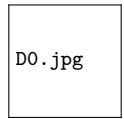
Lecture 08: Integrity Constraints

Domain, NULL and Keys

Z2004: Database Management Systems



IIT Madras Zanzibar

Friday, 13 March 2026

Why Integrity Constraints Matter

The First Line of Defense

An **integrity constraint** is a strict mathematical rule that data must satisfy at all times. They are enforced **automatically by the DBMS**.

The 5 Types of Constraints:

1. **Domain** – Valid values for a column.
2. **Key** – Uniqueness (`PK`, `UNIQUE`).
3. **Entity** – PK cannot be `NULL`.
4. **Referential** – FK values must be valid.
5. **Semantic** – Custom business rules.

⚠ Without Constraints

If we rely solely on application code (which can be bypassed, buggy, or updated), the database **silently corrupts**:

- A student is saved with a GPA of 99.
- An employee is assigned to Dept 500 (non-existent).
- The same Student ID is assigned twice.
- A salary is stored as a negative number.
- An invoice references a deleted customer.

Let DBMS enforce the rules. **Never** trust the application layer to maintain data integrity.

Domain Constraints and CHECK

Restricting Column Values

Built-in Domain Restrictions:

- `INT`, `DECIMAL` – numeric ranges
- `VARCHAR(n)` – string length limits
- `NOT NULL` – mandatory values

User-Defined Rules (`CHECK`):

```
GPA    DECIMAL(3,2) CHECK (GPA >= 0 AND
    GPA <= 4.0),
Grade CHAR(1)       CHECK (Grade IN ('A',
    'B','C','F')),
Age    INT          CHECK (Age >= 18)
```

Three-Valued Logic

Because `NULL` means *unknown*, standard boolean logic breaks down. SQL uses **TRUE, FALSE, and UNKNOWN**.

- `NULL = NULL` → `UNKNOWN`
- `NULL > 5` → `UNKNOWN`

⚠ The WHERE Trap

The `WHERE` clause only keeps rows that evaluate to exactly **TRUE**. **UNKNOWN rows are excluded.**

Always use `IS NULL` or `IS NOT NULL` instead of `= NULL`.

Key Constraints and Entity Integrity

Uniqueness Constraints

Both prevent duplicate values, but have strictly different rules:

- `PRIMARY KEY`**:** Only ONE per table. `NULL`s are strictly forbidden.
- `UNIQUE`**:** Multiple allowed per table. `NULL`s *are* allowed (and treated as distinct).

```
CREATE TABLE Student (
  StudentID   CHAR(6)      PRIMARY KEY,
  Email       VARCHAR(80)  UNIQUE NOT
      NULL,
  NationalID  CHAR(10)     UNIQUE,
  GPA         DECIMAL(3,2)
);
```

Entity Integrity Rule

No attribute in the **Primary Key** of a base relation may hold a `NULL` value.

Why? *A NULL primary key means we cannot uniquely identify the entity—it effectively does not exist in the real world.*

Composite Primary Keys

```
PRIMARY KEY (StudentID, CourseID)
```

- **Neither** attribute may be `NULL`.
- The **combination** of the two must be unique.

Referential Integrity: Handling Deletions

The Ripple Effect

If `Department 5` is deleted, what happens to the Employees assigned to it? The DBMS gives us 4 options:

`NO ACTION`	Reject the deletion entirely.
`CASCADE`	Automatically delete referencing Employees.
`SET NULL`	Change DeptID to `NULL`.
`SET DEFAULT`	Set DeptID to a default value.

Implementation & Design

```
FOREIGN KEY (StudentID)
  REFERENCES Student(StudentID)
  ON DELETE CASCADE
  ON UPDATE CASCADE,

FOREIGN KEY (CourseID)
  REFERENCES Course(CourseID)
  ON DELETE NO ACTION
  ON UPDATE CASCADE
```

Pro-Tip

Use `CASCADE` for weak entities (delete a Course's sections when the Course dies). Use `NO ACTION` to protect critical data.

From Structure to Querying: Relational Algebra

We Built It. Now What?

Up until now, we have focused on **DDL** (Data Definition Language) — creating tables and securing them with constraints.

But how do we extract meaningful answers from this data? We need a formal mathematical query language.

Enter Relational Algebra:

- A procedural query language.
- Takes one or two relations as input.
- **Produces a new relation as output.**

The Basic Operators (Preview)

Just as basic math uses $+$, $-$, $\times$, $\div$, Relational Algebra has 6 fundamental operations:

σ (Select) Pick specific **rows**.

π (Project) Pick specific **columns**.

$\cup$ (Union) Combine tables.

$-$ (Difference) Find what's in A but not B.

$\times$ (Cartesian) Combine everything.

ρ (Rename) Rename outputs.

These operations form the hidden engine that actually executes your SQL queries!

Real-World Scenario: The Unconstrained University App

What happens when the frontend fails?

Imagine a web developer forgets to add validation on a registration form, and the database has **no constraints**. What gets inserted into the Database?

StudentID	Name	Age	Major_ID	GPA
NULL	Alice	20	CS	3.8
102	Bob	-5	ART	9.5
102	*(Blank)*	22	FAKE_DEPT	2.1

Disaster 1

Alice has no ID! (Entity Integrity violation). She can never graduate because we can't look her up.

Disaster 2

Two Student 102s! (Key constraint violation). If 102 pays tuition, whose account is credited?

Disaster 3

Bob is -5 years old? (Domain constraint). And studying in a department that doesn't exist! (Referential).

Understanding NULL: It is NOT Zero!

The Survey Analogy

Imagine you are filling out a census survey. The question asks: **"How many cars do you own?"**



Scenario A: You write '0'

You do not own a car. The value is explicitly known.

Value = 0

Scenario B: You skip it

You might own a car, you might not. You just refused to answer.

Value = NULL



Why it Matters

If we ask the database: *"Show me everyone with fewer than 1 car"* (`Cars < 1`)

Scenario A is included. Scenario B is **ignored**.

Scenario: Closing a Department

Suppose IITM Zanzibar closes the "Arts" Department.

We execute: `DELETE FROM Department WHERE DeptName = 'Arts';`
But Professor John is assigned to 'Arts'. What does the Database do?

ON DELETE CASCADE

The Ruthless Approach:
If the department is deleted, the DBMS automatically deletes Professor John from the system entirely.

(Usually bad for Employees, good for Order Items)

ON DELETE SET NULL

The Limbo Approach:
The department is deleted. Professor John is kept in the database, but his `DeptID` is updated to `NULL` (He has no department).

(Requires DeptID to allow NULLs)

ON DELETE NO ACTION

The Safety Catch:
The DBMS **blocks** the deletion! It throws an error: "Cannot delete Arts, an employee depends on it." You must move John first.

Class Activity: Be the DBMS!

Spot the Integrity Violations

Table: `Employee`

- **PK:** `EmpID`
- **FK:** `DeptID` $\rightarrow$ `Dept` (Valid Depts: 10, 20)
- **Domain:** `Salary > 0`

Row	EmpID	Name	DeptID	Salary
1	1001	Alice	10	50000
2	1001	Bob	20	45000
3	1003	Charlie	99	60000
4	1004	Diana	NULL	-500
5	NULL	Eve	10	55000

Questions?

Which constraint fails for rows 2, 3, 4, and 5?

- **Row 2: Key constraint**
  Duplicate PK (1001 already exists).
- **Row 3: Referential integrity**
  Dept 99 does not exist.
- **Row 4: Domain constraint**
  Salary cannot be negative.
- **Row 5: Entity integrity**
  PK cannot be NULL.

Lecture Summary

Key Takeaways

- **Domain & NULL:** Use `CHECK` for ranges. Remember 3-valued logic (`IS NULL`).
- **Keys:** `PK` is unique + NOT NULL. `UNIQUE` allows NULLs.
- **Entity Integrity:** PK columns must never be NULL.
- **Referential Integrity:** FK must match an existing PK or be NULL.
- **Cascading:** Use `ON DELETE CASCADE` to clean up dependent child records.

Next Lecture (L09)

ER to Relational Mapping

- → Strong vs. weak entities (and identifying relationships)
- → Mapping 1:1 and 1:N relationships using foreign keys
- → Mapping M:N relationships via a junction table (with relationship attributes)
- → Handling multivalued attributes with separate tables
- → ISA / specialization: common mapping strategies