Partie 1 & 2 — Base, commandes & configuration

1. Qu'est-ce qu'un Replica Set dans MongoDB ?
Un *Replica Set* est un groupe de serveurs MongoDB qui contiennent la même base de données copiée automatiquement (1 Primary + plusieurs Secondaries) pour la haute disponibilité.

2. Rôle du Primary
Le *Primary* est le seul nœud qui accepte les écritures. Il enregistre les opérations dans son *oplog* et les Secondaries les recopient.

3. Rôle des Secondaries
Les *Secondaries* recopient en continu les données du Primary. Ils servent de secours en cas de panne du Primary et peuvent aussi servir aux lectures (selon la readPreference).

4. Pourquoi pas d'écritures sur un Secondary ?
Pour éviter les conflits et les incohérences : si plusieurs nœuds écrivaient en même temps, il faudrait gérer des conflits de versions. MongoDB simplifie : **un seul nœud écrit** (Primary).

5. Cohérence forte dans MongoDB
Cohérence forte = quand on lit depuis le Primary, on voit toujours les dernières écritures validées (acknowledged).
En pratique : lecture sur le Primary + writeConcern et readConcern adaptés (ex : majority).

6. Différence readPreference: "primary" et "secondary"

- "primary" : toutes les lectures vont au Primary → données à jour.
- "secondary" : les lectures vont aux Secondaries → possible retard → données potentiellement obsolètes.

7. Pourquoi lire sur un Secondary malgré les risques ?
Pour **décharger le Primary** et répartir la charge de lecture (rapports, statistiques, analytics) où un léger retard est acceptable.

8. Commande pour initialiser un Replica Set
Dans le shell mongo :

```
rs.initiate()
```

9. Ajouter un nœud après initialisation
Depuis le Primary :

```
rs.add("hostname:port")
```

10. Afficher l'état actuel du Replica Set

rs.status()

11. Identifier le rôle d'un nœud (Primary / Secondary / Arbiter)
Depuis le shell du nœud :

```
rs.status()
// ou
db.hello()   // (ou db.isMaster() sur anciennes versions)
```

On regarde le champ stateStr ou les infos retournées (PRIMARY, SECONDARY, ARBITER).

12. Forcer le basculement du Primary
Sur le Primary :

rs.stepDown()

Il renonce à son rôle et un autre nœud est élu Primary.

Partie 3 & 4 — Résilience, pannes, scénarios

13. Comment désigner un nœud comme Arbitre ? Pourquoi ?
Depuis le Primary :

rs.addArb("hostname:port")

L'Arbitre vote dans les élections mais ne stocke pas de données. Il sert à avoir un **nombre de votes impair** sans coûter un vrai serveur de données.

14. Configurer un Secondary avec slaveDelay
On modifie la config du Replica Set :

```
cfg = rs.conf()
cfg.members[1].slaveDelay = 120  // en secondes
rs.reconfig(cfg)
```

(le Secondary concerné a l'index 1 dans members).

15. Si le Primary tombe en panne et qu'il n'y a pas de majorité ?
Aucun nouveau Primary ne peut être élu. Le Replica Set passe en mode **lecture seule** (pas d'écritures possibles).

16. Comment MongoDB choisit un nouveau Primary ?
Lors d'une élection, les nœuds votent. Les critères principaux :

- la **priorité** (priority) la plus élevée,
- la **fraîcheur des données** (oplog le plus à jour),
- la capacité à contacter une majorité de nœuds.

17. Qu'est-ce qu'une élection ?
C'est le **processus automatique de vote** entre les membres du Replica Set pour désigner le nouveau Primary quand l'ancien n'est plus disponible.

18. Auto-dégradation du Replica Set
Un Primary qui **perd la majorité** (ne voit plus assez de nœuds) se dégrade automatiquement en Secondary pour éviter qu'il y ait deux Primary en parallèle (*split-brain*).

19. Pourquoi un nombre impair de nœuds ?
Pour **faciliter le calcul de majorité** et éviter les égalités lors d'un vote (2/3, 3/5, etc.).

20. Effet d'une partition réseau sur le cluster
Le cluster est coupé en deux groupes :

- Le groupe qui garde la majorité peut élire/maintenir un Primary.
- L'autre groupe perd le Primary ou se met en Secondary → pas d'écritures.
  On peut avoir des bascules fréquentes et du retard de réplication.

21. Scénario : 27017 (Primary), 27018 (Secondary), 27019 (Arbitre)
Si 27017 devient injoignable, 27018 + 27019 = 2 votes sur 3 → majorité.
Une élection a lieu et **27018 devient le nouveau Primary**.

22. Secondary avec slaveDelay = 120s : utilité ?
Ce Secondary a toujours 2 minutes de retard.
Usages :

- revenir en arrière après une erreur humaine (suppression, mise à jour massive),

- analyses sur un état “ancien” de la base.
  C’est une sorte de **“time machine”**.

23. Client veut une lecture toujours à jour, même en cas de bascule
Recommandation :

- readConcern: "majority" (lire des données confirmées par la majorité),
- writeConcern: { w: "majority" } (écriture confirmée par plusieurs nœuds).

24. Garantir que l’écriture est confirmée par au moins deux nœuds
Utiliser :

{ writeConcern: { w: 2 } }

(ou w: "majority" si au moins 3 nœuds).

25. Étudiant a lu une donnée obsolète sur un Secondary : pourquoi ? comment éviter ?
Parce que la réplication est **asynchrone** : le Secondary avait du retard par rapport au Primary.
Pour éviter :

- lire sur le Primary (readPreference: "primary"),
- ou utiliser readConcern: "majority" avec writeConcern: "majority".

26. Commande pour voir quel nœud est Primary
Depuis un membre du Replica Set :

rs.status()

On regarde le membre avec stateStr: "PRIMARY".
(on peut aussi utiliser db.hello() / rs.isMaster()).

27. Forcer une bascule manuelle du Primary sans grosse interruption

1. Vérifier que les Secondaries sont à jour.

Sur le Primary :

rs.stepDown()

2.
3. Un Secondary (souvent le plus à jour et/ou avec plus de priorité) devient Primary.
   Les clients doivent être configurés pour se reconnecter automatiquement.

28. Ajouter un nouveau Secondary dans un Replica Set en fonctionnement

1. Installer et démarrer mongod avec le même nom de Replica Set.

Depuis le Primary :

rs.add("hostname:port")

2.
3. Le nouveau nœud fait une **initial sync**, puis devient Secondary.

29. Retirer un nœud défectueux du Replica Set
Depuis le Primary :

rs.remove("hostname:port")

30. Configurer un nœud Secondary caché (hidden)
Dans la config :

```
cfg = rs.conf()
cfg.members[i].hidden = true
cfg.members[i].priority = 0
rs.reconfig(cfg)
```

On fait ça pour avoir un nœud dédié à des tâches spécifiques (backup, analytique) sans qu'il soit utilisé par les clients.

31. Modifier la priorité pour rendre un nœud Primary préféré

```
cfg = rs.conf()
cfg.members[i].priority = 2   // plus que les autres
rs.reconfig(cfg)
```

Un nœud avec une **priority plus haute** est favorisé lors des élections.

32. Vérifier le délai de réplication d'un Secondary
Utiliser :

rs.printSlaveReplicationInfo()

On voit le retard de chaque Secondary par rapport au Primary.