MYSQL 2ITECH

Réactivation mémoire

Les composants SQL

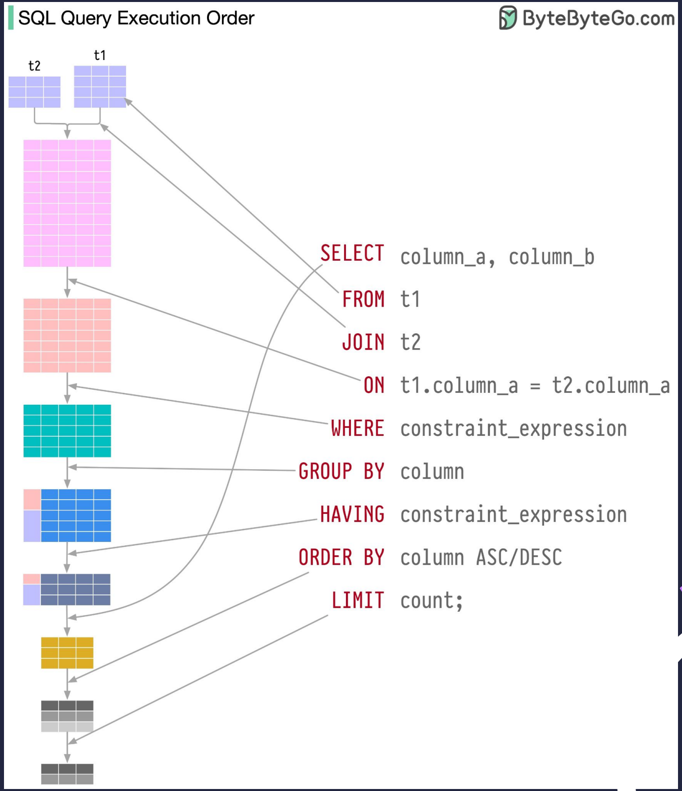
Les composants 5 principaux:

- DDL: data definition language (structure)
- DQL: data query language (lecture et sélection de data)
- DML: data manipulation language (écriture)
- DCL: data control language (contrôle d'accès)
- TCL: transaction control language

+ 3 sous modules:

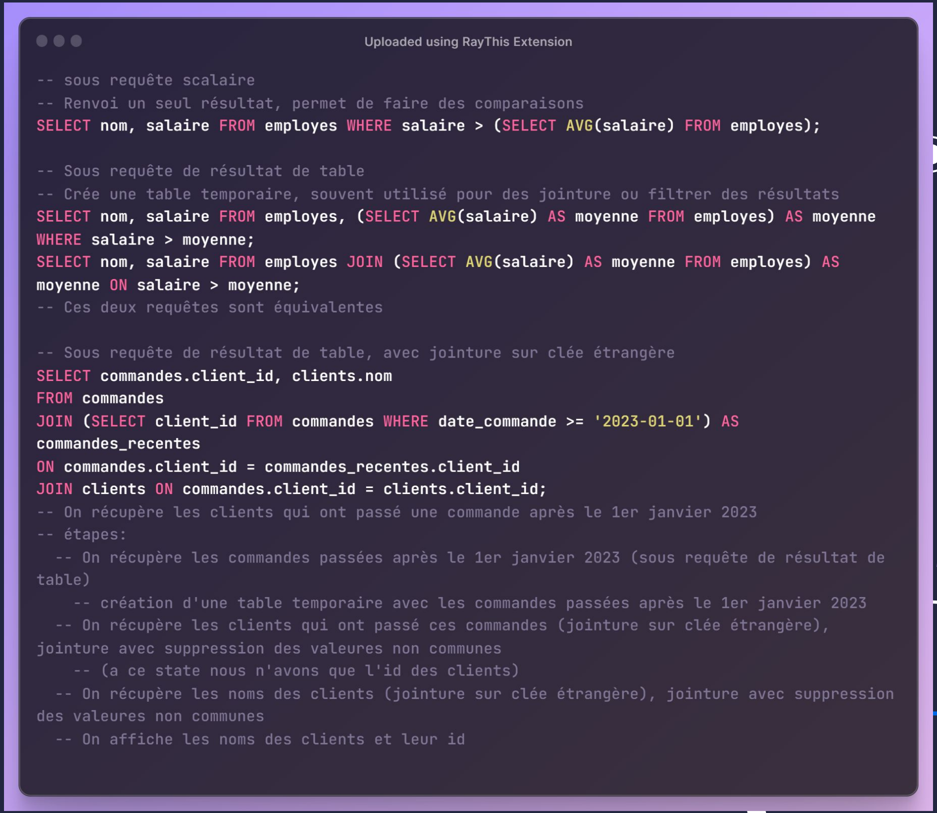
- Data types
- Functions
- Opérators

Query execution order



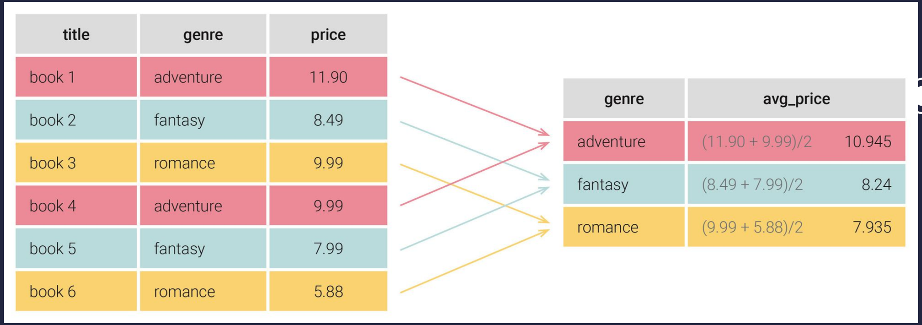
```
SELECT column_a, column_b
  FROM t1
  JOIN t2
    ON t1.column_a = t2.column_a
 WHERE constraint_expression
GROUP BY column
HAVING constraint_expression
ORDER BY column ASC/DESC
 LIMIT count;
```

Sous requêtes


Uploaded using RayThis Extension

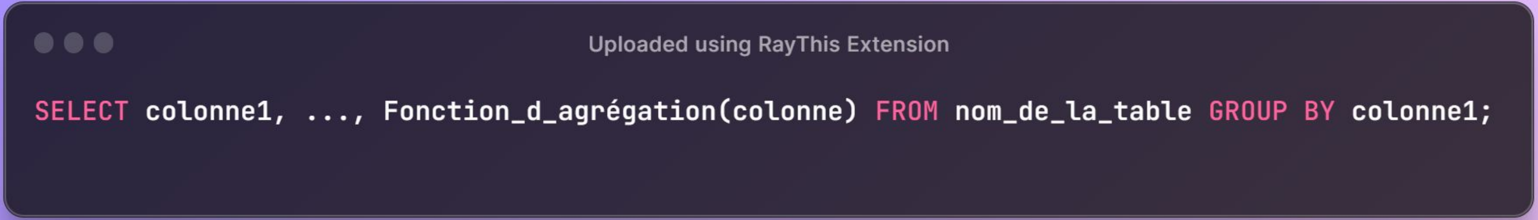
```sql
-- sous requête scalaire
-- Renvoi un seul résultat, permet de faire des comparaisons
SELECT nom, salaire FROM employes WHERE salaire > (SELECT AVG(salaire) FROM employes);

-- Sous requête de résultat de table
-- Crée une table temporaire, souvent utilisé pour des jointure ou filtrer des résultats
SELECT nom, salaire FROM employes, (SELECT AVG(salaire) AS moyenne FROM employes) AS moyenne
WHERE salaire > moyenne;
SELECT nom, salaire FROM employes JOIN (SELECT AVG(salaire) AS moyenne FROM employes) AS
moyenne ON salaire > moyenne;
-- Ces deux requêtes sont équivalentes

-- Sous requête de résultat de table, avec jointure sur clée étrangère
SELECT commandes.client_id, clients.nom
FROM commandes
JOIN (SELECT client_id FROM commandes WHERE date_commande >= '2023-01-01') AS
commandes_recentes
ON commandes.client_id = commandes_recentes.client_id
JOIN clients ON commandes.client_id = clients.client_id;
-- On récupère les clients qui ont passé une commande après le 1er janvier 2023
-- étapes:
  -- On récupère les commandes passées après le 1er janvier 2023 (sous requête de résultat de
table)
    -- création d'une table temporaire avec les commandes passées après le 1er janvier 2023
  -- On récupère les clients qui ont passé ces commandes (jointure sur clée étrangère),
jointure avec suppression des valeures non communes
    -- (a ce state nous n'avons que l'id des clients)
  -- On récupère les noms des clients (jointure sur clée étrangère), jointure avec suppression
des valeures non communes
  -- On affiche les noms des clients et leur id
```

Group By



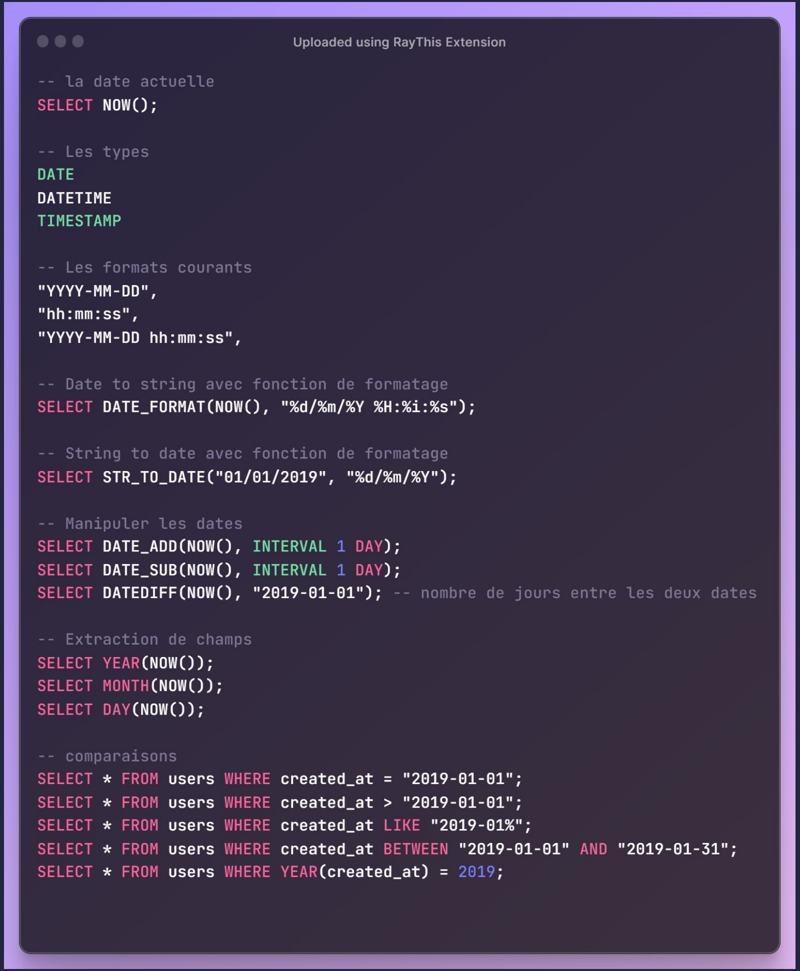
title	genre	price
book 1	adventure	11.90
book 2	fantasy	8.49
book 3	romance	9.99
book 4	adventure	9.99
book 5	fantasy	7.99
book 6	romance	5.88

genre	avg_price	
adventure	(11.90 + 9.99)/2	10.945
fantasy	(8.49 + 7.99)/2	8.24
romance	(9.99 + 5.88)/2	7.935

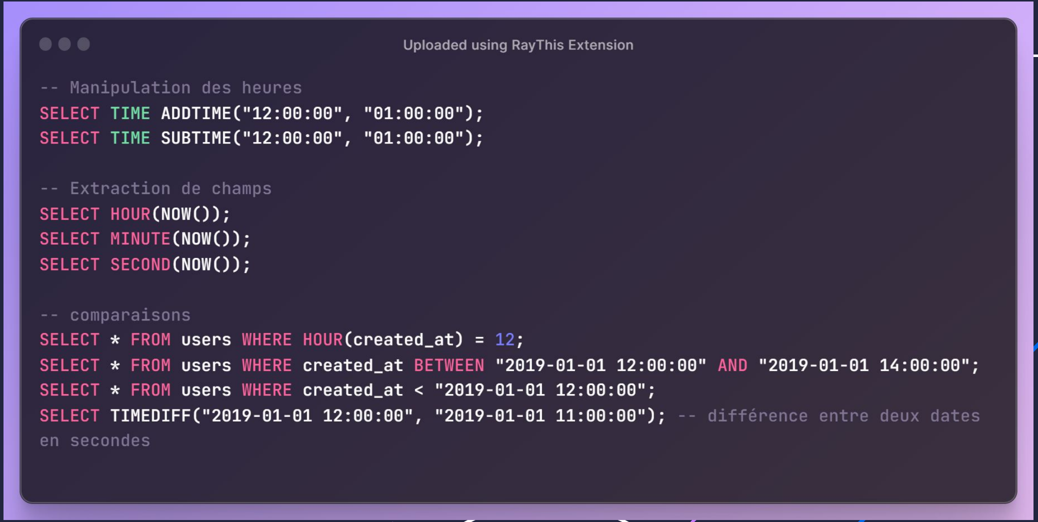


```sql
SELECT colonne1, ..., Fonction_d_agrégation(colonne) FROM nom_de_la_table GROUP BY colonne1;
```

Formatage de date

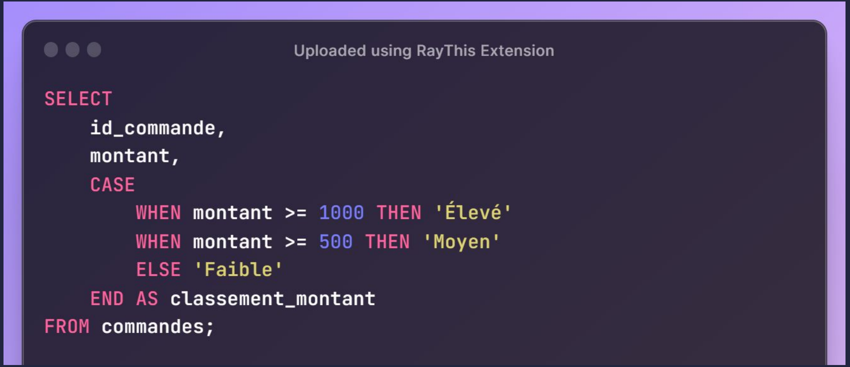


```sql
-- la date actuelle
SELECT NOW();

-- Les types
DATE
DATETIME
TIMESTAMP

-- Les formats courants
"YYYY-MM-DD",
"hh:mm:ss",
"YYYY-MM-DD hh:mm:ss",

-- Date to string avec fonction de formatage
SELECT DATE_FORMAT(NOW(), "%d/%m/%Y %H:%i:%s");

-- String to date avec fonction de formatage
SELECT STR_TO_DATE("01/01/2019", "%d/%m/%Y");

-- Manipuler les dates
SELECT DATE_ADD(NOW(), INTERVAL 1 DAY);
SELECT DATE_SUB(NOW(), INTERVAL 1 DAY);
SELECT DATEDIFF(NOW(), "2019-01-01"); -- nombre de jours entre les deux dates

-- Extraction de champs
SELECT YEAR(NOW());
SELECT MONTH(NOW());
SELECT DAY(NOW());

-- comparaisons
SELECT * FROM users WHERE created_at = "2019-01-01";
SELECT * FROM users WHERE created_at > "2019-01-01";
SELECT * FROM users WHERE created_at LIKE "2019-01%";
SELECT * FROM users WHERE created_at BETWEEN "2019-01-01" AND "2019-01-31";
SELECT * FROM users WHERE YEAR(created_at) = 2019;
```



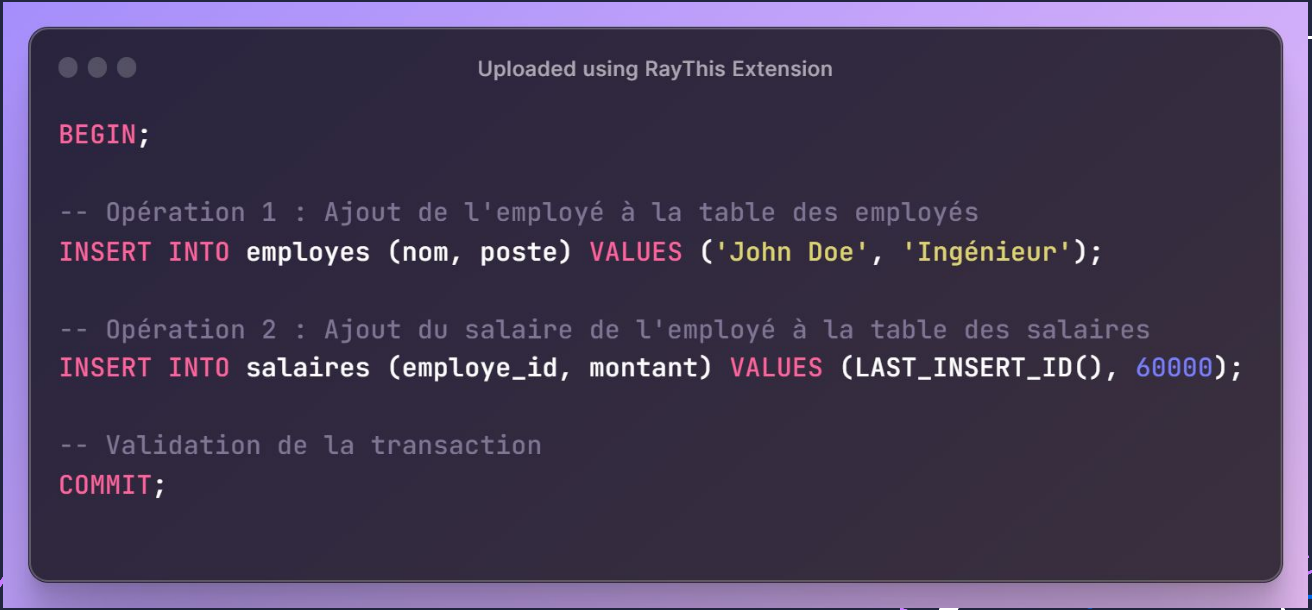
```sql
-- Manipulation des heures
SELECT TIME ADDTIME("12:00:00", "01:00:00");
SELECT TIME SUBTIME("12:00:00", "01:00:00");

-- Extraction de champs
SELECT HOUR(NOW());
SELECT MINUTE(NOW());
SELECT SECOND(NOW());

-- comparaisons
SELECT * FROM users WHERE HOUR(created_at) = 12;
SELECT * FROM users WHERE created_at BETWEEN "2019-01-01 12:00:00" AND "2019-01-01 14:00:00";
SELECT * FROM users WHERE created_at < "2019-01-01 12:00:00";
SELECT TIMEDIFF("2019-01-01 12:00:00", "2019-01-01 11:00:00"); -- différence entre deux dates
en secondes
```

Case


Uploaded using RayThis Extension

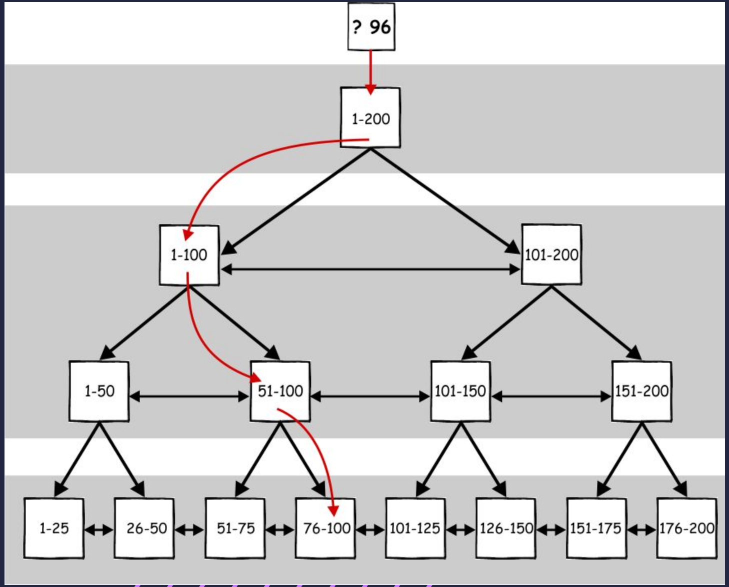
```
SELECT
    id_commande,
    montant,
    CASE
        WHEN montant >= 1000 THEN 'Élevé'
        WHEN montant >= 500 THEN 'Moyen'
        ELSE 'Faible'
    END AS classement_montant
FROM commandes;
```

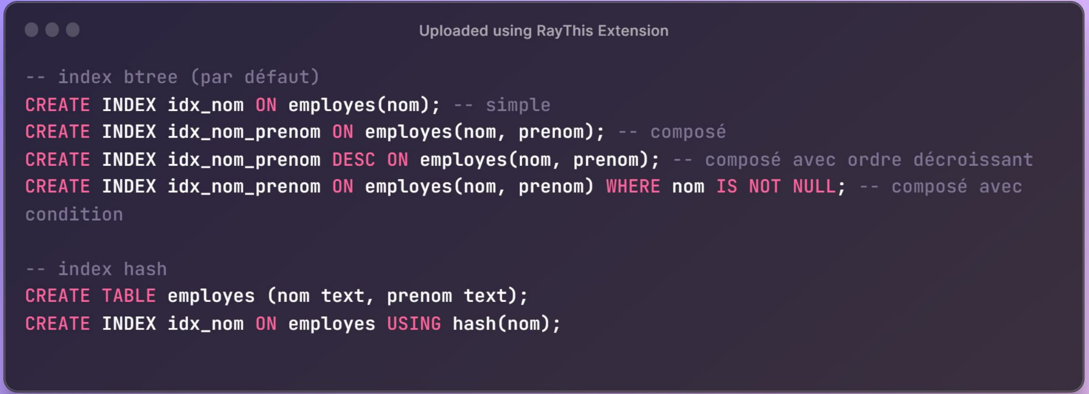
Transactions et contrôle de concurrence



```sql
BEGIN;

-- Opération 1 : Ajout de l'employé à la table des employés
INSERT INTO employes (nom, poste) VALUES ('John Doe', 'Ingénieur');

-- Opération 2 : Ajout du salaire de l'employé à la table des salaires
INSERT INTO salaires (employe_id, montant) VALUES (LAST_INSERT_ID(), 60000);

-- Validation de la transaction
COMMIT;
```

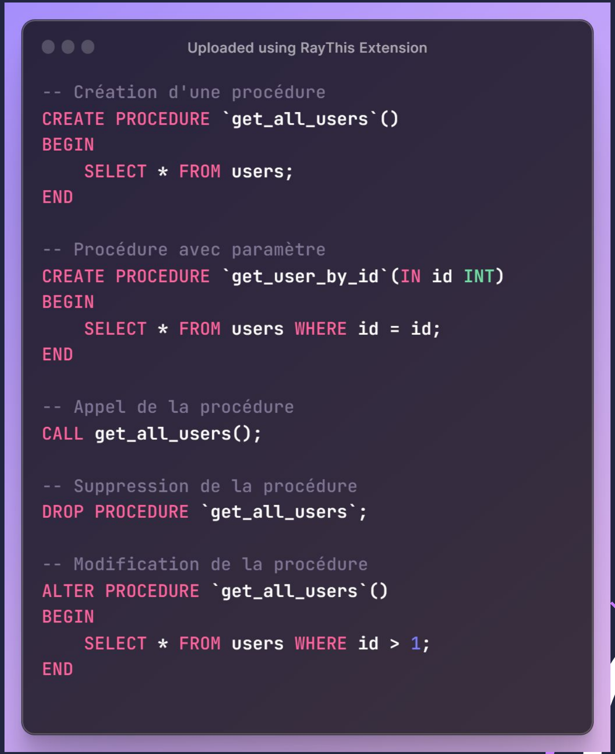
Indexs et optimisation de requêtes





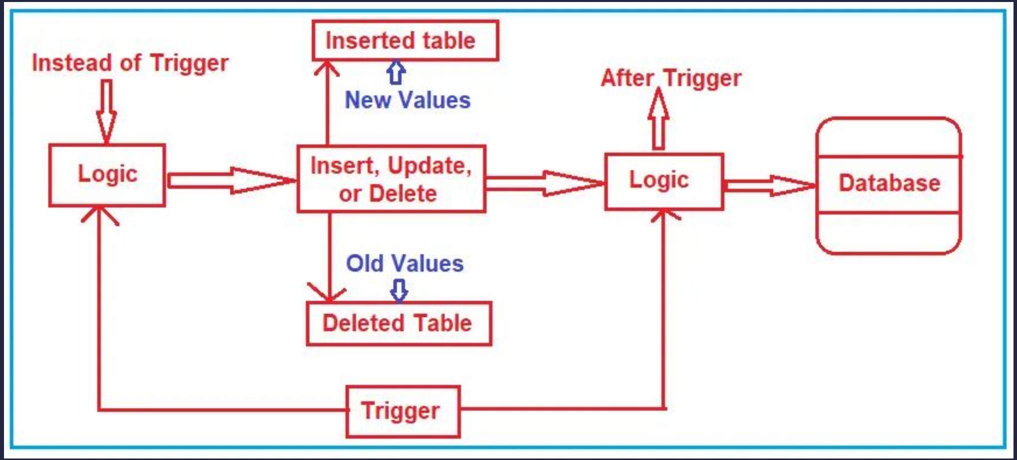
```sql
-- index btree (par défaut)
CREATE INDEX idx_nom ON employes(nom); -- simple
CREATE INDEX idx_nom_prenom ON employes(nom, prenom); -- composé
CREATE INDEX idx_nom_prenom DESC ON employes(nom, prenom); -- composé avec ordre décroissant
CREATE INDEX idx_nom_prenom ON employes(nom, prenom) WHERE nom IS NOT NULL; -- composé avec
condition

-- index hash
CREATE TABLE employes (nom text, prenom text);
CREATE INDEX idx_nom ON employes USING hash(nom);
```

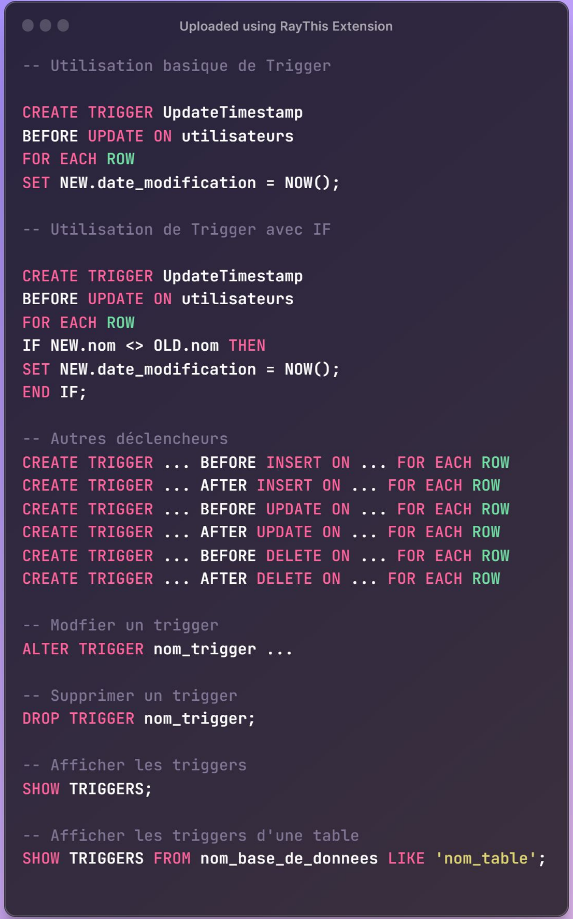
Les procédures


Uploaded using RayThis Extension

```sql
-- Création d'une procédure
CREATE PROCEDURE `get_all_users`()
BEGIN
    SELECT * FROM users;
END

-- Procédure avec paramètre
CREATE PROCEDURE `get_user_by_id`(IN id INT)
BEGIN
    SELECT * FROM users WHERE id = id;
END

-- Appel de la procédure
CALL get_all_users();

-- Suppression de la procédure
DROP PROCEDURE `get_all_users`;

-- Modification de la procédure
ALTER PROCEDURE `get_all_users`()
BEGIN
    SELECT * FROM users WHERE id > 1;
END
```

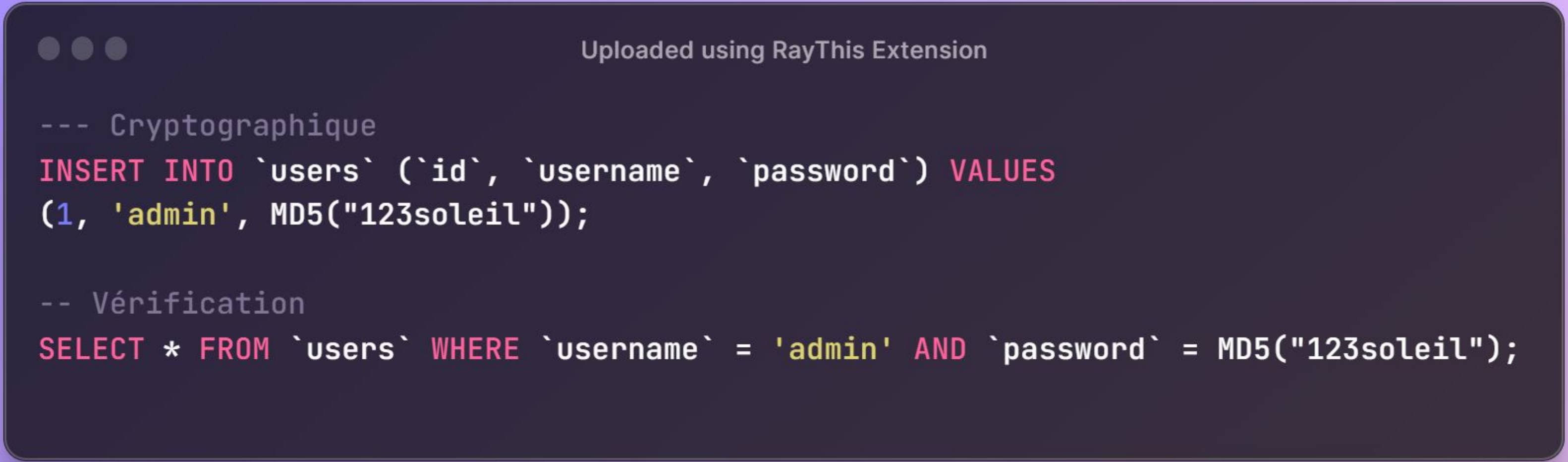
Triggers




Uploaded using RayThis Extension

```sql
-- Utilisation basique de Trigger

CREATE TRIGGER UpdateTimestamp
BEFORE UPDATE ON utilisateurs
FOR EACH ROW
SET NEW.date_modification = NOW();

-- Utilisation de Trigger avec IF

CREATE TRIGGER UpdateTimestamp
BEFORE UPDATE ON utilisateurs
FOR EACH ROW
IF NEW.nom <> OLD.nom THEN
SET NEW.date_modification = NOW();
END IF;

-- Autres déclencheurs
CREATE TRIGGER ... BEFORE INSERT ON ... FOR EACH ROW
CREATE TRIGGER ... AFTER INSERT ON ... FOR EACH ROW
CREATE TRIGGER ... BEFORE UPDATE ON ... FOR EACH ROW
CREATE TRIGGER ... AFTER UPDATE ON ... FOR EACH ROW
CREATE TRIGGER ... BEFORE DELETE ON ... FOR EACH ROW
CREATE TRIGGER ... AFTER DELETE ON ... FOR EACH ROW

-- Modfier un trigger
ALTER TRIGGER nom_trigger ...

-- Supprimer un trigger
DROP TRIGGER nom_trigger;

-- Afficher les triggers
SHOW TRIGGERS;

-- Afficher les triggers d'une table
SHOW TRIGGERS FROM nom_base_de_donnees LIKE 'nom_table';
```

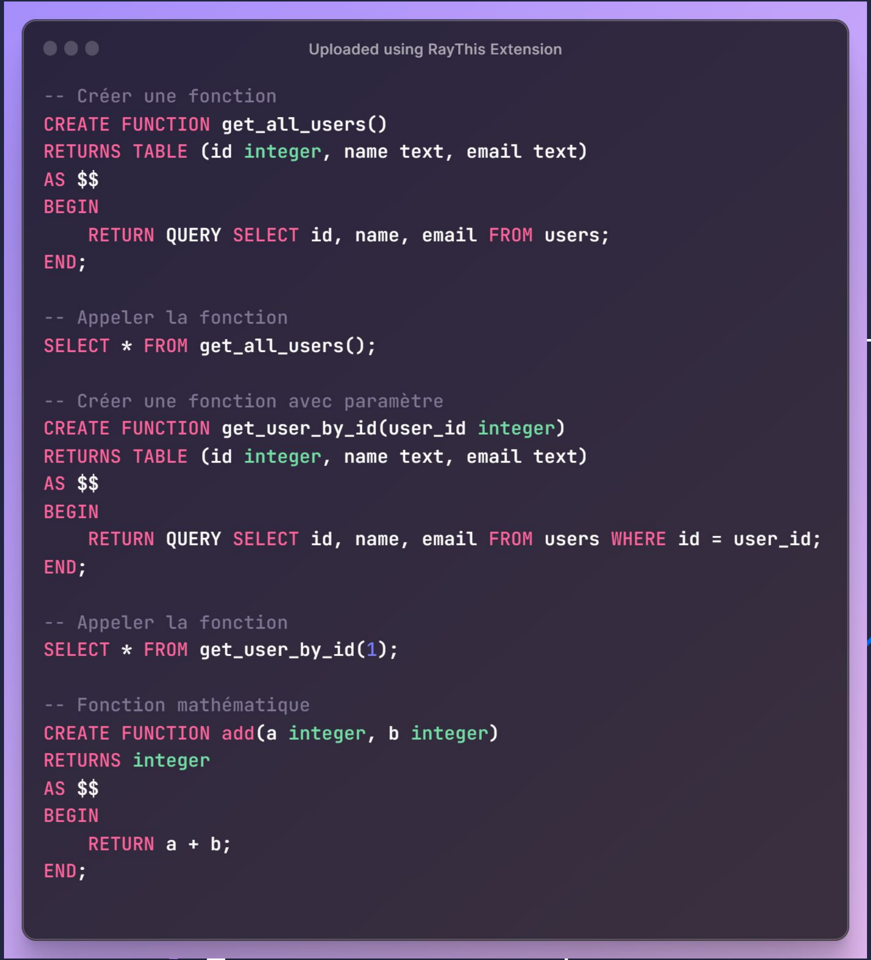
Fonctions avancées



Uploaded using RayThis Extension

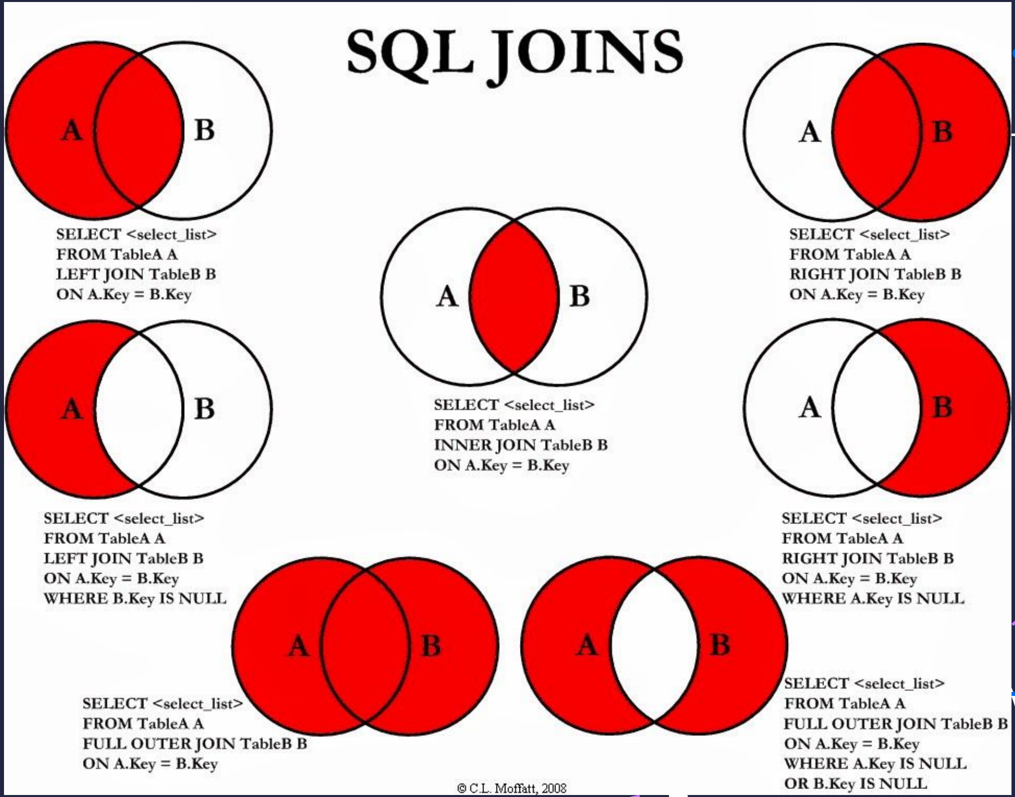
```sql
--- Cryptographique
INSERT INTO `users` (`id`, `username`, `password`) VALUES
(1, 'admin', MD5("123soleil"));

-- Vérification
SELECT * FROM `users` WHERE `username` = 'admin' AND `password` = MD5("123soleil");
```

Fonctions custom



```sql
-- Créer une fonction
CREATE FUNCTION get_all_users()
RETURNS TABLE (id integer, name text, email text)
AS $$
BEGIN
    RETURN QUERY SELECT id, name, email FROM users;
END;

-- Appeler la fonction
SELECT * FROM get_all_users();

-- Créer une fonction avec paramètre
CREATE FUNCTION get_user_by_id(user_id integer)
RETURNS TABLE (id integer, name text, email text)
AS $$
BEGIN
    RETURN QUERY SELECT id, name, email FROM users WHERE id = user_id;
END;

-- Appeler la fonction
SELECT * FROM get_user_by_id(1);

-- Fonction mathématique
CREATE FUNCTION add(a integer, b integer)
RETURNS integer
AS $$
BEGIN
    RETURN a + b;
END;
```

Les jointures en détail



```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

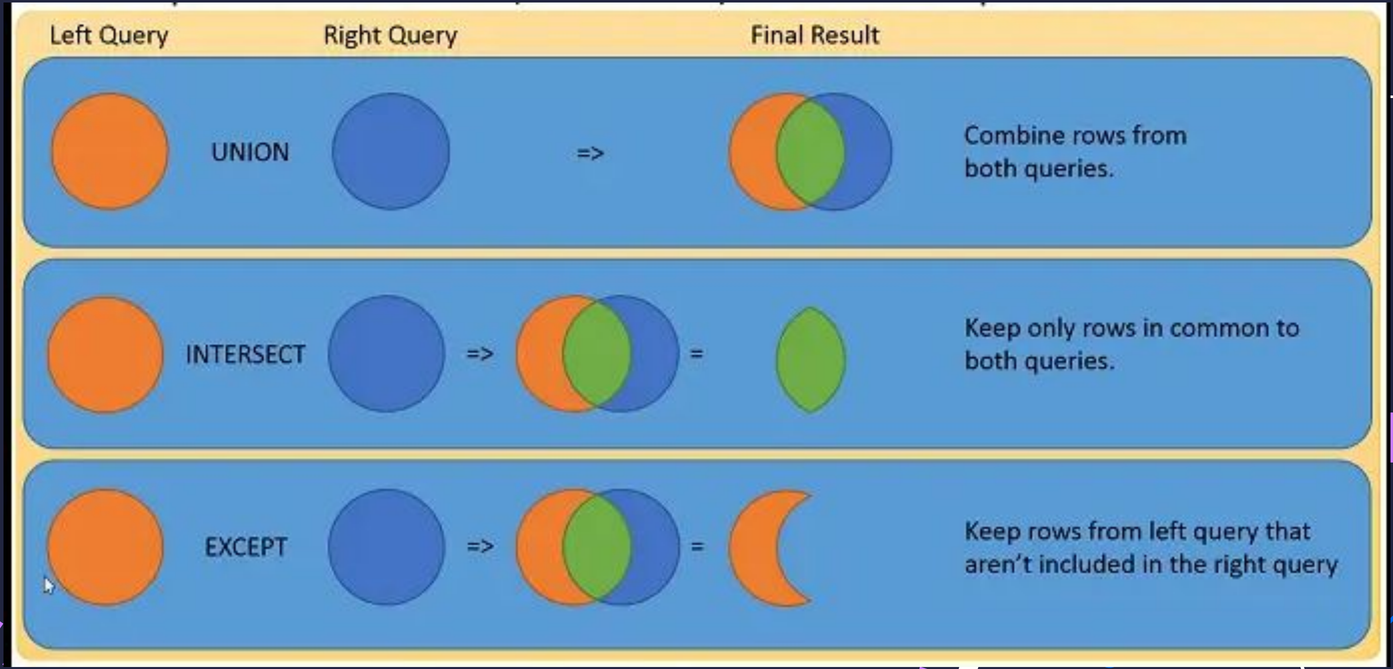
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Les jointures en détail

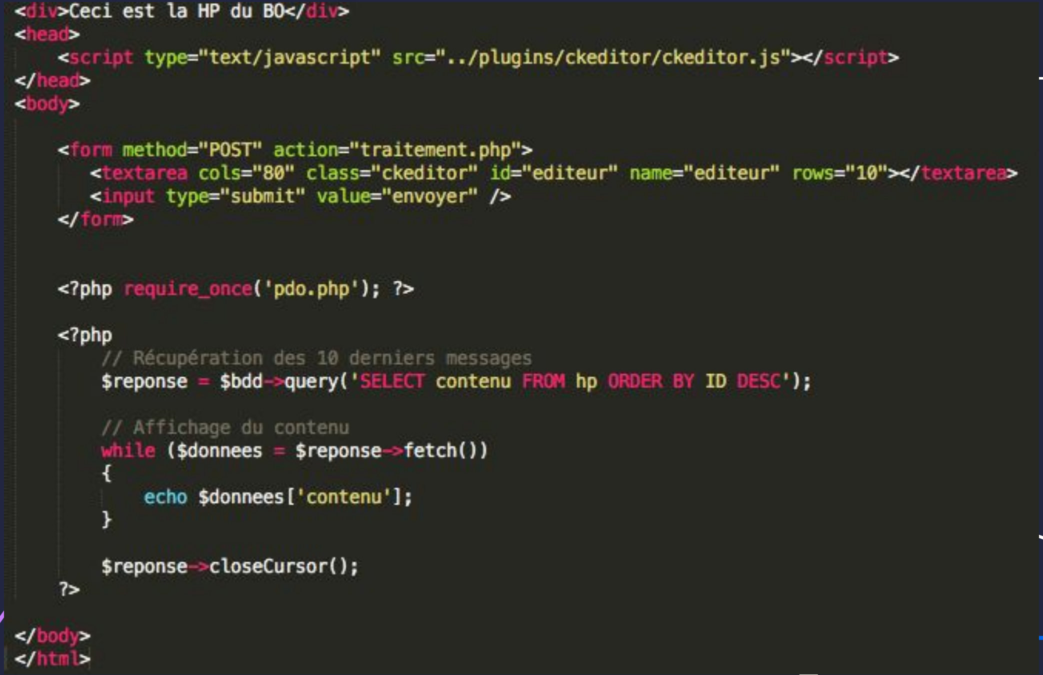


Combine rows from both queries.

Keep only rows in common to both queries.

Keep rows from left query that aren't included in the right query

Connection PDO



```
<div>Ceci est la HP du BO</div>
<head>
    <script type="text/javascript" src="../plugins/ckeditor/ckeditor.js"></script>
</head>
<body>

    <form method="POST" action="traitement.php">
        <textarea cols="80" class="ckeditor" id="editeur" name="editeur" rows="10"></textarea>
        <input type="submit" value="envoyer" />
    </form>


    <?php require_once('pdo.php'); ?>

    <?php
        // Récupération des 10 derniers messages
        $reponse = $bdd->query('SELECT contenu FROM hp ORDER BY ID DESC');

        // Affichage du contenu
        while ($donnees = $reponse->fetch())
        {
            echo $donnees['contenu'];
        }

        $reponse->closeCursor();
    ?>

</body>
</html>
```