Reading: Understanding Relational Model Constraints

Estimated time: 15 minutes

Objectives

After completing this reading, you will be able to:

- Define and identify entity integrity, referential integrity, and domain integrity constraints
- Explain how each constraint maintains data integrity
- Recognize examples of how these constraints are implemented in SQL

Overview

In any well-designed database, maintaining data integrity is essential to ensure the accuracy, consistency, and reliability of the stored information. This lab focuses on three key relational model constraints:

- **Entity Integrity**
- **Referential Integrity**
- **Domain Integrity**

These constraints enforce rules on how data is stored and related within tables.

Imagine you are building a Database **BookShopDB**. You need to ensure that:

- Each book and author has a **unique identifier**.
- Every book must be linked to a valid **author**.
- Attributes like **price, title, and date** must have values within an acceptable range and format.

This reading will walk you through these constraints with examples to solidify your understanding.

Sample view of the tables used in this reading

1. BookShop table

This is a sample table structure and data view for the **BookShop** table used in this reading.

BOOK_ID	TITLE	AUTHOR_ID
1	The Silent Patient	101
2	Educated	102
3	1984	103
4	To Kill a Mockingbird	104
5	The Great Gatsby	105

2. BookShop_AuthorDetails table

This is a sample table structure and data view for the **BookShop_AuthorDetails** table used in this reading.

AUTHOR_ID	AUTHOR_NAME
101	Alex Michaelides
102	Tara Westover
103	George Orwell
104	Harper Lee
105	F. Scott Fitzgerald

Types of relational model constraints

Entity integrity constraint

This constraint ensures that every table in a relational database has a **primary key**. A primary key uniquely identifies each row in the table. A primary key column(s):

- must not contain **NULL** values
- must be **unique across all rows**

This constraint guarantees that each record (or entity) in a table is **distinct and identifiable**, preventing duplication and missing identifiers.

Example:

```
CREATE TABLE BookShop (
    BOOK_ID INT PRIMARY KEY,
    TITLE VARCHAR(100),
    AUTHOR_ID INT
);
```

Explanation

Here, **BOOK_ID** is the **primary key**. It must be:

- Unique (no two books can have the same ID)
- Not **NULL** (every book must have an ID)

Note: Every table in a relational database should have a **primary key** to satisfy entity integrity.

Referential integrity constraint

This constraint ensures that a **foreign key** in one table always refers to a valid **primary key** in another table. This maintains **consistent and meaningful relationships** between tables.

It enforces the **logical link between related data** in different tables, preventing the existence of invalid or "orphaned" references.

Example

```
CREATE TABLE BookShop_AuthorDetails (
    AUTHOR_ID INT PRIMARY KEY,
    AUTHOR_NAME VARCHAR(100)
);
CREATE TABLE BookShop (
    BOOK_ID INT PRIMARY KEY,
    TITLE VARCHAR(100),
    AUTHOR_ID INT,
    FOREIGN KEY (AUTHOR_ID) REFERENCES BookShop_AuthorDetails(AUTHOR_ID)
);
```

Explanation

AUTHOR_ID in **BookShop** references **AUTHOR_ID** in **BookShop_AuthorDetails**.

This means every **AUTHOR_ID** in **BookShop** must exist in **BookShop_AuthorDetails**.

*Trying to insert a book with an **AUTHOR_ID** that doesn't exist in **BookShop_AuthorDetails** will fail.*

Domain integrity constraint

This constraint ensures that all values stored in a column fall within a **defined domain**. This includes rules about:

- Data type
- Format
- Allowed values
- Nullability

It helps ensure that data in a column is **valid, logical, and consistent** with its intended use.

Example:

```
CREATE TABLE BookShop (
    BOOK_ID INT PRIMARY KEY,
    TITLE VARCHAR(100) NOT NULL,
    PRICE DECIMAL(5, 2) CHECK (PRICE >= 0),
    PUBLISHED_DATE DATE
);
```

Explanation

Two standard mechanisms used to enforce domain integrity are the **CHECK** and **NOT NULL** constraints:

1. CHECK constraint

The **CHECK** constraint enforces rules about the range or pattern of acceptable values within a column. It ensures that the data entered meets specific logical conditions.

- The **PRICE** column in the example above has a **CHECK** constraint to ensure it's not negative.

2. NOT NULL constraint

The **NOT NULL** constraint enforces nullability rules, ensuring that a column must contain a value and cannot be left empty **(NULL)**.

- **TITLE** is **NOT NULL**. A title must be provided.

Summary

In this reading, you explored three key types of constraints used in relational databases. By applying these constraints, you can create accurate, consistent, and easy-to-manage databases.

Author(s)

Monika Singh