Bases de données relationnelles

Interfaces et implémentation

Michel Rueher

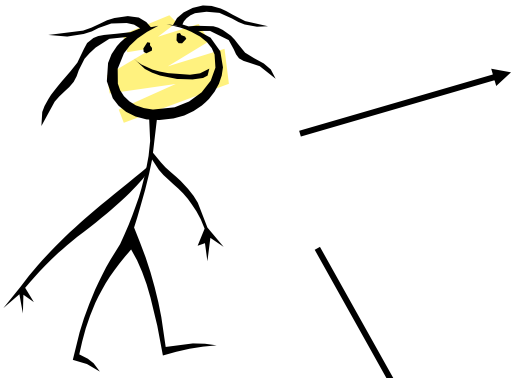
Supports disponibles:

http://www.i3s.unice.fr/~rueher/Cours/BD/

10 Utilisation de SQL dans un langage hôte

- La plupart des applications qui utilisent des bases de données nécessitent l'utilisation conjointe de SQL et d'un langage impératif universel
- Les implémentations SQL2 doivent supporter les langages, **`C, COBOL, Fortran, M, Pascal, JAVA,`** …
- Un des problèmes majeur pour la collaboration réside dans la différence fondamentale des structures de données.

Interface SQL / langage hôte : Schéma général



Langage hôte
+
SQL imbriqué

Pré-processeur

Langage hôte
+
Appels de fonction

Compilateur langage Hôte

Bibliothèque
SQL

Programme
Exécutable

Interface SQL / langage C : principes

- La communication s'effectue via des variables accessibles en C et SQL (préfixées par : en SQL)

```
EXEC SQL BEGIN DECLARE SECTION;
    ...
EXEC SQL END DECLARE SECTION;
```

- Les instructions **SQL** sont préfixées par les mots clés **EXEC SQL** dans de nombreux langages hôtes

- Une variable **SQLSTATE**, tableau de 5 caractères, permet de récupérer le code d'exécution d'une fonction

  00000 : pas d'erreur

  02000 : un tuple demandé n'a pas été trouvé

- Les curseurs permettent d'échanger des ensembles de tuples

Interface SQL / C : exemple de requête SQL avec une seule réponse

```
-- Find the length of the movie for a given title and year.
-- Movie(title, year, length, inColor, sName, presC#)
EXEC SQL BEGIN DECLARE SECTION;
   char theTitle[20];    int theYear;
   int theLength;  char SQLSTATE[6];
EXEC SQL END DECLARE SECTION;
... -- assign the theTitle and theYear
EXEC SQL SELECT length INTO :theLength FROM Movie
WHERE title = :theTitle AND year = :theYear;
if (!strcmp(SQLSTATE, "00000"))
   printf("%d\n", theLength);
else
   printf("Not Found.\n");
```

Interface SQL / C : exemple d'inclusion de SQL

- Insertion d'un nouveau tuple : les variables partagées sont utilisées comme des valeurs

```
Void getStudio () {
   EXEC SQL BEGIN DECLARE SECTION;
        char studioName[15], studioAddr[60];
        char SQLSTATE[6];
   EXEC SQL END DECLARE SECTION;
   --Instanciation studioName et studioAddr
   …
   EXEC SQL INSERT INTO Studio(name, address)
       VALUES (:studioName, :studioAddr);
```

Requêtes à réponses multiples

- Les **curseurs** permettent d'accéder successivement aux différents tuples d'une relation

1. Déclaration d'un curseur

   EXEC SQL DECLARE <cursor_name> CURSOR FOR <query>

2. Initialisation du curseur

   EXEC SQL OPEN <cursor_name>

   % Le curseur est prêt pour l'accès à la première valeur de la relation

3. Accès à un tuple

   EXEC SQL FETCH FROM <cursor_name> INTO <Shared variables>

   -- Le prochain tuple est stocké dans les variables partagées;

   -- s'il n'existe pas: **SQLSTATE** ←'02000'

4. Fermeture du curseur

   EXEC SQL CLOSE <cursor_name>

   -- Le curseur peut être réouvert avec OPEN

Option des curseurs

Il est possible de:

1. Modifier l'ordre dans lequel les tuples sont récupérés en utilisant la clause **`ORDER BY`** après le **`SELECT`** qui définit le curseur
2. Limiter les effets des modifications de la relation sur lequel le curseur est défini

   ```
   EXEC SQL DECLARE <cursor_name> INSENSITIVE
      CURSOR FOR <query>
   ```

   % Le curseur peut aussi être déclaré en READ ONLY
3. Modifier le mode de déplacement sur la séquence de tuples associée à une relation

   ```
   (NEXT, PRIOR, RELATIVE n,…)
   ```

Exemple d'utilisation de curseur

```
#define NO MORE TUPLES !(strcmp(SQLSTATE, “02000”);
void changeWorth()                                {
EXEC SQL BEGIN DECLARE SECTION;
int worth;  char SQLSTATE[6];
EXEC SQL END DECLARE SECTION;
EXEC SQL DECLARE execCur CURSOR FOR
   SELECT netWorth FROM MovieExec;
EXEC SQL OPEN execCur;
while(1)                    {
EXEC SQL FETCH FROM execCur INTO :worth;
if (NO MORE TUPLES) break;
if (worth < 1000)
EXEC SQL DELETE FROM MovieExec WHERE CURRENT OF execCur;
else
EXEC SQL UPDATE MovieExec SET netWorth = 2 * netWorth WHERE
   CURRENT OF execCur;          }
EXEC SQL CLOSE execCursor;                       }
```

Curseurs "READ ONLY"

```
EXEC SQL DECLARE movieStarCursor CURSOR FOR
SELECT title, year, studio, starName
    FROM Movie, StarsIn
    WHERE     title = movieTitle
              AND year = movieYear
FOR READ ONLY;
```

Toute tentative d'exécuter une clause **UPDATE ou DELETE** à travers le curseur `movieStarCursor` va générer une erreur

Interface SQL / langage hôte : PREPARE et EXECUTE

Deux instructions spéciales pour l'incorporation de code SQL:

- **PREPARE** transforme une chaîne de caractères en requête SQL
- **EXECUTE** execute cette requête

```
EXEC SQL PREPARE q FROM :query;
EXEC SQL EXECUTE q;
```

- Une requête "préparée" peut être exécutée plusieurs fois
- PREPARE et EXECUTE peuvent être combinées :

```
EXEC SQL EXECUTE IMMEDIATE :query;
```

11 JDBC

- JDBC (Java Database Connectivity) est une API (Application Programming Interface) fournie avec Java

- L'API JDBC est indépendante du SGBD : elle permet à un programme de se connecter à n'importe quelle base de données relationnelle conforme à ANSI SQL2 en utilisant la même syntaxe

  → **Autorise l'insertion d'instructions SQL dans un programme Java**

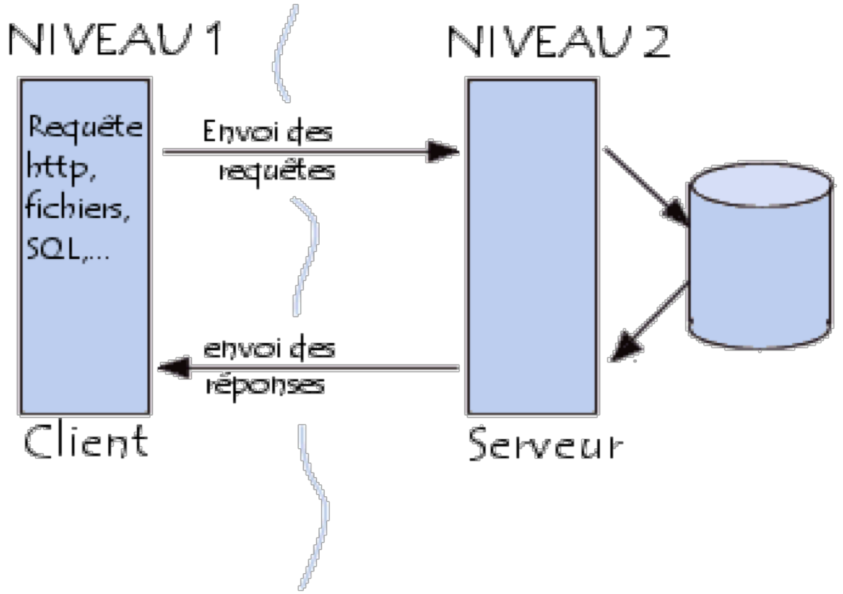
- L'API JDBC assure la portabilité du code

Architecture Client-serveur avec JDBC

- Modèle 2 niveaux :
  - Client ‘’lourd’’: en charge des applications et de l’IHM
  - Serveur ‘’léger’’ : gestion des données

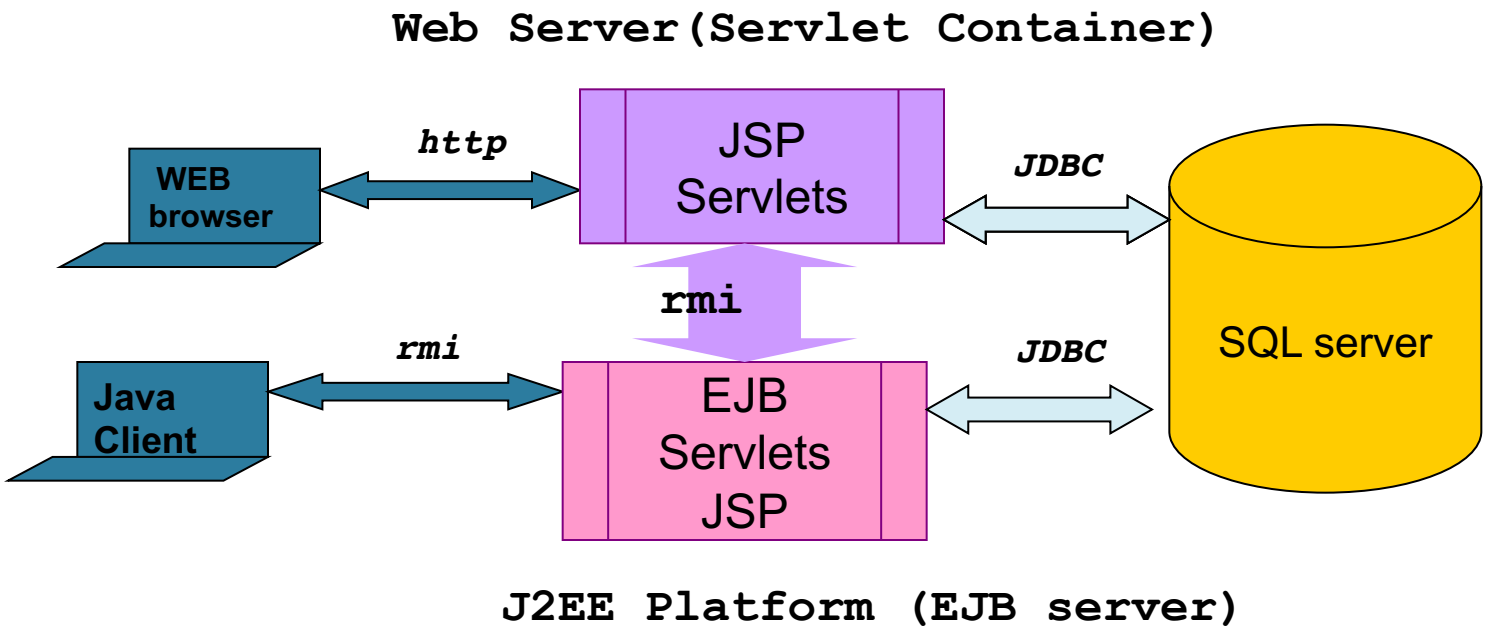
\+ : indépendance totale du DBMS

\- : pas de factorisation de code



Architecture Client-serveur avec JDBC

- Modèle 3 niveaux:
  - Clients "légers": en charge uniquement de l'IHM
  - Serveurs "lourds" (EJB,servlets) : en charge de l'application
  - Serveur DBMS : gestion des données



ODBC(*Open Database Connectivity)*

- **Format propriétaire de Microsoft** pour la communication entre des clients et les SGBD du marché (pour bases de données fonctionnant sous Windows)
- **Résultats identiques** quelque soit le type de base de données, sans avoir à modifier le programme d'appel qui transmet la requête.

Connexion à une base de données avec JDBC

L'API (*Application Programming Interface*) JDBC,
c'est-à-dire la bibliothèque de classes JDBC, se charge de trois étapes indispensables à la connexion à une base de données:

- la **création** d'une connexion à la base
- **l'envoi** d'instructions SQL
- **l'exploitation** des résultats provenant de la base

Package java.sql.*

- **Classes:** **Date, DriverManager**, DriverPropertyInfo
  Time , Timestamp ,Types

- **Interfaces** : Array , Blob , CallableStatement
  Clob , **Connection** ,DatabaseMetaData ,Driver
  PreparedStatement , Ref , **ResultSet** , **ResultSetMetaData**,
  SQLData , SQLInput , SQLOutput , **Statement** , Struct

- **Exceptions** : BatchUpdateException
  DataTruncation , SQLException , SQLWarning

Etablissement de la connexion avec JDBC

Chargement du pilote de la base de données à laquelle on désire se connecter grâce à un appel au DriverManager (gestionnaire de pilotes)

La classe **java.sql.DriverManager** permet d'établir la connexion.

Exemple :

jdbc:postgresql://dbms.polytech.unice.fr:5432/prof x mdp

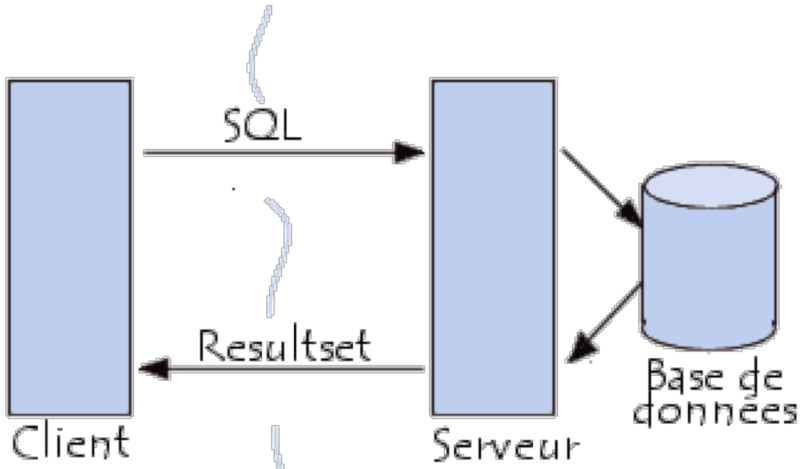
permet à l'utilisateur **x** de se connecter à la base **prof** sur le serveur postgres associé au port **5432** de la machine **dbms.polytech.unice.fr**

Connexion à une BD avec JDBC : exemple

```
import java.sql.*;                    import java.io.*;
public class EdtRemplissage{   public  static void main(String args[])
   {// Parametres de connexion
   String sqlUser = "invited";
   String sqlPwd = "invited";
   String url = "jdbc:postgresql://dbms.polytech.unice.fr:5432/rueher";
        //String url = "jdbc:postgresql://localhost:5432/rueher";
    Connection conn;
        // Chargement du Driver
   try {Class.forName("org.postgresql.Driver");}
   catch(java.lang.ClassNotFoundException e)
        {System.err.print("ClassNotFoundException: ");
        System.err.println(e.getMessage());};
   try {conn = DriverManager.getConnection(url, sqlUser, sqlPwd);
```

L'accès à une base de données avec JDBC

- La classe **java.sql.Statement** fournit les méthodes pour exécuter une requête :
  - **executeQuery() :** consultation
  - **executeUpdate() :** mise à jour
  - **execute() :** requête non définie



- La classe **java.sql.ResultSet** fournit les objets pour obtenir des informations sur la base
  - **DataBaseMetaData :** méta-données
  - **Resultset :** table
  - **ResultsetMetadata:** information sur le nom et le type des données de la table

L'accès aux résultats d'une requête JDBC

Les principales méthodes de l'objet ***ResultSet*** sont les suivantes:

- **getInt(int):** récupère sous forme d'entier le contenu d'une colonne désignée par son numéro
- **getFloat(int):** récupère sous forme de flottant le contenu d'une colonne désignée par son numéro
- **getString(int):** récupère sous forme de chaîne le contenu d'une colonne désignée par son nom
- **next():** déplace le curseur de colonne sur la colonne suivante
- **close():** ferme l'objet
- **getMetaData():** retourne les méta-données de l'objet

L'accès aux résultats d'une requête JDBC

L'objet ***ResultSetMetaData*** permet de connaître le nombre, le nom et le type de chaque colonne à l'aide des méthodes suivantes:

- **getColumnCount():** récupère le nombre de colonnes
- **getColumnName(int):** récupère le nom de la colonne spécifiée
- **getColumnType(int):** récupère le type de données de la colonne spécifiée

Exécution d'une requête avec JDBC : exemple

```
Statement stmt;
    String sqlTxt;              String errMsg;
        // Parametres du formulaire de selection
    String reqCursus ="SI4";
    Int reqSemInt=49;
stmt = conn.createStatement();
stmt.executeUpdate("DELETE FROM affichage");
int[] nbLignes = new int [5];
for (int i=1; i<=5; i++){
stmt.executeUpdate("insert into affichage (jour, heured, duree,
   ligne) " +    "  values(" + Integer.toString(i) + ", 1, 8, 1) " );
nbLignes[i-1]=1;
...
```

requête avec JDBC : exemple (suite)

```
sqlTxt =   " SELECT E.code_module, E.groupe, E.jour, E.heured,
    CT.duree, E.salle " +   " FROM edt_sem  E, cursus_groupe CCT,
    creneau_type CT " +  ….+   " ORDER BY 3,4,5 " ;
    errMsg="PB SQL dans insertion des creneaux a placer : \n" +
    sqlTxt;
PreparedStatement stmtModules =
    conn.prepareStatement(sqlTxt);
ResultSet rsetModules = stmtModules.executeQuery();
// Exploitation des resultats
while (rsetModules.next()){
    String code_module = rsetModules.getString(1); ….
    int jour = rsetModules.getInt(3);
```

Les requêtes pré-formatées

- L'interface **PreparedStatement** fournit la possibilité de lier des paramètres à un appel SQL avant son exécution en associant une valeur à l'indicateur de position (le caractère ?) dans l'instruction prédéfinie

```
PreparedStatement instruction = connection.preparedStatement(
        ''UPDATE comptes'' + ''SET solde = ?'' +''WHERE id = ?'');
int i;
for(i=0; i<comptes.length;i++) {
        instruction.setFloat(1,comptes[i].extraitSolde());
        instruction.setInt(2,comptes[i].extraitIdf());
        instruction.execute();
}
connection.commit();
instruction.close();
```

Évite la coûteuse création d'un plan de requête identique à chaque itération

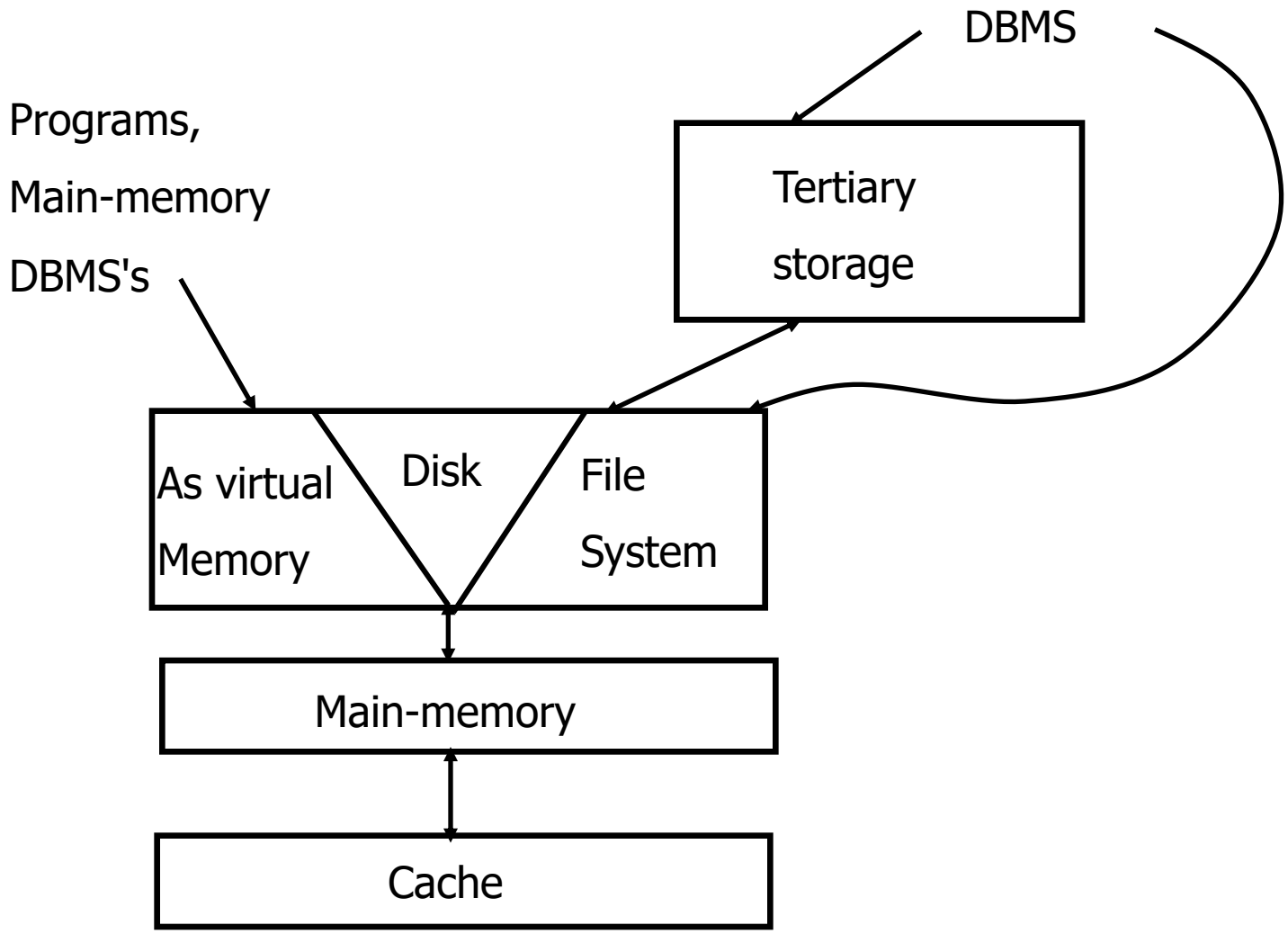
12 DBMS : éléments d'implémentation

- Architecture générale d'un DBMS
- Utilisation efficace de la mémoire secondaire
- Structures d'index

12.1 Architecture générale d'un DBMS

- **Gestion du stockage**
- Traitement des requêtes
- Traitement des transactions

Hiérarchie des supports "mémoires"



Supports "mémoire"

- Le temps d'accès à la mémoire et la vitesse des disques ne respectent pas la loi de Moore
- *Plusieurs ordres de magnitude* entre les temps d'accès et les capacités des différents types de support
- La nature du support est un critère de choix pour les modes de représentation et les algorithmes

Cache

- Circuit intégré (parfois sur le même "chip" que le microprocesseur)
- Contenu : instructions machine et données (copie d'une partie de la mémoire principale)
- Capacité réduite : giga byte
- Accès rapide:
  - Lecture et écriture entre processeur et cache: moins de 10 nano-secondes (10^{-8} secondes)
  - Temps de transfert entre mémoire principale et cache : 100 nano-secondes

Mémoire principale

- Contenu: instructions, données à modifier
- Capacité : giga bytes
- Accès :
  - Aléatoire (temps constant)
  - Temps d'accès : 10 → 100 nano-secondes

Mémoire secondaire (disque)

- Grande capacité mais temps d'accès plus lents : 10^5 fois plus lent (10-30 millisecondes pour lire ou écrire un block)
  - → 100 fois plus de capacités
  - → 1000 fois plus lent

- Mémoire virtuelle:
  - Extension de la mémoire principale (4 giga bytes adressable avec 32 bits)
  - Organisée par blocks (pages): 4 → 56 K bytes
  - Peu utilisée par les grands DBMS

Mémoires tertiaires

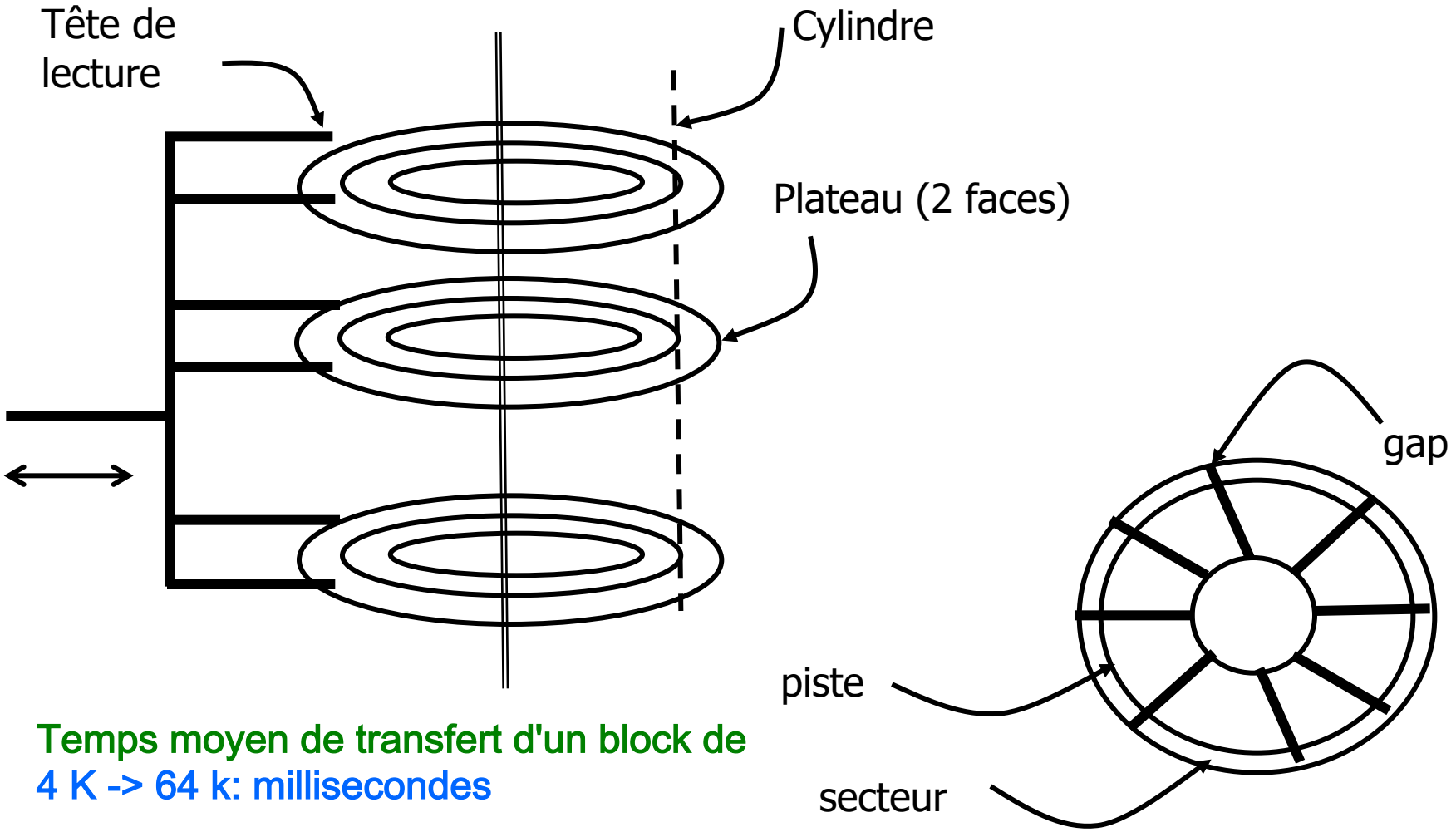
- Cassettes magnétiques, disque optiques, "Tape silos", "juke box" (rack de CD-ROM ou DVD-ROM)

  → 1000 à 10000 fois plus de capacités

  → 1000 à 10000 fois plus lent

  Remarque : *les mémoires secondaires et tertiaires sont rémanentes alors que la mémoire principale est volatile*

Organisation des disques



Temps moyen de transfert d'un block de 4 K -> 64 k: millisecondes

12.2 Utilisation efficace de la mémoire secondaire

- Meilleurs algorithmes en mémoire principale ne sont pas nécessairement les meilleurs algorithmes en mémoire secondaire
- Nombre d'accès à des blocks disque est une bonne approximation du temps d'un algorithme

→ Objectif : ***minimiser les accès à des blocks sur disque***

Exemple du tri-fusion

- **Algorithme**:

  Fusion deux par deux des listes triées en répétant le processus suivant jusqu'à ce que une des listes devient vide:
  - Recherche de la plus petite des clés en tête des listes
  - Retrait de cette clé et écriture sur la sortie standard

- **Exemple**:

Étape	Liste 1	Liste 2	Sortie
start	1,3,4,6	2,5,7,8	rien
1	3,4,6	2,5,7,8	1
2	3,4,6	5,7,8	1,2
3	4,6	5,7,8	1,2,3
4	6	5,7,8	1,2,3,4
5	6	7,8	1,2,3,4,5
6	rien	7,8	1,2,3,4,5,6
7	rien	rien	1,2,3,4,5,6, 7,8

Exemple du tri-fusion (suite)

- **Induction**:

  Toute liste de plus de deux éléments est décomposée en deux listes de "même" taille et l'algorithme de fusion est appliquée récursivement

- **Complexité**: **O(nlogn)**

Tri-fusion en mémoire secondaire

- **Étape 1** :

  Tri en mémoire principale (avec un tri rapide) des différentes parties de la liste

  → génération de *n* sous liste triées

  → Chaque bloc est chargé exactement **deux fois** en mémoire principale (**$\log_2 n$** fois avec l'algorithme classique)

- **Étape 2** : fusion de l'ensemble des sous-listes simultanément
  - Chargement en mémoire du premier block de chaque sous liste
  - Retrait et écriture sur une partition du plus petit des éléments en tête des listes en mémoire

  Partition de sortie pleine → écriture sur disque

  Sous liste vide → chargement du bloc suivant

Technique de réduction du temps d'accès en mémoire secondaire

- Regrouper les blocks accédées simultanément sur le même cylindre → réduction du temps de recherche et de latence
- Répartition des données sur plusieurs petits disques → augmente le nombre de blocks qui peuvent être accédés simultanément
- Utilisation d'un algorithme d'ordonnancement des accès dans le DBMS ou le contrôleur du disque plutôt que l'OS
  → optimisation de l'ordre d'accès aux blocks
- Chargement de blocks de manière anticipées
- Utilisation de clusters et d'index

Les clusters

Un cluster est un regroupement dans un même bloc disque des lignes d'une ou plusieurs tables ayant une caractéristique commune (une même valeur dans une ou plusieurs colonnes) constituant la clé du cluster

Objectifs :

- accélérer la jointure selon la clé de cluster des tables concernées
- accélérer la sélection des lignes d'une table ayant même valeur de clé
- économiser de la place: chaque valeur de la clé du cluster ne sera stockée qu'une seule fois

Les clusters (suite)

- Le regroupement en cluster est totalement transparent à l'utilisateur.
- Pour que l'on puisse mettre une table en cluster il faut que l'une au moins de ses colonnes soit définie comme obligatoire (NOT NULL).
- On peut indexer les colonnes d'une table en cluster

12.3 Structures d'index

Dans le modèle relationnel les sélections peuvent être faites en utilisant le contenu de n'importe quelle colonne. Soit la requête:

```
SELECT * FROM emp WHERE nom = 'MARTIN'
```

Pour retrouver les lignes pour lesquelles nom est égal à MARTIN on peut balayer toute la table.

Un tel moyen d'accès conduit à des temps de réponse prohibitifs

→ **Utilisation d'index**

Index (principes)

- Un index sera matérialisé par la création de blocs disque contenant des couples (valeurs d'index, numéro de bloc) donnant le numéro de bloc disque dans lequel se trouvent les lignes correspondant à chaque valeur d'index

- L'adjonction d'un index à une table ralentit les mises à jour (insertion, suppression, modification de la clé) mais accélère beaucoup la recherche d'une ligne dans la table

- L'index accélère la recherche d'une ligne à partir d'une valeur donnée de clé, mais aussi la recherche des lignes ayant une valeur d'index supérieure ou inférieure à une valeur donnée, car les valeurs de clés sont triées dans l'index.

Index : exemples

- Les requêtes suivantes bénéficieront d'un index sur le champ **n_dept:**

```
SELECT * FROM emp WHERE num = 16034  ;
SELECT * FROM emp WHERE num >= 27234 ;
SELECT * FROM emp WHERE num BETWEEN 16034  AND 27234;
```

- Un index est utilisable même si le critère de recherche est constitué seulement du début de la clé : La requête suivante bénéficiera d'un index sur la colonne nom.

```
SELECT * FROM emp WHERE nom LIKE 'M%';
```

- Si le début de la clé n'est pas connu, l'index est inutilisable

  Exemple: `SELECT * FROM emp WHERE nom LIKE '%M%';`

Valeurs NULL et Conversions

- Les valeurs NULL ne sont pas représentées dans l'index

  → minimiser le volume nécessaire pour stocker l'index

- L'index n'est utilisable que si *le critère de sélection est le contenu de la colonne indexée*, sans aucune transformation.

  Exemple un index sur salaire ne sera pas utilisé pour la requête :

  ```
  SELECT * FROM emp WHERE salaire * 12 > 300000;
  ```

  Mais sera utilisé pour la requête :

  ```
  SELECT * FROM emp WHERE salaire > 300000/12;
  ```

- Attention aux conversions de type qui peuvent empêcher l'utilisation de l'index (e.g., chaîne en date,…)

  Exemple :

  ```
  SELECT * FROM emp WHERE embauche LIKE '…'
  ```

  sera évalué en … `WHERE TO_CHAR(embauche)` …

  Le critère de recherche est une fonction de embauche → l'index est inutilisable

Choix des index

- Indexer en priorité :
  1. les clés primaires
  2. les colonnes servant de critère de jointure
  3. les colonnes servant souvent de critère de recherche

- Ne pas indexer :
  1. les colonnes contenant peu de valeurs distinctes (index peu efficace)
  2. les colonnes fréquemment modifiées

Différents niveaux de stockage des données

- Attributs: champs (colonnes)
- Ensemble d'attributs : record
  - Taille fixe
  - Taille variable (**`varchar`**,…)
- Ensemble de records : bloc
  - "Header" des records:
    - Pointeur vers le schéma
    - Longueur des records
    - "Timestamps": dernière modification ou lecture,
    - …
  - "Header" des blocs:
    - Pointeur vers d'autres blocs (même indexation)
    - Bloc Id
    - "Timestamps": dernière modification ou lecture
    - …