KARNAVATI UNIVERSITY | UIT

Unitedworld Institute Of Technology

B.Tech. Computer Science & Engineering

Semester : 3rd

Introduction To Database Management System

Course Code: 71203002003

UNIT IV: TRANSACTIONS, CONCURRENCY CONTROL AND DEADLOCKS

Prepared by:
Mr. Utsav Kapadiya
Assistant Professor (UIT)

DCL and TCL Commands in SQL

Master the essential commands that control access permissions and manage database transactions in SQL. This guide provides practical, hands-on knowledge for implementing DCL and TCL commands effectively.

First, Let's Recall: Types of SQL Commands

SQL organizes its commands into four distinct categories, each serving a specific purpose in database management. Understanding these categories helps you recognize when and why to use different command types.

Type	Full Form	Purpose
DDL	Data Definition Language	Defines database structure (CREATE, ALTER, DROP)
DML	Data Manipulation Language	Works with data records (INSERT, UPDATE, DELETE)
DCL	Data Control Language	Controls access permissions and security
TCL	Transaction Control Language	Manages transaction lifecycle (save, commit, rollback)

Today's focus is on DCL and TCL commands – the essential tools for database security and transaction integrity.

DCL - Data Control Language

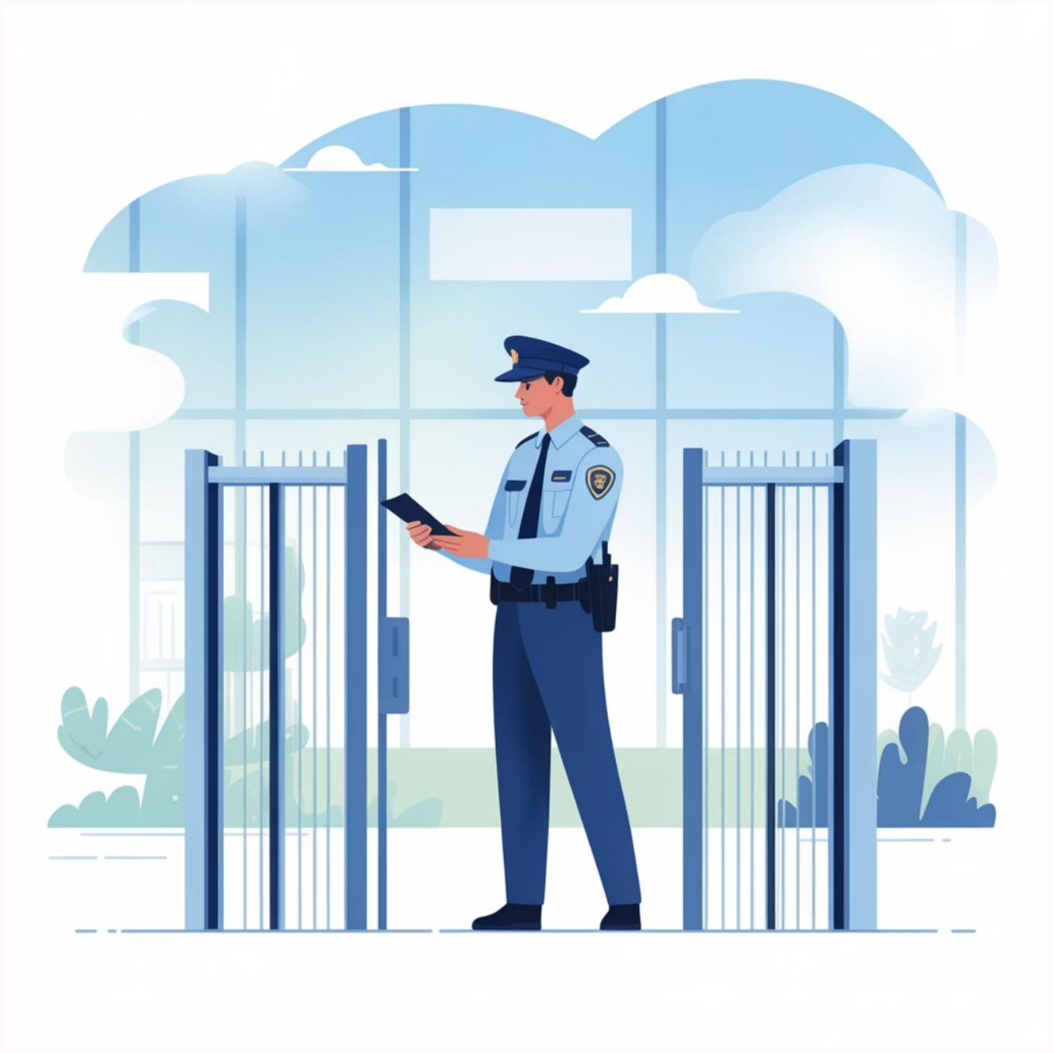
Data Control Language provides the foundation for database security by managing who can access what data and perform which operations.

What is DCL?

DCL commands are the gatekeepers of your database. They control access and permissions, allowing database administrators to grant or revoke rights to users with precision and security.

Think of DCL as a security guard standing at the database entrance, checking credentials and deciding "who can do what" inside. Without proper DCL implementation, your database security would be compromised.

DCL ensures that sensitive data remains protected while authorized users can perform their necessary tasks efficiently.



Main DCL Commands

GRANT

Gives specific permissions to a user, allowing them to perform designated operations on database objects

REVOKE

Takes back previously granted permissions from a user, restricting their access to database objects

These two commands work in tandem to maintain a secure and controlled database environment, ensuring the principle of least privilege is maintained.

GRANT Command: Syntax and Examples

The GRANT command is our primary tool for providing database access. It follows a straightforward syntax pattern that clearly defines what permission is being granted, on which object, and to whom.

Syntax Pattern

```
GRANT privilege_name ON object_name TO user_name;
```

1

Identify the Privilege

Choose the specific permission type (SELECT, INSERT, UPDATE, DELETE, or ALL)

2

Specify the Object

Name the table, view, or database object the permission applies to

3

Designate the User

Identify the user account receiving the permission

Example

```
GRANT SELECT, INSERT ON student TO robin;
```

This command gives user *robin* permission to view and add data to the student table, but not modify or delete existing records.

REVOKE Command: Taking Back Permissions

When permissions need to be withdrawn, whether due to role changes, security concerns, or policy updates , the REVOKE command provides precise control.

Syntax Pattern

```
REVOKE privilege_name ON object_name FROM user_name;
```

Example in Action

```
REVOKE INSERT ON student FROM
robin;
```

This removes the INSERT permission from user *robin* while preserving any other permissions they may have, such as SELECT.

Why REVOKE Matters

- Maintains security when roles change
- Prevents unauthorized data modifications
- Enforces compliance requirements
- Enables granular access control

Common Database Privileges

Understanding the available privileges is crucial for implementing effective database security. Each privilege grants specific capabilities to users.

SELECT

Allows users to read and view data from tables without making any changes

INSERT

Permits users to add new records and data entries to database tables

UPDATE

Enables users to modify existing data records and change field values

DELETE

Grants permission to remove records permanently from database tables

ALL

Provides complete access with all available permissions on the specified object

Real-Life Analogy: The College Library

Understanding DCL Through Everyday Experience

Think of your database like a college library system:

The librarian (Database Administrator) uses GRANT to issue you a library card with specific borrowing privileges

- You can borrow certain types of books based on your access level

If you misuse your privileges—like returning books late repeatedly—the librarian uses REVOKE to suspend your borrowing rights

- Different user types (students, faculty, staff) have different permission levels

This is exactly how DCL manages database access! Just as libraries protect their collections while serving authorized patrons, DCL protects data while enabling legitimate users.

TCL – Transaction Control Language

Transaction Control Language ensures data integrity by managing how changes are applied to your database, providing the foundation for reliable, consistent operations.

The Bank Transfer Analogy

Understanding Transactions Through Money Transfers

Consider a bank transfer scenario:

1 **Step 1**

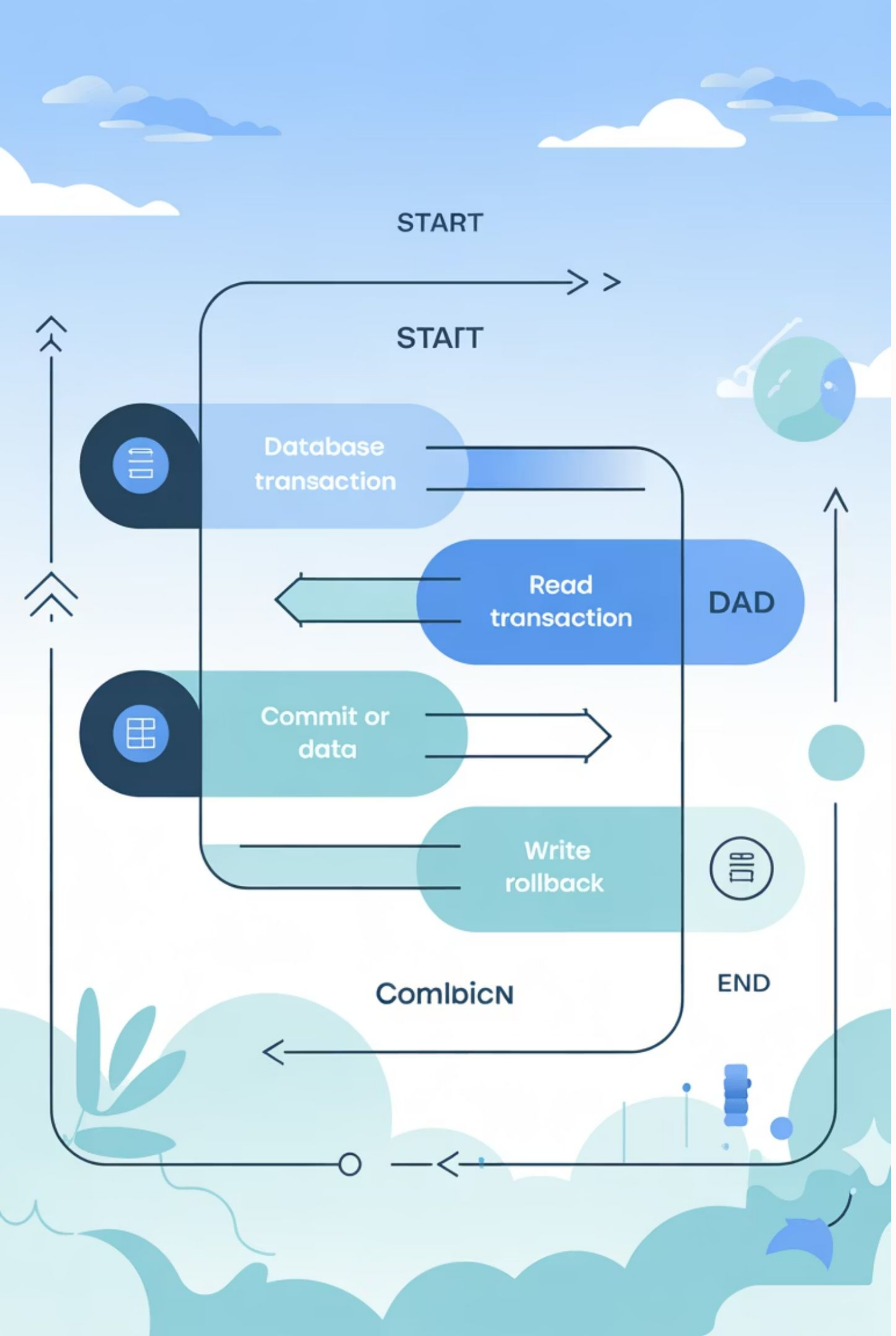
Deduct ₹500 from Account A

2 **Step 2**

Add ₹500 to Account B

Both operations must succeed together or both must fail. You can't have money disappear from one account without appearing in the other!

This is the essence of a transaction , ensuring data consistency through atomic operations.



Main TCL Commands

1

COMMIT

Permanently saves all changes made during the current transaction to the database. Once committed, data survives system crashes and power failures.

2

ROLLBACK

Undoes all changes made since the last COMMIT, restoring the database to its previous consistent state. Essential for error recovery.

3

SAVEPOINT

Creates a named checkpoint within a transaction, allowing partial rollback to that specific point without undoing all changes.

Setting Up Our Example

Let's work with a practical example to see TCL commands in action. We'll create a simple student table and perform various transaction operations to demonstrate each command's behavior.

Creating the Student Table

```
CREATE TABLE student (    id INT,    name VARCHAR(50),    marks INT);
```

Table Structure

id: Student identifier (integer)

name: Student name (up to 50 characters)

marks: Student score (integer)

What We'll Demonstrate

- Inserting data with COMMIT
- Undoing changes with ROLLBACK
- Using SAVEPOINTs for partial rollback
- Real transaction scenarios

Example 1: COMMIT Command

The COMMIT command is your confirmation that changes are ready to be permanently saved. Once executed, your data modifications become a permanent part of the database.

Transaction with COMMIT

```
INSERT INTO student VALUES (1, 'Aarav', 85);INSERT INTO student VALUES (2, 'Isha', 90);COMMIT;
```

0
1
Insert First Record

Aarav's data is added to the transaction buffer (not yet permanent)

0
2
Insert Second Record

Isha's data is also staged in the transaction buffer

0
3
Execute COMMIT

Both records are now permanently written to disk and visible to all users

Result: Changes are now permanently saved in the database. Even if the system crashes immediately after, both Aarav and Isha's records will remain intact.

Example 2: ROLLBACK Command

Undoing Uncommitted Changes

ROLLBACK is your safety net when something goes wrong. It allows you to cancel changes that haven't been committed yet, restoring the database to its last stable state.

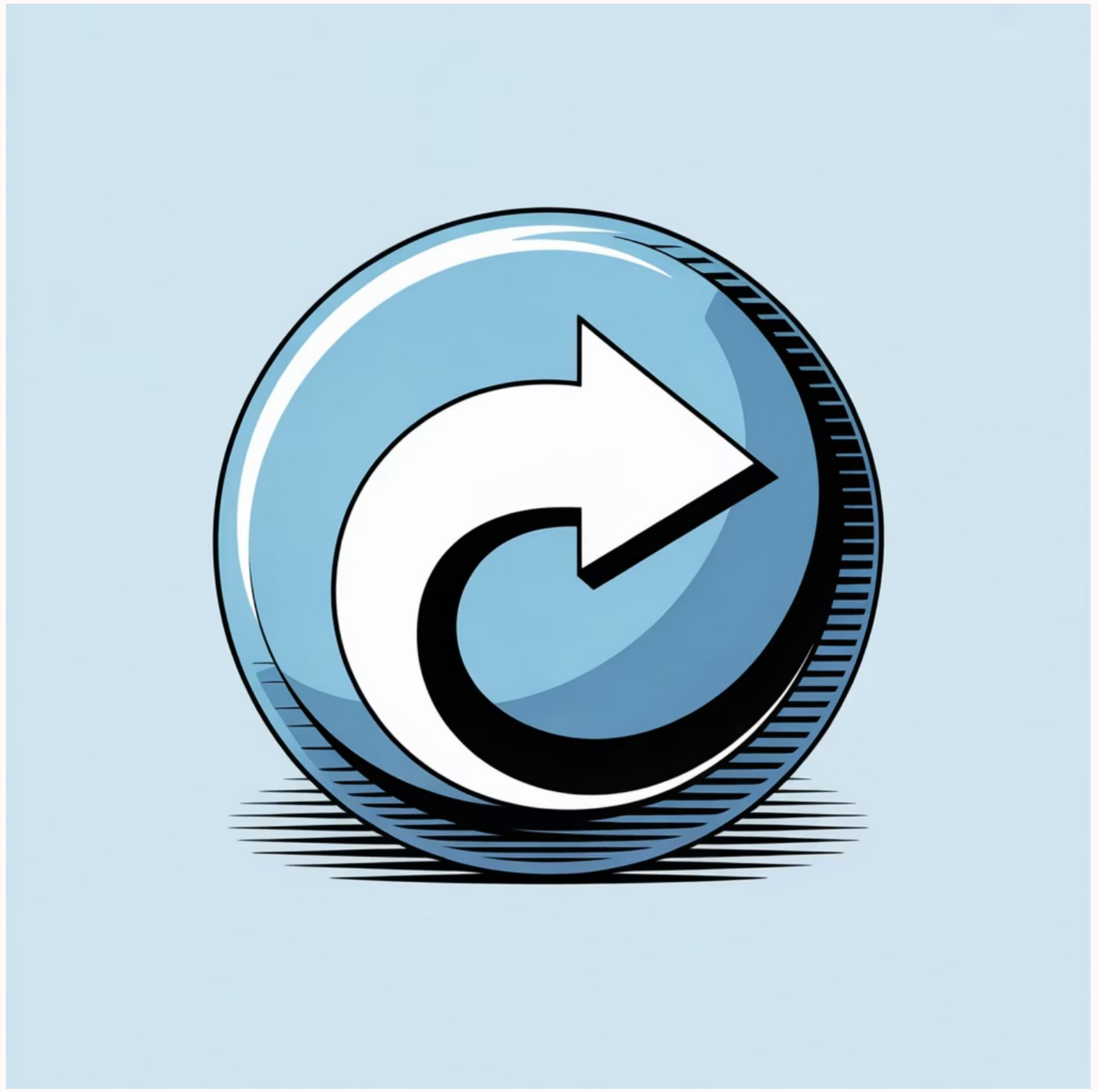
Transaction with ROLLBACK

```
INSERT INTO student VALUES (3, 'Karan', 70);ROLLBACK;
```

What happens: The INSERT operation for Karan is completely undone and discarded. It's as if the INSERT never occurred.

When to Use ROLLBACK

- When you detect an error in your data entry
- If a calculation produces unexpected results
- When business logic validation fails
- During exception handling in application code

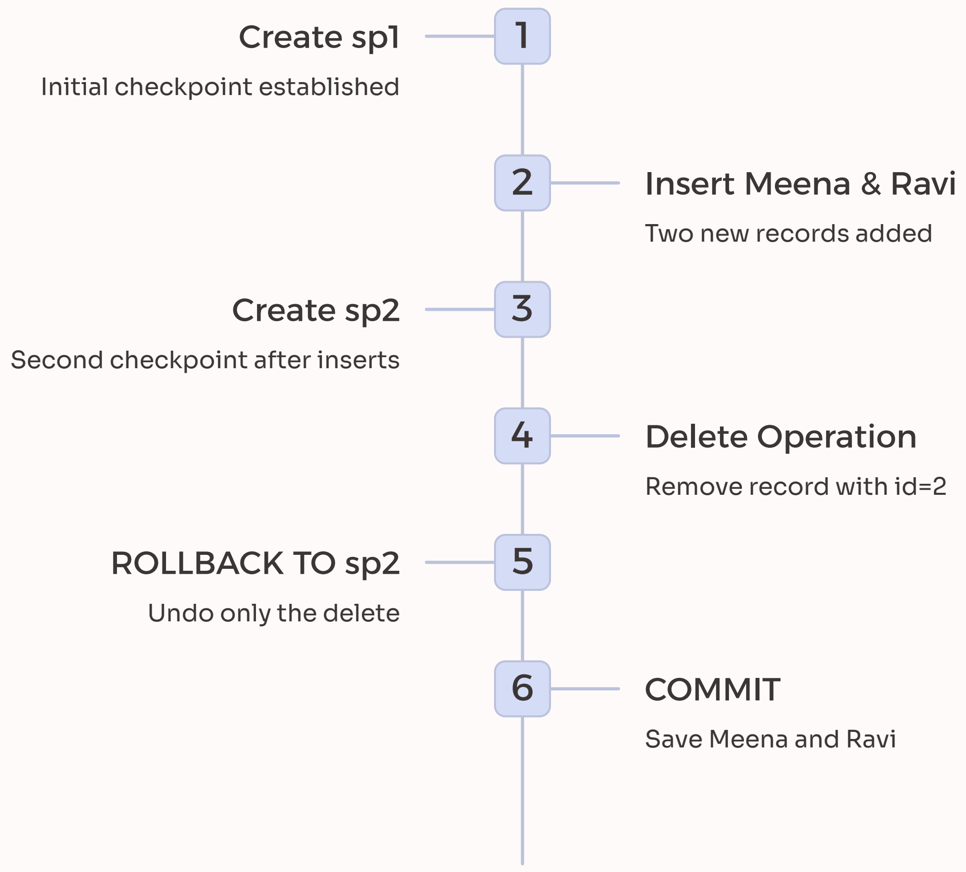


Example 3: SAVEPOINT and Partial Rollback

SAVEPOINTs provide fine-grained transaction control, allowing you to create checkpoints and selectively undo portions of your work without discarding everything.

Complex Transaction with Multiple SAVEPOINTs

```
SAVEPOINT sp1;INSERT INTO student VALUES (4, 'Meena', 88);INSERT INTO student VALUES (5, 'Ravi', 92);SAVEPOINT sp2;DELETE FROM student WHERE id = 2;ROLLBACK TO sp2;  -- Undo only the deleteCOMMIT;
-- Save rest of the changes
```



Final Result: Meena and Ravi are added permanently, but the delete operation was cancelled. This demonstrates selective rollback capability.

Transaction Flow Visualization

Understanding the typical flow of a transaction helps you structure your database operations effectively and handle errors appropriately.

1. Start Transaction

Begin your logical unit of work—changes are now tracked in the transaction buffer

2. Perform Operations

Execute INSERT, UPDATE, or DELETE statements—changes are temporary and reversible

3. Create SAVEPOINT (Optional)

Establish named checkpoints for potential partial rollback scenarios

4. ROLLBACK TO SAVEPOINT (If Needed)

Selectively undo changes back to a specific checkpoint while preserving other work

5. COMMIT Final Work

Permanently save all remaining changes to disk—transaction complete and durable

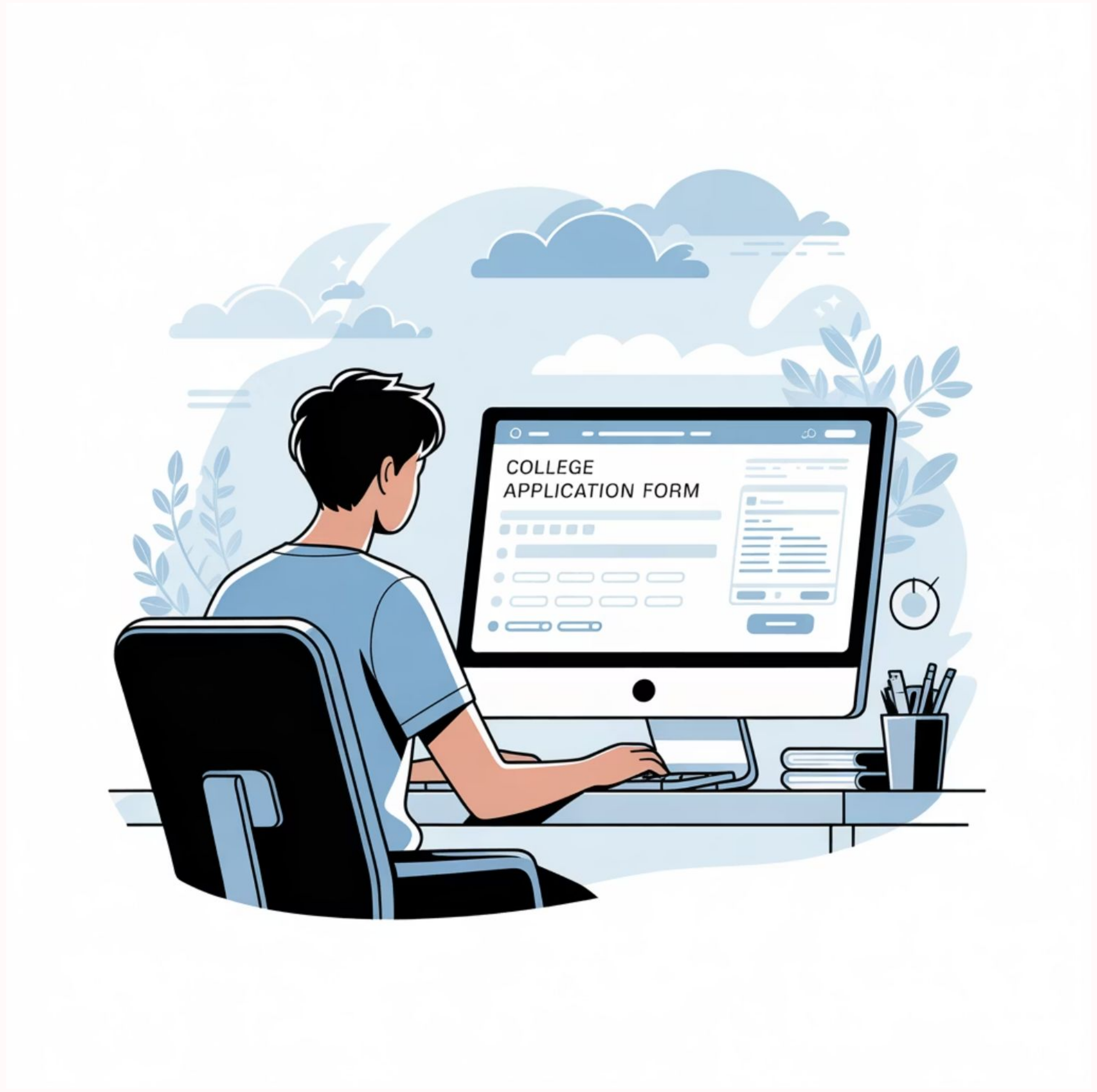
Complete DCL and TCL Command Reference

This comprehensive summary table serves as your quick reference guide for all DCL and TCL commands covered in this presentation.

Category	Command	Purpose & Usage
DCL	**GRANT**	Provide access permissions to users for specific database objects and operations
	REVOKE	Remove previously granted access permissions from users
TCL	**COMMIT**	Save all transaction changes permanently to the database with full durability
	ROLLBACK	Undo all changes made since the last COMMIT, restoring previous state
	SAVEPOINT	Create named checkpoints within transactions for selective partial rollback

Master these commands to ensure database security and transaction reliability in your applications.

The Online Form Submission Analogy



TCL in Everyday Life

Imagine filling out a college admission form online a perfect analogy for understanding transaction control:

1. **SAVEPOINT in Action**

   You save your progress halfway through the form, creating a recovery point in case something goes wrong

2. **ROLLBACK TO SAVEPOINT**

   You notice mistakes in a section and undo those entries back to your last save point without losing everything

3. **Final COMMIT**

   When all information is accurate and complete, you submit the form permanently

That's exactly how TCL commands work! They provide the same safety, flexibility, and control for your database operations that online forms provide for data entry.

Thank You!