What is an ORM?

An **ORM** (Object–Relational Mapper) is a tool that lets your application interact with a relational database using code, instead of writing raw SQL for every operation.

In a typical web app, your backend is written in a programming language like JavaScript or TypeScript, while your database speaks SQL. An ORM sits between the two and translates database rows into language-native objects, and vice versa. This allows you to work with database data using familiar data structures rather than strings of SQL.

For example, instead of writing a SQL query to fetch a user by ID, an ORM might let you write code that looks like “find this user,” and handle generating the SQL behind the scenes. The result is returned as a regular object your application can work with directly.

One major benefit of an ORM is type safety and structure. Many modern ORMs know the shape of your tables ahead of time and can catch mistakes—like missing fields or invalid relationships—before your code ever runs. This is especially helpful in larger projects where the database schema and application logic are tightly connected.

ORMs also help manage relationships between tables. Rather than manually joining tables and stitching results together, an ORM can follow relationships defined in your schema and return connected data in a more ergonomic way. This makes common operations simpler and less error-prone.

ORMs are not magic. They don’t remove the need to understand how databases work. Poorly written ORM queries can still be slow or inefficient, and there are times when writing raw SQL is clearer or more performant. The best developers use ORMs as a tool, not a crutch.

In our scouting application, we use an ORM (Drizzle) to keep database access consistent, readable, and safer to change over time. Learning how an ORM maps tables to code will make it much easier to build features without constantly context-switching between SQL and application logic.

Drizzle ORM

We use Drizzle as our ORM in the scouting site. Drizzle is a modern ORM designed to feel as close to SQL as possible while still giving you the benefits of working in TypeScript. Instead of hiding the database behind abstractions, Drizzle aims to make database access explicit, predictable, and type-safe.

At its core, Drizzle lets you define your database schema directly in TypeScript. Tables, columns, and relationships are written as code, and Drizzle uses that information to generate strongly typed queries. This means your editor can catch mistakes—like invalid column names or incorrect joins—before the code ever runs.

Unlike some ORMs that try to fully replace SQL, Drizzle embraces it SQL. You still think in terms of tables, joins, and queries, but you write them using a structured API instead of raw strings. If you already understand how relational databases work, Drizzle tends to feel very natural.

Another important feature of Drizzle is that your schema is the source of truth. The same definitions are used for:

- Database migrations (i.e., schema changes)
- Query typing
- Application-level validation

This reduces the risk of your database and backend code drifting out of sync over time.

Drizzle also avoids doing too much “magic.” Queries are predictable, easy to inspect, and closely resemble the SQL they generate. This makes debugging easier and helps reinforce good database habits rather than obscuring them. In the context of an FRC scouting application, Drizzle works well because the data is highly structured and relational. Teams, matches, events, and scouting entries all map cleanly to tables, and Drizzle makes it easier to work with those relationships safely as the application grows.

Drizzle is not a replacement for understanding SQL—it assumes you already care about how your data is modeled. Used correctly, it acts as a bridge between clean database design and maintainable backend code.

Revision #2
Created 27 January 2026 23:39:06 by Drew Beamer
Updated 28 January 2026 00:41:06 by Drew Beamer