Bases de Données

De L'AR à SQL
partie 2/2

Anis SAIED

IPEIN

2025/26

SQL

Structured Query Language

Langage universel pour gérer les bases de données relationnelles.

Définit, manipule et interroge les données.

Utilisé pour structurer les tables et gérer les accès.

Principales catégories de commandes SQL :

DDL (Data Definition Language) : Création et modification de la structure des bases de données (ex : ‘CREATE‘, ‘ALTER‘, ‘DROP‘).

DML (Data Manipulation Language) : Manipulation des données à l’intérieur des tables (ex : ‘SELECT‘, ‘INSERT‘, ‘UPDATE‘, ‘DELETE‘).

Commande SQL: CREATE

Définition et Syntaxe

Définition :

La commande `CREATE` est utilisée pour créer une nouvelle table dans une base de données.

Elle définit la structure de la table, y compris les colonnes et leurs types.

Syntaxe Générale :

```
CREATE TABLE TableName (
  Column1 DataType [constraints],
  Column2 DataType [constraints],
  ...
);
```

Types Possibles de Colonnes :

`INT` : Entier

`Text / VARCHAR(n)` : Chaîne de caractères de longueur variable (n est la longueur maximale)

`DATE` : Date

`REAL` : (ou FLOAT) Nombre à virgule flottante

`BOOLEAN` : Valeur booléenne (vrai/faux)

Commande SQL: CREATE

Contraintes sur les Colonnes

`PRIMARY KEY`

Assure l'unicité de la colonne et identifie chaque enregistrement de manière unique. Ex: `ID INT PRIMARY KEY`

`FOREIGN KEY`

Crée une relation entre deux tables en pointant vers une clé primaire d'une autre table. Ex: `FOREIGN KEY (ClientID) REFERENCES Clients(ID)`

`NOT NULL`

Empêche qu'une colonne ait une valeur `NULL`.
Ex: `Nom VARCHAR(100) NOT NULL`

`UNIQUE`

Garantit que tous les éléments d'une colonne sont uniques.
Ex : `Email VARCHAR(255) UNIQUE`

`CHECK`

Implique une condition qui doit être vraie pour chaque enregistrement.
Ex: `Age INT CHECK (Age > 0)`

`DEFAULT`

Définit une valeur par défaut pour une colonne si aucune valeur n'est spécifiée lors de l'insertion. Ex: `CreatedDate DATE DEFAULT CURRENT_DATE`

Commande SQL: CREATE

Exemple 1

Donner la commande SQL permettant de créer la table `Etudiant` du schéma relationnel:

`Etudiant (ID, Nom, Age, Email)`

Requête SQL :

```
CREATE TABLE Etudiant (
  ID INT PRIMARY KEY,
  Nom TEXT NOT NULL,
  Age INT CHECK (Age >= 18),
  Email VARCHAR(100) UNIQUE
);
```

ID	Nom	Age	Email

Table 1: Table `Etudiant` (Vide)

Commande SQL: CREATE

Exemple 2

Departements (ID, Nom)
Cours (ID, Titre, Description, #DepartementID, DateDebut)

Requêtes SQL :

```
CREATE TABLE Departements (
  ID INT PRIMARY KEY,
  Nom VARCHAR(100) NOT NULL
);

CREATE TABLE Cours (
  ID INT PRIMARY KEY,
  Titre VARCHAR(100) NOT NULL,
  Description TEXT,
  DepartementID INT,
  DateDebut DATE DEFAULT CURRENT_DATE,
  FOREIGN KEY (DepartementID) REFERENCES Departements(ID)
);
```

Commande SQL: DROP

Définition et Syntaxe

Définition :

La commande `DROP` est utilisée pour supprimer une table existante dans une base de données.

Cette action est irréversible, ce qui signifie que les données et la structure de la table seront définitivement supprimées.

Syntaxe Générale :

```
DROP TABLE TableName;
```

Commande SQL: DROP

Exemple 1

Donner la commande SQL permettant de supprimer la table `Etudiant` du schéma relationnel:

`Etudiant (ID, Nom, Age, Email)`

Requête SQL :

```
DROP TABLE Etudiant;
```

→ Table `Etudiant` supprimée, n'existe plus.

Commande SQL: ALTER TABLE

Définition et Syntaxe

Définition :

La commande `ALTER TABLE` est utilisée pour modifier la structure d'une table existante dans une base de données.

Elle permet d'ajouter, de supprimer ou de modifier des colonnes ainsi que de changer les contraintes sur les colonnes.

Syntaxe Générale :

```
ALTER TABLE TableName
  ADD ColumnName DataType [constraints];
  DROP COLUMN ColumnName;
  RENAME COLUMN OldColumnName TO NewColumnName;
```

Commande SQL: ALTER TABLE

Exemple 1

Modifier la table `Etudiant` en ajoutant une colonne `Adresse` et en supprimant la colonne `Age`:

De

`Etudiant (ID, Nom, Age, Email)`

à

`Etudiant (ID, Nom, Email, Adresse)`

Requêtes SQL :

```
ALTER TABLE Etudiant
  ADD Adresse VARCHAR(255);
ALTER TABLE Etudiant
  DROP COLUMN Age;
```

ID	Nom	Email	Adresse

Table 2: Table `Etudiant` modifiée

Commande SQL: ALTER TABLE

Exemple 2

Modifier la table Cours pour ajouter une colonne HeureDebut de type TIME, supprimer la colonne Description, et renommer la colonne Titre en NomCours :

Cours (ID, Titre, #DepartementID, DateDebut)

Requêtes SQL :

```
ALTER TABLE Cours
  ADD HeureDebut TIME;

ALTER TABLE Cours
  DROP COLUMN Description;

ALTER TABLE Cours
  RENAME COLUMN Titre TO NomCours;
```

Exercice

Mise en pratique des commandes SQL: CREATE, ALTER, DROP

CREATE : Créez les tables Livres et Etudiants avec les colonnes et contraintes appropriées :

`Livres (ID, Titre, Auteur, Code_Livre, Prix)`

`Etudiants (ID, Nom, Prenom, Adresse, Email)`

`Emprunts (ID, #ID_Livre, #ID_Etudiant, Date_Emprunt, Date_Retour)`

ALTER : Modifiez les tables créées :

Ajoutez une colonne `DateNaissance` à la table `Etudiants`.

Supprimez la colonne `Prix` de la table `Livres`.

Renommez la colonne `Code_Livre` en `ISBN` dans la table `Livres`.

DROP : Supprimez la table `Emprunts`.

Solution

Réponses aux commandes SQL

```
CREATE TABLE Livres (
ID INT PRIMARY KEY,
Titre VARCHAR(255) NOT NULL,
Auteur VARCHAR(255) NOT NULL,
Code_Livre INT UNIQUE,
Prix REAL
);
```

```
CREATE TABLE Etudiants (
ID INT PRIMARY KEY,
Nom VARCHAR(255) NOT NULL,
Prenom VARCHAR(255) NOT NULL,
Adresse TEXT,
Email VARCHAR(255) UNIQUE
);
```

- Ajouter une colonne `DateNaissance` à la table `Etudiants` :
  `ALTER TABLE Etudiants ADD DateNaissance DATE;`
- Supprimer la colonne `Prix` de la table `Livres` :
  `ALTER TABLE Livres DROP COLUMN Prix;`
- Renommer la colonne `Code_Livre` en `ISBN` dans la table `Livres` :
  `ALTER TABLE Livres RENAME COLUMN Code_Livre TO ISBN;`
- Supprimer la table `Emprunts` :
  `DROP TABLE Emprunteurs;`

Commande SQL: INSERT

Définition et Syntaxe

Définition :

La commande `INSERT INTO` est utilisée pour insérer des enregistrements dans une table.

Elle ajoute une ou plusieurs lignes de données à une table existante.

Syntaxe Générale :

```
INSERT INTO TableName (Column1, Column2, ...)
VALUES (Value1, Value2, ...);
```

Exemple :

```
INSERT INTO Etudiant (ID, Nom, Age, Email)
VALUES (1, 'Ali', 20, 'ali@ipei.tn');
```

Commande SQL: INSERT

Exemples

Table : Etudiant (ID, Nom, Age, Email)
Requête SQL :

```
INSERT INTO Etudiant (ID, Nom, Age, Email)
VALUES (1, 'Ali', 20, 'ali@ipei.tn');
```

Table :
Cours (ID, Titre, Description, DepartementID, DateDebut)
Requête SQL :

```
INSERT INTO Cours
VALUES (101, 'Introduction à SQL', 'Cours de base sur
SQL', 1, '2024-09-01');
```

Table : Livres (ID, Titre, Auteur, ISBN)
Requête SQL :

```
INSERT INTO Livres (ID, Titre, ISBN)
VALUES (1, 'Python 3', '978-0451524935');
```

Commande SQL: UPDATE

Définition et Syntaxe

Définition :

La commande `UPDATE` est utilisée pour modifier les enregistrements existants dans une table.

Elle permet de mettre à jour une ou plusieurs colonnes pour une ou plusieurs lignes.

Syntaxe Générale :

```
UPDATE TableName
SET Column1 = Value1, Column2 = Value2, ...
WHERE condition;
```

Exemple :

```
UPDATE Etudiant
SET Email = 'nouveau_email@ipei.tn'
WHERE ID = 1;
```

Commande SQL: UPDATE

Exemples

Exemple 1 :

Table : Cours (ID, Titre, Description, DepartementID, DateDebut)

Mettre à jour la description du cours avec l'ID 101 pour qu'elle soit plus détaillée :

```
UPDATE Cours
SET Description = 'Introduction complète à SQL'
WHERE ID = 101;
```

Exemple 2 :

Table : Livres (ID, Titre, Auteur, AnneePublication, ISBN)

Mettre à jour l'année de publication et l'auteur du livre avec l'ID 1 :

```
UPDATE Livres
SET AnneePublication = 2023, Auteur = 'Ali'
WHERE ID = 1;
```

Commande SQL: DELETE

Définition et Syntaxe

Définition :

La commande `DELETE` est utilisée pour supprimer des enregistrements d'une table.

Elle permet de supprimer une ou plusieurs lignes de données en fonction d'une condition.

Syntaxe Générale :

```
DELETE FROM TableName
WHERE condition;
```

Exemple :

```
DELETE FROM Etudiant
WHERE ID = 1;
```

Commande SQL: DELETE

Exemples

Exemple 1 :

Cours (ID, Titre, Description, DepartementID, DateDebut)

Supprimer le cours avec l'ID 101 :

```
DELETE FROM Cours
WHERE ID = 101;
```

Exemple 2 :

Livres (ID, Titre, Auteur, AnneePublication, ISBN)

Supprimer les livres publiés avant 2000 et écrits par 'Ali' :

```
DELETE FROM Livres
WHERE AnneePublication < 2000 AND Auteur = 'Ali';
```

Commande SQL: SELECT

Définition et Syntaxe

Définition :

La commande SELECT est utilisée pour interroger une base de données et extraire des données des tables.

Elle permet de spécifier les colonnes à afficher et de filtrer les résultats avec des conditions.

Syntaxe Générale :

```
SELECT Column1, Column2, ...
FROM TableName
WHERE condition;
```

Exemple :

```
SELECT Nom, Email
FROM Etudiant
WHERE Age > 18;
```

Commande SQL: SELECT

Opérateur Projection $\pi \rightarrow$ SELECT

L'opérateur Projection π en algèbre relationnelle, qui permet de sélectionner certaines colonnes d'une table, est traduit en SQL par la commande :

```
SELECT Column1, Column2, ...
FROM TableName
```

Exemple :

Table : Etudiant (ID, Nom, Age, Email)

Projection en AR :

$\pi_{Nom,Email}(Etudiant)$

En SQL :

```
SELECT Nom, Email
FROM Etudiant
```

Commande SQL: SELECT

Clause DISTINCT

La clause `DISTINCT` est utilisée en SQL pour éliminer les doublons dans les résultats d'une requête. Elle s'applique à la sélection des colonnes spécifiées dans la commande `SELECT`.

Syntaxe :

```
SELECT DISTINCT Column1, Column2, ...
FROM TableName
```

Exemple :

`Table : Etudiant (ID, Nom, Age, Email)`

Pour afficher des noms distincts des étudiants :

```
SELECT DISTINCT Nom
FROM Etudiant;
```

Note :

La clause `DISTINCT` est utile lorsque vous souhaitez obtenir des valeurs uniques dans le résultat.

Commande SQL: SELECT

Opérateur Renommage $\alpha \rightarrow$ AS

L'opérateur Renommage α en algèbre relationnelle permet de renommer les colonnes ou les tables dans une requête. En SQL, cet opérateur est traduit par la clause AS.

Syntaxe en Algèbre Relationnelle :

$$R1 = \alpha_{AncienNom \leftarrow NouveauNom}(Relation)$$

Syntaxe SQL :

```
SELECT ColumnName AS AliasName
FROM TableName AS AliasName
```

Exemple :

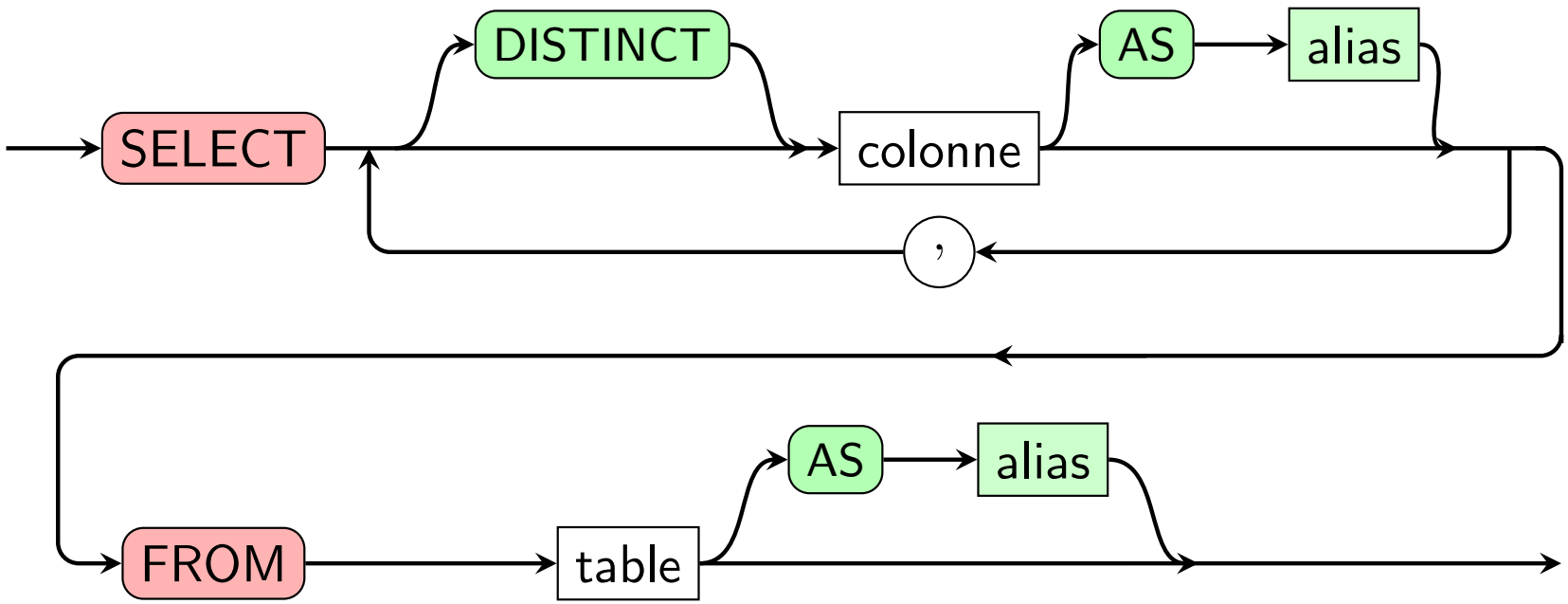
Table : Etudiant (ID, Nom, Age, Email)

Renommer la colonne Nom en NomEtudiant et la table Etudiant en E :

```
SELECT Nom AS NomEtudiant
FROM Etudiant AS E;
```

Commande SQL: SELECT

`SELECT ... FROM`



Commande SQL: SELECT

Opérateur Sélection σ

L'opérateur Sélection σ en algèbre relationnelle permet de filtrer les lignes d'une table en fonction de conditions spécifiques.

Correspondance SQL :

```
WHERE condition
```

Exemple :

Table : Etudiant (ID, Nom, Age, Email)

Sélection en AR:

$\sigma_{Age>18}(Etudiant)$

Sélection en SQL :

```
SELECT *
FROM Etudiant
WHERE Age > 18;
```

Commande SQL: SELECT

Exemple : WHERE & LIKE

Sélectionner les titres des cours qui ont débuté après le 1er janvier 2023 et dont le titre contient le mot "Python".

Table :
Cours (ID, Titre, Description, DepartementID, DateDebut)

Requête SQL :

```
SELECT C.Titre
FROM Cours C
WHERE C.DateDebut > '2023-01-01'
AND C.Titre LIKE '%Python%';
```

Note sur l'opérateur LIKE :

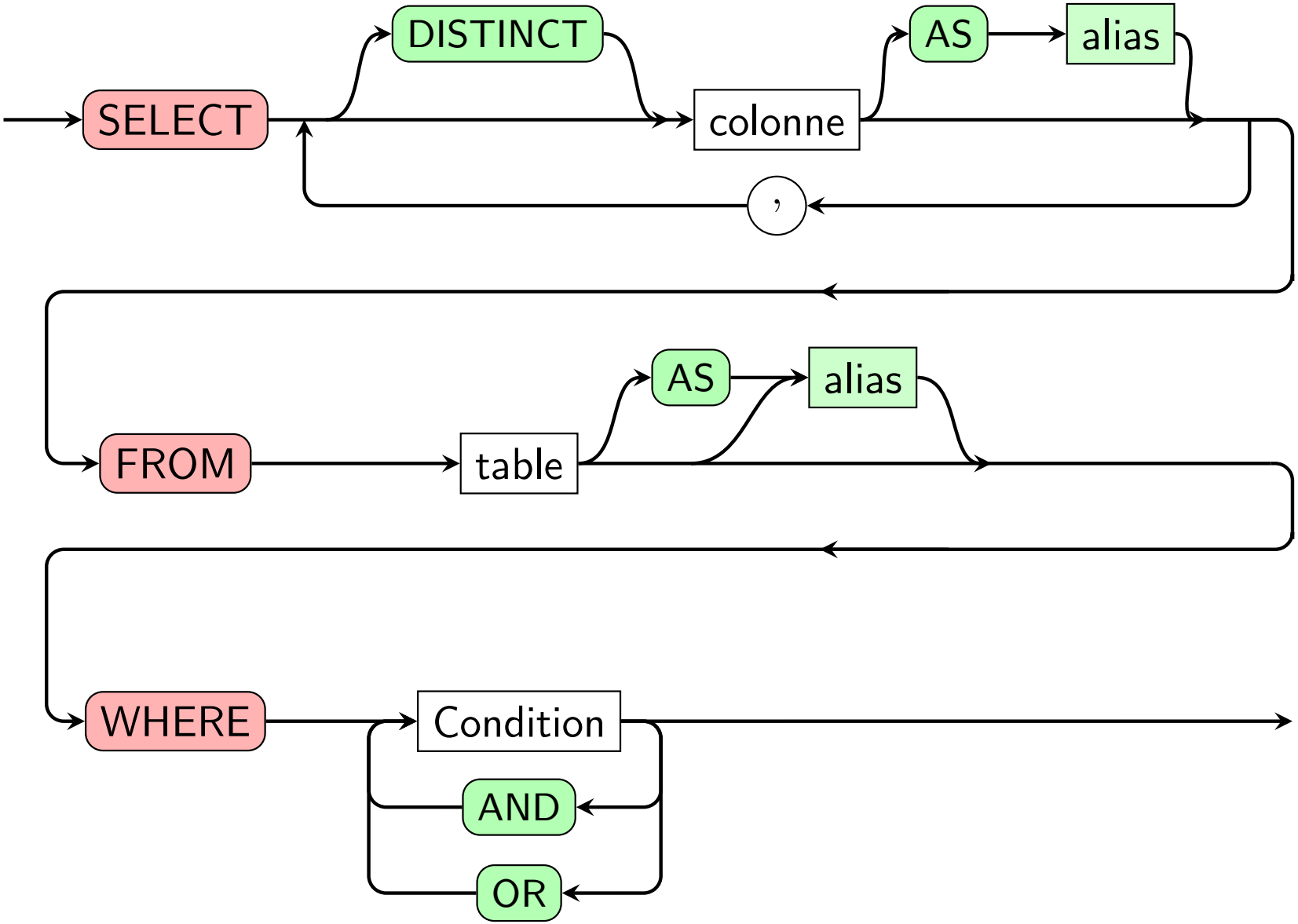
L'opérateur LIKE est utilisé pour rechercher une correspondance dans une colonne de type texte.

Les caractères spéciaux utilisés avec LIKE sont :

- % : Représente zéro ou plusieurs caractères.
- _ : Représente un seul caractère.

Commande SQL: SELECT

`SELECT ... FROM ... WHERE`



Commande SQL: SELECT

Opérateur Jointure $\bowtie$

L'opérateur Jointure $\bowtie$ en algèbre relationnelle combine les tuples de deux relations en fonction d'une condition de jointure.

Correspondance SQL :

```
Table1 JOIN Table2 ON Table1.Column = Table2.Column
```

Exemple :

```
Table :  Cours (ID, Titre, #DepartementID)
Table :  Departement (ID, Nom)
```

Jointure en AR:

$$Cours \underset{\text{Cours.DepartementID} = \text{Departement.ID}}{\bowtie} Departement$$

En SQL :

```sql
SELECT *
FROM Cours C JOIN Departement D ON C.DepartementID = D.ID;
```

Exemple de Jointure

Utilisation de `JOIN`

`Employes` (id, nom, #departementID)
`Departement` (id, nom)

Question : Afficher le département de chaque employé.

Requête SQL :

```sql
SELECT E.Nom AS Employe, D.Nom AS Departement
FROM Employes E
JOIN Departements D ON E.DepartementID = D.ID;
```

Table Employes

ID	Nom	DepartementID
1	Ali Ben Salem	101
2	Mohamed Trabelsi	102
3	Amel Saidi	103
4	Amina Bouaziz	101

Table `Departements`

ID	Nom
101	Informatique
102	Ressources Humaines
103	Comptabilité

Jointure: Combiner une ligne de Employes à une ligne de Departements

ID	Nom	DepartementID	ID	Nom
1	Ali Ben Salem	101	101	Informatique
2	Mohamed Trabelsi	102	102	Ressources Humaines
3	Amel Saidi	103	103	Comptabilité
4	Amina Bouaziz	101	101	Informatique

Employe	Departement
Ali Ben Salem	Informatique
Mohamed Trabelsi	Ressources Humaines
Amel Saidi	Comptabilité
Amina Bouaziz	Informatique

Exemple de Jointure avec Trois Tables

Utilisation de `JOIN` multiples

`Employes` (id, nom, #departementID)
`Departements` (id, nom)
`Projets` (id, nom, #employeID)

Question : Afficher le département et le projet de chaque employé.

```sql
SELECT E.nom Employe, D.nom Departement, P.nom Projet
FROM Employes E
JOIN Departements D ON E.DepartementID = D.ID
JOIN Projets P ON E.ID = P.EmployeID;
```

Table Employes

ID	Nom	DepartementID
1	Ali Ben Salem	101
2	Mohamed Trabelsi	102
3	Amel Saidi	103
4	Amina Bouaziz	101

Table `Departements`

ID	Nom
101	Informatique
102	Ressources Humaines
103	Comptabilité

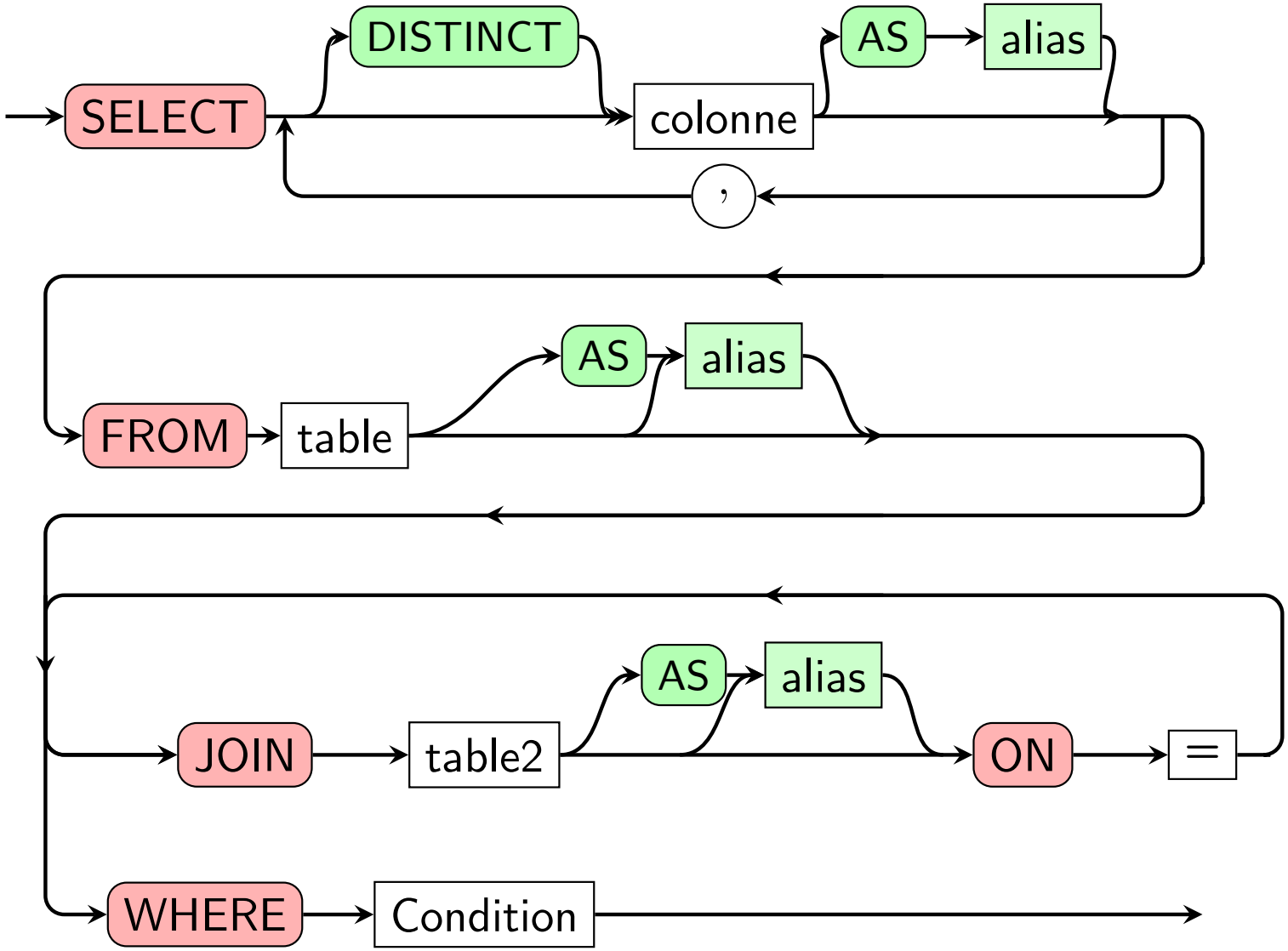
Table `Projets`

ID	Nom	EmployeID
1	Projet A	1
2	Projet B	2
3	Projet C	3
4	Projet D	1

E.ID	E.nom	E.DepartementID	D.id	D.nom	P.ID	P.nom	P.EmployeID
1	Ali Ben Salem	101	101	Informatique	1	Projet A	1
1	Ali Ben Salem	101	101	Informatique	4	Projet D	1
2	Mohamed Trabelsi	102	102	Ressources Humaines	2	Projet B	2
3	Amel Saidi	103	103	Comptabilité	3	Projet C	3

Commande SQL: SELECT

`SELECT ... FROM ... JOIN ... ON`



Commande SQL: SELECT

La clause GROUP BY

La clause GROUP BY en SQL permet de regrouper les lignes d'une table en fonction des valeurs d'une ou plusieurs colonnes, souvent utilisé en combinaison avec des **fonctions d'agrégation** comme COUNT, SUM, AVG, etc.

Syntaxe SQL :

```
GROUP BY colonne1, colonne2, ...
```

Exemple :

Table : Cours (ID, Titre, DepartementID, DateDebut)

Requête SQL : Nombre de cours par département

```
SELECT DepartementID, COUNT(*) AS NombreDeCours
FROM Cours
GROUP BY DepartementID;
```

Exemple de regroupement avec `GROUP BY`

La clause `GROUP BY`: Exemple

Table `Cours (ID, Titre, DepartementID, DateDebut)` :

ID	Titre	DepartementID	DateDebut
1	Mathématiques	1	2024-01-10
3	Informatique	1	2024-01-15
5	Statistiques	1	2024-01-22
2	Physique	2	2024-01-12
7	Économie	2	2024-01-30
4	Chimie	3	2024-01-20
6	Biologie	3	2024-01-25

Requête SQL: Nombre de cours par département.

```
SELECT DepartementID, COUNT(*) AS NombreDeCours
FROM Cours
GROUP BY DepartementID;
```

Résultat :

DepartementID	NombreDeCours
1	3
2	2
3	2

Fonctions d'agrégation en SQL

Principales fonctions d'agrégation

Les fonctions d'agrégation en SQL permettent de réaliser des calculs sur un ensemble de valeurs et de retourner un résultat unique. Elles sont couramment utilisées en combinaison avec la clause `GROUP BY` pour regrouper et résumer des données.

Principales fonctions d'agrégation :

- `COUNT()`: Compte le nombre de lignes dans un groupe.
- `SUM()`: Calcule la somme des valeurs numériques d'une colonne.
- `AVG()`: Calcule la moyenne des valeurs numériques d'une colonne.
- `MIN()`: Retourne la plus petite valeur dans un groupe.
- `MAX()`: Retourne la plus grande valeur dans un groupe.

Exemple d'utilisation :

- Trouver le nombre total d'étudiants par département.
- Calculer la moyenne des salaires par service.
- Déterminer le chiffre d'affaires total par produit.

Commande SQL: SELECT

Exemple : `GROUP BY` & `HAVING`

La clause `HAVING` est utilisée pour filtrer les groupes après l'application de `GROUP BY`, souvent avec des fonctions d'agrégation.

Table: `Cours (ID, Titre, #DepartementID, DateDebut)`
Requête SQL : Les départements ayant plus que 2 cours.

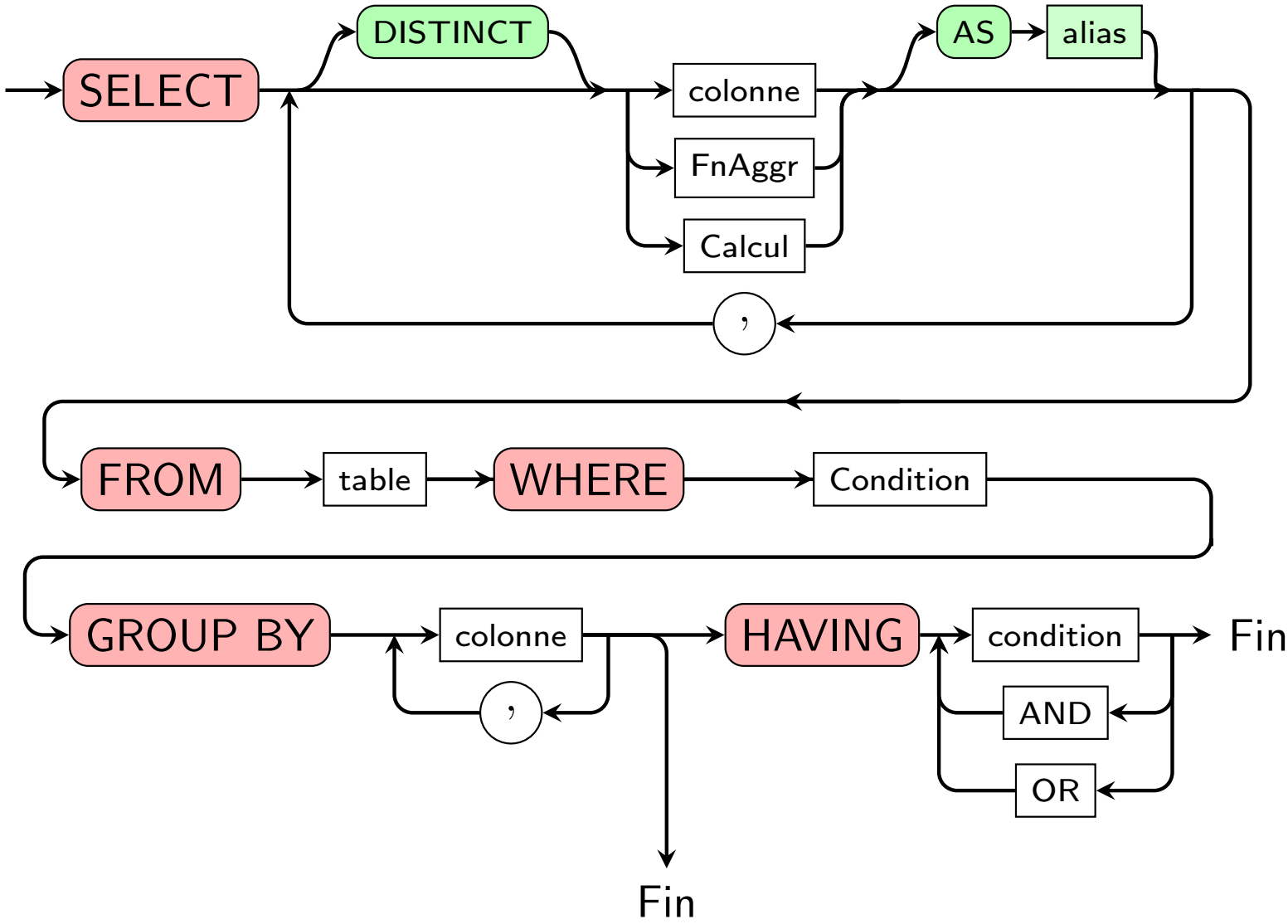
```
SELECT DepartementID, COUNT(*) AS NombreDeCours
FROM Cours
GROUP BY DepartementID HAVING NombreDeCours > 2;
```

ID	Titre	DepartementID	DateDebut
1	Mathématiques	1	2024-01-10
2	Informatique	1	2024-01-15
3	Statistiques	1	2024-01-22
4	Physique	2	2024-01-12
5	Économie	2	2024-01-30
6	Chimie	3	2024-01-20
7	Biologie	3	2024-01-25

DepartementID	NombreDeCours
1	3

Commande SQL: SELECT

`SELECT ... FROM ... GROUP BY ... HAVING`



Exercice d'application

Utilisation de `GROUP BY` et `HAVING`

Exercice : Vous avez une table **Ventes** qui contient les informations suivantes :

ID	ProduitID	Quantité	DateVente
1	101	10	2024-01-05
2	102	15	2024-01-07
3	101	20	2024-01-10
4	103	25	2024-01-12
5	101	5	2024-01-15
6	102	10	2024-01-18
7	103	30	2024-01-20
8	101	15	2024-01-25

1. Écrivez une requête SQL pour calculer la quantité totale vendue par produit.
   a. Utilisez la clause `GROUP BY` pour regrouper les ventes par `ProduitID`.
   b. Affichez les résultats avec les colonnes `ProduitID` et `QuantitéTotale`.
2. Ajoutez une clause `HAVING` pour ne garder que les produits pour lesquels la quantité totale vendue est supérieure à 30.

Solution de l'exercice

Utilisation de `GROUP BY` et `HAVING`

Etape 1 : Regroupe les ventes par identifiant de produit.

ID	ProduitID	Quantité	DateVente
1	101	10	2024-01-05
2	102	15	2024-01-07
3	101	20	2024-01-10
4	103	25	2024-01-12
5	101	5	2024-01-15
6	102	10	2024-01-18
7	103	30	2024-01-20
8	101	15	2024-01-25

ID	ProduitID	Quantité	DateVente
1	101	10	2024-01-05
3	101	20	2024-01-10
5	101	5	2024-01-15
8	101	15	2024-01-25
2	102	15	2024-01-07
6	102	10	2024-01-18
4	103	25	2024-01-12
7	103	30	2024-01-20

Etape 2: Calculer la quantité totale par groupe.

ProduitID	QuantitéTotale
101	50
102	25
103	55

Etape 3: Filtre les groupes pour ne garder que ceux dont la quantité totale dépasse 30.

Résultat final :

ProduitID	QuantitéTotale
101	50
103	55

Requête SQL :

```
SELECT ProduitID, SUM(Quantité) AS QuantitéTotale
FROM Ventes GROUP BY ProduitID HAVING QuantitéTotale > 30;
```

Tri et Pagination

Clauses SQL: `ORDER BY`, `LIMIT`, `OFFSET`, `ASC`, et `DESC`

Les clause de tri et de pagination

`ORDER BY` : Trie les résultats d'une requête SQL par une ou plusieurs colonnes.

`LIMIT` : Limite le nombre de résultats retournés par la requête.

`OFFSET` : Sauter un certain nombre de résultats avant de commencer à renvoyer les résultats.

`ASC` : Ordre croissant (par défaut).

`DESC` : Ordre décroissant.

Syntaxe :

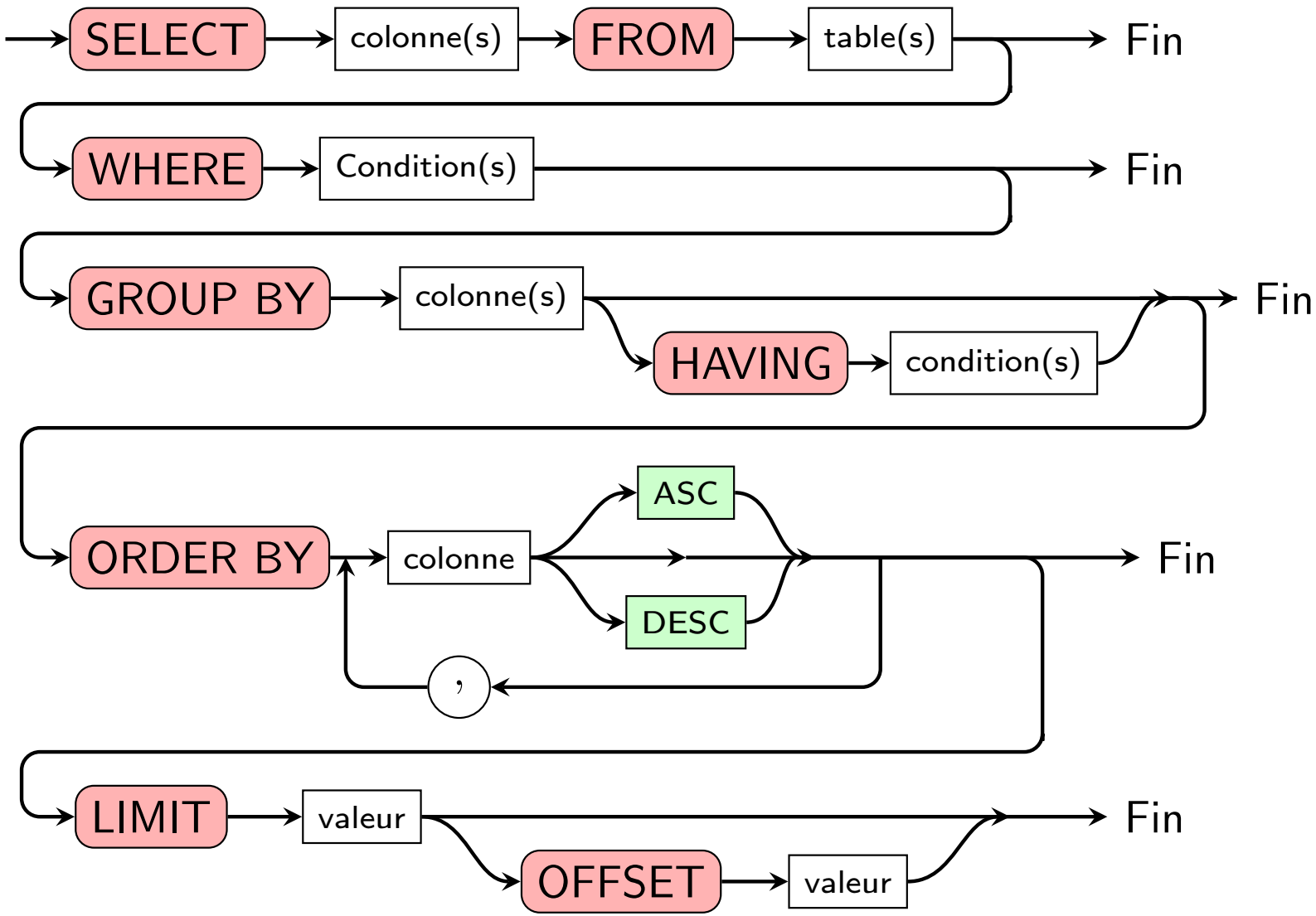
```
SELECT colonne1, colonne2, ...
FROM table
ORDER BY colonne1 [ASC|DESC], colonne2 [ASC|DESC], ...
LIMIT nombre_lignes
OFFSET nombre_lignes;
```

Exemple: Le deuxième produit le plus cher.

```sql
SELECT * FROM Produits
ORDER BY prix DESC
LIMIT 1 OFFSET 1;
```

Commande SQL: SELECT

`SELECT ... FROM ... GROUP BY ... HAVING`



Commande SQL: SELECT

Opérateur Union ∪

L'opérateur Union ∪ en algèbre relationnelle combine les résultats de deux requêtes distinctes en une seule. Par défaut, il élimine les doublons. Pour conserver les doublons, on utilise `UNION ALL`.

Correspondance SQL :

```
Requête1 UNION Requête2
Requête1 UNION ALL Requête2
```

Exemple :

Table : Cours (ID, Titre, #DepartementID)
Table : AnciensCours (ID, Titre, #DepartementID)

Union en AR:

$$Cours \cup AnciensCours$$

En SQL :

```sql
SELECT Titre FROM Cours
UNION
SELECT Titre FROM AnciensCours;
```

Exemple d'Union

Utilisation de `UNION` et `UNION ALL`

`Table Cours` (id, titre, #departementID)
`Table AnciensCours` (id, titre, #departementID)

Question : Afficher les titres de cours actuels et anciens.

Requête SQL :

```sql
SELECT Titre FROM Cours
UNION ALL
SELECT Titre FROM AnciensCours;
```

Table Cours

ID	Titre
1	Mathématiques
2	Informatique
3	Physique

Table AnciensCours

ID	Titre
4	Chimie
5	Biologie
6	Informatique

Résultat: Combinaison des titres des deux colonnes Titre

Titre
Mathématiques
Informatique
Physique
Chimie
Biologie
Informatique

Exercice : Utilisation de l'opérateur `UNION` avec projection

Exercice pratique

Énoncé :

Vous disposez de deux tables : `EtudiantsActuels` et `AnciensEtudiants`. Les tables ont les schémas suivants :

`EtudiantsActuels` (id, nom, prenom, #departementID, annee)

`AnciensEtudiants` (id, nom_complet, #departement)

Question :

Écrivez une requête SQL pour obtenir une liste des noms complets de tous les étudiants (actuels et anciens).

Corrigé :

```sql
SELECT CONCAT(nom, ' ', prenom) AS nom_complet
FROM EtudiantsActuels
UNION
SELECT nom_complet
FROM AnciensEtudiants;
```

Fonctions de manipulation de chaînes en SQL

Présentation des principales fonctions (Partie 1)

CONCAT

Combine plusieurs chaînes en une seule.

Syntaxe : CONCAT(chaîne1, chaîne2, ...)

Requête SQL :

```sql
SELECT CONCAT(Prenom, ' ', Nom) AS NomComplet
FROM etudiants;
```

Résultat :

NomComplet
Aymen zaied

SUBSTRING

Extrait une sous-chaîne d'une chaîne de caractères.

Syntaxe : SUBSTRING(chaîne, position, longueur)

Requête SQL :

```sql
SELECT SUBSTRING(Nom, 1, 3) AS NomPartiel
FROM etudiants;
```

Résultat :

NomPartiel
Aym

Fonctions de manipulation de chaînes en SQL

Présentation des principales fonctions (Partie 2)

LENGTH

Retourne la longueur d'une chaîne de caractères.

Syntaxe : LENGTH(chaîne)

Requête SQL :

```sql
SELECT LENGTH(Nom) AS LongueurNom
FROM etudiants;
```

Résultat :

LongueurNom
5

UPPER et LOWER

Convertit une chaîne en majuscules ou minuscules.

Syntaxe : UPPER(chaîne), LOWER(chaîne)

Requête SQL :

```sql
SELECT UPPER(Nom) AS NomMajuscules, LOWER(Prenom) AS
    PrenomMinuscules
FROM etudiants;
```

Résultat :

NomMajuscules	PrénomMinuscules
ZAIED	aymen

Commande SQL: SELECT

Opérateur Intersection ∩

L'opérateur `Intersection` ∩ en algèbre relationnelle retourne uniquement les enregistrements communs entre deux requêtes.

Correspondance SQL :

`Requête1 INTERSECT Requête2`

Exemple :

`Table : EtudiantsActuels (ID, Nom, #DepartementID)`
`Table : AnciensEtudiants (ID, Nom, #DepartementID)`

`Intersection en AR:`

$$EtudiantsActuels \cap AnciensEtudiants$$

`En SQL :`

```sql
SELECT Nom FROM EtudiantsActuels
INTERSECT
SELECT Nom FROM AnciensEtudiants;
```

Exemple d'Intersection

Utilisation de `INTERSECT`

`Table EtudiantsActuels` (id, nom, #departementID)
`Table AnciensEtudiants` (id, nom, #departementID)

Question : Trouver les étudiants inscrits actuellement et qui ont également été inscrits dans le passé.

Requête SQL :

```sql
SELECT Nom FROM EtudiantsActuels
INTERSECT
SELECT Nom FROM AnciensEtudiants;
```

Table EtudiantsActuels

ID	Nom
1	Ali
2	Bacem
3	Amira

Table AnciensEtudiants

ID	Nom
2	Ali
3	Aymen
4	Bacem

Résultat: Noms communs entre les deux colonnes Nom

Nom
Ali
Bacem

Exercice : Utilisation de l'opérateur INTERSECT

Exercice pratique

Énoncé :
Vous disposez de deux tables : `ProduitsEnStock` et `ProduitsCommandes`. Les tables ont les schémas suivants :

`ProduitsEnStock` (id, nom_produit, categorie)

`ProduitsCommandes` (id, nom_produit, categorie)

Question :
Écrivez une requête SQL pour obtenir une liste des produits qui sont à la fois en stock et ont été commandés.

Corrigé :

```sql
SELECT nom_produit
FROM ProduitsEnStock
INTERSECT
SELECT nom_produit
FROM ProduitsCommandes;
```

Commande SQL: SELECT

Opérateur Différence –

L'opérateur Différence – en algèbre relationnelle retourne les enregistrements présents dans la première requête mais pas dans la seconde.

Correspondance SQL :

```
ProduitsEnStock EXCEPT ProduitsCommandes
```

Exemple :

Table : ProduitsEnStock (ID, Nom_Produit, #Categorie)
Table : ProduitsCommandes (ID, Nom_Produit, #Categorie)

Différence en AR:

ProduitsEnStockExceptProduitsCommandes

En SQL :

```
1 SELECT Nom_Produit FROM ProduitsEnStock
2 EXCEPT
3 SELECT Nom_Produit FROM ProduitsCommandes;
```

Exemple de Différence

Utilisation de EXCEPT

Table ProduitsEnStock (id, nom_produit, #categorie)
Table ProduitsCommandes (id, nom_produit, #categorie)

Question : Trouver les produits qui sont en stock mais n'ont pas été commandés.

Requête SQL :

```sql
SELECT Nom_Produit FROM ProduitsEnStock
EXCEPT
SELECT Nom_Produit FROM ProduitsCommandes;
```

Table ProduitsEnStock

ID	Nom_Produit
1	Télévision
2	Radio
3	Smartphone

Table ProduitsCommandes

ID	Nom_Produit
2	Télévision
3	Radio
4	Ordinateur

Résultat: Noms des produits en stock mais non commandés

Nom_Produit
Smartphone

Exercice : Utilisation de l'opérateur EXCEPT

Exercice pratique

Énoncé :
Vous disposez de deux tables : `Clients` et `Commandes`. Les tables ont les schémas suivants :

`Clients` (id, nom_client, ville)

`Commandes` (id, #id_client, produit)

Question :
Écrivez une requête SQL pour obtenir une liste des clients qui n'ont passé aucune commande.

Corrigé :

```sql
SELECT *
FROM Clients
WEHERE id IN (SELECT id
              FROM Clients
              EXCEPT
              SELECT id_client
              FROM Commandes);
```

Types de Requêtes Imbriquées

Sous-requêtes en SQL

Il existe plusieurs types de sous-requêtes en SQL, chacune ayant son propre usage :

- Sous-requête qui retourne une seule valeur.
- Sous-requête qui retourne plusieurs valeurs ou lignes.
- Sous-requête qui dépend de la requête principale pour chaque ligne.

Exemple :Trouver le nom de l'employé ayant le salaire le plus élevé.

```
1 SELECT nom
2 FROM   Employes
3 WHERE  salaire = (SELECT MAX(salaire) FROM Employes);
```

Exemple de Sous-Requêtes

Exemple 1

Trouver les employés dont le salaire est supérieur au salaire moyen de leur département.

```sql
SELECT nom, salaire
FROM Employes e1
WHERE salaire > (
                  SELECT AVG(salaire)
                  FROM Employes e2
                  WHERE e1.departement_id = e2.departement_id)
```

Explication : La sous-requête calcule le salaire moyen pour chaque département, et la requête principale sélectionne les employés dont le salaire est supérieur à cette moyenne.

Sous-Requêtes dans la clause FROM

Exemple 2

Les sous-requêtes peuvent également être utilisées dans la clause FROM pour créer une table temporaire qui sera utilisée dans la requête principale.

Exemple : Trouver les départements avec un salaire total supérieur à 100 000.

```sql
SELECT departement_id, SUM(salaire) AS total_salaire
FROM (  SELECT departement_id, salaire
        FROM Employes ) AS sous_table
GROUP BY departement_id
HAVING SUM(salaire) > 100000;
```

Explication : La sous-requête crée une table temporaire avec les départements et leurs salaires, puis la requête principale regroupe les résultats et applique la condition.

Exercice : Sous-Requête dans une clause `WHERE`

Exercice pratique

Énoncé :
Vous avez deux tables : `Produits` et `Ventes`. Les tables ont les schémas suivants :

`Produits` (id, nom_produit, prix)

`Ventes` (id, #produit_id, quantite)

Question :
Écrivez une requête SQL pour trouver les produits qui ont été vendus plus de 100 fois.

Corrigé :

```sql
SELECT nom_produit FROM Produits
WHERE id IN (
            SELECT produit_id
            FROM Ventes
            GROUP BY produit_id
            HAVING COUNT(*) > 100
            );
```