Basic Queries for MERN Stack Developers

**1. CRUD Operations**

**Create**

const newUser = new User({ name: "John Doe", email: "john@example.com" });  
await newUser.save();  
  
- \*\*save()\*\*: Saves a new document to the collection.

**Read**

- \*\*Find All\*\*  
  
const users = await User.find();  
  
- \*\*Find One\*\*  
  
const user = await User.findOne({ email: "john@example.com" });  
  
- \*\*Find by ID\*\*  
  
const user = await User.findById("12345");

**Update**

- \*\*Update One Document\*\*  
  
const updatedUser = await User.updateOne({ \_id: "12345" }, { name: "Jane Doe" });  
  
- \*\*Find and Update\*\*  
  
const user = await User.findByIdAndUpdate("12345", { name: "Jane Doe" }, { new: true });  
  
- \*\*{ new: true }\*\*: Returns the updated document.

**Delete**

- \*\*Delete One Document\*\*  
  
const deletedUser = await User.deleteOne({ \_id: "12345" });  
  
- \*\*Find and Delete\*\*  
  
const user = await User.findByIdAndDelete("12345");

**2. Query Operators**

**Comparison Operators**

- \*\*$gt\*\*, \*\*$gte\*\*: Greater than, greater than or equal to  
- \*\*$lt\*\*, \*\*$lte\*\*: Less than, less than or equal to  
- \*\*$ne\*\*: Not equal  
- \*\*$in\*\*: Matches any value in an array  
- \*\*$nin\*\*: Matches none of the values in an array  
  
const users = await User.find({ age: { $gte: 18 } });

**Logical Operators**

- \*\*$and\*\*: Matches documents that satisfy all the conditions  
- \*\*$or\*\*: Matches documents that satisfy at least one condition  
- \*\*$not\*\*: Inverts the effect of a query expression  
- \*\*$nor\*\*: Matches documents that fail all the conditions  
  
const users = await User.find({ $or: [{ age: { $gte: 18 } }, { name: "John" }] });

**3. Array Operators**

- \*\*$all\*\*: Matches arrays that contain all elements specified in the query  
- \*\*$elemMatch\*\*: Matches documents that contain an array field with at least one element that matches all the specified query criteria  
- \*\*$size\*\*: Matches arrays with the specified number of elements  
  
const conversations = await Convo.find({ members: { $all: [userId1, userId2] } });

**4. Aggregation**

**Basic Example**

const result = await User.aggregate([  
 { $match: { age: { $gte: 18 } } },  
 { $group: { \_id: "$age", total: { $sum: 1 } } }  
]);

**Pipeline Stages**

- \*\*$match\*\*: Filters documents  
- \*\*$group\*\*: Groups documents by a specified field  
- \*\*$sort\*\*: Sorts documents  
- \*\*$limit\*\*: Limits the number of documents returned  
- \*\*$skip\*\*: Skips a specified number of documents  
- \*\*$project\*\*: Reshapes each document in the stream

**5. Pagination**

**Skip and Limit**

const users = await User.find().skip(10).limit(10);

**6. Populating References**

**Populating Related Documents**

const post = await Post.findById(postId).populate('author').exec();

**7. Indexing**

**Creating Indexes**

await User.createIndex({ email: 1 });

**8. Text Search**

**Basic Text Search**

const results = await Post.find({ $text: { $search: "keyword" } });

**9. Distinct**

**Get Distinct Values**

const uniqueAges = await User.distinct("age");

**10. Counting Documents**

**Count All Documents**

const count = await User.countDocuments();

**Count with Condition**

const count = await User.countDocuments({ age: { $gte: 18 } });

**11. Error Handling**

**Handling Errors with Try-Catch**

try {  
 const user = await User.findById(userId);  
} catch (error) {  
 console.error(error);  
}

**12. Transactions**

**Using Transactions for Multiple Operations**

const session = await mongoose.startSession();  
session.startTransaction();  
try {  
 await User.create([{ name: "John" }], { session });  
 await Order.create([{ item: "Laptop" }], { session });  
 await session.commitTransaction();  
} catch (error) {  
 await session.abortTransaction();  
} finally {  
 session.endSession();  
}