LAPORAN RELATIONAL DAN NON-RELATIONAL DATABASE

1. Relational Database (MySQL) dengan Docker

a. Pertama kali kita akan melakukan pull mysql dockernya terlebih dahulu

```
PS C:\Users\hp> docker pull mysql:latest
latest: Pulling from library/mysql
Digest: sha256:c69299937e5e2fc9a2cb26f5cd7a7151e48d9d5a3b3679f62bfd1275de698c0c
Status: Image is up to date for mysql:latest
docker.io/library/mysql:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview mysql:latest
PS C:\Users\hp> docker pull mongo:latest
latest: Pulling from library/mongo
Digest: sha256:1ade6afda762cb6a68ba65e83ef305660ef0517d6d4140627211970e85f4588a
Status: Image is up to date for mongo:latest
docker.io/library/mongo:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview mongo:latest
PS C:\Users\hp>
```

b. Selanjutnya kita akan Membuat container, password, dan Databasenya dengan syntax dibawah ini

```
PS C:\Users\hp> docker run --name roni-mysql-relational -e MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=students_db -p
 3306:3306 -d mysql:latest
2c239294248db95407b1e1b4e7a7e379ee58fd800a3171c9ce9b674f5ab1a912
```

c. Selanjutnya kita dapat menjalankan container yang sudah kita buat, ketika kita menjalankan syntax dibawah ini maka kita akan dimintai password yang sudah kita buat sebelumnya pada bagian MYSQL_ROOT_PASSWORD=password

```
PS C:\Users\hp> docker exec -it roni-mysql-relational mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 9.0.1 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

d. Selanjutnya dikarenakan kita sudah membuat databasenya dengan MYSQL_DATABASE=students_db kita dapat langsung menggunakan database tersebut lalu membuat table databasenya

```
mysql> USE students_db;
Database changed
mysql> CREATE TABLE Students (
    -> Student_ID INT AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(100),
    -> Grade VARCHAR(10)
    -> );
Query OK, 0 rows affected (0.10 sec)
```

e. Jika sudah membuat tablenya kita dapat memasukkan data sesuai table database yang sudah kita buat

s

```
mysql> INSERT INTO Students (Name, Grade) VALUES ('Alice', 'A'), ('Bob', 'B'), ('Charlie', 'C');
Query OK, 3 rows affected (0.05 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

f. Jika sudah kita pastikan bahwa data berhasil masuk

```
mysql> SELECT * FROM Students;
+------------+---------+-------+
| Student_ID | Name    | Grade |
+------------+---------+-------+
|          1 | Alice   | A     |
|          2 | Bob     | B     |
|          3 | Charlie | C     |
+------------+---------+-------+
3 rows in set (0.00 sec)

mysql>
```

2. Non-Relational Database (MongDB) dengan Docker

a. Sama Halnnya dengan Relational kita juga akan melakukan pull untuk mongodbnya

```
PS C:\Users\hp> docker pull mongo:latest
latest: Pulling from library/mongo
857cc8cb19c0: Download complete
da4a4cbb623f: Download complete
a54f12bd5819: Download complete
f95b02a6236d: Download complete
0d20d29fe9ca: Download complete
2382733f40de: Download complete
c1458145b657: Download complete
fee77be41765: Download complete
Digest: sha256:1ade6afda762cb6a68ba65e83ef305660ef0517d6d4140627211970e85f4588a
Status: Downloaded newer image for mongo:latest
docker.io/library/mongo:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview mongo:latest
```

b. Lalu kita dapat membuat container mongonya (Disini saya tidak mengikuti syntax bapak karena error ketikas saya mencobanya, sehingga saya menggunakan alternative lain)

```
PS C:\Users\hp> docker run -d -p 27017:27017 -v ~/MongoDb:/data/db --name roni-mongo-nosql mongo:latest
0f655670dafbdc03da96ea30370e45ab432998faee8f4ac635b860a62b6d21b9
```

c. Jikas sudah kita uji running container mongo yang sudah kita buat

```
PS C:\Users\hp> docker exec -it roni-mongo-nosql bash
root@0f655670dafb:/# mongosh
Current Mongosh Log ID: 66e8697f703d56d7d55e739b
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2
.3.0
Using MongoDB:          7.0.14
Using Mongosh:          2.3.0

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/


To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.co
m/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

------
   The server generated these startup warnings when booting
   2024-09-16T17:21:58.654+00:00: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
   2024-09-16T17:21:58.654+00:00: /sys/kernel/mm/transparent_hugepage/enabled is 'always'. We suggest setting it to 'nev
er' in this binary version
   2024-09-16T17:21:58.655+00:00: vm.max_map_count is too low
------

test>
```

d. Jika sudah kita dapat mulai membuat databasenya terlebih dahulu dan memasukkan data yang kita inginkan

```
test> use students_db;
switched to db students_db
students_db> db.Students.insertMany([
... {Name: "Alice", Grade: "A"},
... {Name: "Bob", Grade: "B"},
... {Name: "Charlie", Grade: "C"}
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('66e86abf703d56d7d55e739c'),
    '1': ObjectId('66e86abf703d56d7d55e739d'),
    '2': ObjectId('66e86abf703d56d7d55e739e')
  }
}
```

e. Setelah kita dapat melihat database yang sudah kita buat

```
students_db> db.Students.find().pretty();
[
  {
    _id: ObjectId('66e86abf703d56d7d55e739c'),
    Name: 'Alice',
    Grade: 'A'
  },
  {
    _id: ObjectId('66e86abf703d56d7d55e739d'),
    Name: 'Bob',
    Grade: 'B'
  },
  {
    _id: ObjectId('66e86abf703d56d7d55e739e'),
    Name: 'Charlie',
    Grade: 'C'
  }
]
students_db>
```

TASK FOR Hands-on Practicum

1. Tasks for Relational Database (MySQL)

a. Disini kita akan membuat table baru dengan nama courses

```
mysql> CREATE TABLE Courses (
    -> Course_ID INT AUTO_INCREMENT PRIMARY KEY,
    -> Course_Name VARCHAR(100) NOT NULL,
    -> Credits INT NOT NULL
    -> );
Query OK, 0 rows affected (0.05 sec)
```

b. Selanjutnya kita akan memodifikasi tabel Students yang sudah ada untuk berhubungan degnan tabel Course dengan menambahkan foreign key

```
mysql> INSERt INTO Courses (Course_Name, Credits) VALUES
    -> ('Keamanan Basis Data', 4),
    -> ('PBO', 4),
    -> ('Interkoneksi Jaringan', 4);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM Courses;
+-----------+----------------------+---------+
| Course_ID | Course_Name          | Credits |
+-----------+----------------------+---------+
|         1 | Keamanan Basis Data  |       4 |
|         2 | PBO                  |       4 |
|         3 | Interkoneksi Jaringan |       4 |
+-----------+----------------------+---------+
3 rows in set (0.00 sec)

mysql>
```

```
mysql> ALTER TABLE Students Add Course_ID INT;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE Students
    -> ADD CONSTRAINT FK_Course
    -> FOREIGN KEY (Course_ID)
    -> REFERENCES Courses(Course_ID);
Query OK, 3 rows affected (0.22 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql>
```

```
mysql> UPDATE Students SET Course_ID = 1 WHERE Name = 'Alice';
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE Students SET Course_ID = 2 WHERE Name = 'Bob';
Query OK, 1 row affected (0.90 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> SELECT Students.Name, Courses.Course_Name
    -> FROM Students
    -> JOIN Courses ON Students.Course_ID = Courses.Course_ID;
+-------+---------------------+
| Name  | Course_Name         |
+-------+---------------------+
| Alice | Keamanan Basis Data |
| Bob   | PBO                 |
+-------+---------------------+
2 rows in set (0.00 sec)

mysql>
```

2. Perform Advanced Queries

a. Menghitung jumlah siswa di setiap mata kuliah

```
mysql> SELECT Courses.Course_Name, COUNT(Students.Student_ID) AS NumberOfStudents
    -> FROM Courses
    -> LEFT JOIN Students ON Courses.Course_ID = Students.Course_ID
    -> GROUP BY Courses.Course_Name;
+----------------------+-----------------+
| Course_Name          | NumberOfStudents |
+----------------------+-----------------+
| Keamanan Basis Data  |               1 |
| PBO                  |               1 |
| Interkoneksi Jaringan |               0 |
+----------------------+-----------------+
3 rows in set (0.01 sec)
```

b. menemukan semua siswa yang memiliki nilai 'B’

```
mysql> SELECT Name
    -> FROM Students
    -> WHERE Grade = 'B';
+------+
| Name |
+------+
| Bob  |
+------+
1 row in set (0.00 sec)

mysql>
```

c. mengambil semua siswa yang tidak terdaftar dalam kursus apa pun

```
mysql> SELECT Name
    -> FROM Students
    -> WHERE Course_ID IS NULL;
+---------+
| Name    |
+---------+
| Charlie |
+---------+
1 row in set (0.00 sec)

mysql>
```

3. Undetstanding ACID Properties

a. Dalam suatu transaksi, atomitas memastikan bahwa semua operasi dilakukan sepenuhnya; jika tidak, transaksi dibatalkan. Perintah seperti START TRANSACTION, COMMIT, dan ROLLBACK digunakan untuk menjamin atomisitas di MySQL. Jika ada kesalahan dalam transaksi, setiap perubahan yang dilakukan selama transaksi akan dibatalkan, sehingga kondisi database tetap konsisten.

b. Konsistensi memastikan bahwa database ditransfer dari satu keadaan ke keadaan lain dengan mempertahankan semua aturan yang telah ditentukan, termasuk batasan. MySQL menjaga konsistensi dengan menetapkan batasan (seperti kunci asing dan batasan unik) dan pemicu yang memeriksa validitas data sebelum melakukan komitmen perubahan. Isolasi memastikan bahwa transaksi dieksekusi secara independen satu sama lain. MySQL mempertahankan isolasi melalui tingkat isolasi transaksi (misalnya, READ COMMITTED, REPEATABLE READ, SERIALIZABLE). Tingkat ini mengontrol bagaimana dan kapan perubahan yang dilakukan oleh satu transaksi menjadi terlihat oleh transaksi lainnya.

c. Daya tahan memastikan bahwa data tetap ada di database bahkan jika terjadi kegagalan sistem setelah transaksi dikomit. MySQL menggunakan file data dan log transaksi untuk memastikan daya tahan. Setelah transaksi dikomit, perubahan tersebut dicatat dalam log transaksi, dan proses pemulihan MySQL memastikan bahwa perubahan tersebut diterapkan pada database saat database dimulai ulang.

2. Task for Non-Relational Database (MongoDB)

1. Add a New Field to existing Documents

```
students_db> db.Students.updateMany(
...   {},  // Kondisi filter kosong berarti semua dokumen akan di-update
...   { $set: { Age: 20 } }  // Menambahkan field Age dengan nilai 20
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
students_db> db.Students.find(
...   {},  // Tidak ada filter, menampilkan semua dokumen
...   { Name: 1, Age: 1 }  // Hanya menampilkan field Name dan Age
... ).pretty();

students_db>
```

2. Perform Aggregation ini MongoDb mendapatkan

```
test> use students_db;
switched to db students_db
students_db> db.Students.aggregate([
...   {
...     $group: {
...       _id: "$Grade",            // Mengelompokkan berdasarkan nilai Grade
...       studentCount: { $sum: 1 }  // Menghitung jumlah siswa dalam setiap Grade
...     }
...   }
... ]);

students_db>

students_db> db.Students.aggregate([
...   {
...     $group: {
...       _id: "$Grade",            // Mengelompokkan berdasarkan nilai Grade
...       studentCount: { $sum: 1 }  // Menghitung jumlah siswa dalam setiap Grade
...     }
...   }
... ]);

students_db>

students_db>
```

3. Understanding the CAP Theorem

1. Bagaimana MongoDB Memprioritaskan Ketersediaan dan Toleransi Partisi Daripada Konsistensi Ketat? MongoDB memprioritaskan ketersediaan dan toleransi partisi daripada konsistensi ketat dalam situasi partisi jaringan (ketika komunikasi antara node terputus). Ini menunjukkan bahwa MongoDB terus berusaha memastikan bahwa layanan dapat diakses (Availability) dan sistem tetap beroperasi meskipun partisi jaringan terjadi, meskipun sistem mungkin tidak dapat menjamin konsistensi data di semua node secara bersamaan.

Availability (Ketersediaan): MongoDB memastikan bahwa node yang masih tersedia akan menerima dan memproses permintaan baca dan tulis saat terjadi kegagalan atau partisi.

Toleransi Partisi (Toleransi Partisi): MongoDB dibuat untuk menangani situasi di mana jaringan terpecah. Meskipun data mungkin tidak sepenuhnya konsisten selama partisi, sistem tetap dapat menyesuaikan diri jika ada perubahan.

2. MongoDB menggunakan arsitektur set replika dengan satu node inti dan beberapa node sekunder. Jika ada partisi antara node utama dan node sekunder, node utama yang tersisa tetap dapat melakukan operasi tulis dan baca, tetapi node sekunder tidak akan dapat berkomunikasi dengan node utama. MongoDB akan menjalankan proses pemilihan di antara node tambahan untuk memilih node utama baru jika node utama gagal. Data akan direplikasi dari node utama baru ke node lain setelah jaringan dipulihkan.Namun, ketika jaringan dipartisi, ada kemungkinan bahwa data yang disimpan di satu bagian sistem tidak dapat diakses segera di bagian lain. Ini menunjukkan konsistensi lemah.
3. Eventual Consistency: Replika data akan dikirim ke node tambahan setelah data ditulis di node utama. Jumlah waktu yang diperlukan untuk mencapai konsistensi penuh bervariasi, termasuk kondisi jaringan dan beban sistem.

   Pengguna dapat menyesuaikan tingkat konsistensi dalam MongoDB melalui konfigurasi menulis masalah dan membaca preferensi: Menulis masalah: mengontrol seberapa banyak node yang harus mengakui menulis sebelum dianggap berhasil. Pengguna dapat memilih untuk menerima penulisan dari node utama saja atau menunggu hingga semua replika node memperbarui data. Read Preference: Mengontrol dari mana data akan dibaca. Misalnya, pengguna dapat memilih untuk membaca dari node utama saja (menjamin konsistensi) atau membiarkan pembacaan dari node sekunder (yang mungkin belum sepenuhnya sinkron).

4. **Comparative Tasks: Relational Vs Non-Relational Databases**

   a. **Scheme Flexibility**

      1. Kapan Skema Tetap Lebih Dibutuhkan?
         jika Anda memiliki data yang sangat terorganisir dan aturan validasi yang ketat, skema tetap seperti MySQL akan sangat membantu. Ini termasuk:
         Aplikasi perbankan: Agar data transaksi, pelanggan, dan akun aman, format data harus sangat teratur dan konsisten.
         Sistem ERP (Enterprise Resource Planning): Sistem yang kompleks yang mengelola data perusahaan secara terpusat membutuhkan konsistensi tinggi dan keyakinan bahwa struktur data tidak berubah secara acak.
      2. Kapan Skema Fleksibel Berhasil?
         Ketika struktur data sering berubah atau tidak diketahui sejak awal, skema fleksibel MongoDB lebih baik. Ini sangat menguntungkan dalam:
         Aplikasi Media Sosial: Ini adalah aplikasi di mana data pengguna, postingan, dan konten lainnya dapat sangat beragam dan tidak memiliki pola yang konsisten.
         Prototyping Cepat: Skema yang fleksibel memudahkan perubahan model data tanpa migrasi yang kompleks pada tahap awal pengembangan aplikasi.

   b. **Query Comparison**

      MYSQL

```
mysql> SELECT * FROM Students WHERE Grade = 'A';
+------------+-------+-------+-----------+
| Student_ID | Name  | Grade | Course_ID |
+------------+-------+-------+-----------+
|          1 | Alice | A     |         1 |
+------------+-------+-------+-----------+
1 row in set (0.00 sec)
```

      MONGO

```
students_db> db.Students.find({ Grade: "A" }).pretty();

students_db> db.Students.find({ Age: { $gt: 21 } }).pretty();

students_db>
```

   c. **Performance Considerations**

      1. Penanganan Banyak Data Tidak Terstruktur: MongoDB lebih baik menangani banyak data tidak terstruktur daripada MySQL. Ada sejumlah alasan untuk ini:
      Struktur Data Fleksibel: Model dokumen MongoDB memungkinkan struktur data yang lebih fleksibel. Koleksi dokumen dapat memiliki berbagai struktur, yang bermanfaat untuk data tidak terstruktur seperti log, konten media, atau data dari aplikasi yang berubah secara dinamis.
      Pengelolaan Data Semi-Terstruktur: MongoDB dapat menangani data semi-terstruktur seperti JSON, yang biasa digunakan dalam aplikasi web kontemporer, dan data yang dihasilkan dari berbagai sumber.
      Skalabilitas Vertikal: MongoDB mendukung sharding, yaitu pembagian data di berbagai server, yang memungkinkan penyimpanan dan pemrosesan data yang sangat besar dengan mudah.

2. Kuery Kompleks dengan Banyak Join atau Relasi MySQL bekerja lebih baik untuk kuery kompleks dengan banyak join atau relasi. Penyebab utamanya adalah:

Skema Tetap dan Relasi: MySQL menggunakan model relasional dengan skema tetap, yang memungkinkan pengaturan dan pemeliharaan integritas referensial melalui kunci asing dan relasi antar tabel. Ini juga memudahkan eksekusi kuery yang melibatkan join beberapa tabel.

Optimisasi Pertanyaan: MySQL memiliki mesin optimisasi pertanyaan yang kuat, sehingga dapat menangani operasi join yang kompleks dengan mudah, terutama ketika data diindeks dengan benar.

Transaksi ACID: MySQL mendukung transaksi yang mengikuti prinsip ACID (Atomitas, Konsistensi, Isolasi, dan Ketahanan), yang penting untuk menjaga integritas data dalam operasi yang melibatkan banyak tabel.

4. Final Reflection Task

Disini saya akan membuat database ecmmorece mysql dan mongo.

MYSQL:

```
mysql> CREATE DATABASE ecommerce_db;
Query OK, 1 row affected (0.02 sec)

mysql> USE ecommerce_db;
Database changed
mysql> CREATE TABLE Users (
    ->     User_ID INT AUTO_INCREMENT PRIMARY KEY,
    ->     Username VARCHAR(50) NOT NULL UNIQUE,
    ->     Password_Hash VARCHAR(255) NOT NULL,
    ->     Email VARCHAR(100) NOT NULL UNIQUE,
    ->     Created_At TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.12 sec)

mysql> CREATE TABLE Orders (
    ->     Order_ID INT AUTO_INCREMENT PRIMARY KEY,
    ->     User_ID INT,
    ->     Order_Date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ->     Status VARCHAR(20),
    ->     Total DECIMAL(10, 2),
    ->     FOREIGN KEY (User_ID) REFERENCES Users(User_ID)
    -> );
Query OK, 0 rows affected (0.14 sec)

mysql>
```

```
mysql> INSERT INTO Users (Username, Password_Hash, Email) VALUES ('alice', 'hash1', 'alice@example.com');
Query OK, 1 row affected (0.04 sec)

mysql> INSERT INTO Users (Username, Password_Hash, Email) VALUES ('bob', 'hash2', 'bob@example.com');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO Products (Product_Name, Description, Price, Stock) VALUES ('Laptop', 'High-performance laptop', 1200.
00, 10);
Query OK, 1 row affected (0.04 sec)

mysql> INSERT INTO Products (Product_Name, Description, Price, Stock) VALUES ('Smartphone', 'Latest model smartphone', 8
00.00, 20);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO Orders (User_ID, Status, Total) VALUES (1, 'Completed', 1200.00);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO Orders (User_ID, Status, Total) VALUES (2, 'Pending', 800.00);
Query OK, 1 row affected (0.01 sec)

mysql>
```

```
mysql> SELECT Orders.Order_ID, Users.Username, Orders.Order_Date, Orders.Status, Orders.Total
    -> FROM Orders
    -> JOIN Users ON Orders.User_ID = Users.User_ID;
+----------+----------+---------------------+-----------+---------+
| Order_ID | Username | Order_Date          | Status    | Total   |
+----------+----------+---------------------+-----------+---------+
|        1 | alice    | 2024-09-16 18:29:02 | Completed | 1200.00 |
|        2 | bob      | 2024-09-16 18:29:08 | Pending   |  800.00 |
+----------+----------+---------------------+-----------+---------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM Products;
+------------+--------------+-------------------------+---------+-------+
| Product_ID | Product_Name | Description             | Price   | Stock |
+------------+--------------+-------------------------+---------+-------+
|          1 | Laptop       | High-performance laptop | 1200.00 |    10 |
|          2 | Smartphone   | Latest model smartphone |  800.00 |    20 |
+------------+--------------+-------------------------+---------+-------+
2 rows in set (0.00 sec)

mysql>
```

MONGO:

```
test> use ecommerce;
switched to db ecommerce
ecommerce> db.Users.insertMany([
...   { User_ID: 1, Username: 'alice', Password_Hash: 'hash1', Email: 'alice@example.com', Created_At: new Date() },
...   { User_ID: 2, Username: 'bob', Password_Hash: 'hash2', Email: 'bob@example.com', Created_At: new Date() }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('66e8794f9b94c4040b5e739c'),
    '1': ObjectId('66e8794f9b94c4040b5e739d')
  }
}
ecommerce> db.Products.insertMany([
...   { Product_ID: 1, Product_Name: 'Laptop', Description: 'High-performance laptop', Price: 1200.00, Stock: 10 },
...   { Product_ID: 2, Product_Name: 'Smartphone', Description: 'Latest model smartphone', Price: 800.00, Stock: 20 }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('66e879609b94c4040b5e739e'),
    '1': ObjectId('66e879609b94c4040b5e739f')
  }
}
ecommerce>
```

```
ecommerce> db.Orders.insertMany([
...   { Order_ID: 1, User_ID: 1, Order_Date: new Date(), Status: 'Completed', Total: 1200.00 },
...   { Order_ID: 2, User_ID: 2, Order_Date: new Date(), Status: 'Pending', Total: 800.00 }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('66e879709b94c4040b5e73a0'),
    '1': ObjectId('66e879709b94c4040b5e73a1')
  }
}
ecommerce>
```

```
ecommerce> db.Orders.aggregate([
...   { $lookup: {
...       from: 'Users',
...       localField: 'User_ID',
...       foreignField: 'User_ID',
...       as: 'User_Info'
...     }
...   },
...   { $unwind: '$User_Info' }
... ]);
[
  {
    _id: ObjectId('66e879709b94c4040b5e73a0'),
    Order_ID: 1,
    User_ID: 1,
    Order_Date: ISODate('2024-09-16T18:31:12.191Z'),
    Status: 'Completed',
    Total: 1200,
    User_Info: {
      _id: ObjectId('66e8794f9b94c4040b5e739c'),
      User_ID: 1,
      Username: 'alice',
      Password_Hash: 'hash1',
      Email: 'alice@example.com',
      Created_At: ISODate('2024-09-16T18:30:39.623Z')
    }
  },
  {
    _id: ObjectId('66e879709b94c4040b5e73a1'),
    Order_ID: 2,
    User_ID: 2,
    Order_Date: ISODate('2024-09-16T18:31:12.191Z'),
    Status: 'Pending',
    Total: 800,
    User_Info: {
      _id: ObjectId('66e8794f9b94c4040b5e739d'),
      User_ID: 2,
      Username: 'bob',
      Password_Hash: 'hash2',
      Email: 'bob@example.com',
      Created_At: ISODate('2024-09-16T18:30:39.623Z')
    }
  }
]
ecommerce>
```

```
ecommerce> db.Products.find().pretty();
[
  {
    _id: ObjectId('66e879609b94c4040b5e739e'),
    Product_ID: 1,
    Product_Name: 'Laptop',
    Description: 'High-performance laptop',
    Price: 1200,
    Stock: 10
  },
  {
    _id: ObjectId('66e879609b94c4040b5e739f'),
    Product_ID: 2,
    Product_Name: 'Smartphone',
    Description: 'Latest model smartphone',
    Price: 800,
    Stock: 20
  }
]
ecommerce>
```