# FSD Module 5 – 10 Marks Question Bank Answers

## 1. MongoDB Documents and Collections vs Relational Tables

In MongoDB, a document is a data structure composed of field and value pairs, similar to a JSON object. A collection is a group of MongoDB documents and is akin to a table in a relational database.

Differences from Relational Databases:
- MongoDB documents can contain nested data, arrays, and heterogeneous structures.
- Collections do not require a predefined schema.
- This schema-less nature provides flexibility and adaptability to evolving data models.

Example MongoDB Document:
```json
{
  "name": "John Doe",
  "email": "john@example.com",
  "interests": ["reading", "sports"]
}
```

In contrast, a relational table would require fixed columns, types, and table constraints.

MongoDB's flexibility is advantageous for applications with rapidly changing or diverse data needs.

## 2. MongoDB Query Language (MQL) and Basic Queries

MongoDB uses MongoDB Query Language (MQL) to retrieve and manipulate data.

Basic Queries:
- `db.collection.find()` – returns multiple documents
- `db.collection.findOne()` – returns a single document

Examples:
```js
db.users.find({ name: "Alice" });
db.products.find({ price: { $gt: 100 } });
```

MQL supports powerful operators like `$eq`, `$gt`, `$in`, `$and`, `$or` for querying specific fields, ranges, or conditions.

MQL provides expressive and intuitive query mechanisms similar to JavaScript syntax.

## 3. Installing MongoDB Locally and on Atlas

Installing Locally:
1. Download MongoDB from the official site.
2. Install using installer or package manager (e.g., Homebrew, apt).
3. Start MongoDB server with `mongod` command.
4. Use MongoDB Compass or shell to connect.

Using MongoDB Atlas:
1. Create account at https://www.mongodb.com/cloud/atlas.
2. Create a cluster and choose cloud provider and region.
3. Add database users and whitelist IPs.
4. Connect using connection string via Compass or driver.

Key Configurations:
- Set up authentication and roles.
- Configure firewall rules and connection methods (e.g., SRV records).

## 4. CRUD Operations in MongoDB

CRUD stands for Create, Read, Update, Delete – the four basic operations for managing data.

Create:
```js
db.users.insertOne({ name: "John", age: 30 });
db.users.insertMany([{ name: "Alice" }, { name: "Bob" }]);
```

Read:
```js
db.users.find();
db.users.find({ name: "John" });
```

Update:
```js
db.users.updateOne({ name: "John" }, { $set: { age: 35 } });
```

Delete:
```js
db.users.deleteOne({ name: "John" });
db.users.deleteMany({ age: { $lt: 25 } });
```

These operations form the backbone of all database interactions in MongoDB.

## 5. Projection in MongoDB

Projection refers to selecting only specific fields from a document in a query.

Example:
```js
db.users.find({}, { name: 1, email: 1, _id: 0 });
```

Here, only `name` and `email` are returned, and `_id` is excluded.

Projection is useful for:
- Reducing network load
- Enhancing performance
- Improving readability of results

You can also project nested fields using dot notation.

## 6. Webpack Production Build Configuration

Production builds in Webpack focus on performance, size reduction, and reliability.

Typical Configuration:
```js
mode: "production",
optimization: {
  minimize: true,
  splitChunks: { chunks: "all" },
  usedExports: true
}
```

Optimization Techniques:
- **Minification**: Removes unnecessary whitespace/comments.
- **Code Splitting**: Breaks large bundles into smaller chunks.
- **Tree Shaking**: Eliminates unused code.

- **Caching**: Use hashed filenames for long-term browser caching.

Tools like `TerserPlugin`, `HtmlWebpackPlugin`, and `CleanWebpackPlugin` assist in optimizing the final bundle.

## 7. Handling Libraries in Webpack

Webpack can include or exclude third-party libraries during the bundling process.

Including Libraries:
Install via npm/yarn and import in source files.

Excluding Libraries:
Use `externals` in Webpack config to avoid bundling large libraries already available on CDN.

Example:
```js
externals: {
  react: "React",
  "react-dom": "ReactDOM"
}
```

Benefits:
- Smaller bundle size
- Leverage browser caching of CDN libraries

Be sure to include CDN scripts in your HTML for excluded libraries.

## 8. Managing Assets with Webpack

Webpack uses loaders to process non-JS assets like images and fonts.

Common Loaders:
- `file-loader`: Emits files to output folder and returns public URL.
- `url-loader`: Converts small assets to base64 URIs.

Example Webpack config:
```js
module: {
  rules: [
    {
      test: /\.(png|jpg|gif|svg)$/,
```

```
    use: ["file-loader"]
   },
   {
    test: /\.(woff|woff2|eot|ttf|otf)$/,
    use: ["file-loader"]
   }
 ]
}
```

Loaders enable efficient asset management and bundling.

## 9. Modularization with Entry and Output in Webpack

Webpack uses modular architecture to split code into manageable files.

Entry Point:
- Defines the main file for the app (e.g., `index.js`)

Output:
- Specifies filename and path for the generated bundle

Example:
```js
entry: "./src/index.js",
output: {
 filename: "bundle.js",
 path: path.resolve(__dirname, "dist")
}
```

Impact:
- Better organization of code
- Easier debugging and scaling
- Supports multiple entry points for multi-page apps

## 10. Code Splitting in Webpack

Code splitting divides application code into chunks that are loaded only when needed.

Benefits:
- Faster initial load time
- Reduces JavaScript payload
- Improves performance on slow connections

Implementation using dynamic imports:
```js
import("./utils").then(module => {
  module.default();
});
```

Webpack handles these as separate bundles and loads them asynchronously.

Code splitting is crucial for modern web apps with large or modular codebases.