# LANGAGE de REQUETE MongoDB MQL et AGREGATION MongoDB

**Introduction**

**Comparaison SQL (CREATE, ALTER, SELECT, DROP en SQL) - MongoDB**

**Pipeline d'agrégation MongoDB => Opérations d'agrégation de données courantes en SQL**

## INTRODUCTION

MQL : syntaxe simple pour interroger les documents au sein d'une même collection => collection unique

MQL ne permet pas de traiter les agrégations ou la gestion complexes de documents.

Il faut structurer toutes les opérations complexes en petites opérations via des opérateurs.

Chaque aspect correspond à une étape et tout se déroule en chaîne.

Les fonctions d'agrégations permettent de manipuler les données renvoyées par une requête MongoDB.

Les langages de requête MongoDB sont conçus pour les collections de requêtes uniques.

Le pipeline d'agrégation de MongoDB est construit sous forme d'étapes ou chaque étape opère sur les documents de l'étape précédente.

L'agrégation est utilisée lorsque le traitement de données complexes est requis.

## COMPARAISON SQL (CREATE, ALTER, SELECT, DROP en SQL) – MongoDB

**Terminologie/ Concept SQL    Terminologie/ Concept MongoDB**

| SQL | MongoDB |
|---|---|
| Database | database |
| Table | collection |
| Row | document |
| Column | field |
| Index | index |
| Primary key | Primary key (_id field) |

**Exemples Instructions SQL    //    Instructions MongoDB**

⇨ ***On travaille sur une collection nommée people***

```
{
 _id: ObjectId("509a8fb2f3f4948bd2f983a0"),
 user_id: "abc123",
 age: 55,
 status: 'A'
}
```

| **Création de table SQL** | **Déclaration de Schéma MongoDB**<br>**db.createCollection("people")** |
|---|---|
| <pre>CREATE TABLE people (<br>  Id NOT NULL<br>    AUTO_INCREMENT,<br>  user_id Varchar(30),<br>  age Number,<br>  status char(1),<br>  PRIMARY KEY (id)<br>)</pre> | <pre>db.people.insertOne( {<br>   user_id: "abc123",<br>   age: 55,<br>   status: "A"<br>} )</pre> |

|  |  |
|---|---|
|  | Remarque : avec insertMany()<br>`db.people.insertMany( [`<br>`{ user_id: "abc123", age: 55, status : "A" },`<br>`{ user_id: "ab123", age: 35, status : "B" },`<br>`{ user_id: "abcd123", age: 45, status : "C" }`<br>`] );` |

| Modification de table SQL | Modification MongoDB |
|---|---|
| `ALTER TABLE people`<br>`ADD join_date DATETIME` | Les collections ne décrivent ni n'appliquent la structure de leurs documents, c'est-à-dire qu'il n'y a pas de modification structurelle au niveau de la collection.<br><br>`db.people.updateMany(`<br>`{ },`<br>`{ $set: { join_date: new Date() } }`<br>`)`<br><br>Au niveau du document, updateMany() peut ajouter des champs aux documents existants à l'aide de l'opérateur $set |
| `ALTER TABLE people`<br>`DROP COLUMN join_date` | `db.people.updateMany(`<br>`{ },`<br>`{ $unset: { "join_date": "" } }`<br>`)` |

| Creation INDEX en SQL | Creation index MongoDB |
|---|---|
| CREATE INDEX idx_user_id_asc<br>ON people(user_id)<br><br>CREATE INDEX<br>idx_user_id_asc_age_desc<br>ON people(user_id, age DESC) | db.people.createIndex( { user_id: 1 } )<br><br>db.people.createIndex( { user_id: 1, age: -1 } ) |

| Suppression en SQL | Suppression MongoDB |
|---|---|
| DROP TABLE people | db.people.drop() |

| Insérer en SQL | Insérer MongoDB |
|---|---|
| INSERT INTO people(user_id,<br>age,<br>status)<br>VALUES ("bcd001",<br>45,<br>"A") ; | db.people.insertOne(<br>{ user_id: "bcd001", age: 45, status: "A" }<br>)<br><br>Remarque :<br>db.people.insertMany( [<br>{ user_id: "bcd001",  age: 45, status: "A" },<br>{ user_id: "xxx",  age: 46, status: "B" },<br>{ _ user_id: "yyy",  age: 47, status: "A" }<br>] ); |

| **Sélectionner en SQL** | **Sélectionner MongoDB** |
|---|---|
| SELECT * FROM people | db.people.find() |
| SELECT id, user_id, status FROM people | db.people.find( { }, { user_id: 1, status: 1 } ) |
| SELECT user_id, status FROM people | db.people.find( { }, { user_id: 1, status: 1, _id: 0 } ) |
| SELECT * FROM people WHERE status = "A" | db.people.find( { status: "A" } ) |
| SELECT user_id, status FROM people WHERE status = "A" | db.people.find( { status: "A" }, { user_id: 1, status: 1, _id: 0 } ) |
| SELECT * FROM people WHERE status != "A" | db.people.find( { status: { $ne: "A" } } ) |
| SELECT * FROM people WHERE status = "A" AND age = 50 | db.people.find( { status: "A", age: 50 } ) |
| SELECT * FROM people WHERE status = "A" | db.people.find( { $or: [ { status: "A" } , { age: 50 } ] } |

| | |
|---|---|
| OR age = 50 | ) |
| SELECT *<br>FROM people<br>WHERE age > 25 | db.people.find(<br>{ age: { $gt: 25 } }<br>) |
| SELECT *<br>FROM people<br>WHERE age < 25 | db.people.find(<br>{ age: { $lt: 25 } }<br>) |
| SELECT *<br>FROM people<br>WHERE age > 25<br>AND   age <= 50 | db.people.find(<br>{ age: { $gt: 25, $lte: 50 } }<br>) |
| SELECT *<br>FROM people<br>WHERE user_id like "bc%" | db.people.find( { user_id: /^bc/ } ) |
| SELECT *<br>FROM people<br>WHERE status = "A"<br>ORDER BY user_id ASC | db.people.find( { status: "A" } ).sort(<br>{ user_id: 1 } ) |
| SELECT *<br>FROM people<br>WHERE status = "A"<br>ORDER BY user_id DESC | db.people.find( { status: "A" } ).sort( {<br>user_id: -1 } ) |
| SELECT COUNT(*)<br>FROM people | db.people.count()<br>db.people.find().count() |

| | |
|---|---|
| `SELECT COUNT(user_id)`<br>`FROM people` | `db.people.count( { user_id:`<br>`{ $exists: true } } )` |
| `SELECT COUNT(*)`<br>`FROM people`<br>`WHERE age > 30` | `db.people.count( { age: { $gt: 30 } } )` |
| `SELECT DISTINCT(status)`<br>`FROM people` | `db.people.aggregate( [ { $group : { _id`<br>`: "$status" } } ] )`<br>`db.people.distinct( "status" )` |
| `SELECT *`<br>`FROM people`<br>`LIMIT 1` | `db.people.findOne()`<br><br>`db.people.find().limit(1)` |

| Mise à jour en SQL | Mise à jour MongoDB |
|---|---|
| UPDATE people<br>SET status = "C"<br>WHERE age > 25 | db.people.updateMany(<br>{ age: { $gt: 25 } },<br>{ $set: { status: "C" } } |
| UPDATE people<br>SET age = age + 3<br>WHERE status = "A" | db.people.updateMany(<br>{ status: "A" } ,<br>{ $inc: { age: 3 } }<br>) |

| Supprimer les enregistrements en SQL | Supprimer les enregistrements en MongoDB |
|---|---|
| DELETE FROM people<br>WHERE status = "D" | db.people.deleteMany( { status: "D" } ) |
| DELETE FROM people | db.people.deleteMany({}) |

**PIPELINE D'AGREGATION MongoDB => OPERATION D'AGREGATION DE DONNEES COURANTES EN SQL**

| | |
|---|---|
| WHERE | $match |
| GROUP BY | $group |
| HAVING | $match |
| SELECT | $project |
| ORDER BY | $sort |
| LIMIT | $limit |
| SUM() | $sum |
| COUNT() | $sum $sortByCount |

⇨ ***On travaille sur une collection nommée orders***

```
{
 cust_id: "abc123",
 ord_date: ISODate("2012-11-02T17:04:11.102Z"),
 status: 'A',
 price: 50,
 items: [
        { sku: "xxx", qty: 25, price: 1 },
        { sku: "yyy", qty: 25, price: 1 }
       ]
}
```

| **Instructions d'agrégation SQL** | **Instructions MongoDB** |
|---|---|
| `SELECT COUNT(*) AS count`<br>`FROM orders` | `db.orders.aggregate( [`<br>`  {`<br>`    $group: {`<br>`      _id: null,` |

| | |
|---|---|
| | count: { $sum: 1 }<br>}<br>}<br>] ) |
| SELECT SUM(price) AS total<br>FROM orders | db.orders.aggregate( [<br>{<br>$group: {<br>_id: null,<br>total: { $sum: **"$price"** }<br>}<br>}<br>] )<br><br>On compte tous les enregistrements de orders |
| SELECT cust_id,<br>SUM(price) AS total<br>FROM orders<br>GROUP BY cust_id | db.orders.aggregate( [<br>{<br>$group: {<br>_id: **"$cust_id"**,<br>total: { $sum: **"$price"** }<br>}<br>}<br>] )<br>Pour chaque cust_id on fait la somme sur les colonnes |
| SELECT cust_id,<br>SUM(price) AS total<br>FROM orders<br>GROUP BY cust_id<br>ORDER BY total | db.orders.aggregate( [<br>{<br>$group: {<br>_id: **"$cust_id"**,<br>total: { $sum: **"$price"** }<br>}<br>},<br>{ $sort: { total: 1 } }<br>] ) |

Pour chaque cust_id unique, on additionne le champ de prix, les résultats sont triés par somme.

```sql
SELECT cust_id,
       SUM(price) as total
FROM orders
WHERE status = 'A'
GROUP BY cust_id
```

```
db.orders.aggregate( [
   { $match: { status: 'A' } },
   {
     $group: {
        _id: "$cust_id",
        total: { $sum: "$price" }
     }
   }
] )
```

Pour chaque cust_id unique avec le statut A, on additionne le champ de prix.

```sql
SELECT cust_id,
       SUM(price) as total
FROM orders
WHERE status = 'A'
GROUP BY cust_id
HAVING total > 250
```

```
db.orders.aggregate( [
   { $match: { status: 'A' } },
   {
     $group: {
        _id: "$cust_id",
        total: { $sum: "$price" }
     }
   },
   { $match: { total: { $gt: 250 } } }
] )
```

Pour chaque cust_id unique avec le statut A, on additionne le champ de prix et on doit renvoyer uniquement la somme supérieure à 250.