# SQL (LMD)

## Un langage de manipulation de données

# Manipulation des données

- **INSERT INTO** : ajouter un tuple dans une table ou une vue
- **UPDATE** : changer les tuples d'une table ou d'une vue
- **DELETE FROM** : éliminer les tuples d'une table ou d'une vue

# INSERT INTO

- Syntaxe :

```
INSERT INTO
{nom_table ¦ nom_vue}
[ (nom_col (, nom_col)*) ]
{ VALUES (valeur (, valeur)*) ¦ sous-requête
  };
```

# Insertion sans préciser les colonnes

- Nous travaillons toujours sur la table Emprunteur composée de 4 colonnes : id, nom, prenom, annee_insc

```
INSERT INTO Emprunteur
VALUES (1, 'Buard', 'Jeremy', '2018');

INSERT INTO Emprunteur
VALUES (NULL, 'Zuckerberg', 'Mark', NULL);
```

–> Insert un tuple avec un id=2 et une année = NULL

```
CREATE TABLE Emprunteur(
id SMALLINT UNSIGNED AUTO_INCREMENT
    PRIMARY KEY,
  nom VARCHAR(20) NOT NULL,
  prenom VARCHAR(15) NOT NULL,
  annee_insc YEAR DEFAULT 2021,
)
ENGINE=INNODB;
```

# Insertion en précisant les colonnes

```
INSERT INTO Emprunteur (nom, prenom,
   annee_insc)
VALUES ('Chan', 'Priscilla', '2018');

INSERT INTO Emprunteur (nom, prenom)
VALUES ('Gates', 'Bill');
```

–> Insert un tuple avec une année = 2021

# Insertion multiple

```
INSERT INTO Emprunteur (nom, prenom,
   annee_insc)
VALUES ('Jobes', 'Steve', '2010'),
          ('Moskovitz', 'Dustin', '2011'),
          ('Musk', 'Elon', '2013');
```

# UPDATE

- Syntaxe :
  - UPDATE {nom_table ¦ nom_vue}
    SET { (nom_col)* = (sous-requête)
    ¦ nom_col = { valeur ¦ (sous-requête)} }*
    WHERE condition;
- Exemples :
  - UPDATE Emprunteur
    SET annee_insc = '2019'
    WHERE nom = 'Musk'

  - UPDATE Emprunteur
    SET annee_insc = annee_insc+2
    WHERE id < 3

# DELETE FROM

- Exemple :
  - DELETE FROM Emprunteur
    WHERE annee_insc < 2000

- Syntaxe :
  - DELETE FROM {nom_table ¦ nom_vue}
    WHERE condition;

# SQL

## Un langage de requêtes

# Structure générale d'une requête

- Structure d'une requête formée de trois clauses:

  **SELECT** <liste_attributs>

  **FROM** <liste_tables>

  **WHERE** <condition>

- **SELECT** définit le format du résultat cherché
- **FROM** définit à partir de quelles tables le résultat est calculé
- **WHERE** définit les prédicats de sélection du résultat

# Exemple de requête

SELECT * FROM Emprunteur

→Afficher tous les attributs de tous les tuples dans la table “Emprunteur”

# Opérateurs de comparaison

- = égal
  - WHERE id = 2
- <> différent
  - WHERE nom <> 'Ahmad'
- > plus grand que
  - WHERE annee_insc > 2010
- >= plus grand ou égal
  - WHERE annee_insc >= 2018
- < plus petit que
  - WHERE id < 3
- <= plus petit ou égal
  - WHERE id <= 2

# Opérateurs logiques

- **AND**
  - WHERE annee_insc < 2010 **AND** id<5
- **OR**
  - WHERE annee_insc < 2010 **OR** id<5

- Négation de la condition : **NOT**
  - SELECT *
    FROM Emprunteur
    WHERE nom = 'Badr'
    **AND NOT** annee_insc = '2019' ;

# Expressions logiques

Combinaisons:

WHERE
**(** ensoleillement > 80 **AND** pluviosité < 200 **)**
**OR** température > 30

WHERE
ensoleillement > 80
**AND (** pluviosité < 200 **OR** température > 30 **)**

# **Appartenance** à un ensemble : **IN**

WHERE monnaie = ‘Dollar’

**OR** monnaie = ‘Dirham’

**OR** monnaie = ‘Euro’

Équivalent à:

WHERE monnaie **IN** (‘Dollar’, ‘Dirham’, ‘Euro')

**NOT IN**: non appartenance à un ensemble

# **Comparaison** à un ensemble : **ALL**

```
SELECT * FROM Employe
WHERE salaire >= 1400
AND salaire >= 3000 ;
```

Équivalent à:

```
SELECT * FROM Employe
WHERE salaire >= ALL (1400, 3000);
```

# Valeur dans un intervalle : **BETWEEN**

WHERE population >= 50 **AND** population <= 60

Équivalent à:

WHERE population **BETWEEN** 50 **AND** 60

- **NOT BETWEEN**

# Conditions partielles (**joker**)

- % : un ou plusieurs caractères
  - WHERE nom LIKE '%med'
  - WHERE prenom LIKE '%rem%'
- _ : exactement un caractère
  - WHERE nom LIKE 'B_dr'
- **NOT LIKE**

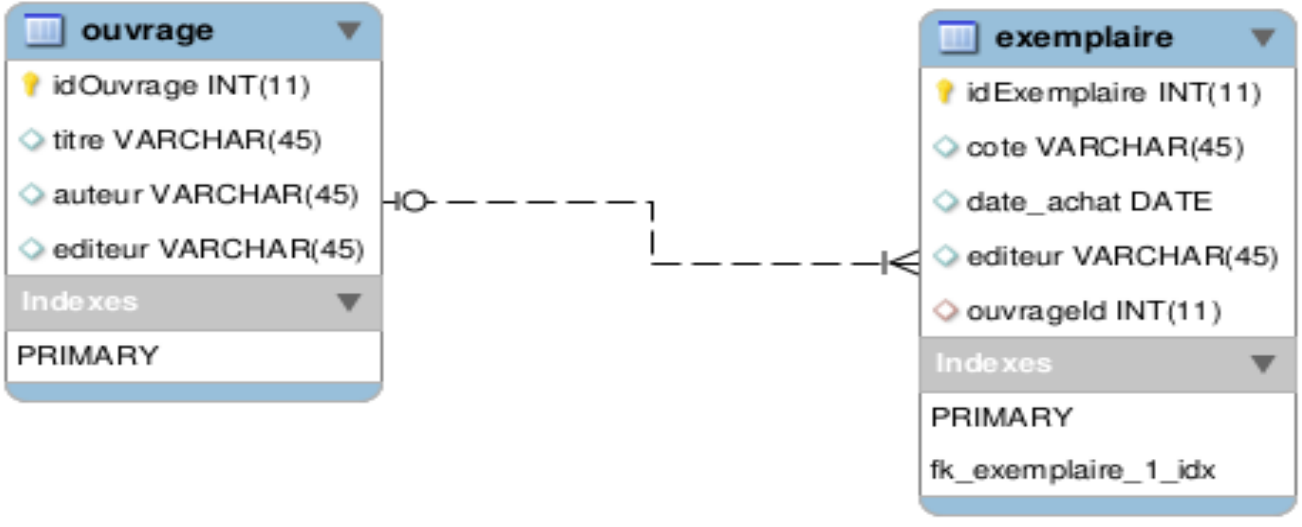
# Valeurs calculées

- SELECT nom, population, surface, natalité
  FROM Pays
  WHERE (population * 1000 / surface) < 50
    AND (population * natalité / surface) > 0

- SELECT nom, (population * 1000 / surface )
  FROM Pays

# Les jointures

- Principe :
  - Joindre plusieurs tables
  - On utilise les informations communes des tables



# Les jointures

- Prenons pour exemple un ouvrage de V. Hugo
- Si l'on souhaite des informations sur la cote d'un exemplaire il faudrait le faire en 2 temps:

  - 1) je récupère l'id de l'ouvrage :

  SELECT id FROM ouvrage where auteur LIKE 'V. Hugo'

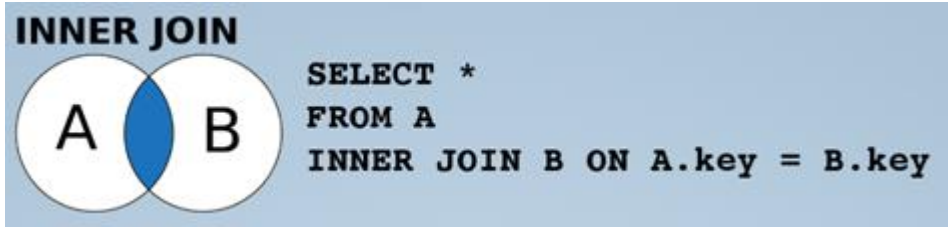
  - 2) Je récupère la ou les cote avec l'id récupéré

  SELECT cote FROM exemplaire WHERE ouvrageId = id_récupéré

# Les jointures

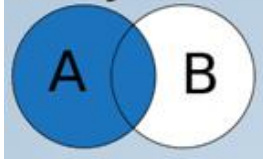
- On peux faire tout ça (et plus encore) en une seule requête
- C'est là que les jointures entrent en jeu:

```
SELECT exemplaire.cote
FROM exemplaire
INNER JOIN ouvrage
        ON exemplaire.ouvrageId = ouvrage.idOuvrage
WHERE ouvrage.auteur LIKE 'V. Hugo' ;
```



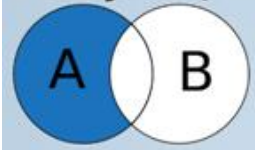
# Les jointures

**LEFT JOIN**



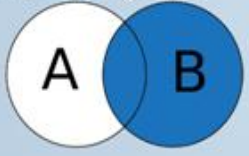

```
SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
```

**LEFT JOIN (sans l'intersection de B)**



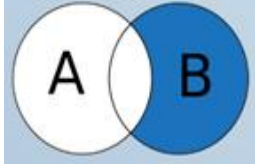

```
SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
WHERE B.key IS NULL
```

**RIGHT JOIN**



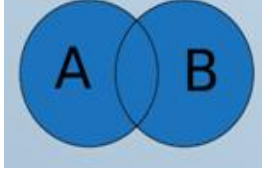

```
SELECT *
FROM A
RIGHT JOIN B ON A.key = B.key
```

**RIGHT JOIN (sans l'intersection de A)**



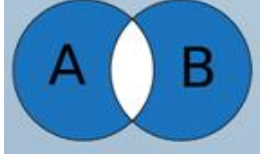

```
SELECT *
FROM A
RIGHT JOIN B ON A.key = B.key
WHERE A.key IS NULL
```

**FULL JOIN**




```
SELECT *
FROM A
FULL JOIN B ON A.key = B.key
```

**FULL JOIN (sans intersection)**




```
SELECT *
FROM A
FULL JOIN B ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```

# Les jointures

**Ville**

| id_client | ville |
|---|---|
| 1 | Marseille |
| 2 | Paris |
| 3 | Lyon |
| 7 | Montpellier |
| 8 | Levallois |

**Email**

| id_client | nom | prenom | email |
|---|---|---|---|
| 1 | Martin | Lucas | lucas.martin@outlook.fr |
| 2 | Clavier | Paul | NULL |
| 3 | Sauron | Benoit | NULL |
| 4 | Oron | Louis | louis.oron@yahoo.fr |
| 5 | Poiret | Michel | michel76@outlook.fr |

**INNER JOIN**

```
SELECT id_client, nom, prenom, ville
FROM ville
INNER JOIN email
ON ville.id_client = email.id_client
```

| id_client | nom | prenom | email | ville |
|---|---|---|---|---|
| 1 | Martin | Lucas | lucas.martin@outlook.fr | Marseille |
| 2 | Clavier | Paul | NULL | Paris |
| 3 | Sauron | Benoit | NULL | Lyon |

**LEFT JOIN**

```
SELECT ID_CLIENT, NOM, PRENOM, VILLE
FROM EMAIL
LEFT JOIN VILLE
ON EMAIL.ID_CLIENT = VILLE.ID_CLIENT
```

| id_client | nom | prenom | email | ville |
|---|---|---|---|---|
| 1 | Martin | Lucas | lucas.martin@outlook.fr | Marseille |
| 2 | Clavier | Paul | NULL | Paris |
| 3 | Sauron | Benoit | NULL | Lyon |
| 4 | Oron | Louis | louis.oron@yahoo.fr | NULL |
| 5 | Poiret | Michel | michel76@outlook.fr | NULL |

**RIGHT JOIN**

```
SELECT id_client, nom, prenom, ville
FROM email
RIGHT JOIN ville
ON email.id_client = ville.id_client
```

| id_client | nom | prenom | email | ville |
|---|---|---|---|---|
| 1 | Martin | Lucas | lucas.martin@outlook.fr | Marseille |
| 2 | Clavier | Paul | NULL | Paris |
| 3 | Sauron | Benoit | NULL | Lyon |
| 7 | NULL | NULL | NULL | Montpellier |
| 8 | NULL | NULL | NULL | Levallois |

# SQL

## Requêtes avec blocs emboîtés

# BD exemple

- Produit(np,nomp,couleur,poids,prix) *les produits*
- Usine(nu,nomu,ville,pays) *les usines*
- Fournisseur(nf,nomf,type,ville,pays) *les fournisseurs*
- Livraison(np,nu,nf,quantité) *les livraisons*
  - *np référence Produit.np*
  - *nu référence Usine.nu*
  - *nf référence Fournisseur.nf*

# Jointure par blocs emboîtés

*Requête: Nom et couleur des produits livrés par le fournisseur 1*

- Solution 1 : la jointure déclarative

```
SELECT nomp, couleur FROM Produit,Livraison
WHERE (Livraison.np = Produit.np) AND nf = 1 ;
```

- Solution 2 : la jointure procédurale (emboîtement)

***Nom et couleur des produits livrés par le fournisseur 1***

```
SELECT nomp, couleur FROM Produit
WHERE np IN
        (SELECT np FROM Livraison WHERE nf = 1) ;
```

*Numéros de produits livrés par le fournisseur 1*

# Jointure par blocs emboîtés

```
SELECT nomp, couleur FROM Produit
WHERE np IN
      ( SELECT np FROM Livraison
                  WHERE nf = 1) ;
```

- **IN** compare chaque valeur de np avec l'ensemble (ou multi-ensemble) de valeurs retournés par la sous-requête
- **IN** peut aussi comparer un tuple de valeurs:

```
SELECT nu FROM Usine
WHERE (ville, pays)
      IN (SELECT ville, pays FROM Fournisseur);
```

# Composition de conditions

*Requête: Nom des fournisseurs qui approvisionnent une usine de Londres ou de Paris en un produit rouge*

```
SELECT nomf
FROM Livraison, Produit, Fournisseur, Usine
WHERE
  couleur = 'rouge'
  AND Livraison.np = Produit.np
  AND Livraison.nf = Fournisseur.nf
  AND Livraison.nu = Usine.nu
  AND (Usine.ville = 'Londres' OR Usine.ville = 'Paris');
```

# Composition de conditions

*<u>Requête:</u>* ***Nom*** *des fournisseurs qui approvisionnent une usine de* ***Londres*** *ou de* ***Paris*** *en un produit* ***rouge***

```
SELECT nomf FROM Fournisseur
WHERE nf IN
(SELECT nf FROM Livraison
  WHERE np IN
        (SELECT np FROM Produit
              WHERE couleur = 'rouge')
  AND nu IN
        (SELECT nu FROM Usine
            WHERE ville = 'Londres' OR ville = 'Paris')
  ) ;
```

# Quantificateur ALL

*<u>Requête:</u> Numéros des fournisseurs qui ne fournissent que des produits rouges*

```
SELECT nf FROM Fournisseur
WHERE 'rouge' = ALL
      (SELECT couleur FROM Produit
       WHERE np IN
            (SELECT np FROM Livraison
             WHERE Livraison.nf = Fournisseur.nf ) ) ;
```

- *La requête imbriquée est ré-évaluée pour chaque tuple de la requête (ici pour chaque nf)*
- *ALL: tous les éléments de l'ensemble doivent vérifier la condition*

# Condition sur des ensembles : **EXISTS**

- Test si l'ensemble n'est pas vide (E ≠ ∅)

Exemple : *Noms des fournisseurs qui fournissent au moins un produit rouge*

```
SELECT nomf
FROM Fournisseur
WHERE EXISTS
    ( SELECT *
      FROM Livraison, Produit
      WHERE Livraison.nf = Fournisseur.nf
     AND Livraison.np = Produit.np
     AND Produit.couleur = 'rouge' );
```

*ce fournisseur*

*Le produit fourni est rouge*

# Blocs emboîtés - récapitulatif

```
SELECT …
FROM …
WHERE …
    attr IN requête
    attr NOT IN requête

    attr opérateur ALL requête

    EXISTS requête
    NOT EXISTS requête
```