='Adjani'

  * : liste de tous les attributs

  $\sigma_{Acteur='Adjani'}$ FILM

- SELECT Titre
  FROM FILM
  WHERE Acteur='Adjani'

  $\pi_{Titre_2}[\sigma_{Acteur='Adjani'} FILM]$

- Sémantique formelle – cas mono-relation :
  SELECT B1 ... Bk
  FROM R
  WHERE C

  $\pi_{B1...Bk}[\sigma_C R]$

Requêtes classiques

- **Renomage** des attributs du schéma cible
  SELECT Titre **AS** Adjani's movies
  FROM FILM
  WHERE Acteur='Adjani'

- **Valeur des attributs = expression arithmétique**
  SELECT Titre, Durée*60 **AS** durée-en-minutes
  FROM FILM-durée

Conditions de la clause SELECT

- **Comparateurs habituels**, arithmétique, concaténation (||), ...
- **Connecteurs : OR, AND et NOT**

```
SELECT Titre
FROM FILM
WHERE Acteur='Adjani' OR réalisateur='Poirier'
```

- Motifs pour la recherche de **chaînes de caractères**

% : une chaîne quelconque

```
SELECT Titre
FROM FILM
WHERE Titre LIKE '%retour%'
```

Le retour du roi, Aliens le retour, …

Chaînes de caractères (suite)

- **Majuscules / Minuscules**
  - Pas de distinction pour les **mots clés**
  - Distinction pour les **valeurs des conditions**
- **Divers**
  - les motifs comme les valeurs sont écrits entre ' '
  - **négation possible** : NOT LIKE
  - **condition d'intervalle** :

att. BETWEEN val1 AND val2

Sémantique formelle – cas multi-relation

```
SELECT B1, ..., Bk
FROM R1 ... Bp
WHERE C
```

$\pi_{B1...Bk}[\sigma_C [R1 \times \ldots \times Bp]]$

Les cinémas qui projettent un film dans lequel Brad Pitt est acteur (pour chaque cinéma donner le titre du film et l'horaire)

```
SELECT Nom-Cine, FILM.Titre, Horaire
FROM FILM , PROG
WHERE FILM.titre=PROG.titre AND Acteur='Brad Pitt'
```

$\pi_{[Nom\text{-}Cine,Titre,Horaire}\ [\sigma_{Acteur='Brad\ Pitt'}\ [FILM\ |X|\ PROG]]$

Introduction de variables

- *Les films avec leur réalisateur et leurs acteurs dans lesquels joue Brad Pitt*

Table FILM(Titre, Acteur, Realisateur)

π Titre, Realisateur, Acteur $[\sigma_{Acteur='Brad\ Pitt'}$ [**FILM 1**]]

- ```
  SELECT F.Titre, F.réalisateur, F.Acteur
  FROM FILM AS F
  WHERE F.Acteur='Brad Pitt'
  ```

Est-ce correct ? Est-ce que cette requête renvoie bien tous les acteurs qui joue dans les film où joue déjà Brad Pitt ?

➔ NON
  ```

Introduction de variables

- *Les films avec leur réalisateur et leurs acteurs dans lesquels joue Brad Pitt*

  Table FILM(Titre, Acteur, Realisateur)

  π Titre, Realisateur, Acteur [σActeur='Brad Pitt' $\wedge$ F1.Titre = F2.Titre [**FILM 1**] X [**FILM 2**]]

- SELECT F2.Titre, F2.réalisateur, F2.Acteur
  FROM FILM **AS F1**, FILM **AS F2** *-- équiv. à FILM F1, FILM F2*
  WHERE F1.Titre = F2.Titre
  AND F1.Acteur='Brad Pitt'

- F1 et F2 sont des **copies virtuelles de FILM**
- F1 sélectionne les films où joue Brad Pitt, F2 est utile pour récupérer tous les autres acteurs de ces films (F1.Titre=F2.Titre)

SQL : Union, Intersection, Différence

- *Les titres des films dans lesquels joue Brad Pitt et qui sont à l'affiche*

  SELECT Titre FROM FILM WHERE Acteur='Brad Pitt'
  INTERSECT
  SELECT Titre FROM PROG

- *Les titres des films qui ne sont pas à l'affiche*

  SELECT Titre FROM FILM
  EXCEPT
  SELECT Titre FROM PROG

- *Toutes les personnes ayant participées au tournage du film "Marion"*

  SELECT Acteur AS Personne FROM FILM
  WHERE Titre = 'Marion'
  UNION
  SELECT réalisateur AS Personne FROM FILM
  WHERE Titre = 'Marion'

SQL : ensembles et multi-ensembles

- {**1**, 2, **1**, 3} est un **multi-ensemble**
- select-from-where (par défaut ALL, multi-ens)
- Union, Except,Intersect (DISTINCT par défaut, ensemble simple)
- **Élimination des dupliqués**

  SELECT **DISTINCT** Titre
  FROM FILM

SQL : Agrégats

- **SUM()** : somme, **AVG()** : moyenne, **MIN()** : minimum, **MAX()** : maximum, **COUNT()** : cardinalité d'un multi-ensemble
- *Le nombre de films dirigés par Bergman*

  SELECT COUNT(Titre)
  FROM PROG
  WHERE réalisateur = 'Bergman'

PB : si un même film de Bergman est plusieurs fois à l'affiche, il est compté plusieurs fois !

- **Éliminer les dupliqués**

  SELECT COUNT (**DISTINCT** Titre)
  FROM PROG
  WHERE réalisateur = 'Bergman'

Groupement

- **Nombre d'acteurs par film**

```
SELECT Titre, COUNT (distinct Acteur)
FROM FILM
GROUP BY Titre
```

- Projection, **regroupement**, calcul de l'agrégat (multi-ensemble)

Film, réalisateur, acteur

T	R	A
t1	r1	a1
t1	r1	a2
t1	r1	a3
t2	r4	a2
t1	r2	a1
t1	r2	a2
t1	r2	a3

T	A
t1	a1
t1	a2
t1	a3
t1	a1
t1	a2
t1	a3
t2	a2

T	
t1	3
t2	1

Groupement - Importance du *Distinct*

```
SELECT Titre, COUNT (Acteur) -- pas de distinct
FROM FILM
GROUP BY Titre
```

- Projection, **regroupement**, calcul de l'agrégat (multi-ensemble)

Film, réalisateur, acteur

T	R	A
t1	r1	a1
t1	r1	a2
t1	r1	a3
t1	r2	a1
t1	r2	a2
t1	r2	a3
t2	r4	a2

T	A
t1	a1
t1	a2
t1	a3
t1	a1
t1	a2
t1	a3
t2	a2

T	
t1	6
t2	1

SQL : Groupement et agrégat

- Ajout d'un schéma Cinephile(NomPers, Nom-Cine)

```
SELECT NomPers, COUNT (DISTINCT Titre)
FROM Cinephile, PROG
WHERE Cinephile.Nom-Cine = PROG.Nom-Cine
GROUP BY NomPers
```

Distinct : Cas où plusieurs salles projettent le même film !

- (1) **jointure naturelle** de Cinephile et PROG, (2) **projection**, (3) **regroupement**, et (4) **calcul de l'agrégat**
  Les personnes et le nombre de films qu'ils peuvent voir
- Clause SELECT en présence d'agrégat

```
SELECT liste1, agg(liste2)
FROM liste-relations WHERE condition
GROUP BY liste1
```


SQL : la clause HAVING

- *Les titre de films et le nombre d'acteurs des films de plus de 3 acteurs*

```
SELECT Titre, COUNT (DISTINCT Acteur)
FROM FILM
GROUP BY Titre
HAVING COUNT(distinct acteur) >= 3
```

- Élimination des **groupes** ne satisfaisant pas la condition
- *Les films dirigés par plus de deux metteurs en scène*

```
SELECT Titre
FROM FILM
GROUP BY Titre
HAVING COUNT(distinct réalisateur) > 2
```

Exercice 2

- Soit le schéma suivant
  - CLIENT (N°C, NomC, AdrC, CP, Ville, Tél, CondPart)
  - PRODUIT (N°P, Description, Prix, QtéP, Ville)
  - COMMANDE (N°Comm, N°C, N°P, QtéC, DateC)
- Répondez aux requêtes suivantes
  - Combien de clients habitent Paris ?
  - Quel est le prix moyen des produits ?
  - Lister toutes les paires de numéros de Clients, tels que ces 2 Clients habitent dans la même ville

Exercice 3

- Considérons le schéma de BD suivant
  - **Appareil**(*aid:* integer, *anom:* string, *autonomie:* integer)
  - **Employes**(*eid:* integer, *enom:* string, *salaire:* integer)
  - **Certifiés**(*eid:* integer, *aid:* integer) *// employés certifiés pour voler sur des appareils*

Exercice 3 (suite)

- Exprimez les requêtes suivantes
- Q1 : Trouvez les noms des pilotes certifiés pour certains appareils Boeing.
- Q2 : Pour chaque pilote certifié sur plus de 3 appareils trouvez son numéro d'employé et l'autonomie maximum de l'appareil pour lequel il est certifié

SQL : Sous-Requêtes et Imbrication

- **Utilisation du résultat d'un Select-From-Where**
  - FILM-DEB(Titre, Acteur) stocke le titre du premier film de chaque acteur.

  Les acteurs du premier film joué par M-F. Pisier ?
- SELECT Acteur
  FROM FILM
  WHERE Titre **=** (SELECT Titre
  FROM FILM-DEB
  WHERE Acteur='M-F. Pisier')
- SELECT FILM.Acteur
  FROM FILM, FILM-DEB
  WHERE FILM.Titre=FILM-DEB.Titre
  AND FILM-DEB.Acteur='M-F. Pisier'

SQL : Sous-requête avec l'opérateur IN

- *Les titres des films dont les réalisateurs sont acteurs (pas forcément dans le même film)*

  SELECT Titre
  FROM FILM
  WHERE réalisateur **IN** (SELECT Acteur FROM FILM)
- *Le résultat de la sous-requête est **un ensemble** de n-uplets*

  SELECT F1.Titre
  FROM FILM AS F1, FILM AS F2
  WHERE F1.réalisateur = F2.acteur

Sous-requêtes coûteuses !

- Soient 2 schémas de relation R(ABC) et S(BCD)
- Deux façons d'exprimer une jointure entre R et S (et proj. s/A)

(1)
```
SELECT A
FROM R
WHERE (R.B, R.C) IN (SELECT B, C
                     FROM S)
```

(2)
```
SELECT A
FROM R AS R1
WHERE EXISTS (SELECT S2.B, S2.C
              FROM S
              WHERE R1.B = S.B
              AND R1.C = S2.C )
```

- Attention : Les sous requêtes sont coûteuses, à éviter lorsque cela est possible

Exercice 4

- Soit le schéma suivant
  - CLIENT (N°C, NomC, AdrC, CP, Ville, Tél, CondPart)
  - PRODUIT (N°P, Description, Prix, QtéP, Ville)
  - COMMANDE (N°Comm, N°C, N°P, QtéC, DateC)
- Répondez à la requête suivante
  - *Quel est le produit le moins cher ?*

En plus… ORDER BY, LIMIT

- On utilise la clause ORDER BY pour trier les résultats d'une requête
- Ordre croissant par défaut (ASC)
- On utilise DESC pour trier par ordre décroissant
- On peut alors utiliser LIMIT n (n un entier fixé) pour extraire les n premiers résultats
- Attention : coût important du tri !

```
SELECT *
FROM FILM
WHERE acteur='Adjani'
ORDER BY titre DESC
LIMIT 20; // 20 films seront renvoyés au plus
```

En plus… SQL : Valeurs nulles

- **Une valeur nulle remplace une valeur d'un attribut**
  - valeur inconnue, attribut inapproprié, valeur incertaine, valeur cachée...
- **Comparaison** avec une valeur nulle
  - vrai=1, faux=0, inconnu=1/2 : logique tri booléene !
  - x AND y =min(x,y) , x OR y =max(x,y), Not x =1-x
- Attention : loi du **tiers exclu n'est plus valide**
  - p OR (NOT p) pour p=1/2

En plus… Jointure externe

- Elle est obtenue en calculant la jointure de R et S puis en y ajoutant
  - **# les n-uplets de R non joingnables** avec un n-uplet de S et complétés avec des **valeurs nulles**
  - **$ les n-uplets de S non joingnables** avec un n-uplet de R et complétés avec des **valeurs nulles**

R

A	B
1	2
3	4

S

B	C
2	5
2	6
7	8

Jointure externe de R et S

A	B	C
1	2	5
1	2	6
3	4	Null
Null	7	8

En plus…Quelques syntaxes *raccourcis* pour les jointures

- **SQL2** propose une variété de formes de jointures : externe, naturelle…
- R **NATURAL JOIN** S correspond exactement à la jointure algébrique (deux colonnes de même nom)
- R **CROSS JOIN** S équivaut à SELECT * FROM R, S
- R **JOIN** S ON R.B = S.B équivaut à
  SELECT * FROM R, S
  WHERE R.B = S.B
- R **OUTER JOIN** S (jointure + $ et #)
- R **RIGHT OUTER JOIN** S (seul $ est effectué)
- R **LEFT OUTER JOIN** S (seul # est effectué)

En plus… Les tables systèmes

- Postgres possède des tables qui contiennent des *meta-données*
- Pg_tables contient la liste des tables
  - SELECT tablename
    FROM pg_tables
    where schemaname='public';

TPs BD

- Page du cours : https://www.lri.fr/~cohen/BD.html
- Se logger sous Postgres : Ligne de commande
  - psql -h tp-postgres -U *login_a*
- Ou avec interface graphique
  - http://tp-sgbd.ups.u-psud.fr/phpPgAdmin
  - Identification adonis requise **puis**
  - Login: votre login unix **court** en concaténant **_a** à la fin du login
  - Mdp = login (=logincourtunix_a)
- Créer un schéma (par exemple TPArticles) dans **votre BD** et y lancer les scripts
  - En ligne de commandes (CREATE SCHEMA mon_schema; puis lancez SET search_path TO mon_schema;)
- Dans l'interface graphique : toujours vérifier où vous travaillez (fenêtre SQL)

Commandes utiles (ligne de commandes)…

- \h pour l'aide-mémoire des commandes SQL
- \? pour l'aide-mémoire des commandes psql
- \q pour quitter
- \dn pour visualiser l'ensemble des schémas de votre base de données
- \dt <nom_schema>.* pour visualiser les tables que contient un schéma
- \d <nom_schema>.<nom_table> pour visualiser les attributs de la table d'un schéma
- \i <chemin>/script.sql pour exécuter un script SQL