GUL'S NOTEBOOK · ENGINEERING · OPEN ACCESS

Designing the database for a school ERP

Muhammad Gulzaib
https://orcid.org/0009-0000-8709-6989
SynergyFlow Labs · Islamabad, Pakistan

Published	2026-05-02 (updated 2026-11-24)
URL	https://gulzaib.vercel.app/blog/designing-the-database-for-a-school-erp
Licence	Creative Commons Attribution 4.0 International (CC BY 4.0), https://creativecommons.org/licenses/by/4.0/

ABSTRACT

A tested MySQL schema for a school ERP: dated enrolments for promotions, composite tenant keys and an append-only fee ledger in paisa. With Laravel migration.

Keywords: mysql, laravel, database, saas, edtech

> **TL;DR**
>
> A school ERP database rests on three decisions. A student's class is a dated **enrolment** row for one academic year, never a column on `students`. Fees are an **append-only ledger** in integer paisa, so arrears are a balance you print, not a charge you add. And every foreign key carries `tenant_id`, so one school's rows can't point at another's. The full MySQL schema below runs in CI against MySQL 8.0 and 8.4, with a Laravel migration for the trickiest table.

I first sketched this schema in May 2026, a few months into building Oxygen LMS. In November I wrote it up, tested every rule on MySQL 8.0 and 8.4, and added the sources.

Most ER diagrams for a "school management system" put `class_id` on the students table. That works until the school promotes everyone at the end of the year. You update the column, and last year's class lists, attendance registers and result sheets now point at the wrong class.

This is the schema I'd start a school ERP with on Laravel and MySQL, for private schools in Pakistan. It goes with two other posts: the multi-tenant Laravel setup covers the application side of keeping schools apart, and what schools pay for explains why the fee tables get the most care. It's a cut-down, tested design for this post, not a dump of a production database.

What tables does a school ERP need?

Fourteen tables cover the daily work. Each one has a clear answer to "what is one row?":

Group	Table	One row is
Tenancy	`tenants`	A school
Calendar	`academic_years`	A session, e.g. 2026-27
	`grade_levels`	"Class 5", permanent
	`sections`	"5-B in 2026-27", one year only
People	`families`	The household that gets one voucher
	`students`	A child's identity, nothing about class
	`enrollments`	A student in a section, from a date to a date
Daily work	`attendance_marks`	One enrolment, one day
Fees	`fee_heads`	Tuition, Transport Route 3, Exam fee
	`fee_rates`	One head for one class in one year
	`concessions`	A sibling, staff or scholarship discount
	`vouchers`	One family, one month
	`payments`	Money received, with the bank reference
	`ledger_entries`	Every rupee owed or paid
Reports	`family_balances`, `section_rosters`	Views, not tables

If you can't say what one row of a table means in a few words, the table is usually two tables.

[Figure: Entity relationship diagram of the school ERP schema. Calendar: sections point to academic_years and grade_levels. People: students point to families, enrollments point to students and sections, attendance_marks point to enrollments. Fees: fee_rates and concessions point to fee_heads, concessions also point to students, payments point to vouchers, ledger_entries point to payments and fee_heads, and vouchers, payments and ledger_entries all point to families.]

The core tables and their main foreign keys. Enrolments and the ledger, outlined in blue, carry most of the design. Select it to open the full-size diagram.

Why shouldn't students have a class_id column?

Because a class is something a student is in for a while, and the database needs to remember the whiles.

Split it into three parts. `grade_levels` holds "Class 5", which exists every year. `sections` holds "5-B in 2026-27", created fresh each session. `enrollments` links a student to a section with dates:

school-erp-schema.sql (excerpt)

```
CREATE TABLE enrollments (
  id              BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  tenant_id       BIGINT UNSIGNED NOT NULL,
  student_id      BIGINT UNSIGNED NOT NULL,
  section_id      BIGINT UNSIGNED NOT NULL,
  starts_on       DATE NOT NULL,
  ends_on         DATE NULL,                          -- NULL = current
  end_reason      VARCHAR(12) NULL,
  open_student_id BIGINT UNSIGNED AS (IF(ends_on IS NULL, student_id, NULL)) VIRTUAL,
  UNIQUE KEY enrollments_tenant_id (tenant_id, id),
  KEY enrollments_section (tenant_id, section_id, ends_on),
  CONSTRAINT enrollments_reason CHECK ((ends_on IS NULL) = (end_reason IS NULL)),
  UNIQUE KEY enrollments_one_open (tenant_id, open_student_id)
  -- plus the date check, the reason values and two composite foreign keys
);
```

A student should be in exactly one section at a time. PostgreSQL can enforce that with an exclusion constraint on a date range [7]. MySQL has nothing like it, so the schema uses a generated column instead.

`open_student_id` holds the student's id while the row is open and `NULL` once it closes. A unique index allows any number of `NULL` values [2], so a student can have ten closed rows but only one open one. Try to enrol Ali in a second section without closing the first and MySQL refuses with error 1062.

The `CHECK` on line 12 says a row has an end reason exactly when it has an end date. MySQL 8.0.16 and later enforce `CHECK` constraints, though they can't call `NOW()` or look at other tables [1]. Keep them to rules about the row itself.

Attendance then hangs off the enrolment, not the student. A mark for 14 October belongs to "Ali in 5-B, 2026-27", so it stays with the right register even after he moves on.

How does year-end promotion work?

Promotion becomes one update and one insert. The update only fills in end dates, so no row ever loses the class it recorded.

Don't hard-code when the year starts. Punjab's 2026 session began on 1 April [3], and a school running another calendar will have different dates, so `academic_years` stores `starts_on` and `ends_on` as data.

1. **Create next year's sections**

   Add the `2027-28` academic year and its sections ("6-B", and so on). Last year's sections stay as they are.

2. **Close this year's rows with a reason**

   Students who repeat a class get `end_reason = 'retained'` and a new row in the same grade next year. Everyone else in the section is promoted:

   ```
   START TRANSACTION;

   UPDATE enrollments
   SET ends_on = '2027-03-31', end_reason = 'promoted'
   WHERE tenant_id = 1 AND section_id = 2 AND ends_on IS NULL;

   INSERT INTO enrollments (tenant_id, student_id, section_id, starts_on)
   SELECT tenant_id, student_id, 5, '2027-04-01'
   FROM enrollments
   WHERE tenant_id = 1 AND section_id = 2
     AND ends_on = '2027-03-31' AND end_reason = 'promoted';

   COMMIT;
   ```

3. **Check the history**

   Ali now has three rows: 4-A (promoted), 5-B (promoted) and 6-B (open). The `section_rosters` view still prints the 2026-27 list for 5-B with him on it.

The order matters. Closing first frees the "one open row" slot, so the insert can't collide with it.

How do you stop one school seeing another school's data?

The multi-tenant post puts a global scope on every model, which filters queries by `tenant_id`. That protects reads. It doesn't stop a bad write, such as a job that saves School A's enrolment with School B's section id. A plain foreign key on `section_id` is happy with that, because the section exists.

Composite keys close the gap. Each parent table gets `UNIQUE (tenant_id, id)`, and each child points at both columns:

```
CONSTRAINT enrollments_section
  FOREIGN KEY (tenant_id, section_id) REFERENCES sections (tenant_id, id)
```

Now a row can only reference a parent from its own school, and the cross-school write fails with error 1452. The cost is one extra unique index per parent table, which the `(tenant_id, …)` indexes from the earlier post mostly cover anyway.

Laravel's schema builder handles all of this except the `CHECK` constraints:

`database/migrations/…_create_enrollments_table.php (excerpt)`

```
$table->unsignedBigInteger('open_student_id')
    ->nullable()
    ->virtualAs('IF(ends_on IS NULL, student_id, NULL)');
$table->unique(['tenant_id', 'open_student_id'], 'enrollments_one_open');

$table->unique(['tenant_id', 'id']);
$table->foreign(['tenant_id', 'student_id'])->references(['tenant_id', 'id'])-
>on('students');
$table->foreign(['tenant_id', 'section_id'])->references(['tenant_id', 'id'])-
>on('sections');

// after Schema::create, since the builder has no CHECK constraints:
DB::statement('ALTER TABLE enrollments ADD CONSTRAINT enrollments_reason
    CHECK ((ends_on IS NULL) = (end_reason IS NULL))');
```

How should a school ERP store fees?

As integers, in a ledger that only grows.

Paisa, in `BIGINT`. Fees are whole rupees, but a 5% late fine on Rs 13,550 is Rs 677.50, and bank files report paisa. `FLOAT` can't hold most decimal fractions exactly, so totals drift. Integer paisa never drifts, and `DIV 100` turns it back into rupees for printing. Discounts are stored in basis points (1000 = 10%) for the same reason.

Rates per year. `fee_rates` has one row per fee head, class and academic year, enforced by a unique key. In Islamabad, PEIRA's rules forbid raising fees during the session [4], so the app never edits a running year's rate. Next year's rates are new rows, and old vouchers can always be explained.

A ledger, not a balance column. Every charge, discount, fine and payment is one row in `ledger_entries`. Charges are positive and payments negative, and a `CHECK` enforces the sign for each type. Two triggers reject `UPDATE` and `DELETE`, so a wrong charge is fixed with a `reversal` row that points at it. A unique key on `reverses_id` stops anyone reversing the same row twice.

Here is the family from the fee voucher post, Ali and Sara, through two months:

Month	Ledger rows	Rs

April	Tuition × 2, sibling discount, transport	11,550
April	1Bill payment, TXN-88231	−10,350
	Balance	**1,200**
May	Tuition × 2, sibling discount, transport, exam fee	12,350
	May voucher: arrears 1,200 + charges 12,350	**13,550**

That matches the voucher in the earlier post to the rupee, and the test suite checks it.

> **Arrears are a balance, not a charge**
>
> The May voucher stores the 1,200 in `balance_bf_paisa`, a snapshot of what the family owed when it was printed. It is never inserted into the ledger. If you add arrears as a charge, the same 1,200 gets billed again every month it stays unpaid, and the defaulter list grows with money nobody owes.

Two more unique keys matter. `vouchers` allows one live voucher per family per month, using the same generated-column trick as enrolments, so a cancelled voucher can be replaced but never duplicated. `payments` has a unique key on the method and bank reference, so importing the same 1Bill file twice can't record a payment twice.

What should a student record hold in Pakistan?

Identity, and nothing that changes each year. Two Pakistan-specific fields matter.

The B-Form number. NADRA issues the Child Registration Certificate, the B-Form, to children under 18 [5]. Store its 13 digits without dashes, with a `CHECK` on the format and a unique key per school. It's nullable, because some children arrive without one.

Names as NADRA has them. Since September 2026, FBISE pulls a candidate's name, father's name and date of birth for board registration straight from NADRA's records [6]. A name typed differently at admission turns into a correction years later, at registration time. So `students` has `father_name` and `date_of_birth` as required columns, and admission staff copy them exactly from the B-Form.

What's left out?

Enough to fill later posts. The same patterns carry over:

- **Exams and marks** hang off `enrollments`, like attendance, so a result always belongs to the class the student was in.
- **Guardians.** `families` holds one guardian and phone. A real school needs several

contacts per family, with a flag for who gets the fee SMS.

- **Staff, HR and payroll** are their own tables with the same composite keys.
- **An audit log** for who changed what, outside the ledger, which already records itself.

How was this tested?

On GitHub Actions, every time the SQL changes. The SQL examples workflow loads the schema into MySQL 8.0 and 8.4, seeds two schools and Ali's family, and runs 29 checks. A second job copies the migration into a fresh Laravel 13 app and runs it. Some of the rules it confirms:

Attempt	MySQL says
Enrol a student in a second section	Error 1062, duplicate on `enrollments_one_open`
Point at another school's section	Error 1452, foreign key fails
Close an enrolment without a reason	Error 3819, check constraint violated
Edit or delete a ledger row	Error 1644, "append-only: add a reversal instead"
Import the same 1Bill transaction twice	Error 1062
Enter a B-Form number with a letter in it	Error 3819

Download the file, run it in an empty database, and try to break it.

> **Key takeaways**
>
> - Put class membership in a dated `enrollments` table. Promotion closes one row and opens the next.
> - In MySQL, a generated column plus a unique index gives you "one current row per student".
> - Add `tenant_id` to every foreign key, so the database refuses cross-school references even when the app forgets.
> - Store money as integer paisa in an append-only ledger, and print arrears from the balance instead of charging them again.
> - Store the B-Form number and the names exactly as NADRA has them.

*[Figure: image]***Multi-tenant Laravel SaaS: the architecture I actually use**: Single database or one per tenant? A production-tested multi-tenant setup for Laravel SaaS: global scopes, tenant resolution, queues and the mistakes to avoid. *[Figure: image]***Fee vouchers, not dashboards: what schools pay for**: What Pakistani private schools asked for when I pitched school software: fee vouchers that follow their rules, a defaulter list they trust and Urdu SMS.

Questions readers ask

What tables does a school management system database need?

At the core: tenants (schools), academic years, grade levels, sections per year, families, students, enrolments, attendance, fee heads, fee rates, concessions, vouchers, payments and a fee ledger. Exams, staff and timetables hang off the same tables later. The key one is enrolments, which links a student to a section for a dated period.

How do you store a student's class history in a database?

Keep the class off the students table. Each academic year gets its own section rows, and an enrolments table links a student to a section with a start date, an end date and a reason (promoted, retained, transferred). Promotion closes one row and opens another, so every past class list stays correct.

How should a school ERP store fees and money?

Store amounts as integers in the smallest unit (paisa for rupees), never FLOAT. Record every charge, discount and payment as its own row in an append-only ledger, fix mistakes with a reversing row, and treat arrears as the ledger balance printed on the next voucher rather than as a new charge.

References

1. Oracle. MySQL 8.4 Reference Manual: CHECK Constraints, 2026.
2. Oracle. MySQL 8.4 Reference Manual: CREATE INDEX Statement, 2026.
3. The Express Tribune. Punjab revises school admissions policy, updates syllabus and textbooks for 2026, 2025.
4. Dawn. Peira directs private schools not to increase fees by more than 5pc, 2021.
5. NADRA. Juvenile Card / Child Registration Certificate (CRC), 2026.
6. NADRA. NADRA and FBISE Sign Agreement to Integrate Student Registration and Educational Certificates System, 2026.
7. PlanSmith. School management system database design, 2026.

CITE AS

Gulzaib, M. (2026, May 2). Designing the database for a school ERP. Gul's Notebook. https://gulzaib.vercel.app/blog/designing-the-database-for-a-school-erp

You may share and adapt this text for any purpose, including commercially, as long as you credit the author and link to the original. Licence: https://creativecommons.org/licenses/by/4.0/