CS4416 PROJECT

Autumn 2018

Adam Aherne – 12159603 | Eoin Purtill - 17185467|
David Jacobs – 16156501 | Ronan McMullen - 0451657

Contents

C.1 Contributors

Adam Aherne – 12159603 | Eoin Purtill – 17185467

David Jacobs – 16156501 | Ronan McMullen - 0451657

Table Creation and Data Insertion: Adam, Eoin, David, Ronan

Views and Indexes: Ronan, David

Triggers, Procedures and Functions: Adam, Eoin

Report: Ronan, Adam, Eoin, David

Percentage of work done by each student: 25% / 25% / 25% / 25%

C.2 Database Description

For this project we attempted to model a typical library database. We aimed to create a functional database that was able to catalogue and access information on all books, loans and members in a library as well as facilitate the interactions between these entities.

Books are uniquely identified by their *ISBN*, members by their *membership_id* and loans by their *loan_id*. These identifiers form the core of our database.

Books are divided into three different tables within our database (*fiction, non-fiction* and *journals*), we felt that by splitting our catalogue of books, which could conceivably be quite large, into three tables it would improve the performance of the database. The table *items* serves as an reference table for the different tables of books.

Members are entered into the table *members*, this table contains membership information including name and email.

Loans consist of a member identifier (*member_id*), a book identifier (*ISBN*) and a date.

One of the main intended functions of the library system is the monitoring and organising of loans. Members will carry scannable library cards (*membership_id*), books will have a scannable section on the spine (*ISBN*). Upon taking a loan the member scans their card and the book being taken, the system passes this information along with date information to the database and a *loan* entry is created. The software will also determine when the *due_date* is for the book to be returned based on the day that it is loaned. The system will update the *loans* table accordingly as books approach their overdue date. We envision the system automatically notifying members when their current loan is approaching overdue status.

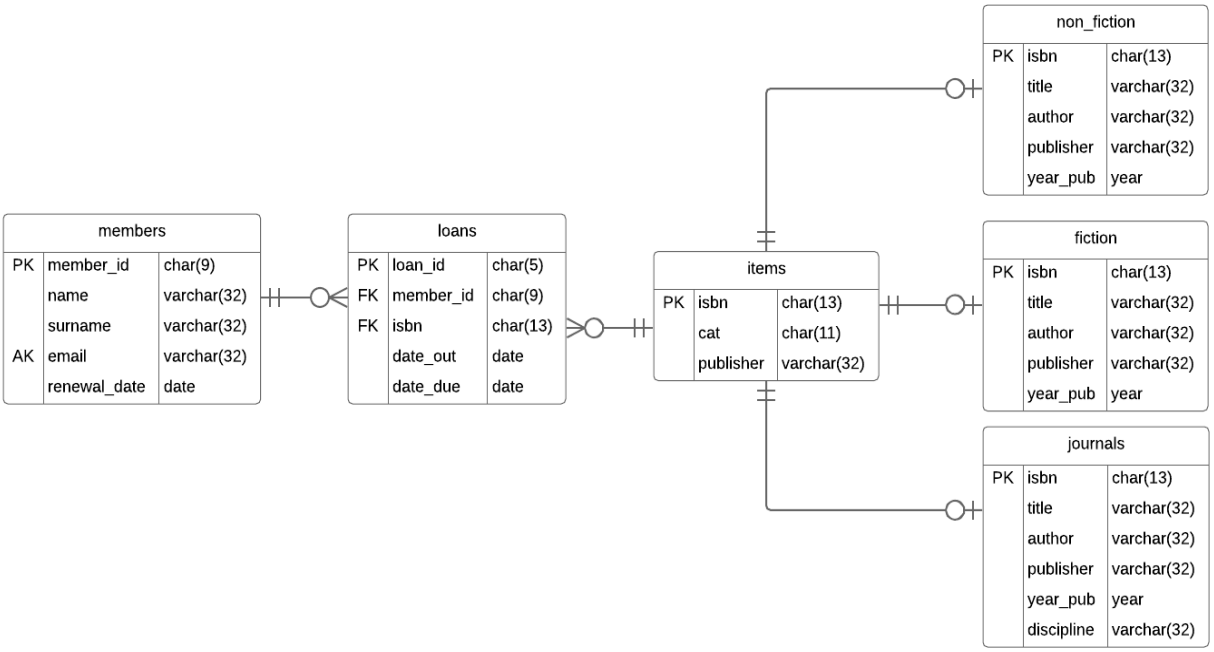
Library staff will also be able to use the software system to view different information relating to books, members and loans.

For the purposes of our project we have made a few assumptions concerning our library system:

- Members cannot take on loan two copies of the same book in one day.
- Books with different ISBN may have the same title.

C.3 Entity Relationship Diagram

The following is the ERD for our file schema.sql



C.4 Table Examples with Primary Key Attributes

Primary Key | **Foreign Key** | **Unique**

Members

member_id	name	surname	email	renewal_date
111123432	Meekus	McNally	abc@123.ie	2018-12-02
499382872	Arthur	Morgan	amorg@gmail.com	2020-02-24

Loans

loan_id	member_id	isbn	date_out	date_due
00443	666444555	093874686263	2018-11-23	2018-12-14
00445	223344098	1234567890943	2018-11-25	2018-12-16

Items

isbn	category	publisher
1238274622773	Non-fiction	Penguin
4953992016374	Fiction	Harper-Collins

Fiction

isbn	title	author	publisher	year_pub
0044492887288	The Amber Spy	Phillip Pullman	Yearling	1998
9948200019928	Wild Bill	Bill Williamson	Big House	2017

Non-Fiction

isbn	title	author	publisher	year_pub
1100434459288	Irish History	Brendan Ryan	O'Rourke	1968
1293481929428	The Wetlands	Janice Cooper	Realt Dearg	2005

Journals

isbn	title	author	publisher	year_pub	Discipline
48848376636333	Biology Review	G.J. Cheung, A. Moore	Oxford Press	1999	Biology
57847377388832	Medical Journal	X. Hernandez, L. Lewis	Cambridge Press	2000	Medicine

C.5 Functional Dependencies

The following are the functional dependencies for each relation in our schema:

Members – Functional Dependencies:

R(A,B,C,D,E) PK = A AK = D

A → BCDE D → ABCE

AB → CDE AC → BDE AD → BCE AE → BCD

DB → ACE DC → ABE DE → ABC

ABC → DE ABD → CE ABE → CD ACD → BE ACE → BD ADE → BC

DBC →AE DBE → AC DCE → AB

ABCD → E ABCE → D ABDE → C ACDE → B BCDE → A

A^+ = {ABCDE}

Loans – Functional Dependencies

R(A,B,C,D,E) PK = A FK = B, C

A → BCDE

AB → CDE AC → BDE AD → BCE AE → BCD

ABC → DE ABD → CE ABE → CD ACD → BE ACE → BD ADE → BC

ABCD → E ABCE → D ABDE → C ACDE → B

A^+={A,B,C,D,E}

Items – Functional Dependencies

R(A,B,C) PK = A FK = B, C

A → B A → C

AB → C AC → B

A^+={A,B,C}

Fiction – Functional Dependencies

R(A,B,C,D,E) PK = A

A → BCDE

AB → CDE AC → BDE AD → BCE AE → BCD

ABC → DE ABD → CE ABE → CD ACD → BE ACE → BD ADE → BC

ABCD → E ABCE → D ABDE → C ACDE → B

A^+={A,B,C,D,E}

Non-Fiction – Functional Dependencies

R(A,B,C,D,E) PK = A

A → BCDE

AB → CDE AC → BDE AD → BCE AE → BCD

ABC → DE ABD → CE ABE → CD ACD → BE ACE → BD ADE → BC

ABCD → E ABCE → D ABDE → C ACDE → B

A^+={A,B,C,D,E}

Journals – Functional Dependencies

R(A,B,C,D,E,F) PK = A

A → BCDEF

AB → CDEF AC → BDEF AD → BCEF AE → BCDF AF → BCDE

ABC → DEF ABD → CEF ABE → CDF ABF → CDE

ACD → BEF ACE → BDF ACF → BDE

ADE → BCF ADF → BCE

AEF → BCD

ABCD → EF ABCE → DF ABCF → DE

ABDE → CF ABDF → CE ABEF → CD

ACDE → BF ACDF → BE ACEF → BD ADEF → BC

ABCDE → F ABCDF → E ABCEF → D ABDEF → C ACDEF → B

A^+={A,B,C,D,E,F}

C.6 3NF Proof

Members

1NF – Yes, because each intersection of a column and row contains one and only one value.

2NF – Yes, because all non-key attributes are dependent on the primary key (*member_id*).

3NF – Yes, because there are no transitive dependencies. All functional dependencies contain the primary key *member_id* on the left-hand side. This agrees with the definition of Third Normal Form.

BCNF – Yes, because the left-hand side of every functional dependency contains a candidate key (A,D).

Loans

1NF – Yes, because each intersection of a column and row contains one and only one value.

2NF – Yes, because all non-key attributes are dependent on the primary key (*loan_id*).

3NF – Yes, because there are no transitive dependencies. As demonstrated in C.5 all the functional dependencies for this table contain the primary key *loan_id* on the left-hand side.

BCNF – Yes, because the left-hand side of every functional dependency contains a candidate key (A).

Items

1NF – Yes, because each intersection of a column and row contains one and only one value.

2NF – Yes, because all non-key attributes are dependent on the primary key (*isbn*).

3NF – Yes, because there are no transitive dependencies. The primary key *isbn* is contained in the left-hand side of all functional dependencies for this table.

BCNF – Yes, because the left-hand side of every functional dependency contains a candidate key (A).

Fiction

1NF – Yes, because each intersection of a column and row contains one and only one value.

2NF – Yes, because all non-key attributes are dependent on the primary key (*isbn*).

3NF – Yes, because there are no transitive dependencies. The primary key *isbn* is contained in the left-hand side of all functional dependencies for this table.

BCNF – Yes, because the left-hand side of every functional dependency contains a candidate key (A).

Non-Fiction

1NF – Yes, because each intersection of a column and row contains one and only one value.

2NF – Yes, because all non-key attributes are dependent on the primary key (*isbn*).

3NF – Yes, because there are no transitive dependencies. The primary key *isbn* is contained in the left-hand side of all functional dependencies for this table.

BCNF – Yes, because the left-hand side of every functional dependency contains a candidate key (A).

Journals

1NF – Yes, because each intersection of a column and row contains one and only one value.

2NF – Yes, because all non-key attributes are dependent on the primary key.

3NF – Yes, because there are no transitive dependencies. The primary key *isbn* is contained in the left-hand side of all functional dependencies for this table.

BCNF – Yes, because the left-hand side of every functional dependency contains a candidate key (A).

C.7 View Justifications

View: members_who_have_had_more_than_one_loan

- This view returns the details of every member that has ever taken more than one loan, it also returns the count of how many loans that the customer has taken.
- We envision this query being used for statistical purposes within the library system.
- We feel this view to be justifiably useful as it provides non-trivial information that could be valuable for the admin staff operating the library system.

View: overdue_loans

- This view displays the details of any loans that are overdue.
- This view was required because the name and contact information of members is not stored in the loans table. A query is needed to tie a member's personal information to a particular loan's information.
- We feel that this is a valid view as it displays the contact information (including email) of members with overdue loans. This information can be used to inform the member in question that they have an overdue book.

View: fiction_currently_on_loan

- This view displays the non-fiction books that are currently on loan.
- This query is important so that a record can be kept of whether a particular book is on the shelf or out on loan.

View: non_fiction_currently_on_loan

- This view displays the non-fiction books that are currently on loan.
- This query is important so that a record can be kept of whether a particular book is on the shelf or out on loan.

View: journals_currently_on_loan

- This view displays the journals that are currently on loan.
- This query is important so that a record can be kept of whether a particular book is on the shelf or out on loan.

C.8 Speed Analysis

For the purposes of analysing our query performance we created a test data set of hundreds of entries for each of the tables in our database. Because we did not want to spend the time required to type each entry by hand, we used Microsoft Excel to populate each column of the tables with trivial data as the set was for performance analysis alone. We ensured that any Primary and Foreign Key constraints were adhered to and exported a .txt file of each table. We opened these files in Notepad++ and with a little help from the *Replace All* command we were able to make the text SQL compatible.

(The following is applicable to the three views *fiction_currently_on_loan, non_fiction_currently_on_loan* and *journals_currently_on_loan*.)

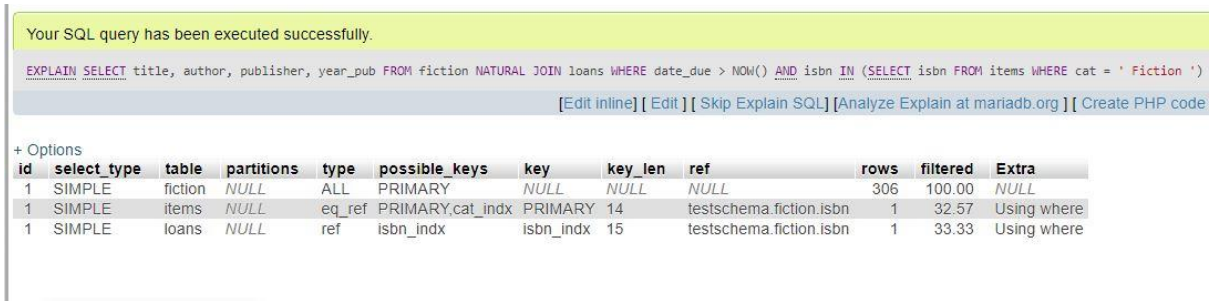
We decided to structure the query *fiction_currently_on_loan* as such:

```
CREATE VIEW fiction_currently_on_loan AS
        SELECT title, author, publisher, year_pub
        FROM fiction NATURAL JOIN loans
        WHERE date_due > NOW() AND isbn IN
                                        (SELECT isbn
                                         FROM items
                                         WHERE cat = 'Fiction');
```

Upon analysing the performance of this query with the EXPLAIN keyword we discovered that it was not operating very efficiently. Although we were only interested in selecting the ISBN of books in the fiction category of the *items* table, our sub-query was visiting 878 of the 918 rows in the table. The screenshot below demonstrates this.

After adding an index (*items_cat_indx*) to the *category* column in the *items* table and adding an index (*loans_cat_indx*) to the *isbn* column in *loans,* we received a notable performance improvement.

Running the same query as above after the addition of our indexes yielded the following analysis:



Your SQL query has been executed successfully.

```
EXPLAIN SELECT title, author, publisher, year_pub FROM fiction NATURAL JOIN loans WHERE date_due > NOW() AND isbn IN (SELECT isbn FROM items WHERE cat = ' Fiction ')
```

[Edit inline] [Edit] [Skip Explain SQL] [Analyze Explain at mariadb.org] [Create PHP code

+ Options

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	fiction	*NULL*	ALL	PRIMARY	*NULL*	*NULL*	*NULL*	306	100.00	*NULL*
1	SIMPLE	items	*NULL*	eq_ref	PRIMARY,cat_indx	PRIMARY	14	testschema.fiction.isbn	1	32.57	Using where
1	SIMPLE	loans	*NULL*	ref	isbn_indx	isbn_indx	15	testschema.fiction.isbn	1	33.33	Using where

As you can see the number of rows visited has dropped from 878 to 306.

C.9 Trigger Justification

Our database implements the following procedures, functions and triggers…

Procedure: *item_table_data_insert*

This procedure is called from several of our triggers. It is passed three arguments and inserts them into a new row in the *items* table. We justify its use as it keeps the *items* table up to date with any insertions into the *fiction, non_fiction* and *journals* tables.

Function: *check_date_due_on_loan*

This function ensures that any loans that are entered have a due date for return of at least 21 days from the day they are loaned. It is called by the *loan_insert* trigger.

Trigger: *loan_insert*

This trigger is called before each row is inserted into the loans table. It checks to see if the due date for a book to be returned is at least 21 days from the day of it is loaned. If this check fails, the trigger calls the function *check_due_date_on_loan*.

Triggers: *fiction_insert, non_fiction_insert, journals_insert*

These triggers are called after every row insert into either *fiction, non_fiction* or *journals.* Each time the trigger executes it calls the *item_table_data_insert* procedure. These triggers ensure that the *items* table is always updated accordingly upon relevant inserts to other tables in the schema.

Triggers: *fiction_delete, non_fiction_delete, journals_delete*

These triggers are called after a row is deleted from either *fiction, non_fiction* or *journals.* They delete the corresponding data from the *items* table to keep our database well ordered.