Mongoose Schema Guide with Examples

Code Example: Sample Schema

```
const mongoose = require("mongoose");

const sampleSchema = new mongoose.Schema({
  name: String, // String
  age: Number, // Number
  isActive: Boolean, // Boolean
  createdAt: Date, // Date
  photo: Buffer, // Binary Buffer
  description: mongoose.Schema.Types.Mixed, // Any type
  userRef: mongoose.Schema.Types.ObjectId, // Reference to another document
  tags: [String], // Array of strings
  price: mongoose.Schema.Types.Decimal128, // High precision number
  settings: {
    type: Map,
    of: String
  },
  uuid: mongoose.Schema.Types.UUID // (Only works if MongoDB 7+)
});

const Sample = mongoose.model("Sample", sampleSchema);

async function insertSample() {
  await mongoose.connect("mongodb://127.0.0.1:27017/test");

  const sample = new Sample({
    name: "Talha Nadeem",
    age: 22,
    isActive: true,
    createdAt: new Date(),
    photo: Buffer.from("Hello, World!"),
    description: { anything: "This is mixed data" },
    userRef: new mongoose.Types.ObjectId(),
    tags: ["mern", "mongodb", "developer"],
    price: mongoose.Types.Decimal128.fromString("999.99"),
    settings: new Map([
      ["theme", "dark"],
      ["notifications", "enabled"]
    ]),
    uuid: new mongoose.Types.UUID()
  });

  const result = await sample.save();
  console.log("\u2705 Sample inserted:\n", result);
```

```
}
insertSample();
```

Mongoose Schema Field Options Explained

🔹 1. required

- **Concept:** Field must exist. If missing, validation fails.
- **Syntax:** `name: { type: String, required: true }`
- **Example:**

```
const user = new User({ age: 22 }); // ❌ Error: name is required.
```

🔹 2. default

- **Concept:** Assigns default value if none is provided.
- **Example:**

```
createdAt: { type: Date, default: () => Date.now() },
isActive: { type: Boolean, default: true }
```

🔹 3. select

- **Concept:** Hides fields from query results unless explicitly selected.
- **Use Case:** Hide passwords.

```
password: { type: String, required: true, select: false }
User.findOne({ name: "Talha" }) // hidden
User.findOne({ name: "Talha" }).select("+password") // visible
```

🔹 4. validate

- **Concept:** Adds custom validator function.
- **Example:**

```
email: {
  type: String,
  validate: {
    validator: (v) => v.includes("@"),
    message: "Invalid email!"
  }
}
```

🔹 5. get / set

- **Concept:** Transform value on get/set.

- **Example:**

```
price: {
  type: Number,
  get: v => (v / 100).toFixed(2),
  set: v => v * 100
}
```

🔹 6. alias

- **Concept:** Virtual shortcut field.
- **Example:**

```
fullName: { type: String, alias: 'name' }
```

🔹 7. immutable

- **Concept:** Can't be changed after creation.
- **Example:**

```
createdAt: {
  type: Date,
  default: Date.now,
  immutable: true
}
```

🔹 8. transform

- **Concept:** Modifies output during JSON conversion.
- **Example:**

```
toJSON: {
  transform: (doc, ret) => {
    delete ret._id;
    delete ret.__v;
    return ret;
  }
}
```

🔹 9. index / unique / sparse

- `index: true` — for faster querying.
- `unique: true` — ensures no duplicates.
- `sparse: true` — index only documents with values.

🔹 10. lowercase / uppercase / trim

- **Example:**

```
username: {
  type: String,
  lowercase: true,
  trim: true
}
```

🔹 **11. match**

- **Concept:** Regex validation.

```
email: {
  type: String,
  match: /.+\@.+\..+/
}
```

🔹 **12. enum**

- **Concept:** Restricts to a set of values.

```
role: {
  type: String,
  enum: ['Admin', 'User', 'Guest']
}
```

🔹 **13. minLength / maxLength (Strings)**

```
password: {
  type: String,
  minLength: 6,
  maxLength: 20
}
```

🔹 **14. min / max (Numbers)**

```
age: {
  type: Number,
  min: 18,
  max: 60
}
```

🔹 **15. populate**

- **Concept:** Join collections using ObjectId reference.

```
userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }
Post.find().populate("userId")
```

✅Complete Schema Example

```
const userSchema = new mongoose.Schema({
  name: { type: String, required: true, trim: true },
  email: { type: String, required: true, lowercase: true, unique: true,
match: /.+@.+\..+/ },
  password: { type: String, required: true, minLength: 6, select: false },
  role: { type: String, enum: ['Admin', 'User'], default: 'User' },
  createdAt: { type: Date, default: Date.now, immutable: true },
  age: { type: Number, min: 18, max: 60 },
  settings: {
    type: Map,
    of: String,
    default: () => new Map([["theme", "light"]])
  }
});
```

🔚Summary Table

Option	Purpose
required	Field must be provided
default	Assigns default if not provided
select	Hide/show in queries
validate	Custom validation logic
get / set	Transform before get/save
alias	Virtual field name
immutable	Can't change after first set
transform	Control JSON output
index	Fast lookup
unique	No duplicates allowed
lowercase	Auto lowercase
match	Regex check
enum	Restrict to certain values
minLength	Min characters in string

Option	Purpose
min	Min value for number
populate	Reference other documents

Let me know if you need real-world MERN use cases for these! 🚫