QUESTION 1

Due to time constraints (full-time work/busy weekend plans) this solution is not as optimised as I would like. The error handling could be improved greatly. Currently there are no custom exception, only generic ones. Also any error will stop the program rather than handling it.

The ID's could be improved, currently they are not unique nor auto-increment and a developer could make an error typing them in manually.

As for 1.2 and 1.3 I have assumed 'reusable method' just applies to a method that can be used for multiple bank accounts. However, it could be interpreted to mean one which can be used with different objects. In which case, the method would be pulled out the 'BankBalanceSheet' class and generics would be used with an interface implemented. The interface would ensure any object methods used within the method would be available to any object that can be passed in.

I also wasn't certain about the 'liability value' so I just made it a variable within the BankBalanceSheet object class.

I also haven't set any unnecessary getters/setters relevant to completing the problem, but again this is a quick fix - I was just really pushed for time.

There is also no test cases but the program should give you the results use are looking for!

QUESTION 2

NOTE: Due to time constraints of trying to get this done while working full time these queries have not been tested against an actual database.

SQL QUERY 1

```
SELECT start_date, COUNT(*) AS new_accounts FROM account_Details
WHERE city = 'London'
GROUP BY start_date;
```

SQL QUERY 2

```
SELECT account_number, daily_usage_mins FROM daily_activity
WHERE daily_usage_mins > 4
```

```
ORDER BY daily_usage_mins DESC;
```

SQL QUERY 3

```
SELECT COUNT(*) FROM account_details INNER JOIN daily_activity
ON account_details.account_number = daily_activity.account_number
WHERE city = 'Glasgow'
AND start_date = '2019-12-01'
AND date > '2019-12-10';
```

QUESTION 3

NOTE: This could have better validation with more time.

JSON SCHEMA

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "title": "Account",
  "description": "A user's account information",
  "type": "object",
  "properties": {
    "account_number": {
      "description": "A user's account number",
      "type": "string",
      "pattern": "^\\d+$",
      "minLength": 8,
      "maxLength": 8
    },
    "start_date": {
      "description": "Date user created account",
      "type": "string",
      "format": "date"
    },
    "city": {
      "description": "A user's city",
      "type": "string"
    },
```

```
    "daily_activity": {
      "daily_usage_mins": {
        "description": "User's daily usage in minutes",
        "type": "integer"
      },
      "date": {
        "description": "User's last activity date",
        "type": "string",
        "format": "date"
      },
      "required": [
        "daily_usage_mins",
        "date"
      ]
    }
  },
  "required": [
    "account_number",
    "start_date",
    "city",
    "daily_activity"
  ]
}
```

JSON INPUT EXAMPLE

```
{
 "account_number": "34353342",
  "start_date": '2023-02-01',
  "city": "Glasgow",
   "daily_activity" : {
        "daily_usage_mins": 43,
         "date": '2018-03-01'
     }
}
```

Advantages and Disadvantages of NoSQL

+ Handles large loads of structured, semi-structured, and unstructured data. SQL only allows for structured data.
+ Agile sprints, quick iteration, and frequent code pushes are all advantages of its unstructured nature.
+ Efficient, scale-out architecture instead of expensive, single architecture. This advantage has already been mentioned but not in terms of cost effectiveness.
- Not ACID compliant - NoSQL provides flexibility and speed at the cost of data integrity
- SQL better for structured data
- Due to the CAP theorem, a noSQL database can be have high consistency or high availability but not both. Either one being a huge sacrifice for a banking system.