ScopeTreasure Courses

SQL COURSES · LESSON DB.7 · CODE SCOPE SERIES

Constraints Deep — Database Ke Sakht Usool

Stop bad data at the door: NOT NULL, UNIQUE, DEFAULT, CHECK, and what happens to children when a parent row is deleted.

Introduction

DB.6 gave our rows identity and guarded links. But a row can have a valid id and still be nonsense: a student with **no name**, two students sharing one **email**, marks of **150 out of 100**, a fee with **no amount**. App code is supposed to check these things — and one day, in one forgotten form, it will not. **Constraints** are the database's own bouncers: rules written into the table that refuse bad data no matter which door it tries.

This lesson goes deep on the four everyday constraints — NOT NULL, UNIQUE, DEFAULT, CHECK — and ends with the question DB.6 left open: when a parent row *is* deleted, what should happen to its children?

(Roman Urdu: Constraint = darwaze ka qaida. Ghalat data andar ghus hi nahi sakta, chahe app so jaye.)

Key points

- **NOT NULL — this column may never be empty:** `name VARCHAR(50) NOT NULL`. An INSERT that leaves it out (or passes NULL) is refused. Use it for facts a record is meaningless without: names, amounts, the linking ids.
- **UNIQUE — no two rows may share this value:** perfect for emails, usernames, roll numbers. A duplicate insert fails with the same 1062 error as a duplicate key. A column can be UNIQUE without being the primary key — a table has one identity, but several "no repeats" promises.
- **DEFAULT — if nothing is given, use this:** `marks INT DEFAULT 0`, or a status that starts as `'Pending'`. Defaults make INSERTs shorter and guarantee a sensible value instead of an accidental NULL.
  (Roman Urdu: DEFAULT woh qeemat hai jo khamoshi par khud lag jati hai.)
- **CHECK — a condition every row must pass:** `CHECK (marks BETWEEN 0 AND 100)`. The database tests it on every INSERT and UPDATE; a row that fails never lands. Ranges, minimums, allowed lists — if a rule can be written as a condition, CHECK can guard it. (MySQL enforces CHECK from version 8.0.16; older versions accepted the words and ignored them — know your version.)
- **Constraints compose:** one column can carry several — `email VARCHAR(100) NOT NULL UNIQUE` is a normal, healthy sight.
- **ON DELETE — choose the children's fate in advance:** when a parent (student) with children (fees) is deleted, the FOREIGN KEY offers behaviours: **RESTRICT** (refuse the delete — the safe default), **CASCADE** (delete the children with the parent — powerful, dangerous), **SET NULL** (keep the children, blank their link — only if the link column allows NULL). The choice belongs in the design, not in a panic later.

Code examples

Example 1 — NOT NULL + UNIQUE: the guarded identity columns

```
CREATE TABLE students (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(50) NOT NULL,
  email VARCHAR(100) NOT NULL UNIQUE,
  class VARCHAR(10),
  marks INT,
  city VARCHAR(50)
);

INSERT INTO students (name, class, marks, city)
VALUES ('Ali Raza', '10-A', 88, 'Sahiwal');

INSERT INTO students (name, email, class, marks, city)
VALUES ('Ali Raza', 'ali@school.pk', '10-A', 88, 'Sahiwal');

INSERT INTO students (name, email, class, marks, city)
VALUES ('Ahmad Raza', 'ali@school.pk', '10-B', 70, 'Okara');
```

What it does: `name` and `email` must always be present; `email` must also be unique. Three inserts attack the rules: the first forgets the email, the second is honest, the third reuses Ali's email for a different boy.

What you will see:

```
ERROR 1364 (HY000): Field 'email' doesn't have a default value
        -- insert 1 refused: email is NOT NULL and was not given

Query OK, 1 row affected
        -- insert 2 lands: Ali is in, with his email

ERROR 1062 (23000): Duplicate entry 'ali@school.pk' for key 'students.email'
        -- insert 3 refused: the email is already taken
```

Two refusals, one success — and the table now *guarantees* that every student has an email and no two share one. That guarantee holds for every future insert, from any app, forever.

Example 2 — DEFAULT + CHECK: marks that behave

```
CREATE TABLE results (
  id INT PRIMARY KEY AUTO_INCREMENT,
  student_id INT NOT NULL,
  marks INT DEFAULT 0 CHECK (marks BETWEEN 0 AND 100)
);

INSERT INTO results (student_id) VALUES (1);
INSERT INTO results (student_id, marks) VALUES (2, 150);
SELECT * FROM results;
```

What it does: The first insert gives no marks at all — DEFAULT supplies 0 and the row lands. The second claims 150 marks out of 100 — CHECK tests `150 BETWEEN 0 AND 100`, finds it false, and refuses the row.

What you will see:

```
Query OK, 1 row affected          -- student 1, marks defaulted to 0

ERROR 3819 (HY000): Check constraint 'results_chk_1' is violated
                                  -- student 2's 150 never lands

+----+------------+-------+
| id | student_id | marks |
+----+------------+-------+
|  1 |          1 |     0 |
+----+------------+-------+
1 row in set
```

The SELECT is the proof: only the honest row exists. An impossible result (150/100) is not "in the table with a warning" — it simply does not exist. Reports built on this table can trust it.

Example 3 — ON DELETE: what happens to the children

```
CREATE TABLE courses (
  id INT PRIMARY KEY AUTO_INCREMENT,
  title VARCHAR(50) NOT NULL
);

CREATE TABLE enrollments (
  id INT PRIMARY KEY AUTO_INCREMENT,
  student_id INT NOT NULL,
  course_id INT,
  FOREIGN KEY (course_id) REFERENCES courses(id) ON DELETE SET NULL
);

INSERT INTO courses (title) VALUES ('Computer Science'), ('Mathematics');
INSERT INTO enrollments (student_id, course_id) VALUES (1, 1), (2, 1), (3, 2);

DELETE FROM courses WHERE id = 1;

SELECT * FROM enrollments;
```

What it does: Two courses, three enrollments — two students in Computer Science (course 1), one in Mathematics (course 2). The foreign key declares `ON DELETE SET NULL`, so deleting course 1 keeps its enrollment rows but blanks their course link. (With the default RESTRICT, the DELETE would have been refused outright; with CASCADE, both enrollment rows would have vanished with the course.)

What you will see:

```
+----+------------+-----------+
| id | student_id | course_id |
+----+------------+-----------+
|  1 |          1 |      NULL |
|  2 |          2 |      NULL |
|  3 |          3 |         2 |
+----+------------+-----------+
3 rows in set
```

The students' enrollment history survives (that they *were* enrolled is a fact), the dead link is honestly NULL, and the Mathematics row is untouched. Three behaviours, three very different tables afterwards — which is why the choice is made at CREATE time, in writing.

Quick recap

- **NOT NULL** refuses emptiness; **UNIQUE** refuses repeats; **DEFAULT** fills silence with a sensible value; **CHECK** refuses any row failing its condition.

- Constraints are enforced by the database on every write — app bugs cannot sneak bad data past them.
- **ON DELETE** decides the children's fate ahead of time: RESTRICT (refuse), CASCADE (delete children too), SET NULL (keep children, blank the link).

Common mistakes

- **Checking in the app only:** the form validates, the API validates — then someone imports a CSV by hand and the table fills with 150-mark rows. If a rule matters, it belongs in the table as a constraint *as well*.
- **NOT NULL everywhere "to be safe":** a column that is sometimes honestly unknown (a student's optional second phone) forced NOT NULL gets filled with junk like `'N/A'` and `'-'`, which is worse than an honest NULL. Constrain what must never be empty; allow what may be.

  (Roman Urdu: Har column par NOT NULL lagana hifazat nahi — jhooti qeematein bhar jayengi.)
- **CASCADE chosen casually:** ON DELETE CASCADE on the wrong link deletes history — fees, orders, results — the day one parent row is removed. CASCADE is for children that are meaningless alone (order items without their order), never for records with their own life.
- **Trusting CHECK on old MySQL:** before 8.0.16, MySQL parsed CHECK and silently ignored it. If a rule is life-or-death, verify your server version — or test the refusal yourself, like Example 2 does.

Practice

1. Rewrite the `students` CREATE TABLE with every constraint this lesson justifies: which columns get NOT NULL, which get UNIQUE, which get DEFAULT or CHECK? Defend each choice in one line.
2. A `products` table needs: name always present, price never negative, stock defaulting to 0. Write the CREATE TABLE, then write the two INSERTs that prove the CHECK and the DEFAULT both work.
3. For each pair, choose RESTRICT, CASCADE or SET NULL and say why: (a) orders + order_items, (b) students + their library fines, (c) blog posts + their comments.

Practice quiz

1. What does NOT NULL do?

- It gives the column a default value
- It refuses emptiness: an INSERT that leaves the column out or passes NULL is refused
- It makes the column unique
- It hides the column

Answer: It refuses emptiness: an INSERT that leaves the column out or passes NULL is refused

2. What does CHECK (marks BETWEEN 0 AND 100) guarantee?

- That marks always equal 100
- That the database tests the condition on every INSERT and UPDATE, and a row that fails never lands
- That the app validates the marks

- That marks can be any number, checked later

Answer: That the database tests the condition on every INSERT and UPDATE, and a row that fails never lands

3. A parent student with fee children is deleted. What does ON DELETE CASCADE do?

- It refuses the delete, the safe default
- It deletes the children with the parent, powerful and dangerous
- It keeps the children and blanks their link
- It does nothing

Answer: It deletes the children with the parent, powerful and dangerous

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/sql-constraints-deep · More free tools and notes at https://scopetreasure.com