Stockage et manipulation de données : des fichiers aux bases de données.

I Données non relationnelles : le système de fichiers d'un OS Libre (Linux)

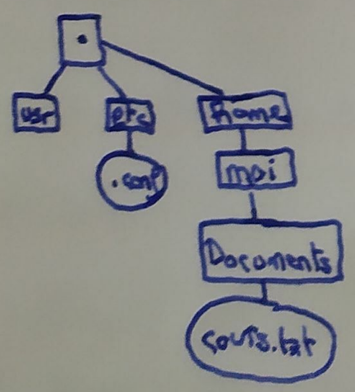
A. Structure générale

Def: Un **fichier** est la représentation de données sous forme d'une suite de bits.

Exemple: un fichier texte encode une suite de caractères par une suite de bits en utilisant l'encodage ASCII ou UTF-8.

Def: Une **arborescence de fichiers** est une représentation organisée de plusieurs fichiers dans un arbre. Les fichiers sont représentés par des feuilles, et les autres noeuds de l'arbre sont appelés **dossiers**.

Exemple:



□ : dossier
○ : fichier

Note: On peut voir un dossier comme un fichier représentant ses enfants dans l'arborescence.

B. Représentation machine

Def: Un **bloc de données** est un espace contigu de taille fixe sur le disque

Def: Un **i-node** est une entrée de taille fixe décrivant un fichier de manière compréhensible par le système d'exploitation. Il contient :

- Des attributs du fichiers (droits de lecture/écriture, nom, etc...)
- Une liste de blocs de données correspondant aux espaces servant au stockage du contenu du fichier.

Note: Afin de pouvoir gérer les fichiers volumineux, les 3 dernières entrées de la liste de blocs pointent vers une liste de bloc, une liste de listes de blocs et une liste de listes de listes de blocs.

Un dossier est simplement représenté comme un fichier dont le contenu est une liste d'i-nodes.

C. Manipulation des données d'un système de fichiers

Def: Un **descripteur de fichier** est une structure maintenue par le système d'exploitation et permettant aux programmes s'exécutant sur la machine d'interagir avec le contenu d'un fichier.
Il contient, entre autres, l'inode du fichier ainsi qu'un curseur indiquant l'emplacement de lecture/écriture dans le fichier.

Syntaxe:

Effet	Python	C
Crée un descripteur de fichier	`f = open('filename', 'mode')`	`fptr = fopen('filename', 'mode');` (de type FILE*)
Libère un descripteur de fichier	`f.close`	`fclose(fptr);`

Effet	Python	C
Lire une partie du fichier	`x=f.read(size)`	`fscanf(fptr, "%d", &x)`
Ecrire dans le fichier	`f.write('data')`	`fprintf(fptr, "%s", 'data')`

D. Autres représentations non relationnelles

Def: Un fichier d'archive est un fichier dont les données représentent une arborescence de fichiers. Il permet, contrairement au système de fichier, de représenter l'arborescence comme une unique suite de bits, ce qui permet l'échange d'arborescences entre différentes machines.

Exemples: Les fichiers de format .zip, .rar, ... sont des fichiers d'archive.

Def: Un système de gestion de version permet de maintenir, en plus d'une arborescence, des versions antérieures de l'arborescence, pouvant être restaurées par l'utilisateur.

Exemple: git

Développement 1: Représentation d'une arborescence dans git.

II Données relationnelles : Bases de données

A. Structure générale

Def: Un schema relationnel est une représentation de la structure d'une base de données. Il contient la donnée de plusieurs relations (ou tables), elles-même décrites par une liste d'attributs.

Les données stockées dans une base de données sont des tuples contenant une valeur pour chacun des attributs d'une table.

Exemple:

T_1

nom	rue	quartier
John	Av. St Michel	5
Alice	Rue de la Cité	1
Bob	Rue St-Jacques	5
Marie	Av. St Michel	5

← attributs →

tuples (données)

T_2

ligne	quartier
7	5
7	1
14	1
14	13

B. Manipulation des bases de données avec SQL

Création d'une table:

On crée une table avec la commande

```
CREATE TABLE table (
    attribut1 type1 contraintes1,
    attribut2 type2 contraintes2,
    CONSTRAINT nom1 contrainte_table1,
    ...
)
```

Exemple:

```
CREATE TABLE T1 (
    nom VARCHAR(20) PRIMARY KEY,
    rue VARCHAR(40),
    quartier INTEGER(2) CHECK (quartier < 21)
)
```

Requêtes de lecture de données :

Pour récupérer le contenu d'une table, on effectue la requête

SELECT * FROM t

De plus, SQL permet d'utiliser de nombreuses primitives permettant d'affiner les données récupérées :

opération	syntaxe	exemple
projection sur une liste d'attributs	SELECT attr1, attr2 FROM t	SELECT nom, rue FROM T1
sélection	SELECT * FROM t WHERE condition	SELECT * FROM T2 WHERE ligne = 7
renommage (d'un attribut ou d'une table)	SELECT attr AS nom FROM t	SELECT quartier AS arrondissement FROM T2
jointure	SELECT * FROM t1 JOIN t2 ON condition	SELECT * FROM T1 JOIN T2 ON T1.quartier = T2.quartier
aggrégation	SELECT count(*) AS n FROM t GROUP BY attr1, attr2,...	SELECT rue, count(*) FROM T1 GROUP BY rue (AS n)
Sélection sur les aggrégats	SELECT attr * FROM t GROUP BY attr HAVING count(*) < 5	SELECT rue FROM T1 GROUP BY rue HAVING count(*) > 1.

On peut bien sur combiner ces opérations. De plus, la table t peut être remplacée par le résultat d'une sous-requête.

Modification des données :

- On insère des données avec la commande

  INSERT INTO t(attr1, attr2,...) VALUES (value1, value2,...)

  ou INSERT INTO t(attr1, attr2, ...) (SELECT ...) ← requête de lecture

- On supprime des données avec la commande

  DELETE FROM t WHERE condition

- On met à jour des tuples avec la commande

  UPDATE t SET attr = value WHERE condition,

C. Dépendances fonctionnelles et formes normales

Déf: Une dépendance fonctionnelle $A_1, \dots, A_k \to B_1, \dots, B_m$ sur une relation R est une restriction des tuples possibles sur R forçant les tuples en accord sur les A_i à l'être sur les B_j.

Exemple: La dépendance fonctionnelle rue → quartier est vérifiée dans T_1.

Le maintient de ces dépendances peut limiter l'efficacité des requêtes de lecture ou modification des données.

Exemple: Si Av. St-Michel est maintenant dans le quartier 6 pour T_1, il faudra modifier plusieurs tuples.

Afin de limiter ces problèmes, il existe des formes normales de schéma de base de données limitant ces anomalies.

Déf: Un **clé** $A_1, \dots, A_k$ **d'une relation R** est un ensemble minimal d'attributs tel que $A_1, \dots, A_k \to B$ pour tout attribut B de R

Un sur-ensemble de $A_1, \dots, A_k$ est une super-clé.

Déf: Une relation R est en **BCNF** si pour toutes ses dépendances fonctionnelles $A_1, \dots, A_k \to B_1, \dots, B_m$, $A_1, \dots, A_k$ est une super-clé de R.

R est en 3NF si pour toute dépendance fonctionnelle $A_1, \dots, A_k \to B_1, \dots, B_m$, $A_1, \dots, A_k$ est une super-clé de R ou $\forall j$, B_j appartient à une clef.

Théorème: Toute relation R peut être décomposée en $R_1, \dots, R_p$ toutes en forme 3NF, sans perte d'information ni de dépendance.

Développement 2 : Algorithme de décomposition en 3NF