University of Victoria
CSC 370: Database Systems

Assignment 4

Due on: Friday, October 18 at 11:59pm (35 marks)

Introduction

The purpose of this assignment is to:

- Practice importing data from a CSV file into a database and then transforming that data to be compatible with a specific relation.
- Practice implementing relations with desired constraints via `CREATE TABLE`.
- Further practice writing queries with PostgreSQL.
- Implicitly introduce you to the `NULLIF` and `COALESCE` functions (look these up if you don't know what they do).

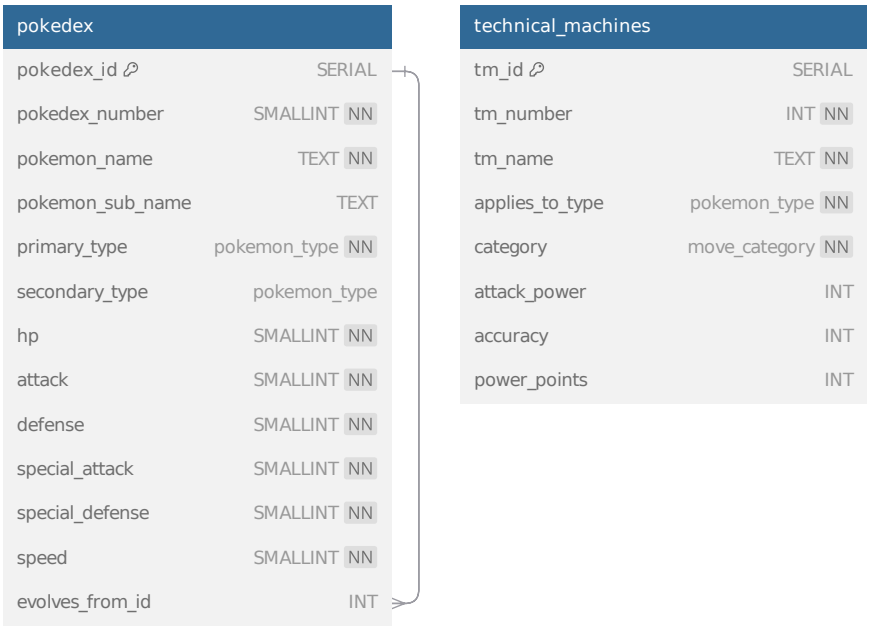

Figure 1: The starting database.

Importing CSV Data

During our class's lectures, we implemented a table to store information about different species of Pokemon and then populated that table by importing data from a CSV file. In this part of the assignment, you will do the same, but this time the table will be storing information about a specific type of item within the Pokemon universe: technical machines.

1. (8 points) A common task when working with database is that of data transformation. In many cases, this can be done entirely in SQL, as you will in this question. Before attempting this question, use the supplied `.sql` file to prepare your database for the CSV import by executing the provided `CREATE TYPE` and `CREATE TABLE` statements.

   With the `technical_machines` table empty, your goal is to fill it with the data found within the supplied CSV file by using steps similar to what you saw in class. Those are:

   - Create a temporary table[1] to hold the data from the CSV file.
   - Import the data via the `COPY`[2] command.
   - Write an `INSERT INTO SELECT`[3] statement to transform and then insert the data from the temporary table into the destination table.
   - Drop the table.

   Use the provided starting file and fill in the blanks. Submit a script that uses a transaction and performs the steps outlined above to populate the `technical_machines` table.

[3]During the lecture I did not have time to explain how true temporary tables work in PostgreSQL, so I used a transaction and dropped the 'temporary' table before committing the transaction. Instead, you can use a temporary table that gets dropped once a transaction is committed or rolled back (as is in the provided file).

[3]If you encounter a permission error when trying to perform a `COPY`, ensure that the CSV file is in a location where other users have permission to view it (such as `C:\Users\Public` on Windows).

[3]You might find it easier to write the `SELECT` statement separately before trying to incorporate it into the `INSERT INTO` statement. The `SELECT` statement must select the data from the temporary table and transforms it into the format required by the destination table.

Extending the Database

The Pokémon Company is interested in adding an online auction system[4] to the next Pokémon game to buy and sell technical machines (single-use items that can be used to teach a Pokémon a specific move). An auction system would allow users to list items (technical machines) for sale. Potential buyers would then be able to place bids on those items. New bids must be higher than prior bids on a particular listing. Listed items have a 'reserve' price, which is the minimum bid that will be accepted — if the highest bid is less than the reserve price, then the item does not get sold. After bidding closes, if the highest bid is above the reserve price, then the bidder wins the item and must pay the price they bid.

2. (13 points) Creating the Tables.

   Based on the requirements, three tables need to be created: A 'users' table, a 'listings' table, and a 'bids' table:

   `users(user_id, username, password, email)`

   `listings(listing_id, tm_id, selling_user_id, listed_on, time_to_list, starting_bid_price, reserve_price, current_bid_price, current_bid_id)`

   `bids(bid_id, listing_id, user_id, bid_time, bid_price)`

   Write and submit a script that uses a transaction to create these tables and add them to our Pokemon database.

   Hints: The tables will use surrogate primary keys, indicated via underlined column names. Your tables will need to be compatible with the provided data (the `INSERT` statements), so ensure that your selected data types for each column are appropriate. For each column, consider all possible constraints, referring to the PostgreSQL documentation to ensure you don't forget about any. Consider the best approach for ensuring referential integrity for each foreign key (restrict, set null, or cascade); for example, upon deleting a user, would deleting information about the bids be appropriate or not?

For the remaining questions, you can run the provided `insert_listings.sql` script to populate your database with data. This data will help you to see the results of your queries.

[4] If you don't know what an auction is, see `https://en.wikipedia.org/wiki/Auction`.

3. (10 points) Creating a View.

   The `listings` table contains two columns (`current_bid_price` and `current_bid_id`) that are not strictly necessary to include in the data model, as they can be determined by finding the most recent `bid` associated with a particular `listing_id`. An alternative approach that would achieve the same result would be to make use of a view.

   Without making use the `current_bid_price` or `current_bid_id` columns from the `listings` table, create a `VIEW` that returns the following columns:

   - listing_id
   - tm_id
   - selling_user_id
   - listed_on
   - time_to_list
   - starting_bid_price
   - reserve_price
   - is_listing_over
   - is_item_sold
   - current_bid_price
   - current_bid_id

   The last four columns of the view need to be determined by the query. `is_listing_over` must calculate whether a listing is over and `is_item_sold` must calculate whether a listing is sold (based on the rules outlined at the start of this part of the assignment). **Submit your SQL code to create the view.**

   Note

   With the present design (using stored values in the tables instead of a view) there is a need to ensure that the `current_bid_price` and `current_bid_id` columns within the `listings` table do not become invalid or out of date. This can be done by writing a trigger that sets these values when a new `bid` is `INSERT`ed into the database. You will learn about triggers later in the course.

   For now, we will write an `UPDATE` query to ensure that this data is accurate.

   ---

4. (4 points) Write an `UPDATE` query that sets the `current_bid_id` and `current_bid_price` for each row in `listings` based on the data in the `bids` table. If the listing has no bids `current_bid_id` and `current_bid_price` should both be be `NULL`. If the listing has one or more bids, then set `current_bid_id` to the `bid_id` with the highest `bid_price` and set the `bid_price`.

   The provided data had the correct values set already, so your query should not change anything unless you insert new data. **Submit your `UPDATE` query.**

5. Optional practice question (not for marks).

   Let's practice writing `SELECT` statements by writing a query that lists all unique technical machines for auction that can be applied to Charizard (that is, the technical machine matches at least one of Charizard's two types). The auction listings must not have already ended. Your query should be ordered by the end date of the listing (with auctions ending sooner at the top) and return the following columns:

   - Pokemon type associated with the technical machine.
   - The closing date and current bid price of the listing.
   - The number and name of the technical machine.