Procurement Card Usage Memo

Phase 1 (Conceptual Modeling)

When we talk about **Procurement Cards** there are five primary types of **procurement cards** that are available to us namely - **Standard Procurement Card, Enhanced Procurement Card, DecliningBalance Procurement Cards, Open Procurement Card,** and **Student Payment Procurement Card**. The Declining Balance Procurement Cards can be further classified as **Effective Date Card** and **Revolving Card**. Each of these procurement cards will contain the **Procurement Card Number, the Cardholder Name, the Effective Date, the Spend limit,** and lastly **the remaining Balance** in the card.

Each of these **Procurement Cards** is issued to an **Employee**. Each **Employee** has an **Employee ID, Name, SSN, Post, Start Date, Department, corresponding Department ID**, and lastly **an Email**. **Employees** can be of two types – the **Regular Faculty** and **Staff** and **Temporary Employees** who are hired as consultants for a short duration of time. We also have the **End Dates** for the **Temporary Employees**.

The **Regular Faculty and Staff** comprises of people who could hold overlapping roles in the University. These roles include **Procurement Card Administrator, Cardholder (Applicant), President/Dean, Editor,** and **Reconciler**.

The **Procurement Card Administrator** reviews the **Procurement Card Application.** This **Procurement Card Application** consists of the following fields: **Reviewer ID, Application Status, Application ID, Applicant Name,** and **Applicant ID**. The **Procurement Card Administrator** also has the power to authorize the **Procurement Card**.

The **President/Dean** reviews and approves this **Procurement Card Application** and if things don't add up then the **President/Dean** can open the **procurement card application** as well.

The **Cardholder (Applicant)** is the one who applies for the **Procurement Card Application**. The **Cardholder (Applicant)** is also required to submit proper **documents** to the appropriate **Editor** in a timely manner. Each **Document** has a **Maintainer ID, Document ID,** and **Submitter ID**.

The **Editor** maintains the **Document** and reviews and edits all **Transactions**. Each **Transaction** contains the **Reviewer ID, Transaction ID, Reviewer Status, Project, Sender ID**, and **Amount**.

Lastly, the **Reconciler** is the one who verifies the above-mentioned **Document** and the **Electronic Record** for the same. This **Electronic Record** must contain the **eRecord ID, the Applicant ID,** and the name of the person it was **reconciled by.**

Design Discussions:

While addressing the **Regular Faculty and Staff**, we were confused about whether the roles of the employees in this would be overlapping or distinct. Upon further discussion, we concluded that these can be overlapping roles.

We pondered upon the loop of the **Procurement Card Application**. We were not very sure of how this would be showcased in the enhanced entity-relationship (EER) diagram.

Determining and effectively illustrating the relationships of the different entities and formulating certain assumptions to make sure the enhanced entity-relationship (EER) diagram made perfect sense.

Phase 2 (Relational Database Design and Queries)

The Team has implemented several modifications to the diagram and devised a relational model utilizing the SQLite database system. The relational schema was derived from our EER Diagram, and SQL queries served as the foundation for the schema's construction.

Revisions to EERD:

Formerly, our intention was to utilize Social Security Numbers (SSN) as the singular and unique foreign key for ***Procurement Card Administrator, Cardholder (Applicant), President/Dean, Editor,*** and ***Reconciler***. However, we have since undergone a shift in approach, resulting in the allocation of a primary key to each of the aforementioned roles, specifically, ***Admin_ID, Cardholder_ID, President_ID, Editor_ID, and Reconciler_ID***.

In addition, we have excluded ***Sender_ID*** from the ***Transaction*** entity, as it was deemed superfluous in the context of our required data and information. The ***Transaction*** table was linked to the cardholder table through the ***Cardholder_ID***.

Database Schema and Constraints:

"**Procurement_Card**" is an Entity with attributes "Card_Number, Cardholder_Name, Effective_Date, Spend_Limit, Balance, Type and Cardholder_ID". "**Procurement_Card_application**" is an Entity with Attributes "Application_ID, Admin_ID, Cardholder_ID, President_ID, Applicant_Name, Application_Status". "**Procurement_Card_Administrator**" is an Entity with Attributes "Admin_ID, SSN, Name, Designation, Department, Department_ID, Email, Start_Date". "**Cardholder**" is an Entity with Attributes "Cardholder_ID, SSN, Name, Designation, Department, Department_ID, Email, Start_Date". "**Documents**" is an Entity with Attributes "Document_ID, Cardholder_ID,Editor_ID,Reconciler_ID". "**President**" is an Entity with Attributes "President_ID, Admin_ID, SSN, Name, Designation, Department, Department_ID, Email, Start_Date". "**Transaction**" is an Entity with Attributes "Transaction_ID, Editor_ID, Review_Status, Project, Cardholder_ID, Amount". "**Editor**" is an Entity with Attributes "Editor_ID, SSN, Name, Designation, Department, Department_ID, Email, Start_Date". "**Reconciler**" is an Entity with Attributes "Reconciler_ID, SSN, Name, Designation, Department, Department_ID, Email, Start_Date". "**Electronic_Record**" is an Entity with Attributes "eRecord_ID, Reconciler_ID".

The diagram denotes Primary Keys as attributes in Underline and Foreign Keys as **attributes** in Bold. The arrows in the diagram indicate the origin of the Foreign Key.

Brief explanation of the SQL queries

1: ***How many procurement cards were distributed to the employees in the last 6 months per department?***

The first query uses the "cardholder" table and selects the "Department", "datename(month, start_date)", and "count(cardholder_ID)" columns. The "where" clause filters out any rows where the department is null. The "group by" clause groups the results by department and the month in which the card was issued. The "order by" clause orders the results by department and the number of cards in descending order.

Query:

```
SELECT
        Department,
        datename(month, start_date)  Card_Issued_Month,
        count(cardholder_ID) Number_Of_Cards
FROM cardholder
        where department is not null
        group by Department, datename(month, start_date)
        order by Department, count(cardholder_ID) desc
```

Output:

	Department	Card_Issued_Month	Number_Of_Cards
1	Architecture	December	3
2	Architecture	February	2
3	Architecture	January	2
4	Architecture	November	1
5	Business	January	3
6	Business	February	2
7	Business	October	2
8	Business	November	1
9	Chemistry	February	3
10	Chemistry	December	2
11	Chemistry	November	2
12	Chemistry	October	1
13	English	November	2
13	English	November	2
14	English	January	2
15	English	December	2
16	English	February	2
17	English	October	1
18	History	October	3
19	History	December	2
20	History	November	2
21	History	January	1
22	History	February	1
23	Physics	December	4
24	Physics	October	2
25	Physics	February	1
26	Physics	November	1

2: ***Provide details for cardholders whose procurement card applications were rejected***

The second query uses the "cardholder" and "procurement_card_application" tables, joining them on the "cardholder_ID" column. The "where" clause filters out any rows where the application status is not "Rejected". The "group by" clause groups the results by department and application status.

Query:

```
SELECT
        Department,
        Application_Status,
        COUNT(application_ID) as NumberOfPeople
FROM cardholder ch
INNER JOIN procurement_card_application pca
        ON ch.cardholder_ID = pca.cardholder_ID
WHERE application_status = 'Rejected'
GROUP BY Department, Application_Status
```

Output:

	Department	Application_Status	NumberOfPeople
1	Architecture	Rejected	2
2	Business	Rejected	2
3	Chemistry	Rejected	4
4	English	Rejected	2
5	History	Rejected	3
6	Physics	Rejected	6

3: ***Departments which have an average spend limit higher than the average spend limit for the organization***

The third query uses the "cardholder" and "procurement_card" tables, joining them on the "cardholder_ID" column. The "group by" clause groups the results by department and calculates the average spend limit for

each department. The "having" clause filters out departments whose average spend limit is less than or equal to the overall average spend limit.

Query:

```
SELECT
        department, AVG(spend_limit) as Dep_Avg_Spend_Limit
FROM cardholder c INNER JOIN procurement_card pc
        ON c.cardholder_ID = pc.cardholder_ID
GROUP BY department
HAVING AVG(spend_limit) > (SELECT AVG(spend_limit) FROM procurement_card)
```

Output:

	department	Dep_Avg_Spend_Limit
1	Architecture	6121
2	History	6993

Design Controversies and further Decision:

During the design phase of our relational model based on the Enhanced Entity Relationship (EER) diagram, we identified a discrepancy in the inclusion of certain attributes such as Sender_ID, which was unnecessary for the purpose of procurement card usage, but rather intended for gathering user information. Consequently, we made the decision to eliminate such attributes from our design to simplify and optimize our database structure.

Upon initial design, we utilized a unique primary ID for each table and linked them to the Procurement_Card table via Procurement card ID. However, after further analysis, we discovered that all tables were already linked through foreign keys, and creating, linking these new IDs would lead to a more complicated design. So, we decided to abandon this approach & instead link only the Cardholder_ID with main table Procurement_Card, utilizing its ID as a foreign key.

Phase 3 (Neo4j)

The Team has worked collaboratively to convert the Database created in Phase 2 into CSV files, importing this database to Neo4j Database, and further creating nodes & relationships to gain deeper insights pertaining to procurement cards and cardholders. We have queried the Neo4j database for various insights regarding the business questions of Procurement card usage.

Queries for Creation of the Database in Neo4j:

```
LOAD CSV WITH HEADERS
FROM 'file:///Query_1.csv' AS row
WITH row WHERE row.Cardholder_ID IS NOT NULL
MERGE (d:Cardholder
{Cardholder_ID: row.Cardholder_ID,
Employee_Name: row.Employee_Name, Department:row.Department, Start_Date:row.Start_Date});

LOAD CSV WITH HEADERS
FROM 'file:///Query_2.csv' AS row
WITH row WHERE row.Cardholder_ID IS NOT NULL
MERGE (d:procurement_card_application
{Cardholder_ID:row.Cardholder_ID,
Application_ID: row.Application_ID,
Application_Status: row.Application_Status});

MATCH (c:Cardholder)
MATCH (p:procurement_card_application)
MERGE (c)-[:HAS_PROCUREMENT_CARD]->(p)
```

1: ***How many procurement cards were distributed to the employees in the last 6 months per department?***

```
MATCH (c:Cardholder)
WHERE c.Department IS NOT NULL
WITH c, date(datetime(c.Start_Date)) AS date
RETURN c.Department AS Department,
    date.month AS Card_Issued_Month,
    count(c.Cardholder_ID) AS Number_Of_Cards
ORDER BY Department, Number_Of_Cards DESC
```

This Cypher query, like the SQL query's WHERE clause, first filters out any Cardholder nodes that do not have a department property set. The date () method is then used to transform the start_date property to a Date type, and the month property is used to extract the month of the date. Finally, it groups the results by Department and Card_Issued_Month and sorts them in decreasing order by Department and Number_Of_Cards.

Because the date() method in Cypher demands a string in the format yyyy-mm-dd, you may need to change the format of the start_date field in your Cardholder nodes.

2: ***Provide details for cardholders whose procurement card applications were rejected***

```
MATCH (ch:Cardholder)-[:HAS_PROCUREMENT_CARD]->(pca:procurement_card_application)
WHERE pca.Application_Status = 'Rejected'
WITH ch.Department AS Department, pca.Application_Status AS Application_Status, COUNT(pca.Application_ID) AS NumberOfPeople
RETURN Department, Application_Status, NumberOfPeople
ORDER BY Department, NumberOfPeople DESC
```

This Cypher query first searches all Cardholder nodes that are attached to a procurement_card_application node with a HAS_PROCUREMENT_CARD relationship and filters out those procurement_card_application nodes whose Application_Status property is equal to 'Rejected'. The remaining procurement_card_application nodes are then grouped by Department and Application_Status, and the number of Application_ID values in each group is counted using the COUNT() method. Finally, it returns the Department, Application_Status, and NumberOfPeople values for each group and arranges the results in descending order by Department and NumberOfPeople.

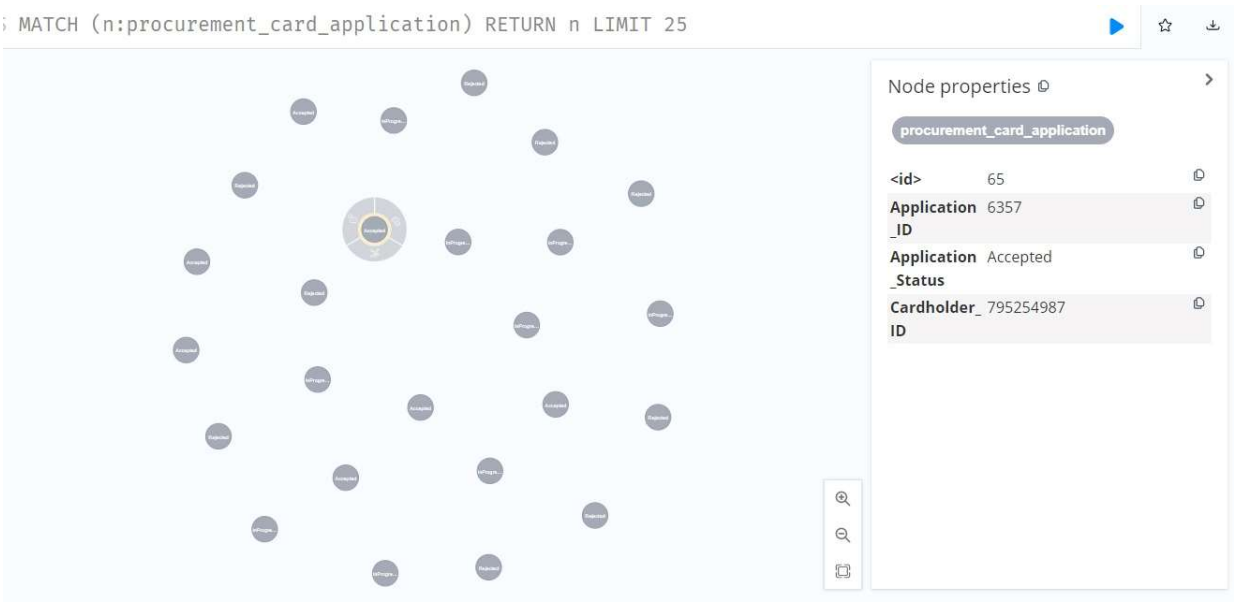
3: ***Departments which have an average spend limit higher than the average spend limit for the organization***

```
MATCH (c:Cardholder)-[:HAS_PROCUREMENT_CARD]->(pc:procurement_card)

WITH c.Department AS department, AVG(pc.Spend_Limit) AS Dep_Avg_Spend_Limit, AVG(pc.Spend_Limit) > AVG(pc.Spend_Limit) OVER () AS exceed_avg

WHERE exceed_avg = true

RETURN department, Dep_Avg_Spend_Limit

ORDER BY Dep_Avg_Spend_Limit DESC
```

This Cypher query first matches all Cardholder nodes that are connected to a procurement_card node with an HAS_PROCUREMENT_CARD relationship. It then groups the results by Department using the GROUP BY clause and calculates the average Spend_Limit for each group using the AVG() function. It also includes a calculated boolean value exceed_avg which is set to true if the average Spend_Limit of the current department is greater than the overall average Spend_Limit of all procurement cards. Finally, it filters the results based on exceed_avg = true using the WHERE clause, and returns the Department and Dep_Avg_Spend_Limit values for each group, ordered by Dep_Avg_Spend_Limit in descending order.

Note that the OVER() function is used with AVG(Spend_Limit) to calculate the average spend limit for all procurement cards, and is used to compare it with the average spend limit for each department.

	Department	Application_Status	NumberOfPeople
1	"Architecture"	"Rejected"	152
2	"Business"	"Rejected"	152
3	"Chemistry"	"Rejected"	152
4	"English"	"Rejected"	171
5	"History"	"Rejected"	171
6	"Physics"	"Rejected"	152



Controversies, Difficulties, and further Decision:

When we started importing our database, designing data model nodes & representing relationships was a little bit challenging since it's different from our traditional relational databases.

Neo4j does not provide detailed error messages during the import database process. So, debugging the simplest Cypher query took a while for us as detailed feedback wasn't present. Because of this, identifying & fixing the issue was challenging during the data import process.