The Honest Bank

CSC375_Project

Abdulrahman Zaatari

Mamdouh Dabjan

Alain Samaha

Kevin Kaddoum

Table of Content:

1- Introduction:

Trust and integrity are the cornerstones of any financial institution. Unfortunately, however, we are suffering from the lack of those essential traits in the financial institutions of our dear Lebanon. That is why we decided to launch the “Honest Bank” as we hope it will later stand as a beacon of transparency and reliability. While the financial landscape has evolved with technological advancements, our commitment to honesty and ethical banking practices shines as our strongest point. In an era of uncertainty, Honest Bank emerges as a trustworthy partner, upholding the highest standards of integrity.

At Honest Bank, we recognize that people seek more than just financial services; they seek confidence and perhaps some guidance in their financial decisions. We take pride in our role as a reliable source for your banking needs, offering not just financial products, but also a commitment to providing you with accurate information and personalized services.

Our online banking platform is designed with your convenience in mind. With user-friendly interfaces and robust security measures, you can access your accounts, manage transactions, and make informed financial choices from the comfort of your home. We prioritize accessibility, ensuring that our services are available to a wide range of clients, regardless of their location or background. We also offer the option of crypto transactions!

Diversity and inclusivity are values that underpin our approach to banking. Honest Bank caters to the unique financial needs of every individual. Our team of dedicated financial experts and advisors brings a wealth of knowledge to cover a wide spectrum of financial topics, ensuring that you find guidance and solutions tailored to your interests and goals.

Effective management is the backbone of our operations. Our commitment to efficient resource management ensures the smooth functioning of our bank. From overseeing the supply of essential resources to timely paperless transactions, our administration team works tirelessly to streamline processes and ensure the security of your financial information, led by our skilled security officer **Alain Samaha**.

We envision a future where banking is not just about numbers but about relationships built on trust. Our team, led by Managing Director **Abdulrahman Zaatari**, combines their expertise to ensure that your banking experience is not just informative but also memorable. Our commitment to transparency extends to our advertising administrator, **Kevin Kaddoum**, who ensures that our advertising partners receive the best value for their investments. We invite you to join us on this journey towards a banking experience built on honesty, integrity, and your financial well-being.

2- Usages of Database System

- Online banking and mobile apps
- Retail Banking
- Corporate Banking
- Cryptocurrency services
- Foreign exchange (Forex)
- Analytics and Business intelligence

3- Tools used to draw the ER:

The tool our team has used to implement the ER Diagram is draw.io. This tool has provided us with a beneficial environment to work on a collective vision. This software also has a selection of diagrams and tools representing the different structures of an ER diagram which simplifies the modeling process. The fluidity of the platform has made it convenient for us to implement our ideas and construct a well-structured diagram.

4- System Description & Requirements:

Our system consists of 17 main entities each with their own attributes. Further information is going to be provided about each entity, attribute, and relationship in section 5.1 & 5.2:

- A customer is a person that has an account with a bank and/or a crypto wallet. Customers can use their accounts to deposit and withdraw money, make transfers, pay bills, apply for loans, and use cryptocurrency functionalities.

- An account is a financial relationship between a customer and a bank. Accounts can be used to deposit and withdraw money, make transfers, pay bills, and receive direct deposits. Each account can be linked to beneficiary for fast transactions or inheritance purposes. An account is used to pay a loan. It is also linked with one cryptowallet.

- A transaction is a financial event that takes place between 1 or 2 accounts. Transactions can be used to deposit and withdraw money, make transfers, pay bills, and purchase goods and services.

- Cryptocurrencies are digital or virtual currencies that can be used to send and receive payments, as well as store value.

- A cryptocurrency wallet is a digital wallet that is used to store and manage cryptocurrency. Each crypto wallet is linked to an account.

- A loan is a financial agreement between a bank and a borrower. The bank agrees to lend money to the borrower, who then repays the loan with interest over time. It consists of several types including: Student Loan, Personal Loan & Home Loan. Each loan can be paid by one account.

- A credit card is owned by a customer who can then use it to purchase products online or withdraw money from any branch of the bank.

- Branches are physical locations where customers can go to interact with their bank. Each branch is managed by an employee.

- ATMs are self-service machines located within bank branches or other physical locations where customers can perform transactions using their accounts and credit cards. Each ATM is typically managed and serviced by bank employees of a branch.

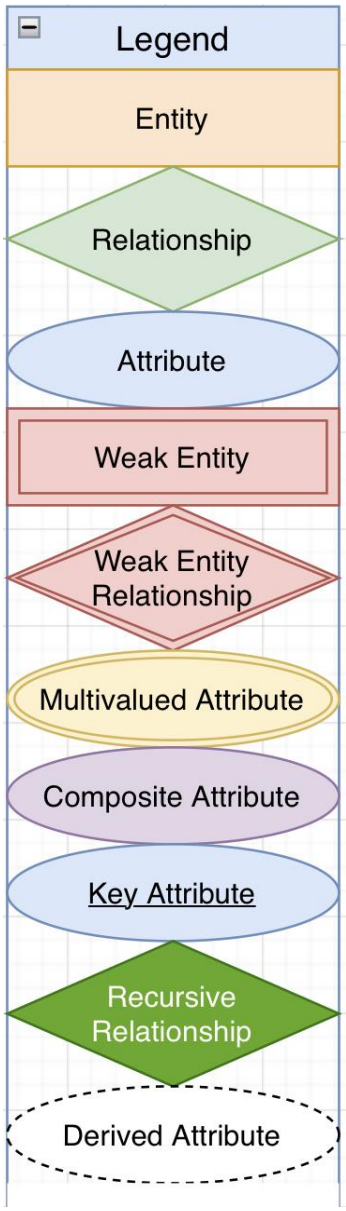
- Employees are the people who work at a bank. They work in specific departments (at least 1). They provide customers with a variety of services, such as opening accounts, making deposits and withdrawals, refilling ATMs and processing loans. They also contribute in publishing cryptonews and help customers with their banking needs. Employees can be supervisors or supervisees and can also have dependants.

- Departments contain employees that work towards one goal. One department can be responsible for transactions, one for publishing, one for IT services and one for crypto… Each department has a manager.

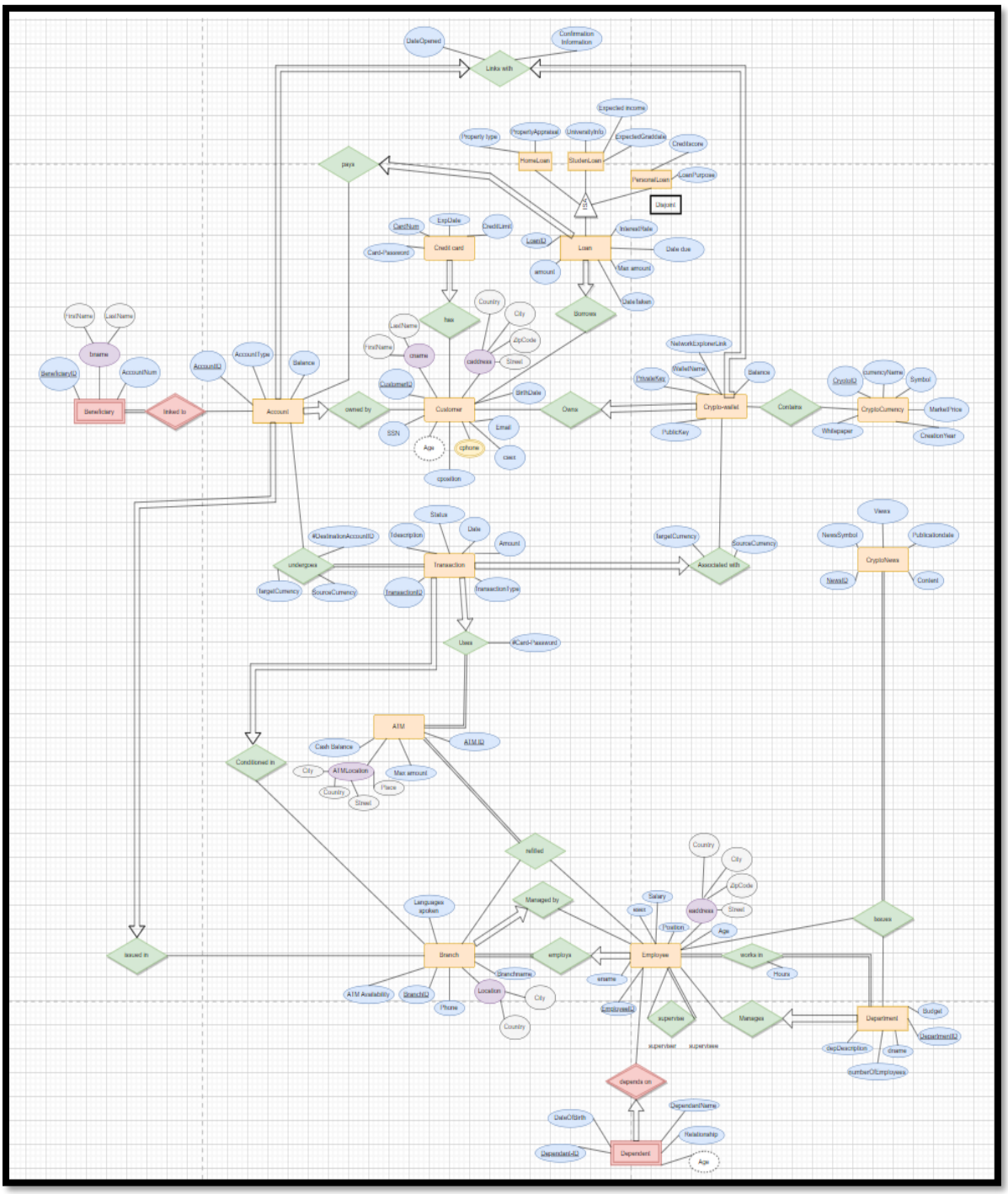
5- Security Measures

We have renamed the attributes of many entities to counter the danger of database manipulation through SQL. The SQL injection attack can delete the database if it knows the names of attributes & entities used. For example, we renamed the Name attribute of Employee to ename, and Sex attribute to esex. We have implemented this throughout all entities in the database.

6- Legend of ER diagram symbols:

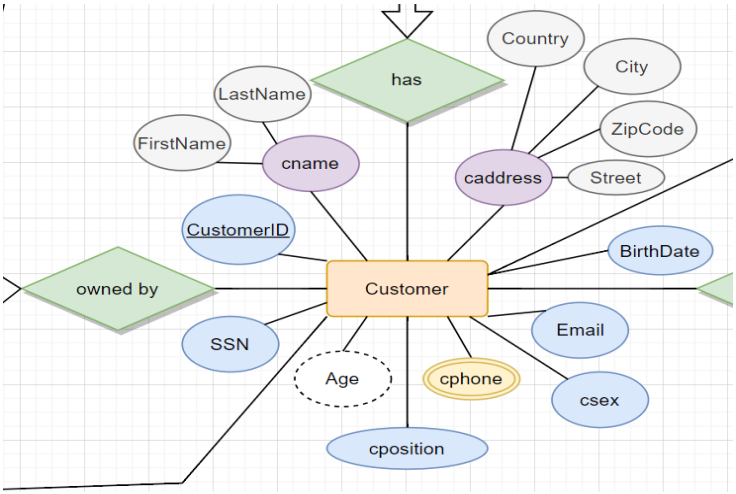


7.1- Updated ER Diagram



7.2- Entity types and their attributes

Customer:

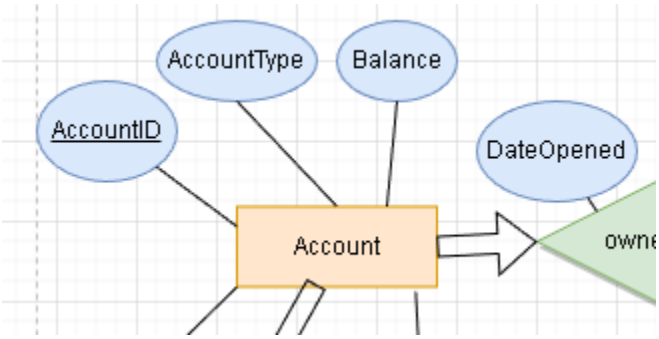


The Customer entity is at the core of the banking system, representing individuals who engage with the bank's services and form the foundation of the financial institution. It is essential for the management of customer relationships and financial transactions.

- **CustomerID:** The primary key and a unique identifier for each customer within the banking system. This CustomerID distinguishes one customer from another.
- **cname:** A composite attribute that combines the first and last names of customers, providing a complete identification.
- **csex:** An attribute to record the gender of customers, allowing for demographic analysis.
- **Email:** Every customer is assigned a unique and personalized email address, serving as the primary means of communication and contact with the bank.
- **Address:** A composite attribute that encompasses multiple sub-attributes, including Country, City, Zip Code, and Street. This comprehensive address information helps track the physical location of customers.
- **Phone:** A multivalued attribute allowing customers to associate one or more phone numbers with their profiles, enhancing contact capabilities.
- **Birthdate:** Day of birth of the customer.
- **Age:** A derived attribute that calculates the age of each customer based on their Date of Birth. It provides information about the customer's demographic.
- **cposition:** A binary attribute that distinguishes between different types of customers based on their roles within the bank. Customers may hold various positions such as account holders, investors, or borrowers.

- **SSN:** This attribute is specifically used to store the SSN of the employee.

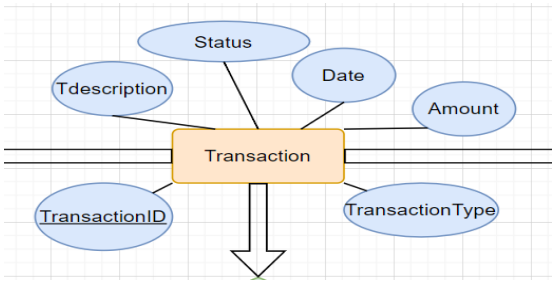
Account:



The Account entity is a fundamental component of the banking system, representing financial accounts held by customers. It includes attributes essential for managing and tracking these accounts.

- **AccountID:** A primary key that uniquely identifies each account in the system.
- **Balance**: Records the current balance of the account, indicating the amount of money held.

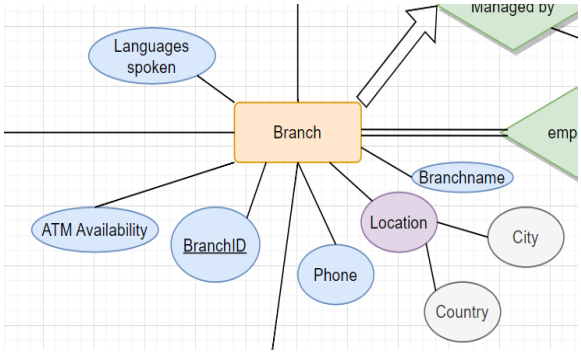
Transaction:



The Transaction entity is responsible for capturing financial activities and operations within the banking system. It includes attributes crucial for recording these transactions.

- **TransactionID:** A primary key that uniquely identifies each transaction.
- **TransactionType:** Specifies the type of transaction, such as deposit, withdrawal, or transfer.
- **Status:** Specifies status (Pending, completed, on hold…)
- **Amount:** Represents the monetary value involved in the transaction.
- **Date:** Records the date when the transaction occurred.
- **Tdescription:** Provides additional information or a description of the transaction.

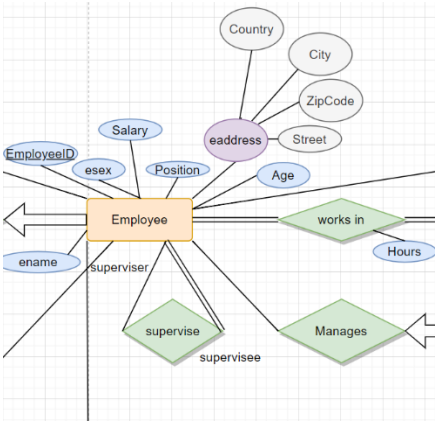
Branch:



The Branch entity represents the physical branches of the bank, providing in-person services to customers. It includes attributes necessary for branch management and identification.

- **<u>BranchID</u>:** A primary key uniquely identifying each branch.
- **BranchName:** The name of the branch, facilitating branch recognition.
- **Language Spoken:** Specifies languages spoken in branch
- **ATM Availability:** Specifies the availability of ATMs.
- **Location:** Specifies the physical location of the branch.
- **PhoneNumber:** Records contact information for the branch.

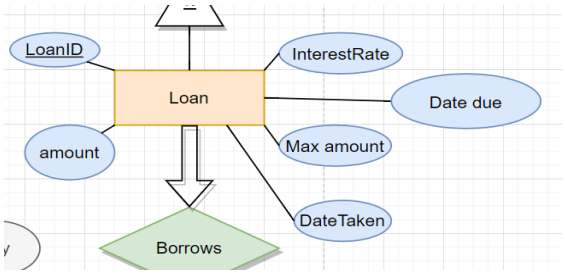
Employee:



The Employee entity represents individuals employed by the bank and is crucial for managing the workforce. It includes attributes essential for employee management.

- **EmployeeID:** A primary key uniquely identifying each employee.
- **ename:** Records the name of the employee.
- **Age:** Age of the employee.
- **Email:** Provides the employee's unique company email address for professional communication.
- **esex:** Determines sex of the employee.
- **Salary:** Records the employee's compensation or wage.
- **Address:** A composite attribute containing sub-attributes like Country, City, Zip Code, and Street, providing comprehensive address information.
- **Position:** Indicates the role or job position held by the employee within the bank.

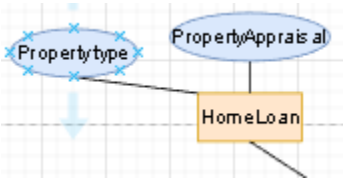
Loan:



The Loan entity represents loans provided by the bank to customers. It includes attributes necessary for managing loan accounts and tracking loan details.

- **LoanID:** A primary key uniquely identifying each loan account.
- **Amount:** Records the loan amount disbursed to the customer.
- **Max amount:** Max amount that can be loaned.
- **InterestRate:** Specifies the interest rate associated with the loan.
- **DateTaken:** Captures the date when the loan was approved or taken by the customer.
- **DateDue:** Records the due date for the loan repayment.

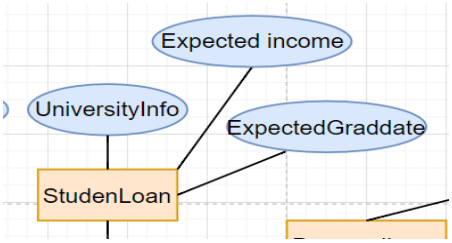
HomeLoan:



A Home Loan is a type of loan specifically designed to help individuals or families purchase residential properties such as houses, apartments, or condominiums. It is also commonly referred to as a mortgage.

- **Property Type:** This attribute specifies the type of property that the home loan is being used to finance. Property types can include single-family homes, condominiums, apartments, townhouses, or other residential structures.
- **Property Appraisal:** Property appraisal refers to the process of determining the current market value of the property that is being purchased with the home loan.

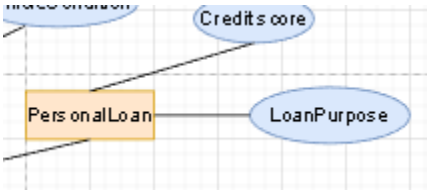
StudentLoan:



A Student loan is a type of loan that is used to finance the purchase of an automobile, including cars, trucks, motorcycles, or other vehicles.

- **University Info:** The educational institution that the student belongs to.
- **Expected Grad-date:** Expected Graduation date of the student.
- **Expected Income:** Expected income which helps decide payments.

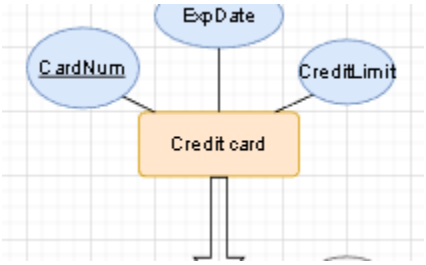
PersonalLoan:



A Personal Loan is a type of loan that is not tied to a specific purpose like buying a home or a car. It is a loan granted to an individual for various personal expenses, including debt consolidation, medical bills, travel, or other personal needs.

- **Credit Score:** Credit score is a numerical representation of an individual's creditworthiness. Lenders use credit scores to assess the risk associated with lending to a borrower.
- **Loan Purpose:** The loan purpose attribute indicates the reason or purpose for which the personal loan is being requested. Common loan purposes can include debt consolidation, medical expenses, home improvement, education, travel, or any other personal financial need.

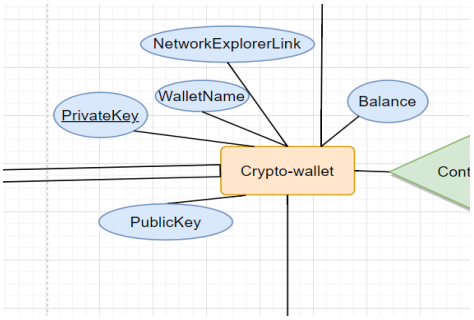
CreditCard:



The CreditCard entity represents credit card accounts issued by the bank. It includes attributes essential for managing credit card accounts.

- **CardNumber:** A primary key uniquely identifying each credit card account.
- **ExpirationDate:** Records the expiration date of the credit card.
- **CreditLimit:** Specifies the credit limit associated with the card.

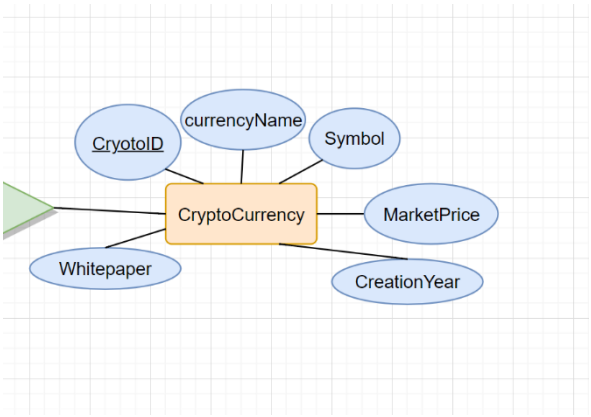
CryptocurrencyWallet:



The CryptocurrencyWallet entity serves as a digital repository for cryptocurrencies and is critical for managing cryptocurrency assets. It includes attributes such as WalletID, WalletName, Balance, and CustomerID.

- **Private key:** A primary key uniquely identifying each cryptocurrency wallet only known by the user.
- **Public key:** Known by other users to do transactions.
- **WalletName:** Provides a name or label for the cryptocurrency wallet.
- **Balance:** Records the current holdings of cryptocurrencies in the wallet.
- **NetworkExplorerLink:** A link to a blockchain explorer to view the wallet's transaction history and balance.

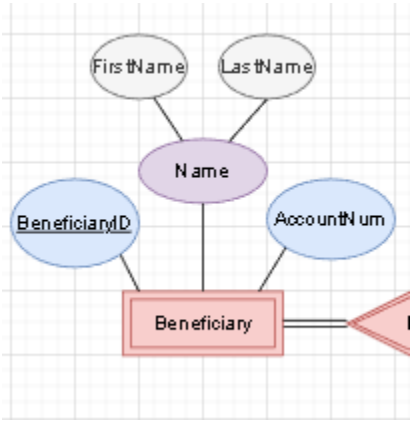
Cryptocurrency:



The Cryptocurrency entity represents general information about cryptocurrencies, which may include Bitcoin, Ethereum, and others.

- **CryptoID:** A primary key uniquely identifying each cryptocurrency.
- **currencyName:** The name of the cryptocurrency, such as Bitcoin or Ethereum.
- **Symbol:** The symbol or code associated with the cryptocurrency.
- **MarketPrice:** Attribute that provides the current market price or value of the cryptocurrency.
- **CreationYear:** Year of creation for the cryptocurrency.
- **Whitepaper:** link to the original whitepaper that describes the cryptocurrency's technology, purpose, and vision.

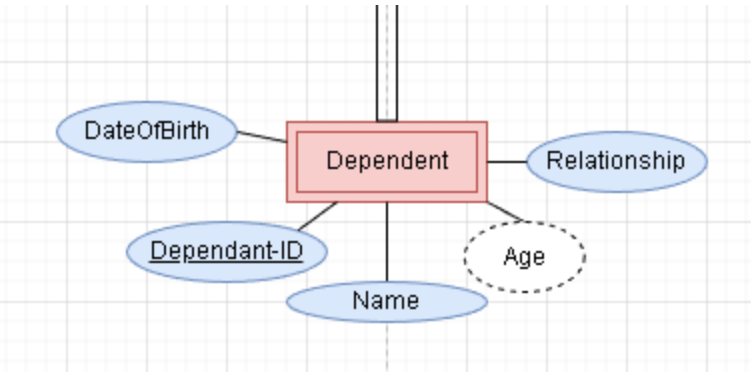
Beneficiary (w):



The Beneficiary entity is a weak entity that plays a role in facilitating fund transfers and ensuring the secure and accurate distribution of assets. It encompasses attributes such as BeneficiaryID (Primary key), Name, AccountNumber, and Relationship. These attributes enable the tracking and management of beneficiaries associated with various financial accounts.

- **BeneficiaryID:** The primary key that uniquely identifies each beneficiary within the banking system.
- **bname:** Records the full name of the beneficiary.
- **AccountNumber:** Specifies the account number associated with the beneficiary, ensuring accurate fund transfers.

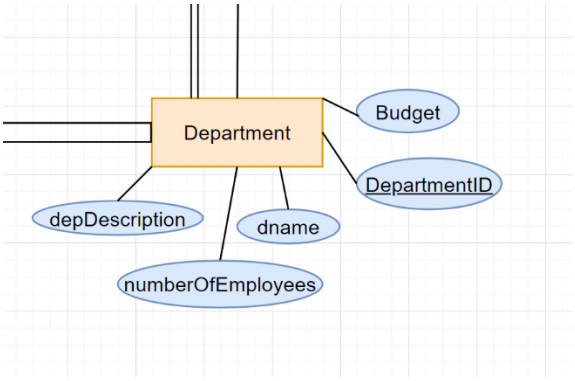
Dependent (w):



The Dependent is a weak entity that represents individuals who rely on employees within the organization for financial support and benefits. It plays a role in understanding the familial and financial dependencies within the workforce.

- **DependantID:** The primary key that uniquely identifies each dependant within the banking system.
- **DependantName:** Records the full name of the dependant.
- **Age:** It is a derived attribute that Indicates the age of the dependant, which helps in understanding their life stage and financial needs.
- **DateOfBirth:** Captures the date of birth of the dependant.
- **Relationship:** Specifies the relationship between the dependant and the employee, such as spouse, child, or other family ties.

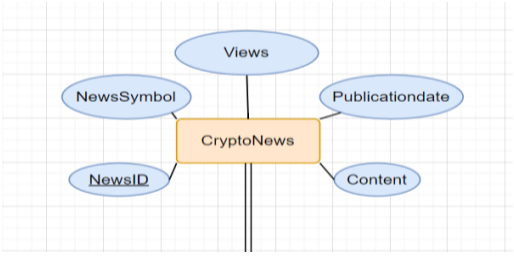
Department:



The Department entity is created to streamline and manage various aspects of the bank activities efficiently.

- **DepartmentID:** DepartmentID is a unique identifier or code assigned to each department within an organization. This attribute is used to distinguish one department from another.
- **NumberOfEmployees:** NumberOfEmployees represents the count or total number of individuals who are employed or assigned to work within a specific department. It provides an indication of the department's size in terms of staff.
- **Budget:** The Budget attribute refers to the financial allocation or resources allocated to a department for its operations, projects, and expenses. It typically includes funds for salaries, equipment, supplies, and other operational costs.
- **dname:** The dname attribute is the formal or commonly recognized title or label given to the department. It helps in identifying and referring to the department within the organization.
- **depDescription:** depDescription provides a brief explanation or overview of the department's purpose, functions, or responsibilities. It helps employees and outsiders understand the role of the department within the bank.

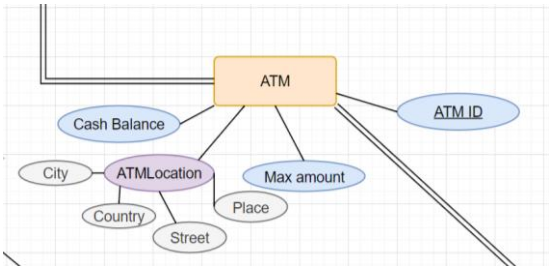
CryptoNews:



Refers to news and information related to cryptocurrencies and the broader blockchain technology ecosystem. Cryptocurrencies are digital or virtual currencies that use cryptography for security and operate on decentralized ledger technology called blockchain.

- **NewsID:** This is the primary key for the table, which means it uniquely identifies each row in the table. It should be an integer and serve as a unique identifier for each crypto news article.
- **Views:** This field represents the number of views the news article has received. It's of type integer.
- **Publicationdate:** This field stores the date when the news article was published. It's of type date.
- **Content:** This field stores the content or text of the news article. It's of type text, which is suitable for storing longer text data.
- **NewsSymbol:** Associate each news article with a particular cryptocurrency or symbol.

ATM:



ATMs are machines found in branches used by customers to withdraw money from their account usings their credit cards passwords for access.

ATM ID : This is the primary key for the ATM entity, uniquely identifying each ATM in the system. It should be of type integer and serves as a unique identifier for each ATM.

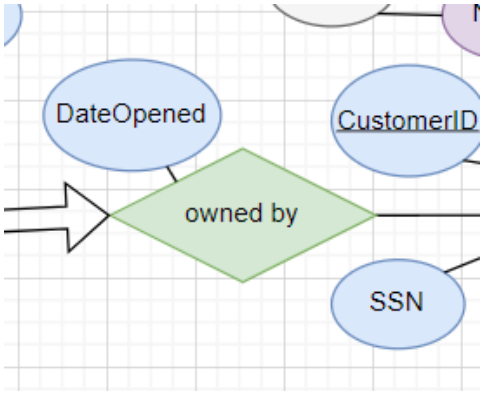
Cash Balance: This attribute represents the amount of cash currently available in the ATM for dispensing to customers. It is of type decimal or currency, depending on your database system.

Maximum amount: This attribute specifies the maximum amount of cash that a user can withdraw at once. It's also of type decimal.

ATMLocation: A composite attribute specifying the Country, City, Street and place the ATM is found in.

7.3 Relationships and their explanations

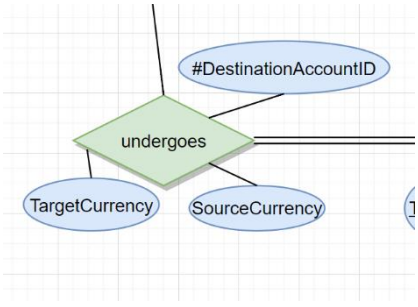
Customer-Owns-Account:



The "Customer-Owns-Account" relationship establishes ownership connections between customers and their respective accounts. It exhibits a one-to-many relationship, signifying that a customer may own multiple accounts, while an account can belong to only one customer. This relationship is crucial for maintaining a record of account ownership and facilitating various financial transactions and services for customers.

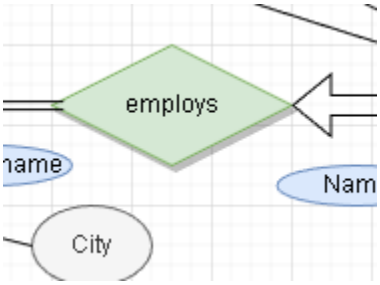
- DateOpened: Represents the date when the account was opened, providing historical data.

Account-Transaction Undergoes relationship:



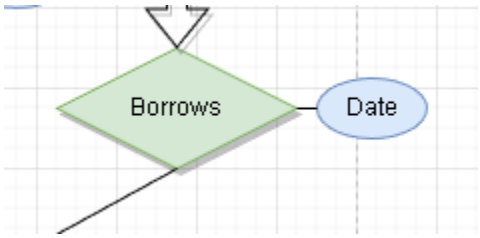
The " Account-Transaction Undergoes relationship:" relationship links accounts to their associated transactions. It exemplifies a many to many relationship, indicating that an account can have multiple transactions, such as deposits, withdrawals, transfers, and more. Conversely, each transaction is associated with a specific account. It includes attributes such as TargetCurrency, SourceCurrency, DestinationAccountID.

Branch-Employee-Works-At:



The "Branch-Employee-Works-At" relationship connects branch locations with employees who work at those branches. It demonstrates a one-to-many relationship, allowing a branch to have multiple employees, each fulfilling different roles. Employees, in turn, are associated with a specific branch where they perform their job responsibilities. This relationship facilitates the organization and management of branch operations and staffing.

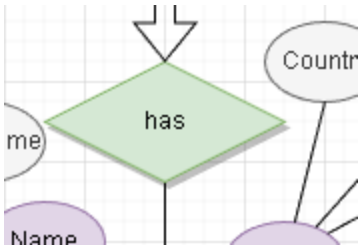
Customer-Borrows-Loan:



The "Customer-Borrows-Loan" relationship associates customers with loans they have acquired. It represents a one-to-many relationship, indicating that a customer can have multiple loans, such as personal loans or mortgages. Each loan is linked to a particular customer who is responsible for repayment. This relationship plays a pivotal role in loan management and tracking customer obligations.

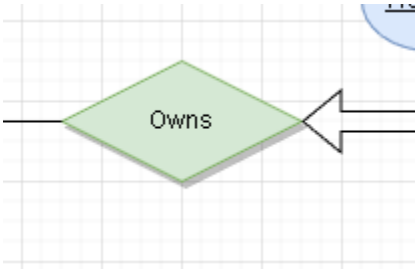
-Date: It is a attribute which keeps track of the loan date.

Customer-Has-CreditCard:



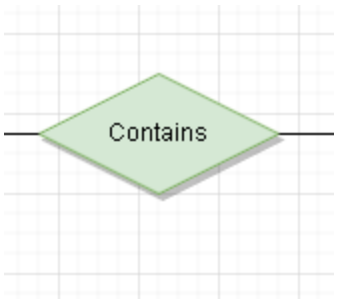
The "Customer-Has-CreditCard" relationship establishes connections between customers and their credit cards. It exhibits a one-to-many relationship, allowing customers to possess multiple credit cards, each with its unique features and limits. Each credit card is linked to a specific customer who uses it for financial transactions and credit-related activities. This relationship is vital for credit card issuance and management.

Customer-Owns-CryptocurrencyWallet:



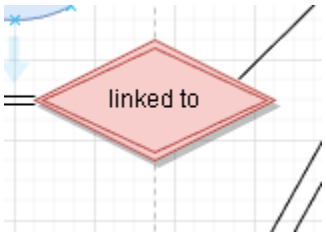
The "Customer-Owns-CryptocurrencyWallet" relationship links customers to their cryptocurrency wallets, serving as digital repositories for cryptocurrencies. It represents a one-to-many relationship, enabling customers to own multiple cryptocurrency wallets of various types. Each wallet is associated with a specific customer who manages their cryptocurrency assets. This relationship is crucial for cryptocurrency asset management and security.

CryptocurrencyWallet-Contains-Cryptocurrency:



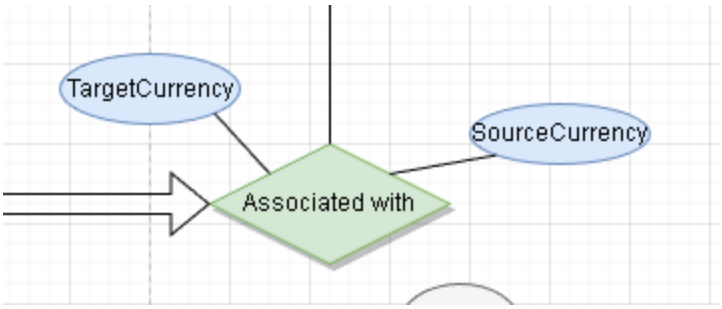
The "CryptocurrencyWallet-Contains-Cryptocurrency" relationship illustrates the contents of cryptocurrency wallets. It signifies a one-to-many relationship, indicating that a cryptocurrency wallet can contain multiple types of cryptocurrencies, such as Bitcoin, Ethereum, or others. Each cryptocurrency is held within a specific wallet, and this relationship aids in tracking cryptocurrency holdings and transactions.

Account-Has-Beneficiary:



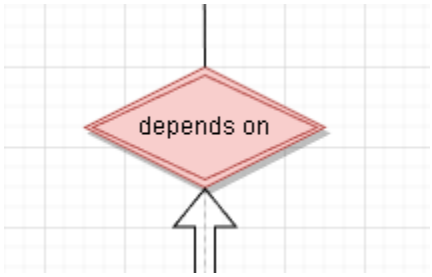
The "Account-Has-Beneficiary" relationship establishes associations between accounts and their designated beneficiaries. It represents a one-to-many relationship, permitting an account to have multiple beneficiaries. Beneficiaries are individuals or entities designated to receive funds from the account in specific circumstances. This relationship aids in efficient fund transfers and inheritance planning.

Transaction-AssociatedWith-CryptocurrencyWallet:



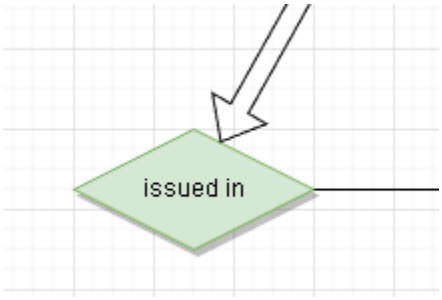
The "Transaction-AssociatedWith-CryptocurrencyWallet" relationship connects cryptocurrency wallets to the transactions conducted within those wallets. It demonstrates a one-to-many relationship, allowing a cryptocurrency wallet to have multiple associated transactions, such as cryptocurrency transfers or exchanges. Each transaction is linked to a specific wallet, facilitating the monitoring of cryptocurrency transaction history and keeps track of the Target and source currencies.

Dependant-DependsOn-Employee:



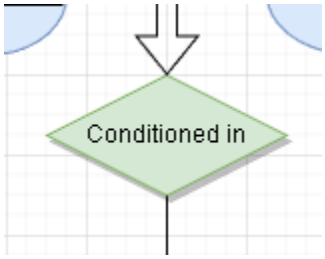
The "Dependent-DependsOn-Employee" relationship signifies the dependency connections between dependents and employees within the organization. It establishes a one-to-many relationship, allowing an employee to have multiple dependents. Dependents are individuals, such as spouses, children, or other family members, who rely on the employee for financial support and benefits.

Account-IssuedIn-Branch:



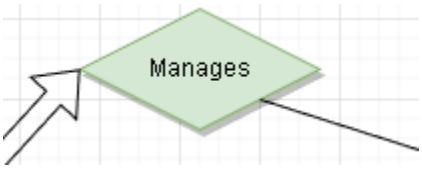
This relationship represents the association between customer accounts and the bank branches where these accounts are held. In a bank, customers open accounts at specific branches, and this relationship captures that connection. One-to-Many (1:N) since each branch can have multiple accounts associated with it, but each account is associated with only one branch.

Transaction-Conditioned In-Branch:



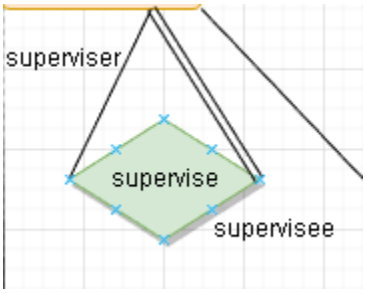
This relationship represents the connection between banking transactions and the bank branches where these transactions occur or are associated with. In a bank, transactions such as deposits, withdrawals, transfers, and more are carried out by customers at specific branches, and this relationship captures that association. Many-to-One (N:1): Many transactions can be associated with a single branch.

Branch-ManagedBy-Employee:



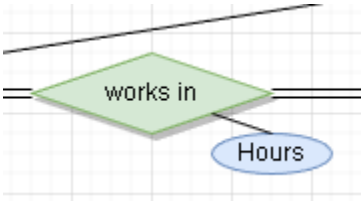
This relationship represents the management hierarchy within a bank branch. In this context, a branch manager is an employee responsible for overseeing the operations and staff within a specific branch. One-to-Many (1:N) as each branch can have one branch manager.

Supervisor:



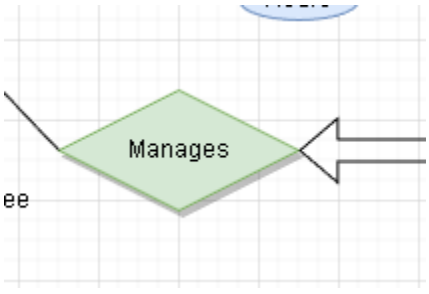
This is a recursive relationship that shows that an employee can be a supervisor of another employee. An employee that is a supervisor can have many supervisees.

Employee-Works in-Department:



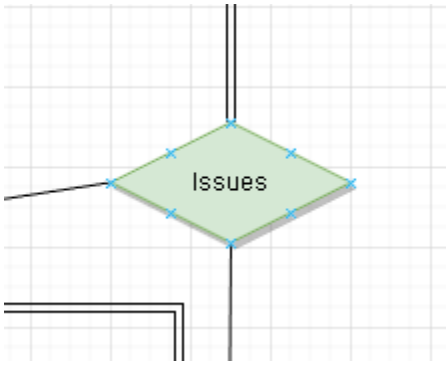
This relationship captures the association between employees and the departments within the bank where they work. In a bank, employees are typically assigned to specific departments based on their roles and responsibilities. Many-to-One (N:1): Many employees can work in one department, but each employee works in only one department at a time.

Employee-Manages-Department:



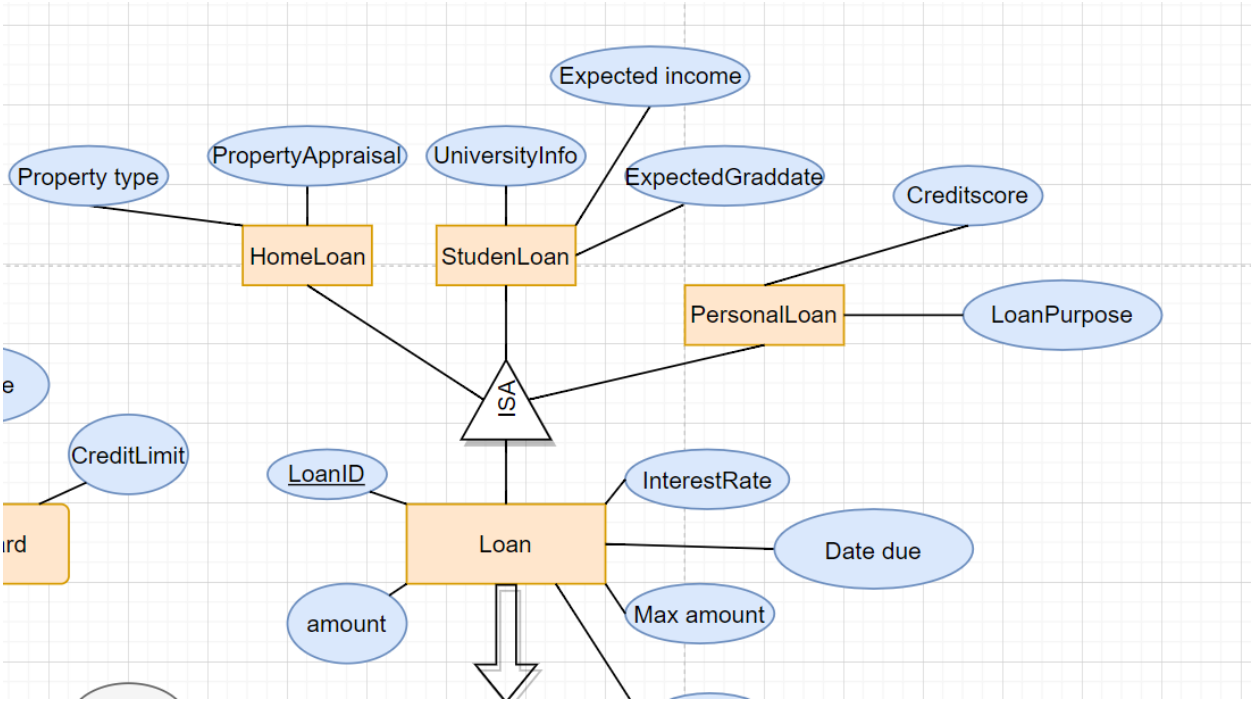
This relationship represents the management hierarchy within a bank, where certain employees hold managerial positions and are responsible for overseeing specific departments. In this context, an employee may serve as the manager of one or more departments within the bank. An employee, such as a department manager or supervisor, can manage one or more departments. However, each department is managed by only one employee at a time.

Issues Ternary Relationship:



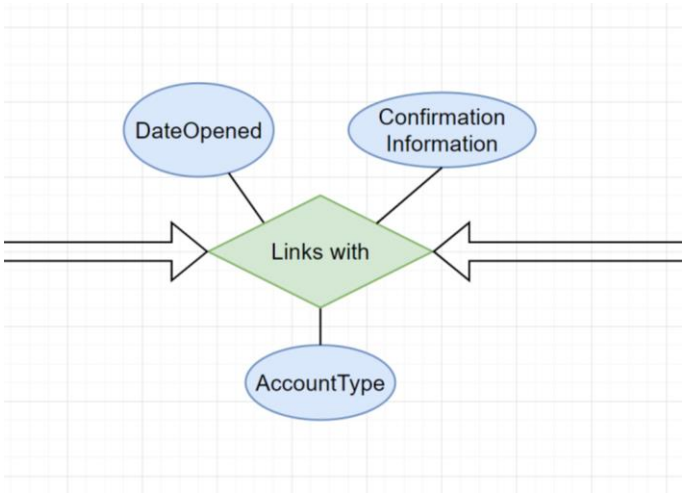
This relationship captures the connection between bank departments, employees and crypto-related news or updates. In the context of our bank, certain employees can work with a department on crypto-related news, reports, or announcements to the bank's customers or the public.

Loan-ISA-HomeLoan, PersonalLoan, AutoLoan:



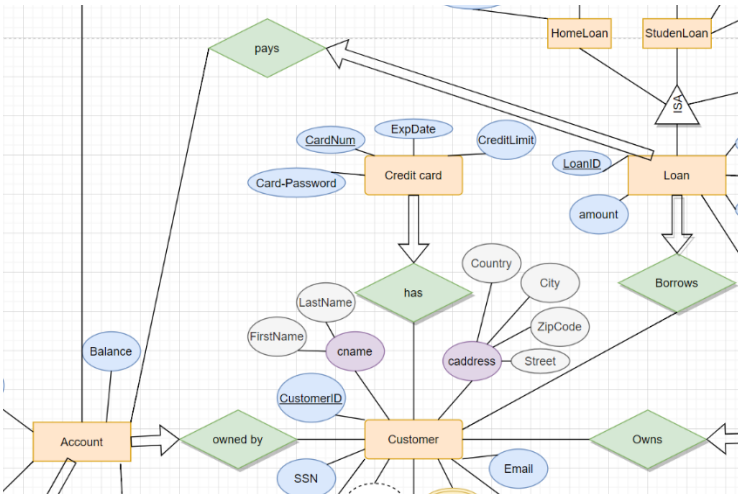
This relationship represents the inheritance or specialization hierarchy within the "Loan" entity. In this context, "Loan" is considered the parent entity, and "HomeLoan", "StudentLoan", and "PersonalLoan" are child entities that inherit attributes and properties from the parent entity but also have their attributes and characteristics specific to the loan types. The type ISA relationship.

Wallet Links with Account:



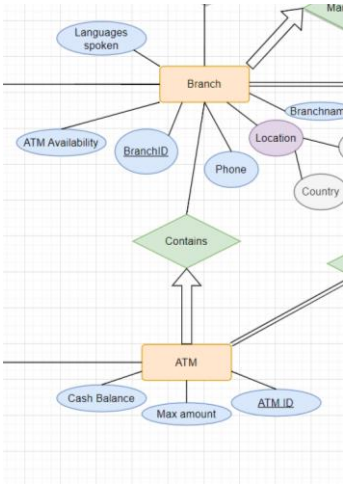
- This relationship links one account with one wallet that contains all the details related to the account.
- **Confirmation Information:** The number of confirmations received for the cryptocurrency transaction and any acknowledgment from the bank regarding the deposit or withdrawal.
- **AccountType:** Specifies the type of the account, such as checking, savings, or others.
- **Date Opened:** Date of account opening.

Account pays Loan:



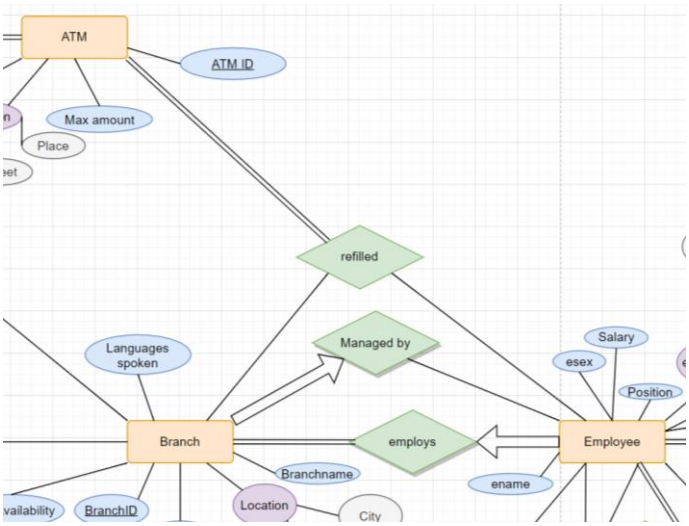
This relationship specifies which account is responsible for paying a specific loan.

Branch contains ATM:



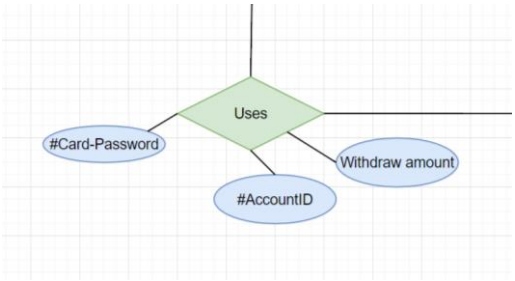
This relationship specifies in which branch an ATM is located. A branch can contain multiple ATMs.

ATM refilled by Employee & branch



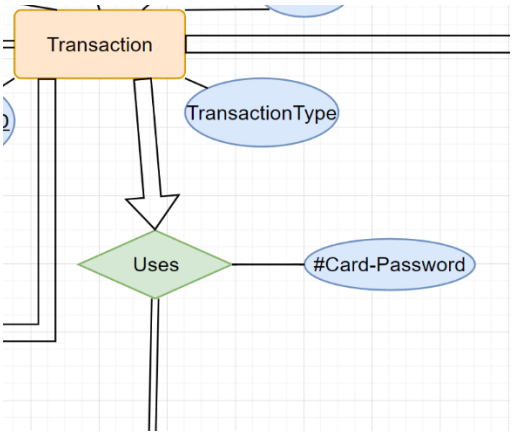
This relationship specifies which employees from which branches can refill a specific ATM. One employee and One branch can refill many ATMs, and an ATM must be refilled by at least one Employee and one branch.

Customer Uses ATM:



Many customers can use many ATMs, and an ATM can be used by many customers to withdraw their money.
A Card-Password is needed to access the ATM, and an AccountID is necessary for knowing which account to withdraw from the amount specified.

Transaction uses ATM:



The "Transaction-ATM Uses relationship**:**" relationship links a transactions to their used ATM. It exemplifies a one to many relationship, indicating that an ATM can be used by multiple transactions. Conversely, each transaction can use a specific ATM. It includes attribute Card password which is used to access ATM account services.

8- ER to Relation Model

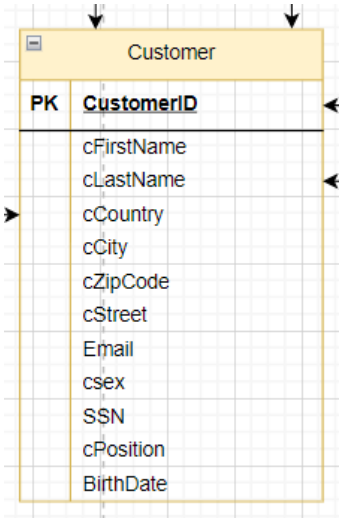
8.1- Strong entities

Every strong entity will undergo a transformation process, resulting in the table. Within this table, every attribute from the strong entity's schema will be included, except for multivalued attributes (represented in different table). Moreover, composite attributes will be dissected and represented by their individual simple attributes.

The key objectives during this conversion process are to establish a structured framework for data storage and retrieval, ensuring that each attribute is accurately represented within the relational model.

We have the following Strong entities: **Customer, Account, Employee, Transaction, CreditCard, Branch, Department, Loan, ATM, CryptoNews, CryptoWallet, and CryptoCurrency.**

Customer:



Customer: (**CustomerID**, cFirstName, cLastName, cCountry, cCity, cZipcode, cStreet, Email, csex, SSN, cPosition, BirthDate)

Phone Number: (**#CustomerID, PhoneNum**)

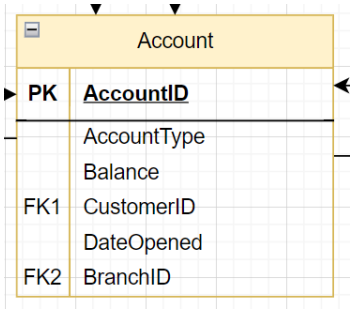
Primary Key (PK): "CustomerID" serves as a unique identifier for each customer, ensuring each customer's record is distinct.

Multivalued Attribute: "cphone" indicates that a customer may have multiple phone numbers, typically managed through a separate related table.

Derived Attribute: "Age" isn't directly stored in the table but can be calculated from the customer's date of birth when needed, reducing redundancy in the database.

Composite Attributes: cAddress and cName that are represented by their simple attributes.

Account:



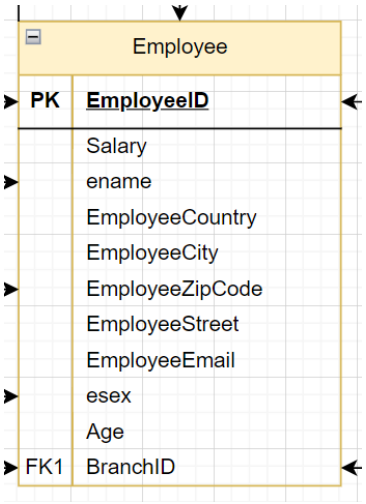
Account: (**AccountID**, AccountType, Balance, DateOpened, #CustomerID (NOT NULL), #BranchID (NOT NULL))

Primary Key (PK): "AccountID" acts as the unique identifier for each account, ensuring the distinctness of account records.

Foreign Key (FK1): "CustomerID" links the account to a specific customer in the "Customer" entity, establishing a relationship between accounts and their respective customers.

Foreign Key (FK2): "BranchID" connects the account to a particular branch in the "Branch" entity, allowing the association of accounts with their branches.

Employee:



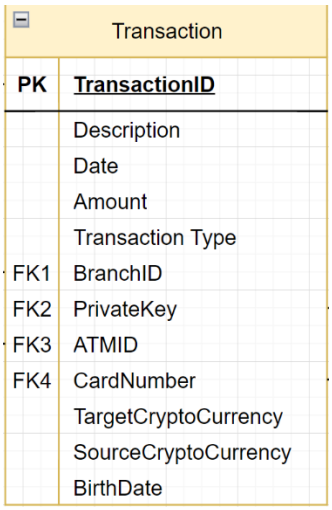
Employee: (**EmployeeID**, Salary, ename, EmployeeCountry, EmployeeCity, EmployeeZipCode,EmployeeStreet, EmployeeEmail, esex, Age, #BranchID (NOT NULL))

Primary Key (PK): "EmployeeID" functions as the unique identifier for each employee, ensuring the distinctiveness of employee records.

Foreign Key (FK1): "BranchID" is used to establish a connection between the employee and a specific branch, facilitating the association of employees with their respective branches.

Composite Attribute: "eAddress" represents a composite attribute, combining attributes such as street, city, country, and state.

Transaction:



Transaction: (**TransactionID**, Description, Date, Amount, TransactionType, TargetCryptocurrency, SourceCryptoCurrency, #BranchID (NOT NULL), #PrivateKey (NOT NULL))

Primary Key (PK): "TransactionID" serves as the unique identifier for each transaction, ensuring the uniqueness of transaction records.

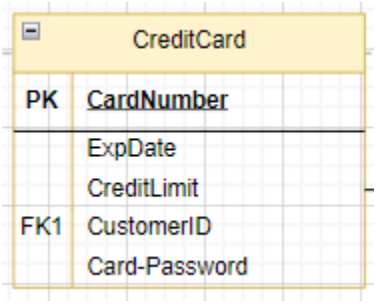
Foreign Key (FK1): "BranchID" establishes a relationship between the transaction and a specific branch, allowing for the association of transactions with their respective branches.

Foreign Key (FK2): "PrivateKey" is sourced from the "cryptoWallet" entity, linking the transaction to a specific wallet's private key, enabling the tracking of transactions associated with a particular wallet.

Foreign Key (FK2): "ATM-ID" is sourced from the "ATM" entity, linking the transaction to a specific ATM, enabling transactions to be done from a specific ATM.

Foreign Key (FK2): "Card Password" is sourced from the "Credit Card" entity, used to access accounts from an ATM to do transactions.

CreditCard:

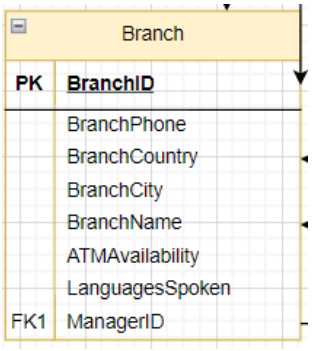


CreditCard: (**CardNumber**, ExpDate, CreditLimit, Card-Password, #CustomerID (NOT NULL))

Primary Key (PK): "CardNumber" acts as the unique identifier for each credit card, ensuring that each credit card has its own specific identity.

Foreign Key (FK): "CustomerID" establishes a connection between the credit card and a particular customer in the "Customer" entity, allowing the credit cards to be linked with their matching customers.

Branch:



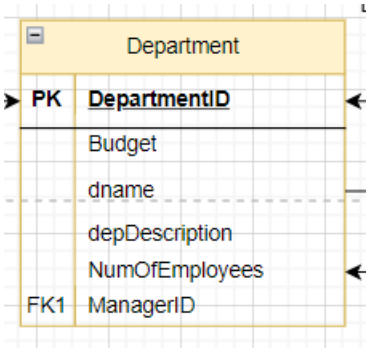
Branch: (**BranchID**, BranchPhone, BranchCountry, BranchCity, BranchName, ATMAvailability, LanguagesSpoken, #ManagerID (NOT NULL))

Primary Key (PK): "BranchID" serves as the unique identifier for each branch, ensuring that each branch has its own specific identity.

Foreign Key (FK1): "ManagerID" establishes a connection to the manager of the branch in the "Employee" entity, enabling the association of branches with their respective managers.

Composite Attribute: "Location" is a composite attribute that has simple attributes “city" and "country" to have the location information of the branch.

Department:

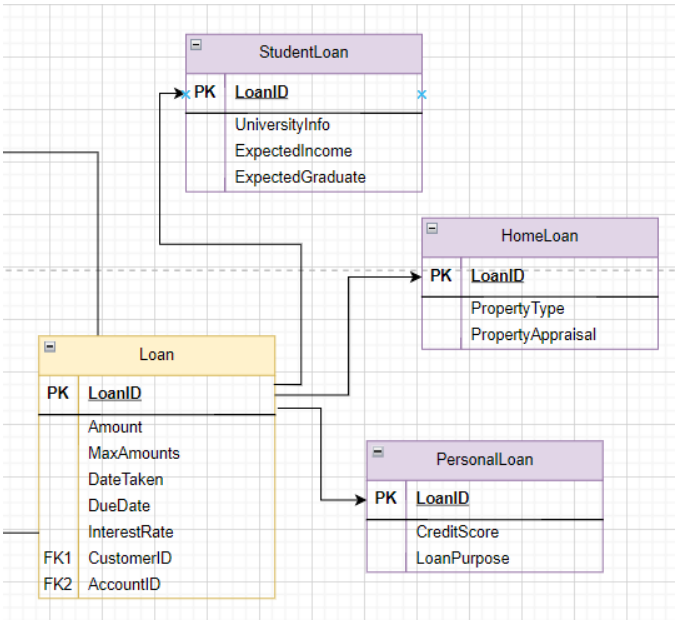


Department: (**DepartmentID**, Budget, dname, depDescription, NumOfEmployees, #ManagerID (NOT NULL))

Primary Key (PK): "DepartmentID" is the unique identifier for each department.

Foreign Key (FK): "ManagerID" establishes a link to the manager of the department in the "Employee" entity, facilitating the association of departments with their respective managers.

Loan:



Loan: (**LoanID**, Amount, MaxAmount, DateTaken, DueDate, InterestRate, #CustomerID (NOT NULL), #AccountID (NOT NULL))

StudenLoan: (**LoanID**, UniversityInfo, ExpectedIncome, ExpectedGradDate)

HomeLoan: (**LoanID**, PropertyType, PropertyAppraisal)

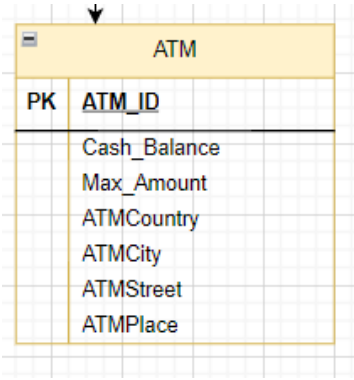
PersonalLoan: (**LoanID**, CreditScore, LoanPurpose)

Primary Key (PK): "LoanID" acts as the unique identifier for each loan.

Foreign Keys (FK): "CustomerID" and "AccountID" link loans to specific customers and accounts in the "Customer" and "Account" entities, respectively, allowing loans to relate to their relevant customers and accounts.

Loan has a ISA relationship with StudentLoan, HomeLoan, and PersonLoan which these entities take the primary key of Loan entity.

ATM:

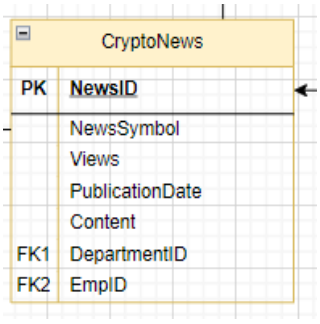


ATM: (**ATM-ID**, Cash-Balance, Max-Amount, ATMCountry, ATMCity, ATMStreet, ATMPLace)

Primary Key (PK): "ATM_ID" serves as the unique identifier for each ATM.

Composite Attribute: "ATMLocation" combines attributes, including "City," "Country," "Street," and "Place," to represent the complete location information of the ATM.

CryptoNews:

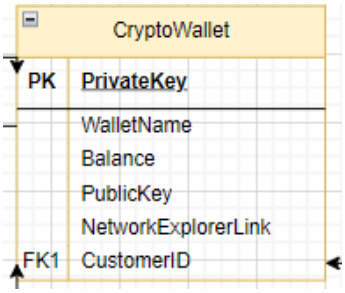


CryptoNews: (**NewsID**, NewsSymbol, Views, PublicationDate, Content)

Primary Key (PK): "NewsID" is the unique identifier for each crypto news article.

Foreign Keys (FK): "DepartmentID" and "EmployeeID" establish connections to the department responsible for the news and the employee who authored it, facilitating the association of news articles with their respective departments.

CryptoWallet:

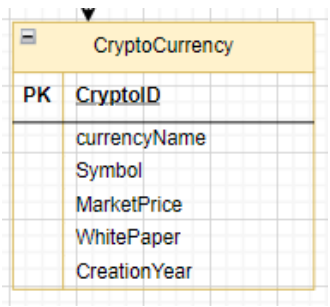


CryptoWallet: (**PrivateKey**, WalletName, Balance, PublicKey, NetworkExplorerLink, #CustomerID (NOT NULL))

Primary Key (PK): "PrivateKey" serves as the unique identifier for each crypto wallet.

Foreign Key (FK): "CustomerID" links each wallet to a specific customer in the "Customer" entity, allowing wallets to be associated with their corresponding customers.

CryptoCurrency:



CryptoCurrency: (**CryptoID**, currencyName, Symbol, MarketPrice, WhitePaper, CreationYear)

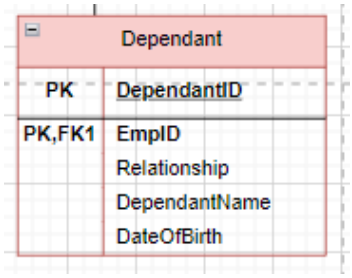
Primary Key (PK): "CryptoID" acts as the unique identifier for each cryptocurrency.

8.2- Weak entities

Converting a weak entity into a relational model involves representing it as tables in a database. A weak entity is one that doesn't have a primary key attribute on its own.

We have the following Strong entities: **Dependent and Beneficiary.**

Dependent (w):

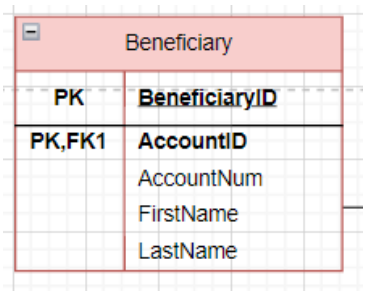


Dependant: (**DependantID**, **#EmpID**, Relationship, DependantName, DateOfBirth)

"DependentID" is a partial key with the primary key “EmployeeID” for each dependent. "EmployeeID" establishes a relationship with the employee in the "Employee" strong entity, allowing the association of dependents with their respective employees.

Derived Attribute: "Age" isn't directly stored in the table.

Beneficiary (w):



Beneficiary: (**BeneficiaryID**, **#AccountID**, AccountNum, FirstName, LastName)

"BeneficiaryID" is a partial key identifier for each beneficiary with the help of the primary key of the Account. "AccountID" links beneficiaries to specific accounts in the "Account" strong entity, facilitating the association of beneficiaries with their respective accounts.

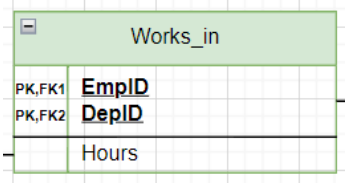
Composite Attribute: "bname" represents a composite attribute, combining attributes such as "Firstname" and "LastName".

8.3- Relationships

Creating a table for the relationship. Then, include foreign keys in the relationship table that reference the primary keys of the connected entities. Any additional attributes of the relationship are represented as columns in the table.

We have the following relationships table: **Works_in, Supervise, Links_with, Contains, Issues, Undergoes, and Refilled.**

Works_in:

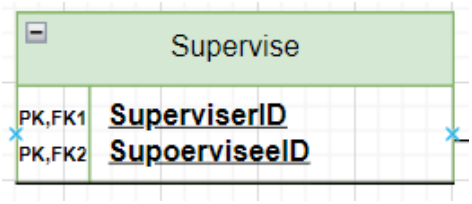


Works-in: (**#EmpID, #DepID**, Hours)

Primary Key (PK): The primary key could consist of attributes like "EmpID" from the "Employee" entity and "DepID" from the "Department" entity, ensuring the uniqueness of each relationship. "EmpID" acts linking to an employee, indicating their role as a employee, while "DepID" functions as another foreign key linking to a department, identifying the department having employees.

Attribute: The "hours" attribute represents the number of hours an employee works in a department.

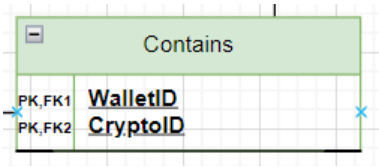
Supervise:



Supervise: (#SuperviserID, #SuperviseeID)

"SuperviserID" is used to link to another employee who serves as the supervisor, while "SuperviseeID" is used to link to an employee who is being supervised. These foreign keys establish relationships within the same "Employee" entity, allowing the association of employees as supervisors and supervisees.

Contains:

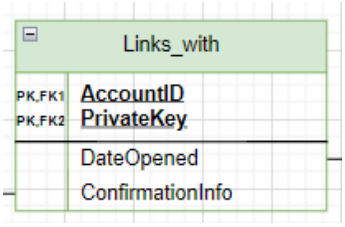


Contains: (#WalletID, #CryptoID)

Primary Key (PK): The primary key is typically formed by combining the "WalletID" attribute from the "CryptoWallet" entity and the "CryptoID" attribute from the "CryptoCurrency" entity, ensuring the uniqueness of each

record within the "Contains" relationship. "WalletID" serves as a foreign key, linking to a specific crypto wallet in the "CryptoWallet" entity, while "CryptoID" acts as another foreign key, connecting to a particular cryptocurrency in the "CryptoCurrency" entity. These foreign keys establish relationships between crypto wallets and the cryptocurrencies they contain, facilitating the association of wallets with their cryptocurrencies.

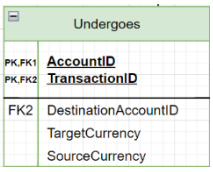
Links_with:



Links_with: (**#AccountID, #PrivateKey**, DateOpened, ConfirmationInfo)

Primary Key (PK): The primary key is typically formed by combining the "AccountID" attribute from the "Account" entity and the "PrivateKey" attribute from the "CryptoWallet" entity. "AccountID" and "CryptoWallet" serve as a foreign key. These foreign keys establish relationships between accounts and the crypto wallets they are linked with, enabling the association of accounts with their linked crypto wallets.

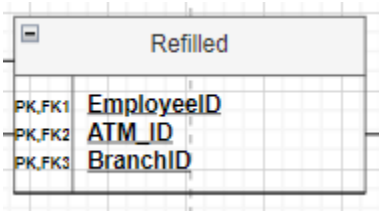
Undergoes:



Undergoes: (**#AccountID, #TransactionID**, #DestinationAccountID, TargetCurrency, SourceCurrency)

Primary Key (PK): The primary key is typically formed by combining the "AccountID" attribute from the "Account" entity and the "TransactionID" attribute from the "Transaction" entity. "DestinationAccountID" serves as a foreign key to specify which account will be the destination of the transactional process. TargetCurrency and SourceCurrency specify the needed CurrencyExchange.

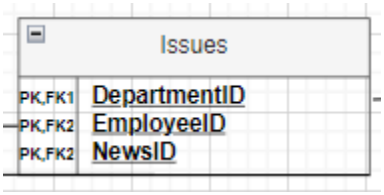
Refilled (Ternary):



Refilled: (**#EmployeeID, #ATM-ID, #BranchID**)

"EmployeeID" links to the employee involved in the transaction, "ATM_ID" connects to the specific ATM used, and "BranchID" relates to the branch where the transaction occurs.

Issues (Ternary):



Issues: (**#DepartmentID, #EmployeeID, #NewsID**)

"EmployeeID" links to the employee involved in the news, "NewsID" connects to the specific news article, and "DepartmentID" relates to the department responsible for the news.

9- Relational Model

Entities:

1. **Customer:** (**CustomerID**, cFirstName, cLastName, cCountry, cCity, cZipcode, cStreet, Email, csex, SSN, cPosition, BirthDate)
2. **Account:** (**AccountID**, AccountType, Balance, DateOpened, #CustomerID (NOT NULL), #BranchID (NOT NULL))

3. **Transaction:** (**TransactionID**, Description, Date, Amount, TransactionType, TargetCryptocurrency, SourceCryptoCurrency, #BranchID (NOT NULL), #PrivateKey (NOT NULL), #ATMID (NOT NULL), #cardPassword(NOT NULL))
4. **CreditCard:** (**CardNumber**, ExpDate, CreditLimit, Card-Password, #CustomerID (NOT NULL))
5. **Branch:** (**BranchID**, BranchPhone, BranchCountry, BranchCity, BranchName, ATMAvailability, LanguagesSpoken, #ManagerID (NOT NULL))
6. **Department:** (**DepartmentID**, Budget, dname, depDescription, NumOfEmployees, #ManagerID (NOT NULL))
7. **Employee:** (**EmployeeID**, Salary, ename, EmployeeCountry, EmployeeCity, EmployeeZipCode,EmployeeStreet, EmployeeEmail, esex, Age, #BranchID (NOT NULL))
8. **CryptoNews:** (**NewsID**, NewsSymbol, Views, PublicationDate, Content)
9. **CryptoWallet:** (**PrivateKey**, WalletName, Balance, PublicKey, NetworkExplorerLink, #CustomerID (NOT NULL))
10. **CryptoCurrency:** (**CryptoID**, currencyName, Symbol, MarketPrice, WhitePaper, CreationYear)
11. **Loan:** (**LoanID**, Amount, MaxAmount, DateTaken, DueDate, InterestRate, #CustomerID (NOT NULL), #AccountID (NOT NULL))
12. **StudenLoan:** (**LoanID**, UniversityInfo, ExpectedIncome, ExpectedGradDate)
13. **HomeLoan:** (**LoanID**, PropertyType, PropertyAppraisal)
14. **PersonalLoan:** (**LoanID**, CreditScore, LoanPurpose)
15. **ATM:** (**ATM-ID**, Cash-Balance, Max-Amount, ATMCountry, ATMCity, ATMStreet, ATMPLace)

Weak Entities:

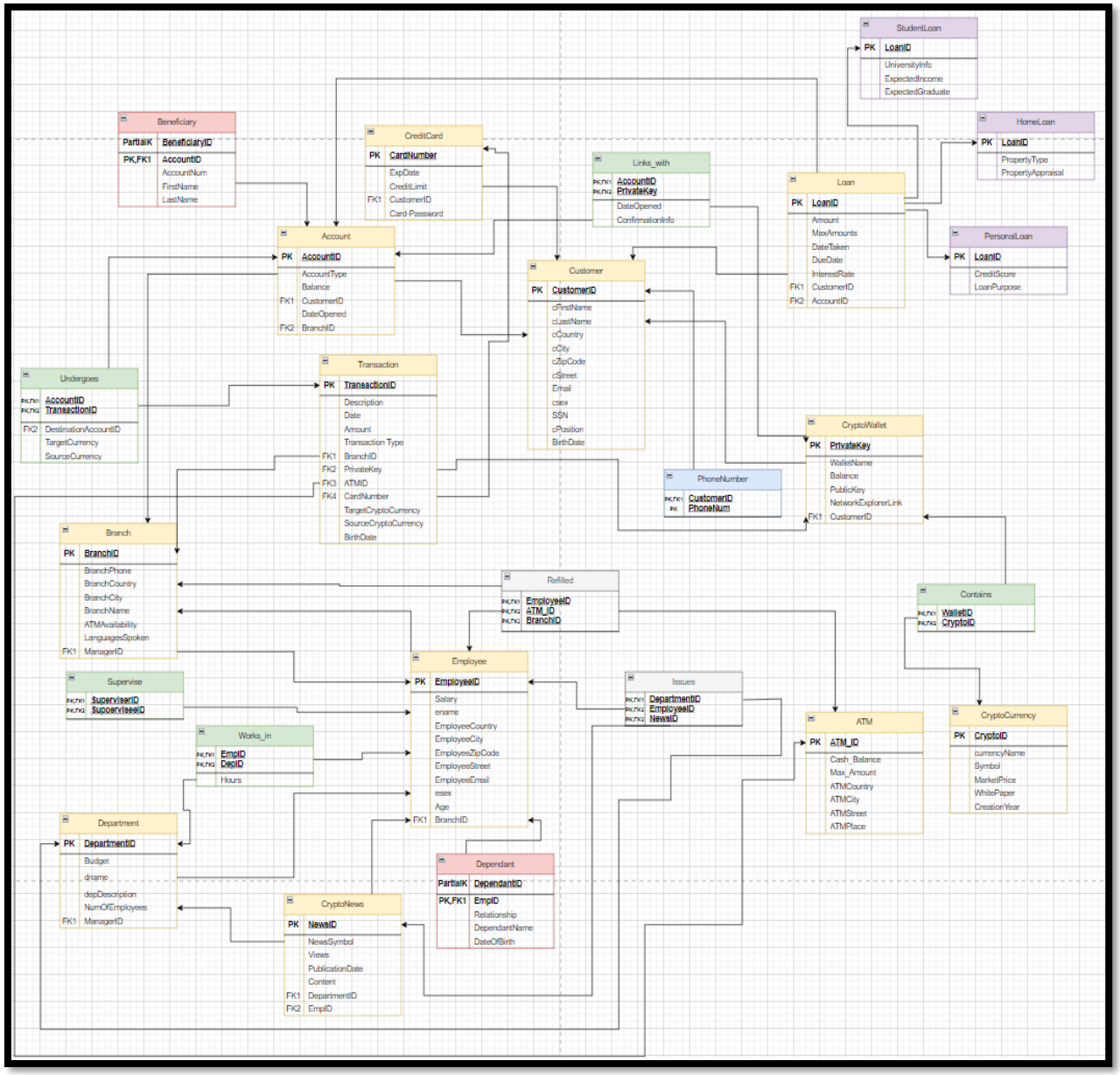
1. **Dependant:** (**DependantID**, **#EmpID**, Relationship, DependantName, DateOfBirth)
2. **Beneficiary:** (**BeneficiaryID**, #**AccountID**, AccountNum, FirstName, LastName)

Relationships:

1. **Links_with:** (**#AccountID, #PrivateKey**, DateOpened, ConfirmationInfo)
2. **Contains:** (**#WalletID, #CryptoID**)
3. **Supervise:** (**#SuperviserID, #SuperviseeID**)
4. **Works-in:** (**#EmpID, #DepID**, Hours)
5. **Refilled:** (**#EmployeeID, #ATM-ID, #BranchID**)
6. **Issues:** (**#DepartmentID, #EmployeeID, #NewsID**)
7. **Undergoes:** (**#AccountID, #TransactionID**, #DestinationAccountID, TargetCurrency, SourceCurrency)

Multivalued Attributes:

1- **Phone Number**: (**#CustomerID, PhoneNum**)


StudentLoan
PK LoanID
UniversityInfo
ExpectedIncome
ExpectedGraduate
Beneficiary
PartialK BeneficiaryID
PK,FK1 AccountID
AccountNum
FirstName
LastName
CreditCard
PK CardNumber
ExpDate
CreditLimit
FK1 CustomerID
Card-Password
Links_with
PK,FK1 AccountID
PK,FK2 PrivateKey
DateOpened
ConfirmationInfo
HomeLoan
PK LoanID
PropertyType
PropertyAppraisal
Loan
PK LoanID
Amount
MaxAmounts
DateTaken
DueDate
InterestRate
FK1 CustomerID
FK2 AccountID
PersonalLoan
PK LoanID
CreditScore
LoanPurpose
Account
PK AccountID
AccountType
Balance
FK1 CustomerID
DateOpened
FK2 BranchID
Customer
PK CustomerID
cFirstName
cLastName
cCountry
cCity
cZipCode
cStreet
Email
csex
SSN
cPosition
BirthDate
Undergoes
PK,FK1 AccountID
PK,FK2 TransactionID
FK2 DestinationAccountID
TargetCurrency
SourceCurrency
Transaction
PK TransactionID
Description
Date
Amount
Transaction Type
FK1 BranchID
FK2 PrivateKey
FK3 ATMID
FK4 CardNumber
TargetCryptoCurrency
SourceCryptoCurrency
BirthDate
CryptoWallet
PK PrivateKey
WalletName
Balance
PublicKey
NetworkExplorerLink
FK1 CustomerID
PhoneNumber
PK,FK1 CustomerID
PK PhoneNum
Branch
PK BranchID
BranchPhone
BranchCountry
BranchCity
BranchName
ATMAvailability
LanguagesSpoken
FK1 ManagerID
Refilled
PK,FK1 EmployeeID
PK,FK2 ATM_ID
PK,FK3 BranchID
Contains
PK,FK1 WalletID
PK,FK2 CryptoID
Employee
PK EmployeeID
Salary
ename
EmployeeCountry
EmployeeCity
EmployeeZipCode
EmployeeStreet
EmployeeEmail
esex
Age
FK1 BranchID
Supervise
PK,FK1 SupervisorID
PK,FK2 SuperviseeID
Issues
PK,FK1 DepartmentID
PK,FK2 EmployeeID
PK,FK2 NewsID
ATM
PK ATM_ID
Cash_Balance
Max_Amount
ATMCountry
ATMCity
ATMStreet
ATMPlace
CryptoCurrency
PK CryptoID
currencyName
Symbol
MarketPrice
WhitePaper
CreationYear
Works_in
PK,FK1 EmpID
PK,FK2 DeptID
Hours
Department
PK DepartmentID
Budget
dname
depDescription
NumOfEmployees
FK1 ManagerID
Dependant
PartialK DependantID
PK,FK1 EmpID
Relationship
DependantName
DateOfBirth
CryptoNews
PK NewsID
NewsSymbol
Views
PublicationDate
Content
FK1 DepartmentID
FK2 EmpID

Honest Bank Code

Database:

Create Database The_Honest_Bank;

Strong Entities:

Customer:

```
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY,
    cFirstName VARCHAR(30),
    cLastName VARCHAR(30),
    cCountry VARCHAR(30),
    cCity VARCHAR(30),
    cZipcode VARCHAR(10),
    cStreet VARCHAR(255),
    Email VARCHAR(255),
    csex VARCHAR(10),
    SSN VARCHAR(20) UNIQUE,
    cPosition VARCHAR(255),
    BirthDate DATE,
    CHECK (csex IN ('Male', 'Female')),
    CHECK (BirthDate <= CURDATE())
);
```

Customer Tuples:

CustomerID	cFirstName	cLastName	cCountry	cCity	cZipcode	cStreet	Email	csex	SSN	cPosition	BirthDate
1	John	Johnson	USA	New York	10001	123 Main St	john.johnson@email.com	Male	123-85-6789	Manager	5/15/1980
2	Jane	Smith	Canada	Toronto	M5V 2L7	456 Maple Ave	jane.smith@email.com	Female	988-65-4321	Developer	8/22/1985
3	Khaleel	Johnson	UK	London	SW1A 1AA	789 Oak Blvd	khaleel.johnson@email.com	Male	345-67-8501	Analyst	2/10/1990
4	Emily	Davis	Australia	Sydney	2000	101 Pine St	emily.davis@email.com	Female	569-89-0123	Designer	11/28/1988
5	Abdulrahman	Brown	Germany	Berlin	10117	202 Cedar Rd	abdulrahman.brown@email.com	Male	789-08-2345	Engineer	4/3/1983
6	Sophia	Garcia	Spain	Madrid	28001	303 Elm Ln	sophia.garcia@email.com	Female	812-34-5678	Consultant	7/19/1995

7	Daniel	Martinez	France	Paris	75001	404 Birch St	daniel.martinez@email.com	Male	23-56-7890	Supervisor	9/14/1982
8	Olivia	Rodriguez	Italy	Rome	00185	505 Oak Ave	olivia.rodriguez@email.com	Female	416-78-9012	Manager	1/26/1987
9	William	Lopez	Brazil	Rio de Janeiro	20040-080	606 Maple Blvd	william.lopez@email.com	Male	678-95-1234	Developer	12/7/1992
10	Abdulrahman	Rodriguez	India	Mumbai	400001	707 Pine Rd	abdulrahman.hernandez@email.com	Male	890-12-3256	Analyst	6/30/1986
11	John	Smith	USA	Los Angeles	90001	111 Oak St	john.smith@email.com	Male	111-22-3344	Manager	9/8/1975
12	Kevin	Williams	Canada	Vancouver	V6C 1A1	222 Maple Ave	kevin.williams@email.com	Male	444-65-6677	Developer	12/18/1988
13	Kevin	Williams	UK	Manchester	M1 1AB	333 Pine Blvd	kevin.williams1@email.com	Male	999-88-6777	Analyst	3/22/1993
14	Sophie	Anderson	Australia	Melbourne	3000	444 Cedar Ln	sophie.anderson@email.com	Female	122-45-6689	Designer	7/15/1982
15	Liam	Brown	Germany	Hamburg	20095	555 Elm St	liam.brown@email.com	Male	234-56-7810	Engineer	11/30/1989
16	Kevin	Williams	Spain	Barcelona	08001	666 Oak Ave	zoe.martinez@email.com	Male	769-01-2345	Consultant	4/25/1996
17	Alain	Garcia	France	Nice	06000	777 Maple Rd	mason.garcia@email.com	Male	345-67-8701	Supervisor	1/12/1984
18	Alain	Rodriguez	Italy	Milan	20121	888 Pine Blvd	alain.rodriguez@email.com	Male	567-89-0823	Manager	8/7/1980
19	Logan	Lopez	Brazil	Sao Paulo	01000-000	999 Cedar Ln	logan.lopez@email.com	Male	012-94-5678	Developer	6/19/1991
20	Alain	Hernandez	India	Delhi	110001	1010 Oak St	madison.hernandez@email.com	Male	890-22-3456	Analyst	2/3/1987
21	Caleb	Taylor	USA	Chicago	60601	111 Maple Ave	caleb.taylor@email.com	Male	345-67-8981	Manager	4/14/1978
22	Scarlett	Brown	Canada	Montreal	H3C 1A1	222 Pine Blvd	scarlett.miller@email.com	Female	567-99-0123	Developer	9/26/1989
23	Jackson	Harris	UK	Birmingham	B1 1AB	333 Cedar Ln	jackson.harris@email.com	Male	789-01-2345	Analyst	12/8/1994
24	Jackson	Harris	Australia	Sydney	2000	444 Oak Ave	chloe.king@email.com	Female	012-34-5668	Designer	3/17/1981
25	Elijah	Young	Germany	Munich	80331	555 Maple Rd	elijah.young@email.com	Male	134-56-7790	Engineer	7/2/1986
26	Grace	Fisher	Spain	Seville	41001	666 Cedar Ln	grace.fisher@email.com	Female	348-67-8901	Consultant	10/15/1992
27	Kevin	Ward	France	Marseille	13001	777 Oak Ave	lucas.ward@email.com	Male	567-89-0223	Supervisor	1/28/1987
28	Aubrey	Reed	Italy	Florence	50123	888 Pine Blvd	aubrey.reed@email.com	Female	779-01-2345	Manager	6/9/1984
29	Kevin	Williams	Brazil	Brasilia	70000-000	999 Cedar Ln	kevin.williams2@email.com	Male	012-34-5658	Developer	9/22/1990
30	Lily	Baker	India	Chennai	600001	1010 Maple Rd	lily.baker@email.com	Female	234-56-7990	Analyst	12/3/1983

The "Customer" table serves as a repository for customer information, encompassing details like names, addresses, email, gender, social security numbers, positions, and birthdates.

- **Primary Key Constraint:**

a) The **CustomerID** column serves as the primary key, ensuring a unique identifier for each Customer.

- **Check Constraints:**

a) The **csex** column employs a **CHECK** constraint, ensuring that only 'Male' or 'Female' values are accepted. This guarantees that gender representation adheres to predefined options.

b) The **BirthDate** column features a **CHECK** constraint to ensure birthdates are not in the future. This maintains accuracy in the stored data.

- **Unique Constraint:**

a) The **SSN** (Social Security Number) column is marked as **UNIQUE**. This constraint prevents the duplication of social security numbers across different customer records, maintaining uniqueness and avoiding conflicts.

Branch:

```
CREATE TABLE Branch (
    BranchID INT PRIMARY KEY,
    BranchPhone VARCHAR(25),
    BranchCountry VARCHAR(20),
    BranchCity VARCHAR(20),
    BranchName VARCHAR(255),
    ATMAvailability BOOLEAN,
    LanguagesSpoken VARCHAR(255),
    ManagerID INT,
    FOREIGN KEY (ManagerID) REFERENCES Employee(EmployeeID),
    CHECK (BranchCountry IN ('USA', 'Lebanon', 'UK')),
    CHECK (BranchCity IN ('New York', 'Los Angeles', 'Beirut', 'Byblos', 'London','Chicago',
            'Birmingham','Tripoli','Tyre','Virginia'))
);
```

BranchID	BranchPhone	BranchCountry	BranchCity	BranchName	ATMAvailability	LanguagesSpoken	ManagerID
1	+1-123-456-7890	USA	New York	Main Street Branch	1	English, Spanish	1
2	+1-987-654-3210	USA	Los Angeles	Sunset Boulevard Branch	0	English, Chinese	7
3	+961-71-234567	Lebanon	Beirut	Downtown Branch	1	Arabic, French, English	4
4	+961-76-543210	Lebanon	Byblos	Seaside Branch	0	Arabic, English	9
5	+44-20-1234-5678	UK	London	City Centre Branch	1	English	10
6	+1-555-123-4567	USA	Chicago	Downtown Chicago Branch	1	English, Spanish	14
7	+1-333-987-6543	USA	Virginia	Harrisonbourgh	0	English, French	12

8	+44-20-5678-1234	UK	Birmingham	Midlands Branch	1	English	5
9	+961-70-987654	Lebanon	Tripoli	Northern Lebanon Branch	0	Arabic, English	8
10	+961-79-123456	Lebanon	Tyre	Southern Lebanon Branch	1	Arabic, English	15

The "Branch" table is designed to store information about different branches, including their identifiers, contact details, geographical location, branch name, ATM availability, languages spoken, and the manager's identifier.

- **Primary Key Constraint:**
    a) The **BranchID** column serves as the primary key, ensuring a unique identifier for each branch.
- **Foreign Key Constraint:**
    a) The **ManagerID** column has a **NOT NULL** constraint, requiring each branch to have a manager. Additionally, it features a foreign key constraint referencing the **Employee** table's **EmployeeID**. This ensures that the **ManagerID** in the "Branch" table corresponds to a valid employee in the "Employee" table, establishing a relationship between branches and employees.
- **Check Constraints:**
    a) The **BranchCountry** column has a **CHECK** constraint, restricting values to 'USA', 'Lebanon', 'Canada' or 'UK'. This ensures that branches are associated with valid countries.
    b) The **BranchCity** column has a **CHECK** constraint, limiting values to specific cities ('New York', 'Los Angeles', 'Beirut', 'Byblos', 'London','Chicago', 'Birmingham', 'Tripoli', 'Tyre', 'Virginia'). This maintains consistency in the recorded cities for branches.
- **Not Null Constraint:**
    a) The **ManagerID** column is marked as **NOT NULL**, ensuring that every branch has an assigned manager.

Employee:

```sql
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    Salary DECIMAL(10, 2),
    ename VARCHAR(255),
    EmployeeCountry VARCHAR(20),
    EmployeeCity VARCHAR(20),
    EmployeeZipCode VARCHAR(10),
    EmployeeStreet VARCHAR(255),
    EmployeeEmail VARCHAR(255),
    esex VARCHAR(10),
    Age INT CHECK (Age >= 18),
    BranchID INT not null,
    FOREIGN KEY (BranchID) REFERENCES Branch(BranchID),
    CHECK (Salary >= 0),
    CHECK (esex IN ('Male', 'Female'))
);
```

Employee Tuples:

EmployeeID	Salary	ename	EmployeeCountry	EmployeeCity	EmployeeZipCode	EmployeeStreet	E
1	80000	Robert Johnson	USA	New York	10001	456 Oak St	r
2	75000	Sarah White	UK	London	SW1A 1AA	789 Pine Ave	s
3	90000	Chris Smith	USA	Los Angeles	90001	101 Maple Rd	c
4	82000	Emily Davis	Lebanon	Beirut	12345	303 Cedar Ln	e
5	70000	Daniel Martinez	Lebanon	Byblos	54321	202 Birch Blvd	d
6	85000	Ava Hernandez	USA	New York	10002	404 Elm St	a
7	88000	Michael Brown	USA	Los Angeles	90002	606 Pine Ave	r
8	95000	Saad Hamdoun	Lebanon	Beirut	54322	707 Cedar Rd	s
9	78000	William Lopez	Lebanon	Byblos	12346	808 Maple Blvd	v
10	92000	Sophia Garcia	UK	London	SW1A 1AB	909 Oak Ln	s
11	80000	Jason Turner	USA	Chicago	60601	111 Oak St	j
12	75000	Mia Cooper	USA	Virginia	M5V 2L7	222 Maple Ave	r
13	90000	Ryan Walker	UK	Manchester	M1 1AB	333 Pine Blvd	r
14	82000	Emma Robinson	Australia	Melbourne	3000	444 Cedar Ln	e
15	70000	James Reed	Germany	Hamburg	20095	555 Elm St	j
16	85000	Sophie Turner	Spain	Barcelona	08001	666 Oak Ave	s
17	88000	David Hall	France	Nice	06000	777 Maple Rd	d
18	95000	Chloe Evans	Italy	Milan	20121	888 Pine Blvd	c
19	78000	Daniel Parker	Brazil	Sao Paulo	01000-000	999 Cedar Ln	d
20	92000	Aria Hughes	India	Delhi	110001	1010 Oak St	a
21	80000	Elijah Taylor	USA	Los Angeles	90001	456 Oak St	e
22	75000	Lily Adams	UK	London	SW1A 1AA	789 Pine Ave	li
23	90000	Mason Miller	USA	Chicago	60601	101 Maple Rd	n
24	82000	Zoe Parker	Lebanon	Beirut	12345	303 Cedar Ln	z
25	70000	Aiden Scott	Lebanon	Byblos	54321	202 Birch Blvd	a

26	85000	Scarlett Young	USA	New York	10002	404 Elm St	s
27	88000	Owen Adams	USA	Los Angeles	90002	606 Pine Ave	c
28	95000	Ava Turner	Lebanon	Beirut	54322	707 Cedar Rd	a
29	78000	Logan Hall	Lebanon	Byblos	12346	808 Maple Blvd	l
30	92000	Emma Walker	UK	London	SW1A 1AB	909 Oak Ln	e

The "Employee" table is designed to store information about employees, encompassing details such as employee ID, salary, name, address, email, gender, age, and the branch to which they are assigned.

- **Primary Key Constraint:**
    a) The **EmployeeID** column serves as the primary key, ensuring a unique identifier for each employee.
- **Foreign Key Constraint:**
    a) The **BranchID** column has a **NOT NULL** constraint, indicating that each employee must be associated with a specific branch. Additionally, it features a foreign key constraint referencing the **Branch** table's **BranchID**. This ensures that the **BranchID** in the "Employee" table corresponds to a valid branch in the "Branch" table, establishing a relationship between employees and branches.
- **Check Constraints:**
    a) The **Age** column has a **CHECK** constraint, ensuring that the age of an employee is 18 years or older. This helps maintain age eligibility for employment.
    b) The **Salary** column has a **CHECK** constraint, ensuring that the salary is greater than or equal to 0. This prevents negative salary values, ensuring financial consistency.
    c) The **esex** column has a **CHECK** constraint, allowing only 'Male' or 'Female' values. This enforces gender representation consistency.
- **Not Null Constraint:**
    a) The **BranchID** column is marked as **NOT NULL**, ensuring that every employee is associated with a specific branch.

Account:

```
CREATE TABLE Account (
    AccountID INT PRIMARY KEY,
    AccountType VARCHAR(20),
    Balance DECIMAL(10, 2),
    DateOpened DATE,
    CustomerID INT not null,
    BranchID INT not null,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
    FOREIGN KEY (BranchID) REFERENCES Branch(BranchID),
    CHECK (AccountType IN ('Savings', 'Checking', 'Investment', 'Credit')),
    CHECK (Balance >= 0)
);
```

Account Tuples:

AccountID	AccountType	Balance	DateOpened	CustomerID	BranchID
1	Savings	5000	2020-01-15	1	1
2	Checking	2500	2019-05-20	2	2
3	Investment	10000	2022-02-10	3	3
4	Credit	-500	2021-08-05	4	4
5	Savings	7500	2023-01-30	5	5
6	Checking	3000	2020-12-12	6	1
7	Investment	12000	2021-10-18	7	2
8	Credit	-200	2018-07-25	8	3
9	Savings	6000	2019-11-08	9	4
10	Checking	4000	2022-05-03	10	5
11	Savings	5500	2021-02-20	11	6
12	Checking	3200	2020-08-15	12	7
13	Investment	15000	2023-03-05	13	8
14	Credit	-700	2022-02-12	14	9
15	Savings	8000	2022-11-28	15	10
16	Checking	3500	2021-07-01	16	1
17	Investment	18000	2020-12-10	17	2
18	Credit	-300	2019-05-22	18	3
19	Savings	7000	2020-03-18	19	4
20	Checking	4500	2023-06-09	20	5
21	Savings	5800	2022-04-14	21	6
22	Checking	2800	2021-10-05	22	7
23	Investment	13000	2020-07-22	23	8
24	Credit	-600	2022-09-15	24	9
25	Savings	8500	2019-12-01	25	10

26	Checking	3200	2019-06-28	26	1
27	Investment	16000	2022-01-12	27	2
28	Credit	-250	2018-04-05	28	3
29	Savings	6700	2021-11-08	29	4
30	Checking	5000	2023-05-03	30	5
31	Savings	6000	2019-10-15	1	6
32	Checking	3800	2018-04-10	2	7
33	Investment	12000	2021-02-28	3	8
34	Credit	-400	2019-09-01	4	9
35	Savings	7200	2020-06-15	5	10

The "Account" table is designed to store information about customer accounts, including account ID, type, balance, date opened, and the associated customer and branch.

- **Primary Key Constraint:**
    a) The **AccountID** column serves as the primary key, ensuring a unique identifier for each account.
- **Foreign Key Constraints:**
    a) The **CustomerID** column has a **NOT NULL** constraint, indicating that each account must be associated with a specific customer. It features a foreign key constraint referencing the **Customer** table's **CustomerID**, ensuring that the **CustomerID** in the "Account" table corresponds to a valid customer in the "Customer" table.
    b) The **BranchID** column also has a **NOT NULL** constraint, ensuring that each account is associated with a specific branch. It features a foreign key constraint referencing the **Branch** table's **BranchID**, ensuring that the **BranchID** in the "Account" table corresponds to a valid branch in the "Branch" table.
- **Check Constraints:**
    a) The **AccountType** column has a **CHECK** constraint, allowing only specific values ('Savings', 'Checking', 'Investment', 'Credit'). This ensures consistency in account types.
    b) The **Balance** column has a **CHECK** constraint, ensuring that the balance is greater than or equal to 0. This prevents negative balance values, maintaining financial consistency.
- **Not Null Constraints:**
    a) The **CustomerID** and **BranchID** columns are marked as **NOT NULL**, ensuring that each account is associated with a specific customer and branch.

Transaction:

```
CREATE TABLE Transaction (
    TransactionID INT PRIMARY KEY,
    PrivateKey INT,
    Description VARCHAR(255),
    Date DATE,
    Amount DECIMAL(10, 2),
    TransactionType VARCHAR(20),
    TargetCryptocurrency VARCHAR(20),
    SourceCryptoCurrency VARCHAR(20),
    BranchID INT not null,
    CardPassword VARCHAR(20),
    FOREIGN KEY (CardPassword) REFERENCES CreditCard(CardPassword),
    FOREIGN KEY (ATMID) REFERENCES ATM(ATMID),
    FOREIGN KEY (BranchID) REFERENCES Branch(BranchID),
    FOREIGN KEY (PrivateKey) REFERENCES CryptoWallet(PrivateKey),
    CHECK (Amount >= 0)
);
```

The "Transaction" table is designed to record various financial transactions, including details such as transaction ID, description, date, amount, transaction type, cryptocurrencies involved, and the associated branch, credit card, ATM, and crypto wallet.

- **Primary Key Constraint:**
    a) The **TransactionID** column serves as the primary key, ensuring a unique identifier for each transaction.
- **Foreign Key Constraints:**
    a) The **CardPassword** column has a **NOT NULL** constraint, indicating that each transaction may be associated with a credit card. It features a foreign key constraint referencing the **CreditCard** table's **CardPassword**, ensuring that the **CardPassword** in the "Transaction" table corresponds to a valid credit card in the "CreditCard" table.
    b) The **BranchID** column also has a **NOT NULL** constraint, indicating the branch associated with each transaction. It features a foreign key constraint referencing the **Branch** table's **BranchID**, ensuring that the **BranchID** in the "Transaction" table corresponds to a valid branch in the "Branch" table.
- **Check Constraints:**
    - The **Amount** column has a **CHECK** constraint, ensuring that the amount is greater than or equal to 0. This prevents negative transaction amounts, maintaining financial consistency.
- **Not Null Constraints:**
    - The **BranchID** column is marked as **NOT NULL**, ensuring that each transaction is associated with a specific branch.

CreditCard:

```
CREATE TABLE CreditCard (
    CardNumber VARCHAR(16) PRIMARY KEY,
    ExpDate DATE,
    CreditLimit DECIMAL(10, 2),
    CardPassword VARCHAR(20),
    CustomerID INT not null,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
    CHECK (LENGTH(CardNumber) = 16),
    CHECK (ExpDate >= CURDATE()),
    CHECK (CreditLimit >= 0)
);
```

CreditCard Tuples:

CardNumber	ExpDate	CreditLimit	CardPassword	CustomerID
1111222233334444	5/1/2023	5000	1122	1
1234567812345670	7/1/2024	3500	1343	10
2222333344445555	8/1/2024	8000	3344	2
2345678923456781	4/1/2023	4800	5342	11
3333444455556666	12/1/2023	3000	7788	3
3456789034567892	2/1/2024	6200	5233	12
4444555566667777	10/1/2024	6000	9900	4
4567890145678903	8/1/2023	2800	7765	13
5555666677778888	6/1/2023	7000	5566	5
5678901256789014	1/1/2024	4200	7875	14
6666777788889999	3/1/2024	9000	1234	6
6789012367890125	7/1/2023	5800	9879	15
7777888899990000	9/1/2023	4000	5678	7
7890123478901236	9/1/2024	3200	9865	16
8888999900001111	5/1/2024	5500	9101	8
8901234589012347	10/1/2023	4900	8567	17
9999000011112222	11/1/2023	7500	5323	9

The "CreditCard" table is designed to store information about credit cards, including details such as card number, expiration date, credit limit, card password, and the associated customer.

- **Primary Key Constraint:**
    a) The **CardNumber** column serves as the primary key, ensuring a unique identifier for each credit card.
- **Foreign Key Constraint:**

a) The **CustomerID** column has a **NOT NULL** constraint, indicating that each credit card must be associated with a specific customer. It features a foreign key constraint referencing the **Customer** table's **CustomerID**, ensuring that the **CustomerID** in the "CreditCard" table corresponds to a valid customer in the "Customer" table.

- **Check Constraints:**

  a) The **LENGTH(CardNumber) = 16** constraint ensures that the card number has exactly 16 characters. This maintains consistency in the length of credit card numbers.

  b) The **ExpDate >= CURDATE()** constraint ensures that the expiration date is not in the past, maintaining accuracy.

  c) The **CreditLimit >= 0** constraint ensures that the credit limit is greater than or equal to 0, preventing negative credit limits and ensuring financial consistency.

- **Not Null Constraints:**

  a) The **CustomerID** column is marked as **NOT NULL**, ensuring that each credit card is associated with a specific customer.

Department:

```
CREATE TABLE Department (
    DepartmentID INT PRIMARY KEY,
    Budget DECIMAL(15, 2),
    dname VARCHAR(30),
    depDescription TEXT,
    NumOfEmployees INT,
    ManagerID INT not null,
    FOREIGN KEY (ManagerID) REFERENCES Employee(EmployeeID),
    CHECK (Budget >= 0),
    CHECK (NumOfEmployees >= 0)
);
```

Department Tuples:

DepartmentID	Budget	dname	depDescription	NumOfEmployees	ManagerID
1	800000	Corporate Banking	Provides financial services to businesses.	5	2
2	1500000	Investment Banking	Handles investment and financial transactions.	5	4
3	900000	Information Technology	Manages the bank's information systems.	5	1
4	600000	Customer Support	Provides assistance and support to bank customers.	5	7

5	450000	Branch Operations	Manages day-to-day operations of bank branches.	5	10

The "Department" table is designed to store information about different departments within an organization, including details such as department ID, budget, department name, description, number of employees, and the manager overseeing the department.

- **Primary Key Constraint:**
    a) The **DepartmentID** column serves as the primary key, ensuring a unique identifier for each department.
- **Foreign Key Constraint:**
    a) The **ManagerID** column has a **NOT NULL** constraint, indicating that each department must have a manager. It features a foreign key constraint referencing the **Employee** table's **EmployeeID**, ensuring that the **ManagerID** in the "Department" table corresponds to a valid employee in the "Employee" table.
- **Check Constraints:**
    a) The **Budget >= 0** constraint ensures that the budget for each department is non-negative, maintaining financial consistency.
    b) The **NumOfEmployees >= 0** constraint ensures that the number of employees in each department is non-negative, maintaining consistency in employee count.
- **Not Null Constraints:**
    a) The **ManagerID** column is marked as **NOT NULL**, ensuring that each department has a manager.

CryptoNews:

```
CREATE TABLE CryptoNews (
    NewsID INT PRIMARY KEY,
    NewsSymbol VARCHAR(10),
    Views INT,
    PublicationDate DATE,
    Content TEXT
);
```

CryptoNews Tuples:

NewsID	NewsSymbol	Views	PublicationDate	Content

1	BTC	5000	1/15/2023	Bitcoin reaches new all-time high.
2	ETH	3500	2/10/2023	Ethereum upgrades its consensus mechanism.
3	BNB	2000	3/5/2023	Binance Coin becomes a major player in DeFi.
4	ADA	8000	4/20/2023	Cardano introduces smart contract capabilities.
5	SOL	4500	5/15/2023	Solana gains popularity for its fast transactions.
6	XRP	6000	6/1/2023	XRP used for cross-border payments on a large scale.
7	DOT	3000	7/10/2023	Polkadot implements parachains for better scalability.
8	DOGE	7000	8/25/2023	Dogecoin community continues to grow.
9	AVAX	2500	9/18/2023	Avalanche introduces new features for developers.
10	LTC	4000	10/5/2023	Litecoin celebrates its 10th anniversary.

The "CryptoNews" table is designed to store information about news articles related to cryptocurrencies, including details such as news ID, cryptocurrency symbol, number of views, publication date, and content.

- **Primary Key Constraint:**

    a) The **NewsID** column serves as the primary key, ensuring a unique identifier for each crypto news article.

CryptoWallet:

```
CREATE TABLE CryptoWallet (
    PrivateKey INT PRIMARY KEY,
    WalletName VARCHAR(30),
    Balance DECIMAL(10, 2),
    PublicKey VARCHAR(30),
    NetworkExplorerLink VARCHAR(255),
    CustomerID INT not null,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
    CHECK (Balance >= 0)
);
```

CryptoWallet Tuples:

PrivateKey	WalletName	Balance	PublicKey	NetworkExplorerLink	CustomerID
101	Main Wallet	5	pubkey1	https://explorer.link/pubkey1	1
102	Savings Wallet	8.2	pubkey2	https://explorer.link/pubkey2	2
103	Trading Wallet	12.5	pubkey3	https://explorer.link/pubkey3	3
104	Investment Wallet	3.7	pubkey4	https://explorer.link/pubkey4	10
105	Crypto Ventures	6.8	pubkey5	https://explorer.link/pubkey5	9

106	Daily Transactions	9.3	pubkey6	https://explorer.link/pubkey6	2
107	Personal Savings	2.5	pubkey7	https://explorer.link/pubkey7	7
108	Emergency Fund	7.1	pubkey8	https://explorer.link/pubkey8	2
109	Retirement Savings	11	pubkey9	https://explorer.link/pubkey9	9
110	Long-Term Investments	4.5	pubkey10	https://explorer.link/pubkey10	10

The "CryptoWallet" table is designed to store information about cryptocurrency wallets, including details such as private key, wallet name, balance, public key, network explorer link, and the associated customer.

- **Primary Key Constraint:**
    a) The **PrivateKey** column serves as the primary key, ensuring a unique identifier for each crypto wallet.
- **Foreign Key Constraint:**
    a) The **CustomerID** column has a **NOT NULL** constraint, indicating that each crypto wallet must be associated with a specific customer. It features a foreign key constraint referencing the **Customer** table's **CustomerID**, ensuring that the **CustomerID** in the "CryptoWallet" table corresponds to a valid customer in the "Customer" table.
- Check Constraint (Balance):
    a) The Balance column has a check constraint (CHECK (Balance >= 0)) to ensure that the wallet balance is always greater than or equal to zero. This constraint helps maintain data integrity by preventing the insertion of negative balances, which might be invalid in the context of a wallet balance.
- Not Null Constraint (CustomerID):
    a) The CustomerID column is marked as NOT NULL, ensuring that each wallet is associated with a valid customer.

CryptoCurrency:

```
CREATE TABLE CryptoCurrency (
    CryptoID INT PRIMARY KEY,
    currencyName VARCHAR(30),
    Symbol VARCHAR(10),
    MarketPrice DECIMAL(20, 10),
    WhitePaper TEXT,
    CreationYear YEAR,
    CHECK (CreationYear <= YEAR(CURDATE())),
    CHECK (currencyName IN ('Bitcoin', 'Ethereum', 'Binance Coin', 'Cardano', 'Solana',
                    'XRP', 'Polkadot', 'Dogecoin', 'Avalanche', 'Litecoin', 'Chainlink', 'Stellar')),
    CHECK (Symbol IN ('BTC', 'ETH', 'BNB', 'ADA', 'SOL', 'XRP', 'DOT',
                        'DOGE', 'AVAX', 'LTC', 'LINK', 'XLM'))
);
```

CryptoCurrency Tuples:

CryptoID	currencyName	Symbol	MarketPrice	WhitePaper	CreationYear
101	Bitcoin	BTC	60000.12	Link to Bitcoin Whitepaper	2009
102	Ethereum	ETH	2000.988	Link to Ethereum Whitepaper	2015
103	Binance Coin	BNB	400.5679	Link to Binance Coin Whitepaper	2017
104	Cardano	ADA	2.345679	Link to Cardano Whitepaper	2015
105	Solana	SOL	150.6789	Link to Solana Whitepaper	2020
106	XRP	XRP	1.234568	Link to XRP Whitepaper	2012
107	Polkadot	DOT	30.09877	Link to Polkadot Whitepaper	2020
108	Dogecoin	DOGE	0.56789	Link to Dogecoin Whitepaper	2013
109	Avalanche	AVAX	80.12346	Link to Avalanche Whitepaper	2020
110	Litecoin	LTC	180.9877	Link to Litecoin Whitepaper	2011

- Primary Key Constraint (CryptoID):
    a) The CryptoID column is set as the primary key for the CryptoCurrency table. This primary key constraint ensures that each cryptocurrency entry is uniquely identified within the table.

- Check Constraint (CreationYear):
    a. The CreationYear column has a check constraint (CHECK (CreationYear <= YEAR(CURDATE()))) ensuring that the specified creation year is not in the future. This constraint helps maintain the integrity of the data by preventing the insertion of cryptocurrencies with a creation year beyond the current year.

- Check Constraints (currencyName, Symbol):
    a) The currencyName and Symbol columns have check constraints restricting their values to a predefined set. This ensures that only valid cryptocurrency names and symbols are entered into the table. The check constraints for currencyName and Symbol help

maintain consistency and prevent the insertion of cryptocurrencies with unrecognized names or symbols.

Loan:

```
CREATE TABLE Loan (
    LoanID INT PRIMARY KEY,
    Amount DECIMAL(10, 2),
    MaxAmount DECIMAL(10, 2),
    DateTaken DATE,
    DueDate DATE,
    InterestRate DECIMAL(8, 2),
    CustomerID INT not null,
    AccountID INT not null,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
    FOREIGN KEY (AccountID) REFERENCES Account(AccountID),
    CHECK (InterestRate >= 0),
    CHECK (Amount >= 0),
    CHECK (MaxAmount >= 0),
    CHECK (DateTaken <= DueDate)
);
```

LoanID	Amount	MaxAmount	DateTaken	DueDate	InterestRate	CustomerID	AccountID
1	5000	10000	1/15/2023	1/15/2024	5.5	1	1
2	10000	20000	2/10/2023	2/10/2024	7.2	2	2
3	7500	12000	3/5/2023	3/5/2024	6	3	3
4	12000	25000	4/20/2023	4/20/2024	8.3	4	4
5	20000	35000	5/15/2023	5/15/2024	6.8	5	5
6	15000	28000	6/1/2023	6/1/2024	7.5	6	6
7	18000	30000	7/10/2023	7/10/2024	6.2	1	7
8	25000	40000	8/25/2023	8/25/2024	8	8	8
9	3000	6000	9/18/2023	9/18/2024	5	9	9
10	22000	45000	10/5/2023	10/5/2024	7.9	1	10

The "Loan" table is designed to store information about loans, including details such as loan ID, amount, maximum amount, date taken, due date, interest rate, and the associated customer and account.

- Primary Key Constraint (LoanID):
    a) The LoanID column serves as the primary key for the Loan table. This primary key constraint ensures that each loan record has a unique identifier.
- Foreign Key Constraints (CustomerID, AccountID):
    a) The CustomerID and AccountID columns have foreign key constraints that reference the Customer and Account tables, respectively. These constraints establish relationships

between the Loan table and the Customer and Account tables, linking each loan to a specific customer and account.

- Check Constraints (InterestRate, Amount, MaxAmount, DateTaken, DueDate):
    a) CHECK (InterestRate >= 0): Ensures that the interest rate is non-negative.
    b) CHECK (Amount >= 0): Ensures that the loan amount is non-negative.
    c) CHECK (MaxAmount >= 0): Ensures that the maximum loan amount is non-negative.
    d) CHECK (DateTaken <= DueDate): Ensures that the date the loan was taken is before or equal to the due date.

StudentLoan:

```
-- StudentLoan
CREATE TABLE StudentLoan (
    LoanID INT PRIMARY KEY,
    UniversityInfo TEXT,
    ExpectedIncome DECIMAL(10, 2),
    ExpectedGradDate DATE,
    CHECK (ExpectedIncome >= 0)
);
```

LoanID	UniversityInfo	ExpectedIncome	ExpectedGradDate
1	LAU, Computer Science	60000	5/15/2024
2	AUB, Business Administration	80000	8/10/2024
3	University DEF, Engineering	70000	6/5/2024
4	University GHI, Medicine	90000	9/20/2024
5	University JKL, Engineering	50000	7/15/2024
6	LAU, Finance	75000	8/1/2025
7	AUB, Computer Science	65000	7/10/2024
8	LAU, Psychology	50000	10/25/2024
9	University VWX, Sociology	55000	9/18/2024
10	University YZ, History	70000	10/5/2025

The "StudentLoan" table is designed to store information specifically related to student loans, including details such as loan ID, university information, expected income, and expected graduation date.

- **Primary Key Constraint:**
    a) The **LoanID** column serves as the primary key, ensuring a unique identifier for each student loan.

HomeLoan:

```sql
CREATE TABLE HomeLoan (
    LoanID INT PRIMARY KEY,
    PropertyType VARCHAR(20),
    PropertyAppraisal DECIMAL(10, 2)
);
```

LoanID	PropertyType	PropertyAppraisal
101	Single Family Home	300000
102	Condo	200000
103	Townhouse	250000
104	Urban Loft	280000
105	Vacation Home	400000
106	Duplex	350000
107	Apartment	180000
108	Mansion	800000
109	Rural Property	220000
110	Cottage	180000

- **Primary Key Constraint:**
    a) The **LoanID** column serves as the primary key, ensuring a unique identifier for each home loan.

PersonalLoan:

```sql
-- PersonalLoan
CREATE TABLE PersonalLoan (
    LoanID INT PRIMARY KEY,
    CreditScore INT,
    LoanPurpose VARCHAR(255),
    CHECK (CreditScore >= 0)
);
```

LoanID	CreditScore	LoanPurpose
101	720	Home Improvement
102	680	Debt Consolidation
103	750	Travel
104	700	Education
105	780	Medical Expenses
106	650	Wedding
107	710	Car Purchase
108	740	Business Startup
109	690	Emergency
110	760	Vacation

- **Primary Key Constraint:**

    a) The **LoanID** column serves as the primary key, ensuring a unique identifier for each personal loan.

ATM:

```sql
CREATE TABLE ATM (
    ATMID INT PRIMARY KEY,
    CashBalance DECIMAL(10, 2),
    MaxAmount DECIMAL(10, 2),
    ATMCountry VARCHAR(20),
    ATMCity VARCHAR(20),
    ATMStreet VARCHAR(30),
    ATMPlace VARCHAR(30),
    CHECK (CashBalance >= 0),
    CHECK (MaxAmount >= 0),
    CHECK (BranchCountry IN ('USA', 'Lebanon', 'UK'))
);
```

ATM Tuples:

ATMID	CashBalance	MaxAmount	ATMCountry	ATMCity	ATMStreet	ATMPlace
1	15000	2000	USA	New York	123 Main St	Near Central Park
2	20000	2500	USA	Los Angeles	456 Oak Ave	Downtown LA
3	18000	2500	Lebanon	Beirut	789 Pine Rd	Hamra District
4	12000	1500	Lebanon	Byblos	101 Maple Blvd	Old Souk Area
5	25000	3000	UK	London	202 Cedar Ln	Financial District
6	22000	2800	USA	Chicago	303 Elm St	Downtown Loop
7	20000	2600	USA	Houston	404 Birch Ave	Midtown
8	28000	3500	Lebanon	Tripoli	505 Oak Rd	City Center
9	16000	2000	UK	Manchester	606 Pine Blvd	Northern Quarter
10	24000	3000	USA	San Francisco	707 Cedar Ave	Financial District
11	13000	1800	USA	Los Angeles	808 Maple St	Pioneer Square
12	18000	2200	Lebanon	Sidon	909 Oak Ave	Downtown Souk
13	20000	2500	UK	Glasgow	121 Birch Ln	City Center
14	21000	2500	USA	Miami	232 Elm Rd	South Beach
15	18000	2500	USA	Atlanta	343 Pine Blvd	Downtown
16	30000	4000	USA	New York	444 Oak St	Times Square
17	26000	3200	USA	Los Angeles	555 Pine Ave	Hollywood
18	22000	2500	Lebanon	Beirut	666 Cedar Rd	Downtown Gemmayze
19	17000	1800	Lebanon	Byblos	777 Maple Blvd	Byblos Old Port
20	32000	4500	UK	London	888 Elm Ln	Westminster
21	28000	3800	USA	Chicago	999 Birch St	Magnificent Mile
22	24000	3200	USA	Houston	1010 Oak Ave	Downtown Theater District
23	35000	5000	Lebanon	Byblos	1111 Pine Blvd	Al Mina
24	29000	4200	UK	Manchester	1212 Cedar Rd	Castlefield
25	26000	3500	USA	San Francisco	1313 Maple Blvd	Chinatown

The "ATM" table is designed to store information about automated teller machines (ATMs), including details such as ATM ID, cash balance, maximum withdrawal amount, and ATM location.

- Primary Key Constraint:
    a) The ATMID column serves as the primary key, ensuring a unique identifier for each ATM.
- Check Constraints:
    a) The CashBalance and MaxAmount columns have check constraints ensuring that the cash balance and maximum withdrawal amount are non-negative. This helps maintain the integrity of the financial data in the table.
    b) The BranchCountry column has a check constraint (CHECK (BranchCountry IN ('USA', 'Lebanon', 'UK', ‘Canada’))) ensuring that the ATM is associated with a valid country. This

constraint helps maintain consistency in recording the countries where ATMs are located.

Weak Entities:

Dependent:

```
CREATE TABLE Dependant (
    DependantID INT PRIMARY KEY,
    EmpID INT,
    Relationship VARCHAR(255),
    DependantName VARCHAR(255),
    DateOfBirth DATE,
    FOREIGN KEY (EmpID) REFERENCES Employee(EmployeeID),
    CHECK (DateOfBirth <= CURDATE())
);
```

Dependent Tuples:

DependantID	EmpID	Relationship	DependantName	DateOfBirth
1	1	Spouse	Emma Johnson	3/12/1985
2	3	Child	Alex Smith	9/5/2008
3	5	Spouse	Alex Brown	11/18/1990
4	7	Child	Ethan Martinez	7/22/2015
5	9	Child	Isabella Lopez	4/30/2012
6	2	Spouse	Alex White	6/14/1982
7	1	Uncle	Oliver Davis	12/8/2007
8	6	Child	Emma Garcia	4/25/2019
9	9	Spouse	Liam Johnson	8/29/1987
10	10	Child	Aria White	10/12/2011

The "Dependant" table is designed to store information about dependents of employees, including details such as dependent ID, employee ID, relationship, dependent name, and date of birth.

- **Primary Key Constraint:**
    a) The **DependantID** column serves as the primary key, ensuring a unique identifier for each dependent.
- **Foreign Key Constraint:**
    a) The **EmpID** column has a **NOT NULL** constraint, indicating that each dependent must be associated with a specific employee. It features a foreign key constraint referencing the

Employee table's **EmployeeID**, ensuring that the **EmpID** in the "Dependant" table corresponds to a valid employee in the "Employee" table.

- **Check Constraints:**

  a) The **DateOfBirth <= CURDATE()** constraint ensures that the date of birth is not in the future, maintaining temporal accuracy.

- **Not Null Constraints:**

  a) The **EmpID** column is marked as **NOT NULL**, ensuring that each dependent is associated with a specific employee.

Beneficiary:

```
CREATE TABLE Beneficiary (
    BeneficiaryID INT PRIMARY KEY,
    AccountID INT,
    AccountNum VARCHAR(255),
    FirstName VARCHAR(255),
    LastName VARCHAR(255),
    FOREIGN KEY (AccountID) REFERENCES Account(AccountID)
);
```

Beneficiary Tuples:

BeneficiaryID	AccountID	AccountNum	FirstName	LastName
1	1	123456789	Alice	Johnson
2	3	987654321	Charlie	Smith
3	5	567890123	Emily	Brown
4	7	234567890	Mia	Martinez
5	9	678901234	Olivia	Lopez
6	2	345678901	Mia	White
7	4	789012345	Harper	Davis
8	6	012345678	Sebastian	Johnson
9	8	456789012	Mia	Rodriguez
10	10	901234567	Jackson	Hernandez

The "Beneficiary" table is designed to store information about beneficiaries associated with bank accounts, including details such as beneficiary ID, account ID, account number, first name, and last name.

- **Primary Key Constraint:**

  a) The **BeneficiaryID** column serves as the primary key, ensuring a unique identifier for each beneficiary.

- **Foreign Key Constraint:**

    a) The **AccountID** column has a **NOT NULL** constraint, indicating that each beneficiary must be associated with a specific bank account. It features a foreign key constraint referencing the **Account** table's **AccountID**, ensuring that the **AccountID** in the "Beneficiary" table corresponds to a valid account in the "Account" table.

- **Not Null Constraints:**

    a) The **AccountID** column is marked as **NOT NULL**, ensuring that each beneficiary is associated with a specific bank account.

Relationships

Links_with:

```
CREATE TABLE Links_with (
    AccountID INT,
    PrivateKey INT,
    DateOpened DATE,
    ConfirmationInfo VARCHAR(255),
    FOREIGN KEY (AccountID) REFERENCES Account(AccountID),
    FOREIGN KEY (PrivateKey) REFERENCES CryptoWallet(PrivateKey),
    CHECK (DateOpened <= CURDATE())
);
```

Links_with:

AccountID	PrivateKey	DateOpened	ConfirmationInfo
1	101	1/15/2022	Confirmation123
2	102	2/20/2022	Confirmation234
3	103	3/25/2022	Confirmation345
10	104	4/30/2022	Confirmation456
7	105	5/15/2022	Confirmation567
9	106	6/20/2022	Confirmation678
8	107	7/25/2022	Confirmation789
4	108	8/30/2022	Confirmation890
9	109	9/5/2022	Confirmation901
6	110	10/10/2022	Confirmation102

The "Links_with" table is designed to establish links between bank accounts and cryptocurrency wallets, recording details such as account ID, private key, date opened, and confirmation information.

- **Foreign Key Constraints:**

    a) The **AccountID** column has a foreign key constraint referencing the **Account** table's **AccountID**, ensuring that the **AccountID** in the "Links_with" table corresponds to a valid account in the "Account" table.

    b) The **PrivateKey** column has a foreign key constraint referencing the **CryptoWallet** table's **PrivateKey**, ensuring that the **PrivateKey** in the "Links_with" table corresponds to a valid private key in the "CryptoWallet" table.

- **Check Constraints:**

    a) The **DateOpened <= CURDATE()** constraint ensures that the date the link was opened is not in the future, maintaining temporal accuracy.

Contains:

```
CREATE TABLE Contains (
    WalletID INT,
    CryptoID INT,
    FOREIGN KEY (WalletID) REFERENCES CryptoWallet(PrivateKey),
    FOREIGN KEY (CryptoID) REFERENCES CryptoCurrency(CryptoID)
);
```

Contains Tuples:

WalletID	CryptoID
101	101
102	102
103	103
104	104
105	105
101	102
107	101
101	108
109	101
110	101

The "Contains" table is designed to represent a relationship between cryptocurrency wallets and the cryptocurrencies they contain.

- **Foreign Key Constraints:**

a) The **WalletID** column has a foreign key constraint referencing the **CryptoWallet** table's **PrivateKey**, ensuring that the **WalletID** in the "Contains" table corresponds to a valid private key in the "CryptoWallet" table.

b) The **CryptoID** column has a foreign key constraint referencing the **CryptoCurrency** table's **CryptoID**, ensuring that the **CryptoID** in the "Contains" table corresponds to a valid cryptocurrency in the "CryptoCurrency" table.

Supervise:

```
CREATE TABLE Supervise (
    SuperviserID INT,
    SuperviseeID INT,
    FOREIGN KEY (SuperviserID) REFERENCES Employee(EmployeeID),
    FOREIGN KEY (SuperviseeID) REFERENCES Employee(EmployeeID),
    CHECK (SuperviserID <> SuperviseeID) -- supervisor is not their own supervisee
);
```

Supervise Tuples:

SuperviserID	SuperviseeID
1	2
1	3
1	4
1	5
2	6
2	7
2	8
3	9
3	10
3	11
4	12
4	13
4	14
5	15
5	16
5	17
6	18
6	19
6	20
7	21
7	22
7	23
8	24

8	**25**
8	**26**
9	**27**
9	**28**
9	**29**
10	**30**

- **Foreign Key Constraints:**
    a) The **SuperviserID** column has a foreign key constraint referencing the **Employee** table's **EmployeeID**, ensuring that the **SuperviserID** in the "Supervise" table corresponds to a valid employee in the "Employee" table.
    b) The **SuperviseeID** column has a foreign key constraint referencing the **Employee** table's **EmployeeID**, ensuring that the **SuperviseeID** in the "Supervise" table corresponds to a valid employee in the "Employee" table.
- **Check Constraint:**
    a) The **CHECK (SuperviserID <> SuperviseeID)** constraint ensures that a supervisor cannot be their own supervisee, preventing circular relationships.

Works_in:

```
CREATE TABLE Works_in (
    EmpID INT,
    DepID INT,
    Hours DECIMAL(5, 2),
    FOREIGN KEY (EmpID) REFERENCES Employee(EmployeeID),
    FOREIGN KEY (DepID) REFERENCES Department(DepartmentID),
    CHECK (DepartmentID = (SELECT DepartmentID FROM Employee WHERE EmployeeID = EmployeeID))
);
```

Works_in:

EmpID	DepID	Hours
1	**1**	40
2	**2**	35.5
3	**3**	38
4	**4**	42.5
5	**5**	37
6	**1**	41.5
7	**2**	36
8	**3**	39.5
9	**4**	40
10	**5**	35.5

11	**1**	38
12	**2**	42.5
13	**3**	37
14	**4**	41.5
15	**5**	36
16	**1**	39.5
17	**2**	40
18	**3**	35.5
19	**4**	38
20	**5**	42.5
21	**1**	37
22	**2**	41.5
23	**3**	36
24	**4**	39.5
25	**5**	40
26	**1**	35.5
27	**2**	38
28	**3**	42.5
29	**4**	37
30	**5**	41.5

- **Foreign Key Constraints:**

    a) The **EmpID** column has a foreign key constraint referencing the **Employee** table's **EmployeeID**, ensuring that the **EmpID** in the "Works_in" table corresponds to a valid employee in the "Employee" table.

    b) The **DepID** column has a foreign key constraint referencing the **Department** table's **DepartmentID**, ensuring that the **DepID** in the "Works_in" table corresponds to a valid department in the "Department" table.

- **Check Constraint:**

    a) The **CHECK (DepID = (SELECT DepartmentID FROM Employee WHERE EmployeeID = EmpID))** constraint needs to be modified to reference the correct columns. It seems you want to ensure that the department ID in "Works_in" matches the department ID of the employee in the "Employee" table. The correct constraint is **CHECK (DepID = (SELECT DepartmentID FROM Employee WHERE EmployeeID = EmpID))**.

Refilled:

```
CREATE TABLE Refilled (
    EmployeeID INT,
    ATMID INT,
    BranchID INT,
    FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID),
    FOREIGN KEY (ATMID) REFERENCES ATM(ATMID),
    FOREIGN KEY (BranchID) REFERENCES Branch(BranchID),
    CHECK (BranchID = (SELECT BranchID FROM Employee WHERE EmployeeID = EmployeeID))
);
```

Refilled Tuples:

EmployeeID	ATMID	BranchID
24	8	9
8	8	9
11	6	6
10	5	5
23	6	6
4	3	3
6	1	1
12	7	7
3	2	2
16	2	2
20	6	6
21	6	6
14	6	6
13	9	8

The "Refilled" table is designed to represent instances where an employee refills an ATM at a specific branch.

- **Foreign Key Constraints:**

    a) The **EmployeeID** column has a foreign key constraint referencing the **Employee** table's **EmployeeID**, ensuring that the **EmployeeID** in the "Refilled" table corresponds to a valid employee in the "Employee" table.

    b) The **ATMID** column has a foreign key constraint referencing the **ATM** table's **ATMID**, ensuring that the **ATMID** in the "Refilled" table corresponds to a valid ATM in the "ATM" table.

    c) The **BranchID** column has a foreign key constraint referencing the **Branch** table's **BranchID**, ensuring that the **BranchID** in the "Refilled" table corresponds to a valid branch in the "Branch" table.

- **Check Constraint:**

    a) The **CHECK (BranchID = (SELECT BranchID FROM Employee WHERE EmployeeID = EmployeeID))** constraint needs to be modified to reference the correct columns. It seems you want to ensure that the branch ID in "Refilled" matches the branch ID of the employee in the "Employee" table. The correct constraint is **CHECK (BranchID = (SELECT BranchID FROM Employee WHERE EmployeeID = Refilled.EmployeeID))**.

Issues:

```
CREATE TABLE Issues (
    DepartmentID INT,
    EmployeeID INT,
    NewsID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID),
    FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID),
    FOREIGN KEY (NewsID) REFERENCES CryptoNews(NewsID)
);
```

DepartmentID	EmployeeID	NewsID
2	2	2
1	1	10
2	2	2
3	3	10
4	9	10
2	2	7
3	8	10
1	6	7
4	4	9
1	1	6
5	30	6
2	22	1
2	17	2
5	20	8
2	7	1
4	9	5
1	6	7

The "Issues" table is designed to represent instances where employees from a specific department are involved in cryptocurrency news-related issues.

- **Foreign Key Constraints:**

a) The **DepartmentID** column has a foreign key constraint referencing the **Department** table's **DepartmentID**, ensuring that the **DepartmentID** in the "Issues" table corresponds to a valid department in the "Department" table.

b) The **EmployeeID** column has a foreign key constraint referencing the **Employee** table's **EmployeeID**, ensuring that the **EmployeeID** in the "Issues" table corresponds to a valid employee in the "Employee" table.

c) The **NewsID** column has a foreign key constraint referencing the **CryptoNews** table's **NewsID**, ensuring that the **NewsID** in the "Issues" table corresponds to a valid news article in the "CryptoNews" table..

Undergoes:

```
CREATE TABLE Undergoes (
    AccountID INT,
    TransactionID INT,
    DestinationAccountID INT,
    TargetCurrency VARCHAR(255),
    SourceCurrency VARCHAR(255),
    FOREIGN KEY (AccountID) REFERENCES Account(AccountID),
    FOREIGN KEY (TransactionID) REFERENCES Transaction(TransactionID),
    FOREIGN KEY (DestinationAccountID) REFERENCES Account(AccountID)
);
```

Undergoes Tuples:

AccountID	TransactionID	DestinationAccountID	TargetCurrency	SourceCurrency
5	14	7	EUR	EUR
10	9	5	USD	JPY
20	16	13	USD	USD
18	1	6	LBP	USD
14	10	2	GBP	USD
5	3	7	EUR	USD
7	4	5	USD	EUR
17	20	12	LBP	LBP
2	19	11	Bitcoin	USD
7	6	9	EUR	Ethereum
2	18	10	USD	Bitcoin
13	2	3	USD	USD
15	5	17	Bitcoin	Bitcoin
6	7	14	Bitcoin	Ethereum
3	11	4	USD	Litecoin
6	12	14	EUR	Litecoin
6	15	18	USD	USD

6	17	14	USD	Bitcoin
1	13	19	USD	EUR
19	8	16	Bitcoin	USD

The "Undergoes" table is designed to capture information about transactions where funds are moved between accounts, specifying the source and destination accounts, along with details about the currencies involved.

- **Foreign Key Constraints:**

  a) The **AccountID** column has a foreign key constraint referencing the **Account** table's **AccountID**, ensuring that the **AccountID** in the "Undergoes" table corresponds to a valid account in the "Account" table.

  b) The **TransactionID** column has a foreign key constraint referencing the **Transaction** table's **TransactionID**, ensuring that the **TransactionID** in the "Undergoes" table corresponds to a valid transaction in the "Transaction" table.

  c) The **DestinationAccountID** column has a foreign key constraint referencing the **Account** table's **AccountID**, ensuring that the **DestinationAccountID** in the "Undergoes" table corresponds to a valid account in the "Account" table.

Multivalued Attribute

PhoneNumber:

```
CREATE TABLE PhoneNumber (
    CustomerID INT,
    PhoneNum VARCHAR(20),
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
);
```

PhoneNumber Tuples:

CustomerID	PhoneNum
1	+1-123-456-7890
2	+1-987-654-3210
3	+961-71-234567
4	+961-76-543210
5	+44-20-1234-5678
6	+1-234-567-8901
7	+961-81-987654
8	+44-20-9876-5432
9	+1-345-678-9012
10	+61-2-9876-5432

11	+44-20-8765-4321
12	+961-70-987654
13	+1-555-123-4567
14	+44-20-3456-7890
15	+961-81-987654
16	+1-123-987-6543
17	+961-70-987654
18	+44-20-8765-4321
19	+61-2-9876-5432
20	+1-555-123-4567
21	+44-20-3456-7890
22	+961-71-234567
23	+961-76-543210
24	+44-20-1234-5678
25	+1-234-567-8901
26	+961-81-987654
27	+44-20-9876-5432
28	+1-345-678-9012
29	+61-2-9876-5432
30	+1-123-987-6543
1	+961-76-123456
2	+1-555-987-6543
3	+44-20-8765-1234
4	+61-2-3456-7890
5	+961-70-987123
6	+1-555-987-3210
7	+44-20-8765-4321
8	+961-71-987654
8	+961-71-917654
8	+961-71-917614
9	+61-2-9876-9876
1	+961-71-123455

The "PhoneNumber" table is designed to store phone numbers associated with customer accounts.

- **Foreign Key Constraint:**

    a) The **CustomerID** column has a foreign key constraint referencing the **Customer** table's **CustomerID**, ensuring that the **CustomerID** in the "PhoneNumber" table corresponds to a valid customer in the "Customer" table.

Basic Queries:

Task 1: IT and Customer Support Employees in the US.

Query 1:

Scenario: A manager wants to evaluate salaries in different departments. They are interested in IT employees in the US earning above $90,000 and Customer Support employees earning less than $80,000. Select employees that work in IT who live in the US and make over 90 thousand a year, or work in Customer support but make less than 80 thousand a year. Display their name, salary, sex, and age.

```
SELECT
    ename,
    salary,
    esex,
    age
FROM employee e
JOIN works_in w
    ON w.empid = e.employeeid
JOIN department d
    ON d.departmentID = w.depID
WHERE dname = 'Information Technology' AND salary > 90000
UNION
SELECT
    ename, salary, esex, age
FROM employee e
JOIN works_in w
    ON w.empid = e.employeeid
JOIN department d
    ON d.departmentID = w.depID
WHERE dname = 'Customer Support' AND salary < 80000;
```

Test Query:

	ename	salary	esex	age
▶	Olivia Rodriguez	95000.00	Female	35
	Chloe Evans	95000.00	Female	36
	Ava Turner	95000.00	Female	35
	William Lopez	78000.00	Male	31
	Daniel Parker	78000.00	Male	32
	Logan Hall	78000.00	Male	31

Query 2:

Task 2: Branches with Highest Average Transactions.

Scenario: A financial analysis is being conducted to identify which branches have the highest average transactions in each country, helping us understand market dynamics. Showcase which branches have the highest average transactions for each country and then display the branch id, name, country, and average transactions ordered descendingly by the average transactions and country name. Round the average to 2 decimal places.

```sql
SELECT
    b.BranchID,
    BranchName,
    BranchCountry,
    ROUND(AVG(amount), 2) AS average_transactions
FROM Branch b
NATURAL JOIN transaction
GROUP BY b.BranchID, BranchName, BranchCountry
ORDER BY BranchCountry, ROUND(AVG(amount), 2) DESC;
```

Test Query:

BranchID	BranchName	BranchCountry	average_transactions
3	Downtown Branch	Lebanon	107.00
10	Southern Lebanon Branch	Lebanon	85.00
9	Northern Lebanon Branch	Lebanon	75.00
8	Midlands Branch	UK	168.33
5	City Centre Branch	UK	113.00
1	Main Street Branch	USA	500.00
7	Harrisonbourgh	USA	250.00
2	Sunset Boulevard Branch	USA	120.00
6	Downtown Chicago Branch	USA	80.00

Query 3:

Task 3: Average Balance and Loan per Account Type

Scenario: A bank's marketing team needs data on the average balance and loan amount per account type to tailor their advertising campaigns. Display the average balance and average loan per account type. Show the Account type, average balance, and average loan. Round the averages to 2 decimal places

```sql
SELECT
    AccountType,
    ROUND(AVG(balance), 2) AS average_balance,
    ROUND(AVG(Amount), 2) AS average_loan_amount
FROM Account a
JOIN Loan USING(AccountID)
GROUP BY  AccountType;
```

Test Query:

AccountType	average_balance	average_loan_amount
Savings	6600.00	17750.00
Checking	3611.11	18055.56
Credit	-425.00	23300.00
Investment	13666.67	27333.33

Query 4:

Task 4: Cryptowallet Details of Customers Named 'J' and 'A'.

Scenario: For a special promotion, the bank targets customers whose first name starts with 'J' or 'A', providing insights on their cryptowallet holdings. For each customer whose first name starts with a J or an A, display the amount of money in their cryptowallets as well as the crypto currencies in them. Display the customer's full name, balance, and currency name.

```
SELECT
    concat(CFirstName,' ', CLastName) AS full_name,
    balance,
    currencyName
FROM customer c
JOIN CryptoWallet cw
    ON c.CustomerID = cw.CustomerID
JOIN contains con
    ON con.WalletID = cw.PrivateKey
JOIN CryptoCurrency cc
    ON cc.CryptoID = con.CryptoID
WHERE CFirstName LIKE 'J%' OR CFirstName LIKE 'A%';
```

Test Query:

full_name	balance	currencyName
John Johnson	5.00	Bitcoin
John Johnson	5.00	Ethereum
John Johnson	5.00	Dogecoin
Jane Smith	8.20	Ethereum
Abdulrahman Rodriguez	3.70	Cardano
Abdulrahman Rodriguez	4.50	Bitcoin

Query 5:

Task 5: Average Monthly Views on Crypto News.

Scenario: A media company wants to analyze the average number of views per month on their cryptocurrency news articles to plan content strategies. Display the average number of views per month.

```
SELECT
    EXTRACT(MONTH FROM publicationDate) As month,
    AVG(views) as average_views
FROM CryptoNews cn
GROUP BY EXTRACT(MONTH FROM publicationDate);
```

Test Query:

	month	average_views
▶	1	5000.0000
	2	3500.0000
	3	2000.0000
	4	8000.0000
	5	4500.0000
	6	6000.0000
	7	3000.0000
	8	7000.0000
	9	2500.0000
	10	4000.0000

Query 6:

Task 6: Total Transactions Per Account

Scenario: The bank is assessing customer engagement by counting the total number of transactions per account. Display the total number of transactions per account. The final output should contain the accountID, customer's full name, and the total number of transactions

```
SELECT
    a.AccountID,
    CONCAT(CFirstName, ' ', CLastName) AS full_name,
    COALESCE(SUM(t.TransactionID), 0) AS total_number_of_transactions
FROM Customer c
JOIN Account a
    ON a.CustomerID = c.CustomerID
JOIN Undergoes u
    ON u.AccountID = a.AccountID
LEFT JOIN transaction t
    ON t.TransactionID = u.TransactionID
GROUP BY a.AccountID, CONCAT(CFirstName, ' ', CLastName);
```

Test Query:

	AccountID	full_name	total_number_of_transactions
▶	1	John Johnson	48
	2	Jane Smith	51
	3	Khaleel Johnson	28
	4	Emily Davis	64
	5	Abdulrahman Brown	62
	6	Sophia Garcia	33
	7	Daniel Martinez	44
	14	Sophie Anderson	29

Query 7:

Task 7: Employees with Children Over 10.
Scenario: For a family benefits program, the company needs to identify employees with at least one child over the age of 10. For each employee that has at least one child that is over 10, showcase his/her name, salary, department, sex, and age.

```
SELECT
    ename,
    salary,
    dname,
    esex,
    age
FROM employee e
JOIN works_in wi
    ON wi.empID = e.EmployeeID
JOIN Department d
    ON d.DepartmentID = wi.DepID
WHERE EmployeeID IN (SELECT EmpID FROM dependant
                WHERE TIMESTAMPDIFF(YEAR, dateOFBirth, current_date()) > 10 and relationship = 'Child');
```

Test Query:

	ename	salary	dname	esex	age
▶	Chris Smith	90000.00	Information Technology	Male	35
	William Lopez	78000.00	Customer Support	Male	31
	Sophia Garcia	92000.00	Branch Operations	Female	34

Query 8:

Task 8: Top 5 Countries in Average Loan Amounts

Scenario: An international bank is evaluating which countries have the highest average loan amounts for strategic lending policies. Select the top 5 countries in terms of average loan amounts. Order the results from highest to lowest.

```
SELECT
    cCountry As Country,
    AVG(amount) AS average_loan_amount
FROM customer c
JOIN loan l
    ON l.CustomerID = c.CustomerID
GROUP BY cCountry
ORDER BY AVG(amount) DESC
LIMIT 5;
```

Test Query:

	Country	average_loan_amount
▶	Germany	30000.000000
	UK	30000.000000
	Italy	27400.000000
	France	26000.000000
	India	23333.333333

Query 9:

Task 9: Customer Phone Number Registrations.

Scenario: The telecom department is analyzing where most of the bank's customers' phone numbers are registered to optimize network coverage. Find out where most customers' phone numbers are registered display the count of customers while grouping by the telephone country code. Order the results by the count of customers from highest to lowest

```
SELECT
    LEFT(PhoneNum,LOCATE('-', PhoneNum) - 1) AS country_code,
    COUNT(CustomerID) as Number_of_customers
FROM Phonenumber
GROUP BY LEFT(PhoneNum,LOCATE('-', PhoneNum) - 1)
ORDER BY COUNT(CustomerID) DESC;
```

Test Query:

	country_code	Number_of_customers
▶	+961	15
	+1	12
	+44	10
	+61	5

Query 10:

Task 10: Total Cash in ATMs by Country.

Scenario: The bank's logistics team is assessing the cash distribution in ATMs, focusing on countries with above-average cash holdings. What is the total amount of cash available in ATMs across different countries. Showcase only countries that have more or equal to the average of the total amount of money across countries.

```sql
SELECT
    ATMCity,
    ROUND(SUM(cashbalance)) As Total_cash_available
FROM ATM
GROUP BY ATMCity
HAVING ROUND(SUM(CashBalance)) >
    (SELECT
        AVG((Total_cash_available))
    FROM (
        SELECT
            ATMCity,
            ROUND(SUM(cashbalance)) As Total_cash_available
        FROM ATM
        GROUP BY ATMCity)
    AS subquery) ;
```

Test Query:

	ATMCity	Total_cash_available
▶	New York	45000
	Los Angeles	59000
	Beirut	40000
	Byblos	64000
	London	57000
	Chicago	50000
	Houston	44000
	Manchester	45000
	San Francisco	50000

Query 11:

Task 11: Student Loans Per Major.

Scenario: A financial aid department is analyzing the distribution of student loans across different majors to tailor their support programs. Show the total number of students per major who have taken a loan. The result should contain the number of students aggregated by the major.

```sql
SELECT
    SUBSTRING(UniversityInfo, LOCATE(',', UniversityInfo) +2) AS major,
    COUNT(DISTINCT c.customerID) AS number_of_students
FROM StudentLoan sl
JOIN Loan l
    ON sl.LoanID = l.LoanID
JOIN customer c
    ON c.CustomerID = l.CustomerID
GROUP BY SUBSTRING(UniversityInfo, LOCATE(',', UniversityInfo) +2)
ORDER BY number_of_students DESC;
```

Test Query:

major	number_of_students
Computer Science	2
Engineering	2
Business Administration	1
Finance	1
History	1
Medicine	1
Psychology	1
Sociology	1

Query 12:

Task 12: Number of Transactions Per Country.

Scenario: The bank's international division is looking at the number of transactions per country to understand global transaction patterns. What is the number of transactions per country?

```
SELECT
    COUNT(transactionID) AS number_of_transactions,
    BranchCountry
FROM Branch b
LEFT JOIN transaction t
    ON b.BranchID = t.BranchID
GROUP BY BranchCountry;
```

Test Query:

number_of_transactions	BranchCountry
8	USA
14	Lebanon
8	UK

Query 13:

Task 13: UK Account Holders' Balances

Scenario: For a regional performance assessment, the bank needs to know the total balances of customers who have accounts in UK branches. For customers who have accounts in UK branches, display their total balance. The result should include the id, full name, balance and should be ordered in alphabetical order.

```sql
SELECT
    c.CustomerID,
    CONCAT(cFirstName, ' ', CLastName) AS full_name,
    balance
FROM customer c
JOIN account a
    ON c.CustomerID = a.CustomerID
JOIN Branch b
    ON b.BranchID = a.BranchID
WHERE BranchCountry = 'UK'
ORDER BY full_name;
```

Test Query:

CustomerID	full_name	balance
5	Abdulrahman Brown	7500.00
10	Abdulrahman Rodriguez	4000.00
20	Alain Hernandez	4500.00
23	Jackson Harris	13000.00
13	Kevin Williams	15000.00
3	Khaleel Johnson	12000.00
30	Lily Baker	5000.00

Query 14:

Task 14: Employee-Supervisor Relationships.

Scenario: The HR department is creating an organizational chart, showing each employee alongside their direct supervisor. For each employee, display their supervisor. The result should showcase both the name of the employee and the supervisor as well as the IDs

```sql
WITH CTE AS (
SELECT ename, SuperviserID, EmployeeID FROM supervise s
RIGHT JOIN employee e
    ON e.Employeeid = s.superviseeId)
SELECT t1.ename AS employee,
       COALESCE(t2.ename) AS superviserID,
       t1.employeeID,
       t2.employeeID AS superviserID
FROM CTE t1
LEFT JOIN CTE t2
ON t2.EmployeeID = t1.SuperviserID;
```

#	Time	Action	Message
1	18:47:35	use test4 WITH CTE AS (SELECT ename, SuperviserID, EmployeeID FROM supervis...	Error Code: 1064. You have an error in your SQL syntax; check the manual that corres...

Query 15:

Task 15: Non-Personal Loans Future Amounts.

Scenario: The credit risk team is analyzing the future value of non-personal loans for different interest rates to assess financial risks. For every loan that is not personal, display what the amount will be in a year for every interest rate on the table. The interest is compound annually.

```
WITH CTE AS (
    (SELECT
    LoanID
    FROM Loan)
    EXCEPT
    (
    SELECT
        LoanID
    FROM PersonalLoan))
SELECT
    l1.amount,
    l2.interestrate,
    l1.amount*(1+l2.interestRate/100) AS amount_a_year_from_now
FROM loan l1
JOIN CTE
    ON CTE.LoanID = l1.LoanID
cross join loan l2
ORDER BY amount DESC;
```

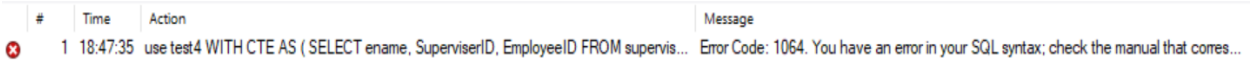
#	Time	Action	Message
1	18:47:35	use test4 WITH CTE AS (SELECT ename, SuperviserID, EmployeeID FROM supervis...	Error Code: 1064. You have an error in your SQL syntax; check the manual that corres...

Query 16:

Task 16: Personal Loans Future Amounts.

Scenario: As part of a risk assessment, the bank is calculating the future amounts of personal loans at various interest rates. Do the same task as the previous one, only this time for Personal Loans only.

```
WITH CTE AS (
    (SELECT
    LoanID
    FROM Loan)
    INTERSECT
    (
    SELECT
        LoanID
    FROM personalLoan))
SELECT
    l1.amount,
    l2.interestrate,
    l1.amount*(1+l2.interestRate/100) AS amount_a_year_from_now
FROM loan l1
JOIN CTE
    ON CTE.LoanID = l1.LoanID
cross join loan l2
ORDER BY amount DESC
```



#	Time	Action	Message
1	18:47:35	use test4 WITH CTE AS (SELECT ename, SuperviserID, EmployeeID FROM supervis...	Error Code: 1064. You have an error in your SQL syntax; check the manual that corres...

Advanced Queries:

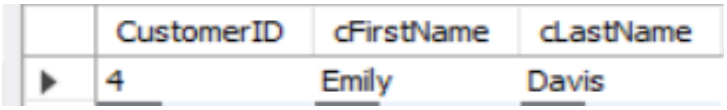
Query 1:

```
SELECT DISTINCT C.CustomerID, C.cFirstName, C.cLastName
FROM Customer C
WHERE C.CustomerID IN (
    SELECT CustomerID
    FROM Account A
    JOIN Undergoes u
        ON u.AccountID = A.AccountID
    JOIN Transaction T
        ON T.TransactionID = u.TransactionID
    WHERE T.Amount > 10000
);
```

Task: Extract a list of all customers who have conducted transactions exceeding $100

Scenario: A bank wants to identify customers who have made high-value transactions in order to offer them exclusive services or promotions. The bank defines a high-value transaction as any transaction exceeding $100. The marketing team plans to use this information to tailor special offers, such as higher interest rates on deposits or personalized wealth management services, to these high-value customers.

Test Query:



CustomerID	cFirstName	cLastName
4	Emily	Davis

Query 2:

```
SELECT *
FROM Employee E
WHERE Salary >
    (SELECT
        AVG(Salary)
    FROM Employee WHERE BranchID = E.BranchID);
```

Task: Generate a report showing employees whose salaries are above the average salary in their branch. The report should include employee IDs, names, salaries, and branch IDs.

Scenario: A multinational corporation wants to identify employees who are earning more than the average salary in their respective branches. This analysis is part of a performance and compensation review to ensure equitable pay scales across different locations.

Test Query:

EmployeeID	Salary	ename	EmployeeCountry	EmployeeCity	EmployeeZipCode	EmployeeStreet	EmployeeEmail	esex	Age	BranchID
3	90000.00	Chris Smith	USA	Los Angeles	90001	101 Maple Rd	chris.smith@email.com	Male	35	2
6	85000.00	Ava Hernandez	USA	New York	10002	404 Elm St	ava.hernandez@email.com	Female	33	1
7	88000.00	Michael Brown	USA	Los Angeles	90002	606 Pine Ave	michael.brown@email.com	Male	30	2
8	95000.00	Saad Hamdoun	Lebanon	Beirut	54322	707 Cedar Rd	sosoh@email.com	Male	35	9
10	92000.00	Sophia Garcia	UK	London	SW1A 1AB	909 Oak Ln	sophia.garcia@email.com	Female	34	5
13	90000.00	Ryan Walker	UK	Manchester	M1 1AB	333 Pine Blvd	ryan.walker@email.com	Male	36	8
17	88000.00	David Hall	France	Nice	06000	777 Maple Rd	david.hall@email.com	Male	31	2
18	95000.00	Chloe Evans	Italy	Milan	20121	888 Pine Blvd	chloe.evans@email.com	Female	36	3
20	92000.00	Aria Hughes	India	Delhi	110001	1010 Oak St	aria.hughes@email.com	Female	35	6
23	90000.00	Mason Miller	USA	Chicago	60601	101 Maple Rd	mason.miller@email.com	Male	35	6
26	85000.00	Scarlett Young	USA	New York	10002	404 Elm St	scarlett.young@email.com	Female	33	1
27	88000.00	Owen Adams	USA	Los Angeles	90002	606 Pine Ave	owen.adams@email.com	Male	30	2
28	95000.00	Ava Turner	Lebanon	Beirut	54322	707 Cedar Rd	ava.turner@email.com	Female	35	3
30	92000.00	Emma Walker	UK	London	SW1A 1AB	909 Oak Ln	emma.walker@email.com	Female	34	5

Query 3:

```
SELECT COUNT(*) AS number_of_customers
FROM Customer
WHERE CustomerID IN (
    SELECT CustomerID
    FROM Account
    GROUP BY CustomerID
    HAVING COUNT(AccountID) >= 2);
```

Task: Create a database query to count customers with two or more accounts.

Scenario: A bank is running a promotional campaign targeted at multi-account holders. They want to count the number of customers who have two or more accounts of any type (savings, checking, etc.) to tailor their marketing strategy accordingly.

Test Query:

number_of_customers
5

Query 4:

```
SELECT E.EmployeeID, E.ename
FROM Employee E
WHERE E.BranchID IN (
    SELECT A.BranchID
    FROM CryptoWallet CW
    JOIN Customer C
        ON CW.CustomerID = C.CustomerID
    JOIN Account A
        ON a.CustomerID = C.CustomerID
    WHERE CW.Balance > 10000
    AND a.CustomerID IN
        (SELECT cw.CustomerID
        FROM CryptoCurrency cc
        JOIN contains c
            ON cc.CryptoID = c.CryptoID
        JOIN CryptoWallet CW
            ON CW.PrivateKey = c.WalletID
        JOIN customer cr
            ON cr.CustomerID = CW.CustomerID
        GROUP BY cw.customerID
        HAVING COUNT(DISTINCT WalletID) >= 2));
```

Task: Identify employees working in branches where there are customers with high-value cryptocurrency accounts (over $10,000) and at least 2 wallets. The output should include employee IDs and their names.

Scenario: Identify Employees Working in Branches with High-Value Cryptocurrency Accounts
Objective: Find employees who work in branches where at least one customer has a cryptocurrency wallet balance exceeding a certain high-value threshold, say $10,000. We are only interested in customers who have at least 2 wallets. This scenario assumes that a higher wallet balance may indicate a branch with more affluent clientele or a focus on cryptocurrency services.

Test Query:

	EmployeeID	ename
*	NULL	NULL

Query 5:

```
SELECT CustomerID FROM Customer C
WHERE NOT EXISTS (
    (SELECT BranchID FROM Branch)
    EXCEPT
    (SELECT BranchID
    FROM Account WHERE CustomerID = C.CustomerID)
);
```

Task: Compile a list of 'elite' customers who hold accounts in every branch. The list should include customer IDs, names, the number of branches where they hold accounts, and the account details.

Scenario: For a customer loyalty program, a financial institution wants to identify 'elite' customers who have accounts in every branch of the bank. This is to offer them exclusive services and benefits as part of a premium customer engagement program.

Query 6:

```
SELECT BranchID, AVG(Balance) AS AverageBalance
FROM (SELECT BranchID, Balance FROM Account) AS BranchBalances
GROUP BY BranchID;
```

Task: Calculate and report the average account balance for each branch. The report should include branch IDs, and the average balance per branch.

Scenario: A regional bank is conducting an internal audit to assess the financial health of each branch. They need to calculate the average balance maintained in accounts at each branch to evaluate the branches' performance and identify areas needing attention.

Test Query:

BranchID	AverageBalance
1	3675.000000
2	12125.000000
3	2312.500000
4	4800.000000
5	5250.000000
6	5766.666667
7	3266.666667
8	13333.333333
9	-566.666667
10	7900.000000

Query 7:

```
SELECT
    B.BranchID,
    B.BranchName,
    (SELECT COUNT(*) FROM Employee WHERE BranchID = B.BranchID) AS EmployeeCount,
    (SELECT AVG(Salary) FROM Employee WHERE BranchID = B.BranchID) AS AverageSalary
FROM
    Branch B;
```

Task: For each branch, find the branch ID, branch name, the number of employees working there, and the average salary of these employees.
Scenario: A corporation seeks to evaluate the staffing and salary metrics of its various branches. The goal is to obtain a snapshot of each branch's workforce size and the average salary for strategic planning. This data will help in assessing whether some branches are understaffed or overstaffed and if the salary distributions align with the company's standards and regional economic conditions. The information is crucial for informed decision-making in human resource management and financial budgeting across the company's network of branches.

Test Query:

BranchID	BranchName	EmployeeCount	AverageSalary
1	Main Street Branch	3	83333.333333
2	Sunset Boulevard Branch	5	87800.000000
3	Downtown Branch	3	90666.666667
4	Seaside Branch	3	78000.000000
5	City Centre Branch	3	86333.333333
6	Downtown Chicago Branch	5	84800.000000
7	Harrisonbourgh	2	75000.000000
8	Midlands Branch	2	80000.000000
9	Northern Lebanon Branch	2	88500.000000
10	Southern Lebanon Branch	2	70000.000000

Query 8:

```
UPDATE CreditCard cc
JOIN Customer c ON c.CustomerID = cc.CustomerID
JOIN CryptoWallet CW ON CW.CustomerID = c.CustomerID
SET cc.CreditLimit = CASE
    WHEN CW.Balance < 5000 THEN 10000
    WHEN CW.Balance BETWEEN 5000 AND 10000 THEN 15000
    ELSE 20000
END;
```

Task: Credit Limit Adjustment Based on Account Balance

Scenario: A financial institution is implementing a new policy to adjust the credit limits of its customers' credit cards based on their account balances. The idea is to reward customers with higher credit limits if they maintain higher balances in their accounts, thereby encouraging more deposits and providing an incentive for customers to use the bank's financial services more actively. The bank wants to automatically update the credit limits of all credit cards based on the current balance in the associated accounts. The policy stipulates the following rules. If the account balance is less than $5,000, the credit limit is set to $10,000. If the account balance is between $5,000 and $10,000, the credit limit is increased to $15,000. For balances exceeding $10,000, the credit limit is boosted to $20,000.

Test Query:

```
41  23:01:29  UPDATE CreditCard cc JOIN Customer c ON c.CustomerID = cc.CustomerID JOIN ...  6 row(s) affected Rows matched: 6  Changed: 6  Warnings: 0
```

Query 9:

```
SELECT EmployeeID, ename, dname
FROM Employee
LEFT OUTER JOIN Department
    ON Employee.BranchID = Department.DepartmentID;
```

Task: Update the internal directory by creating a comprehensive list of all employees, including those not assigned to any department. The list should include employee IDs, their names and department names.

Scenario: A large enterprise is updating its internal directory and wants to create a comprehensive list of all employees, including their names, employee IDs, and the departments they belong to. This includes employees who are currently not assigned to any department.

Test Query:

EmployeeID	ename	dname
1	Robert Johnson	Corporate Banking
2	Sarah White	Branch Operations
3	Chris Smith	Investment Banking
4	Emily Davis	Information Technology
5	Daniel Martinez	NULL
6	Ava Hernandez	Corporate Banking
7	Michael Brown	Investment Banking
8	Saad Hamdoun	NULL
9	William Lopez	Customer Support
10	Sophia Garcia	Branch Operations
11	Jason Turner	NULL
12	Mia Cooper	NULL
13	Ryan Walker	NULL
14	Emma Robinson	NULL
15	James Reed	NULL
16	Sophie Turner	Investment Banking

17	David Hall	Investment Banking
18	Chloe Evans	Information Technology
19	Daniel Parker	Customer Support
20	Aria Hughes	NULL
21	Elijah Taylor	NULL
22	Lily Adams	NULL
23	Mason Miller	NULL
24	Zoe Parker	NULL
25	Aiden Scott	NULL
26	Scarlett Young	Corporate Banking
27	Owen Adams	Investment Banking
28	Ava Turner	Information Technology
29	Logan Hall	Customer Support
30	Emma Walker	Branch Operations

Query 10:

```
CREATE ASSERTION MaxEmployeeLimit
CHECK (
    NOT EXISTS (
        SELECT BranchID
        FROM Employee
        GROUP BY BranchID
        HAVING COUNT(EmployeeID) > 100
    )
);
```

Task: Ensure that no branch has more than 100 employees.
Scenario: The bank manager believes having more than 100 employees in one branch will lessen productivity. A new rule is thus announced that no branch shall have more than a 100 employees.

Test Query:

```
43  23:08:27  CREATE ASSERTION MaxEmployeeLimit CHECK (  NOT EXISTS (  SELEC...  Error Code: 1064. You have an error in your SQL syntax; check the manual that corr...
```

Query 11:

```
CREATE VIEW CustomerLoanDetails AS
SELECT
    c.CustomerID,
    cFirstName,
    cLastName,
    SUM(Amount) AS TotalLoanAmount
FROM Customer c
JOIN Loan l
    ON c.CustomerID = l.CustomerID
GROUP BY CustomerID;
```

Task: Create a view to show customer details along with their total loan amount.
Scenario: A financial institution is looking to streamline its customer service and account management processes. Specifically, the bank wants to have a quick and efficient way to access detailed loan information for each customer. This need arises from the bank's customer service representatives

frequently requiring a comprehensive view of a customer's total loan amount when handling inquiries, offering new services, or conducting regular financial reviews.

Test Query:

```
44  23:11:48  CREATE VIEW CustomerLoanDetails AS SELECT  c.CustomerID,    cFirstName,  ...  0 row(s) affected
```

Query 12:

```
CREATE FUNCTION CalculateTotalBalance(customer_id INT)
RETURNS DECIMAL(10, 2)
BEGIN
    DECLARE total_balance DECIMAL(10, 2);

    SELECT SUM(Balance) INTO total_balance
    FROM Account
    WHERE CustomerID = customer_id;

    RETURN total_balance;
END;
```

Task: Calculate Total Balance through all accounts for customer.
Scenario: A customer service representative at the bank needs to review a customer's overall financial standing with the bank. A long-standing customer, Mr. Johnson, calls in to inquire about his total funds available across all his accounts before considering a major purchase. The representative uses the CalculateTotalBalance function, inputting Mr. Johnson's customer ID. The function quickly calculates the total balance by summing up the funds in all of Mr. Johnson's accounts, enabling the representative to inform him about his total available balance in a matter of seconds.

Test Query:

```
1 •  select calculateTotalBalance(1);
```

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

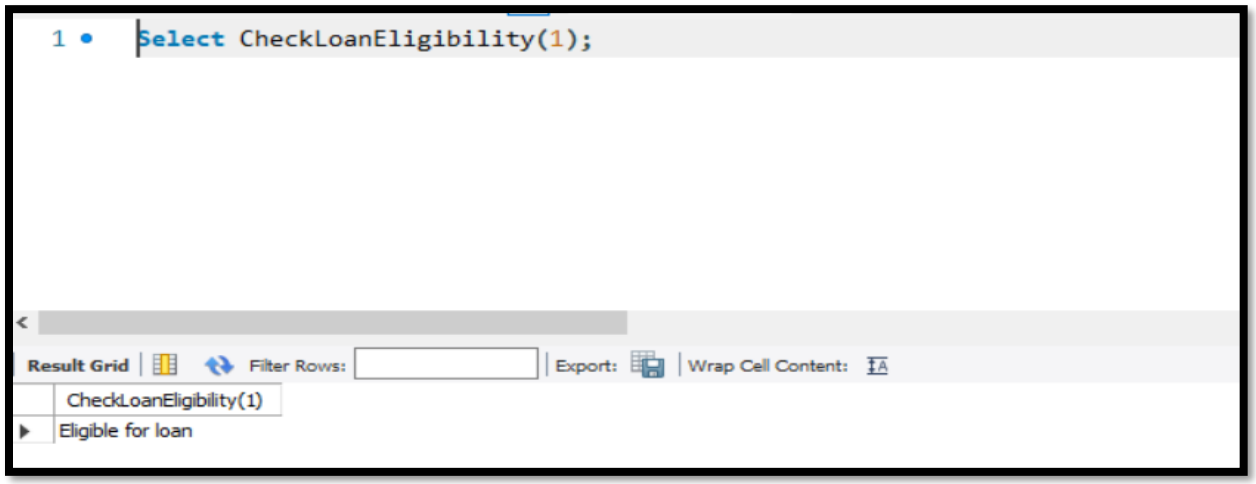
calculateTotalBalance(1)
11000.00

Query 13:

```
CREATE FUNCTION CheckLoanEligibility(customer_id INT)
RETURNS VARCHAR(100)
BEGIN
    DECLARE eligibility VARCHAR(100);
    DECLARE total_balance DECIMAL(10, 2);
    DECLARE account_opening_date DATE;
    DECLARE years_with_bank INT;

    -- Calculate total balance
    SET total_balance = CalculateTotalBalance(customer_id);

    -- Find the earliest account opening date
    SELECT MIN(DateOpened) INTO account_opening_date
    FROM Account
    WHERE CustomerID = customer_id;

    -- Calculate the number of years with the bank
    SET years_with_bank = TIMESTAMPDIFF(YEAR, account_opening_date, CURDATE());

    -- Determine eligibility based on balance and years with bank

    IF total_balance >= 5000 AND years_with_bank >= 2 THEN
        SET eligibility = 'Eligible for loan';
    ELSE
        SET eligibility = 'Not eligible for loan';
    END IF;

    RETURN eligibility;
END;
```

Task: Check if customer is eligible for a loan
Scenario: A customer asks for a loan from the bank, the bank tests their eligibility using this function.
CREATE FUNCTION CheckLoanEligibility(customer_id INT)

Test Query:



Query 14:

```
CREATE PROCEDURE UpdateCustomerAddress(
    IN cust_id INT,
    IN new_country VARCHAR(30),
    IN new_city VARCHAR(30),
    IN new_zipcode VARCHAR(10),
    IN new_street VARCHAR(255)
)
BEGIN
    UPDATE Customer
    SET cCountry = new_country,
        cCity = new_city,
        cZipcode = new_zipcode,
        cStreet = new_street
    WHERE CustomerID = cust_id;
END;
```

Task: Update Customer Address
Scenario: A customer moves to a new address and informs the bank. The bank uses this procedure to update the customer's address in their records.

Test Query:

```
1 •  CALL UpdateCustomerAddress(1, 'Lebanon', 'Sidon', '00000', 'Nejme');
```

136 01:57:53 CALL UpdateCustomerAddress(1, 'Lebanon', 'Sidon', '00000', 'Nejme') 1 row(s) affected 0.015 sec

Query 15:

```
CREATE PROCEDURE CreateNewAccount(
    IN account_type VARCHAR(20),
    IN initial_balance DECIMAL(10, 2),
    IN customer_id INT,
    IN branch_id INT
)
BEGIN
    DECLARE new_account_id INT;

    -- Generate new AccountID
    SELECT MAX(AccountID) + 1 INTO new_account_id FROM Account;

    -- Create new account
    INSERT INTO Account(AccountID, AccountType, Balance, DateOpened, CustomerID, BranchID)
    VALUES (new_account_id, account_type, initial_balance, CURDATE(), customer_id, branch_id);
END;
```

Task: Open new Account for customer.
Scenario: A customer decided to open a new Account.

Test Query:

```
1 •  CALL CreateNewAccount('Savings', 1000.00, 1, 1);
```

✔	138	02:01:30	CALL CreateNewAccount('Savings', 1000.00, 1, 1)	1 row(s) affected	0.000 sec

WE HOPE YOU ENJOYED OUR HONEST BANK!