

# ACCA7002 Database Systems

## Tutorial 15

### Car Parking Management System with MongoDB

Activity 1: Setup MongoDB (if you haven't installed)

Install MongoDB on your machine. You can follow the installation instructions provided on the official MongoDB website (<https://docs.mongodb.com/manual/installation/>).

Start the MongoDB server.

Alternatively, you can use <https://www.mycompiler.io/new/mongodb>

Activity 2: Create a Database and Collection

Open the MongoDB shell.

Create a new database for the Car Parking Management System:

```
use car_parking
```

Create a collection named "cars" within the "car\_parking" database:

```
db.createCollection("cars")
```

Activity 3: Insert Data

Insert sample data into the "cars" collection:

```
db.cars.insertMany([
  { plateNumber: "ABC123", brand: "Toyota", model: "Camry",
    color: "Blue" },
  { plateNumber: "XYZ789", brand: "Honda", model: "Civic",
    color: "Red" },
  // Add more sample data as needed
])
```

Activity 4: CRUD Operations

Read Operation:

Retrieve all cars in the collection:

```
db.cars.find()
```

Create Operation:

Add a new car to the collection:

```
db.cars.insert({ plateNumber: "JKL456", brand: "Ford", model: "Focus", color: "Green" })
```

#### Update Operation:

Update the color of a specific car:

```
db.cars.update({ plateNumber: "ABC123" }, { $set: { color: "Silver" } })
```

#### Delete Operation:

Remove a car from the collection:

```
db.cars.remove({ plateNumber: "XYZ789" })
```

### Activity 5: Query and Review

#### Query Operation:

Retrieve cars of a specific color (e.g., all Blue cars):

```
db.cars.find({ color: "Blue" })
```

#### Activity 5: Update Database Schema:

Add a new collection named "reviews" within the "car\_parking" database:

```
db.createCollection("reviews")
```

Define the schema for reviews. For example:

```
// Sample Review Schema
{
  carPlateNumber: "ABC123",
  rating: 4,
  comment: "Great parking experience!",
  timestamp: ISODate("2023-12-05T12:00:00Z")
}
```

### Activity 6: CRUD Operations for Reviews

#### Create Operation for Reviews:

Allow customers to submit reviews for a specific car:

```
db.reviews.insert({
  carPlateNumber: "ABC123",
  rating: 5,
  comment: "Excellent service!",
  timestamp: new Date()
})
```

### Read Operation for Reviews:

Retrieve all reviews for a particular car:

```
db.reviews.find({ carPlateNumber: "ABC123" })
```

### Update Operation for Reviews:

Allow customers to update their reviews:

```
db.reviews.update(
  { carPlateNumber: "ABC123", /* add additional criteria if
needed */ },
  { $set: { rating: 4, comment: "Good experience!" } }
)
```

### Delete Operation for Reviews:

Allow customers to delete their reviews:

```
db.reviews.remove({ carPlateNumber: "ABC123", /* add
additional criteria if needed */ })
```

## Activity 7: Integrate Reviews with Car Data

### Query and Aggregate:

Retrieve car information along with aggregated review data (average rating, total reviews, etc.):

```
db.cars.aggregate([
  {
    $lookup: {
      from: "reviews",
      localField: "plateNumber",
      foreignField: "carPlateNumber",
      as: "reviews"
    }
  },
  {
    $addFields: {
      averageRating: { $avg: "$reviews.rating" },
      totalReviews: { $size: "$reviews" }
    }
  }
])
```

## Activity 8: Implementing Review Filtering in Car Parking Management System

### Step 1: Update Review Schema (if necessary)

#### Add Indexing:

Consider adding indexes on fields that users might filter on (e.g., rating, timestamp) for better query performance:

```
db.reviews.createIndex({ rating: 1 })
db.reviews.createIndex({ timestamp: 1 })
```

### Step 2: Implement Review Filtering

#### Filter by Rating:

Allow users to filter reviews based on a specific rating:

```
// Example: Get all reviews with a rating of 4
db.reviews.find({ rating: 4 })
```

#### Filter by Timestamp Range:

Allow users to filter reviews based on a timestamp range (e.g., from a specific start date to an end date):

```
// Example: Get reviews within a specific date range
const startDate = ISODate("2023-01-01T00:00:00Z");
const endDate = ISODate("2023-12-31T23:59:59Z");
db.reviews.find({ timestamp: { $gte: startDate, $lte: endDate
} })
```

#### Combine Filters:

Allow users to combine multiple filters (e.g., rating and timestamp):

```
// Example: Get reviews with a rating of 4 within a specific
date range
const startDate = ISODate("2023-01-01T00:00:00Z");
const endDate = ISODate("2023-12-31T23:59:59Z");
db.reviews.find({ rating: 4, timestamp: { $gte: startDate,
$lte: endDate } })
```

#### Query Optimization:

Monitor the performance of filtered queries, and optimize them if needed. Use the explain method to analyze query execution plans.

```
db.reviews.find({ rating: 4 }).explain("executionStats")
```

-----End-----