# SUMMARY: TCL AND DCL COMMANDS

## SESSION OVERVIEW:

By the end of this session, you will be able to:

- Understand the various commands used in TCL.
- Understand the various commands used in DCL.
- Understand the utilization of AI in SQL.

## KEY TOPICS AND EXAMPLES:

### Understanding the various commands used in TCL:

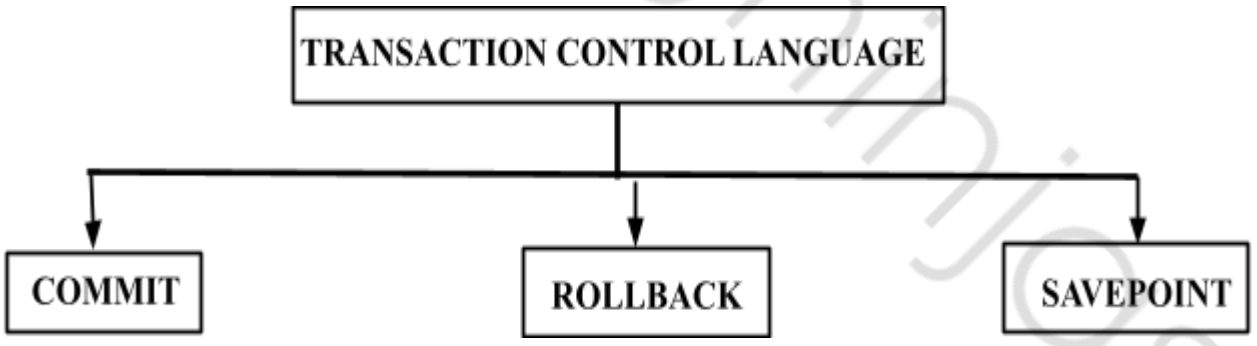
### What is a Transaction?

A transaction is a unit of work that is performed against a database in SQL. In other words, a transaction is a single, indivisible database action. If the transaction contains multiple statements, it is called a multi-statement transaction (MST). By default, all transactions are multi-statement transactions.

For example, suppose we are creating a new record or updating or deleting any record from a table (in general, performing any changes on the table). In that case, we are performing a transaction on the table.

In SQL, each transaction begins with a particular set of task and ends only when all the tasks in the set are completed successfully. However, if any (or a single) task fails, the transaction is said to fail.

### Types of TCL commands:




1. **Concept on COMMIT:**

    a. **What is COMMIT?**
    COMMIT command in SQL is used to save all the transaction-related changes permanently to the disk. Whenever DDL commands such as INSERT, UPDATE, and DELETE are used, the changes made by these commands are permanent only after closing the current session. So before closing the session, one can easily roll back the

changes made by the DDL commands. Hence, if we want the changes to be saved permanently to the disk without closing the session, we will use the commit command.

**b. Uses of COMMIT:**

**Saving Changes:** Permanently save all the changes made during the current transaction to the database. COMMIT ensures that all changes made by the transaction are permanently saved and cannot be undone

**Ending a Transaction:** Mark the end of a transaction, signaling that all operations within the transaction are complete and finalized.

**Ensuring Data Integrity:** Maintain data consistency and durability, especially in multi-user database environments where transactions might be interleaved.

**Releasing Resources:** Free up resources, such as locks on data, that were held during the transaction to ensure data consistency.

**Checkpoint in Batch Processing:** Save intermediate results periodically during long-running scripts or batch processing to reduce the risk of data loss due to unexpected failures.

**c. Examples:**

```
START TRANSACTION;

INSERT INTO products (ProductKey, productsubcategorykey, productSKU,
Productname, modelname, productcost, productprice)
VALUES (101, 5, 'SKU12345', 'New Product', 'Model X', 100.00, 150.00);

COMMIT;

SELECT * FROM products
```

**Output:**

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| ProductKey | ProductSubcategoryKey | ProductSKU | ProductName | ModelName | ProductDescription |
|---|---|---|---|---|---|
| 603 | 5 | BB-9108 | HL Bottom Bracket | HL Bottom Bracket | Aluminum alloy cups and a hollow axle. |
| 604 | 2 | BK-R19B-44 | Road-750 Black, 44 | Road-750 | Entry level adult bike; offers a comfortable ride ... |
| 605 | 2 | BK-R19B-48 | Road-750 Black, 48 | Road-750 | Entry level adult bike; offers a comfortable ride ... |
| 606 | 2 | BK-R19B-52 | Road-750 Black, 52 | Road-750 | Entry level adult bike; offers a comfortable ride ... |
| 101 | 5 | SKU12345 | New Product | Model X | NULL |

In the above output, we can observe that a new entry has been added.

Autocommit is by default enabled in MySQL. To turn it off, we will set the value of autocommit as 0.

```
SET autocommit = 0;
```

**Example 2: (COMMIT after Multiple Operations)**

```
START TRANSACTION;

-- Update product prices
UPDATE products
SET productprice = productprice * 1.10
WHERE productsubcategorykey = 5;

-- Insert a new customer
INSERT INTO Customers (customerkey, prefix, firstname, lastname)
VALUES (301, 'Mr.', 'John', 'Doe');

COMMIT;
```

2. Concepts on SAVEPOINT:
    a. What is SAVEPOINT?
    We can divide the database operations into parts.

    For example, we can consider all the insert-related queries that we will execute consecutively as one part of the transaction and the delete command as the other part of the transaction.

    Using the SAVEPOINT command in SQL, we can save these different parts of the same transaction using different names.

    For example, we can save all the insert-related queries with the savepoint named INS. To save all the insert-related queries in one savepoint, we have to execute the SAVEPOINT query followed by the savepoint name after finishing the insert command execution.

    **b. Benefits of SAVEPOINT:**

    **Partial Rollbacks:** SAVEPOINT allows you to roll back parts of a transaction without affecting the entire transaction. This enables more fine-grained control over which operations to undo.

    **Error Handling:** In complex transactions, you can set savepoints to isolate problematic sections. If an error occurs, you can roll back to a specific savepoint instead of rolling back the entire transaction.

    **Nested Transactions:** SAVEPOINT effectively supports nested transactions, providing a way to manage multiple levels of transactions within a single outer transaction.

**Improved Flexibility:** With SAVEPOINT, you can structure your transaction logic more flexibly, accommodating various scenarios and conditions that might require partial rollbacks.

**Better Resource Management:** By allowing partial rollbacks, SAVEPOINT helps in managing resources efficiently, as you don't need to restart entire transactions from scratch in case of minor issues.

**Enhanced Debugging:** Savepoints can be used to mark specific stages of a transaction, making it easier to debug and test different parts of a transaction independently.

**Reduced Transaction Time:** By avoiding the need to restart the entire transaction, savepoints can help reduce the overall time required for transaction processing, particularly in long-running transactions.

c. Examples:

```
START TRANSACTION;

INSERT INTO products (ProductKey, productsubcategorykey, productSKU,
Productname, modelname, productcost, productprice)
VALUES (112, 11, 'SKU44444', 'Product D', 'Model D', 140.00, 210.00);

SAVEPOINT after_insert;

INSERT INTO products (ProductKey, productsubcategorykey, productSKU,
Productname, modelname, productcost, productprice)
VALUES (113, 12, 'SKU55555', 'Product E', 'Model E', 150.00, 220.00);

SAVEPOINT after_second_insert;

COMMIT;
```

**Output:**

- The SQL script begins by starting a transaction with START TRANSACTION;, indicating that a series of database operations following this command should be treated as a single unit of work.
- Within this transactional context, the script first inserts a new product (Product D) into the products table. This insertion includes details such as the product's key, SKU, name, model, cost, and price, ensuring these values are committed together if the transaction is successful.
- After the first insertion, a savepoint named after_insert is established using SAVEPOINT after_insert;. Savepoints allow the transaction to mark specific points where it can later roll back without affecting operations that occurred after the savepoint.
- Following the savepoint creation, the script attempts to insert another product (Product E) into

the products table. Similar to the first insertion, this operation adds a new set of product details to the database.

- After completing the second insertion, another savepoint named after_second_insert is set with SAVEPOINT after_second_insert;, marking the completion of the second insert operation within the transaction.
- To finalize the transaction, the script executes COMMIT;, which commits all changes made within the transaction to the database. In this case, both insert operations (Product D and Product E) are successfully committed, thereby adding these products to the products table permanently.

### 3. Concepts on ROLLBACK:

#### a. What is ROLLBACK?

While carrying out a transaction, we must create savepoints to save different parts of the transaction. According to the user's changing requirements, he/she can roll back the transaction to different savepoints.
Consider a scenario: We have initiated a transaction followed by the table creation and record insertion into the table. After inserting records, we created a savepoint INS. Then we executed a delete query, but later we thought that mistakenly we had removed the useful record. Therefore in such situations, we have the option of rolling back our transaction. In this case, we have to roll back our transaction using the ROLLBACK command to the savepoint INS, which we have created before executing the DELETE query.

#### b. Benefits of ROLLBACK:

- **Data Integrity and Consistency:** Preserves data state by reverting to the previous state if an error occurs. Ensures atomic transactions where all operations succeed or none do.
- **Error Handling and Recovery:** Facilitates easy recovery from errors or exceptions. Allows undoing unintended or incorrect modifications.
- **Safe Experimentation and Testing:** Enables safe testing of database operations without making permanent changes. Allows simulation of transactions to analyze impact without committing changes.
- **Controlled Transaction Management:** Supports savepoints for finer control, enabling partial rollback to specific points. Allows undoing specific parts of a transaction while retaining successful operations.
- **User Protection:** Provides a safety net against accidental data modifications or deletions. Allows users to make informed commit or rollback decisions based on validation checks. ROLLBACK undoes all changes made by the transaction up to the point where ROLLBACK is issued or to a specific savepoint

#### c. Examples:

```
START TRANSACTION;
```

```
INSERT INTO products (ProductKey, productsubcategorykey, productSKU,
Productname, modelname, productcost, productprice)
VALUES (109, 8, 'SKU11111', 'Product A', 'Model A', 100.00, 150.00);

UPDATE products
SET productprice = productprice * 1.10
WHERE ProductKey = 109;

ROLLBACK;

select * from products
```

**Output:**

The provided SQL script demonstrates the use of a transaction block to manage database operations with the ability to rollback changes if necessary.

- The START TRANSACTION statement begins a new transaction, ensuring that subsequent operations are treated as a single unit.
- The script first inserts a new product with ProductKey 109 into the products table.
- Immediately after, it updates the price of this newly inserted product by increasing it by 10%. However, before the transaction is committed, the ROLLBACK statement is executed, which undoes all changes made within the transaction.
- This means that both the insert and the update operations are reverted, and no changes are applied to the database.
- Finally, the SELECT * FROM products query retrieves all records from the products table, showing that the new product and the price update are not present because the transaction was rolled back. This ensures that the database remains in its previous consistent state, demonstrating the importance and functionality of transactional control in maintaining data integrity.

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| ProductKey | ProductSubcategoryKey | ProductSKU | ProductName | ModelName | ProductDescription |
|---|---|---|---|---|---|
| 603 | 5 | BB-9108 | HL Bottom Bracket | HL Bottom Bracket | Aluminum alloy cups and a hollow axle. |
| 604 | 2 | BK-R19B-44 | Road-750 Black, 44 | Road-750 | Entry level adult bike; offers a comfortable ride ... |
| 605 | 2 | BK-R19B-48 | Road-750 Black, 48 | Road-750 | Entry level adult bike; offers a comfortable ride ... |
| 606 | 2 | BK-R19B-52 | Road-750 Black, 52 | Road-750 | Entry level adult bike; offers a comfortable ride ... |
| 101 | 5 | SKU12345 | New Product | Model X | NULL |

**Example 2:**

```
START TRANSACTION;

INSERT INTO products (ProductKey, productsubcategorykey, productSKU,
Productname, modelname, productcost, productprice)
VALUES (112, 11, 'SKU44444', 'Product D', 'Model D', 140.00, 210.00);
```

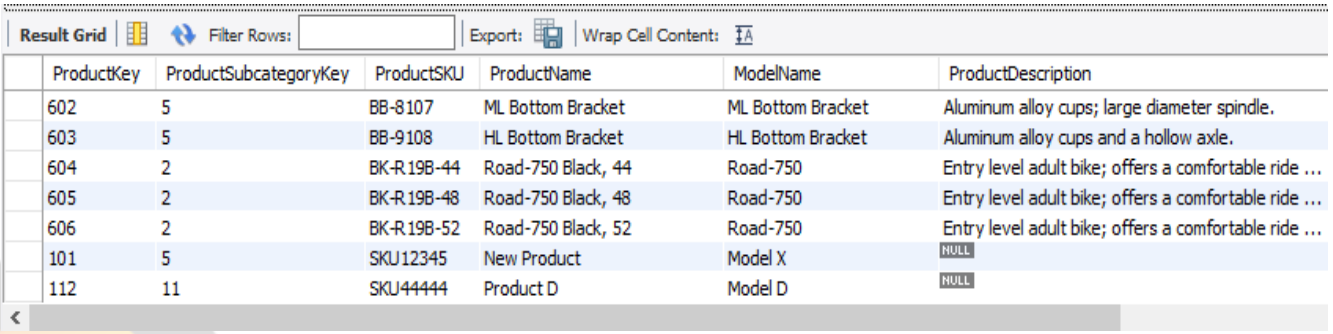
```
SAVEPOINT first_insert;

INSERT INTO products (ProductKey, productsubcategorykey, productSKU,
Productname, modelname, productcost, productprice)
VALUES (113, 12, 'SKU55555', 'Product E', 'Model E', 150.00, 220.00);

SAVEPOINT second_insert;

ROLLBACK TO SAVEPOINT first_insert;

COMMIT;
```

**Output:**

- The script begins by starting a transaction using START TRANSACTION;. This ensures that subsequent operations are treated as a single atomic unit, where either all changes are committed or rolled back together.
- Following this, the script inserts a new product into the products table using the INSERT INTO statement. This product ('Product D') is identified by ProductKey 112, belongs to productsubcategorykey 11, has a SKU 'SKU44444', and includes details such as Productname 'Product D', modelname 'Model D', productcost 140.00, and productprice 210.00.
- A savepoint named first_insert is then set immediately after the first insert operation. Savepoints allow for establishing points within a transaction from which you can later roll back if needed.
- Continuing with the transaction, another product ('Product E') is inserted into the products table. This second product has ProductKey 113, productsubcategorykey 12, SKU 'SKU55555', Productname 'Product E', modelname 'Model E', productcost 150.00, and productprice 220.00. A savepoint named second_insert is set after this second insert operation.
- Next, the script simulates an error condition that necessitates a rollback to the first_insert savepoint. This rollback operation effectively undoes the insertion of the second product ('Product E') into the products table.
- Finally, the transaction is committed using COMMIT;, which would typically make all changes permanent. However, due to the preceding rollback, only the first insert operation (insertion of 'Product D') remains committed in this specific scenario.

After executing this script, the products table will contain only the first inserted product (Product D) with ProductKey = 112 and its associated details (productsubcategorykey, productSKU, Productname, modelname, productcost, productprice).

The second insert (Product E) is rolled back, so it will not appear in the products table after the transaction is completed.



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| ProductKey | ProductSubcategoryKey | ProductSKU | ProductName | ModelName | ProductDescription |
|---|---|---|---|---|---|
| 602 | 5 | BB-8107 | ML Bottom Bracket | ML Bottom Bracket | Aluminum alloy cups; large diameter spindle. |
| 603 | 5 | BB-9108 | HL Bottom Bracket | HL Bottom Bracket | Aluminum alloy cups and a hollow axle. |
| 604 | 2 | BK-R19B-44 | Road-750 Black, 44 | Road-750 | Entry level adult bike; offers a comfortable ride ... |
| 605 | 2 | BK-R19B-48 | Road-750 Black, 48 | Road-750 | Entry level adult bike; offers a comfortable ride ... |
| 606 | 2 | BK-R19B-52 | Road-750 Black, 52 | Road-750 | Entry level adult bike; offers a comfortable ride ... |
| 101 | 5 | SKU12345 | New Product | Model X | NULL |
| 112 | 11 | SKU44444 | Product D | Model D | NULL |

**How TCL commands help in ACID properties:**

- **Atomicity:** Ensures all-or-nothing execution of transactions.

```
START TRANSACTION;
INSERT INTO orders (order_id, customer_id, order_date) VALUES (1, 101,
'2024-01-01');
INSERT INTO order_details (order_id, product_id, quantity) VALUES (1,
202, 5);

-- Suppose the following update fails due to a constraint violation
UPDATE products SET stock = stock - 5 WHERE product_id = 202;

-- If the update fails, the transaction is rolled back
ROLLBACK;
```

- **Consistency**: Ensures the database moves from one valid state to another.

```
START TRANSACTION;
INSERT INTO accounts (account_id, balance) VALUES (1, 1000);
INSERT INTO accounts (account_id, balance) VALUES (2, 2000);

-- Transfer $500 from account 1 to account 2
UPDATE accounts SET balance = balance - 500 WHERE account_id = 1;
UPDATE accounts SET balance = balance + 500 WHERE account_id = 2;

-- If any update fails, the transaction is rolled back to maintain
consistency
COMMIT;
```

- **Isolation**: Ensures that the intermediate states of a transaction are not visible to other transactions.

```
START TRANSACTION;

-- Transaction 1: Read balance of account 1
SELECT balance FROM accounts WHERE account_id = 1;

-- Transaction 2: Update balance of account 1
UPDATE accounts SET balance = balance + 100 WHERE account_id = 1;

-- Transaction 1: Should still see the original balance, not the updated
one by Transaction 2
SELECT balance FROM accounts WHERE account_id = 1;

COMMIT;
```

- **Durability**: Ensures that once a transaction is committed, it remains so, even in the event of a failure.

```
START TRANSACTION;
INSERT INTO logs (log_id, message) VALUES (1, 'Transaction started');
UPDATE accounts SET balance = balance - 100 WHERE account_id = 1;
UPDATE accounts SET balance = balance + 100 WHERE account_id = 2;
COMMIT;

-- Even if the system crashes now, the changes are permanent and will be
reflected after recovery
```

### Understanding the DCL Commands in MySQL:

### What is a DCL Command?

DCL is an abbreviation for Data Control Language in SQL. It is used to provide different users access to the stored data. It enables the data administrator to grant or revoke the required access to act as the database. When DCL commands are implemented in the database, there is no feature to perform a rollback. The administrator must implement the other DCL command to reverse the action.

### Benefits of implementing DCL:

There are several advantages of implementing Data Control Language commands in a database. Let's see some most common reasons why the user implements DCL commands on the database.

- **Security:** the primary reason to implement DCL commands in the database is to manage the access to the database and its object between different users. This limits the actions that can be performed by specific users on the different elements in the database. It ensures the security and integrity of the data stored in the database.

- **Granular control:** DCL commands provide granular control to the data administrator over the database. It allows the administrator to provide or remove specific privileges or permissions from other users using a database for information. Thus, it enables the admin to create different levels of access to the database.

- **Flexibility:** The data administrator can implement DCL commands on specific commands and queries in the database. It allows the administrator to grant or revoke user permissions and privileges as per their needs. It provides flexibility to the administrator that allows them to manage access to the database.

**Disadvantages of implementing DCL:**

- **Complexity:** It increases the complexity of database management. If many users are accessing the database, keeping track of permission and privileges provided to every user in the database becomes very complex.

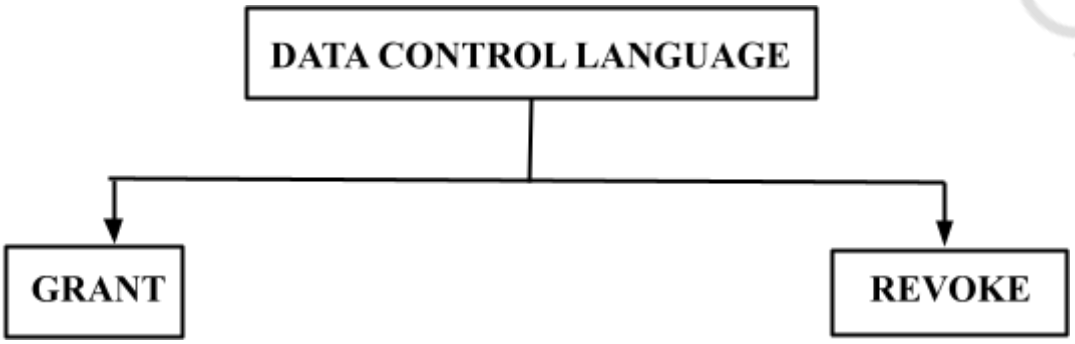
- **Time-Consuming:** In most organizations, several users access the database, and different users have different access levels to organization data. It is time-consuming to assign the permissions and privileges to each user separately.

- **Risk of human error:** Human administrators execute DCL commands and can make mistakes in granting or revoking privileges. Thus, giving unauthorized access to data or imposing unintended restrictions on access.

- **Lack of audit trail:** There may be no built-in mechanism to track changes to privileges and permissions over time. Thus, it is extremely difficult to determine who has access to the data and when that access was granted or revoked.

**Types of DCL Commands:**

Two types of DCL commands can be used by the user in SQL. These commands are useful, especially when several users access the database. It enables the administrator to manage access control. The two types of DCL commands are as follows:



## 1. Concepts on GRANT Command:

GRANT, as the name itself suggests, provides. This command allows the administrator to provide particular privileges or permissions over a database object, such as a table, view, or procedure. It can provide user access to perform certain database or component operations.

In simple language, the GRANT command allows the user to implement other SQL commands on the database or its objects. The primary function of the GRANT command in SQL is to provide administrators the ability to ensure the security and integrity of the data is maintained in the database.

The Syntax for the GRANT command is:

```
GRANT privilege_name
ON object_name
TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];
```

*NOTE:*

- *privilege_name is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, and SELECT.*
- *object_name is the name of an database object like TABLE, VIEW, STORED PROC and SEQUENCE.*
- *user_name is the name of the user to whom an access right is being granted.*
- *user_name is the name of the user to whom an access right is being granted.*
- *PUBLIC is used to grant access rights to all users.*
- *ROLES are a set of privileges grouped together.*
- *WITH GRANT OPTION - allows a user to grant access rights to other users.*

**Example:**

```
GRANT SELECT ON employee TO user1;
```

This command grants a SELECT permission on employee table to user1.You should use the WITH GRANT option carefully because for example if you GRANT SELECT privilege on employee table to user1 using the WITH GRANT option, then user1 can GRANT SELECT privilege on employee table to another user, such as user2 etc. Later, if you REVOKE the SELECT privilege on employee from user1, still user2 will have SELECT privilege on employee table.

*NOTE: In the context of the GRANT command in MySQL, user is an example username. When you see user1, it refers to a specific user account in the MySQL database system to which you are granting specific privileges.*

**Example 2:**

```
GRANT SELECT (Productname, productprice), INSERT (productSKU,
Productname) ON mydatabase.products TO 'user4'@'localhost';
```

**NOTE:**

- **GRANT:** This keyword starts the command to grant privileges.
- **SELECT (Productname, productprice):** This specifies that the user will have SELECT privileges on the Productname and productprice columns of the products table.
- **INSERT (productSKU, Productname):** This specifies that the user will have INSERT privileges on the productSKU and Productname columns of the products table.
- **ON mydatabase.products:** This specifies the database and table on which the privileges will be granted.
- **TO 'user4'@'localhost':**
    - **'user4':** The username of the MySQL account to which the privileges are being granted. This is an example, and in practice, you would replace user4 with the actual username of the user account.
    - **'localhost':** Specifies that this user can connect to the MySQL server from the local machine. This can be changed to a specific IP address or '% to allow connections from any host.

2. Concepts on Revoke Command:
   As the name suggests, revoke is to take away. The REVOKE command enables the database administrator to remove the previously provided privileges or permissions from a user over a database or database object, such as a table, view, or procedure. The REVOKE commands prevent the user from accessing or performing a specific operation on an element in the database.

   In simple language, the REVOKE command terminates the ability of the user to perform the mentioned SQL command in the REVOKE query on the database or its component. The primary reason for implementing the REVOKE query in the database is to ensure the data's security and integrity.

   Consider a scenario where the user is the database administrator. In the above implementation of the GRANT command, the user Aman was provided permission to implement a SELECT query on the student table that allowed Aman to read or retrieve the data from the table. Due to certain circumstances, the administrator wants to revoke the abovementioned permission. To do so, the administrator can implement the below REVOKE statement:

```
REVOKE SELECT ON student FROM Aman;
```

This will stop the user Aman from implementing the SELECT query on the student table. The user may be able to implement other queries in the database.

**Benefits of DCL Commands:**

- **Security:** the primary reason to implement DCL commands in the database is to manage the access to the database and its object between different users. This limits the actions that can be performed by specific users on the different elements in the database. It ensures the security and integrity of the data stored in the database.

- **Granular control:** DCL commands provide granular control to the data administrator over the database. It allows the administrator to provide or remove specific privileges or permissions from other users using a database for information. Thus, it enables the admin to create different levels of access to the database.

- **Flexibility:** The data administrator can implement DCL commands on specific commands and queries in the database. It allows the administrator to grant or revoke user permissions and privileges as per their needs. It provides flexibility to the administrator that allows them to manage access to the database.

Disadvantages of DCL Commands:

- Complexity: It increases the complexity of database management. If many users are accessing the database, keeping track of permission and privileges provided to every user in the database becomes very complex.

- Time-Consuming: In most organizations, several users access the database, and different users have different access levels to organization data. It is time-consuming to assign the permissions and privileges to each user separately.
- Risk of human error: Human administrators execute DCL commands and can make mistakes in granting or revoking privileges. Thus, giving unauthorized access to data or imposing unintended restrictions on access.

- Lack of audit trail: There may be no built-in mechanism to track changes to privileges and permissions over time. Thus, it is extremely difficult to determine who has access to the data and when that access was granted or revoked.

**Example 1:**

```
REVOKE SELECT (Productname, productprice), INSERT (productSKU,
Productname) ON mydatabase.products FROM 'user4'@'localhost';
```

This command revokes user4's ability to select the Productname and productprice columns and insert into the productSKU and Productname columns of the products table in the mydatabase database.

**Example 2:**

```
REVOKE SELECT, INSERT ON mydatabase.products FROM 'user5'@'localhost';
REVOKE SELECT, INSERT ON mydatabase.product_subcategories FROM
'user5'@'localhost';
```

These commands revoke user5's ability to select and insert data into both the products and product_subcategories tables in the mydatabase database.

### Understanding AI features in MySQL Workbench:

1. **Smart code completion:**

   This feature uses machine learning algorithms to predict and suggest relevant SQL syntax, table names, and column names as you type. It analyzes your database schema, query history, and common SQL patterns to provide context-aware suggestions. This speeds up query writing and reduces syntax errors.

2. **Query optimization suggestions:**

   The AI analyzes your SQL queries and compares them against performance best practices. It considers factors like indexing, join order, and subquery usage. The system then provides specific recommendations to improve query performance, such as suggesting alternative query structures or advising on index creation.

3. **Schema design assistance:**

   This feature uses AI to analyze your database schema and compare it against established design patterns and best practices. It can suggest improvements like normalizing tables, optimizing data types, or adding appropriate constraints. This helps in creating more efficient and maintainable database structures.

4. **Data modeling:**

   AI assists in creating entity-relationship diagrams by automatically suggesting relationships between tables based on column names and data types. It can also optimize the layout of complex diagrams for better readability. The system learns from common data modeling patterns to provide intelligent suggestions.

5. **Automated documentation:**

   The AI analyzes your database schema, relationships, and even query patterns to generate comprehensive documentation. This includes table descriptions, column details, relationship explanations, and common usage patterns. The documentation is kept up-to-date as your schema evolves.

6. **Predictive indexing:**

   By analyzing query patterns and database structure, the AI can suggest optimal indexes to improve overall database performance. It considers factors like query frequency, data distribution, and update patterns to balance read and write performance.

7. **Anomaly detection:**

   This feature uses machine learning to establish baseline patterns for your data and query execution. It can then alert you to unusual activities, such as unexpected data values, sudden changes in query performance, or potential security issues.