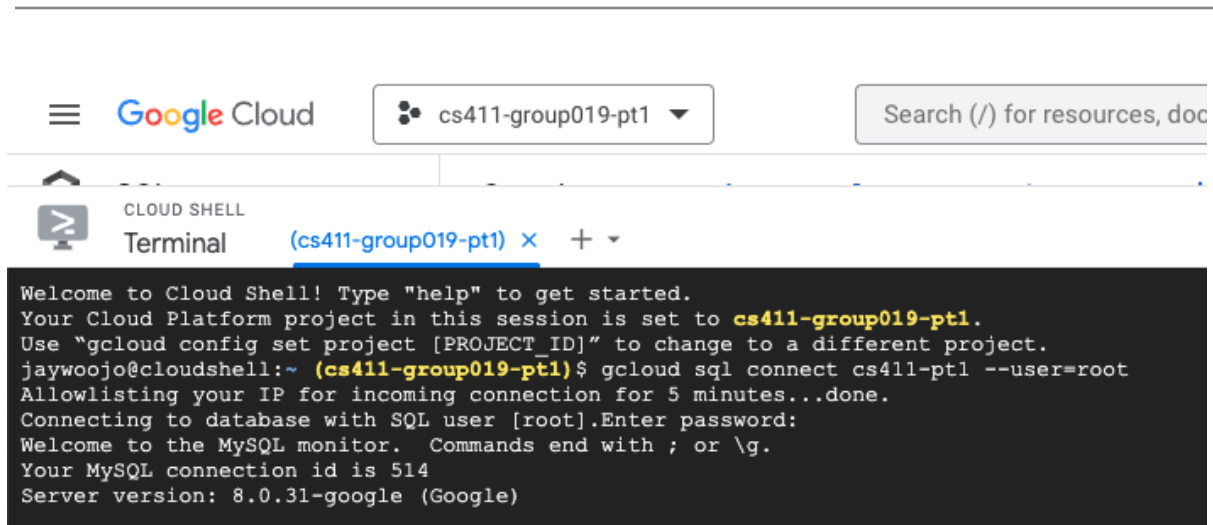


Illini BookBridge Database Implementation and Indexing

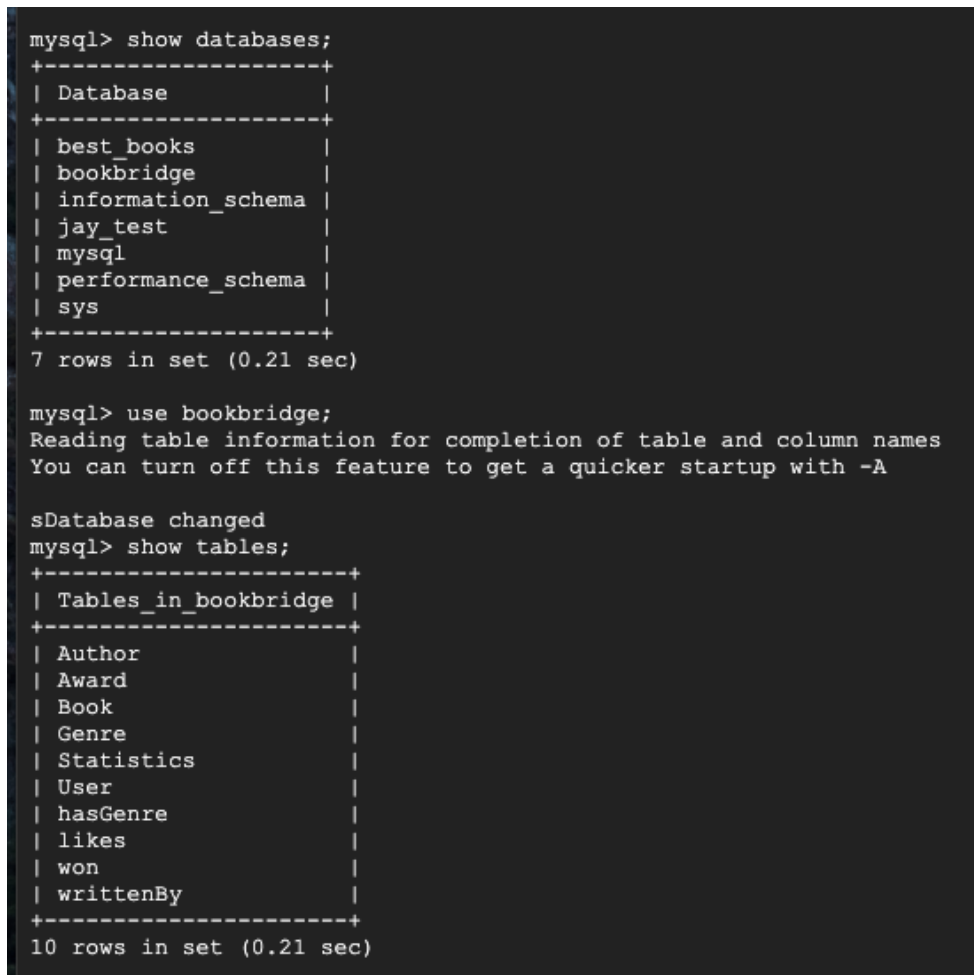
By Jaywoo Jo, Ruhana Azam, John Shen, Alan Zhang

Screenshot of connection



The screenshot shows the Google Cloud Shell interface. At the top, there's a navigation bar with the Google Cloud logo, a dropdown menu showing 'cs411-group019-pt1', and a search bar. Below the navigation bar, there's a 'Terminal' tab for '(cs411-group019-pt1)'. The terminal window displays the following text:

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to cs411-group019-pt1.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
jaywoojo@cloudshell:~ (cs411-group019-pt1)$ gcloud sql connect cs411-pt1 --user=root
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 514
Server version: 8.0.31-google (Google)
```



The screenshot shows a MySQL terminal session. The user has entered the following commands and received the following output:

```
mysql> show databases;
+-----+
| Database |
+-----+
| best_books |
| bookbridge |
| information_schema |
| jay_test |
| mysql |
| performance_schema |
| sys |
+-----+
7 rows in set (0.21 sec)

mysql> use bookbridge;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

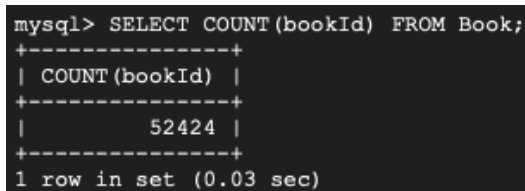
sDatabase changed
mysql> show tables;
+-----+
| Tables_in_bookbridge |
+-----+
| Author |
| Award |
| Book |
| Genre |
| Statistics |
| User |
| hasGenre |
| likes |
| won |
| writtenBy |
+-----+
10 rows in set (0.21 sec)
```

DDL Commands

```
CREATE TABLE User(  
    illiniEmail VARCHAR(255) PRIMARY KEY,  
    firstName VARCHAR(50),  
    lastName VARCHAR(50),  
    password VARCHAR(20),  
    gender VARCHAR(20),  
    introduction VARCHAR(255)  
);
```

(User table does not have any data yet but will be filled with auto-generated data)

```
CREATE TABLE Book (  
    bookId VARCHAR(255) PRIMARY KEY,  
    title VARCHAR(255),  
    series VARCHAR(255),  
    description TEXT,  
    language VARCHAR(30),  
    isbn VARCHAR(13),  
    characters TEXT,  
    publisher VARCHAR(255),  
    publishDate DATE,  
    firstPublicationDate DATE,  
    setting TEXT,  
    coverImg VARCHAR(255),  
    bbeScore REAL,  
    bbeVotes REAL,  
    price REAL  
);
```



```
mysql> SELECT COUNT(bookId) FROM Book;  
+-----+  
| COUNT(bookId) |  
+-----+  
|          52424 |  
+-----+  
1 row in set (0.03 sec)
```

```
CREATE TABLE Author(  
    authorName VARCHAR(50) PRIMARY KEY,  
    averageRating REAL  
);
```

```
mysql> SELECT COUNT(authorName) FROM Author;
+-----+
| COUNT(authorName) |
+-----+
|          34488 |
+-----+
1 row in set (0.63 sec)
```

```
CREATE TABLE Award(
    awardName VARCHAR(255) PRIMARY KEY,
    description TEXT,
    yearCreated INT
);
```

```
mysql> SELECT COUNT(awardName) FROM Award;
+-----+
| COUNT(awardName) |
+-----+
|          5478 |
+-----+
1 row in set (0.66 sec)
```

```
CREATE TABLE Genre(
    genreName VARCHAR(30) PRIMARY KEY,
    description TEXT
);
```

```
mysql> SELECT COUNT(genreName) FROM Genre;
+-----+
| COUNT(genreName) |
+-----+
|          984 |
+-----+
1 row in set (0.11 sec)
```

```
CREATE TABLE Statistics(
    bookID VARCHAR(255),
    rating REAL,
    numRatings INT,
    likedPercent REAL,
    FOREIGN KEY(bookID) REFERENCES Book(bookID)
    ON UPDATE CASCADE ON DELETE CASCADE
);
```

```
mysql> SELECT COUNT(bookId) FROM Statistics;
+-----+
| COUNT(bookId) |
+-----+
|          52478 |
+-----+
1 row in set (0.03 sec)
```

```
CREATE TABLE likes(
    bookID VARCHAR(255),
    email VARCHAR(255),
    FOREIGN KEY(bookID) REFERENCES Book(bookID)
    ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY(email) REFERENCES User(illiniEmail)
    ON UPDATE CASCADE ON DELETE CASCADE
);
```

(Likes table does not have any data yet but will be filled with auto generated data for testing)

```
CREATE TABLE won(
    bookID VARCHAR(255),
    awardName VARCHAR(255),
    yearWon INT,
    FOREIGN KEY(bookID) REFERENCES Book(bookID)
    ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY(awardName) REFERENCES Award(awardName)
    ON UPDATE CASCADE ON DELETE CASCADE
);
```

```
mysql> SELECT COUNT(bookId) FROM won;
+-----+
| COUNT(bookId) |
+-----+
|          20892 |
+-----+
1 row in set (0.33 sec)
```

```
CREATE TABLE hasGenre(
    genreName VARCHAR(255),
    bookId VARCHAR(255),
    FOREIGN KEY(genreName) REFERENCES Genre(genreName)
    ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY(bookId) REFERENCES Book(bookId)
    ON UPDATE CASCADE ON DELETE CASCADE
);
```

```
mysql> SELECT COUNT(bookId) FROM hasGenre;
+-----+
| COUNT(bookId) |
+-----+
|          412321 |
+-----+
1 row in set (0.22 sec)
```

```
CREATE TABLE writtenBy(
    bookId VARCHAR(255),
    authorName VARCHAR(24),
    FOREIGN KEY(bookId) REFERENCES Book(bookId)
    ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY(authorName) REFERENCES Author(authorName)
    ON UPDATE CASCADE ON DELETE CASCADE
);
```

```
mysql> SELECT COUNT(bookId) FROM writtenBy;
+-----+
| COUNT(bookId) |
+-----+
|          68401 |
+-----+
1 row in set (0.04 sec)
```

Advanced Queries

1. This query returns the Authors who have won the most amount of “Goodreads” related-awards on English books after 2010. (Meets requirement for JOIN and GROUP BY. We added the WHERE clauses to explore more indexing options to get better index performance).

```
SELECT authorName, COUNT(awardName) AS numAwards
FROM Book JOIN writtenBy USING(bookId) JOIN won USING(bookId)
WHERE yearWon > 2010 AND language = "English" AND awardName LIKE
    "%Goodreads%"
GROUP BY authorName
ORDER BY COUNT(awardName) DESC
LIMIT 15;
```

```
mysql> SELECT authorName, COUNT(awardName) AS numAwards
-> FROM Book JOIN writtenBy USING(bookId) JOIN won USING(bookId)
-> WHERE yearWon > 2010 AND awardName LIKE "%Goodreads%" AND language = "English"
-> GROUP BY authorName
-> ORDER BY COUNT(awardName) DESC
-> LIMIT 15;
```

authorName	numAwards
Rick Riordan (Goodreads Author)	15
Cassandra Clare (Goodreads Author)	13
Brandon Sanderson (Goodreads Author)	12
Stephen King (Goodreads Author)	11
Sarah J. Maas (Goodreads Author)	11
Colleen Hoover (Goodreads Author)	10
J.R. Ward (Goodreads Author)	9
Richelle Mead (Goodreads Author)	9
Ilona Andrews (Goodreads Author)	9
James S.A. Corey (Goodreads Author)	8
Marie Lu (Goodreads Author)	8
Marissa Meyer (Goodreads Author)	8
Leigh Bardugo (Goodreads Author)	8
Maggie Stiefvater (Goodreads Author)	7
Chris Colfer	7

15 rows in set (0.08 sec)

2. This query returns the Books who have won the most amount of “Goodreads” related-awards on English books after 2010. They are sorted in descending order by rating and then number of awards. (Meets requirement for JOIN and GROUP BY. We added the WHERE clauses to explore more indexing options to get better index performance).

```
SELECT title, rating, COUNT(awardName) as numAwards
FROM Book JOIN Statistics USING(bookId) JOIN won USING(bookId)
WHERE yearWon > 2010 AND language = "English" AND awardName LIKE
      "%Goodreads%"
GROUP BY bookId, rating
ORDER BY rating DESC, numAwards DESC
LIMIT 15;
```

```
mysql> SELECT title, rating, COUNT(awardName) as numAwards
-> FROM Book JOIN Statistics USING(bookId) JOIN won USING(bookId)
-> GROUP BY bookId, rating
-> ORDER BY rating DESC, numAwards DESC
-> LIMIT 15;
```

title	rating	numAwards
Furniture Of The New Zealand Colonial Era: An Illustrated History, 1830 1900	5	2
You, My Love: A Diary in Verse	5	1
PMO Governance: Practical Strategies to Govern Portfolio, Program, and Project Delivery	4.8	5
Dreaming My Animal Selves/Le Songe de Mes Ames Animales	4.79	9
Words of Radiance	4.75	5
Harry Potter Series Box Set	4.73	1
Know My Name	4.71	2
That Ever Died So Young	4.7	1
The Woman in the Woods: Linked Stories	4.68	1
Life Application Study Bible: NIV	4.68	1
The Harry Potter Collection 1-4	4.68	1
Judging Angels	4.67	1
Vattenhjärtat	4.67	1
The Life and Times of Scrooge McDuck	4.67	1
Born to Die in My Place: A Story of Unconditional Love	4.67	1

15 rows in set (0.22 sec)

Index Analysis

Index analysis for Advanced Query #1:

This query returns the Author with most awards and award counts (meet requirement for JOIN and GROUP BY).

```
SELECT authorName, COUNT(awardName) AS numAwards
FROM Book JOIN writtenBy USING(bookId) JOIN won USING(bookId)
WHERE yearWon > 2010 AND awardName LIKE '%Goodreads%' AND language =
      'English'
GROUP BY authorName
ORDER BY COUNT(awardName) DESC
LIMIT 15;
```

Explain analyze default output:

```
| -> Limit: 15 row(s) (actual time=113.941..113.945 rows=15 loops=1)
| -> Sort: numAwards DESC, limit input to 15 row(s) per chunk (actual time=113.941..113.943 rows=15 loops=1)
| -> Table scan on <temporary> (actual time=113.043..113.526 rows=1382 loops=1)
| -> Aggregate using temporary table (actual time=113.040..113.040 rows=1382 loops=1)
| -> Nested loop inner join (cost=2541.02 rows=512) (actual time=0.283..102.480 rows=2087 loops=1)
| -> Nested loop inner join (cost=2361.96 rows=379) (actual time=0.264..75.780 rows=1739 loops=1)
| -> Filter: ((won.yearWon > 2010) and (won.awardName like '%Goodreads%') and (won.bookId is not null)) (cost=2089.55 rows=759) (actual time=0.241..50.988 rows=1791 loops=1)
| -> Table scan on won (cost=2089.55 rows=20493) (actual time=0.031..28.499 rows=20892 loops=1)
| -> Filter: (Book.language = 'English') (cost=0.26 rows=0.5) (actual time=0.012..0.013 rows=1 loops=1791)
| -> Single-row index lookup on Book using PRIMARY (bookId=won.bookId) (cost=0.26 rows=1) (actual time=0.012..0.012 rows=1 loops=1791)
| -> Index lookup on writtenBy using bookId (bookId=won.bookId) (cost=0.34 rows=1) (actual time=0.013..0.015 rows=1 loops=1739)
|
```

Explain analyze after adding index on won.yearWon output:

```
| -> Limit: 15 row(s) (actual time=43.117..43.120 rows=15 loops=1)
    -> Sort: numAwards DESC, limit input to 15 row(s) per chunk (actual time=43.116..43.118 rows=15 loops=1)
    -> Table scan on <temporary> (actual time=42.605..42.904 rows=1382 loops=1)
    -> Aggregate using temporary table (actual time=42.603..42.603 rows=1382 loops=1)
    -> Nested loop inner join (cost=2541.02 rows=512) (actual time=0.174..0.39.759 rows=2087 loops=1)
    -> Nested loop inner join (cost=2361.96 rows=379) (actual time=0.153..0.29.675 rows=1739 loops=1)
    -> Filter: ((won.yearWon > 2010) and (won.awardName like '%Goodreads%') and (won.bookId is not null)) (cost=2089.55 rows=759) (actual time=0.135..0.21.008 rows=1791 loops=1)
    -> Table scan on won (cost=2009.55 rows=20493) (actual time=0.020..0.13.949 rows=20892 loops=1)
    -> Filter: (Book.'language' = 'English') (cost=0.26 rows=0.5) (actual time=0.004..0.005 rows=1 loops=1791)
    -> Single-row index lookup on Book using PRIMARY (bookId=won.bookId) (cost=0.26 rows=1) (actual time=0.004..0.004 rows=1 loops=1791)
    -> Index lookup on writtenBy using bookId (bookId=won.bookId) (cost=0.34 rows=1) (actual time=0.004..0.005 rows=1 loops=1739)
```

Explain analyze after adding index on won.awardName output:

```
| -> Limit: 15 row(s) (actual time=44.920..44.923 rows=15 loops=1)
    -> Sort: numAwards DESC, limit input to 15 row(s) per chunk (actual time=44.919..44.920 rows=15 loops=1)
    -> Table scan on <temporary> (actual time=44.383..44.697 rows=1382 loops=1)
    -> Aggregate using temporary table (actual time=44.380..44.380 rows=1382 loops=1)
    -> Nested loop inner join (cost=2541.02 rows=512) (actual time=0.170..0.41.545 rows=2087 loops=1)
    -> Nested loop inner join (cost=2361.96 rows=379) (actual time=0.154..0.31.023 rows=1739 loops=1)
    -> Filter: ((won.yearWon > 2010) and (won.awardName like '%Goodreads%') and (won.bookId is not null)) (cost=2089.55 rows=759) (actual time=0.134..0.21.278 rows=1791 loops=1)
    -> Filter: (Book.'language' = 'English') (cost=0.26 rows=0.5) (actual time=0.005..0.005 rows=1 loops=1791)
    -> Single-row index lookup on Book using PRIMARY (bookId=won.bookId) (cost=0.26 rows=1) (actual time=0.004..0.005 rows=1 loops=1791)
    -> Index lookup on writtenBy using bookId (bookId=won.bookId) (cost=0.34 rows=1) (actual time=0.005..0.006 rows=1 loops=1739)
```

Explain analyze after adding index on Book.language output:

```
| -> Limit: 15 row(s) (actual time=44.865..44.868 rows=15 loops=1)
    -> Sort: numAwards DESC, limit input to 15 row(s) per chunk (actual time=44.864..44.866 rows=15 loops=1)
    -> Table scan on <temporary> (actual time=44.343..44.650 rows=1382 loops=1)
    -> Aggregate using temporary table (actual time=44.340..44.340 rows=1382 loops=1)
    -> Nested loop inner join (cost=2541.02 rows=512) (actual time=0.164..0.41.402 rows=2087 loops=1)
    -> Nested loop inner join (cost=2361.96 rows=379) (actual time=0.149..0.31.006 rows=1739 loops=1)
    -> Filter: ((won.yearWon > 2010) and (won.awardName like '%Goodreads%') and (won.bookId is not null)) (cost=2089.55 rows=759) (actual time=0.130..0.21.501 rows=1791 loops=1)
    -> Table scan on won (cost=2009.55 rows=20493) (actual time=0.023..0.14.307 rows=20892 loops=1)
    -> Filter: (Book.'language' = 'English') (cost=0.26 rows=0.5) (actual time=0.005..0.005 rows=1 loops=1791)
    -> Single-row index lookup on Book using PRIMARY (bookId=won.bookId) (cost=0.26 rows=1) (actual time=0.004..0.004 rows=1 loops=1791)
    -> Index lookup on writtenBy using bookId (bookId=won.bookId) (cost=0.34 rows=1) (actual time=0.004..0.006 rows=1 loops=1739)
```

We attempted three indexings 1) on won.yearName 2) on year.awardNames 3) on Book.language. When we compare the cost and the times of indexing, we found that there is no major difference between the cost or time between the additionally explored indexed queries and our default, primary key indexed query. We also found that only the primary key index, bookId, is used in all three attempts.

We think this is because of the way we constructed our tables. Because bookId is a primary key for many tables, we think the bookId index is the most efficient and thus SQL always chooses to use it, regardless of the other indices we add.

We experimented with adding WHERE clauses to this query so that we had more attributes to try and use to index but this returned the same results. These can be seen in the screenshots. The EXPLAIN ANALYZE only shows the book.bookid index being utilized for all 4 queries regardless of the indices we attempted.

Index analysis for Advanced Query 2:

This query returns the Books with the highest ratings and number of awards, sorting them in descending order by rating and then number of awards (meets requirement for JOIN and GROUP BY).

```
SELECT title, rating, COUNT(awardName) as numAwards
FROM Book JOIN Statistics USING(bookId) JOIN won USING(bookId)
WHERE yearWon > 2010 AND language = 'English' AND awardName LIKE
```



```
'%Goodreads%'
GROUP BY bookId, rating
ORDER BY rating DESC, numAwards DESC
LIMIT 15;
```

Explain analyze default:

```
-> Limit: 15 row(s) (actual time=48.811..48.815 rows=15 loops=1)
-> Sort: Statistics.rating DESC, numAwards DESC, limit input to 15 row(s) per chunk (actual time=48.810..48.813 rows=15 loops=1)
-> Table scan on <temporary> (actual time=47.925..48.328 rows=1739 loops=1)
-> Aggregate using temporary table (actual time=47.912..47.912 rows=1739 loops=1)
-> Nested loop inner join (cost=2642.87 rows=549) (actual time=0.248..0.738 rows=1739 loops=1)
-> Nested loop inner join (cost=2450.61 rows=503) (actual time=0.227..0.320 rows=1739 loops=1)
-> Filter: ((won.yearWon > 2010) and (won.awardName like '%Goodreads%') and (won.bookId is not null)) (cost=2089.55 rows=1006) (actual time=0.202..0.214 rows=1791 loops=1)
-> Table scan on won (cost=2089.55 rows=20493) (actual time=0.058..0.142 rows=20892 loops=1)
-> Filter: (Book.language = 'English') (cost=0.26 rows=0.5) (actual time=0.005..0.006 rows=1 loops=1791)
-> Single-row index lookup on Book using PRIMARY (bookId=won.bookId) (cost=0.26 rows=1) (actual time=0.005..0.005 rows=1 loops=1791)
-> Index lookup on Statistics using bookId (bookId=won.bookId) (cost=0.27 rows=1) (actual time=0.005..0.006 rows=1 loops=1739)
```

Explain analyze with indexing on Statistics.rating

```
-> Limit: 15 row(s) (actual time=48.463..48.467 rows=15 loops=1)
-> Sort: Statistics.rating DESC, numAwards DESC, limit input to 15 row(s) per chunk (actual time=48.462..48.464 rows=15 loops=1)
-> Table scan on <temporary> (actual time=47.647..48.021 rows=1739 loops=1)
-> Aggregate using temporary table (actual time=47.642..47.642 rows=1739 loops=1)
-> Nested loop inner join (cost=2642.87 rows=549) (actual time=0.175..0.432 rows=1739 loops=1)
-> Nested loop inner join (cost=2450.61 rows=503) (actual time=0.155..0.315 rows=1739 loops=1)
-> Filter: ((won.yearWon > 2010) and (won.awardName like '%Goodreads%') and (won.bookId is not null)) (cost=2089.55 rows=1006) (actual time=0.133..0.212 rows=1791 loops=1)
-> Table scan on won (cost=2089.55 rows=20493) (actual time=0.024..0.140 rows=20892 loops=1)
-> Filter: (Book.language = 'English') (cost=0.26 rows=0.5) (actual time=0.005..0.005 rows=1 loops=1791)
-> Single-row index lookup on Book using PRIMARY (bookId=won.bookId) (cost=0.26 rows=1) (actual time=0.005..0.005 rows=1 loops=1791)
-> Index lookup on Statistics using bookId (bookId=won.bookId) (cost=0.27 rows=1) (actual time=0.006..0.007 rows=1 loops=1739)
```

Explain analyze with indexing on Book.title:

```
-> Limit: 15 row(s) (actual time=47.493..47.496 rows=15 loops=1)
-> Sort: Statistics.rating DESC, numAwards DESC, limit input to 15 row(s) per chunk (actual time=47.492..47.494 rows=15 loops=1)
-> Table scan on <temporary> (actual time=46.678..47.058 rows=1739 loops=1)
-> Aggregate using temporary table (actual time=46.673..46.673 rows=1739 loops=1)
-> Nested loop inner join (cost=2642.87 rows=549) (actual time=0.195..0.420 rows=1739 loops=1)
-> Nested loop inner join (cost=2450.61 rows=503) (actual time=0.176..0.315 rows=1739 loops=1)
-> Filter: ((won.yearWon > 2010) and (won.awardName like '%Goodreads%') and (won.bookId is not null)) (cost=2089.55 rows=1006) (actual time=0.153..0.212 rows=1791 loops=1)
-> Table scan on won (cost=2089.55 rows=20493) (actual time=0.026..0.141 rows=20892 loops=1)
-> Filter: (Book.language = 'English') (cost=0.26 rows=0.5) (actual time=0.005..0.005 rows=1 loops=1791)
-> Single-row index lookup on Book using PRIMARY (bookId=won.bookId) (cost=0.26 rows=1) (actual time=0.005..0.005 rows=1 loops=1791)
-> Index lookup on Statistics using bookId (bookId=won.bookId) (cost=0.27 rows=1) (actual time=0.005..0.006 rows=1 loops=1739)
```

Explain analyze with indexing on Book.language:

```
-> Limit: 15 row(s) (actual time=46.993..46.997 rows=15 loops=1)
-> Sort: Statistics.rating DESC, numAwards DESC, limit input to 15 row(s) per chunk (actual time=46.992..46.994 rows=15 loops=1)
-> Table scan on <temporary> (actual time=46.192..46.567 rows=1739 loops=1)
-> Aggregate using temporary table (actual time=46.188..46.188 rows=1739 loops=1)
-> Nested loop inner join (cost=2507.02 rows=414) (actual time=0.226..0.426 rows=1739 loops=1)
-> Nested loop inner join (cost=2361.96 rows=379) (actual time=0.206..0.316 rows=1739 loops=1)
-> Filter: ((won.yearWon > 2010) and (won.awardName like '%Goodreads%') and (won.bookId is not null)) (cost=2089.55 rows=759) (actual time=0.182..0.214 rows=1791 loops=1)
-> Table scan on won (cost=2089.55 rows=20493) (actual time=0.032..0.142 rows=20892 loops=1)
-> Filter: (Book.language = 'English') (cost=0.26 rows=0.5) (actual time=0.005..0.005 rows=1 loops=1791)
-> Single-row index lookup on Book using PRIMARY (bookId=won.bookId) (cost=0.26 rows=1) (actual time=0.005..0.005 rows=1 loops=1791)
-> Index lookup on Statistics using bookId (bookId=won.bookId) (cost=0.27 rows=1) (actual time=0.005..0.006 rows=1 loops=1739)
```

We attempted three indexings 1) on Statistics.rating 2) on Book.title 3) on Book.language. When we compare the cost and the times of indexing, we found that there is no major difference between the cost or time between the additional indexed queries and our default, primary key indexed query. Additionally, we found that only the default index, the primary key bookId, is used in all three attempts.

Again, we believe this is because of how our tables are constructed. In our tables, bookId is a primary key for most tables so we think the bookId index is the most efficient and thus SQL always chooses to use it, regardless of the other indices we add.

We experimented with adding WHERE clauses to this query so that we had more attributes to try and use to index but this returned the same results. These can be seen in the screenshots. The EXPLAIN

ANALYZE only shows the book.bookid index being utilized for all 4 queries regardless of the indices we attempted.