Information Management & Systems Engineering

Milestone 2: Online shop [Presentation Slides]

Bryan Yi Jue Tan 11930138
Marwa Wahdan 00727689

1. NoSQL Design

1.1. Compare the design of the RDBS data model to the NoSQL design

SQL :
CREATE TABLE user (
user_id INT AUTO_INCREMENT PRIMARY KEY,
username VARCHAR(50) UNIQUE,
email VARCHAR(50) UNIQUE,
password VARCHAR(50)
);

NoSQL:
example_user = {
“user_id": 1,
"username": "Julia Barrett",
"email": "GreenApple88@gmail.com",
“password": "Juliabar"
}

SQL:
CREATE TABLE customer (
phone_number BIGINT,
delivery_address VARCHAR(100),
bonus_points INT,
user_id INT AUTO_INCREMENT,
FOREIGN KEY (user_id) REFERENCES user(user_id));

NoSQL:
example_customer= {
“phone_number": 06817198109,
"delivery_address": "Koloniestrasse 01a, 9.OG, 9812,
Krumbach, Niederösterreich, Austria",
“bonus_points": 334,
"user_id": 1,
}

2. NoSQL Design

1.1. Compare the design of the RDBS data model to the NoSQL design

SQL:
CREATE TABLE item (
item_id INT PRIMARY KEY,
description VARCHAR(200),
price DECIMAL(10,2),
category VARCHAR(200)
);

NoSQL:
example_item = {
"item_id": 8,
"description": "Swimwear",
"price": 23.49,
"category": "Women" }

SQL:
CREATE TABLE review (
review_id INT AUTO_INCREMENT PRIMARY KEY,
publish_timestamp TIMESTAMP,
title VARCHAR(200),
description VARCHAR(200),
rating INT,
user_id INT,
item_id INT,
FOREIGN KEY (user_id) REFERENCES user(user_id),
FOREIGN KEY (item_id) REFERENCES item(item_id)
);

NoSQL:
example_review= { "review_id":7,
"publish_timestamp": (2023-03-13 04:53:39),
"title": "very Big ",
"description": "very nice but a bit big. ",
"rating":4,
"user_id":11,
"item_id":43 }

2. NoSQL Design

1.1. Compare the design of the RDBS data model to the NoSQL design

SQL:
CREATE TABLE comment (
comment_id INT PRIMARY KEY,
publish_timestamp TIMESTAMP,
content VARCHAR(200), review_id INT,
user_id INT,
FOREIGN KEY (review_id) REFERENCES
review(review_id),
FOREIGN KEY (user_id) REFERENCES user(user_id)
);

NoSQL:
example_comment= {
"comment_id":7,
"publish_timestamp": (2023-04-04 20:07:43),
"content": "I took a size smaller ",
"review_id": 7,
"user_id":11,
}

SQL :
CREATE TABLE orderItem (
order_id INT,
item_id INT,
FOREIGN KEY (order_id) REFERENCES orders(order_id),
FOREIGN KEY (item_id) REFERENCES item (item_id)
);

NoSQL :
example_orderItem= {
"order_id":7,
"item_id":43,
}

2. NoSQL Design

1.1. Compare the design of the RDBS data model to the NoSQL design

SQL:

```
CREATE TABLE orders (
order_id INT AUTO_INCREMENT PRIMARY KEY,
quantity INT,
total_price INT,
delivery_date DATE,
user_id INT,
FOREIGN KEY (user_id) REFERENCES user(user_id)
);
```

NoSQL :

```
example_order= {
"order_id": "10",
"quantity": 2,
"total_price": 38.99, "delivery_date": 20.03.2023,
"user_id": 9,
"item":[ {
  “item_id": 9,
  "description": "Coat",
  "price": 27.99,
  "category": "Women",
 }, {
  "item_id": 8,
  "description": "Swimwear",
  "price": 23.49,
  "category": "Women",
 }
] }
```

2.2 Performance implications of the chosen NoSQL design

Main Use Case 1 (Marwa Wahdan): Register a new customer

```python
@app.route('/use_case1', methods=['POST'])
def use_case1():

    username_signup = request.form['username']
    password_signup = request.form['password']
    if len(password_signup) < 6:
        return "weak password"
    email = request.form['email']
    user_type = request.form['user_type']

    if is_migrated == 0:
        cur = db.cursor()
        cur.execute('SELECT * FROM user where username=%s', (username_signup,))
        result = cur.fetchall()
        print(len(result))
        if len(result) > 0:
            return 'duplicate user name '
        else:
            cur.execute('SELECT * FROM user where email=%s', (email,))
            result_email = cur.fetchall()
            print(len(result_email))
            if len(result_email) != 0:
                return 'duplicate mail'
            else:
                cur.execute("INSERT INTO user (username,email, password) VALUES (%s, %s, %s)",
                            (username_signup, email, password_signup))

                if user_type == 'merchant':
                    website = request.form['website']
                    cur.execute(
                        "INSERT INTO merchant (merchant_name,website) VALUES (%s, %s)", (username_signup, website))

                elif user_type == 'customer':
                    phone_number = request.form['phone_number']
                    delivery_address = request.form['delivery_address']
                    bonus = 0
                    cur.execute("INSERT INTO customer (phone_number,delivery_address,bonus_points) VALUES (%s, %s,%s)", (
                        phone_number, delivery_address, bonus))
                cur.close()
    else:
        collection = mongo_db.user
        last_document = collection.find().sort('user_id', -1).limit(1)
        last_user_id = last_document[0]['user_id']
        result_1 = collection.find_one({'username': username_signup})
        if result_1:
            return 'duplicate user name'
        else:
            result_2 = collection.find_one({'email': email})
            if result_2:
                return 'duplicate email'
            else:
                new_user = {
                    'username': username_signup,
                    'email': email,
                    'password': password_signup,
                    'user_id': last_user_id+1
                }
                result = collection.insert_one(new_user)

                if user_type == 'merchant':
                    website = request.form['website']
                    new_merchant = {
                        'merchant_name': username_signup, 'website': website}
                    collection = mongo_db.merchant
                    result = collection.insert_one(new_merchant)

                elif user_type == 'customer':
                    phone_number = request.form['phone_number']
                    delivery_address = request.form['delivery_address']
                    bonus = 0
                    new_customer = {
                        'phone_number': phone_number,
                        'delivery_address': delivery_address,
                        'bonus_points': bonus}
                    collection = mongo_db.customer
                    result = collection.insert_one(new_customer)

    return render_template('success.html', message="Signed up successfully")
```

2.2 Performance implications of the chosen NoSQL design

Main Use Case 2 (Bryan Yi Jue Tan): Add item to the order

```python
@app.route('/use_case2', methods=['POST'])
def use_case2():
    global is_migrated
    id = request.form['id']
    quantity = request.form['anzahl']
    current_date = date.today()
    formatted_date = current_date.strftime("%Y-%m-%d")
    if is_migrated == 0:
        cur = db.cursor(buffered=True)
        cur.execute('SELECT user_id FROM user where username=%s', (username,))
        user_id = cur.fetchone()[0]
        cur.execute("select price from item where item_id=%s", (id,))
        price = cur.fetchall()[0][0]
        price = float(price)
        print(type(quantity), type(price))

        cur.execute("INSERT INTO orders (quantity,total_price,delivery_date,user_id) VALUES (%s, %s, %s,%s)",
                    (quantity, int(quantity)*price, formatted_date, user_id))
        cur.execute("select max(order_id) from orders")
        order_id_needed = cur.fetchall()
        order_id_needed = str(order_id_needed[0][0])
        cur.execute(
            "INSERT INTO orderItem (order_id,item_id) VALUES (%s, %s)", (order_id_needed, id))
        db.commit()
        totalprices = int(quantity)*price
    else:
        collection_user = mongo_db.user
        user = collection_user.find_one({'username': username}, {'user_id': 1})
        if user:
            user_id = user['user_id']
        collection_item = mongo_db.item
        item_col = collection_item.find_one(
            {'item_id': int(id)}, {'description': 1, 'price': 1})
        if item_col:
            item_desc = item_col['description']
            item_price = item_col['price']
            print(item_desc, item_price, quantity)
            totalprices = item_price*int(quantity)
        collection_order = mongo_db.orders
        last_document = collection_order.find().sort('order_id', -1).limit(1)
        last_order_id = last_document[0]['order_id']
        new_order = {
            'order_id': last_order_id+1,
            'item_id': id,
            'description': item_desc,
            'quantity': quantity,
            'user_id': int(user_id),
            "delivery_date": formatted_date,
            "total_price": float(totalprices)
        }
        result = collection_order.insert_one(new_order)
    return "delivery is on your way and the total price is " + str(totalprices)
```

2.3 Compare the SQL Select statements to the MongoDB query statements

Report 1 (Marwa Wahdan): Top Items

```python
@app.route('/report1', methods=['GET', 'POST'])
def report1():
    if request.method == 'POST':
        selected_category = request.form.get('category')
        if selected_category == 'All':
            if is_migrated == 0:
                cursor = db.cursor()
                cursor.execute("""SELECT top_items.item_id, item.description, item.category, top_items.count AS item_count
                               FROM (
                                   SELECT item_id, COUNT(item_id) AS count
                                   FROM review
                                   GROUP BY item_id
                                   ORDER BY count DESC
                                   LIMIT 10
                               ) AS top_items
                               JOIN item ON top_items.item_id = item.item_id
                               LIMIT 10;""")
                order = cursor.fetchall()
            else:
                review_collection = mongo_db.review
                pipeline = [
                    {'$group': {
                        '_id': '$item_id',
                        'count': {'$sum': 1}
                    }},
                    {'$lookup': {
                        'from': 'item',
                        'localField': '_id',
                        'foreignField': 'item_id',
                        'as': 'item'
                    }},
                    {'$unwind': '$item'},
                    {'$project': {
                        '_id': 0,
                        'item_id': '$_id',
                        'description': '$item.description',
                        'category': '$item.category',
                        'count': 1
                    }},
                    {'$sort': {'count': -1}},
                    {'$limit': 10}
                ]
                result = review_collection.aggregate(pipeline)
                order = [[item['item_id'], item['description'],
                          item['category'], item['count']] for item in result]
    else:
        if is_migrated == 0:
            cursor = db.cursor()
            cursor.execute("""SELECT top_items.item_id, item.description, item.category, top_items.count AS item_count
                           FROM (
                               SELECT item_id, COUNT(item_id) AS count
                               FROM review
                               GROUP BY item_id
                               ORDER BY count DESC
                           ) AS top_items
                           JOIN item ON top_items.item_id = item.item_id
                           WHERE item.category = %s
                           LIMIT 10;""", (selected_category,))
            order = cursor.fetchall()
        else:
            review_collection = mongo_db.review
            pipeline = [
                {'$group': {
                    '_id': '$item_id',
                    'count': {'$sum': 1}
                }},
                {'$lookup': {
                    'from': 'item',
                    'localField': '_id',
                    'foreignField': 'item_id',
                    'as': 'item'
                }},
                {'$unwind': '$item'},
                {'$project': {
                    '_id': 0,
                    'item_id': '$_id',
                    'description': '$item.description',
                    'category': '$item.category',
                    'count': 1
                }},
                {'$match': {'category': selected_category}},
                {'$sort': {'count': -1}},
                {'$limit': 10}
            ]
            result = review_collection.aggregate(pipeline)
            order = [[item['item_id'], item['description'],
                      item['category'], item['count']] for item in result]
    return render_template('report1.html', employees=order)
else:
    return render_template('report1.html', employees=[])
```

2.3 Compare the SQL Select statements to the MongoDB query statements

Report 2 (Bryan Yi Jue Tan): Top Users

```python
@app.route('/report2', methods=['GET', 'POST'])
def report2():
    global user_id, username
    if is_migrated == 0:
        cursor = db.cursor()
        cursor.execute("""
            SELECT u.user_id, u.username, MAX(c.bonus_points), COUNT(DISTINCT r.review_id)
            FROM user u
            JOIN customer c ON c.user_id = u.user_id
            JOIN review r ON c.user_id = r.user_id
            WHERE c.bonus_points > 100
            GROUP BY u.user_id, u.username
            ORDER BY MAX(c.bonus_points) DESC
        """)
        order = cursor.fetchall()
    else:
        customer_collection = mongo_db.user
        pipeline = [
            {'$lookup': {
                'from': 'user',
                'localField': 'user_id',
                'foreignField': 'user_id',
                'as': 'user_info'
            }},
            {'$lookup': {
                'from': 'customer',
                'localField': 'user_id',
                'foreignField': 'user_id',
                'as': 'customer_info'
            }},
            {'$lookup': {
                'from': 'review',
                'localField': 'user_id',
                'foreignField': 'user_id',
                'as': 'reviews'
            }},
            {'$group': {
                '_id': '$user_id',
                'review_count': {'$sum': {'$size': '$reviews'}},
                'username': {'$first': {'$arrayElemAt': ['$user_info.username', 0]}},
                'bonus_points': {'$first': {'$arrayElemAt': ['$customer_info.bonus_points', 0]}}
            }},
            {'$match': {'bonus_points': {'$gt': 100}}},
            {'$sort': {'bonus_points': -1
                       }},
            {'$project': {
                'user_id': '$_id',
                'username': 1,
                'review_count': 1,
                'bonus_points': 1,
                '_id': 0
            }}
        ]
        result = customer_collection.aggregate(pipeline)
        order = [[item['user_id'], item['username'],
                  item['bonus_points'], item['review_count']] for item in result]
    return render_template('report2.html', users=order)
```