PUNE INSTITUTE OF COMPUTER TECHNOLOGY

Subject: ADBMS (LP LAB)

Name: Aditya Kangune Roll No.: 33323

Batch: K11 Academic Year: 2021-22

---

# Assignment 1
# MONGO CRUD OPERATIONS

---

## Aim:

Create a database with suitable examples using MongoDB and implement:

- Inserting and saving document (batch insert, insert validation).
- Removing document
- Updating document (document replacement, using modifiers, up inserts, updating multiple documents, returning updated documents)
- Execute at least 10 queries on any suitable MongoDB database that demonstrates the following:
    - Find and find One (specific values)
    - Query criteria (Query conditionals, OR queries, $not, Conditional semantics)
    - Type-specific queries (Null, Regular expression, Querying arrays)
    - $ where queries
    - Cursors (Limit, skip, sort, advanced query options)

## Objective:

Following are the objectives:

1. To understand MongoDB basic commands.
2. To implement the concept of document-oriented databases.
3. To understand MongoDB retrieval commands.

## Theory:

### Comparison between MongoDB and SQL concepts:

| SQL concepts | MongoDB concepts |
|---|---|
| Table | Collection |
| Row | Document |
| Column | Field |
| Table Join | Embedded Documents |
| Primary Key | Primary Key |
| Specify any Unique column as a Primary key | _Id field is the primary key |
| Aggregation (Group By) | Aggregation Pipeline |

| SQL Database | NoSQL Database (MongoDB) |
| --- | --- |
| Relational database | Non-relational database |
| Supports SQL query language | Supports JSON query language |
| Table based | Collection based and key-value pair |
| Row based | Document based |
| Column based | Field based |
| Support foreign key | No support for foreign key |
| Support for triggers | No Support for triggers |
| Contains schema which is predefined | Contains dynamic schema |
| Not fit for hierarchical data storage | Best fit for hierarchical data storage |

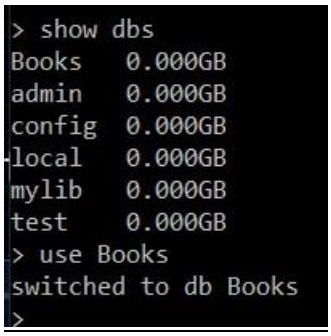
| | |
|---|---|
| Vertically scalable - increasing RAM | Horizontally scalable - add more servers |
| Emphasizes on ACID properties (Atomicity, Consistency, Isolation and Durability) | Emphasizes on CAP theorem (Consistency, Availability and Partition tolerance) |

## Commands:

## // Opening MongoDB in windows command shell:

i) Download MongoDB and install it.
ii) Create a new path variable in environment variables to use mongo shell anywhere in the device.
iii) Use commands:
(1) mongod
(2) mongo

## // Create commands:




> show collections
Book store
> db.createCollection(Books)

## // Shows current database

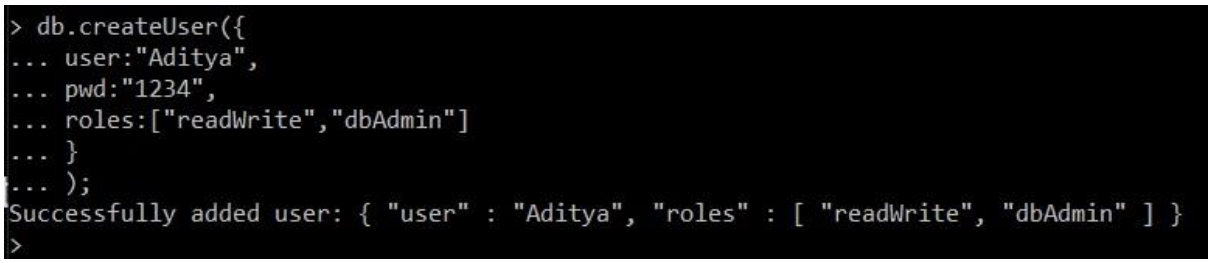
{ "ok" : 1 }
> show collections
Book store
Books
> use Books
switched to db Books
> db.Books.find().pretty()

> db.Books.insertMany([{name:"Harry Potter", author:"JK Rowling", id:123456789, price:490, pages:370}, {name:"The mortal instruments", author:"Cassandra Clare", id:4453356543, price:260, pages:290}, {name:"Who will cry when you die", id:7656445678, price:240, pages:120}, {name:"An Autobiography of a Yogi", id:8768905678, author:"Paramhansa Yogananda", price:390, pages:560}, {name:"The Subtle art of not giving a *uck", author:"Mark Manson", id:2574103698, price:450, pages:230}, {name:"Who Moved My Cheese?", author:"Spencer Johnson", id:9865741520, price:345, pages:180}])

```
{
	"acknowledged" : true,
	"insertedIds" : [
		ObjectId("611fa8bffc6e6eecbe413420"),
		ObjectId("611fa8bffc6e6eecbe413421"),
		ObjectId("611fa8bffc6e6eecbe413422"),
		ObjectId("611fa8bffc6e6eecbe413423"),
		ObjectId("611fa8bffc6e6eecbe413424"),
		ObjectId("611fa8bffc6e6eecbe413425")
	]
}
```

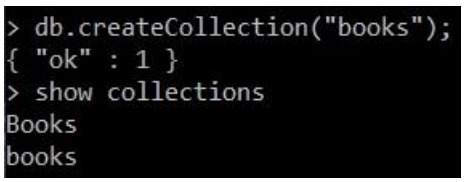
// Adding user

```
> db.createUser({
... user:"Aditya",
... pwd:"1234",
... roles:["readWrite","dbAdmin"]
... }
... );
Successfully added user: { "user" : "Aditya", "roles" : [
"readWrite", "dbAdmin" ] }
```




// Creating new collection(like tables in NoSQL)

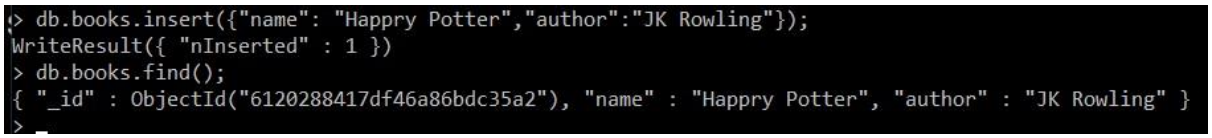
```
> db.createCollection("books");
{ "ok" : 1 }
> show collections
Books
books
```



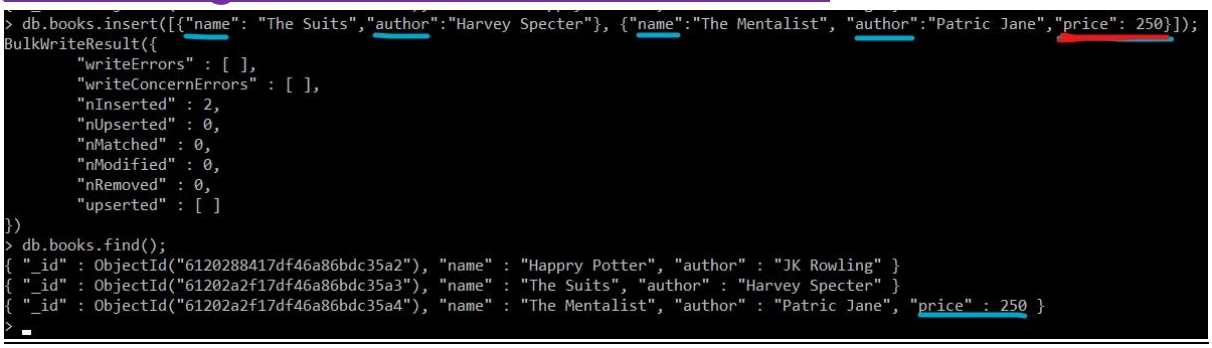

// Inserting one record and viewing it

// It automatiacally adds _id, which is a unique value so we don't have to worry about auto increament etc like we do in SQL

> db.books.insert({"name": "Happry Potter","author":"JK Rowling"});
WriteResult({ "nInserted" : 1 })
> db.books.find();
{ "_id" : ObjectId("6120288417df46a86bdc35a2"), "name" : "Happry Potter", "author" : "JK Rowling" }




// Inserting a record with an extra field:




// Using pretty to get cleaner output

## // Output after create commands

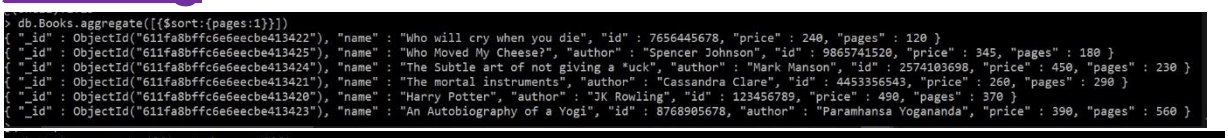
```
> db.Books.find().pretty()
{
        "_id" : ObjectId("611fa8bffc6e6eecbe413420"),
        "name" : "Harry Potter",
        "author" : "JK Rowling",
        "id" : 123456789,
        "price" : 490,
        "pages" : 370
}
{
        "_id" : ObjectId("611fa8bffc6e6eecbe413421"),
        "name" : "The mortal instruments",
        "author" : "Cassandra Clare",
        "id" : 4453356543,
        "price" : 260,
        "pages" : 290
}
{
        "_id" : ObjectId("611fa8bffc6e6eecbe413422"),
        "name" : "Who will cry when you die",
        "id" : 7656445678,
        "price" : 240,
        "pages" : 120
}
{
        "_id" : ObjectId("611fa8bffc6e6eecbe413423"),
        "name" : "An Autobiography of a Yogi",
        "id" : 8768905678,
        "author" : "Paramhansa Yogananda",
        "price" : 390,
        "pages" : 560
}
{
        "_id" : ObjectId("611fa8bffc6e6eecbe413424"),
        "name" : "The Subtle art of not giving a *uck",
        "author" : "Mark Manson",
        "id" : 2574103698,
        "price" : 450,
        "pages" : 230
}
{
        "_id" : ObjectId("611fa8bffc6e6eecbe413425"),
        "name" : "Who Moved My Cheese?",
        "author" : "Spencer Johnson",
        "id" : 9865741520,
        "price" : 345,
        "pages" : 180
}
```

```
>  db.Books.find({},{_id:0});
{ "name" : "Harry Potter", "author" : "JK Rowling", "id" : 123456789, "price" : 490, "pages" : 370 }
{ "name" : "The mortal instruments", "author" : "Cassandra Clare", "id" : 4453356543, "price" : 260, "pages" : 290 }
{ "name" : "Who will cry when you die", "id" : 7656445678, "price" : 240, "pages" : 120 }
{ "name" : "An Autobiography of a Yogi", "id" : 8768905678, "author" : "Paramhansa Yogananda", "price" : 390, "pages" : 560 }
{ "name" : "The Subtle art of not giving a *uck", "author" : "Mark Manson", "id" : 2574103698, "price" : 450, "pages" : 230 }
{ "name" : "Who Moved My Cheese?", "author" : "Spencer Johnson", "id" : 9865741520, "price" : 345, "pages" : 180 }
>
```

```
> db.Books.find({pages:{$lt:200}},{_id:0}).pretty();
{
        "name" : "Who will cry when you die",
        "id" : 7656445678,
        "price" : 240,
        "pages" : 120
}
{
        "name" : "Who Moved My Cheese?",
        "author" : "Spencer Johnson",
        "id" : 9865741520,
        "price" : 345,
        "pages" : 180
```

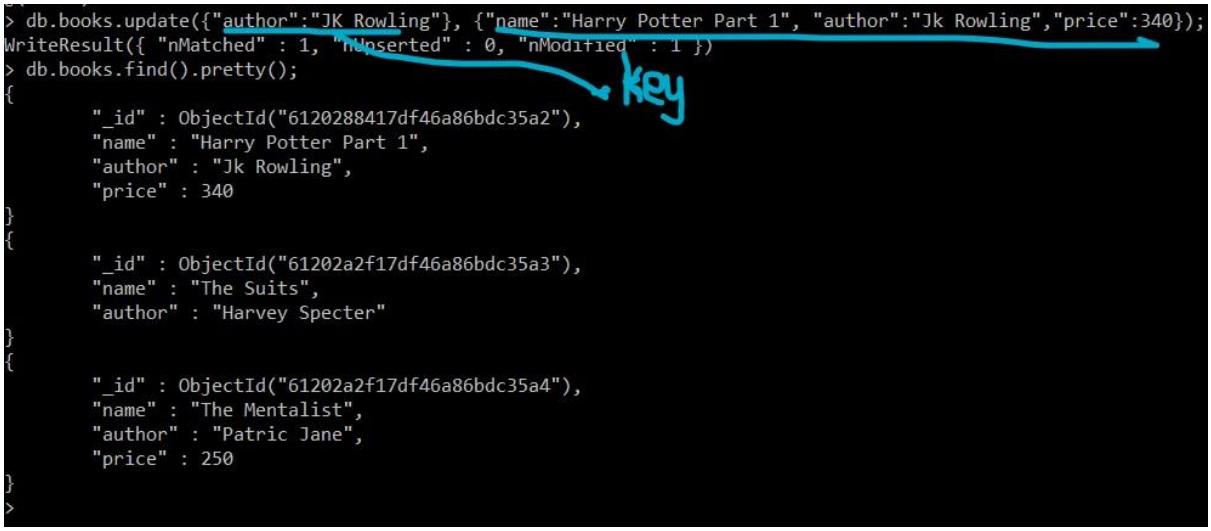
```
> db.Books.find({pages:{$gt:200}},{_id:0}).pretty();
{
        "name" : "Harry Potter",
        "author" : "JK Rowling",
        "id" : 123456789,
        "price" : 490,
        "pages" : 370
}
{
        "name" : "The mortal instruments",
        "author" : "Cassandra Clare",
        "id" : 4453356543,
        "price" : 260,
        "pages" : 290
}
{
        "name" : "An Autobiography of a Yogi",
        "id" : 8768905678,
        "author" : "Paramhansa Yogananda",
        "price" : 390,
        "pages" : 560
}
{
        "name" : "The Subtle art of not giving a *uck",
        "author" : "Mark Manson",
        "id" : 2574103698,
        "price" : 450,
        "pages" : 230
```

## // Sorting




```
> db.Books.aggregate([{$sort:{pages:1}}])
{ "_id" : ObjectId("611fa8bffc6e6eecbe413422"), "name" : "Who will cry when you die", "id" : 7656445678, "price" : 240, "pages" : 120 }
{ "_id" : ObjectId("611fa8bffc6e6eecbe413425"), "name" : "Who Moved My Cheese?", "author" : "Spencer Johnson", "id" : 9865741520, "price" : 345, "pages" : 180 }
{ "_id" : ObjectId("611fa8bffc6e6eecbe413424"), "name" : "The Subtle art of not giving a *uck", "author" : "Mark Manson", "id" : 2574103698, "price" : 450, "pages" : 230 }
{ "_id" : ObjectId("611fa8bffc6e6eecbe413421"), "name" : "The mortal instruments", "author" : "Cassandra Clare", "id" : 4453356543, "price" : 260, "pages" : 290 }
{ "_id" : ObjectId("611fa8bffc6e6eecbe413420"), "name" : "Harry Potter", "author" : "JK Rowling", "id" : 123456789, "price" : 490, "pages" : 370 }
{ "_id" : ObjectId("611fa8bffc6e6eecbe413423"), "name" : "An Autobiography of a Yogi", "id" : 8768905678, "author" : "Paramhansa Yogananda", "price" : 390, "pages" : 560 }
>
```

// Updating by using “author” as key, and changing the name as well as adding a new field of “price” to it.




```
> db.books.update({"author":"JK Rowling"}, {"name":"Harry Potter Part 1", "author":"Jk Rowling","price":340});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.books.find().pretty();
{
        "_id" : ObjectId("6120288417df46a86bdc35a2"),
        "name" : "Harry Potter Part 1",
        "author" : "Jk Rowling",
        "price" : 340
}
{
        "_id" : ObjectId("61202a2f17df46a86bdc35a3"),
        "name" : "The Suits",
        "author" : "Harvey Specter"
}
{
        "_id" : ObjectId("61202a2f17df46a86bdc35a4"),
        "name" : "The Mentalist",
        "author" : "Patric Jane",
        "price" : 250
}
>
```

// Using $set to keep whatever was there previously and adding new field

```
> db.books.update({"name":"The Suits"}, {$set:{"price": 120}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.books.find().pretty();
{
        "_id" : ObjectId("6120288417df46a86bdc35a2"),
        "name" : "Harry Potter Part 1",
        "author" : "Jk Rowling",
        "price" : 340
}
{
        "_id" : ObjectId("61202a2f17df46a86bdc35a3"),
        "name" : "The Suits",
        "author" : "Harvey Specter",
        "price" : 120
}
{
        "_id" : ObjectId("61202a2f17df46a86bdc35a4"),
        "name" : "The Mentalist",
        "author" : "Patric Jane",
        "price" : 250
}
```

// Increameting price by 500

```
> db.books.update({"name":"The Suits"}, {$inc:{price: 500}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.books.find().pretty();
{
        "_id" : ObjectId("6120288417df46a86bdc35a2"),
        "name" : "Harry Potter Part 1",
        "author" : "Jk Rowling",
        "price" : 340
}
{
        "_id" : ObjectId("61202a2f17df46a86bdc35a3"),
        "name" : "The Suits",
        "author" : "Harvey Specter",
        "price" : 620
}
{
        "_id" : ObjectId("61202a2f17df46a86bdc35a4"),
        "name" : "The Mentalist",
        "author" : "Patric Jane",
        "price" : 250
}
```

// Using unset to remove a field

```
> db.books.update({"name":"The Suits"}, {$unset:{price:1}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.books.find().pretty();
{
        "_id" : ObjectId("6120288417df46a86bdc35a2"),
        "name" : "Harry Potter Part 1",
        "author" : "Jk Rowling",
        "price" : 340
}
{
        "_id" : ObjectId("61202a2f17df46a86bdc35a3"),
        "name" : "The Suits",
        "author" : "Harvey Specter"
}
{
        "_id" : ObjectId("61202a2f17df46a86bdc35a4"),
        "name" : "The Mentalist",
        "author" : "Patric Jane",
        "price" : 250
}
```

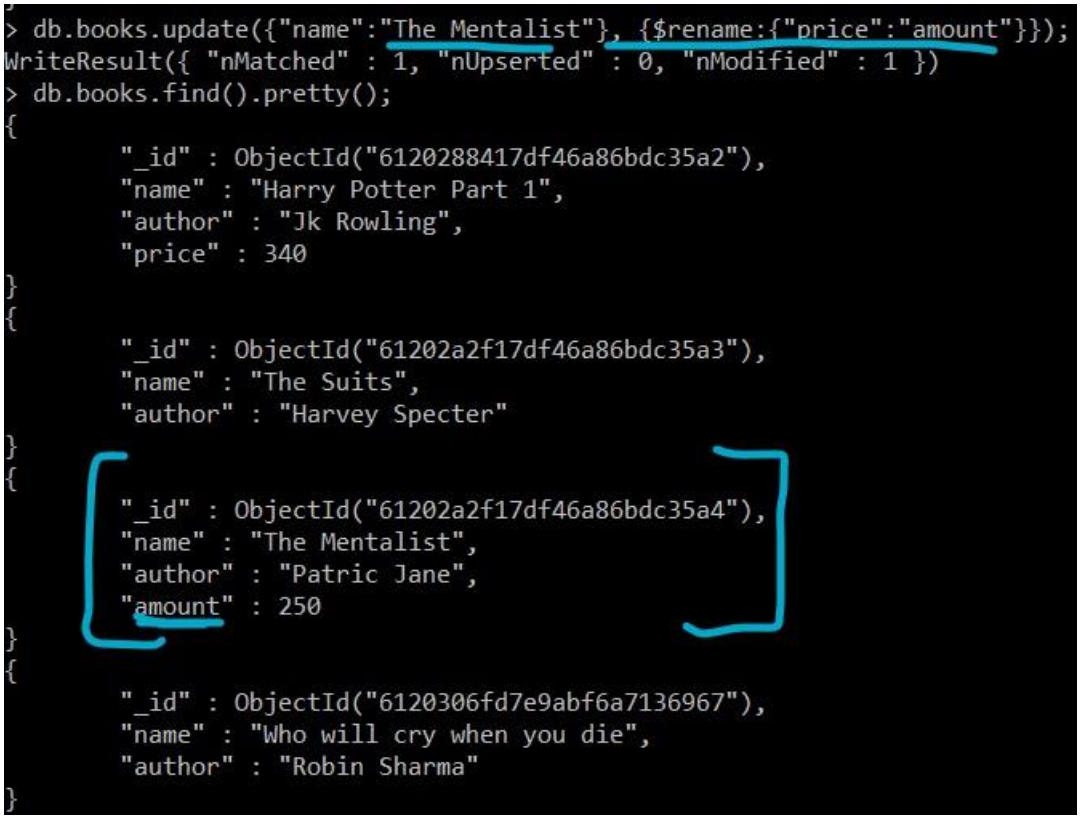
// Searching for an entry that is not there – We don't ger any error nor is the record created

```
> db.books.update({name:"Who will cry when you die"}, {name:"Who will cry when you die","author":"Robin Sharma"});
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
> db.books.find().pretty();
{
        "_id" : ObjectId("6120288417df46a86bdc35a2"),
        "name" : "Harry Potter Part 1",
        "author" : "Jk Rowling",
        "price" : 340
}
{
        "_id" : ObjectId("61202a2f17df46a86bdc35a3"),
        "name" : "The Suits",
        "author" : "Harvey Specter"
}
{
        "_id" : ObjectId("61202a2f17df46a86bdc35a4"),
        "name" : "The Mentalist",
        "author" : "Patric Jane",
        "price" : 250
}
>
```

// Creating a new entry if not found in the collection by setting upsert to true

```
> db.books.update({name:"Who will cry when you die"}, {name:"Who will cry when you die","author":"Robin Sharma"}, {upsert:true});
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 1,
        "nModified" : 0,
        "_id" : ObjectId("6120306fd7e9abf6a7136967")
})
> db.books.find().pretty();
{
        "_id" : ObjectId("6120288417df46a86bdc35a2"),
        "name" : "Harry Potter Part 1",
        "author" : "Jk Rowling",
        "price" : 340
}
{
        "_id" : ObjectId("61202a2f17df46a86bdc35a3"),
        "name" : "The Suits",
        "author" : "Harvey Specter"
}
{
        "_id" : ObjectId("61202a2f17df46a86bdc35a4"),
        "name" : "The Mentalist",
        "author" : "Patric Jane",
        "price" : 250
}
{
        "_id" : ObjectId("6120306fd7e9abf6a7136967"),
        "name" : "Who will cry when you die",
        "author" : "Robin Sharma"
}
```

// Changing “price” to “amount” using rename



```
> db.books.update({"name":"The Mentalist"}, {$rename:{"price":"amount"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.books.find().pretty();
{
        "_id" : ObjectId("6120288417df46a86bdc35a2"),
        "name" : "Harry Potter Part 1",
        "author" : "Jk Rowling",
        "price" : 340
}
{
        "_id" : ObjectId("61202a2f17df46a86bdc35a3"),
        "name" : "The Suits",
        "author" : "Harvey Specter"
}
{
        "_id" : ObjectId("61202a2f17df46a86bdc35a4"),
        "name" : "The Mentalist",
        "author" : "Patric Jane",
        "amount" : 250
}
{
        "_id" : ObjectId("6120306fd7e9abf6a7136967"),
        "name" : "Who will cry when you die",
        "author" : "Robin Sharma"
}
```

// Removing a particular record

```
> db.books.remove({"name":"The Suits"});
WriteResult({ "nRemoved" : 1 })
> db.books.find().pretty();
{
        "_id" : ObjectId("6120288417df46a86bdc35a2"),
        "name" : "Harry Potter Part 1",
        "author" : "Jk Rowling",
        "price" : 340
}
{
        "_id" : ObjectId("61202a2f17df46a86bdc35a4"),
        "name" : "The Mentalist",
        "author" : "Patric Jane",
        "amount" : 250
}
{
        "_id" : ObjectId("6120306fd7e9abf6a7136967"),
        "name" : "Who will cry when you die",
        "author" : "Robin Sharma"
}
>
```

//Removing only one record if there are multiple records with same name

```
> db.books.remove({"name":"The Suits"},{justOne:true});
WriteResult({ "nRemoved" : 0 })
```

## // Adding nested records using arrays

```
> db.inventory.insertOne(
...     { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } }
... )
{
        "acknowledged" : true,
        "insertedId" : ObjectId("61203c3617df46a86bdc35a6")
}
> db.inventory.find().pretty();
{
        "_id" : ObjectId("6120393c17df46a86bdc35a5"),
        "item" : "canvas",
        "qty" : 100,
        "tags" : [
                "cotton"
        ],
        "size" : {
                "h" : 28,
                "w" : 35.5,
                "uom" : "cm"
        }
}
{
        "_id" : ObjectId("61203c3617df46a86bdc35a6"),
        "item" : "canvas",
        "qty" : 100,
        "tags" : [
                "cotton"
        ],
        "size" : {
                "h" : 28,
                "w" : 35.5,
                "uom" : "cm"
        }
}
>
```

## // Adding many records with nested arrays

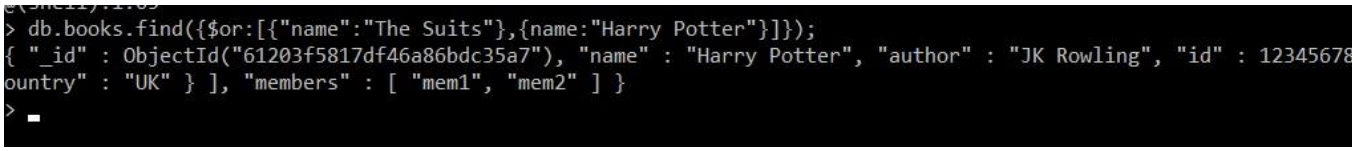
```
{
        "_id" : ObjectId("61203f5817df46a86bdc35a7"),
        "name" : "Harry Potter",
        "author" : "JK Rowling",
        "id" : 123456789,
        "price" : 490,
        "pages" : 370,
        "address" : [
                {
                        "street" : "Bakers Street",
                        "city" : "London",
                        "country" : "UK"
                }
        ],
        "members" : [
                "mem1",
                "mem2"
        ]
}
```

```
> db.books.find().pretty();
{
        "_id" : ObjectId("6120288417df46a86bdc35a2"),
        "name" : "Harry Potter Part 1",
        "author" : "Jk Rowling",
        "price" : 340
}
{
        "_id" : ObjectId("61202a2f17df46a86bdc35a4"),
        "name" : "The Mentalist",
        "author" : "Patric Jane",
        "amount" : 250
}
{
        "_id" : ObjectId("6120306fd7e9abf6a7136967"),
        "name" : "Who will cry when you die",
        "author" : "Robin Sharma"
}
{
        "_id" : ObjectId("61203f5817df46a86bdc35a7"),
        "name" : "Harry Potter",
        "author" : "JK Rowling",
        "id" : 123456789,
        "price" : 490,
        "pages" : 370,
        "address" : [
                {
                        "street" : "Bakers Street",
                        "city" : "London",
                        "country" : "UK"
                }
        ],
        "members" : [
                "mem1",
                "mem2"
```

```
        ]
}
{
        "_id" : ObjectId("61203f5817df46a86bdc35a8"),
        "name" : "The mortal instruments",
        "author" : "Cassandra Clare",
        "id" : 4453356543,
        "price" : 260,
        "pages" : 290,
        "address" : [
                {
                        "street" : "Times Square",
                        "city" : "New York",
                        "country" : "USA"
                }
        ],
        "members" : [
                "mem1",
                "mem2",
                "mem3"
        ]
}
{
        "_id" : ObjectId("61203f5817df46a86bdc35a9"),
        "name" : "Who will cry when you die",
        "id" : 7656445678,
        "price" : 240,
        "pages" : 120
}
{
        "_id" : ObjectId("61203f5817df46a86bdc35aa"),
        "name" : "An Autobiography of a Yogi",
        "id" : 8768905678,
        "author" : "Paramhansa Yogananda",
        "price" : 390,
        "pages" : 560
}
```

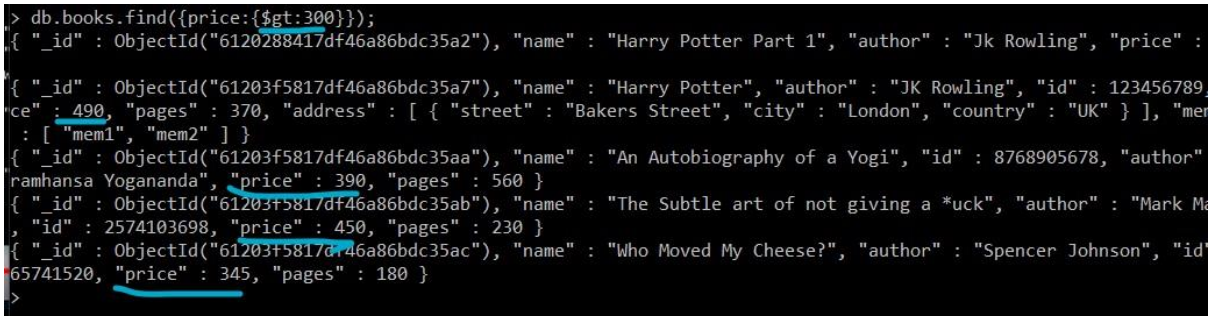
```
{
        "_id" : ObjectId("61203f5817df46a86bdc35ab"),
        "name" : "The Subtle art of not giving a *uck",
        "author" : "Mark Manson",
        "id" : 2574103698,
        "price" : 450,
        "pages" : 230
}
{
        "_id" : ObjectId("61203f5817df46a86bdc35ac"),
        "name" : "Who Moved My Cheese?",
        "author" : "Spencer Johnson",
        "id" : 9865741520,
        "price" : 345,
        "pages" : 180
}
>
```

// $or operator

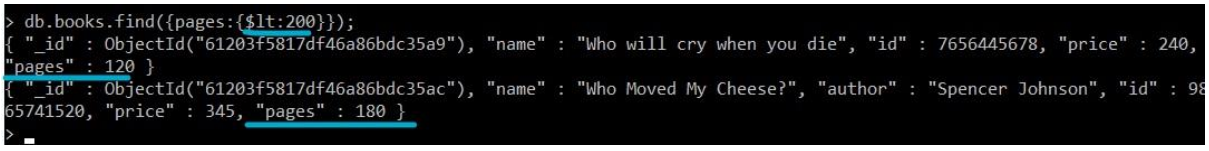



```
> db.books.find({$or:[{"name":"The Suits"},{name:"Harry Potter"}]});
{ "_id" : ObjectId("61203f5817df46a86bdc35a7"), "name" : "Harry Potter", "author" : "JK Rowling", "id" : 123456789, "price" : 490, "pages" : 370, "address" : [ { "street" : "Bakers Street", "city" : "London", "country" : "UK" } ], "members" : [ "mem1", "mem2" ] }
>
```

// Getting book with price greater than 300

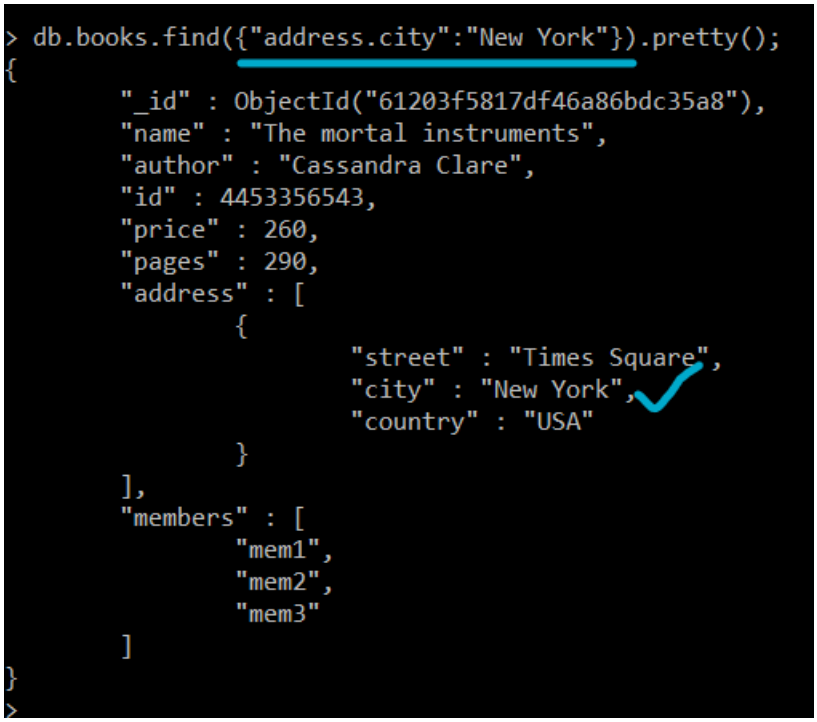



// Getting book with less than 200 pages

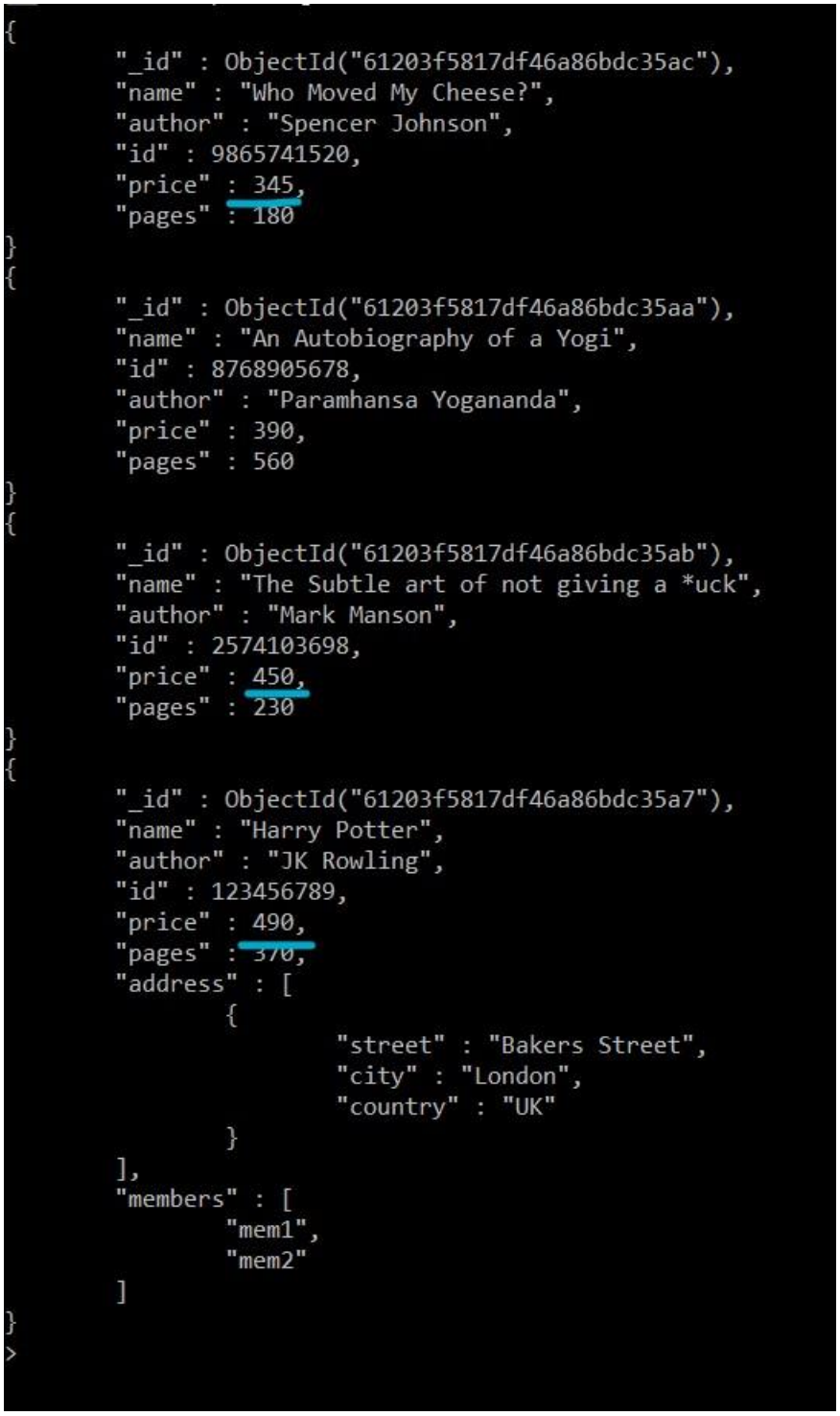



> db.books.find({pages:{$lt:200}});
{ "_id" : ObjectId("61203f5817df46a86bdc35a9"), "name" : "Who will cry when you die", "id" : 7656445678, "price" : 240, "pages" : 120 }
{ "_id" : ObjectId("61203f5817df46a86bdc35ac"), "name" : "Who Moved My Cheese?", "author" : "Spencer Johnson", "id" : 9865741520, "price" : 345, "pages" : 180 }
>

// Finding records inside a nested record(row)


> db.books.find({"address.city":"New York"}).pretty();
{
"_id" : ObjectId("61203f5817df46a86bdc35a8"),
"name" : "The mortal instruments",
"author" : "Cassandra Clare",
"id" : 4453356543,
"price" : 260,
"pages" : 290,
"address" : [
{
"street" : "Times Square",
"city" : "New York",
"country" : "USA"
}
],
"members" : [
"mem1",
"mem2",
"mem3"
]
}
>

// Sorting all the books according to price(ascending)



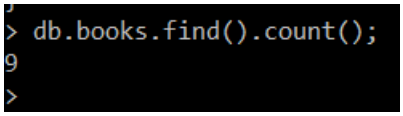

```
> db.books.find().sort({price:1}).pretty();
{
	"_id" : ObjectId("61202a2f17df46a86bdc35a4"),
	"name" : "The Mentalist",
	"author" : "Patric Jane",
	"amount" : 250
}
{
```

```
        "_id" : ObjectId("6120306fd7e9abf6a7136967"),
        "name" : "Who will cry when you die",
        "author" : "Robin Sharma"
}
{
        "_id" : ObjectId("61203f5817df46a86bdc35a9"),
        "name" : "Who will cry when you die",
        "id" : 7656445678,
        "price" : 240,
        "pages" : 120
}
{
        "_id" : ObjectId("61203f5817df46a86bdc35a8"),
        "name" : "The mortal instruments",
        "author" : "Cassandra Clare",
        "id" : 4453356543,
        "price" : 260,
        "pages" : 290,
        "address" : [
                {
                        "street" : "Times Square",
                        "city" : "New York",
                        "country" : "USA"
                }
        ],
        "members" : [
                "mem1",
                "mem2",
                "mem3"
        ]
}
{
        "_id" : ObjectId("6120288417df46a86bdc35a2"),
        "name" : "Harry Potter Part 1",
        "author" : "Jk Rowling",
        "price" : 340
}
```

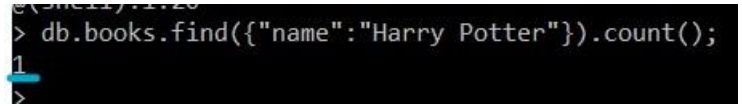
```
{
	"_id" : ObjectId("61203f5817df46a86bdc35ac"),
	"name" : "Who Moved My Cheese?",
	"author" : "Spencer Johnson",
	"id" : 9865741520,
	"price" : 345,
	"pages" : 180
}
{
	"_id" : ObjectId("61203f5817df46a86bdc35aa"),
	"name" : "An Autobiography of a Yogi",
	"id" : 8768905678,
	"author" : "Paramhansa Yogananda",
	"price" : 390,
	"pages" : 560
}
{
	"_id" : ObjectId("61203f5817df46a86bdc35ab"),
	"name" : "The Subtle art of not giving a *uck",
	"author" : "Mark Manson",
	"id" : 2574103698,
	"price" : 450,
	"pages" : 230
}
{
	"_id" : ObjectId("61203f5817df46a86bdc35a7"),
	"name" : "Harry Potter",
	"author" : "JK Rowling",
	"id" : 123456789,
	"price" : 490,
	"pages" : 370,
	"address" : [
		{
			"street" : "Bakers Street",
			"city" : "London",
			"country" : "UK"
		}
```

```
        ],
        "members" : [
                "mem1",
                "mem2"
        ]
}
>
```

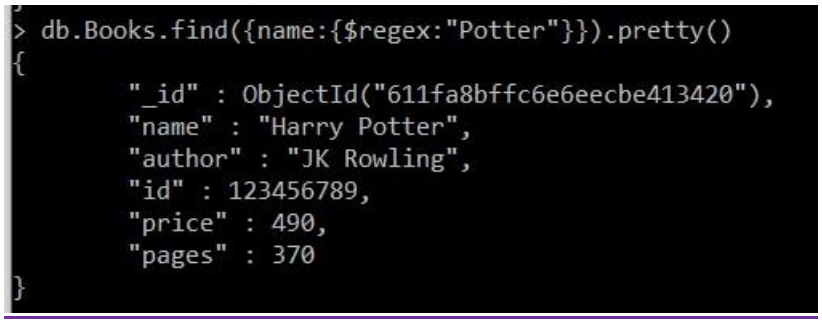
// Using count() function




// Counting occurrence of a specific record




// Using regex to find the word “Potter”



## // Using validators

```
> db.createCollection("Books1", {
...    validator: {
...       $jsonSchema: {
...          bsonType: "object",
...          required: [ "name", "author", "pages", "price" ],
...          properties: {
...             name: {
...                bsonType: "string",
...                description: "must be a string and is required"
...             },
...             pages: {
...                bsonType: "int",
...                minimum: 50,
...                maximum: 1000,
...                description: "must be an integer in [ 50, 1000 ] and is required"
...             },
...             price: {
...               bsonType: "int",
...               minimum: 0,
...               description: "Required and should be greater than 0"
...             },
...          }
...       }
...    }
... });
{ "ok" : 1 }
```

```
> db.createCollection("Books1", {
...   validator: {
...     $jsonSchema: {
...       bsonType: "object",
...       required: [ "name", "author", "pages", "price" ],
...       properties: {
...         name: {
...           bsonType: "string",
...           description: "must be a string and is required"
...         },
...         pages: {
...           bsonType: "int",
...           minimum: 50,
...           maximum: 1000,
...           description: "must be an integer in [ 50, 1000 ] and is required"
...         },
...         price: {
...           bsonType: "int",
...           minimum: 0,
...           description: "Required and should be greater than 0"
...         },
...       }
...     }
...   }
```

... });
{ "ok" : 1 }

## // Using push, pop for arrays:

## BEFORE push:


## AFTER push:

## Conclusion:

- Commands for insert, saving removing document, updating document were executed successfully.
- Queries were executed on a “book” database such as find, find one, conditions, OR, $ not, conditional semantics, Type-specific queries (Null, Regular expression, Querying arrays), $ where, cursors (Limit, skip, sort, advanced query options), etc.

**PUNE INSTITUTE OF COMPUTER TECHNOLOGY**

**Subject: ADBMS (LP LAB)**

**Name: Aditya Kangune** **Roll No. : 33323**

**Batch: K11** **Academic Year: 2021-22**

---

## Assignment 2
## Map reduce and aggregate

---

### Aim:

Implement Map reduces operation with suitable example on above MongoDB database and implement the following:

- Aggregation framework
- Create and drop different types of indexes and explain () to show the advantage of the indexes.

### Objective:

- To understand the concept of Map-reduce in mongodb.
- To understand the concept of Aggregation in mongodb.
- To implement the concept of document-oriented databases.

## Theory:

### Map-reduce:

1. MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.
2. A MapReduce program is composed of a map procedure, which performs filtering and sorting, and a reduce method, which performs a summary operation.
3. To perform map-reduce operations, MongoDB provides the mapReduce database command.
4. MapReduce functions are written in JavaScript.
5. Uses a "pipeline" approach where objects are transformed as they pass through a series of pipeline operators such as matching, projecting, sorting, and grouping. Pipeline operators need not produce one output document for every input document: operators may also generate new documents or filter out documents.

Map/Reduce involves two steps:

- First, map the data from the collection specified;
- Second, reduce the results.

### Map Function:

- var mapFunction1 = function()
- { emit(this.cust_id, this.amount);};

## Reduce Function:

- var reduceFunction1 = function(key, values)
- {return Array.sum(values); };

## Aggregation:

Aggregation operations process multiple documents and return computed results. We can use aggregation operations to:

- Group values from multiple documents together.
- Perform operations on the grouped data to return a single result.
- Analyse data changes over time.

## Indexing:

Indexes are special data structures that store a small portion of the collection's data set in an easy to traverse form.

The index stores the value of a specific field or set of fields, ordered by the value of the field.

The ordering of the index entries supports efficient equality matches and range-based query operations.

In addition, MongoDB can return sorted results by using the ordering in the index.

**Output:**

**Current DB**

```
db.Employee.find().pretty();
{
        "_id" : ObjectId("611f52deaf8bf9764e431af4"),
        "empName" : "Suresh",
        "uan" : 54356212,
        "dept" : {
                "location" : 412,
                "name" : "IT"
        },
        "designation" : "Teacher",
        "emailID" : "abc@gmail.com",
        "salary" : 20000
}
{
        "_id" : ObjectId("611f52f7af8bf9764e431af5"),
        "empName" : "Ramesh",
        "uan" : 34554222,
        "dept" : {
                "location" : 522,
```

```
            "name" : "IT"
        },
        "designation" : "Teacher",
        "emailID" : "cde@gmail.com",
        "salary" : 60000
}
{
        "_id" : ObjectId("611f59cfaf8bf9764e431af6"),
        "empName" : "Mangesh",
        "uan" : 362926328,
        "dept" : {
            "location" : 301,
            "name" : "IT"
        },
        "designation" : "HOD",
        "emailID" : "abcde@gmail.com",
        "salary" : 10000
}
{
        "_id" : ObjectId("611f657c76ba0a7a09d22ef5"),
        "empName" : "Rangesh",
        "uan" : 362926328,
```

```
        "dept" : {
                "location" : 513,
                "name" : "IT"
        },
        "designation" : "HOD",
        "emailID" : "abcdef@gmail.com",
        "salary" : 50000
}
{
        "_id" : ObjectId("611f659976ba0a7a09d22ef6"),
        "empName" : "Tungesh",
        "uan" : 362900328,
        "dept" : {
                "location" : 618,
                "name" : "Comp"
        },
        "designation" : "Teacher",
        "emailID" : "abcdefg@gmail.com",
        "salary" : 60000
}
{
        "_id" : ObjectId("611f65b676ba0a7a09d22ef7"),
```

```
        "empName" : "Sangesh",
        "uan" : 452900328,
        "dept" : {
                "location" : 102,
                "name" : "Comp"
        },
        "designation" : "Teacher",
        "emailID" : "abcdefgh@gmail.com",
        "salary" : 70000
}
{
        "_id" : ObjectId("611f65d176ba0a7a09d22ef8"),
        "empName" : "Vangesh",
        "uan" : 452900833,
        "dept" : {
                "location" : 213,
                "name" : "ENTC"
        },
        "designation" : "Teacher",
        "emailID" : "abcdefghi@gmail.com",
        "salary" : 80000
}
```

```
{
        "_id" : ObjectId("611f65e576ba0a7a09d22ef9"),
        "empName" : "Zangesh",
        "uan" : 722900833,
        "dept" : {
                "location" : 503,
                "name" : "ENTC"
        },
        "designation" : "Teacher",
        "emailID" : "abcdefghij@gmail.com",
        "salary" : 90000
}
{
        "_id" : ObjectId("611f65f876ba0a7a09d22efa"),
        "empName" : "Dangesh",
        "uan" : 7229002133,
        "dept" : {
                "location" : 321,
                "name" : "ENTC"
        },
        "designation" : "Teacher",
        "emailID" : "xyz@gmail.com",
```

```
        "salary" : 100000
}
{
        "_id" : ObjectId("611f662076ba0a7a09d22efb"),
        "empName" : "Mohan",
        "salary" : 140000,
        "uan" : 123441121,
        "dept" : {
                "location" : 542,
                "name" : "comp"
        },
        "designation" : "Teacher",
        "emailID" : "xyz@gmail.com",
        "teams" : [
                "technical"
        ]
}
{
        "_id" : ObjectId("61260b295e64af69224b448a"),
        "empName" : "Rohan",
        "salary" : 120000,
        "uan" : 123441121,
```

```
        "dept" : {
                "location" : 542,
                "name" : "comp"
        },
        "designation" : "Teacher",
        "emailID" : "abcd@gmail.com"
}
```


**Aggregate**

```
db.Employee.aggregate([{$group:{_id:"$dept"}}]);
{ "_id" : { "location" : 321, "name" : "ENTC" } }
{ "_id" : { "location" : 503, "name" : "ENTC" } }
{ "_id" : { "location" : 213, "name" : "ENTC" } }
{ "_id" : { "location" : 102, "name" : "Comp" } }
{ "_id" : { "location" : 618, "name" : "Comp" } }
{ "_id" : { "location" : 542, "name" : "comp" } }
{ "_id" : { "location" : 513, "name" : "IT" } }
{ "_id" : { "location" : 301, "name" : "IT" } }
{ "_id" : { "location" : 522, "name" : "IT" } }
{ "_id" : { "location" : 412, "name" : "IT" } }
```

```
> db.Employee.aggregate([{$group:{_id:"$dept.name"}}]);
{ "_id" : "comp" }
{ "_id" : "ENTC" }
{ "_id" : "Comp" }
{ "_id" : "IT" }

> db.Employee.aggregate([{$group:{_id:"$dept",totalSalary:{$sum:"$salary"}}}]);
{ "_id" : { "location" : 321, "name" : "ENTC" }, "totalSalary" : 100000 }
{ "_id" : { "location" : 503, "name" : "ENTC" }, "totalSalary" : 90000 }
{ "_id" : { "location" : 213, "name" : "ENTC" }, "totalSalary" : 80000 }
{ "_id" : { "location" : 102, "name" : "Comp" }, "totalSalary" : 70000 }
{ "_id" : { "location" : 618, "name" : "Comp" }, "totalSalary" : 60000 }
{ "_id" : { "location" : 542, "name" : "comp" }, "totalSalary" : 260000 }
{ "_id" : { "location" : 513, "name" : "IT" }, "totalSalary" : 50000 }
{ "_id" : { "location" : 301, "name" : "IT" }, "totalSalary" : 10000 }
{ "_id" : { "location" : 522, "name" : "IT" }, "totalSalary" : 60000 }
{ "_id" : { "location" : 412, "name" : "IT" }, "totalSalary" : 20000 }
```

>
db.Employee.aggregate([{$group:{_id:"$dept.name",totalSalary:{$sum:"$salary
"}}},{$match:{totalSalary:{$gte:150000}}}]);

{ "_id" : "comp", "totalSalary" : 260000 }

{ "_id" : "ENTC", "totalSalary" : 270000 }

>
db.Employee.aggregate([{$group:{_id:"$dept.name",totalSalary:{$sum:"$salary
"}}},{$match:{totalSalary:{$lte:150000}}}]);

{ "_id" : "Comp", "totalSalary" : 130000 }

{ "_id" : "IT", "totalSalary" : 140000 }

>
db.Employee.aggregate([{$group:{_id:"$dept.name",totalSalary:{$sum:"$salary
"}}},{$sort:{totalSalary:1}}]);

{ "_id" : "Comp", "totalSalary" : 130000 }

{ "_id" : "IT", "totalSalary" : 140000 }

{ "_id" : "comp", "totalSalary" : 260000 }

{ "_id" : "ENTC", "totalSalary" : 270000 }

>
db.Employee.aggregate([{$group:{_id:"$dept.name",totalSalary:{$sum:"$salary
"}}},{$sort:{totalSalary:-1}}]);

{ "_id" : "ENTC", "totalSalary" : 270000 }

{ "_id" : "comp", "totalSalary" : 260000 }

{ "_id" : "IT", "totalSalary" : 140000 }

{ "_id" : "Comp", "totalSalary" : 130000 }

>
db.Employee.aggregate([{$group:{_id:"$dept.name",avgSalary:{$avg:"$salary"}}
},{$sort:{totalSalary:-1}}]);

{ "_id" : "comp", "avgSalary" : 130000 }

{ "_id" : "ENTC", "avgSalary" : 90000 }

{ "_id" : "Comp", "avgSalary" : 65000 }

{ "_id" : "IT", "avgSalary" : 35000 }

>
db.Employee.aggregate([{$group:{_id:"$dept.name",avgSalary:{$avg:"$salary"}}
},{$limit:2}]);

{ "_id" : "comp", "avgSalary" : 130000 }

{ "_id" : "ENTC", "avgSalary" : 90000 }

>
db.Employee.aggregate([{$group:{_id:"$dept.name",avgSalary:{$avg:"$salary"}}
},{$limit:2},{$group:{_id:"$_id.dept.name",avgOfAll:{$avg:"$avgSalary"}}}]);

{ "_id" : null, "avgOfAll" : 110000 }

```
db.Employee.aggregate([{$group:{_id:"$dept.name",maxSalary:{$max:"$salary"
}}}]);

{ "_id" : "comp", "maxSalary" : 140000 }

{ "_id" : "ENTC", "maxSalary" : 100000 }

{ "_id" : "Comp", "maxSalary" : 70000 }

{ "_id" : "IT", "maxSalary" : 60000 }
```

```
db.Employee.aggregate([{$match:{salary:{$lt:100000}}},{$group:{_id:"$dept.na
me",maxSalary:{$max:"$salary"}}}]);

{ "_id" : "ENTC", "maxSalary" : 90000 }

{ "_id" : "Comp", "maxSalary" : 70000 }

{ "_id" : "IT", "maxSalary" : 60000 }
```

**Map Reduce:**

```
> var map1=function(){emit(this.empName,this.salary);};

> var reduce1=function(key,values){return Array.sum(values);};


db.Employee.mapReduce(map1,reduce1,{out:"total_salary"});

{
        "result" : "total_salary",

        "timeMillis" : 77,
```

```
        "counts" : {
                "input" : 11,
                "emit" : 11,
                "reduce" : 0,
                "output" : 11
        },
        "ok" : 1
}
> db.total_salary.find().pretty();
{ "_id" : "Dangesh", "value" : 100000 }
{ "_id" : "Mangesh", "value" : 10000 }
{ "_id" : "Mohan", "value" : 140000 }
{ "_id" : "Ramesh", "value" : 60000 }
{ "_id" : "Rangesh", "value" : 50000 }
{ "_id" : "Rohan", "value" : 120000 }
{ "_id" : "Sangesh", "value" : 70000 }
{ "_id" : "Suresh", "value" : 20000 }
{ "_id" : "Tungesh", "value" : 60000 }
{ "_id" : "Vangesh", "value" : 80000 }
{ "_id" : "Zangesh", "value" : 90000 }
>
```

```
db.Employee.mapReduce(map1,reduce1,{query:{salary:{$gte:100000}},out:"total_salary"});

{
        "result" : "total_salary",
        "timeMillis" : 62,
        "counts" : {
                "input" : 3,
                "emit" : 3,
                "reduce" : 0,
                "output" : 3
        },
        "ok" : 1
}

> db.total_salary.find().pretty();

{ "_id" : "Dangesh", "value" : 100000 }

{ "_id" : "Mohan", "value" : 140000 }

{ "_id" : "Rohan", "value" : 120000 }



db.Employee.mapReduce(map1,reduce1,{query:{salary:{$lt:100000}},out:"total_salary"});

{
        "result" : "total_salary",
```

```
        "timeMillis" : 72,
        "counts" : {
                "input" : 8,
                "emit" : 8,
                "reduce" : 0,
                "output" : 8
        },
        "ok" : 1
}
> db.total_salary.find().pretty();
{ "_id" : "Mangesh", "value" : 10000 }
{ "_id" : "Ramesh", "value" : 60000 }
{ "_id" : "Rangesh", "value" : 50000 }
{ "_id" : "Sangesh", "value" : 70000 }
{ "_id" : "Suresh", "value" : 20000 }
{ "_id" : "Tungesh", "value" : 60000 }
{ "_id" : "Vangesh", "value" : 80000 }
{ "_id" : "Zangesh", "value" : 90000 }



> var map1=function(){emit(this.dept.name,this.salary);};
```

```
> var reduce1=function(key,values){return Array.sum(values);};

> db.Employee.mapReduce(map1,reduce1,{query:{salary:{$lt:100000}},out:"total_salary"});

{
        "result" : "total_salary",
        "timeMillis" : 78,
        "counts" : {
                "input" : 8,
                "emit" : 8,
                "reduce" : 3,
                "output" : 3
        },
        "ok" : 1
}

> db.total_salary.find().pretty();

{ "_id" : "Comp", "value" : 130000 }

{ "_id" : "ENTC", "value" : 170000 }

{ "_id" : "IT", "value" : 140000 }
```

**Conclusion:**

- Map reduce operations were implemented on a database.
- The Aggregation framework was executed.
- Different types of indexes were created and dropped.

**PUNE INSTITUTE OF COMPUTER TECHNOLOGY**

**Subject: ADBMS (LP LAB)**

**Name: Aditya Kangune** **Roll No. : 33323**

**Batch: K11** **Academic Year: 2021-22**

---

## Case Study
## Star and Snowflake schema

---

### Aim:

Design conceptual model using Star and Snowflake schema for any one database.

### Theory:

### Multidimensional Schema:

The relations in a data warehouse schema can be classified as fact tables and dimension tables.

### Fact table:

1. Fact tables record information about individual events, such as sales in this case, and are usually very large.
2. The attributes in the fact table can be classified as either dimension attributes or measure attributes.

### Measure Attributes:

The measure attributes store quantitative information, which can be aggregated upon.

### Dimension Attributes:

1. Dimension attributes are dimensions upon which measure attributes, and summaries of measure attributes, are grouped and viewed.
2. To minimize storage requirements, dimension attributes are usually short identifiers that are foreign keys into other tables called dimension tables.

Data that can be modeled using dimension attributes and measure attributes is called multidimensional data.

Following is the Star and snowflake schema for a sales database:

### 1. Star Schema:

1. Star schema includes a fact table at the center with multiple dimension tables.
2. It consists of having foreign keys from the fact table to the dimension tables.
3. Star schema is the simplest data warehouse schema and is optimized for querying large databases.

### Model of Star schema:

1. In Star Schema, Business process data, which holds the quantitative data about a business is distributed in fact tables, and dimensions which are descriptive characteristics related to fact data.

2. Often, A Star Schema having multiple dimensions is termed a Centipede Schema. It is easy to handle a star schema that has dimensions of a few attributes.

## Advantages of star schema:

**Simpler Queries:**

Join logic of star schema is quite cinch in comparison to other join logic which is needed to fetch data from a transactional schema that is highly normalized.

**Simplified Business Reporting Logic:**

In comparison to a transactional schema that is highly normalized, the star schema makes simpler common business reporting logic, such as as-of reporting and period-over-period.

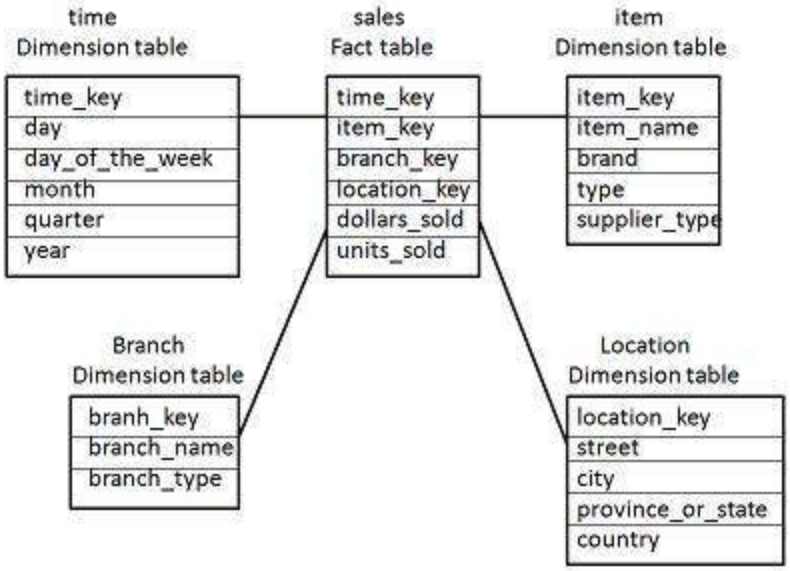
**Feeding Cubes:**

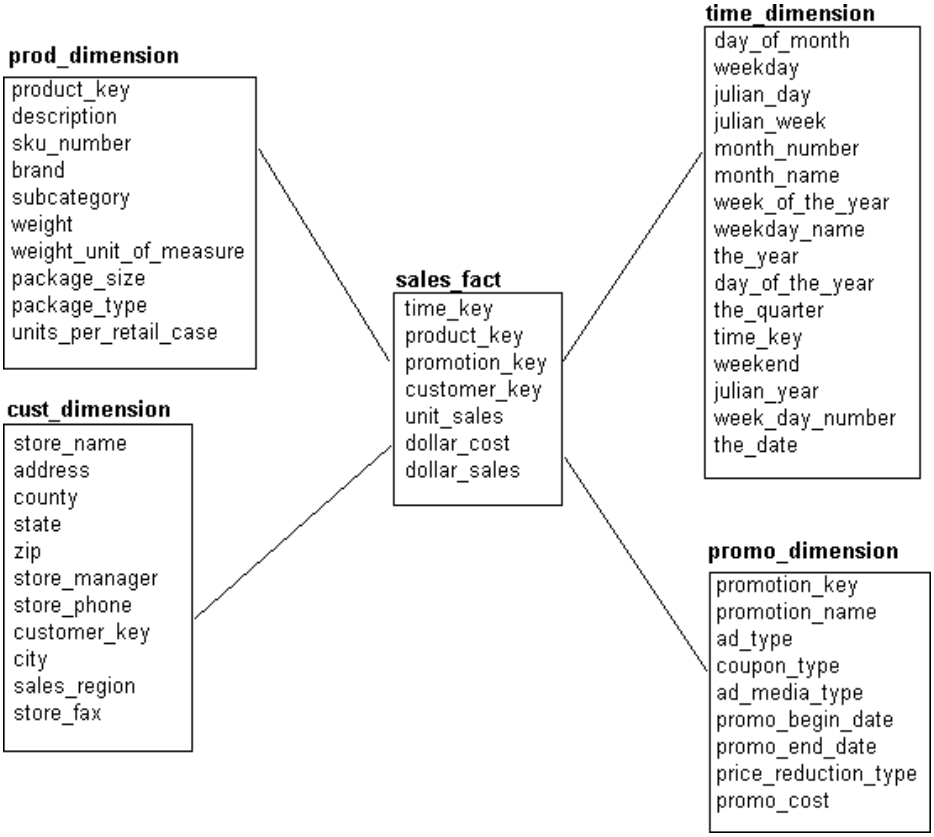
Star schema is widely used by all OLAP systems to design OLAP cubes efficiently. In fact, major OLAP systems deliver a ROLAP mode of operation which can use a star schema as a source without designing a cube structure.

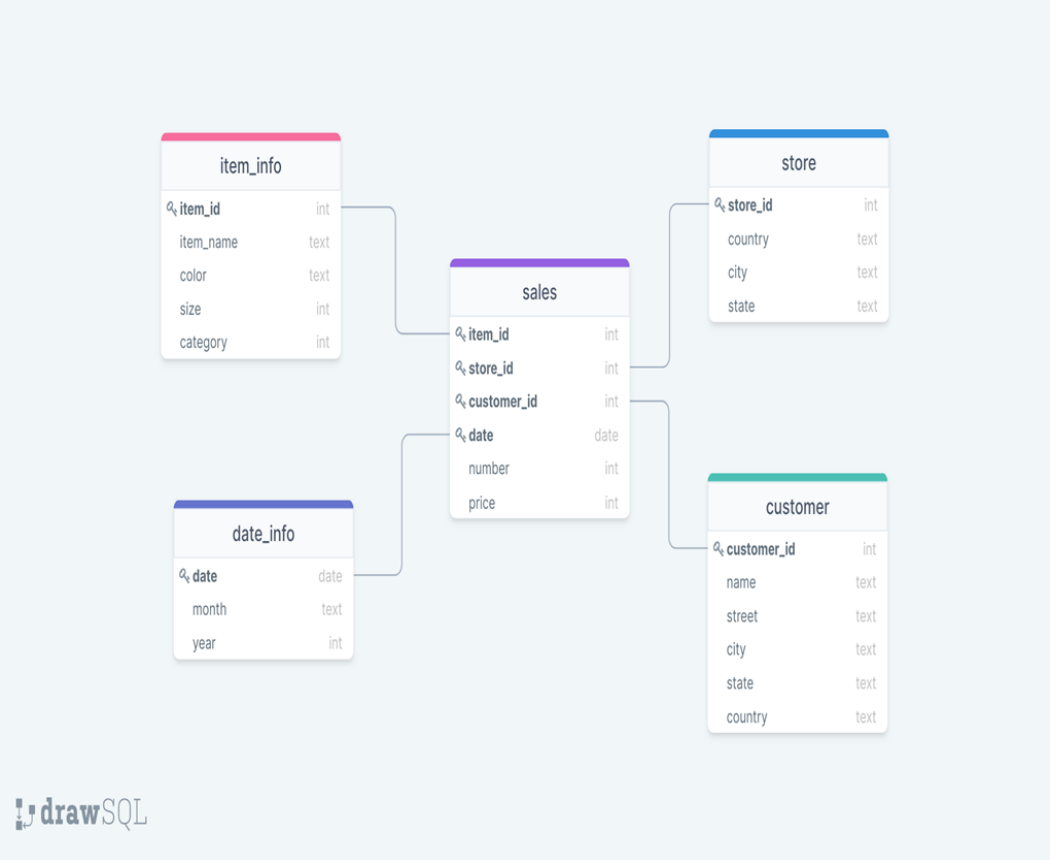
## Disadvantages of star schema:

1. **Data integrity** is not enforced well since in a highly de-normalized schema state.
2. **Not flexible** in terms if analytical needs as a normalized data model.
3. Star schemas don't **reinforce many-to-many relationships** within business entities – at least not frequently.

## Star schema examples:


time
Dimension table
time_key
day
day_of_the_week
month
quarter
year
sales
Fact table
time_key
item_key
branch_key
location_key
dollars_sold
units_sold
item
Dimension table
item_key
item_name
brand
type
supplier_type
Branch
Dimension table
branh_key
branch_name
branch_type
Location
Dimension table
location_key
street
city
province_or_state
country


prod_dimension
product_key
description
sku_number
brand
subcategory
weight
weight_unit_of_measure
package_size
package_type
units_per_retail_case
cust_dimension
store_name
address
county
state
zip
store_manager
store_phone
customer_key
city
sales_region
store_fax
sales_fact
time_key
product_key
promotion_key
customer_key
unit_sales
dollar_cost
dollar_sales
time_dimension
day_of_month
weekday
julian_day
julian_week
month_number
month_name
week_of_the_year
weekday_name
the_year
day_of_the_year
the_quarter
time_key
weekend
julian_year
week_day_number
the_date
promo_dimension
promotion_key
promotion_name
ad_type
coupon_type
ad_media_type
promo_begin_date
promo_end_date
price_reduction_type
promo_cost




## 2 . Snowflake Schema:

1. Snowflake Schema is an extension of the star schema.
2. A snowflake schema is used to model complex data warehouse designs that have multiple levels of dimensions.
3. Snowflake schema uses smaller disk space, but multiple dimension tables reduce the query performance.
4. The snowflake schema is a variant of the star schema.
5. Here, the centralized fact table is connected to multiple dimensions. In the snowflake schema, dimensions are present in a normalized form in multiple related tables.
6. The snowflake structure materialized when the dimensions of a star schema are detailed and highly structured, having several levels of relationship, and the child tables have multiple parent tables.
7. The snowflake effect affects only the dimension tables and does not affect the fact tables.

## What is snowflaking?

1. The snowflake design is the result of further expansion and normalized of the dimension table. In other words, a dimension table is said to be snowflaked if the low-cardinality attribute of the dimensions has been divided into separate normalized tables.
2. These tables are then joined to the original dimension table with referential constraints (foreign key constrain).
3. Generally, snowflaking is not recommended in the dimension table, as it hampers the understandability and performance of the dimension model as more tables would be required to be joined to satisfy the queries.

## Characteristics of snowflake schema:

The dimension model of a snowflake under the following conditions:

1. The snowflake schema uses small disk space.
2. It is easy to implement the dimension that is added to the schema.
3. There are multiple tables, so performance is reduced.
4. The dimension table consists of two or more sets of attributes that define information at different grains.
5. The sets of attributes of the same dimension table are being populated by different source systems.

## Advantages:

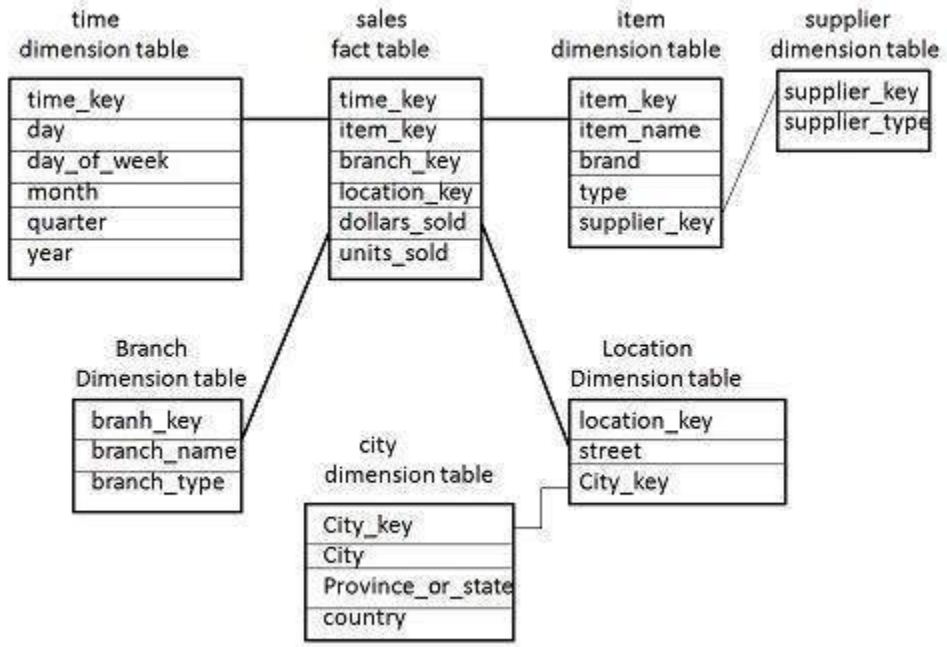
There are two main advantages of snowflake schema given below:

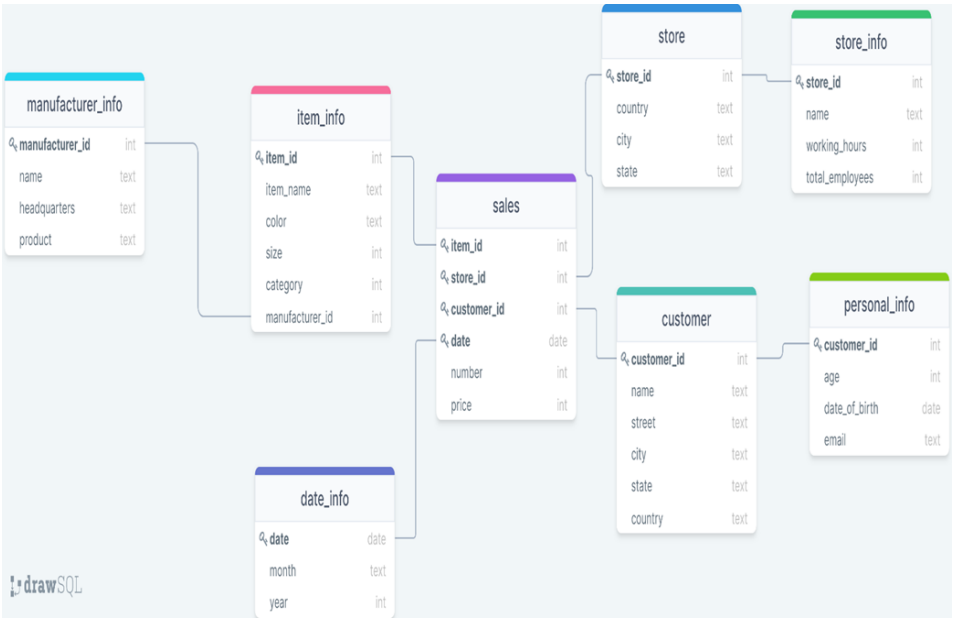
1. It provides structured data which reduces the problem of data integrity.
2. It uses small disk space because data are highly structured.

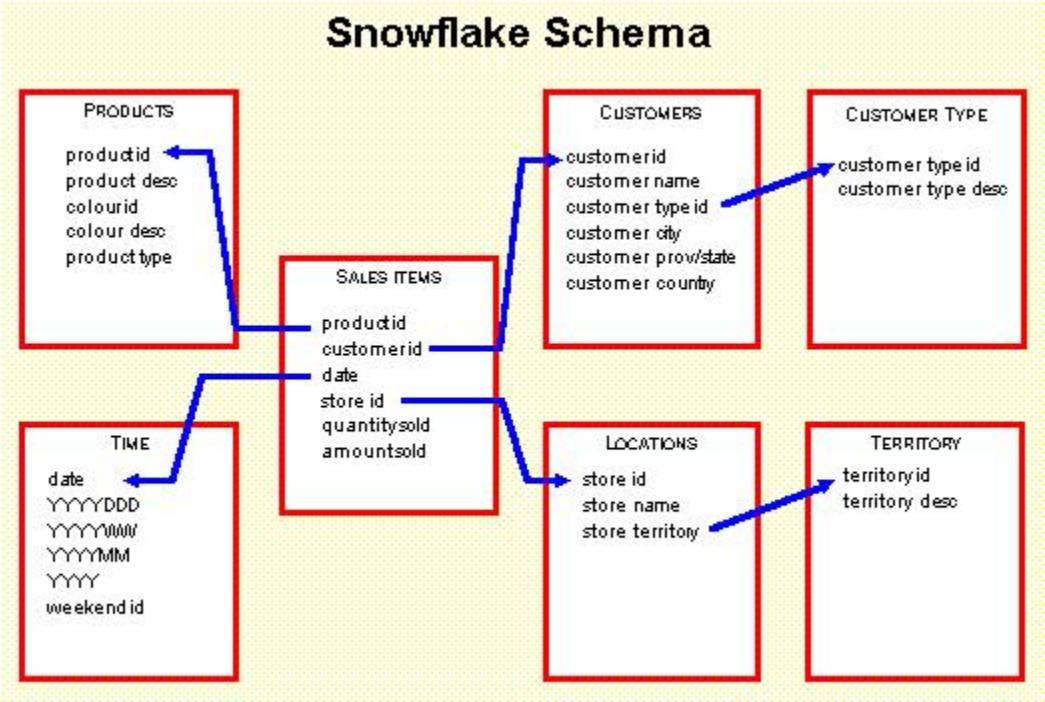
## Disadvantages:

1. Snowflaking reduces space consumed by dimension tables but compared with the entire data warehouse the saving is usually insignificant.
2. Avoid snowflaking or normalization of a dimension table, unless required and appropriate.
3. Do not snowflake hierarchies of one dimension table into separate tables. Hierarchies should belong to the dimension table only and should never be snowflakes.
4. Multiple hierarchies that can belong to the same dimension have been designed at the lowest possible detail.

## Snowflake schema examples:




manufacturer_info
manufacturer_id int
name text
headquarters text
product text
item_info
item_id int
item_name text
color text
size int
category int
manufacturer_id int
sales
item_id int
store_id int
customer_id int
date date
number int
price int
store
store_id int
country text
city text
state text
store_info
store_id int
name text
working_hours int
total_employees int
customer
customer_id int
name text
street text
city text
state text
country text
personal_info
customer_id int
age int
date_of_birth date
email text
date_info
date date
month text
year int
drawSQL


Snowflake Schema
PRODUCTS
productid
product desc
colourid
colour desc
product type
SALES ITEMS
productid
customerid
date
store id
quantitysold
amountsold
CUSTOMERS
customerid
customer name
customer type id
customer city
customer prov/state
customer country
CUSTOMER TYPE
customer type id
customer type desc
TIME
date
YYYYDDD
YYYYWW
YYYYMM
YYYY
weekend id
LOCATIONS
store id
store name
store territory
TERRITORY
territoryid
territory desc

## Conclusion:

Conceptual models using Star and Snowflake schema were designed after the completion of this case study.