**Appendix 1 - CET341 Assignment 2 Structure Template**

**Task One:**

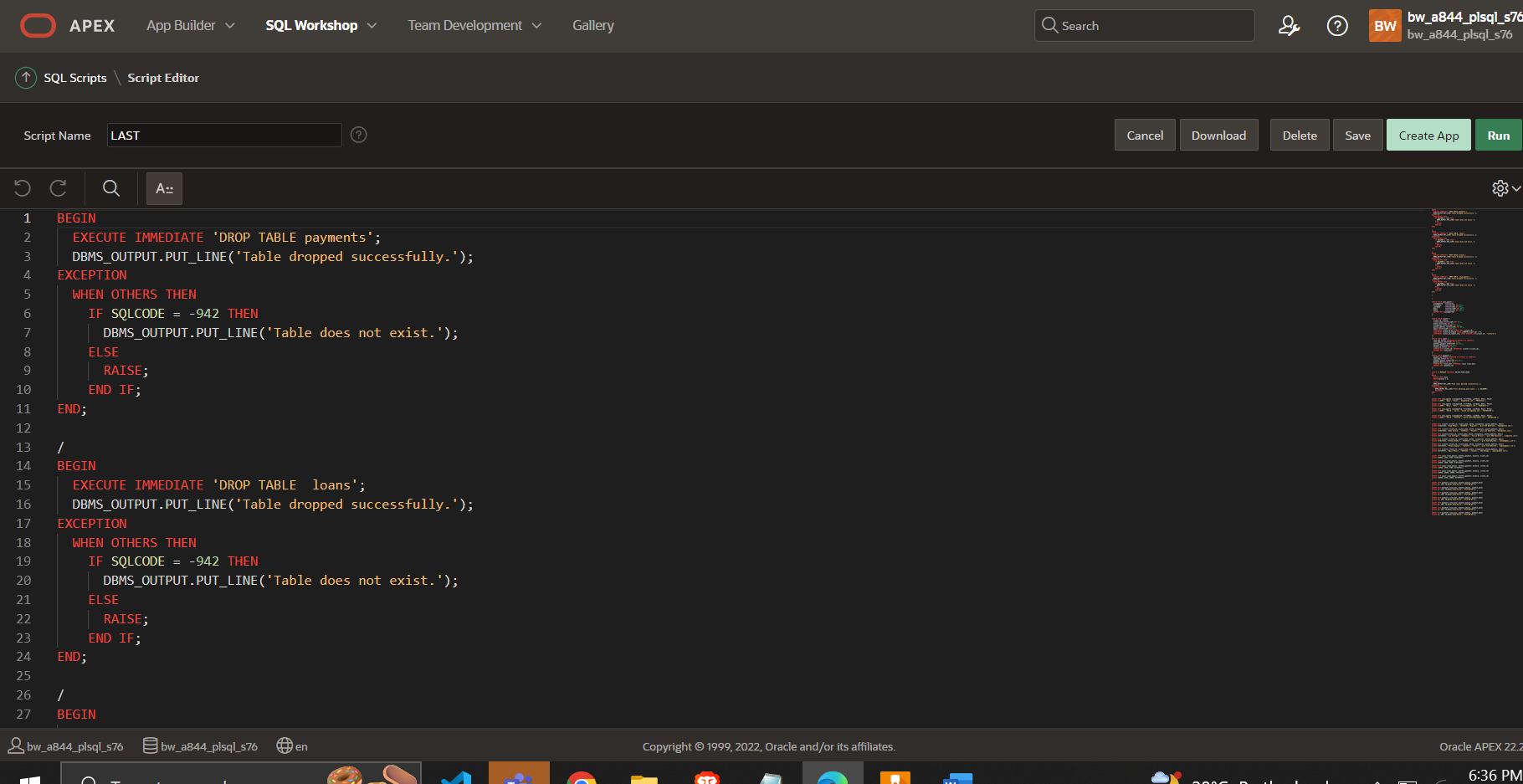
**Loan Management System UML**

*A picture containing text, diagram, screenshot, line

Description automatically generated*

**Task Two:**

*Include screenshots to demonstrate that ALL of your SQL code works.*

**

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

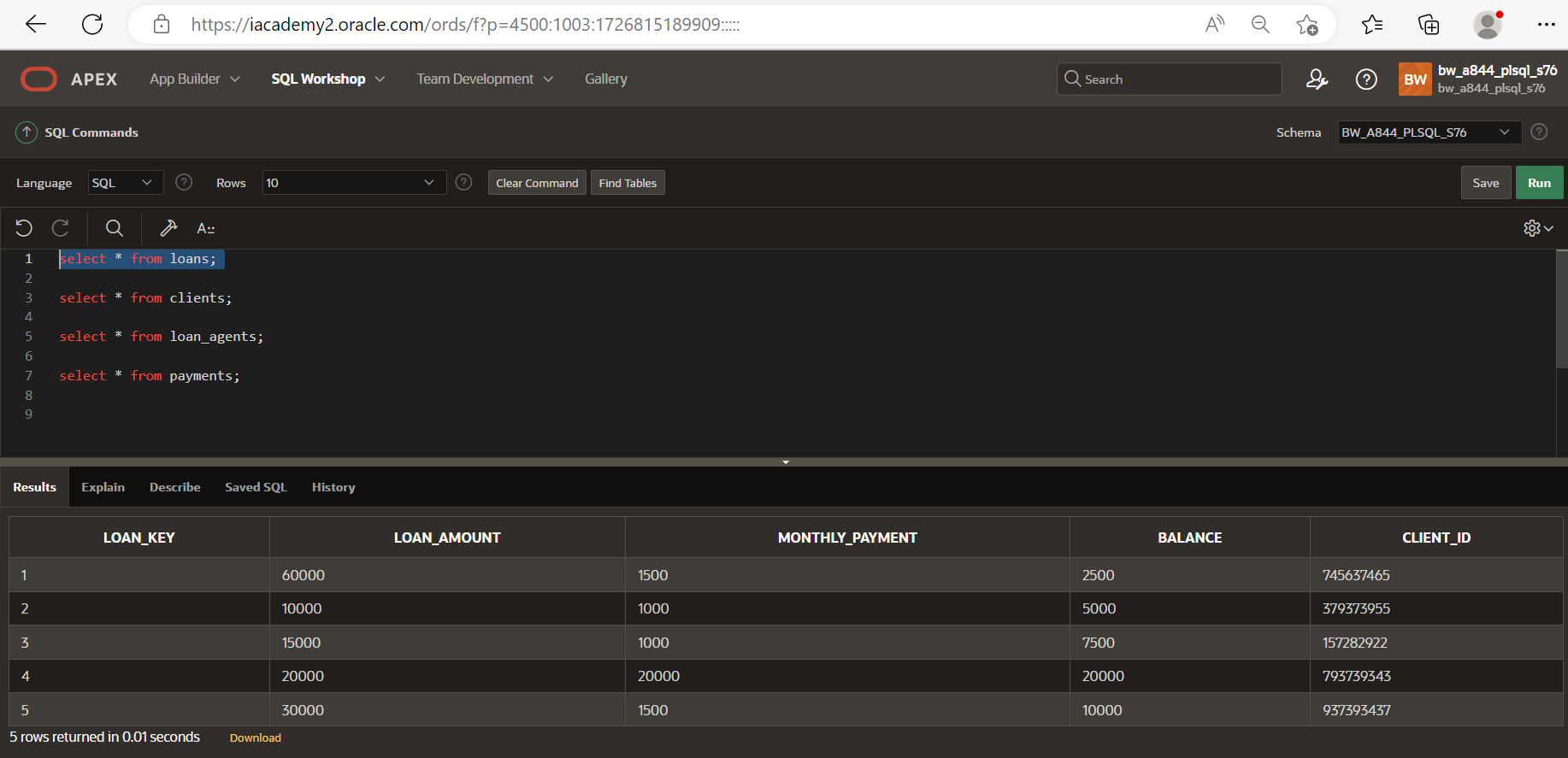
Description automatically generated*

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

**

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

The database consists of four tables: `Loan\_agents`, `clients`, `loans`, and `payments`.

The `Loan\_agents` table stores information about loan agents, including their unique ID, first name, last name, email, and phone number.

The `clients` table contains details of clients who have taken loans. It includes their ID, name, phone number, occupation, postal address, and email.

The `loans` table holds information about the loans themselves. It includes a unique loan key, the loan amount, monthly payment amount, outstanding balance, and the client ID associated with each loan.

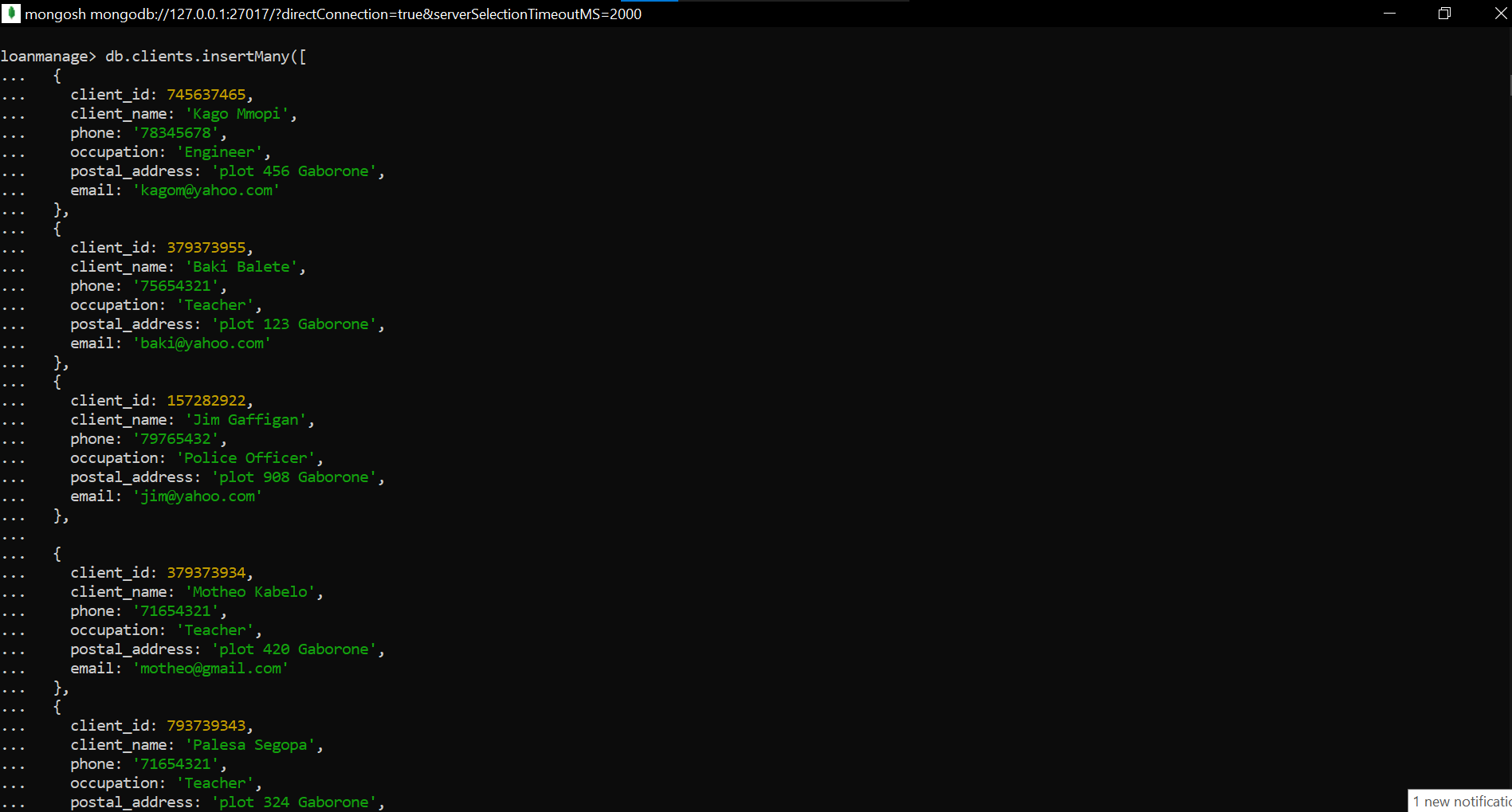
The `payments` table stores data related to loan payments made by clients. It includes a payment ID, the loan key associated with the payment, the payment amount, and the payment date.

The database design allows for the management of loans, clients, loan agents, and their interactions. Loan agents can be associated with multiple clients, clients can have multiple loans, and payments can be tracked for each loan. The stored procedure `DELETE\_PAID\_LOANS` facilitates the deletion of loans with a balance of zero.

**Task Three:**

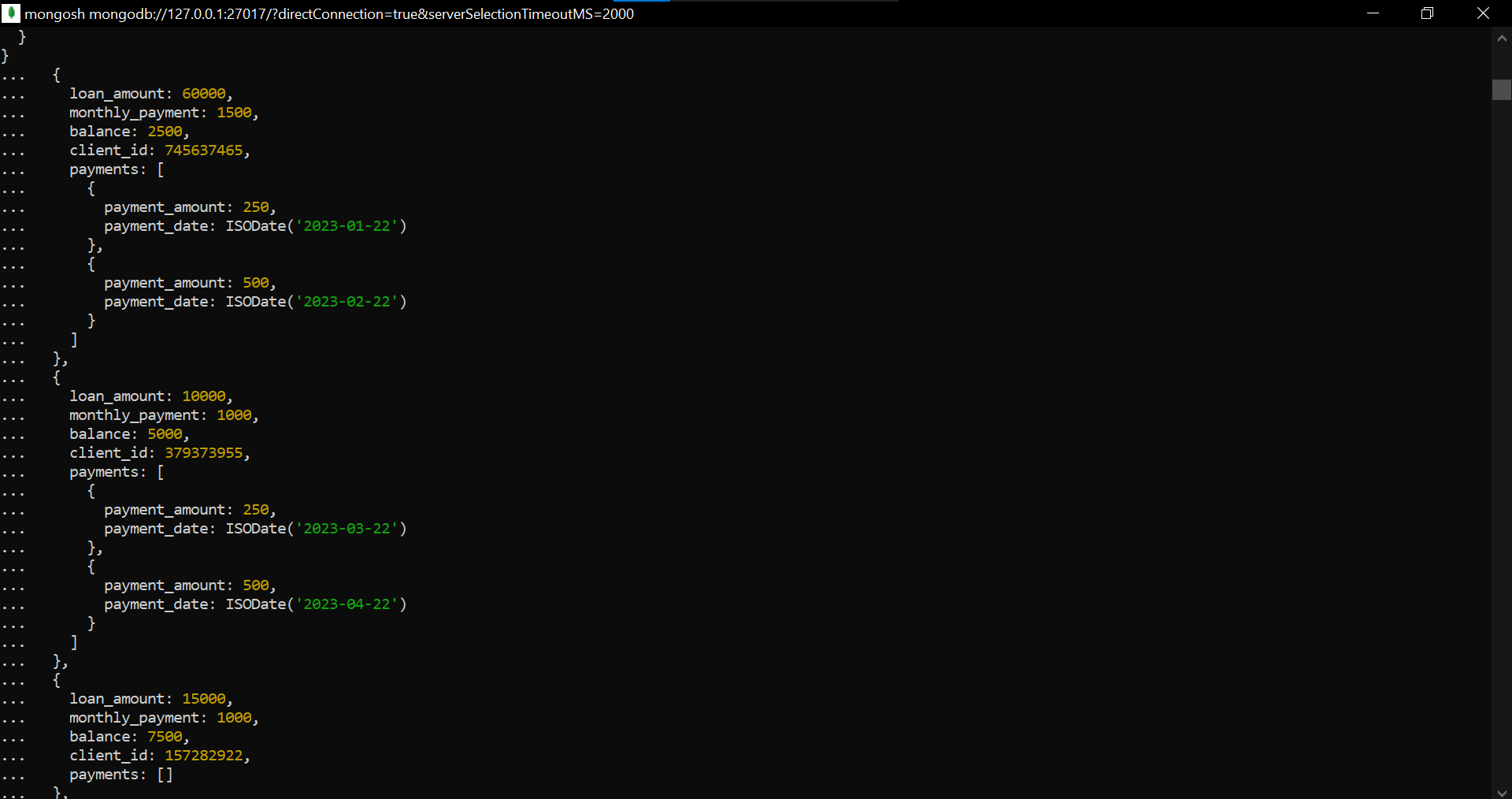
*A picture containing screenshot, text, multimedia software, software

Description automatically generated*

**

*A picture containing text, screenshot, multimedia software, software

Description automatically generated*

**

*A screen shot of a computer

Description automatically generated with medium confidence*

The MongoDB code creates three collections: 'Loan\_agents', 'clients', and 'loans', and inserts documents into each collection. The 'Loan\_agents' collection contains information about loan agents, including their IDs, names, email addresses, and phone numbers. The 'clients' collection stores client information such as client IDs, names, contact details, occupations, and postal addresses. The 'loans' collection stores loan details, including loan amounts, monthly payments, balances, and associated client IDs. It also includes nested documents for payments made towards each loan, specifying the payment amounts and dates

**Tasks Four to Six**

*Complete the following tables:*

|  |  |
| --- | --- |
| **Query a: A join of three or more tables – you should consider various types of join in this query (e.g. inner join, left/right/full outer joins, etc.) and the query must include a restriction on the rows selected**  The query retrieves the client\_name column from the clients table, the loan\_amount column from the loans table, and the payment\_amount column from the payments table.the restriction applied is amount being equal to 500 | |
| **SQL code** | **MongoDB code** |
| SELECT c.client\_name, l.loan\_amount, p.payment\_amount  FROM clients c  LEFT JOIN loans l ON c.client\_id = l.client\_id  LEFT JOIN payments p ON l.loan\_key = p.loan\_key  WHERE p.payment\_amount = 500 ; | db.clients.aggregate([  {  $lookup: {  from: "loans",  localField: "client\_id",  foreignField: "client\_id",  as: "loan"  }  },  {  $unwind: "$loan"  },  {  $lookup: {  from: "payments",  localField: "loan.loan\_key",  foreignField: "loan\_key",  as: "payment"  }  },  {  $unwind: "$payment"  },  {  $match: {  "payment.payment\_amount": 500  }  },  {  $project: {  \_id: 0,  client\_name: "$client\_name",  loan\_amount: "$loan.loan\_amount",  payment\_amount: "$payment.payment\_amount"  }  }  ]) |
| **Screenshots** | |
|  | |
|  | |

|  |  |
| --- | --- |
| **Query b: A query which uses one (or more) of the UNION, DIFFERENCE or INTERSECT operators.**  This query, we have two SELECT statements separated by the INTERSECT operator. The first SELECT statement retrieves the client names from the clients table where the occupation is 'Teacher'. The second SELECT statement retrieves the client names from the same table where the postal address contains the string 'Gaborone’ | |
| **SQL code** | **MongoDB code** |
| SELECT client\_name  FROM clients  WHERE occupation = 'Teacher'  INTERSECT  SELECT client\_name  FROM clients  WHERE postal\_address LIKE '%Gaborone%'; | *db.clients.aggregate([*  *{*  *$match: {*  *$or: [*  *{ occupation: "Teacher" },*  *{ postal\_address: /Gaborone/ }*  *]*  *}*  *},*  *{*  *$group: {*  *\_id: "$client\_name",*  *count: { $sum: 1 }*  *}*  *},*  *{*  *$match: {*  *count: { $gt: 1 }*  *}*  *},*  *{*  *$project: {*  *\_id: 0,*  *client\_name: "$\_id"*  *}*  *}*  *])* |
| **Screenshots** | |
|  | |
|  | |

|  |  |
| --- | --- |
| **Query c: A query which requires use of either a nested table or subtypes**  *Provide a description of your query here* | |
| **SQL code** | **MongoDB code** |
| *Insert the SQL query code here* | *Insert the MongoDB query code here* |
| **Screenshots** | |
|  | |
|  | |

|  |  |
| --- | --- |
| **Query d: A query using temporal features (e.g., timestamps, intervals, etc.) of Oracle SQL**  **To calculate the sum of payments made per month using the given schema, we can use the payment\_date column from the payments** | |
| **SQL code** | **MongoDB code** |
| SELECT EXTRACT(YEAR FROM payment\_date) AS payment\_year,  EXTRACT(MONTH FROM payment\_date) AS payment\_month,  SUM(payment\_amount) AS total\_payment  FROM payments  GROUP BY EXTRACT(YEAR FROM payment\_date), EXTRACT(MONTH FROM payment\_date)  ORDER BY payment\_year, payment\_month; |  |
| **Screenshots** | |
|  | |
| *Insert a screenshot of the output from your MongoDB query here* | |

|  |  |
| --- | --- |
| **Query e: A query using OLAP (e.g., ROLLUP, CUBE, PARTITION) features of Oracle SQL**  CUBE operation is applied to the client\_id column. It generates results that include subtotals for each individual client\_id, as well as subtotals for all combinations of client\_id. The grand total is also included in the result set. | |
| **SQL code** | **MongoDB code** |
| SELECT client\_id, SUM(loan\_amount) AS total\_loan\_amount  FROM loans  GROUP BY CUBE (client\_id); | db.collection.aggregate([  {  $group: {  \_id: null,  cubeResult: {  $sum: {  $pow: ["client\_id", 3] //  }  }  }  }  ]); |
| **Screenshots** | |
|  | |
|  | |