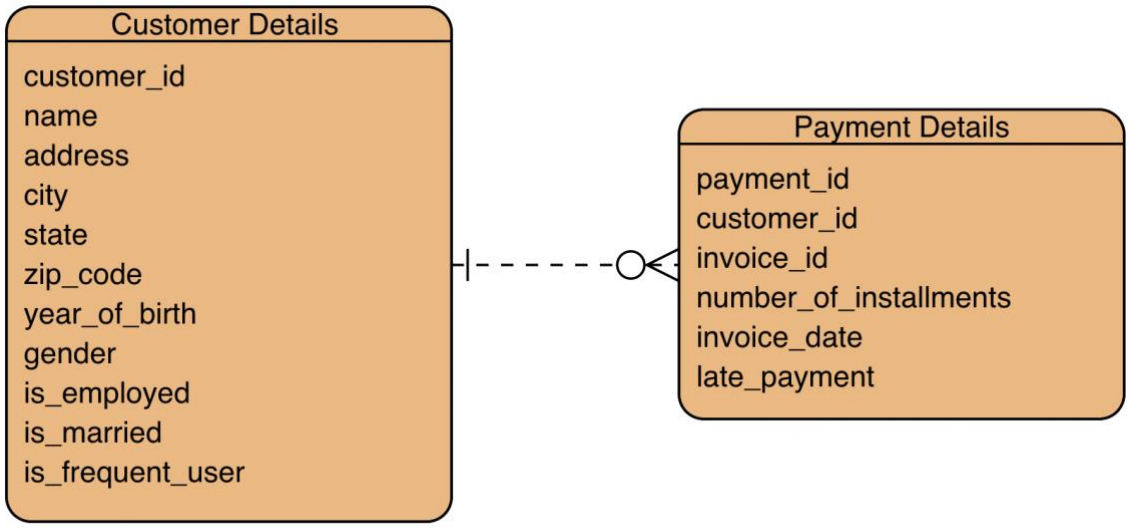
Entity Relationship Diagram



Redis

The records are stored as Hash in Redis. This is because, hash can record unique fields and their values, and is perfect to represent an object. Also, hash can store multiple keys, therefore it is a good way to store our data in Redis.

To store the CustDetails records, HMSET has been used, and to get the keys HGETALL has been used. It is as follows:

```
HMSET CustDetails:1 customer_id 171 name "Madelyn Hensley" address "10075 Thierer Plaza"  city "New York" state "New York" zip_code 81377 year_of_birth 1976 gender "M"  is_employed TRUE is_married FALSE is_frequent_user FALSE

HGETALL CustDetails:1
```

This gives the following output:

```
> HMSET CustDetails:1 customer_id 171 name "Madelyn Hensley" address
"10075 Thierer Plaza" city "New York" state "New York" zip_code 81377
year_of_birth 1976 gender "M" is_employed TRUE is_married FALSE
is_frequent_user FALSE

OK

> HGETALL CustDetails:1

1) "customer_id"
2) "171"
3) "name"
4) "Madelyn Hensley"
5) "address"
6) "10075 Thierer Plaza"
7) "city"
8) "New York"
9) "state"
10) "New York"
11) "zip_code"
12) "81377"
13) "year_of_birth"
14) "1976"
15) "gender"
16) "M"
17) "is_employed"
18) "TRUE"
19) "is_married"
20) "FALSE"
21) "is_frequent_user"
22) "FALSE"
```

In the similar way, the LatePmts records can also be stored in Redis. The syntax is as follows:

```
HMSET LatePmts:1 payment_id 1 cutomer_id 181 invoice_id 146268743 number_of_installments 8 invoice_date "23-8-2020"  late_payment TRUE
```

```
HGETALL LatePmts:1
```

This gives the following output:

```
> HMSET LatePmts:1 payment_id 1 cutomer_id 181 invoice_id 146268743
number_of_installments 8 invoice_date "23-8-2020" late_payment TRUE

OK

> HGETALL LatePmts:1

1) "payment_id"
2) "1"
3) "cutomer_id"
4) "181"
5) "invoice_id"
6) "146268743"
7) "number_of_installments"
8) "8"
9) "invoice_date"
10) "23-8-2020"
11) "late_payment"
12) "TRUE"
```

MongoDB

First, a collection for CustDetails is created in MongoDB using createCollection, followed by storing the data using insertMany(). Finally, the records can be retrieve using find(). The following syntax has been used:

```
use test

db.createCollection('CustDetails')

db.CustDetails.insertMany([{customer_id: 171, name: "Madelyn Hensley", address: {street:"10075 Thierer Plaza", city:"New York", state:"New York", zip_code: 81377}, year_of_birth: 1976, gender: "M", is_employed: "TRUE", is_married:"FALSE", is_frequent_user: "FALSE"}, {customer_id: 172, name: "Lonny Foster", address: {street:"23901 Park Meadow Dr", city: "Austin", state: "Texas", zip_code: 13498}, year_of_birth: 1981, gender: "F", is_employed: "TRUE", is_married: "FALSE", is_frequent_user: "FALSE"}])

db.CustDetails.find()
```

```
>>> db.createCollection('CustDetails')
{ "ok" : 1 }
>>> db.CustDetails.insertMany([{customer_id: 171, name: "Madelyn Hensley", address: {street:"10075
Thierer Plaza", city:"New York", state:"New York", zip_code: 81377}, year_of_birth: 1976, gender:
"M", is_employed: "TRUE", is_married:"FALSE", is_frequent_user: "FALSE"}, {customer_id: 172, name:
"Lonny Foster", address: {street:"23901 Park Meadow Dr", city: "Austin", state: "Texas", zip_code:
13498}, year_of_birth: 1981, gender: "F", is_employed: "TRUE", is_married: "FALSE",
is_frequent_user: "FALSE"}])
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("619dce48f3146575c4bd89af"),
        ObjectId("619dce48f3146575c4bd89b0")
    ]
}
>>> db.CustDetails.find()
{ "_id" : ObjectId("619dce48f3146575c4bd89af"), "customer_id" : 171, "name" : "Madelyn Hensley",
"address" : { "street" : "10075 Thierer Plaza", "city" : "New York", "state" : "New York",
"zip_code" : 81377 }, "year_of_birth" : 1976, "gender" : "M", "is_employed" : "TRUE", "is_married"
: "FALSE", "is_frequent_user" : "FALSE" }
{ "_id" : ObjectId("619dce48f3146575c4bd89b0"), "customer_id" : 172, "name" : "Lonny Foster",
"address" : { "street" : "23901 Park Meadow Dr", "city" : "Austin", "state" : "Texas", "zip_code" :
13498 }, "year_of_birth" : 1981, "gender" : "F", "is_employed" : "TRUE", "is_married" : "FALSE",
"is_frequent_user" : "FALSE" }
```

Similarly, a collection for LatePmts is created using createCollection, enteries are stored using insertMany, and finally we use find to see the enteries.

```
db.createCollection('LatePmts')
```

```
db.LatePmts.insertMany([{payment_id: 1 ,  cutomer_id: 181, invoice_id: 146268743, number_of_installments: 8, invoice_date: "23-8-2020",  late_payment: "TRUE"}, {payment_id: 2 ,  cutomer_id: 172, invoice_id: 396589804, number_of_installments: 7, invoice_date: "28-9-2020",  late_payment: "FALSE"}])
```

```
db.LatePmts.find()
```

```
>>> db.createCollection('LatePmts')
{ "ok" : 1 }
>>> db.LatePmts.insertMany([{payment_id: 1 ,  cutomer_id: 181, invoice_id: 146268743,
number_of_installments: 8, invoice_date: "23-8-2020",  late_payment: "TRUE"}, {payment_id: 2 ,
 cutomer_id: 172, invoice_id: 396589804, number_of_installments: 7, invoice_date: "28-9-2020",
 late_payment: "FALSE"}])
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("619dce62f3146575c4bd89b1"),
        ObjectId("619dce62f3146575c4bd89b2")
    ]
}
>>> db.LatePmts.find()
{ "_id" : ObjectId("619dce62f3146575c4bd89b1"), "payment_id" : 1, "cutomer_id" : 181, "invoice_id"
: 146268743, "number_of_installments" : 8, "invoice_date" : "23-8-2020", "late_payment" : "TRUE" }
{ "_id" : ObjectId("619dce62f3146575c4bd89b2"), "payment_id" : 2, "cutomer_id" : 172, "invoice_id"
: 396589804, "number_of_installments" : 7, "invoice_date" : "28-9-2020", "late_payment" : "FALSE" }
```

The collections made previously can be deleted using drop. First, we check the collection names using getCollectionNames(), drop the collections using drop(), and then again check the collection names to reassure that the collections has been deleted. The syntax is as follows:

```
db.getCollectionNames()
```

```
db.CustDetails.drop()
```

```
db.LatePmts.drop()
```

```
db.getCollectionNames()
```

```
>>> db.getCollectionNames()
[ "CustDetails", "LatePmts" ]
>>> db.CustDetails.drop()
true
>>> db.LatePmts.drop()
true
>>> db.getCollectionNames()
[ ]
>>>
```

The records are then rewritten in XML format, and then stored in MongoDB by first creating collections and then storing the XML as a string.

XML format for CustDetails:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <row>
    <customer_id>171</customer_id>
    <name>Madelyn Hensley</name>
    <address>
      <street>10075 Thierer Plaza</street>
      <city>New York</city>
      <state>New York</state>
      <zip_code>81377</zip_code>
    </address>
    <year_of_birth>1976</year_of_birth>
    <gender>M</gender>
    <is_employed>TRUE</is_employed>
    <is_married>FALSE</is_married>
    <is_frequent_user>FALSE</is_frequent_user>
  </row>
  <row>
    <customer_id>172</customer_id>
    <name>Lonny Foster</name>
    <address>
      <street>23901 Park Meadow Dr</street>
      <city>Austin</city>
      <state>Texas</state>
      <zip_code>13498</zip_code>
    </address>
    <year_of_birth>1981</year_of_birth>
    <gender>F</gender>
    <is_employed>TRUE</is_employed>
    <is_married>FALSE</is_married>
    <is_frequent_user>FALSE</is_frequent_user>
  </row>
</root>
```

The following syntax has been used:

```
db.createCollection('CustDetails_XML')
```

```
db.CustDetails_XML.insert({XML: '<?xml version="1.0" encoding="UTF-8"?> <root> <row> <customer_id>171</customer_id> <name>Madelyn Hensley</name> <address> <street> 10075 Thierer Plaza</street> <city>New York</city> <state>New York</state> <zip_code>81377</zip_code> </address> <year_of_birth>1976<year_of_birth> <gender>M</gender> <is_employed> FALSE</is_employed> <is_married>FALSE</is_married> <is_frequent_user>FALSE</is_frequent_user> </row> <row> <customer_id>172</customer_id> <name>Lonny Foster</name> <address> <street> 23901 Park Meadow</street> <city>Austin</city> <state>Texas</state> <zip_code>13498</zip_code> </address> <year_of_birth>1981<year_of_birth> <gender>F</gender> <is_employed> TRUE</is_employed> <is_married>FALSE</is_married> <is_frequent_user>FALSE</is_frequent_user> </row> </root>'})
```

```
db.CustDetails_XML.find()
```

```
>>> db.CustDetails_XML.insert({XML:'<?xml version="1.0" encoding="UTF-8" ?> <root> <row>
<customer_id>171</customer_id> <name>Madelyn Hensley</name> <address> <street>10075 Thierer
Plaza</street> <city>New York</city> <state>New York</state> <zip_code>81377</zip_code>
</address> <year_of_birth>1976</year_of_birth> <gender>M</gender>
<is_employed>TRUE</is_employed> <is_married>FALSE</is_married>
<is_frequent_user>FALSE</is_frequent_user> </row> <row> <customer_id>172</customer_id>
<name>Lonny Foster</name> <address> <street>23901 Park Meadow Dr</street> <city>Austin</city>
<state>Texas</state> <zip_code>13498</zip_code> </address> <year_of_birth>1981</year_of_birth>
<gender>F</gender> <is_employed>TRUE</is_employed> <is_married>FALSE</is_married>
<is_frequent_user>FALSE</is_frequent_user> </row> </root>'})
WriteResult({ "nInserted" : 1 })
>>> db.CustDetails_XML.find()
{ "_id" : ObjectId("619dd901bf2fc39358fd0527"), "XML" : "<?xml version=\"1.0\" encoding=\"UTF-
8\" ?> <root> <row> <customer_id>171</customer_id> <name>Madelyn Hensley</name> <address>
<street>10075 Thierer Plaza</street> <city>New York</city> <state>New York</state>
<zip_code>81377</zip_code> </address> <year_of_birth>1976</year_of_birth> <gender>M</gender>
<is_employed>TRUE</is_employed> <is_married>FALSE</is_married>
<is_frequent_user>FALSE</is_frequent_user> </row> <row> <customer_id>172</customer_id>
<name>Lonny Foster</name> <address> <street>23901 Park Meadow Dr</street> <city>Austin</city>
<state>Texas</state> <zip_code>13498</zip_code> </address> <year_of_birth>1981</year_of_birth>
<gender>F</gender> <is_employed>TRUE</is_employed> <is_married>FALSE</is_married>
<is_frequent_user>FALSE</is_frequent_user> </row> </root>" }
```

Similarly, the records of LatePmts can be rewritten in XML format and then stored as string in MongoDB.

XML format for LatePmts:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <row>
    <payment_id>1</payment_id>
    <cutomer_id>181</cutomer_id>
    <invoice_id>146268743</invoice_id>
    <number_of_installments>8</number_of_installments>
    <invoice_date>23-8-2020</invoice_date>
    <late_payment>TRUE</late_payment>
  </row>
  <row>
    <payment_id>2</payment_id>
    <cutomer_id>172</cutomer_id>
    <invoice_id>396589804</invoice_id>
    <number_of_installments>7</number_of_installments>
    <invoice_date>28-9-2020</invoice_date>
    <late_payment>FALSE</late_payment>
  </row>
</root>
```

The following syntax has been used:

```
db.createCollection('LatePmts_XML')
```

```
db.LatePmts_XML.insert({XML: '<?xml version="1.0" encoding="UTF-8"?> <root> <row>
<payment_id>1</payment_id> <customer_id>181</customer_id> <invoice_id>146268743</invoice_id>
<number_of_installments>8</number_of_installments> <invoice_date>23-8-2020</invoice_date>
<late_payment>TRUE</late_payment> </row> <row> <payment_id>2</payment_id>
<customer_id>172</customer_id> <invoice_id>396589804</invoice_id>
<number_of_installments>7</number_of_installments> <invoice_date>28-9-2020</invoice_date>
<late_payment>FALSE</late_payment> </row> <root>'})
```

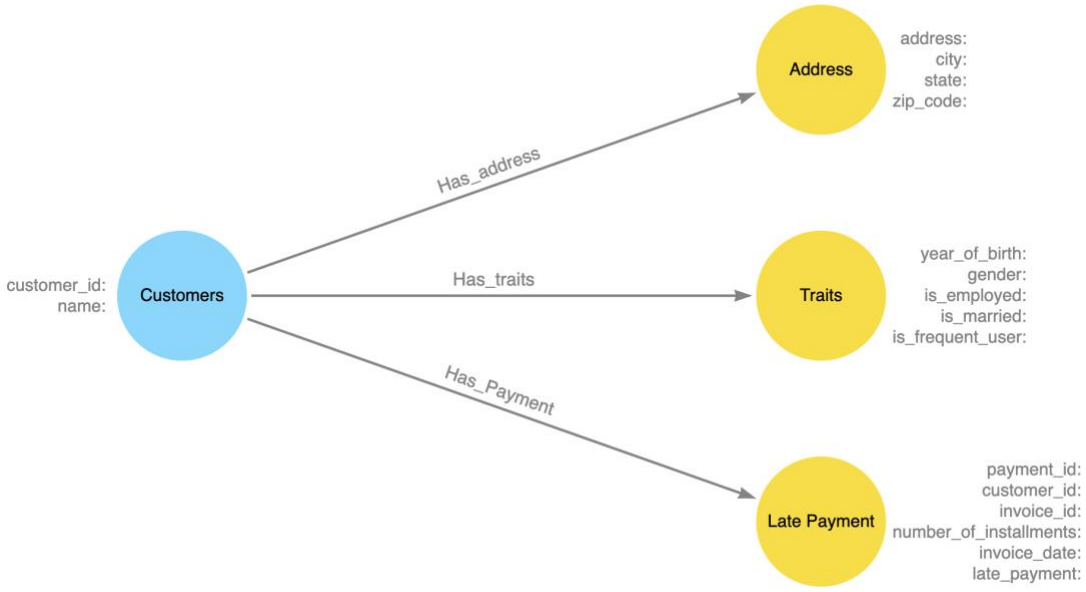
```
db.LatePmts_XML.find()
```

```
>>> db.createCollection('LatePmts_XML')
{ "ok" : 1 }
>>> db.LatePmts_XML.insert({XML:'<?xml version="1.0" encoding="UTF-8" ?> <root> <row> <payment_id>1</payment_id>
<cutomer_id>181</cutomer_id> <invoice_id>146268743</invoice_id> <number_of_installments>8</number_of_installments>
<invoice_date>23-8-2020</invoice_date> <late_payment>TRUE</late_payment> </row> <row> <payment_id>2</payment_id>
<cutomer_id>172</cutomer_id> <invoice_id>396589804</invoice_id> <number_of_installments>7</number_of_installments>
<invoice_date>28-9-2020</invoice_date> <late_payment>FALSE</late_payment> </row> </root>'})
WriteResult({ "nInserted" : 1 })
>>> db.LatePmts_XML.find()
{ "_id" : ObjectId("619dd916bf2fc39358fd0528"), "XML" : "<?xml version=\"1.0\" encoding=\"UTF-8\" ?> <root> <row>
<payment_id>1</payment_id> <cutomer_id>181</cutomer_id> <invoice_id>146268743</invoice_id>
<number_of_installments>8</number_of_installments> <invoice_date>23-8-2020</invoice_date>
<late_payment>TRUE</late_payment> </row> <row> <payment_id>2</payment_id> <cutomer_id>172</cutomer_id>
<invoice_id>396589804</invoice_id> <number_of_installments>7</number_of_installments> <invoice_date>28-9-
2020</invoice_date> <late_payment>FALSE</late_payment> </row> </root>" }
```

Graph Database and Neo4j

The graph-based diagram has been used to show the information and relation of customers with different entities like address, traits, and payments. Four nodes are created which are customer, address, traits, and payments. A customer has an address, has multiple traits, and has a payment/transaction, which has been depicted using arrows (showing the direction of relationship) in the diagram. In each node we have different properties (for example, traits contain information on *year of birth*, *gender*, *employed*, *married*, and *frequent user*). Finally, each diagram shows the information on one customer.

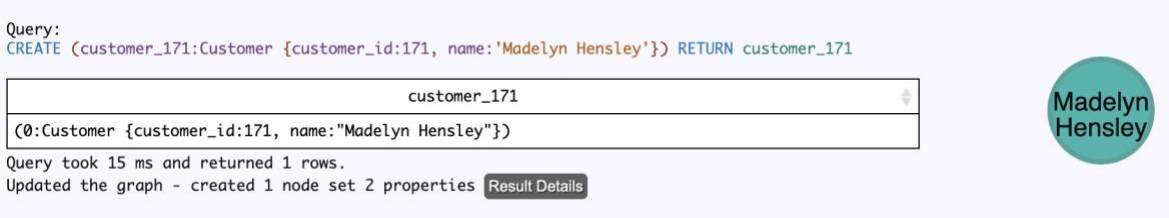
The graph-based diagram is prepared in the following manner:



Neo4j is used to store the first record of the CustDetails. We start by creating the node for the customer and add information on *customer*_id, and *name*.

The following syntax has been used:

```
CREATE (customer_171:Customer {customer_id:171, name:'Madelyn Hensley'}) RETURN customer_171
```

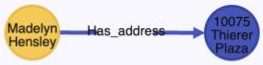


Next, the second node namely address is added and the relationship between customer and address has been defined as follows:

```
MERGE (customer_171:Customer{customer_id:171, name:'Madelyn Hensley'}) CREATE(customer_171)-[:Has_address]-> (address_171:Address {address: '10075 Thierer Plaza', city: 'New York', state: 'New York', zip_code: 81377 }) RETURN address_171
```

```
Query:
MERGE (customer_171:Customer{customer_id:171, name:'Madelyn Hensley'}) CREATE(customer_171)-[:Has_address]-> (address_171:Address
{address: '10075 Thierer Plaza', city: 'New York', state: 'New York', zip_code: 81377 }) RETURN address_171
```

address_171
(1:Address {address:"10075 Thierer Plaza", city:"New York", state:"New York", zip_code:81377})



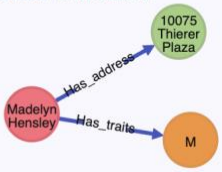
Query took 42 ms and returned 1 rows.
Updated the graph - created 1 node and 1 relationship set 4 properties Result Details

Similarly, we add the third node (Traits) and define the relationship between customer and traits.

```
MERGE (customer_171:Customer{customer_id:171, name:'Madelyn Hensley'}) CREATE(customer_171)-[:Has_traits]-
> (traits_171:Traits {year_of_birth: 1976, gender: 'M', is_employed: TRUE, is_married: FALSE, is_frequent_user:
FALSE}) RETURN traits_171
```

```
Query:
MERGE (customer_171:Customer{customer_id:171, name:'Madelyn Hensley'}) CREATE(customer_171)-[:Has_traits]-> (traits_171:Traits
{year_of_birth: 1976, gender: 'M', is_employed: TRUE, is_married: FALSE, is_frequent_user: FALSE}) RETURN traits_171
```

traits_171
(34:Traits {gender:"M", is_employed:true, is_frequent_user:false, is_married:false, year_of_birth:1976})



Query took 12 ms and returned 1 rows.
Updated the graph - created 1 node and 1 relationship set 5 properties Result Details

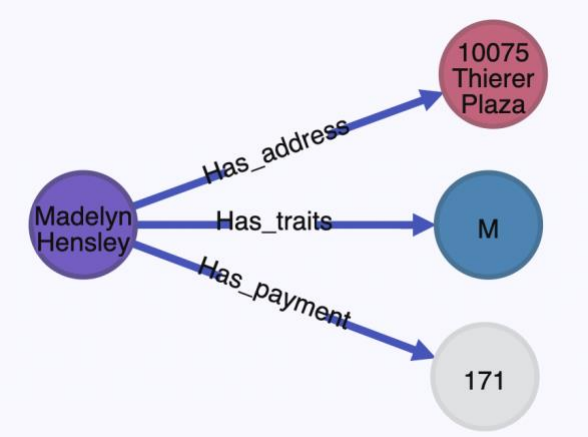
Finally, the last node is created, and the relationship between customer and LatePmts is defined as customers has LatePmts payments/transactions. The syntax and output is as follows:

```
MERGE (customer_171:Customer{customer_id:171, name:'Madelyn Hensley'}) CREATE(customer_171)-
[:Has_payment]-> (payment_171:LatePmts {payment_id: 14, customer_id: 171, invoice_id: 887428332,
number_of_installments: 0, invoice_date: '4-7-2020', late_payment: FALSE}) RETURN payment_171
```

```
Query:
MERGE (customer_171:Customer{customer_id:171, name:'Madelyn Hensley'}) CREATE(customer_171)-[:Has_payment]->
(payment_171:LatePmts {payment_id: 14, customer_id: 171, invoice_id: 887428332, number_of_installments: 0,
invoice_date: '4-7-2020',   late_payment: FALSE}) RETURN payment_171
```

payment_171
(6:LatePmts {customer_id:171, invoice_date:"4-7-2020", invoice_id:887428332, late_payment:false, number_of_installments:0, payment_id:14})

Query took 13 ms and returned 1 rows.
Updated the graph - created 1 node and 1 relationship set 6 properties Result Details



Online Resources:

- https://try.redis.io/
- https://mws.mongodb.com/?version=4.4
- https://console.neo4j.org/
- https://online.visual-paradigm.com/app/diagrams/ - diagram:proj=0&type=ERDiagram
- https://arrows.app/ - /local/id=dlEP_zcgu6s7CCRPaHLa