Brandon Swanson
CS 340 Final Project
7/9/15

Outline

This database is designed to track all skateboarding parts owned and used by a household or some other skateboard club/organization. Me and my wife are avid skateboards and after only a few years in the hobby we have amassed a decent collection of different boards, and spare pieces and have disassembled and re-assembled them in many different configurations. This is the purpose of this database, to track the various existing and possible combinations of skateboard parts and their riders.

The database interface provides the user with a list of all currently owned inventory, which consists of Decks (the board part of a skateboard), Trucks (the wheel mount) in sets of 2, and Wheels in sets of 4. The 'Build a Skateboard' tab will allow the user to assemble one of each of these parts into a new skateboard that can be viewed in the Skateboard tab where skateboards can be linked with riders, or disassembled to make those parts available.

Look for the following Icons to perform said actions


INSERT


DELETE


UPDATE


INSERT/DELETE skateboards

Outline in Words

The entities in this database are as follows:

- Riders
- Skateboards
- Skateboard parts (Deck, Truck, Wheel)
- Brands

Riders and Brands are the only entities that are self contained in their own table, The others are made up of relationships. There are 3 inventory tables, one for Deck, Truck, and Wheels. Each row in these tables represents a real world item, an existing part, and is composed of unique information (color) and a foreign key to a Part Type which is manufactured by a Brand (foreign key in the type table row to the Brand table). For example in our household we have 2 boards made by Penny brand, they have differently sized decks, differently sized trucks, but the same type of wheels. So there is one entry for the Penny Brand in the Brands table, 2 entries in deck types, 2 entries in truck types and 1 entries in wheel type. The 6 real world parts are each represented as a row in their respective table, with the 2 entries in the wheel inventory table referring to the same type and all 5 types referring to the same brand. The type tables contain information about the part like Diameter and Durometer (a measure of hardness) for wheel types, information that is consistent across all instances of this type of part.

A Skateboard is composed of a total participation relationship with exactly 1 Deck, 1 Truck, and 1 Wheel from the inventory tables. With a name, and picture as relationship attributes. The interface allows for the assembling and disassembling of skateboards, in other words inserting or deleting these relationship rows.

Finally there is a many to many relationship between riders and boards. In our household we posses around 5 boards and each of the 2 of use make use of some subset of them. This many to many relationship is contained within the table sk8_riders_skateboards as pairs of foreign keys.

Constraints

- Foreign key constraints
    - all inventory items must reference a part type
    - all part types must reference a brand
    - all skateboards must reference a Deck, Truck, and Wheel from inventory.
    - Rider-Board relationship must reference riders table and boards table
        - relationship is deleted on one of those keys being deleted
        - ON DELETE CASCADE

A type is deleted when it no longer references any part in inventory. This is done with using a delete query and WHERE NOT IN a subquery of all part type foreign keys in the respective inventory table. This is because the interface allows for adding a duplicate of an established type or creating a new type, It would be opaque to the user if they had removed all items of a type but that type still existed.

Each inventory part can only be used on one skateboard at a time, This is enforced with a UNIQUE constraint on each foreign key in the skateboard table. When attempting to create a new skateboard the PHP checks for the unique constraint error code and then runs a query that selects the name and id of each board that contains one of the parts the user had selected. It

then presents them with the option to disassemble these boards or cancel. If they chose to disassemble those boards then the appropriate rows are removed, the parts are freed up for use, and their original query to insert into the skateboards table runs again.

Brand names must be unique, This is a real world constraint as well due to trademark laws. There should be no reason for the user to enter the same brand twice, and then be linking parts from the same brand to two different entries. This is enforced by the database using a UNIQUE constraint.

Adding/Removing/Updating Rows

All tables can have new rows inserted via the interface. Most rows (excluding brands) can have rows deleted. Updating is restricted to changing the img_url for riders or skateboards.

From the inventory screen you can add or remove instances of a part type, create a new of a new part type, or add a brand. From the 'Build a Skateboard' screen these parts can be combined to form a new row in the skateboards relationship table.The many to many relationship of skateboards can be managed from the skateboards and riders tabs.

Updating is restricted because for the items in the parts inventory an update on most of its fields is not very logical because it is composed of information from 3 different tables. Would it make sense to the user if their edits on one item propagated to the other items of that type in the inventory.

I believe though that there are sufficiently complicated queries at use in this database application, making use of Subqueries, NOT IN, and complex joins. Of note are

- LINE:256 query to select all information pertaining to a skateboard including each of its parts, their specifications, and brand names and images.
- LINE: 287 query to find all not yet in existence relationships between riders/skateboards for making a drop down menu. This query makes use of subqueries and a LEFT OUTER JOIN.