**Requêtes SQL**

**1)Sélectionnez les dresseurs ayant au moins 7 pokemons (JOIN, COUNT, GROUP BY, HAVING)**

SELECT d.id, d.nom, COUNT(p.id) AS nombre\_pokemons

FROM dresseur d

JOIN pokemons p ON d.id = p.id\_dresseur

GROUP BY d.id, d.nom

HAVING COUNT(p.id) >= 7;

**2) Selectionne les 5 pokemons les plus haut niveau (LIMIT, ORDER BY)**

SELECT \* FROM pokemons

ORDER BY niveau DESC

LIMIT 5;

**3)Afficher le nombres de pokemons pour chaque types ( COUNT, JOIN, GROUP BY)**

SELECT t.nom AS type, COUNT(p.id) AS nombre\_de\_pokemons

FROM types t

JOIN est\_de\_type edt ON t.id = edt.id\_type

JOIN pokemons p ON edt.id\_pokemon = p.id

GROUP BY t.nom;

**4) On veut les noms des dresseurs commençant par la lettre « P » (LIKE)**

SELECT \* FROM dresseur

WHERE nom LIKE 'B%';

**5) Afficher les noms des Pokémons appartenant aux types 'Plante' ou 'Eau' (IN, JOIN)**

SELECT p.nom AS nom\_du\_pokemon, t.nom AS type

FROM pokemons p

JOIN est\_de\_type edt ON p.id = edt.id\_pokemon

JOIN types t ON edt.id\_type = t.id

WHERE t.nom IN ('Plante', 'Eau');

**6) Afficher les pokemons dont les noms commencent par "R" et dont le niveau est compris entre 20 et 30 (LIKE,BETWEEN)**

SELECT \* FROM pokemons

WHERE nom LIKE 'R%' AND niveau BETWEEN 7 AND 10

**7) Augmenter les niveaux des pokémons dont le nom commence par "S" de 10. (UPDATE, LIKE)**

UPDATE pokemons

SET niveau = niveau + 10

WHERE nom LIKE 'S%';

**8) Afficher les noms des dresseurs qui ont des Pokémon dont le niveau est supérieur à la moyenne des pokemons. Utilisez Union. (DINSTINCT,JOIN,AVG, requete imbriqué, UNION)**

SELECT DISTINCT d.nom

FROM dresseur d

JOIN pokemons p ON d.id = p.id\_dresseur

WHERE p.niveau > (SELECT AVG(niveau) FROM pokemons)

UNION

SELECT DISTINCT d.nom

FROM dresseur d

JOIN pokemons p ON d.id = p.id\_dresseur

WHERE d.id IN (

SELECT DISTINCT d.id

FROM dresseur d

JOIN pokemons p ON d.id = p.id\_dresseur

WHERE p.niveau > (SELECT AVG(niveau) FROM pokemons)

);

**9) Une erreur a été fait lors de la création de la base de donnée. En effet, nous avons crée une table « dresseur » au lieu de « dresseurs ». Afin de rester cohérent avec les noms des autres tables, nous aimerions palier à cela. (CREATE TABLE , INSERT INTO, SELECT, DROP TABLE). Pour assurer l’intégrité des données, nous allons utiliser les transactions.**

CREATE TABLE dresseurs(

id INT PRIMARY KEY AUTO\_INCREMENT,

nom VARCHAR(50) NOT NULL,

age INT,

genre VARCHAR(1)

);

INSERT INTO dresseurs(nom, age, genre)

SELECT nom, age, genre FROM dresseur;

DROP TABLE dresseur;

**10) Nous avons 3 pikachu, nous aimerions supprimé celui appartenant au dresseur Charlie. (DELETE, SELECT imbriqué)**

DELETE FROM pokemons

WHERE nom = "pikachu" AND id\_dresseur = (SELECT id FROM dresseur WHERE nom = "Charlie");

**11) Sélectionner les noms des dresseurs en fonction du genre en utilisant CASE. (DISTINCT,JOIN,CASE)**

SELECT DISTINCT d.nom

FROM dresseur d

JOIN pokemons p ON d.id = p.id\_dresseur

WHERE CASE

WHEN d.genre = 'F' THEN 1

ELSE 0

END = 1;

**12) On aimerait ajouter une table qui permet d’enregistrer les resultats des combats entre les dresseurs. Pour ce faire, on crée une table combat avec l’id du dresseur\_vainqueur et du dresseur\_perdant et le match\_nul.**

CREATE TABLE combat (

id INT PRIMARY KEY AUTO\_INCREMENT,

id\_dresseur\_vainqueur INT NOT NULL REFERENCES dresseurs(id),

id\_dresseur\_perdant INT NOT NULL REFERENCES dresseurs(id),

match\_nul INT NOT NULL

);

**13) Finalement on voudrait aussi avoir la date du combat dans la table. (ALTER TABLE)**

ALTER TABLE combat

ADD COLUMN date\_combat DATE NOT NULL;

**14) On veut inserer quelques lignes de données pour tester la table. (INSERT INTO)**

INSERT INTO combat (id\_dresseur\_vainqueur, id\_dresseur\_perdant, match\_nul, date\_combat)

VALUES

(1, 2, FALSE, '2024-01-20'), -- Victoire du dresseur 1 sur le dresseur 2

(3, 4, FALSE, '2024-02-05'), -- Victoire du dresseur 3 sur le dresseur 4

(2, 5, TRUE, '2024-03-10'); -- Match nul entre le dresseur 2 et le dresseur 5

**15) Maintenant qu’on a verifié que la table fonctionnait correctement, on aimerait vider la table. (TRUNCATE TABLE)**

TRUNCATE TABLE combat;

**16) Affiche moi le plan d’exécution de la requête SQL « SELECT \* FROM dresseurs » (EXPLAIN)**

EXPLAIN SELECT \* FROM dresseurs;

**17) Rajouter un dresseur s’appelant Charlie, ayant 18 ans qui est de genre N. On souhaite maintenant supprimer le dresseur Charlie ayant 18 ans que l’on vient d’ajouter. (SELECT, DELETE, WHERE, LIMIT, ORDER BY, TRANSACTION)**

INSERT INTO dresseurs(nom, age, genre) VALUES(‘Charlie’, 18, ‘N’) ;

SET autocommit=0 ;

SELECT \* FROM dresseurs ;

DELETE FROM dresseurs WHERE nom = ‘Charlie’ AND age = 18 AND genre = ‘N’ ;

SELECT \* FROM dresseurs ;

ROLLBACK ;

SELECT \* FROM dresseurs ;

DELETE FROM dresseurs WHERE id = (SELECT id FROM dresseurs WHERE nom = ‘Charlie’ AND age = 18 AND genre = ‘N’ ORDER BY id DESC LIMIT 1) ;

SELECT \* FROM dresseurs ;

COMMIT ;

SET autocommit=1 ;

**18) Créer une vue où l’on veut uniquement les dresseurs de genre masculin dans la table des dresseurs.**

CREATE VIEW vue\_dresseurs AS SELECT \* FROM dresseurs WHERE genre = 'H';

**Pour afficher la table on aura juste à mettre :**

SELECT \* FROM vue\_dresseurs;

**19) Créons une vue qui rassemble des informations sur les Pokémon, les dresseurs, les types et les attaques, pour qu’on puisse y acceder plus facilement.**

CREATE VIEW Infopokemons AS

SELECT

p.id AS id\_pokemon,

p.nom AS nom\_pokemon,

p.niveau,

p.exp\_en\_cours,

d.id AS id\_dresseur,

d.nom AS nom\_dresseur,

t.nom AS type,

a.nom AS attaque

FROM pokemons p JOIN dresseurs d ON p.id\_dresseur = d.id

JOIN est\_de\_type et ON p.id = et.id\_pokemon

JOIN types t ON et.id\_type = t.id

JOIN connait\_attaque ca ON p.id = ca.id\_pokemon

JOIN attaques a ON ca.id\_attaque = a.id;

SELECT \* FROM Infopokemons;