LAB CYCLE 4

Date : 05/05/2025

Experiment No: 7

AIM : Familiarization of TCL Commands.

Transaction Control Language (TCL) – These SQL commands are used to manage transactions in a database, ensuring data consistency and integrity. They help in controlling changes made by DML statements. The main TCL commands are **COMMIT, ROLLBACK, and SAVEPOINT**.

1. Create a Table 'Order_Details' with the given data and perform the following operations.

```
>> create table OrderDetails(OrderID int PRIMARY KEY, Product_Name
   varchar(50),Order_Num int, Order_Date date );

   insert into OrderDetails values(1,'Laptop',5544,'2020-02-01');
```

OUTPUT:

```
+---------+--------------+-----------+------------+
| OrderID | Product_Name | Order_Num | Order_Date |
+---------+--------------+-----------+------------+
|       1 | Laptop       |      5544 | 2020-02-01 |
|       2 | Mouse        |      3322 | 2020-02-11 |
|       3 | Desktop      |      2135 | 2020-01-05 |
|       4 | Mobile       |      3432 | 2020-02-22 |
|       5 | Anti-Virus   |      5648 | 2020-03-10 |
+---------+--------------+-----------+------------+
```

a) Start a new transaction and insert 2 rows into the table Order_Details.

```
>> SET AUTOCOMMIT = 0

   START TRANSACTION;

   SAVEPOINT S1;

   insert into OrderDetails values(12,'keyboard',5667,'2020-02-15');
   insert into OrderDetails values(14,'printer',5321,'2020-05-12');
```

b) Display the details of Order_Details.

>> Select * from OrderDetails;

OUTPUT:

OrderID	Product_Name	Order_Num	Order_Date
1	Laptop	5544	2020-02-01
2	Mouse	3322	2020-02-11
3	Desktop	2135	2020-01-05
4	Mobile	3432	2020-02-22
5	Anti-Virus	5648	2020-03-10
12	keyboard	5667	2020-02-15
14	printer	5321	2020-05-12

c) Rollback the current transaction and check the contents of the table Order_Details.

>> ROLLBACK TO S1;

OUTPUT:

OrderID	Product_Name	Order_Num	Order_Date
1	Laptop	5544	2020-02-01
2	Mouse	3322	2020-02-11
3	Desktop	2135	2020-01-05
4	Mobile	3432	2020-02-22
5	Anti-Virus	5648	2020-03-10

d) Start a new transaction and delete 2 rows from the table Order_Details.

>> START TRANSACTION;

delete from OrderDetails where OrderID = 3;

delete from OrderDetails where OrderID = 5;

select * from OrdersDetails

e) Display the details of Order_Details.

>> select * from OrderDetails;

OrderID	Product_Name	Order_Num	Order_Date
1	Laptop	5544	2020-02-01
2	Mouse	3322	2020-02-11
4	Mobile	3432	2020-02-22

f) Commit the current transaction and check the details of the table Order_Details.

>> COMMIT;

OrderID	Product_Name	Order_Num	Order_Date
1	Laptop	5544	2020-02-01
2	Mouse	3322	2020-02-11
4	Mobile	3432	2020-02-22

g) Disable autocommit and create a savepoint named ‘Check_Updates’.

>> SET AUTOCOMMIT = 0;

SAVEPOINT Check_Updates;

h) Update details of the order with Order_Num 5544.

>> update OrderDetails set Product_Name = 'Laptop HP' where Order_Num=5544;

i) Insert 2 more rows into the table.

>> insert into OrderDetails values(25,'keyboard',5632,'2020-02-15');

insert into OrderDetails values(28,'printer',7521,'2020-05-12');

OUTPUT:

OrderID	Product_Name	Order_Num	Order_Date
1	Laptop HP	5544	2020-02-01
2	Mouse	3322	2020-02-11
4	Mobile	3432	2020-02-22
25	keyboard	5632	2020-02-15
28	printer	7521	2020-05-12

j) Create a savepoint named ‘Check_delete’

>> SAVEPOINT Check_delete;

k) Delete order details of a product named ‘Laptop’.

>> delete from OrderDetails where Product_Name = 'Laptop HP';

l) Display the current details of the table Order_Details.

>> select * from OrderDetails;

OUTPUT:

OrderID	Product_Name	Order_Num	Order_Date
2	Mouse	3322	2020-02-11
4	Mobile	3432	2020-02-22
25	keyboard	5632	2020-02-15
28	printer	7521	2020-05-12

m) Rollback to savepoint ‘Check_delete’ and check the details of the table Order_Details.

>> ROLLBACK TO Check_Delete;

OUTPUT:

OrderID	Product_Name	Order_Num	Order_Date
1	Laptop HP	5544	2020-02-01
2	Mouse	3322	2020-02-11
4	Mobile	3432	2020-02-22
25	keyboard	5632	2020-02-15
28	printer	7521	2020-05-12

n) Rollback to savepoint ‘Check_Updates’ and check the details of the table Order_Details.

>> ROLLBACK TO Check_Delete;

OUTPUT:

OrderID	Product_Name	Order_Num	Order_Date
1	Laptop	5544	2020-02-01
2	Mouse	3322	2020-02-11
4	Mobile	3432	2020-02-22

o) Commit the changes and enable autocommit.

>> COMMIT;

SET AUTOCOMMIT = 1;

LAB CYCLE 5

Date : 06/05/2025

Experiment No: 8

AIM : Familiarization of MongoDB CRUD operations.

CRUD Operations in MongoDB – These are the basic operations used to interact with data in MongoDB collections. CRUD stands for Create, Read, Update, and Delete. MongoDB uses methods like insertOne(), find(), updateOne(), and deleteOne() to perform these operation efficiently on documents.

1. Create a Mongodb Database named “Inventory”.

>> use Inventory49

2. Create a collection named ‘Products’ and Insert the following documents.

>> db.createCollection("products")

```
db.products.insertMany([
{
  "_id": 1,
  "name": "xPhone",
  "price": 799,
  "releaseDate": ISODate("2011-05-14"),
  "spec": { "ram": 4, "screen": 6.5, "cpu": 2.66 },
  "color": ["white", "black"],
  "storage": [64, 128, 256]
 },
```

3. Display all documents in the collection product.

>> db.products.find()

OUTPUT:

```
Inventory49> db.products.find()
[
  {
    '0': {
      _id: 1,
      name: 'xPhone',
      price: 799,
      releaseDate: ISODate('2011-05-14T00:00:00.000Z'),
      spec: { ram: 4, screen: 6.5, cpu: 2.66 },
      color: [ 'white', 'black' ],
      storage: [ 64, 128, 256 ]
    },
    _id: ObjectId('6819cfb899f85ef3846b140f')
  },
  {
    _id: 2,
    name: 'xTablet',
    price: 899,
    releaseDate: ISODate('2011-09-01T00:00:00.000Z'),
    spec: { ram: 16, screen: 9.5, cpu: 3.66 },
    color: [ 'white', 'black', 'purple' ],
    storage: [ 128, 256, 512 ]
  },
  {
    _id: 3,
    name: 'SmartTablet',
    price: 899,
    releaseDate: ISODate('2015-01-14T00:00:00.000Z'),
    spec: { ram: 12, screen: 9.7, cpu: 3.66 },
    color: [ 'blue' ],
    storage: [ 16, 64, 128 ]
  },
  {
    _id: 4,
    name: 'SmartPad',
    price: 699,
    releaseDate: ISODate('2020-05-14T00:00:00.000Z'),
    spec: { ram: 8, screen: 9.7, cpu: 1.66 },
    color: [ 'white', 'orange', 'gold', 'gray' ],
    storage: [ 128, 256, 1024 ]
  },
  {
    _id: 5,
    name: 'SmartPhone',
    price: 599,
    releaseDate: ISODate('2022-09-14T00:00:00.000Z'),
    spec: { ram: 4, screen: 9.7, cpu: 1.66 },
    color: [ 'white', 'orange', 'gold', 'gray' ],
    storage: [ 128, 256 ]
  }
]
```

4. Display all the details of product with _ id is 2.

>> db.products.find({_id: 2})

```
Inventory49> db.products.find({_id: 2})
[
  {
    _id: 2,
    name: 'xTablet',
    price: 899,
    releaseDate: ISODate('2011-09-01T00:00:00.000Z'),
    spec: { ram: 16, screen: 9.5, cpu: 3.66 },
    color: [ 'white', 'black', 'purple' ],
    storage: [ 128, 256, 512 ]
  }
]
```

5. Display the first document in the collection product.

>> db.products.findone()

```
Inventory49> db.products.findOne()
{
  '0': {
    _id: 1,
    name: 'xPhone',
    price: 799,
    releaseDate: ISODate('2011-05-14T00:00:00.000Z'),
    spec: { ram: 4, screen: 6.5, cpu: 2.66 },
    color: [ 'white', 'black' ],
    storage: [ 64, 128, 256 ]
  },
  _id: ObjectId('6819cfb899f85ef3846b140f')
}
```

6. Display name and price of product with _ id is 5.

>> db.products.find({ _id: 5}, { name: 1,price: 1})

```
Inventory49> db.products.find({ _id: 5}, { name: 1,price: 1})
[ { _id: 5, name: 'SmartPhone', price: 599 } ]
```

7. Query the products collection to select all documents where the value of the price field equals 899.

>> db.products.find({price: {$eq: 899}}, {name: 1,price: 1})

```
[
  { _id: 2, name: 'xTablet', price: 899 },
  { _id: 3, name: 'SmartTablet', price: 899 }
]
```

8. Search for documents where the value of the ram field in the spec document equals 4.

>> db.products.find({"spec.ram": {$eq: 4}}, {name: 1,"spec.ram": 1})

```
[ { _id: 5, name: 'SmartPhone', spec: { ram: 4 } } ]
```

9. Query the products collection to find all documents where the array color contains an element with the value "black".

>> db.products.find({color: {$eq: "black"}}, {name: 1,color: 1})

```
[ { _id: 2, name: 'xTablet', color: [ 'white', 'black', 'purple' ] } ]
```

10. Select documents in the products collection with the published date is 2020-05-14

>> db.products.find({releaseDate: {$eq: new ISODate("2020-05-14")}},
{name:1,releaseDate:1})

```
[
  {
    _id: 4,
    name: 'SmartPad',
    releaseDate: ISODate('2020-05-14T00:00:00.000Z')
  }
]
```

11. select documents from the products collection where price is less than 799.

>> db.products.find({price: {$lt: 799}}, {name: 1,price: 1})

```
[
  { _id: 4, name: 'SmartPad', price: 699 },
  { _id: 5, name: 'SmartPhone', price: 599 }
]
```

12. select documents where the value of the screen field in the spec document is less than 7.

>> db.products.find({"spec.screen": {$lt: 7}}, {name: 1,"spec.screen": 1})

```
[ { _id: 1, name: 'xPhone', spec: { screen: 6.5 } } ]
```

13. query the products collection to find all documents where the array storage has at least one element less than 128.

>> db.products.find({storage: {$lt: 128}}, {name: 1,storage: 1})

```
[
  { _id: 1, name: 'xPhone', storage: [ 64, 128, 256 ] },
  { _id: 3, name: 'SmartTablet', storage: [ 16, 64, 128 ] }
]
```

14. Display documents from the products collection whose the price is either 699 or 799.

>> db.products.find({price: {$in: [699, 799]}}, {name: 1,price: 1})

```
[
  { _id: 1, name: 'xPhone', price: 799 },
  { _id: 4, name: 'SmartPad', price: 699 }
]
```

15. Display documents where the color array has at least one element either "black" or "white".

>> db.products.find({color: {$in: ["black", "white"]}}, { name: 1,color: 1})

```
[
  { _id: 2, name: 'xTablet', color: [ 'white', 'black', 'purple' ] },
  {
    _id: 4,
    name: 'SmartPad',
    color: [ 'white', 'orange', 'gold', 'gray' ]
  },
  {
    _id: 5,
    name: 'SmartPhone',
    color: [ 'white', 'orange', 'gold', 'gray' ]
  }
]
```

16. Display documents from the products collection whose price is neither 699 or 799.

>> db.products.find({price: {$nin: [699, 799]}}, {name: 1,price: 1})

```
[
  { _id: ObjectId('6819cfb899f85ef3846b140f') },
  { _id: 2, name: 'xTablet', price: 899 },
  { _id: 3, name: 'SmartTablet', price: 899 },
  { _id: 5, name: 'SmartPhone', price: 599 }
]
```

17. Display documents where the color array doesn't have an element that is either "black" or "white".

>> db.products.find({color: {$nin: ["black", "white"]}}, {name: 1,color: 1})

```
[
  { _id: ObjectId('6819cfb899f85ef3846b140f') },
  { _id: 3, name: 'SmartTablet', color: [ 'blue' ] }
]
```

18. Display all documents in the products collection where the value in the price field is equal to 899 and the value in the color field is either "white" or "black"

>> db.products.find({$and: [{price: 899}, {color: {$in: ["white", "black"]}}]}, {name: 1, price: 1,color: 1})

```
[
  {
    _id: 2,
    name: 'xTablet',
    price: 899,
    color: [ 'white', 'black', 'purple' ]
  }
]
```

19. Select all documents where the price is less than 699 or greater than 799.

>> db.products.find({$or: [{ price: {$lt: 699} },{ price: {$gt: 799} }]}, {name: 1,price: 1});

```
[
  { _id: 2, name: 'xTablet', price: 899 },
  { _id: 3, name: 'SmartTablet', price: 899 },
  { _id: 5, name: 'SmartPhone', price: 599 }
]
```

20. Sorts the products by the values in the ram field in the spec embedded documents. It includes the _id, name, and spec fields in the matching documents.

>> db.products.find({}, {name: 1,spec: 1}).sort({"spec.ram": 1});

```
[
  { _id: ObjectId('6819cfb899f85ef3846b140f') },
  {
    _id: 5,
    name: 'SmartPhone',
    spec: { ram: 4, screen: 9.7, cpu: 1.66 }
  },
  {
    _id: 4,
    name: 'SmartPad',
    spec: { ram: 8, screen: 9.7, cpu: 1.66 }
  },
  {
    _id: 3,
    name: 'SmartTablet',
    spec: { ram: 12, screen: 9.7, cpu: 3.66 }
  },
  {
    _id: 2,
    name: 'xTablet',
    spec: { ram: 16, screen: 9.5, cpu: 3.66 }
  }
]
```

21. Sorts the products by the values in the releaseDate field in descending order.

>> db.products.find({releaseDate: {$exists: 1}}, {name: 1,releaseDate: 1}).sort({
releaseDate: -1});

```
[
  {
    _id: 5,
    name: 'SmartPhone',
    releaseDate: ISODate('2022-09-14T00:00:00.000Z')
  },
  {
    _id: 4,
    name: 'SmartPad',
    releaseDate: ISODate('2020-05-14T00:00:00.000Z')
  },
  {
    _id: 3,
    name: 'SmartTablet',
    releaseDate: ISODate('2015-01-14T00:00:00.000Z')
  },
  {
    _id: 2,
    name: 'xTablet',
    releaseDate: ISODate('2011-09-01T00:00:00.000Z')
  }
]
```

22. Sort the products by name and price in ascending order. It selects only documents where the price field exists and includes the _id, name, and price fields in the matching documents.

>> db.products.find({'price': {$exists: 1}}, { name: 1, price: 1}).sort({price: 1,name: 1})

```
[
  { _id: 5, name: 'SmartPhone', price: 599 },
  { _id: 4, name: 'SmartPad', price: 699 },
  { _id: 3, name: 'SmartTablet', price: 899 },
  { _id: 2, name: 'xTablet', price: 899 }
]
```

23. Get the most expensive product in the products collection. It includes the _id, name, and price fields in the returned documents:

>> db.products.find({}, {name: 1,price: 1}).sort({price: -1,name: 1}).limit(1);

```
[ { _id: 3, name: 'SmartTablet', price: 899 } ]
```

LAB CYCLE 6

Date : 07/05/2025

Experiment No: 9

AIM : Familiarization of MongoDB aggregation operations.

Aggregation Operations in MongoDB – These operations process data records and return computed results, allowing for data transformation and analysis. MongoDB uses the aggregation pipeline, which includes stages like $match, $group, $sort, and $project to filter, group, and format data.

>> use Inventory49
>> db.createCollection(“sales”)

```
db.sales.insertMany([
{ "_id": 1, "item": "Americanos", "price": 5, "size": "Short", "quantity": 22,
"date": ISODate("2022-01-15T08:00:00Z") },
{ "_id": 2, "item": "Cappuccino", "price": 6, "size": "Short", "quantity": 12,
  "date": ISO  Date("2022-01-16T09:00:00Z") }
    ])
```

```
>> db.sales.find()
```

```
Inventory49> db.sales.find()
[
  {
    _id: 1,
    item: 'Americanos',
    price: 5,
    size: 'Short',
    quantity: 22,
    date: ISODate('2022-01-15T08:00:00.000Z')
  },
  {
    _id: 2,
    item: 'Cappuccino',
    price: 6,
    size: 'Short',
    quantity: 12,
    date: ISODate('2022-01-16T09:00:00.000Z')
  },
  {
    _id: 3,
    item: 'Lattes',
    price: 15,
    size: 'Grande',
    quantity: 25,
    date: ISODate('2022-01-16T09:05:00.000Z')
  },
  {
    _id: 4,
    item: 'Mochas',
    price: 25,
    size: 'Tall',
    quantity: 11,
    date: ISODate('2022-02-17T08:00:00.000Z')
  },
  {
    _id: 5,
    item: 'Americanos',
    price: 10,
    size: 'Grande',
    quantity: 12,
    date: ISODate('2022-02-18T21:06:00.000Z')
  },
```

```
{
  _id: 6,
  item: 'Cappuccino',
  price: 7,
  size: 'Tall',
  quantity: 20,
  date: ISODate('2022-02-20T10:07:00.000Z')
},
{
  _id: 7,
  item: 'Lattes',
  price: 25,
  size: 'Tall',
  quantity: 30,
  date: ISODate('2022-02-21T10:08:00.000Z')
},
{
  _id: 8,
  item: 'Americanos',
  price: 10,
  size: 'Grande',
  quantity: 21,
  date: ISODate('2022-02-22T14:09:00.000Z')
},
{
  _id: 9,
  item: 'Cappuccino',
  price: 10,
  size: 'Grande',
  quantity: 17,
  date: ISODate('2022-02-23T14:09:00.000Z')
},
{
  _id: 10,
  item: 'Americanos',
  price: 8,
  size: 'Tall',
  quantity: 15,
  date: ISODate('2022-02-25T14:09:00.000Z')
}
```

1. Groups the documents by the item field and use the $avg to calculate the average amount for each group

>> db.sales.aggregate([{$group: {_id: '$item',averageAmount: { $avg: { $multiply: ['$quantity', '$price'] } },},},{ $sort: { averageAmount: 1 } },]);

```
[
  { _id: 'Cappuccino', AverageAmount: 127.33333333333333 },
  { _id: 'Americanos', AverageAmount: 140 },
  { _id: 'Mochas', AverageAmount: 275 },
  { _id: 'Lattes', AverageAmount: 562.5 }
]
```

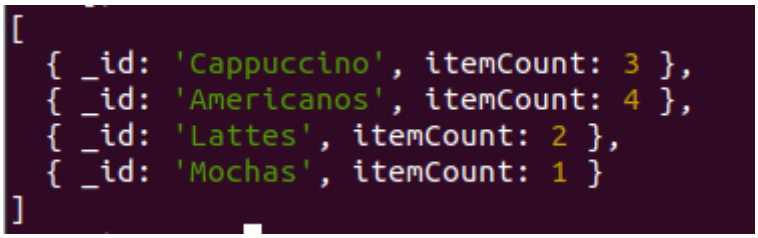
2. Calculate the average amount per group and returns the group with the average amount greater than 150.

>> db.sales.aggregate([{$group: {_id: '$item',averageAmount: { $avg: { $multiply: ['$quantity', '$price'] } },},},{ $match: { averageAmount: { $gt: 150 } } },{ $sort: { averageAmount: 1

```
[
  { _id: 'Mochas', averageAmount: 275 },
  { _id: 'Lattes', averageAmount: 562.5 }
]
```

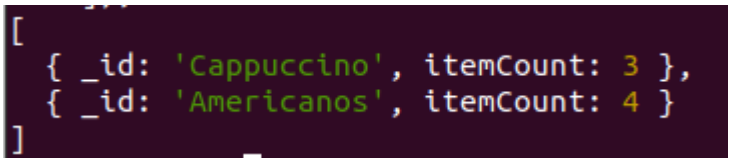
3. Return the number of items in the sales collection.

>> db.sales.aggregate([{$group: {_id: '$item',itemCount: { $count: {} },},},]);



4. Calculate the number of documents per item and returns the item with a count greater than two.

>> db.sales.aggregate([{$group: {_id: '$item',itemCount: { $count: {} },},},{$match: { itemCount: { $gt: 2 } },},]);

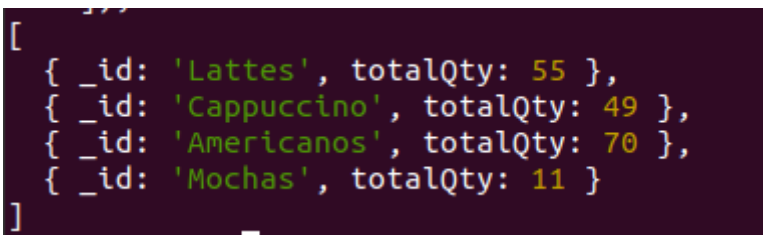


5. Calculates the total quantity of coffee sales in the sales collection

>> db.sales.aggregate([{$group: {_id: null,totalQty: { $sum: '$quantity' },},},{ $project: { _id: 0 } },]);

6. Calculate the sum of quantity grouped by items

>> db.sales.aggregate([{$group: {_id: '$item',totalQty: { $sum: '$quantity' },},},]);



7. Returns the total quantity of each item and sorts the result documents by the totalQty in descending order.

>> db.sales.aggregate([{$group: {_id: '$item',totalQty: { $sum: '$quantity' },},},{ $sort: { totalQty: -1 } },]);

```
[
  { _id: 'Americanos', totalQty: 70 },
  { _id: 'Lattes', totalQty: 55 },
  { _id: 'Cappuccino', totalQty: 49 },
  { _id: 'Mochas', totalQty: 11 }
]
```

8. Find the maximum quantity from all the sales documents.

>> db.sales.aggregate([{$group: {_id: null,maxQty: { $max: '$quantity' },},},{$project: {_id: 0,},},]);

```
[ { maxQuantity: 30 } ]
```

9. Group documents in the item field and returns the maximum quantity per group of Documents.

>> db.sales.aggregate([{$group: {_id: '$item',maxQty: { $max: '$quantity' },},},]);

```
[
  { _id: 'Lattes', maxQty: 30 },
  { _id: 'Cappuccino', maxQty: 20 },
  { _id: 'Americanos', maxQty: 22 },
  { _id: 'Mochas', maxQty: 11 }
]
```

10. Groups the documents by the item field and returns the maximum amount for each group of sales.

>> db.sales.aggregate([{$group: {_id: '$item',maxQty: { $max: { $multiply: ['$quantity', '$price'] } },},},]);

```
[
  { _id: 'Lattes', maxQty: 750 },
  { _id: 'Cappuccino', maxQty: 170 },
  { _id: 'Mochas', maxQty: 275 },
  { _id: 'Americanos', maxQty: 210 }
]
```