Use Case Study Report

Group 4
Srinivas Murali , Monisha Prasad

Executive Summary:

The objective of this study was to design an implement a relational database model that could be of use in a rewards loyalty program. The database was modeled taking into consideration the program details, member details, the various types of transactions a member can make along with keeping a log of their rewards (points earned) as well as allotting each of them a tier level that is subject to change with time. The EER diagram was designed first, followed by the UML diagram. The conceptual models were then mapped to a relation model with the primary and foreign keys as recorded in the EER and UML diagrams. The relational database was implemented in MySQL and a smaller preliminary model was used to implement NoSQL queries on Neo4j to study the database functioning in a NoSQL environment.

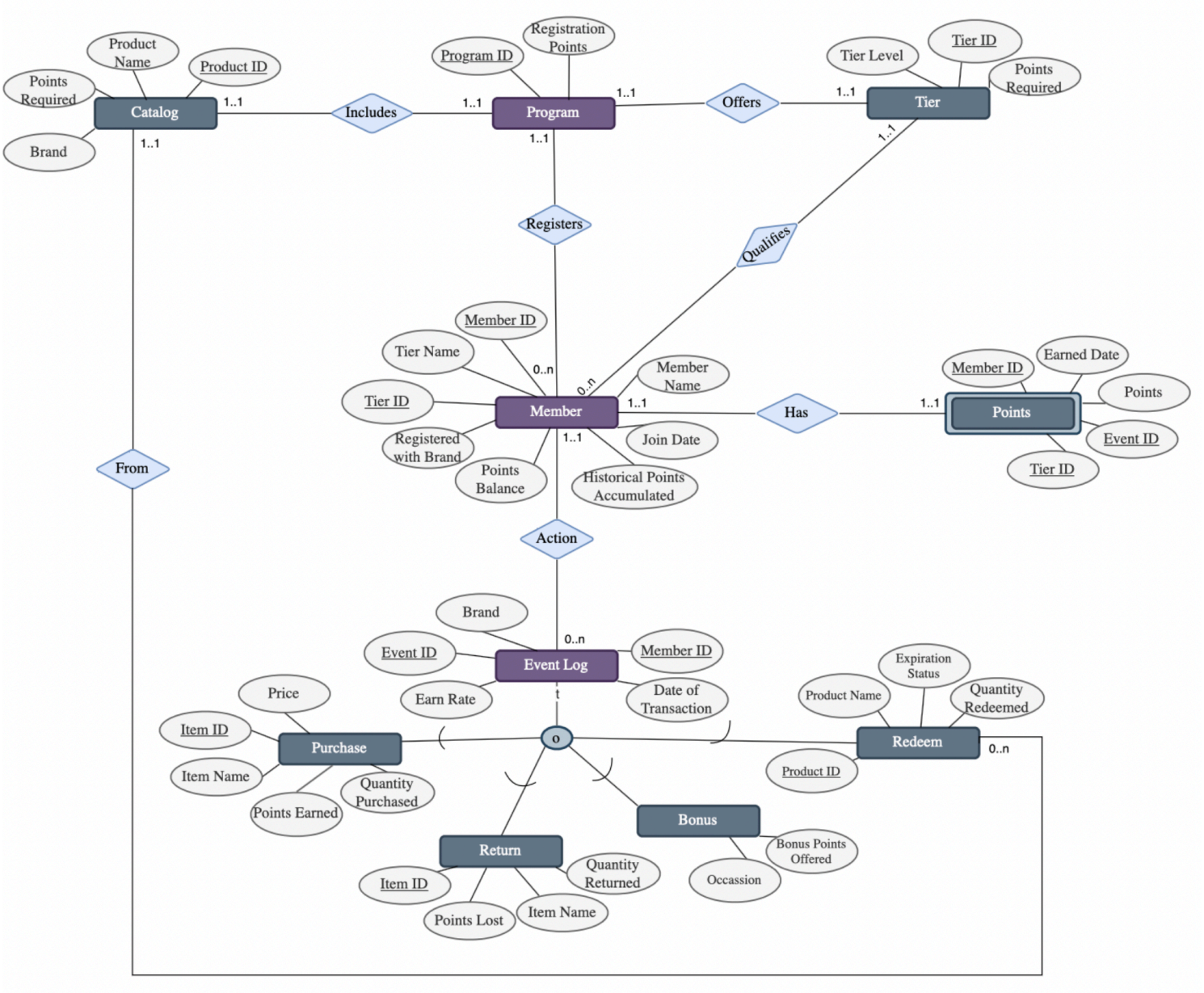
1) Introduction

A retailer has realized that their business has been doing extremely well and wants to capitalize on this potential to increase interaction with the customers. They would like to introduce a loyalty program to reward their customers for shopping with them. The program is open to all present and potential customers, where they can earn points and redeem exciting and exclusive products. In order to roll-out this program, they are faced with the task of setting up backend capabilities which appear in the form of designing a database to capture all program activities.

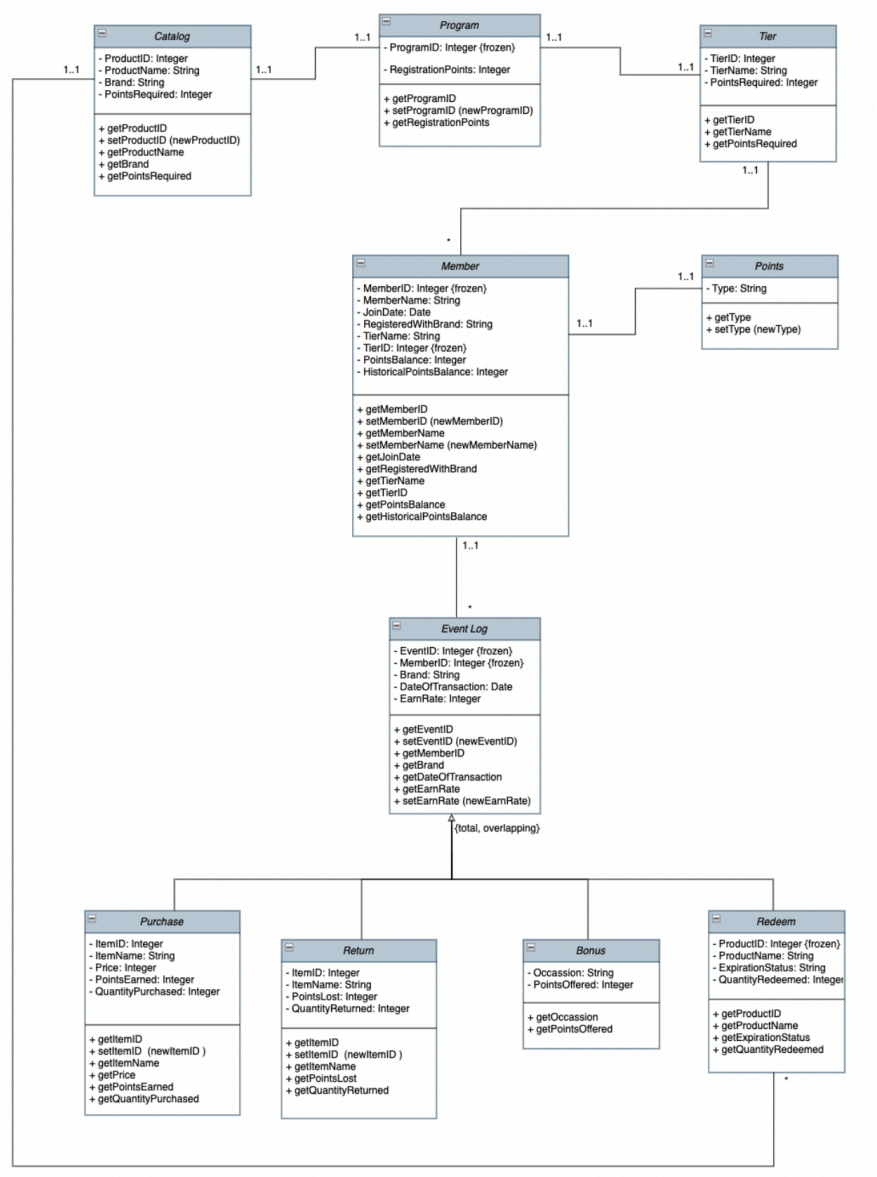
There are certain requirements in order to successfully run a loyalty program.
They are as follows –
1] A customer table to capture customer details when they register online/offline. This table must have a unique identifier for every member followed by information around which tier they fall into and their points balance. They must also capture which brand they are registered through into the program
2] An events log table to capture every event (interaction) that a customer has with the program. Some examples are registration, purchase, return, redemption, etc. This table is the backbone to the program as every other table can be derived using this
3] A rewards catalog that can be accessed by members of the program to redeem the points that they accumulate over time due to their interaction with the programs. The catalog is not just restricted to products but will have choices like monetary rewards, access to exclusive events, vouchers and coupons, and other benefits
4] Tiers are awarded to members that are involved more in the program via spending and other interactions. These tier thresholds need to be in place to clearly distinguish members are reward them accordingly as each tier will have different access levels to rewards catalog and ultimately make the program tier centric. This will drive members to spend more and this is beneficial for the business

5] Points are rewarded to members that interact with the program. They can be earned via transactional and non-transactional ways. Different tiers earn points at different rates. The lowest tier earns 1pt/$ spent whereas the topmost tier can earn 5pt/$ spent. Checks need to be in place where the points earned will expire within 12 months if left untouched

2) A complete conceptual data model using the EER model



A complete conceptual data model using UML class diagram



3) Mapping of conceptual data model into a relational model (primary keys are underlined; foreign keys are in italics):

Program (Program ID, Registration Points)
Tier (Tier ID, Tier Name, Points Required, *Program ID*)
Catalog (Product ID, Product Name, Points Required, Brand, *Program ID*)
Member (Member ID, Member Name, Join Date, Points Balance, Historic Points Balance, Registered with Brand, Tier Name, *Tier ID*, *Program ID*)
Points (Points, Tier ID, Tier Name, Earned Date, *Member ID*, *Event ID*)
Event Log (Event ID, Date of Transaction, Brand, Earn Rate, *Member ID*)
Purchase(Item ID, Item Name, Price, Points Earned, Quantity Purchased, *Event ID*, *Member ID*)
Return(Item ID, Item Name, Points Lost, Quantity Returned, *Event ID*, *Member ID*)
Redeem(Product Name, *Product ID*, Expiration Status, Quantity Redeemed, *Event ID*, *Member ID*)
Bonus(Occassion, Bonus Points Offered, *Event ID*, *Member ID*)

4) Implementation of Relation Model via MySQL and NoSQL

MySQL Implementation:

Q1.

Find tier distribution at the start v/s the end of the year (tier shift)

```
SELECT st.user_id AS 'UserID'
        ,st.tier_name AS 'Starting Tier'
        ,et.tier_name AS 'Ending Tier'
        ,CASE
                WHEN st.tier_name = et.tier_name
                        THEN 0
                ELSE (et.event_time - st.event_time)
                END AS 'Time (in months)'
FROM (SELECT a.user_id
                ,a.tier_name
                ,a.event_time
        FROM finaldataset a
        WHERE a.event_time = (
                        SELECT min(event_time)
                        FROM finaldataset b
                        WHERE b.user_id = a.user_id)
        GROUP BY a.user_id
                ,a.tier_name
                ,a.event_time) AS st
LEFT JOIN (SELECT a.user_id
                ,a.tier_name
                ,a.event_time
        FROM finaldataset a
        WHERE a.event_time = (
                        SELECT max(event_time)
                        FROM finaldataset b
                        WHERE b.user_id = a.user_id)
        GROUP BY a.user_id
                ,a.tier_name
                ,a.event_time) AS et ON st.user_id = et.user_id
```

Result Grid | Filter Rows: | Export:

UserID	Starting Tier	Ending Tier	Time (in months)
348405118	Bronze	Bronze	0
412120092	Bronze	Gold	7
494077766	Bronze	Bronze	0
546170008	Bronze	Bronze	0
560109803	Bronze	Silver	3
576005683	Bronze	Silver	7
576802932	Bronze	Platinum	8
579751441	Bronze	Gold	8
579966747	Bronze	Silver	6
579969717	Bronze	Bronze	0

Q2.

Find revenue, points earned, points redeemed by member and tier

- **By Member**

```
SELECT sales.user_id AS 'MemberID'
        ,sales.Sales
        ,IFNULL(redemption.`Points Redeemed`, 0) AS 'Points Redeemed'
        ,IFNULL(RETURNS.`Returns`, 0) AS 'Returns'
        ,IFNULL(bonus.`Bonus Earned`, 0) AS
```

Result Grid | Filter Rows: | Export: | Wrap Ce

MemberID	Sales	Points Redeemed	Returns	Bonus Earned
348405118	74.13	-7	-11.07	20
412120092	425.95	-35	-51.11	0
494077766	110.9	0	-11.27	1
546170008	22.49	-29	0	0
560109803	137.85	0	0	0
576005683	159.45	-40	-27.6	0
576802932	1330.73	0	-834.35	211
579751441	851.67	-600	-275.87	0
579966747	194.04	0	-15.89	0
579969717	10.95	-21	0	0

```
'Bonus Earned'
FROM (SELECT user_id
            ,round(sum(price), 2) AS 'Sales'
      FROM finaldataset
      WHERE event_type = 'purchase'
      GROUP BY user_id) sales
LEFT JOIN (SELECT user_id
            ,sum(points_earned) AS 'Points Redeemed'
      FROM redemption
      GROUP BY user_id) redemption ON sales.user_id = redemption.user_id
LEFT JOIN (SELECT user_id
            ,round(sum(price), 2) AS 'Returns'
      FROM
      RETURNS
      GROUP BY user_id)
RETURNS ON sales.user_id =
RETURNS.user_id
LEFT JOIN (SELECT user_id, sum(points_earned) AS 'Bonus Earned'
          FROM bonus
          GROUP BY user_id) bonus ON sales.user_id = bonus.user_id
```

- **By tier**

```
SELECT main.`Tier Name`
      ,sum(main.Sales) AS `Sales`
      ,sum(main.`Points Redeemed`) AS `Points Redeemed`
      ,sum(main.`Returns`) AS `Returns`
      ,sum(main.`Bonus Earned`) AS `Bonus Earned`
FROM (SELECT sales.user_id AS ‘MemberID' , tier.`Tier Name`, sales.Sales
            ,IFNULL(redemption.`Points Redeemed`, 0) AS 'Points Redeemed'
            ,IFNULL(RETURNS.`Returns`, 0) AS 'Returns'
            ,IFNULL(bonus.`Bonus Earned`, 0) AS 'Bonus Earned'
      FROM (SELECT user_id
                  ,round(sum(price), 2) AS 'Sales'
            FROM finaldataset
            WHERE event_type = 'purchase'
            GROUP BY user_id
            ) sales
      LEFT JOIN (SELECT user_id, sum(points_earned) AS 'Points Redeemed'
            FROM redemption
            GROUP BY user_id
            ) redemption ON sales.user_id = redemption.user_id
      LEFT JOIN (SELECT user_id,round(sum(price), 2) AS 'Returns'
            FROM
            RETURNS
            GROUP BY user_id)
      RETURNS ON sales.user_id =
      RETURNS.user_id
```

```
        LEFT JOIN (SELECT user_id
                        ,sum(points_earned) AS 'Bonus Earned'
                FROM bonus
                GROUP BY user_id
                ) bonus ON sales.user_id = bonus.user_id
        LEFT JOIN (SELECT a.user_id
                        ,a.tier_name AS 'Tier Name'
                FROM finaldataset a
                WHERE a.event_time = (
                                SELECT max(event_time)
                                FROM finaldataset b
                                WHERE b.user_id = a.user_id)
                GROUP BY a.user_id
                        ,a.tier_name
                ) tier ON sales.user_id = tier.user_id) main
GROUP BY `Tier Name`
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Tier Name	Sales	Points Redeemed	Returns	Bonus Earned
Bronze	218.47	-57	-22.34	21
Gold	1277.62	-635	-326.98	0
Silver	491.3399999999999	-40	-43.49	0
Platinum	1330.73	0	-834.35	211

Q3.

Find most popular brands by different levels (like customer, tier, etc)

- **by tier**

```
SELECT final.`Tier Name`, final.brand, sum(final.timespurchased) AS timespurchased
FROM (
        SELECT b.user_id AS 'MemberID'
                ,tier.`Tier Name`
                ,b.brand
                ,b.timespurchased
        FROM (
                SELECT user_id
                        ,brand
                        ,count(brand) AS timespurchased
                FROM purchases
                GROUP BY user_id
                        ,brand) b
        LEFT JOIN (
                SELECT a.user_id
                        ,a.tier_name AS 'Tier Name'
```

Result Grid | Filter Rows:

Tier Name	brand	timespurchased
Bronze	2	25
Bronze	3	25
Gold	3	86
Silver	3	45
Platinum	2	109

```
        FROM finaldataset a
        WHERE a.event_time = (
                SELECT max(event_time)
                FROM finaldataset b
                WHERE b.user_id = a.user_id)
        GROUP BY a.user_id
            ,a.tier_name) tier ON b.user_id = tier.user_id) final
GROUP BY final.`Tier Name`,final.brand
```

- **by customer (follows the same code as tier but using Member)**

create a view to consolidate tier level popular brand

```
SELECT b.`Tier Name`
    ,a.brand
    ,b.timespurchased
FROM loyaltyprogram.maxbrand a
RIGHT JOIN (
    SELECT `Tier Name`
        ,MAX(timespurchased) AS timespurchased
    FROM loyaltyprogram.maxbrand
    GROUP BY `Tier Name`
    ) b ON a.`Tier Name` = b.`Tier Name`
    AND a.timespurchased = b.timespurchased
GROUP BY b.`Tier Name`
    ,a.brand
    ,b.timespurchased
```

Result Grid | Filter Rows:

Tier Name	brand	timespurchased
Bronze	2	25
Bronze	1	13
Bronze	4	8
Bronze	3	25
Gold	3	86
Gold	1	47
Gold	2	67
Gold	4	33
Silver	2	39
Silver	1	25
Silver	4	25
Silver	3	45
Platinum	2	109
Platinum	1	63
Platinum	3	103
Platinum	4	57

Q4.

Find average basket value and average # of items purchased by customer_id and tier

- **by customer**

```
SELECT p.user_id AS 'MemberID'
    ,(p.sales - IFNULL(r.RETURNS, 0)) AS 'Sales'
    ,p.pt AS 'Transactions'
    ,((p.sales - IFNULL(r.RETURNS, 0)) / p.pt) AS 'Avg Basket Price'
    ,(p.prc / p.pt) AS 'Avg Basket Product Count'
FROM (
    SELECT user_id
        ,sum(price) AS sales
        ,count(DISTINCT event_id) AS pt
        ,count(DISTINCT product_id) AS prc
    FROM purchases
    GROUP BY user_id
    ) p
LEFT JOIN (
```

```
        SELECT user_id
                ,sum(price) AS
        RETURNS
        FROM
        RETURNS
        GROUP BY user_id
        ) r ON p.user_id = r.user_id
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

MemberID	Sales	Transactions	Avg Basket Price	Avg Basket Product Count
348405118	85.2	6	14.2	3.67
412120092	477.06	7	68.15	3.43
494077766	122.17	5	24.43	7.20
546170008	22.49	3	7.5	1.00
560109803	137.85	5	27.57	5.00
576005683	187.05	4	46.76	4.25
576802932	2165.08	22	98.41	10.50
579751441	1127.54	4	281.89	29.25
579966747	209.93	5	41.99	8.40
579969717	10.95	1	10.95	1.00

- **by tier**

```
SELECT final.`Tier Name`
        ,sum(final.Sales) /
sum(final.Transactions) AS 'Avg Basket Price'
        ,sum(final.`Product Count`) / sum(final.Transactions) AS 'Avg Basket Product Count'
FROM (
        SELECT p.user_id AS 'MemberID'
                ,tier.`Tier Name`
                ,(p.sales - IFNULL(r.RETURNS, 0)) AS 'Sales'
                ,p.pt AS 'Transactions'
                ,p.prc AS `Product Count`
        FROM (
                SELECT user_id
                        ,sum(price) AS sales
                        ,count(DISTINCT event_id) AS pt
                        ,count(DISTINCT product_id) AS prc
                FROM purchases
                GROUP BY user_id) p
        LEFT JOIN (
                SELECT user_id
                        ,sum(price) AS
                RETURNS
                FROM
                RETURNS
                GROUP BY user_id
                ) r ON p.user_id = r.user_id
        LEFT JOIN (
                SELECT a.user_id
                        ,a.tier_name AS 'Tier Name'
                FROM finaldataset a
                WHERE a.event_time = (
                                SELECT max(event_time)
                                FROM finaldataset b
                                WHERE b.user_id = a.user_id)
                GROUP BY a.user_id
                        ,a.tier_name
                ) tier ON p.user_id = tier.user_id
        ) final
```

Result Grid | Filter Rows: | Export:

Tier Name	Avg Basket Price	Avg Basket Product Count
Bronze	16.05	4.13
Gold	145.87	12.82
Silver	38.2	6.00
Platinum	98.41	10.50

```
GROUP BY final.`Tier Name`
```

Q5.

Calculate average time between purchases using lead/lag

```
SELECT final.user_id
	,avg(diff) AS `Avg Purchase TIME(days) `
FROM (
	SELECT base.user_id
		,base.event_time
		,base.previoustx
		,datediff(str_to_date(cast(base.event_time AS
CHAR), '%m/%d/%Y'), str_to_date(cast(base.previoustx AS
CHAR), '%m/%d/%Y')) AS diff
	FROM (
		SELECT user_id
			,event_time
			,LAG(event_time) OVER (
				PARTITION BY user_id
ORDER BY user_id
				) AS previoustx
		FROM purchases
		GROUP BY user_id
			,event_time) base
	WHERE base.previoustx IS NOT NULL) final
GROUP BY final.user_id
```

Result Grid | Filter Rows:

user_id	Avg Purchase Time(days)
348405118	52
412120092	25
494077766	41
546170008	40
560109803	23
576005683	29
576802932	17
579751441	54
579966747	35

Q6.

Rank every member's transaction value

```
SELECT user_id
	,event_id
	,round(sum(price), 2) AS sales
	,RANK() OVER (
		PARTITION BY user_id ORDER BY sum(price)
DESC
		) AS rankk
FROM purchases
GROUP BY user_id
	,event_id
```

Result Grid | Filter Rows:

user_id	event_id	sales	rankk
348405118	event15	40.14	1
348405118	event19	13.34	2
348405118	event1	10.48	3
348405118	event11	5.24	4
348405118	event7	3.18	5
348405118	event9	1.75	6
412120092	event30	111.27	1
412120092	event32	82.55	2
412120092	event31	77.38	3
412120092	event36	71.83	4
412120092	event35	64.03	5
412120092	event33	10.95	6
412120092	event34	7.94	7
494077766	event43	39.61	1
494077766	event42	26.28	2
494077766	event41	23.57	3
494077766	event44	15.09	4

NoSQL Implementation:

Q1.

Finding tier level of each member

```
MATCH (m:Member)-[q:Qualifies]-(t:Tier)
RETURN m.Member_ID,t.Tier_ID,t.Tier_Level
```

"m.Member_ID"	"t.Tier_ID"	"t.Tier_Level"
"579751441"	"3"	"Gold"
"412120092"	"3"	"Gold"
"576802932"	"4"	"Platinum"
"560109803"	"2"	"Silver"
"576005683"	"2"	"Silver"
"579966747"	"2"	"Silver"
"579969717"	"1"	"Bronze"
"546170008"	"1"	"Bronze"
"494077766"	"1"	"Bronze"
"348405118"	"1"	"Bronze"

Q2.

Find all bonus points earned by member_id ’348405118’

```
MATCH (b:Bonus)
WHERE b.member_id='348405118'
RETURN b.occassion, b.bonus_points_offered,
b.member_id;
```

"b.occassion"	"b.bonus_points_offered"	"b.member_id"
"bonus"	"1"	"348405118"
"bonus"	"2"	"348405118"
"bonus"	"2"	"348405118"
"bonus"	"2"	"348405118"
"bonus"	"1"	"348405118"
"bonus"	"2"	"348405118"
"bonus"	"2"	"348405118"
"bonus"	"2"	"348405118"
"bonus"	"1"	"348405118"
"bonus"	"1"	"348405118"
"bonus"	"2"	"348405118"
"bonus"	"2"	"348405118"
"registration"	"10"	"348405118"

Q3.

Finding products of brand ‘2’ offered in the catalog

```
MATCH (p:Program)-[i:Includes]-(c:Catalog)
WHERE c.brand = '2'
RETURN c.brand,c.product_id,c.points_required;
```

"c.brand"	"c.product_id"	"c.points_required"
"2"	"5809911"	"5"
"2"	"5854832"	"5"
"2"	"5904253"	"2"
"2"	"5904747"	"2"
"2"	"5904256"	"2"

5) Database Access via R

The database is accessed using R and we plot the following graphs to observe certain trends in our dataset:

Chart 1: Brand popularity by member - Members interacted with brands 1,2,4 equally followed by brand 3

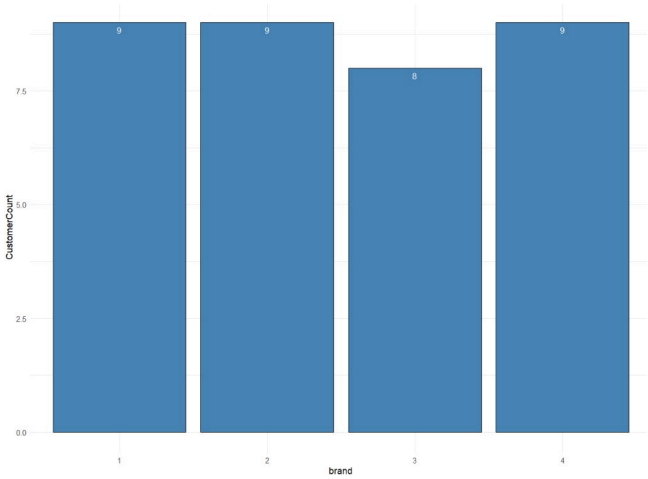


Chart 2: Spendings per customer - Shows us how much was made in sales by customer id

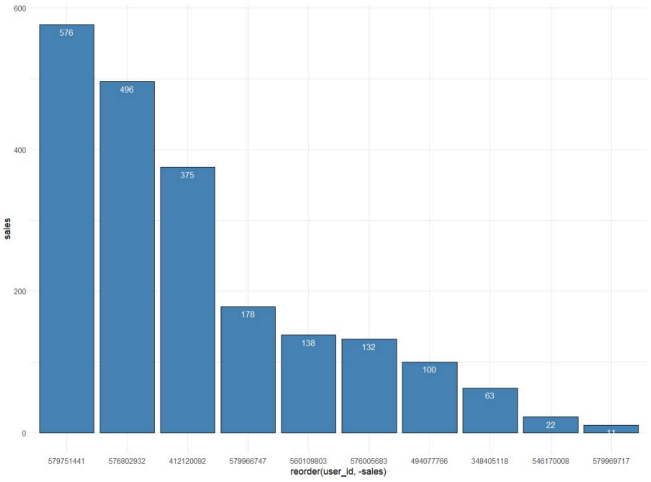
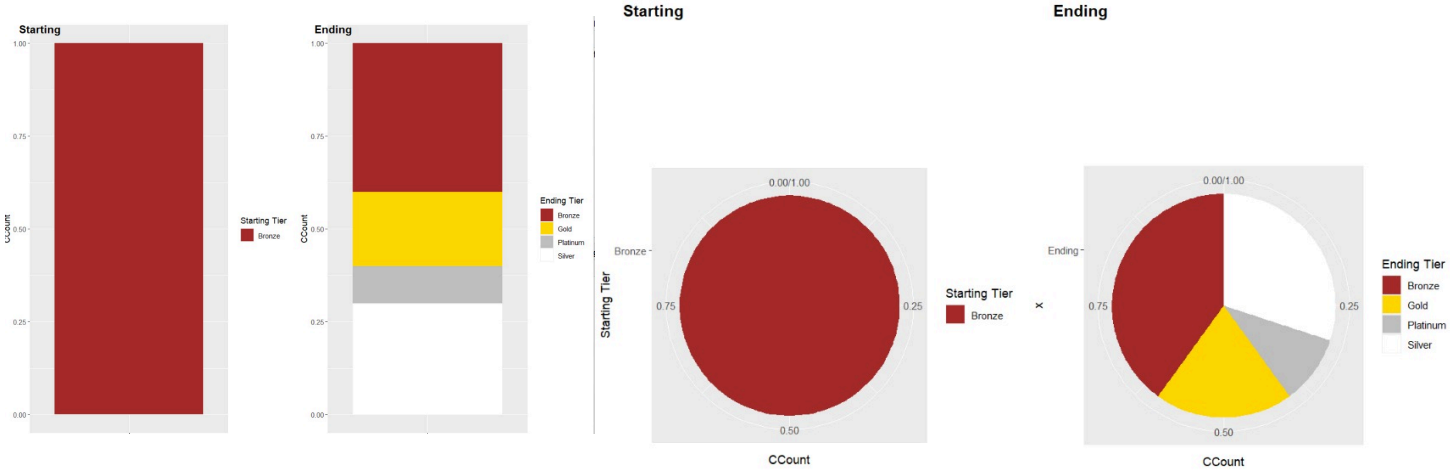


Chart 3: Tier Distribution - Shows tier distribution change over the one year period recorded in the database in a stacked bar form as well as a pie chart (started with all bronze, then some members gained tier levels)



6) Summary and Recommendation

Using this relational database a retailer can keep track of their loyalty program. We have allowed for there to be a scope of expansion to the data model as well, like in terms of tier level, as the years go by the cut off points for each tier level can be changed or even expanded. Limitations of this project include the lack of being able to implement all the tables in NoSQL given the size of our database (event log containing 1000's of records). However, when completed this relational database can be lucrative to any retailer wanting to implement a rewards loyalty program as well as be useful to their customers by providing them access to exclusive products and offers.