Bases de données relationnelles

Modèle, languages, optimisation

v1.9

B. Amann, C. Crochepeyre, M. Crucianu, M. Ferecatu,
D. Gross-Amblard, P. Rigaux, V. Thion, N. Travers, D. Vodislav,
V. Sans, M. Scholl

Plan du cours

Plan du cours

Objectif du cours

COMPRENDRE et MAÎTRISER
la technologie des
BASES DE DONNÉES RELATIONNELLES

Bibliographie

Ouvrages en français

1. Carrez C., *Des Structures aux Bases de Données*, Masson
2. Gardarin G., *Maîtriser les Bases de Données: modèles et langages*, Eyrolles
3. Date C.J, *Introduction aux Bases de Données*, Vuibert, 970 Pages, Janvier 2001
4. Akoka J. et Comyn-Wattiau I., *Conception des Bases de Données Relationnelles*, Vuibert Informatique
5. Rigaux P., *Cours de Bases de Données*, *http://dept25.cnam.fr/BDA/DOC/cbd.pdf*

Bibliographie

Ouvrages en anglais

1. R. Ramakrishnan et J. Gehrke, *DATABASE MANAGEMENT SYSTEMS*, MacGraw Hill
2. R. Elmasri, S.B. Navathe, *Fundamentals of database systems*, 3e édition, 1007 pages, 2000, Addison Wesley
3. Ullman J.D. and Widom J. *A First Course in Database Systems*, Prentice Hall, 1997
4. H. Garcia - Molina, J.D. Ullman, J. Widom, *Database Systems : The Complete Book*, Hardcover, 2002
5. Garcia-Molina H., Ullman J. and Widom J., *Implementation of Database Systems*, Prentice Hall, 1999
6. Ullman J.D., *Principles of Database and Knowledge-Base Systems*, 2 volumes, Computer Science Press
7. Abiteboul S., Hull R., Vianu V., *Foundations of Databases*,Addison-Wesley

Webographie

1. Serge Abiteboul, Cours du collège de France
   `http://www.college-de-france.fr/site/serge-abiteboul/`
2. Jennifer Widom, Mooc de Stanford
   `https://www.db-class.org/db/Winter2013/preview/`

Bibliographie

Le standard SQL

1. Date C.J., *A Guide to the SQL Standard*, Addison-Wesley

Quelques systèmes

1. DB2 (IBM),
2. Oracle (actuellement 11g),
3. SQL Server (Microsoft),
4. PostgreSQL,
5. MySQL.

Bibliographie

SQL “à la maison”

1. MySQL, `http://www.mysql.org` (MS Windows, Linux)
   - Installation et interface Web via EasyPhp, `http://www.easyphp.org/`
   - Administration via MySQL Workbench, `http://dev.mysql.com/doc/workbench/en/`
2. PostgreSQL, `http://www.postgresql.org` (MS Windows, Linux)
   - Interface Web via PhpPgAdmin, `http://phppgadmin.sourceforge.net/`
   - Administration via PgAdmin, `http://www.pgadmin.org/`

Applications des bases de données

1. **Applications “classiques” :**
   - Gestion de données : salaires, stocks, ...
   - Transactionnel : comptes bancaires, centrales d'achat, réservations
2. **Applications “modernes” :**
   - Documents électroniques : bibliothèques, journaux
   - Web : commerce électronique, serveurs Web
   - Logiciel : persistance de données (*Hibernate*)
   - Génie logiciel : gestion de programmes, manuels, ...
   - Documentation technique : plans, dessins, ...
   - Bases de données spatiales : cartes routières, systèmes de guidage GPS, ...
   - Données embarquées : SQLite dans votre mobile

Problème central : comment stocker et manipuler les données?

Une base de données est

- un grand ensemble de données
- structurées et
- mémorisées sur un support permanent

Un système de gestion de bases de données (SGBD) est

- un logiciel de haut niveau d'abstraction qui permet de manipuler ces informations

Diversité → Complexité

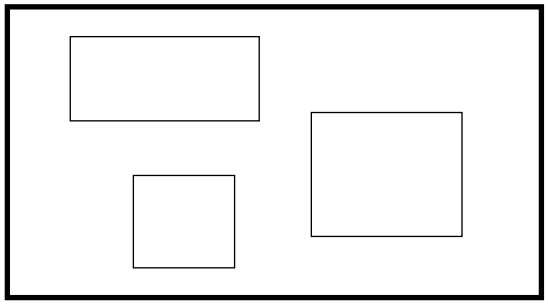
Diversité des utilisateurs, des interfaces et des architectures :

1. diversité des utilisateurs : administrateurs, programmeurs, non informaticiens, ...
2. diversité des interfaces : langages BD, ETL, menus, saisies, rapports, ...
3. diversité des architectures : client-serveur centralisé/distribué
   Aujourd'hui : accès à plusieurs bases hétérogènes accessibles par réseau, bases NoSQL déployées en nuage

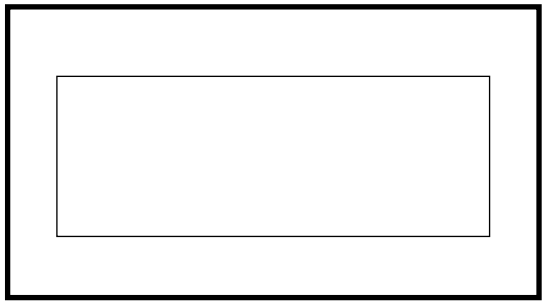
Architecture d'un SGBD : ANSI-SPARC (1975)

NIVEAU EXTERNE

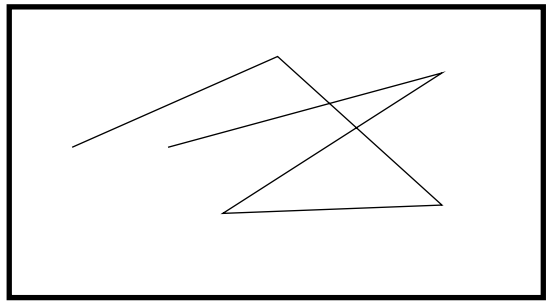
vue 1



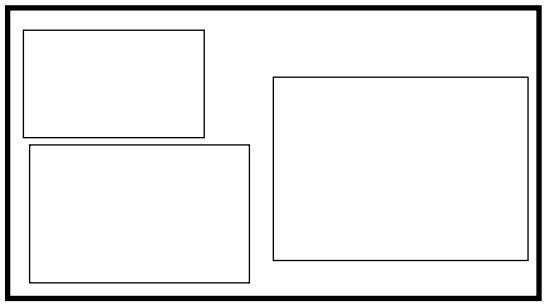
vue 2



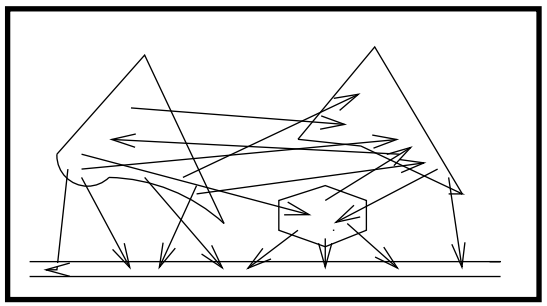
vue 3



NIVEAU LOGIQUE



NIVEAU PHYSIQUE



Architecture d'un SGBD

Chaque niveau du SGBD réalise un certain nombre de fonctions :
NIVEAU PHYSIQUE

- Accès aux données, gestion sur mémoire secondaire (fichiers) des données, des index
- Partage de données et gestion de la concurrence d'accès
- Reprise sur pannes (fiabilité)
- Distribution des données et interopérabilité (accès aux réseaux)

Architecture d'un SGBD

NIVEAU LOGIQUE

- Définition de la structure des données :
  Langage de Description de Données (LDD)
- Consultation et mise à jour des données :
  Langages de Requêtes (LR) et
  Langage de Manipulation de Données (LMD)

Architecture d'un SGBD

NIVEAU EXTERNE : Vues utilisateurs

Exemple : base(s) de données de gestion des étudiants :

1. **Vue de la planification des salles :** pour chaque cours
   - Noms des enseignants
   - Horaires et salles
2. **Vue de la paye :** pour chaque enseignant
   - Nom, prénom, adresse, indice, nombre d'heures
3. **Vue du service de scolarité :** pour chaque élève
   - Nom, prénom, adresse, no d'immatriculation, inscriptions aux cours, résultats

Intégration de ces vues

1. On laisse chaque usager avec sa vision du monde
2. Passage du niveau externe au niveau logique

On “intègre” l'ensemble de ces vues en une description **unique** :

SCHÉMA LOGIQUE

Les avantages de cette abstraction

1. **SIMPLICITÉ**
   Les structures et les langages sont plus simples (“logiques”, déclaratifs), donc plus faciles pour l'usager non expert
2. **INDÉPENDANCE PHYSIQUE**
   On peut modifier l'implantation/la représentation physique sans modifier les programmes/l'application
3. **INDÉPENDANCE LOGIQUE**
   On peut modifier les programmes/l'application sans toucher à la représentation physique des données

Autour du SGBD

- Outils d'aide à la conception de schémas
- Outils de saisie et d'impression
- Outils d'extraction / importation (ETL)
- Interfaces d'interrogation (textuel / graphique)
- Fouille de données
- Visualisation de données
- Environnement de programmation : intégration des langages SGBD (LDD, LR, LMD) avec un langage de programmation (C++, Java, Php, Cobol, ...)
- API standards : ODBC, JDBC
- Importation/exportation de données (ex. documents XML)
- Passerelles (réseaux) vers d'autres SGBD

En résumé

Un système de gestion de bases de données (SGBD) est un logiciel permettant de gérer un grand volume de données

- structurées,
- persistantes (stockées sur disque),
- cohérentes,
- fiables (protégées contres les pannes) et
- partagées entre utilisateurs et applications
- indépendamment de leur organisation physique

Historique des modèles SGBD

À chaque génération correspond un modèle logique

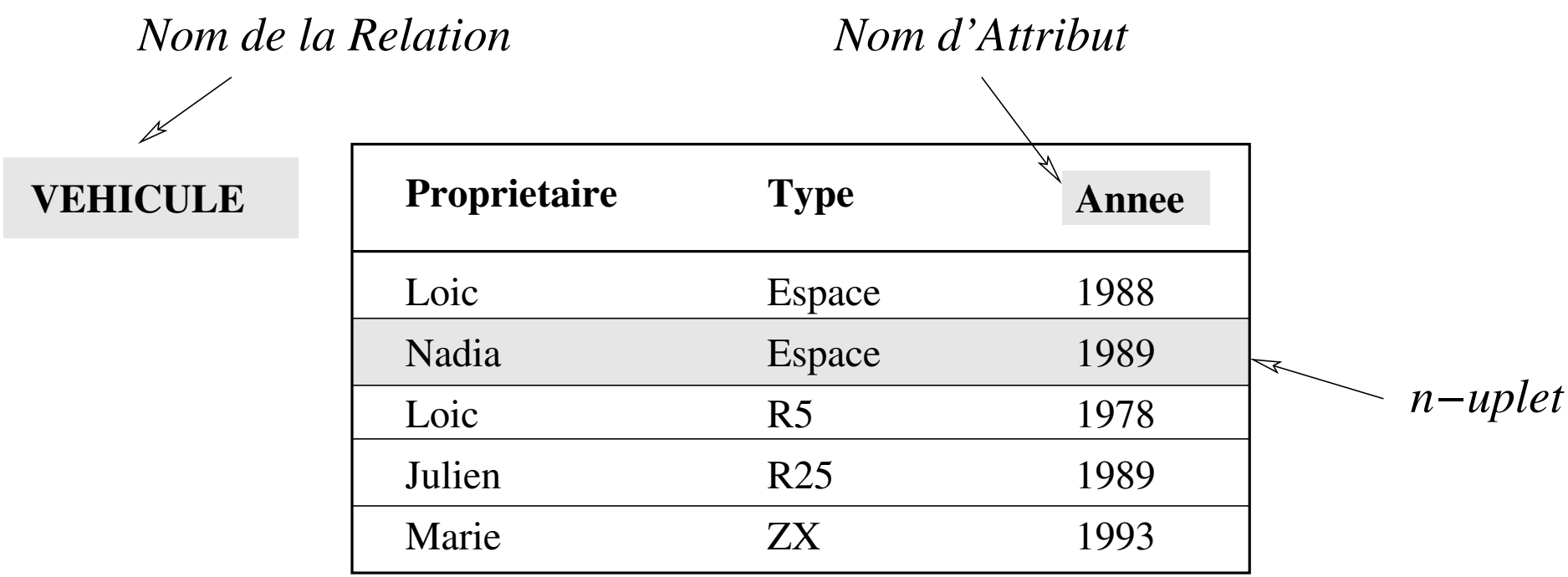
< 60	S.G.F. (e.g. COBOL)	
mi-60	HIÉRARCHIQUE IMS (IBM)	navigationnel
	RÉSEAU (CODASYL)	navigationnel
73-80	RELATIONNEL	déclaratif
mi-80	RELATIONNEL	explosion sur micro
Fin 80	ORIENTÉ-OBJET	navig. + déclaratif
	RELATIONNEL ETENDU	nouvelles applications
	DATALOG (SGBD déductifs)	pas encore de marché
Fin 90	XML	navig. + déclaratif
Fin 90	NoSQL	peu de structure mais Big Data

Les acteurs du SGBD

- **Le concepteur**
  - évalue les besoins de l'application
  - conçoit le schéma logique de la base
- **L'administrateur du SGBD**
  - installe le système et créé la base
  - conçoit le schéma physique
  - fait des réglages fins (*tuning*)
  - gère avec le concepteur l'évolution de la base (nouveaux besoins, utilisateurs)
- **L'éditeur du SGBD**
  - fournit le système et les outils
- **Les utilisateurs**
  - Ajoutent des données
  - Posent des requêtes

Plan du cours

Exemple de relation



FOURNISSEURS

FNOM	FADRESSE
Abounayan	92190 Meudon
Cima	75010 Paris
Preblocs	92230 Gennevilliers
Sarnaco	75116 Paris

FOURNITURES

FNOM	PNOM	PRIX
Abounayan	sable	300
Abounayan	briques	1500
Preblocs	parpaing	1200
Sarnaco	parpaing	NULL
Sarnaco	ciment	125

CLIENTS

NOM	CADRESSE	BALANCE
Jean	75006 Paris	-12000
Paul	75003 Paris	0
Vincent	94200 Ivry	3000
Pierre	92400 Courbevoie	7000

COMMANDES

NUM_COMDE	NOM	PNOM	QUANTITE
1	Jean	briques	5
2	Jean	ciment	10
3	Paul	briques	3
4	Paul	parpaing	9
5	Vincent	parpaing	7

Plan du cours

Domaines, n-uplets et relations

- Un domaine est un *ensemble de valeurs*.
  Exemples : $\{0, 1\}$, $\mathbb{N}$, l'ensemble des chaînes de caractères, l'ensemble des chaînes de caractères de longueur 10.
- Un n-uplet est une *liste de valeurs* $[v_1, \ldots, v_n]$ où chaque valeur v_i est la valeur d'un domaine D_i : $v_i \in D_i$
- Le produit cartésien $D_1 \times \ldots \times D_n$ entre des domaines $D_1, \ldots, D_n$ est *l'ensemble de* **tous** *les n-uplets* $[v_1, \ldots, v_n]$ où $v_i \in D_i$.
- Une relation R est *un sous-ensemble fini* d'un produit cartésien $D_1 \times \ldots \times D_n$: R est un ensemble de n-uplets.
- Une base de données est un *ensemble de relations*.

Attributs

Une relation $R \subset D_1 \times \ldots \times D_n$ est représentée sous forme d'une table où chaque ligne correspond à un élément de l'ensemble R (un n-uplet) :

- L'ordre des lignes n'a pas d'importance (ensemble).
- Les colonnes sont distinguées par leur ordre ou par un nom d'attribut. Soit A_i le i-ème attribut de R :
  - n est appelé l'arité de la relation R.
  - D_i est appelé le domaine de A_i.
  - Tous les attributs d'une relation ont un nom différent.
  - Un même nom d'attribut peut apparaître dans différentes relations.
  - Plusieurs attributs de la même relation peuvent avoir le même domaine.

Schéma d'une base de données

- Le schéma d'une relation R est défini par le nom de la relation et la liste des attributs avec pour chaque attribut son domaine :

$$R(A_1 : D_1, \ldots, A_n : D_n)$$

ou, plus simplement :

$$R(A_1, \ldots, A_n)$$

Exemple : VEHICULE(NOM:CHAR(20), TYPE:CHAR(10), ANNEE:ENTIER)

- Le schéma d'une base de données est l'ensemble des schémas de ses relations.

Exemple de base de données

SCHÉMA :

- FOURNISSEURS(FNOM:CHAR(20), FADRESSE:CHAR(30))
- FOURNITURES(FNOM:CHAR(20), PNOM:CHAR(10), PRIX:ENTIER))
- COMMANDES(NUM_COMDE:ENTIER, NOM:CHAR(20), PNOM:CHAR(10), QUANTITE:ENTIER))
- CLIENTS(NOM: CHAR(20), CADRESSE:CHAR(30), BALANCE:RELATIF)

FOURNISSEURS

FNOM	FADRESSE
Abounayan	92190 Meudon
Cima	75010 Paris
Preblocs	92230 Gennevilliers
Sarnaco	75116 Paris

FOURNITURES

FNOM	PNOM	PRIX
Abounayan	sable	300
Abounayan	briques	1500
Preblocs	parpaing	1200
Sarnaco	parpaing	NULL
Sarnaco	ciment	125

Valeurs spéciale NULL

- Fréquemment : on ne connaît pas toutes les valeurs demandées
- En pratique, valeur spéciale NULL : «valeur inconnue»
- Valeur spéciale disponible dans chaque domaine
- (on verra plus tard son fonctionnement détaillé)

Attributs identifiants, clés, clé primaire

1. La valeur d'un attribut ou ensemble d'attributs peut servir à identifier chaque *n*-uplet

Exemple

- Ex. valeur de FNOM dans FOURNISSEURS
- Mais pas valeur FNOM dans FOURNITURES

2. On dit qu'un ensemble d'attributs est clé pour une relation s'il ne peut y avoir deux *n*-uplets ayant les mêmes valeurs pour ces attributs.
3. Il peut exister plusieurs clés pour une relation
4. La clé choisie en pratique pour jouer le rôle d'indentifiant : clé primaire

Exemple

- Choisir FNOM comme clé primaire de FOURNISSEURS

Référencement entre relations, clés étrangères

1. Des valeurs d'attributs peuvent servir à désigner d'autres attributs dans d'autres relations

Exemple

- Ex. valeur de FNOM dans FOURNITURES fait référence au n-uplet identifié par la valeur de FNOM dans FOURNISSEURS

2. On dit qu'un ensemble d'attributs est clé étrangère pour une relation s'ils sont clé primaire dans une autre relation
3. Il peut exister plusieurs clés étrangères dans une relation

Plan du cours

Opérations sur une base de données relationnelle

- Langage de définition des données (LDD) (définition et MAJ du schéma) :
  - création et destruction d'une relation ou d'une base
  - ajout, suppression d'un attribut
  - définition des contraintes (clés, références, ...)
- Langage de manipulation des données (LMD)
  - saisie des n-uplets d'une relation
  - affichage d'une relation
  - modification d'une relation : insertion, suppression et maj des n-uplets

Création de tables

Une table (relation) est créée avec la commande **CREATE TABLE** :

```
CREATE TABLE Produit (pnom VARCHAR(20),
                      prix INTEGER);

CREATE TABLE Fournisseur(fnom VARCHAR(20),
                         ville VARCHAR(16));
```

- Pour chaque attribut, on indique le domaine (type)

Nombreux types : exemple d'Oracle 8i

`decimal(p, s)`	<p digits>,<s digits>
`integer`	nombre entier
`real`	nombre réel
`char (size)`	chaîne de caractère de taille fixe
`varchar (size)`	chaîne de caractère de taille variable
`date,timestamp`	horodatage
`boolean`	booléen
`blob`	données binaires de grande taille
et aussi	image, son, vidéo, objets 3D, ...

Contraintes d'intégrité

Pour une application donnée, pour un schéma relationnel donné, toutes les instances ne sont pas significatives

Exemple

- Champs important non renseigné
- Prix négatifs
- Code de produit dans une commande ne correspondant à aucun produit dans le catalogue

Contraintes d'intégrité

Pour une application donnée, pour un schéma relationnel donné, toutes les instances ne sont pas significatives

Exemple

- Champs important non renseigné : autorisation des `NULL`
- Prix négatifs : contrainte d'intégrité sémantique
- Code de produit dans une commande ne correspondant à aucun produit dans le catalogue : contrainte d'intégrité référentielle

Valeurs NULL

La valeur NULL peut être interdite :

```
CREATE TABLE Fourniture (pnom VARCHAR(20) NOT NULL,
                         fnom VARCHAR(20) NOT NULL
)
```

Unicité des valeurs

Interdiction de deux valeurs identiques pour le même attribut :

```
CREATE TABLE Fourniture (pnom VARCHAR(20) UNIQUE,
                         fnom VARCHAR(20)
)
```

- `UNIQUE` et `NOT NULL` : l'attribut peut servir de clé primaire

Ajout de contraintes référentielles : clés primaires

```
CREATE TABLE Produit (pnom VARCHAR(20),
                      prix INTEGER,
                      PRIMARY KEY (pnom));

CREATE TABLE Fournisseur(fnom VARCHAR(20) PRIMARY KEY,
                         ville VARCHAR(16));
```

- L'attribut `pnom` est une clé dans la table `Produit`
- L'attribut `fnom` est une clé dans la table `Fournisseur`
- Une seule clé primaire par relation
- Une clé primaire peut être référencée par une autre relation

Ajout de contraintes référentielles : clés étrangères

La table Fourniture relie les produits à leurs fournisseurs :

```
CREATE TABLE Fourniture (pnom VARCHAR(20) NOT NULL,
                         fnom VARCHAR(20) NOT NULL,
        FOREIGN KEY (pnom) REFERENCES Produit,
        FOREIGN KEY (fnom) REFERENCES Fournisseur);
```

- Les attributs pnom et fnom sont des clés étrangères (pnom et fnom existent dans les tables référencées)
- Pour sélectionner un attribut de nom différent :

```
FOREIGN KEY (pnom) REFERENCES Produit(autrenom)
```

Valeurs par défaut

```
CREATE TABLE Fournisseur(fnom VARCHAR(20),
                         ville VARCHAR(16) DEFAULT 'Carcassonne');
```

- Valeur utilisée lorsque l'attribut n'est pas renseigné
- Sans précision, la valeur par défaut est `NULL`

Contraintes sémantiques

- Clause `CHECK`, suivie d'une condition

Exemple : prix positifs

```
prix INTEGER CHECK (prix>0)
```

- Condition générale : requête booléenne (dépend du SGBD)

Destruction de tables

On détruit une table avec la commande **DROP TABLE** :

```
DROP TABLE Fourniture;
DROP TABLE Produit;
DROP TABLE Fournisseur;
```

La table `Fourniture` doit être détruite en premier car elle contient des clés étrangères vers les deux autres tables;

Insertion de n-uplets

On insère dans une table avec la commande **INSERT** :

$$\textbf{INSERT INTO } R(A_1, A_2, \ldots, A_n) \textbf{ VALUES } (v_1, v_2, \ldots v_n)$$

Donc on donne deux listes : celles des attributs (les A_i) de la table et celle des valeurs respectives de chaque attribut (les v_i).

1. Bien entendu, chaque A_i doit être un attribut de R
2. Les attributs non-indiqués restent à **NULL** ou à leur valeur par défaut.
3. On doit toujours indiquer une valeur pour un attribut déclaré **NOT NULL**

Insertion : exemples

Insertion d'une ligne dans *Produit* :

INSERT INTO *Produit (pnom, prix)*
VALUES (*'Ojax', 15*)

Insertion de deux fournisseurs :

INSERT INTO *Fournisseur (fnom, ville)*
VALUES (*'BHV', 'Paris'*), (*'Casto', 'Paris'*)

Il est possible d'insérer plusieurs lignes en utilisant une requête

INSERT INTO *NomsProd (pnom)*
<requete>

Modification

On modifie une table avec la commande **UPDATE** :

UPDATE R **SET** $A_1 = v_1, A_2 = v_2, \ldots, A_n = v_n$
WHERE *condition*

Contrairement à **INSERT**, **UPDATE** s'applique à un ensemble de lignes.

1. On énumère les attributs que l'on veut modifier.
2. On indique à chaque fois la nouvelle valeur.
3. La clause **WHERE** *condition* permet de spécifier les lignes auxquelles s'applique la mise à jour.

Bien entendu, on ne peut pas violer les contraintes sur la table.

Modification : exemples

Mise à jour du prix d'Ojax :

UPDATE *Produit* **SET** *prix=17*
WHERE *pnom = 'Ojax'*

Destruction

On détruit une ou plusieurs lignes dans une table avec la commande **DELETE** :

DELETE FROM *R*
WHERE *condition*

C'est la plus simple des commandes de mise-à-jour puisque elle s'applique à des lignes et pas à des attributs.

Destruction : exemples

Destruction du fournisseur BHV :

DELETE FROM *Fournisseur*
WHERE *fnom* = *'BHV'*

Déclencheurs associés aux destructions de *n*-uplets

- Que faire lorsque le *n*-uplet référence une autre table ?

```
CREATE TABLE Produit (pnom VARCHAR(20),
                      prix INTEGER,
                      PRIMARY KEY (pnom));

CREATE TABLE Fourniture (pnom VARCHAR(20) NOT NULL,
                         fnom VARCHAR(20) NOT NULL,
       FOREIGN KEY (pnom) REFERENCES Produit
       on delete <action>);
```

<action> à effectuer lors de la destruction dans Produit :

- CASCADE : destruction si destruction dans Produit
- RESTRICT : interdiction si existe dans Fourniture
- SET NULL : remplacer par NULL
- SET DEFAULT <valeur> : remplacement par une valeur par défaut

Déclencheurs associés aux mise à jour de *n*-uplets

```
CREATE TABLE Fourniture (pnom VARCHAR(20) NOT NULL,
fnom VARCHAR(20) NOT NULL,
FOREIGN KEY (pnom) REFERENCES Produit
on update <action>);
```

<action> à effectuer lors d'un changement de clé dans Produit :

- CASCADE : propagation du changement de clé de Produit
- RESTRICT : interdiction si clé utilisée dans Fourniture
- SET NULL : remplace la clé dans Fourniture par NULL
- SET DEFAULT <valeur> : remplace la clé par une valeur par défaut

Objectifs

- Interroger les données
- De façon flexible
- De façon efficace
- Sans refaire un programme à chaque fois
- $\rightarrow$ langage de requête riche
- $\rightarrow$ approche déclarative plutôt que procédurale

Langages de requêtes relationnels

Pouvoir d'expression : Qu'est-ce qu'on peut calculer ? Quelles opérations peut-on faire ?

Fondements théoriques :

- calcul relationnel : $\{p : \exists n \exists d \ \ ACTEUR(n, p, d)\}$
  - logique du premier ordre, très étudiée (théorèmes)
  - langage déclaratif : on indique les propriétés que doivent vérifier les réponses à la requête
  - on n'indique pas comment les trouver
  - facile pour un utilisateur expert (logicien . . .)
- algèbre relationnelle : $\Pi_p ACTEUR$
  - langage procédural, évaluateur facile à programmer
  - on indique comment trouver le résultat
  - difficile pour un utilisateur

Langages de requêtes relationnels

- Théorème de Codd[1] : ces deux langages ont le même pouvoir d'expression
- Edgar Frank Codd (1923-2003), mathématicien anglais, inventeur des SGBD relationels (chez IBM), Prix Turing

[1] A Relational Model of Data for Large Shared Data Banks, CACM, 1970

Langages de requêtes relationnels

En pratique, langage SQL : `select PRENOM from ACTEUR`

- Langage déclaratif
- Plus naturel que logique du premier ordre
  - facile pour tout utilisateur
- Traduction automatique en algèbre relationnelle
- Évaluation de la requête à partir de l'algèbre
  - évaluation facile à programmer

Algèbre relationnelle

Opérations “relationnelles” (ensemblistes) :

- une opération prend en entrée une ou deux relations (ensembles de n-uplets) de la base de données
- le résultat est toujours une relation (un ensemble)

5 opérations de base (pour exprimer toutes les requêtes) :

- opérations unaires : sélection, projection
- opérations binaires : union, différence, produit cartésien

Autres opérations qui s'expriment en fonction des 5 opérations de base : jointure, intersection et division

Plan du cours

Projection

Projection sur une partie (un sous-ensemble) des attributs d'une relation R.
Notation :

$$\pi_{A_1,A_2,\ldots,A_k}(R)$$

$A_1, A_2, \ldots, A_k$ sont des attributs (du schéma) de la relation R. La projection "élimine" tous les autres attributs (colonnes) de R.

Projection: Exemples

a) On élimine la colonne C dans la relation R :

R	A	B	C
→	a	b	c
	d	a	b
	c	b	d
→	a	b	e
	e	e	a

$\Rightarrow$ $\pi_{A,B}(R)$

A	B
a	b
d	a
c	b
e	e

Le résultat est une relation (un ensemble) : le n-uplet (a, b) n'apparaît qu'**une seule** fois dans la relation $\pi_{A,B}(R)$, bien qu'il existe **deux** n-uplets (a, b, c) et (a, b, e) dans R.

Projection: Exemples

b) On élimine la colonne B dans la relation R (on garde A et C) :

R

A	B	C
a	b	c
d	a	b
c	b	d
a	b	e
e	e	a

$\Rightarrow$

$\pi_{A,C}(R)$

A	C
a	c
d	b
c	d
a	e
e	a

Plan du cours

Sélection

Sélection avec une condition $\mathcal{C}$ sur les attributs d'une relation R: on garde les n-uplets de R dont les attributs satisfont $\mathcal{C}$.
NOTATION :

$$\sigma_C(R)$$

Sélection : exemples

a) On sélectionne les n-uplets dans la relation R tels que l'attribut B vaut "b" :

R

A	B	C
a	b	1
d	a	2
c	b	3
a	b	4
e	e	5

$\Rightarrow$ $\sigma_{B=''b''}(R)$

A	B	C
a	b	1
c	b	3
a	b	4

Sélection : exemples

b) On sélectionne les n-uplets tels que

$$(A ='' a'' \vee B ='' a'') \wedge C \leq 3 :$$

R

A	B	C
a	b	1
d	a	2
c	b	3
a	b	4
e	e	5

$\Rightarrow$ $\sigma_{(A=''a'' \vee B=''a'') \wedge C \leq 3}(R)$

A	B	C
a	b	1
d	a	2

Sélection : exemples

c) On sélectionne les n-uplets tels que la 1re et la 2e colonne sont identiques :

R

A	B	C
a	b	1
d	a	2
c	b	3
a	b	4
e	e	5

$\Rightarrow \quad \sigma_{A=B}(R)$

A	B	C
e	e	5

Condition de sélection

La condition $\mathcal{C}$ d'une sélection $\sigma_C(R)$ est une **formule logique** qui relie des termes de la forme $A_i \theta A_j$ ou $A_i \theta a$ avec les connecteurs logiques **et** $(\wedge)$ et **ou** $(\vee)$ où

- A_i et A_j sont des attributs de la relation R,
- a est un élément (une valeur) du domaine de A_i,
- θ est un prédicat de comparaison $(=, <, \leq, >, \geq, \neq)$.

Plan du cours

Expressions (requêtes) de l'algèbre relationnelle

Fermeture :

- Le résultat d'une opération est à nouveau une **relation**
- Sur cette relation, on peut faire une **autre opération** de l'algèbre

$\Rightarrow$ *Les opérations peuvent être* composées *pour former des expressions plus complexes de l'algèbre relationnelle.*

Expressions de l'algèbre relationnelle

Exemple : $COMMANDES(NOM, PNOM, NUM, QTE)$

$$R'' = \pi_{PNOM}(\overbrace{\sigma_{NOM=''Jean''}(COMMANDES)}^{R'})$$

La relation $R'(NOM, PNOM, NUM, QTE)$ contient les n-uplets dont l'attribut NOM a la valeur $''Jean''$. La relation $R''(PNOM)$ contient tous les produits commandés par Jean.

Plan du cours

Produit cartésien

- NOTATION : $R \times S$
- ARGUMENTS : 2 relations quelconques :

$$R(A_1, A_2, \ldots, A_n) \quad S(B_1, B_2, \ldots, B_k)$$

- SCHÉMA DE $T = R \times S$: $T(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_k)$.
  On introduit les règles de renommage suivantes pour lever les éventuelles ambiguités sur le schéma de T:
  Si le produit cartésien est le produit d'une relation avec elle-même alors le nom de la relation est numéroté pour indentifier les deux rôles (par 1 et 2).
  Si les relations ont des attributs en commum, les noms des attributs en commun sont prefixés par le nom de la relation d'origine.
- VALEUR DE $T = R \times S$: ensemble de tous les n-uplets ayant $n + k$ composants (attributs)
  - dont les n premiers composants forment un n-uplet de R
  - et les k derniers composants forment un n-uplet de S

Exemple de produit cartésien

R

A	B
1	1
1	2
3	4

$|R|$

S

C	D	E
a	b	a
a	b	c
b	a	a

$|S|$

$\Rightarrow$

R $\times$ **S**

A	B	C	D	E
1	1	a	b	a
1	1	a	b	c
1	1	b	a	a
1	2	a	b	a
1	2	a	b	c
1	2	b	a	a
3	4	a	b	a
3	4	a	b	c
3	4	b	a	a

$|R| \times |S|$

Plan du cours

Jointure naturelle

- NOTATION : $R \bowtie S$
- ARGUMENTS : 2 relations quelconques :

$$R(A_1, \ldots, A_m, X_1, \ldots, X_k) \quad S(B_1, \ldots, B_n, X_1, \ldots, X_k)$$

où $X_1, \ldots, X_k$ sont les attributs en commun.

- SCHÉMA DE $T = R \bowtie S$: $T(A_1, \ldots, A_m, B_1, \ldots, B_n, X_1, \ldots, X_k)$
- VALEUR DE $T = R \bowtie S$: ensemble de tous les n-uplets ayant $m + n + k$ attributs dont les m premiers et k derniers composants forment un n-uplet de R et les $n + k$ derniers composants forment un n-uplet de S.

Jointure naturelle: exemple

R

A	B	C
a	b	c
d	b	c
b	b	f
c	a	d

S

B	C	D
b	c	d
b	c	e
a	d	b
a	c	c

$\Rightarrow$

R $\bowtie$ S

A	B	C	D
a	b	c	d
a	b	c	e
d	b	c	d
d	b	c	e
c	a	d	b

Jointure naturelle

Soit $U = \{A_1, \ldots, A_m, B_1, \ldots, B_n, X_1, \ldots, X_k\}$ l'ensemble des attributs des 2 relations et $V = \{X_1, \ldots, X_k\}$ l'ensemble des attributs en commun.

$$R \bowtie S = \pi_U\big(\sigma_{\forall X \in V : R.X = S.X}\big(R \times S\big)\big)$$

NOTATION : $R.X$ signifie "l'attribut X de la relation R".

Jointure naturelle : exemple

R

A	B
1	a
1	b
4	a

S

A	B	D
1	a	b
2	c	b
4	a	a

$\Rightarrow$

R × S

	R.A	R.B	S.A	S.B	D
	1	a	1	a	b
$R.A \neq S.A \wedge R.B \neq S.B \rightarrow$	**1**	**a**	**2**	**c**	b
$R.A \neq S.A \rightarrow$	**1**	**a**	**4**	**a**	a
$R.B \neq S.B \rightarrow$	**1**	**b**	**1**	**a**	b
$R.A \neq S.A \wedge R.B \neq S.B \rightarrow$	**1**	**b**	**2**	**c**	b
$R.A \neq S.A \wedge R.B \neq S.B \rightarrow$	**1**	**b**	**4**	**a**	a
$R.A \neq S.A \rightarrow$	**4**	**a**	**1**	**a**	b
$R.A \neq S.A \wedge R.B \neq S.B \rightarrow$	**4**	**a**	**2**	**c**	b
	4	a	4	a	a

$\Downarrow$

Jointure naturelle : exemple

$$\pi_{R.A,R.B,D}\left(\sigma_{R.A=S.A \wedge R.B=S.B}\left(R \times S\right)\right)$$

$\Rightarrow$

R ⋈ S

A	B	D
1	a	b
4	a	a

Jointure naturelle : algorithme de calcul

Pour chaque n-uplet a dans R et pour chaque n-uplet b dans S :

1. on concatène a et b et on obtient un n-uplet qui a pour attributs

$$\overbrace{A_1, \ldots, A_m, X_1, \ldots, X_k}^{a}, \overbrace{B_1, \ldots, B_n, X_1, \ldots, X_k}^{b}$$

2. on ne le garde que si chaque attribut X_i de a est égal à l'attribut X_i de b : $\forall_{i=1..k} a.X_i = b.X_i$.
3. on élimine les valeurs (colonnes) dupliquées : on obtient un n-uplet qui a pour attributs

$$\overbrace{A_1, \ldots, A_m}^{a}, \overbrace{B_1, \ldots, B_m}^{b}, \overbrace{X_1, \ldots, X_k}^{a \text{ et } b}$$

θ-Jointure

- ARGUMENTS : deux relations qui ne partagent pas d'attributs :

$$R(A_1, \ldots, A_m) \qquad S(B_1, \ldots, B_n)$$

- NOTATION : $R \bowtie_{A_i \theta B_j} S$, $\theta \in \{=, \neq, <, \leq, >, \geq\}$
- SCHÉMA DE $T = R \bowtie_{A_I \theta B_J} S$: $T(A_1, \ldots, A_m, B_1, \ldots, B_n)$
- VALEUR DE $T = R \bowtie_{A_I \theta B_J} S$: $T = \sigma_{A_i \theta B_j}(R \times S))$
- ÉQUIJOINTURE : θ est l'**égalité**.

Exemple de θ-Jointure : R $\bowtie_{A \leq C}$ S

R

A	B
1	a
1	b
3	a

S

C	D	E
1	b	a
2	b	c
4	a	a

T := R × S

	A	B	C	D	E
	1	a	1	b	a
	1	a	2	b	c
	1	a	4	a	a
	1	b	1	b	a
	1	b	2	b	c
	1	b	4	a	a
$A > C \rightarrow$	**3**	a	**1**	b	a
$A > C \rightarrow$	**3**	a	**2**	b	c
	3	a	4	a	a

$\Rightarrow$

$\sigma_{A \leq C}(\mathbf{T}) = \mathbf{R} \bowtie_{A \leq C} \mathbf{S}$

A	B	C	D	E
1	a	1	b	a
1	a	2	b	c
1	a	4	a	a
1	b	1	b	a
1	b	2	b	c
1	b	4	a	a
3	a	4	a	a

Exemple d'équijointure : $R \bowtie_{B=D} S$

R

A	B
1	a
1	b
3	a

S

C	D	E
1	b	a
2	b	c
4	a	a

T := R × S

	A	B	C	D	E
$B \neq D \rightarrow$	1	**a**	1	**b**	a
$B \neq D \rightarrow$	1	**a**	2	**b**	c
	1	a	4	a	a
	1	b	1	b	a
	1	b	2	b	c
$B \neq D \rightarrow$	1	**b**	4	**a**	a
$B \neq D \rightarrow$	3	**a**	1	**b**	a
$B \neq D \rightarrow$	3	**a**	2	**b**	c
	3	a	4	a	a

$\Rightarrow$

$\sigma_{B=D}(\mathbf{T}) = \mathbf{R} \bowtie_{B=D} \mathbf{S}$

A	B	C	D	E
1	a	4	a	a
1	b	1	b	a
1	b	2	b	c
3	a	4	a	a

Utilisation de l'équijointure et jointure naturelle I

$$IMMEUBLE(ADI, NBETAGES, DATEC, PROP)$$

$$APPIM(ADI, NAP, OCCUP, ETAGE)$$

1. Nom du propriétaire de l'immeuble où est situé l'appartement occupé par *Durand* :

$$\pi_{PROP}(\overbrace{IMMEUBLE \bowtie \sigma_{OCCUP=''DURAND''}(APPIM)}^{JointureNaturelle})$$

2. Appartements occupés par des propriétaires d'immeuble :

$$\pi_{ADI,NAP,ETAGE}\overbrace{(APPIM \bowtie_{OCCUP=PROP} IMMEUBLE)}^{équijointure}$$

Utilisation de l'équijointure et jointure naturelle II

(Exercice)

IMMEUBLE

ADI	NBETAGES	DATEC	PROP
2 rue Conté	4	1973	Durand
3 rue de Paris	2	1987	Smith
7 rue des Lilas	3	1979	Durand
6 rue Monod	2	1977	Danes

APPIM

ADI	NAP	OCCUP	ETAGE
3 rue de Paris	7	Nardini	1
2 rue Conté	011	Gilmore	0
2 rue Conté	012	Danes	0
2 rue Conté	027	Geller	0
2 rue Conté	137	Doose	1
6 rue Monod	4	Patty	2
6 rue Monod	2	Durand	1
3 rue de Paris	5	Smith	1

Autre exemple de requête : Nom et adresse des clients qui ont commandé des parpaings.

- Schéma Relationnel :

$$COMMANDES(PNOM, CNOM, NUM\_CMDE, QTE)$$

$$CLIENTS(CNOM, CADRESSE, BALANCE)$$

- Requête Relationnelle :

$$\pi_{CNOM,CADRESSE}(CLIENTS \bowtie \sigma_{PNOM=''PARPAING''}(COMMANDES))$$

Plan du cours

Union

- ARGUMENTS : 2 relations de même schéma :

$$R(A_1, \ldots, A_m) \qquad S(A_1, \ldots, A_m)$$

- NOTATION : $R \cup S$
- SCHÉMA DE $T = R \cup S$: $T(A_1, \ldots, A_m)$
- VALEUR DE T : Union ensembliste sur $D_1 \times \ldots \times D_m$:

$$T = \{t \mid t \in R \vee t \in S\}$$

Exemple d'union

R

A	B
a	b
a	c
d	e

S

A	B
a	b
a	e
d	e
f	g

$\Rightarrow$

$R \cup S$

A	B
a	b
a	c
d	e
a	e
f	g

Plan du cours

Différence

- ARGUMENTS : 2 relations de même schéma :

$$R(A_1, \ldots, A_m) \quad S(A_1, \ldots, A_m)$$

- NOTATION : $R - S$
- SCHÉMA DE $T = R - S$: $T(A_1, \ldots, A_m)$
- VALEUR DE T : Différence ensembliste sur $D_1 \times \ldots \times D_m$:

$$T = \{t \mid t \in R \wedge t \notin S\}$$

Exemple de différence

R	A	B
→	a	b
	a	c
→	d	e

S	A	B
→	a	b
	a	e
→	d	e
	f	g

R - S	A	B
	a	c

S - R	A	B
	a	e
	f	g

Plan du cours

Intersection

- ARGUMENTS : 2 relations de même schéma :

$$R(A_1, \ldots, A_m) \qquad S(A_1, \ldots, A_m)$$

- NOTATION : $R \cap S$
- SCHÉMA DE $T = R \cap S$: $T(A_1, \ldots, A_m)$
- VALEUR DE T : Intersection ensembliste sur $D_1 \times \ldots \times D_m$:

$$T = \{t \mid t \in R \wedge t \in S\}$$

Exemple d'intersection

R	A	B
→	a	b
	a	c
→	d	e

S	A	B
→	a	b
	a	e
→	d	e
	f	g

R - S

A	B
a	c

$R \cap S = R - (R - S)$

A	B
a	b
d	e

Plan du cours

Semi-jointure

- ARGUMENTS : 2 relations quelconques :

$$R(A_1, \ldots, A_m, X_1, \ldots, X_k)$$

$$S(B_1, \ldots, B_n, X_1, \ldots, X_k)$$

où $X_1, \ldots, X_k$ sont les attributs en commun.

- NOTATION : $R \ltimes S$
- SCHÉMA DE $T = R \ltimes S$: $T(A_1, \ldots, A_m, X_1, \ldots, X_k)$
- VALEUR DE $T = R \ltimes S$: Projection sur les attributs de R de la jointure naturelle entre R et S.

Semi-jointure

La semi-jointure correspond à une sélection où la condition de sélection est définie par le biais d'une autre relation.
Soit U l'ensemble des attributs de R.

$$R \ltimes S = \pi_U(R \bowtie S)$$

Exemple de semi-jointure

R

A	B	C
a	b	c
d	b	c
b	b	f
c	a	d

S

B	C	D
b	c	d
b	c	e
a	d	b

$\Rightarrow \pi_{A,B,C}(R \bowtie S) \Rightarrow$

R $\ltimes$ S

A	B	C
a	b	c
d	b	c
c	a	d

Plan du cours

Exemple de division

REQUÊTE : Clients qui commandent tous les produits:

COMM

NUM	NOM	PNOM	QTE
1	Jean	briques	100
2	Jean	ciment	2
3	Jean	parpaing	2
4	Paul	briques	200
5	Paul	parpaing	3
6	Vincent	parpaing	3

$R = \pi_{NOM,PNOM}(COMM)$:

R

NOM	PNOM
Jean	briques
Jean	ciment
Jean	parpaing
Paul	briques
Paul	parpaing
Vincent	parpaing

PROD

PNOM
briques
ciment
parpaing

$\Downarrow$

$R \div PROD$

NOM
Jean

Exemple de division

R

A	B	C	D
a	b	x	m
a	b	y	n
a	b	z	o
b	c	x	o
b	d	x	m
c	e	x	m
c	e	y	n
c	e	z	o
d	a	z	p
d	a	y	m

S

C	D
x	m
y	n
z	o

R÷S

A	B
a	b
c	e

Division

- ARGUMENTS : 2 relations :

$$R(A_1, \ldots, A_m, X_1, \ldots, X_k) \qquad S(X_1, \ldots, X_k)$$

où **tous** les attributs de S sont des attributs de R.

- NOTATION : $R \div S$
- SCHÉMA DE $T = R \div S$: $T(A_1, \ldots, A_m)$
- VALEUR DE $T = R \div S$:

$$R \div S = \{(a_1, \ldots, a_m) \mid \forall (x_1, \ldots, x_k) \in S \ : (a_1, \ldots, a_m, x_1, \ldots, x_k) \in R\}$$

Division

La division s'exprime en fonction du produit cartésien, de la projection et de la différence : $R \div S = R_1 - R_2$ où

$$R_1 = \pi_{A_1,\ldots,A_m}(R) \text{ et } R_2 = \pi_{A_1,\ldots,A_m}((R_1 \times S) - R)$$

Plan du cours

Renommage

- NOTATION : ρ
- ARGUMENTS : 1 relation :

$$R(A_1, \ldots, A_n)$$

- SCHÉMA DE $T = \rho_{A_i \to B_i} R$: $T(A_1, \ldots, A_{i-1}, B_i, A_{i+1}, \ldots, A_n)$
- VALEUR DE $T = \rho_{A_i \to B_i} R$: $T = R$. La valeur de R est inchangée. Seul le nom de l'attribut A_i a été remplacé par B_i

Plan du cours

Principe

- SQL (Structured Query Language) est le Langage de Requêtes standard pour les SGBD relationnels
- Expression d'une requête par un bloc *SELECT FROM WHERE*

SELECT	<liste des attributs a projeter>
FROM	<liste des relations arguments>
WHERE	<conditions sur un ou plusieurs attributs>

- Dans les requêtes simples, la correspondance avec l'algèbre relationnelle est facile à mettre en évidence.

Plan du cours

Projection

SCHÉMA : **COMMANDES**(NUM,CNOM,PNOM,QUANTITE)
REQUÊTE : *Toutes les commandes*
ALGÈBRE : *COMMANDES*
SQL:

```
SELECT NUM,CNOM,PNOM,QUANTITE
FROM   COMMANDES
```

ou

```
SELECT *
FROM   COMMANDES
```

Projection avec élimination de doublons

SCHÉMA : **COMMANDES**(NUM,CNOM,PNOM,QUANTITE)
REQUÊTE: *Produits commandés*
ALGÈBRE : $\pi_{PNOM}(COMMANDES)$
SQL :

```
SELECT PNOM
FROM   COMMANDES
```

NOTE: Contrairement à l'algèbre relationnelle, SQL n'élimine pas les doublons (sémantique multi-ensemble). Pour les éliminer on utilise DISTINCT :

```
SELECT DISTINCT PNOM
FROM   COMMANDES
```

Le DISTINCT peut être remplacé par la clause UNIQUE.

Plan du cours

Sélection

SCHÉMA : **COMMANDES**(NUM,CNOM,PNOM,QUANTITE)
REQUÊTE: *Produits commandés par Jean*
ALGÈBRE: $\pi_{PNOM}(\sigma_{CNOM=\text{"}JEAN\text{"}}(COMMANDES))$
SQL:

```
SELECT PNOM
FROM   COMMANDES
WHERE  CNOM = 'JEAN'
```

REQUÊTE: *Produits commandés par Jean en quantité supérieure à 100*
ALGÈBRE: $\pi_{PNOM}(\sigma_{CNOM=\text{"}JEAN\text{"}\wedge QUANTITE>100}(COMMANDES))$
SQL:

```
SELECT PNOM
FROM   COMMANDES
WHERE  CNOM = 'JEAN'
AND    QUANTITE > 100
```

Conditions simples

Les conditions de base sont exprimées de deux façons:

1. *attribut comparateur valeur*
2. *attribut comparateur attribut*

où *comparateur* est $=, <, >, !=, \ldots,$
Soit le schéma de relation **FOURNITURE**(PNOM,FNOM,PRIX)
Exemple :

```
SELECT PNOM FROM FOURNITURE WHERE PRIX > 2000
```

Exemple

SCHÉMA : **FOURNITURE**(PNOM,FNOM,PRIX)
REQUÊTE: *Produits dont le nom est celui du fournisseur*
SQL:

```
SELECT PNOM
FROM   FOURNITURE
WHERE  PNOM = FNOM
```

Appartenance à une intervalle : BETWEEN

SCHÉMA : **FOURNITURE**(PNOM,FNOM,PRIX)
REQUÊTE: *Produits avec un coût entre 1000F et 2000F*
SQL:

```
SELECT PNOM
FROM   FOURNITURE
WHERE  PRIX BETWEEN 1000 AND 2000
```

NOTE: La condition y BETWEEN x AND z est équivalente à $y <= z$ AND $x <= y$.

Chaînes de caractères : LIKE

SCHÉMA : **COMMANDES**(NUM,CNOM,PNOM,QUANTITE)
REQUÊTE: *Clients dont le nom commence par "C"*
SQL:

```
SELECT CNOM
FROM   COMMANDES
WHERE  CNOM LIKE 'C%'
```

NOTE: Le littéral qui suit LIKE doit être une chaîne de caractères éventuellement avec des caractères jokers _ (un caractère quelconque) et % (une chaîne de caractères quelconque). Pas exprimable avec l'algèbre relationnelle.

Plan du cours

Valeurs inconnues : NULL

La valeur NULL est une valeur "spéciale" qui représente une *valeur (information) inconnue*.

1. $A\ \theta\ B$ est `inconnu` (ni vrai, ni faux) si la valeur de A ou/et B est NULL (θ est l'un de $=, <, >, !=, ...$).
2. $A\ op\ B$ est NULL si la valeur de A ou/et B est NULL (op est l'un de $+, -, *, /$).

Comparaison avec valeurs nulles

SCHÉMA et INSTANCE :

FOURNISSEUR	FNOM	VILLE
	Toto	Paris
	Lulu	*NULL*
	Marco	Marseille

REQUÊTE: *Les Fournisseurs de Paris.*
SQL:

```
SELECT FNOM
FROM   FOURNISSEUR
WHERE  VILLE = 'Paris'
```

RÉPONSE : Toto

Comparaison avec valeurs nulles

REQUÊTE: *Fournisseurs dont la ville est inconnue.*
SQL:

```
SELECT FNOM
FROM   FOURNISSEUR
WHERE  VILLE = NULL
```

La réponse est vide. Pourquoi?
SQL:

```
SELECT FNOM
FROM   FOURNISSEUR
WHERE  VILLE IS NULL
```

RÉPONSE : Lulu

Trois valeurs de vérité

Trois valeurs de vérité: `vrai`, `faux` et **inconnu**

1. `vrai AND inconnu = inconnu`
2. `faux AND inconnu = faux`
3. `inconnu AND inconnu = inconnu`
4. `vrai OR inconnu = vrai`
5. `faux OR inconnu = inconnu`
6. `inconnu OR inconnu = inconnu`
7. `NOT inconnu = inconnu`

Exemple

SCHÉMA : **EMPLOYE**(EMPNO,ENOM,DEPNO,SAL)
SQL:

```
SELECT ENOM
  FROM EMPLOYE
 WHERE SAL > 20000 OR SAL <= 20000
```

On ne trouve que les noms des employés avec un salaire connu. Pourquoi?

Plan du cours

Jointures : exemple

SCHÉMA : **COMMANDES**(NUM,CNOM,PNOM,QUANTITE)
FOURNITURE(PNOM,FNOM,PRIX)

REQUÊTE : *Nom, Coût, Fournisseur des Produits commandés par Jean*

ALGÈBRE :

$$\pi_{PNOM,PRIX,FNOM}(\sigma_{CNOM="JEAN"}(COMMANDES) \bowtie (FOURNITURE))$$

Jointure : exemple

SCHÉMA : **COMMANDES**(NUM,CNOM,PNOM,QUANTITE)
FOURNITURE(PNOM,FNOM,PRIX)

SQL :

```
SELECT COMMANDES.PNOM, PRIX, FNOM
FROM   COMMANDES, FOURNITURE
WHERE  CNOM = 'JEAN' AND
       COMMANDES.PNOM = FOURNITURE.PNOM
```

NOTES:

- On exprime une jointure comme un produit cartésien suivi d'une sélection et d'une projection (on a déjà vu ça?)
- Algèbre : la requête contient une jointure naturelle.
- SQL : il faut expliciter les attributs de jointure.

Auto-jointure et renommage

SCHÉMA : **FOURNISSEUR**(FNOM,STATUT,VILLE)
REQUÊTE: *"Couples" de fournisseurs situés dans la même ville*
SQL:

```
SELECT PREM.FNOM, SECOND.FNOM
FROM   FOURNISSEUR PREM, FOURNISSEUR SECOND
WHERE  PREM.VILLE =  SECOND.VILLE AND
       PREM.FNOM < SECOND.FNOM
```

La deuxième condition permet

1. l'élimination des paires (x,x)
2. de garder un exemplaire parmi les couples symétriques (x,y) et (y,x)

NOTE: PREM représente une instance de FOURNISSEUR, SECOND une autre instance de FOURNISSEUR.

Auto-jointure

SCHÉMA : **EMPLOYE**(EMPNO,ENOM,DEPNO,SAL)
REQUÊTE: *Nom et Salaire des Employés gagnant plus que l'employé de numéro 12546*
SQL:

```
SELECT E1.ENOM, E1.SAL
FROM   EMPLOYE E1, EMPLOYE E2
WHERE  E2.EMPNO = 12546 AND
       E1.SAL > E2.SAL
```

- On confond souvent les auto-jointures avec des sélections simples.
- Requête en algèbre?

Opérations de jointure

SQL2 introduit des opérations de jointure dans la clause FROM :

SQL2	opération	**Algèbre**
R1 CROSS JOIN R2	produit cartesien	$R1 \times R2$
R1 JOIN R2 ON R1.A < R2.B	théta-jointure	$R1 \bowtie_{R1.A<R2.B} R2$
R1 NATURAL JOIN R2	jointure naturelle	$R1 \bowtie R2$

Jointure naturelle : exemple

SCHEMA: **EMP**(EMPNO,ENOM,DEPNO,SAL)
DEPT(DEPNO,DNOM)

REQUÊTE: *Numéros des départements avec les noms de leurs employés.*

SQL2:

```
SELECT DEPNO, ENOM
FROM   DEPT NATURAL JOIN EMP
```

Note : L'expression `DEPT NATURAL JOIN EMP` fait la jointure naturelle (sur les attributs en commun) et l'attribut `DEPNO` n'apparaît qu'une seule fois dans le schéma du résultat.

θ-jointure : exemple

REQUÊTE: *Nom et salaire des employés gagnant plus que l'employé 12546*
SQL2:

```
SELECT E1.ENOM, E1.SAL
FROM   EMPLOYE E1 JOIN EMPLOYE E2 ON E1.SAL > E2.SAL
WHERE  E2.EMPNO = 12546
```

Jointure interne

EMP

EMPNO	*DEPNO*	SAL
Tom	1	10000
Jim	2	20000
Karin	3	15000

DEPT

DEPNO	DNOM
1	Comm.
2	Adm.
4	Tech.

Jointure (interne) : les n-uplets qui ne peuvent pas être joints sont éliminés :

EMP NATURAL JOIN DEPT

Tom	1	10000	Comm.
Jim	2	20000	Adm.

Jointure externe

Jointure externe : les n-uplets qui ne peuvent pas être joints *ne sont pas éliminés*.

- On garde tous les n-uplets des deux relations :

EMP NATURAL *FULL* OUTER JOIN DEPT

Tom	1	10000	Comm.
Jim	2	20000	Adm.
Karin	3	15000	NULL
NULL	4	NULL	Tech.

- On garde tous les n-uplets de la première relation (gauche) :

EMP NATURAL *LEFT* OUTER JOIN DEPT

Tom	1	10000	Comm.
Jim	2	20000	Adm.
Karin	3	15000	NULL

- On peut aussi écrire (dans Oracle) :

```
select EMP.*, DEP.DNOM
from EMP, DEPT
where EMP.DEPNO = DEPT.DEPNO (+)
```

- On garde tous les n-uplets de la deuxième relation (droite) :

EMP NATURAL *RIGHT* OUTER JOIN DEPT

Tom	1	10000	Comm.
Jim	2	20000	Adm.
NULL	4	NULL	Tech.

- On peut aussi écrire (dans Oracle) :

```
select EMP.*, DEP.DNOM
from EMP, DEPT
where EMP.DEPNO (+) = DEPT.DEPNO
```

Jointures externes dans SQL2

- R1 NATURAL FULL OUTER JOIN R2 : Remplir R1.* et R2.*
- R1 NATURAL LEFT OUTER JOIN R2 : Remplir R2.*
- R1 NATURAL RIGHT OUTER JOIN R2 : Remplir R1.*

avec NULL quand nécessaire.
D'une manière similaire on peut définir des théta-jointures externes :

- R1 (FULL|LEFT|RIGHT) OUTER JOIN R2 ON prédicat

Plan du cours

Union

COMMANDES(NUM,CNOM,PNOM,QUANTITE)
FOURNITURE(PNOM,FNOM,PRIX)
REQUÊTE: *Produits qui coûtent plus que 1000F ou ceux qui sont commandés par Jean*

ALGÈBRE:

$$\pi_{PNOM}(\sigma_{PRIX>1000}(FOURNITURE))$$
$$\cup$$
$$\pi_{PNOM}(\sigma_{CNOM='Jean'}(COMMANDES))$$

SQL:

```
SELECT PNOM
FROM   FOURNITURE
WHERE  PRIX >= 1000
UNION
SELECT PNOM
FROM   COMMANDES
WHERE  CNOM = 'Jean'
```

NOTE: L'union élimine les dupliqués. Pour garder les dupliqués on utilise l'opération `UNION ALL` : le résultat contient chaque n-uplet $a + b$ fois, où a et b est le nombre d'occurrences du n-uplet dans la première et la deuxième requête.

Plan du cours

Différence

La différence ne fait pas partie du standard.
EMPLOYE(EMPNO,ENOM,DEPTNO,SAL)
DEPARTEMENT(DEPTNO,DNOM,LOC)
REQUÊTE: *Départements sans employés*
ALGÈBRE: $\pi_{DEPTNO}(DEPARTEMENT) - \pi_{DEPTNO}(EMPLOYE)$
SQL:

```
SELECT DEPTNO FROM   DEPARTEMENT
   EXCEPT
SELECT DEPTNO FROM   EMPLOYE
```

NOTE: La différence élimine les dupliqués. Pour garder les dupliqués on utilise l'opération `EXCEPT ALL` : le résultat contient chaque n-uplet $a - b$ fois, où a et b est le nombre d'occurrences du n-uplet dans la première et la deuxième requête.

Plan du cours

Intersection

L'intersection ne fait pas partie du standard.
EMPLOYE(EMPNO,ENOM,DEPTNO,SAL)
DEPARTEMENT(DEPTNO,DNOM,LOC)
REQUÊTE: *Départements ayant des employés qui gagnent plus que 20000F et qui se trouvent à Paris*

ALGÈBRE :

$$\pi_{DEPTNO}(\sigma_{LOC="Paris"}(DEPARTEMENT)) \cap \pi_{DEPTNO}(\sigma_{SAL>20000}(EMPLOYE))$$

SQL:

```
SELECT DEPTNO
FROM   DEPARTEMENT
WHERE  LOC = 'Paris'
INTERSECT
SELECT DEPTNO
FROM   EMPLOYE
WHERE  SAL > 20000
```

NOTE: L'intersection élimine les dupliqués. Pour garder les dupliqués on utilise l'opération `INTERSECT ALL` : le résultat contient chaque n-uplet $min(a, b)$ fois, où a et b est le nombre d'occurrences du n-uplet dans la première et la deuxième requête.

Plan du cours

Requêtes imbriquées simples

La Jointure s'exprime par deux blocs SFW imbriqués

Soit le schéma de relations
COMMANDES(NUM,CNOM,PNOM,QUANTITE)
FOURNITURE(PNOM,FNOM,PRIX)
REQUÊTE: *Nom, prix et fournisseurs des Produits commandés par Jean*
ALGÈBRE:

$$\pi_{PNOM,PRIX,FNOM}(\sigma_{CNOM="JEAN"}(COMMANDES) \bowtie (FOURNITURE))$$

SQL:

```
SELECT PNOM,PRIX,FNOM
FROM   FOURNITURE
WHERE  PNOM IN (SELECT PNOM
                FROM   COMMANDES
                WHERE  CNOM = 'JEAN')
```

ou

```
SELECT DISTINCT FOURNITURE.PNOM,PRIX,FNOM
FROM   FOURNITURE,COMMANDES
WHERE  FOURNITURE.PNOM = COMMANDES.PNOM
AND    CNOM = ``JEAN''
```

La Différence s'exprime aussi par deux blocs SFW imbriqués

Soit le schéma de relations
EMPLOYE(EMPNO,ENOM,DEPNO,SAL)
DEPARTEMENT(DEPTNO,DNOM,LOC)
REQUÊTE: *Départements sans employés*
ALGÈBRE :

$$\pi_{DEPTNO}(DEPARTEMENT) - \pi_{DEPTNO}(EMPLOYE)$$

SQL:

```
SELECT DEPTNO
FROM   DEPARTEMENT
WHERE  DEPTNO NOT IN (SELECT DEPTNO FROM   EMPLOYE)
```

ou

```
SELECT DEPTNO
FROM DEPARTEMENT
EXCEPT
SELECT DEPTNO
FROM EMPLOYE
```

Requêtes imbriquées plus complexes : ANY - ALL

SCHÉMA: **FOURNITURE**(PNOM,FNOM,PRIX)
REQUÊTE: *Fournisseurs des briques à un coût inférieur au coût maximum des ardoises*

```
SQL :     SELECT FNOM
          FROM   FOURNITURE
          WHERE  PNOM = 'Brique'
          AND    PRIX < ANY (SELECT PRIX
                             FROM   FOURNITURE
                             WHERE  PNOM = 'Ardoise')
```

La condition $f\ \theta$ ANY (SELECT ... FROM ...) est vraie ssi la comparaison $f\theta v$ est vraie au moins pour une valeur v du résultat du bloc (SELECT F FROM ...).

"IN" et "= ANY"

COMMANDE(NUM,CNOM,PNOM,QUANTITE)
FOURNITURE(PNOM,FNOM,PRIX)
REQUÊTE: *Nom, prix et fournisseur des produits commandés par Jean*
SQL:

```
SELECT PNOM, PRIX, FNOM
FROM   FOURNITURE
WHERE  PNOM = ANY (SELECT PNOM
                   FROM   COMMANDE
                   WHERE  CNOM = 'JEAN')
```

NOTE: Les prédicats IN et = ANY sont utilisés de façon équivalente.

ALL

SCHÉMA: **COMMANDE**(NUM,CNOM,PNOM,QUANTITE)
REQUÊTE: *Client ayant commandé la plus petite quantité de briques*
SQL:

```
SELECT CNOM
FROM   COMMANDE
WHERE  PNOM = 'Brique' AND
       QUANTITE <= ALL (SELECT QUANTITE
                        FROM   COMMANDE
                        WHERE  PNOM = 'Brique')
```

La condition $f\ \theta$ ALL (SELECT ... FROM ...) est vraie ssi la comparaison $f\theta v$ est vraie pour toutes les valeurs v du résultat du bloc (SELECT ... FROM ...).

"NOT IN" et "NOT = ALL"

EMPLOYE(EMPNO,ENOM,DEPNO,SAL)
DEPARTEMENT(DEPTNO,DNOM,LOC)
REQUÊTE: *Départements sans employés*
SQL:

```
SELECT DEPTNO
FROM   DEPARTEMENT
WHERE  DEPTNO NOT = ALL (SELECT DEPTNO
                         FROM   EMPLOYE)
```

NOTE: Les prédicats NOT IN et NOT = ALL sont utilisés de façon équivalente.

EXISTS

FOURNISSEUR(FNOM,STATUS,VILLE)
FOURNITURE(PNOM,FNOM,PRIX)
REQUÊTE: *Fournisseurs qui fournissent au moins un produit*
SQL :

```
SELECT FNOM
  FROM FOURNISSEUR
 WHERE EXISTS (SELECT *
                 FROM FOURNITURE
                WHERE FNOM = FOURNISSEUR.FNOM)
```

La condition EXISTS (SELECT * FROM ...) est vraie ssi le résultat du bloc (SELECT F FROM ...) n'est pas vide.

NOT EXISTS

FOURNISSEUR(FNOM,STATUS,VILLE)
FOURNITURE(PNOM,FNOM,PRIX)
REQUÊTE: *Fournisseurs qui ne fournissent aucun produit*
SQL:

```
SELECT FNOM
FROM   FOURNISSEUR
WHERE  NOT EXISTS (SELECT *
                   FROM   FOURNITURE
                   WHERE  FNOM = FOURNISSEUR.FNOM)
```

La condition NOT EXISTS (SELECT * FROM . . .) est vraie ssi le résultat du bloc (SELECT F FROM . . .) est vide.

Formes équivalentes de quantification

Si θ est un des opérateurs de comparaison $<, =, >, \ldots$

- La condition

  x θ ANY (SELECT Ri.y FROM R1, ... Rn WHERE p)

  est équivalente à

  EXISTS (SELECT * FROM R1, ... Rn WHERE p AND x θ Ri.y)

- La condition

  x θ ALL (SELECT Ri.y FROM R1, ... Rn WHERE p)

  est équivalente à

NOT EXISTS (SELECT * FROM R1, ... Rn WHERE (p) AND NOT (x θ Ri.y))

Exemple : "EXISTS" et "= ANY"

COMMANDE(NUM,CNOM,PNOM,QUANTITE)
FOURNITURE(PNOM,FNOM,PRIX)
REQUÊTE: *Nom, prix et fournisseur des produits commandés par Jean*

```
SELECT PNOM, PRIX, FNOM  FROM   FOURNITURE
WHERE  EXISTS  (SELECT * FROM   COMMANDE
                WHERE CNOM = 'JEAN'
                  AND PNOM = FOURNITURE.PNOM)
```

ou

```
SELECT PNOM, PRIX, FNOM FROM   FOURNITURE
WHERE  PNOM = ANY (SELECT PNOM  FROM   COMMANDE
                   WHERE  CNOM = 'JEAN')
```

Encore plus compliqué...

SCHÉMA: **FOURNITURE**(PNOM,FNOM,PRIX)
REQUÊTE: *Fournisseurs qui fournissent au moins un produit avec un coût supérieur au coût de tous les produits fournis par Jean*

```
SELECT DISTINCT P1.FNOM
FROM   FOURNITURE P1
WHERE  NOT EXISTS  (SELECT * FROM   FOURNITURE P2
                    WHERE  P2.FNOM = 'JEAN'
                    AND    P1.PRIX <= P2.PRIX)
```

ou

```
SELECT DISTINCT FNOM FROM   FOURNITURE
WHERE  PRIX > ALL (SELECT PRIX FROM   FOURNITURE
                   WHERE FNOM = 'JEAN')
```

Et la division?

FOURNITURE(FNUM,PNUM,QUANTITE)
PRODUIT(PNUM,PNOM,PRIX)
FOURNISSEUR(FNUM,FNOM,STATUS,VILLE)
REQUÊTE: *Noms des fournisseurs qui fournissent tous les produits*
ALGÈBRE:

$$\text{R1} := \pi_{FNUM,PNUM}(FOURNITURE) \div \pi_{PNUM}(PRODUIT)$$
$$\text{R2} := \pi_{FNOM}(FOURNISSEUR \bowtie R1)$$

SQL:

```
SELECT FNOM
FROM   FOURNISSEUR
WHERE  NOT EXISTS
       (SELECT *
        FROM  PRODUIT
        WHERE NOT EXISTS
              (SELECT *
               FROM  FOURNITURE
               WHERE FOURNITURE.FNUM = FOURNISSEUR.FNUM
               AND   FOURNITURE.PNUM = PRODUIT.PNUM))
```

Plan du cours

COUNT, SUM, AVG, MIN, MAX

REQUÊTE: *Nombre de fournisseurs parisiens*

```
SELECT COUNT(*)
FROM   FOURNISSEUR
WHERE  VILLE = 'Paris'
```

REQUÊTE: *Nombre de fournisseurs qui fournissent des produits*

```
SELECT COUNT(DISTINCT FNOM)
FROM   FOURNITURE
```

NOTE: La fonction COUNT(*) compte le nombre des n-uplets du résultat d'une requête sans élimination des dupliqués ni vérification des valeurs nulles. Dans le cas contraire on utilise la clause COUNT(DISTINCT . . .).

SUM et AVG

REQUÊTE: *Quantité totale de Briques commandées*

```
SELECT SUM (QUANTITE)
FROM   COMMANDES
WHERE  PNOM = 'Brique'
```

REQUÊTE: *Coût moyen de Briques fournies*

```
SELECT AVG (PRIX)
FROM   FOURNITURE
WHERE  PNOM = 'Brique'
```

ou

```
SELECT SUM (PRIX)/COUNT(PRIX)
FROM FOURNITURE
WHERE PNOM = 'Brique'
```

MIN et MAX

REQUÊTE: *Le prix des briques le moins chères.*

```
SELECT MIN(PRIX)
FROM   FOURNITURE
WHERE  PNOM = 'Briques';
```

REQUÊTE: *Le prix des briques le plus chères.*

```
SELECT MAX(PRIX)
FROM   FOURNITURE
WHERE  PNOM = 'Briques';
```

Comment peut-on faire sans MIN et MAX?

Requête imbriquée avec fonction de calcul

REQUÊTE: *Fournisseurs de briques dont le prix est en dessous du prix moyen*

```
SELECT FNOM
FROM   FOURNITURE
WHERE  PNOM = 'Brique' AND
       PRIX < (SELECT AVG(PRIX)
               FROM   FOURNITURE
               WHERE PNOM = 'Brique')
```

Division et fonctions de calcul

FOURNITURE(FNUM,PNUM,QUANTITE)
PRODUIT(PNUM,PNOM,PRIX)
FOURNISSEUR(FNUM,FNOM,STATUS,VILLE)
REQUÊTE: *Noms des fournisseurs qui fournissent tous les produits*
SQL:

```
SELECT FNOM
FROM   FOURNISSEUR
WHERE  NOT EXISTS
       (SELECT *
        FROM  PRODUIT
        WHERE NOT EXISTS
              (SELECT *
               FROM  FOURNITURE
               WHERE FOURNITURE.FNUM = FOURNISSEUR.FNUM
               AND   FOURNITURE.PNUM = PRODUIT.PNUM))
```

Division et fonctions de calcul

FOURNITURE(FNUM,PNUM,QUANTITE)
PRODUIT(PNUM,PNOM,PRIX)
FOURNISSEUR(FNUM,FNOM,STATUS,VILLE)
REQUÊTE: *Noms des fournisseurs qui fournissent tous les produits*
SQL:

```
SELECT FNOM
FROM   FOURNISSEUR
WHERE (SELECT COUNT(*)
              FROM  FOURNITURE
              WHERE FOURNITURE.FNUM = FOURNISSEUR.FNUM
              AND   FOURNITURE.PNUM = PRODUIT.PNUM)
      =(SELECT COUNT(*) from PRODUIT)
```

Plan du cours

GROUP BY

REQUÊTE: *Nombre de fournisseurs* ***par ville***

VILLE	FNOM
PARIS	TOTO
PARIS	DUPOND
LYON	DURAND
LYON	LUCIEN
LYON	REMI

VILLE	COUNT(FNOM)
PARIS	2
LYON	3

```
SELECT VILLE, COUNT(FNOM) FROM FOURNISSEUR GROUP BY VILLE
```

NOTE: La clause GROUP BY permet de préciser les attributs de partitionnement des relations declarées dans la clause FROM.

REQUÊTE: *Donner pour chaque produit son prix moyen*

```
SELECT PNOM, AVG (PRIX)
FROM   FOURNITURE
GROUP BY PNOM
```

RÉSULTAT:

PNOM	AVG (PRIX)
BRIQUE	10.5
ARDOISE	9.8

NOTE: Les fonctions de calcul appliquées au résultat de régroupement sont directement indiquées dans la clause SELECT: le calcul de la moyenne se fait par produit obtenu au résultat après le regroupement.

HAVING

REQUÊTE: *Produits fournis par deux ou plusieurs fournisseurs avec un prix supérieur à 100 Euros*

```
SELECT PNOM
FROM   FOURNITURE
WHERE  PRIX > 100
GROUP BY PNOM
HAVING COUNT(*) >= 2
```

HAVING

AVANT LA CLAUSE HAVING

PNOM	FNOM	PRIX
BRIQUE	TOTO	105
ARDOISE	LUCIEN	110
ARDOISE	DURAND	120

APRÈS LA CLAUSE HAVING

PNOM	FNOM	PRIX
ARDOISE	LUCIEN	110
ARDOISE	DURAND	120

NOTE: La clause HAVING permet d'éliminer des partitionnements, comme la clause WHERE élimine des n-uplets du résultat d'une requête: on garde les produits dont le nombre des fournisseurs est $>= 2$.
Des conditions de sélection peuvent être appliquées avant le calcul d'agrégat (clause WHERE) mais aussi après (clause HAVING).

REQUÊTE: *Nom et prix moyen des produits fournis par des fournisseurs Parisiens et dont le prix minimum est supérieur à 1000 Euros*

```
SELECT PNOM, AVG(PRIX)
FROM   FOURNITURE, FOURNISSEUR
WHERE  VILLE = 'Paris' AND
       FOURNITURE.FNOM = FOURNISSEUR.FNOM
GROUP BY PNOM
HAVING MIN(PRIX) > 1000
```

ORDER BY

En général, le résultat d'une requête SQL n'est pas trié. Pour trier le résultat par rapport aux valeurs d'un ou de plusieurs attributs, on utilise la clause `ORDER BY` :

```
SELECT VILLE, FNOM, PNOM
  FROM FOURNITURE, FOURNISSEUR
 WHERE FOURNITURE.FNOM = FOURNISSEUR.FNOM
 ORDER BY VILLE, FNOM DESC
```

Le résultat est trié par les villes (ASC) et le noms des fournisseur dans l'ordre inverse (DESC).

Plan du cours

Historique

SQL86 - SQL89 ou SQL1 La référence de base:

- Requêtes compilées puis exécutées depuis un programme d'application.
- Types de données simples (entiers, réels, chaînes de caractères de taille fixe)
- Opérations ensemblistes restreintes (UNION).

SQL91 ou SQL2 Standard actuel:

- Requêtes dynamiques
- Types de données plus riches (intervalles, dates, chaînes de caractères de taille variable)
- Différents types de jointures: jointure naturelle, jointure externe
- Opérations ensemblistes: différence (EXCEPT), intersection (INTERSECT)
- Renommage des attributs dans la clause SELECT

Historique

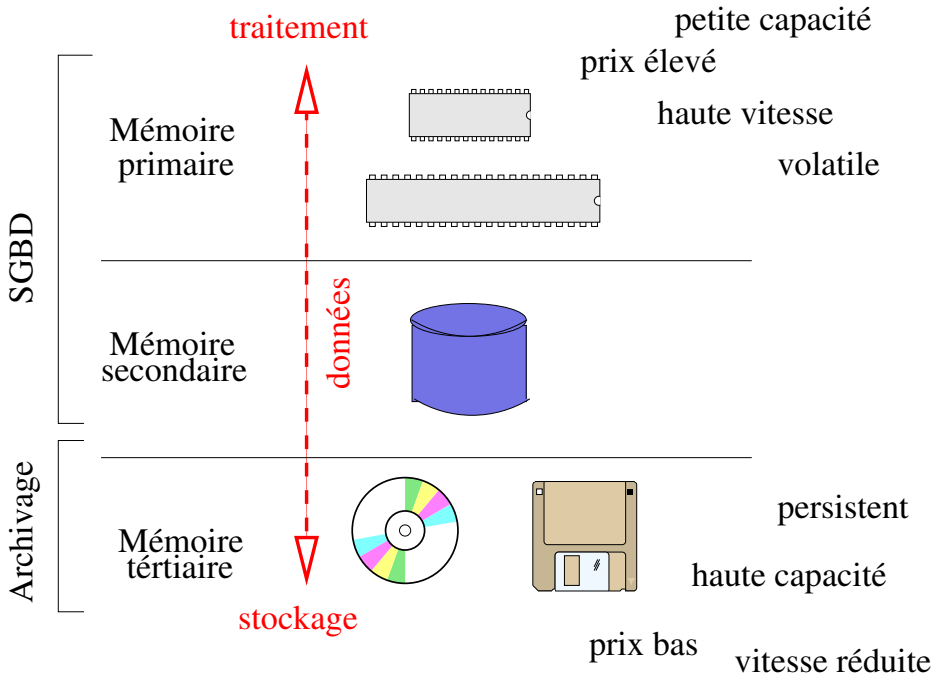
SQL:1999 (SQL3) : SQL devient un langage de programmation :

- Extensions orientées-objet (héritage, méthodes)
- Types structurés
- BLOB, CLOB
- Opérateur de fermeture transitive (recursion)

Plan du cours

Stockage de données

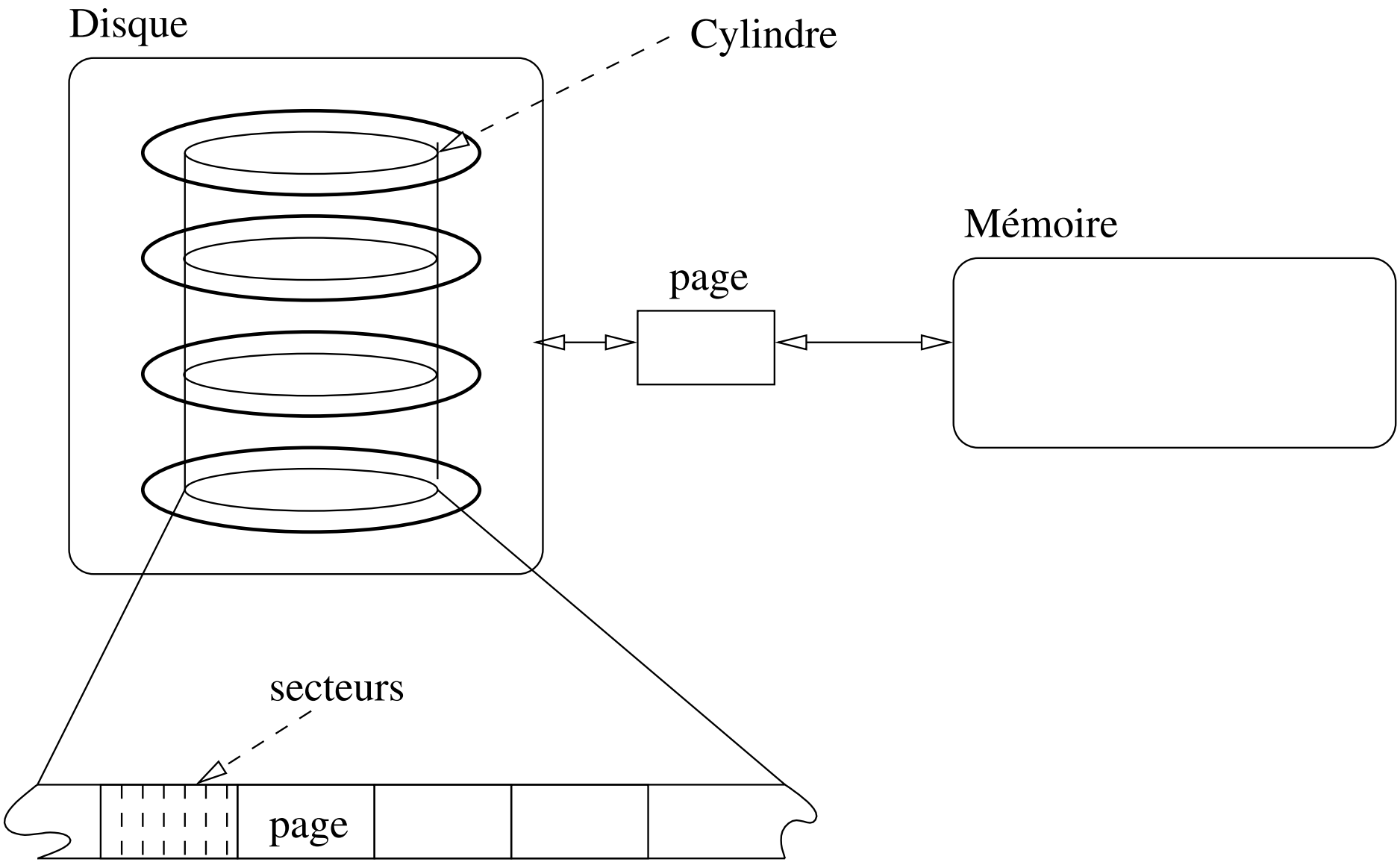
Hiérarchie de mémoire (vitesse d'accès, prix, capacité de stockage, volume) :



Comparaison

Mémoire	AZEN 256MB, SDRAM DIMM
taille	256MB
prix	90 euros
temps d'accès	8ns
euros/MB	90E/256MB = 0.350 euros/MB
Disque	Western Digital 80GB 7200RPM
taille	80GB = 81920 MB
prix	160 euros
temps d'accès 9ms	9 000 000 ns
euros/MB	160/81962 = 0.002 euros/MB

Architecture d'un disque



Organisation d'un disque

1. Un disque est divisé en **blocs physiques** (ou **pages**) de tailles égales (en nombre de secteurs).
2. Accès à un bloc par son adresse (le numéro de cylindre + le numéro du premier secteur + le numéro de face).
3. Le bloc est *l'unité d'échange* entre la mémoire secondaire et la mémoire principale

Exemple : (38792 cylindres)x(16 blocs/cylindre)x(63 secteurs/bloc)x(512 octets/secteur)=20,020,396,000 octets = 20 GO

Les fichiers

Les données sont stockées dans des **fichiers** :

- Un fichier occupe un ou plusieurs blocs (pages) sur un disque.
- L'accès aux fichiers est géré par un logiciel spécifique : le Système de Gestion de Fichiers (SGF).
- Un fichier est caractérisé par son nom.

Les articles

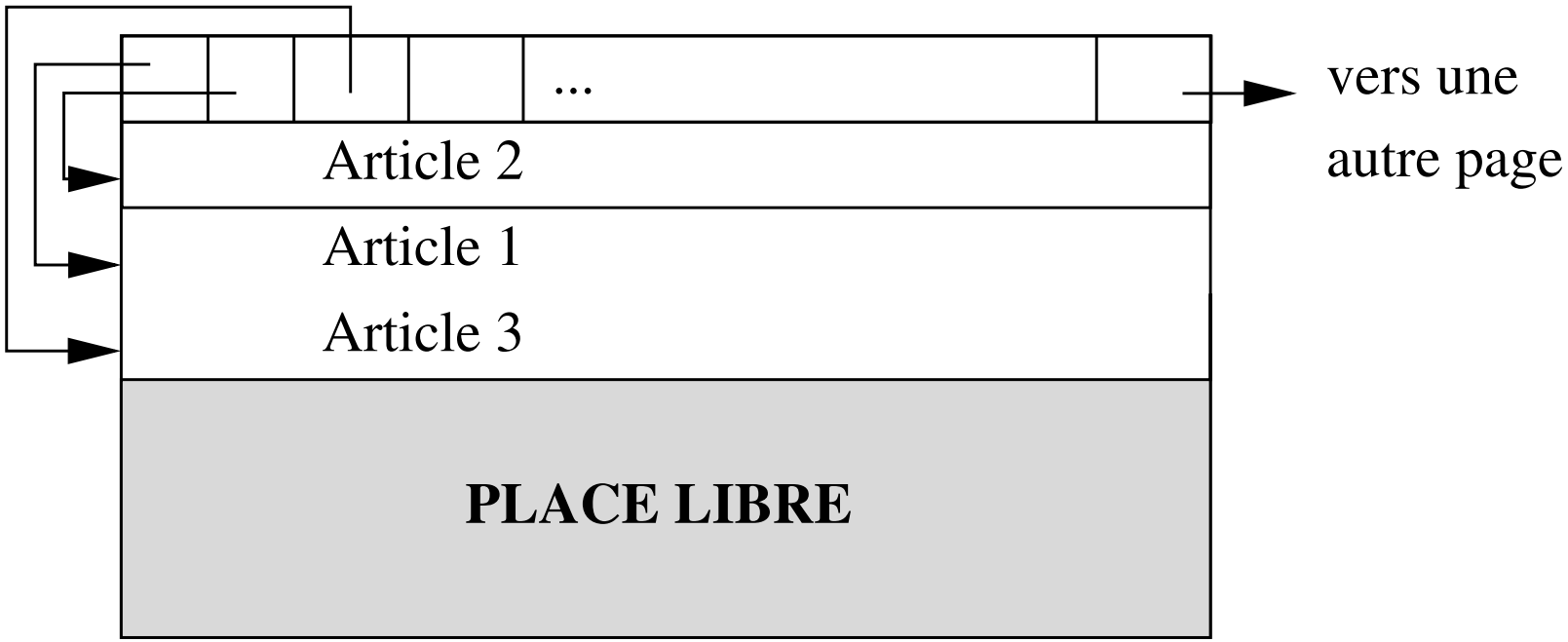
Un fichier est un ensemble **d'articles** (enregistrements, n-uplets) et un article est une séquence de **champs** (attributs).

1. **Articles en format fixe.**
   1. La taille de chaque champ est fixée.
   2. Taille et nom des champs dans le **descripteur** de fichier.
2. **Articles en format variable**.
   1. La taille de chaque champ est variable.
   2. Taille du champ dans son entête.

Articles et pages

- Les articles sont stockés dans les pages (taille article $<$ taille de page)
- L'adresse d'un article est constituée de
  1. L'adresse de la page dans laquelle il se trouve.
  2. Un entier : indice d'une table placée en début de page qui contient l'adresse réelle de l'article dans la page.

Structure interne d'une page



Opérations sur les fichiers

1. **Insérer un article.**
2. **Modifier un article**
3. **Détruire un article**
4. **Rechercher un ou plusieurs article(s)**
   - Par adresse
   - Par valeur d'un ou plusieurs champs
     - ★ requête ponctuelle : $x = cst$
     - ★ requête d'intervalle : $cst_1 < x < cst_2$

Hypothèse: Le **coût** d'une opération est surtout fonction du **nombre d'E/S** (nb de pages échangées)!

Organisation de fichiers

L'organisation d'un fichier est caractérisée par le mode de répartition des articles dans les pages

Il existe trois sortes d'organisation principales :

1. Fichiers séquentiels
2. Fichiers indexés (séquentiels indexés et arbres-B)
3. Fichiers hachés

Exemple de référence

Organisation d'un fichier contenant les articles suivants :

Vertigo 1958	***Annie Hall 1977***
Brazil 1984	***Jurassic Park 1992***
Twin Peaks 1990	***Metropolis 1926***
Underground 1995	***Manhattan 1979***
Easy Rider 1969	***Reservoir Dogs 1992***
Psychose 1960	***Impitoyable 1992***
Greystoke 1984	***Casablanca 1942***
Shining 1980	***Smoke 1995***

Plan du cours

Organisation séquentielle

- **Insertion** : les articles sont stockés séquentiellement dans les pages au fur et à mesure de leur création.
- **Recherche** : le fichier est parcouru séquentiellement.
- **Destruction** : recherche, puis destruction (par marquage d'un bit par exemple).
- **Modification** : recherche, puis réécriture.

Coût des opérations

Nombre moyen de lectures/écritures sur disque d'un fichier de n pages :

- **Recherche** : $\frac{n}{2}$. On parcourt en moyenne la moitié du fichier.
- **Insertion** : $n+1$. On vérifie que l'article n'existe pas avant d' écrire.
- **Destruction et mises-à-jour** : $\frac{n}{2}+1$.

$\Rightarrow$ organisation utilisée pour les fichiers de petite taille.

Fichiers séquentiels triés

Une première amélioration consiste à **trier** le fichier sur sa clé d'accès. On peut alors effectuer une recherche par **dichotomie** :

1. On lit la page qui est "au milieu" du fichier.
2. Selon la valeur de la clé du premier enregistrement de cette page, on sait si l'article cherché est "avant" ou "après".
3. On recommence avec le demi-fichier où se trouve l'article recherché.

Coût de l'opération : $\log_2(n)$.

Plan du cours

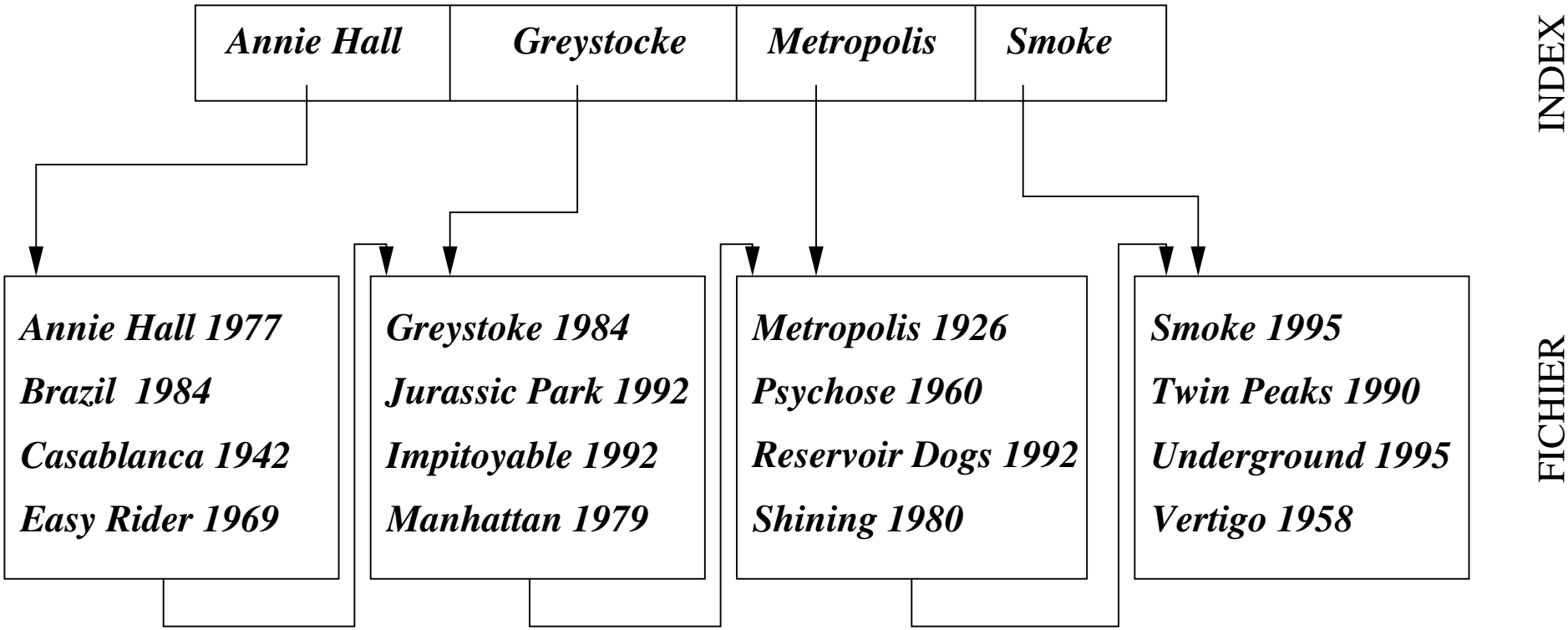
Ajout d'un index

L'opération de recherche peut encore être améliorée en utilisant un **index** sur un fichier **trié**. Le(s) champ(s) sur le(s)quel(s) le fichier est trié est appelé **clé**.
Un index est un *second fichier* possédant les caractéristiques suivantes :

1. Les articles sont des couples ($valeurdecl$, $adressedepage$)
2. Une occurence (v, b) dans un index signifie que le premier article dans la page b du fichier trié a pour valeur de clé v.

L'index est lui-même **trié** sur la valeur de clé.

Exemple


Annie Hall
Greystocke
Metropolis
Smoke
INDEX
Annie Hall 1977
Brazil 1984
Casablanca 1942
Easy Rider 1969
Greystoke 1984
Jurassic Park 1992
Impitoyable 1992
Manhattan 1979
Metropolis 1926
Psychose 1960
Reservoir Dogs 1992
Shining 1980
Smoke 1995
Twin Peaks 1990
Underground 1995
Vertigo 1958
FICHIER

Recherche avec un index

Un index permet d'accélérer les recherches : chercher l'article dont la valeur de clé est v_1.

1. On recherche dans l'index (séquentiellement ou - mieux - par dichotomie) la plus grande valeur v_2 telle que $v_2 < v_1$.
2. On lit la page désignée par l'adresse associée à v_2 dans l'index.
3. On cherche séquentiellement les articles de clé v_1 dans cette page.

Coût d'une recherche avec ou sans index

Soit un fichier F contenant 1000 pages. On suppose qu'une page d'index contient 100 entrées, et que l'index occupe donc 10 pages.

- F non trié et non indexé. Recherche séquentielle : **500 pages**.
- F trié et non indexé. Recherche dichotomique : $\log_2(1000)$=**10 pages**
- F trié et indexé. Recherche dichotomique sur l'index, puis lecture d'une page : $\log_2(10) + 1$= **5 pages**

Autres opérations : insertion

Etant donné un article A de clé v_1, on effectue d'abord une recherche pour savoir dans quelle page p il doit être placé. Deux cas de figure :

1. Il y une place libre dans p. Dans ce cas on réorganise le contenu de p pour placer A à la bonne place.
2. Il n'y a plus de place dans p. Plusieurs solutions, notamment : créer une **page de débordement**.

NB : il faut éventuellement réorganiser l'index.

Autres opérations : destructions et mises-à-jour

Relativement facile en général :

1. On recherche l'article.
2. On applique l'opération.

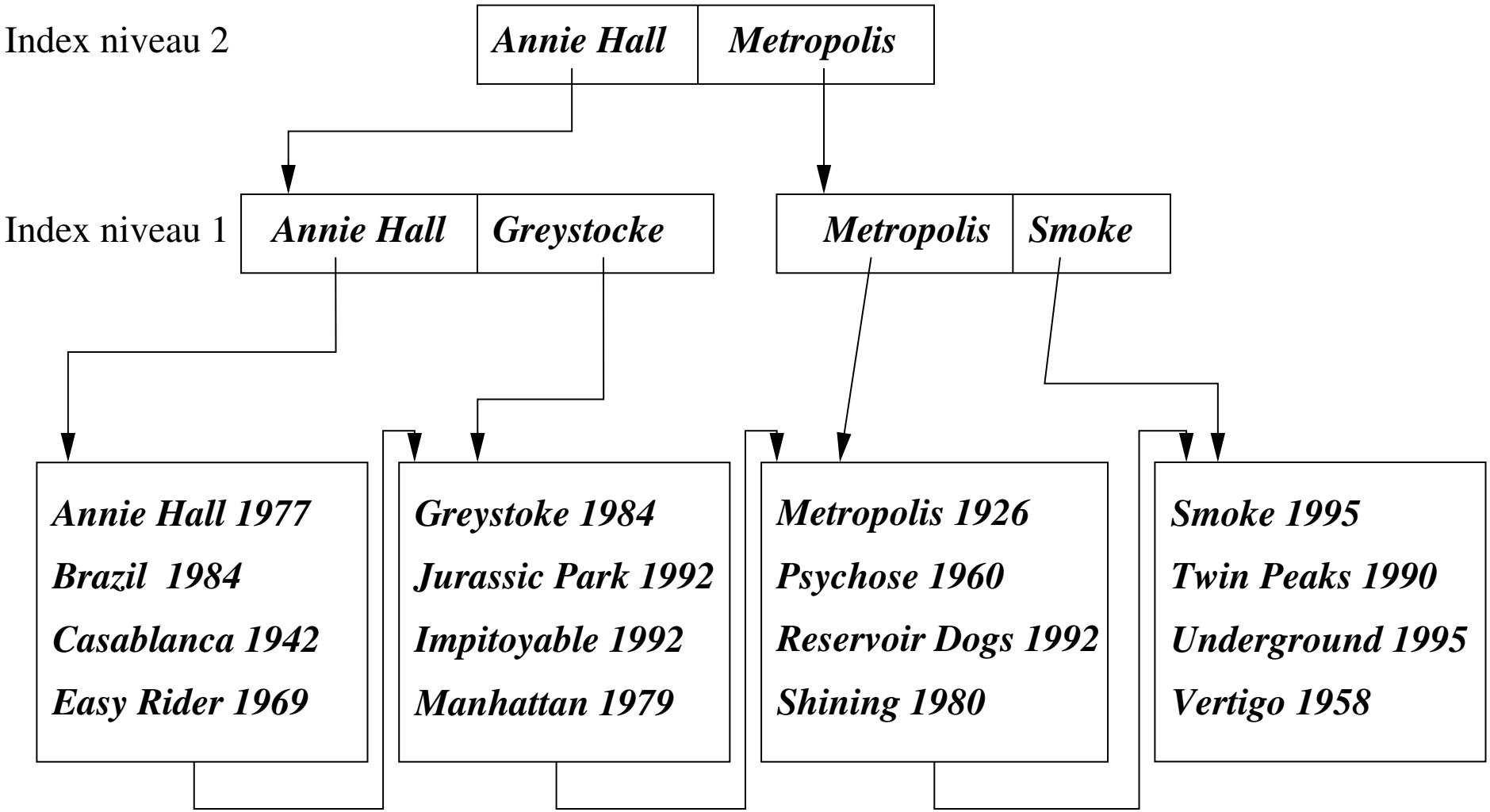
$\Rightarrow$ on peut avoir à réorganiser le fichier et/ou l'index, ce qui peut être couteux.

Séquentiel indexé : définition

Un index est un fichier qu'on peut à nouveau indexer :

1. On trie le fichier sur la clé.
2. On répartit les articles triés dans n pages, en laissant de la place libre dans chaque page.
3. On constitue un index à plusieurs niveaux sur la clé.

$\Rightarrow$ on obtient un **arbre** dont les feuilles constituent le fichier et les noeuds internes l'index.

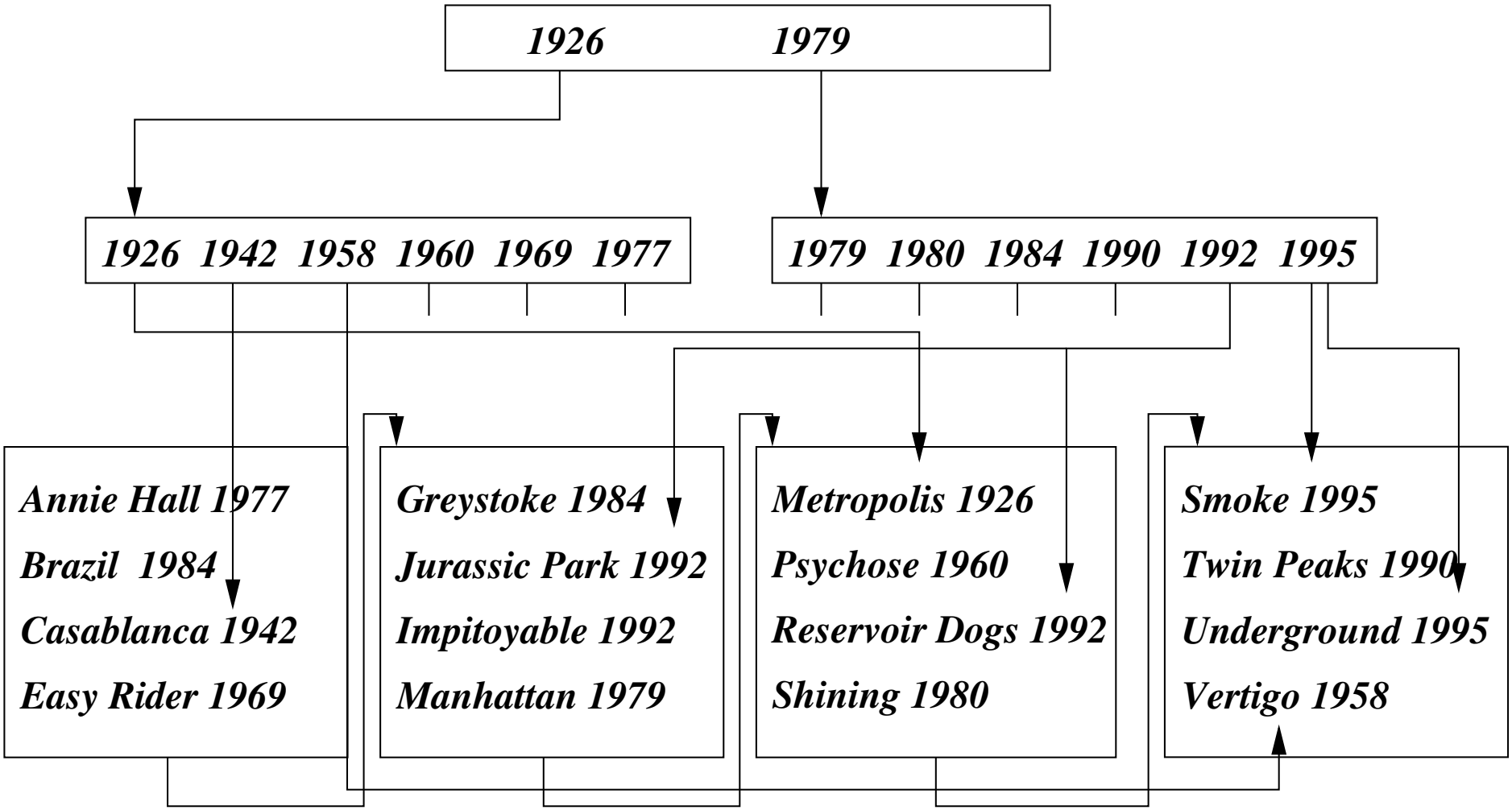

Index niveau 2
Annie Hall
Metropolis
Index niveau 1
Annie Hall
Greystocke
Metropolis
Smoke
Annie Hall 1977
Brazil 1984
Casablanca 1942
Easy Rider 1969
Greystoke 1984
Jurassic Park 1992
Impitoyable 1992
Manhattan 1979
Metropolis 1926
Psychose 1960
Reservoir Dogs 1992
Shining 1980
Smoke 1995
Twin Peaks 1990
Underground 1995
Vertigo 1958

Index dense et index non dense

L'index ci-dessus est **non dense** : **une seule valeur de clé** dans l'index pour l'ensemble des articles du fichier indexé F situés dans une même page.
Un index est **dense** ssi il existe une valeur de clé dans l'index pour chaque article dans le fichier F.
Remarques :

1. On ne peut créer un index non-dense que sur un fichier trié (et un seul index non-dense par fichier).
2. Un index non-dense est beaucoup moins volumineux qu'un index dense.

Exemple d'index dense



Inconvénients du séquentiel indexé

Organisation bien adaptée aux fichiers qui évoluent peu. En cas de grossissement :

1. Une page est trop pleine $\rightarrow$ on crée une page de débordement.
2. On peut aboutir à des chaînes de débordement importantes pour certaines pages.
3. Le temps de réponse peut se dégrader et dépend de l'article recherché

$\Rightarrow$ on a besoin d'une structure permettant une réorganisation dynamique sans dégradation de performances.

Plan du cours

Arbres-B

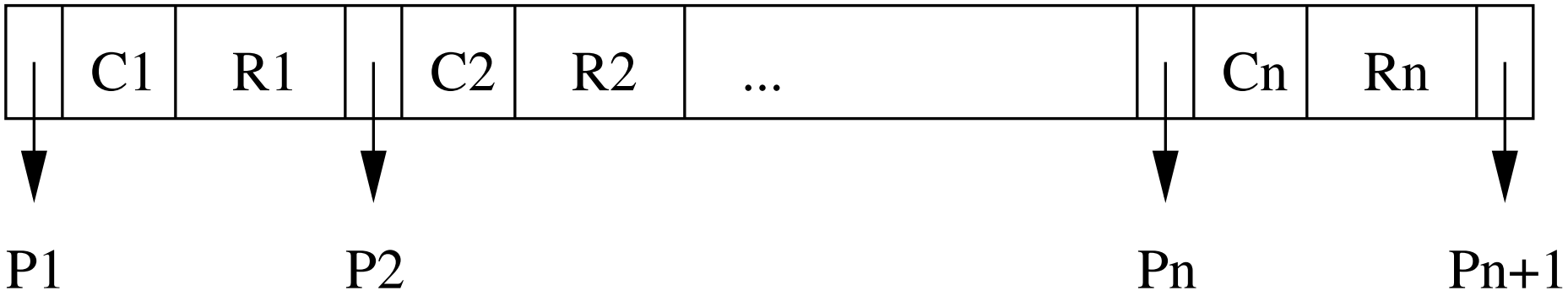
Un arbre-B (pour *balanced tree* ou **arbre équilibré**) est une structure arborescente dans laquelle tous les chemins de la racine aux feuilles ont même longueur.
Si le fichier grossit : la hiérarchie grossit **par le haut**.
L'arbre-B est utilisé dans **tous** les SGBD relationnels (avec des variantes).

Arbre-B : définition

Un arbre-B **d'ordre k** est un arbre équilibré tel que :

1. Chaque noeud est une page contenant au moins k et au plus $2k$ articles, $k \in N$.
2. Les articles dans un noeud sont triés sur la clé.
3. Chaque "père" est un index pour l'ensemble de ses fils/descendants.
4. Chaque noeud (sauf la racine) contient n articles a $n+1$ fils.
5. La racine a 0 ou au moins deux fils.

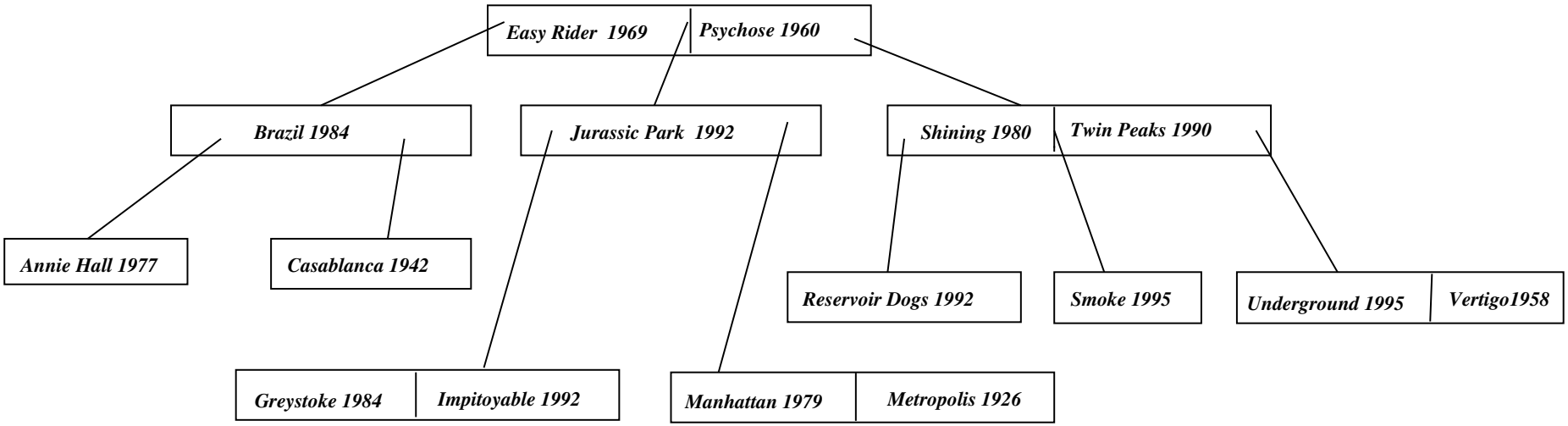
Structure d'un noeud dans un arbre-B d'ordre k



Les C_i sont les clés des articles, les R_i représentent le reste des attributs d'un article de clé C_i. Les P_i sont les pointeurs vers les noeuds fils dans l'index. NB : $k \leq n \leq 2k$.

Tous les articles référencés par P_i ont une valeur de clé x entre C_{i-1} et C_i (pour $i = 1 : x \leq C_1$ et $i = n+1 : x \geq C_n$)

Exemple d'un arbre-B



Recherche dans un arbre-B

Rechercher les articles de clé C. A partir de la racine, appliquer récursivement l'algorithme suivant :
Soit $C_1, \ldots C_n$ les valeurs de clés de la page courante.

1. Chercher C dans la page courante. Si C y est, accéder aux valeurs des autres champs, Fin.
2. Sinon, Si $C < C_1$ (ou $C > C_n$), on continue la recherche avec le noeud référencé par P_1 (ou P_{n+1}).
3. Sinon, il existe $i \in [1, k[$ tel que $C_i < C < C_{i+1}$, on continue avec la page référencée par le pointeur P_{i+1}.

Insertion dans un arbre-B d'ordre k

On recherche la feuille de l'arbre où l'article doit prendre place et on l'y insère. Si la page p déborde (elle contient $2k+1$ éléments) :

1. On alloue une nouvelle page p'.
2. On place les k premiers articles (ordonnés selon la clé) dans p et les k derniers dans p'.
3. On insère le $k+1^e$ article dans le père de p. Son pointeur gauche référence p, et son pointeur droit référence p'.
4. Si le père déborde à son tour, on continue comme en 1.

Destruction dans un arbre-B d'ordre k

Chercher la page p contenant l'article. Si c'est une feuille :

1. On détruit l'article.
2. S'il reste au moins k articles dans p, c'est fini.
3. Sinon :
   1. Si une feuille "soeur" contient plus de k articles, on effectue une permutation pour rééquilibrer les feuilles. Ex : destruction de *Smoke*.
   2. Sinon on "descend" un article du père dans p, et on réorganise le père. Ex : destruction de *ReservoirDogs*

Réorganisation de l'arbre

Supposons maintenant qu'on détruise un article dans un noeud interne. Il faut réorganiser :

1. On détruit l'article
2. On le remplace par l'article qui a la plus grande valeur de clé dans le sous-arbre gauche. Ex : destruction de *Psychose*, remplacé par *Metropolis*
3. On vient de retirer un article dans une feuille : si elle contient moins de k éléments, on procède comme indiqué précédemment.

$\Rightarrow$ toute destruction a un effet seulement **local**.

Quelques mesures pour l'arbre-B

Hauteur h d'un arbre-B d'ordre k contenant n articles :

$$\log_{2k+1}(n+1) \leq h \leq \log_{k+1}(\tfrac{n+1}{2})$$

1. $\log_{2k+1}(n+1)$: meilleure des cas (arbre balancé)
2. $\log_{k+1}(\frac{n+1}{2})$: pire des cas (insertion dans la même branche)

Exemple pour $k = 100$:

1. si $h = 2$, $n \leq 8 \times 10^6$
2. si $h = 3$, $n \leq 1,6 \times 10^9$

Les opérations d'accès coûtent au maximum h E/S (en moyenne $\geq h - 1$).

Variante de l'arbre-B

1. L'arbre B contient à la fois l'index et le fichier indexé.
2. Si la taille d'un article est grande, chaque noeud en contient peu, ce qui augmente la hauteur de l'arbre
3. On peut alors séparer l'index (arbre B) du fichier : stocker les articles dans un fichier F, remplacer l'article R_i dans les pages de l'arbre par un pointeur vers l'article dans F.

L'arbre B+

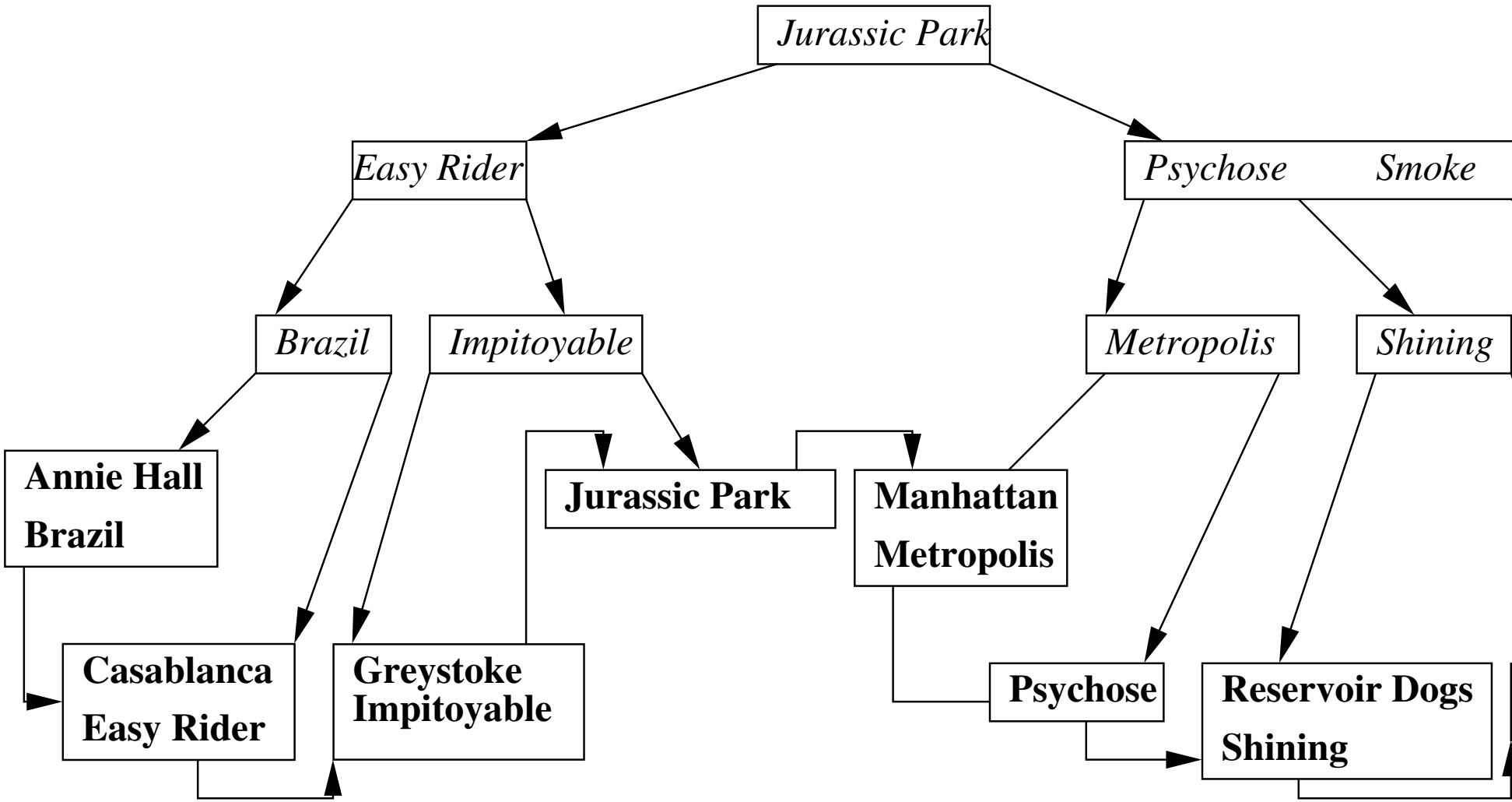
Inconvénient de l'arbre B (et de ses variantes):

- Les recherches sur des intervalles de valeurs sont complexes.

D'où l'arbre-B+ :

- *Seules les feuilles de l'arbre pointent sur les articles du fichier.*
- Toutes les clés sont dans les feuilles
- De plus ces feuilles sont chaînées entre elles.
- **Variante: les feuilles contiennent les articles (MySQL - moteur InnoDB)**

Exemple d'un arbre-B+



Plan du cours

Hachage

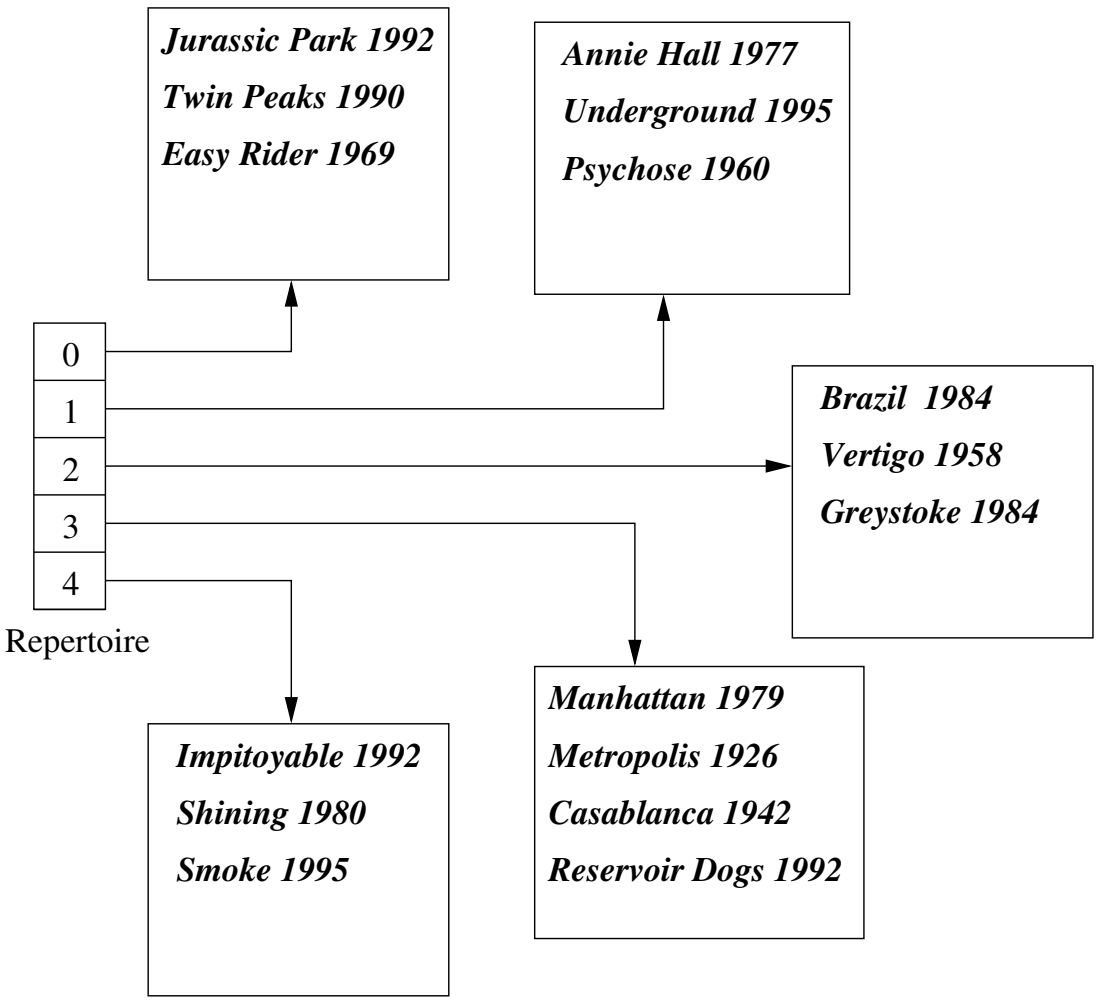
Accès direct à la page contenant l'article recherché :

1. On estime le nombre N de pages qu'il faut allouer au fichier.
2. **fonction de hachage** H : à toute valeur de la clé de domaine V associe un nombre entre 0 et $N-1$.
$$H : V \to \{0, 1, \ldots, N-1\}$$
3. On range dans la page de numéro i tous les articles dont la clé c est telle que $H(c) = i$.

Exemple : hachage sur le fichier *Films*

On suppose qu'une page contient 4 articles :

1. On alloue 5 pages au fichier.
2. On utilise une fonction de hachage H définie comme suit :
   1. Clé : nom d'un film, on ne s'intéresse qu'à l'initiale de ce nom.
   2. On numérote les lettres de l'alphabet de 1 à 26 :
      $No('a') = 1, No('m') = 13$, etc.
   3. Si l est une lettre de l'alphabet, $H(l) = MODULO(No(l), 5)$.


Jurassic Park 1992
Twin Peaks 1990
Easy Rider 1969
Annie Hall 1977
Underground 1995
Psychose 1960
0
1
2
3
4
Repertoire
Brazil 1984
Vertigo 1958
Greystoke 1984
Manhattan 1979
Metropolis 1926
Casablanca 1942
Reservoir Dogs 1992
Impitoyable 1992
Shining 1980
Smoke 1995

Remarques

1. Le nombre $H(c) = i$ n'est pas une adresse de page, mais l'indice d'une table ou "répertoire" R. $R(i)$ contient l'adresse de la page associée à i
2. Si ce répertoire ne tient pas en mémoire centrale, la recherche coûte plus cher.
3. Une propriété essentielle de H est que la distribution des valeurs obtenues soit uniforme dans $\{0, \ldots N-1\}$
4. Quand on alloue un nombre N de pages, il est préférable de prévoir un remplissage partiel(non uniformité, grossissement du fichier). On a choisi 5 pages alors que 4 (16 articles / 4) auraient suffi.

Hachage : recherche

Etant donné une valeur de clé v :

1. On calcule $i = H(v)$.
2. On consulte dans la case i du répertoire l'adresse de la page p.
3. On lit la page p et on y recherche l'article.

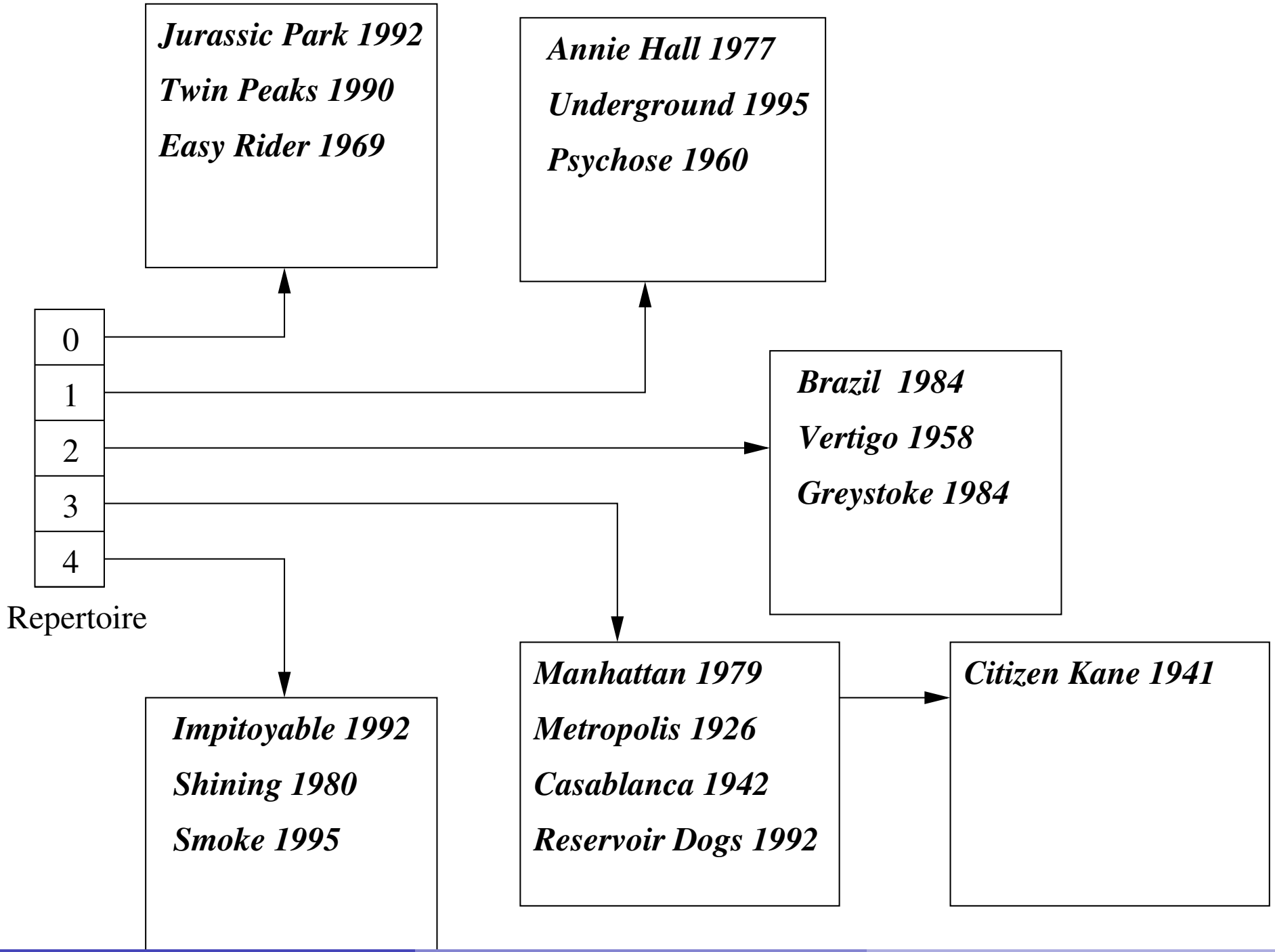
$\Rightarrow$ **donc une recherche ne coûte qu'une seule lecture**.

Hachage : insertion

Recherche par $H(c)$ la page p où placer A et l'y insèrer.
Si la page p est pleine, il faut :

1. Allouer une nouvelle page p' (de débordement).
2. Chaîner p' à p.
3. Insérer A dans p'.

$\Rightarrow$ lors d'une recherche, il faut donc en fait parcourir la liste des pages chaînées correspondant à une valeur de $H(v)$.
Moins la répartition est uniforme, plus il y aura de débordements


Jurassic Park 1992
Twin Peaks 1990
Easy Rider 1969
Annie Hall 1977
Underground 1995
Psychose 1960
0
1
2
3
4
Repertoire
Brazil 1984
Vertigo 1958
Greystoke 1984
Impitoyable 1992
Shining 1980
Smoke 1995
Manhattan 1979
Metropolis 1926
Casablanca 1942
Reservoir Dogs 1992
Citizen Kane 1941

Hachage : avantages et inconvénients

Intérêt du hachage :

1. **Très rapide**. Une seule E/S dans le meilleur des cas pour une recherche.
2. Le hachage, contrairement à un index, **n'occupe aucune place disque**.

En revanche :

1. Il faut penser à réorganiser les fichiers qui évoluent beaucoup.
2. **Les recherches par intervalle sont impossibles.**

Plan du cours

Comparatif

Organisation	Coût	Avantages	Inconvénients
Sequentiel	$\frac{n}{2}$	Simple	Très coûteux !
Indexé	$\log_2(n)$	Efficace Intervalles	Peu évolutive
Arbre-B	$\log_k(n)$	Efficace Intervalles	Traversée
Hachage	1+	Le plus efficace	Intervalles impossibles

Plan du cours

Pourquoi l'optimisation ?

Les langages de requêtes de haut niveau comme SQL sont **déclaratifs**.

L'utilisateur :

1. indique ce qu'il veut obtenir.
2. n'indique pas **comment** l'obtenir.

Donc le système doit faire le reste :

1. Déterminer le (ou les) chemin(s) d'accès aux données, les stratégies d'évaluation de la requête
2. **Choisir la meilleure stratégie** (ou une des meilleures . . .)

Pourquoi l'optimisation ?

Les langages de requêtes de haut niveau comme SQL sont **déclaratifs**. L'utilisateur :

1. indique ce qu'il veut obtenir.
2. n'indique pas **comment** l'obtenir.

Donc le système doit faire le reste :

1. Déterminer le (ou les) chemin(s) d'accès aux données, les stratégies d'évaluation de la requête
2. **Choisir la meilleure stratégie** (ou une des meilleures . . .)

Pourquoi l'optimisation ?

Les langages de requêtes de haut niveau comme SQL sont **déclaratifs**. L'utilisateur :

1. indique ce qu'il veut obtenir.
2. n'indique pas **comment** l'obtenir.

Donc le système doit faire le reste :

1. Déterminer le (ou les) chemin(s) d'accès aux données, les stratégies d'évaluation de la requête
2. **Choisir la meilleure stratégie** (ou une des meilleures . . .)

L'optimisation sur un exemple

Considérons le schéma :
$CINEMA(Cinéma, Adresse, Gérant)$
$SALLE(Cinéma, NoSalle, Capacité)$
$SEANCE(NoSalle, jour, heure - debut, film)$

Avec les hypothèses :

1. Il y a 300 n-uplets dans CINEMA, occupant 30 pages (10 cinémas/page).
2. Il y a 1200 n-uplets dans SALLE, occupant 120 pages(10 salles/page).
3. La mémoire centrale (buffer) ne contient qu'une seule page par relation.

Expression d'une requête

On considère la requête : *Cinémas ayant des salles de plus de 150 places*
En SQL, cette requête s'exprime de la manière suivante :

```
SELECT CINEMA.*
FROM   CINEMA, SALLE
WHERE  capacite > 150
AND    CINEMA.cinema = SALLE.cinema
```

En algèbre relationnelle

Traduit en algèbre, on a plusieurs possibilités. En voici deux :

1. $\pi_{CINEMA.*}(\sigma_{Capacité>150}(CINEMA \bowtie SALLE))$
2. $\pi_{CINEMA.*}(CINEMA \bowtie \sigma_{Capacité>150}(SALLE))$

Soit une jointure suivie d'une sélection, ou l'inverse.

Evaluation des coûts

On suppose qu'il n'y a que 5 % de salles de plus de 150 places (haute séléctivité) et que les résultats intérmédiaires d'une opération et le résultat final sont écrits sur disque (10 n-uplets par page).

1. Jointure d'abord :
   - **Jointure** : on lit 3 600 pages (120 x 30);
   - On écrit le résultat intermédiaire (120 pages);
   - **Sélection** : on relit le résultat et comme on projète sur tous les attributs de CINEMA, on obtient 5 % de 120 pages, soit 6 pages;
   - Nombre d'E/S : 3 600E + 120 x 2E/S + 6S = 3 846.
2. Sélection d'abord :
   - **Sélection** : on lit 120 pages (salles) et on obtient (écrit) 6 pages;
   - **Jointure** : on lit 180 pages (6x30) et on obtient 6 pages;
   - Nombre d'E/S : 120E + 6S + 180E + 6S = 312.

$\Rightarrow$ la deuxième stratégie est de loin la meilleure !

Evaluation des coûts

On suppose qu'il n'y a que 5 % de salles de plus de 150 places (haute séléctivité) et que les résultats intérmédiaires d'une opération et le résultat final sont écrits sur disque (10 n-uplets par page).

1. Jointure d'abord :
   - **Jointure** : on lit 3 600 pages (120 x 30);
   - On écrit le résultat intermédiaire (120 pages);
   - **Sélection** : on relit le résultat et comme on projète sur tous les attributs de CINEMA, on obtient 5 % de 120 pages, soit 6 pages;
   - Nombre d'E/S : 3 600E + 120 x 2E/S + 6S = 3 846.
2. Sélection d'abord :
   - **Sélection** : on lit 120 pages (salles) et on obtient (écrit) 6 pages;
   - **Jointure** : on lit 180 pages (6x30) et on obtient 6 pages;
   - Nombre d'E/S : 120E + 6S + 180E + 6S = 312.

$\Rightarrow$ la deuxième stratégie est de loin la meilleure !

Evaluation des coûts

On suppose qu'il n'y a que 5 % de salles de plus de 150 places (haute séléctivité) et que les résultats intérmédiaires d'une opération et le résultat final sont écrits sur disque (10 n-uplets par page).

1. Jointure d'abord :
   - **Jointure** : on lit 3 600 pages (120 x 30);
   - On écrit le résultat intermédiaire (120 pages);
   - **Sélection** : on relit le résultat et comme on projète sur tous les attributs de CINEMA, on obtient 5 % de 120 pages, soit 6 pages;
   - Nombre d'E/S : 3 600E + 120 x 2E/S + 6S = 3 846.
2. Sélection d'abord :
   - **Sélection** : on lit 120 pages (salles) et on obtient (écrit) 6 pages;
   - **Jointure** : on lit 180 pages (6x30) et on obtient 6 pages;
   - Nombre d'E/S : 120E + 6S + 180E + 6S = 312.

$\Rightarrow$ la deuxième stratégie est de loin la meilleure !

Optimisation de requêtes : premières conclusions

1. Il faut **traduire** une requête exprimée avec un langage déclaratif en une suite d'opérations (similaires aux opérateurs de l'algèbre relationnelle).
2. En fonction :
   - des coûts de chaque opération,
   - des caractéristiques de la base,
   - des algorithmes utilisés,

   On cherche à estimer la meilleure stratégie.
3. On obtient le **plan d'exécution** de la requête. Il n'y a plus qu'à le traiter au niveau physique.

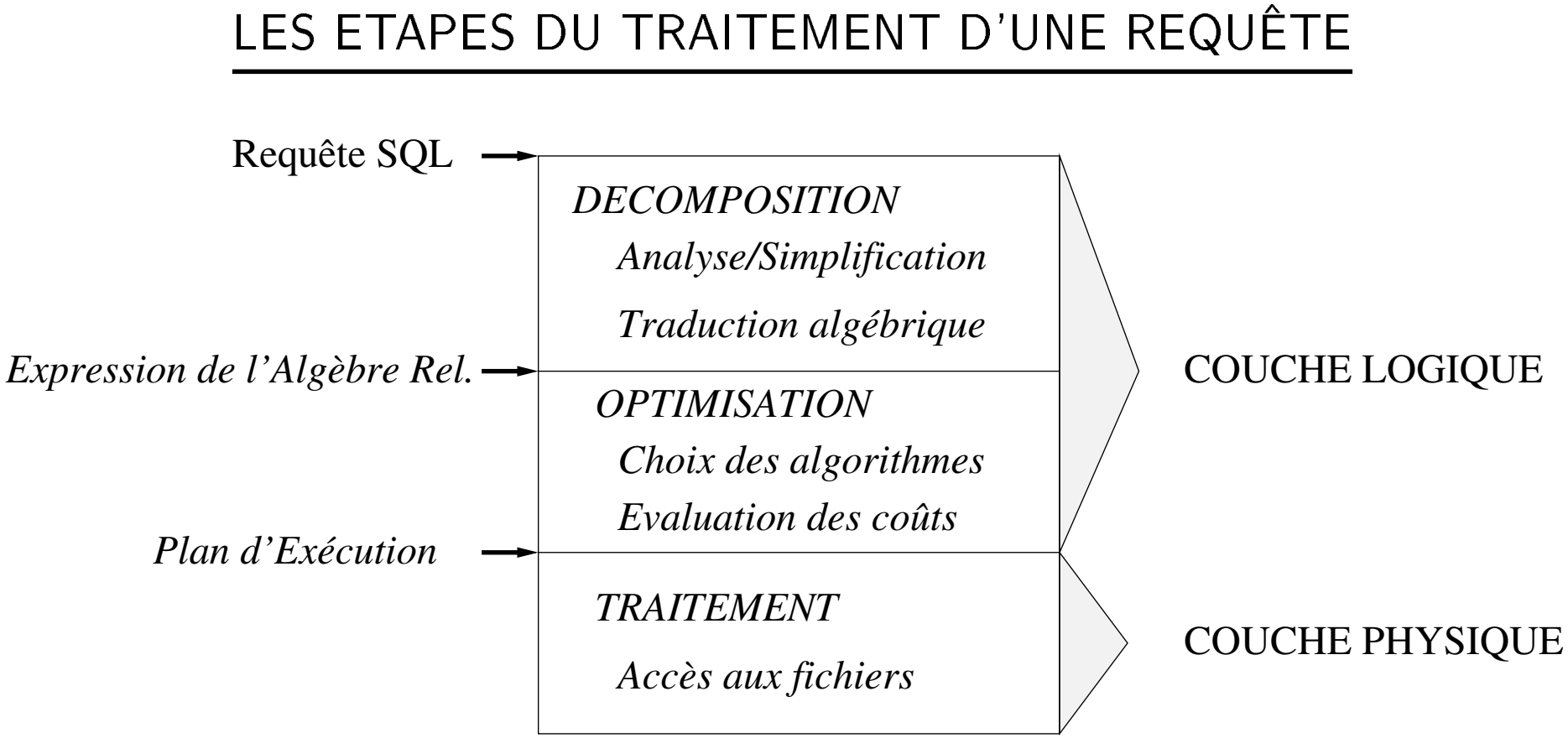
Les paramètres de l'optimisation

Comme on l'a vu sur l'exemple, l'optimisation s'appuie sur :

1. Des **règles de réécriture** des expressions de l'algèbre.
2. Des connaissances sur **l'organisation physique** de la base (index)
3. Des **statistiques** sur les caractéristiques de la base (taille des relations par exemple).

Un **modèle de coût** permet de classer les différentes stratégies envisagées

Architecture d'un SGBD et optimisation



Plan du cours

Analyse syntaxique

On vérifie la validité (syntaxique) de la requête.

1. Contrôle de la structure grammaticale.
2. Vérification de l'existence des relations et des noms d'attributs.

$\Rightarrow$ On utilise le "dictionnaire" de la base qui contient le schéma.

Analyse, simplification et normalisation

D'autres types de transformations avant optimisation :

1. **Analyse sémantique** pour la détection d'incohérences.
   Exemple : "*NoSalle* = 11 *AND NoSalle* = 12"
2. **Simplification** de clauses inutilement complexes. Exemple :
   (*A OR NOT B*) *AND B* est équivalent à *A AND B*.
3. **Normalisation** de la requête.
   Exemple : transformation des conditions en forme normale conjonctive) et décomposition en *blocs SELECT-FROM-WHERE* pour faciliter la traduction algébrique.

Analyse, simplification et normalisation

D'autres types de transformations avant optimisation :

1. **Analyse sémantique** pour la détection d'incohérences.
   Exemple : "*NoSalle* = 11 *AND NoSalle* = 12"
2. **Simplification** de clauses inutilement complexes. Exemple :
   (*A OR NOT B*) *AND B* est équivalent à *A AND B*.
3. **Normalisation** de la requête.
   Exemple : transformation des conditions en forme normale conjonctive) et décomposition en *blocs SELECT-FROM-WHERE* pour faciliter la traduction algébrique.

Analyse, simplification et normalisation

D'autres types de transformations avant optimisation :

1. **Analyse sémantique** pour la détection d'incohérences.
   Exemple : "*NoSalle* = 11 *AND NoSalle* = 12"
2. **Simplification** de clauses inutilement complexes. Exemple :
   (*A OR NOT B*) *AND B* est équivalent à *A AND B*.
3. **Normalisation** de la requête.
   Exemple : transformation des conditions en forme normale conjonctive) et décomposition en *blocs SELECT-FROM-WHERE* pour faciliter la traduction algébrique.

Traduction algébrique

Déterminer l'expression algébrique équivalente à la requête :

1. Arguments du SELECT :
2. Arguments du WHERE :

On obtient une expression algébrique qui peut être représentée par un arbre de requête.

Traduction algébrique

Déterminer l'expression algébrique équivalente à la requête :

1. Arguments du SELECT : **projections**
2. Arguments du WHERE :

On obtient une expression algébrique qui peut être représentée par un **arbre de requête**.

Traduction algébrique

Déterminer l'expression algébrique équivalente à la requête :

1. Arguments du SELECT : **projections**
2. Arguments du WHERE :
   - *NomAttr*1 = *NomAttr*2 correspond en général à une **jointure**
   - *NomAttr* = *constante* à une **sélection**.

On obtient une expression algébrique qui peut être représentée par un **arbre de requête**.

Traduction algébrique

Déterminer l'expression algébrique équivalente à la requête :

1. Arguments du SELECT : **projections**
2. Arguments du WHERE :
   - *NomAttr*1 = *NomAttr*2 correspond en général à une **jointure**
   - *NomAttr* = *constante* à une **sélection**.

On obtient une expression algébrique qui peut être représentée par un **arbre de requête**.

Traduction algébrique : exemple

Considérons l'exemple suivant :

Quels films passent au REX à 20 heures ?

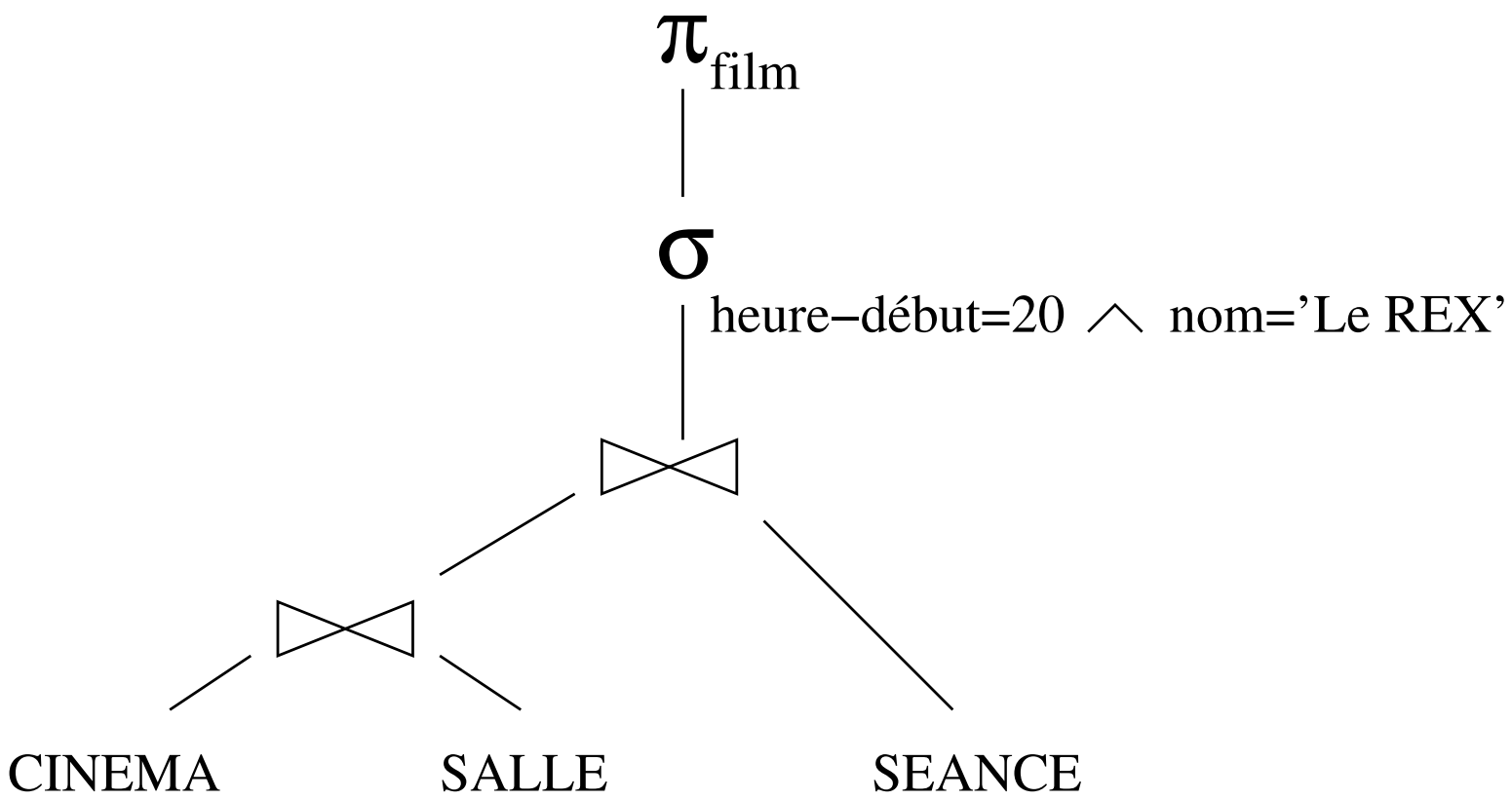
```
SELECT film
FROM   CINEMA, SALLE, SEANCE
WHERE  CINEMA.nom-cinema = 'Le Rex'
AND    SEANCE.heure-debut = 20
AND    CINEMA.nom-cinema = SALLE.nom-cinema
AND    SALLE.salle = SEANCE.salle
```

Expression algébrique et arbre de requête

$$\pi_{film}(\sigma_{Nom='Le\ Rex' \wedge heure-début=20}((CINEMA \bowtie SALLE) \bowtie SEANCE))$$

Expression algébrique et arbre de requête

$$\pi_{film}(\sigma_{Nom='Le\ Rex' \wedge heure-début=20}((CINEMA \bowtie SALLE) \bowtie SEANCE))$$



Restructuration

Il y a plusieurs expressions **équivalentes** pour une même requête.
ROLE DE L'OPTIMISEUR

1. Trouver les expressions équivalentes à une requête.
2. Les évaluer et choisir la "meilleure".

On convertit une expression en une expression équivalente en employant des **règles de réécriture**.

Règles de réécriture

Il en existe beaucoup. En voici huit parmi les plus importantes :

1. **Commutativité des jointures :**
   $R \bowtie S \equiv S \bowtie R$
2. **Associativité des jointures :**
   $(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$
3. **Regroupement des sélections :**
   $\sigma_{A='a' \wedge B='b'}(R) \equiv \sigma_{A='a'}(\sigma_{B='b'}(R))$
4. **Commutativité de la sélection et de la projection**
   $\pi_{A_1,A_2,\ldots A_p}(\sigma_{A_i='a}(R)) \equiv \sigma_{A_i='a}(\pi_{A_1,A_2,\ldots A_p}(R)),\ i \in \{1,\ldots,p\}$

Règles de réécriture

Il en existe beaucoup. En voici huit parmi les plus importantes :

1. **Commutativité des jointures :**
   $R \bowtie S \equiv S \bowtie R$
2. **Associativité des jointures :**
   $(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$
3. **Regroupement des sélections :**
   $\sigma_{A='a' \wedge B='b'}(R) \equiv \sigma_{A='a'}(\sigma_{B='b'}(R))$
4. **Commutativité de la sélection et de la projection**
   $\pi_{A_1,A_2,\ldots A_p}(\sigma_{A_i='a}(R)) \equiv \sigma_{A_i='a}(\pi_{A_1,A_2,\ldots A_p}(R)),\ i \in \{1,\ldots,p\}$

Règles de réécriture

Il en existe beaucoup. En voici huit parmi les plus importantes :

1. **Commutativité des jointures :**
   $R \bowtie S \equiv S \bowtie R$
2. **Associativité des jointures :**
   $(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$
3. **Regroupement des sélections :**
   $\sigma_{A='a' \wedge B='b'}(R) \equiv \sigma_{A='a'}(\sigma_{B='b'}(R))$
4. **Commutativité de la sélection et de la projection**
   $\pi_{A_1,A_2,\ldots A_p}(\sigma_{A_i='a}(R)) \equiv \sigma_{A_i='a}(\pi_{A_1,A_2,\ldots A_p}(R)),\ i \in \{1,\ldots,p\}$

Règles de réécriture

Il en existe beaucoup. En voici huit parmi les plus importantes :

1. **Commutativité des jointures :**
   $R \bowtie S \equiv S \bowtie R$
2. **Associativité des jointures :**
   $(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$
3. **Regroupement des sélections :**
   $\sigma_{A='a' \wedge B='b'}(R) \equiv \sigma_{A='a'}(\sigma_{B='b'}(R))$
4. **Commutativité de la sélection et de la projection**
   $\pi_{A_1,A_2,\ldots A_p}(\sigma_{A_i='a}(R)) \equiv \sigma_{A_i='a}(\pi_{A_1,A_2,\ldots A_p}(R)),\ i \in \{1,\ldots,p\}$

Règles de réécriture

5 **Commutativité de la sélection et de la jointure.**
$\sigma_{A='a'}(R(\ldots A\ldots) \bowtie S) \equiv \sigma_{A='a'}(R) \bowtie S$

6 **Distributivité de la sélection sur l'union.**
$\sigma_{A='a'}(R \cup S) \equiv \sigma_{A='a'}(R) \cup \sigma_{A='a'}(S)$
NB : valable aussi pour la différence.

7 **Commutativité de la projection et de la jointure**
$\pi_{A_1\ldots A_p B_1\ldots B_q}(R \bowtie_{A_i=B_j} S) \equiv$
$\pi_{A_1\ldots A_p}(R) \bowtie_{A_i=B_j} \pi_{B_1\ldots B_q}(S)$,
$(i \in \{1,\ldots,p\}, j \in \{1,\ldots,q\})$

8 **Distributivité de la projection sur l'union**
$\pi_{A_1 A_2\ldots A_p}(R \cup S) \equiv \pi_{A_1 A_2\ldots A_p}(R) \cup \pi_{A_1 A_2\ldots A_p}(S)$

Règles de réécriture

5. **Commutativité de la sélection et de la jointure.**
   $\sigma_{A='a'}(R(\ldots A\ldots) \bowtie S) \equiv \sigma_{A='a'}(R) \bowtie S$
6. **Distributivité de la sélection sur l'union.**
   $\sigma_{A='a'}(R \cup S) \equiv \sigma_{A='a'}(R) \cup \sigma_{A='a'}(S)$
   NB : valable aussi pour la différence.
7. **Commutativité de la projection et de la jointure**
   $\pi_{A_1\ldots A_p B_1\ldots B_q}(R \bowtie_{A_i=B_j} S) \equiv$
   $\pi_{A_1\ldots A_p}(R) \bowtie_{A_i=B_j} \pi_{B_1\ldots B_q}(S)$,
   $(i \in \{1,\ldots,p\}, j \in \{1,\ldots,q\})$
8. **Distributivité de la projection sur l'union**
   $\pi_{A_1 A_2\ldots A_p}(R \cup S) \equiv \pi_{A_1 A_2\ldots A_p}(R) \cup \pi_{A_1 A_2\ldots A_p}(S)$

Règles de réécriture

5. **Commutativité de la sélection et de la jointure.**
   $\sigma_{A='a'}(R(\ldots A\ldots) \bowtie S) \equiv \sigma_{A='a'}(R) \bowtie S$
6. **Distributivité de la sélection sur l'union.**
   $\sigma_{A='a'}(R \cup S) \equiv \sigma_{A='a'}(R) \cup \sigma_{A='a'}(S)$
   NB : valable aussi pour la différence.
7. **Commutativité de la projection et de la jointure**
   $\pi_{A_1\ldots A_p B_1\ldots B_q}(R \bowtie_{A_i=B_j} S) \equiv$
   $\pi_{A_1\ldots A_p}(R) \bowtie_{A_i=B_j} \pi_{B_1\ldots B_q}(S)$,
   $(i \in \{1,\ldots,p\}, j \in \{1,\ldots,q\})$
8. **Distributivité de la projection sur l'union**
   $\pi_{A_1 A_2\ldots A_p}(R \cup S) \equiv \pi_{A_1 A_2\ldots A_p}(R) \cup \pi_{A_1 A_2\ldots A_p}(S)$

Règles de réécriture

5. **Commutativité de la sélection et de la jointure.**
   $\sigma_{A='a'}(R(\ldots A\ldots) \bowtie S) \equiv \sigma_{A='a'}(R) \bowtie S$
6. **Distributivité de la sélection sur l'union.**
   $\sigma_{A='a'}(R \cup S) \equiv \sigma_{A='a'}(R) \cup \sigma_{A='a'}(S)$
   NB : valable aussi pour la différence.
7. **Commutativité de la projection et de la jointure**
   $\pi_{A_1\ldots A_p B_1\ldots B_q}(R \bowtie_{A_i=B_j} S) \equiv$
   $\pi_{A_1\ldots A_p}(R) \bowtie_{A_i=B_j} \pi_{B_1\ldots B_q}(S)$,
   $(i \in \{1,\ldots,p\}, j \in \{1,\ldots,q\})$
8. **Distributivité de la projection sur l'union**
   $\pi_{A_1 A_2\ldots A_p}(R \cup S) \equiv \pi_{A_1 A_2\ldots A_p}(R) \cup \pi_{A_1 A_2\ldots A_p}(S)$

Exemple d'un algorithme de restructuration

Voici un algorithme basé sur les propriétés précédentes.

1. Séparer les sélections avec plusieurs prédicats en plusieurs sélections à un prédicat (**règle 3**).
2. Descendre les sélections le plus bas possible dans l'arbre (**règles 4, 5, 6**).
3. Regrouper les sélections sur une même relation (**règle 3**).
4. Descendre les projections le plus bas possible (**règles 7 et 8**).
5. Regrouper les projections sur une même relation.

Exemple d'un algorithme de restructuration

Voici un algorithme basé sur les propriétés précédentes.

1. Séparer les sélections avec plusieurs prédicats en plusieurs sélections à un prédicat (**règle 3**).
2. Descendre les sélections le plus bas possible dans l'arbre (**règles 4, 5, 6**).
3. Regrouper les sélections sur une même relation (**règle 3**).
4. Descendre les projections le plus bas possible (**règles 7 et 8**).
5. Regrouper les projections sur une même relation.

Exemple d'un algorithme de restructuration

Voici un algorithme basé sur les propriétés précédentes.

1. Séparer les sélections avec plusieurs prédicats en plusieurs sélections à un prédicat (**règle 3**).
2. Descendre les sélections le plus bas possible dans l'arbre (**règles 4, 5, 6**).
3. Regrouper les sélections sur une même relation (**règle 3**).
4. Descendre les projections le plus bas possible (**règles 7 et 8**).
5. Regrouper les projections sur une même relation.

Exemple d'un algorithme de restructuration

Voici un algorithme basé sur les propriétés précédentes.

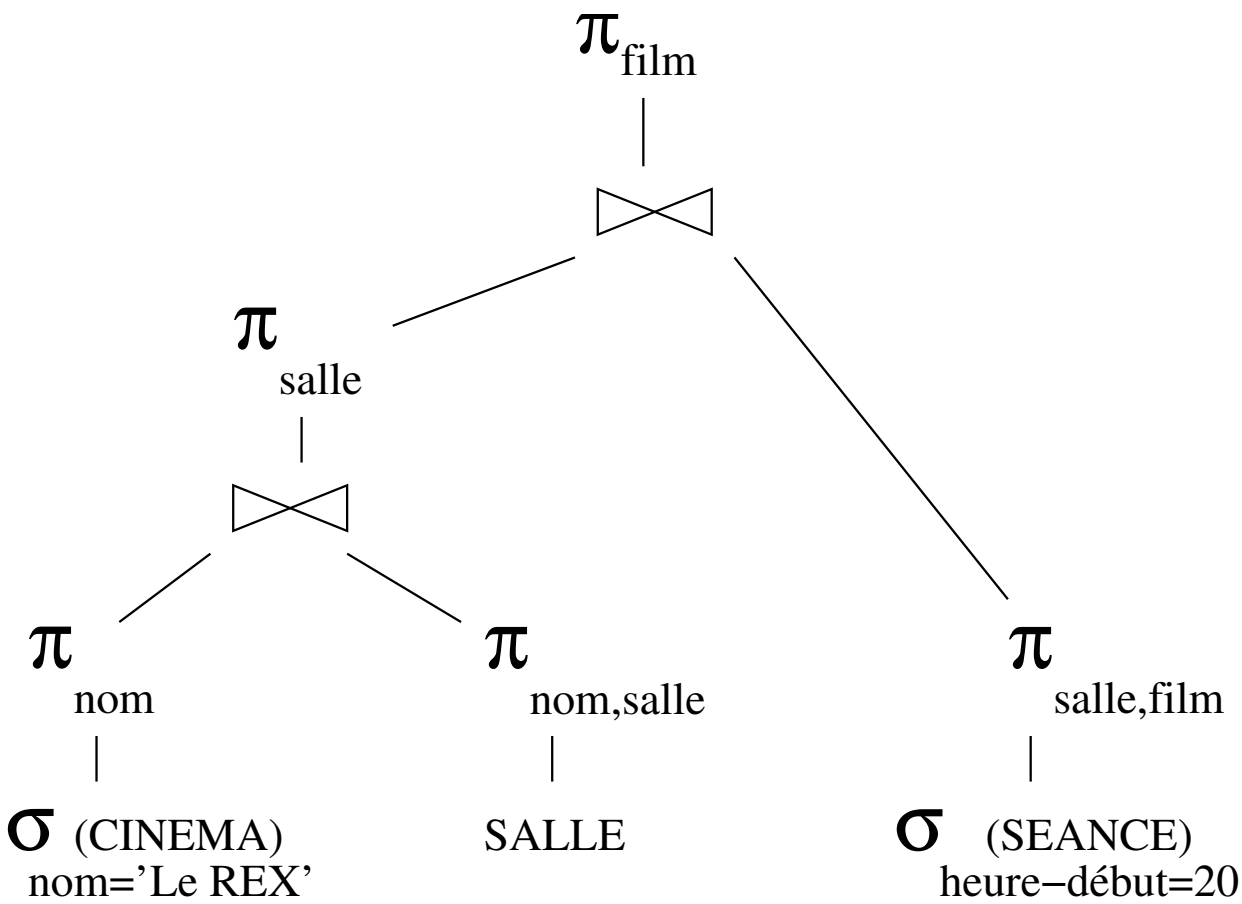
1. Séparer les sélections avec plusieurs prédicats en plusieurs sélections à un prédicat (**règle 3**).
2. Descendre les sélections le plus bas possible dans l'arbre (**règles 4, 5, 6**).
3. Regrouper les sélections sur une même relation (**règle 3**).
4. Descendre les projections le plus bas possible (**règles 7 et 8**).
5. Regrouper les projections sur une même relation.

Exemple d'un algorithme de restructuration

Voici un algorithme basé sur les propriétés précédentes.

1. Séparer les sélections avec plusieurs prédicats en plusieurs sélections à un prédicat (**règle 3**).
2. Descendre les sélections le plus bas possible dans l'arbre (**règles 4, 5, 6**).
3. Regrouper les sélections sur une même relation (**règle 3**).
4. Descendre les projections le plus bas possible (**règles 7 et 8**).
5. Regrouper les projections sur une même relation.

Arbre de requête après restructuration



Quelques remarques sur l'algorithme précédent

L'idée de base est de réduire le plus tôt possible (en bas de l'arbre) la taille des relations manipulées. Donc :

1. On effectue les sélections d'abord car on considère que c'est l'opérateur le plus "réducteur".
2. On élimine dès que possible les attributs inutiles par projection.
3. Enfin on effectue les jointures.

Le plan obtenu est-il TOUJOURS optimal (pour toutes les bases de données) ?

Quelques remarques sur l'algorithme précédent

L'idée de base est de réduire le plus tôt possible (en bas de l'arbre) la taille des relations manipulées. Donc :

1. On effectue les sélections d'abord car on considère que c'est l'opérateur le plus "réducteur".
2. On élimine dès que possible les attributs inutiles par projection.
3. Enfin on effectue les jointures.

Le plan obtenu est-il TOUJOURS optimal (pour toutes les bases de données) ?

Quelques remarques sur l'algorithme précédent

L'idée de base est de réduire le plus tôt possible (en bas de l'arbre) la taille des relations manipulées. Donc :

1. On effectue les sélections d'abord car on considère que c'est l'opérateur le plus "réducteur".
2. On élimine dès que possible les attributs inutiles par projection.
3. Enfin on effectue les jointures.

Le plan obtenu est-il TOUJOURS optimal (pour toutes les bases de données) ?

Quelques remarques sur l'algorithme précédent

L'idée de base est de réduire le plus tôt possible (en bas de l'arbre) la taille des relations manipulées. Donc :

1. On effectue les sélections d'abord car on considère que c'est l'opérateur le plus "réducteur".
2. On élimine dès que possible les attributs inutiles par projection.
3. Enfin on effectue les jointures.

Le plan obtenu est-il TOUJOURS optimal (pour toutes les bases de données) ?

Quelques remarques sur l'algorithme précédent

L'idée de base est de réduire le plus tôt possible (en bas de l'arbre) la taille des relations manipulées. Donc :

1. On effectue les sélections d'abord car on considère que c'est l'opérateur le plus "réducteur".
2. On élimine dès que possible les attributs inutiles par projection.
3. Enfin on effectue les jointures.

Le plan obtenu est-il TOUJOURS optimal (pour toutes les bases de données) ?
La réponse est NON!

Un contre-exemple

Quels sont les films visibles entre 14h et 22h?

Voici deux expressions de l'algèbre, dont l'une "optimisée" :

1. $\pi_{film}(\sigma_{h-début>14 \wedge h-début<22}(FILM \bowtie SEANCE))$
2. $\pi_{film}(FILM \bowtie \sigma_{h-début>14 \wedge h-début<22}(SEANCE))$

La relation FILM occupe 8 pages, la relation SEANCE 50.

Contre-exemple : évaluation des coûts

Hypothèses :

- 90 % des séances ont lieu entre 14 et 22 heures,
- seulement 20 % des films de la table FILM existent dans la table SEANCE.

1. **Jointure** : on lit 400 pages (50 x 8) et on aboutit à 10 pages (20% de 50 pages).
   Sélection : on se ramène à 9 pages (90%).
   Nombre d'E/S : 400E + 10x2E/S + 9S = 429E/S.
2. **Sélection** : on lit 50 pages et on aboutit à 45 pages (90%).
   Jointure : on lit 360 (45x8) pages et on aboutit à 9 pages (20% de 45).
   Nombre d'E/S : 50E + 45S + 360E +9S = 464E/S.

$\Rightarrow$ la première stratégie est la meilleure !
Ici la jointure est plus sélective que la sélection (cas rare).

Contre-exemple : évaluation des coûts

Hypothèses :

- 90 % des séances ont lieu entre 14 et 22 heures,
- seulement 20 % des films de la table FILM existent dans la table SEANCE.

1. **Jointure** : on lit 400 pages (50 x 8) et on aboutit à 10 pages (20% de 50 pages).
   Sélection : on se ramène à 9 pages (90%).
   Nombre d'E/S : 400E + 10x2E/S + 9S = 429E/S.
2. **Sélection** : on lit 50 pages et on aboutit à 45 pages (90%).
   Jointure : on lit 360 (45x8) pages et on aboutit à 9 pages (20% de 45).
   Nombre d'E/S : 50E + 45S + 360E +9S = 464E/S.

⇒ la première stratégie est la meilleure !
Ici la jointure est plus sélective que la sélection (cas rare).

Contre-exemple : évaluation des coûts

Hypothèses :

- 90 % des séances ont lieu entre 14 et 22 heures,
- seulement 20 % des films de la table FILM existent dans la table SEANCE.

1. **Jointure** : on lit 400 pages (50 x 8) et on aboutit à 10 pages (20% de 50 pages).
   Sélection : on se ramène à 9 pages (90%).
   Nombre d'E/S : 400E + 10x2E/S + 9S = 429E/S.
2. **Sélection** : on lit 50 pages et on aboutit à 45 pages (90%).
   Jointure : on lit 360 (45x8) pages et on aboutit à 9 pages (20% de 45).
   Nombre d'E/S : 50E + 45S + 360E +9S = 464E/S.

$\Rightarrow$ la première stratégie est la meilleure !
Ici la jointure est plus sélective que la sélection (cas rare).

Contre-exemple : évaluation des coûts

Hypothèses :

- 90 % des séances ont lieu entre 14 et 22 heures,
- seulement 20 % des films de la table FILM existent dans la table SEANCE.

1. **Jointure** : on lit 400 pages (50 x 8) et on aboutit à 10 pages (20% de 50 pages).
   Sélection : on se ramène à 9 pages (90%).
   Nombre d'E/S : 400E + 10x2E/S + 9S = 429E/S.
2. **Sélection** : on lit 50 pages et on aboutit à 45 pages (90%).
   Jointure : on lit 360 (45x8) pages et on aboutit à 9 pages (20% de 45).
   Nombre d'E/S : 50E + 45S + 360E +9S = 464E/S.

$\Rightarrow$ la première stratégie est la meilleure !
Ici la jointure est plus sélective que la sélection (cas rare).

Traduction algébrique : conclusion

La réécriture algébrique est nécessaire mais pas suffisante. L'optimiseur tient également compte :

1. Des chemins d'accès aux données (index).
2. Des différents algorithmes implantant une même operation algébrique (jointures).
3. De propriétés statistiques de la base.

Plan du cours

Les chemins d'accès

Ils dépendent des organisations de fichiers existantes :

1. Balayage séquentiel
2. Parcours d'index
3. Accès par hachage

Attention ! Dans certains cas un balayage peut être préférable à un parcours d'index.

Algorithmes pour les opérations algébriques

On a généralement le choix entre plusieurs algorithmes pour effectuer une opération.
L'opération la plus étudiée est la JOINTURE (pourquoi ?) :

1. Boucle imbriquée sans index,
2. Tri-fusion,
3. Jointure par hachage,
4. Boucles imbriquées avec accès à une des relations par index.

Le choix dépend essentiellement - mais pas uniquement - du chemin d'accès disponible.

Algorithmes de jointure sans index

En l'absence d'index, les principaux algorithmes sont :

1. Boucles imbriquées
2. Tri-fusion
3. Jointure par hachage

Jointure par boucles imbriquées

A utiliser quand les tailles des relations sont petites.
Soit les deux relations R et S :
Algorithme :

- Pour chaque page r de $\mathcal{R}$ (table directrice)
  - Pour chaque page s de $\mathcal{S}$
    Joindre en mémoire les tuples de r avec ceux de s

NestedLoopJoin : Exemple

Cours

nfe106	Ingénierie des ...
nfp107	Systèmes de ...
nfe204	Base de données...
nfe205	Base de données...
nsy122	Analyse des imag...
nfa011	Approfondissem...
nfe102	Infrastructures...
nsy218	Vision par ordina...
nsy219	Vision par ordina...

Auditeurs

Id1	nfe205
Id2	nsy218
Id3	nfe106
Id2	nfe204
Id10	nfa011
Id2	nsy122
Id5	nsy218
Id3	nsy218
Id4	nfe106
Id2	nfp107
Id7	nfe204
Id2	nfe205
Id5	nfe205
Id6	nfp107
Id1	nfe102
Id4	nfe204
Id6	nsy122
Id5	nsy218
Id3	nfa011
Id2	nfe106

NestedLoopJoin : Exemple

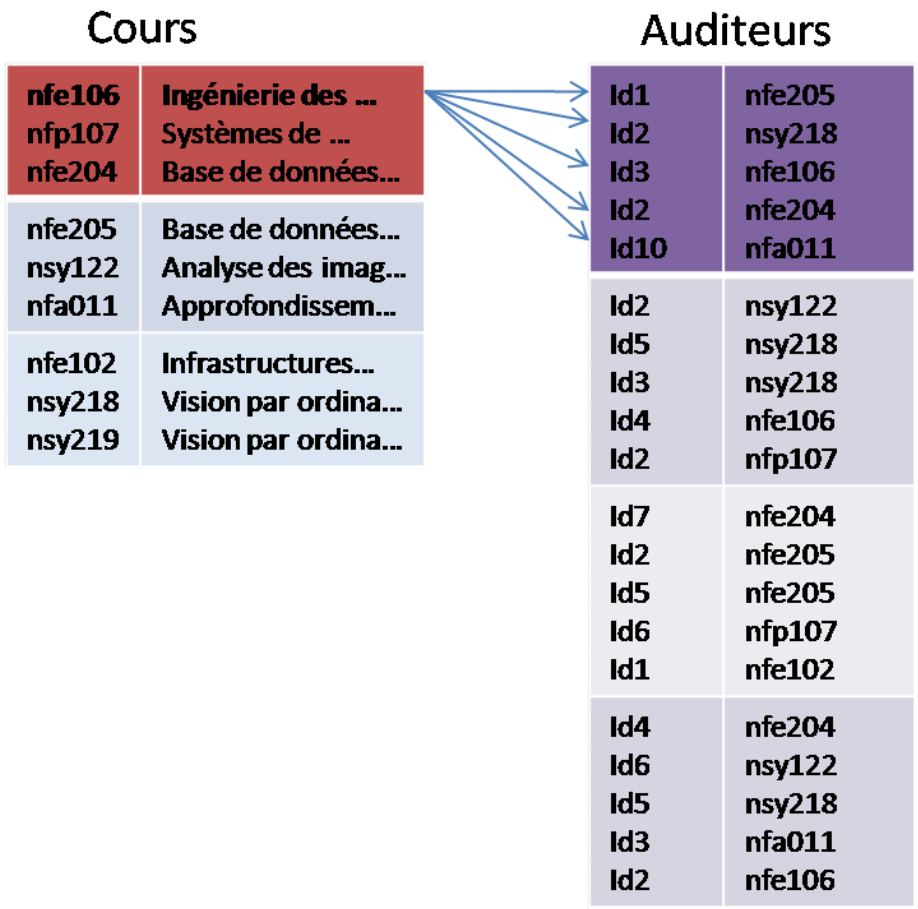
Cours

nfe106	Ingénierie des ...
nfp107	Systèmes de ...
nfe204	Base de données...
nfe205	Base de données...
nsy122	Analyse des imag...
nfa011	Approfondissem...
nfe102	Infrastructures...
nsy218	Vision par ordina...
nsy219	Vision par ordina...

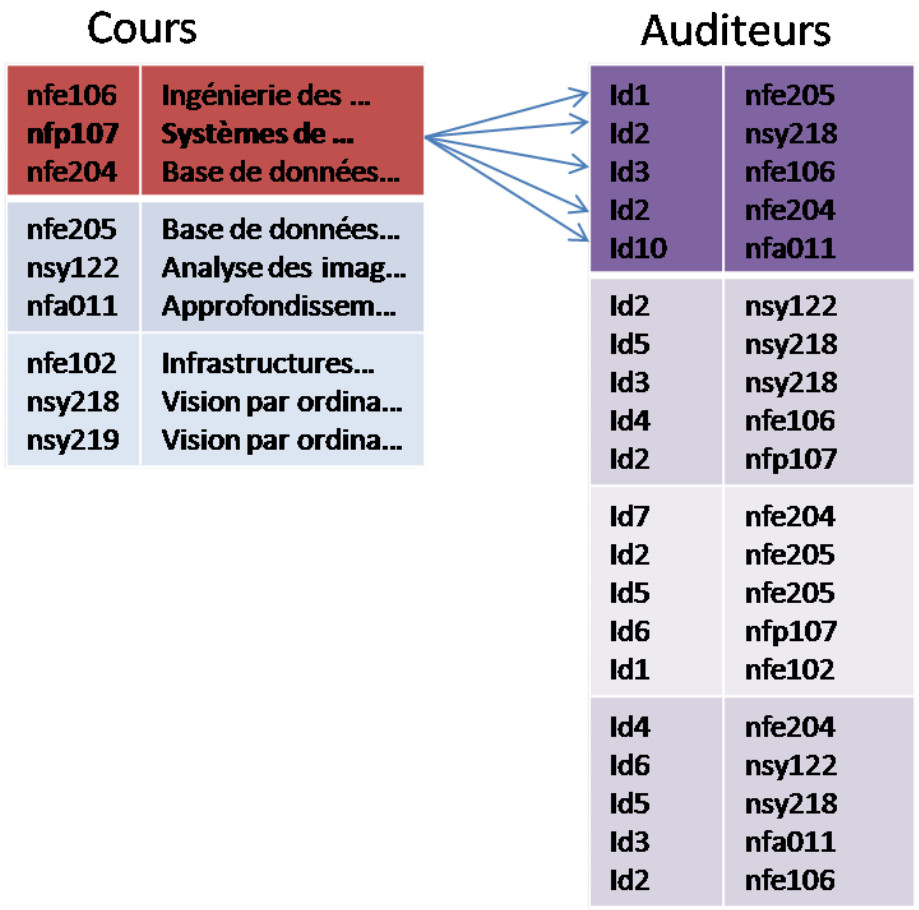
Auditeurs

Id1	nfe205
Id2	nsy218
Id3	nfe106
Id2	nfe204
Id10	nfa011
Id2	nsy122
Id5	nsy218
Id3	nsy218
Id4	nfe106
Id2	nfp107
Id7	nfe204
Id2	nfe205
Id5	nfe205
Id6	nfp107
Id1	nfe102
Id4	nfe204
Id6	nsy122
Id5	nsy218
Id3	nfa011
Id2	nfe106

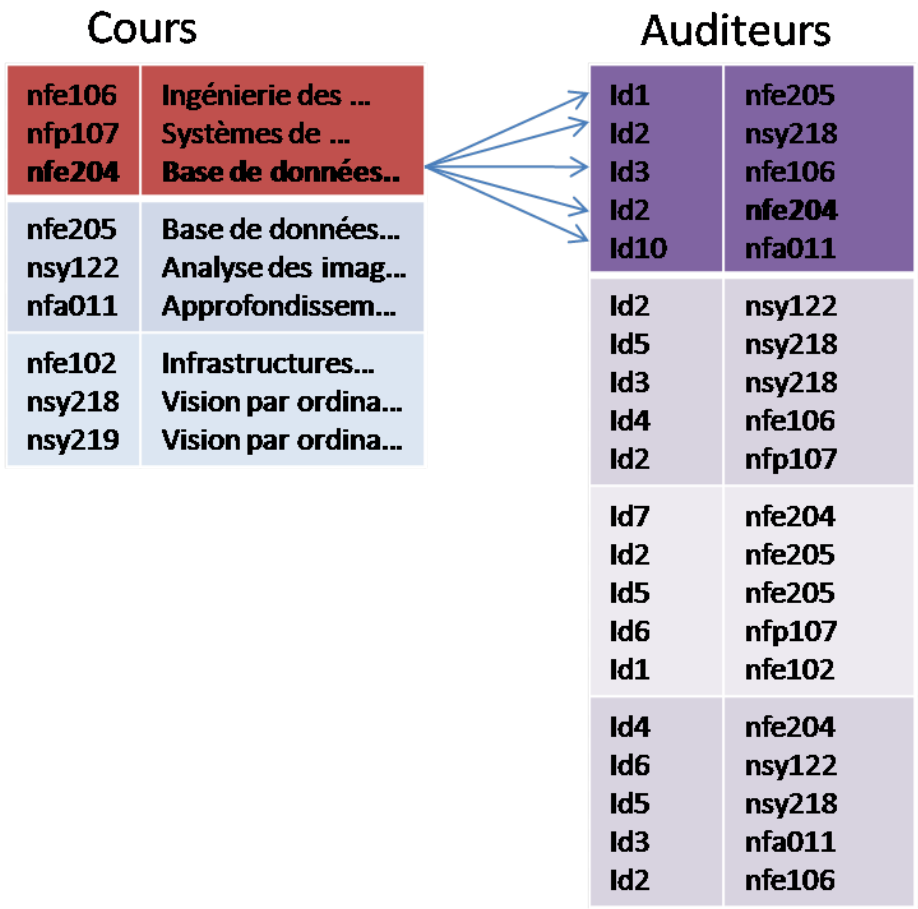
NestedLoopJoin : Exemple



NestedLoopJoin : Exemple



NestedLoopJoin : Exemple



NestedLoopJoin : Exemple

Cours

nfe106	Ingénierie des ...
nfp107	Systèmes de ...
nfe204	Base de données...
nfe205	Base de données...
nsy122	Analyse des imag...
nfa011	Approfondissem...
nfe102	Infrastructures...
nsy218	Vision par ordina...
nsy219	Vision par ordina...

Auditeurs

Id1	nfe205
Id2	nsy218
Id3	nfe106
Id2	nfe204
Id10	nfa011
Id2	nsy122
Id5	nsy218
Id3	nsy218
Id4	nfe106
Id2	nfp107
Id7	nfe204
Id2	nfe205
Id5	nfe205
Id6	nfp107
Id1	nfe102
Id4	nfe204
Id6	nsy122
Id5	nsy218
Id3	nfa011
Id2	nfe106

NestedLoopJoin : Exemple

Cours

nfe106	Ingénierie des ...
nfp107	Systèmes de ...
nfe204	Base de données...
nfe205	Base de données...
nsy122	Analyse des imag...
nfa011	Approfondissem...
nfe102	Infrastructures...
nsy218	Vision par ordina...
nsy219	Vision par ordina...

Auditeurs

Id1	nfe205
Id2	nsy218
Id3	nfe106
Id2	nfe204
Id10	nfa011
Id2	nsy122
Id5	nsy218
Id3	nsy218
Id4	nfe106
Id2	nfp107
Id7	nfe204
Id2	nfe205
Id5	nfe205
Id6	nfp107
Id1	nfe102
Id4	nfe204
Id6	nsy122
Id5	nsy218
Id3	nfa011
Id2	nfe106

NestedLoopJoin : Exemple

Cours

nfe106	Ingénierie des ...
nfp107	Systèmes de ...
nfe204	Base de données...
nfe205	Base de données...
nsy122	Analyse des imag...
nfa011	Approfondissem...
nfe102	Infrastructures...
nsy218	Vision par ordina...
nsy219	Vision par ordina...

Auditeurs

Id1	nfe205
Id2	nsy218
Id3	nfe106
Id2	nfe204
Id10	nfa011
Id2	nsy122
Id5	nsy218
Id3	nsy218
Id4	nfe106
Id2	nfp107
Id7	nfe204
Id2	nfe205
Id5	nfe205
Id6	nfp107
Id1	nfe102
Id4	nfe204
Id6	nsy122
Id5	nsy218
Id3	nfa011
Id2	nfe106

NestedLoopJoin : Exemple

Cours

nfe106	Ingénierie des ...
nfp107	Systèmes de ...
nfe204	Base de données...
nfe205	Base de données...
nsy122	Analyse des imag...
nfa011	Approfondissem...
nfe102	Infrastructures...
nsy218	Vision par ordina...
nsy219	Vision par ordina...

Auditeurs

Id1	nfe205
Id2	nsy218
Id3	nfe106
Id2	nfe204
Id10	nfa011
Id2	nsy122
Id5	nsy218
Id3	nsy218
Id4	nfe106
Id2	nfp107
Id7	nfe204
Id2	nfe205
Id5	nfe205
Id6	nfp107
Id1	nfe102
Id4	nfe204
Id6	nsy122
Id5	nsy218
Id3	nfa011
Id2	nfe106

NestedLoopJoin : Exemple

Cours

nfe106	Ingénierie des ...
nfp107	Systèmes de ...
nfe204	Base de données...
nfe205	Base de données...
nsy122	Analyse des imag...
nfa011	Approfondissem...
nfe102	Infrastructures...
nsy218	Vision par ordina...
nsy219	Vision par ordina...

Auditeurs

Id1	nfe205
Id2	nsy218
Id3	nfe106
Id2	nfe204
Id10	nfa011
Id2	nsy122
Id5	nsy218
Id3	nsy218
Id4	nfe106
Id2	nfp107
Id7	nfe204
Id2	nfe205
Id5	nfe205
Id6	nfp107
Id1	nfe102
Id4	nfe204
Id6	nsy122
Id5	nsy218
Id3	nfa011
Id2	nfe106

Jointure par boucles imbriquées : Analyse

La boucle s'effectue à deux niveaux :

1. Au niveau des **pages** pour les charger en mémoire.
2. Au niveau des **n-uplets** des pages chargées en mémoire.

Si T_R et T_S représentent le nombre de pages de R et S respectivement, le coût de la jointure est :

$$T_R + T_R \times T_S$$

On ne tient pas compte :

- Jointure des tuples en mémoire
- Cout d'écriture du résultat sur disque, lequel dépend de la taille du résultat

Jointure par tri-fusion

Soit l'expression $\pi_{R.Ap,S.Bq}(R \bowtie_{A_i=B_j} S)$.
Algorithme :

- **Projeter** R sur $\{A_p, A_i\}$
- **Trier** R sur A_i
- **Projeter** S sur $\{B_q, B_j\}$
- **Trier** S sur B_j
- **Fusionner** les deux listes triées.

On les parcourt en parallèle en joignant les n-uplets ayant même valeur pour A_i et B_j.

Jointure par tri-fusion

Soit l'expression $\pi_{R.Ap,S.Bq}(R \bowtie_{A_i=B_j} S)$.
Algorithme :

- **Projeter** R sur $\{A_p, A_i\}$
- **Trier** R sur A_i
- **Projeter** S sur $\{B_q, B_j\}$
- **Trier** S sur B_j
- **Fusionner** les deux listes triées.

On les parcourt en parallèle en joignant les n-uplets ayant même valeur pour A_i et B_j.

Jointure par tri-fusion

Soit l'expression $\pi_{R.Ap,S.Bq}(R \bowtie_{A_i=B_j} S)$.
Algorithme :

- **Projeter** R sur $\{A_p, A_i\}$
- **Trier** R sur A_i
- **Projeter** S sur $\{B_q, B_j\}$
- **Trier** S sur B_j
- **Fusionner** les deux listes triées.

On les parcourt en parallèle en joignant les n-uplets ayant même valeur pour A_i et B_j.

Etape de Tri

- Algorithme par dichotomie des pages de la table[2]
- Cout : $|R| \times \log(|R|)$

[2] Algorithme non vu en NFP107, voir cours NFE106

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat :
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1,
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1,
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1,
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1,
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1, 4,
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1, 4,
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1, 4,
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1, 4,
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1, 4, 10,
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1, 4, 10,
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1, 4, 10, 13
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1, 4, 10, 13
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1, 4, 10, 13
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1, 4, 10, 13
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1, 4, 10, 13
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1, 4, 10, 13
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1, 4, 10, 13
- Coût de lecture ? $|R| + |S|$

Fusion de listes

- Listes stockées sur plusieurs pages

R	1, 4, 8	10, 13, 27	29, 55, 60	65
S	1, 2, 3	4, 5, 10	13, 30, 40	

- Résultat : 1, 4, 10, 13
- Coût de lecture ? $|R| + |S|$

Jointure par tri-fusion : Performance

- Coût dominé par la phase de tri :
$$\mathcal{O}(|R| + |R| \times \log(|R|) + |S| + |S| \times \log(|S|)).$$
- L'étape de fusion est un simple parcours en parallèle
- Algorithme intéressant quand les données sont déjà triées en entrée.

Jointure par tri-fusion : Discussion

Pour de grandes relations et en l'absence d'index, la jointure par tri-fusion présente les avantages suivants :

1. **Efficacité** : bien meilleure que les boucles imbriquées.
2. **Manipulation de données triées** : facilite l'élimination de dupliqués ou l'affichage ordonné.
3. **Très général** : non compatible avec tous les types de θ-jointure ($<, >$ demande des retours en arrière)

Jointure par hachage

Comme la jointure par tri-fusion, la jointure par hachage permet de limiter le nombre de comparaisons entre n-uplets.

1. R et S sont hachées sur l'attribut de jointure avec une fonction H.
2. Création de $2 \times N$ paquets (de tailles $\frac{|R|}{N}$ et $\frac{|S|}{N}$). Chaque paquet de même indice ont des valeurs identiques (Fonction H)
3. On joint les paquets de même indice par Boucle Imbriquée. On évite ainsi de joindre avec des valeurs inutiles

Jointure par hachage : Algorithme

Pour une jointure $R \bowtie_{A=B} S$.

Pour chaque n-uplet r de R **faire**
placer r dans le paquet R' indiquée par $H(r.A)$

Pour chaque n-uplet s de S **faire**
placer s dans le paquet S' indiquée par $H(s.B)$

Pour chaque couple (R'_i, S'_i) **faire**
Joindre les deux paquets (les paquets sont petits)

Jointure par hachage : Performance

Coût (en E/S) :

1. Phase 1 :
   - Coût du hachage de R : $|R|$ (lecture) + $|R|$ (écriture des paquets)
   - Coût du hachage de S : $|S|$ (lecture) + $|S|$ (écriture des paquets)
2. Phase 2 : jointure de R et S par paquets :
   - N jointures à effectuer
   - lecture de $\frac{|R|}{N}$ et $\frac{|S|}{N}$ à chaque jointure
   - Coût de la jointure : $N \times (\frac{|R|}{N} + \frac{|S|}{N}) = |R| + |S|$
3. Coût total $= 3 \times |R| + 3 \times |S|$

Jointure par hachage : Performance

Coût (en E/S) :

1. Phase 1 :
   - Coût du hachage de R : $|R|$ (lecture) + $|R|$ (écriture des paquets)
   - Coût du hachage de S : $|S|$ (lecture) + $|S|$ (écriture des paquets)
2. Phase 2 : jointure de R et S par paquets :
   - N jointures à effectuer
   - lecture de $\frac{|R|}{N}$ et $\frac{|S|}{N}$ à chaque jointure
   - Coût de la jointure : $N \times (\frac{|R|}{N} + \frac{|S|}{N}) = |R| + |S|$
3. Coût total $= 3 \times |R| + 3 \times |S|$

Jointure par hachage : Performance

Coût (en E/S) :

1. Phase 1 :
   - Coût du hachage de R : $|R|$ (lecture) + $|R|$ (écriture des paquets)
   - Coût du hachage de S : $|S|$ (lecture) + $|S|$ (écriture des paquets)
2. Phase 2 : jointure de R et S par paquets :
   - N jointures à effectuer
   - lecture de $\frac{|R|}{N}$ et $\frac{|S|}{N}$ à chaque jointure
   - Coût de la jointure : $N \times (\frac{|R|}{N} + \frac{|S|}{N}) = |R| + |S|$
3. Coût total $= 3 \times |R| + 3 \times |S|$

Jointure par hachage : Performance

Coût (en E/S) :

1. Phase 1 :
   - Coût du hachage de R : $|R|$ (lecture) + $|R|$ (écriture des paquets)
   - Coût du hachage de S : $|S|$ (lecture) + $|S|$ (écriture des paquets)
2. Phase 2 : jointure de R et S par paquets :
   - N jointures à effectuer
   - lecture de $\frac{|R|}{N}$ et $\frac{|S|}{N}$ à chaque jointure
   - Coût de la jointure : $N \times (\frac{|R|}{N} + \frac{|S|}{N}) = |R| + |S|$
3. Coût total $= 3 \times |R| + 3 \times |S|$

Jointure par hachage : Performance

Coût (en E/S) :

1. Phase 1 :
   - Coût du hachage de R : $|R|$ (lecture) + $|R|$ (écriture des paquets)
   - Coût du hachage de S : $|S|$ (lecture) + $|S|$ (écriture des paquets)
2. Phase 2 : jointure de R et S par paquets :
   - N jointures à effectuer
   - lecture de $\frac{|R|}{N}$ et $\frac{|S|}{N}$ à chaque jointure
   - Coût de la jointure : $N \times (\frac{|R|}{N} + \frac{|S|}{N}) = |R| + |S|$
3. Coût total $= 3 \times |R| + 3 \times |S|$

Jointure par hachage : Performance

Coût (en E/S) :

1. Phase 1 :
   - Coût du hachage de R : $|R|$ (lecture) + $|R|$ (écriture des paquets)
   - Coût du hachage de S : $|S|$ (lecture) + $|S|$ (écriture des paquets)
2. Phase 2 : jointure de R et S par paquets :
   - N jointures à effectuer
   - lecture de $\frac{|R|}{N}$ et $\frac{|S|}{N}$ à chaque jointure
   - Coût de la jointure : $N \times (\frac{|R|}{N} + \frac{|S|}{N}) = |R| + |S|$
3. Coût total $= 3 \times |R| + 3 \times |S|$

Jointure par hachage : Performance

Coût (en E/S) :

1. Phase 1 :
   - Coût du hachage de R : $|R|$ (lecture) + $|R|$ (écriture des paquets)
   - Coût du hachage de S : $|S|$ (lecture) + $|S|$ (écriture des paquets)
2. Phase 2 : jointure de R et S par paquets :
   - N jointures à effectuer
   - lecture de $\frac{|R|}{N}$ et $\frac{|S|}{N}$ à chaque jointure
   - Coût de la jointure : $N \times (\frac{|R|}{N} + \frac{|S|}{N}) = |R| + |S|$
3. Coût total $= 3 \times |R| + 3 \times |S|$

Jointure par hachage : Discussion

Il est préférable d'effectuer le hachage lorsque:

- l'une des deux tables est très grandes
- l'autre de taille moyenne (ou petite) $\Rightarrow$ coût de jointure normal $(|R| + |S|)$

La jointure par hachage n'est pas adaptée aux jointures avec inégalités.

Jointure avec une table indexée

1. Parcourt séquentiel de R (table directrice sans index)
2. Pour chaque n-uplet r de R :
   - Recherche de $r.A$ dans l'index sur $S.B$
   - Récupérer chaque n-uplet s pointé par l'index
   - Joindre r avec chaque s

Boucles imbriquées avec une table indexée

ALGORITHME **boucles-imbriquées-index**

begin
$J := \emptyset$
for each r **in** R
  for each s **in** $Index_{S_B}(r.A)$
    $J := J + \{r \bowtie s\}$
end

Coût:

- Parcourt sequentiel : $|R|$
- Nombre de recherche dans l'index $||R||$ (nombre de n-uplets)
- Recherche et accès aux n-uplets : $|Index_{S_B}|$ (hauteur de l'arbre) + $sel \times ||S||$ (selectivité de l'index)
- Coût total : $\mathcal{O}(|R| + ||R|| \times (|Index_{S_B}| + sel \times ||S||))$

Boucles imbriquées avec une table indexée

ALGORITHME **boucles-imbriquées-index**

begin
$J := \emptyset$
for each r **in** R
  for each s **in** $Index_{S_B}(r.A)$
    $J := J + \{r \bowtie s\}$
end

Coût:

- Parcourt sequentiel : $|R|$
- Nombre de recherche dans l'index $||R||$ (nombre de n-uplets)
- Recherche et accès aux n-uplets : $|Index_{S_B}|$ (hauteur de l'arbre) + $sel \times ||S||$ (selectivité de l'index)
- Coût total : $\mathcal{O}(|R| + ||R|| \times (|Index_{S_B}| + sel \times ||S||))$

Boucles imbriquées avec une table indexée

ALGORITHME **boucles-imbriquées-index**

begin
$J := \emptyset$
for each r **in** R
  for each s **in** $Index_{S_B}(r.A)$
    $J := J + \{r \bowtie s\}$
end

Coût:

- Parcourt sequentiel : $|R|$
- Nombre de recherche dans l'index $||R||$ (nombre de n-uplets)
- Recherche et accès aux n-uplets : $|Index_{S_B}|$ (hauteur de l'arbre) + $sel \times ||S||$ (selectivité de l'index)
- Coût total : $\mathcal{O}(|R| + ||R|| \times (|Index_{S_B}| + sel \times ||S||))$

Boucles imbriquées avec une table indexée

ALGORITHME **boucles-imbriquées-index**

begin
$J{:=}\emptyset$
for each r **in** R
  for each s **in** $Index_{S_B}(r.A)$
    $J := J + \{r \bowtie s\}$
end

Coût:

- Parcourt sequentiel : $|R|$
- Nombre de recherche dans l'index $||R||$ (nombre de n-uplets)
- Recherche et accès aux n-uplets : $|Index_{S_B}|$ (hauteur de l'arbre) + $sel \times ||S||$ (selectivité de l'index)
- Coût total : $\mathcal{O}(|R| + ||R|| \times (|Index_{S_B}| + sel \times ||S||))$

Boucles imbriquées avec une table indexée

ALGORITHME **boucles-imbriquées-index**

begin
$J := \emptyset$
for each r **in** R
  for each s **in** $Index_{S_B}(r.A)$
    $J := J + \{r \bowtie s\}$
end

Coût:

- Parcourt sequentiel : $|R|$
- Nombre de recherche dans l'index $||R||$ (nombre de n-uplets)
- Recherche et accès aux n-uplets : $|Index_{S_B}|$ (hauteur de l'arbre) + $sel \times ||S||$ (selectivité de l'index)
- Coût total : $\mathcal{O}(|R| + ||R|| \times (|Index_{S_B}| + sel \times ||S||))$

Boucles imbriquées avec une table indexée

ALGORITHME **boucles-imbriquées-index**

begin
$J := \emptyset$
for each r **in** R
  for each s **in** $Index_{S_B}(r.A)$
    $J := J + \{r \bowtie s\}$
end

Coût:

- Parcourt sequentiel : $|R|$
- Nombre de recherche dans l'index $||R||$ (nombre de n-uplets)
- Recherche et accès aux n-uplets : $|Index_{S_B}|$ (hauteur de l'arbre) + $sel \times ||S||$ (selectivité de l'index)
- Coût total : $\mathcal{O}(|R| + ||R|| \times (|Index_{S_B}| + sel \times ||S||))$

Jointure par boucle imbriquée avec index : Discussion

Il est préférable d'effectuer la jointure avec index lorsque:

- Une table est indexé sur l'attribut de jointure
- L'autre table a peu de n-uplets á rechercher dans l'index (peu de parcours d'index)

La jointure par index est adaptée aux jointures avec inégalités.

Jointure avec deux tables indexées

Si les deux tables sont indexées sur les deux attributs de jointure, on peut utiliser une variante de l'algorithme de tri-fusion :

1. On fusionne les deux index (déjà triés) pour constituer une liste (Rid, Sid) de couples d'adresses pour les articles satisfaisant la condition de jointure.
2. On parcourt la liste en accédant aux tables pour constituer le résultat.

Inconvénient : on risque de lire plusieurs fois la même page. En pratique, on préfère utiliser une boucle imbriquée en prenant la plus petite table comme table directrice.

Choix de l'algorithme

		A		
		Petite	**Moyenne**	**Grande**
B	**Petite**	BI[3]	BI ($B \bowtie A$)	BI ($B \bowtie A$) ou JH[4]
	Moyenne	BI	JH	JH
	Grande	BI ou JH	JH	TF[5]
	Grande & Indexée	BI Index	BI Index ou JH	TF

On suppose :

1. Module récoltant périodiquement des statistiques sur la base
2. Estimation en temps réel des statistiques par échantillonnage.

[3] BI: Boucle Imbriquée
[4] JH: Jointure par Hachage
[5] TF : Jointure par Tri-Fusion

Plan du cours

Plans d'exécution

Le résultat de l'optimisation est un **plan d'exécution**: c'est un ensemble d'opérations de niveau intermédiaire, dit **algèbre "physique"** constituée :

1. De chemins d'accès aux données
2. D'opérations manipulant les données, (correspondant aux noeuds internes de l'arbre de requête).

Algèbre physique

CHEMINS D'ACCES

Sequentiel

Parcours sequentiel

Adresse

Acces par adresse

Attribut(s)

Parcours d'index

OPERATIONS PHYSIQUES

Critere

Selection selon un critere

Attribut(s)

Tri sur un attribut

Critere

Fusion de deux ensembles tries

Critere

Filtre d'un ensemble en fonction d'un autre

Critere

Jointure selon un critere

Attribut(s)

Projection sur des attributs

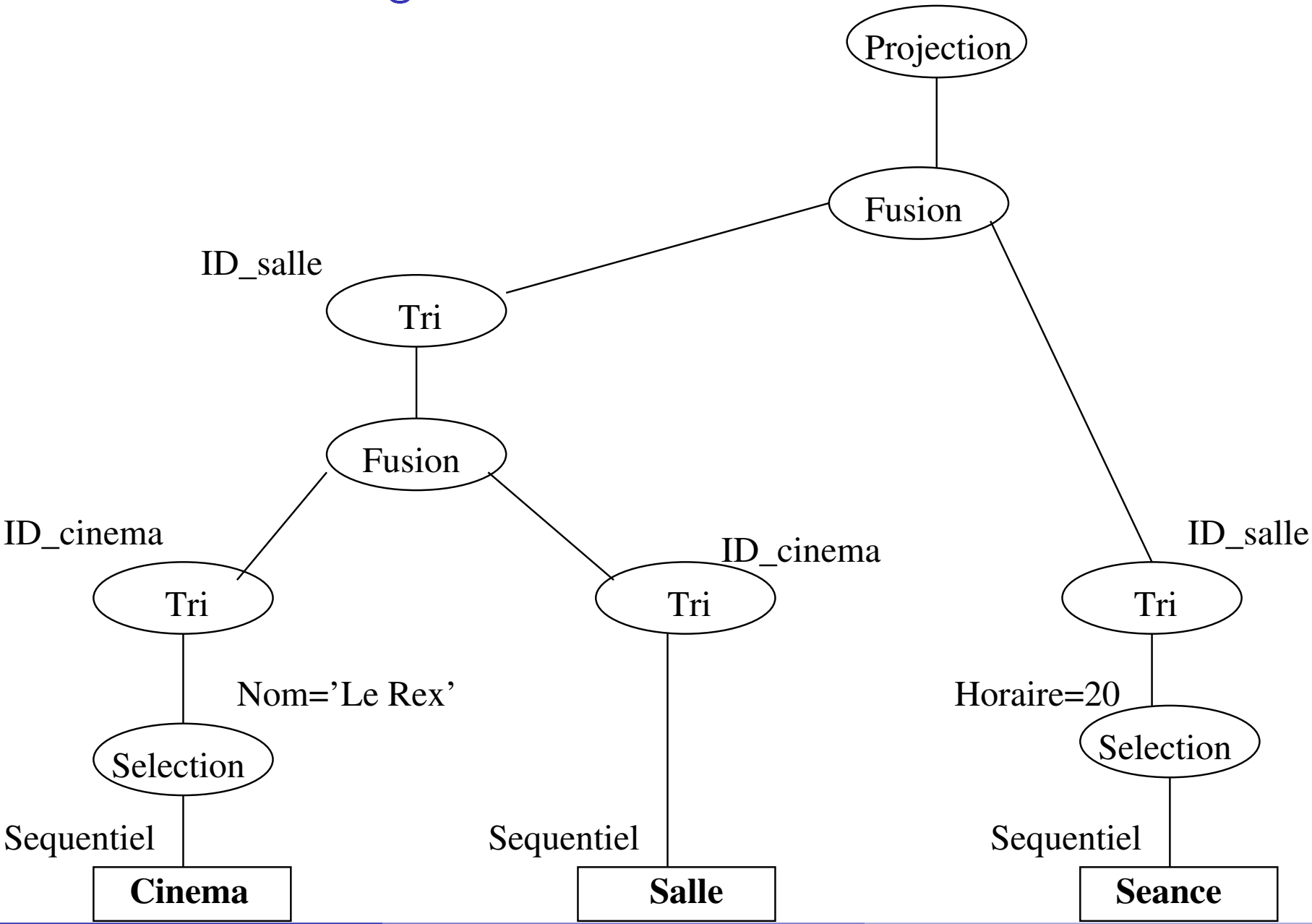
Exemple

Quels films passent au REX à 20 heures ?

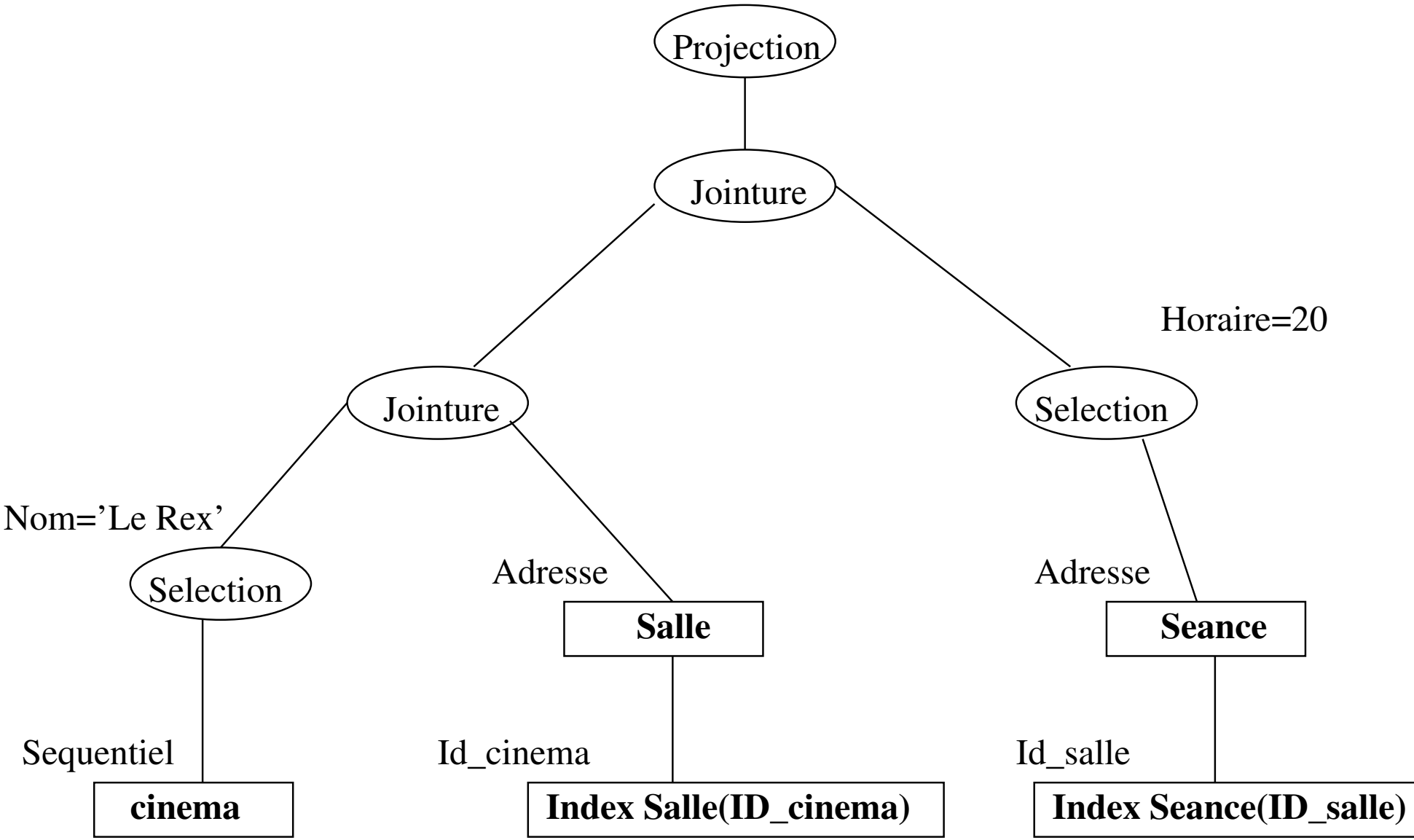
```
select Titre
  from Cinema, Salle, Seance
 where Cinema.nom = 'Le Rex'
   and Cinema.ID_cinema = Salle.ID_cinema
   and Salle.ID_salle=Seance.ID_salle
   and Seance.horaire='20'
```

La requête contient deux selections et deux jointures.

Sans index ni hachage



Avec un index sur les attributs de jointure



Plan du cours

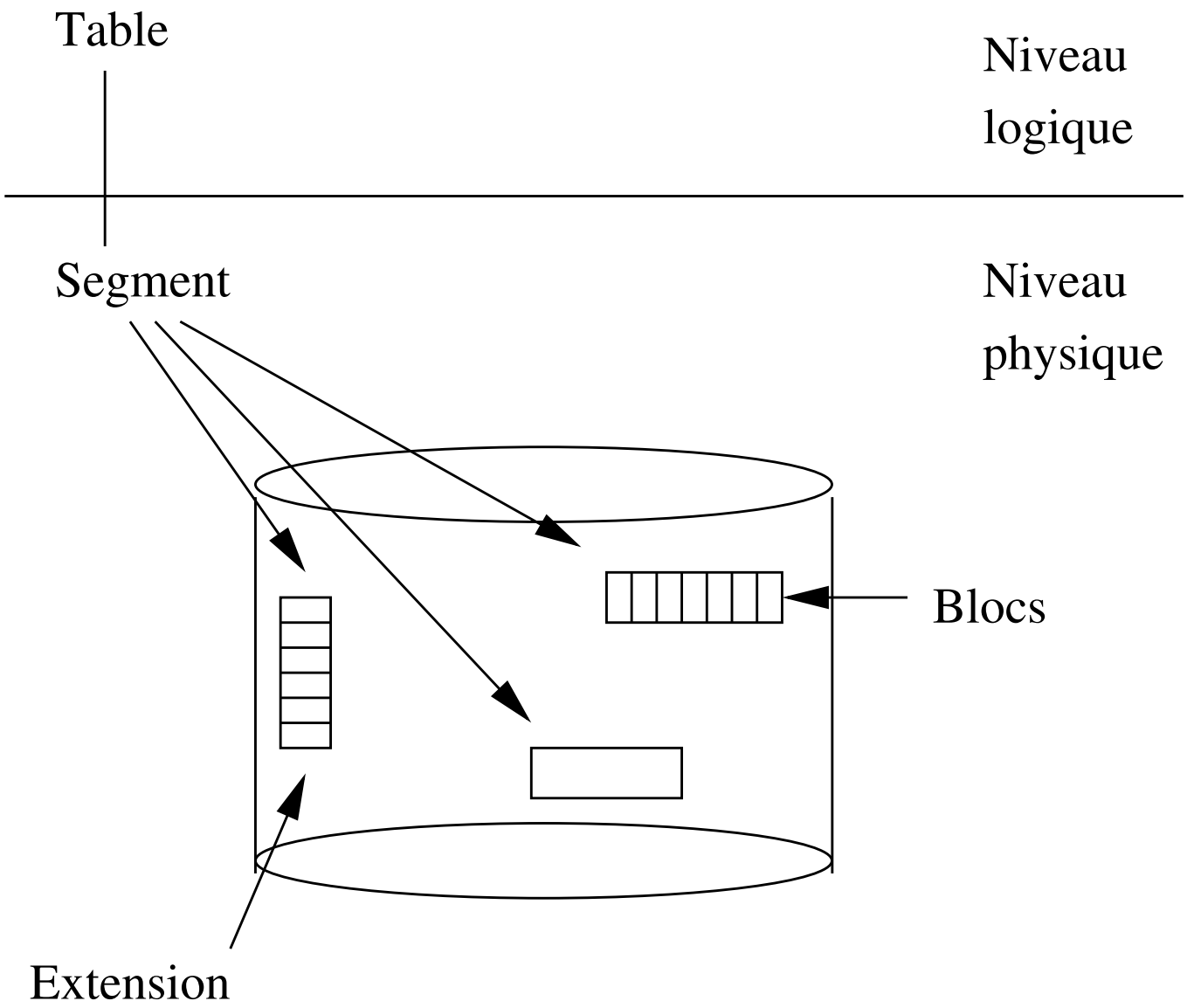
7. Representation physique des données dans Oracle
   - Fichiers
   - Stockage
   - Index et Clusters

Représentation physique des données dans ORACLE depuis V7

Les principales structures physiques utilisées dans ORACLE sont :

1. Le **bloc** est l'unité physique d'E/S (entre 1KO et 8KO). La taille d'un bloc ORACLE est un multiple de la taille des blocs (pages) du système sous-jacent.
2. L'**extension** est un ensemble de blocs *contigus* contenant un même type d'information.
3. Le **segment** est un ensemble d'extensions stockant un objet logique (une table, un index ...).

Tables, segments, extensions et blocs



Le segment ORACLE

Le segment est la zone physique contenant un objet logique. Il existe quatre types de segments :

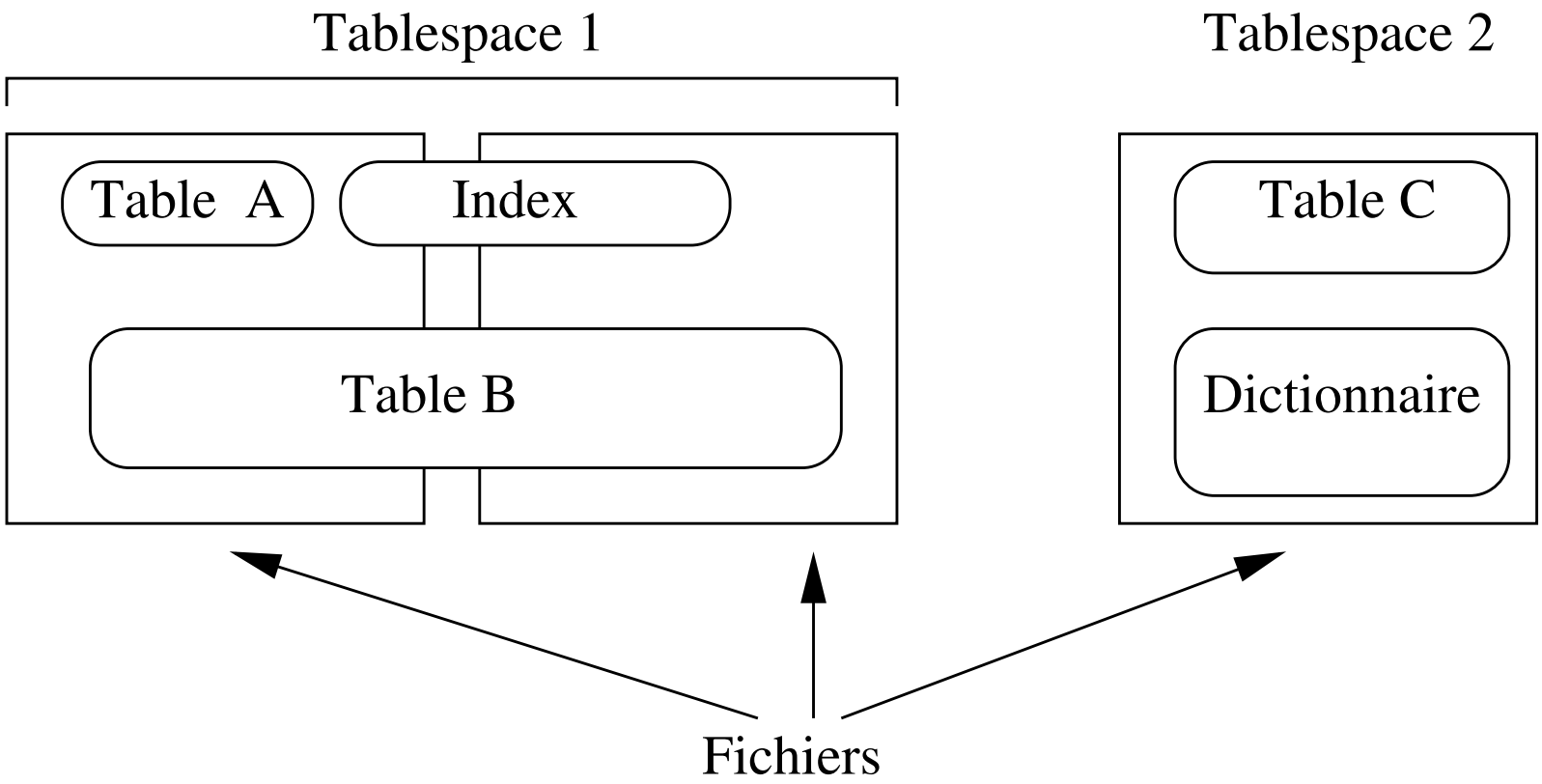
1. Le segment de données (pour une table ou un *cluster*).
2. Le segment d'index.
3. Le *rollback segment* utilisé pour les transactions.
4. Le segment temporaire (utilisé pour les tris par exemple).

Base ORACLE, fichiers et *TABLESPACE*

1. **Physiquement**, une base ORACLE est un ensemble de fichiers.
2. **Logiquement**, une base est divisée par l'administrateur en *tablespace*. Chaque *tablespace* consiste en un ou plusieurs fichiers.

La notion de *tablespace* permet :

1. De contrôler l'emplacement physique des données. (par ex. : le dictionnaire sur un disque, les données utilisateur sur un autre).
2. de faciliter la gestion (sauvegarde, protection, etc).


Tablespace 1
Tablespace 2
Table A
Index
Table B
Table C
Dictionnaire
Fichiers

Plan du cours

7. Representation physique des données dans Oracle
   - Fichiers
   - Stockage
   - Index et Clusters

Stockage des données

Il existe deux manières de stocker une table :

1. **Placement indépendant :** Les segments sont automatiquement alloués à la table. Il est possible de spécifier des paramètres pour la création d'un nouveau segment :
   1. Sa taille initiale.
   2. Le pourcentage d'espace libre dans chaque bloc.
   3. La taille des extensions.
2. **Dans un** *cluster*.

TABLESPACE : définition

Exemple : Création d'un "tablespace" *tabspace_2* dans le fichier *diska:tabspace_file2.dat* de taille 2Go :

```
CREATE TABLESPACE tabspace_2
  DATAFILE 'diska:tabspace_file2.dat' SIZE 2G
  DEFAULT STORAGE (INITIAL 500K NEXT 1M
                   MINEXTENTS 1 MAXEXTENTS 999
                   PCTINCREASE 50);
```

La taille de la première extension est 500 ko, de la deuxième extension 1 Mo avec un taux de croissement de 20% pour les extensions suivantes : 1.5 Mo, 2.25 Mo, 3.375 Mo, ... (défaut: 50%)

CREATE TABLE

Exemple : Création d'une table *salgrade* dans le tablespace *tabspace_2* :

```
CREATE TABLE salgrade (
   grade NUMBER PRIMARY KEY USING INDEX TABLESPACE users_a
   losal NUMBER,
   hisal NUMBER )
  TABLESPACE tabspace_2
  PCTFREE 10 PCTUSED 75
```

L'index est stocké dans le "tablespace" *users_a*. Pour les données, 10% dans chaque bloc sont réservés pour les mise-à-jours. On insère des n-uplets dans un bloc si l'espace occupé descend en dessous de 75%.

Stockage des n-uplets

En règle générale un n-uplet est stocké dans un seul bloc. L'adresse physique d'un n-uplet est le *ROWID* qui se décompose en trois parties :

1. Le numéro du n-uplet dans la page disque.
2. Le numéro de la page, relatif au **fichier** dans lequel se trouve le n-uplet.
3. Le numéro du fichier.

Exemple : $\overbrace{00000DD5}^{page}.\overbrace{000}^{n-uplets}.\overbrace{001}^{fichier}$ est l'adresse du premier n-uplet de la page DD5 dans le premier fichier.

Structures de données pour l'optimisation

ORACLE propose plusieurs structures pour l'optimisation de requêtes.
Voici ceux correspondantes aux cours :

1. Les index.
2. Le hachage.

Plan du cours

7. Representation physique des données dans Oracle
   - Fichiers
   - Stockage
   - Index et Clusters

Les index ORACLE

On peut créer des index sur tout attribut (ou tout ensemble d'attributs) d'une table. ORACLE utilise l'arbre B+.

1. Les noeuds contiennent les valeurs de l'attribut (ou des attributs) clé(s).
2. Les feuilles contiennent chaque valeur indexée et le *ROWID* correspondant.

Un index est stocké dans un segment qui lui est propre. On peut le placer par exemple sur un autre disque que celui contenant la table.

Création d'un index B+Tree

Index de nom *index_cin_nom* sur la table CINEMA sur l'attribut NOM :

```
CREATE INDEX ind_cin_nom ON CINEMA (nom) ;
```

Le hachage (hash cluster)

On définit un *hash cluster* décrivant les caractéristiques physiques de la table :

```
CREATE CLUSTER hash-cinema (ID-cinema NUMBER(10))
    HASH IS ID-cinema
    HASHKEYS 1000
    SIZE 2K
```

- *HASH IS* (optionnel) spécifie la clé à hacher.
- *HASHKEYS* est le nombre de valeurs de la clé de hachage.
- *SIZE* est la taille des données pour une clé donnée.
- ORACLE détermine le nombre de pages allouées, ainsi que la fonction de hachage.

CREATE TABLE

On fait référence au *hash cluster* pendant la création de la table :

```
CREATE TABLE Cinema (
    id-cinema NUMBER(10) PRIMARY KEY,
    nom-cinema VARCHAR(32),
    adresse VARCHAR(64)
)
 CLUSTER cluster-cinema(id-cinema);
```

Plan du cours

L'optimiseur

L'optimiseur ORACLE suit une approche classique :

1. Génération de plusieurs plans d'exécution.
2. Estimation du coût de chaque plan généré.
3. Choix du meilleur et exécution.

Estimation du coût d'un plan d'exécution

Beaucoup de paramètres entrent dans l'estimation du coût :

1. Les chemins d'accès disponibles.
2. Les opérations physiques de traitement des résultats intermédiaires.
3. Des statistiques sur les tables concernées (taille, sélectivité). Les statistiques sont calculées par appel explicite à l'outil ANALYSE.
4. Les ressources disponibles.

Plan du cours

Sélection / Accès aux données

Opération	Option	Description
`TABLE ACCESS`	`FULL`	*SelSeq*
	`BY INDEX ROWID`	*SelInd*, avec l'attribut indexé (cf index) et la valeur

Accès à un Index

Opération	Option	Description
`INDEX`	`UNIQUE SCAN`	*SelInd*, clé primaire/unique
	`RANGE SCAN`	*SelInd*, données multivaluées

Accès à un Index Hash

Opération	Option	Description
PARTITION HASH	SINGLE	Accès à la partition sélectionné par la clé d'accès (sur le FULL SCAN)
	ALL	Accès à toutes les partitions (pas de filtrage sur la clé hachée)

Ordonnancement

Opération	Option	Description
SORT	ORDER BY	*Sort* pour les résultats
	JOIN	*SortMergeJoin*, étape de tri de l'algo de jointure
	UNIQUE	*SortProj*, supprime les doublons
	GROUP BY	*Sort*, produit une ligne pour chaque groupement de valeurs (sans fonctions)
	AGGREGATE	*Sort*, produit une ligne pour chaque groupement de valeurs avec application de fonction

Jointures

Opération	Description
`NESTED LOOPS`	*NLJ*/*NLJI*, Boucles imbriquées
`MERGE JOIN`	*SMJ*, Etape de fusion des tables
`HASH JOIN`	*HashJoin*, construction des partitions, puis suite de *NLJ*
`AND-EQUAL`	Intersection de deux listes de ROWIDs pour une jointure entre deux indexes

Options : `OUTER`, `ANTI`, `SEMI`, `CARTESIAN`.

Plan du cours

L'outil EXPLAIN

L'outil EXPLAIN donne le plan d'exécution d'une requête. La description comprend :

1. Le chemin d'accès utilisé.
2. Les opérations physiques (tri, fusion, intersection, ...).
3. L'ordre des opérations. Il est représentable par un arbre.

EXPLAIN par l'exemple : schéma de la base

CINEMA
(ID-cinéma*,
Nom,
Adresse);

SALLE
(ID-salle*,
Nom,
Capacité+,
ID-cinéma+);

FILM
(ID-film,
Titre,
Année,
ID-réalisateur+)

SEANCE (ID-séance*,
Heure-début, Heure-fin,
ID-salle+,ID-film

ARTISTE (ID-artiste*,
Nom,
Date-naissance)

Attributs avec une * : index unique.
Attributs avec une + : index non unique.

Interprétation d'une requête par EXPLAIN

Reprenons l'exemple : Quels films passent aux Rex à 20 heures ?

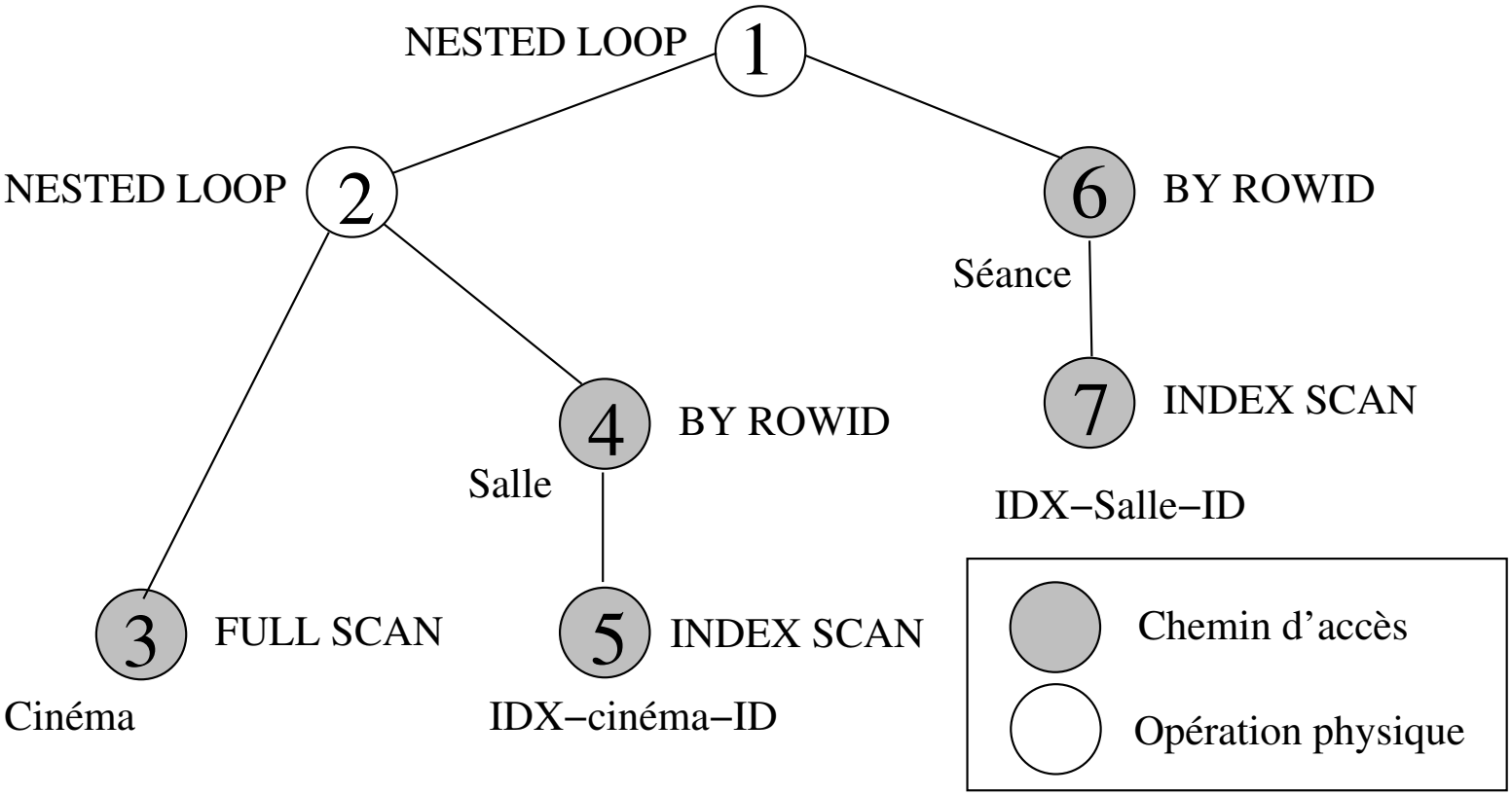
```
EXPLAIN PLAN SET statement-id = 'cin'
FOR     SELECT ID-film
        FROM   Cinéma, Salle, Séance
        WHERE  Cinéma.ID-cinéma = Salle.ID-cinéma
        AND    Salle.ID-salle = Séance.ID-salle
        AND    Cinéma.nom = 'Le Rex'
        AND    Séance.heure-début = '20H'
```

Plan d'exécution donné par EXPLAIN

```
0 SELECT STATEMENT
1   NESTED LOOP
2     NESTED LOOPS
3*      TABLE ACCESS FULL              CINEMA
4       TABLE ACCESS BY ROWID          SALLE
5*        INDEX RANGE SCAN             IDX-SALLE_IDCINEMA
6*    TABLE ACCESS BY ROWID            SEANCE
7*      INDEX RANGE SCAN               IDX-SEANCE_IDSALLE

3 - access("nom"='Le Rex')
4 - access(CINEMA.ID_CINEMA = SALLE.ID_CINEMA)
6 - access("heure-début"='20H')
7 - access(SALLE.ID_SALLE = SEANCE.ID_SALLE)
```

Représentation arborescente du plan d'exécution



Quelques remarques sur EXPLAIN

EXPLAIN utilise un ensemble de primitives que nous avons appelé "algèbre physique": des opérations comme le tri n'existent pas au niveau relationnel. D'autres opérations de l'algèbre relationnelle sont regroupées en une seule opération physique.

- Par exemple, la sélection sur l'horaire des séances est effectuée en même temps que la recherche par ROWID (étape 6).

Exemple : sélection sans index

```
SELECT * FROM cinéma WHERE  nom = 'Le Rex'
```

Plan d'exécution :

```
0 SELECT STATEMENT
1*  TABLE ACCESS FULL       CINEMA

1 - access("nom"='Le Rex')
```

Sélection avec index

```
SELECT * FROM cinéma WHERE  ID-cinéma = 1908
```

Plan d'exécution :

```
0 SELECT STATEMENT
1   TABLE ACCESS BY ROWID    CINEMA
2*    INDEX UNIQUE SCAN      IDX-CINEMA_IDCINEMA

2 - access("ID-cinéma"=1908)
```

Sélection conjonctive avec un index

```
SELECT capacité FROM Salle
WHERE  ID-cinéma =187 AND nom = 'Salle 1'
```

Plan d'exécution :

```
0 SELECT STATEMENT
1*  TABLE ACCESS BY ROWID     SALLE
2*    INDEX RANGE SCAN        IDX-SALLE_IDCINEMA

1 - access("nom"='Salle 1')
2 - access("ID-cinéma"=187)
```

Sélection conjonctive avec deux index

```
SELECT  nom FROM   Salle
 WHERE ID-cinéma = 1908 AND capacité = 150
```

Plan d'exécution :

```
0 SELECT STATEMENT
1   TABLE ACCESS BY ROWID    SALLE
2     AND-EQUAL
3*      INDEX RANGE SCAN     IDX-SALLE_IDCINEMA
4*      INDEX RANGE SCAN     IDX-SALLE_CAPACITE

3 - access("ID-cinéma"=1908)
4 - access("capacité"=150)
```

Sélection disjonctive avec index

SELECT nom FROM Salle
WHERE ID-cinéma = 1908 OR capacité > 150

Plan d'exécution :

```
0 SELECT STATEMENT
1   CONCATENATION
2     TABLE ACCESS BY ROWID  SALLE
3*      INDEX RANGE SCAN     IDX-SALLE_CAPACITE
4     TABLE ACCESS BY ROWID  SALLE
5*      INDEX RANGE SCAN     IDX-SALLE_IDCINEMA

3 - access("capacité">150)
5 - access("ID-cinéma"=1908)
```

Sélection disjonctive sans index

```
SELECT  nom FROM   Salle
WHERE   ID-cinema = 1908 OR nom = 'Salle 1'
```

Plan d'exécution :

```
0 SELECT STATEMENT
1*  TABLE ACCESS FULL        SALLE

1 - access("ID-cinéma"=1908 OR "nom"='Salle 1')
```

Jointure avec index

```
SELECT Cinéma.nom,capacité FROM cinéma, salle
WHERE  Cinéma.ID-cinéma = salle.ID-cinéma
```

Plan d'exécution :

```
0 SELECT STATEMENT
1   NESTED LOOPS
2     TABLE ACCESS FULL       SALLE
3     TABLE ACCESS BY ROWID   CINEMA
4*      INDEX UNIQUE SCAN     IDX-CINEMA_IDCINEMA

4 - access (Cinéma.ID-cinéma = salle.ID-cinéma)
```

Jointure et sélection avec index

SELECT Cinéma.nom,capacité FROM Cinéma, Salle
WHERE Cinema.ID-cinéma = salle.ID-cinéma
AND capacité > 150

Plan d'exécution :

```
0 SELECT STATEMENT
1  NESTED LOOPS
2     TABLE ACCESS BY ROWID   SALLE
3*      INDEX RANGE SCAN      IDX-SALLE_CAPACITE
4     TABLE ACCESS BY ROWID   CINEMA
5*      INDEX UNIQUE SCAN     IDX-CINEMA_IDCINEMA

3 - access("Capacité">150)
5 - access(Cinéma.ID-cinéma = salle.ID-cinéma)
```

Jointure sans index

```
SELECT titre
FROM Film, Séance
WHERE  Film.ID-film  = Séance.ID-film
AND    heure-début = '14H00'
```

Plan d'exécution :

```
0 SELECT STATEMENT
1*  MERGE JOIN
2     SORT JOIN
3*      TABLE ACCESS FULL   SEANCE
4     SORT JOIN
5       TABLE ACCESS FULL   FILM

1 - access(Cinéma.ID-cinéma = salle.ID-cinéma)
3 - access("heure-début"='14H00')
```

Différence

Dans quel cinéma ne peut-on voir de film après 23H ?

```
SELECT Cinéma.nom
FROM   Cinéma
WHERE NOT EXISTS (SELECT * FROM séance, Salle
                  WHERE Cinéma.ID-cinéma = Salle.ID-cinéma
                  AND Salle.ID-salle = Séance.ID-salle
                  AND heure-fin > '23H00')
```

Plan d'exécution donné par EXPLAIN

```
0 SELECT STATEMENT
1* 1 FILTER
2      TABLE ACCESS FULL        CINEMA
3      NESTED LOOPS
4        TABLE ACCESS BY ROWID  SALLE
5          INDEX RANGE SCAN     IDX-SALLE_IDCINEMA
6*       TABLE ACCESS BY ROWID  SEANCE
7*         INDEX RANGE SCAN     IDX-SEANCE_IDSALLE

1 - filter(LNNVL(SALLE.IDCINEMA)) *LNNVL = negation
6 - access("heure-fin">'23H00')
7 - access(Seance.id-salle = salle.ID-salle)
```