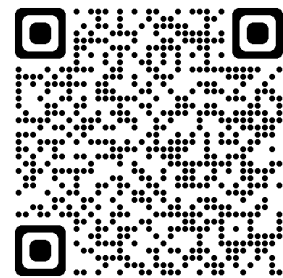


# Project Comprehensive



*CS232-Database Management Systems*

*DATE: 04/06/2022*

*Group 6*

*Muhammad Faraz (2020292)*

*Muhammad Umer Asif (2020372)*

*Muhammed Hamza Malik (2020382)*

*Submitted to: Mr. Muhammad Mohsin Zafar*

## Development Description

Our group decided to implement the project on an online bank management system case-study. Throughout the duration of the project, we acquired new skills handling software and programming languages we had not used before. In order to get started, we consulted tutorials available online to better understand conceptually what was to be achieved in our project and more so importantly, how we would implement it.

For backend, we settled on PostgreSQL due to its widespread usage in industry and its ease of use. After the database had been conceptually constructed using EER, the design was mapped to a relational schema, which was then used to create the tables alongside their constraints in Postgres using its pgAdmin 4 user interface. For the sake of backend logic and its completeness, we populated the tables with dummy data and defined procedures, functions, triggers/trigger functions, indexes, and views where we felt it was appropriate to do so. DataGrip was used for writing PostgreSQL related queries and statements while Visual Studio Code (VS Code) was used to manage rest of the project files.

For the intermediate business layer, Python's microframework Flask was used in conjunction with an Object-Relational Mapping (ORM) called SQLAlchemy. Although Postgres was relatively easier to learn, this part of the project was arguably where we faced the most challenges. This was mainly due to the team never having done a project in the past which required multiple layers to its implementation. Figuring out which libraries to import such as psycopg2 and figuring out the overall set up to connect to our intermediate layer to the database layer took some time initially.

For the frontend application layer, we used HTML and CSS to display the website and to decide its appearance, styling, and layout. This part of the project was not as lengthy as the intermediate layer was and understanding the basics of HTML and CSS did not take too much time after we got the hang of the foundations. Later, we attempted to make use of Bootstrap in order to enhance the overall look of our website and modify its appearance to be more user friendly.

Overall, our group worked well as a team. The tasks of the project were subdivided into portions that each member could independently work on. Owing to this project, we learned all the significant practices of project management with your team and how the use of Git and Github facilitates the collaboration process. Majority of the issues faced during the project were related to bugs in programs and connection problems between the different layers of the application. By working hard with the intent to learn as much as possible, we were able to solve these problems by consulting the help available online such as documentation of languages/tools, websites such as Stack Overflow, and learning to use the debugger of the Integrated Development Environment (IDE) more efficiently. The project underwent design changes continually in almost all stages of the implementation; however, we believe our team was able to develop a working application up to the mark of what we had envisioned to implement.

# Central Bank Management Portal (CBMP)

*CS232-Database Management Systems*

*DATE: 21/03/2022*

*Group 6*

*Muhammad Faraz (2020292)*

*Muhammad Umer Asif (2020372)*

*Muhammed Hamza Malik (2020382)*

*Submitted to: Mr. Muhammad Mohsin Zafar*

# Synopsis

In this course we have learned the fundamental constituents of a database management system and its uses. We believe this knowledge will be suitable in implementing a fully functional Central Bank Management Portal (CBMP) that will serve account holders of all categories and serve multiple uses as outlined later in this document.

Modern banking systems fulfill the need to digitize financial transactions of large-scale datasets of vast number of customers as efficiently as possible. In addition, the use of a database system is substantiated by the requirements of CBMP such as concurrency access and the miniworld described reflects real-world banking systems that are used ubiquitously in our society. Our goal is to develop a banking system that checks all the functionality requirement boxes defined in the 'Functionality Requirements' Section.

In summary, the CBMP has been proposed as the hallmark project idea to be implemented by Group 6 members. Due to the extensive handling of data in the form of retrievals, updates, deletions and insertions required in this case-study, the team believes the miniworld would be a suitable scenario to apply our newly acquired knowledge of database systems alongside a cohort of complementary knowledge pertaining to web development and frontend programming in general.

## User Story

The interface of the project will be implemented on a web application program, that will enable all users of CBMP to interact with the system through their own personal login accounts, and specific privileges will be administered to various users depending on their status and category of financial needs.

- There is a bank branch having the following information stored in it: name of that branch, branch code, address with unique code assigned to each bank branch.
- Administrator level privileges are given to the bank branch manager who has the highest-level privilege to the database and can delete the account of any person and update any parameter of that database.
- There will be other bank staff such as clerks who are supervised by bank managers who by themselves are also part of the bank staff team. Bank staff IT managers, receptionist, chartered accountant and will have their own names, bank assigned id, date of birth, age, gender, manager/supervisor id and category of role assigned.
- Account holders/customers will have their unique IBAN assigned to them, alongside their other attributes such as date of birth, gender, age, phone number, email, address.
- There are multiple types of distinct accounts of the person that can be created such as a Savings (stores minimum required balance and interest rate), Current (stores deposit limit). Information stored includes total balance, bank account history, transaction details, the billing information, installments etc.
- The customer has many categories of distinct loans from the bank to complete his life requirements such as purchasing a home, car and for students for their education. In these cases, the home address, car model, and student's duration of loan repayment will be recorded respectively. Banks also take records of that person and other information such as loan numbers and the amount. There are several modes of payment to pay the loan such as

paying installment wise or paying the entire loan in one installment. It stores the information of the person such as payment type whether the customer wants to pay the loan through cash or to pay the loan through cheque and the other necessary information such as payment date, payment time, loan number, and the total amount paid.

- The person can perform transactions which are a weak entity and identified by the account it was performed for. It stores payment date, payment time, account debit, and account credit amount depending upon receipt or transfer of funds.

## Functional Requirements

Tentatively, the plan of the project is to follow the standard procedure of database system development in which the requirements and functionality analysis is performed first, followed by the conceptual, logical and physical database modelling and design.

CBMP will provide real-time Online Transaction Processing (OLTP) to users in an integrated platform of 3-tier architecture model in which the client will access web interface, the intermediate web server will act as the conduit through which database and user data flows through, and the database server will seamlessly and securely safekeep the user data.

Our MVP (Minimum Value Product) is an ideal web application that at the very least conforms to modern implementation strategies of database systems in which the following considerations are dealt with fully:

- Because the data stored in the CBMP must be retained for indefinitely long periods of time, the database implementation is the perfect solution.
- Moreover, multiple users are expected to access the database to process their own transactions simultaneously, thus the concurrent user access property of CBMP is another asset.
- Information to be stored in CBMP is a type of formatted homogeneous data made up of components related and associated with each other, necessitating the use of a database system.
- Debatably the most important feature of CBMP is that data will be processed in such a manner that integrity constraints would be enforced on the data storage mechanisms to guarantee correctness, validity, and consistency of private information. Without such private user information, there would be a higher risk of user data getting corrupted.

Due to its application driven nature, CBMP will provide dynamic views based on the category of user that accesses the system. Broadly speaking, there will be an account holder view in which the interface is tailored to meet the functional requirements of the user such as login, validation, get balance information, withdrawal of money, money transfer, set beneficiary, transaction history, and much more.

Administrative users on the other hand will have most if not all the access privileges of the account holders whilst also being able to create, read, update, and delete (i.e., CRUD operations) user account information.

## Technology Stack

For the purpose of implementing our project, a repertoire of software applications and technologies is expected to be used. Although the list is not exhaustive, it encapsulates what the team believes will enable us to build an exceptional product meeting all the enumerated functional requirements.

The tentative list of software to be included in the project implementation is not limited to:

BACKEND: **Postgres** or **MySQL** (for the backend relational DBMS programming), **Python** (in order to make use of its microframeworks such as **Flask** or high-level frameworks like **Django** and to process the database information entered the program by Object-Relational Mapping or ORM), etc.

FRONTEND: **Javascript** (to create and design dynamic web pages for the sake of our web applications), **HTML/CSS/SCSS** (for webpage construction and its styling of webpages), etc.

MISCELLANEOUS: **Visual Studio Code** (to fulfill IDE purposes), **Git Bash & Github** (to organize the individual contributions of the team and enhance organization), etc.

## **Team Github Repository Link**

*Project Hosting Link:* <https://github.com/CS-232>

# Central Bank Management Portal (CBMP)

*Assignment 1: Enhanced Entity-Relationship Diagram (EERD)*

*CS232-Database Management Systems*

*DATE: 09/05/2022*

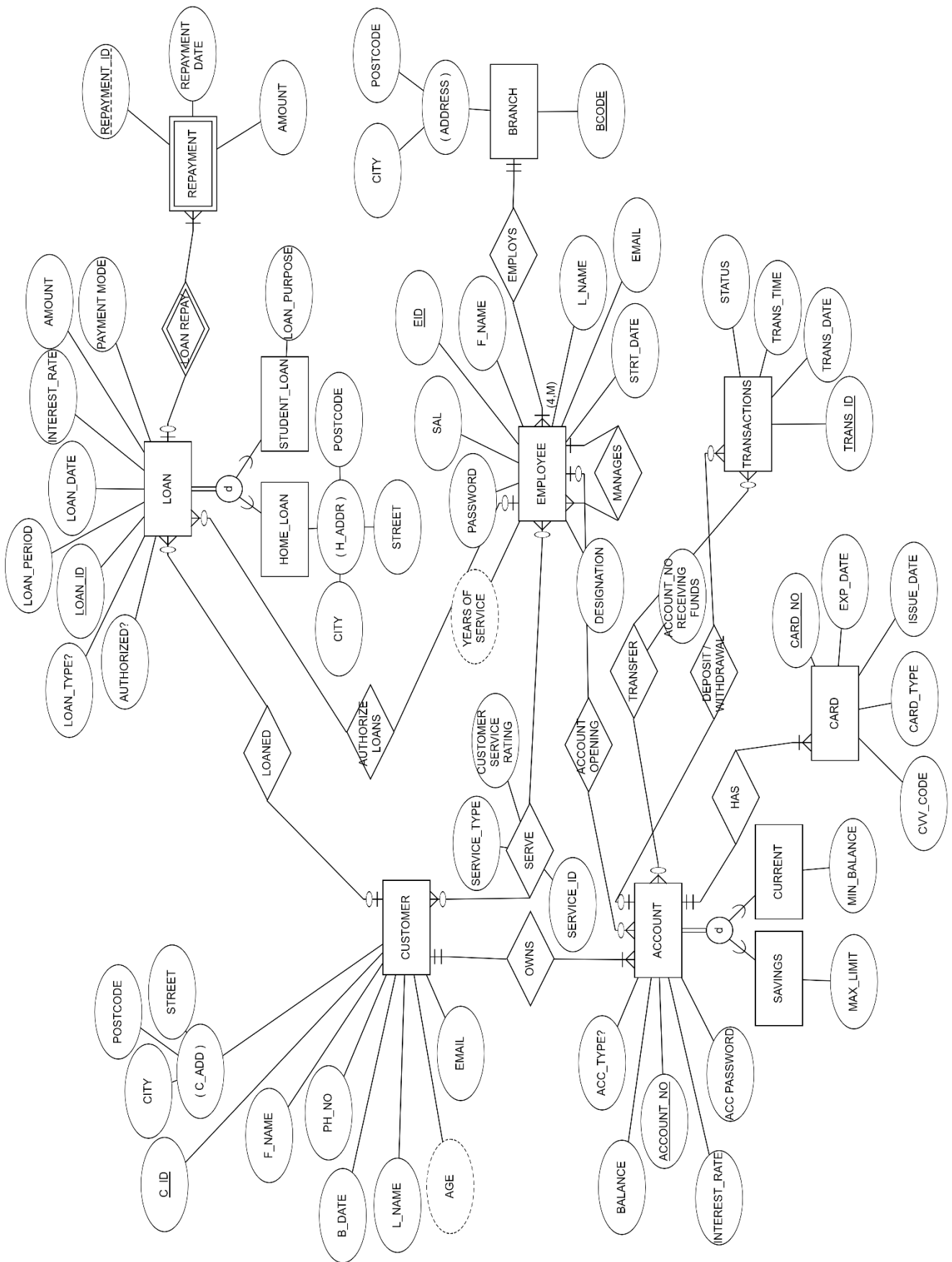
*Group 6*

*Muhammad Faraz (2020292)*

*Muhammad Umer Asif (2020372)*

*Muhammed Hamza Malik (2020382)*

*Submitted to: Mr. Muhammad Mohsin Zafar*





## ***Relationships:***

- LOANED: One customer can apply for many loans (1:M).
- AUTHORIZE LOANS: One employee can authorize multiple loans, but a single loan can only be authorized by one employee (1:M).
- LOAN REPAY: For one loan, there can be multiple repayments, but a particular repayment can only be intended for a single loan (1:M).
- OWNS ACCOUNT: One customer can own many bank accounts in the bank, but a particular account cannot be linked with more than one customer (1:M).
- HAS CARD: One account can have multiple credit or debit cards affiliated but a particular card cannot be linked to multiple accounts (1:M).
- DEPOSIT/WITHDRAWAL: One account can have multiple money related deposits or withdrawals but a particular transaction with a unique transaction id can be traced to a single bank account that initiated the request (1:M).
- TRANSFER: One account can place multiple money transfer transactions and one particular transaction can be tied to more than one bank account (i.e. the bank account from which the money was sent, and the bank account receiving the money) (M:N).
- ACCOUNT OPENING: One employee can open many accounts but not the other way around (1:M).
- SERVE: One customer can be served by many employees and one employee can serve many customers (M:N).
- MANAGES: One bank manager can manage many other subordinate employees but not the other way around (1:M).
- EMPLOYS: One bank branch employs at least 4 employees but a single employee cannot work across multiple branches (1:M).

## ***Please Note:***

- The EERD diagram was created using erdplus.com
- Crow's Foot notation has been used to denote the cardinalities on the EERD diagram.
- Some attribute names may have been deliberately shortened to enhance the tidiness of the diagram (e.g. employee\_id written as eid).

## ***Assumptions:***

- Current accounts are assumed to have no limit to their daily transaction and do not require a minimum balance to be always maintained, in contrast to savings accounts which do require a minimum balance threshold.
- Customers are assumed to reside at a single residence without multivalued option for home addresses; a customer must choose one location to be their main address registered with their accounts on the database.

# Central Bank Management Portal (CBMP)

*Assignment 2: Relational  
Schema and Normalization*

*CS232-Database Management Systems*

*DATE: 10/05/2022*

*Group 6*

*Muhammad Faraz (2020292)*

*Muhammad Umer Asif (2020372)*

*Muhammed Hamza Malik (2020382)*

*Submitted to: Mr. Muhammad Mohsin Zafar*

## Introduction

In assignment 2, we converted the EER diagram completed in assignment 1 and converted the said conceptual design into a relational schema. The relations, primary/foreign keys, domain constraints have been outlined.

Afterwards, normalization was pursued where necessary until the 3<sup>rd</sup> Normal Form which assures lossless and dependency preserving decomposition (unlike Boyce-Codd Normalization which does not assure dependency preservation. With the finalized schema, six sample records for each relation have been defined and listed accordingly.

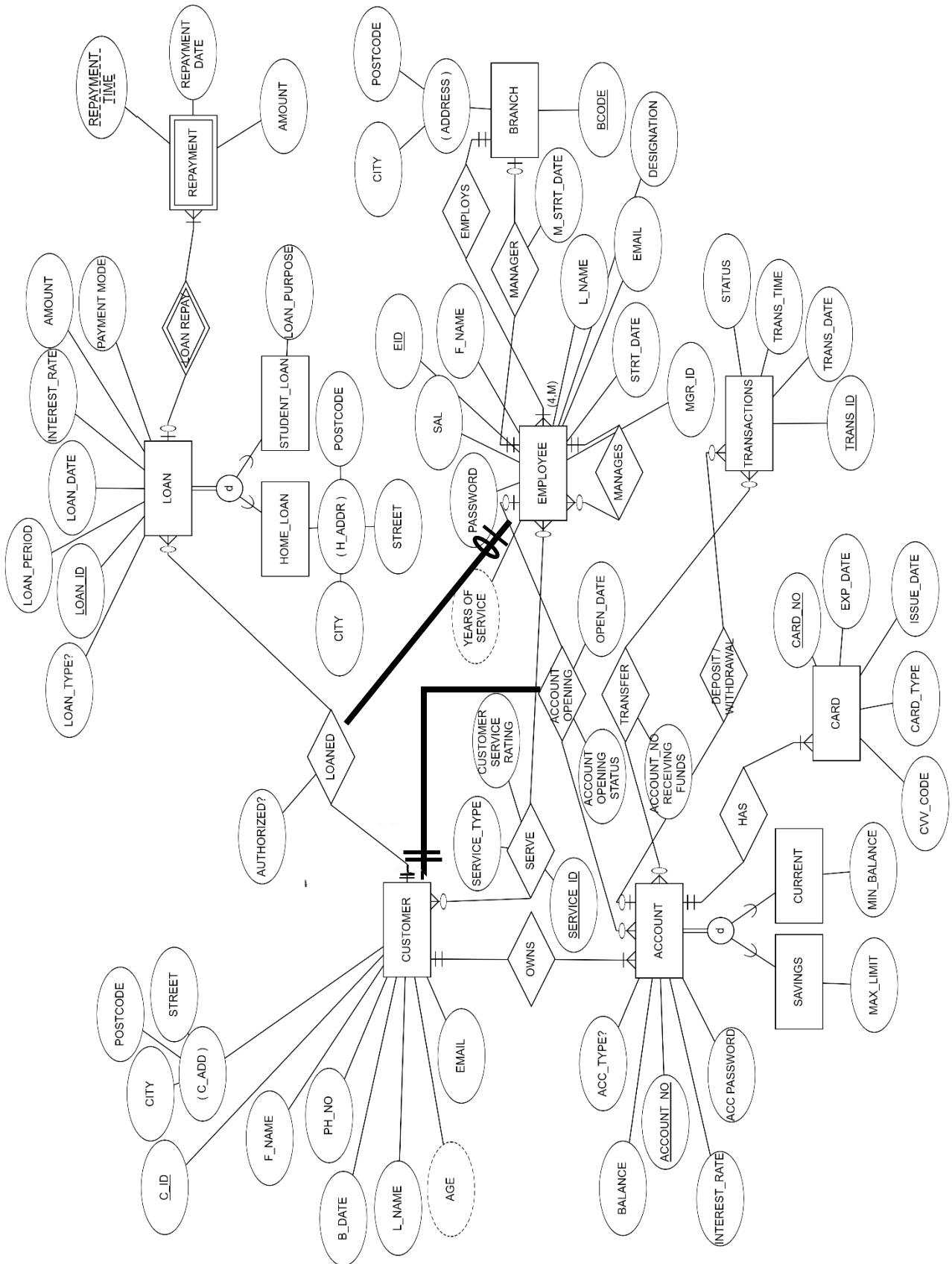
## Minor Updates to the EER Conceptual Design

When we were completing assignment 1, we encountered an issue at erdplus.com where we faced problems when trying to add tertiary relationships to our conceptual design. As a result, we could not add relationships interlinking more than 2 entities. The changes made have been listed below alongside the updated diagram on the next page. Some other issues also noticed now were amended as listed below:

- Employee entity was given an attribute MGR\_ID.
- We added a relationship in assignment 1 between the entities BRANCH and EMPLOYEE called MANAGER. The relationship was given an attribute M\_STRT\_DATE as well. *Every bank branch has exactly one bank manager who is an employee but not every employee has the designation of a bank manager and if they do, they can only manage one branch at a time (1:1).*
- The weak entity REPAYMENT has an attribute repayment\_id, which in reality makes it a strong entity. The attribute was replaced with repayment\_time to maintain the weak entity relationship it has with loan.
- The relationships LOANED and AUTHORIZE LOANS were replaced by a single relationship LOANING linking customers, employees, and the loans. The Boolean attribute named 'Authorized?' for the loan entity was instead made this relationship's attribute. *One customer can place request for many loans to a particular employee.*
- The relationship ACCOUNT OPENING was changed from a binary relationship to a tertiary relationship linking customers, employees, and accounts. Attributes to keep track of account opening status an account opening date were added to the relationship. *One customer can request to open multiple accounts and each account was authorized to be opened by a particular employee.*

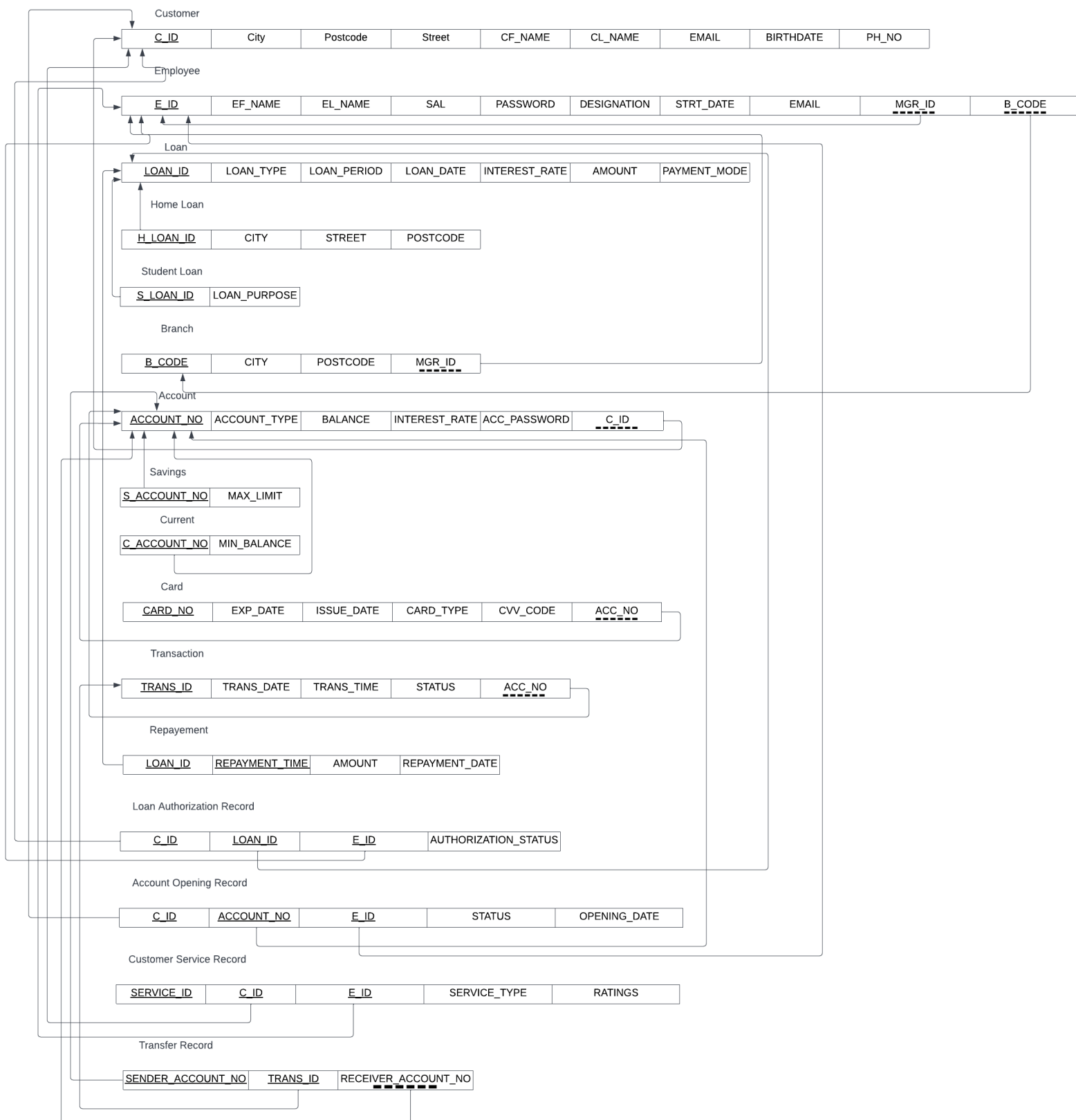
The updated EER diagram has been shown below. Please note that in order to make it possible to visually draw tertiary relationships, MS Word's line shapes were used to overlay the EER diagram to counteract the aforementioned issue.

## Updated EER Diagram



# Relational Schema Diagram

The relational schema was created based on the updated EER diagram and it is displayed below.



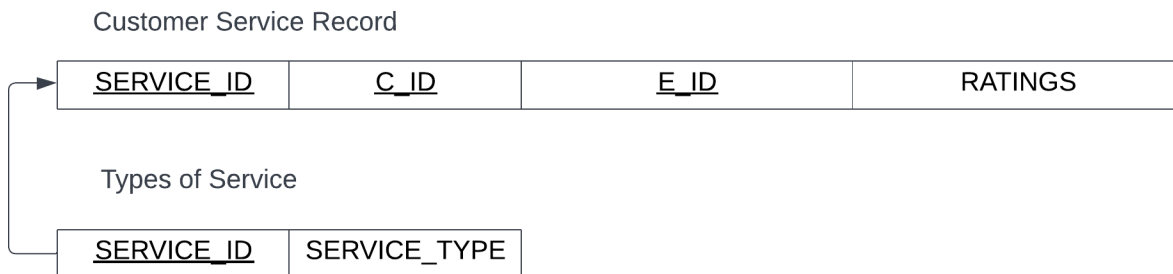
# Normalization

## 1<sup>st</sup> Normal Form:

All relational schema are automatically in the first normal form.

## 2<sup>nd</sup> Normal Form:

All relations fulfill the requirement of 2NF except 'Customer Service Record', since the key Service\_id is part of the composite key but it can fully identify on its own the service\_type attribute; thus the relation is split into two tables as shown below. Please note that only changes have been shown below and the rest of the relational schema remains same as before.



## 3<sup>rd</sup> Normal Form:

All relations fulfill the 3NF except those with the attributes CITY and POSTCODE, since the nonprime attribute POSTCODE can identify the nonprime attribute CITY on its own. The relations with this problem are 'Customer', 'Branch', and 'Home Loan'. The 3 relations are restructured as below with the inclusion of new relation 'Postal Codes'. As mentioned before, the rest of the diagram remains the same. Only the new relations and attributes are outlined below.



## Tables/Relations

The below table summarizes the relations, their primary and foreign keys, and domain of each attribute has been described in brackets.

<i>Relation Name</i>	Primary Key (PK)	Foreign Keys (FK)	Other Attributes
<b>Customer</b>	C_ID (number)	---	CITY (string, major cities in Pakistan), POSTCODE (number), STREET (string), CF_NAME (string), CL_NAME (string), EMAIL (string), BIRTHDATE (date), PH_NO (string)
<b>Employee</b>	E_ID (number)	BRANCH_CODE (number, only the numbers that match registered branch codes)  MGR_ID (number)	EF_NAME (string), EL_NAME (string), SAL (number), PASSWORD (string), DESIGNATION (string), STRT_DATE (date), EMAIL (string)
<b>Loan</b>	LOAN_ID (number)	---	LOAN_TYPE (string, either home or student loan), LOAN_PERIOD (number), LOAN_DATE (date), INTEREST_RATE (number), AMOUNT (number), PAYMENT_MODE (string, can be cheque or cash)
<b>Home Loan</b>	H_LOAN_ID (number)	---	CITY (string, major cities in Pakistan), STREET (string), POSTCODE (number)
<b>StudentLoan</b>	S_LOAN_ID (number)	---	LOAN_PURPOSE (string)
<b>Branch</b>	B_CODE (number)	MGR_ID (number)	CITY (string, major cities), POSTCODE (number)
<b>Account</b>	ACCOUNT_NO (number)	C_ID (number)	ACCOUNT_TYPE (string), BALANCE (number), INTEREST_RATE (number), ACC_PASSWORD (string)
<b>Savings</b>	S_ACCOUNT_NO (number)	---	MAX_LIMIT (number)
<b>Current</b>	C_ACCOUNT_NO (number)	---	MIN_BALANCE (number)
<b>Card</b>	CARD_NO (number)	ACC_NO (number)	EXP_DATE (date), ISSUE_DATE (date), CARD_TYPE (string), CVV_CODE (number)
<b>Transaction</b>	TRANS_ID (number)	ACC_NO (number)	TRANS_DATE (date), TRANS_TIME (time), STATUS (string, e.g. 'processed')
<b>Repayment</b>	LOAN_ID (number), REPAYMENT_TIME (time)	---	AMOUNT (number), REPAYMENT_DATE (date)
<b>Loan Auth. Record</b>	C_ID (number), LOAN_ID (number), E_ID (number)	C_ID (number), LOAN_ID (number), E_ID (number)	AUTHORIZATION_STATUS (string)

<b>Account Auth. Record</b>	C_ID (number), ACCOUNT_NO (number), E_ID (number)	C_ID (number), ACCOUNT_NO (number), E_ID (number)	STATUS (string), OPENING_DATE (date)
<b>Customer Service Record</b>	SERVICE_ID (number), C_ID (number), E_ID (number)	C_ID (number), E_ID (number)	SERVICE_TYPE (string), RATINGS (number, on a scale of 1 to 5)
<b>Transfer Record</b>	SENDER_ACCOUNT_NO (number), TRANS_ID (number)	SENDER_ACCOUNT_NO (number), TRANS_ID (number), RECEIVER_ACCOUNT_NO (number)	---

## Sample Records for Each Relation

CUSTOMER (, POSTCODE, STREET, C\_ID, F\_NAME, L\_NAME, EMAIL, B\_DATE, PH\_NO)

```
(4400,13,1,'JAVED','AWAN','JAVED_AWAN@GMAIL.COM');
(4445,2,2,'IQBAL','TAHIR','IQBAL_TAHIR@GMAIL.COM');
(46000,12,3,'HARIS','GHAFOOR','HARIS_GHAFOOR@GMAIL.COM');
(47040,14,4,'FAISAL','ANWAR','FAISAL_ANWAR@GMAIL.COM');
(25200,3,5,'IMRAN','HASHMI','IMRAN_HASHMI@GMAIL.COM');
(60001,16,6,'NASIR','SOHAIL','NASIR_SOHAIL@GMAIL.COM');
```

EMPLOYEE (EID, F\_NAME, L\_NAME, SAL, PASSWORD, DESIGNATION, STRT\_DATE, EMAIL, BCODE, MANAGES\_EID)

```
(1,'IZHAR','AWAN',82000,4322,'PHONE OPERATOR',21/2/2022,'IZHARAWAN@gmail.com',444,NULL);
(2,'ZAIN','INAM',79000,4342,'FINANCE ANALYST',22/4/2021,'ZAININAM@gmail.com',444,NULL);
(25,'HAMZA','HASHMI',50000,3433,'CHARTERED ACCOUNTANT',19/3/2020,'HAMZAHASHMI@gmail.com',444,1);
(26,'AMIR','SULTAN',48000,4564,'MANAGER',3/8/2020,'AMIRSULTAN@gmail.com',444,1);
(27,'AZHAR','AWAN',43000,7567,'COUNTER AGENT',4/4/2022,'AZHARAWAN@gmail.com',444,1);
(28,'ASIM','MALIK',80000,4564,'FINANCE ANALYST',4/9/2022,'ASIMMALIK@gmail.com',444,1);
(29,'SAIFULLAH','AWAN',82000,6734,'PHONE OPERATOR',5/2/2022,'SAIFULLAHAWAN@gmail.com',466,1);
(30,'MUSA','INAM',79000,5784,'FINANCE ANALYST',12/4/2021,'MUSAINAM@gmail.com',466,2);
(31,'QASIM','HASHMI',50000,8946,'CHARTERED ACCOUNTANT',18/5/2020,'QASIMHASHMI@gmail.com',466,2);
(32,'SAIM','SULTAN',48000,4532,'MANAGER',3/5/2020,'SAIMSULTAN@gmail.com',466,2);
(33,'SHERYAR','AWAN',43000,6543,'COUNTER AGENT',2/9/2022,'SHERYARAWAN@gmail.com',466,2);
(34,'ABDULLAH','MALIK',80000,7489,'FINANCE ANALYST',1/9/2022,'ABDULLAHMALIK@gmail.com',466,2);
```

LOAN (LOAN\_ID, AMOUNT, PAYMENT\_MODE, INTEREST\_RATE, LOAN\_PERIOD, LOAN\_DATE, LOAN\_TYPE, AUTHORIZED, C\_ID, EID)

```
(233,10000,'ONLINE',5,12,4/3/2022,'HOME',TRUE,1,16);
```



```
(234,23000,'ONLINE',6.5,5/2/2022,'STUDENT',TRUE,1,16);
(235,20000,'CASH',7,30/3/2022,'HOME',TRUE,2,16);
(236,30000,'CASH',5,4/1/2022,'STUDENT',FALSE,3,16);
(237,35000,'ONLINE',8.5,8/6/2022,'HOME',TRUE,2,11);
(238,90000,'CASH',6,2/6/2022,'STUDENT',TRUE,5,10);
```

#### HOME\_LOAN (STREET,POSTCODE,LOAN\_ID)

```
(13,4400,233);
(2,4445,234);
(12,46000,235);
(14,47040,236);
(3,25200,237);
(16,60001,238);
```

#### STUDENT\_LOAN (LOAN\_PURPOSE, LOAN\_ID)

```
('FOR PAYMENT OF LAB EQUIPMENT',233);
('PAYING FOR INTERNSHIP APPLICATIONS',234);
('LACK OF EDUCATIONAL RESOURCES SUCH AS TEXTBOOKS',235);
('FOR MEDICAL TREATMENT',236);
('FOR PAYMENT OF TUITION FEE',237);
('FOR DORMITORY/ACCOMMODATION FEES',238);
```

#### BRANCH (B\_CODE,POSTCODE, MGR\_ID)

```
(444,4400,1);
(466,4445,2);
```

#### ACCOUNT (ACCOUNT\_NO, BALANCE, ACC\_TYPE, ACC\_PASSWORD, INTEREST\_RATE, C\_ID, EID)

```
(1123,4334,'CURRENT',5444,6,1,11);
(1124,5454,'CURRENT',3282,5,2,12);
(1125,3455,'SAVING',9334,8,3,13);
(1126,5444,'CURRENT',3554,7,4,14);
(1127,9999,'SAVING',5454,5,5,15);
(1128,9000,'CURRENT',4593,7,6,16);
```

#### SAVINGS (MAX\_LIMIT,ACCOUNT\_NO)

```
(10000000,1123);
(10000000,1124);
(10000000,1125);
(10000000,1126);
(10000000,1127);
(10000000,1128);
```

#### CURRENT(MIN\_BALANCE,ACCOUNT\_NO)

```
(100,1123);
(100,1124);
```

```
(100,1125);  
(100,1126);  
(100,1127);  
(100,1128);
```

CARD (CARD\_NO, EXP\_DATE, ISSUE\_DATE, CARD\_TYPE, CVV\_CODE, ACCOUNT\_NO)

```
(89324327,2/9/2026,2/9/2022,'CREDIT',9123,1123);  
(89324328,2/9/2026,2/9/2022,'DEBIT',8934,1124);  
(89324329,2/9/2026,2/9/2022,'CREDIT',3456,1125);  
(89324330,2/9/2026,2/9/2022,'DEBIT',9304,1126);  
(89324331,2/9/2026,2/9/2022,'CREDIT',9035,1127);  
(89324332,2/9/2026,2/9/2022,'CREDIT',8945,1128);
```

TRANSACTION (TRANS\_ID,TRANS\_DATE,TRANS\_TIME,STATUS,ACCOUNT\_NO)

```
(100,3/5/2020,'00:00:00','ACCEPTED',1123);  
(101,4/5/2020,'00:00:00','PROCESSED',1124);  
(102,4/6/2020,'00:00:00','PENDING',1125);  
(103,4/7/2020,'00:00:00','ACCEPTED',1126);  
(104,6/9/2020,'00:00:00','ACCEPTED',1127);  
(105,4/9/2020,'00:00:00','ACCEPTED',1128);
```

REPAYMENT (REPAYMENT\_ID,REPAYMENT\_DATE,AMOUNT,LOAN\_ID)

```
(200,5/10/2022,10000,233);  
(201,6/12/2022,2200,234);  
(202,4/12/2022,8000,235);  
(203,5/12/2022,5000,236);  
(204,6/4/2022,9000,237);  
(205,8/4/2022,10000,238);
```

SERVICE RECORD (SERVICE\_TYPE, SERVICE\_ID, CUSTOMER\_SERVICE\_RATING, C\_ID,EID)

```
('ACCOUNT MANAGEMENT',400,5,1,11);  
( 'LOAN MANAGEMENT',401,4,2,12);  
( 'GENERAL INFORMATION',402,3,3,13);  
( 'GENERAL INFORMATION',403,5,4,14);  
( 'CARD RELATED',404,5,5,15);  
( 'ACCOUNT MANAGEMENT',405,5,6,16);
```

TYPES OF SERVICE (SERVICE\_ID, SERVICE\_TYPE)

```
(400,'GENERAL ACCOUNT ENQUIRY');  
(401,'LOAN MANAGEMENT');  
(402,'GENERAL INFORMATION');  
(403,'CUSTOMER FEEDBACK');  
(404,'CARD RELATED');  
(405,'ACCOUNT CLOSURE');
```

## TRANSFER RECORD

(ACCOUNT\_NO\_RECEIVING\_FUNDS,TRANS\_ID,ACCOUNT\_NO)

```
(1123,233,1124);  
(1124,234,1126);  
(1125,235,1127);  
(1126,236,1128);  
(1127,237,1123);  
(1128,238,1125);
```

## LOAN AUTHORIZATION RECORD (C\_ID, LOAN\_ID, E\_ID, AUTHORIZATION\_STATUS)

```
(1,233,11,'PENDING');  
(2,234,12,'PENDING');  
(3,235,13,'PROCESSED');  
(4,236,14,'REJECTED');  
(5,237,15,'PROCESSED');  
(6,238,16,'PROCESSED');
```

## POSTAL CODES (POSTCODE, CITY)

```
(4400,'ISLAMABAD');  
(4445,'RAWALPINDI');  
(46000,'SIALKOT');  
(47040,'WAH CANTT');  
(25200,'PESHAWAR');  
(60001,'MULTAN');
```

## Further Assumptions

Below we have mentioned a few assumptions we will follow for our implementation of the web application.

- The shortest repayment period of a loan is assumed to be at least 6 months and all repayments are done on a monthly basis.
- Any account that has been successfully created must have at least one card.
- The SERVE relation deals with many types of customer to employee interaction except account opening and loan authorization, both of which have dedicated separate relationships.

# **Central Bank Management Portal (CBMP)**

*Assignment 3: Basic SQL and  
Data Population*

*CS232-Database Management Systems*

*DATE: 03/06/2022*

**Group 6**

*Muhammad Faraz (2020292)*

*Muhammad Umer Asif (2020372)*

*Muhammed Hamza Malik (2020382)*

*Submitted to: Mr. Muhammad Mohsin Zafar*

## Tables/Relations

The below table summarizes the relations, their primary and foreign keys, and domain of each attribute has been described in brackets.

Please note that DML queries for inserting data were written into the query console directly. Under each DDL statement, we have given the generalized DML insert, and then underneath the actual dummy data stored into the tables.

Please also note that the table schemas and their constraints displayed in the DDL statements may have been changed during the course of the project when the web application was being developed, thus the final combined document will contain the most up-to-date table DDL definitions.

Relation Name	Primary Key (PK)	Foreign Keys (FK)	Other Attributes
<b>Customer</b>	C_ID (number)	---	CITY (string, major cities in Pakistan), POSTCODE (number), STREET (string), CF_NAME (string), CL_NAME (string), EMAIL (string), BIRTHDATE (date), PH_NO (string)
<b>Employee</b>	E_ID (number)	BRANCH_CODE (number, only the numbers that match registered branch codes)  MGR_ID (number)	EF_NAME (string), EL_NAME (string), SAL (number), PASSWORD (string), DESIGNATION (string), STRT_DATE (date), EMAIL (string)
<b>Loan</b>	LOAN_ID (number)	---	LOAN_TYPE (string, either home or student loan), LOAN_PERIOD (number), LOAN_DATE (date), INTEREST_RATE (number), AMOUNT (number), PAYMENT_MODE (string, can be cheque or cash)
<b>Home Loan</b>	H_LOAN_ID (number)	---	CITY (string, major cities in Pakistan), STREET (string), POSTCODE (number)
<b>StudentLoan</b>	S_LOAN_ID (number)	---	LOAN_PURPOSE (string)
<b>Branch</b>	B_CODE (number)	MGR_ID (number)	CITY (string, major cities), POSTCODE (number)
<b>Account</b>	ACCOUNT_NO (number)	C_ID (number)	ACCOUNT_TYPE (string), BALANCE (number), INTEREST_RATE (number), ACC_PASSWORD (string)
<b>Savings</b>	S_ACCOUNT_NO (number)	---	MAX_LIMIT (number)
<b>Current</b>	C_ACCOUNT_NO (number)	---	MIN_BALANCE (number)
<b>Card</b>	CARD_NO (number)	ACC_NO (number)	EXP_DATE (date), ISSUE_DATE (date), CARD_TYPE (string), CVV_CODE (number)
<b>Transaction</b>	TRANS_ID (number)	ACC_NO (number)	TRANS_DATE (date), TRANS_TIME (time), STATUS (string, e.g. 'processed')

<b>Repayment</b>	LOAN_ID (number), REPAYMENT_TIME (time)	---	AMOUNT (number), REPAYMENT_DATE (date)
<b>Loan Auth. Record</b>	C_ID (number), LOAN_ID (number), E_ID (number)	C_ID (number), LOAN_ID (number), E_ID (number)	AUTHORIZATION_STATUS (string)
<b>Account Auth. Record</b>	C_ID (number), ACCOUNT_NO (number), E_ID (number)	C_ID (number), ACCOUNT_NO (number), E_ID (number)	STATUS (string), OPENING_DATE (date)
<b>Customer Service Record</b>	SERVICE_ID (number), C_ID (number), E_ID (number)	C_ID (number), E_ID (number)	SERVICE_TYPE (string), RATINGS (number, on a scale of 1 to 5)
<b>Transfer Record</b>	SENDER_ACCOUNT_NO (number), TRANS_ID (number)	SENDER_ACCOUNT_NO (number), TRANS_ID (number), RECEIVER_ACCOUNT_NO (number)	---

## Data Definition Language (DDL) SQL

### Account

```
CREATE TABLE IF NOT EXISTS public.account
(
    account_no integer NOT NULL,
    balance integer NOT NULL,
    acc_type character varying(50) COLLATE pg_catalog."default" NOT NULL,
    acc_password integer,
    interest_rate numeric(2,1) DEFAULT 1.03,
    c_id integer NOT NULL,
    CONSTRAINT account_pkey PRIMARY KEY (account_no),
    CONSTRAINT acco FOREIGN KEY (c_id)
        REFERENCES public.customer (c_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT account_acc_type_check CHECK (acc_type::text = ANY (ARRAY['CURRENT'::character varying::text, 'SAVING'::character varying::text])),
    CONSTRAINT account_interest_rate_check CHECK (interest_rate = ANY (ARRAY[5.5, 6::numeric, 6.5, 7::numeric, 7.5, 8::numeric, 8.5, 9::numeric, 9.5, 10::numeric, 10.5, 11::numeric, 11.5, 12::numeric]))
)

INSERT INTO public.account(
    account_no, balance, acc_type, acc_password, interest_rate, c_id)
```

VALUES (?, ?, ?, ?, ?, ?);

	account_no [PK] integer	balance integer	acc_type character varying (50)	acc_password integer	interest_rate numeric (2,1)	c_id integer
1	3000	53500	CURRENT	5444	6.0	1000
2	3001	5454	CURRENT	3282	5.5	1001
3	3002	3455	SAVING	9334	8.5	1002
4	3003	5444	CURRENT	3554	7.0	1003
5	3004	8000	SAVING	5454	6.5	1004
6	3005	9000	CURRENT	5005	7.0	1005
7	3006	8080	CURRENT	5050	7.5	1002
8	3007	2900	SAVING	9091	7.0	1003
9	3008	1000	CURRENT	[null]	[null]	1
10	3009	79905500	SAVING	[null]	[null]	1
11	3010	3500	SAVING	[null]	[null]	2

## Account\_Openning\_Record

```
CREATE TABLE IF NOT EXISTS public.account_openning_record
(
    c_id integer NOT NULL,
    account_no integer NOT NULL,
    e_id integer NOT NULL,
    authorization_status character varying(50) COLLATE pg_catalog."default" NOT NULL,
    opening_date date NOT NULL,
    CONSTRAINT account_openning_record_pkey PRIMARY KEY (c_id, account_no, e_id),
    CONSTRAINT acc_o_r FOREIGN KEY (c_id)
        REFERENCES public.customer (c_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT acc_o_r_ FOREIGN KEY (account_no)
        REFERENCES public.account (account_no) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT acc_o_r__ FOREIGN KEY (e_id)
        REFERENCES public.employee (e_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT auth_status_check CHECK (authorization_status::text = ANY
(ARRAY['PENDING'::character varying::text, 'PROCESSED'::character varying::text,
```

```

'REJECTED'::character varying::text, 'REMOVED'::character varying::text,
'ACCEPTED'::character varying::text]))
)

INSERT INTO public.account_openning_record(
    c_id, account_no, e_id, authorization_status, opening_date)
VALUES (?, ?, ?, ?, ?);

```

Query Editor	Query History	Data Output	Explain	Messages	Notificat
	c_id [PK] integer	account_no [PK] integer	e_id [PK] integer	authorization_status character varying (50)	opening_date date
1	1	3009	100	PROCESSED	2022-05-29
2	2	3010	100	REMOVED	2022-05-29
3	1000	3000	25	PROCESSED	2002-09-06
4	1001	3001	26	PROCESSED	2002-09-06
5	1002	3002	27	PROCESSED	2002-09-06
6	1002	3006	31	PROCESSED	2002-09-06
7	1003	3003	28	PROCESSED	2002-09-06
8	1003	3007	31	PROCESSED	2002-09-06
9	1004	3004	29	PROCESSED	2002-09-06
10	1005	3005	30	PROCESSED	2002-09-06

## Branch

```

CREATE TABLE IF NOT EXISTS public.branch
(
    b_code integer NOT NULL,
    postcode integer NOT NULL,
    mgr_id integer,
    CONSTRAINT branch_pkey PRIMARY KEY (b_code),
    CONSTRAINT postcode_branch_fk FOREIGN KEY (postcode)
        REFERENCES public.postal (postcode) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

INSERT INTO public.branch(
    b_code, postcode, mgr_id)
VALUES (?, ?, ?);

```



	Query Editor	Query History	Data
	b_code [PK] integer	postcode integer	mgr_id integer
1	343	60001	6
2	434	46000	3
3	436	47040	4
4	444	44000	1
5	466	44450	2
6	490	25200	5

## Card\_

```
CREATE TABLE IF NOT EXISTS public.card_
(
    card_no bigint NOT NULL,
    exp_date date NOT NULL,
    issue_date date NOT NULL,
    card_type character varying(50) COLLATE pg_catalog."default" NOT NULL,
    cvv_code integer NOT NULL,
    account_no integer NOT NULL,
    CONSTRAINT card__pkey PRIMARY KEY (card_no),
    CONSTRAINT car FOREIGN KEY (account_no)
        REFERENCES public.account (account_no) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

INSERT INTO public.card_(
    card_no, exp_date, issue_date, card_type, cvv_code, account_no)
VALUES (?, ?, ?, ?, ?, ?);
```

	Query Editor	Query History	Data Output	Explain	Messages	Notifications
	card_no [PK] bigint	exp_date date	issue_date date	card_type character varying (50)	cvv_code integer	account_no integer
1	89324327	2010-02-04	2002-12-04	Platinum	800	3000
2	89324328	2010-02-04	2002-12-03	Silver	801	3001
3	89324329	2010-02-09	2002-12-06	Gold	802	3002
4	89324330	2010-02-10	2002-12-08	Master	803	3003
5	89324331	2010-02-01	2002-12-03	Gold	804	3004
6	89324332	2010-02-07	2002-12-01	Silver	805	3005

## Current

```
CREATE TABLE IF NOT EXISTS public.current
(
    min_balance integer NOT NULL,
    account_no integer NOT NULL,
    CONSTRAINT current_pkey PRIMARY KEY (account_no),
    CONSTRAINT curr FOREIGN KEY (account_no)
        REFERENCES public.account (account_no) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

INSERT INTO public.current(
    min_balance, account_no)
VALUES (?, ?);
```

	min_balance integer	account_no [PK] integer
1	100	3000
2	100	3001
3	100	3003
4	100	3005
5	100	3006

# Customer

```
CREATE TABLE IF NOT EXISTS public.customer
(
    postcode integer,
    street character varying(50) COLLATE pg_catalog."default" NOT NULL,
    c_id integer NOT NULL,
    f_name character varying(50) COLLATE pg_catalog."default" NOT NULL,
    l_name character varying(50) COLLATE pg_catalog."default" NOT NULL,
    email character varying(50) COLLATE pg_catalog."default" NOT NULL,
    b_date date,
    ph_no bigint,
    CONSTRAINT customer_pkey PRIMARY KEY (c_id)
)

INSERT INTO public.customer(
    postcode, street, c_id, f_name, l_name, email, b_date, ph_no)
VALUES (?, ?, ?, ?, ?, ?, ?, ?);
```

	postcode integer	street character varying (50)	c_id [PK] integer	f_name character varying (50)	l_name character varying (50)	email character varying (50)	b_date date	ph_no bigint
1	[null]		1	It	Worked		[null]	[null]
2	[null]	2	2	test2	test2	test2@whatever.com	[null]	222222222
3	[null]	4	3	4	4	4@gmail.com	[null]	[null]
4	44000	Constitution Avenue	1000	Muhammad	Farae	sample@xyz.com	2002-01-01	345876549
5	44450	Umer's Crib	1001	Umer	Asif	sample1@xyz.com	2002-01-02	345876550
6	46000	Hamza's Minions Headquarters	1002	Hamza	Chan	sample2@xyz.com	2002-01-03	345876551
7	47040	Quaid Avenue	1003	NASIR	SOHAIL	sample3@xyz.com	2002-01-04	345876552
8	25200	Brother Musa Street	1004	IMRAN	HASHMI	sample4@xyz.com	2002-01-05	345876553
9	60001	Zaeem Parizad Road	1005	JAVED	AWAN	sample5@xyz.com	2002-01-05	345876554

# Customer\_service\_record

```
CREATE TABLE IF NOT EXISTS public.customer_service_record
(
    service_id integer NOT NULL,
    customer_service_rating integer NOT NULL,
    c_id integer NOT NULL,
    e_id integer NOT NULL,
    "SERVICE_TIME" time without time zone NOT NULL,
    CONSTRAINT service_record_pk PRIMARY KEY (service_id, c_id, e_id, "SERVICE_TIME"),
    CONSTRAINT service_id_unique UNIQUE (service_id),
    CONSTRAINT c_s_r_ FOREIGN KEY (c_id)
        REFERENCES public.customer (c_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
```

```

CONSTRAINT c_s_r__ FOREIGN KEY (e_id)
REFERENCES public.employee (e_id) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE NO ACTION,
CONSTRAINT customer_service_record_customer_service_rating_check CHECK
(customer_service_rating = ANY (ARRAY[5, 4, 3, 2, 1, 0]))
)

INSERT INTO public.customer_service_record(
service_id, customer_service_rating, c_id, e_id, "SERVICE_TIME")
VALUES (?, ?, ?, ?, ?);

```

	service_id [PK] integer	customer_service_rating integer	c_id [PK] integer	e_id [PK] integer	SERVICE_TIME [PK] time without time zone
1	1	4	1000	22	08:05:50.907637
2	2	5	1001	30	08:05:50.907637
3	3	4	1002	31	08:05:50.907637
4	4	2	1003	29	08:05:50.907637

## Customer\_service\_types

```

CREATE TABLE IF NOT EXISTS public.customer_service_types
(
service_id integer NOT NULL,
service_type character varying(50) COLLATE pg_catalog."default",
CONSTRAINT service_id_pk PRIMARY KEY (service_id),
CONSTRAINT serv_id_fk FOREIGN KEY (service_id)
REFERENCES public.customer_service_record (service_id) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE NO ACTION,
CONSTRAINT serv_type_check CHECK (service_type::text = ANY (ARRAY['ACCOUNT
MANAGER'::character varying::text, 'LOAN MANAGEMENT'::character varying::text, 'GENERAL
INFORMATION'::character varying::text, 'CARD RELATED'::character varying::text, 'ACCOUNT
MANAGEMENT'::character varying::text]))
)

INSERT INTO public.customer_service_types(
service_id, service_type)
VALUES (?, ?);

```

	service_id [PK] integer	service_type character varying (50)
1	1	ACCOUNT MANAGEMENT
2	2	LOAN MANAGEMENT
3	3	GENERAL INFORMATION
4	4	CARD RELATED

## Employee

```
CREATE TABLE IF NOT EXISTS public.employee
(
    e_id integer NOT NULL,
    f_name character varying(50) COLLATE pg_catalog."default" NOT NULL,
    l_name character varying(50) COLLATE pg_catalog."default" NOT NULL,
    sal integer NOT NULL,
    password_ integer NOT NULL,
    designation character varying(50) COLLATE pg_catalog."default" NOT NULL,
    strt_date date NOT NULL,
    email character varying(50) COLLATE pg_catalog."default" NOT NULL,
    bcode integer NOT NULL,
    manages_eid integer,
    CONSTRAINT employee_pkey PRIMARY KEY (e_id),
    CONSTRAINT empl FOREIGN KEY (bcode)
        REFERENCES public.branch (b_code) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT empl_ FOREIGN KEY (manages_eid)
        REFERENCES public.employee (e_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT employee_desig_check CHECK (designation::text = ANY
(ARRAY['EXECUTIVE'::character varying::text, 'TELLER'::character varying::text,
'CASHIER'::character varying::text])) NOT VALID
)

INSERT INTO public.employee(
    e_id, f_name, l_name, sal, password_, designation, strt_date, email, bcode, manages_eid)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?);
```

e_id [PK] integer	f_name character varying (50)	Lname character varying (50)	sal integer	password_ integer	designation character varying (50)	strt_date date	email character varying (50)	bcode integer	manages_eid integer
1	ZAIN	INAM	79000	4342	EXECUTIVE	2002-07-09	ZAININAM@gmail.com	444	100
2	HAMZA	HASHMI	50000	3433	EXECUTIVE	2002-07-08	HAMZAHASHMI@gmail.com	444	100
3	AMIR	SULTAN	48000	4564	EXECUTIVE	2002-07-06	AMIRSULTAN@gmail.com	444	100
4	AZHAR	AWAN	43000	7567	EXECUTIVE	2002-04-04	AZHARAWAN@gmail.com	444	100
5	ASIM	MALIK	80000	4564	EXECUTIVE	2002-04-05	ASIMMALIK@gmail.com	444	100
6	SAIFULLAH	AWAN	82000	6734	EXECUTIVE	2002-07-09	SAIFULLAHAWAN@gmail.com	466	100
7	MUSA	INAM	79000	5784	TELLER	2002-07-02	MUSAINAM@gmail.com	466	1
8	QASIM	HASHMI	50000	8946	TELLER	2002-07-01	QASIMHASHMI@gmail.com	466	1
9	SAIM	SULTAN	48000	4532	CASHIER	2002-07-17	SAIMSULTAN@gmail.com	466	1
10	SHERYAR	AWAN	43000	6543	CASHIER	2002-05-04	SHERYARAWAN@gmail.com	466	1
11	ABDULLAH	MALIK	80000	7489	TELLER	2002-01-03	ABDULLAHMALIK@gmail.com	466	2
12	ASHIR	AWAN	82000	9000	TELLER	2002-04-02	ASHIRAWAN@gmail.com	434	2
13	ZAIN	BHATTI	79000	2345	CASHIER	2002-07-04	ZAININAM@gmail.com	434	2
14	IBRAHIM	HASHMI	50000	8596	CASHIER	2002-07-01	IBRAHIMHASHMI@gmail.com	434	2
15	NOSHAIR	SULTAN	48000	9345	CASHIER	2002-07-01	NOSHAIRSULTAN@gmail.com	434	2
16	HAIDER	AWAN	43000	283	TELLER	2002-09-08	HAIDERAWAN@gmail.com	434	3
17	AKBAR	MALIK	80000	384	TELLER	2002-04-09	AKBARMALIK@gmail.com	434	3
18	JAWAD	AWAN	82000	6666	CASHIER	2002-04-02	JAWADAWAN@gmail.com	436	3

Query Editor Query History **Data Output** Explain Messages Notifications

e_id [PK] integer	f_name character varying (50)	Lname character varying (50)	sal integer	password_ integer	designation character varying (50)	strt_date date	email character varying (50)	bcode integer	manages_eid integer
16	HAIDER	AWAN	43000	283	TELLER	2002-09-08	HAIDERAWAN@gmail.com	434	3
17	AKBAR	MALIK	80000	384	TELLER	2002-04-09	AKBARMALIK@gmail.com	434	3
18	JAWAD	AWAN	82000	6666	CASHIER	2002-04-02	JAWADAWAN@gmail.com	436	3
19	TAHA	INAM	79000	493	CASHIER	2002-07-07	TAHAINAM@gmail.com	436	3
20	ALI	HASHMI	50000	7384	TELLER	2002-07-08	ALIHASHMI@gmail.com	436	4
21	ADNAN	SULTAN	48000	3484	TELLER	2002-07-18	ADNANSULTAN@gmail.com	436	4
22	SAMI	AWAN	43000	133	CASHIER	2002-07-09	SAMIAWAN@gmail.com	436	4
23	NOUMAN	MALIK	80000	920	CASHIER	2002-07-07	NOUMANMALIK@gmail.com	436	4
24	AMIR	AWAN	82000	7823	TELLER	2002-07-20	AMIRAWAN@gmail.com	490	5
25	WAQAS	INAM	79000	8934	TELLER	2002-07-20	WAQASINAM@gmail.com	490	5
26	BILAL	HASHMI	50000	8924	CASHIER	2002-07-02	BILALHASHMI@gmail.com	490	5
27	USAMA	SULTAN	48000	9083	CASHIER	2002-07-20	USAMASULTAN@gmail.com	490	5
28	RAUF	AWAN	43000	9285	TELLER	2002-07-02	RAUFAWAN@gmail.com	490	6
29	EJAZ	MALIK	80000	9835	TELLER	2002-07-20	EJAZMALIK@gmail.com	490	6
30	FAZAL	AWAN	82000	3263	CASHIER	2002-07-02	FAZALAWAN@gmail.com	444	6
31	JAMSHED	INAM	79000	9293	CASHIER	2002-07-02	JAMSHEDINAM@gmail.com	444	6
100	IZHAR	AWAN	82000	4322	EXECUTIVE	2002-07-09	IZHARAWAN@gmail.com	444	[null]

## Home\_loan

```
CREATE TABLE IF NOT EXISTS public.home_loan
(
    street character varying(50) COLLATE pg_catalog."default" NOT NULL,
    postcode integer NOT NULL,
    loan_id integer NOT NULL,
    CONSTRAINT home_loan_pkey PRIMARY KEY (loan_id),
    CONSTRAINT hom FOREIGN KEY (loan_id)
        REFERENCES public.loan (loan_id) MATCH SIMPLE
        ON UPDATE NO ACTION
```

```

        ON DELETE NO ACTION,
    CONSTRAINT postcode_home_fk FOREIGN KEY (postcode)
        REFERENCES public.postal (postcode) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

INSERT INTO public.home_loan(
    street, postcode, loan_id)
VALUES (?, ?, ?);

```

	street character varying (50)	postcode integer	loan_id [PK] integer
1	1	44000	233
2	2	44000	235
3	3	44450	237

## Loan

```

CREATE TABLE IF NOT EXISTS public.loan
(
    loan_id integer NOT NULL,
    amount integer NOT NULL,
    payment_mode character varying(50) COLLATE pg_catalog."default" NOT NULL,
    interest_rate integer NOT NULL,
    loan_period integer NOT NULL,
    loan_date date NOT NULL,
    loan_type character varying(50) COLLATE pg_catalog."default" NOT NULL,
    authorized boolean NOT NULL,
    CONSTRAINT loan_pkey PRIMARY KEY (loan_id),
    CONSTRAINT loan_payment_mode_check CHECK (payment_mode::text = ANY
(Array['ONLINE'::character varying::text, 'CASH'::character varying::text]))
)

INSERT INTO public.loan(
    loan_id, amount, payment_mode, interest_rate, loan_period, loan_date, loan_type,
    authorized)
VALUES (?, ?, ?, ?, ?, ?, ?, ?);

```

	loan_id [PK] integer	amount integer	payment_mode character varying (50)	interest_rate integer	loan_period integer	loan_date date	loan_type character varying (50)	authorized boolean
1	233	10000	ONLINE	5	12	2002-11-04	HOME	true
2	234	23000	ONLINE	6	24	2002-12-09	STUDENT	true
3	235	20000	CASH	8	32	2002-12-09	HOME	true
4	236	30000	CASH	10	12	2002-12-03	STUDENT	false
5	237	35000	ONLINE	12	24	2002-12-04	HOME	true
6	238	90000	CASH	15	18	2002-12-09	STUDENT	true

## Loan\_authorization Record

```
CREATE TABLE IF NOT EXISTS public.loan_authorization_record
(
    c_id integer NOT NULL,
    loan_id integer NOT NULL,
    e_id integer NOT NULL,
    authorization_status character varying(50) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT loan_authorization_record_pkey PRIMARY KEY (c_id, loan_id, e_id),
    CONSTRAINT loan_a FOREIGN KEY (c_id)
        REFERENCES public.customer (c_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT loan_a_ FOREIGN KEY (loan_id)
        REFERENCES public.loan (loan_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT loan_a__ FOREIGN KEY (e_id)
        REFERENCES public.employee (e_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT loan_authorization_record_authorization_status_check CHECK
(authorization_status::text = ANY (ARRAY['PENDING'::character varying::text,
'PROCESSED'::character varying::text, 'REJECTED'::character varying::text]))
)

INSERT INTO public.loan_authorization_record(
    c_id, loan_id, e_id, authorization_status)
VALUES (?, ?, ?, ?);
```



	<b>c_id</b> [PK] integer	<b>loan_id</b> [PK] integer	<b>e_id</b> [PK] integer	<b>authorization_status</b> character varying (50)
1	1000	233	24	PENDING
2	1001	234	25	PENDING
3	1002	235	26	PROCESSED
4	1003	236	27	REJECTED
5	1004	237	28	PROCESSED
6	1005	238	29	PROCESSED

## Postal

```
CREATE TABLE IF NOT EXISTS public.postal
(
    postcode integer NOT NULL,
    city character varying(50) COLLATE pg_catalog."default",
    CONSTRAINT postcode_pk PRIMARY KEY (postcode)
)

INSERT INTO public.postal(
    postcode, city)
VALUES (?, ?);
```

	<b>postcode</b> [PK] integer	<b>city</b> character varying (50)
1	25200	PESHAWAR
2	44000	ISLAMABAD
3	44450	RAWALPINDI
4	46000	SIALKOT
5	47040	WAH CANTT
6	60001	MULTAN

# Repayment

```
CREATE TABLE IF NOT EXISTS public.repayment
(
    repayment_date date NOT NULL,
    amount integer NOT NULL,
    loan_id integer NOT NULL,
    repayment_time time without time zone NOT NULL,
    CONSTRAINT repayment_pk PRIMARY KEY (loan_id, repayment_time),
    CONSTRAINT repay FOREIGN KEY (loan_id)
        REFERENCES public.loan (loan_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)



INSERT INTO public.repayment(
    repayment_date, amount, loan_id, repayment_time)
VALUES (?, ?, ?, ?);
```

Please note that this table was not populated as it has an identifying relationship with the loan table and we decided to populate dummy data into this table directly from the frontend side of the application. Thus, only this table does not have data at this point.

# Savings

```
CREATE TABLE IF NOT EXISTS public.savings
(
    max_limit bigint NOT NULL,
    account_no integer NOT NULL,
    CONSTRAINT savings_pkey PRIMARY KEY (account_no),
    CONSTRAINT sav FOREIGN KEY (account_no)
        REFERENCES public.account (account_no) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

INSERT INTO public.savings(
    max_limit, account_no)
VALUES (?, ?);
```

	 max_limit bigint	 account_no [PK] integer
1	10000000	3002
2	10000000	3004
3	10000000	3007

## Student\_loan

```
CREATE TABLE IF NOT EXISTS public.student_loan
(
    loan_purpose character varying(50) COLLATE pg_catalog."default" NOT NULL,
    loan_id integer NOT NULL,
    CONSTRAINT student_loan_pkey PRIMARY KEY (loan_id),
    CONSTRAINT stu FOREIGN KEY (loan_id)
        REFERENCES public.loan (loan_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

INSERT INTO public.student_loan(
    loan_purpose, loan_id)
VALUES (?, ?);
```

	loan_purpose character varying (50)	loan_id [PK] integer
1	PAYING FOR INTERNSHIP APPLICATIONS	234
2	FOR PAYMENT OF TUITION FEE	236
3	FOR DORMITORY/ACCOMODATION FEES	238

## Transactions

```
CREATE TABLE IF NOT EXISTS public.transactions
(
    trans_id integer NOT NULL,
    trans_date date NOT NULL,
    status_ character varying(50) COLLATE pg_catalog."default" NOT NULL,
    account_no integer,
    trans_time time without time zone,
    "Debit" double precision,
    "Credit" double precision,
    CONSTRAINT transactions_pkey PRIMARY KEY (trans_id),
    CONSTRAINT trans FOREIGN KEY (account_no)
        REFERENCES public.account (account_no) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
```

```

CONSTRAINT transactions_status__check CHECK (status_::text = ANY
(ARRAY['ACCEPTED'::character varying::text, 'PROCESSED'::character varying::text,
'PENDING'::character varying::text]))
)

INSERT INTO public.transactions(
  trans_id, trans_date, status_, account_no, trans_time, "Debit", "Credit")
VALUES (?, ?, ?, ?, ?, ?, ?);




```

	trans_id [PK] integer	trans_date date	status_ character varying (50)	account_no integer	trans_time time without time zone
1	100	2003-12-04	PROCESSED	3000	19:43:00
2	101	2003-12-03	PROCESSED	3001	16:47:00
3	102	2003-12-08	PENDING	3002	18:20:00
4	103	2003-12-07	PROCESSED	3003	13:27:00
5	104	2003-12-05	PROCESSED	3004	19:09:00
6	105	2003-12-04	PROCESSED	3005	15:08:00
7	106	2003-12-04	PROCESSED	3009	[null]
8	107	2022-05-31	PROCESSED	3009	[null]
9	108	2022-05-31	PROCESSED	3009	14:07:51.908111
10	109	2022-05-31	PROCESSED	3009	14:17:21.833502
11	110	2022-05-31	PROCESSED	3009	14:20:35.73941
12	111	2022-05-31	PROCESSED	3009	21:04:15.84969
13	112	2022-05-31	PROCESSED	3009	21:04:33.906092

## Transfer Record

```
CREATE TABLE IF NOT EXISTS public.transfer_record
(
    account_no_receiving_funds integer NOT NULL,
    trans_id integer NOT NULL,
    account_no integer NOT NULL,
    CONSTRAINT transfer_record_pkey PRIMARY KEY (trans_id, account_no),
    CONSTRAINT acc_rec_fk FOREIGN KEY (account_no_receiving_funds)
        REFERENCES public.account (account_no) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT trf_rec_ FOREIGN KEY (trans_id)
        REFERENCES public.transactions (trans_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT trf_rec__ FOREIGN KEY (account_no)
        REFERENCES public.account (account_no) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

INSERT INTO public.transfer_record(
    account_no_receiving_funds, trans_id, account_no)
VALUES (?, ?, ?);
```

	 <b>account_no_receiving_funds</b> integer	 <b>trans_id</b> [PK] integer	 <b>account_no</b> [PK] integer
1	3001	100	3002
2	3002	101	3003
3	3004	102	3000
4	3010	115	3000
5	3010	117	3000

## Running Sample Queries on the Database

QUERY	CODE	OUTPUT
1	<pre>select c_id,balance,acc_type,acc_password,s.max_limit from account a inner join savings s on a.account_no=s.account_no;</pre>	<b>Please see screenshot (SC) number 1.</b>
2	<pre>select c_id,balance,acc_type,acc_password,s.min_balance from account a inner join current s on a.account_no=s.account_no;</pre>	<b>Please see screenshot (SC) number 2.</b>
3	<pre>select * from employee where employee.e_id in(select distinct(manages_eid) from employee);</pre>	<b>Please see screenshot (SC) number 3.</b>
4	<pre>select * from employee where employee.e_id in (select e_id from account_openning_record group by account_openning_record.e_id having e_id&gt;1);</pre>	<b>Please see screenshot (SC) number 4.</b>
5	<pre>select t_.trans_id,t_.trans_date,t_.account_no AS SENDER_ACCOUNT,tr.account_no_receiving_funds AS RECIEVER_ACCOUNT from transactions t_ inner join transfer_record tr on t_.account_no=tr.account_no;</pre>	<b>Please see screenshot (SC) number 5.</b>

# Screenshots of Sample Queries

1)

	c_id integer	balance integer	acc_type character varying (50)	acc_password integer	max_limit bigint
1	1002	3455	SAVING	9334	10000000
2	1004	8000	SAVING	5454	10000000
3	1003	2900	SAVING	9091	10000000

2)

	c_id integer	balance integer	acc_type character varying (50)	acc_password integer	min_balance integer
1	1000	4334	CURRENT	5444	100
2	1001	5454	CURRENT	3282	100
3	1003	5444	CURRENT	3554	100
4	1005	9000	CURRENT	5005	100
5	1002	8080	CURRENT	5050	100

3)

e_id [PK] integer	f_name character varying (50)	l_name character varying (50)	sal integer	password_ integer	designation character varying (50)	strt_date date	email character varying (50)	bcode integer	manages_eid integer
100	IZHAR	AWAN	82000	4322	EXECUTIVE	2002-07-09	IZHARAWAN@gmail.com	444	[null]
1	ZAIN	INAM	79000	4342	EXECUTIVE	2002-07-09	ZAININAM@gmail.com	444	100
2	HAMZA	HASHMI	50000	3433	EXECUTIVE	2002-07-08	HAMZAHASHMI@gmail.com	444	100
3	AMIR	SULTAN	48000	4564	EXECUTIVE	2002-07-06	AMIRSULTAN@gmail.com	444	100
4	AZHAR	AWAN	43000	7567	EXECUTIVE	2002-04-04	AZHARAWAN@gmail.com	444	100
5	ASIM	MALIK	80000	4564	EXECUTIVE	2002-04-05	ASIMMALIK@gmail.com	444	100
6	SAIFULLAH	AWAN	82000	6734	EXECUTIVE	2002-07-09	SAIFULLAHAWAN@gmail.com	466	100

4)

	e_id [PK] integer	f_name character varying (50)	L_name character varying (50)	sal integer	password integer	designation character varying (50)	strt_date date	email character varying (50)	bcode integer	manages_eid integer
1	100	IZHAR	AWAN	82000	4322	EXECUTIVE	2002-07-09	IZHARAWAN@gmail.com	444	[null]
2	26	BILAL	HASHMI	50000	8924	CASHIER	2002-07-02	BILALHASHMI@gmail.com	490	5
3	27	USAMA	SULTAN	48000	9083	CASHIER	2002-07-20	USAMASULTAN@gmail.com	490	5
4	30	FAZAL	AWAN	82000	3263	CASHIER	2002-07-02	FAZALAWAN@gmail.com	444	6
5	31	JAMSHED	INAM	79000	9293	CASHIER	2002-07-02	JAMSHEDINAM@gmail.com	444	6
6	25	WAQAS	INAM	79000	8934	TELLER	2002-07-20	WAQASINAM@gmail.com	490	5
7	28	RAUF	AWAN	43000	9285	TELLER	2002-07-02	RAUFAWAN@gmail.com	490	6
8	29	EJAZ	MALIK	80000	9835	TELLER	2002-07-20	EJAZMALIK@gmail.com	490	6

5)

	trans_id integer	trans_date date	sender_account integer	reciever_account integer
1	100	2003-12-04	3000	3004
2	102	2003-12-08	3002	3001
3	103	2003-12-07	3003	3002



# **Central Bank Management Portal (CBMP)**

*Assignment 4: Advanced SQL*

*CS232-Database Management Systems*

*DATE: 03/06/2022*

*Group 6*

*Muhammad Faraz (2020292)*

*Muhammad Umer Asif (2020372)*

*Muhammed Hamza Malik (2020382)*

*Submitted to: Mr. Muhammad Mohsin Zafar*

## Procedures and Functions

The below procedure calculates increases employee salary by 3% for every year they've worked in the company. Two screenshots are attached to show the procedure successfully updates the salary attribute of employees.

```
CREATE OR REPLACE FUNCTION INCREASE_SAL(SAL_RISE double precision,STRT_DATE date)
    returns double precision LANGUAGE PLPGSQL AS $$
DECLARE
    salary double precision;
    NEW_SAL NUMERIC(10,2) := SAL_RISE;
BEGIN
    if current_date - STRT_DATE < 365 then
        NEW_SAL:=SAL_RISE;
    ELSE
        NEW_SAL:=((SAL_RISE/100)*3)+SAL_RISE;
    END if;
    salary:=NEW_SAL;
    RETURN salary;
END;$$;

CREATE OR REPLACE PROCEDURE public.raise_sal(
)
LANGUAGE 'plpgsql'
AS $BODY$
    DECLARE
        S CURSOR FOR SELECT E_ID,SAL,STRT_DATE FROM EMPLOYEE FOR UPDATE;
        SAL double precision;
        S_DATE DATE;
        STORE RECORD;
        NEW_SAL double precision;
BEGIN
    OPEN S;
    LOOP
        FETCH S INTO STORE;
        EXIT WHEN NOT FOUND;
        SAL:=STORE.SAL;
        S_DATE:=STORE.STRT_DATE;
        NEW_SAL:=INCREASE_SAL(SAL,S_DATE);
        IF SAL=NEW_SAL THEN
            RAISE INFO' EMPLOYEE WITH ID NO % HAS SPENT LESS THAN 1 YEAR IN HIS/HER
DEPARTMENT',SE.E_ID;
        ELSE
            UPDATE EMPLOYEE
            SET SAL =NEW_SAL
            WHERE CURRENT OF S;
        END IF;
    END LOOP;
END $BODY$;
```

```

CLOSE S;
COMMIT;
END;
$BODY$;
ALTER PROCEDURE public.raise_sal()
OWNER TO postgres;

```

## BEFORE

	e_id [PK] integer	f_name character varying (50)	L_name character varying (50)	sal integer	password_ integer	designation character varying (50)	strt_date date	email character varying (50)	bcode integer
1	1	ZAIN	INAM	83811	4342	EXECUTIVE	2002-07-09	ZAININAM@gmail.com	444
2	2	HAMZA	HASHMI	53045	3433	EXECUTIVE	2002-07-08	HAMZAHASHMI@gmail.com	444
3	3	AMIR	SULTAN	50923	4564	EXECUTIVE	2002-07-06	AMIRSULTAN@gmail.com	444
4	4	AZHAR	AWAN	45619	7567	EXECUTIVE	2002-04-04	AZHARAWAN@gmail.com	444
5	5	ASIM	MALIK	84872	4564	EXECUTIVE	2002-04-05	ASIMMALIK@gmail.com	444
6	6	SAIFULLAH	AWAN	86994	6734	EXECUTIVE	2002-07-09	SAIFULLAHAWAN@gmail.com	466
7	7	MUSA	INAM	83811	5784	TELLER	2002-07-02	MUSAINAM@gmail.com	466
8	8	QASIM	HASHMI	53045	8946	TELLER	2002-07-01	QASIMHASHMI@gmail.com	466
9	9	SAIM	SULTAN	50923	4532	CASHIER	2002-07-17	SAIMSULTAN@gmail.com	466
10	10	SHERYAR	AWAN	45619	6543	CASHIER	2002-05-04	SHERYARAWAN@gmail.com	466
11	11	ABDULLAH	MALIK	84872	7489	TELLER	2002-01-03	ABDULLAHMALIK@gmail.com	466
12	12	ASHIR	AWAN	86994	9000	TELLER	2002-04-02	ASHIRAWAN@gmail.com	434
13	13	ZAIN	BHATTI	83811	2345	CASHIER	2002-07-04	ZAININAM@gmail.com	434
14	14	IBRAHIM	HASHMI	53045	8596	CASHIER	2002-07-01	IBRAHIMHASHMI@gmail.com	434
15	15	NOSHAIR	SULTAN	50923	9345	CASHIER	2002-07-01	NOSHAIRSULTAN@gmail.com	434
16	16	HAIDER	AWAN	45619	283	TELLER	2002-09-08	HAIDERAWAN@gmail.com	434
17	17	AKBAR	MALIK	84872	384	TELLER	2002-04-09	AKBARMALIK@gmail.com	434
18	18	JAWAD	AWAN	86994	6666	CASHIER	2002-04-02	JAWADAWAN@gmail.com	436
19	19	TAHA	INAM	83811	493	CASHIER	2002-07-07	TAHAINAM@gmail.com	436
20	20	ALI	HASHMI	53045	7384	TELLER	2002-07-08	ALIHASHMI@gmail.com	436
21	21	ADNAN	SULTAN	50923	3484	TELLER	2002-07-18	ADNANSULTAN@gmail.com	436
22	22	SAMI	AWAN	45619	133	CASHIER	2002-07-09	SAMIAWAN@gmail.com	436
23	23	NOUMAN	MALIK	84872	920	CASHIER	2002-07-07	NOUMANMALIK@gmail.com	436
24	24	AMIR	AWAN	86994	7823	TELLER	2002-07-20	AMIRAWAN@gmail.com	490
25	25	WAQAS	INAM	83811	8934	TELLER	2002-07-20	WAQASINAM@gmail.com	490
26	26	BILAL	HASHMI	53045	8924	CASHIER	2002-07-02	BILALHASHMI@gmail.com	490
27	27	USAMA	SULTAN	50923	9083	CASHIER	2002-07-20	USAMASULTAN@gmail.com	490

## AFTER

	e_id [PK] integer	f_name character varying (50)	L_name character varying (50)	sal integer	password_ integer	designation character varying (50)	strt_date date	email character varying (50)	bcode integer
1	1	ZAIN	INAM	86325	4342	EXECUTIVE	2002-07-09	ZAININAM@gmail.com	444
2	2	HAMZA	HASHMI	54636	3433	EXECUTIVE	2002-07-08	HAMZAHASHMI@gmail.com	444
3	3	AMIR	SULTAN	52451	4564	EXECUTIVE	2002-07-06	AMIRSULTAN@gmail.com	444
4	4	AZHAR	AWAN	46988	7567	EXECUTIVE	2002-04-04	AZHARAWAN@gmail.com	444
5	5	ASIM	MALIK	87418	4564	EXECUTIVE	2002-04-05	ASIMMALIK@gmail.com	444
6	6	SAIFULLAH	AWAN	89604	6734	EXECUTIVE	2002-07-09	SAIFULLAHAWAN@gmail.com	466
7	7	MUSA	INAM	86325	5784	TELLER	2002-07-02	MUSAINAM@gmail.com	466
8	8	QASIM	HASHMI	54636	8946	TELLER	2002-07-01	QASIMHASHMI@gmail.com	466
9	9	SAIM	SULTAN	52451	4532	CASHIER	2002-07-17	SAIMSULTAN@gmail.com	466
10	10	SHERYAR	AWAN	46988	6543	CASHIER	2002-05-04	SHERYARAWAN@gmail.com	466
11	11	ABDULLAH	MALIK	87418	7489	TELLER	2002-01-03	ABDULLAHMALIK@gmail.com	466
12	12	ASHIR	AWAN	89604	9000	TELLER	2002-04-02	ASHIRAWAN@gmail.com	434
13	13	ZAIN	BHATTI	86325	2345	CASHIER	2002-07-04	ZAININAM@gmail.com	434
14	14	IBRAHIM	HASHMI	54636	8596	CASHIER	2002-07-01	IBRAHIMHASHMI@gmail.com	434
15	15	NOSHAIR	SULTAN	52451	9345	CASHIER	2002-07-01	NOSHAIRSULTAN@gmail.com	434
16	16	HAIDER	AWAN	46988	283	TELLER	2002-09-08	HAIDERAWAN@gmail.com	434
17	17	AKBAR	MALIK	87418	384	TELLER	2002-04-09	AKBARMALIK@gmail.com	434
18	18	JAWAD	AWAN	89604	6666	CASHIER	2002-04-02	JAWADAWAN@gmail.com	436
19	19	TAHA	INAM	86325	493	CASHIER	2002-07-07	TAHAINAM@gmail.com	436
20	20	ALI	HASHMI	54636	7384	TELLER	2002-07-08	ALIHASHMI@gmail.com	436
21	21	ADNAN	SULTAN	52451	3484	TELLER	2002-07-18	ADNANSULTAN@gmail.com	436
22	22	SAMI	AWAN	46988	133	CASHIER	2002-07-09	SAMIAWAN@gmail.com	436
23	23	NOUMAN	MALIK	87418	920	CASHIER	2002-07-07	NOUMANMALIK@gmail.com	436
24	24	AMIR	AWAN	89604	7823	TELLER	2002-07-20	AMIRAWAN@gmail.com	490
25	25	WAQAS	INAM	86325	8934	TELLER	2002-07-20	WAQASINAM@gmail.com	490
26	26	BILAL	HASHMI	54636	8924	CASHIER	2002-07-02	BILALHASHMI@gmail.com	490
27	27	USAMA	SULTAN	52451	9083	CASHIER	2002-07-1	USAMASULTAN@gmail.com	490
28	28	SAIF	AWAN	46988	6734	EXECUTIVE	2002-07-09	SAIFAWAN@gmail.com	466

✓ Query returned successfully in 30 msec.

The procedure below displays the information of customers having more than one account.

```
CREATE OR REPLACE PROCEDURE public.no_of_accounts_of_customers(
)
LANGUAGE 'plpgsql'
AS $BODY$
    DECLARE
        ACC CURSOR FOR SELECT * FROM CUSTOMER WHERE C_ID IN (SELECT C_ID FROM ACCOUNT GROUP BY
C_ID HAVING COUNT(ACCOUNT_NO)>1);
        REC RECORD;
    BEGIN
        OPEN ACC;
        LOOP
            FETCH ACC INTO REC;
            EXIT WHEN NOT FOUND;
            RAISE INFO 'NAME:% % ID:% PHONE
NO:% BIRTHDAY:% CITY:% EMAIL:%',REC.F_NAME,REC.L_NAME,REC.C_ID,REC.PH_NO,REC.B_DATE,REC.PO
STCODE,REC.EMAIL;
        END LOOP;
        CLOSE ACC;
```

```

END;
$BODY$;
ALTER PROCEDURE public.no_of_accounts_of_customers()
OWNER TO postgres;

```

```

INFO:  NAME:Hamza Chan  ID:1002  PHONE NO:345876551  BIRTHDAY:2002-01-03  CITY:46000  EMAIL:sample2@xyz.com
INFO:  NAME:NASIR SOHAIL  ID:1003  PHONE NO:345876552  BIRTHDAY:2002-01-04  CITY:47040  EMAIL:sample3@xyz.com


```

Function to calculate the total number of accounts in saving and current combined using cursors.

```

CREATE OR REPLACE FUNCTION TOTAL_ACCOUNT()
RETURNS INT language plpgsql as $$
declare
AC CURSOR FOR SELECT A_.ACCOUNT_NO
FROM ACCOUNT A_
INNER JOIN SAVINGS S
ON S.ACCOUNT_NO=A_.ACCOUNT_NO;
SA CURSOR FOR SELECT A_.ACCOUNT_NO
FROM ACCOUNT A_
INNER JOIN CURRENT C
ON C.ACCOUNT_NO=A_.ACCOUNT_NO;
TOTAL_ACC INT;
TOTAL INT ;
SAV_ACC INT := 0;
CURR_ACC INT := 0;
REC RECORD;
REC1 RECORD;
BEGIN
OPEN AC;
OPEN SA;
LOOP
FETCH AC INTO REC;
EXIT WHEN NOT FOUND;
SAV_ACC:=SAV_ACC+1;
END LOOP;
LOOP
FETCH SA INTO REC1;
EXIT WHEN NOT FOUND;
CURR_ACC:=CURR_ACC+1;
END LOOP;
TOTAL :=SAV_ACC+CURR_ACC;
RAISE INFO 'TOTAL CURRENT ACCOUNTS :% ',CURR_ACC;
RAISE INFO 'TOTAL SAVING ACCOUNTS :% ',SAV_ACC;
TOTAL_ACC := TOTAL;
CLOSE AC;
CLOSE SA;
return TOTAL_ACC;
END;$$;

```

	total_account	
	integer	
1		8

Procedure to return the employee details who have customer ratings greater than 2.

```
CREATE OR REPLACE PROCEDURE COUNT_RATIINGS() LANGUAGE PLPGSQL AS $$
    DECLARE
        C CURSOR FOR SELECT * FROM EMPLOYEE E INNER JOIN CUSTOMER_SERVICE_RECORD R ON
E.E_ID=R.E_ID WHERE R.CUSTOMER_SERVICE_RATING >2;
        REC RECORD;
    BEGIN
        OPEN C;
        LOOP
            FETCH C INTO REC;
            EXIT WHEN NOT FOUND;
            RAISE INFO 'NAME:% % ID:% B-
CODE:% DESIGNATION:% SALARY:% EMAIL:%',REC.F_NAME,REC.L_NAME,REC.E_ID,REC.BCODE,REC.DESIGN
ATION,REC.SAL,REC.EMAIL;
            END LOOP;
        CLOSE C;
    END;$$;

call count_ratiings();
```

```
INFO: NAME:SAMI AWAN ID:22 B-CODE:436 DESIGNATION:CASHIER SALARY:46988 EMAIL:SAMIWAN@gmail.com
INFO: NAME:FAZAL AWAN ID:30 B-CODE:444 DESIGNATION:CASHIER SALARY:89604 EMAIL:FAZALAWAN@gmail.com
INFO: NAME:JAMSHED INAM ID:31 B-CODE:444 DESIGNATION:CASHIER SALARY:86325 EMAIL:JAMSHEDINAM@gmail.com
CALL
```

```
--FUNCTION TO DISPLAY ALL THE CARDS THAT HAVE NOT EXPIRED AND HOW MUCH TIME THEY HAVE LEFT --
--UNTIL THEIR EXPIRY
```

```
CREATE OR REPLACE PROCEDURE EXPIRE(DATE_ DATE)
LANGUAGE PLPGSQL AS $$
    DECLARE
        S CURSOR FOR SELECT EXP_DATE,CARD_NO FROM CARD_;
        EXPIRE BOOLEAN;
        DAYS INTEGER;
        D2 INTEGER;
        REC RECORD;
        EX BOOLEAN;
    BEGIN
        OPEN S;
        LOOP
            FETCH S INTO REC;
            EXIT WHEN NOT FOUND;
            DAYS :=DATE_ - REC.EXP_DATE ;
```

```

IF DATE_ - REC.EXP_DATE >= 0 THEN
    RAISE INFO ' CARD NO % HAS ALREADY EXPIRED',REC.CARD_NO;
ELSE
    DAYS :=DATE_ - REC.EXP_DATE;
    D2 := DAYS * -1;
    RAISE INFO ' CARD NO % WILL EXPIRE AFTER % DAYS ',REC.CARD_NO,D2;
    EX:=FALSE;
END if;
END LOOP;
CLOSE S;
END;$$;

call EXPIRE('2005-01-01');

```

```

INFO: CARD NO 89324327 WILL EXPIRE AFTER 1860 DAYS
INFO: CARD NO 89324328 WILL EXPIRE AFTER 1860 DAYS
INFO: CARD NO 89324329 WILL EXPIRE AFTER 1865 DAYS
INFO: CARD NO 89324330 WILL EXPIRE AFTER 1866 DAYS
INFO: CARD NO 89324331 WILL EXPIRE AFTER 1857 DAYS
INFO: CARD NO 89324332 WILL EXPIRE AFTER 1863 DAYS
CALL

```

## Triggers/Trigger Functions

This trigger function will calculate the years of service of that employee whenever the data of that employee gets modified in our database. It will display the ID of the recently modified employee in database.

```

CREATE OR REPLACE function y_o_s()
RETURNS TRIGGER AS $$
    DECLARE
    C_ CURSOR FOR SELECT STRT_DATE,E_ID FROM employee;
    DAYS INT;
    YEARS NUMERIC(10,2);
    REC RECORD;
    BEGIN
    OPEN C_;
    LOOP
    FETCH C_ INTO REC;
    EXIT WHEN NOT FOUND;
    IF new.e_id=REC.e_id THEN
    DAYS := current_date - REC.STRT_DATE;
    IF DAYS<31 THEN
        YEARS:=DAYS;
        RAISE NOTICE 'EMPLOYEE ID NO. % HAS SPENT LESS THAN 31 days IN THIS
ORGANIZATION',REC.E_ID;
    ELSIF DAYS<365 THEN

```

```

        YEARS:=DAYS/12;
        RAISE NOTICE 'EMPLOYEE ID NO. % HAS SPENT 1 YEAR  IN THIS ORGANIZATION',REC.E_ID;
    ELSIF  DAYS>=365 THEN
        YEARS :=(DAYS/365);
        RAISE NOTICE 'EMPLOYEE ID NO. % HAS SPENT % YEARS IN THIS
ORGANIZATION',REC.E_ID,YEARS;
    END IF;
END IF;
END LOOP;
CLOSE C_;
RETURN NEW;
END;$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER CALCULATE_y_O_S
AFTER UPDATE
ON EMPLOYEE
FOR EACH ROW
EXECUTE PROCEDURE y_O_S();

UPDATE EMPLOYEE SET SAL=SAL+1 WHERE e_id=1;

```

Data Output   Explain   Messages   Notifications

NOTICE:   EMPLOYEE ID NO. 1 HAS SPENT 19.00 YEARS IN THIS ORGANIZATION  
UPDATE 1

Query returned successfully in 25 msec.

This trigger function will stop the insert or update query from executing if the new balance of the saving account exceeds the maximum balance limit set by the bank.

```

CREATE OR REPLACE function ACC_HAVING_MAX_LIMIT()
RETURNS TRIGGER AS $$
    DECLARE
    C CURSOR FOR  SELECT A.C_ID,A.ACCOUNT_NO,A.BALANCE
    FROM ACCOUNT A
    WHERE A.ACC_TYPE='SAVING';
    ACC_NO INT;
    REC RECORD;
    REC1 RECORD;
    BEGIN
    OPEN C;
    LOOP
    FETCH C INTO REC;
    EXIT WHEN NOT FOUND;
    --Raise INFO '1';

```



```

--RAISE INFO '%',NEW.ACCOUNT_NO;
--RAISE INFO '%',REC.ACCOUNT_NO;

IF REC.ACCOUNT_NO=NEW.ACCOUNT_NO THEN
RAISE INFO '%',NEW.ACCOUNT_NO;
  IF 10000000<NEW.BALANCE THEN
    Raise EXCEPTION 'Exceeds allowed limit';
    RETURN NULL;
  END IF;
END IF;
END LOOP;
CLOSE C;
RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';
CREATE OR REPLACE TRIGGER CALCULATE_ACC
BEFORE INSERT OR UPDATE
ON ACCOUNT
FOR EACH ROW
EXECUTE PROCEDURE ACC_HAVING_MAX_LIMIT();

UPDATE ACCOUNT SET balance=100000000 WHERE account_no=9993;

```

INFO: 9993

ERROR: Exceeds allowed limit

CONTEXT: PL/pgSQL function acc\_having\_max\_limit() line 21 at RAISE  
SQL state: P0001

This trigger function will stop the insert or update query from executing if the balance of the current account is below the minimum balance required by the bank account to remain functional.

```

CREATE OR REPLACE function ACC_HAVING_MIN_LIMIT()
RETURNS TRIGGER AS $$
  DECLARE
  C CURSOR FOR SELECT A.C_ID,A.ACCOUNT_NO,A.BALANCE
  FROM ACCOUNT A
  WHERE A.ACC_TYPE='CURRENT';
  ACC_NO INT;
  REC RECORD;
  REC1 RECORD;
  BEGIN
  OPEN C;
  LOOP
  FETCH C INTO REC;
  EXIT WHEN NOT FOUND;

```

```

IF REC.ACCOUNT_NO=NEW.ACCOUNT_NO THEN
RAISE INFO '%',NEW.ACCOUNT_NO;
  IF 100>NEW.BALANCE THEN
    Raise EXCEPTION 'The balance has dropped below the allowed limit';
    RETURN NULL;
  END IF;
END IF;
END LOOP;
CLOSE C;
RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';
CREATE OR REPLACE TRIGGER CALCULATE_ACC
BEFORE INSERT OR UPDATE
ON ACCOUNT
FOR EACH ROW
EXECUTE PROCEDURE ACC_HAVING_MIN_LIMIT();

UPDATE ACCOUNT SET balance=99 WHERE account_no=3008;

```

Data Output	Explain	Messages	Notifications
-------------	---------	----------	---------------

INFO: 3008

ERROR: The balance has dropped below the allowed limit  
CONTEXT: PL/pgSQL function acc\_having\_min\_limit() line 18 at RAISE  
SQL state: P0001

## Views

Views were created between tables that were created from tertiary relationships and also for those tables that were derived and their corresponding base tables (e.g. savings account table and main account table).

```

CREATE OR REPLACE VIEW public.account_assoc_card
AS
SELECT a_.account_no,
       a_.balance,
       a_.c_id,
       c_.card_no,
       c_.exp_date,
       c_.card_type
FROM account a_

```

```
JOIN card_ c_ ON a_.account_no = c_.account_no;
```

	account_no integer	balance integer	c_id integer	card_no bigint	exp_date date	card_type character varying (50)
1	3000	53500	1000	89324327	2010-02-04	Platinum
2	3001	5454	1001	89324328	2010-02-04	Silver
3	3002	3455	1002	89324329	2010-02-09	Gold
4	3003	5444	1003	89324330	2010-02-10	Master
5	3004	8000	1004	89324331	2010-02-01	Gold
6	3005	9000	1005	89324332	2010-02-07	Silver

```
CREATE OR REPLACE VIEW public.account_link
AS
SELECT cu.c_id,
       cu.f_name,
       cu.l_name,
       cu.email,
       acc.account_no,
       acc.balance,
       acc.acc_type,
       accop.e_id,
       accop.authorization_status
FROM customer cu
     JOIN account acc USING (c_id)
     JOIN account_opening_record accop USING (account_no);
```

	c_id integer	f_name character varying (50)	l_name character varying (50)	email character varying (50)	account_no integer	balance integer	acc_type character varying (50)	e_id integer	authorization_status character varying (50)
1	1001	Umer	Asif	sample1@xyz.com	3001	5454	CURRENT	26	PROCESSED
2	1002	Hamza	Chan	sample2@xyz.com	3002	3455	SAVING	27	PROCESSED
3	1003	NASIR	SOHAIL	sample3@xyz.com	3003	5444	CURRENT	28	PROCESSED
4	1004	IMRAN	HASHMI	sample4@xyz.com	3004	8000	SAVING	29	PROCESSED
5	1005	JAVED	AWAN	sample5@xyz.com	3005	9000	CURRENT	30	PROCESSED
6	1002	Hamza	Chan	sample2@xyz.com	3006	8080	CURRENT	31	PROCESSED
7	1003	NASIR	SOHAIL	sample3@xyz.com	3007	2900	SAVING	31	PROCESSED
8	2	test2	test2	test2@whatever.com	3010	3500	SAVING	100	REMOVED
9	1000	Muhammad	Farae	sample@xyz.com	3000	53500	CURRENT	25	PROCESSED
10	1	It	Worked		3009	79905500	SAVING	100	PROCESSED

```
CREATE OR REPLACE VIEW public.current_accounts
AS
SELECT a_.account_no,
       a_.acc_type,
       a_.balance,
```

```
s.min_balance
FROM account a_
JOIN current s ON a_.account_no = s.account_no;
```

	account_no integer	acc_type character varying (50)	balance integer	min_balance integer
1	3001	CURRENT	5454	100
2	3003	CURRENT	5444	100
3	3005	CURRENT	9000	100
4	3006	CURRENT	8080	100
5	3000	CURRENT	53500	100

```
CREATE OR REPLACE VIEW public.info_
AS
SELECT a_.account_no AS "SENDER",
       a_.acc_type,
       a_.balance,
       t_.trans_id,
       t_.trans_date,
       t_.status_ AS "STATUS",
       tr.account_no_receiving_funds AS "RECIEVER"
FROM account a_
JOIN transactions t_ ON a_.account_no = t_.account_no
JOIN transfer_record tr ON a_.account_no = tr.account_no;
```

	SENDER integer	acc_type character varying (50)	balance integer	trans_id integer	trans_date date	STATUS character varying (50)	RECIEVER integer
1	3000	CURRENT	53500	100	2003-12-04	PROCESSED	3004
2	3000	CURRENT	53500	100	2003-12-04	PROCESSED	3010
3	3000	CURRENT	53500	100	2003-12-04	PROCESSED	3010
4	3002	SAVING	3455	102	2003-12-08	PENDING	3001
5	3003	CURRENT	5444	103	2003-12-07	PROCESSED	3002
6	3000	CURRENT	53500	113	2022-06-01	PROCESSED	3004
7	3000	CURRENT	53500	113	2022-06-01	PROCESSED	3010
8	3000	CURRENT	53500	113	2022-06-01	PROCESSED	3010
9	3000	CURRENT	53500	114	2022-06-01	PROCESSED	3004
10	3000	CURRENT	53500	114	2022-06-01	PROCESSED	3010
11	3000	CURRENT	53500	114	2022-06-01	PROCESSED	3010
12	3000	CURRENT	53500	116	2022-06-03	PROCESSED	3004
13	3000	CURRENT	53500	116	2022-06-03	PROCESSED	3010
14	3000	CURRENT	53500	116	2022-06-03	PROCESSED	3010
15	3000	CURRENT	53500	118	2022-06-03	PROCESSED	3004
16	3000	CURRENT	53500	118	2022-06-03	PROCESSED	3010
17	3000	CURRENT	53500	118	2022-06-03	PROCESSED	3010

```
CREATE OR REPLACE VIEW public.loan_info
AS
```

```
SELECT l.loan_id,
       l.loan_period,
       l.amount,
       r.repayment_date
FROM loan l
FULL JOIN repayment r ON l.loan_id = r.loan_id;
```

	loan_id integer	loan_period integer	amount integer	repayment_date date
1	238	18	90000	[null]
2	234	24	23000	[null]
3	233	12	10000	[null]
4	235	32	20000	[null]
5	236	12	30000	[null]
6	237	24	35000	[null]

```
CREATE OR REPLACE VIEW public.saving_accounts
AS
SELECT a_.account_no,
       a_.acc_type,
       a_.balance,
       s.max_limit
FROM account a_
JOIN savings s ON a_.account_no = s.account_no;
```

	account_no integer	acc_type character varying (50)	balance integer	max_limit bigint
1	3002	SAVING	3455	10000000
2	3004	SAVING	8000	10000000
3	3007	SAVING	2900	10000000

## Indexes

As learned in the course, creating indexes decreases access time but takes up considerable storage if used excessively, and thus we only declared indexes using B-Tree data structure occasionally for some foreign keys and other miscellaneous attributes which were forecasted to require substantial number of access during runtime of the application.

```
CREATE UNIQUE INDEX IF NOT EXISTS account_opening_index
ON public.account_opening_record USING btree
(c_id ASC NULLS LAST, account_no ASC NULLS LAST, e_id ASC NULLS LAST)
TABLESPACE pg_default;

CREATE INDEX IF NOT EXISTS fki_postcode_branch_fk
ON public.branch USING btree
```

```

(postcode ASC NULLS LAST)
TABLESPACE pg_default;

CREATE INDEX IF NOT EXISTS mgr_id_index
ON public.branch USING btree
(mgr_id ASC NULLS LAST)
TABLESPACE pg_default;

CREATE INDEX IF NOT EXISTS card_account_no_index
ON public.card_ USING btree
(account_no ASC NULLS LAST)
TABLESPACE pg_default;

CREATE INDEX IF NOT EXISTS current_account_no
ON public.current USING btree
(account_no ASC NULLS LAST)
TABLESPACE pg_default;

CREATE UNIQUE INDEX IF NOT EXISTS customer_postcode
ON public.customer USING btree
(postcode ASC NULLS LAST)
TABLESPACE pg_default;

CREATE INDEX IF NOT EXISTS fki_postcode_cust_fk
ON public.customer USING btree
(postcode ASC NULLS LAST)
TABLESPACE pg_default;

CREATE INDEX IF NOT EXISTS fki_serv_id_fk
ON public.customer_service_types USING btree
(service_id ASC NULLS LAST)
TABLESPACE pg_default;

CREATE INDEX IF NOT EXISTS fki_postcode_home_fk
ON public.home_loan USING btree
(postcode ASC NULLS LAST)
TABLESPACE pg_default;

CREATE INDEX IF NOT EXISTS fki_acc_rec_fk
ON public.transfer_record USING btree
(account_no_receiving_funds ASC NULLS LAST)
TABLESPACE pg_default;

```



## Miscellaneous

Below shows the recursive query showing the managers hierarchy of the employees from the employee table.

```
WITH recursive WORKERS_DESIGNATION AS
(
  SELECT e_id,manages_eid -- base condition or anchor condition
  FROM employee where manages_eid is NULL

  UNION
  -- recursive member,
  SELECT E_.e_id,E_.manages_eid
  FROM WORKERS_DESIGNATION E1
  JOIN employee E_ ON E1.e_id=E_.manages_eid
  WHERE E1.E_ID IS NOT NULL
)

SELECT e_id, MANAGES_EID FROM WORKERS_DESIGNATION
GROUP BY MANAGES_EID, e_id
order by e_id;
```

	Data Output	Explain	Messages
	 e_id integer	 manages_eid integer	
1	1	100	
2	2	100	
3	3	100	
4	4	100	
5	5	100	
6	6	100	
7	7	1	
8	8	1	
9	9	1	
10	10	1	
11	11	2	
12	12	2	
13	13	2	
14	14	2	
15	15	2	
16	16	3	
17	17	3	
18	18	3	
19	19	3	
20	20	4	
21	21	4	
22	22	4	
23	23	4	
24	24	5	
25	25	5	
26	26	5	
27	27	5	
28	28	6	