What is a Database?

If you've done some of the basic web challenges (React Tic-Tac-Toe, To-do List, etc.) you may have wondered how to persist data. While things like session and local storage exist in the browser, they're limited to a single device and can be cleared at any time. This isn't super useful for things like the scouting site, where we need to aggregate data from multiple devices.

Databases are how applications store data permanently. It lives on a server, and allows for data to persist across logins, devices, and users. When working on a backend, a database is where things like user information, settings (and in our case scouting data) actually lives. In a typical flow, a frontend application sends a request to an API, which can perform operations on the database to return data to the user.

Kinds of Databases

There are a few kinds of databases, which are store data in different ways. The right choice is application-specific, and depends on things like how structured your data is and how you want to query it.

Relational Databases

Relational databases are the most common kind of database, and often the best starting point for applications. Relational databases store data in *tables.* Tables represent a kind of data, where each *column* represents an attribute about the data and each *row* represents a data point.

Relational databases are very similar to how a spreadsheet works. Each table can be thought of as a tab in the spreadsheet, whereas rows and columns are very similar to how a spreadsheet functions.

However, relational databases power lies in their ability to relate tables to each other, making it easy to model (and store) real world relationships between data. Imagine we have a database containing information on cars. We might have a "cars" table and a "manufacturers" table. The cars could be linked to a manufacturer through a manufacturer id that exists on both the cars table and the manufacturers table (this is an example of a one-to-many relationship, more on that later...). When querying the data, I might want to get all cars manufactured by “Ford”. To do this, we use *SQL* (Structured Query Language), the language used to read and write data in relational databases.

SQL lets you:

- Select data from tables
- Filter rows on conditions
- Join related tables together
- More functions that will be discussed in later pages...

To get all the cars made by Ford, we can write a query like:

```
SELECT
  car.id,
  car.model,
  car.year,
FROM car
JOIN manufacturer
  ON car.manufacturer_id = manufacturer.id
WHERE manufacturer.name = 'Ford';
```

This query combines the car and manufacturer tables using the shared id, filters the results to only manufacturers named "Ford," and returns the car's id, the model, and the year. The key idea is that the car does **not** store the manufacturer's name directly, as the manufacturer's name belongs to the manufacturer. Instead, it stores a reference (manufacturer_id). While a manufacturer may change their name, the manufacturer_id attribute is stable, unique to our database, and should never change. If the manufacturer name ever changes, we only need to update it in the manufacturer table. Every related car automatically reflects that change because it points to the same manufacturer record.

Note: our scouting site uses Postgres, which is a kind of relational database. As such, there is a bit more detail here than will be included for other kinds of databases.

NoSQL Databases

NoSQL databases are a category of databases that store data in ways other than tables with rows and columns. Instead of enforcing a rigid structure, most NoSQL databases store data as documents, key–value pairs, or other flexible formats. In web development, the most common type you'll encounter is a **document database** (e.g., MongoDB).

In a document database, data is stored as individual documents that look very similar to JSON objects. Each document typically represents a single entity and contains all of the data associated with it. Because the data closely resembles JavaScript objects,

NoSQL databases often feel intuitive to developers working with Node or frontend frameworks like React.

Relationships are handled differently in NoSQL databases as well. Since most NoSQL systems do not support joins, related data is often stored together or referenced manually. For example, instead of having a separate manufacturers table, a car document might store the manufacturer’s name directly. This can simplify reads, but it introduces duplication. If that manufacturer’s name changes, every document containing it must be updated to stay consistent.

NoSQL databases are often a good fit when data is unstructured, changes frequently, or does not have many relationships. They are commonly used for things like logs, analytics events, session data, or caching. In these cases, flexibility and speed are often more important than strict structure.

However, this flexibility comes with tradeoffs. Because relationships and consistency are handled mostly in application code, complex data models can become harder to reason about as an application grows. For systems that need strong guarantees and well-defined relationships—such as teams, matches, events, and scouting entries—this can quickly add complexity.

For most FRC applications, relational databases remain the better default choice. NoSQL databases are still an important tool to understand, but they are best used intentionally, once you have a clear reason to trade structure for flexibility.

Revision #6
Created 27 January 2026 23:38:52 by Drew Beamer
Updated 28 January 2026 00:17:04 by Drew Beamer