Phrase2

Yuewei Wang, Yongyi Xu

Design Descisions

Regarding the feedback of phase1, we are encouraged to find additional datasets, and generate more investigative questions.

Since the evaluation terms of the world happiness score are consistent (GDP, life expectancy, generosity, social support, freedom, and corruption), we found that other available datasets have exactly the same columns as our old dataset. The only variant column would be year. In this case, with only year-difference to generate interesting and investigative questions is difficult. Additionally, based on year-difference to build schema into different relations would be redundant. It can be expressed into one big table rather than within a schema, thus the good design of schema is also hard to achieve.

Therefore, we decided to replace a new dataset of best-selling books. This dataset contains multi-dimensional relations in the original data, which is more suitable to generate schema with multiple relations than the old dataset. It is also suitable to generate investigative questions since more characters are involved. Due to the richness of column categories, we could generate questions in different aspects.

Here is the new schema design for phase 2:

- **Domain**: The summary of Amazon top 50 bestselling books from 2009 to 2019.
- **Link**: https://www.kaggle.com/sootersaalu/amazon-top-50-bestselling-books-2009-2019
- **Questions**:
    1. How are the booktype and popularity related. Are fiction books always more popular than non-fiction books? How are the readers' feedback and popularity related?
    2. For the books that are bestsellers for multiple years, do they always have higher ratings than the books who only be the bestseller for one year? Are these books always having non-decreasing reviews or reader ratings?
    3. For the authors who have multiple published books and are bestsellers, do they always focus on writing single genre types?
- **Schema**:**Relations:**
    - Book (<u>BID</u>**,** bookname, price)
      An Amazon's Top 50 bestselling book

Attribute	Description	Type
BID	the ID of the book	int
bookname	the name of the book	TEXT
price	the price of the book	int

This is the ‘center relation’ as we are discussing the bestselling book on Amazon. Bookname and price are original attributes in the dataset. We also created a new attribute BID to keep track of every book as BID is unique.

- BookType (BID, bookname, genre)
  Foreign (BID) References Book(BID)
  The type of an Amazon's Top 50 bestselling book.

Attribute	Description	Type
BID	the ID of the book	int
bookname	the name of the book	TEXT
genre	the type of the book	Genre

Since we want to make discussion respect of two types of books, we created BookType relation.
We created a new type called Genre and it will be used to describe the genre of an Amazon's Top 50 bestselling book. We decided to make an attribute genre of type Genre because the type of the book can only be either ‘Fiction’ or ‘Non-fiction’. We want to make it more specific than simply assigning the type TEXT.

- Author (AID, authorname)
  The author of an Amazon's Top 50 bestselling book.

Attribute	Description	Type
AID	the ID of the author	int
authorname	the name of the author	TEXT

- Written (AID, BID)
  Indicates which author wrote which book.
  FOREIGN KEY (BID) REFERENCES Book(BID),
  FOREIGN KEY (AID) REFERENCES Author(AID)

Attribute	Description	Type
AID	the ID of the author	int
BID	the ID of the book	int

We create this relation to avoid redundancy: if we put BID directly inside Author relation, it will cause redundancy as one author may write different books so there may be multiple (AID, authorname) tuples with the same value. Hence, we decide to make this relation to describe the relation between authors and books.

- Feedback (<u>BID, year</u>, userrating, review)
  Foreign (BID) References Book(BID)
  The review and user rating for the book at the year.

Attribute	Description	Type
BID	the ID of the book	int
year	the year of the book being the bestseller	int
userrating	the overall Amazon user rating for the book at the year	FLOAT
review	the number of reviews written for the book at the year	int

We built this relation to avoid redundancy: There are books that become bestsellers in multiple years. If we put the column ‘year’, ’review’ and ‘userrating’ directly in the Book relation, it would cause redundancy since the same book name may be included multiple times, duplicate (BID, bookname) tuples would occur in Book relation. Also for the purpose of separating different category information (review and rating), we create this relation to describe the relation between books and each bestselling year along with reviews and ratings.

Cleaning Process:

- <u>Preview for NULL values and redundant columns</u>:

  For a cleaning schema, removing the irrelevant columns and NULL data are necessary before importing the data into SQL. We observed the columns of ‘bestsellers_data.csv’ has, which are all needed for creating the schema, thus we kept all columns.
  We also observed all rows to detect any NULL value. All rows have content in every attribute, thus we kept all rows in the csv file.

- Fix the original data csv:

  To import the values of each row, we deleted the first row (column names) of the 'bestsellers_data.csv' so that we could use '\copy' loading all data values at one time. Here we used vim to edit the file in terminal end:

  ```
  vi bestsellers_data.csv
  ```

  and use the `dd` command to delete the row with all column names. Thus the content csv file would start with values directly as shown below, and ready to be copied:

```
10-Day Green Smoothie Cleanse,JJ Smith,4.7,17350,8,2016,Non Fiction
11/22/63: A Novel,Stephen King,4.6,2052,22,2011,Fiction
12 Rules for Life: An Antidote to Chaos,Jordan B. Peterson,4.7,18979,15,2018,Non Fiction
```

- Create the raw table regarding the data in 'bestsellers_data.csv' and fix the column names:

  Now, we are on the stage to import the above csv file into psql. The preliminary stage of importing the data from 'bestsellers_data.csv' is to create a table holding all information. Initially, building a table for these raw data is essential.

  For the attribute names of the table, we maintained most of the column names as the original column name in the file. But replaced the original name of 'User Rating' into 'userrating' in SQL to avoid the invalid space in the middle. Thus, the command in psql are:

```
CREATE TABLE data (
      name text,
      author text,
      userrating double precision,
      review integer,
      price integer,
      year integer,
      genre text
);
```

  After creating the table, the dataset could be populated by using the '\copy' command.

- Check any violating rows for relation 'Book' construction:

  Consider the case that two different authors may write two books with the exact same name, or a book has two authors at the same time. If the case occurs, it is necessary to remove the ambiguous rows. The below query was performed to check whether exists such row that we need to clean:

```
csc343h-xuyongy1=> select * from book;
csc343h-xuyongy1=> create view t1 as
csc343h-xuyongy1-> select name, genre from data;
CREATE VIEW

csc343h-xuyongy1=> select * from
csc343h-xuyongy1-> t1 c1, t1 c2
csc343h-xuyongy1-> where c1.name = c2.name and c1.genre != c2.genre;
 name | genre | name | genre
------+-------+------+-------
(0 rows)
```

`t1` is the view that selects all book names from the table `data`. Self-product of `t1`, then select the rows that have exact same book names but written by different authors. As the figure shown, the query result was 0 rows. Therefore, the ambiguous row does not exist, there is no row needed to be clean regarding this case.

- Check any violating rows for relation 'Feedback' construction:

  Considering the case that two different reviews or user ratings may be included in the dataset. If the case occurs, it is necessary to remove the ambiguous rows. The below query was performed to check whether exists such row that we need to clean:

```
csc343h-xuyongy1=> create view f as
csc343h-xuyongy1-> select name, year, userrating, reviews
csc343h-xuyongy1-> from data;
CREATE VIEW
csc343h-xuyongy1=> select * from f f1, f f2
csc343h-xuyongy1-> where f1.name = f2.name and f1.year = f2.year and
csc343h-xuyongy1-> (f1.userrating != f2.userrating or f1.reviews != f2.reviews);
 name | year | userrating | reviews | name | year | userrating | reviews
------+------+------------+---------+------+------+------------+---------
(0 rows)
```

`f` is the view that selects all book names, record year, user rating at the year, and reviews at the year from the table `data`. Self-product of `f`, then select the rows that have the exact same book name and record year but different reviews or user ratings. As the figure shown, the query result was 0 rows. Therefore, the ambiguous row does not exist, there is no row needed to be clean regarding this case.