Common TCL Commands:

1. **COMMIT**
2. **ROLLBACK**
3. **SAVEPOINT**
4. **SET TRANSACTION**

1. COMMIT

The COMMIT command is used to save all the changes made during the current transaction permanently to the database.

-- Example

BEGIN;

UPDATE office2 SET salary = 60000 WHERE rollno = 1;

INSERT INTO office2 (rollno, empname, empwork, salary) VALUES (2, 'Alice', 'Manager', 70000);

COMMIT;

In this example, the COMMIT command ensures that both the UPDATE and INSERT operations are saved to the database.

2. ROLLBACK

The ROLLBACK command is used to undo changes made during the current transaction, effectively restoring the database to its state before the transaction began.

-- Example

BEGIN;

UPDATE office2 SET salary = 60000 WHERE rollno = 1;

DELETE FROM office2 WHERE rollno = 2;

ROLLBACK;

In this example, the ROLLBACK command undoes the UPDATE and DELETE operations, so the database remains unchanged.

3. SAVEPOINT

The SAVEPOINT command creates a point within a transaction to which you can later roll back without affecting the entire transaction. This is useful for partial rollbacks within a transaction.

-- Example

BEGIN;

UPDATE office2 SET salary = 60000 WHERE rollno = 1;

SAVEPOINT sp1;

```
DELETE FROM office2 WHERE rollno = 2;


-- Something went wrong, rollback to SAVEPOINT sp1

ROLLBACK TO sp1;


-- The DELETE is undone, but the UPDATE remains

COMMIT;
```

In this example, after the ROLLBACK TO sp1, only the DELETE operation is undone, while the UPDATE operation remains, and then the transaction is committed.

4. SET TRANSACTION

The SET TRANSACTION command is used to specify characteristics for the current transaction, such as the isolation level.

```
-- Example

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN;

UPDATE office2 SET salary = 60000 WHERE rollno = 1;

COMMIT;
```

In this example, the transaction is set to use the SERIALIZABLE isolation level, which is the strictest level and ensures that transactions are executed in a completely isolated manner.

Summary

- **COMMIT**: Permanently saves the changes made in a transaction.
- **ROLLBACK**: Reverts the changes made in a transaction.
- **SAVEPOINT**: Sets a point within a transaction to which you can later roll back.
- **SET TRANSACTION**: Configures the characteristics of a transaction, such as isolation level.

Common DCL Commands:

1. **GRANT**
2. **REVOKE**

1. GRANT

The GRANT command is used to give specific privileges to users or roles. These privileges can include the ability to perform operations like SELECT, INSERT, UPDATE, DELETE, and more on database objects (tables, views, etc.).

```
-- Example: Grant SELECT and INSERT privileges on the office2 table to user 'username'

GRANT SELECT, INSERT ON office2 TO 'username';
```

-- Example: Grant all privileges on the office2 table to user 'admin'

GRANT ALL PRIVILEGES ON office2 TO 'admin';

In these examples:

- The first command grants SELECT and INSERT privileges on the office2 table to the user username.
- The second command grants all possible privileges on the office2 table to the user admin.

2. REVOKE

The REVOKE command is used to remove previously granted privileges from users or roles.

-- Example: Revoke INSERT privilege on the office2 table from user 'username'

REVOKE INSERT ON office2 FROM 'username';

-- Example: Revoke all privileges on the office2 table from user 'admin'

REVOKE ALL PRIVILEGES ON office2 FROM 'admin';

In these examples:

- The first command revokes the INSERT privilege on the office2 table from the user username.
- The second command revokes all privileges on the office2 table from the user admin.

Summary

- **GRANT**: Assigns specific privileges to a user or role, allowing them to perform certain operations on database objects.
- **REVOKE**: Removes previously granted privileges from a user or role, restricting their ability to perform certain operations on database objects.

Database Design

Database design is the process of creating a detailed data model of a database. This involves defining the structure of the database, including tables, fields, relationships, and constraints. A well-designed database ensures efficient data storage, retrieval, and integrity, while minimizing redundancy and ensuring consistency.

The design process typically involves several stages, including:

1. **Requirement Analysis**: Understanding what the database needs to accomplish based on the needs of the users or the application.
2. **Conceptual Design**: Creating an abstract model of the database using tools like Entity-Relationship Diagrams (ERDs).
3. **Logical Design**: Translating the conceptual model into a logical structure that can be implemented in a specific database management system (DBMS).

4. **Physical Design**: Optimizing the database structure for performance, storage efficiency, and scalability.

Database Models

A **database model** defines the logical structure of a database and determines how data is stored, organized, and manipulated. There are several types of database models, with the most common being:

1. **Hierarchical Model**:
    - Data is organized into a tree-like structure.
    - Each record has a single parent, and records are linked through parent-child relationships.
    - Example: File systems.
2. **Network Model**:
    - An extension of the hierarchical model, allowing records to have multiple parent-child relationships.
    - Uses a graph structure to represent relationships.
    - Example: Early database systems like IDS (Integrated Data Store).
3. **Relational Model**:
    - Data is organized into tables (also known as relations), where each table represents an entity.
    - Tables are linked through foreign keys.
    - Most widely used model in modern databases.
    - Example: MySQL, PostgreSQL.
4. **Object-Oriented Model**:
    - Data is stored as objects, similar to objects used in object-oriented programming.
    - Supports inheritance, encapsulation, and polymorphism.
    - Example: ObjectDB, db4o.
5. **Document Model**:
    - Data is stored in document format, typically as JSON, BSON, or XML.
    - Often used in NoSQL databases.
    - Example: MongoDB.
6. **Key-Value Model**:
    - Data is stored as key-value pairs.
    - Simple and efficient for large amounts of data with no complex relationships.

- Example: Redis, DynamoDB.

Normalization (1NF, 2NF, 3NF, BCNF)

Normalization is the process of organizing the fields and tables of a relational database to minimize redundancy and dependency. It typically involves dividing large tables into smaller, more manageable pieces without losing data integrity. The most common forms of normalization are:

1. **First Normal Form (1NF)**:
    - Ensures that each table column contains atomic (indivisible) values and that each record is unique.
    - There should be no repeating groups or arrays in a table.
    - Example: A table with a column for phone numbers should store each phone number in a separate row, not as a comma-separated list.
2. **Second Normal Form (2NF)**:
    - Achieved when a table is in 1NF, and all non-key columns are fully dependent on the primary key.
    - Removes partial dependencies, where a non-key column depends only on part of a composite primary key.
    - Example: If a table has a composite primary key (OrderID, ProductID), all other columns should depend on both OrderID and ProductID, not just one.
3. **Third Normal Form (3NF)**:
    - Achieved when a table is in 2NF, and all non-key columns are directly dependent on the primary key, not on other non-key columns (no transitive dependency).
    - Example: If a table has columns EmployeeID, DepartmentID, and DepartmentName, DepartmentName should be removed because it depends on DepartmentID, not directly on EmployeeID.
4. **Boyce-Codd Normal Form (BCNF)**:
    - A stricter version of 3NF where every determinant (a column or set of columns that can uniquely identify the rest of the columns) is a candidate key.
    - Ensures there are no anomalies in the table structure.
    - Example: If a table has columns TeacherID, Subject, and TeacherAge, and both TeacherID and Subject together uniquely identify TeacherAge, the table should be split to ensure that each determinant is a candidate key.

Entity-Relationship Diagrams (ERD)

An **Entity-Relationship Diagram (ERD)** is a graphical representation of entities (tables) and their relationships to each other within a database. ERDs are a crucial part of the database design process, helping to visualize the structure of the database and how data elements are interconnected.

Components of an ERD:

1. **Entities**: Represented by rectangles, entities are objects or concepts that can have data stored about them. In a database, entities typically become tables.
    - Example: Customer, Order, Product.
2. **Attributes**: Represented by ovals connected to entities, attributes describe properties of the entities.
    - Example: For the Customer entity, attributes might include CustomerID, Name, Address, etc.
3. **Relationships**: Represented by diamonds, relationships show how entities are related to each other.
    - Example: A Customer can place an Order, so there might be a relationship between the Customer and Order entities.
4. **Primary Key**: A unique identifier for each record in an entity, typically underlined in an ERD.
    - Example: CustomerID in the Customer entity.
5. **Foreign Key**: An attribute that creates a link between two entities, pointing from one entity's attribute to another entity's primary key.
    - Example: CustomerID in the Order entity that references CustomerID in the Customer entity.

Cardinality: Indicates the nature of the relationship between entities (one-to-one, one-to-many, many-to-many).

- Example: A one-to-many relationship where one Customer can place many Orders.

Example of an ERD:

[Customer] ----(places)----< [Order]

| |

+-- CustomerID +-- OrderID

| +-- OrderDate

+-- Name +-- CustomerID (FK)

+-- Address +-- TotalAmount

This simple ERD shows that a Customer can place many Orders, with each Order linked to a Customer by the CustomerID foreign key.

Conclusion

- **Database Design**: The process of creating a structured and efficient database.
- **Database Models**: Different ways to organize and structure data.
- **Normalization**: The process of organizing data to reduce redundancy and improve data integrity.

- **Entity-Relationship Diagrams (ERD)**: Visual tools to represent the structure and relationships within a database.