## 1. Graph Database Modelling

### 1.1 Translating EER Model to Graph Model

The Enhanced Entity-Relationship (EER) model serves as the foundation for our graph model. In the graph model, entities become nodes, and the relationships between them become edges. Here's a breakdown of the entities and relationships:

**Nodes:**

* **Customer:** Represents individual customers with attributes such as id, name, branch\_id, and collector\_id.
* **User:** Represents users (collectors) with attributes such as id, name, and branch\_id.
* **Billing:** Represents billing records with attributes such as id, user\_id, total\_amount, and fiscal\_year\_id.
* **BillingDetail:** Represents details of each billing record with attributes like id, billing\_id, billing\_amount, and billing\_category\_id.
* **Branch:** Represents branches with attributes including id, branch\_title, branch\_code, status, etc.
* **CustomerCategory:** Represents categories for customers with attributes like id, category\_title, and status.
* **BillingCategory:** Represents categories for billing with attributes like id, title, and monthly\_charge.
* **FiscalYear:** Represents fiscal year records with an identifier.

**Relationships:**

* Customer -[

]-> CustomerCategory: Shows which category a customer belongs to.

* User -[

]-> Branch: Indicates the branch to which a user belongs.

* Billing -[

]-> BillingDetail: Connects billing records to their details.

* Billing -[

]-> FiscalYear: Links billing to its respective fiscal year.

* BillingDetail -[

]-> BillingCategory: Indicates the category of each billing detail.

* Customer -[

]-> Branch: Shows the branch associated with each customer.

* User -[

]-> Customer: Links users to the customers they manage.

* User -[

]-> Billing: Shows which user generated each billing record.

### 1.2 Graph Structure Visualization

To visualize this structure, a graph diagram is created using graph modeling tools such as Neo4j Desktop or draw.io. Each node is represented as a circle with labels (e.g., Customer, User), while the relationships are represented as arrows between these nodes.

## 2. Graph Database Implementation

### 2.1 Cypher Scripts for Data Migration

Cypher is the query language for Neo4j, used for creating nodes, establishing relationships, and querying the database.

#### Importing Data from CSV Files

To migrate data from a relational database to Neo4j, we use the following Cypher scripts to import data from CSV files:

LOAD CSV WITH HEADERS FROM "file:///User/beck/CDU/Semester3/Advanced Data Management/billing\_categories.csv" AS row

CREATE (:BillingCategory {id: row.id, title: row.title, default\_price: row.default\_price, monthly\_charge: row.monthly\_charge, status: row.status, created\_by: row.created\_by, updated\_by: row.updated\_by, created\_at: row.created\_at, updated\_at: row.updated\_at});

LOAD CSV WITH HEADERS FROM "file:/// User/beck/CDU/Semester3/Advanced Data Management/customer\_category.csv" AS row

CREATE (:CustomerCategory {id: row.id, category\_title: row.category\_title, status: row.status, created\_at: row.created\_at, updated\_at: row.updated\_at});

LOAD CSV WITH HEADERS FROM "file:/// User/beck/CDU/Semester3/Advanced Data Management/branches.csv" AS row

CREATE (:Branch {id: row.id, branch\_title: row.branch\_title, status: row.status, created\_at: row.created\_at, updated\_at: row.updated\_at, branch\_code: row.branch\_code});

LOAD CSV WITH HEADERS FROM 'file:/// User/beck/CDU/Semester3/Advanced Data Management/customers.csv' AS row

CREATE (c:Customer {id: row.id, name: row.name, branch\_id: row.branch\_id, collector\_id: row.collector\_id});

LOAD CSV WITH HEADERS FROM 'file:/// User/beck/CDU/Semester3/Advanced Data Management/users.csv' AS row

CREATE (u:User {id: row.id, name: row.name, branch\_id: row.branch\_id});

LOAD CSV WITH HEADERS FROM 'file:/// User/beck/CDU/Semester3/Advanced Data Management/billing.csv' AS row

CREATE (b:Billing {id: row.id, user\_id: row.user\_id, total\_amount: row.total\_amount, fiscal\_year\_id: row.fiscal\_year\_id});

LOAD CSV WITH HEADERS FROM 'file:/// User/beck/CDU/Semester3/Advanced Data Management/billing\_details.csv' AS row

CREATE (bd:BillingDetail {id: row.id, billing\_id: row.billing\_id, billing\_amount: row.billing\_amount, billing\_category\_id: row.billing\_category\_id});

#### Creating Relationships

Once the nodes are created, we establish relationships between them using additional Cypher scripts.

// Customer to CustomerCategory

MATCH (c:Customer), (cc:CustomerCategory) WHERE c.customer\_category\_id = cc.id

CREATE (c)-[:BELONGS\_TO]->(cc);

// User to Branch

MATCH (u:User), (b:Branch) WHERE u.branch\_id = b.id

CREATE (u)-[:COLLECTOR\_BELONGS\_TO\_BRANCH]->(b);

// Billing to BillingDetail

MATCH (b:Billing), (bd:BillingDetail) WHERE b.id = bd.billing\_id

CREATE (b)-[:HAS\_DETAIL]->(bd);

// Billing to FiscalYear

MATCH (b:Billing), (f:FiscalYear) WHERE b.fiscal\_year\_id = f.id

CREATE (b)-[:BELONGS\_TO\_FISCAL\_YEAR]->(f);

// BillingDetail to BillingCategory

MATCH (bd:BillingDetail), (bc:BillingCategory) WHERE bd.billing\_category\_id = bc.id

CREATE (bd)-[:BELONGS\_TO\_CATEGORY]->(bc);

// Customer to Branch

MATCH (c:Customer), (b:Branch) WHERE c.branch\_id = b.id

CREATE (c)-[:BELONGS\_TO\_BRANCH]->(b);

// User to Customer

MATCH (u:User), (c:Customer) WHERE u.id = c.collector\_id

CREATE (u)-[:COLLECTOR\_HAS\_CUSTOMER]->(c);

// User to Billing

MATCH (u:User), (b:Billing) WHERE u.id = b.user\_id

CREATE (u)-[:COLLECTOR\_GENERATE\_BILL]->(b);

**2.2 Creating Indexes**

To improve query performance, we create indexes on frequently accessed fields. The following scripts set up these indexes:

CREATE INDEX FOR (c:Customer) ON (c.id);

CREATE INDEX FOR (u:User) ON (u.id);

CREATE INDEX FOR (b:Billing) ON (b.id);

CREATE INDEX FOR (bd:BillingDetail) ON (bd.id);

CREATE INDEX FOR (f:FiscalYear) ON (f.id);

CREATE INDEX FOR (bc:BillingCategory) ON (bc.id);

CREATE INDEX FOR (cc:CustomerCategory) ON (cc.id);

CREATE INDEX FOR (br:Branch) ON (br.id);

**3. Graph Database Queries**

After setting up the database, various queries can be executed to retrieve meaningful insights.

#### Use Case 1: Total Billing per Branch

**Description:** This query retrieves the total billing amount for each branch along with the names of customers associated with that branch.

MATCH (c:Customer)-[:BELONGS\_TO\_BRANCH]->(b:Branch)

MATCH (c)-[:HAS\_BILL]->(bill:Billing)

WITH b, c, SUM(toFloat(bill.total\_amount)) AS TotalBilling

RETURN b.branch\_title AS BranchName,

COLLECT(c.name) AS CustomerNames,

TotalBilling

ORDER BY TotalBilling DESC;

**Expected Result:** A list of branches with their total billing amounts and the corresponding customers.

#### Use Case 2: Top Customers per Branch

**Description:** This query finds the top three customers with the highest billing amounts for each branch.

MATCH (b:Branch)<-[:BELONGS\_TO\_BRANCH]-(c:Customer)-[:HAS\_BILL]->(bill:Billing)

WITH b, c, SUM(toFloat(bill.total\_amount)) AS TotalBilling

ORDER BY b.id, TotalBilling DESC

WITH b,

COLLECT(c.name) AS Customers,

COLLECT(TotalBilling) AS Billings

RETURN b.branch\_title AS BranchName, [i IN RANGE(0, 2) | {name: Customers[i], amount: Billings[i]}] AS TopCustomers;  
  
**Expected Result:** Branch names with the top three customers by billing amounts.

#### Use Case 3: Total Customers per Collector

**Description:** This query provides a count of customers managed by each collector.

cypher

#### MATCH (u:User)-[:COLLECTOR\_HAS\_CUSTOMER]->(c:Customer)

RETURN u.name AS CollectorName, COUNT(c) AS TotalCustomers

ORDER BY TotalCustomers DESC;

**Expected Result:** List of collectors with the number of customers they handle.

#### Use Case 4: Billing Amount per Category

**Description:** This query aggregates total billing amounts per billing category.

MATCH (bd:BillingDetail)-[:BELONGS\_TO\_CATEGORY]->(bc:BillingCategory)

RETURN bc.title AS BillingCategory, SUM(toFloat(bd.billing\_amount)) AS TotalBillingAmount

ORDER BY TotalBillingAmount DESC;  
  
**Expected Result:** A summary of billing categories with total amounts for each.