Objectifs

- Comprendre le mécanisme des requêtes récursives en SQL.
- Manipuler des structures hiérarchiques (employés, catégories).
- Afficher les données sous forme d'arborescence avec profondeur.
- Réaliser un cas d'usage métier : notification hiérarchique pour validation de congés.

Exemple guidé : Arborescence d'employés

Création de la table `employes`

```
CREATE TABLE employes (
    id INT PRIMARY KEY,
    nom VARCHAR(50),
    manager_id INT,
    FOREIGN KEY (manager_id) REFERENCES employes(id)
);
```

Insertion de données

```
INSERT INTO employes (id, nom, manager_id) VALUES
(1, 'Alice', NULL),
(2, 'Bob', 1),
(3, 'Charlie', 1),
(4, 'David', 2),
(5, 'Eva', 2),
(6, 'Frank', 3);
```

Requête récursive pour trouver tous les subordonnés de Alice

```
WITH RECURSIVE sous_ordres AS (
    SELECT id, nom, manager_id, 0 AS niveau
    FROM employes
    WHERE nom = 'Alice'

    UNION ALL

    SELECT e.id, e.nom, e.manager_id, s.niveau + 1
    FROM employes e
    JOIN sous_ordres s ON e.manager_id = s.id
)
```

```
SELECT * FROM sous_ordres;
```

Explication

- La première requête (ancrage) récupère l'employé de départ.
- La seconde (récursive) récupère les employés dont le manager est dans les résultats précédents.
- Le champ `niveau` permet de représenter la profondeur dans l'arbre hiérarchique.

TP Étudiant : Arborescence de catégories

Création de la table `categories`

```
CREATE TABLE categories (
    id INT PRIMARY KEY,
    nom VARCHAR(100),
    parent_id INT,
    FOREIGN KEY (parent_id) REFERENCES categories(id)
);
```

Insertion de données

```
INSERT INTO categories (id, nom, parent_id) VALUES
(1, '  lectronique  ', NULL),
(2, 'Informatique', 1),
(3, 'Smartphones', 1),
(4, 'Ordinateurs portables', 2),
(5, 'Composants', 2),
(6, 'M moire RAM', 5),
(7, 'Disques SSD', 5),
(8, 'Accessoires', 1);
```

Travail demandé

1. Écrire une requête récursive pour récupérer toutes les sous-catégories de **Électronique**.
2. Ajouter un champ `niveau` pour afficher la profondeur de chaque catégorie.
3. Afficher une arborescence textuelle à l'aide de `REPEAT()` ou `LPAD()` selon le SGBD :

```
- Électronique
  - Informatique
```



```
      - Ordinateurs portables
      - Composants
        - Mémoire RAM
        - Disques SSD
    - Smartphones
    - Accessoires
```

Questions théoriques

- Quelle est la différence entre une requête récursive et une jointure classique ?
- Que se passe-t-il en cas de boucle infinie ? (ex : un parent se référence lui-même)
- Comment limiter la profondeur de la récursion avec une condition ?

Livrables attendus

- Script SQL (création + insertion + requêtes).
- Résultat de la requête récursive (arborescence affichée).
- Réponses aux questions théoriques.

Exemple avancé : Gestion des notifications et validations de congés avec hiérarchie ascendante

Ce script SQL illustre une gestion avancée des demandes de congés en entreprise, avec prise en compte de la hiérarchie managériale pour validation progressive et envoi automatique de notifications.

Création des tables

On commence par créer les tables nécessaires :

- `employes` : pour les employés avec leur manager direct.
- `conges` : pour les demandes de congé.
- `validations` : pour suivre la validation par chaque manager.
- `notifications` : pour informer les managers et employés des actions.

```
DROP TABLE IF EXISTS notifications;
DROP TABLE IF EXISTS validations;
DROP TABLE IF EXISTS conges;
DROP TABLE IF EXISTS employes;

CREATE TABLE employes (
```



```
    id INTEGER PRIMARY KEY,
    nom VARCHAR(50),
    manager_id INTEGER,
    FOREIGN KEY (manager_id) REFERENCES employes(id)
);

CREATE TABLE conges (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    employe_id INTEGER,
    date_debut DATE,
    date_fin DATE,
    status VARCHAR(20) DEFAULT 'en attente',
    FOREIGN KEY (employe_id) REFERENCES employes(id)
);

CREATE TABLE validations (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    conge_id INTEGER,
    manager_id INTEGER,
    statut VARCHAR(20) DEFAULT 'en attente',
    date_validation DATETIME,
    FOREIGN KEY (conge_id) REFERENCES conges(id),
    FOREIGN KEY (manager_id) REFERENCES employes(id)
);

CREATE TABLE notifications (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    employe_id INTEGER,
    message TEXT,
    date_notification DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (employe_id) REFERENCES employes(id)
);
```

Insertion des employés avec hiérarchie

```
INSERT INTO employes (id, nom, manager_id) VALUES
(1, 'Alice', NULL),         -- PDG
(2, 'Bob', 1),              -- Manager de Charlie
(3, 'Charlie', 2),          -- Manager de David
(4, 'David', 3);            -- Employé
```

David fait une demande de congé

```
INSERT INTO conges (employe_id, date_debut, date_fin)
VALUES (4, '2025-06-01', '2025-06-10');
```

Ajout automatique des managers pour validation (hiérarchie ascendante)

Cette requête récursive remonte la chaîne des managers pour la demande de congé et insère une ligne dans `validations` pour chaque manager.

```
INSERT INTO validations (conge_id, manager_id)
SELECT
    (SELECT id FROM conges WHERE employe_id = 4 ORDER BY id DESC LIMIT 1),
    id
FROM (
    WITH RECURSIVE managers AS (
        SELECT id, manager_id FROM employes WHERE id = 4
        UNION ALL
        SELECT e.id, e.manager_id FROM employes e JOIN managers m ON e.id = m.manager_id
    )
    SELECT * FROM managers WHERE id != 4
) AS manager_list;
```

Validation successive par chaque manager avec notification automatique

Manager Charlie valide :

```
UPDATE validations
SET statut = 'accepté', date_validation = CURRENT_TIMESTAMP
WHERE conge_id = 1 AND manager_id = 3;
```

Notification au manager suivant (Bob) :

```
INSERT INTO notifications (employe_id, message)
SELECT manager_id, 'Une demande de congé de David vous attend.'
FROM validations
WHERE conge_id = 1 AND statut = 'en attente'
ORDER BY manager_id ASC
LIMIT 1;
```

Manager Bob valide :

```
UPDATE validations
SET statut = 'accepté', date_validation = CURRENT_TIMESTAMP
WHERE conge_id = 1 AND manager_id = 2;
```

Notification au manager final (Alice) :

```
INSERT INTO notifications (employe_id, message)
SELECT manager_id, 'Une demande de congé de David vous attend.'
FROM validations
WHERE conge_id = 1 AND statut = 'en attente'
ORDER BY manager_id ASC
LIMIT 1;
```

Manager Alice valide :

```
UPDATE validations
SET statut = 'accepté', date_validation = CURRENT_TIMESTAMP
WHERE conge_id = 1 AND manager_id = 1;
```

Mise à jour finale du congé et notification à David

Si tous les managers ont validé, le statut du congé est mis à jour, et David est notifié.

```
UPDATE conges
SET status = 'accepté'
WHERE id = 1
  AND (
    SELECT COUNT(*) FROM validations WHERE conge_id = 1
  ) = (
    SELECT COUNT(*) FROM validations WHERE conge_id = 1 AND statut = 'accepté'
  );

INSERT INTO notifications (employe_id, message)
SELECT employe_id, 'Votre demande de congé a été validée par tous les managers.'
FROM conges
WHERE id = 1
  AND status = 'accepté';
```

Affichage des résultats

Statut final de la demande de congé de David :

```
SELECT * FROM conges WHERE employe_id = 4;
```

Validations liées à cette demande de congé :

```
SELECT v.id, v.conge_id, e.nom AS manager_nom, v.statut, v.date_validation
FROM validations v
JOIN employes e ON v.manager_id = e.id
WHERE v.conge_id = 1
ORDER BY v.manager_id;
```

Notifications reçues par David :

```
SELECT * FROM notifications WHERE employe_id = 4 ORDER BY date_notification;
```

Notes

- La table `validations` suit l'état de validation par chaque manager.
- Les notifications sont envoyées automatiquement au manager suivant après chaque validation.
- La validation est ascendante, du manager direct à la direction générale.
- Le congé ne devient effectif que si tous les managers l'ont accepté.