Bases de données

Master 1 Humanités numériques

Cécile Favre

https://eric.univ-lyon2.fr/~cfavre/

Requêtes

- Quésaco ?

Pourquoi des requêtes

- Il est intéressant de ne pouvoir retrouver que certaines informations dans la base de données.
- Lire tous les n-uplets peut être fastidieux.
- Les requêtes vont donc permettre de n'afficher que les données qui nous intéressent.

Requêtes de base

- Avec SQL

Présentation générale

- Les SGBD peuvent être interrogés via des commandes spécifiques.

- Il existe un langage commun, normalisé, utilisé par la grande majorité des SGBD.

- Ce langage s'appelle SQL pour *Structured Query Language*.

Présentation générale

- SQL : définition, manipulation et contrôle d'une base de données relationnelle (basé sur l'algèbre relationnelle)

- SQL est subdivisé en 3 sous langages :
  - LDD (Langage de Définition des Données) : création, modification et suppression des tables
  - LMD (Langage de Manipulation de Données) : ajout, suppression, modification et interrogation des données
  - LCD (Langage de Contrôle de Données) : gestion des protections d'accès

Présentation générale

- SQL : définition, manipulation et contrôle d'une base de données relationnelle (basé sur l'algèbre relationnelle)

- SQL subdivisé en 3 sous langages :
  - LDD (Langage de Définition des Données) : création, modification et suppression définitions des tables
  - **LMD (Langage de Manipulation de Données) : ajout, suppression, modification et interrogation des données**
  - LCD (Langage de Contrôle de Données) : gestion des protections d'accès

Les possibilités de SQL

- SQL permet de :
  - faire une requête sur les tables (requête avec projection, sélection, jointure)
  - créer/modifier/supprimer des tables ou des bases
  - ajouter/modifier/supprimer des enregistrements
  - gérer les droits

- Chaque instruction SQL se termine par « ; »

Trois types d'opérations

- Opérateurs relationnels de l'algèbre relationnelle
- Trois types :
  - Projection : on ne conserve que les attributs intéressants
  - Sélection : on ne conserve que les n-uplets intéressants
  - Jointure : on met ensemble des données situées dans différentes tables
- Possibilité de mélanger les opérations au sein d'une même requête

Projection / sélection

- Soit une table Etudiants(N°, nom, prénom, âge, année, établissement) qui contient 150 étudiant.es
- Projection : n'afficher que les noms et prénoms des étudiant.es
- Requête de sélection : n'afficher que les étudiants dont le nom commence par 'M'

N°	Nom	Prénom	Age	Année	Faculté
10	MERWEMT	Cécile	24	4	Sociologie
25	MUTARGO	Cédric	21	2	Histoire

Requête de sélection

Requête de projection

Requête de base

- SELECT champs FROM table;
- SELECT champs FROM table WHERE predicats;

- Permet de conserver les « champs » de la « table » répondant aux « prédicats »
- * dans la clause SELECT signifie « tous les champs »

- Ex :

```
SELECT Nom, Prénom FROM Etudiant
   WHERE Ville = 'Lyon';
SELECT * FROM Etudiant WHERE Ville = 'Lyon';
```

Les prédicats – 2

- AND/OR : pour composer différents prédicats

- SELECT * FROM Etudiant
  WHERE Ville = 'Lyon'
  AND Nom = 'PERSONNE';

  - Sélectionne les étudiant.es habitant Lyon et dont le nom est PERSONNE

Les prédicats – 3

- LIKE : comparaison de chaînes (« comme »)
- SELECT * FROM Etudiant
  WHERE Adresse LIKE '%rue%';
  - Sélectionne les étudiant.es qui habitent dans une rue
- Utilisation de deux caractères spéciaux :
  - % : n'importe quoi en nombre quelconque
  - _ : un caractère quelconque

Les prédicats – 4

- IN : présence dans une liste
- SELECT Nom, Prénom FROM Etudiant WHERE Ville IN ('Lyon', 'Bron', 'Brignais');
  - Sélectionne les étudiant.es qui habitent Lyon, Bron ou Brignais

Les prédicats – 5

- BETWEEN ... AND ... : pour donner une fourchette

- SELECT * FROM Etudiant
  WHERE Naissance BETWEEN '01/01/1980' AND '31/12/1980';

  - Sélectionne les informations des étudiant.es né.es en 1980

Les prédicats – 6

- NOT : permet de faire la négation d'un prédicat

- SELECT * FROM Etudiant
  WHERE Ville NOT IN ('Lyon', 'Brignais');

  - Sélectionne les étudiant.es, sauf les personnes habitant Lyon ou Brignais

Autre clause : ORDER BY

- ORDER BY (attribut) ASC/DESC : permet de trier les n-uplets
  - ASC : par ordre croissant ou alphabétique
  - DESC : par ordre décroissant
  -
- Par défaut (sans précision) : ordre ascendant

- SELECT * FROM Etudiant
  ORDER BY Nom DESC;
  - Donne toutes les infos des étudiant.es triées par nom de famille (inversement à l'ordre alphabétique)

Requêtes calculs et regroupement

- Avec SQL

Les calculs dans les requêtes

- Pour certains calculs, il est inutile d'enregistrer toutes les données calculées.
- On ne stocke donc que les données de base et les calculs seront refaits à chaque requête.
- Compromis entre :
  - Place sur le disque nécessaire pour la base
  - Capacités de calcul de l'ordinateur

Rappels

- Requête simple :

  SELECT attributs
  FROM tables
  WHERE predicats;

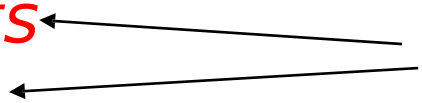
- Permet de faire à la fois des projections (SELECT) et des sélections (WHERE)

Requête SELECT complète

- En réalité la requête SELECT contient beaucoup plus de clauses

- Voilà une version plus complète :

SELECT *attributs*
FROM *tables*



Seules clauses obligatoires

WHERE *prédicats*
GROUP BY *critères de regroupements*
HAVING *critères sur le regroupement*
ORDER BY *attribut de tri* ASC/DESC;

Clause GROUP BY

- Permet de regrouper les attributs
- On peut mettre un ou plusieurs attributs

Calculs sur les regroupements

- Pour faire des sommes, moyennes… il faut l'indiquer dans la clause SELECT.

- Ex : pour faire la moyenne des notes par étudiant.e :

```
SELECT NumEtu, avg(Ch_Note)
FROM Passer
GROUP BY NumEtu;
```

Calculs sur les groupements 2

- Voici quelques fonctions pour les agrégats :
  - Sum() : somme
  - Avg() : moyenne
  - Min() : valeur minimale
  - Max() : valeur maximale
  - Count() : compte du nombre d’enregistrements

Renommage d'un champ

- Avec l'utilisation des groupements, il est souvent souhaitable d'avoir un nom plus court ou pour les calculs.

- Dans la clause SELECT, on peut renommer un champ avec AS :

```
SELECT NumEtu, avg(Ch_Note) AS Moy
FROM Passer
GROUP BY NumEtu;
```

Calculs dans les requêtes

- Il suffit d'indiquer le calcul dans la clause SELECT.
- Par exemple :

```
SELECT    NumEtu, CodeEpreuve,
          Ch_Note + 2 AS NoteAugment
FROM Passer;
```

- L'utilisation du renommage est alors fortement conseillé pour un résultat clair.

Clause HAVING

- Il s’agit de contraintes non pas sur les données d’origine mais sur les regroupements, portant sur des calculs d’agrégats (sum, avg…)

- Ne pas confondre WHERE et HAVING !

WHERE vs HAVING

- Voici deux requêtes différentes :

```
SELECT  IdClient,
        sum(montant)
FROM Commandes
WHERE montant > 100
GROUP BY IdClient;
```

Fait pour chaque client.e la somme de ses commandes dont le montant individuel dépasse 100 €.

≠

```
SELECT  IdClient,
        sum(montant)
FROM Commandes
GROUP BY IdClient
HAVING
        sum(montant) > 100
```

Fait pour chaque client.e la somme de ses commandes et n'affiche que les client.es qui ont commandé pour plus de 100 € au total.

Requêtes de jointure

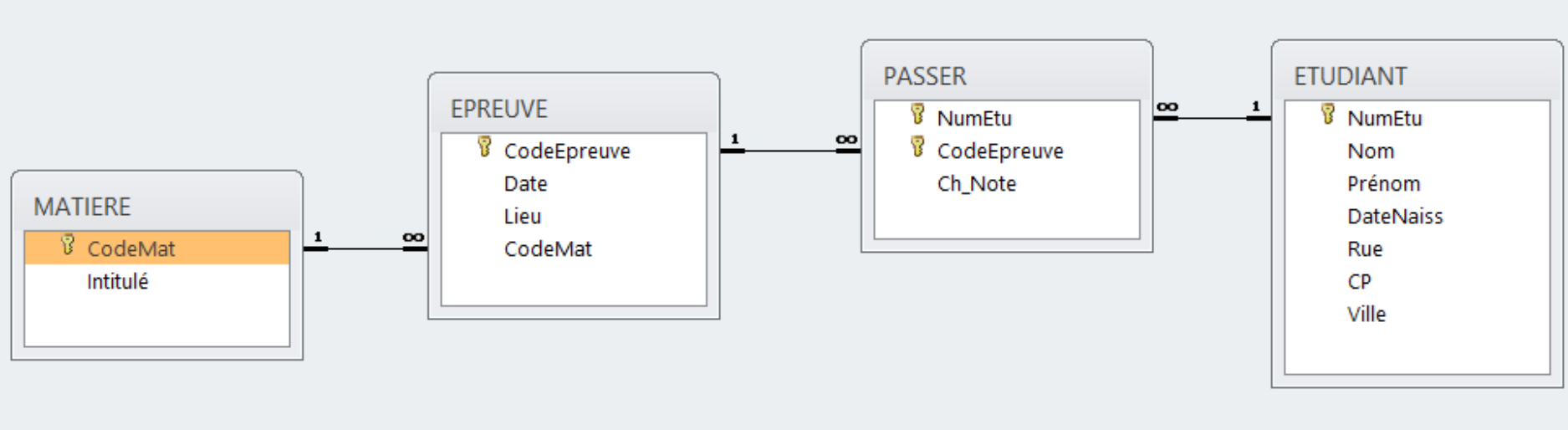
- Avec SQL

Requête sur une table

- Si on veut récupérer les informations d'une table, il suffit de faire une requête de sélection / projection.

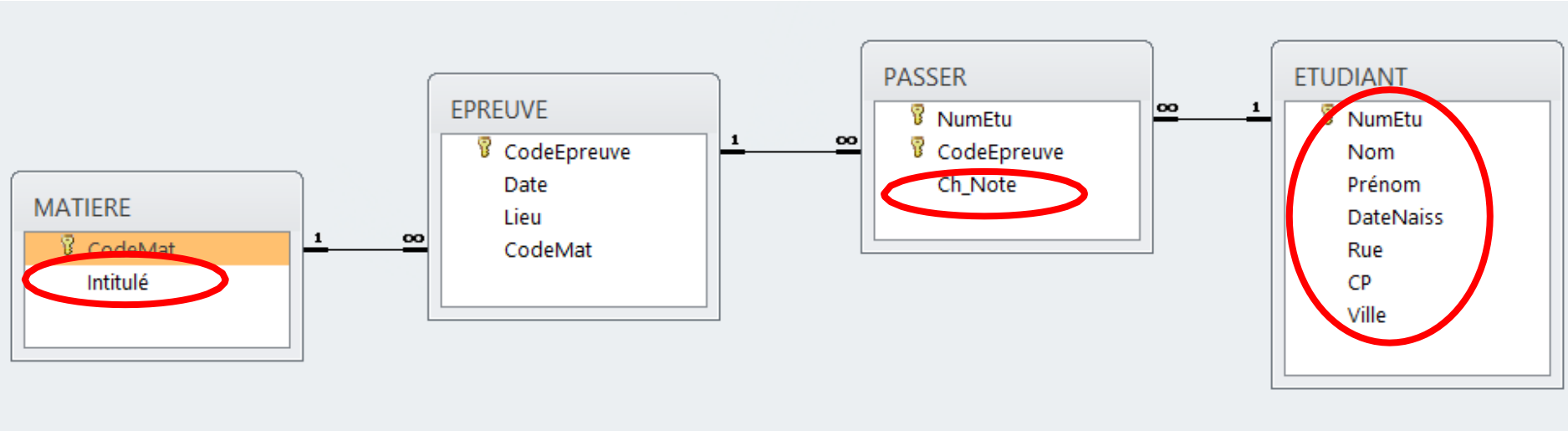
- Problème : et si mes données sont dans deux tables différentes (ou plus) ???

Exemple : base Etudiant



Exemple de questions

- Pour chaque étudiant.e, donner les informations sur les étudiant.es, les matières et les notes obtenues.



Repérage des tables

- Choix des tables qui vont intervenir dans la requête + les tables qui les lient (toutes les tables choisies doivent être liées entre elles) : pas +, pas -

- Choix des champs et spécification des critères s'il y en a, comme dans toute requête.

Jointure

- Une jointure est une combinaison des n-uplets de deux ou plusieurs tables.
- La combinaison se fait grâce aux liens clé primaire–clé étrangère.

Rappel

- Requête simple :

SELECT attributs
FROM tables
WHERE prédicats;

- Permet de faire à la fois des projections (SELECT) et des sélections (WHERE)

Jointures

- Il existe deux façons pour faire des jointures :
  - Utilisation du mot clé INNER JOIN
  - Utilisation de la clause WHERE

- Attention à ne pas confondre avec le produit cartésien.

Jointure – solution 1

- On utilise la clause WHERE pour indiquer l'attribut de jointure.
- Ex :
  - Etudiant(NumEtu, Nom, Prénom, DateNaiss, Rue, CP, Ville)
  - Passer(#NumEtu, #CodeEpreuve, Ch_Note)

```
SELECT NumEtu, Nom, Prénom, Ch_Note
FROM Etudiant, Passer
WHERE Etudiant.NumEtu = Passer.NumEtu;
```

Attention à la désambiguisation dans le SELECT, sinon erreur :

Etudiant.NumEtu plutôt que NumEtu par exemple.

Jointure – solution 2

- Cette solution utilise un mot-clé de SQL : INNER JOIN, qui se met dans FROM.

- La requête précédente donne alors :

```
SELECT NumEtu, Nom, Prénom, Ch_Note
FROM Etudiant INNER JOIN Passer
ON Etudiant.NumEtu = Passer.NumEtu;
```

Attention à la désambiguisation dans le SELECT, sinon erreur :

Etudiant.NumEtu plutôt que NumEtu par exemple.

Jointure avec 3 tables

```
SELECT NumEtu, Nom, Prénom, CodeEpreuve, Date, Ch_Note
FROM Etudiant, Passer, Epreuve
WHERE Etudiant.NumEtu=Passer.NumEtu
AND Passer.CodeEpreuve=Epreuve.CodeEpreuve AND Nom LIKE ‘M*’;

SELECT NumEtu, Nom, Prénom, CodeEpreuve, Date, Ch_Note
FROM (Etudiant INNER JOIN Passer
    ON Etudiant.NumEtu = Passer.NumEtu) INNER JOIN Epreuve
    ON Passer.CodeEpreuve=Epreuve.CodeEpreuve
WHERE Nom LIKE ‘M*’;
```

Attention à la désambiguisation dans le SELECT, sinon erreur :

Etudiant.NumEtu plutôt que NumEtu par exemple.

Jointure – comparaison

- La première solution est plus simple à écrire (surtout si + de 2 tables).

- Mais la deuxième a l'avantage d'être plus claire :
  - Dès la clause FROM on sait qu'on fait une jointure.
  - La clause WHERE ne sert que pour les prédicats de sélection et pas pour la jointure.

- Au final : à vous de choisir ce que vous préférez, mais vous devez savoir faire les deux.

Jointure procédurale

- Usage de requêtes imbriquées avec le prédicat IN (sous-requête)
  - Exemple 1 : Nom et prénom des étudiant.es ayant passé une épreuve

```
SELECT Nom, Prenom FROM etudiant WHERE NumEtu
IN (SELECT NumEtu FROM Passer);
```

  - Exemple 2 : Nom et prénom des étudiant.es ayant passé l'épreuve d'INFO1

```
SELECT Nom, Prenom FROM etudiant WHERE NumEtu
IN (SELECT NumEtu FROM Passer WHERE
CodeEpreuve='INFO1');
```

Auto-jointure

- Besoin selon

Table utilisateur :

id	prenom	nom	email	manager_id
1	Sebastien	Martin	s.martin@example.com	NULL
2	Gustave	Dubois	g.dubois@example.com	NULL
3	Georgette	Leroy	g.leroy@example.com	1
4	Gregory	Roux	g.roux@example.com	2

Source : https://sql.sh/cours/jointures/self-join

```
SELECT    u1.id, u1.prenom, u1.nom, u2.id as id_manager,
          u2.prenom as prenom_manager,
          u2.nom as nom_manager
FROM utilisateur as u1 LEFT JOIN utilisateur as u2
ON u1.manager=u2.id;
```

Requêtes autres prédicats

- Avec SQL

Suppression de doublons dans l'affichage de résultats de requête

- Usage de DISTINCT après le SELECT
- Exemple : de quelles villes sont les étudiant.es recensé.es ?

```
SELECT DISTINCT Ville
FROM etudiant;
```

Autres Prédicats

- Prédicats ALL / ANY / EXISTS / NOT EXISTS
  - Dans la clause WHERE
  - Précision d'une requête

- Opérateurs ensemblistes :

UNION, INTERSECT, MINUS

Prédicats de dénombrement ALL/ANY

- ALL/ANY : Teste si la valeur d'un attribut satisfait un critère de comparaison avec tous les résultats d'une sous- requête ou avec au moins un résultat d'une sous-requête.

- Numéros des étudiant·es qui ont obtenu au moins une note supérieure à chacune des notes obtenues par l'étudiant·e n° 1000.

```
SELECT DISTINCT NumEtu FROM Passer
WHERE Note > ALL
   (SELECT Note FROM Passer WHERE NumEtu = 1000);
```

- Numéros des étudiant·es qui ont obtenu au moins une note supérieure à au moins une des notes obtenues par l'étudiant·e n° 1000

```
SELECT DISTINCT NumEtu FROM Passer
WHERE Note >  ANY(    SELECT Note FROM Passer
                      WHERE NumEtu = 1000);
```

Prédicats d'existence EXIST / NOT EXIST

- EXISTS / NOT EXISTS

- Ex : Étudiant·es qui ont passé au moins une épreuve

```
SELECT * FROM Etudiant E
WHERE EXISTS(  SELECT * FROM Passer P
                    WHERE E.NumEtu = P.NumEtu);
```

- Ex : Étudiant·es qui n'ont passé aucune épreuve

```
SELECT * FROM Etudiant E
WHERE NOT  EXISTS(  SELECT * FROM Passer P
                         WHERE E.NumEtu =  P.NumEtu);
```

Opérations ensemblistes UNION / INTERSECT / MINUS

- **Union**
  - Élimination automatique des doublons
  - Ex : Code des épreuves ayant soit lieu dans l'Amphi Aubrac, soit ayant été passées par l'étudiant·e n° 1000

```
SELECT CodeEpr FROM Epreuve WHERE Lieu = 'Amphi  Aubrac'
UNION
SELECT CodeEpr FROM Passer WHERE NumEtu = 1000;
```

- **Intersection**
  - Ex : Code des épreuves ayant lieu dans l'Amphi Aubrac et ayant été passées par l'étudiant·e n° 1000

```
SELECT CodeEpr FROM Epreuve WHERE Lieu = 'Amphi Aubrac'
INTERSECT
SELECT CodeEpr FROM Passer WHERE NumEtu = 1000;
```

- **Différence**
  - Ex : Code des épreuves ayant lieu dans l'Amphi Aubrac mais sans celles ayant été passées par l'étudiant·e n° 1000

```
SELECT CodeEpr FROM Epreuve WHERE Lieu = 'Amphi Aubrac'
MINUS
SELECT CodeEpr FROM Passer WHERE NumEtu = 1000;
```

LMD

- Manipulation des données

LMD

- LMD (Langage de Manipulation de Données) :
  - Ajout (insertion)
  - Modification (mise à jour)
  - Suppression

  - Interrogation (clause SELECT : déjà abordée)

Insertion / mise à jour / suppression

- **Ajout d'un n-uplet**

INSERT INTO nom_table [(att1, att2,…)] VALUES (val_att1, val_att2,…);

Ex : INSERT INTO Matiere VALUES ('BD', 'Bases de données');

- **Mise à jour d'un attribut**

UPDATE nom_table SET attribut = valeur
[WHERE condition];

Ex : UPDATE Etudiant SET Nom='Tartampion' WHERE NumEtu = 333333;

UPDATE Passer SET Ch_Note = Ch_Note + 1;

- **Suppression de n-uplets**

DELETE FROM nom_table [WHERE condition];

Ex : DELETE FROM Etudiant WHERE Ville = 'Lyon';

LDD

- Définition des données

LDD

- LDD (Langage de Définition des Données) :
  - Création de tables
  - Modification de tables
  - Suppression de tables

  - (Mais aussi ces opérations sur d’autres structures : index ou autre)

Types de données

- Les types de données :

  https://mariadb.com/kb/en/data-types/

Contraintes intégrité

- Mot clé : CONSTRAINT

- **Clé primaire**

CONSTRAINT nom_c PRIMARY KEY (attribut_clé)

- **Clé étrangère**

CONSTRAINT nom_c FOREIGN KEY (attribut_clé_etr) REFERENCES table(attribut)

- **Contrainte de domaine**

CONSTRAINT nom_c CHECK (condition)

Création de table

- CREATE TABLE nom_table (Attribut1 TYPE,
  Attribut2 TYPE, …,
  contrainte_intégrité1,
  contrainte_intégrité2,
  …);

Création de tables : exemples

```
CREATE TABLE Etudiant(      NumEtu INT,
                            Nom VARCHAR(255),
                            Prenom  VARCHAR(255),
                            DateNaiss DATE,
                            Rue VARCHAR(255),
                            CP VARCHAR(5),
                            Ville  VARCHAR(255),
   CONSTRAINT EtuClePri PRIMARY KEY (NumEtu)
);
```

Création de tables : exemples

```
CREATE TABLE Passer (  NumEtu INT,
                       CodeEpr VARCHAR(10),
                       Ch_Note FLOAT,
   CONSTRAINT PassClePri PRIMARY KEY (NumEtu,CodeEpr),
   CONSTRAINT PassCleEtrEtu FOREIGN KEY (NumEtu)
                                REFERENCES Etudiant(NumEtu),
   CONSTRAINT PassCleEtrEpr FOREIGN KEY (CodeEpr)
                                REFERENCES Epreuve(CodeEpr),
   CONSTRAINT NoteValide CHECK (Ch_Note BETWEEN 0 AND 20)
);
```

Modification structurelle de table

- **Ajout d'attributs**

ALTER TABLE nom_table ADD (attribut TYPE, ...);

  - Ex : ALTER TABLE Etudiant ADD (tel VARCHAR(8));

- **Modifications d'attributs**

ALTER TABLE nom_table MODIFY (attribut TYPE, ...);

  - Ex : ALTER TABLE Etudiant MODIFY (tel VARCHAR(10));

- **Suppression d'attributs**

ALTER TABLE nom_table DROP COLUMN attribut;

  - Ex : ALTER TABLE Etudiant DROP COLUMN tel;

Modification structurelle de table

- **Ajout de contrainte**

ALTER TABLE nom_table ADD CONSTRAINT nom_contrainte définition_contrainte;

  - Ex : ALTER TABLE Epreuve
    ADD CONSTRAINT LieuValide
    CHECK (Lieu IN (‘Say’, ‘Aubrac’));

- **Suppression de contrainte**

ALTER TABLE nom_table DROP CONSTRAINT nom_contrainte;

  - Ex : ALTER TABLE Epreuve
    DROP CONSTRAINT LieuValide;

Copie et destruction de tables

- **Destruction**

DROP TABLE nom_table;

- **Copie**

CREATE TABLE copie AS requete;