HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

DATABASE LAB - IT3290E

LIBRARY MANAGEMENT SYSTEM

Guided by:

Lecturer Nguyen Thi Oanh

Group members:

Vu Binh Minh - 20226058

Ngo Anh Tu - 20226005

Dang Trong Van - 20226072

Contents

1 Job delegation

Student name	Student id	Contributions
Vu Binh Minh	20226058	Design ERD, relational schema, write report
Ngo Anh Tu	20226005	Create functions, triggers, participate in writing report
Dang Trong Van	20226072	Create dataset, participate in writing report

2 Description

Library Management System is designed to streamline and automate library workflows and activities, helping them manage resources efficiently while providing a better user experience for borrowers.

2.1 Business context

The business operates in the library, where the primary goal is to manage and facilitate the lending of books and other resources to borrowers. The system automates tasks like managing books, authors, publishers, borrowers, and library staff operations. It tracks book loans, returns, updates book information, and handles any compensations for late returns or damaged books.

2.2 Users

- **Borrowers:** People who borrow books from the library. They interact with the system to check out, return, and, if necessary, compensate for lost or damaged books.
- **Staff:** Responsible for updating the book inventory, managing loans, and overseeing the overall operations of book lending and returns.

2.3 Business processes

- **Book updating process:** When a book is acquired, the staff member responsible for entering the book's details into the system. A book can be assigned to one or many authors, but it must be assigned to one publisher. The genre of the book is classified and entered the system. The book's data (book_id, title, language, publish year, price) is updated and categorized according to the system.
- **Book borrowing process:** When user want to borrow a book, they can search them by the book's title. By default if the user doesn't input any title then the system will return all the books in the library. If there is no books whose title the user searching for, the system will show out a message to announce that the user can't borrow that book now.
- **Staff management process:** The library's staff are managed within the system with details such as name, phone, date of birth and staff_id. Staff members are responsible for managing various library operations such as updating books, borrower registration, loan management and compensations.

2.4 Business rules

- When the borrower borrows the books, the loan period for borrowing the books is 3 months.
- The number of books that are borrowed must be less than or equal to 3 books.
- When creating accounts, users will have to pay a deposit of 100,000 VND.
- If the borrower returns the book past the due date, the compensation fee will be calculated in the deposit, and the value will be 100,000 VND.
- 10 days after the expected return date, if the borrower does not return the books, the borrower will be blacklisted and will no longer be allowed to borrow books.
- If the borrower damages or loses the book, they will have to compensate an amount equivalent to the price of the book.
- Borrowers must maintain a deposit of 100,000 VND to be able to read books. In cases where the deposit balance is 0, they must deduct 100,000 VND if they wish to continue reading.
- If readers no longer wish to continue reading, the deposit amount will be refunded.

3 Appication description

3.1 General view

Borrowers are individuals who interact with the system to access and manage library resources. They can perform the following functions:

1. **Search and view books:** Check the availability of books by title, genre, author, or publisher.
2. **Borrow books:**
    - Request to borrow up to 3 books at a time. They will also need to provide their deposit and borrowing status that meets the requirements.
    - Allowed to access information on loan periods and due dates for borrowed books.
3. **Return books:**
    - Return borrowed books and confirm their status as undamaged.
    - Pay fines for late returns, damages, or lost books, calculated automatically by the system.
4. **View profile:** Check personal details, borrowing history, and penalties (if any).

3.2 Functions

- **check_borrower_eligibility():** This function check borrower's eligibility including borrower's black_list status, check if borrower has sufficient deposit, check if borrower has less than 3 books currently borrowed.
- **set_loan_period():** This function set loan period and due date including Set borrow date to current date if not specified and Set due date to 3 months from borrow date.
- **process_book_return():** This function to handle book returns and calculate fees including calculate overdue days and handle damaged or lost books.
- **check_overdue_books():** This function check for overdue books and blacklist borrowers including Update blacklist status for borrowers with books overdue by 10 days or more.
- **initialize_borrower():** This function initialize new borrower accounts including initial the deposit amount to 100000 and black_list status to false.
- **search_books(p_title):** This function search books based on the title, genre, author_id and publisher_id.

3.3 Triggers

- **check_eligibility_before_loan:** Trigger to check eligibility before allowing new loans.
- **set_loan_period_trigger:** Trigger to set loan period automatically.
- **process_return_trigger:** Trigger to process returns.
- **initialize_borrower_trigger:** Trigger to initialize new borrower accounts.

4 Entities relationship diagram

4.1 Features

The system stores and manages several important features:

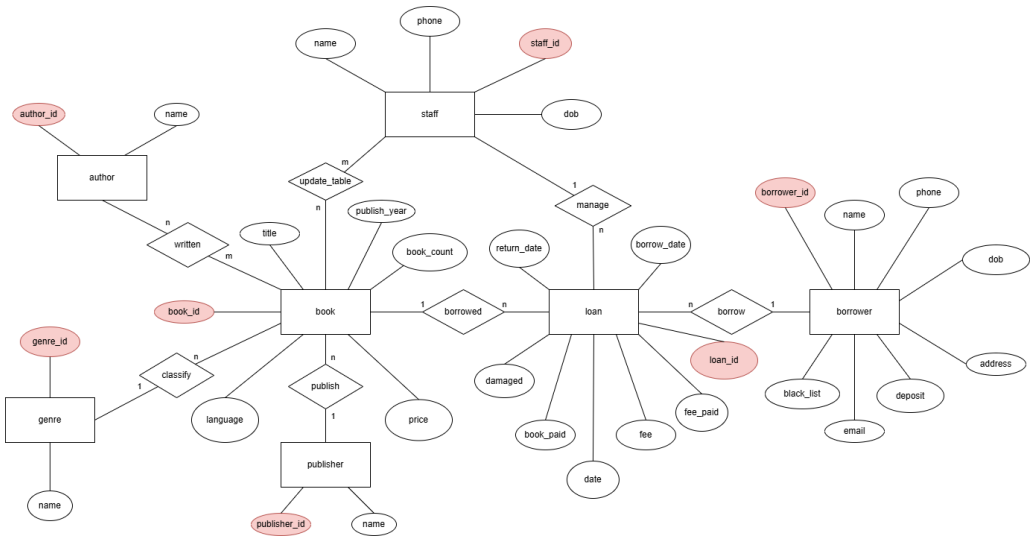
- **Book:** Information about each book, including book_id, title, publish_year, price and language.
- **Borrower:** Information about each borrower, including borrower_id, name, phone, date of birth, address, email, deposit and black_list status.
- **Staff:** Details about the staff members managing the system, including staff_id, name, phone, date of birth.
- **Loan:** Information about the loan_id, return_date, borrow_date, the damaged status of books, the return date of the borrower, the book paid status, the fee paid status and the fee.

- **Author:** Details about author_id and author's name.
- **Publisher:** Details about publisher_id and publisher's name.
- **Genre:** Details about genre_id and genre's name.

4.2 Relationship

- **written:** one book can be written by many authors and an author can write many books so this is n - m relationship.
- **classify:** many books can have the same genre so this is 1 - n relationship.
- **publish:** many books can be published by a publisher so this is 1 - n relationship.
- **update_table:** a staff can update many type of books and a book can be updated by many staffs so this is m - n relationship.
- **manage:** a staff can manage many loans so this is 1 - n relationship
- **borrowed:** One book can be associated with multiple borrow transactions so this is 1 - n relationship.
- **borrow:** one borrower can have multiple borrow transactions so this is 1 - n relationship.

4.3 Design



5 Relational schema

book(book_id, *publisher_id*, *genre_id*, title, language, publish_year, price)

written(*author_id*, *book_id*)

author(author_id, name)

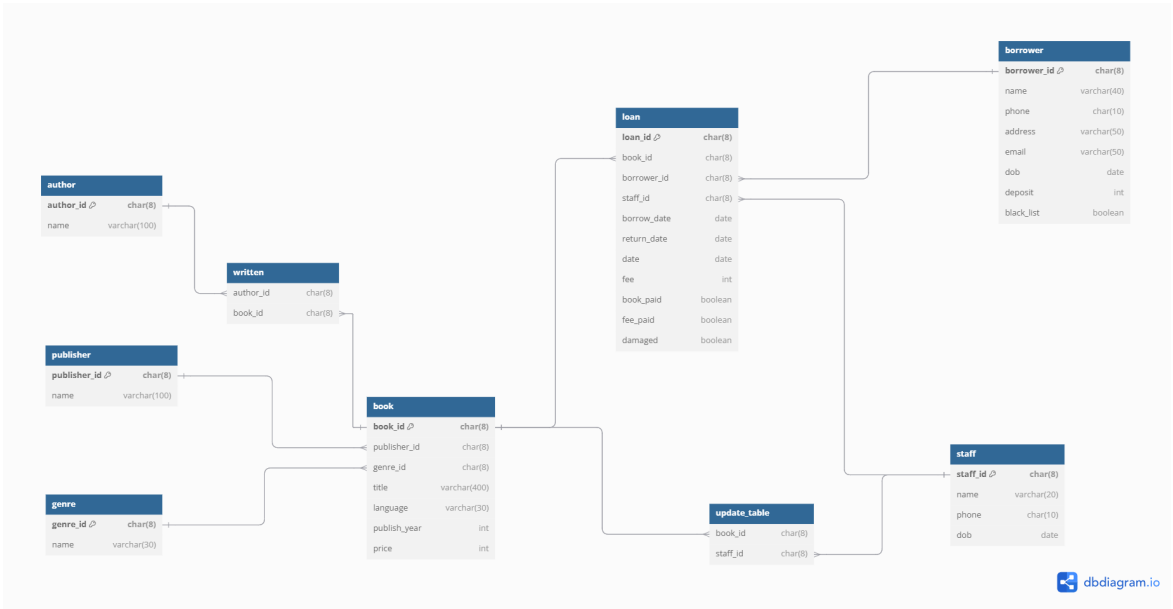
genre(genre_id, name)

loan(loan_id, *book_id*, *borrower_id*, *staff_id*, borrow_date, return_date, date, book_paid, fee_paid, damaged)

borrower(borrower_id, name, phone, address, email, dob, deposit, black_list)

staff(staff_id, name, phone, dob)

update_table(*book_id*, *staff_id*)



6 Queries

Query #1: Retrieve the list of books published by a publisher whose name is Ace

```
SELECT b.*
FROM book b
JOIN publisher p ON p.publisher_id = b.publisher_id
WHERE p.name = 'Ace';
```

	QUERY PLAN text
1	Hash Join (cost=45.31..345.85 rows=5 width=77) (actual time=10.974..20.665 rows=36 loops=1)
2	Hash Cond: (b.publisher_id = p.publisher_id)
3	-> Seq Scan on book b (cost=0.00..271.27 rows=11127 width=77) (actual time=0.928..12.037 rows=1112…
4	-> Hash (cost=45.30..45.30 rows=1 width=9) (actual time=3.386..3.387 rows=1 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Seq Scan on publisher p (cost=0.00..45.30 rows=1 width=9) (actual time=1.317..3.368 rows=1 loops…
7	Filter: ((name)::text = 'Ace'::text)
8	Rows Removed by Filter: 2263
9	Planning Time: 9.673 ms
10	Execution Time: 22.210 ms

	QUERY PLAN text
1	Hash Join (cost=8.31..308.84 rows=5 width=77) (actual time=1.547..4.462 rows=36 loops=1)
2	Hash Cond: (b.publisher_id = p.publisher_id)
3	-> Seq Scan on book b (cost=0.00..271.27 rows=11127 width=77) (actual time=0.033..1.538 rows=11127 loops=1)
4	-> Hash (cost=8.30..8.30 rows=1 width=9) (actual time=0.124..0.126 rows=1 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Index Scan using idx_1 on publisher p (cost=0.28..8.30 rows=1 width=9) (actual time=0.118..0.120 rows=1 loop…
7	Index Cond: ((name)::text = 'Ace'::text)
8	Planning Time: 4.170 ms
9	Execution Time: 5.313 ms

```
CREATE INDEX idx_1 ON publisher(name);
```

We can see that after using index in this query, the execution time decreases, the initial cost and the end cost of all task also decreases.

Query #2: Retrieve the list of borrowers who borrowed at least one book in 2021

```
SELECT DISTINCT b.*
FROM borrower b
JOIN loan l ON l.borrower_id = b.borrower_id
WHERE EXTRACT(YEAR FROM l.borrow_date) = 2021;
```

	QUERY PLAN text
1	Unique (cost=644.27..646.79 rows=112 width=83) (actual time=53.481..63.040 rows=1188 loops=1)
2	-> Sort (cost=644.27..644.55 rows=112 width=83) (actual time=53.479..54.373 rows=7493 loops=1)
3	Sort Key: b.borrower_id, b.name, b.phone, b.address, b.email, b.dob, b.deposit, b.black_list
4	Sort Method: quicksort Memory: 1009kB
5	-> Hash Join (cost=74.00..640.45 rows=112 width=83) (actual time=3.760..20.171 rows=7493 loops=1)
6	Hash Cond: (l.borrower_id = b.borrower_id)
7	-> Seq Scan on loan l (cost=0.00..566.16 rows=112 width=9) (actual time=0.479..13.193 rows=7493 loops=1)
8	Filter: (EXTRACT(year FROM borrow_date) = '2021'::numeric)
9	Rows Removed by Filter: 14851
10	-> Hash (cost=49.00..49.00 rows=2000 width=83) (actual time=3.170..3.171 rows=2000 loops=1)
11	Buckets: 2048 Batches: 1 Memory Usage: 247kB
12	-> Seq Scan on borrower b (cost=0.00..49.00 rows=2000 width=83) (actual time=1.154..2.637 rows=2000 loop...
13	Planning Time: 18.436 ms
14	Execution Time: 64.505 ms

	QUERY PLAN text
1	Unique (cost=644.27..646.79 rows=112 width=83) (actual time=52.866..59.926 rows=1188 loops=1)
2	-> Sort (cost=644.27..644.55 rows=112 width=83) (actual time=52.864..53.490 rows=7493 loops=1)
3	Sort Key: b.borrower_id, b.name, b.phone, b.address, b.email, b.dob, b.deposit, b.black_list
4	Sort Method: quicksort Memory: 1009kB
5	-> Hash Join (cost=74.00..640.45 rows=112 width=83) (actual time=1.487..16.164 rows=7493 loops=1)
6	Hash Cond: (l.borrower_id = b.borrower_id)
7	-> Seq Scan on loan l (cost=0.00..566.16 rows=112 width=9) (actual time=0.050..10.166 rows=7493 loops=1)
8	Filter: (EXTRACT(year FROM borrow_date) = '2021'::numeric)
9	Rows Removed by Filter: 14851
10	-> Hash (cost=49.00..49.00 rows=2000 width=83) (actual time=1.407..1.409 rows=2000 loops=1)
11	Buckets: 2048 Batches: 1 Memory Usage: 247kB
12	-> Seq Scan on borrower b (cost=0.00..49.00 rows=2000 width=83) (actual time=0.023..0.361 rows=2000 loop...
13	Planning Time: 4.375 ms
14	Execution Time: 60.350 ms

Total rows: 14 of 14 Query complete 00:00:00.140 Ln 1, Col 1

CREATE INDEX idx_2 ON loan(borrower_id);

We can see that after using index, the execution time decreases but not much. The initial cost and the end cost of all tasks reduces slightly. Also, the query plan shows only seq scan without any changing so in this query using index does not useful.

Query #3: List of books have been borrowed by the current date

```
SELECT b.*
FROM book b
JOIN loan l ON b.book_id = l.book_id
WHERE l.borrow_date = CURRENT_DATE;
```

	QUERY PLAN text
1	Nested Loop (cost=0.29..707.91 rows=19 width=77) (actual time=4.913..4.914 rows=0 loops=1)
2	-> Seq Scan on loan l (cost=0.00..566.16 rows=19 width=9) (actual time=4.911..4.912 rows=0 loops...
3	Filter: (borrow_date = CURRENT_DATE)
4	Rows Removed by Filter: 22344
5	-> Index Scan using pk_book on book b (cost=0.29..7.46 rows=1 width=77) (never executed)
6	Index Cond: (book_id = l.book_id)
7	Planning Time: 6.827 ms
8	Execution Time: 4.964 ms

	QUERY PLAN text
1	Nested Loop (cost=4.72..206.12 rows=19 width=77) (actual time=0.541..0.543 rows=0 loops=1)
2	-> Bitmap Heap Scan on loan l (cost=4.44..64.37 rows=19 width=9) (actual time=0.541..0.541 rows=0 loops=1)
3	Recheck Cond: (borrow_date = CURRENT_DATE)
4	-> Bitmap Index Scan on idx_3 (cost=0.00..4.43 rows=19 width=0) (actual time=0.537..0.538 rows=0 loops...
5	Index Cond: (borrow_date = CURRENT_DATE)
6	-> Index Scan using pk_book on book b (cost=0.29..7.46 rows=1 width=77) (never executed)
7	Index Cond: (book_id = l.book_id)
8	Planning Time: 3.920 ms
9	Execution Time: 1.238 ms

CREATE INDEX idx_3 ON loan(borrow_date);

We can see that after using index the execution time decreases. The initial cost and the end cost also reduces significantly. So using index in this query is an optimization solution.

Query #4: Retrieve the list of books that have amount greater than 3

```
SELECT title, COUNT(*) AS number_of_book
FROM book
GROUP BY title
HAVING COUNT(*) > 3;
```

	QUERY PLAN text
1	HashAggregate (cost=326.90..456.30 rows=3451 width=45) (actual time=13.230..17.053 rows=70 loops=1)
2	Group Key: title
3	Filter: (count(*) > 3)
4	Batches: 1 Memory Usage: 1681kB
5	Rows Removed by Filter: 10282
6	-> Seq Scan on book (cost=0.00..271.27 rows=11127 width=37) (actual time=0.061..2.174 rows=11127 loops...
7	Planning Time: 0.434 ms
8	Execution Time: 18.416 ms

	QUERY PLAN text
1	HashAggregate (cost=326.90..456.30 rows=3451 width=45) (actual time=8.925..11.965 rows=70 loops=1)
2	Group Key: title
3	Filter: (count(*) > 3)
4	Batches: 1 Memory Usage: 1681kB
5	Rows Removed by Filter: 10282
6	-> Seq Scan on book (cost=0.00..271.27 rows=11127 width=37) (actual time=0.042..1.356 rows=11127 loops...
7	Planning Time: 2.671 ms
8	Execution Time: 12.715 ms

CREATE INDEX idx_4 on book(title);

We can see that after using index the initial cost and the end cost still remain. The execution time decreases slightly. So using index in this query is not useful.

Query #5: Retrieve the list of books and their genre

```
SELECT b.*, g.name
```

FROM book b
JOIN genre g ON g.genre_id = b.genre_id;

	QUERY PLAN text
1	Hash Join (cost=1.36..309.57 rows=11127 width=155) (actual time=0.658..6.713 rows=11127 loops=1)
2	Hash Cond: (b.genre_id = g.genre_id)
3	-> Seq Scan on book b (cost=0.00..271.27 rows=11127 width=77) (actual time=0.040..1.143 rows=11127 l...
4	-> Hash (cost=1.16..1.16 rows=16 width=114) (actual time=0.586..0.587 rows=16 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Seq Scan on genre g (cost=0.00..1.16 rows=16 width=114) (actual time=0.548..0.552 rows=16 loops...
7	Planning Time: 7.159 ms
8	Execution Time: 7.162 ms

	QUERY PLAN text
1	Hash Join (cost=1.36..309.57 rows=11127 width=155) (actual time=0.142..10.894 rows=11127 loops=1)
2	Hash Cond: (b.genre_id = g.genre_id)
3	-> Seq Scan on book b (cost=0.00..271.27 rows=11127 width=77) (actual time=0.035..1.820 rows=11127 l...
4	-> Hash (cost=1.16..1.16 rows=16 width=114) (actual time=0.072..0.074 rows=16 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Seq Scan on genre g (cost=0.00..1.16 rows=16 width=114) (actual time=0.024..0.034 rows=16 loops...
7	Planning Time: 2.956 ms
8	Execution Time: 11.744 ms

CREATE INDEX idx_5 on book(genre_id);
We can see that after using index, the execution time decreases slightly but the cost still remains. Also the query plan still shows seq scan. So using index in this query is not a good choice.
Query #6: List of borrowers who have borrowed more than 5 books in June
SELECT br.*
FROM borrower br
JOIN loan l ON l.borrower_id = br.borrower_id
JOIN book bk ON bk.book_id = l.book_id
WHERE EXTRACT(MONTH FROM l.borrow_date) = 6
GROUP BY br.borrower_id
HAVING COUNT(l.book_id) > 5;

	QUERY PLAN text
4	Rows Removed by Filter: 420
5	-> Sort (cost=878.15..878.43 rows=112 width=92) (actual time=46.952..47.060 rows=1664 loops=1)
6	Sort Key: br.borrower_id
7	Sort Method: quicksort Memory: 262kB
8	-> Nested Loop (cost=74.28..874.33 rows=112 width=92) (actual time=2.757..40.244 rows=1664 loops=1)
9	-> Hash Join (cost=74.00..640.45 rows=112 width=92) (actual time=2.179..12.302 rows=1664 loops=1)
10	Hash Cond: (l.borrower_id = br.borrower_id)
11	-> Seq Scan on loan l (cost=0.00..566.16 rows=112 width=18) (actual time=0.113..9.004 rows=1664 loops=1)
12	Filter: (EXTRACT(month FROM borrow_date) = '6'::numeric)
13	Rows Removed by Filter: 20680
14	-> Hash (cost=49.00..49.00 rows=2000 width=83) (actual time=1.928..1.929 rows=2000 loops=1)
15	Buckets: 2048 Batches: 1 Memory Usage: 247kB
16	-> Seq Scan on borrower br (cost=0.00..49.00 rows=2000 width=83) (actual time=0.033..0.562 rows=2000 loops=1)
17	-> Index Only Scan using pk_book on book bk (cost=0.29..2.09 rows=1 width=9) (actual time=0.016..0.016 rows=1 loops=1...
18	Index Cond: (book_id = l.book_id)
19	Heap Fetches: 0
20	Planning Time: 6.060 ms
21	Execution Time: 48.358 ms

	QUERY PLAN text
4	Rows Removed by Filter: 420
5	-> Sort (cost=878.15..878.43 rows=112 width=92) (actual time=30.029..30.150 rows=1664 loops=1)
6	Sort Key: br.borrower_id
7	Sort Method: quicksort Memory: 262kB
8	-> Nested Loop (cost=74.28..874.33 rows=112 width=92) (actual time=0.839..23.300 rows=1664 loops=1)
9	-> Hash Join (cost=74.00..640.45 rows=112 width=92) (actual time=0.801..9.789 rows=1664 loops=1)
10	Hash Cond: (l.borrower_id = br.borrower_id)
11	-> Seq Scan on loan l (cost=0.00..566.16 rows=112 width=18) (actual time=0.056..8.047 rows=1664 loops=1)
12	Filter: (EXTRACT(month FROM borrow_date) = '6'::numeric)
13	Rows Removed by Filter: 20680
14	-> Hash (cost=49.00..49.00 rows=2000 width=83) (actual time=0.722..0.723 rows=2000 loops=1)
15	Buckets: 2048 Batches: 1 Memory Usage: 247kB
16	-> Seq Scan on borrower br (cost=0.00..49.00 rows=2000 width=83) (actual time=0.012..0.190 rows=2000 loops=1)
17	-> Index Only Scan using pk_book on book bk (cost=0.29..2.09 rows=1 width=9) (actual time=0.008..0.008 rows=1 loops=1...
18	Index Cond: (book_id = l.book_id)
19	Heap Fetches: 0
20	Planning Time: 6.587 ms
21	Execution Time: 31.363 ms

CREATE INDEX idx_6 on loan(book_id);

We can see that after using index, the query plan still shows seq scan. The execution time decreases slightly but the initial cost and the end cost still remains. So using index in this query is not a good choice.

Query #7: Determine the number of books borrowed by borrower with ID "1576"

SELECT COUNT(l.book_id)
FROM loan l
WHERE borrower_id = '1576';

	QUERY PLAN text
1	Aggregate (cost=510.38..510.39 rows=1 width=8) (actual time=13.243..13.244 rows=1 loops=1)
2	-> Seq Scan on loan l (cost=0.00..510.30 rows=32 width=9) (actual time=0.089..13.214 rows=32 loops...
3	Filter: (borrower_id = '1576'::bpchar)
4	Rows Removed by Filter: 22312
5	Planning Time: 2.449 ms
6	Execution Time: 13.331 ms

	QUERY PLAN text
1	Aggregate (cost=92.58..92.59 rows=1 width=8) (actual time=0.221..0.223 rows=1 loops=1)
2	-> Bitmap Heap Scan on loan l (cost=4.54..92.50 rows=32 width=9) (actual time=0.178..0.204 rows=32 loops=1)
3	Recheck Cond: (borrower_id = '1576'::bpchar)
4	Heap Blocks: exact=9
5	-> Bitmap Index Scan on idx_7 (cost=0.00..4.53 rows=32 width=0) (actual time=0.155..0.155 rows=32 loops...
6	Index Cond: (borrower_id = '1576'::bpchar)
7	Planning Time: 3.749 ms
8	Execution Time: 0.322 ms

CREATE INDEX idx_7 on loan(borrower_id);

We can see that after using index the execution time decreases. The initial cost and the end cost also reduces significantly. So using index in this query is an optimization solution.

Query #8: List the borrowers who have borrowed both book titled "Ukridge" and "The Log from the Sea of Cortez"

```
SELECT br.*
FROM borrower br
JOIN loan l ON l.borrower_id = br.borrower_id
JOIN book bk ON bk.book_id = l.book_id
WHERE bk.title = 'The Wise Woman'
INTERSECT
SELECT br.*
FROM borrower br
JOIN loan l ON l.borrower_id = br.borrower_id
JOIN book bk ON bk.book_id = l.book_id
WHERE bk.title = 'Spares';
```

	QUERY PLAN text
1	HashSetOp Intersect (cost=299.38..1625.78 rows=2 width=427) (actual time=19.470..19.481 rows=0 loops=1)
2	-> Append (cost=299.38..1625.70 rows=4 width=427) (actual time=5.587..19.465 rows=3 loops=1)
3	-> Subquery Scan on "*SELECT* 1" (cost=299.38..812.84 rows=2 width=87) (actual time=5.586..8.802 rows=1 loops=1)
4	-> Nested Loop (cost=299.38..812.82 rows=2 width=83) (actual time=5.584..8.799 rows=1 loops=1)
5	-> Hash Join (cost=299.10..812.22 rows=2 width=9) (actual time=5.047..8.260 rows=1 loops=1)
6	Hash Cond: (l.book_id = bk.book_id)
7	-> Seq Scan on loan l (cost=0.00..454.44 rows=22344 width=18) (actual time=0.034..2.228 rows=22344 loops=1)
8	-> Hash (cost=299.09..299.09 rows=1 width=9) (actual time=2.007..2.010 rows=1 loops=1)
9	Buckets: 1024 Batches: 1 Memory Usage: 9kB
10	-> Seq Scan on book bk (cost=0.00..299.09 rows=1 width=9) (actual time=0.025..1.999 rows=1 loops=1)
11	Filter: ((title)::text = 'The Wise Woman'::text)
12	Rows Removed by Filter: 11126
13	-> Index Scan using pk_borrower on borrower br (cost=0.28..0.30 rows=1 width=83) (actual time=0.529..0.530 rows=1 loops=...
14	Index Cond: (borrower_id = l.borrower_id)
15	-> Subquery Scan on "*SELECT* 2" (cost=299.38..812.84 rows=2 width=87) (actual time=5.563..10.656 rows=2 loops=1)
16	-> Nested Loop (cost=299.38..812.82 rows=2 width=83) (actual time=5.562..10.651 rows=2 loops=1)
17	-> Hash Join (cost=299.10..812.22 rows=2 width=9) (actual time=5.053..9.907 rows=2 loops=1)
18	Hash Cond: (l_1.book_id = bk_1.book_id)

	QUERY PLAN text
1	HashSetOp Intersect (cost=4.58..623.10 rows=2 width=427) (actual time=4.655..4.660 rows=0 loops=1)
2	-> Append (cost=4.58..623.02 rows=4 width=427) (actual time=0.139..4.650 rows=3 loops=1)
3	-> Subquery Scan on "*SELECT* 1" (cost=4.58..311.50 rows=2 width=87) (actual time=0.138..2.333 rows=1 loops=1)
4	-> Nested Loop (cost=4.58..311.48 rows=2 width=83) (actual time=0.137..2.331 rows=1 loops=1)
5	-> Nested Loop (cost=4.30..310.88 rows=2 width=9) (actual time=0.108..2.301 rows=1 loops=1)
6	-> Seq Scan on book bk (cost=0.00..299.09 rows=1 width=9) (actual time=0.025..2.216 rows=1 loops=1)
7	Filter: ((title)::text = 'The Wise Woman'::text)
8	Rows Removed by Filter: 11126
9	-> Bitmap Heap Scan on loan l (cost=4.30..11.77 rows=2 width=18) (actual time=0.069..0.071 rows=1 loops=1)
10	Recheck Cond: (book_id = bk.book_id)
11	Heap Blocks: exact=1
12	-> Bitmap Index Scan on idx_8 (cost=0.00..4.30 rows=2 width=0) (actual time=0.063..0.063 rows=1 loops=1)
13	Index Cond: (book_id = bk.book_id)
14	-> Index Scan using pk_borrower on borrower br (cost=0.28..0.30 rows=1 width=83) (actual time=0.026..0.026 rows=1 loops=...
15	Index Cond: (borrower_id = l.borrower_id)
16	-> Subquery Scan on "*SELECT* 2" (cost=4.58..311.50 rows=2 width=87) (actual time=0.100..2.313 rows=2 loops=1)
17	-> Nested Loop (cost=4.58..311.48 rows=2 width=83) (actual time=0.099..2.311 rows=2 loops=1)
18	-> Nested Loop (cost=4.30..310.88 rows=2 width=9) (actual time=0.076..2.273 rows=2 loops=1)

CREATE INDEX idx_8 on loan(book_id);

We can see that after using index the execution time decreases. The initial cost and the end cost also reduces significantly. So using index in this query is an optimization solution.

Query #9: Retrieve the list of authors and their title books

```
SELECT a.*, b.title
FROM author a
JOIN written w ON a.author_id = w.author_id
JOIN book b ON w.book_id = b.book_id;
```

	QUERY PLAN text
1	Hash Join (cost=778.14..1160.23 rows=17642 width=147) (actual time=21.152..41.878 rows=17642 loops=1)
2	Hash Cond: (w.book_id = b.book_id)
3	-> Hash Join (cost=367.79..703.54 rows=17642 width=119) (actual time=13.514..25.696 rows=17642 loops=1)
4	Hash Cond: (w.author_id = a.author_id)
5	-> Seq Scan on written w (cost=0.00..289.42 rows=17642 width=18) (actual time=0.768..5.205 rows=17642 loops...
6	-> Hash (cost=252.35..252.35 rows=9235 width=110) (actual time=12.572..12.573 rows=9235 loops=1)
7	Buckets: 16384 Batches: 1 Memory Usage: 1410kB
8	-> Seq Scan on author a (cost=0.00..252.35 rows=9235 width=110) (actual time=0.933..8.090 rows=9235 loop...
9	-> Hash (cost=271.27..271.27 rows=11127 width=46) (actual time=7.545..7.545 rows=11127 loops=1)
10	Buckets: 16384 Batches: 1 Memory Usage: 977kB
11	-> Seq Scan on book b (cost=0.00..271.27 rows=11127 width=46) (actual time=0.032..2.811 rows=11127 loops=1)
12	Planning Time: 11.883 ms
13	Execution Time: 42.922 ms

	QUERY PLAN text
1	Hash Join (cost=778.14..1160.23 rows=17642 width=147) (actual time=14.958..44.571 rows=17642 loops=1)
2	Hash Cond: (w.book_id = b.book_id)
3	-> Hash Join (cost=367.79..703.54 rows=17642 width=119) (actual time=6.556..22.794 rows=17642 loops=1)
4	Hash Cond: (w.author_id = a.author_id)
5	-> Seq Scan on written w (cost=0.00..289.42 rows=17642 width=18) (actual time=0.041..2.560 rows=17642 loops...
6	-> Hash (cost=252.35..252.35 rows=9235 width=110) (actual time=6.370..6.371 rows=9235 loops=1)
7	Buckets: 16384 Batches: 1 Memory Usage: 1410kB
8	-> Seq Scan on author a (cost=0.00..252.35 rows=9235 width=110) (actual time=0.023..1.655 rows=9235 loop...
9	-> Hash (cost=271.27..271.27 rows=11127 width=46) (actual time=8.277..8.277 rows=11127 loops=1)
10	Buckets: 16384 Batches: 1 Memory Usage: 977kB
11	-> Seq Scan on book b (cost=0.00..271.27 rows=11127 width=46) (actual time=0.040..3.226 rows=11127 loops=1)
12	Planning Time: 5.716 ms
13	Execution Time: 46.185 ms

CREATE INDEX idx_9 on written(author_id);

We can see that after using index, the query plan still shows seq scan. The execution time decreases slightly but the initial cost and the end cost still remains. So using index in this query is not a good choice.

Query #10: List of the most borrowed book genre

```
SELECT g.name, COUNT(g.genre_id) AS count
FROM genre g
JOIN book b ON g.genre_id = b.genre_id
JOIN loan l ON l.book_id = b.book_id
GROUP BY g.genre_id
HAVING COUNT(g.genre_id) ≥ ALL
(
SELECT COUNT(g.genre_id) AS count_book
FROM genre g
JOIN book b ON g.genre_id = b.genre_id
JOIN loan l ON l.book_id = b.book_id
GROUP BY g.genre_id
);
```

	QUERY PLAN text
1	HashAggregate (cost=2221.47..2223.91 rows=8 width=122) (actual time=143.153..143.175 rows=1 loops=1)
2	Group Key: g.genre_id
3	Filter: (SubPlan 1)
4	Batches: 1 Memory Usage: 24kB
5	Rows Removed by Filter: 15
6	-> Hash Join (cost=411.72..999.02 rows=22344 width=114) (actual time=9.396..53.652 rows=22344 loops=1)
7	Hash Cond: (b.genre_id = g.genre_id)
8	-> Hash Join (cost=410.36..923.47 rows=22344 width=9) (actual time=9.321..36.392 rows=22344 loops=1)
9	Hash Cond: (l.book_id = b.book_id)
10	-> Seq Scan on loan l (cost=0.00..454.44 rows=22344 width=9) (actual time=0.018..4.312 rows=22344 loops=1)
11	-> Hash (cost=271.27..271.27 rows=11127 width=18) (actual time=9.160..9.161 rows=11127 loops=1)
12	Buckets: 16384 Batches: 1 Memory Usage: 672kB
13	-> Seq Scan on book b (cost=0.00..271.27 rows=11127 width=18) (actual time=0.018..3.570 rows=11127 loops=1)
14	-> Hash (cost=1.16..1.16 rows=16 width=114) (actual time=0.056..0.057 rows=16 loops=1)
15	Buckets: 1024 Batches: 1 Memory Usage: 9kB
16	-> Seq Scan on genre g (cost=0.00..1.16 rows=16 width=114) (actual time=0.033..0.038 rows=16 loops=1)
17	SubPlan 1
18	-> Materialize (cost=1110.74..1110.98 rows=16 width=44) (actual time=4.596..4.615 rows=4 loops=16)
19	-> HashAggregate (cost=1110.74..1110.90 rows=16 width=44) (actual time=73.512..73.529 rows=16 loops=1)
20	Group Key: g_1.genre_id
21	Batches: 1 Memory Usage: 24kB
22	-> Hash Join (cost=411.72..999.02 rows=22344 width=36) (actual time=10.456..57.163 rows=22344 loops=1)
23	Hash Cond: (b_1.genre_id = g_1.genre_id)
24	-> Hash Join (cost=410.36..923.47 rows=22344 width=9) (actual time=10.362..41.167 rows=22344 loops=1)
25	Hash Cond: (l_1.book_id = b_1.book_id)
26	-> Seq Scan on loan l_1 (cost=0.00..454.44 rows=22344 width=9) (actual time=0.022..5.166 rows=22344 loops=1)
27	-> Hash (cost=271.27..271.27 rows=11127 width=18) (actual time=10.216..10.217 rows=11127 loops=1)
28	Buckets: 16384 Batches: 1 Memory Usage: 672kB
29	-> Seq Scan on book b_1 (cost=0.00..271.27 rows=11127 width=18) (actual time=0.023..3.879 rows=11127 loop…
30	-> Hash (cost=1.16..1.16 rows=16 width=36) (actual time=0.057..0.058 rows=16 loops=1)
31	Buckets: 1024 Batches: 1 Memory Usage: 9kB
32	-> Seq Scan on genre g_1 (cost=0.00..1.16 rows=16 width=36) (actual time=0.037..0.041 rows=16 loops=1)
33	Planning Time: 3.000 ms
34	Execution Time: 144.115 ms

	QUERY PLAN text
1	HashAggregate (cost=2221.47..2223.91 rows=8 width=122) (actual time=74.100..74.121 rows=1 loops=1)
2	Group Key: g.genre_id
3	Filter: (SubPlan 1)
4	Batches: 1 Memory Usage: 24kB
5	Rows Removed by Filter: 15
6	-> Hash Join (cost=411.72..999.02 rows=22344 width=114) (actual time=5.884..31.914 rows=22344 loops=1)
7	Hash Cond: (b.genre_id = g.genre_id)
8	-> Hash Join (cost=410.36..923.47 rows=22344 width=9) (actual time=5.816..22.171 rows=22344 loops=1)
9	Hash Cond: (l.book_id = b.book_id)
10	-> Seq Scan on loan l (cost=0.00..454.44 rows=22344 width=9) (actual time=0.015..3.394 rows=22344 loops=1)
11	-> Hash (cost=271.27..271.27 rows=11127 width=18) (actual time=5.690..5.691 rows=11127 loops=1)
12	Buckets: 16384 Batches: 1 Memory Usage: 672kB
13	-> Seq Scan on book b (cost=0.00..271.27 rows=11127 width=18) (actual time=0.015..2.153 rows=11127 loops=1)
14	-> Hash (cost=1.16..1.16 rows=16 width=114) (actual time=0.054..0.055 rows=16 loops=1)
15	Buckets: 1024 Batches: 1 Memory Usage: 9kB
16	-> Seq Scan on genre g (cost=0.00..1.16 rows=16 width=114) (actual time=0.035..0.039 rows=16 loops=1)
17	SubPlan 1
18	-> Materialize (cost=1110.74..1110.98 rows=16 width=44) (actual time=2.096..2.097 rows=4 loops=16)
19	-> HashAggregate (cost=1110.74..1110.90 rows=16 width=44) (actual time=33.502..33.511 rows=16 loops=1)
20	Group Key: g_1.genre_id
21	Batches: 1 Memory Usage: 24kB
22	-> Hash Join (cost=411.72..999.02 rows=22344 width=36) (actual time=4.147..25.655 rows=22344 loops=1)
23	Hash Cond: (b_1.genre_id = g_1.genre_id)
24	-> Hash Join (cost=410.36..923.47 rows=22344 width=9) (actual time=4.094..17.524 rows=22344 loops=1)
25	Hash Cond: (l_1.book_id = b_1.book_id)
26	-> Seq Scan on loan l_1 (cost=0.00..454.44 rows=22344 width=9) (actual time=0.009..2.282 rows=22344 loops=1)
27	-> Hash (cost=271.27..271.27 rows=11127 width=18) (actual time=4.009..4.010 rows=11127 loops=1)
28	Buckets: 16384 Batches: 1 Memory Usage: 672kB
29	-> Seq Scan on book b_1 (cost=0.00..271.27 rows=11127 width=18) (actual time=0.010..1.576 rows=11127 loop...
30	-> Hash (cost=1.16..1.16 rows=16 width=36) (actual time=0.036..0.036 rows=16 loops=1)
31	Buckets: 1024 Batches: 1 Memory Usage: 9kB
32	-> Seq Scan on genre g_1 (cost=0.00..1.16 rows=16 width=36) (actual time=0.026..0.028 rows=16 loops=1)
33	Planning Time: 7.070 ms
34	Execution Time: 74.569 ms

CREATE INDEX idx_10 on book(genre_id);
This query use ALL so index is not probably used in this query. If I use index, the execution time reduces but the initial cost and the end cost still remains.

Query #11: List books borrowed more than once

SELECT book_id, COUNT(*) AS borrow_count
FROM loan
GROUP BY book_id
HAVING COUNT(*) > 1;

```
1  EXPLAIN ANALYZE
2  SELECT book_id, COUNT(*) AS borrow_count
3  FROM loan
4  GROUP BY book_id
5  HAVING COUNT(*) > 1;
```

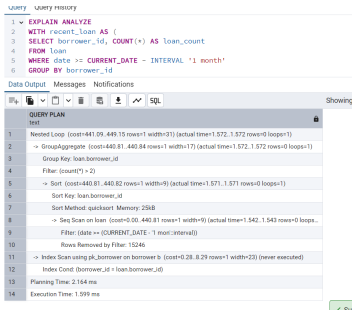
Data Output Messages Notifications

	QUERY PLAN text
1	HashAggregate (cost=402.69..506.24 rows=2761 width=17) (actual time=3.001..3.555 rows=4450 loops=1)
2	Group Key: book_id
3	Filter: (count(*) > 1)
4	Batches: 1 Memory Usage: 1169kB
5	Rows Removed by Filter: 3834
6	-> Seq Scan on loan (cost=0.00..326.46 rows=15246 width=9) (actual time=0.010..0.660 rows=15246 loops...
7	Planning Time: 1.128 ms
8	Execution Time: 3.827 ms

Query #12: List borrowers who borrowed more than 2 books in the last month

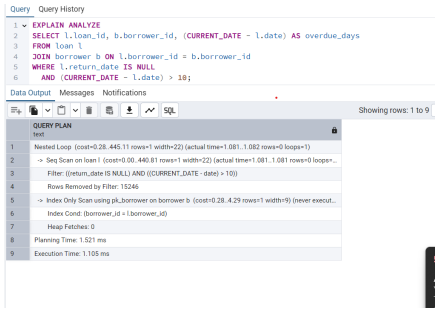
WITH recent_loan AS (
SELECT borrower_id, COUNT(*) AS loan_count
FROM loan
WHERE $date \geq CURRENT\_DATE$ - INTERVAL '1 month'
GROUP BY borrower_id

)
SELECT b.borrower_id, b.name, r.loan_count
FROM borrower b
JOIN recent_loan r ON b.borrower_id = r.borrower_id
WHERE r.loan_count > 2;



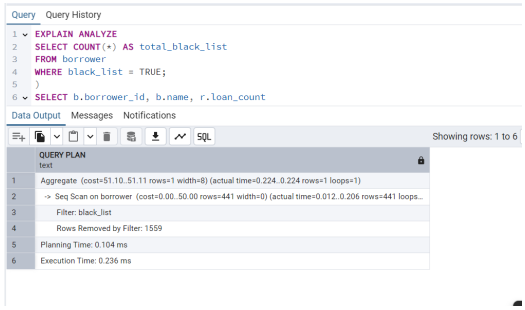
Query #13: List overdue loans greater than 10 days

SELECT l.loan_id, b.borrower_id, (CURRENT_DATE - l.date) AS overdue_days
FROM loan l
JOIN borrower b ON l.borrower_id = b.borrower_id
WHERE l.return_date IS NULL
AND (CURRENT_DATE - l.date) > 10;



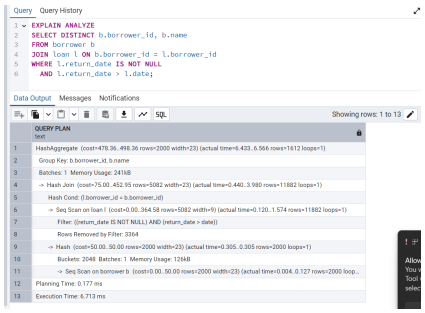
Query #14: Count of blacklisted borrowers

SELECT COUNT(*) AS total_black_list
FROM borrower
WHERE black_list = TRUE;



Query #15: Borrowers who have never returned a book on time

```
SELECT DISTINCT b.borrower_id, b.name
FROM borrower b
JOIN loan l ON b.borrower_id = l.borrower_id
WHERE l.return_date IS NOT NULL
AND l.return_date > l.date;
```



Query #16: Average book price by languages

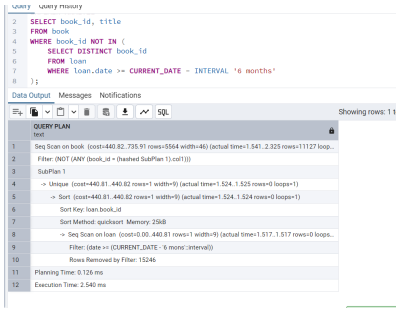
```
SELECT languages, AVG(price) AS avg_price
FROM book
GROUP BY languages
HAVING AVG(price) > 0;
```

Query #17: List of books not borrowed in the last 6 months

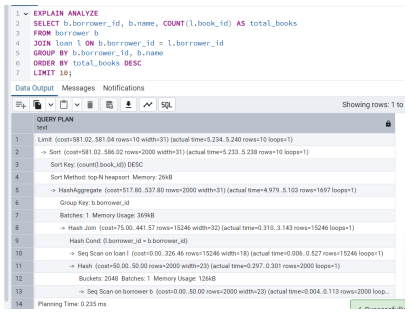
```
SELECT book_id, title
FROM book
WHERE book_id NOT IN (
SELECT DISTINCT book_id
FROM loan
WHERE loan.date ≥ CURRENT_DATE - INTERVAL '6 months'
);
```

Query #18: Top 10 borrowers who borrowed the most books

```
SELECT b.borrower_id, b.name, COUNT(l.book_id) AS total_books
FROM borrower b
JOIN loan l ON b.borrower_id = l.borrower_id
GROUP BY b.borrower_id, b.name
ORDER BY total_books DESC
```

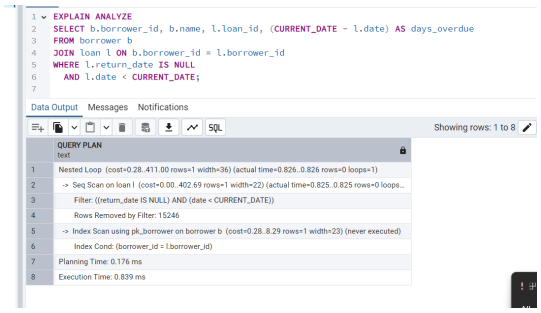


LIMIT 10;



Query #19: List borrowers who have overdue loans

SELECT b.borrower_id, b.name, l.loan_id, (CURRENT_DATE - l.date) AS days_overdue
FROM borrower b
JOIN loan l ON b.borrower_id = l.borrower_id
WHERE l.return_date IS NULL
AND l.date < CURRENT_DATE;



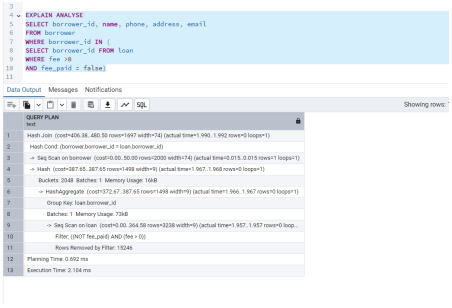
Query #20: List borrowers with total borrowed books and total deposit

SELECT b.borrower_id, b.name, b.deposit,
(SELECT COUNT(*) FROM loan l WHERE l.borrower_id b.borrower_id) AS total_loan
FROM borrower b;

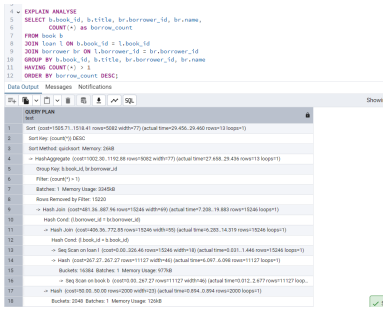
Query #21: Contact list of members that have not pay their fees

SELECT borrower_id, name, phone, address, email

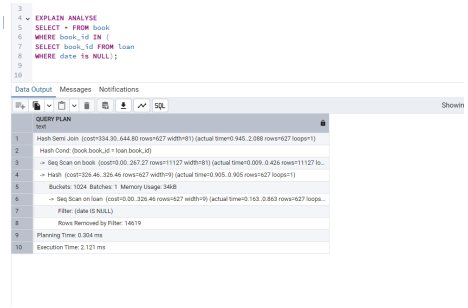
FROM borrower
WHERE borrower_id IN (
SELECT borrower_id FROM loan
WHERE fee >0
AND fee_paid = false);



Query #22: List of borrowers who repeat borrow for same books SELECT b.book_id, b.title, br.borrower_id, br.name, COUNT(*) as borrow_count FROM book b JOIN loan l ON b.book_id = l.book_id JOIN borrower br ON l.borrower_id = br.borrower_id GROUP BY b.book_id, b.title, br.borrower_id, br.name HAVING COUNT(*) ¿ 1 ORDER BY borrow_count DESC;

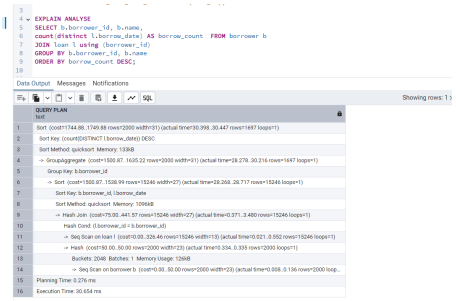


Query #23: List all books cannot be borrowed
SELECT * FROM book
WHERE book_id IN (
SELECT book_id FROM loan
WHERE date is NULL);



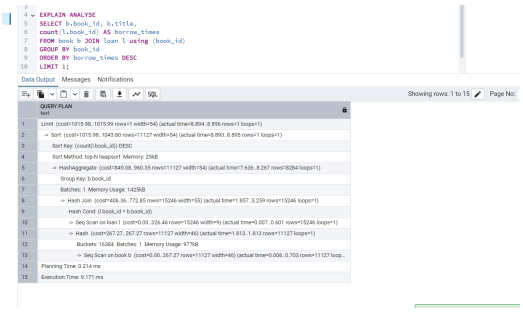
Query #24: Frequency of borrowing books of members

```
SELECT b.borrower_id, b.name,
count(distinct l.borrow_date) AS borrow_count FROM borrower b
JOIN loan l using (borrower_id)
GROUP BY b.borrower_id, b.name
ORDER BY borrow_count DESC;
```



Query #25: Most borrowed book

```
SELECT b.book_id, b.title,
count(l.book_id) AS borrow_times
FROM book b JOIN loan l using (book_id)
GROUP BY book_id
ORDER BY borrow_times DESC
LIMIT 1;
```



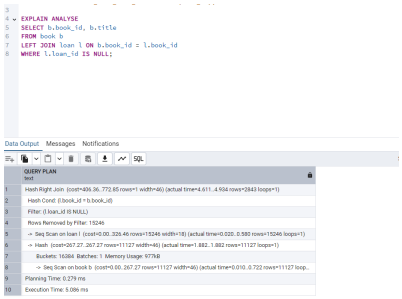
Query #26: List of staff whose age is over 50

```
WITH tmp AS (
SELECT staff_id, name,
```

EXTRACT(YEAR FROM AGE(CURRENT_DATE, dob)) AS age
FROM staff)
SELECT * FROM tmp
WHERE age ¿ 50;

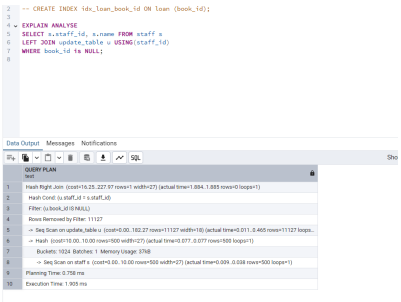
Query #27: List staffs who nevers update book

SELECT s.staff_id, s.name FROM staff s
LEFT JOIN update_table u USING(staff_id)
WHERE book_id is NULL;



Query #28: List numbers of book in each language

SELECT languages, count(*) FROM book
GROUP BY languages;

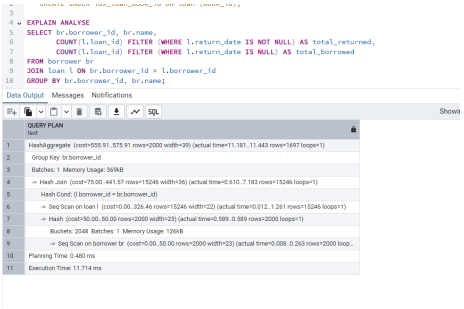


Query #29: List all overdue loans along with borrower and book details

SELECT l.loan_id, br.name AS borrower_name, b.title AS book_title,
(CURRENT_DATE - l.borrow_date) AS overdue_days
FROM loan l

JOIN borrower br ON l.borrower_id = br.borrower_id
JOIN book b ON l.book_id = b.book_id
WHERE l.return_date IS NULL AND (CURRENT_DATE - l.borrow_date) ¿ 10;

Query #30:Find the total number of books borrowed and returned by each borrower
SELECT br.borrower_id, br.name,
COUNT(l.loan_id) FILTER (WHERE l.return_date IS NOT NULL) AS total_returned,
COUNT(l.loan_id) FILTER (WHERE l.return_date IS NULL) AS total_borrowed
FROM borrower br
JOIN loan l ON br.borrower_id = l.borrower_id
GROUP BY br.borrower_id, br.name;



7 Difficulties and Evaluation

7.1 Difficulties

There are several challenges we encountered while carrying out the project. The first issue was managing the level of damaged status. Although we considered addressing this problem in a concrete way, it proved too complex to implement both in the ERD and the functionality. As a result, we decided to remove it to simplify the process. The second challenge involved the data. When generating data, we struggled with figuring out how to include all the books a borrower borrowed in the loan table. We could only add the books sequentially. These two significant problems were the main obstacles our team faced during the project. However, through this experience, we have learned a great deal and plan to apply these lessons to future projects.

7.2 Advantages

- Faster Search and Query Performance: Quick Retrieval of Data, Efficient Borrower Lookups
- Improved Book and Borrower Lookup: Faster Book Lookup, Efficient Transaction Processing
- Faster Fine and Fee Calculations
- Better Resource Management

7.3 Disadvantages

- Increased Storage Requirements: Additional Storage Space, Index Overhead
- Slower Write Operations
- Complexity in Query Optimization: Suboptimal Query Plans, Index Bloat