Mongoose Cheatsheet (Concisa): Sintassi Rapida

Schema: Definizione Base

- Struttura dati per MongoDB in Mongoose.
- Definisce tipi, validazioni, middleware.

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

const schema = new mongoose.Schema({
  nome: String, // Tipo Stringa
  eta: Number, // Tipo Numero
  isStudente: Boolean, // Tipo Booleano
  dataIscrizione: Date // Tipo Data
  // ... altri campi
});

const Modello = mongoose.model("NomeModello", schema); // Crea Modello
```

Tipi di Schema Comuni (Sintassi Rapida)

- **String:** `String` / `{ type: String, ... }`
- **Number:** `Number` / `{ type: Number, ... }`
- **Boolean:** `Boolean` / `{ type: Boolean, ... }`
- **Date:** `Date` / `{ type: Date, ... }`
- **Buffer:** `Buffer` / `{ type: Buffer, ... }`
- **ObjectId:** `mongoose.Schema.Types.ObjectId` / `{ type: mongoose.Schema.Types.ObjectId, ... }`
- **Array:** `[String]`, `[Number]`, `[{ type: String }]`, `[Schema]`
- **Mixed:** `mongoose.Schema.Types.Mixed` / `{ type: mongoose.Schema.Types.Mixed, ... }`
- **Schema (Subdocument):** `Schema` / `{ type: Schema, ... }`

Esempio Tipi Schema + Validazioni:

```
const schemaEsempio = new mongoose.Schema({
  nome: { type: String, required: true, trim: true },
  punteggio: { type: Number, min: 0, max: 100, default: 0 },
  email: {
    type: String,
    lowercase: true,
    validate: { validator: v => /regex_email/.test(v), message: "Email non valida" }
  },
  tags: [String],
  dataCreazione: { type: Date, default: Date.now }
});
```

Subdocument: Definizione e Uso Rapido

- Schemi annidati dentro altri schemi.

- Per dati "parte-di" o "has-a" embedded.

Definizione Subdocument:

```
const IndirizzoSchema = new mongoose.Schema({
  via: String,
  citta: String,
  cap: String
});

const UtenteSchema = new mongoose.Schema({
  nome: String,
  eta: Number,
  indirizzo: IndirizzoSchema // Subdocument 'indirizzo'
});

const Utente = mongoose.model("Utente", UtenteSchema);
```

Creazione con Subdocument:

```
const nuovoUtente = new Utente({
  nome: "Mario Rossi",
  eta: 30,
  indirizzo: { via: "Via Roma, 1", citta: "Milano", cap: "20100" }
});
nuovoUtente.save();
```

ObjectId Reference: Definizione e Popolazione Rapida

- `ObjectId` per ID unici e reference tra collezioni.
- `ref: 'NomeModello'` per collegare a un modello.
- `populate('campoReference')` per ottenere dati relazionati.

Definizione Reference (ObjectId):

```
const AutoreSchema = new mongoose.Schema({ nome: String, bio: String });
const ArticoloSchema = new mongoose.Schema({
  titolo: String,
  contenuto: String,
  autore: { type: mongoose.Schema.Types.ObjectId, ref: "Autore" } // Reference
});

const Autore = mongoose.model("Autore", AutoreSchema);
const Articolo = mongoose.model("Articolo", ArticoloSchema);
```

Creazione con Reference:

```
const autoreSalvato = await new Autore({ nome: "Giovanni Verdi" }).save();
const articoloSalvato = await new Articolo({
  titolo: "Titolo Articolo",
```

```
  contenuto: "...",
  autore: autoreSalvato._id
}).save();
```

Popolazione Reference (`populate()`):

```
const articoloPopolato = await
Articolo.findById(articoloSalvato._id).populate("autore");
console.log(articoloPopolato.autore.nome); // Accedi ai dati dell'autore
```

Opzioni Schema (Sintassi Rapida)

- `timestamps: true` // `createdAt` , `updatedAt`
- `versionKey: false` // Rimuove `__v`
- `toJSON: { virtuals: true }` // Virtuals in JSON

Esempio Opzioni Schema:

```
const schemaOpzioni = new mongoose.Schema(
  {
    nome: String,
    prezzo: Number
  },
  {
    timestamps: true,
    versionKey: false,
    toJSON: { virtuals: true }
  }
);

schemaOpzioni.virtual("prezzoEuro").get(() => (this.prezzo * 0.85).toFixed(2) +
"€");
const ModelloOpzioni = mongoose.model("ModelloOpzioni", schemaOpzioni);
```

Modelli: Operazioni CRUD Rapide

- `mongoose.model('NomeModello', schema)` : Crea modello.
- `Modello.create({ ... })` / `new Modello({ ... }).save()` : Crea.
- `Modello.find({ ... })` , `findOne({ ... })` , `findById(id)` : Leggi.
- `updateOne({ ... }, { ... })` , `updateMany({ ... }, { ... })` , `findByIdAndUpdate(id, { ... })` : Aggiorna.
- `deleteOne({ ... })` , `deleteMany({ ... })` , `findByIdAndDelete(id)` : Cancella.

Esempio Operazioni CRUD:

```
// Crea
const nuovoUtente = await ModelloUtente.create({ nome: "Luca Bianchi", eta: 25 });
// Leggi (Find)
const utentiGiovani = await ModelloUtente.find({ eta: { $lt: 30 } });
// Leggi (FindById)
```

```
const utenteTrovato = await ModelloUtente.findById(nuovoUtente._id);
// Aggiorna
await ModelloUtente.updateOne({ _id: nuovoUtente._id }, { eta: 26 });
// Cancella
await ModelloUtente.deleteOne({ _id: nuovoUtente._id });
```

Esempio Blog (Conciso): Subdocument e Reference

```
const CategoriaSchema = new mongoose.Schema({ nome: String });
const CommentoSchema = new mongoose.Schema({
  testo: String,
  autore: { type: mongoose.Schema.Types.ObjectId, ref: "Utente" },
  data: { type: Date, default: Date.now }
});
const ArticoloBlogSchema = new mongoose.Schema({
  titolo: String,
  contenuto: String,
  autore: { type: mongoose.Schema.Types.ObjectId, ref: "Autore" }, // Reference
Autore
  categoria: CategoriaSchema, // Subdocument Categoria
  tags: [String],
  commenti: [CommentoSchema], // Array Subdocument Commenti
  dataPubblicazione: { type: Date, default: Date.now }
});

const ArticoloBlog = mongoose.model("ArticoloBlog", ArticoloBlogSchema);
```