Online Consumer Banking Database
Database Design and Initial ERD
Database Specification: Purpose, Business Problems Addressed, and Business Rules

Database Purpose:

The purpose of this database is to support efficient storage, update, and retrieval of consumer banking account and transaction data. Aiming to facilitate daily banking activities, our database will enable individual users to manage and track their account activities by providing real-time access to accurate and secure information.

Business Problems Addressed:

- Facilitate daily banking activities including account opening and transaction recording (i.e., transfers, deposits, or withdrawal)
- Streamline account management and tracking process for customers with the ease to:
    - Access account information such as account number and balances
    - Retrieve transaction history and detailed transaction information
    - Generate detailed financial statements on account balances, transaction history, spending & budgeting, etc.
    - Set up alerts with regard to low balance, overdraft, or transaction limits, etc.

Business Rules:

- Each client may have one or more accounts
- Each client may have zero or more transactions
- Each account can only belong to one client
- Each account may have one and only one account type
- Each account may have zero or more transactions
- Each account may belong to one and only one br

Design Requirements:

- Eliminate any multi-valued and/or composite attribute
- Remove any repeating group
- Avoid a many-to-many relationship
- Pay attention to the multiplicity (cardinality and partition)
- Make sure the type of a relationship is correct (Identifying vs Non-Identifying)
- All entities must be connected
- Avoid any unnecessary relationship
- Make sure the ERD and design document match

Design Decisions:

Entity Name	**Why Entity Included**	**How Entity is Related to** Other Entities
Client	One of the main purposes of this database is to support data storage, update, and retrieval for individual online banking customers. The **Client** entity stores essential client information such as clientID, name, etc.	As one of the core entities of this database, the **Client** entity's primary key, clientID, serves as the foreign key to **Account** entity. The clientID makes it possible to retrieve data of individual clients. One client could have multiple accounts. Therefore, the **Client** entity is involved in a one-to-many relationship with **Account** and it is the parent entity.
Account	The **Account** entity is used to store and retrieve basic account information, including account ID, client ID, account type ID, routing number, account balance, registration branch, overdraft limit and account warning.	The **Account** entity is also one of the core entities. The primary key, accountID, serves as the foreign key to **Transaction** and **Overdraft** entities. The **Account** entity is the child entity to the **Client** entity since one client could have multiple accounts. Since one account could have multiple transactions and multiple overdraft records, the **Account** entity has one-to-many relationships with **Transaction** and **Overdraft** entities and it is the parent entity relative to these two entities. After each withdraw, paying, and transfer out transaction, if the overdraft balance exceeds account overdraft limit, then AccountWarning will be on. If the warning account client pays back the overdraft amount (account balance becomes positive), then AccountWarning will be off.
AccountType	The **AccountType** entity stores information on the types of account the bank offers to clients. It is used to identify which types of account it is. For instance, checking accounts, saving account, etc.	The **AccountType** entity only relates to the **Account** entity. One account may have one and only one account type. Zero or many accounts may be classified to one account type. The **AccountType** entity relates to the **Account** entity through its AccountTypeID as primary key, which is a foreign key in the **Account** entity.

Transaction	The purpose of the **Transaction** entity is to capture the details of the transactions that occurred on a particular account, including ClientAccountID, TransactionTypeID, TransactionAmount, TransactionCurrency, TransactionDate, TransactionTime, TransactionStatus, TransactionNotes. This information will help clients to be aware of their account transactions at a glance. In the event of unauthorized or malicious use of individual bank accounts, clients can immediately retrieve more details by searching for transaction information.	The **Transaction** entity connects to the **TransactionType, TransactionStatus**, **Account,** and five main subtype entities, which represents five main online banking functions (transferIn, transferOut, withdraw, saving, paying). Since each transaction has one specific function, subtypes are **disjoint or**. The discrimination attribute is the transaction type. Since we assume that there might be some other transactions that do not belong to any of the five types, the subtypes are partially completed, and the **partition constraint** is **optional**. We are implementing the disjoint or rule here, which means that once a transaction is identified as a Transfer transaction, it will not be a Withdrawal transaction, and it is independent of other subtype entities.
TransactionType	The **TransactionType** entity contains information about the types of transactions. Serving as the subtype discriminator, this entity allows us to easily distinguish different types of transactions while ensuring the ease of data retrieval.	The **TransactionType** entity connects with the **Transaction** entity. One transaction only has one and only transaction type, one transaction type could be shown in zero or many transactions.
TransactionStatus	The **TransactionStatus** entity stores information on the current status of the transaction. It could be viewed as a helper entity to the **Transaction** entity. It includes status such as ongoing, finished, failed, etc. Each status is given an unique id.	The **TransactionStatus** entity connects with the **Transaction** entity. One transaction only has one and only transaction status at one time, one transaction status could be shown in zero or many transactions.
TransferIn	A sub-entity of **Transaction**. Record the transfer in activities. The unique field is the senderAccountID.	**TransferIn** is linked to **Transaction** through its primary key and foreign key of Transaction, TransactionID, and the relationship between these two entities is identifying.

TransferOut	A sub-entity of **Transaction**. Record the transfer out activities, and determine whether this transaction leads to **Overdraft**. The unique fields are receiverAccountID, Overdraft, OverdraftID	**TransferOut** is linked to **Transaction** through its primary key and foreign key of Transaction, TransactionID, and the relationship between these two entities is identifying. **Transfer** is also linked to **Overdraft** through OverdraftID. One transfer out could be one and only one overdraft activity. Therefore, the relationship is Optional one - Mandatory one.
Withdraw	A sub-entity of **Transaction**. Record the withdrawing activities of an account, and determine whether this transaction leads to **Overdraft**. The unique fields are Overdraft, OverdraftID	**Withdraw** is linked to **Transaction** through its primary key and foreign key of Transaction, TransactionID, and the relationship between these two entities is identifying. One withdraw could be one and only one overdraft activity. Therefore, the relationship is Optional one - Mandatory one.
Saving	A sub-entity of **Transaction**. Record the saving/ depositing activities of the account. The unique field is interestRate	**Saving** is linked to **Transaction** through its primary key and foreign key of Transaction, TransactionID, and the relationship between these two entities is identifying.
Paying	A sub-entity of **Transaction**, Record the paying activities between the sender account and the merchant, and determine whether this transaction leads to **Overdraft**. The unique fields are Overdraft, OverdraftID, MerchantAccountID, MerchantDescription, MerchantType	**Paying** is linked to **Transaction** through its primary key and foreign key of Transaction, TransactionID, and the relationship between these two entities is identifying. One paying transaction could be one and only one overdraft activity. Therefore, the relationship is Optional one - Mandatory one.
OverDraft	Keep the log for each overdraft transaction and the overdraft amount, and identify whether each overdraft exceeds the account overdraft limit.	**Overdraft** is linked to three types of transaction activities: **TransferOut**, **Withdraw**, **Paying**, through its primary key OverdraftID. It is linked with **Account** through foreign key AccountID. An overdraft activity must be traced back to a transaction activity.
Branch	The branch where the account is opened	The primary key of **Branch**, BranchID, serves as the foreign key of **Account**. A branch can have zero or many

		accounts, but an account must belong to a branch.