CPSC 449- Section 01: Web Back-End Engineering

Project - 3 Polyglot Persistence Report
Fall 2023

Group Members:

Sanjyot Satvi
Divya Tanwar
Nathan Storm
Ethan Davidson
Anurag Ganji

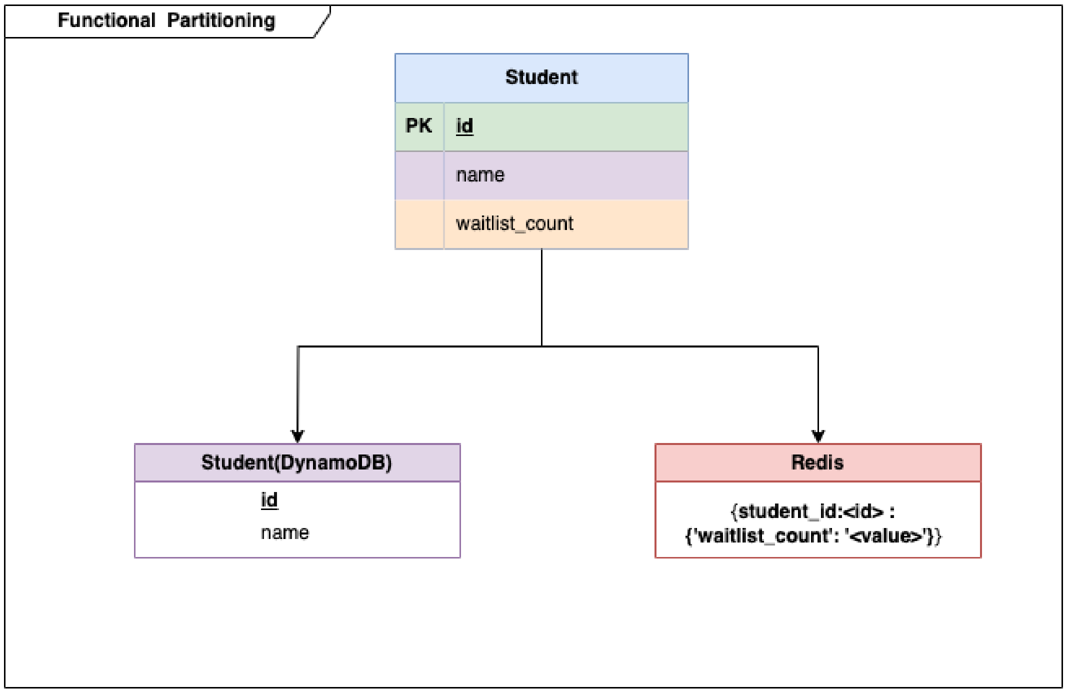
GitHub Repo: sanjyot242/backend-project3 (github.com)

Install and configure databases, tools, and libraries

`run `sh ./bin/install.sh``

Partition the data for the enrollment service

Functional Partitioning:



For Redis No-SQL database, key is: **student_id:<id>** and mapping fields for corresponding key is: **{'waitlist_count': '<value>'}**

Redis Client:

```
redis_client = redis.StrictRedis(host='localhost', port=6379, db =0,
decode_responses=True)
```

Redis Queries:

```
from redis import Redis
def get_waitlist_count(student_id:int, redis_client: Redis):
    key = f'student_id:{student_id}'
    waitlist_count = str(redis_client.hget(key, "waitlist_count"))
    if waitlist_count is None or waitlist_count == 'None':
        return 0
    return int(waitlist_count)

def increment_wailist_count(student_id:int, redis_client: Redis):
  key = f'student_id:{student_id}'
  waitlist_count = 0
  current_waitlist_count =  str(redis_client.hget(key, "waitlist_count"))
  if current_waitlist_count is not None and current_waitlist_count != 'None':
    current_waitlist_count = int(current_waitlist_count) + 1
    redis_client.hset(key, mapping={'waitlist_count': current_waitlist_count})
  else:
    redis_client.hset(key, mapping={'waitlist_count': waitlist_count + 1})

def decrement_wailist_count(student_id:int, redis_client: Redis):
  key = f'student_id:{student_id}'
  current_waitlist_count =  str(redis_client.hget(key, "waitlist_count"))
  current_waitlist_count = int(current_waitlist_count) - 1
  redis_client.hset(key, mapping={'waitlist_count': current_waitlist_count})
```

DynamoDB Design:

Client:

```
dynamodb = boto3.resource('dynamodb',endpoint_url='http://localhost:5500)
```

Class Table:

```
def create_class_table():
    # Define the table structure
    table = dynamodb.create_table(
        TableName='class',
        KeySchema=[
            {'AttributeName': 'id', 'KeyType': 'HASH'}  # Primary key
        ],
        AttributeDefinitions=[
            {'AttributeName': 'id', 'AttributeType': 'N'},  # Attribute type N for number
            {'AttributeName': 'department_id', 'AttributeType': 'N'},
            {'AttributeName': 'instructor_id', 'AttributeType': 'N'},
            {'AttributeName': 'available_slot', 'AttributeType': 'N'},
            {'AttributeName': 'constantGSI', 'AttributeType': 'S'},
        ],
        ProvisionedThroughput={
            'ReadCapacityUnits': 10,
            'WriteCapacityUnits': 10
        },
        GlobalSecondaryIndexes=[
            {
                'IndexName': 'DepartmentIndex',
                'KeySchema': [
                    {'AttributeName': 'department_id', 'KeyType': 'HASH'}
                ],
                'Projection': {
                    'ProjectionType': 'ALL'
                },
                'ProvisionedThroughput': {
                    'ReadCapacityUnits': 10,
                    'WriteCapacityUnits': 10
                }
            },
            {
                'IndexName': 'InstructorIndex',
                'KeySchema': [

                    {'AttributeName': 'instructor_id', 'KeyType': 'HASH'}
                ],
                'Projection': {
                    'ProjectionType': 'ALL'
                },
                'ProvisionedThroughput': {
                    'ReadCapacityUnits': 10,
                    'WriteCapacityUnits': 10
                }
            },
            {
                'IndexName': 'AvailableSlotsIndex',
                'KeySchema': [
                    {'AttributeName': 'constantGSI', 'KeyType': 'HASH'},
                    {'AttributeName': 'available_slot', 'KeyType': 'RANGE'}
                ],
                'Projection': {
                    'ProjectionType': 'ALL'
                },
                'ProvisionedThroughput': {
                    'ReadCapacityUnits': 10,
                    'WriteCapacityUnits': 10
                }
            }
        ]
    )
    table.meta.client.get_waiter('table_exists').wait(TableName='class')
    print("Table created successfully.")
```

Department Table:

```
def create_department_table():
    table = dynamodb.create_table(
        TableName='department',
        KeySchema= [
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'  # Partition key
        }
    ],
    AttributeDefinitions = [
        {
            'AttributeName': 'id',
            'AttributeType': 'N'  # Number
        }
        # If you need to use 'id' as a GSI in another table, define the GSI here
    ],
    ProvisionedThroughput= {
        'ReadCapacityUnits': 5,
        'WriteCapacityUnits': 5
    }
    )
    table.meta.client.get_waiter('table_exists').wait(TableName='department')
    print("Table created successfully.")
```

Instructor Table:

```
def create_instructor_table():
    table = dynamodb.create_table(
        TableName='instructor',
        KeySchema= [
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'  # Partition key
        }
    ],
    AttributeDefinitions = [
        {
            'AttributeName': 'id',
            'AttributeType': 'N'  # Number
        }
    ],
    ProvisionedThroughput= {
        'ReadCapacityUnits': 5,
        'WriteCapacityUnits': 5
    }
    )
    table.meta.client.get_waiter('table_exists').wait(TableName='instructor')
    print("Table created successfully.")
```

Student Table:

```
def create_student_table():
    table = dynamodb.create_table(
        TableName='student',
        KeySchema= [
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'  # Partition key
        }
    ],
    AttributeDefinitions = [
        {
            'AttributeName': 'id',
            'AttributeType': 'N'  # Number
        }
    ],
    ProvisionedThroughput= {
        'ReadCapacityUnits': 5,
        'WriteCapacityUnits': 5
    }
    )
    table.meta.client.get_waiter('table_exists').wait(TableName='student')
    print("Table created successfully.")
```

Dropped Table:

```
def create_dropped_table():
    table = dynamodb.create_table(
        TableName='dropped',
        KeySchema= [
        {
            'AttributeName': 'class_id',
            'KeyType': 'HASH'  # Partition key
        },
        {
            'AttributeName': 'student_id',
            'KeyType': 'RANGE'  # Sort key
        }
    ],
    AttributeDefinitions=[
        {
            'AttributeName': 'class_id',
            'AttributeType': 'N'  # Number
        },
        {
            'AttributeName': 'student_id',
            'AttributeType': 'N'  # Number
        }
    ],
    ProvisionedThroughput= {
        'ReadCapacityUnits': 5,
        'WriteCapacityUnits': 5
    }
    )
    table.meta.client.get_waiter('table_exists').wait(TableName='dropped')
    print("Table created successfully.")
```

Enrollment Table:

```
def create_enrollment_table():
    table = dynamodb.create_table(
    TableName= "enrollment",
        KeySchema= [
            {
                'AttributeName': 'student_id',
                'KeyType': 'HASH'  # Partition key
            },
            {
                'AttributeName': 'class_id',
                'KeyType': 'RANGE'  # Sort key
            }
        ],
        AttributeDefinitions= [
            {
                'AttributeName': 'class_id',
                'AttributeType': 'N'  # Number
            },
            {
                'AttributeName': 'student_id',
                'AttributeType': 'N'  # Number
            }
            # Additional attributes can be defined here if needed for secondary indexes
        ],
        ProvisionedThroughput= {
            'ReadCapacityUnits': 5,
            'WriteCapacityUnits': 5
        }
    )
    table.meta.client.get_waiter('table_exists').wait(TableName='enrollment')
    print("Table created successfully.")
```

Testing

We tested all the endpoints after removing SQLite database and tested successfully all the endpoints with DynamoDB and Redis database.