



MongoDB Day

# Schema Design for Great Application Performance



**Ingo Hofmann**  
Solutions Architect

# Agenda

What is “good performance”?

How does the storage engine help us?

Correct doesn't always look the same.

Real Life Examples.



What is “good  
performance”?

# Factors

**Response Time**

**Throughput**

**Efficiency**

**Scalability**

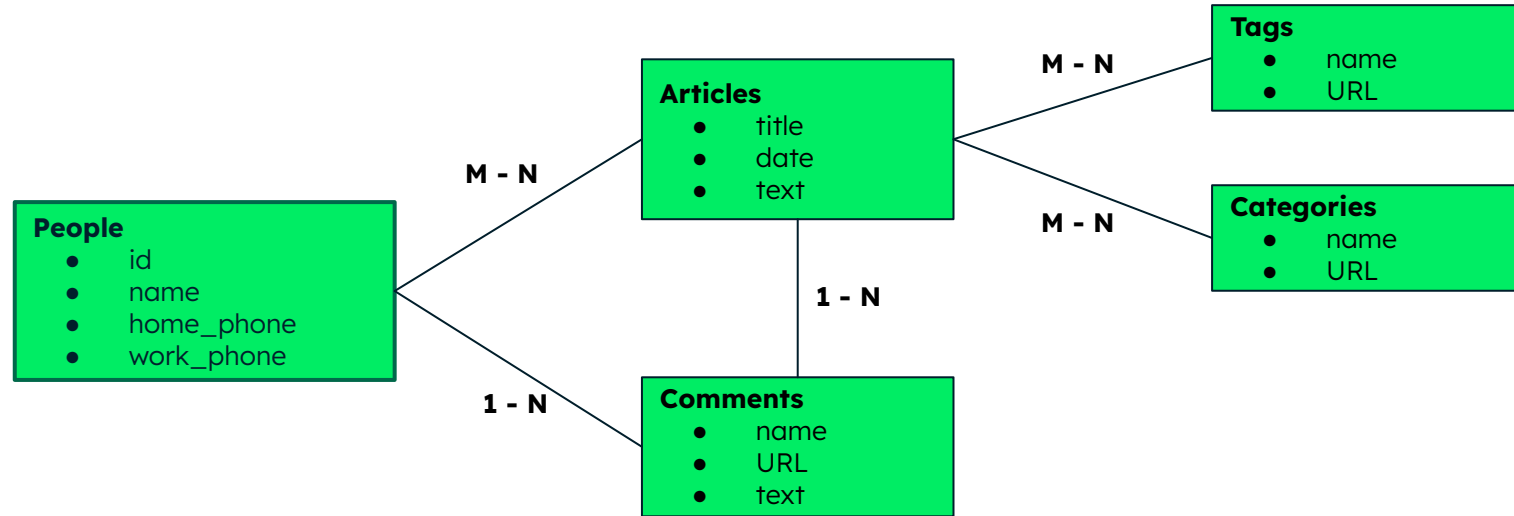


# Impactors

- Doing more work than is needed to process a request.
- Complex queries or too many requests for small bits of data.
- Not paying attention to how the system accesses data.
- Poorly distributed workloads.



# Tabular Database: 5 Tables

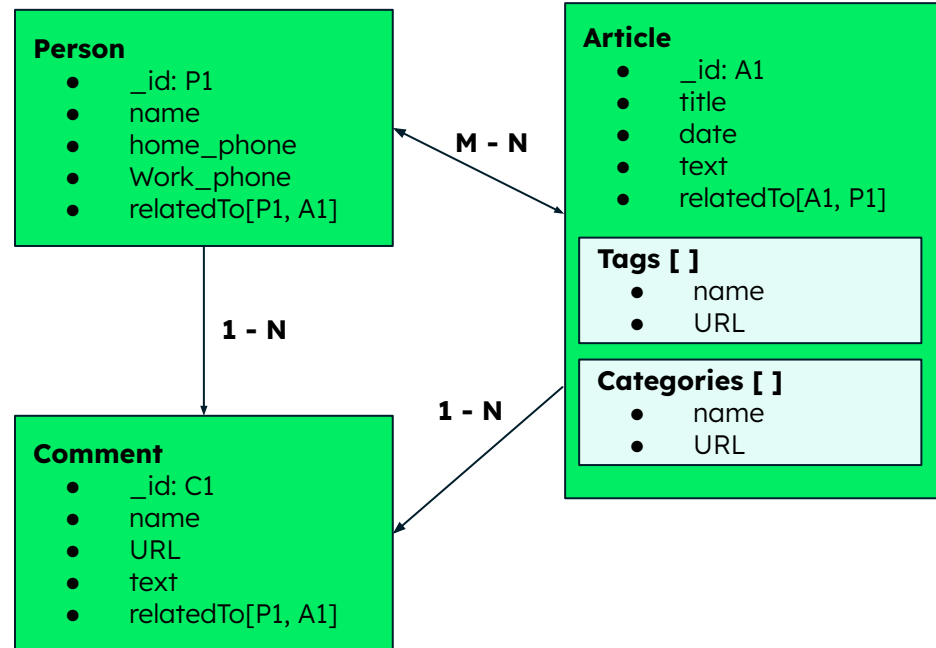




# MongoDB: 1, 2 or more Documents?



or





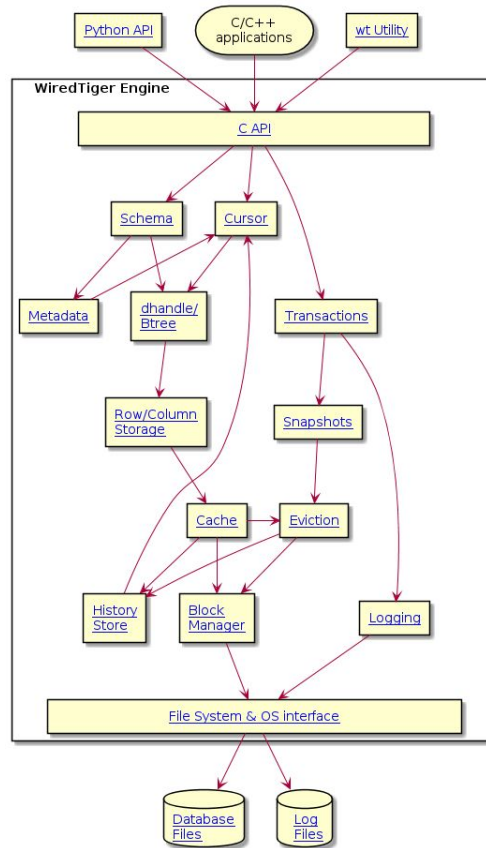
“

The optimal grouping of objects into documents or collections is determined by the workload.

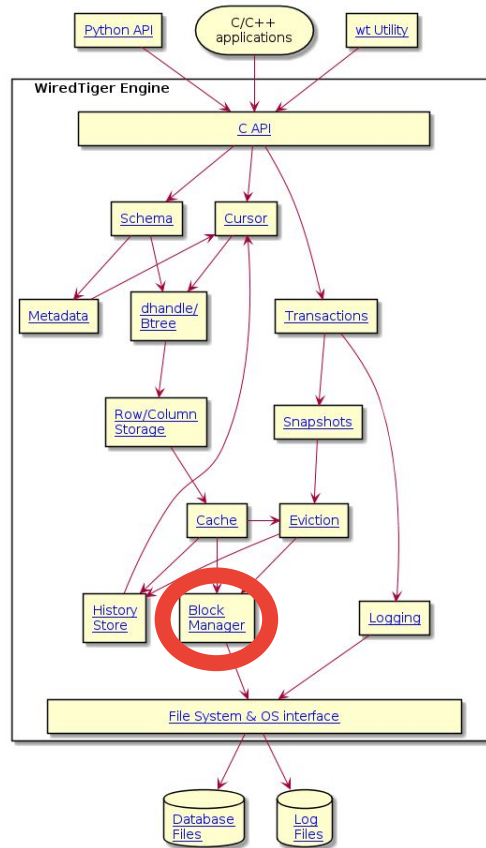


How does the storage  
engine help us?

# WiredTiger Architecture



# WiredTiger Architecture



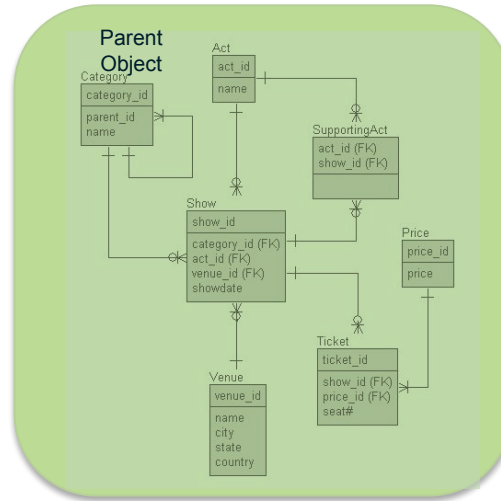


# How Document Storage Works

- WiredTiger is an **Non-Overwriting** storage engine.
- **Documents < 32KB** (uncompressed) will be aggregated and written to single pages.
- Related data stored in multiple documents will fragment as it is updated over time.
- High *read* velocity workloads benefit from **embedded document patterns**.
- High *write* velocity workloads benefit from **multiple document patterns**.
- Least Recently Used (LRU) Cache: Use **Covered Indexes** and **In Memory Working Sets** to maximize efficiency of hybrid workloads.



# Schema Anti-Patterns



OVER-EMBEDDING



# Schema Anti-Patterns: over-embedding

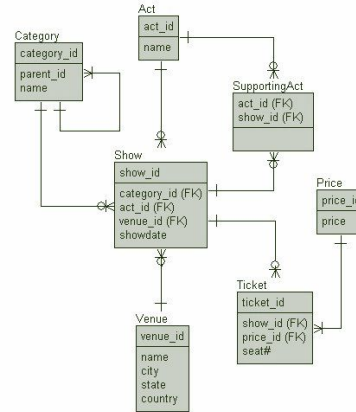
Data is unbounded

Data not required for  
query

Data not needed at all



# Schema Anti-Patterns



OVER-NORMALIZATION

# Schema Anti-Patterns: over-normalizing



Unbounded growth

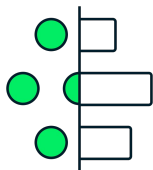
**JOINS**

Deeply nested arrays

Really large documents



# Schema Design Patterns Families



## Computation

- Approximation
- Computed



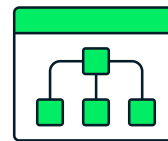
## Grouping

- Bucket
- Outlier
- Preallocated



## Life Cycle

- Archive
- Document Versioning
- Envelope
- Schema Versioning



## Polymorphism

- Inheritance
- Single Collection



## Relationships

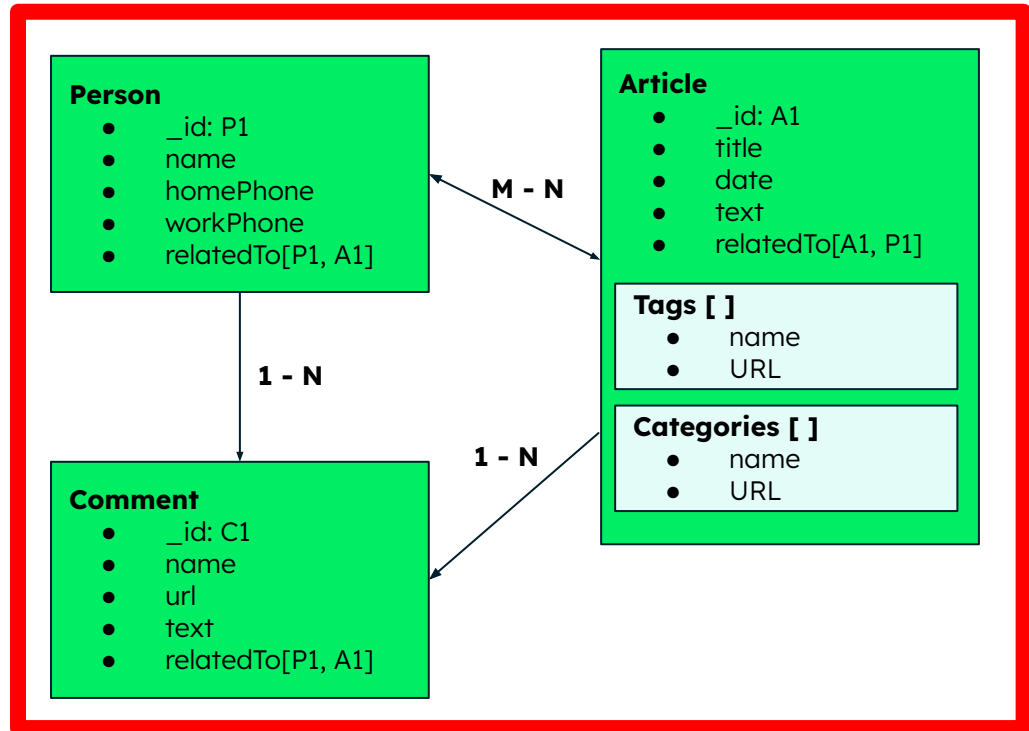
- Attribute
- Extended Reference
- Graph
- Subset
- Tree



# MongoDB: 1, 2 or more Documents?

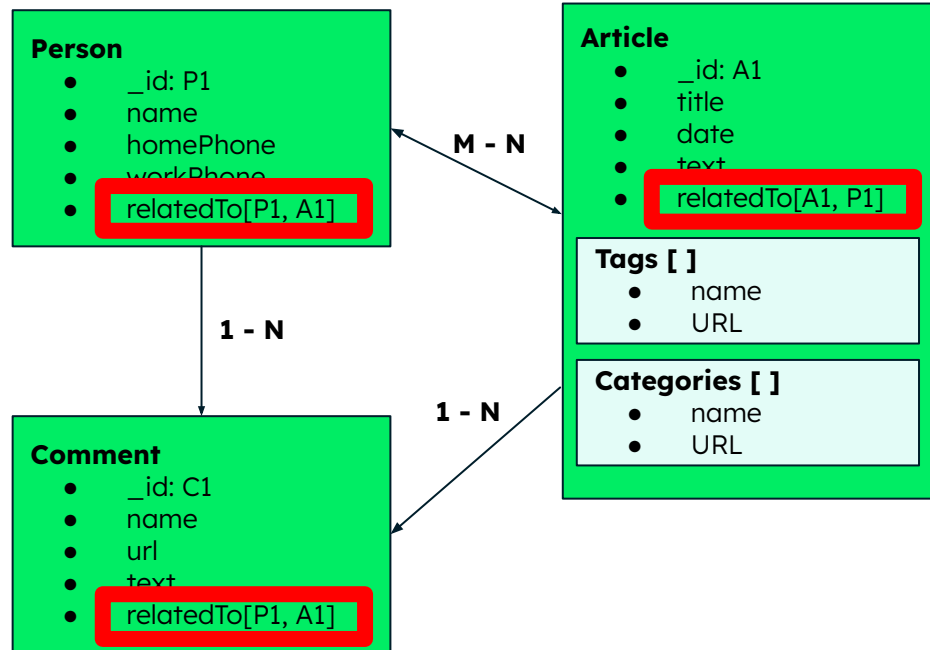


or



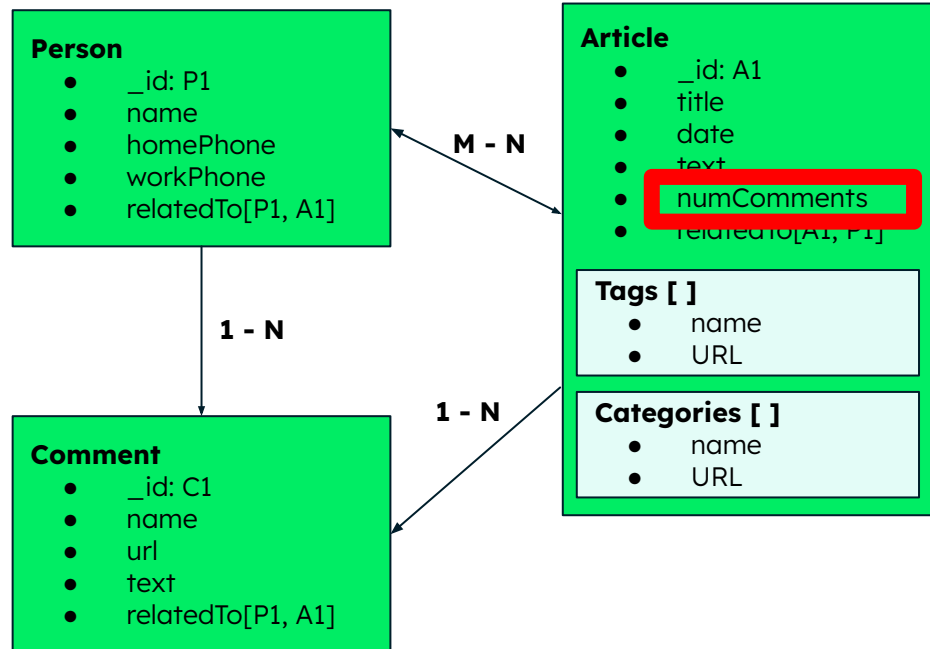


# Applying the Single Collection Pattern



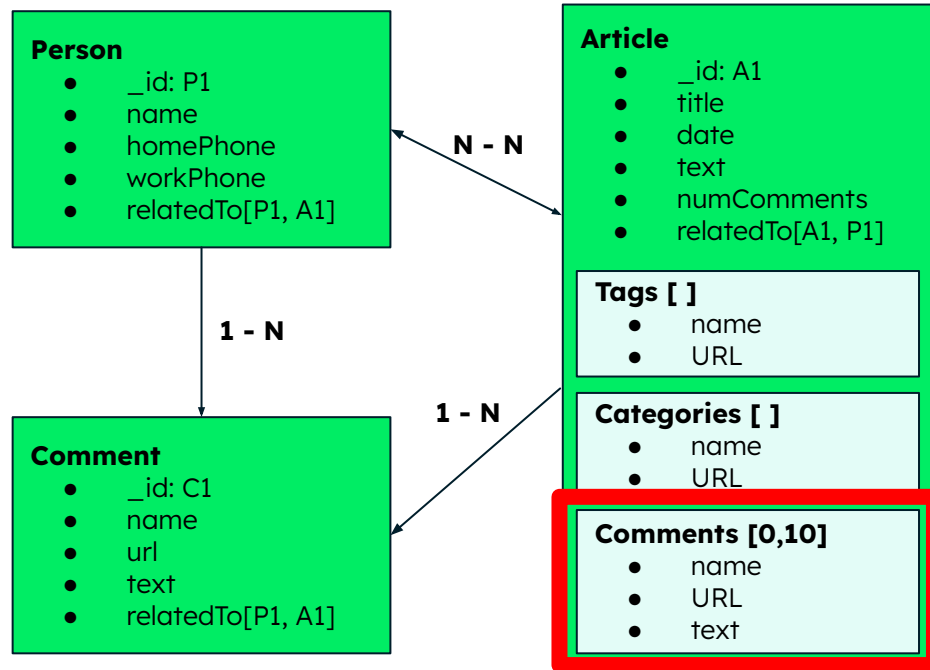


# Applying the Computed Pattern





# Applying the Subset Pattern





Correct does not always  
look the same.



# Signs of Trouble

Optimizing the wrong  
pattern

# Optimizing the wrong pattern



```
{
  "_id": "123456",
  "orderDate": "2023-10-09",
  "customer": {
    "customerId": "789012",
    "name": "John Doe",
    "address": {
      "street": "123 Main St",
      "city": "Anytown",
      "state": "CA",
      "postalCode": "12345",
      "country": "USA"
    },
    "email": "johndoe@example.com",
    "phoneNumber": "+1234567890"
  },
  "items": [
    {
      "itemId": "001",
      "sku": "P111",
      "description": "Laptop",
      "quantity": 1,
      "unitPrice": 1000.00,
      "total": 1000.00
    },
    {
      "itemId": "002",
      "sku": "P112",
      "description": "Mouse",
      "quantity": 2,
      "unitPrice": 15.00,
      "total": 30.00
    }
  ],
  "shipment": {
    "shipmentMethod": "Express",
    "trackingNumber": "EXP123456",
    "estimatedDeliveryDate": "2023-10-11"
  },
  "orderTotal": 1030.00,
  "status": "Shipped",
  "notes": "Customer requested to leave package at the back door if not at home."
}
```

## Access patterns:

| Pattern                                               | Volume     | Peak        |
|-------------------------------------------------------|------------|-------------|
| Create Order                                          | 1M daily   | 500 req/sec |
| View Order                                            | 250K daily | 125 req/sec |
| Get Product Summary<br>(how many products sold today) | Once daily | End of day  |

# Optimizing the wrong pattern



```
{
  "id": "123456",
  "orderDate": "2023-10-09",
  "customer": {
    "customerId": "789012",
    "name": "John Doe",
    "address": {
      "street": "123 Main St",
      "city": "Anytown",
      "state": "CA",
      "postalCode": "12345",
      "country": "USA"
    },
    "email": "johndoe@example.com",
    "phoneNumber": "+1234567890"
  },
  "items": [
    {
      "itemId": "001",
      "sku": "P111",
      "description": "Laptop",
      "quantity": 1,
      "unitPrice": 1000.00,
      "total": 1000.00
    },
    {
      "itemId": "002",
      "sku": "P112",
      "description": "Mouse",
      "quantity": 2,
      "unitPrice": 15.00,
      "total": 30.00
    }
  ],
  "shipment": {
    "shipmentMethod": "Express",
    "trackingNumber": "EXP123456",
    "estimatedDeliveryDate": "2023-10-11"
  },
  "orderTotal": 1030.00,
  "status": "Shipped",
  "notes": "Customer requested to leave package at the back door if not at home."
}
```

Access patterns:

| Pattern                                                      | Volume     | Peak        |
|--------------------------------------------------------------|------------|-------------|
| Create Order                                                 | 1M daily   | 500 req/sec |
| View Order                                                   | 250K daily | 125 req/sec |
| <b>Get Product Summary</b><br>(how many products sold today) | Once daily | End of day  |

`createIndex({items.sku: 1, orderDate:1})`

Will increase overhead of every write!



# Signs of Trouble

Optimizing the wrong  
pattern

Containers with more  
data than you need

# More data than you need...



```
{
  "_id": "123456",
  "orderDate": "2023-10-09",
  "customer": {
    "customerId": "789012",
    "name": "John Doe",
    "address": {
      "street": "123 Main St",
      "city": "Anytown",
      "state": "CA",
      "postalCode": "12345",
      "country": "USA"
    },
    "email": "johndoe@example.com",
    "phoneNumber": "+1234567890"
  },
  "items": [
    {
      "itemId": "001",
      "sku": "P111",
      "description": "Laptop",
      "quantity": 1,
      "unitPrice": 1000.00,
      "total": 1000.00
    },
    {
      "itemId": "002",
      "sku": "P112",
      "description": "Mouse",
      "quantity": 2,
      "unitPrice": 15.00,
      "total": 30.00
    }
  ],
  "shipment": {
    "shipmentMethod": "Express",
    "trackingNumber": "EXP123456",
    "estimatedDeliveryDate": "2023-10-11"
  },
  "orderTotal": 1030.00,
  "status": "Shipped",
  "notes": "Customer requested to leave package at the back door if not at home."
}
```

## Access patterns:

| Pattern                | Volume           | Peak                |
|------------------------|------------------|---------------------|
| Create Order           | 1M daily         | 500 req/sec         |
| <b>Add/Remove Item</b> | <b>20M daily</b> | <b>2.5K req/sec</b> |
| View Order             | 250K daily       | 125 req/sec         |
| Get Product Summary    | Once daily       | End of day          |

# Tuning the data model for the workload



## Orders

```
{
  "_id": "123456",
  "orderDate": "2023-10-09",
  "customer": {
    "customerId": "789012",
    "name": "John Doe",
    "address": {
      "street": "123 Main St",
      "city": "Anytown",
      "state": "CA",
      "postalCode": "12345",
      "country": "USA"
    },
    "email": "johndoe@example.com",
    "phoneNumber": "+1234567890"
  },
  "orderTotal": 1030.00,
  "status": "Shipped",
  "notes": "Customer requested to leave package at the back door if not at home."
}
```

## Order Items

```
{
  "_id": "123456_P111",
  "associatedOrder": "123456_order",
  "description": "Mouse",
  "quantity": 2,
  "unitPrice": 15.00
}

{
  "_id": "123456_P112",
  "associatedOrder": "123456_order",
  "description": "Laptop",
  "quantity": 1,
  "unitPrice": 1000.00
}
```

## Shipments

```
{
  "_id": "123456_shipment",
  "associatedOrder": "123456_order",
  "shipmentMethod": "Express",
  "trackingNumber": "EXP123456",
  "estimatedDeliveryDate": "2023-10-11"
}
```

# Implementation in Spring Data - 2 Queries



## Entities

```
@Document(collection = "orders")
public class Order {
    @Id
    private String id;
    ...
}

@Document(collection = "orders")
public class OrderItem {
    @Id
    private String id;
    ...
}
```

## Service

```
@Service
public class OrderService {

    @Autowired private OrderRepository orderRepository;
    @Autowired private OrderItemRepository orderItemRepository;
    ...

    public OrderWithItems getOrderWithItems(String orderId) {
        Order order = orderRepository.findById(orderId)
            .orElseThrow(() -> new OrderNotFoundException());
        List<OrderItem> items = orderItemRepository.findByIdStartingWith(orderId + "_");
        return new OrderWithItems(order, items);
    }
}
```

## Repositories

```
public interface OrderRepository extends MongoRepository<Order, String> {}

public interface OrderItemRepository extends MongoRepository<OrderItem,
String> {
    List<OrderItem> findByIdStartingWith(String orderIdPrefix);
}
```

## DTO

```
public class OrderWithItems {
    private Order order;
    private List<OrderItem> items;
    public OrderWithItems(Order order, List<OrderItem> items) {
        this.order = order;
        this.items = items;
    }
    ...
}
```

# Single Collection in Spring Data - One Query



```
@Repository
public class OrderRepositoryCustomImpl implements OrderRepositoryCustom {
    @Autowired private MongoTemplate mongoTemplate;

    @Override
    public Order findOrderWithItemsByIdPrefix(String orderId) {

        Query query = new Query(Criteria.where("_id").regex("^" + orderId + "($|_)"));
        List<Document> docs = mongoTemplate.find(query, Document.class, "orders");
        if (docs.isEmpty()) {
            return null;
        }
        Order order = null;
        List<OrderItem> items = new ArrayList<>();
        for (Document doc : docs) {
            String id = doc.getString("_id");
            if (id.equals(orderId)) {
                order = mongoTemplate.getConverter().read(Order.class, doc);
            } else {
                OrderItem item = mongoTemplate.getConverter().read(OrderItem.class, doc);
                items.add(item);
            }
        }
        if (order != null) {
            order.setItems(items);
        }
        return order;
    }
}
```

```
@Document(collection = "orders")
public class Order {
    @Id
    private String id;

    @Transient
    private List<OrderItem> items = new ArrayList<>();

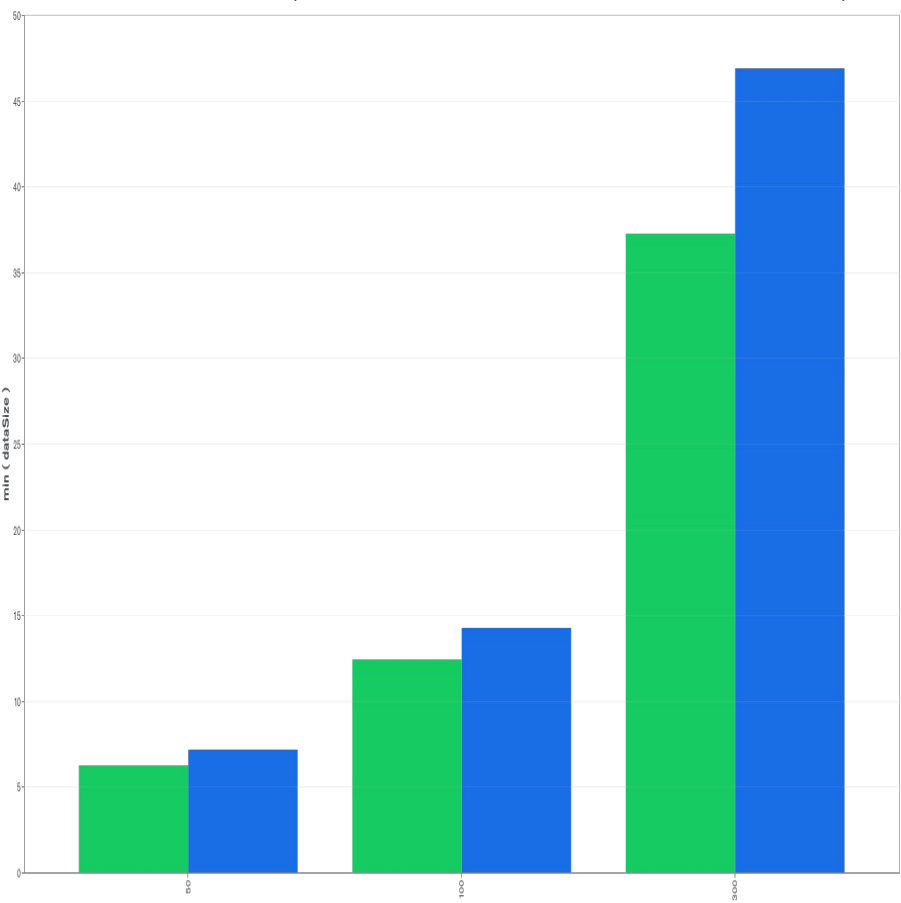
    ...

}
```



# Real Life Examples: TPC-C

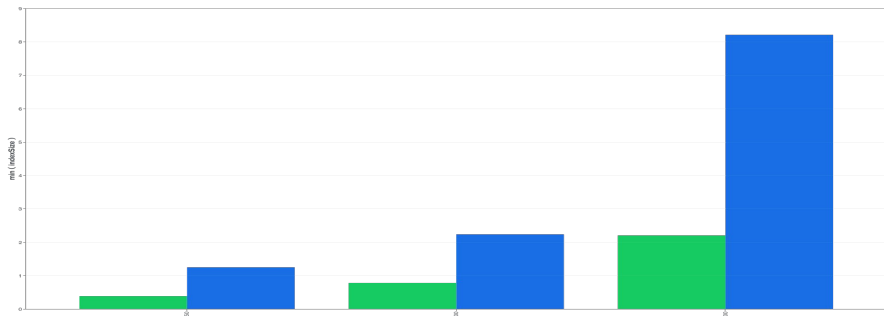
**Data Sizes (Number of warehouses x GB)**

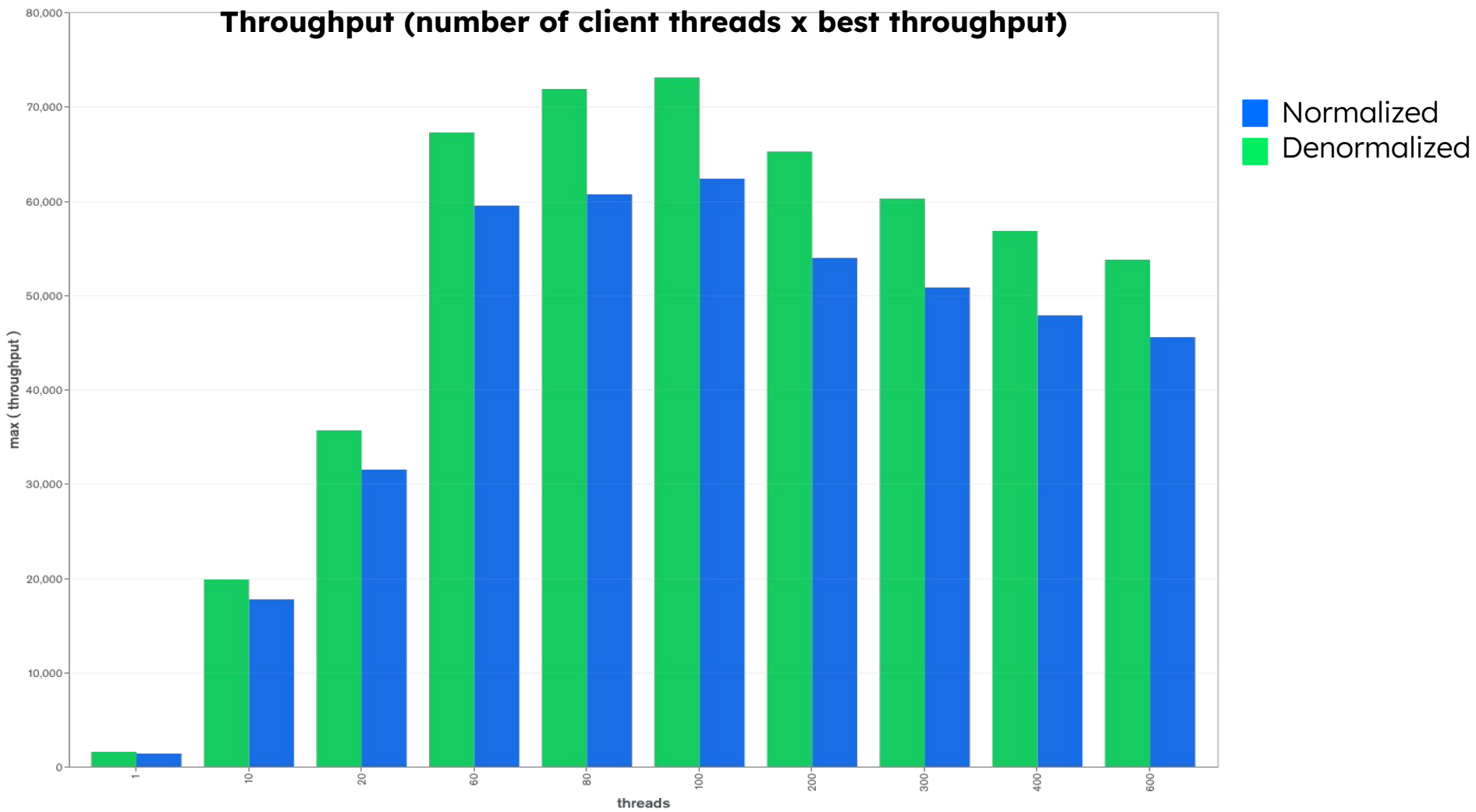


Normalized (Tables)

Denormalized (Documents)

**Index Sizes**







# Real Life Examples: Trading



# Trade execution service

```
{
  "_id": ObjectId("617c1f77a6f4441a0c0f1a0a"),
  "tradeId": "TRADE1234",
  "security": "MDB",
  "action": "BUY",
  "shares": 200,
  "creationDate": ISODate("2023-10-09T09:00:00Z"),
  "executions": [
    {
      "executionId": "EXEC5678",
      "shares": 200,
      "executionPrice": 150.00,
      "executionDate": ISODate("2023-10-09T10:00:00Z")
    }
  ]
}
```

| Pattern       | Volume           | Peak        |
|---------------|------------------|-------------|
| Insert Trade  | Up to 25M daily  | 25K req/sec |
| Add Execution | Up to 75M daily  | 75K req/sec |
| Get Trade     | Below 1K per day | <10 req/sec |



# Trade execution service

```
{
  "_id": ObjectId("617c1f77a6f4441a0c0f1a0a"),
  "tradeId": "TRADE1234",
  "security": "MDB",
  "action": "BUY",
  "shares": 200,
  "creationDate": ISODate("2023-10-09T09:00:00Z"),
  "executions": [
    {
      "executionId": "EXEC5678",
      "shares": 200,
      "executionPrice": 150.00,
      "executionDate": ISODate("2023-10-09T10:00:00Z")
    }
  ]
}
```

| Pattern       | Volume           | Peak        |
|---------------|------------------|-------------|
| Insert Trade  | Up to 25M daily  | 25K req/sec |
| Add Execution | Up to 75M daily  | 75K req/sec |
| Get Trade     | Below 1K per day | <10 req/sec |

createIndex({tradeId:1})

find({tradeId: "TRADE1234" })

**Expensive and slow!**



# Trade execution service

```
{
  {
    "_id": "TRADE1234",
    "security": "MDB",
    "action": "BUY",
    "shares": 200,
    "creationDate": ISODate("2023-10-09T09:00:00Z")
  },
  {
    "_id": "TRADE1234#EXEC5678",
    "shares": 200,
    "executionPrice": 150.00,
    "executionDate": ISODate("2023-10-09T10:00:00Z")
  },
  {
    "_id": "TRADE1234#EXEC91011",
    "shares": 50,
    "executionPrice": 152.00,
    "executionDate": ISODate("2023-10-09T10:00:10Z")
  }
}
```

| Pattern       | Volume           | Peak        |
|---------------|------------------|-------------|
| Insert Trade  | Up to 25M daily  | 25K req/sec |
| Add Execution | Up to 75M daily  | 75K req/sec |
| Get Trade     | Below 1K per day | <10 req/sec |

**find({\_id:{\$regex: "^TRADE1234" }})**



Align your schema with how data  
is used.

# MongoDB Skill Badges

Quickly learn and validate  
specific MongoDB skills

<https://learn.mongodb.com>

MongoDB Skill

Vector Search  
Fundamentals



MongoDB Skill

RAG with  
MongoDB



MongoDB Skill

Relational to  
Document Model



MongoDB Skill

Schema Patterns and  
Antipatterns



MongoDB Skill

Advanced Schema  
Patterns and Antipatterns



MongoDB Skill

Schema Design  
Optimization



Thank You

