# Cours SQL - BTS CIEL - IR

## Objectifs

- Comprendre les concepts de base des bases de données relationnelles
- Maîtriser les commandes SQL essentielles (CRUD)
- Savoir utiliser les fonctions d'agrégation et les jointures
- Appliquer SQL dans un contexte professionnel, notamment en utilisant des concepts avancés comme les sous-requêtes et les jointures multiples.

## Pré-requis

- Connaissances de base en informatique et algorithmique
- Familiarité avec les concepts de base de données

## Plan du Cours



---

## 1. Introduction à SQL

**SQL (Structured Query Language)** est le langage de programmation standard pour interroger et manipuler des bases de données relationnelles. SQL permet de :

- Créer, modifier et supprimer des bases de données et des tables.
- Ajouter, modifier, supprimer et récupérer des données.
- Gérer les permissions des utilisateurs.

### Concepts clés

| Terme | Définition |
|---|---|
| **BDD** | Base de données (ensemble structuré de données) |
| **Table** | Structure organisée en lignes (tuples) et colonnes |
| **Colonne** | Type de donnée spécifique dans une table (ex : `email VARCHAR`) |
| **Ligne (Tuple)** | Enregistrement complet dans une table |
| **Clé Primaire (PK)** | Identifie de manière unique chaque ligne dans une table |

| Terme | Définition |
| --- | --- |
| **Clé Étrangère (FK)** | Relie une table à une autre via une référence à la clé primaire d'une autre table |

## 2. Installation et Configuration

### Outils recommandés

- **XAMPP** : Package qui inclut Apache, MySQL, PHP, etc. pour un environnement local de développement.
- **phpMyAdmin** : Interface graphique pour gérer les bases de données MySQL via un navigateur.

### Étapes d'installation

1. Téléchargez et installez XAMPP.
2. Lancez Apache et MySQL via le panneau de contrôle XAMPP.
3. Accédez à phpMyAdmin via `http://localhost/phpmyadmin` pour gérer vos bases de données.

## 3. Création et Gestion des Bases de Données et Tables

### Création d'une base de données

```sql
-- Créer une base de données
CREATE DATABASE bibliotheque;

-- Sélectionner une base de données
USE bibliotheque;

-- Supprimer une base de données (ATTENTION : irréversible !)
DROP DATABASE bibliotheque;
```

> **La Base de données `bibliotheque` est utilsée dans tous les exemples.**

### Création et gestion des tables

```sql
-- Créer une table livres
CREATE TABLE livres (
    id INT AUTO_INCREMENT PRIMARY KEY,
    titre VARCHAR(255) NOT NULL,
    auteur VARCHAR(255) NOT NULL,
    annee_publication INT,
    isbn VARCHAR(20) UNIQUE
);

-- Ajouter une colonne
ALTER TABLE livres ADD COLUMN prix DECIMAL(10,2);

-- Supprimer la table livres
DROP TABLE livres;
```

Diagramme Entité-Relation (ERD) Exemple

```
erDiagram
  LIVRES {
      INT id PK
      VARCHAR titre
      VARCHAR auteur
      INT annee_publication
      VARCHAR isbn "UNIQUE"
      DECIMAL prix
  }
```

## TP 1.1 : Création de Base de Données et Tables

1. Créez une base de données appelée `ecole`.
2. Dans cette base de données, créez une table `etudiants` avec les colonnes suivantes :
    - `id` (INT, AUTO_INCREMENT, PRIMARY KEY)
    - `nom` (VARCHAR(255), NOT NULL)
    - `prenom` (VARCHAR(255), NOT NULL)
    - `email` (VARCHAR(255), NOT NULL)
    - `age` (INT)
3. Ajoutez une colonne `moyenne` de type DECIMAL(10,2) à la table `etudiants`.

```
erDiagram
    ETUDIANTS {
      INT id PK
      VARCHAR nom
      VARCHAR prenom
      VARCHAR email "UNIQUE"
      INT age
      DECIMAL moyenne
    }
```

## 4. Manipulation des Données (CRUD)

Insertion (CREATE)

```
-- Insertion simple
INSERT INTO livres (titre, auteur, annee_publication, isbn, prix)
VALUES ('Le titre 1', 'Auteur 1', 1983, '9782070612758', 8.50);

-- Insertion multiple
INSERT INTO livres (titre, auteur, annee_publication, isbn, prix) VALUES
('Le titre 2', 'Auteur 2', 2005, '9782070612702', 12.90),
```

```
('Le titre 3', 'Auteur 3', 1994, '9782070612703', 7.95),
('Le titre 4', 'Auteur 4', 2005, '9782070612704', 17.50);
```

### Sélection des données (READ)

```sql
-- Sélectionner toutes les colonnes
SELECT * FROM livres;

-- Sélectionner certaines colonnes
SELECT titre, auteur FROM livres;

-- Avec condition
SELECT titre, auteur FROM livres WHERE annee_publication > 2000;

-- Avec tri
SELECT titre, auteur FROM livres ORDER BY auteur ASC;

-- Limiter les résultats
SELECT titre, auteur FROM livres LIMIT 2;

-- Éviter les doublons
SELECT DISTINCT annee_publication FROM livres;
```

### Mise à jour des données (UPDATE)

```sql
-- Modifier un enregistrement
UPDATE livres SET prix = 9.00 WHERE titre = 'Le titre 1';

-- Modifier plusieurs champs
UPDATE livres SET prix = 23.90, isbn = '9782070612701' WHERE id = 1;
```

### Suppression des données (DELETE)

```sql
-- Supprimer un enregistrement
DELETE FROM livres WHERE id = 1;

-- Supprimer avec condition
DELETE FROM livres WHERE prix < 18 AND auteur = 'Auteur 4';
```

## TP 1.2 : Manipulation des Données

1. Insérez 6 étudiants dans la table `etudiants`.

| nom | prenom | email | age | moyenne |
|---|---|---|---|---|

| nom | prenom | email | age | moyenne |
|---|---|---|---|---|
| Carnus | Charles | charles.carnus@carnus.fr | 18 | 11.9 |
| Martin | Paul | paul.martin@carnus.fr | 21 | 13.5 |
| Durand | Alice | alice.durand@carnus.fr | 19 | 14.0 |
| Dupont | Jean | jean.dupont@carnus.fr | 18 | 15.2 |
| Bernard | Clara | clara.bernard@carnus.fr | 21 | 12.8 |
| Simon | Alex | alex.petit@carnus.fr | 20 | 16.1 |

2. Sélectionnez tous les étudiants dont l'âge est supérieur à 20 ans.
3. Mettez à jour l'âge de l'étudiant dont le nom est 'Martin' à 22 ans.
4. Supprimez l'étudiant dont le nom est 'Durand'.
5. Sélectionnez les noms et prénoms des étudiants, triés par nom de famille en ordre décroissant.

## 5. Fonctions d'Agrégation

Les fonctions d'agrégation permettent de résumer et analyser les données d'une manière plus globale.

```
-- Compter le nombre de livres
SELECT COUNT(*) AS nombre_livres FROM livres;

-- Calculer la moyenne des prix
SELECT AVG(prix) AS prix_moyen FROM livres;

-- Trouver le prix minimum et maximum
SELECT MIN(prix) AS prix_min, MAX(prix) AS prix_max FROM livres;

-- Calculer la somme des prix
SELECT SUM(prix) AS total_prix FROM livres;
```

## TP 1.3 : Fonctions d'Agrégation

1. Comptez le nombre total d'étudiants dans la table `etudiants`.
2. Calculez l'âge moyen des étudiants.
3. Trouvez l'âge minimum et maximum des étudiants.
4. Calculez la somme des âges de tous les étudiants.
5. Comptez le nombre d'étudiants dont l'âge est supérieur à 20 ans.

## 6. Clauses Avancées

### GROUP BY et HAVING

```
-- Grouper les livres par auteur
SELECT auteur, COUNT(*) AS nb_livres
FROM livres
```

```sql
GROUP BY auteur;

-- Filtrer les groupes (avoir plus de 2 livres par auteur)
SELECT auteur, COUNT(*) AS nb_livres
FROM livres
GROUP BY auteur
HAVING COUNT(*) > 2;
```

### ORDER BY

```sql
-- Tri croissant
SELECT * FROM livres ORDER BY auteur ASC;

-- Tri décroissant
SELECT * FROM livres ORDER BY prix DESC;

-- Tri multiple (par auteur croissant et prix décroissant)
SELECT * FROM livres ORDER BY auteur ASC, prix DESC;
```

## TP 1.4 : Clauses Avancées

1. Groupez les étudiants par âge et comptez le nombre d'étudiants pour chaque âge.
2. Trouvez les âges pour lesquels il y a plus de 2 étudiants.
3. Sélectionnez tous les étudiants et triez-les par nom de famille en ordre croissant.
4. Sélectionnez tous les étudiants et triez-les par âge en ordre décroissant.
5. Sélectionnez tous les étudiants et triez-les par nom de famille en ordre croissant et par âge en ordre décroissant.

## 7. Les Jointures

Les jointures permettent de combiner des données provenant de plusieurs tables liées entre elles (associer plusieurs tables dans une même requête).

### Exemple de Structure de Tables

```sql
-- Table abonnés
CREATE TABLE abonnes (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nom VARCHAR(255),
  prenom VARCHAR(255)
);

-- Table emprunts (table de liaison)
CREATE TABLE emprunts (
  id INT AUTO_INCREMENT PRIMARY KEY,
  abonne_id INT,
  livre_id INT,
  date_emprunt DATE,
```

```
  date_retour DATE,
  FOREIGN KEY (abonne_id) REFERENCES abonnes(id),
  FOREIGN KEY (livre_id) REFERENCES livres(id)
);
```

```
erDiagram
    ABONNE {
      INT id PK
      VARCHAR nom
      VARCHAR prenom
    }
    LIVRE {
      INT id PK
      VARCHAR titre
    }
    EMPRUNT {
      INT id PK
      INT abonne_id FK
      INT livre_id FK
      DATE date_emprunt
      DATE date_retour
    }
    ABONNE ||--o{ EMPRUNT : fait
    LIVRE  ||--o{ EMPRUNT : concerne
```

Structure de la requête :

```
-- INNER JOIN : seuls les enregistrements qui ont une correspondance
SELECT a.nom, l.titre
FROM abonnes a
INNER JOIN emprunts e ON a.id = e.abonne_id
-- mot-clé ON pour indiquer à la base de données quelles colonnes doivent
être utilisées pour faire correspondre les enregistrements
INNER JOIN livres l ON e.livre_id = l.id;
```

1. **Sélection des colonnes :**

   ```
   SELECT a.nom, l.titre
   ```

   - Cette partie indique que la requête doit retourner deux colonnes :
     - `a.nom`, qui est le nom de l'abonné (la colonne `nom` de la table `abonnes`).
     - `l.titre`, qui est le titre du livre (la colonne `titre` de la table `livres`).

2. **Tables utilisées :**

```
FROM abonnes a
```

- La requête commence par sélectionner la table `abonnes` et l'alias `a` est utilisé pour la désigner plus facilement dans la suite de la requête.

3. **Première jointure (INNER JOIN avec la table `emprunts`) :**

```
INNER JOIN emprunts e ON a.id = e.abonne_id
```

- Cette partie réalise une jointure interne (`INNER JOIN`) entre la table `abonnes` (alias `a`) et la table `emprunts` (alias `e`).
- La condition de jointure est `a.id = e.abonne_id`. Cela signifie qu'on relie chaque abonné à ses emprunts en utilisant l'ID de l'abonné (`a.id`) et l'ID de l'abonné dans la table `emprunts` (`e.abonne_id`).
- Cette jointure assure que seuls les abonnés ayant effectué un emprunt apparaissent dans le résultat.

4. **Deuxième jointure (INNER JOIN avec la table `livres`) :**

```
INNER JOIN livres l ON e.livre_id = l.id;
```

- Ici, on effectue une autre jointure interne entre la table `emprunts` (alias `e`) et la table `livres` (alias `l`).
- La condition de jointure est `e.livre_id = l.id`, ce qui signifie que chaque emprunt est lié à un livre spécifique via l'ID du livre (`l.id`).
- Cette jointure permet de récupérer les informations relatives aux livres empruntés.

> La requête retourne la liste des abonnés ayant emprunté des livres, en affichant pour chaque abonné son **nom** et le **titre** du livre qu'il a emprunté.

---

## TP 1.5 : Jointures

1. Créez une table `cours` avec les colonnes suivantes :
   - `id` (INT, AUTO_INCREMENT, PRIMARY KEY)
   - `nom` (VARCHAR(255), NOT NULL)
   - `description` (TEXT)
2. Créez une table `inscriptions` avec les colonnes suivantes :
   - `id` (INT, AUTO_INCREMENT, PRIMARY KEY)
   - `etudiant_id` (INT, FOREIGN KEY)
   - `cours_id` (INT, FOREIGN KEY)
   - `date_inscription` (DATE, DEFAULT CURRENT_DATE)
3. Insérez 3 cours dans la table `cours`.

| nom | description | coefficient |
|---|---|---|
| Bases de Données | Introduction aux BDD et SQL | 3 |
| Réseaux | Fondamentaux des réseaux informatiques | 3 |
| Cybersécurité | Sécurité des systèmes d"information | 2 |

4. Insérez 5 inscriptions dans la table `inscriptions`.

| etudiant_id | cours_id |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 2 | 1 |
| 2 | 3 |
| 3 | 2 |
| 3 | 3 |
| 4 | 1 |
| 4 | 2 |
| 4 | 3 |
| 5 | 3 |

5. Utilisez une jointure interne pour sélectionner les noms et prénoms des étudiants ainsi que les noms des cours auxquels ils sont inscrits.
6. Utilisez une jointure gauche pour sélectionner tous les étudiants, même ceux qui ne sont pas inscrits à un cours.
7. Utilisez une jointure droite pour sélectionner tous les cours, même ceux qui n'ont pas d'étudiants inscrits.
8. Trouvez les cours les plus populaires

```
erDiagram
    ETUDIANTS {
        INT id PK
        VARCHAR nom
        VARCHAR prenom
        VARCHAR email
        INT age
    }
    COURS {
        INT id PK
        VARCHAR nom
        TEXT description
    }
    INSCRIPTIONS {
```

```
        INT id PK
        INT etudiant_id FK
        INT cours_id FK
        DATE date_inscription
    }

    ETUDIANTS ||--o{ INSCRIPTIONS : inscrit
    COURS ||--o{ INSCRIPTIONS : concerne
```

## 8. Sous-Requêtes

Les sous-requêtes permettent d'effectuer une requête à l'intérieur d'une autre, ce qui est utile pour des filtres plus complexes.

```sql
-- Trouver les livres dont le prix est supérieur à la moyenne
SELECT titre, prix
FROM livres
WHERE prix > (SELECT AVG(prix) FROM livres);
```

## TP 1.6 : Sous-Requêtes

1. Sélectionnez les noms et prénoms des étudiants dont l'âge est supérieur à l'âge moyen des étudiants.
2. Sélectionnez les noms des cours qui ont plus de 2 inscriptions.
3. Sélectionnez les noms et prénoms des étudiants qui sont inscrits à un cours dont le nom commence par 'Info'.
4. Sélectionnez les noms des cours qui n'ont pas d'inscriptions.
5. Sélectionnez les noms et prénoms des étudiants qui sont inscrits à tous les cours.

---

## 9. Corrigés des TP SQL

### TP 1.1 : Création de Base de Données et Tables

```sql
-- 1. Créer la base
CREATE DATABASE ecole;
USE ecole;

-- 2. Créer la table
CREATE TABLE etudiants (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nom VARCHAR(255) NOT NULL,
  prenom VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE,
  age INT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```sql
-- 3. Ajouter la colonne moyenne
ALTER TABLE etudiants ADD COLUMN moyenne DECIMAL(10,2);
```

---

### TP 1.2 : Manipulation des Données

```sql
-- 1. Insérer 5 étudiants
INSERT INTO etudiants (nom, prenom, email, age, moyenne) VALUES
('Martin', 'Paul', 'paul.martin@example.com', 22, 13.5),
('Durand', 'Alice', 'alice.durand@example.com', 19, 14.0),
('Dupont', 'Jean', 'jean.dupont@example.com', 21, 15.2),
('Bernard', 'Clara', 'clara.bernard@example.com', 23, 12.8),
('Petit', 'Luc', 'luc.petit@example.com', 20, 16.1);

-- 2. Étudiants de plus de 20 ans
SELECT * FROM etudiants WHERE age > 20;

-- 3. Mettre à jour l'âge de Martin
UPDATE etudiants SET age = 23 WHERE nom = 'Martin';

-- 4. Supprimer l'étudiant Durand
DELETE FROM etudiants WHERE nom = 'Durand';

-- 5. Trier par nom décroissant
SELECT nom, prenom FROM etudiants ORDER BY nom DESC;
```

---

### TP 1.3 : Fonctions d'Agrégation

```sql
-- 1. Nombre total d'étudiants
SELECT COUNT(*) FROM etudiants;

-- 2. Âge moyen
SELECT AVG(age) FROM etudiants;

-- 3. Âge min et max
SELECT MIN(age), MAX(age) FROM etudiants;

-- 4. Somme des âges
SELECT SUM(age) FROM etudiants;

-- 5. Étudiants > 20 ans
SELECT COUNT(*) FROM etudiants WHERE age > 20;
```

---

### TP 1.4 : Clauses Avancées

```sql
-- 1. Grouper par âge
SELECT age, COUNT(*) AS nb_etudiants
FROM etudiants
GROUP BY age;

-- 2. Âges avec plus de 2 étudiants
SELECT age, COUNT(*) AS nb
FROM etudiants
GROUP BY age
HAVING COUNT(*) > 2;

-- 3. Trier par nom croissant
SELECT * FROM etudiants ORDER BY nom ASC;

-- 4. Trier par âge décroissant
SELECT * FROM etudiants ORDER BY age DESC;

-- 5. Nom croissant + âge décroissant
SELECT * FROM etudiants ORDER BY nom ASC, age DESC;
```

---

## TP 1.5 : Jointures

```sql
-- 1. Créer table cours
CREATE TABLE cours (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nom VARCHAR(255) NOT NULL,
  description TEXT
);

-- 2. Créer table inscriptions
CREATE TABLE inscriptions (
  id INT AUTO_INCREMENT PRIMARY KEY,
  etudiant_id INT,
  cours_id INT,
  date_inscription DATE DEFAULT CURRENT_DATE,
  FOREIGN KEY (etudiant_id) REFERENCES etudiants(id),
  FOREIGN KEY (cours_id) REFERENCES cours(id)
);

-- 3. Insérer 3 cours
INSERT INTO cours (nom, description) VALUES
('Informatique', 'Cours d'introduction à la programmation'),
('Mathématiques', 'Cours d'algèbre et d'analyse'),
('Physique', 'Cours de mécanique et d'optique');

-- 4. Insérer 5 inscriptions (exemple)
INSERT INTO inscriptions (etudiant_id, cours_id) VALUES
(1,1), (1,2), (2,1), (3,3), (4,2);
```

```sql
-- 5. Jointure interne
SELECT e.nom, e.prenom, c.nom AS cours
FROM etudiants e
INNER JOIN inscriptions i ON e.id = i.etudiant_id
INNER JOIN cours c ON i.cours_id = c.id;

-- 6. Jointure gauche (tous les étudiants)
SELECT e.nom, e.prenom, c.nom AS cours
FROM etudiants e
LEFT JOIN inscriptions i ON e.id = i.etudiant_id
LEFT JOIN cours c ON i.cours_id = c.id;

-- 7. Jointure droite (tous les cours)
SELECT e.nom, e.prenom, c.nom AS cours
FROM etudiants e
RIGHT JOIN inscriptions i ON e.id = i.etudiant_id
RIGHT JOIN cours c ON i.cours_id = c.id;

-- 8. Cours les plus populaires
SELECT c.nom, COUNT(i.etudiant_id) AS nb_inscrits
FROM cours c
LEFT JOIN inscriptions i ON c.id = i.cours_id
GROUP BY c.id, c.nom
ORDER BY nb_inscrits DESC;
```

---

## TP 1.6 : Sous-Requêtes

```sql
-- 1. Étudiants avec âge > moyenne
SELECT nom, prenom
FROM etudiants
WHERE age > (SELECT AVG(age) FROM etudiants);

-- 2. Cours avec plus de 2 inscriptions
SELECT c.nom
FROM cours c
WHERE c.id IN (
  SELECT cours_id
  FROM inscriptions
  GROUP BY cours_id
  HAVING COUNT(*) > 2
);

-- 3. Étudiants inscrits à un cours 'Info%'
SELECT DISTINCT e.nom, e.prenom
FROM etudiants e
JOIN inscriptions i ON e.id = i.etudiant_id
JOIN cours c ON i.cours_id = c.id
WHERE c.nom LIKE 'Info%';

-- 4. Cours sans inscriptions
```

```sql
SELECT nom
FROM cours
WHERE id NOT IN (SELECT cours_id FROM inscriptions);

-- 5. Étudiants inscrits à TOUS les cours
SELECT e.nom, e.prenom
FROM etudiants e
WHERE NOT EXISTS (
  SELECT *
  FROM cours c
  WHERE c.id NOT IN (
    SELECT i.cours_id FROM inscriptions i WHERE i.etudiant_id = e.id
  )
);
```

## 10. Tableau Récapitulatif des Commandes SQL

### Commandes de Base

| Commande | Description | Exemple |
|---|---|---|
| `CREATE DATABASE` | Créer une base de données | `CREATE DATABASE ma_base;` |
| `USE` | Sélectionner une base | `USE ma_base;` |
| `CREATE TABLE` | Créer une table | `CREATE TABLE users (id INT, nom VARCHAR(50));` |
| `INSERT INTO` | Ajouter des données | `INSERT INTO users (nom) VALUES ('Alice');` |
| `SELECT` | Lire des données | `SELECT * FROM users;` |
| `UPDATE` | Modifier des données | `UPDATE users SET nom='Bob' WHERE id=1;` |
| `DELETE` | Supprimer des données | `DELETE FROM users WHERE id=1;` |
| `WHERE` | Filtrer les résultats | `SELECT * FROM users WHERE age > 18;` |
| `ORDER BY` | Trier les résultats | `SELECT * FROM users ORDER BY nom;` |
| `LIMIT` | Limiter les résultats | `SELECT * FROM users LIMIT 10;` |

### Commandes Avancées

| Commande | Description | Exemple |
|---|---|---|
| `COUNT()` | Compter les lignes | `SELECT COUNT(*) FROM users;` |
| `AVG()` | Calculer une moyenne | `SELECT AVG(age) FROM users;` |

| Commande | Description | Exemple |
| --- | --- | --- |
| `SUM()` | Calculer une somme | `SELECT SUM(salaire) FROM employes;` |
| `MIN()/MAX()` | Valeur min/max | `SELECT MIN(age), MAX(age) FROM users;` |
| `GROUP BY` | Grouper les résultats | `SELECT ville, COUNT(*) FROM users GROUP BY ville;` |
| `HAVING` | Filtrer les groupes | `SELECT ville, COUNT(*) FROM users GROUP BY ville HAVING COUNT(*) > 2;` |
| `INNER JOIN` | Jointure interne | `SELECT * FROM a INNER JOIN b ON a.id = b.a_id;` |