Mount Royal University

COMP 2521 002

Data Modeling and Query Languages

Assignment Three

Author
Zyrel Arandia
Author
Andrew Bown
Author
Bryce Carson

December 3, 2023

1 Introduction

The DDL for the assignment SQL is summarized in this top-level chunk. The tables are created in the order the referenced chunks are named. The parental chunk has the same name as the table in the database, but child chunks should have descriptive names (if any child chunks exist).

The SQL server used is MariaDB. The ⟨*MariaDB version*⟩ used is contained in the chunk just referenced.

Occasionally, throughout this document, chunk names may be repeated in SQL comments to ensure that the tangled SQL is readable without the literate document.

⟨*MariaDB version*⟩≡

```
-- 10.3.39-MariaDB; as of 2023-12-02T23:11PM on macomydb.mtroyal.ca
SELECT VERSION();
```

Overall, the SQL script to create the databse is outlined.

⟨*A3.sql*⟩≡
  ⟨*DDL: tables*⟩

The file is not too complex.

2 DDL

⟨*DDL: tables*⟩≡
  ⟨*USER*⟩
  ⟨*BOOK*⟩
  ⟨*AUTHOR*⟩
  ⟨*BOOKAUTHOR*⟩
  ⟨*READBOOK*⟩

3 Entities and Relations

The business rules of the database are not complex, and can be stated in few words. The names of the sections that follow state the rules. Further explanation is given in the body text of a section if necessary.

3.1 USER is an entity

MariaDB VARCHAR data types support different character sets; the UTF-8 character set, which is used for international email addresses, has a maximum length of 21,844 characters in a VARCHAR attribute. Given that, email addresses, nick names, and profiles should each be constrained to this limit.

A profile, or biography, is usually longer than a person's name. Email addresses are often names, and with international email addresses, these may be rather long. The length is indeterminate, so rather than truncating names or biographies, the maximum character length is permited for each string.

Any email address that does not contain an `@` character between a "local part" and a "domain part" is invalid.

From the perspective of a software engineer, almost any string is a valid international email address. ASCII-only email addresses are also quite complex! A great video of a presentation on the topic is available here on YouTube (the presentation was given by Dylan Beattie at NDC { Oslo }). More technical information is contained in this archived document from the Universal Acceptance Steering Group of ICANN.

> In some cases it may be useful to assign both an EAI and a legacy address for a mailbox. (See Downgrading, below.) In some cases there may be a straightforward transliteration, such as борис@domain to boris@domain or 李伟 @domain to liwei@domain. In other cases, there may be no natural way to transliterate, and the two names may have no obvious connection.

Client-side and server-side validation of email addresses should be used; prevent malicious actors. Assuming that client-side validation of email addresses has been implemented properly, and that server-side validation has also occurred, what remains is to insert the email address into the database and check that it has an @ sign in the string, at *minimum* (or *maximum*, depending on personal engineering perspective).

⟨*USER*⟩≡
```
CREATE TABLE USER (
    Email VARCHAR(21,844) CHARACTER SET utf8 PRIMARY KEY
        CHECK Email LIKE "%@%",
    DateAdded DATETIME NOT NULL,
    NickName VARCHAR(21,844) CHARACTER SET utf8 UNIQUE,
    Profile VARCHAR(21,844) CHARACTER SET utf8
);
```

3.2 BOOK is an entity

⟨*BOOK*⟩≡
```
CREATE TABLE BOOK (
  BookID INT PRIMARY KEY AUTO_INCREMENT,
  Title VARCHAR(255) NOT NULL,
  Year INT CHECK (Year >= ⟨date of the Kish Tablet⟩ AND
                  Year <= ⟨the current year⟩),
  NumRaters INT DEFAULT 0,
  Rating DECIMAL(2,1) DEFAULT 0.0
      CHECK (Rating >= 0.0 AND Rating <= 5.0)
);
```

Users obviously cannot read books that have not been written yet, so ⟨*the current year*⟩ is used to limit what can be inserted into the database.

⟨*the current year*⟩≡
```
YEAR(CURDATE())
```

The Kish Tablet is believed to date from 3500 BCE. It is likely the oldest confirmed writing known, so any books a user claims to have read should be more recent than this.

⟨*date of the Kish Tablet*⟩≡
```
-3500
```

3.3 AUTHOR is an entity

⟨*AUTHOR*⟩≡
```
CREATE TABLE AUTHOR (
    AuthorID INT PRIMARY KEY AUTO_INCREMENT,
    Lastname VARCHAR(50) CHARACTER SET utf8,
    FirstName VARChar(50) CHARACTER SET utf8 NOT NULL,
    MiddleName VARCHAR(50) CHARACTER SET utf8
);
```

3.4 BOOK has AUTHOR

⟨*BOOKAUTHOR*⟩≡
```
CREATE TABLE BOOKAUTHOR (
    AuthorID INT FOREIGN KEY REFERENCES AUTHOR(AuthorID),
    BookID INT FOREIGN KEY REFERENCES BOOK(BookID)
);
```

3.5 USER reads BOOK

Users can read many books, so necessarily the Email address in a READBOOK should not be unique. This will lead to an excessive amount of storage usage, but this is the specification given in the database design (by our instructor).

> When a user is deleted, all the READBOOK records associated with the user must also be deleted.

Ergo, a trigger should be implemented to delete the records `WHERE READBOOK.Email = USER.Email` whenever the trigger condition is satisfied.

⟨*READBOOK*⟩≡
```
CREATE TABLE READBOOK (
    BookID INT FOREIGN KEY REFERENCES BOOK(BookID) NOT NULL,
    Email VARCHAR(...) FOREIGN KEY REFERENCES USER(Email)
        NOT NULL
        ON DELETE CASCADE,
    DateRead DATE NOT NULL,
    Rating INT(2) CHECK Rating >= 1 AND Rating <= 10 NOT NULL,
    PRIMARY KEY (BookID, Email)
) ENGINE=INNODB;
```

⟨*Update ratings in a BOOK on changes to BOOKTABLE*⟩

Ray Ferrell explains in the linked blog how to make changes to a foreign key's referent domain propagate (cascade) to the referring (child) table. This eliminates the need to programmatically delete rows in the table when the referent is updated, and instead the SQL server and supporting engines do the work for us. The only remaining task is to then trigger a call to the user-defined `CalculateRating` aggregate function to update 'BOOK.Rating' when this table is updated.

⟨*Update ratings in BOOK on changes to BOOKTABLE*⟩≡

```
CREATE TRIGGER afterUpdateRow_recalculateBookRating
    AFTER UPDATE ON READBOOK FOR EACH ROW CalculateRating();

CREATE TRIGGER afterInsertRow_recalculateBookRating
    AFTER INSERT ON READBOOK FOR EACH ROW CalculateRating();

CREATE TRIGGER afterDeleteRow_recalculateBookRating
    AFTER DELETE ON READBOOK FOR EACH ROW CalculateRating();
```