AWD 1111

# MongoDB Query Operators Guide

Comparison, Logical, Array, and Element Operators

# Query Operators Overview

MongoDB query operators allow you to build powerful queries beyond simple equality matches. Operators are prefixed with $ and can be combined to create complex filters.

| Category | Operators | Purpose |
|---|---|---|
| Comparison | $eq, $ne, $gt, $gte, $lt, $lte | Compare field values |
| Comparison | $in, $nin | Match against arrays of values |
| Logical | $and, $or, $not, $nor | Combine multiple conditions |
| Element | $exists, $type | Check field existence and type |
| Array | $all, $elemMatch, $size | Query array fields |
| Evaluation | $regex, $expr | Pattern matching and expressions |

**Database Used in Examples**
All examples use the concert_demo database with collections: concerts, venues, reviews, and fans.

# 1. Comparison Operators

Comparison operators filter documents based on field value comparisons. These are the most commonly used operators.

## 1.1 $eq (Equals)

Matches documents where the field equals the specified value. This is the default behavior when no operator is specified.

```
// Explicit $eq
db.concerts.find({ releaseYear: { $eq: 2024 } })

// Implicit equals (same result)
db.concerts.find({ releaseYear: 2024 })

// Find fans in Missouri
db.fans.find({ state: { $eq: "MO" } })
```

## 1.2 $ne (Not Equals)

Matches documents where the field does NOT equal the specified value.

```
// Concerts NOT from 2024
db.concerts.find({ releaseYear: { $ne: 2024 } })

// Fans who are NOT Gold members
db.fans.find({ membershipLevel: { $ne: "Gold" } })

// Venues not in Missouri
db.venues.find({ state: { $ne: "MO" } })
```

## 1.3 $gt (Greater Than)

Matches documents where the field is greater than the specified value.

```
// Concerts with attendance over 50,000
db.concerts.find({ attendance: { $gt: 50000 } })

// Tickets more than $100
db.concerts.find({ ticketPrice: { $gt: 100 } })

// Venues with capacity greater than 10,000
db.venues.find({ capacity: { $gt: 10000 } })
```

## 1.4 $gte (Greater Than or Equal)

Matches documents where the field is greater than OR equal to the specified value.

```
// Concerts from 2020 or later
db.concerts.find({ releaseYear: { $gte: 2020 } })

// Tickets $100 or more
db.concerts.find({ ticketPrice: { $gte: 100 } })

// Venues with capacity of at least 18,000
db.venues.find({ capacity: { $gte: 18000 } })
```

## 1.5 $lt (Less Than)

Matches documents where the field is less than the specified value.

```
// Concerts with attendance under 10,000
db.concerts.find({ attendance: { $lt: 10000 } })

// Tickets under $50
db.concerts.find({ ticketPrice: { $lt: 50 } })

// Venues with capacity less than 5,000
db.venues.find({ capacity: { $lt: 5000 } })
```

## 1.6 $lte (Less Than or Equal)

Matches documents where the field is less than OR equal to the specified value.

```
// Concerts from 2020 or earlier
db.concerts.find({ releaseYear: { $lte: 2020 } })

// Tickets $50 or less
db.concerts.find({ ticketPrice: { $lte: 50 } })

// Venues with capacity up to 20,000
db.venues.find({ capacity: { $lte: 20000 } })
```

## 1.7 Combining Comparison Operators (Range Queries)

You can combine comparison operators to create range queries.

```
// Tickets between $50 and $150 (inclusive)
db.concerts.find({
ticketPrice: { $gte: 50, $lte: 150 }
})

// Concerts from 2015 to 2020
db.concerts.find({
releaseYear: { $gte: 2015, $lte: 2020 }
})

// Venues with capacity between 5,000 and 20,000
db.venues.find({
capacity: { $gte: 5000, $lte: 20000 }
})

// Attendance between 10,000 and 50,000 (exclusive)
db.concerts.find({
attendance: { $gt: 10000, $lt: 50000 }
})
```

## 1.8 $in (In Array)

Matches documents where the field value equals ANY value in the specified array.

```
// Fans in Missouri, Illinois, or New York
db.fans.find({ state: { $in: ["MO", "IL", "NY"] } })

// Concerts from specific years
db.concerts.find({ releaseYear: { $in: [2023, 2024] } })

// Specific membership levels
db.fans.find({ membershipLevel: { $in: ["Gold", "Platinum"] } })

// Venues in specific cities
db.venues.find({ city: { $in: ["St. Louis", "Chicago"] } })
```

**$in with Arrays**
When used on an array field like genres, $in matches if ANY element of the field array matches ANY element of the query array.

```
// Concerts that are Rock OR Jazz (matches either genre)
db.concerts.find({ genres: { $in: ["Rock", "Jazz"] } })
```

## 1.9 $nin (Not In Array)

Matches documents where the field value does NOT equal any value in the specified array.

```
// Fans NOT in Missouri or Illinois
db.fans.find({ state: { $nin: ["MO", "IL"] } })

// Concerts NOT from 2023 or 2024
db.concerts.find({ releaseYear: { $nin: [2023, 2024] } })

// Not Silver or Bronze members
db.fans.find({ membershipLevel: { $nin: ["Silver", "Bronze"] } })
```

# 2. Logical Operators

Logical operators combine multiple query conditions. They allow you to build complex queries with AND, OR, and NOT logic.

## 2.1 Implicit $and

When you specify multiple conditions in a query document, MongoDB treats them as an implicit AND.

```
// Both conditions must be true (implicit AND)
db.concerts.find({
releaseYear: 2024,
ticketPrice: { $lt: 100 }
})

// Same as above, explicit $and
db.concerts.find({
$and: [
{ releaseYear: 2024 },
{ ticketPrice: { $lt: 100 } }
]
})
```

## 2.2 $and (Explicit AND)

Use explicit $and when you need multiple conditions on the SAME field, or for clarity.

```
// Multiple conditions on the same field
db.concerts.find({
$and: [
{ ticketPrice: { $gte: 50 } },
{ ticketPrice: { $lte: 150 } }
]
})

// Complex query with multiple conditions
db.concerts.find({
$and: [
{ releaseYear: { $gte: 2020 } },
{ attendance: { $gt: 10000 } },
{ ticketPrice: { $lt: 200 } }
]
})
```

**When to Use Explicit $and**
Use explicit $and when you have multiple conditions on the same field. For different fields, implicit AND (comma-separated) is cleaner.

## 2.3 $or

Matches documents that satisfy AT LEAST ONE of the conditions.

```
// Rock OR Jazz concerts
db.concerts.find({
$or: [
{ genres: "Rock" },
{ genres: "Jazz" }
]
})

// Free concerts OR concerts under $50
db.concerts.find({
$or: [
{ ticketPrice: 0 },
{ ticketPrice: { $lt: 50 } }
]
})

// Fans from Missouri OR Gold members
db.fans.find({
$or: [
{ state: "MO" },
{ membershipLevel: "Gold" }
]
})
```

## 2.4 Combining $and and $or

You can nest logical operators for complex queries.

```
// 2024 concerts that are either Rock or under $100
db.concerts.find({
releaseYear: 2024,
$or: [
{ genres: "Rock" },
{ ticketPrice: { $lt: 100 } }
]
})

// (Missouri OR Illinois) AND (Gold OR Platinum)
db.fans.find({
$and: [
{ $or: [{ state: "MO" }, { state: "IL" }] },
{ $or: [{ membershipLevel: "Gold" }, { membershipLevel: "Platinum" }] }
]
})
```

## 2.5 $not

Inverts the effect of a query expression. Matches documents that do NOT match the expression.

```
// Tickets NOT greater than $100 (same as $lte: 100)
db.concerts.find({
ticketPrice: { $not: { $gt: 100 } }
})

// Attendance NOT less than 10000
db.concerts.find({
attendance: { $not: { $lt: 10000 } }
})

// Release year NOT in range 2020-2024
db.concerts.find({
releaseYear: { $not: { $gte: 2020, $lte: 2024 } }
})
```

**$not Syntax**
$not requires an operator expression. Use { field: { $not: { $operator: value } } }. It cannot be used with simple values directly.

## 2.6 $nor

Matches documents that fail ALL of the specified conditions (NOT OR).

```
// Concerts that are NEITHER Rock NOR expensive (>$150)
db.concerts.find({
$nor: [
{ genres: "Rock" },
{ ticketPrice: { $gt: 150 } }
]
})

// Fans who are NEITHER from MO NOR Gold members
db.fans.find({
$nor: [
{ state: "MO" },
{ membershipLevel: "Gold" }
]
})
```

# 3. Element Operators

Element operators query based on field existence or data type, rather than field values.

## 3.1 $exists

Matches documents where the field exists (or doesn't exist).

```
// Documents where setList field exists
db.concerts.find({ setList: { $exists: true } })

// Documents where setList field does NOT exist
db.concerts.find({ setList: { $exists: false } })

// Venues that have a location field
db.venues.find({ location: { $exists: true } })

// Fans without an email field (if any)
db.fans.find({ email: { $exists: false } })
```

**$exists and null**
$exists: true matches documents where the field exists, even if the value is null. To find non-null values, combine with $ne: null.

```
// Field exists AND is not null
db.concerts.find({
setList: { $exists: true, $ne: null }
})
```

## 3.2 $type

Matches documents where the field is of the specified BSON type.

```
// Fields that are strings
db.concerts.find({ artist: { $type: "string" } })

// Fields that are numbers
db.concerts.find({ ticketPrice: { $type: "number" } })

// Fields that are arrays
db.concerts.find({ genres: { $type: "array" } })

// Fields that are dates
db.reviews.find({ date: { $type: "date" } })
```

Common BSON type identifiers:

| Type | Number | Alias |
|---|---|---|
| Double | 1 | "double" |
| String | 2 | "string" |
| Object | 3 | "object" |
| Array | 4 | "array" |
| ObjectId | 7 | "objectId" |
| Boolean | 8 | "bool" |
| Date | 9 | "date" |
| Null | 10 | "null" |
| 32-bit Integer | 16 | "int" |
| 64-bit Integer | 18 | "long" |

# 4. Array Operators

Array operators are designed specifically for querying array fields. These are essential when working with fields like genres, venue_ids, and setList.

## 4.1 Basic Array Matching

Without any operator, querying an array field matches if ANY element matches.

```
// Matches if "Rock" is anywhere in the genres array
db.concerts.find({ genres: "Rock" })

// Matches if "Alive" is in the setList
db.concerts.find({ setList: "Alive" })

// Matches concerts at a specific venue
db.concerts.find({
venue_ids: ObjectId("5f8d04b3b54764421b7156d2")
})
```

## 4.2 $all

Matches documents where the array contains ALL of the specified elements (in any order).

```
// Concerts that are BOTH Rock AND Classic Rock
db.concerts.find({
genres: { $all: ["Rock", "Classic Rock"] }
})

// Concerts that played at BOTH Enterprise Center AND MSG
db.concerts.find({
venue_ids: {
$all: [
ObjectId("5f8d04b3b54764421b7156d2"),
ObjectId("5f8d04b3b54764421b7156d4")
]
}
})

// SetList must include both "Alive" and "Even Flow"
db.concerts.find({
setList: { $all: ["Alive", "Even Flow"] }
})
```

**$all vs $in**
$all requires ALL elements to match. $in requires ANY element to match. Think of $all as AND, $in as OR.

## 4.3 $size

Matches documents where the array has exactly the specified number of elements.

```
// Concerts with exactly 2 genres
db.concerts.find({ genres: { $size: 2 } })

// Concerts that played at exactly 3 venues
db.concerts.find({ venue_ids: { $size: 3 } })

// Concerts with exactly 3 songs in setList
db.concerts.find({ setList: { $size: 3 } })

// Concerts at only 1 venue
db.concerts.find({ venue_ids: { $size: 1 } })
```

**$size Limitation**
$size only accepts exact numbers. It cannot be used with comparison operators like $gt. For "more than X elements," use aggregation $expr.

## 4.4 $elemMatch

Matches documents where at least one array element matches ALL specified conditions. Most useful for arrays of objects.

```
// For arrays of simple values, $elemMatch with single condition
// is the same as direct matching
db.concerts.find({
genres: { $elemMatch: { $eq: "Rock" } }
})

// More useful example: Finding within a range
// (if setList had numeric values)
db.concerts.find({
setList: { $elemMatch: { $regex: /^Love/ } }
})
```

$elemMatch is most powerful with arrays of embedded documents:

```
// If we had an array of objects like:
// scores: [{ type: "exam", score: 85 }, { type: "quiz", score: 90 }]

// Find where ONE element has type "exam" AND score > 80
db.students.find({
scores: {
$elemMatch: { type: "exam", score: { $gt: 80 } }
}
})
```

# 5. Evaluation Operators

Evaluation operators perform more complex matching including regular expressions and custom expressions.

## 5.1 $regex (Regular Expressions)

Matches documents where the string field matches a regular expression pattern.

```
// Artists starting with "T"
db.concerts.find({ artist: { $regex: /^T/ } })

// Artists starting with "T" (case-insensitive)
db.concerts.find({ artist: { $regex: /^t/i } })

// Tour names containing "Tour"
db.concerts.find({ tourName: { $regex: /Tour/ } })

// Fan names ending with "Smith"
db.fans.find({ name: { $regex: /Smith$/ } })

// Email addresses from example.com
db.fans.find({ email: { $regex: /@example\.com$/ } })
```

Common regex patterns:

| Pattern | Meaning | Example |
|---|---|---|
| ^text | Starts with | /^Taylor/ matches 'Taylor Swift' |
| text$ | Ends with | /Stones$/ matches 'Rolling Stones' |
| /i | Case-insensitive | /rock/i matches 'Rock', 'ROCK' |
| .* | Any characters | /T.*Swift/ matches 'Taylor Swift' |
| \\. | Literal dot | /@example\\.com/ matches email |

### Alternative $regex syntax

```
// Using $options for flags
db.concerts.find({
artist: { $regex: "taylor", $options: "i" }
})

// Using string instead of regex literal
db.concerts.find({
artist: { $regex: "^The" }
})
```

## 5.2 $expr

Allows using aggregation expressions in query filters. Useful for comparing two fields in the same document.

```
// This operator is more advanced - shown for reference

// Compare two fields (if we had budget vs actual fields)
// db.projects.find({
// $expr: { $gt: ["$actual", "$budget"] }
// })

// Using with aggregation operators
db.concerts.find({
$expr: {
$gt: [{ $size: "$genres" }, 1]
}
})
// Finds concerts with more than 1 genre
```

**$expr for Array Size Comparisons**
Since $size cannot use comparison operators, $expr with aggregation $size is how you find arrays with more/less than X elements.

# 6. Geospatial Operators

Geospatial operators allow you to query documents based on geographic location. This is useful for finding places within a certain distance of a point.

## 6.1 GeoJSON Format

MongoDB uses GeoJSON format to store location data. A point location looks like this:

```
{
"location": {
"type": "Point",
"coordinates": [-90.2026, 38.6268] // [longitude, latitude]
}
}
```

**Coordinate Order Warning!**
MongoDB uses [longitude, latitude] order, which is OPPOSITE of Google Maps (which shows latitude, longitude). This is a common source of errors!

## 6.2 Creating a Geospatial Index

Before using geospatial queries, you must create a 2dsphere index on the location field:

```
// Create a 2dsphere index on the location field
db.venues.createIndex({ location: "2dsphere" })

// Verify the index was created
db.venues.getIndexes()
```

**Index Required**
Geospatial queries like $near will fail with an error if the collection does not have a 2dsphere index on the location field.

## 6.3 $near - Find Nearby Locations

$near returns documents sorted by distance from a specified point. It requires a 2dsphere index.

### Basic $near Syntax

```
db.collection.find({
: {
$near: {
$geometry: {
type: "Point",
coordinates: [, ]
},
$maxDistance:
}
}
})
```

### Example: Find venues near Ranken Technical College

Ranken's coordinates are approximately: longitude -90.2265, latitude 38.6488

```
// Find venues within 10 miles of Ranken (10 miles = 16,093 meters)
db.venues.find({
location: {
$near: {
$geometry: {
type: "Point",
coordinates: [-90.2265, 38.6488]
},
$maxDistance: 16093
}
}
})
```

### Example: Find venues within 100 miles

```
// Find venues within 100 miles of Ranken (100 miles = 160,934 meters)
db.venues.find({
location: {
$near: {
$geometry: {
type: "Point",
coordinates: [-90.2265, 38.6488]
},
$maxDistance: 160934
}
}
})
```

## 6.4 Distance Conversions

$maxDistance is always specified in meters. Here are common conversions:

| Distance | Meters | Calculation |
|---|---|---|
| 1 mile | 1,609.34 | miles × 1609.34 |
| 5 miles | 8,047 | 5 × 1609.34 |
| 10 miles | 16,093 | 10 × 1609.34 |
| 25 miles | 40,234 | 25 × 1609.34 |
| 50 miles | 80,467 | 50 × 1609.34 |
| 100 miles | 160,934 | 100 × 1609.34 |
| 1 kilometer | 1,000 | km × 1000 |
| 10 kilometers | 10,000 | 10 × 1000 |

## 6.5 $nearSphere

$nearSphere is similar to $near but calculates distances using spherical geometry, which is more accurate for large distances on Earth.

```
// Find venues within 50 miles using spherical calculation
db.venues.find({
location: {
$nearSphere: {
$geometry: {
type: "Point",
coordinates: [-90.2265, 38.6488]
},
$maxDistance: 80467 // 50 miles in meters
}
}
})
```

## 6.6 $geoWithin - Find Within a Shape

$geoWithin finds documents with locations inside a specified shape (circle, polygon, etc.). Unlike $near, results are NOT sorted by distance.

```
// Find venues within a circular area (center point + radius)
db.venues.find({
location: {
$geoWithin: {
$centerSphere: [
[-90.2265, 38.6488], // center point [lng, lat]
50 / 3963.2 // radius in radians (miles / Earth radius)
]
}
}
})
```

**$near vs $geoWithin**
$near returns results sorted by distance (closest first) and requires an index. $geoWithin does not sort results and does not require an index (but performs better with one).

## 6.7 Finding Coordinates

To find longitude and latitude coordinates for a location:

- Google Maps: Right-click on a location and click the coordinates to copy them. Note: Google shows (latitude, longitude) - you need to REVERSE the order for MongoDB!
- latlong.net: Enter an address to get coordinates
- geocode.maps.co: Free geocoding API

### Common St. Louis Area Coordinates

| Location | Longitude | Latitude |
|---|---|---|
| Ranken Technical College | -90.2265 | 38.6488 |
| Downtown St. Louis (Arch) | -90.1848 | 38.6249 |
| Enterprise Center | -90.2026 | 38.6268 |
| Lambert Airport | -90.3599 | 38.7487 |

**Remember**
MongoDB coordinate order: [longitude, latitude] — Longitude comes FIRST!

# 7. Querying Nested Documents

MongoDB supports dot notation to query fields within embedded documents.

## 7.1 Dot Notation

Use dot notation to access nested fields. Always wrap the field path in quotes.

```
// Query nested location.coordinates
db.venues.find({
"location.type": "Point"
})

// Query nested location coordinates (longitude)
db.venues.find({
"location.coordinates.0": { $lt: -90 }
})

// If concerts had nested rating object:
// db.concerts.find({ "rating.imdb": { $gte: 8 } })
```

**Quotes Required**
Always use quotes around dot notation paths: "location.type" not location.type

## 7.2 Array Index with Dot Notation

Access specific array elements by index using dot notation.

```
// First genre equals "Pop"
db.concerts.find({ "genres.0": "Pop" })

// Second song in setList
db.concerts.find({ "setList.1": "Love Story" })

// First coordinate (longitude) is negative
db.venues.find({ "location.coordinates.0": { $lt: 0 } })
```

# 8. Query Operators Quick Reference

## Comparison Operators

| Operator | Description | Example |
|---|---|---|
| $eq | Equals | { field: { $eq: value } } |
| $ne | Not equals | { field: { $ne: value } } |
| $gt | Greater than | { field: { $gt: value } } |
| $gte | Greater than or equal | { field: { $gte: value } } |
| $lt | Less than | { field: { $lt: value } } |
| $lte | Less than or equal | { field: { $lte: value } } |
| $in | In array | { field: { $in: [v1, v2] } } |
| $nin | Not in array | { field: { $nin: [v1, v2] } } |

## Logical Operators

| Operator | Description | Example |
|---|---|---|
| $and | All conditions true | { $and: [{...}, {...}] } |
| $or | Any condition true | { $or: [{...}, {...}] } |
| $not | Inverts expression | { field: { $not: {...} } } |
| $nor | None of the conditions | { $nor: [{...}, {...}] } |

## Array Operators

| Operator | Description | Example |
|---|---|---|
| $all | All elements match | { arr: { $all: [v1, v2] } } |
| $size | Array length equals | { arr: { $size: 3 } } |
| $elemMatch | Element matches all conditions | { arr: { $elemMatch: {...} } } |

## Element & Evaluation

| Operator | Description | Example |
|---|---|---|
| $exists | Field exists | { field: { $exists: true } } |
| $type | Field is BSON type | { field: { $type: 'string' } } |
| $regex | Matches pattern | { field: { $regex: /pattern/ } } |
| $expr | Aggregation expression | { $expr: { $gt: ['$a', '$b'] } } |

## Geospatial Operators

| Operator | Description | Example |
|---|---|---|
| $near | Near point (sorted) | { loc: { $near: { $geometry: {...}, $maxDistance: m |
| $nearSphere | Near point (spherical) | Same as $near, spherical calculation |
| $geoWithin | Within shape | { loc: { $geoWithin: { $centerSphere: [...] } } } |

# 9. Reference Links

For more information, see the official MongoDB documentation:

- Query Operators: https://www.mongodb.com/docs/manual/reference/operator/query/
- Comparison Operators: https://www.mongodb.com/docs/manual/reference/operator/query-comparison/
- Logical Operators: https://www.mongodb.com/docs/manual/reference/operator/query-logical/
- Array Operators: https://www.mongodb.com/docs/manual/reference/operator/query-array/
- Element Operators: https://www.mongodb.com/docs/manual/reference/operator/query-element/
- Evaluation Operators: https://www.mongodb.com/docs/manual/reference/operator/query-evaluation/
- Geospatial Operators: https://www.mongodb.com/docs/manual/reference/operator/query-geospatial/