Benagoub Omar - SWD - Verifica Finale NoSQL

Esercizio – Gestione di un Servizio di Consegna Pasti

Modello documentale e implementazione di un database per un'applicazione di food delivery. Il progetto include clienti, piatti e ordini, con esempi pratici, query e giustificazioni delle scelte progettuali.

- *Struttura del DataBase (Collections e Campi)*

DISHES :

-_id: ObjectId
- name: String
- description: String
- price: Number
- category: String
- preparationTime: Number (opzionale)
- calories: Number (opzionale)

CUSTOMERS :

-_id: ObjectId
- firstName: String
- lastName: String
- email: String (univoco)
- registrationDate: Date
- addresses: [Embedded]
    - street
    - city
    - zipCode
    - unit (opzionale)

ORDERS :

-_id: ObjectId
- customerId: ObjectId (referenza a customers)
- createdAt: Date
- status: String
- items: [Embedded]
    - dishId: ObjectId (referenza a dishes)
    - quantity
    - priceSnapshot

- *Scelte Progettuali (Embedding vs Reference)*

- **Gli indirizzi** sono **__embedded__** nei clienti, perché sono pochi e servono spesso assieme ai dati cliente.
- **Gli items** dell'ordine sono **__embedded__** nell'ordine per performance e snapshot del prezzo.
- **I riferimenti** a piatti e clienti sono **__reference__** per evitare duplicazioni e mantenere i dati aggiornati.

- *QUERIES CON SPIEGAZIONI*

Q1: Trova tutti i piatti con prezzo > 15€

Q1: Trova tutti i piatti con prezzo > 15€
```js
db.dishes.find({ price: { $gt: 15 } })

$gt = greater than
```

Q2: Elenca gli ordini effettuati da un determinato cliente (dato il suo `_id` o `email`)

con find: db.orders.find({ customerId: ObjectId("66a2e2b4c45ff6a2a86512a1") })
con email:
const customer = db.customers.findOne({ email: "maria.rossi@gmail.com" })
db.orders.find({ customerId: customer._id })

spiegazione: Cerca **nella collezione `customers`** un **cliente con quella email**.

Restituisce **un solo documento** (grazie a `findOne()`).

Lo **salva nella variabile `customer`**.

Cerca **nella collezione `orders`** tutti gli ordini dove il campo `customerId` è **uguale all'`_id` del cliente trovato** prima.

Q3: Aggiorna lo stato di un ordine (dato l'ID) a "completed"

```
db.orders.updateOne(
  { _id: ObjectId("66a2e3c0f47aa4829a1b8ccf") },
  { $set: { status: "completed" } }
)
```

$set = Aggiorna lo status da "in preparation" a "completed"

Q4: Trova i clienti che hanno effettuato almeno 2 ordini

```
db.orders.aggregate([
  { $group: { _id: "$customerId", orderCount: { $sum: 1 } } },
  { $match: { orderCount: { $gte: 2 } } },
  {
    $lookup: {
      from: "customers",
      localField: "_id",
      foreignField: "_id",
      as: "customerInfo"
    }
  },
  { $unwind: "$customerInfo" },
  {
    $project: {
      _id: 0,
      customerId: "$_id",
      firstName: "$customerInfo.firstName",
      lastName: "$customerInfo.lastName",
      email: "$customerInfo.email",
      orderCount: 1
    }
  }
])
```

Operatori usati: `$group`, `$match`, `$lookup`, `$unwind`, `$project`

Spiegazione: Raggruppa per cliente, filtra chi ha ≥ 2 ordini, restituisce nome e email.

$group = Raggruppa tutti gli ordini per `customerId`

$match = Filtra i gruppi trovati sopra
$lookup = Fa il join con la collection “customers”
$unwind = Estrae l’oggetto da “customerInfo[]” e lo trasforma da array a oggetto singolo
$project = Definisce quali campi mostrare nel risultato finale

Q5: Trova il piatto più ordinato nel mese di giugno 2025

```
db.orders.aggregate([
  {
    $match: {
      createdAt: {
        $gte: ISODate("2025-06-01T00:00:00Z"),
        $lte: ISODate("2025-06-30T23:59:59Z")
      }
    }
  },
  { $unwind: "$items" },
  {
    $group: {
      _id: "$items.dishId",
      totalOrdered: { $sum: "$items.quantity" }
    }
  },
  { $sort: { totalOrdered: -1 } },
  { $limit: 1 },
  {
    $lookup: {
      from: "dishes",
      localField: "_id",
      foreignField: "_id",
      as: "dishInfo"
    }
  },
  { $unwind: "$dishInfo" },
  {
    $project: {
      _id: 0,
      dishId: "$_id",
      dishName: "$dishInfo.name",
      totalOrdered: 1
    }
  }
])
```

$match: Filtra i documenti in base a una condizione. In questo caso seleziona solo gli ordini compresi tra due date.

$unwind: Divide un array in documenti singoli, uno per ogni elemento dell'array. Serve per analizzare ogni piatto ordinato separatamente.

$group: Raggruppa i documenti in base a un campo comune. Qui raggruppa per dishId per sommare le quantità.

$sum: Somma i valori di un campo all'interno di un gruppo. Viene usato per calcolare quante volte è stato ordinato ciascun piatto.

$sort: Ordina i risultati secondo un criterio. Qui ordina in ordine decrescente per quantità totale ordinata.

$limit: Limita il numero di risultati restituiti. In questa query prende solo il piatto più ordinato.

$lookup: Unisce dati da un'altra collezione. Qui collega il dishId con i dettagli del piatto nella collezione dishes.

$project: Seleziona quali campi mostrare nel risultato finale e può rinominarli. Serve per mostrare solo dishId, nome del piatto e quantità totale.

Fonti:
Durante la realizzazione del progetto ho utilizzato, in modo mirato e consapevole, il supporto di un Large Language Model, **ChatGPT di OpenAI (modello GPT-4)**. Questo strumento è stato impiegato come assistente per validare alcune scelte progettuali, verificare la correttezza della sintassi delle query MongoDB e migliorare la chiarezza espositiva della documentazione. L'interazione con l'LLM ha rappresentato anche un'opportunità per consolidare conoscenze già acquisite e approfondirne di nuove, soprattutto in ambito di modellazione documentale e aggregazioni avanzate. Ogni contenuto generato è stato attentamente revisionato, compreso e adattato personalmente, con l'obiettivo di garantire un lavoro originale, solido e pienamente coerente con gli obiettivi didattici.