page_list.pdf

data_model_diagram.pdf

CS 4332 – Round Rock Group 4: Jeffrey Hogan, William Howland, Luis Pena, Robert Preston

Web page summary with SQL queries for each.

Endpoint: / (Main index/home)
Information: The main page, shows books and lists.

Endpoint: /books/author (books_by_authors.html)
Information: Books sorted by author.
Queries:
Python:

```
with easypg.cursor() as cur:
    if flask.ext.login.current_user.is_authenticated():
        total_pages, book_info = books.get_books_by_authors(cur, start, limit, flask.session['user_id'],
sorting, sort_direction)
    else:
        total_pages, book_info = books.get_books_by_authors(cur, start, limit, None, sorting,
sort_direction)
```

SQL:

```
SELECT author_id, author_name, AVG(rating) as avg_rating, COUNT(DISTINCT core_id) as
num_books, SUM(page_count) as num_pages
    FROM author
    JOIN authorship USING (author_id)
    LEFT JOIN ratings ON core_id = ratings.book_id
    LEFT JOIN books USING (core_id)
    %s
    GROUP BY author_id, author_name
    %s
    LIMIT %s OFFSET %s
```

Endpoint: /author/<author_name>(author.html)
Information: Displays information about an author, including books written, information about the author, total pages written and more.
Queries:
Python:

```
total_pages, author_info = books.get_books_by_author(cur, start, limit, author_name,
flask.session['user_id'], sorting, sort_direction)
```

SQL:

```
 SELECT author_id, author_name, AVG(rating) as avg_rating, COUNT(DISTINCT core_id) as
num_books, SUM(page_count) as num_pages
    FROM author
    JOIN authorship USING (author_id)
    LEFT JOIN ratings ON core_id = ratings.book_id
    LEFT JOIN books USING (core_id)
    %s
    GROUP BY author_id, author_name
    %s
    LIMIT %s OFFSET %s
```

Endpoint: /books/publishers (books_by_publishers.html)
Information: Displays publishers and information about them in a clickable list format on thepage.
Queries:
Python:

```
    with easypg.cursor() as cur:
        if flask.ext.login.current_user.is_authenticated():
            total_pages, publisher_info = books.get_books_by_publishers(cur, start, limit,
flask.session['user_id'], sorting, sort_direction)
        else:
            total_pages, publisher_info = books.get_books_by_publishers(cur, start, limit, None, sorting,
sort_direction)
```

SQL:

```
   SELECT publisher_id, publisher_name, AVG(rating) as avg_rating, COUNT(DISTINCT core_id) as
num_books, SUM(page_count) as num_pages
        FROM publisher
        JOIN book_publisher USING (publisher_id)
        LEFT JOIN ratings USING (book_id)
        LEFT JOIN books USING (book_id)
        %s
        GROUP BY publisher_id, publisher_name
        %s
        LIMIT %s OFFSET %s
```

Endpoint: /publisher/<pid> (author.html)
Information: Displays information about a publisher on its own page, and gives information on which books they have published.
Queries:
Python:

```
with easypg.cursor() as cur:
        if flask.ext.login.current_user.is_authenticated():
            total_pages, publisher_info = books.get_books_by_publisher(cur, start, limit, pid,
flask.session['user_id'], sorting, sort_direction)
        else:
            total_pages, publisher_info = books.get_books_by_publisher(cur, start, limit, pid, None,
sorting, sort_direction)

    sort_options = {"Title": "book_title", "Publication Date": "publication_date", "Avg. Rating":
"avg_rating",
                    "Number of Readers": "num_readers"}
```

SQL:

```
   SELECT publisher_id, publisher_name, AVG(rating) as avg_rating, COUNT(DISTINCT core_id) as
num_books, SUM(page_count) as num_pages
        FROM publisher
        JOIN book_publisher USING (publisher_id)
        LEFT JOIN ratings USING (book_id)
        LEFT JOIN books USING (book_id)
        %s
        GROUP BY publisher_id, publisher_name
```

```
    %s
    LIMIT %s OFFSET %s
```

Endpoint: /books/subjects (books_by_subjects.html)
Information: Lists books by tagged subjects.
Queries:
Python:

```
  with easypg.cursor() as cur:
    if flask.ext.login.current_user.is_authenticated():
      total_pages, book_info = books.get_books_by_subjects(cur, start, limit, flask.session['user_id'],
sorting, sort_direction)
    else:
      total_pages, book_info = books.get_books_by_subjects(cur, start, limit, None, sorting,
sort_direction)
```

SQL:

```
SELECT subject_id, subject_name, AVG(rating) as avg_rating, COUNT(DISTINCT core_id) as
num_books, SUM(page_count) as num_pages
    FROM subject_genre
    JOIN book_categorization USING (subject_id)
    LEFT JOIN ratings ON core_id = ratings.book_id
    LEFT JOIN books USING (core_id)
    %s
    GROUP BY subject_id, subject_name
    %s
    LIMIT %s OFFSET %s
```

Endpoint: /books/subjects/<subject> (books_index.html)
Information: Lists all books with share a tag. User will have clicked on a tag to reach this page, so it will look for other books with that tag and list them here.
Queries:
Python:

```
with easypg.cursor() as cur:
    if flask.ext.login.current_user.is_authenticated():
      total_pages, book_info = books.get_books_by_subject(cur, start, limit, subject,
flask.session['user_id'], sorting, sort_direction)
    else:
      total_pages, book_info = books.get_books_by_subject(cur, start, limit, subject, None, sorting,
sort_direction)

  sort_options = {"Title": "book_title", "Publication Date": "publication_date", "Avg. Rating":
"avg_rating",
          "Number of Readers": "num_readers"}
  parameters = "&sorting=%s&sort_direction=%s" % (sorting, sort_direction)
```

SQL:

```
SELECT subject_id, subject_name, AVG(rating) as avg_rating, COUNT(DISTINCT core_id) as
num_books, SUM(page_count) as num_pages
    FROM subject_genre
    JOIN book_categorization USING (subject_id)
```

```
    LEFT JOIN ratings ON core_id = ratings.book_id
    LEFT JOIN books USING (core_id)
    %s
    GROUP BY subject_id, subject_name
    %s
    LIMIT %s OFFSET %s
```

Endpoint: /books/<bid> (book.html)
Information: Displays detailed information about a book, including publisher, author, page count, ISBN, and more.
Queries:

```
SELECT DISTINCT core_id,book_title, isbn, page_count, COALESCE(publisher_name,'Unknown'), book_description,
    to_char(publication_date,'Mon. DD, YYYY') as publication_date,
to_char(publication_date,'MM/DD/YYYY') as publication_date_fmt,
    COALESCE(cover_name,'_placeholder') as cover_name, AVG(rating) as avg_rating
    FROM book_core
    LEFT JOIN books USING (core_id)
    LEFT JOIN ratings ON core_id = ratings.book_id
    LEFT JOIN book_publisher ON core_id = book_publisher.book_id
    LEFT JOIN publisher USING (publisher_id)
    WHERE core_id = %s
    GROUP BY core_id, book_title, book_description, cover_name, isbn, page_count,
publication_date, publication_date_fmt, publisher_name
  '", (book_id,))
```

Endpoint: /books (books_index.html)
Information: Displays a list of books sorted by book ID by default. Also shows author name, and you can add the books to a list or write reviews of the book from here.
Queries:

```
 SELECT core_id, book_title, book_description, isbn, page_count, cover_name,
      AVG(rating) as avg_rating, COUNT(DISTINCT log_id) as num_readers
    FROM book_core
    JOIN books USING (core_id)
    LEFT JOIN ratings ON core_id = ratings.book_id
    LEFT JOIN user_log ON core_id = user_log.book_id
    WHERE books.is_active = TRUE
    GROUP BY core_id, book_title, book_description, cover_name, isbn, page_count,
publication_date
    %s
    LIMIT %s OFFSET %s
```

Endpoint: /books/covers (books_index.html)
Information: An extra addition to the site that shows the books that have covers. The site does support showing covers, so this is a good way to see the cover viewing functionality since only a very small percentage of books have covers in the imported data.
Queries:
Python:

```
  total_pages = get_total_pages(cur, QUERIES['books_with_covers_count'])
```

```
    cur.execute(QUERIES['select_books'] % (order_by,'%s','%s'), (amount, start))
    query = QUERIES['select_books_where'] % ('', '', ' AND cover_name IS NOT NULL', '',
order_by,'%s','%s')
    cur.execute(query, (amount, start))


    book_info = []
    for core_id, book_title, description, isbn, page_count, cover_name, avg_rating, num_readers in cur:

        print cover_name
        if avg_rating:
            discrete_rating = round(avg_rating*2) / 2
        else:
            discrete_rating = 0
        try:
            avg_rating = round(avg_rating,2)
        except TypeError:
            pass
        book_info.append({'core_id':core_id, 'title': str(book_title).decode('utf8', 'xmlcharrefreplace'),
                    'cover_name': cover_name, 'authors': [], 'subjects': [], 'isbn': isbn, 'num_pages':
page_count,
                    'num_readers': num_readers, 'avg_rating': avg_rating, 'discrete_rating': discrete_rating,
'user_rating': None})
    for book in book_info:
        cur.execute('''
        SELECT author_name
        FROM author JOIN authorship USING (author_id)
        WHERE core_id = %s
        ''', (book['core_id'],))
        author_info = []
        for author_name in cur:
            author_info.append(str(author_name[0]).decode('utf8', 'xmlcharrefreplace'))

        book['authors'] = author_info
        cur.execute('''
        SELECT subject_name
        FROM subject_genre JOIN book_categorization USING (subject_id)
        WHERE core_id = %s
        ''', (book['core_id'],))
        for subject_name in cur:
            book['subjects'].append(subject_name[0].decode('utf8', 'xmlcharrefreplace'))
        # book['subjects'] = subject_info
        if (user_id):
            cur.execute('''
            SELECT rating
            FROM ratings
            WHERE book_id = %s AND rater = %s
            ''', (book['core_id'], user_id))
            if cur.rowcount > 0:
```

```
            book['user_rating'] = cur.fetchone()[0]
```
SQL:

```
SELECT core_id, book_title, book_description, isbn, page_count, cover_name,
        AVG(rating) as avg_rating, COUNT(DISTINCT log_id) as num_readers
       FROM book_core
       JOIN books USING (core_id)
       LEFT JOIN ratings ON core_id = ratings.book_id
       LEFT JOIN user_log ON core_id = user_log.book_id
       WHERE books.is_active = TRUE
       GROUP BY core_id, book_title, book_description, cover_name, isbn, page_count,
publication_date
       %s
       LIMIT %s OFFSET %s

SELECT %s core_id, book_title, book_description, isbn, COALESCE(page_count,0),
COALESCE(cover_name,'_placeholder') as cover_name, AVG(rating) as avg_rating,
COUNT(DISTINCT log_id) as num_readers
       FROM book_core
       LEFT JOIN books USING (core_id)
       %s
       LEFT JOIN ratings ON core_id = ratings.book_id
       LEFT JOIN user_log ON core_id = user_log.book_id
       LEFT JOIN book_categorization USING (core_id)
       WHERE books.is_active = TRUE %s AND cover_name IS NOT NULL
       GROUP BY %s core_id, book_title, book_description, cover_name, isbn, page_count,
publication_date
       %s
       LIMIT %s OFFSET %s
```

Endpoint: /book/add
Information: Part of the book adding function.

Endpoint: /book/add/<bid> (book_edit_form.html)
Information: The main book adding form, allows a user to edit information on a new book and then puts it in the moderation queue to be inserted when the user is finished.
Queries:
Python:
```
if core_id == 0:
       cur.execute('''
          SELECT core_id
          FROM book_core
          WHERE book_title = %s
       ''', (book_title,))
       if cur.rowcount < 1:
          cur.execute('''
            INSERT INTO book_core (book_title, book_description, edition, is_active)
            VALUES(%s, %s, %s, %s)
            RETURNING core_id
```

```
        ''', (book_title, book_description, 1, False))
    core_id = cur.fetchone()[0]

cur.execute('''
    INSERT INTO books (core_id, publication_date, isbn, book_type, page_count, is_active)
    VALUES(%s, %s, %s, %s, %s, %s)
    RETURNING book_id
''', (core_id, publication_date, isbn, book_type, page_count, False))

if cur.rowcount > 0:
    book_id = cur.fetchone()[0]
    message.append("New book %s queued for insertion!" % book_title)

else:
    message.append("Unknown error!")
    return False, message, 0

for position,author in author_names.iteritems():
    # First check to see if we have a matching author
    cur.execute('''
    SELECT author_id
    FROM author
    WHERE author_name = %s
    ''', (author,))
    if cur.rowcount == 0:
        # Need to insert this author
        cur.execute('''
        INSERT INTO author (author_name)
        VALUES(%s)
        RETURNING author_id
        ''', (author,))
    author_id = cur.fetchone()[0]

    # Now we see if this author is already associated with our book
    cur.execute('''
    SELECT authorship_id
    FROM authorship
    JOIN author USING (author_id)
    WHERE core_id = %s AND author_name = %s
    ''', (core_id,author))
    if cur.rowcount == 0:
        # Need to insert this author
        cur.execute('''
        INSERT INTO authorship (core_id, author_id, position)
        VALUES(%s, %s, %s)
        RETURNING authorship_id
        ''', (core_id,author_id,position))
    authorship_id = cur.fetchone()[0]
```

```
    for subject in subjects:

        cur.execute('''
        SELECT subject_id
        FROM subject_genre
        WHERE subject_name = %s
        ''', (subject,))
        if cur.rowcount == 0:
            # Need to insert this subject
            cur.execute('''
            INSERT INTO subject_genre (subject_name)
            VALUES(%s)
            RETURNING subject_id
            ''', (subject,))
        subject_id = cur.fetchone()[0]

        # Now we see if this author is already associated with our book
        cur.execute('''
        SELECT categorize_id
        FROM book_categorization
        JOIN subject_genre USING (subject_id)
        WHERE core_id = %s AND subject_name = %s
        ''', (core_id,subject))
        if cur.rowcount == 0:
            cur.execute('''
            INSERT INTO book_categorization (core_id, subject_id)
            VALUES(%s, %s)
            RETURNING categorize_id
            ''', (core_id,subject_id))
        categorize_id = cur.fetchone()[0]


    # And finally enter the new book in the queue for approval
    cur.execute('''
        INSERT INTO request (user_id, type, date_requested, priority, status, date_of_status)
        VALUES(%s, %s, NOW(), %s, %s, NOW())
        RETURNING request_id
    ''', (user_id, "Add Book", '1', "Awaiting Review"))
    request_id = cur.fetchone()[0]

        # And finally enter the new book in the queue for approval
    cur.execute('''
        INSERT INTO request_on_book (request_id, book_id, request_type, request_text)
        VALUES(%s, %s, %s, %s)
        RETURNING request_on_book_id
    ''', (request_id, book_id, "Add Book", request_text))
    request_on_book_id = cur.fetchone()[0]
```

SQL:

```
SELECT core_id
          FROM book_core
          WHERE book_title = %s
       ''', (book_title,))
            INSERT INTO book_core (book_title, book_description, edition, is_active)
            VALUES(%s, %s, %s, %s)
            RETURNING core_id
          ''', (book_title, book_description, 1, False))

 INSERT INTO books (core_id, publication_date, isbn, book_type, page_count, is_active)
       VALUES(%s, %s, %s, %s, %s, %s)
       RETURNING book_id
   ''', (core_id, publication_date, isbn, book_type, page_count, False))

 SELECT author_id
       FROM author
       WHERE author_name = %s
       ''', (author,))
       if cur.rowcount == 0:
          cur.execute('''
          INSERT INTO author (author_name)
          VALUES(%s)
          RETURNING author_id
          ''', (author,))
 SELECT authorship_id
       FROM authorship
       JOIN author USING (author_id)
       WHERE core_id = %s AND author_name = %s
       ''', (core_id,author))

       INSERT INTO authorship (core_id, author_id, position)
          VALUES(%s, %s, %s)
          RETURNING authorship_id
          ''', (core_id,author_id,position))

 SELECT subject_id
       FROM subject_genre
       WHERE subject_name = %s
       ''', (subject,))

 INSERT INTO subject_genre (subject_name)
          VALUES(%s)
          RETURNING subject_id
          ''', (subject,))

 SELECT categorize_id
       FROM book_categorization
       JOIN subject_genre USING (subject_id)
       WHERE core_id = %s AND subject_name = %s
```

```
        ''', (core_id,subject))

    INSERT INTO book_categorization (core_id, subject_id)
            VALUES(%s, %s)
            RETURNING categorize_id
            ''', (core_id,subject_id))

INSERT INTO request (user_id, type, date_requested, priority, status, date_of_status)
        VALUES(%s, %s, NOW(), %s, %s, NOW())
        RETURNING request_id
    ''', (user_id, "Add Book", '1', "Awaiting Review"))

INSERT INTO request_on_book (request_id, book_id, request_type, request_text)
        VALUES(%s, %s, %s, %s)
        RETURNING request_on_book_id
    ''', (request_id, book_id, "Add Book", request_text))
```

Endpoint: /book/edit/<bid> (book_edit_form.html)
Information: Allows a user to edit book information and then submit the changes.
Queries:
Python:
print form

```
    update_status = True

    author_names = {}
    subjects = []
    for key, value in form.iteritems():
        if 'inputBookAuthor' in key:
            author_names[key[15:]] = value #try to keep position
        if 'inputBookSubject' in key:
            subjects.append(value)

    publication_date = form['inputBookPubDate']
    isbn = form['inputBookISBN']
    page_count = form['inputBookPageCount']
    book_type = form['inputBookType']
    book_title = form['inputBookTitle']
    book_description = form['inputBookDescription']

    cur.execute('''
        UPDATE books SET
        publication_date = %s
        ISBN = %s
        book_type = %s
        page_count = %s
        book_title = %s
        WHERE book_id = %s
```

```
    RETURNING book_id
''', (publication_date, isbn, book_type, page_count, book_title, book_id))
if cur.rowcount == 1:
    message = "Book %s updated!" % book_title

else:
    message = "Unknown error!"

for position,author in author_names.iteritems():
    # First check to see if we have a matching author
    cur.execute('''
    SELECT author_id
    FROM author
    WHERE author_name = %s
    ''', (author,))
    if cur.rowcount == 0:
        # Need to insert this author
        cur.execute('''
        INSERT INTO author (author_name)
        VALUES(%s)
        RETURNING author_id
        ''', (author,))
    author_id = cur.fetchone()[0]

    # Now we see if this author is already associated with our book
    cur.execute('''
    SELECT authorship_id
    FROM authorship
    WHERE core_id = %s AND author_id = %s
    ''', (book_id,author))
    if cur.rowcount == 0:
        # Need to insert this author
        cur.execute('''
        INSERT INTO authorship (core_id, author_id, position)
        VALUES(%s, %s, %s)
        RETURNING authorship_id
        ''', (book_id,author_id,position))
    authorship_id = cur.fetchone()[0]

for subject in subjects:

    cur.execute('''
    SELECT subject_id
    FROM subject_genre
    WHERE subject_name = %s
    ''', (subject,))
    if cur.rowcount == 0:
        # Need to insert this subject
        cur.execute('''
```

```
            INSERT INTO subject_genre (subject)
            VALUES(%s)
            RETURNING subject_id
            ''', (subject,))
        subject_id = cur.fetchone()[0]

        # Now we see if this author is already associated with our book
        cur.execute('''
        SELECT categorize_id
        FROM book_categorization
        WHERE core_id = %s AND subject_id = %s
        ''', (book_id,subject))
        if cur.rowcount == 0:
            cur.execute('''
            INSERT INTO book_categorization (core_id, subject_id)
            VALUES(%s, %s)
            RETURNING authorship_id
            ''', (book_id,author_id))
        categorize_id = cur.fetchone()[0]
    return True, message
SQL:
UPDATE books SET
        publication_date = %s
        ISBN = %s
        book_type = %s
        page_count = %s
        book_title = %s
        WHERE book_id = %s
        RETURNING book_id

SELECT author_id
        FROM author
        WHERE author_name = %s
        ''', (author,))

 INSERT INTO author (author_name)
            VALUES(%s)
            RETURNING author_id
            ''', (author,))

 SELECT authorship_id
        FROM authorship
        WHERE core_id = %s AND author_id = %s
        ''', (book_id,author))

 INSERT INTO authorship (core_id, author_id, position)
            VALUES(%s, %s, %s)
            RETURNING authorship_id
            ''', (book_id,author_id,position))
```

```
  SELECT subject_id
      FROM subject_genre
      WHERE subject_name = %s

  INSERT INTO subject_genre (subject)
         VALUES(%s)
         RETURNING subject_id

SELECT categorize_id
      FROM book_categorization
      WHERE core_id = %s AND subject_id = %s

    INSERT INTO book_categorization (core_id, subject_id)
         VALUES(%s, %s)
         RETURNING authorship_id
```

Endpoint: /log/<lid> (log.html)
Information: Displays logs.
Queries:

```
   SELECT log_id, core_id, book_id, book_title, reader, login_name, log_text,
to_char(date_started,'Mon. DD, YYYY') as date_started, to_char(date_completed,'Mon. DD, YYYY')
as date_completed
      FROM user_log
      JOIN books USING (book_id)
      JOIN book_core USING (core_id)
      JOIN booknet_user ON reader = user_id
      WHERE log_id = %s
```

Endpoint: /log/<year>
Information: Will display logs sorted by year.

Endpoint: /books/log/add
Information: Not so much a page as something that is processed by our site, but this is how a user adds a log.
Queries:
Python:

```
create_status = True
   message = []
   log_id = 0

   book_id = form['book_id']
   book_title, book_core = get_book_title(cur, book_id);
   log_text = form['log_input']
   date_started = form['starting_date']
   date_completed = form['date_completed']

   cur.execute('''
     INSERT INTO user_log (book_id, reader, status, date_completed, pages_read, log_text,
```

```
date_started)
      VALUES(%s, %s, 1, %s, 1, %s, %s)
      RETURNING log_id
    ''', (book_id, user_id, date_completed, log_text, date_started))
    if cur.rowcount == 1:
        log_id = cur.fetchone()[0]
        message.append("Created new log for: %s!" % book_title)
    else:
        message.append("Unknown error!")
        create_status = False

    return create_status, message, log_id
SQL:
cur.execute('''
      INSERT INTO user_log (book_id, reader, status, date_completed, pages_read, log_text,
date_started)
      VALUES(%s, %s, 1, %s, 1, %s, %s)
      RETURNING log_id
    ''', (book_id, user_id, date_completed, log_text, date_started))
```

Endpoint: /log/_current_user/<bid>
Information: Another function of the site, this is the log retrieval function.
Queries:

```
 SELECT log_id, log_text, to_char(date_started,'Mon. DD, YYYY') as date_started,
      to_char(date_completed,'Mon. DD, YYYY') as date_completed
      FROM user_log
      WHERE reader = %s AND book_id = %s;
```

Endpoint: /list/add/bid
Information: This is our add-book-to-list functionality.
Queries:
relevant Python:

```
cur.execute(query, (user_id, book_id))

    for log_id, log_text, date_completed, date_started in cur:
        logs[cur.rownumber] = {'log_id': log_id, 'log_text': log_text, 'date_completed': date_completed,
'date_started': date_started};

    return logs
SQL queries:
SELECT DISTINCT core_id,book_title, isbn, page_count, COALESCE(publisher_name,'Unknown'),
book_description,
        to_char(publication_date,'Mon. DD, YYYY') as publication_date,
to_char(publication_date,'MM/DD/YYYY') as publication_date_fmt,
        COALESCE(cover_name,'_placeholder') as cover_name, AVG(rating) as avg_rating
        FROM book_core
        LEFT JOIN books USING (core_id)
        LEFT JOIN ratings ON core_id = ratings.book_id
        LEFT JOIN book_publisher ON core_id = book_publisher.book_id
```

```
        LEFT JOIN publisher USING (publisher_id)
        WHERE core_id = %s
        GROUP BY core_id, book_title, book_description, cover_name, isbn, page_count,
publication_date, publication_date_fmt, publisher_name

SELECT booklist_id
        FROM book_list
        WHERE book_id = %s AND list_id = %s

 INSERT INTO book_list (book_id, list_id, date_added)
          VALUES(%s, %s, current_timestamp)
          RETURNING booklist_id
        ''', (book_info['core_id'], list_id))
```

Endpoint: /list/<lid> (list.html)
Information: Displays a list of the book lists created by users. Sorted by list id.
Queries:

```
    SELECT list_id, list_name, user_id, login_name, to_char(list.date_created,'Mon. DD, YYYY') as
date_created, list_description, COUNT(DISTINCT book_id) as num_books
        FROM list
        JOIN book_list USING (list_id)
        JOIN booknet_user USING (user_id)
        WHERE list_id = %s
        GROUP BY list_id, list_name, user_id, login_name, list.date_created, list_description
```

Endpoint: /list/ (lists_list.html)
Information: List of all lists.
Queries:

```
 SELECT COUNT(*)
        FROM list;

SELECT list_id, list_name, user_id, login_name, to_char(list.date_created,'Mon. DD, YYYY') as
date_created, list_description, COUNT(DISTINCT book_id) as num_books
        FROM list
        JOIN book_list USING (list_id)
        JOIN booknet_user USING (user_id)
        GROUP BY list_id, list_name, user_id, login_name, list.date_created, list_description
        LIMIT %s OFFSET %s
```

Endpoint: /list/_current_user
Information: Display the current user's lists.
Queries:

```
SELECT list_id, list_name, COUNT(DISTINCT book_id) as num_books
      FROM list JOIN book_list USING (list_id)
      WHERE user_id = %s
      GROUP BY list_id, list_name;
```

Endpoint: /books/ratings/add/<bid>
Information: A function to add a rating to a book.

Queries:
Python:

```
book_info = get_book(cur, book_id)
   # try:
   cur.execute('''
      SELECT ratings
      FROM ratings
      WHERE book_id = %s AND rater = %s
   ''', (book_id, user_id))
   # first two queries should be able to be combined
   if cur.rowcount == 1:
      cur.execute('''
        UPDATE ratings SET rating = %s
        WHERE book_id = %s AND rater = %s
        RETURNING ratings
      ''', (rating, book_info['core_id'], user_id))
      if cur.rowcount == 1:
         message = "Rating of %s updated for %s!" % (rating, book_info['title'])
      else:
         message = "Unknown error!"
   # except Exception, e:
   #     message = errorcodes.lookup(e.pgcode[:2])
   else:
      cur.execute('''
        INSERT INTO ratings (book_id, rater, rating, date_rated)
        VALUES(%s, %s, %s, current_timestamp)
        RETURNING ratings
      ''', (book_info['core_id'], user_id, rating))
      if cur.rowcount == 1:
         message = "Rating of %s saved for %s!" % (rating, book_info['title'])
      else:
         message = "Unknown error!"
```

SQL:

```
   SELECT ratings
      FROM ratings
      WHERE book_id = %s AND rater = %s

SELECT DISTINCT core_id,book_title, isbn, page_count, COALESCE(publisher_name,'Unknown'),
book_description,
      to_char(publication_date,'Mon. DD, YYYY') as publication_date,
to_char(publication_date,'MM/DD/YYYY') as publication_date_fmt,
      COALESCE(cover_name,'_placeholder') as cover_name, AVG(rating) as avg_rating
      FROM book_core
      LEFT JOIN books USING (core_id)
      LEFT JOIN ratings ON core_id = ratings.book_id
      LEFT JOIN book_publisher ON core_id = book_publisher.book_id
      LEFT JOIN publisher USING (publisher_id)
      WHERE core_id = %s
      GROUP BY core_id, book_title, book_description, cover_name, isbn, page_count,
```

publication_date, publication_date_fmt, publisher_name

```
UPDATE ratings SET rating = %s
        WHERE book_id = %s AND rater = %s
        RETURNING ratings

INSERT INTO ratings (book_id, rater, rating, date_rated)
        VALUES(%s, %s, %s, current_timestamp)
        RETURNING ratings
```

Endpoint: /books/rating/remove/<bid>
Information: A function to remove ratings from books..
Queries:

```
 SELECT DISTINCT core_id,book_title, isbn, page_count, COALESCE(publisher_name,'Unknown'),
book_description,
       to_char(publication_date,'Mon. DD, YYYY') as publication_date,
to_char(publication_date,'MM/DD/YYYY') as publication_date_fmt,
       COALESCE(cover_name,'_placeholder') as cover_name, AVG(rating) as avg_rating
       FROM book_core
       LEFT JOIN books USING (core_id)
       LEFT JOIN ratings ON core_id = ratings.book_id
       LEFT JOIN book_publisher ON core_id = book_publisher.book_id
       LEFT JOIN publisher USING (publisher_id)
       WHERE core_id = %s
       GROUP BY core_id, book_title, book_description, cover_name, isbn, page_count,
publication_date, publication_date_fmt, publisher_name

   DELETE FROM ratings
      WHERE book_id = %s AND rater = %s
```

Endpoint: /reviews/book/<bid> (review_list.html)
Information: Displays a list of reviews for a book.
Queries:
Python:

```
   total_pages = get_total_pages(cur, QUERIES['reviews_by_book_count'] % bid)
   query = QUERIES['select_reviews_where'] % ('WHERE review.book_id = %s', '%s', '%s')
   book_info = books.get_book(cur,bid,current_user_id)

   cur.execute(query, (bid, amount, start))

   review_info = {'book_info': book_info, 'reviews': []}
   for  review_id, core_id, book_id, book_title, reviewer, login_name, date_reviewed, review_text in
cur:
       review_info['reviews'].append({'core_id':core_id, 'id':review_id, 'book_id': book_id,
'book_title':book_title.decode('utf8', 'xmlcharrefreplace'), 'reviewer':reviewer, 'reviewer_name':
login_name, 'date_reviewed': date_reviewed, 'review_text':review_text})

   return total_pages, review_info
```

SQL:
SELECT DISTINCT core_id,book_title, isbn, page_count, COALESCE(publisher_name,'Unknown'), book_description,
to_char(publication_date,'Mon. DD, YYYY') as publication_date,
to_char(publication_date,'MM/DD/YYYY') as publication_date_fmt,
COALESCE(cover_name,'_placeholder') as cover_name, AVG(rating) as avg_rating
FROM book_core
LEFT JOIN books USING (core_id)
LEFT JOIN ratings ON core_id = ratings.book_id
LEFT JOIN book_publisher ON core_id = book_publisher.book_id
LEFT JOIN publisher USING (publisher_id)
WHERE core_id = %s
GROUP BY core_id, book_title, book_description, cover_name, isbn, page_count, publication_date, publication_date_fmt, publisher_name

SELECT review_id, book_core.core_id, book_id, book_title, reviewer, login_name, to_char(date_reviewed,'Mon. DD, YYYY'), review_text
FROM review
JOIN book_core ON review.book_id = book_core.core_id
JOIN books USING (book_id)
JOIN booknet_user ON reviewer = user_id
%s
WHERE review.book_id = %s
LIMIT %s OFFSET %s

Endpoint: /reviews/<rid> (review.html)
Information: Gets a review and shows details of that review.
Queries:
SELECT review_id, core_id, book_id, book_title, reviewer, login_name, to_char(date_reviewed,'Mon. DD, YYYY'), review_text
FROM review
JOIN books USING (book_id)
JOIN book_core USING (core_id)
JOIN booknet_user ON reviewer = user_id
WHERE review_id = %s

Endpoint: /reviews/add/<bid> (review_add_form.html)
Information: Adds a review to a book.
Queries:
INSERT INTO review (book_id, reviewer, review_text, date_reviewed)
VALUES(%s, %s, %s, current_timestamp)
RETURNING review_id

SELECT DISTINCT core_id,book_title, isbn, page_count, COALESCE(publisher_name,'Unknown'), book_description,
to_char(publication_date,'Mon. DD, YYYY') as publication_date,
to_char(publication_date,'MM/DD/YYYY') as publication_date_fmt,
COALESCE(cover_name,'_placeholder') as cover_name, AVG(rating) as avg_rating
FROM book_core

LEFT JOIN books USING (core_id)
LEFT JOIN ratings ON core_id = ratings.book_id
LEFT JOIN book_publisher ON core_id = book_publisher.book_id
LEFT JOIN publisher USING (publisher_id)
WHERE core_id = %s
GROUP BY core_id, book_title, book_description, cover_name, isbn, page_count, publication_date, publication_date_fmt, publisher_name

Endpoint: /reviews/ (review_list.html)
Information: Shows a list of reviews.
Queries:
Python:

```
total_pages = get_total_pages(cur, QUERIES['reviews_count'])
  query = QUERIES['select_reviews_where'] % ('', '%s', '%s')
  cur.execute(query, (amount, start))
  review_info = []
  for  review_id, core_id, book_id, book_title, reviewer, login_name, date_reviewed, review_text in cur:
      review_info.append({'core_id':core_id, 'id':review_id, 'book_id': book_id, 'book_title':book_title.decode('utf8', 'xmlcharrefreplace'), 'reviewer':reviewer, 'reviewer_name': login_name, 'date_reviewed': date_reviewed, 'review_text':review_text})
```

SQL:

```
SELECT review_id, book_core.core_id, book_id, book_title, reviewer, login_name, to_char(date_reviewed,'Mon. DD, YYYY'), review_text
      FROM review
      JOIN book_core ON review.book_id = book_core.core_id
      JOIN books USING (book_id)
      JOIN booknet_user ON reviewer = user_id
      %s
      LIMIT %s OFFSET %s
```

Endpoint: /dashboard/ (dashboard_overview.html)
Information: Shows the dashboard for a user.
Queries:

```
SELECT user_id, login_name, level_name, COALESCE(is_followed, FALSE) as is_followed, COUNT(DISTINCT book_list.book_id) as num_unique_list_books,
      COUNT(DISTINCT user_log.book_id) as num_total_books_read, COUNT(DISTINCT log_id) as num_unique_books_read, COUNT(DISTINCT review_id) as num_reviews
      FROM booknet_user
      LEFT JOIN review ON user_id = reviewer
      LEFT JOIN list USING (user_id)
      LEFT JOIN book_list USING (list_id)
      LEFT JOIN user_level USING (level_id)
      LEFT JOIN user_log ON user_id = reader
      LEFT JOIN  ( SELECT user_followed, is_followed FROM follow WHERE follower = %s ) is_followed_table ON user_id = user_followed
```

WHERE user_id = %s
GROUP BY user_id, login_name, level_name, is_followed

Endpoint: /dashboard/logs (dashboard_logs.html)
Information: Shows a user's logs on the dashboard.
Queries:
SELECT user_id, login_name, level_name, COALESCE(is_followed, FALSE) as is_followed, COUNT(DISTINCT book_list.book_id) as num_unique_list_books,
COUNT(DISTINCT user_log.book_id) as num_total_books_read, COUNT(DISTINCT log_id) as num_unique_books_read, COUNT(DISTINCT review_id) as num_reviews
FROM booknet_user
LEFT JOIN review ON user_id = reviewer
LEFT JOIN list USING (user_id)
LEFT JOIN book_list USING (list_id)
LEFT JOIN user_level USING (level_id)
LEFT JOIN user_log ON user_id = reader
LEFT JOIN (SELECT user_followed, is_followed FROM follow WHERE follower = %s) is_followed_table ON user_id = user_followed
WHERE user_id = %s
GROUP BY user_id, login_name, level_name, is_followed

Endpoint: /dashboard/lists (dashboard_lists.html)
Information: Shows lists on the user's dashboard.
Queries:
SELECT user_id, login_name, level_name, COALESCE(is_followed, FALSE) as is_followed, COUNT(DISTINCT book_list.book_id) as num_unique_list_books,
COUNT(DISTINCT user_log.book_id) as num_total_books_read, COUNT(DISTINCT log_id) as num_unique_books_read, COUNT(DISTINCT review_id) as num_reviews
FROM booknet_user
LEFT JOIN review ON user_id = reviewer
LEFT JOIN list USING (user_id)
LEFT JOIN book_list USING (list_id)
LEFT JOIN user_level USING (level_id)
LEFT JOIN user_log ON user_id = reader
LEFT JOIN (SELECT user_followed, is_followed FROM follow WHERE follower = %s) is_followed_table ON user_id = user_followed
WHERE user_id = %s
GROUP BY user_id, login_name, level_name, is_followed

Endpoint: /dashboard/reviews (dashboard_reviews.html)
Information: Shows reviews on the user's dashboard.
Queries:
SELECT user_id, login_name, level_name, COALESCE(is_followed, FALSE) as is_followed, COUNT(DISTINCT book_list.book_id) as num_unique_list_books,
COUNT(DISTINCT user_log.book_id) as num_total_books_read, COUNT(DISTINCT log_id) as num_unique_books_read, COUNT(DISTINCT review_id) as num_reviews
FROM booknet_user
LEFT JOIN review ON user_id = reviewer

```
        LEFT JOIN list USING (user_id)
        LEFT JOIN book_list USING (list_id)
        LEFT JOIN user_level USING (level_id)
        LEFT JOIN user_log ON user_id = reader
        LEFT JOIN  ( SELECT user_followed, is_followed FROM follow WHERE follower = %s )
is_followed_table ON user_id = user_followed
        WHERE user_id = %s
        GROUP BY user_id, login_name, level_name, is_followed
```

Endpoint: /dashboard/followers
Information: Shows followers on user's dashboard.

Endpoint: /dashboard/following
Information: Shows who the user is following, on the dashboard.
Queries:

```
  SELECT user_id, login_name, level_name, COALESCE(is_followed, FALSE), COUNT(DISTINCT
review_id) as num_reviews, COUNT(DISTINCT list_id) as num_lists
        FROM booknet_user
        JOIN review ON user_id = reviewer
        JOIN list USING (user_id)
        JOIN user_level USING (level_id)
        LEFT JOIN  follow ON user_id = user_followed
        WHERE follower = %s
        GROUP BY user_id, login_name, level_name, is_followed

SELECT review_id, book_id, to_char(date_reviewed,'Mon. DD, YYYY') as date_reviewed, book_title,
review_text
            FROM review
            JOIN book_core ON book_id = core_id
            WHERE reviewer = %s AND date_reviewed > NOW() - INTERVAL '60 days'
```

Endpoint: /profile
Information: Shows user's profile.

Endpoint: /dashboard/moderation (dashboard_moderation.html)
Information: Shows moderation options, if the user has them available, on the dashboard.
Queries:

```
SELECT request_id, request_on_book_id, request_type, book_id, request_text, book_title, login_name,
user_id, to_char(date_requested,'Mon. DD, YYYY') as date_requested
        FROM request
        JOIN request_on_book USING (request_id)
        JOIN books USING (book_id)
        JOIN book_core USING (core_id)
        JOIN booknet_user USING (user_id)
        WHERE status = 'Awaiting Review'
```

Endpoint: /dashboard/presentation (dashboard_presentation.html)
Information: The section of the dashboard we used during the final presentation. Lists a few notable queries from our various files for the site, also has links to the github we are using.

Endpoint: /dashboard/approve/<request_id>
Information: A function that allows a moderator to approve a request from a user.
Queries:

```
SELECT request_id, request_on_book_id, request_type, book_id, request_text, book_title, login_name,
user_id, to_char(date_requested,'Mon. DD, YYYY') as date_requested
        FROM request
        JOIN request_on_book USING (request_id)
        JOIN books USING (book_id)
        JOIN book_core USING (core_id)
        JOIN booknet_user USING (user_id)
        WHERE request_id = %s

  UPDATE request SET
            status = '0'
            WHERE request_id = %s
            RETURNING status
```

Endpoint: /dashboard/reject/<request_id>
Information: A function to allow a moderator to reject a user's request.

Endpoint: /user/<uid> (user_profile.html)
Information: Displays user's profile information.
Queries:

```
   SELECT user_id, login_name, level_name, COALESCE(is_followed, FALSE) as is_followed,
COUNT(DISTINCT book_list.book_id) as num_unique_list_books,
        COUNT(DISTINCT user_log.book_id) as num_total_books_read, COUNT(DISTINCT log_id) as
num_unique_books_read, COUNT(DISTINCT review_id) as num_reviews
        FROM booknet_user
        LEFT JOIN review ON user_id = reviewer
        LEFT JOIN list USING (user_id)
        LEFT JOIN book_list USING (list_id)
        LEFT JOIN user_level USING (level_id)
        LEFT JOIN user_log ON user_id = reader
        LEFT JOIN  ( SELECT user_followed, is_followed FROM follow WHERE follower = %s )
is_followed_table ON user_id = user_followed
        WHERE user_id = %s
        GROUP BY user_id, login_name, level_name, is_followed
```

Endpoint: /user/follow/<uid>
Information: The function to follow a user.
Queries:

```
   SELECT follow_id
        FROM follow
        WHERE user_followed = %s AND follower = %s

  INSERT INTO follow (follower, user_followed, date_followed, is_followed)
```

```
        VALUES(%s, %s, current_timestamp, True)
        RETURNING follow_id
```

Endpoint: /user/unfollow/<uid>
Information: A function for unfollowing a user.
Queries:

```
 DELETE FROM follow
      WHERE user_followed = %s AND follower = %s
```

Endpoint: /users (users.html)
Information: A list of the site's users.
Queries:

```
 SELECT user_id, login_name, level_name, COALESCE(is_followed, FALSE), COUNT(DISTINCT
review_id) as num_reviews, COUNT(DISTINCT list_id) as num_lists
      FROM booknet_user
      JOIN review ON user_id = reviewer
      JOIN list USING (user_id)
      JOIN user_level USING (level_id)
      LEFT JOIN  ( SELECT user_followed, is_followed FROM follow WHERE follower = %s )
is_followed_table ON user_id = user_followed
      GROUP BY user_id, login_name, level_name, is_followed
      LIMIT %s OFFSET %s

SELECT COUNT(log_id) as num_books_read
         FROM user_log
         WHERE reader = %s
```

Endpoint: /user/login (login.html)
Information: The login page.
Queries:

```
SELECT user_id, login_name, password
     FROM booknet_user
     WHERE login_name = %s
```

Endpoint: /user/registration (register.html)
Information: The user registration page. Flask handles this part.

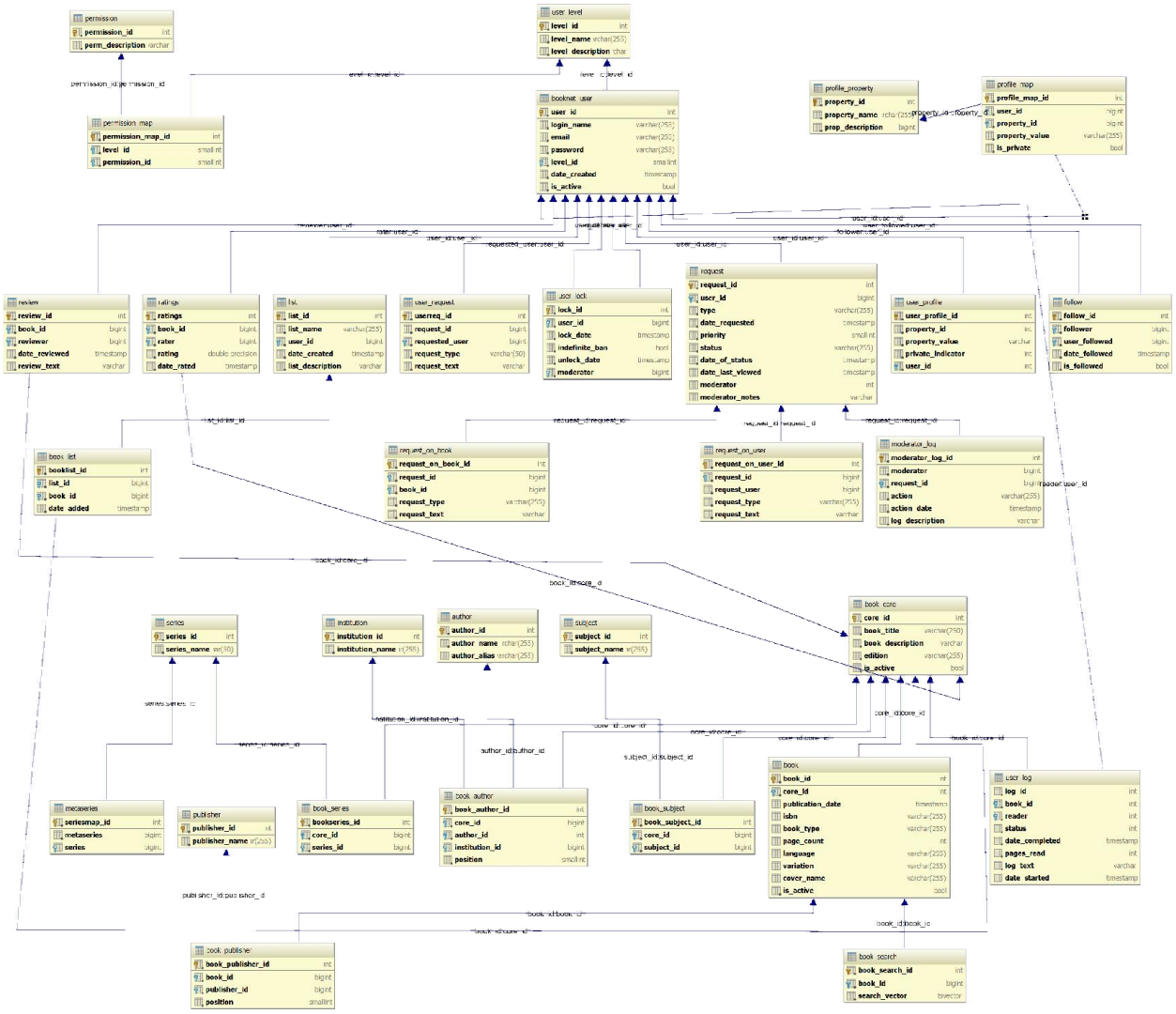
Endpoint: /user/logout
Information: Flask function for logging out the user.

Endpoint: /search (book_search_results.html)
Information: The book search function.
Queries:
Python:

```
order_by = 'ORDER BY ts_rank(search_vector, plainto_tsquery(\'%s\')) DESC'
   try:
      order_by += ', '+SORTING[sorting]+' '+SORT_DIRECTION[sort_direction]
   except KeyError:
      pass
```

```
    total_pages = get_total_pages(cur, QUERIES['books_search_count'] % search_query)

    query = QUERIES['select_books_where'] % ('', 'JOIN book_search USING (book_id)',
                                        ' AND search_vector @@ plainto_tsquery(\'%s\')', 'search_vector,',
order_by,'%s','%s')
    print query
    cur.execute(query % (search_query, search_query,'%s','%s'), (amount, start))



    book_info = []
    # print "Retrieved %s book rows..." % cur.rowcount
    for core_id, book_title, description, isbn, page_count, cover_name, avg_rating, num_readers in cur:
        # if not cover_name:
        #     cover_name = '_placeholder'
        if avg_rating:
            discrete_rating = round(avg_rating*2) / 2
        else:
            discrete_rating = 0
        try:
            avg_rating = round(avg_rating,2)
        except TypeError:
            # not a float / null
            pass
        book_info.append({'core_id':core_id, 'title': str(book_title).decode('utf8', 'xmlcharrefreplace'),
                    'cover_name': cover_name, 'authors': [], 'subjects': [], 'isbn': isbn, 'num_pages':
page_count,
                    'num_readers': num_readers, 'avg_rating': avg_rating, 'discrete_rating': discrete_rating,
'user_rating': None})
    for book in book_info:
        cur.execute('''
        SELECT author_name
        FROM author JOIN authorship USING (author_id)
        WHERE core_id = %s
        ''', (book['core_id'],))
        author_info = []
        for author_name in cur:
            author_info.append(str(author_name[0]).decode('utf8', 'xmlcharrefreplace'))

        book['authors'] = author_info
        # print book['authors']
        cur.execute('''
        SELECT subject_name
        FROM subject_genre JOIN book_categorization USING (subject_id)
        WHERE core_id = %s
        ''', (book['core_id'],))
        # subject_info = []
        # print cur.fetchone()
```

```
        for subject_name in cur:
            book['subjects'].append(subject_name[0].decode('utf8', 'xmlcharrefreplace'))
        # book['subjects'] = subject_info
        if (user_id):
            cur.execute('''
            SELECT rating
            FROM ratings
            WHERE book_id = %s AND rater = %s
            ''', (book['core_id'], user_id))
            if cur.rowcount > 0:
                book['user_rating'] = cur.fetchone()[0]
        # print book['subjects']


    return total_pages, book_info

SQL:
SELECT %s core_id, book_title, book_description, isbn, COALESCE(page_count,0),
COALESCE(cover_name,'_placeholder') as cover_name, AVG(rating) as avg_rating,
COUNT(DISTINCT log_id) as num_readers
        FROM book_core
        LEFT JOIN books USING (core_id)
        JOIN book_search USING (book_id)
        %s
        LEFT JOIN ratings ON core_id = ratings.book_id
        LEFT JOIN user_log ON core_id = user_log.book_id
        LEFT JOIN book_categorization USING (core_id)

        WHERE books.is_active = TRUE AND search_vector @@ plainto_tsquery(\'search_vector,\')
        GROUP BY %s core_id, book_title, book_description, cover_name, isbn, page_count,
publication_date
        %s
        LIMIT %s OFFSET %s
```


permisson
permission_id int
perm_description varchar
user_level
level_id int
level_name varchar(255)
level_description char
permission_map
permission_map_id int
level_id small int
permission_id small int
booknet_user
user_id int
login_name varchar(255)
email varchar(255)
password varchar(255)
level_id small int
date_created timestamp
is_active bool
profile_property
property_id int
property_name varchar(255)
prop_description bigint
profile_map
profile_map_id int
user_id big int
property_id big int
property_value varchar(255)
is_private bool
review
review_id int
book_id bigint
reviewer bigint
date_reviewed timestamp
review_text varchar
ratings
ratings int
book_id bigint
rater bigint
rating double precision
date_rated timestamp
list
list_id int
list_name varchar(255)
user_id bigint
date_created timestamp
list_description varchar
user_request
userreq_id int
request_id bigint
requested_user bigint
request_type varchar(30)
request_text varchar
user_lock
lock_id int
user_id bigint
lock_date timestamp
indefinite_ban bool
unlock_date timestamp
moderator bigint
request
request_id int
user_id bigint
type varchar(255)
date_requested timestamp
priority small int
status varchar(255)
date_of_status timestamp
date_last_viewed timestamp
moderator int
moderator_notes varchar
user_profile
property_id int
property_value varchar
private_indicator int
user_id int
follow
follow_id int
follower bigint
user_followed bigint
date_followed timestamp
is_followed bool
booklist
booklist_id int
list_id bigint
book_id bigint
date_added timestamp
request_on_book
request_on_book_id int
request_id bigint
book_id bigint
request_type varchar(255)
request_text varchar
request_on_user
request_on_user_id int
request_id bigint
request_user bigint
request_type varchar(255)
request_text varchar
moderator_log
moderator_log_id int
moderator bigint
request_id bigint
action varchar(255)
action_date timestamp
log_description varchar
book_core
core_id int
book_title varchar(250)
book_description varchar
edition varchar(255)
is_active bool
series
series_id int
series_name varchar(90)
institution
institution_id int
institution_name varchar(255)
author
author_id int
author_name varchar(255)
author_alias varchar(255)
subject
subject_id int
subject_name varchar(255)
book
book_id int
core_id int
publication_date timestamp
isbn varchar(255)
book_type varchar(255)
page_count int
language varchar(255)
variation varchar(255)
cover_name varchar(255)
is_active bool
user_log
log_id int
book_id int
reader int
status int
date_completed timestamp
pages_read int
log_text varchar
date_started timestamp
metaseries
seriesmap_id int
metaseries bigint
series bigint
publisher
publisher_id int
publisher_name varchar(250)
book_series
bookseries_id int
core_id bigint
series_id bigint
book_author
book_author_id int
core_id bigint
author_id int
institution_id bigint
position smallint
book_subject
book_subject_id int
core_id bigint
subject_id bigint
book_publisher
book_publisher_id int
book_id bigint
publisher_id bigint
position smallint
book_search
book_search_id int
book_id bigint
search_vector tsvector