MongoDB

Cheat Sheet — technique + fondements (GL3)

A4 • Landscape | 4 colonnes | Print-friendly

0) FONDEMENTS

Modèle document

- **Base → collections → documents → champs**
- Document = structure **JSON-like** (stockée en **BSON** côté disque)
- **Schéma flexible** : documents souvent similaires, pas obligatoires

Terminologie SQL vs MongoDB

- Database = **database**
- Table = **collection**
- Row = **document**
- Column = **field (champ)**
- Join = **embedding (imbrication)** ou **référence**
- Primary key = **_id** (unique)

Champ `_id`

- Champ **obligatoire** (clé primaire) dans chaque collection
- Plusieurs types possibles (souvent `ObjectId`)

Limites & règles

- Taille max d'un document : **16MB** (au-delà : **GridFS**)
- Noms de champs : éviter `.` et `\0`, ne pas commencer par `$`

1) DÉMARRAGE RAPIDE

Connexion

- `mongosh` (shell)

Bases

```
show dbs
use mydb
show collections
```

Inspection rapide

```
db.users.findOne()
db.users.find().limit(5)
```

2) CRUD

2.1 Create (insert)

Si tu ne fournis pas `_id`, MongoDB le génère.

```
db.users.insertOne({
  name: "sue",
  age: 26,
  status: "A",
  groups: ["news","sports"]
})
```

```
db.users.insertMany([
  { name: "ali", age: 18, status:
  { name: "bob", age: 31, status:
])
```

2.2 Read (find / findOne)

```
db.collection.find(<filter>, <proj
// filter: critères (conditions)
// projection: champs à inclure(1)
```

Exemples

```
db.inventory.fi
db.inventory.fi
db.inventory.fi
db.inventory.fi
```

Projection

```
// Inclure cham
db.users.find({

// Exclure _id
db.users.find(
  { status: "A"
  { item: 1, st
)
```

Règle : ne pas mélanger inclusion/exclusion (sauf `_id`).

Tri / Limit / Skip

```
db.users.find({ age: { $gt: 18 } }
db.users.find({}).sort({ createdAt
db.users.find({}).skip(20).limit(1
```

Comptage

```
db.users.countDocuments({ status:
```

2.3 UPDATE

Syntaxe générale

```
db.collection.updateOne(filter, up
db.collection.updateMany(filter, u
db.collection.replaceOne(filter, r
```

$set

```
db.users.updateOne(
  { name: "sue" },
  { $set: { age: 27, city: "Tunis"
)
```

$unset

```
db.users.updateOne(
  { name: "sue" },
  { $unset: { city: "" } }
)
```

$inc

```
db.users.updateOne(
  { name: "sue" },
  { $inc: { points: 5 } }
)
```

Arrays

```
// sans doublon
db.users.updateOne(
  { name: "sue" },
  { $addToSet: { groups: "tech" }
)

// avec doublon possible
db.users.updateOne(
  { name: "sue" },
  { $push: { groups: "tech" } }
)

// enlever une valeur
db.users.updateOne(
  { name: "sue" },
  { $pull: { groups: "sports" } }
)
```

Upsert

```
db.users.updateOne(
  { email: "a@b.com" },
  { $set: { name: "Ali", status:
  { upsert: true }
)
```

Par défaut `updateOne` modifie 1 document, `updateMany` plusieurs.

2.4 DELETE

Supprimer un doc

```
db.users.deleteOne({ name: "sue"
```

Supprimer plusieurs

```
db.users.deleteMany({ status: "B"
```

Danger : tout supprimer

```
db.users.deleteMany({})
```

3) QUERY OPERATORS

Comparaison

```
{ age: { $eq: 1
{ age: { $ne: 1
{ age: { $gt: 1
{ age: { $gte:
{ age: { $lt: 1
{ age: { $lte:
```

Ensembles

```
{ status: { $in
{ status: { $ni
```

Logique

```
{ $and: [ { status: "A" }, { age:
{ $or:  [ { status: "A" }, { statu
{ $not: { age: { $gt: 18 } } }
```

Note :
`{ status:"A", age:{ $gt:18 } }`
équivaut souvent à un AND.

Existence / type

```
{ field: { $exi
{ field: { $typ
```

Texte (si index texte)

```
{ $text: { $sea
```

4) IMBRICATION & ARRAYS

Dot notation

```
// doc: { name: { first:"Alan", la
db.people.find({ "name.last": "Tur
```

Array contient une valeur

```
db.users.find({ groups: "sports" }
```

$elemMatch

```
db.orders.find({
  items: { $elemMatch: { qty: { $g
})
```

5) AGRÉGATION

Une agrégation = pipeline :
`[{ $stage1 }, { $stage2 }, ...]`

Exemple : match + group

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

Stages ultra-utiles

- `$match` : filtre (comme WHERE)
- `$group` : group by + `$sum/$avg/$min/$max/$push`
- `$project` : calculer/choisir champs
- `$sort` : trier
- `$limit` / `$skip`
- `$unwind` : éclater un tableau
- `$lookup` : "join" (si références)

6) MODÉLISATION

Références (normalisé)

- Un doc contient l'`_id` d'un autre doc
- ✅ many-to-many, gros volumes, éviter duplication
- ❌ lecture peut nécessiter plusieurs requêtes (ou `$lookup`)

Imbrication (dénormalisé)

- Sous-documents dans le même document
- ✅ lecture/écriture plus simple et rapide
- ❌ duplication possible, doc peut grossir

Règles pratiques

- **Imbriquer** quand relation **contains** (one-to-one) ou one-to-many "toujours avec le parent"
- **Référencer** quand imbrication = duplication sans gain, many-to-many, gros ensembles hiérarchiques

7) ATOMICITÉ

- Écritures **atomiques au niveau d'un document**
- Pas de garantie d'atomicité multi-documents/collections en une seule opération
- L'imbrication aide quand tu veux modifier des données liées "d'un coup"

8) INDEX

Créer / voir

```
db.users.createIndex({ email: 1 }
db.users.getIndexes()
```

Index composés

```
db.users.createIndex({ status: 1,
```

Index = reads plus rapides, coût en writes + mémoire/disque.

9) RÉPLICATION (REPLICA SET)

- **Replica set** = groupe de `mongod` pour haute dispo
- **Primaire** : reçoit écritures
- **Secondaires** : répliquent via **oplog**
- **Arbitre** : pas de données, vote (utile si nombre pair)
- Si primaire tombe : **élection**

10) SHARDING

Pourquoi

- Scalabilité : répartir données et charge sur plusieurs serveurs

Composants

- **Shards** : stockent données (souvent en replica sets)
- **mongos** : routeur de requêtes
- **Config servers** : métadonnées (chunks → shards)

Shard key

- Partitionnement au niveau collection via une **shard key**
- Mongo découpe en **chunks** et distribue

Équilibrage

- **Splitting** : découpe chunks trop gros (sans migrer)
- **Balancing** : migre chunks pour équilibrer

11) PIÈGES CLASSIQUES

- Faute de frappe sur champ → Mongo renvoie souvent **0 résultat** (pas d'erreur)
- Éviter documents qui grossissent sans contrôle (arrays "infinies")
- Choisir imbrication vs référence selon lectures/écritures

12) MINI MÉMO SQL ↔ MONGO

- `SELECT * FROM col` → `db.col.find({})`
- `WHERE x = 5` → `db.col.find({ x: 5 })`
- `WHERE x IN (...)` → `{ x: { $in: [...] } }`
- `ORDER BY x DESC` → `.sort({ x: -1 })`
- `LIMIT 10` → `.limit(10)`
- `GROUP BY k SUM(v)` → `aggregate([{ $group: ... }])`

13) TEMPLATES (COPIER-COLLER)

Read : filtre + projection + tri

```
db.COL.find(
  { status: "A", age: { $gte: 18 }
  { name: 1, age: 1, status: 1, _
).sort({ age: 1 }).limit(20)
```

Update safe

```
db.COL.updateOne(
  { _id: ObjectId("...") },
  { $set: { field: "new" } }
)
```

Delete safe

```
db.COL.deleteOne({ _id: ObjectId(
```

14) CHECKLIST EXAM / PROJET

- Expliquer : **document / collection / BSON / _id**
- Imbrication vs référence + quand choisir
- CRUD + opérateurs `$gt/$in/$set/$unset/$inc...`
- Agrégation : `$match` + `$group`
- Atomicité au niveau document
- Replica set + sharding (**mongos/config/chunks**)