UNIT-IV

- Relational Data Model:
- Relational model concepts, relational constraints,
- relational algebra
- SQL: SQL queries, programming using SQL.

. Relational Data Model

A **relational data model** represents data in the form of **tables (relations)**.

- A **relation** is a **table** with rows and columns.
- Each row is a **tuple**; each column is an **attribute**.
- Developed by **E.F. Codd** in 1970.

Example Table:

StudentID	Name	Age	Marks
S101	Shalini	21	85
S102	Rahul	22	90
S103	Anita	20	78

Relational Model Concepts

1.Tuple (Row) A tuple is a single record in a table.
Represents one instance of the entity.
Example: (S101, Shalini, 21, 85)

2.Attribute (Column) An attribute is a property or column of a relation.
Defines the type of data stored.
Example: Name, Age, Marks

3.Domain The domain of an attribute is the set of allowed values for that attribute.
Example: Age domain: 18–30
Marks domain: 0–100

4.Degree

- **Degree** = Number of attributes (columns) in a relation.
- **Example:** Student table has 4 attributes → **Degree = 4**

5.Cardinality

- **Cardinality** = Number of tuples (rows) in a relation.
- **Example:** Student table has 3 rows → **Cardinality = 3**

6.Keys

•**Keys** uniquely identify tuples in a table.

Key Type	Description	Example
Primary Key (PK)	Unique identifier, cannot be NULL	StudentID
Candidate Key	All possible unique identifiers	StudentID, EmailID
Foreign Key (FK)	References PK of another table	DeptID in Student referencing Department table
Composite Key	Combination of attributes to form a unique key	(CourseID, StudentID)

7. Integrity Rules

- **Entity Integrity:** Primary key cannot be NULL.
- **Referential Integrity:** Foreign key must match a primary key in another table or be NULL.

StudentID (PK)	Name	Age	DeptID (FK)
S101	Shalini	21	D01
S102	Rahul	22	D02
S103	Anita	20	D01

- **PK = StudentID** → Unique, Not NULL
- **FK = DeptID** → References Department table

Relational Constraints

Relational constraints are **rules applied on a table** to maintain **data integrity and accuracy**.

1.Primary Key (PK)

Uniquely identifies each row in a table.

•Rules:

- Must be **unique**.
- Cannot be **NULL**.

```
CREATE TABLE Student (
   StudentID INT PRIMARY KEY,
   Name VARCHAR(50),
   Age INT
);
```

StudentID is the primary key.

2.Foreign Key (FK)

- **Definition:** A key in one table that **references the primary key of another table**.
- **Purpose:** Maintain **referential integrity**.
- **Example:**

Department Table

```
CREATE TABLE Department (
    DeptID INT PRIMARY KEY,
    DeptName VARCHAR(50)
);
```

- `DeptID` is the **primary key** — it uniquely identifies each department.

Student Table

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(50),
    Age INT,
    DeptID INT,
    FOREIGN KEY (DeptID) REFERENCES Department(DeptID)
);
```

- DeptID in Student is a foreign key.
- It references the DeptID column in the Department table.

3.Unique Constraint

- **Definition:** Ensures that **no two rows have the same value** in a column.
- **Example:**

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Email VARCHAR(50) UNIQUE
);
```

Email must be **unique** for each student.

4.Not Null Constraint

- **Definition:** Ensures that a column **cannot have NULL values**.
- **Example:**

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(50) NOT NULL
);
```

5.Check Constraint

•**Definition:** Ensures that **values in a column satisfy a specific condition**.

•**Example:**

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(50),
    Age INT CHECK (Age >= 18)
);
```

6. Default Constraint

•**Definition:** Provides a **default value** for a column if none is specified.

•**Example:**

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(50),
    Status VARCHAR(10) DEFAULT 'Active'
);
```

Relational Constraints

Constraints ensure **data integrity**.

Constraint	Purpose	Example Column
Primary Key	Unique + Not NULL	StudentID
Foreign Key	Reference another table	DeptID
Unique	No duplicate values	Email
Not Null	Column must have a value	Name
Check	Values satisfy condition	Age >= 18
Default	Provide default value if not specified	Status = 'Active'

Relational Algebra (RA)

Relational Algebra is a **procedural query language** that uses a set of operations to manipulate and retrieve data from **relations (tables)** in a database.

It works on **one or more relations** to produce another relation.

Basic Operations of Relational Algebra

Student Table

StudentID	Name	Age	DeptID
1	Aditi	20	101
2	Bharat	17	102
3	Chaya	21	101
4	Deep	19	103

Department Table

DeptID	DeptName
101	ComputerSci
102	Commerce
103	Arts

Another Table for Examples: Alumni

StudentID	Name
3	Chaya
4	Deep

Relational Algebra Operations with Examples

1.Selection (σ)
Filter rows based on a condition.
Expression:
`σ Age > 18 (Student)`
Result: (students older than 18)

StudentID	Name	Age	DeptID
1	Aditi	20	101
3	Chaya	21	101
4	Deep	19	103

2.Projection (π)
Choose specific columns (attributes).
Expression:
`π Name, Age (Student)`

Name	Age
Aditi	20
Bharat	17
Chaya	21
Deep	19

3. Union (∪)

Combine two tables with same structure, removing duplicates.

Expression:

```
π StudentID, Name (Student) ∪ Alumni
```

StudentID	Name
1	Aditi
2	Bharat
3	Chaya
4	Deep

4. Set Difference (–)

Rows present in one relation but not in another.

Expression:

```
π StudentID, Name (Student) – Alumni
```

Result: (Students not Alumni)

StudentID	Name
1	Aditi
2	Bharat

5. Cartesian Product (×)

Combine each row of one table with each row of another table.

Expression:

`Student × Department`

Result: (Only first few rows shown)

StudentID	Name	Age	DeptID	DeptID	DeptName
1	Aditi	20	101	101	ComputerSci
1	Aditi	20	101	102	Commerce
…	…	…	…	…	…

StudentID	Name	Age	DeptID
1	Aditi	20	101

DeptID	DeptName
101	ComputerSci
102	Commerce
103	Arts

6. Rename (ρ)

Rename a table or its attributes.

Expression:

```
ρ S(Student)
```

This renames **Student** relation to **S**.

7. Intersection (∩)

Common rows between two tables.

Expression:

```
π StudentID, Name (Student) ∩ Alumni
```

StudentID	Name
3	Chaya
4	Deep

8. Natural Join (⋈)

Combines rows from two tables based on same attribute name.

Expression:

`Student ⋈ Department`

StudentID	Name	Age	DeptID	DeptName
1	Aditi	20	101	ComputerSci
2	Bharat	17	102	Commerce
3	Chaya	21	101	ComputerSci
4	Deep	19	103	Arts

9. Theta Join (⋈θ)

Join based on a specific condition.

Expression:

```
Student ⋈ Student.DeptID =
Department.DeptID AND Age>18
```

Result: (only students Age>18 joined with Dept)

StudentID	Name	Age	DeptID	DeptName
1	Aditi	20	101	ComputerSci
3	Chaya	21	101	ComputerSci
4	Deep	19	103	Arts

10. Division (÷)

Used to find tuples in one relation related to **all** tuples in another.

Example Scenario:

- **Takes(StudentID, CourseID)** – which student took which course
- **AllCourses(CourseID)** – all required courses

Expression:

```
Takes ÷ AllCourses
```

→ Students who took **all courses** listed in AllCourses.

(Division result depends on data; it returns StudentIDs who match **all** courses.)

Operator	Symbol	Description	Example
1. Selection	σ (sigma)	Selects rows (tuples) satisfying a condition.	σ Age>18 (Student) – selects students older than 18.
2. Projection	π (pi)	Selects columns (attributes).	π Name,Age (Student) – shows only Name and Age.
3. Union	∪	Combines tuples from two relations (no duplicates).	Student ∪ Alumni – all students or alumni.
4. Set Difference	−	Tuples in one relation but not in another.	Student − Alumni – students not alumni.
5. Cartesian Product	×	Combines each tuple of one relation with each tuple of another.	Student × Course – all combinations.
6. Rename	ρ (rho)	Renames the relation or its attributes.	ρ S(Student) – renames Student table to S.

Additional Operations

Operator	Symbol	Description	Example
7. Intersection	∩	Tuples common to both relations.	Student ∩ Alumni
8. Natural Join	⋈	Combines related tuples from two relations on common attributes.	Student ⋈ Department
9. Theta Join	⋈θ	Join using a specific condition.	Student ⋈ (Student.DeptID=Department.DeptID)
10. Division	÷	Finds tuples in one relation that are related to **all** tuples in another.	R ÷ S

1. SQL

SQL (Structured Query Language) is a standard language used to **interact with a database**.
It allows you to **create, modify, manage, and retrieve data** from relational databases.

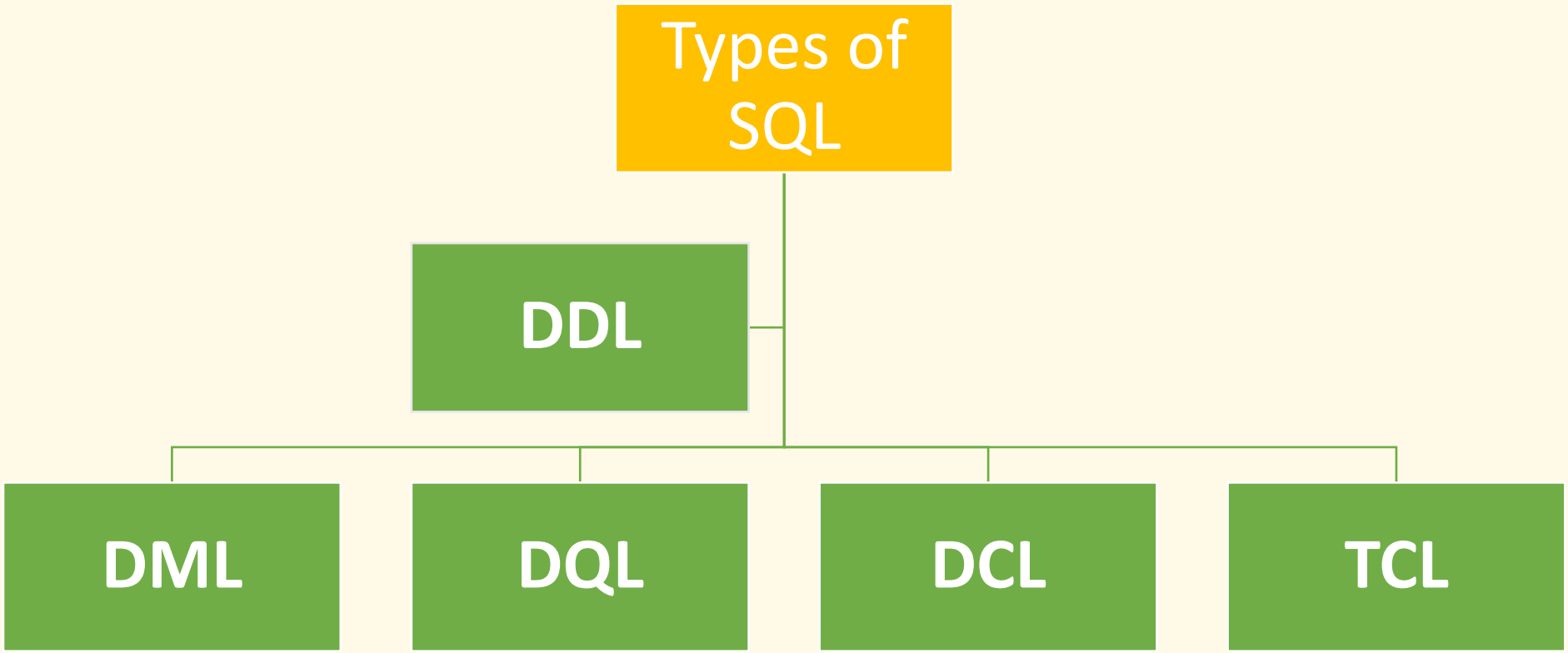
•SQL is **non-procedural**, meaning you specify *what* you want, not *how* to get it.

•It is used in systems like MySQL, Oracle, SQL Server, PostgreSQL, etc.

https://onecompiler.com

2. Types of SQL Commands
SQL commands are divided into **five main categories**:

Type	Full Form	Purpose	Example
DDL	Data Definition Language	Define or modify database structure	CREATE TABLE, ALTER TABLE, DROP TABLE
DML	Data Manipulation Language	Manipulate data inside tables	INSERT, UPDATE, DELETE
DQL	Data Query Language	Query or retrieve data	SELECT
DCL	Data Control Language	Control access and permissions	GRANT, REVOKE
TCL	Transaction Control Language	Manage transactions	COMMIT, ROLLBACK, SAVEPOINT



1. DDL (Data Definition Language)
Used to **define or modify database structure**

Command	Purpose	Example
CREATE	Create a table or database	CREATE TABLE Student (ID INT, Name VARCHAR(50), Age INT);
ALTER	Modify table structure	ALTER TABLE Student ADD COLUMN Marks INT;
DROP	Delete table or database	DROP TABLE Student;
TRUNCATE	Remove all rows from table (faster than DELETE)	TRUNCATE TABLE Student;

Feature	**CHAR**	**VARCHAR**
Length	Fixed	Variable
Storage	Pads with spaces	Stores only actual data
Speed	Faster for fixed data	Slightly slower
Use Case	Codes, short fixed text	Names, addresses, long text
Example	CHAR(5) → 'Hi '	VARCHAR(5) → 'Hi'

1. DDL (Data Definition Language)

1.Create Database

```
CREATE DATABASE CollegeDB;
```

2.Create Table

```
CREATE TABLE Student (
    ID INT PRIMARY KEY,
    Name VARCHAR(50),
    Age INT,
    Marks INT
);
```

3. Alter Table (Add Column)

```
ALTER TABLE Student ADD COLUMN Grade CHAR(1);
```

4. Alter Table (Modify Column)

```
ALTER TABLE Student MODIFY COLUMN Age SMALL INT;
```

5. Drop Table

```
DROP TABLE Student;
```

6. Truncate Table

```
TRUNCATE TABLE Student;
```

2. DML (Data Manipulation Language)
Used to **manipulate data** in tables.

Command	Purpose	Example
INSERT	Add new records	INSERT INTO Student (ID, Name, Age) VALUES (1, 'Shalu', 21);
UPDATE	Modify existing data	UPDATE Student SET Age = 22 WHERE ID = 1;
DELETE	Remove records	DELETE FROM Student WHERE ID = 1;

2. DML (Data Manipulation Language)

1.Insert Single Record

```
INSERT INTO Student (ID, Name, Age, Marks)
VALUES (1, 'Shalini', 21, 85);
```

2.Insert Multiple Records

```
INSERT INTO Student (ID, Name, Age, Marks) VALUES
(2, 'Rahul', 22, 90),
(3, 'Anita', 20, 78);
```

3.Update Data

```
UPDATE Student SET Marks = 88 WHERE ID = 1;
```

4.Delete Record

```
DELETE FROM Student WHERE ID = 3;
```

3. DQL (Data Query Language)
Used to **fetch data** from tables.

Command	Purpose	Example
SELECT	Retrieve data	SELECT * FROM Student; SELECT Name, Age FROM Student WHERE Age > 20;
WHERE	Filter rows	SELECT * FROM Student WHERE Marks > 80;
ORDER BY	Sort data	SELECT * FROM Student ORDER BY Name ASC;
GROUP BY	Group rows for aggregation	SELECT Age, COUNT(*) FROM Student GROUP BY Age;
HAVING	Filter groups	SELECT Age, COUNT(*) FROM Student GROUP BY Age HAVING COUNT(*) > 1;

1.Select All Columns

```
SELECT * FROM Student;
```

2.Select Specific Columns

```
SELECT Name, Marks FROM Student;
```

3.Select with Condition

```
SELECT * FROM Student WHERE Marks > 80;
```

4.Select with AND / OR

```
SELECT * FROM Student WHERE Age > 20 AND Marks > 80;
SELECT * FROM Student WHERE Age < 21 OR Marks > 85;
```

5.Order By (Ascending / Descending)

```
SELECT * FROM Student ORDER BY Marks DESC;
SELECT * FROM Student ORDER BY Name ASC;
```

6.Group By

```
SELECT Age, COUNT(*) AS Count FROM Student GROUP BY Age;
```

7.Having (Filter Groups)

```
SELECT Age, COUNT(*) AS Count
FROM Student
GROUP BY Age
HAVING COUNT(*) > 1;
```

4. DCL (Data Control Language)
Used to **control access/permissions**.

Command	Purpose	Example
GRANT	Give permission	GRANT SELECT ON Student TO User1;
REVOKE	Remove permission	REVOKE SELECT ON Student FROM User1;

1.**Grant Permission**

```
GRANT SELECT, INSERT ON Student TO User1;
```

2.Revoke Permission

```
REVOKE INSERT ON Student FROM User1;
```

5. TCL (Transaction Control Language)
Used to **manage transactions**.

Command	Purpose	Example
COMMIT	Save transaction permanently	COMMIT;
ROLLBACK	Undo transaction	ROLLBACK;
SAVEPOINT	Set a point to rollback to	SAVEPOINT sp1;

1.Commit Transaction

```
COMMIT;
```

2.Rollback Transaction

```
ROLLBACK;
```

3.Savepoint

```
SAVEPOINT sp1;
ROLLBACK TO sp1;
```

```
CREATE TABLE students (
  id INTEGER PRIMARY KEY,
  name TEXT,
  course TEXT,
  marks INTEGER
);

INSERT INTO students (id, name, course, marks) VALUES
(1, 'Asha', 'DBMS', 85),
(2, 'Ravi', 'DBMS', 72),
(3, 'Sita', 'AI', 91),
(4, 'Rahul', 'DBMS', 66),
(5, 'Meena', 'AI', 78);

SELECT * FROM students;
```

SQL Server Download

Output

id	name	course	marks
1	Asha	DBMS	85
2	Ravi	DBMS	72
3	Sita	AI	91
4	Rahul	DBMS	66
5	Meena	AI	78

```
SELECT * FROM students;
SELECT name, course, marks
FROM students
WHERE marks > 75;
```

```
SELECT *
FROM students
ORDER BY marks DESC;
```

```
CREATE TABLE Student (
 ID INT PRIMARY KEY,
Name VARCHAR(50),
Age INT,
 Marks INT);
INSERT INTO Student (ID, Name, Age, Marks)
VALUES (2, 'Rahul', 22, 90),
(3, 'Anita', 20, 78);
SELECT * FROM Student;
```

OUTPUT

ID	Name	Age	Marks
2	Rahul	22	90
3	Anita	20	78

```
CREATE TABLE students (
  id INTEGER PRIMARY KEY,
  name TEXT,
  course TEXT,
  marks INTEGER
);

INSERT INTO students (id, name, course, marks)
VALUES
(1, 'Asha', 'DBMS', 85),
(2, 'Ravi', 'DBMS', 72),
(3, 'Sita', 'AI', 91),
(4, 'Rahul', 'DBMS', 66),
(5, 'Meena', 'AI', 78);
SELECT * FROM Student;
```

```
-- Step 3: View all data (SELECT)
SELECT * FROM students;
-- Step 4: Show students with marks greater than 75
(WHERE)
SELECT name, course, marks
FROM students
WHERE marks > 75;

-- Step 5: Sort students by marks (ORDER BY)
SELECT * FROM students
ORDER BY marks DESC;

-- Step 6: Update marks (UPDATE)
UPDATE students
SET marks = 95
WHERE name = 'Ravi';
```


```
-- Step 7: Check the update
SELECT * FROM students WHERE name = 'Ravi';

-- Step 8: Delete a record (DELETE)
DELETE FROM students
WHERE name = 'Rahul';

-- Step 9: Check table after deletion
SELECT * FROM students;

-- Step 10: Add a new column (ALTER TABLE)
ALTER TABLE students ADD COLUMN city TEXT;
```

```
Step 11: Update new column
UPDATE students
SET city = 'Delhi'
WHERE course = 'DBMS';

UPDATE students
SET city = 'Mumbai'
WHERE course = 'AI';

-- Step 12: Display table with city
SELECT * FROM students;

-- Step 13: Aggregate functions (GROUP BY)
SELECT course, AVG(marks) AS avg_marks
FROM students
GROUP BY course;
```

Step 14: Create second table for JOIN example

```
CREATE TABLE courses (
  course TEXT PRIMARY KEY,
  teacher TEXT
);
INSERT INTO courses VALUES
('DBMS', 'Dr. Neha'),
('AI', 'Dr. Rakesh’);
```

Step 15: JOIN two tables

```
SELECT s.name, s.course, s.marks, c.teacher
FROM students s
JOIN courses c
ON s.course = c.course;
```

Step 16: Show top scorer (LIMIT)

```
SELECT name, course, marks
FROM students
ORDER BY marks DESC
LIMIT 1;
```


```
COMPLETE SQL SCRIPT (DDL, DML, DQL, DCL, TCL)
-- 💾 CREATE DATABASE (DDL)
CREATE DATABASE CollegeDB;

-- 🏫 USE DATABASE
USE CollegeDB;

-- 🧱 CREATE TABLE (DDL)
CREATE TABLE Students (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  course VARCHAR(50),
  marks INT
);
```

```
-- ✨ INSERT VALUES (DML)
INSERT INTO Students (id, name, course,
marks) VALUES
(1, 'Asha', 'DBMS', 85),
(2, 'Ravi', 'DBMS', 72),
(3, 'Sita', 'AI', 91),
(4, 'Rahul', 'DBMS', 66),
(5, 'Meena', 'AI', 78);
```


```
-- 📋 SELECT DATA (DQL)
SELECT * FROM Students;

-- 🖍️ UPDATE DATA (DML)
UPDATE Students SET marks = 88 WHERE name = 'Ravi';

-- ❌ DELETE A RECORD (DML)
DELETE FROM Students WHERE name = 'Rahul';

-- 🔍 SELECT WITH CONDITION (DQL)
SELECT * FROM Students WHERE marks > 80;

-- 📃 ALTER TABLE (DDL)
ALTER TABLE Students ADD email VARCHAR(100);
```


```
-- 🔄 MODIFY COLUMN TYPE (DDL)
ALTER TABLE Students ALTER COLUMN marks TYPE FLOAT;

-- 🏷️ RENAME COLUMN (DDL)
ALTER TABLE Students RENAME COLUMN name TO student_name;

-- ❌ DROP COLUMN (DDL)
ALTER TABLE Students DROP COLUMN email;

-- 🪪 RENAME TABLE (DDL)
RENAME TABLE Students TO College_Students;

-- 🧹 TRUNCATE TABLE (DDL)
TRUNCATE TABLE College_Students;
```

- 🗑️ DROP TABLE (DDL)

```
DROP TABLE College_Students;

-- 💣 DROP DATABASE (DDL)
DROP DATABASE CollegeDB;

-- ============================
-- 🔐 DATA CONTROL LANGUAGE
-- ============================

-- GRANT PERMISSION (DCL)
GRANT SELECT, INSERT ON CollegeDB.* TO 'user1'@'localhost';

-- REVOKE PERMISSION (DCL)
REVOKE INSERT ON CollegeDB.* FROM 'user1'@'localhost’;
```