Relational Schema Design Notes

Incentivize high fidelity—data that reflects their real-world meaning as closely as possible—and disincentivize errors—data that is inaccurate, contradictory, or needlessly missing

• It should be no surprise to state that we want our database to reflect our information accurately

• But to paraphrase the so-called “Rule of Repair,” we also want to make attempts to insert bad data fail and fail noisily and as soon as possible:

  ◦ Inconsistent data—pieces of information in the same database that contradict each other

  ◦ Redundant data—pieces of information that always change together (meaning that, if you don’t, you will then get inconsistent data)

  ◦ Dangling data—information that references other information that doesn’t exist
Relationships

• A key aspect of the “high fidelity” side is representing how different entities relate to each other

• An objective characteristic of such relationships is cardinality—i.e., how many of one side of the relationship can connect to the other side?

• For each type of cardinality, a fairly standard “recipe” exists for modeling each in a relational database—these recipes aren’t absolute, but they represent a great starting point

• One-to-one: An instance of one side can connect to only one instance of the other side, and vice versa

  e.g., A driver’s license relating to its owner

• One-to-many: An instance of one side can connect to more than one instance of the other side, but not the other way around (a “many-to-one” relationship is essentially the same thing, but the other way around)

  e.g., A parent relating to their biological children

• Many-to-many: An instance from either side can connect to more than one instance of the other side

  e.g., Classes relating to students that are taking them
**One-to-One Recipe**

Put both entities in the same table -or- use the same primary key for each table.

**One-to-Many Recipe**

Make the “many”-side entity have a non-primary-key reference to the primary key of the “one”-side entity.
Many-to-Many Recipe

Define a **third** table with references to the primary keys of the related entities (the references can be the primary key to avoid relationship duplications)

---

<table>
<thead>
<tr>
<th>entity A</th>
<th>entity B</th>
<th>A_B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

“‘It’s Complicated’”

- Consider if either side of a relationship can be empty—this determines whether a reference can be null
- Relationships between the same entity types (e.g., friends; items assembled from other items [a.k.a. “bill of materials”]) follow similar recipes—but watch for directionality (i.e., \( A \ifdir B \) is the same \( B \ifdir A \)?)
- Recipes shown here are for relationships involving two groups of entities—relationships that involve more than two have different approaches
Minimizing Repetition

- Although not an absolute, relational databases seek to minimize the repetition of data—e.g., having a person’s name appear in more than one table or in multiple rows of the same table.

- Repetitions not only waste space, but make updates error-prone: if you update a repeated value, you need to update it everywhere in order to stay consistent.

- If a value (except for a key) appears more than once, you may have a relationship—so model it that way.

- The process of eliminating repetitions is typically called normalization because relational database theory has the concept of normal forms—formalized descriptions of how tables are defined.

- Intuitively speaking, the theoretical third normal form denotes a relational database schema that has eliminated redundant attributes among its tables.

- The single major factor against normalization is performance: normalization implies joins, and joins can be costly—thus, applications with specific performance needs may denormalize their data as long as they accept the consequence of doing extra work to keep the data self-consistent.
What’s in a Name?

• This one is a bit of a soapbox—but if it isn’t pointed out now, how can we possibly seek change in the future: Many databases still model name fields in ways that hold overly-narrow assumptions

• Here’s something to read and bookmark, to share with anyone you encounter whose idea of names remains limited and oversimplified:


• Note how many of the things stated in that article are directly contradictory—highlighting the inherent challenge of how to approach them in a database

• The key here isn’t necessarily to find a “one true way” for modeling names, but simply to spread awareness that storing names isn’t as straightforward as some people [still] think

• Here’s a slightly newer take, this time more specific to data modeling and how they affect a user interface:

  https://softwareforgood.com/why-you-should-stop-asking-for-first-and-last-names-on-forms/

• Spread the word! Call this out whenever you spot it
Structured Attributes

• The aforementioned practices were largely developed when relational database columns/attributes were always single-valued or scalar—i.e., they weren’t allowed to have intrinsic structure, like the primitive data types of a programming language.

• Things have gotten even more complicated with the emergence of structured attributes such as JSON or similar data types—all of a sudden, a column/attribute can have parts, can be a list, etc.

• This raises questions like: Can an array column replace a simple one-to-many relationship? Can a column hold an object to represent a one-to-one relationship? How does one avoid redundancy of values within a structured attribute? 😞😢😢

• On the one hand, one can ignore this very feature—relational databases were without this for decades after all, and they did just fine (mostly)

• But that begs the question of why this feature emerged in the first place—something compels it, right?

• For these (and all future changes in database technology), keep first principles in mind: maximize fidelity to your data’s meaning and minimize errors.