16. Database Design

Github Repository | Vimeo Video Recording

Topics to cover

- [X] 1. Tables
- [X] 2. Data types
- [X] 3. Naming Conventions
- [X] 4. Primary Keys and Foreign Keys
- [X] 5. Relationships
- [X] 6. Design Concepts
- [X] 7. ERDs

Primary Key

- A way of uniquely identifying a particular record within a table
- Must be unique (within the table) and can never be null
- The usual data type is auto-incrementing integer (`INTEGER` or `BIGINT`)
- A Primary Key stored in another table is known as a `Foreign Key`
- The Primary Key and Foreign Key **MUST** be the same data type

Naming Conventions

- Table and field names are written in `snake_case`
- Table names are always pluralized
- The primary key for each table will simply be called `id`
- A foreign key is made up of the singular of the primary keys table and the suffix `_id` (eg. `user_id` is the foreign key for the `id` field in the `users` table)

Data Types

- Each field in a table **must** have a data type defined for it
- The data type tells the database how much room to set aside to store the value *and* allows the database to perform type validation on data before insertion (to protect the data integrity of the table)
- Choosing the perfect data type is less of a concern nowadays because memory is now comparably cheap

Design Concepts/Heuristics

- Make fields required based on the records state upon initial creation (remember that additional data can be added to a record after it has been created)
- Intelligent default values can be set for fields (such as the current timestamp for a `created_on` field)
- Don't use calculated fields (a field that can be derived from one or more other fields, such as `full_name` is a combination of `first_name` and `last_name`)
- Pull repeated values out to their own table and make reference to them with a foreign key
- Try not to delete anything (use a boolean flag instead to mark a record as active or inactive)
- Consider using a `type` field instead of using two (or more) tables to store very similar data (eg. create an `orders` table with an `order_type` field instead of a `purchase_orders` and a `sales_orders` table)luralize table names: `authors`
  - Always call your primary keys: `id`
  - Foreign keys are made from the table name singularized plus `_id` :
    - A foreign key referencing the `authors` table: `author_id`
    - A foreign key referencing the `books` table: `book_id`
    - A foreign key referencing the `authors_books` table: `authors_book_id`

Relationship Types

- **One-to-One**: Pretty Rare
- **One-to-Many**: Very Common
- **Many-to-Many**: Very Common - can be thought of as two One-to-Many

Normalization

- About data duplication
- We don't need to know the academic underpinnings
- Any time we have the same data stored in more than one location
  - Writing that data becomes complicated
- The advantage to breaking data normalization is Query Performance

Design Concepts/Heuristics

- `NOT NULL` should be prefered, and you should utilize intelligent defaults

- Try to get the database to work as hard as you can
- Avoid calculated fields: `first_name`, `last_name` and `full_name`. (Depends)
- Any repeated values are an indication that Normalization can happen here
- Don't use `DELETE` statements, instead add a `deleted_at` column. (Depends)
- Sometimes use `type` columns to differentiate things rather than multiple tables with very similar data

Entity Relationship Diagram (ERD)

- A visual depiction of the database tables and how they are related to each other
- Extremely useful for reasoning about how the database should be structured
- Can be created using pen and paper, a whiteboard, or using an online application

Useful Links

- Database Normalization
- Postgres Data Types
- Relationship Types
- draw.io (online ERD)

View on Compass | Leave Feedback