Veille

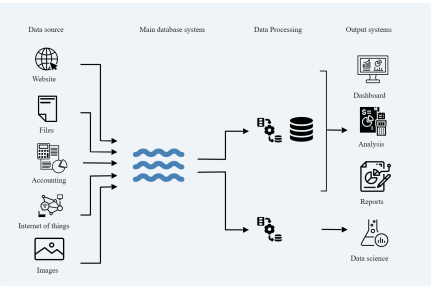
A. **Qu’est ce qu’une donnée ? Sous quelle forme peut-elle se présenter ?**
Une donnée est la représentation d’une information.
Les données peuvent être sous forme:
1. Numérique
2. Textuelle
3. Visuel
4. Audio

B. **Donnez et expliquez les critères de mesure de qualité des données.**

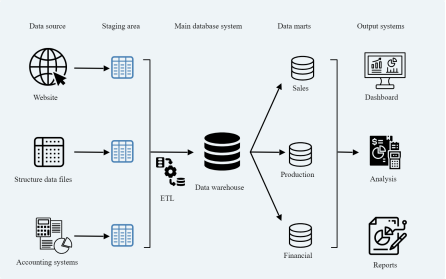
1. **Exactitude** :
   Les données doivent être correctes et précises.
2. **Exhaustivité**:
   Toutes les données nécessaires doivent être présentes.
3. **Actualité** :
   Les données doivent être à jour.
4. **Cohérence** :
   Les données doivent être cohérentes à travers l'ensemble du système.
5. **Fiabilité** :
   Les données doivent être fiables et crédibles.
6. **Précision** :
   Les données doivent être suffisamment détaillées pour répondre aux besoins de l'utilisateur.
7. **Accessibilité** :
   Les données doivent être facilement accessibles.
8. **Intégrité** :
   Les données doivent être protégées contre la corruption ou la perte.
9. **Sécurité**
   Les données doivent être protégées contre les accès non autorisés, la modification ou la destruction.

C. **Définissez et comparez les notions de Data Lake, Data Warehouse et Lake. House. Illustrez les différences à l’aide de schémas.**

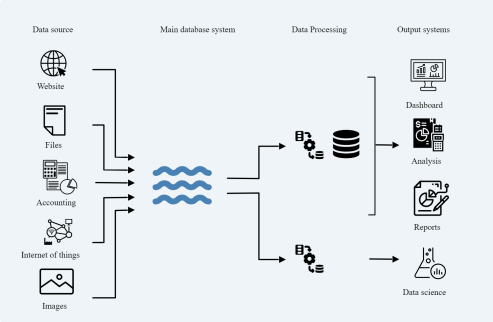
1. Un **Data Lake** est un référentiel de stockage qui permet de stocker des données brutes, non structurées et structurées, à grande échelle et permet de stocker ces données dans leur format d'origine sans nécessiter une structuration préalable.



2. Un **Data Warehouse** est un système centralisé de stockage de données. Il intègre des données de différentes sources, les transforme en un format structuré et offre une vue consolidée des données pour faciliter l'analyse.



3. Le concept de **Lake House** vise à combiner les avantages du Data Lake (stockage flexible de données brutes) et du Data Warehouse (structure et performances pour l'analyse). Il cherche à intégrer la simplicité du stockage dans un Data Lake avec la fiabilité et la performance d'un Data Warehouse.



D. Donnez une définition et des exemples de systèmes de gestion de bases de données avec des illustrations.

Un système de gestion de base de données (SGBD) est un logiciel conçu pour gérer et organiser des données de manière efficace. Il offre un ensemble de fonctionnalités permettant de stocker, d'organiser, de récupérer et de manipuler des données de manière sécurisée et cohérente. Parmis eux:

1. **MySQL** est un SGBD open source très populaire, largement utilisé pour les applications web. Il prend en charge le langage SQL (Structured Query Language) pour la gestion des bases de données relationnelles.
2. **MongoDB** est un SGBD NoSQL qui utilise un modèle de données de type document (JSON-like). Il est adapté au stockage de données semi-structurées ou non structurées.

E. Qu'est ce qu'une base de données relationnelle ? Qu'est ce qu'une base de données non relationnelle ? Donnez la différence entre les deux avec des exemples d'applications.

1. Une **base de données relationnelle** est un type de base de données qui organise les données sous forme de tables interconnectées. Ces tables sont composées de lignes et de colonnes, et les relations entre les tables sont établies à l'aide de clés primaires et étrangères. Ce modèle relationnel offre une structure rigide et organise les données de manière à assurer l'intégrité, la cohérence et la normalisation.
   Applications:
   i. Les applications commerciales utilisent des bases de données relationnelles pour stocker des données sur les clients, les produits, les finances et les opérations.
   ii. Les applications Web utilisent des bases de données relationnelles pour stocker des données sur les utilisateurs, les articles et les commandes.
   iii. Les applications scientifiques utilisent des bases de données relationnelles pour stocker des données expérimentales et des résultats de simulation.
2. Une **base de données non relationnelle**, également appelée NoSQL (Not Only SQL), est un type de base de données qui ne suit pas le modèle tabulaire des bases de données relationnelles. Ces bases de données sont conçues pour gérer des types de données variés et peuvent être plus flexibles en termes de schéma.
   Applications:
   i. Les applications de médias sociaux utilisent des bases de données non relationnelles pour stocker des données sur les utilisateurs, les publications et les interactions.
   ii. Les applications de commerce électronique utilisent des bases de données non relationnelles pour stocker des données sur les produits, les commandes et les retours.
   iii. Les applications de traitement du langage naturel utilisent des bases de données non relationnelles pour stocker des données sur le texte et le langage.

F. Définissez les notions de clé étrangère et clé primaire

1. **Clé primaire:**
   Une clé primaire est une colonne ou un ensemble de colonnes dans une table de base de données qui identifie de manière unique chaque enregistrement dans cette table. Elle garantit qu'il n'y a pas de doublons dans cette colonne, et chaque valeur de la clé primaire doit être unique et non nulle.
2. **Clé étrangère:**
   Une clé étrangère est une colonne (ou un ensemble de colonnes) dans une table qui fait référence à la clé primaire d'une autre table. Elle établit une relation entre les deux tables en reliant les valeurs de la clé étrangère aux valeurs correspondantes de la clé primaire dans une autre table.

G. Quelles sont les propriétés ACID ?

Les propriétés ACID sont un ensemble de caractéristiques qui garantissent la fiabilité et la cohérence des transactions dans les bases de données relationnelles. L'acronyme ACID représente les quatre propriétés suivantes :

1. **Atomicité** (Atomicity) :

   Définition : Une transaction est atomique, ce qui signifie qu'elle est traitée comme une seule unité indivisible. Soit toutes les modifications de la transaction sont effectuées, soit aucune ne l'est.

   Exemple : Si une transaction inclut deux opérations, comme le transfert d'argent d'un compte à un autre et la mise à jour du solde des comptes, alors soit les deux opérations sont effectuées avec succès, soit aucune d'entre elles ne l'est.

2. **Cohérence** (Consistency) :

   Définition : La transaction assure que la base de données passe d'un état valide à un autre état valide. La cohérence garantit que toutes les contraintes d'intégrité sont respectées avant et après l'exécution d'une transaction.

   Exemple : Si une transaction doit respecter une contrainte d'intégrité, comme la non-négativité du solde d'un compte, alors cette contrainte doit être vérifiée et maintenue avant et après l'exécution de la transaction.

3. **Isolation** (Isolation) :

   Définition : Chaque transaction s'exécute de manière isolée par rapport aux autres transactions. Les résultats intermédiaires d'une transaction ne sont pas visibles par d'autres transactions tant que la transaction n'est pas validée.

   Exemple : Si deux transactions concurrentes tentent de mettre à jour les mêmes données, l'isolation garantit que chaque transaction voit une vue cohérente de la base de données, même si les transactions sont exécutées simultanément.

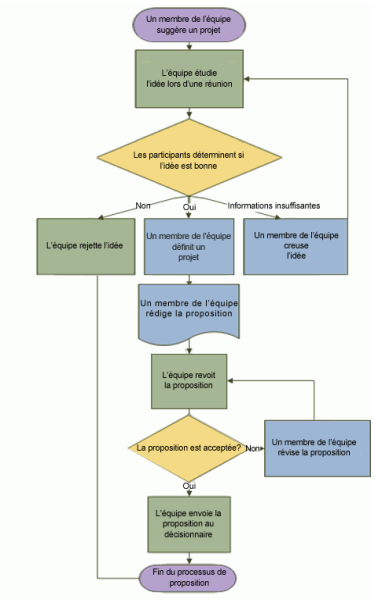
4. **Durabilité** (Durability) :

   Définition : Une fois qu'une transaction est validée, ses modifications sont permanentes, même en cas de panne du système. Les données modifiées sont persistantes et survivent à un redémarrage du système.

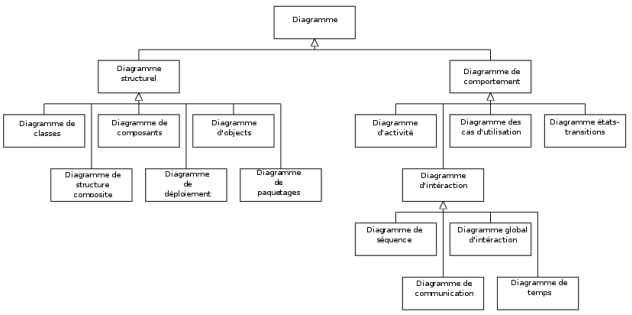
Exemple : Après la confirmation de la réussite d'une transaction, les modifications apportées à la base de données, telles que l'écriture sur le disque, sont durables et ne seront pas perdues même en cas de panne du système.

H. **Définissez les méthodes Merise et UML. Quelles sont leur utilité dans le monde de l'informatique ? Donnez des cas précis d'utilisation avec des schémas.**

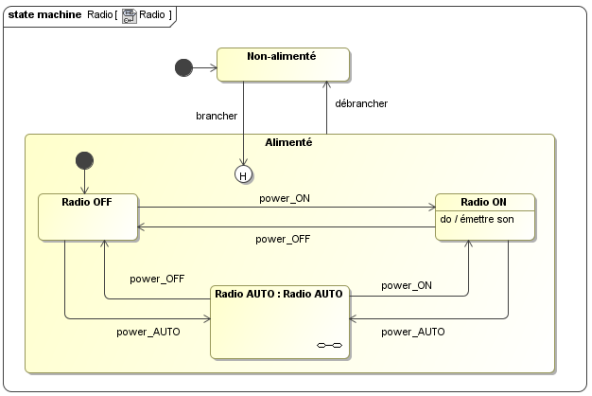
1. **Merise** est une méthode d'analyse et de conception de systèmes d'information (SI) développée en France dans les années 1970. Elle est basée sur un cycle de vie en quatre phases :
    i. Phase d'analyse : identification des besoins des utilisateurs et des exigences du SI.
    ii. Phase de conception : définition des structures de données, des traitements et des interfaces du SI.
    iii. Phase de réalisation : développement du SI en fonction de la conception.
    iv. Phase de maintenance : maintenance et évolution du SI.
2. **UML** (Unified Modeling Language) est un langage de modélisation graphique des systèmes d'information développé par l'Object Management Group (OMG). Il est basé sur un ensemble de diagrammes et de schémas pour représenter les différents aspects d'un système.
3. Les diagrammes suivants peuvent être utilisés:
    i. Diagramme de flux



ii. Diagramme de classe



iii. Diagramme d'état



I. Définissez le langage SQL. Donnez les commandes les plus utilisées de ce langage et les différentes jointures qu'il est possible de faire.

SQL (Structured Query Language) est un langage informatique normalisé servant à exploiter des bases de données relationnelles. Il permet de créer, de modifier et d'accéder aux données d'une base de données.

Les commandes les plus utilisées de SQL sont les suivantes :

1. SELECT: permet de sélectionner des données d'une base de données.
2. INSERT: permet d'insérer des données dans une base de données.
3. UPDATE: permet de modifier des données dans une base de données.
4. DELETE: permet de supprimer des données d'une base de données.

Les jointures permettent de combiner les données de deux ou plusieurs tables. Il existe différents types de jointures, dont les suivantes :

1. JOINTURE INNER: retourne les enregistrements qui correspondent aux deux tables.
2. JOINTURE LEFT OUTER: retourne tous les enregistrements de la première table, même si ceux-ci ne correspondent pas à la deuxième table.
3. JOINTURE RIGHT OUTER: retourne tous les enregistrements de la deuxième table, même si ceux-ci ne correspondent pas à la première table.
4. JOINTURE FULL OUTER: retourne tous les enregistrements des deux tables, même si ceux-ci ne correspondent pas à l'autre table.

JOB 0

Outils installé

JOB 1

1. **SELECT** Population **FROM** WORLD
   WHERE **TRIM**(Country) = 'Germany'

2. **SELECT** Population **FROM** WORLD
   WHERE **TRIM**(Country) = 'Sweden'

   SELECT Population **FROM** WORLD
   WHERE **TRIM**(Country) = 'Norway'

   SELECT Population **FROM** WORLD
   WHERE **TRIM**(Country) = 'Denmark'

3. **SELECT** Country **FROM** WORLD
   WHERE **"Area (sq. mi.)" BETWEEN 200000 AND 300000**

JOB 2

1. **SELECT** Country **FROM** WORLD
   WHERE **TRIM**(Country) **LIKE** "B%"

2. **SELECT** Country **FROM** WORLD
   WHERE **TRIM**(Country) **LIKE** "Al%"

3. **SELECT** Country **FROM** WORLD
   WHERE **TRIM**(Country) **LIKE** "%y"

4. **SELECT** Country **FROM** WORLD
   WHERE **TRIM**(Country) **LIKE** "%land"

5. **SELECT** Country **FROM** WORLD
   WHERE **TRIM**(Country) **LIKE** "%w%"

6. **SELECT** Country **FROM** WORLD
   WHERE **TRIM**(Country) **LIKE** "%oo%" **OR** **TRIM**(Country) **LIKE** "%ee%"

7. **SELECT** Country **FROM** WORLD
   WHERE **TRIM**(Country) **LIKE** "%a%a%a%"

8. ```
   SELECT Country FROM WORLD
   WHERE  TRIM(Country) LIKE "_r%"
   ```

JOB 3

1. ```
   PRAGMA table_info(WORLD)
   ```

2. ```
   SELECT * FROM students
   WHERE age > 20
   ```

JOB 4

1. ```
   SELECT * FROM nobel
   WHERE yr = 1986
   ```

2. ```
   SELECT * FROM nobel
   WHERE yr = 1967 AND subject = 'Literature'
   ```

3. ```
   SELECT yr, subject FROM nobel
   WHERE winner = 'Albert Einstein'
   ```

4. ```
   SELECT * FROM nobel
   WHERE yr BETWEEN 1980 AND 1989
   ```

5. 0, ca n’existe pas

JOB 5

1. ```
   SELECT Country FROM WORLD
   WHERE Population > (SELECT Population FROM WORLD WHERE TRIM(Country) =
   'Russia')
   ```

2. ```
   SELECT Country FROM WORLD
   WHERE "GDP ($ per capita)" > (SELECT "GDP ($ per capita)"  FROM WORLD WHERE
   TRIM(Country) = 'Italy')  AND Region LIKE '%EUROPE%'
   ```

3. ```
   SELECT Country FROM WORLD
   WHERE Population BETWEEN (SELECT Population FROM WORLD WHERE
   TRIM(Country) = 'United Kingdom') + 1 AND (SELECT Population FROM WORLD WHERE
   TRIM(Country) = 'Germany') - 1
   ```

4. ```
   SELECT Country,
   ```
   ```

```
CONCAT(ROUND(100 * CAST(Population AS REAL) / CAST((SELECT Population FROM
WORLD WHERE TRIM(Country) = 'Germany') AS REAL), 1), '%')  AS
pourcentage_Allemagne
FROM WORLD
WHERE TRIM(Country) <> 'Germany' AND Region LIKE '%EUROPE%'
```

5.
```
WITH ClassementParContinent AS(
        SELECT
                Country,
                Region,
                "Area (sq. mi.)",
                ROW_NUMBER() OVER (PARTITION BY Region ORDER BY "Area (sq.
mi.)" DESC) AS Rang
        FROM WORLD
        )
SELECT
        Country,
        Region,
        "Area (sq. mi.)"
FROM ClassementParContinent
WHERE Rang = 1
ORDER BY "Area (sq. mi.)" DESC
```

6.
```
SELECT Region
FROM WORLD
GROUP BY Region
HAVING MAX(Population) < 25000000
```

JOB 6

1.
```
SELECT SUM(Population)
FROM WORLD
```

2.
```
SELECT Region , SUM(Population) AS "Population par continent"
FROM WORLD
GROUP BY Region
ORDER BY "Population par continent" DESC
```

3.
```
SELECT Region , SUM(Population * "GDP ($ per capita)") AS "Population par continent"
FROM WORLD
GROUP BY Region
ORDER BY "Population par continent" DESC
```

4.
```
SELECT SUM(Population * "GDP ($ per capita)") AS "Population par continent"
FROM WORLD
WHERE TRIM(Region) LIKE '%AFRICA%'
```

5.
```
SELECT COUNT(*)  AS "Nb de pays dont la superficie est supérieure à 1000000m²"
```

```
FROM WORLD
WHERE Population >= 1000000/2560000
```

6.
```
SELECT Country , Population
FROM WORLD
WHERE TRIM(Region) = 'BALTICS'
```

7.
```
SELECT Region , COUNT(*) AS "Nombre de pays"
FROM WORLD
GROUP BY Region
ORDER BY "Nombre de pays" DESC
```

8.
```
SELECT Region , SUM(Population)  AS "Population par continent"
FROM WORLD
GROUP BY Region
HAVING  "Population par continent" > 100000000
ORDER BY "Population par continent" DESC
```

JOB 7

1. Etude de la cardinalité:
    a. Un match peut contenir plusieurs équipes (en théorie 2).
    b. Une équipe ne peut être qu'une seule fois dans un match
    c. 1 but ne peut être associé qu'à une seule équipe lors d'un match
    d. 1 but ne peut être associés qu'à un seul match
    e. Une équipe peut avoir plusieurs buts
    f. Un match peut avoir plusieurs buts

2.
```
SELECT matchid, player
FROM Goal
WHERE teamid = 'GER'
```

3.
```
SELECT id, stadium, team1, team2
FROM Game
WHERE id = 1012
```

4.
```
SELECT player, teamid, stadium, mdate
FROM Game JOIN Goal ON (id=matchid)
WHERE teamid = 'GER'
```

5.
```
SELECT team1, team2, player
FROM Game JOIN Goal ON (id=matchid)
WHERE player LIKE 'Mario%'
```

6.
```
SELECT *
FROM Goal JOIN Eteam ON (teamid=id)
```

7.
```
SELECT player, teamid, coach, gtime
FROM Goal JOIN Eteam ON (teamid=id)
```

```
WHERE gtime <= 10
```

8. ```
   SELECT mdate, teamname
   FROM Game JOIN Eteam ON (team1=Eteam.id)
   WHERE coach = 'Fernando Santos'
   ```

9. ```
   SELECT player
   FROM Game JOIN Goal ON (id=matchid)
   WHERE stadium LIKE 'National Stadium, Warsaw'
   ```

10. ```
    SELECT teamname, COUNT(*) AS "Nombre de but par équipe"
    FROM Goal JOIN Eteam ON (teamid=id)
    GROUP BY teamname
    ORDER BY "Nombre de but par équipe" DESC
    ```

11. ```
    SELECT stadium COUNT(*) AS "Nombre de but par stade"
    FROM Game JOIN Goal ON (id=matchid)
    GROUP BY stadium
    ORDER BY "Nombre de but par stade" DESC
    ```

12. ```
    SELECT id, mdate, COUNT(*) AS "Nombre de but par match de l'équipe de France"
    FROM Game JOIN Goal ON (id=matchid)
    WHERE teamid = 'FRA'
    GROUP BY id
    ORDER BY "Nombre de but par match de l'équipe de France" DESC
    ```

JOB 8

1. sqlite3 SomeCompany.db

2. ```
   CREATE TABLE Employes (
     employee_id INT PRIMARY KEY,
     first_name TEXT,
     last_name TEXT,
     birthdate DATE,
     position TEXT,
     department_id INT
   )
   ```

3. ```
   CREATE TABLE Departments (
     department_id INT PRIMARY KEY,
     department_name TEXT,
     department_head INT,
     location TEXT
   )
   ```

4. ```
   INSERT INTO
   Employes (employee_id, first_name, last_name, birthdate, "position", department_id)
   VALUES
   ```

```
(1, 'John', 'Doe', '1990-05-15', 'Software Engineer', 1),
(2, 'Jane', 'Smith', '1985-08-20', 'Project Manager', 2),
(3, 'Mike', 'Johnson', '1992-03-10', 'Data Analyst', 1),
(4, 'Emily', 'Brown', '1988-12-03', 'UX Designer', 1),
(5, 'Alex', 'Williams', '1995-06-28', 'Software Developer', 1),
(6, 'Sarah', 'Miller', '1987-09-18', 'HR Specialist', 3),
(7, 'Ethan', 'Clark', '1991-02-14', 'Database Administrator', 1),
(8, 'Olivia', 'Garcia', '1984-07-22', 'Marketing Manager', 2),
(9, 'Emilia', 'Clark', '1986-01-12', 'HR Manager', 3)
```

5. ```
   SELECT last_name , "position"
   FROM Employes
   ```

6. ```
   UPDATE Employes
   SET last_name = 'Biden'
   WHERE last_name = 'Garcia'
   ```

7. ```
   DELETE FROM Employes
   WHERE last_name = 'Biden'
   ```

8. ```
   SELECT last_name , department_name, location
   FROM Employes JOIN Departments ON (Employes.department_id =
   Departments.department_id)
   ```

9. ```
   SELECT *
   FROM Employes JOIN Departments ON (Employes.department_id =
   Departments.department_id)
   ORDER BY department_id
   ```

10. ```
    SELECT department_name , CONCAT(Employes.first_name, ' ' , Employes.last_name) AS
    manager
    FROM Departments JOIN Employes ON (Departments.department_head =
    Employes.employee_id)
    ORDER BY department_name
    ```

11. Réponse
    a. Pour ajouter le nouveau service à la table departments:
    ```
    INSERT INTO
    Departments (department_id, department_name, department_head, location)
    VALUES
    (4, 'Marketing', 4, 'Branch Office Middle')
    ```

    b. Pour modifier les employés:
    ```
    UPDATE Employes
    SET department_id = 4
    WHERE employee_id IN (3,4,10)
    ```

12. Réponse
    a. Création de la table:
    ```
    CREATE TABLE Project (
      project_id INT PRIMARY KEY,
    ```

```
  project_name TEXT,
  start_date DATE,
  end_date DATE,
  department_id INT
)
```

b. Ajout des projets

```
INSERT INTO
Project (project_id , project_name , start_date , end_date , department_id)
VALUES
(1, 'Rocket to the moon', '2023-05-15', '2025-09-25', 1),
(2, 'Development', '2023-01-17', '2024-11-01', 2),
(3, 'Test', '2024-11-02', '2025-09-25', 2),
(4, 'Hiring', '2023-04-30', '2023-06-30', 3),
(5, 'Firing', '2023-10-26', '2023-12-25', 3),
(6, 'Ads', '2023-04-30', '2025-09-25', 3)
```

JOB 9

Pour rendre les chiffres réels exploitables:

```
UPDATE WORLD
SET
"Pop. Density (per sq. mi.)" = REPLACE ("Pop. Density (per sq. mi.)", ",", "."),
"Coastline (coast/area ratio)" = REPLACE ("Coastline (coast/area ratio)", ",", "."),
"Net migration" = REPLACE ("Net migration", ",", "."),
"Infant mortality (per 1000 births)" = REPLACE ("Infant mortality (per 1000 births)", ",", "."),
"Literacy (%)" = REPLACE ("Literacy (%)", ",", "."),
"Phones (per 1000)" = REPLACE ("Phones (per 1000)", ",", "."),
"Arable (%)" = REPLACE ("Arable (%)", ",", "."),
"Crops (%)" = REPLACE ("Crops (%)", ",", "."),
"Other (%)" = REPLACE ("Other (%)", ",", "."),
"Birthrate" = REPLACE ("Birthrate", ",", "."),
"Deathrate" = REPLACE ("Deathrate", ",", "."),
"Agriculture" = REPLACE ("Agriculture", ",", "."),
"Industry" = REPLACE ("Industry", ",", "."),
"Service" = REPLACE ("Service", ",", ".")
```

1. **Liste des pays sans côte maritime**

```
SELECT Country AS "Pays sans cote maritime"
FROM WORLD
WHERE "Coastline (coast/area ratio)" = 0
```

2. **Les 10 pays les plus denses**

```
SELECT Country, "Pop. Density (per sq. mi.)"
FROM WORLD
ORDER BY "Pop. Density (per sq. mi.)" DESC
LIMIT 10
```

3. **Liste des pays ou il y a plus d'un téléphone par personne**

```
SELECT Country,"Phones (per 1000)"
FROM WORLD
WHERE "Phones (per 1000)" > 1000 AND COALESCE("Phones (per 1000)", '')
```

4. **Taux d'alphabétisation par continent**

```
SELECT Region , AVG("Literacy (%)") AS "Taux d'alphabétisation"
FROM WORLD
GROUP BY Region
ORDER BY "Taux de lettrisme par continent" DESC
```

5. **Liste des pays dans lesquels la démographie diminue**

```
SELECT Country , (Birthrate - Deathrate) AS "croissance"
FROM WORLD
WHERE "croissance" < 0
```

6. **Les 10 pays les plus pauvres**

```
SELECT Country, "GDP ($ per capita)"
FROM WORLD
ORDER BY "GDP ($ per capita)" ASC
LIMIT 10
```