Fiche de révision MongoDB - Requêtes avec pymongo

📌 Connexion à MongoDB

```
import pymongo
import pandas

connex = pymongo.MongoClient("mongodb://127.0.0.1:27017/")
db = connex.nom_base  # ex: db = connex.resto
```

🔍 REQUÊTES SIMPLES (find, count, distinct)

Compter des documents

```
db.collection.count_documents({})  # Tous
db.collection.count_documents({"champ": "valeur"})  # Avec filtre
db.collection.count_documents({"champ.sous_champ": "valeur"})  # Nested
```

Lister des valeurs uniques

```
db.collection.distinct("champ")
db.collection.distinct("champ.sous_champ")  # Pour nested
db.collection.distinct("tableau.champ")  # Dans un tableau
```

Recherche simple

```
# Afficher avec pandas
pandas.DataFrame(db.collection.find({}))

# Projection (sélectionner colonnes)
db.collection.find({}, {"nom": 1, "adresse": 1, "_id": 0})

# Recherche avec find_one
db.collection.find_one({"_id": "12345"}, {"_id": 0})
```

🎯 FILTRES & OPÉRATEURS

Opérateurs de comparaison

```
{"champ": {"$gt": 50}}   # > 50
{"champ": {"$gte": 50}}  # >= 50
{"champ": {"$lt": 50}}   # < 50
{"champ": {"$lte": 50}}  # <= 50
{"champ": {"$ne": 0}}    # != 0
{"champ": {"$eq": "A"}}  # == "A"
```

Opérateurs logiques

```
# $in – appartient à la liste
{"champ": {"$in": ["val1", "val2"]}}

# $and (implicite ou explicite)
{"champ1": "val1", "champ2": "val2"}  # AND implicite
{"$and": [{"champ1": "val1"}, {"champ2": "val2"}]}  # AND explicite

# Entre deux valeurs
{"champ": {"$gte": -74.2, "$lte": -74.1}}
```

Recherche dans tableaux

```
# Élément présent dans tableau
{"tableau": "valeur"}

# Plusieurs éléments présents
{"$and": [{"tableau": "val1"}, {"tableau": "val2"}]}

# Taille du tableau
{"tableau": {"$size": 2}}

# Existence d'un champ
{"champ": {"$exists": True}}
```

Regex (recherche de texte)

```
{"champ": {"$regex": "Union"}}  # Contient "Union"
```

Dates

```
from datetime import datetime
{"champ_date": datetime(2014, 2, 1)}
```

📊 TRI & LIMITE

```
# Tri
db.collection.find({}, sort=[("champ", 1)])   # Croissant
db.collection.find({}, sort=[("champ", -1)])  # Décroissant

# Tri multiple
sort=[("champ1", 1), ("champ2", -1)]

# Limite
.limit(10)
```

🔥 AGRÉGATIONS (aggregate)

Structure de base

```
db.collection.aggregate([
    {"$stage1": {...}},
    {"$stage2": {...}},
    {"$stage3": {...}}
])
```

$group - Regrouper

```
# Compter
{"$group": {"_id": "$champ", "count": {"$sum": 1}}}

# Moyenne
{"$group": {"_id": "$champ", "moyenne": {"$avg": "$nombre"}}}

# Min/Max
{"$group": {"_id": "$champ", "min": {"$min": "$val"}, "max": {"$max":
"$val"}}}

# Grouper sans clé
{"$group": {"_id": None, "total": {"$sum": 1}}}

# Récupérer premier/dernier
{"$group": {"_id": "$id", "premier": {"$first": "$champ"}}}

# Liste unique (set)
{"$group": {"_id": "$rue", "quartiers": {"$addToSet": "$quartier"}}}

# Liste (push)
{"$group": {"_id": "$quartier", "cuisines": {"$push": "$cuisine"}}}
```

$sort - Trier

```
{"$sort": {"champ": -1}}  # -1 décroissant, 1 croissant
```

$limit - Limiter

```
{"$limit": 10}
```

$project - Projeter/Renommer

```
{"$project": {
    "_id": 0,
    "nouveau_nom": "$ancien_nom",
    "champ": 1,  # Garder
    "calcul": {"$round": ["$moyenne", 2]}  # Arrondir
}}
```

$match - Filtrer

```
{"$match": {"champ": {"$gt": 100}}}
{"$match": {"tableau": {"$size": 2}}}
{"$match": {"tableau": {"$eq": ["A"]}}}
```

$unwind - Déplier un tableau

```
# Avant: {_id: 1, notes: [10, 15, 20]}
{"$unwind": "$notes"}
# Après: 3 documents: {_id: 1, notes: 10}, {_id: 1, notes: 15}, {_id: 1,
notes: 20}
```

$sortByCount - Compter et trier (raccourci)

```
# Équivalent à $group + $sort
{"$sortByCount": "$champ"}

# Équivalent à:
{"$group": {"_id": "$champ", "count": {"$sum": 1}}},
{"$sort": {"count": -1}}
```

Fonctions de dates

```
{"$project": {"dayOfWeek": {"$dayOfWeek": "$date_field"}}}
# 1 = Dimanche, 2 = Lundi, ..., 7 = Samedi
```

$slice - Limiter un tableau dans projection

```
{"$project": {"top3": {"$slice": ["$tableau", 3]}}}
```

🎓 PATTERNS COURANTS

TOP N

```
db.collection.aggregate([
    {"$group": {"_id": "$champ", "count": {"$sum": 1}}},
    {"$sort": {"count": -1}},
    {"$limit": 10}
])
```

Moyenne par groupe (avec $unwind)

```
db.collection.aggregate([
    {"$unwind": "$notes"},
    {"$group": {
        "_id": "$quartier",
        "moyenne": {"$avg": "$notes.score"}
    }}
])
```

Moyenne des moyennes (2 étapes)

```
db.collection.aggregate([
    {"$unwind": "$notes"},
    # Étape 1: Moyenne par document
    {"$group": {
        "_id": "$_id",
        "quartier": {"$first": "$quartier"},
        "moy_doc": {"$avg": "$notes.score"}
    }},
    # Étape 2: Moyenne par quartier
    {"$group": {
        "_id": "$quartier",
```

```
        "nb": {"$sum": 1},
        "moy_finale": {"$avg": "$moy_doc"}
    }}
])
```

Filtrer sur taille de tableau groupé

```
db.collection.aggregate([
    {"$group": {"_id": "$rue", "quartiers": {"$addToSet": "$quartier"}}},
    {"$match": {"quartiers": {"$size": 2}}}
])
```

Top N par catégorie

```
db.collection.aggregate([
    {"$group": {"_id": {"cat": "$categorie", "item": "$item"}, "count":
{"$sum": 1}}},
    {"$sort": {"_id.cat": 1, "count": -1}},
    {"$group": {
        "_id": "$_id.cat",
        "top": {"$push": {"item": "$_id.item", "nb": "$count"}}
    }},
    {"$project": {"top": {"$slice": ["$top", 3]}}}
])
```

📈 VISUALISATION avec Seaborn

🔧 Préparation des données

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Convertir résultat aggregate en DataFrame
result = list(db.collection.aggregate([...]))
df = pd.DataFrame(result)

# Nettoyer et convertir types
df["colonne"] = df["colonne"].astype(str).str.replace(",", ".")
df["colonne"] = pd.to_numeric(df["colonne"], errors="coerce")

# Renommer colonnes depuis _id
df = df.rename(columns={"_id": "categorie"})

# Gérer valeurs manquantes
```

```
df = df.dropna()  # Supprimer lignes avec NaN
df = df.fillna(0)  # Remplacer NaN par 0
```

📊 Barplot (Diagramme en barres)

```
# Barplot simple
plt.figure(figsize=(12, 6))
sns.barplot(data=df, x="categorie", y="valeur", palette="viridis")
plt.title("Titre du graphique", fontsize=16)
plt.xlabel("Catégorie", fontsize=12)
plt.ylabel("Valeur", fontsize=12)
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

# Barplot horizontal (nombreuses catégories)
plt.figure(figsize=(10, 8))
sns.barplot(data=df, x="valeur", y="categorie", palette="Blues_r")
plt.xlabel("Nombre")
plt.tight_layout()
plt.show()

# Barplot avec hue (sous-catégories)
sns.barplot(data=df, x="quartier", y="count", hue="cuisine",
palette="Set2")
```

📈 Lineplot (Courbe)

```
# Évolution temporelle
plt.figure(figsize=(12, 6))
sns.lineplot(data=df, x="date", y="valeur", marker="o")
plt.title("Évolution dans le temps")
plt.xticks(rotation=45)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# Plusieurs lignes
sns.lineplot(data=df, x="mois", y="ventes", hue="categorie", marker="o")
```

🎯 Scatterplot (Nuage de points)

```
# Points simples
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x="note", y="count", s=100)
plt.title("Relation entre variables")
```

```
plt.show()

# Avec taille et couleur
sns.scatterplot(data=df, x="x", y="y", size="count", hue="categorie",
                sizes=(50, 500), alpha=0.6, palette="viridis")
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
```

🥧 Pieplot (Camembert)

```
# Matplotlib pour pie chart
plt.figure(figsize=(10, 8))
plt.pie(df["count"], labels=df["categorie"], autopct="%1.1f%%",
        startangle=90, colors=sns.color_palette("pastel"))
plt.title("Répartition")
plt.axis("equal")
plt.show()

# Top N + Autres
df_top = df.nlargest(10, "count")
autres = pd.DataFrame([{"categorie": "Autres",
                        "count": df.iloc[10:]["count"].sum()}])
df_pie = pd.concat([df_top, autres])
```

📦 Boxplot (Boîte à moustaches)

```
# Distribution par catégorie
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x="quartier", y="score", palette="Set3")
plt.xticks(rotation=45)
plt.title("Distribution des scores")
plt.tight_layout()
plt.show()
```

🎻 Violinplot

```
plt.figure(figsize=(12, 6))
sns.violinplot(data=df, x="categorie", y="valeur", palette="muted")
plt.xticks(rotation=45)
plt.show()
```

📊 Countplot (Comptage automatique)

```
# Compte automatiquement les occurrences
plt.figure(figsize=(10, 6))
```

```
sns.countplot(data=df, x="categorie", palette="rocket")
plt.xticks(rotation=45)
plt.show()
```

🔥 Heatmap (Carte de chaleur)

```
# Créer pivot table d'abord
pivot = df.pivot_table(values="count", index="quartier",
                       columns="cuisine", fill_value=0)

plt.figure(figsize=(12, 8))
sns.heatmap(pivot, annot=True, fmt=".0f", cmap="YlOrRd",
            linewidths=0.5, cbar_kws={"label": "Nombre"})
plt.title("Heatmap")
plt.tight_layout()
plt.show()

# Matrice de corrélation
corr = df[["col1", "col2", "col3"]].corr()
sns.heatmap(corr, annot=True, cmap="coolwarm", center=0,
            vmin=-1, vmax=1, square=True)
```

📉 Histogramme & Distribution

```
# Histogramme
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x="score", bins=20, kde=True)
plt.title("Distribution des scores")
plt.show()

# Distribution par catégorie
sns.histplot(data=df, x="score", hue="quartier", multiple="stack")
```

🎨 Palettes de couleurs

```
# Palettes séquentielles (ordre)
"Blues", "Greens", "Reds", "YlOrRd", "viridis", "plasma"

# Palettes divergentes (autour d'un centre)
"coolwarm", "RdBu", "RdYlGn"

# Palettes qualitatives (catégories)
"Set1", "Set2", "Set3", "Paired", "tab10", "husl"

# Inverser palette
"Blues_r"
```

```
# Créer palette custom
sns.set_palette(sns.color_palette("husl", 8))
```

⚙️ Configuration globale

```
# Style
sns.set_style("whitegrid")  # whitegrid, darkgrid, white, dark, ticks
sns.set_context("notebook")  # paper, notebook, talk, poster

# Taille de police
sns.set(font_scale=1.2)

# Réinitialiser
sns.reset_defaults()

# Figure size par défaut
plt.rcParams["figure.figsize"] = (12, 6)
```

🎯 Subplots (Plusieurs graphiques)

```
# 2x2 grid
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

sns.barplot(data=df, x="a", y="b", ax=axes[0, 0])
axes[0, 0].set_title("Graphique 1")

sns.lineplot(data=df, x="c", y="d", ax=axes[0, 1])
axes[0, 1].set_title("Graphique 2")

sns.boxplot(data=df, x="e", y="f", ax=axes[1, 0])
sns.scatterplot(data=df, x="g", y="h", ax=axes[1, 1])

plt.tight_layout()
plt.show()
```

💾 Sauvegarder graphiques

```
# Sauvegarder
plt.savefig("graphique.png", dpi=300, bbox_inches="tight")
plt.savefig("graphique.pdf")  # Format vectoriel
plt.savefig("graphique.svg")

# Après avoir créé le plot
fig = plt.gcf()  # Get current figure
fig.savefig("output.png", dpi=300, bbox_inches="tight")
```

🔧 Astuces formatage

```
# Rotation labels
plt.xticks(rotation=45, ha="right")
plt.yticks(rotation=0)

# Limites axes
plt.xlim(0, 100)
plt.ylim(0, 50)

# Grille
plt.grid(True, alpha=0.3, linestyle="--")

# Légende personnalisée
plt.legend(title="Catégorie", loc="upper right",
           bbox_to_anchor=(1.15, 1))

# Retirer légende
plt.legend([])

# Format nombres axes
from matplotlib.ticker import FuncFormatter
ax = plt.gca()
ax.yaxis.set_major_formatter(FuncFormatter(lambda x, p: f"{int(x):,}"))

# Tight layout (éviter coupures)
plt.tight_layout()

# Titre avec sous-titre
plt.suptitle("Titre principal", fontsize=16, y=1.02)
plt.title("Sous-titre", fontsize=12)
```

📋 Workflow complet

```
# 1. Requête MongoDB
pipeline = [
    {"$group": {"_id": "$categorie", "total": {"$sum": 1}}},
    {"$sort": {"total": -1}},
    {"$limit": 10}
]
result = list(db.collection.aggregate(pipeline))

# 2. Créer DataFrame
df = pd.DataFrame(result)
df = df.rename(columns={"_id": "categorie", "total": "nombre"})

# 3. Nettoyer données
df["nombre"] = pd.to_numeric(df["nombre"], errors="coerce")
df = df.dropna()
```

```
# 4. Créer graphique
plt.figure(figsize=(12, 6))
sns.barplot(data=df, x="categorie", y="nombre", palette="viridis")
plt.title("Top 10 des catégories", fontsize=16, pad=20)
plt.xlabel("Catégorie", fontsize=12)
plt.ylabel("Nombre", fontsize=12)
plt.xticks(rotation=45, ha="right")
plt.tight_layout()

# 5. Sauvegarder
plt.savefig("top_categories.png", dpi=300, bbox_inches="tight")
plt.show()
```
