MySQL Data Types

1. Numeric Data Types

Used for numbers (integers, decimals, floating-point, etc.).

Integer Types

Type	Storage	Range (Signed)	Range (Unsigned)
TINYINT	1 byte	-128 to 127	0 to 255
SMALLINT	2 bytes	-32,768 to 32,767	0 to 65,535
MEDIUMINT	3 bytes	-8,388,608 to 8,388,607	0 to 16,777,215
INT / INTEGER	4 bytes	-2,147,483,648 to 2,147,483,647	0 to 4,294,967,295
BIGINT	8 bytes	-2^{63} to 2^{63}-1	0 to 2^{64}-1

Example:

```
age TINYINT UNSIGNED, salary INT, population BIGINT
```

2.Fixed-Point Types

Type	Description
DECIMAL(M, D) / NUMERIC(M, D)	Exact numeric values. `M` = total digits, `D` = digits after decimal. Good for money.

Example:

```
price DECIMAL(10, 2)  -- max 10 digits, 2 after decimal
```

Floating-Point Types

Type	Storage	Description
FLOAT(M, D)	4 bytes	Approximate numeric, single precision
DOUBLE(M, D) / REAL	8 bytes	Approximate numeric, double precision

Example:

temperature FLOAT, pi DOUBLE

3. String Data Types

Used for text, characters, and binary data.

Type	Description
CHAR(n)	Fixed-length string (padded with spaces). Up to 255 chars.
VARCHAR(n)	Variable-length string (up to 65,535 depending on row size).
TEXT types	Large text storage:
- TINYTEXT (255)	SMALLTEXT (65,535)
BLOB types	Binary large objects (images, files): TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB
ENUM	One value from a predefined list. Example:

Type	Description
	`'small','medium','large'`
SET	Multiple values from a predefined list.

Example:

```
name VARCHAR(100),
gender ENUM('male','female','other'),
hobbies SET('reading','music','sports')
```

NOTE :

- `CHAR = fixed-size box → if value is smaller, it fills with spaces.`

code CHAR(5)

Insert 'IN' → stored as 'IN ' (3 spaces).

- `VARCHAR = flexible string → only uses space equal to actual characters.`

```
name VARCHAR(5)
```

```
Insert 'IN' → stored as 'IN' (just 2 chars).
```

4. Date & Time Data Types

Used for storing dates and times.

Type	Format	Range
DATE	`YYYY-MM-DD`	1000-01-01 to 9999-12-31
DATETIME	`YYYY-MM-DD HH:MM:SS`	1000-01-01 00:00:00 to 9999-12-31 23:59:59
TIMESTAMP	`YYYY-MM-DD HH:MM:SS`	1970-01-01 UTC to 2038-01-19 UTC
TIME	`HH:MM:SS`	-838:59:59 to 838:59:59
YEAR(2 or 4)	`YYYY`	1901 to 2155

Example:

```
birth_date DATE,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAM
```

MySQL CRUD Operations

Assume we have a table `users` with schema:

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100),
    age INT
);
```

1. INSERT (Create)

```
INSERT INTO users (name, email, age)
VALUES ('Mehul', 'mehul@example.com', 25);
```

2. SELECT (Read)

- Fetch all users:

```
SELECT * FROM users;
```

- Fetch specific user:

```
SELECT * FROM users WHERE id = 1;

SELECT name, email FROM users WHERE id = 1;
```

3. UPDATE

```
UPDATE users
SET age = 26, email = 'mehul.raj@example.com'
WHERE id = 1;
```

4. DELETE

```
DELETE FROM users
WHERE id = 1;
```

MongoDB CRUD Operations

Assume we have a collection `users` with documents like:

```
{
  "_id": ObjectId("..."),
  "name": "Mehul",
  "email": "mehul@example.com",
  "age": 25
}
```

1. INSERT (Create)

- Insert one:

```
db.users.insertOne({ name: "Mehul", email: "mehul@example.com", age: 25 });
```

- Insert many:

```
db.users.insertMany([
  { name: "Riya", email: "riya@example.com", age: 24 },
  { name: "Raj", email: "raj@example.com", age: 28 }
]);
```

2. FIND (Read)

- Fetch all:

```
db.users.find();
```

- Fetch specific user:

```
db.users.findOne({ name: "Mehul" });
```

- With condition:

```
db.users.find({ age: { $gt: 25 } });
```

3. UPDATE

- Update one:

```
db.users.updateOne(
  { name: "Mehul" },
  { $set: { age: 26, email: "mehul.raj@example.com" } }
);
```

- Update many:

```
db.users.updateMany(
  { age: { $lt: 25 } },
  { $set: { status: "young" } }
);
```

4. DELETE

- Delete one:

```
db.users.deleteOne({ name: "Mehul" });
```

- Delete many:

```
db.users.deleteMany({ age: { $gt: 30 } });
```

Key Difference:

- **MySQL** uses **tables, rows, columns** with structured schema.
- **MongoDB** uses **collections, documents** with flexible schema (JSON-like).

Mongoose CRUD Operations

const mongoose = require("mongoose");

const userSchema = new mongoose.Schema({

  name: String,

  email: String,

  age: Number,

});

const User = mongoose.model("User", userSchema);

1. **CREATE (Insert)**

- Insert one:

```
const user = new User({ name: "Mehul", email: "mehul@example.com", age: 25 });
await user.save();
```

- Insert many:

```
await User.insertMany([
  { name: "Riya", email: "riya@example.com", age: 24 },
  { name: "Raj", email: "raj@example.com", age: 28 }
]);
```

2. **READ (Find/Select)**

- Find all:

```
const users = await User.find();
```

- Find one:

```
const user = await User.findOne({ email: "mehul@example.com" });
```

- Find by ID:

```
const user = await User.findById("64fe0b123abc456def789ghi");
```

- Conditional + projection:

```
const users = await User.find({ age: { $gt: 25 } }).select("name email");
```

3. **UPDATE**

- Update one:

```
await User.updateOne(
  { email: "mehul@example.com" },
  { $set: { age: 26 } }
);
```

- Update by ID:

```
await User.findByIdAndUpdate("64fe0b123abc456def789ghi", { age: 27 });
```

- Update many:

```
await User.updateMany({ age: { $lt: 25 } }, { $set: { status: "young" } });
```

4. **DELETE**

- Delete one:

```
await User.deleteOne({ email: "mehul@example.com" });
```

- Delete by ID:

```
await User.findByIdAndDelete("64fe0b123abc456def789ghi");
```

- Delete many:

```
await User.deleteMany({ age: { $gt: 30 } });
```

General CRUD Interview Questions

1. **What does CRUD stand for?**
   Create, Read, Update, Delete – the four basic operations on persistent storage.
2. **Difference between SQL (MySQL) and NoSQL (MongoDB) in terms of CRUD?**

- MySQL → Structured, uses tables/rows, schema required.
- MongoDB → Document-oriented, flexible schema, JSON-like docs.

3. **Which operation is most expensive in databases?**
   Depends on indexes, but usually `UPDATE` and `DELETE` can be more costly than `SELECT`, especially without indexes.

MySQL CRUD Interview Questions

Q. How do you insert multiple records in MySQL at once?

```
INSERT INTO users (name, email, age)
VALUES ('A', 'a@gmail.com', 21), ('B', 'b@gmail.com', 22);
```

Q. **How do you fetch only unique values from a column?**

Use `DISTINCT`:

```
SELECT DISTINCT age FROM users;
```

Q. **How do you update multiple rows in MySQL?**

Using `WHERE`:

```
UPDATE users SET age = age + 1 WHERE age < 25;
```

Q. **What is the difference between `DELETE`, `TRUNCATE`, and `DROP`?**

- `DELETE`: Removes rows, can use `WHERE`, can rollback.
- `TRUNCATE`: Removes all rows, faster, cannot use `WHERE`.
- `DROP`: Deletes the entire table/schema.

Q. **What happens if you don’t use `WHERE` in UPDATE or DELETE?**
It will update/delete *all* rows.

MongoDB CRUD Interview Questions

Q. **How do you insert multiple documents in MongoDB?**
Using `insertMany()`

Q. **What is the difference between `find()`, `findOne()`, and `find().limit(1)`?**

- `findOne()` → Returns the first matching document.
- `find()` → Returns a cursor of multiple docs.
- `find().limit(1)` → Cursor limited to 1 doc, still wrapped in cursor.

Q. **How do you update a document without replacing the whole object?**
Use `$set`:

```
db.users.updateOne({ name: "Mehul" }, { $set: { age: 26 } });
```

Q.**What is the difference between `updateOne` and `updateMany`?**

- `updateOne` → Updates the first matching document.
- `updateMany` → Updates all matching documents.

Q. **What is the difference between `remove()`, `deleteOne()`, and `deleteMany()`?**

- `remove()` → Older method, deprecated.
- `deleteOne()` → Deletes the first match.
- `deleteMany()` → Deletes all matches.

Q. **How do you query documents with age > 25?**

```
db.users.find({ age: { $gt: 25 } });
```

Q. **What happens if you insert a document without `_id` in MongoDB?**
MongoDB automatically generates an `_id` (ObjectId).

Mongoose Interview Questions

Q. **What is Mongoose?**
Mongoose is an ODM (Object Data Modeling) library for MongoDB and Node.js. It provides schema-based modeling, validation, and query building.

Q. **Difference between MongoDB driver and Mongoose?**

- **MongoDB driver** → low-level, flexible, but more manual work.
- **Mongoose** → higher-level, schema-based, has middleware, validation, and helper methods

Q. **How do you define a schema in Mongoose?**
Using `mongoose.Schema({ field: Type })`.

Q. **Difference between `findOne()` and `findById()`?**

- `findOne()` → Query with condition (e.g., `{ email: ... }`).
- `findById()` → Query directly by `_id`.

Q. **Difference between `updateOne()` vs `findByIdAndUpdate()`?**

- `updateOne()` → Updates without returning the document.
- `findByIdAndUpdate()` → Updates and can return the updated document.

Q. **What happens if you use `updateOne()` without `$set`?**
It replaces the entire document with only the fields you provide (dangerous mistake).

Q. **How do you implement soft delete in Mongoose?**
Add a field `isDeleted: Boolean` instead of actually deleting, then filter in queries.

Q. **What are Mongoose middleware (pre and post hooks)?**
Functions that run before/after events (e.g., before saving, before deleting).
Example:

```
userSchema.pre("save", function(next) {

  console.log("Before saving:", this.name);

  next();

});
```

Q. **What is the difference between `lean()` and normal queries in Mongoose?**

- `.lean()` returns **plain JavaScript objects**, faster, but no Mongoose methods.
- Normal queries return **Mongoose documents** (with methods like `.save()`).

Q. **How do you handle validation in Mongoose?**
Define validation in schema (e.g., `required`, `minlength`, `match`), or use `validate()`.

Q. **What happens if you update a document without `$set`?**

Example:

```
await User.updateOne({ _id: id }, { age: 30 });
```

- This **replaces the entire document** with only `{ age: 30 }`.

  Correct way:

```
await User.updateOne({ _id: id }, { $set: { age: 30 } });
```

Q. **Difference between `remove()`, `deleteOne()`, `findOneAndDelete()`, and `findByIdAndDelete()`?**

- `remove()` → Deprecated.
- `deleteOne()` → Deletes first matching doc (no return).
- `findOneAndDelete()` → Deletes and returns the deleted doc.
- `findByIdAndDelete()` → Same but directly by `_id`.

Q. **How do you implement soft delete in Mongoose?**

Instead of deleting docs, add a flag:

```
isDeleted: { type: Boolean, default: false }
```

Then filter queries:

```
User.find({ isDeleted: false });
```

Q. **What is the difference between `save()` and `insertMany()`?**

- `save()` → Document instance method, runs **middleware (pre/post hooks)**, validation.
- `insertMany()` → Bulk insert, **skips middleware by default**, faster.

Q. **Difference between `find()` vs `lean()`?**

- `find()` → Returns full **Mongoose document** with methods (`.save()`, `.validate()`).
- `lean()` → Returns plain JS object → faster, but no methods.

Used for **read-heavy APIs** where you don’t need doc methods.

Q. **What is the difference between `populate()` and `$lookup` in MongoDB aggregation?**

- `populate()` → High-level Mongoose method for joining references.
- `$lookup` → MongoDB aggregation operator, lower-level but more flexible.

Example:

```
User.find().populate("posts");
```

Q. **What happens if you store a field not defined in Schema?**

Depends on `strict` option:

- `strict: true` → Ignores extra fields.
- `strict: false` → Stores them.
- `strict: "throw"` → Throws an error.

Q. **How would you handle unique constraints in Mongoose?**

```
email: { type: String, unique: true }
```

Note: `unique: true` is **not validation**, it just creates an **index**.
You must also handle duplicate key errors.

Q. . **How to ensure atomicity when updating multiple documents?**

👉 Use **Transactions** (with MongoDB sessions):

```
const session = await mongoose.startSession();

session.startTransaction();



await User.updateOne({ _id: id }, { $set: { age: 30 } }, { session
});

await Order.updateOne({ userId: id }, { $set: { status:
"updated" } }, { session });
```

```
await session.commitTransaction();

session.endSession();
```

Q. **How do you validate an email field in Mongoose Schema?**

```
email: {

  type: String,

  required: true,

  match: /.+\@.+\..+/

}
```

Scenario-based Questions

Q. **If you run `UPDATE users SET age = 30;` in MySQL, what happens?**
All rows' age column will become 30.

Q. **In MongoDB, if you forget `$set` in an update, what happens?**
The whole document gets replaced with only the new fields. (Big mistake in real systems.)

Q. **How would you implement soft delete in MySQL/MongoDB?**

- Add a field `isDeleted: true/false` instead of actually deleting the record.

Q. **Which CRUD operation is most optimized in MongoDB?**
Read (`find`) is optimized when you use indexes.

Mongoose

Q. Scenario: You want to return only non-sensitive fields (`name`, `email`) but hide `password`. How do you do it?

Option 1: Use `select` in query:

```
User.find().select("name email");
```

Option 2: Use `toJSON` transformation in schema:

```
userSchema.set("toJSON", {

  transform: (doc, ret) => {

    delete ret.password;
    return ret;

  }

});
```

Q. **Scenario: How would you log every time a document is deleted?**

Use middleware (hooks):

```
userSchema.pre("deleteOne", { document: true, query: false },
function(next) {

  console.log(`User deleted: ${this._id}`);

  next();

});
```

Q. **What's the difference between `SchemaType options` like `default`, `required`, `enum`?**

- `default` → Assigns a value if none provided.
- `required` → Validation rule.
- `enum` → Restricts to predefined values.

Q. **What happens if two users try to update the same document at the same time?**

Possible **race condition**.
Solution: Use **Optimistic Concurrency Control** → Mongoose adds __v version key. If mismatch → update fails.

Q. **How do you implement pagination with Mongoose?**

Using `skip()` and `limit()`:

```
User.find().skip(10).limit(5);
```

Or better, use `cursor-based pagination` with `_id`.