Data Models and Query Language

CSE 560

Prof. Dr. Sreyasee DAS BHATTACHARJEE

Group 80 Report

UB LMS - Library Management System

Team Members:

Kedar Amar Nayak (kedarama)

Annu Priya (apriya3)

Prashant Kumar (pkumar32)

Section 1: Problem Statement:

Our use case is to build a library management system. We are building a library database that can manage the library efficiently and eliminate the manual effort of managing the inventory on physical copies at times prone to human errors.

Our task is to collect and store data of books, customers visiting the library, adding new books to the inventory, rewarding loyal customers and managing all these using SQL. This will solve the existing problem of handling and issuing books, collecting late fine from defaulting readers, and efficiently managing the library database of reducing human effort by automating the search and management process.

We will be defining our schema and describing all the tables that we have used to build the database.

Why do we need a database instead of an excel file?

There are several advantages of using database to store and retrieve data as compared to excel files. Firstly, a database can store huge volume of data up to 1 GB per field and 32 TB per relation whereas an excel crashes beyond 1M rows. Secondly, using a database allows concurrent access among multiple users and, we can enable row-level or column-level security which allows only the super users to make changes to sensitive information. This enables data privacy and security where all the users do not have access to the entire database.

Who will be our target user and who will use our database?

Our target user will be a mixture of the staff working at the library, customers namely the residents of the area where the library is located and the shops which are used to procure the books from.

Real-life scenario for our use case:

An example would the UB Library which comprises of 5 libraries. Currently, it does not allow the cross functionality of searching or issuing books across all the locations. One must physically go to the Law Library to search for genre specific books and similarly for the music library. If one stays in South campus and wishes to search for a book in the Abbott Library, they must go there and do a holistic search. Instead, one can search a specific book from any location and go the right place. A hassle-free process will save time of both the student and the staff.

Section 2: Database Design:

Our database consists of 9 tables. The description of attributes, data type, purpose and description are listed below.

Table	Attributes	PK	FK	Data Type	Purpose	Description	Null Values Allowed
author	author_id	Y	N	int	Uniquely identifies an author of a book	Unique id of an author	N
	name	N	N	varchar(255)	Name of the author	Name of the author	N
category	category_id	Y	Y	int	Assigns an id to a category	Assigns an id to a book depending on the category/genre	N
	category_name	N	N	varchar(255)	Assigns a category name to a book	Assigns a category to a book depending on the category/genre	N
book	book_id	Y	N	int	Uniquely identifies a book	Assigns a unique id to a book	N
	book_title	N	N	varchar(255)	Identifies the book through its name	Displays the name of the book	N
	book_type_id	N	Y	int	Displays the id of the book which is the foreign key referring to the book_type_id of the table "book_type"	Displys the id of the the type of the book	N
	price	N	N	float	Displays the price of the book	Displays the price of the book	N
book_type	book_type_id	Y	Y	int	Uniquely identifies each book type	Unique id of a book type	N
	book_type	N	N	varchar(255)	Assigns a type to the book- for children and for adults [tentative]	Type of book available	N
book_copy	unique_book_id	Y	N	int	Uniquely identifies each copy of a particular book	Assigns a unique ID to a particular copy of a book	N
	published_year	N	N	date	Displays the year when the copy of the book was published	Displays the year when the copy of the book was published	N
	book_id	N	Y	int	Displays the id of the book of which the unique is a copy	This is a foreign key whyich is referred from the book_id of the "book" table	N
	published_by	N	N	varchar(255)	Displays the name of the publication house	Assigns the name of the publication house	N
	available	N	N	varchar(10)	Displays if the copy is available or not	Can have either of the two values "available" and "unavailable", if book is checked out by a customer, the field must display "unavailable". If not, then displays as "available"	N
book_category_relation	book_id	Y (combined with category_id)	Y	int	Used to create an association between category table and book table. book_id and category_id together act as the primary key of this table.	Primary key of book table, used as a foreign key in the table	N
	category_id	Y (combined with book_id)	Y	int	Used to create an association between category table and book table. book_id and category_id together act as the primary key of this table.	Primary key of category table, used as a foreign key in the table	N
book_author_relation	book_id	Y (combined with author_id)	Y	int	Used to create an association between author table and book table. book_id and author_id together act as the primary key of this table.	Primary key of book table, used as a foreign key in the table	N
	author_id	Y (combined with book_id)	Y	int	Used to create an association between author table and book table. book_id and author_id together act as the primary key of this table.	Primary key of author table, used as a foreign key in the table	N
Customer	customer_id_card_number	Y	Y	serial	This field uniquely identifies each customer of the library	Assigns a unique ID to the customer	N
	customer_first_name	N	N	varchar(255)	Displays the first name of the customer	Displays the first name of the customer	N
	customer_last_name	N	N	varchar(255)	Displayes the last name of the customer	Displayes the last name of the customer	N
	customer_age	N	N	int	Displays the age of the customer	Displays the age of the customer	N
	customer_address	N	N	varchar(255)	Displays the address of the customer	Displays the address of the customer	N
	customer_email	N	N	varchar(255)	Displays the email ID of the customer. The email ID must be unique but it can be null as some customers may not have an email ID	Displays the email ID of the customer	Y
	customer_status	N	N	varchar(10)	Displays if the customer is still a member of the library or not, "active" if the customer is a member, else "inactive"	Displays the status of the customer	N
checkout	checkout_id	Y	N	serial	Uniquely identifies the checkout tansaction of a particular book		N
	issue_date	N	N	date	Date on which a book is issued	Date on which a customer issues a book from the library	N
	due_date	N	N	date	Expected return date of a book	Expected date by which a book should be returned to the library	N
	actual_return_date	N	N	date	Actual return date of a book	Actual date on which an issued book is returned in the library	Y
	fine	N	N	Float	Amount paid for late return	Amount paid by a customer for returning a book after the expected date	Y
	unique_book_id	N	Y	int	Uniquely identifies a book. This is a foreign key which refers to the unique_book_id of the "book_copy" table.	Uniquely identifies a book in the library	N
	customer_id_card_number	N	Y	int	Unique number assigned to a customer in the library. This field is a foreign key which refers to the customer_id_number of the "customer" table	Unique id associated to a customer in the library	N

Figure 1: description of attributes, data type, purpose and description

ER Diagram:

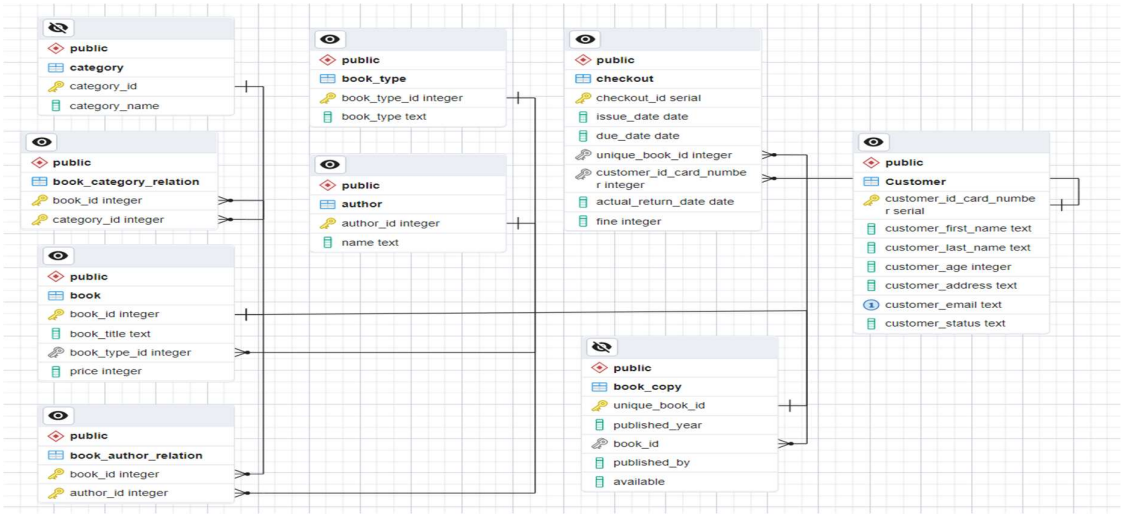

Figure 2:ER Diagram 1

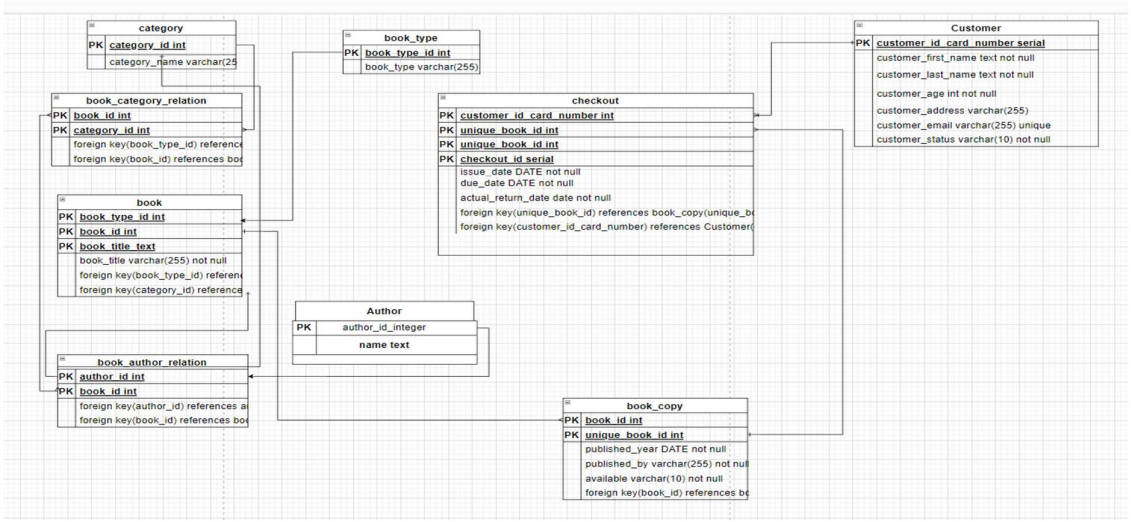

Figure 3: ER Diagram 2

Data Creation and Record Insertion:

We have used python script to generate custom data by using the Faker library of Python. Lists and sets were used to store the generated data. These were later used to generate .csv files through python script. Below is the list of .csv files generated.

- author.csv
- Book.csv
- book_author_relation.csv
- book_category_relation.csv
- book_copy.csv
- book_type.csv
- Category.csv

- Customer.csv
- checkout.csv

After creating the .csv files, the data were entered into the PostgresQL database using python script.The .pynb file containing the python scripts can be found below.

DMQL Project.ipynb

Documentation of all 9 tables from the library database

Author relation:

	author_id [PK] integer	name character varying (255)
1	974856	John Sanchez
2	458761	Kenneth Carlson
3	647178	Melissa Vega
4	131081	Brian Gates
5	489486	Robert Smith
6	73746	Craig Sampson
7	626715	Ryan Potts
8	845854	Stephanie Armstrong
9	716831	Robin Ramirez
10	823327	Allison Becker
11	608291	Donald Mcdonald
12	350247	Jill Powers
13	616494	Jenna Adams
14	778291	Thomas Rosales
15	235576	Denise White
16	430137	Thomas Mejia
17	645179	David Stewart
18	649278	Eric Davis
19	686145	Ms. Kimberly Grant
20	483397	Helen Baldwin
21	639046	Roger Bryan

Total rows: 1000 of 1000 Query complete 00:00:00.333

Figure 4: Author Relation

Category relation:

	category_id [PK] integer	category_name character varying (255)
1	1	Fiction
2	2	Non-Fiction
3	3	Biography
4	4	Autobiography
5	5	History
6	6	Science
7	7	Philosophy
8	8	Religion
9	9	Politics
10	10	Business
11	11	Cooking
12	12	Travel
13	13	Art
14	14	Music
15	15	Sports
16	16	Technology
17	17	Self-Help
18	18	Romance
19	19	Mystery
20	20	Horror
21	21	Fantasy

Total rows: 70 of 70 Query complete 00:00:00.215

Figure 5: Category relation

Customer relation:

	customer_id_card_number [PK] integer	customer_first_name text	customer_last_name text	customer_age integer	customer_address character varying (255)	customer_email character varying (255)	customer_status character varying (10)
1	1	Scott	Cooper	70	1047 Allen Rapid Suite 747 New Austinland, ID 38266	woodskristin@example.org	Active
2	2	Jeremy	Cruz	41	4606 Gonzalez Well Jamesstad, PR 24258	diana26@example.net	Inactive
3	3	Sean	Anthony	79	7524 Thomas Isle Apt. 428 Aaronborough, ID 87985	nathanlarsen@example.org	Inactive
4	4	Michael	Charles	78	1780 Sanchez Skyway Brittanyton, SD 30324	stephenadams@example.org	Inactive
5	5	Jesus	Ballard	42	80850 Kayla Ways Apt. 688 Greenstad, MI 37573	cgregory@example.org	Inactive
6	6	Christine	Martin	39	01665 Erika Light Moorefurt, CO 17492	andrewanderson@example.com	Active
7	7	Meagan	Rhodes	41	1035 Jessica Neck Suite 628 Brianberg, NE 85465	wrightjulie@example.com	Inactive
8	8	Natasha	Taylor	38	09532 Joseph Station Apt. 389 West Veronica, NV 67990	joshua59@example.net	Active
9	9	Karen	Rogers	34	3192 Munoz Lane Suite 972 West Monicafurt, DE 73681	kgutierrez@example.org	Active
10	10	Darren	Fields	40	0343 Garrett Brook Apt. 992 East Davidport, TN 81534	kentadams@example.net	Active
11	11	John	Andrews	35	76745 Cindy Ville Lake Theresa, VA 07518	glenndavid@example.com	Active
12	12	Joe	Tate	83	Unit 9013 Box 4153 DPO AP 81876	gilbertkathleen@example.org	Inactive
13	13	Laura	Anderson	45	PSC 2376, Box 0771 APO AP 75075	umccormick@example.net	Active
14	14	Timothy	Stevenson	26	3098 Garcia Lakes Nelsontown, DE 73894	ramoschad@example.com	Active
15	15	Kenneth	Schwartz	18	7671 Tamara Ferry Suite 498 Davidtown, NM 70805	victorrobinson@example.com	Active
16	16	Joseph	Carlson	94	5281 Jennifer Vista Port Karen, CA 22265	myerstyler@example.com	Active
17	17	Gregory	James	70	39966 Justin Stream Reevesstad, CO 74537	kristopher28@example.com	Inactive
18	18	Frank	Martinez	50	5252 Nicholas Center Suite 062 East Jeremy, VT 84671	meghanlee@example.org	Inactive
19	19	Katie	Romero	20	12500 Joshua Mill Jasonmouth, HI 96209	diana42@example.net	Active
20	20	Bethany	Brown	63	9410 Rice Landing Apt. 955 Lauraside, CO 77799	amurray@example.com	Active
21	21	Benjamin	Garcia	95	0422 Stephen Way East Melissa, SD 39300	fhayes@example.net	Inactive

Total rows: 1000 of 25000 Query complete 00:00:00.247 Ln 19, Col 1

Figure 6:Customer relation:

Book_type relation:

	book_type_id [PK] integer	book_type character varying (255)
1	1	For Children
2	2	For Adults

Figure 7: Book_type relation

Book relation:

	book_id [PK] integer	book_title character varying (255)	book_type_id integer	price double precision
1	360467	Appear opportunity.	1	15.729783747767987
2	32797	Small step.	2	18.174270342648754
3	196641	Ability create range.	1	24.906249461709482
4	360486	Thank onto fly.	1	27.057841042643005
5	917551	Hold for.	2	23.525798704386016
6	950322	Article decide leader.	1	14.279267637590843
7	753714	Network floor.	1	11.759296813604163
8	426036	Quite ago nor.	2	18.03504039967695
9	294969	Television administration.	1	23.735242986680984
10	622654	Police doctor including particular.	1	25.745639457220506
11	753747	Participant.	1	7.451826023566681
12	163933	Might culture.	1	12.626010995375186
13	262257	Case administration keep.	1	13.350236031722611
14	196725	Her.	2	11.511130549304148
15	950425	Have approach director.	1	26.331016319651788
16	327837	Course later.	1	20.28113980055981
17	167	Week start.	1	17.05527508028535
18	360622	Type than goal.	2	16.79744052355874
19	32954	May against.	1	18.728536984641686
20	164037	Reason international which.	2	22.63044285924934
21	262354	Career owner begin.	1	19.592178855190813

Total rows: 1000 of 5000 Query complete 00:00:00.329

Figure 8: Book relation

Book_copy relation:

	unique_book_id [PK] integer	published_year date	book_id integer	published_by character varying (255)	available character varying (20)
1	524288	1911-07-16	181506	Johnson-Jenkins	Unavailable
2	786437	1906-08-26	349670	Holmes-Bates	Available
3	655366	2008-11-13	178329	Harris, Young and Mcbride	Unavailable
4	393226	1961-09-21	101867	Hughes-Acosta	Unavailable
5	11	1924-03-04	749864	Pugh and Sons	Available
6	262161	1924-08-05	227745	Lee, Romero and Smith	Unavailable
7	917521	1962-01-15	615868	Nelson Ltd	Unavailable
8	131089	1917-04-04	410795	Hurst Inc	Unavailable
9	524307	1930-11-09	45213	Zimmerman-Holt	Unavailable
10	131092	1984-11-08	643507	Vega, Smith and Mckee	Unavailable
11	262163	1944-04-07	514933	Wagner Group	Available
12	131096	1963-02-03	732648	Foley, Myers and Ellis	Unavailable
13	655387	1964-09-08	488947	Hendricks Group	Unavailable
14	655389	2019-02-12	349988	Smith, Walker and Harvey	Unavailable
15	655393	1952-12-22	462862	Cook, Chung and Holmes	Available
16	131107	1924-07-11	199917	Williamson Inc	Available
17	524324	1936-09-17	179664	Frank, Wiley and Douglas	Unavailable
18	35	1970-05-14	375487	Richards LLC	Available
19	524329	1947-12-28	849637	Silva-Palmer	Unavailable
20	917545	1958-08-24	638149	Burgess PLC	Available
21	524330	1988-11-17	776905	Osborn-Walker	Unavailable

Total rows: 1000 of 45940 Query complete 00:00:00.447

Figure 9:Book_copy relation

Book_category_relation relation:

	book_id [PK] integer	category_id [PK] integer
1	98177	38
2	397965	42
3	581598	45
4	719075	43
5	457672	68
6	12970	53
7	244789	35
8	134074	62
9	285346	7
10	226221	63
11	729336	5
12	791566	8
13	52214	17
14	136071	53
15	649217	37
16	734105	11
17	203708	4
18	759340	56
19	970129	4
20	392004	38
21	976940	68

Total rows: 1000 of 38359 Query complete 00:00:00.257

Figure 10::Book_author_relation relation

Book_author_relation relation:

	book_id [PK] integer	author_id [PK] integer
1	632470	929605
2	149662	862302
3	419159	14869
4	862395	601215
5	460617	733897
6	373824	40816
7	617659	918335
8	496370	928487
9	601799	81859
10	553821	32379
11	319586	172544
12	34467	620084
13	857597	262132
14	956320	978382
15	163752	837334
16	470962	706923
17	807896	334279
18	520745	195483
19	364993	375220
20	269258	218693
21	69755	203696

Total rows: 1000 of 46991 Query complete 00:00:00.259

Figure 11:Book_category_relation relation:

Checkout relation:

	checkout_id [PK] integer	issue_date date	due_date date	actual_return_date date	fine double precision	unique_book_id integer	customer_id_card_number integer
1	1	2023-02-22	2023-04-21	2023-04-27	3.7309494038337045	497424	1023
2	2	2023-03-21	2023-04-05	2023-04-23	11.192848211501115	213587	15568
3	3	2023-01-17	2023-04-07	2023-04-12	3.109124503194754	501039	22544
4	4	2023-04-23	2023-04-29	2023-05-04	3.109124503194754	852794	6458
5	5	2023-02-20	2023-03-28	2023-03-29	0.6218249006389508	742196	3379
6	6	2023-01-19	2023-05-03	2023-05-06	1.8654747019168523	420639	18712
7	7	2023-04-21	2023-05-02	2023-05-04	1.2436498012779016	40401	18605
8	8	2023-01-15	2023-04-30	2023-05-06	3.7309494038337045	634073	13805
9	9	2023-01-07	2023-03-11	2023-04-11	19.276571919807473	672650	9003
10	10	2023-02-11	2023-03-26	2023-05-01	22.38569642300223	634080	17541
11	11	2023-04-07	2023-04-24	2023-04-29	3.109124503194754	809736	24992
12	12	2023-03-25	2023-04-17	2023-04-27	6.218249006389508	46454	1195
13	13	2023-01-22	2023-02-22	2023-03-16	13.680147814056918	208777	9624
14	14	2023-04-08	2023-04-15	2023-04-26	6.840073907028459	272643	23004
15	15	2023-01-27	2023-03-11	2023-03-12	0.6218249006389508	954523	20564
16	16	2023-03-22	2023-04-12	2023-04-27	9.327373509584262	35548	8431
17	17	2023-01-18	2023-02-28	2023-03-03	1.8654747019168523	543955	7067
18	18	2023-01-08	2023-04-23	2023-04-25	1.2436498012779016	468448	14076
19	19	2023-01-08	2023-02-25	2023-04-10	27.360295628113835	161264	22641
20	20	2023-01-17	2023-05-01	2023-05-05	2.487299602555803	104374	7521
21	21	2023-01-12	2023-03-10	2023-03-29	11.814673112140063	499262	18474

Total rows: 1000 of 23064 Query complete 00:00:00.326

Figure 12: Checkout relation:

DDL Commands for Database Creation:

- Author Table:
  CREATE TABLE author(author_id int, name varchar(255) not null, primary key(author_id));

- Category Table:
  CREATE TABLE category(category_id int, category_name varchar(255) not null, primary key(category_id));

- Book_Type Table:
  CREATE TABLE book_type(book_type_id int, book_type varchar(255) not null, primary key(book_type_id));

- Book Table:
  CREATE TABLE book(book_id int, book_title varchar(255) not null, book_type_id int, price float not null, primary key(book_id), foreign key(book_type_id) references book_type(book_type_id));

- Book_coy Table:
  CREATE TABLE book_copy(unique_book_id int, published_year DATE not null, book_id int,published_by varchar(255) not null, available varchar(10) not null, primary key(unique_book_id), foreign key(book_id) references book(book_id));

- Book_category_relation Table:
  CREATE TABLE book_category_relation(book_id int, category_id int,primary key(book_id,category_id), foreign key(category_id) references category(category_id), foreign key(book_id) references book(book_id));

- book_author_relation :
  CREATE TABLE book_author_relation(book_id int,author_id int,primary key(book_id,author_id), foreign key(author_id) references author(author_id), foreign key(book_id) references book(book_id));

- Customer Table:
  CREATE TABLE Customer(customer_id_card_number serial, customer_first_name text not null, customer_last_name text not null, customer_age int not null, customer_address varchar(255) not null, customer_email varchar (255) unique, customer_status varchar (10) not null, primary key(customer_id_card_number));

- Checkout Table:
  CREATE TABLE checkout(checkout_id serial, issue_date DATE not null, due_date DATE not null, actual_return_date date, fine float, unique_book_id int, customer_id_card_number int, primary key(checkout_id), foreign key(unique_book_id) references book_copy(unique_book_id), foreign key(customer_id_card_number) references

Customer(customer_id_card_number));

SQL Queries:

- **Generate fine of the book, we charge 5% for each day extended**

  UPDATE checkout SET fine = (actual_return_date-due_date)*0.05*(select b.price from book as b, book_copy as c where b.book_id=c.book_id and c.unique_book_id in (select unique_book_id from checkout) LIMIT 1) WHERE actual_return_date>due_date;

- **Update a copy of the book to null if it is not returned yet:**

  UPDATE book_copy SET available = 'Not'
  WHERE unique_book_id = (select unique_book_id from checkout where actual_return_date Is Null);

- **Calculate the number of copies of the books in the library:**

  select b.book_title, count(c.book_id) as books
  from book b, book_copy c
  where b.book_id=c.book_id
  group by b.book_title
  order by count(c.book_id) desc;

```
select b.book_title, count(c.book_id) as books from book b, book_copy c
where b.book_id=c.book_id group by b.book_title order by count(c.book_id) desc;
```

Data Output | Messages | Notifications

	book_title character varying (255)	books bigint
1	Pride and Prejudice	3
2	The Adventures of Tom Sawyer	2
3	The Great Gatsby	2
4	The Fall of the House of Usher	2
5	The Metamorphosis	1
6	Jane Eyre	1
7	Ulysses	1
8	To the Lighthouse	1
9	Harry Potter and the Philosophers Stone	1

Figure 13: Number of copies of the books in the library

- **Demonstrating window function to calculate the number of copies of the books which are available in the library**

```
select b.book_title, count(c.book_id) as books
from book b, book_copy c
where b.book_id=c.book_id and available='Available'
group by b.book_title
order by count(c.book_id) desc;
```

```
select b.book_title, count(c.book_id) as books from book b, book_copy c
where b.book_id=c.book_id and available='Available' group by b.book_title order by count(c.book_id) desc
```

Data Output | Messages | Notifications

	book_title character varying (255)	books bigint
1	Pride and Prejudice	3
2	The Great Gatsby	2
3	The Adventures of Tom Sawyer	2
4	The Fall of the House of Usher	2
5	Jane Eyre	1
6	Kafka on the Shore	1
7	Leaves of Grass	1
8	Middlemarch	1

- **Demonstrating window function to find out the latest date on which a book was issued in the library**

```
select b.book_title, Max(issue_date)
from checkout c, book_copy bc, book b
where c.unique_book_id=bc.unique_book_id and bc.book_id=b.book_id
group by b.book_title order by Max(issue_date) desc limit 1;
```

```
select b.book_title, Max(issue_date) from checkout c, book_copy bc, book b
where c.unique_book_id=bc.unique_book_id and bc.book_id=b.book_id group by b.book_title order by Max(issue_date) desc limit 1;
```

Data Output | Messages | Notifications

	book_title character varying (255)	max date
1	Frankenstein	2023-03-03

Figure 14: latest date on which a book was issued in the library

- **Retrieves information about books borrowed by active customers, including the book title, author name, category name, customer name, and the corresponding checkout details:**

```
SELECT  b.book_title AS "Book Title",  a.name AS "Author Name",  cat.category_name AS
"Category Name",  CONCAT (cust.customer_first_name, ' ', cust.customer_last_name) AS
"Customer Name",  ch.issue_date AS "Issue Date",  ch.due_date AS "Due Date",
ch.actual_return_date AS "Actual Return Date",  ch.fine AS "Fine"
FROM  book_copy bc
INNER JOIN book b ON bc.book_id = b.book_id
INNER JOIN book_author_relation ba ON b.book_id = ba.book_id
```

```
INNER JOIN author a ON a.author_id = ba.author_id
INNER JOIN book_category_relation bcat ON b.book_id = bcat.book_id
INNER JOIN category cat ON bcat.category_id = cat.category_id
INNER JOIN checkout ch ON bc.unique_book_id = ch.unique_book_id
INNER JOIN customer cust ON ch.customer_id_card_number =
cust.customer_id_card_number
WHERE cust.customer_status = 'Active';
```

	Book Title character varying (255)	Author Name character varying (255)	Category Name character varying (255)	Customer Name text	Issue Date date	Due Date date	Actual Return Date date	Fine double precision
1	Always upon.	Carlos Shaw	Astronomy	Alison Campbell	2023-03-29	2023-04-24	2023-05-04	6.218249006389508
2	Always upon.	Erica Walker	Astronomy	Alison Campbell	2023-03-29	2023-04-24	2023-05-04	6.218249006389508
3	Always upon.	Cynthia Acosta	Astronomy	Alison Campbell	2023-03-29	2023-04-24	2023-05-04	6.218249006389508
4	Always upon.	Elizabeth Washington	Astronomy	Alison Campbell	2023-03-29	2023-04-24	2023-05-04	6.218249006389508
5	Always upon.	Kelly Miller	Astronomy	Alison Campbell	2023-03-29	2023-04-24	2023-05-04	6.218249006389508
6	Always upon.	Donald Newton	Astronomy	Alison Campbell	2023-03-29	2023-04-24	2023-05-04	6.218249006389508
7	Always upon.	Erin Schwartz	Astronomy	Alison Campbell	2023-03-29	2023-04-24	2023-05-04	6.218249006389508
8	Always upon.	Jose Hester	Astronomy	Alison Campbell	2023-03-29	2023-04-24	2023-05-04	6.218249006389508
9	Always upon.	Jamie Long MD	Astronomy	Alison Campbell	2023-03-29	2023-04-24	2023-05-04	6.218249006389508
10	Always upon.	Amy Rollins	Astronomy	Alison Campbell	2023-03-29	2023-04-24	2023-05-04	6.218249006389508
11	Always upon.	Erik House	Astronomy	Alison Campbell	2023-03-29	2023-04-24	2023-05-04	6.218249006389508
12	Always upon.	Keith Key	Astronomy	Alison Campbell	2023-03-29	2023-04-24	2023-05-04	6.218249006389508
13	Always upon.	William Acosta	Astronomy	Alison Campbell	2023-03-29	2023-04-24	2023-05-04	6.218249006389508
14	Always upon.	Austin Stevens	Astronomy	Alison Campbell	2023-03-29	2023-04-24	2023-05-04	6.218249006389508
15	Always upon.	Craig Jackson	Astronomy	Alison Campbell	2023-03-29	2023-04-24	2023-05-04	6.218249006389508
16	Always upon.	Carlos Shaw	Astronomy	Brian Hobbs	2023-03-08	2023-04-21	2023-04-24	1.8654747019168523
17	Always upon.	Erica Walker	Astronomy	Brian Hobbs	2023-03-08	2023-04-21	2023-04-24	1.8654747019168523
18	Always upon.	Cynthia Acosta	Astronomy	Brian Hobbs	2023-03-08	2023-04-21	2023-04-24	1.8654747019168523
19	Always upon.	Elizabeth Washington	Astronomy	Brian Hobbs	2023-03-08	2023-04-21	2023-04-24	1.8654747019168523
20	Always upon.	Kelly Miller	Astronomy	Brian Hobbs	2023-03-08	2023-04-21	2023-04-24	1.8654747019168523
21	Always upon.	Donald Newton	Astronomy	Brian Hobbs	2023-03-08	2023-04-21	2023-04-24	1.8654747019168523

Total rows: 1000 of 1665876 Query complete 00:00:02.753

Figure 15: Information of books borrowed by active customers

- **<u>Retrieves information about books, authors, and their respective categories in ascending order based on the book title where Book Type- "For Adults",Category - either "Fiction" or "Non-Fiction", book price> 20</u>**

```
SELECT  b.book_title AS "Book Title", a.name AS "Author Name", cat.category_name AS
"Category Name"
FROM book b
INNER JOIN book_author_relation ba ON b.book_id = ba.book_id
INNER JOIN author a ON a.author_id = ba.author_id
INNER JOIN book_category_relation bcat ON b.book_id = bcat.book_id
INNER JOIN category cat ON bcat.category_id = cat.category_id
WHERE b.book_type_id = 2 AND cat.category_name IN ('Fiction', 'Non-Fiction')
AND b.price > 20
ORDER BY
b.book_title ASC;
```

	Book Title character varying (255)	Author Name character varying (255)	Category Name character varying (255)
1	A system.	Dalton Johnson	Fiction
2	A system.	Tara Williams	Fiction
3	A system.	Michael Davis	Fiction
4	A system.	Charles Oliver	Fiction
5	A system.	Miranda Hammond	Fiction
6	A system.	Nicholas Gonzales	Fiction
7	A system.	Erik Williams	Fiction
8	Ability final fire.	Carrie Williams	Non-Fiction
9	Ability final fire.	Dr. Tanner Peterson	Non-Fiction
10	Ability final fire.	Donald Bush	Non-Fiction
11	Ability final fire.	Elizabeth King	Non-Fiction
12	Ability final fire.	Steven Dickson	Non-Fiction
13	Ability final fire.	Zachary Gibbs	Non-Fiction
14	Ability final fire.	Mary Wu	Non-Fiction
15	Ability final fire.	Elizabeth King	Fiction
16	Ability final fire.	Scott Dillon	Fiction
17	Ability final fire.	Donald Bush	Fiction
18	Ability final fire.	Dr. Tanner Peterson	Fiction
19	Ability final fire.	Carrie Williams	Fiction
20	Ability final fire.	Scott Dillon	Non-Fiction
21	Ability final fire.	David Avila	Non-Fiction

Total rows: 1000 of 2094 Query complete 00:00:00.193

Figure 16: Information about books, authors, and their respective categories in ascending order

- **Calculate the total number of books written by each author, the average and maximum fine amount for the checkouts associated with each book.**

```
SELECT a.name AS "Author Name", COUNT(DISTINCT b.book_id) AS "Total Books",
AVG(ch.fine) AS "Average Fine", MAX(ch.fine) AS "Max Fine"
FROM author a
INNER JOIN book_author_relation bar ON a.author_id = bar.author_id
INNER JOIN book b ON bar.book_id = b.book_id
INNER JOIN book_copy bc ON b.book_id = bc.book_id
INNER JOIN checkout ch ON bc.unique_book_id = ch.unique_book_id
GROUP BY a.name;
```

	Author Name character varying (255)	Total Books bigint	Average Fine double precision	Max Fine double precision
1	Aaron Hall	46	9.993371763820512	62.182490063895074
2	Aaron Murray	51	10.81312726807814	60.31701536197822
3	Aaron Smith	45	10.695388290989937	52.85511655431082
4	Aaron Torres	43	10.600918738777484	51.61146675303292
5	Aaron Wilson	44	12.398034616863193	67.77891416964563
6	Aaron Zimmerman	56	10.251642428043713	54.72059125622767
7	Abigail Lara	43	11.183613188224292	50.36781695175501
8	Abigail Sullivan	28	10.014389730451486	67.77891416964563
9	Adam Little	45	9.290629310910141	62.80431496453403
10	Adam Mcdaniel	61	10.216959310498341	54.72059125622767
11	Adam Perez	53	9.353381395162314	51.61146675303292
12	Adrian Liu	50	9.809723666154651	52.23329165367187
13	Adrian Santos	47	9.968805501684804	53.47694145494977
14	Adrienne Edwards	43	11.131632288640281	67.77891416964563
15	Alexa Wall	49	10.465123000272982	63.42613986517298
16	Alexander Day	52	9.904179412891876	62.80431496453403
17	Alexander Pham	51	9.547603161893885	60.31701536197822
18	Alexander Shaffer	42	11.105267239580126	70.8880386728404
19	Alexandra Terrell	45	10.061719106529305	57.82971575942243
20	Alexis Adkins	41	10.09493862131047	60.31701536197822
21	Alexis Carlson	45	9.813587660615784	57.207890858783465

Total rows: 1000 of 1000 Query complete 00:00:01.008

Figure 17:Total number of books written by each author, average and maximum fine amount

- **<u>Calculate the total number of books in each category, the average and maximum price of the books in each category</u>**

```
SELECT  cat.category_name AS "Category", COUNT (DISTINCT b.book_id) AS "Total Books",
AVG(b.price) AS "Average Price", MAX(b.price) AS "Max Price"
FROM category cat
INNER JOIN book_category_relation bcr ON cat.category_id = bcr.category_id
INNER JOIN book b ON bcr.book_id = b.book_id
INNER JOIN book_copy bc ON b.book_id = bc.book_id
GROUP BY   cat.category_name;
```

	Category character varying (255)	Total Books bigint	Average Price double precision	Max Price double precision
1	Accounting	545	17.786497211516263	29.929260080540384
2	Anthropology	558	17.190724837325313	29.99292147865067
3	Architecture	572	17.463120745442357	29.951476903729617
4	Art	524	17.6962768455466	29.94221819329737
5	Astronomy	563	17.273675457322483	29.956224994571155
6	Autobiography	572	17.79463131642904	29.995264266075097
7	Biography	548	17.40676642903598	29.975361961535164
8	Biology	568	17.874458832843423	29.986155578478982
9	Business	546	17.598127167756594	29.995264266075097
10	Chemistry	563	17.334512984463977	29.978749526019193
11	Children	529	17.71777243221474	29.995264266075097
12	Comics	573	17.56315123810675	29.777941226879538
13	Computer Science	564	17.382549134853857	29.91453307850596
14	Cooking	543	17.80558346072371	29.975361961535164
15	Culture	537	17.481141535035977	29.956224994571155
16	Cycling	544	17.51957679866721	29.94221819329737
17	Drama	561	17.499992453198477	29.89046950909259
18	Economics	519	18.44133866166155	29.862029945910887
19	Education	535	17.43765633217239	29.984772007085088
20	Engineering	575	17.458779955200043	29.983998260138247
21	Environmental Science	587	17.618119640202803	29.995264266075097

Total rows: 70 of 70 Query complete 00:00:00.918

Figure 18: Total number of books, the average and maximum price for each category.

- **<u>Retrieve information about customers and their checkouts count for each customer in descending order.</u>**

```
SELECT  cust.customer_id_card_number AS "Customer ID",
CONCAT(cust.customer_first_name, ' ', cust.customer_last_name) AS "Customer Name",
COUNT(ch.checkout_id) AS "Checkouts Made"
FROM customer cust
LEFT JOIN checkout ch ON cust.customer_id_card_number = ch.customer_id_card_number
GROUP BY cust.customer_id_card_number, cust.customer_first_name,
cust.customer_last_name
Order By "Checkouts Made" desc
```

	Customer ID integer	Customer Name text	Checkouts Made bigint
1	10022	Michael Huang	10
2	20570	William Lopez	8
3	20866	Richard Flores	8
4	23275	Adrian Moore	8
5	1499	Kevin Tucker	8
6	65	Charles Oliver	8
7	19267	Tony Palmer	8
8	20435	John Chavez	7
9	1559	Jeffrey Bennett	7
10	22715	Anthony Murillo	7
11	14726	Lacey Ortiz	7
12	17813	Zachary Rosario	7
13	1766	Kristi Gray	7
14	14166	Elizabeth Mahoney	7
15	2774	Jose Torres	7
16	22332	Aaron Lewis	7
17	16926	Alexandra Baker	7
18	2069	Alexander Chavez	7
19	1149	Noah Ayers	7
20	23364	Alexander Velasquez	7
21	21171	Randall Jones	7

Total rows: 1000 of 25000 Query complete 00:00:00.135

Figure 19: Information about customers and their checkouts count in descending order.

Foreign Key Deletion Scenarios - Possibility and Removal:

1. The book_author_relation table forms an association between author table and book table. So, the PK of author table which is author_id and the PK of book table, both can only be removed only after this association table is first removed.

2. The book_category_relation table forms an association between category table and book table. So, the PK of category table which is category_id and the PK of book table, both can only be removed only after this association table is first removed.

3. The checkout table uses unique_book_id and customer_id_card_number as its foreign key. So, in order to remove the PK of customer table which is customer_id_card_number and the PK of book_copy table which is unique_book_id, we must first drop the checkout table. Only then, the PK of both the referenced tables can be removed.

4. The book table uses book_type_id as the foreign key referencing the book_type table. So, to remove the PK of book_type table which is book_type_id, we must first drop the book table.

5. Similarly, the book_copy table uses book_id as the foreign key referencing the book table. So, to remove the PK of book table which is book_id, we must first drop the book_copy table.

Section 3: Normalization:

Normalization is a database design technique aimed at minimizing data redundancy and removing unwanted issues such as Insertion, Update, and Deletion Anomalies. It involves breaking down larger tables into smaller ones and establishing relationships between them. The goal of normalization in SQL is to eliminate repetitive data and ensure logical storage of data. By adhering to normalization rules, databases can be organized more efficiently and maintain data integrity.

Boyce Codd Normal Form:

BCNF and 3NF are two normalization techniques used in database design to eliminate data redundancy and improve data integrity. BCNF is based on functional dependencies, which are relationships between attributes in a relation. It aims to eliminate non-trivial functional dependencies by ensuring that every determinant is a candidate key. BCNF guarantees the preservation of all functional dependencies and decomposes relations into separate tables based on determinants.3NF also focuses on functional dependencies but specifically targets transitive dependencies. It aims to remove dependencies where an attribute depends on another attribute through a third attribute. 3NF reduces data redundancy by breaking down relations into smaller tables.

In terms of normalization levels, BCNF is a higher level than 3NF as it eliminates all non-trivial functional dependencies. To determine the BCNF (Boyce-Codd Normal Form) and 3NF (Third Normal Form) for each table, we need to analyze the functional dependencies and the primary keys. Here's the analysis for each table.

Table: author

Primary Key: author_id

Functional Dependencies:
author_id -> name

BCNF: The table is already in BCNF since the primary key determines all other attributes.

3NF: The table is already in 3NF since there are no transitive dependencies.

Table: category

Primary Key: category_id

Functional Dependencies:
category_id -> category_name

BCNF: The table is already in BCNF since the primary key determines all other attributes.

3NF: The table is already in 3NF since there are no transitive dependencies.

Table: book_type

Primary Key: book_type_id

Functional Dependencies:
book_type_id -> book_type

BCNF: The table is already in BCNF since the primary key determines all other attributes.

3NF: The table is already in 3NF since there are no transitive dependencies.

Table: book

Primary Key: book_id

Functional Dependencies:
book_id -> book_title, book_type_id, price

BCNF: The table is already in BCNF since the primary key determines all other attributes.

3NF: The table is already in 3NF since there are no transitive dependencies.

Table: book_category

Primary Key: (book_id, category_id)

Functional Dependencies:
(book_id, category_id) -> None

BCNF: The table is already in BCNF since the primary key determines all other attributes.

3NF: The table is already in 3NF since there are no transitive dependencies.

Table: book_author

Primary Key: (book_id, author_id)

Functional Dependencies:
(book_id, author_id) -> None

BCNF: The table is already in BCNF since the primary key determines all attributes of the table.

3NF: The table is already in 3NF since there are no transitive dependencies.

Table: customer

Primary Key: customer_id_card_number

Functional Dependencies:
customer_id_card_number -> customer_first_name, customer_last_name, customer_age, customer_address, customer_email, customer_status

BCNF: The table is already in BCNF since the primary key determines all other attributes.

3NF: The table is already in 3NF since there are no transitive dependencies.

Table: book_copy

Primary Key: unique_book_id

Functional Dependencies:
unique_book_id -> published_year, book_id, published_by, available

BCNF: The table is already in BCNF since the primary key determines all other attributes.

3NF: The table is already in 3NF since there are no transitive dependencies.

Table: checkout
Primary Key: checkout_id

Functional Dependencies:
checkout_id -> issue_date, due_date, actual_return_date, fine, unique_book_id, customer_id_card_number

BCNF: The table is already in BCNF since the primary key determines all other attributes.

3NF: The table is already in 3NF since there are no transitive dependencies.

All the tables are already in BCNF and 3NF, meaning they meet the highest normal form requirements.

Query Optimization:

SQL query optimization is the ongoing process of improving the performance of a query by reducing its execution time, minimizing disk accesses, and optimizing other cost-related factors. Efficient data access is crucial for any application, as it directly impacts the user experience. The aim is to retrieve data as quickly as possible, ensuring a smooth and responsive application usage.

Why Do We Need To Query Optimization?

Query optimization is required for several reasons as given below:

1. Enhanced Performance: By optimizing the execution plan, query optimization improves the performance of database queries. This results in faster response times and overall system performance improvement.
2. Efficient Resource Utilization: Optimized queries consume fewer system resources such as CPU, memory, and disk I/O. This efficient resource utilization allows the system to handle more concurrent queries without performance bottlenecks.
3. Scalability: Query optimization becomes crucial as data volume and user access increase. Optimized queries ensure the system can scale effectively and handle larger workloads without significant performance degradation.
4. Cost Savings: Query optimization reduces the cost of executing queries by minimizing disk I/O operations and optimizing resource usage. This leads to hardware cost savings and improved efficiency of the database system.
5. Improved User Experience: Faster query response times and improved system performance provide a better user experience. Applications that deliver quick and responsive results are more likely to satisfy users and maintain their engagement.
6. SLA Compliance: Query optimization helps meet service level agreements (SLAs) by ensuring that the database system meets the agreed-upon performance standards and response time requirements.

Implementation of Query optimization:

Two indexing methods have been used for the query optimization namely B-tree indexing and hash indexing.

In B-tree indexing, data is organized in a balanced tree structure, sorted in either ascending or descending order. It can handle duplicate keys and store them in a sorted order, and it is optimized

for disk access and well-suited for scenarios where data resides on disk.

Hash indexing uses a hash function to directly compute the storage location of data and it is ideal for exact match lookups. It is primarily optimized for in-memory data access.

Query Optimization 1 using B-tree indexing:

The query for selecting entries from Customer table based on customer status and customer age has been optimized using indexing. Before implementation of the index, the Postgres performed a sequential scan and the execution time for the query was 4.863 ms. A B-tree index 'index_customer_id' is implemented on the 'customer_id_card_number' column of the customer table and the time required to execute the same query was reduced to 4.275 ms, thus optimizing the query by 0.588 ms.

Before Indexing:

```
explain analyze select * from customer where customer_status='Active' and customer_age > 50
```

	QUERY PLAN text
1	Seq Scan on customer (cost=0.00..779.00 rows=7516 width=95) (actual time=0.012..4.627 rows=7451 loops...
2	Filter: ((customer_age > 50) AND ((customer_status)::text = 'Active'::text))
3	Rows Removed by Filter: 17549
4	Planning Time: 0.098 ms
5	Execution Time: 4.863 ms

Figure 20: Execution time before B+-Tree Indexing

After Indexing:

```
create index index_customer_id on customer(customer_id_card_number)

explain analyze select * from customer where customer_status='Active' and customer_age > 50
```

	QUERY PLAN text
1	Seq Scan on customer (cost=0.00..779.00 rows=7516 width=95) (actual time=0.012..4.056 rows=7451 loops...
2	Filter: ((customer_age > 50) AND ((customer_status)::text = 'Active'::text))
3	Rows Removed by Filter: 17549
4	Planning Time: 1.406 ms
5	Execution Time: 4.275 ms

Figure 21: Execution time after B+-Tree Indexing

Query Optimization 1 using hash indexing:

The query to calculate the total number of books in each category, the average and maximum price of the books in each category has been optimized using indexing. Before implementation of the index, the Postgres performed a sequential scan and the execution time for the query was 1524.522 ms. A hash index 'ind1' was implemented on the book_id' column of the book table and a hash index 'ind2' was implemented on the 'category_name' column of the category table. The time required to execute the same query was reduced to 1326.997 ms, thus optimizing the query by 197.525 ms.

Before Indexing:

```
explain analyze
SELECT
  cat.category_name AS "Category",
  COUNT(DISTINCT b.book_id) AS "Total Books",
  AVG(b.price) AS "Average Price",
  MAX(b.price) AS "Max Price"
FROM
  category cat
  INNER JOIN book_category_relation bcr ON cat.category_id = bcr.category_id
  INNER JOIN book b ON bcr.book_id = b.book_id
  INNER JOIN book_copy bc ON b.book_id = bc.book_id
GROUP BY
  cat.category_name;
```

	QUERY PLAN text
3	-> Sort (cost=46132.94..47014.05 rows=352442 width=21) (actual time=1214.745..1371.550 rows=352855 loops=1)
4	Sort Key: cat.category_name
5	Sort Method: external merge Disk: 12200kB
6	-> Hash Join (cost=1717.42..6429.65 rows=352442 width=21) (actual time=37.144..227.943 rows=352855 loops=1)
7	Hash Cond: (bcr.book_id = b.book_id)
8	-> Hash Join (cost=2.58..662.95 rows=38359 width=13) (actual time=0.043..23.298 rows=38359 loops=1)
9	Hash Cond: (bcr.category_id = cat.category_id)
10	-> Seq Scan on book_category_relation bcr (cost=0.00..553.59 rows=38359 width=8) (actual time=0.011..6.452 rows=38359 loops=1)
11	-> Hash (cost=1.70..1.70 rows=70 width=13) (actual time=0.023..0.024 rows=70 loops=1)
12	Buckets: 1024 Batches: 1 Memory Usage: 12kB
13	-> Seq Scan on category cat (cost=0.00..1.70 rows=70 width=13) (actual time=0.005..0.011 rows=70 loops=1)
14	-> Hash (cost=1140.59..1140.59 rows=45940 width=16) (actual time=36.906..36.908 rows=45940 loops=1)
15	Buckets: 65536 Batches: 1 Memory Usage: 2845kB
16	-> Hash Join (cost=152.50..1140.59 rows=45940 width=16) (actual time=1.340..23.976 rows=45940 loops=1)
17	Hash Cond: (bc.book_id = b.book_id)
18	-> Seq Scan on book_copy bc (cost=0.00..867.40 rows=45940 width=4) (actual time=0.007..3.840 rows=45940 loops=1)
19	-> Hash (cost=90.00..90.00 rows=5000 width=12) (actual time=1.310..1.311 rows=5000 loops=1)
20	Buckets: 8192 Batches: 1 Memory Usage: 299kB
21	-> Seq Scan on book b (cost=0.00..90.00 rows=5000 width=12) (actual time=0.005..0.578 rows=5000 loops=1)
22	Planning Time: 0.820 ms
23	Execution Time: 1524.522 ms

Total rows: 23 of 23 | Query complete 00:00:01.544

Figure 22: Executiion time before using hash indexing

After Indexing:

```
1  create index ind1 on book USING hash(book_id);
2  create index ind2 on category USING hash(category_name);
3
4  explain analyze
5  SELECT
6    cat.category_name AS "Category",
7    COUNT(DISTINCT b.book_id) AS "Total Books",
8    AVG(b.price) AS "Average Price",
9    MAX(b.price) AS "Max Price"
10 FROM
11   category cat
12   INNER JOIN book_category_relation bcr ON cat.category_id = bcr.category_id
13   INNER JOIN book b ON bcr.book_id = b.book_id
14   INNER JOIN book_copy bc ON b.book_id = bc.book_id
15 GROUP BY
16   cat.category_name;
```

	QUERY PLAN text
3	-> Sort (cost=46132.94..47014.05 rows=352442 width=21) (actual time=1029.564..1175.594 rows=352855 loops=1)
4	Sort Key: cat.category_name
5	Sort Method: external merge Disk: 12200kB
6	-> Hash Join (cost=1717.42..6429.65 rows=352442 width=21) (actual time=53.278..230.458 rows=352855 loops=1)
7	Hash Cond: (bcr.book_id = b.book_id)
8	-> Hash Join (cost=2.58..662.95 rows=38359 width=13) (actual time=0.055..22.045 rows=38359 loops=1)
9	Hash Cond: (bcr.category_id = cat.category_id)
10	-> Seq Scan on book_category_relation bcr (cost=0.00..553.59 rows=38359 width=8) (actual time=0.015..5.481 rows=38359 loops=1)
11	-> Hash (cost=1.70..1.70 rows=70 width=13) (actual time=0.030..0.032 rows=70 loops=1)
12	Buckets: 1024 Batches: 1 Memory Usage: 12kB
13	-> Seq Scan on category cat (cost=0.00..1.70 rows=70 width=13) (actual time=0.009..0.016 rows=70 loops=1)
14	-> Hash (cost=1140.59..1140.59 rows=45940 width=16) (actual time=53.008..53.011 rows=45940 loops=1)
15	Buckets: 65536 Batches: 1 Memory Usage: 2845kB
16	-> Hash Join (cost=152.50..1140.59 rows=45940 width=16) (actual time=1.708..35.380 rows=45940 loops=1)
17	Hash Cond: (bc.book_id = b.book_id)
18	-> Seq Scan on book_copy bc (cost=0.00..867.40 rows=45940 width=4) (actual time=0.011..5.245 rows=45940 loops=1)
19	-> Hash (cost=90.00..90.00 rows=5000 width=12) (actual time=1.669..1.670 rows=5000 loops=1)
20	Buckets: 8192 Batches: 1 Memory Usage: 299kB
21	-> Seq Scan on book b (cost=0.00..90.00 rows=5000 width=12) (actual time=0.007..0.767 rows=5000 loops=1)
22	Planning Time: 4.596 ms
23	Execution Time: 1326.997 ms

Total rows: 23 of 23 | Query complete 00:00:01.351

Figure 23: Executiion time after using hash indexing

Triggers and functions

Triggers and functions are powerful tools in database systems that help in the automation, validation, and customization of data processing tasks. They help in improving the data management and processing tasks.

Triggers are database objects associated with a table that automatically respond to specific events like data insertions, updates, or deletions. They enforce constraints, validate data, audit changes, and initiate actions based on conditions. Triggers can execute before or after events, ensuring data

integrity and consistency. They are implemented using SQL statements and allow for complex logic to enforce business rules.

Functions are named blocks of code used in databases to perform specific tasks and return values or execute actions. They encapsulate reusable logic, accept input parameters, and can be written in different programming languages. Functions enhance code reuse, modularity, and maintainability by encapsulating complex logic into reusable units. They are utilized in queries, stored procedures, triggers, and other database objects to perform calculations, data transformations, or complex operations.

A trigger has been created for the checkout table. The trigger function 'calculate_fine()' is invoked after an insertion occurs on the checkout table. The function updates the 'fine' column for the newly inserted row by calculating the fine amount which is 5% of the price of the book per day.

The trigger and function work together to automatically calculate and assign a fine value for new record inserted into the checkout table. The trigger is triggered and invokes the function to determines the fine based on the actual due date and actual return date of the new record. The calculated fine is then stored as the fine value for that specific record. This process ensures that fines are generated and assigned automatically, relieving the need for manual intervention.

Trigger Function for Checkout Table:

```
/* Drop Trigger*/
DROP TRIGGER after_checkout_insert ON checkout;

/* Drop Fucntion*/
DROP FUNCTION calculate_fine();


/* Function which calculates the fine for each entry in the checkout table*/
CREATE OR REPLACE FUNCTION calculate_fine()
    RETURNS TRIGGER AS
$$
BEGIN
    UPDATE checkout
    SET
        fine = (NEW.actual_return_date - NEW.due_date) * 0.05 * (
            SELECT b.price FROM book AS b, book_copy as c
            where b.book_id=c.book_id
            and c.unique_book_id in
            (select unique_book_id from checkout) LIMIT 1)
    WHERE actual_return_date>due_date and NEW.checkout_id = checkout_id;

    RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER after_checkout_insert
AFTER INSERT ON checkout
FOR EACH ROW
EXECUTE FUNCTION calculate_fine();
```

Records in the Checkout Table before inserting:

	checkout_id [PK] integer	issue_date date	due_date date	actual_return_date date	fine double precision	unique_book_id integer	customer_id_card_number integer
1	23064	2023-01-23	2023-01-31	2023-04-25	88.6857350612252	186654	5608
2	23063	2023-04-03	2023-04-21	2023-05-06	15.836738403790214	198231	17270
3	23062	2023-01-24	2023-04-20	2023-05-01	11.613608162779492	817277	437
4	23061	2023-02-06	2023-04-02	2023-04-11	9.50204304227413	682783	21115
5	23060	2023-01-30	2023-03-21	2023-03-24	3.1673476807580427	984930	18597
6	23059	2023-04-25	2023-05-04	2023-05-06	2.1115651205053623	210891	22615
7	23058	2023-04-22	2023-04-29	2023-05-03	4.2231302410107245	75484	4472
8	23057	2023-04-08	2023-04-12	2023-05-06	25.338781446064342	474708	12018
9	23056	2023-03-02	2023-04-17	2023-05-06	20.05986864480094	172103	6774
10	23055	2023-03-30	2023-05-04	2023-05-05	1.0557825602526811	733007	17560
11	23054	2023-01-06	2023-03-28	2023-04-07	10.55782560252681	608306	12884
12	23053	2023-01-03	2023-02-18	2023-02-24	6.3346953615160855	203701	19526
13	23052	2023-03-13	2023-04-08	2023-04-14	6.3346953615160855	790061	21023
14	23051	2023-01-19	2023-04-21	2023-04-23	2.1115651205053623	209239	6365
15	23050	2023-03-06	2023-03-08	2023-04-22	47.51021521137064	217745	18525
16	23049	2023-02-18	2023-04-09	2023-04-28	20.05986864480094	129910	21753
17	23048	2023-04-18	2023-04-26	2023-05-05	9.50204304227413	772040	10540
18	23047	2023-02-01	2023-02-12	2023-04-15	65.45851873566622	389822	24879
19	23046	2023-02-18	2023-03-11	2023-04-27	49.62178033187601	549707	16797
20	23045	2023-04-03	2023-04-13	2023-05-01	19.00408608454826	574472	6575
21	23044	2023-03-20	2023-04-26	2023-04-28	2.1115651205053623	582759	21248

Total rows: 1000 of 23064 Query complete 00:00:00.156

Figure 24: Records in the Checkout Table before inserting

Inserting an Entry in the Checkout Table and Retrieving the Entries:

```
insert into checkout(checkout_id,issue_date,due_date, actual_return_date, fine, unique_book_id, customer_id_card_number)
values(23065, '2023-04-30', '2023-05-02', '2023-05-05', null, 186654, 4472);

select * from checkout order by checkout_id desc;
```

The fine is inserted as null value for the checkout_id 23065.

Records in the Checkout Table after insertion:

	checkout_id [PK] integer	issue_date date	due_date date	actual_return_date date	fine double precision	unique_book_id integer	customer_id_card_number integer
1	23065	2023-04-30	2023-05-02	2023-05-05	3.1673476807580427	186654	4472
2	23064	2023-01-23	2023-01-31	2023-04-25	88.6857350612252	186654	5608
3	23063	2023-04-03	2023-04-21	2023-05-06	15.836738403790214	198231	17270
4	23062	2023-01-24	2023-04-20	2023-05-01	11.613608162779492	817277	437
5	23061	2023-02-06	2023-04-02	2023-04-11	9.50204304227413	682783	21115
6	23060	2023-01-30	2023-03-21	2023-03-24	3.1673476807580427	984930	18597
7	23059	2023-04-25	2023-05-04	2023-05-06	2.1115651205053623	210891	22615
8	23058	2023-04-22	2023-04-29	2023-05-03	4.2231302410107245	75484	4472
9	23057	2023-04-08	2023-04-12	2023-05-06	25.338781446064342	474708	12018
10	23056	2023-03-02	2023-04-17	2023-05-06	20.05986864480094	172103	6774
11	23055	2023-03-30	2023-05-04	2023-05-05	1.0557825602526811	733007	17560
12	23054	2023-01-06	2023-03-28	2023-04-07	10.55782560252681	608306	12884
13	23053	2023-01-03	2023-02-18	2023-02-24	6.3346953615160855	203701	19526
14	23052	2023-03-13	2023-04-08	2023-04-14	6.3346953615160855	790061	21023
15	23051	2023-01-19	2023-04-21	2023-04-23	2.1115651205053623	209239	6365
16	23050	2023-03-06	2023-03-08	2023-04-22	47.51021521137064	217745	18525
17	23049	2023-02-18	2023-04-09	2023-04-28	20.05986864480094	129910	21753
18	23048	2023-04-18	2023-04-26	2023-05-05	9.50204304227413	772040	10540
19	23047	2023-02-01	2023-02-12	2023-04-15	65.45851873566622	389822	24879
20	23046	2023-02-18	2023-03-11	2023-04-27	49.62178033187601	549707	16797
21	23045	2023-04-03	2023-04-13	2023-05-01	19.00408608454826	574472	6575

Total rows: 1000 of 23065 | Query complete 00:00:00.147

Figure 25: Records in the Checkout Table after insertion:

The fine is calculated and entered as 3.1673476807580427 for the new record with checkout_id 23065 by the trigger and function.

User-Interface Implementation:

We have created a web interface which houses all the library data and displays it in a user-friendly way. Currently, it is deployed on the local server and can be easily scaled to cater a large-scale environment. Currently the capability of the website incorporates information related to author, books and number of copies of a book available in the library. We would like to expand the capability to display several other data by relevance to different use cases.

The website is hosted here: http://127.08.0.1:5000

Here is a screenshot of the main home page.

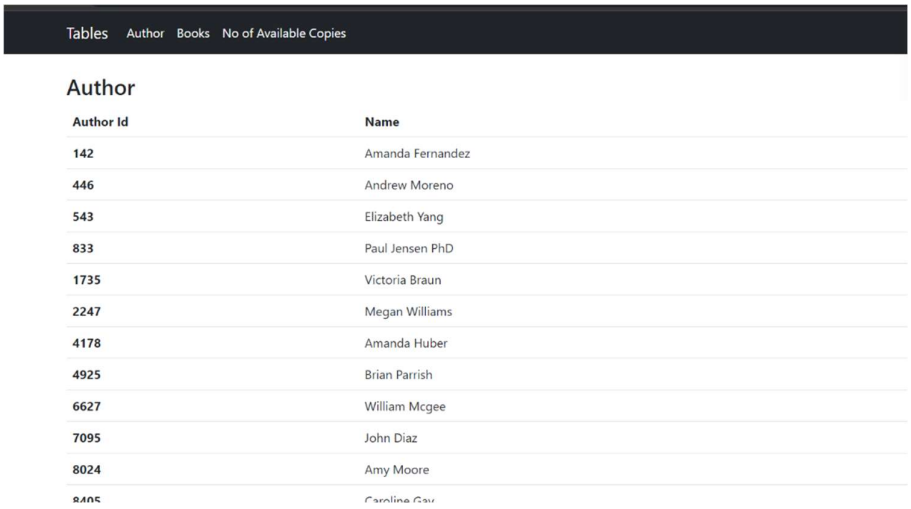

Figure 26: Author page

Different tabs with information fetched from SQL queries.

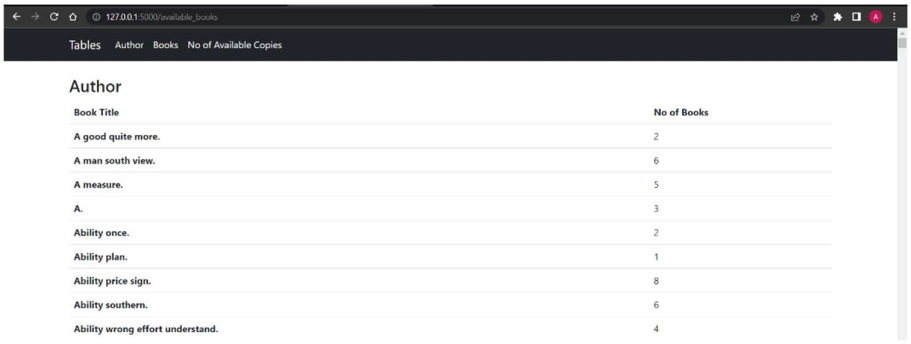

Figure 27: Number of Books

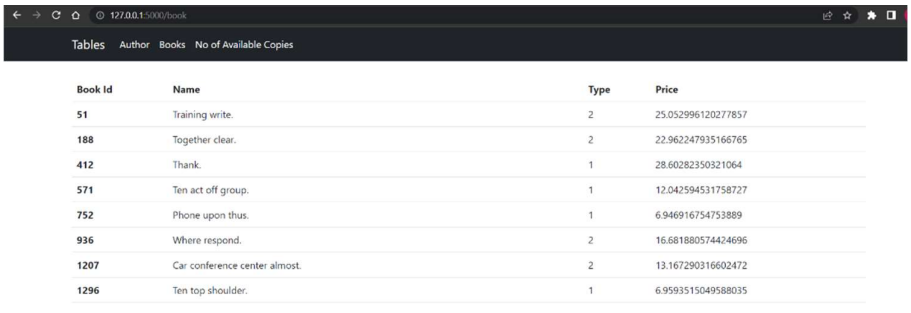

Figure 28: Number of available copies of a book.

Contributions:

Each team member has made equal contributions and worked collaboratively to successfully complete the project.

References:

1. Avi Silberschatz, Henry F. Korth, S. Sudarshan, Database System Concepts, Seventh Edition, McGraw-Hill, 2019.
2. H. Garcia-Molina, J. D. Ullman, J. Widom, Database Systems: The Complete Book, Second Edition, Prentice Hall, 2009.
3. https://www.postgresql.org/
4. https://www.geeksforgeeks.org/
5. https://stackoverflow.com/
6. https://hasura.io/docs/latest/schema/postgres/postgres-guides/import-data-from-csv/
7. https://mkzia.github.io/eas503-notes/intro.html
8. https://www.python.org/