CS331-003

GROUP #7

FALL 2024

BANK DATABASE REPORT

PREPARED BY

FARAZ MERCHANT
NONA HARRIS
ANGEL CAZARES

fbm@njit.edu - Faraz nrf@nji.edu - Nona ajc253@njit.edu - Angel

DESIGN AND REQUIREMENT SUMMARY

We were tasked with creating a database system for a bank organization, managing different internal and external components of the organization and the entities within it. Beginning with the bank, we identified each unique branch with its branch-ID, recorded its name, address (decomposed into city, zip, state, and street), manager and assistant manager. In the relational database, branchID was the unique primary key. Initially, when posed with keeping track of the bank's numerous assets, we decided that this attribute was multivalued, and the relationship would be stored as its own relation when designing the schema, ASSETS, with its unique name and the branch-ID it is associated with being its primary key. This decision was appropriate because each branch might have multiple assets and, to prevent redundancy, while preserving the relationship between the two entities through a foreign key (branchID), a new relation was necessary. In the actual relational database design, BRANCH and ASSET are separate relations, with the branchID serving as a foreign key reference. Upon deletion of the branch from the system, we decided a corresponding behavior of cascade was appropriate, as it would no longer be necessary to keep track of the assets for a branch that is no longer in the database.

Another component of the design was the customer managing the customer information. Since each one is uniquely identified by their social security number, this was mapped as the primary key (SSN), and remained so in the relational database design. We also had to keep track of their name (decomposed into first, last, middle initial in the entity relation design), address (decomposed into apartment number, street number, state, city, and zip code in the entity relation design), managing accounts, branch, and personal banker. The accounts, branch, and personal banker are all separate entities in the system, and we had to decide how to map the relations.

Since each customer may have a number of accounts (1:N), which is its own entity, we decided that a new relation, mapping the primary identifier for customer (SSN), and the identifier for account (accountNo) would be created in the schema, with the customer SSN and accountNo serving as the primary key in the relational database design, preserving the relationship while minimizing redundancy. Since a customer may be associated with a branch, but a branch could have multiple customers, when mapping the entity in the schema, the CUSTOMER entity got mapped the attribute branchID, as the relationship is necessarily one-to-one for the customer, but one-to-many for the branch. In the relational database, this was a foreign key, relating CUSTOMER to BRANCH, and was kept optional as a customer may not be associated with a branch at all. The same occurred for the relationship with the bank employee, as one customer is only associated with one employee, so this field was mapped to the CUSTOMER entity in the entity relation design, with the foreign key of SSN (EMPLOYEE). However, in the relational database design, this field was required to be non-null, as every customer must have a banker, and upon deletion of the employee (when removed from the system), the action is restricted; this is so that the banker of the customer is required to be updated any time an employee leaves the system, a design requirement we deemed appropriate.

The next requirement was keeping track of the employee information. Their primary identifier, SSN, was kept as the primary key in the relational database. Their telephone number, name (decomposed into first, last, middle initial in the entity relation design), address (decomposed into street, state, city, and zip code) start date, dependents, length of employment, manager, assistant manager, and associated branch were also other required attributes. However, when mapping this in an entity-relational model, we noticed that length of employment was a derived attribute from start date, so we did not include this in our relational design. This is because, in a relational database, a simple query can be run with the information of the stored attribute (start date) in order to obtain that information. Additionally, we felt it necessary to enhance the entity relation design by storing the sex and date of birth of each employee in the system.

Since each employee could have multiple dependents, and dependents represent physical entities in the system, albeit weak entities, we decided to create a new relation in the design phase, DEPENDENT, to keep track of the information for the dependents. As the entity is weak, its primary key is a combination of its identifying entity's (employee) primary key, SSN, and its name (partial key), which will be unique for each tuple in the relational database. To enhance the design further, we also decided to store the sex and date of birth for each dependent. In the actual database design, a component of the primary key (ESSN) was a foreign key reference to employee; upon deletion of an employee, the result cascades in the dependent relation, as there would no longer be a foreign key to reference in the employee table, and it would no longer be necessary to keep track of the dependents for employees that are not a part of the system.

Since each employee can have a manager or assistant manager, but those individuals can oversee multiple employees, these relationships are a 1:N recursive relationship (as outlined in the system requirements), as shown in the ER diagram. To map these in the schema design and relational database, we thus created separate attributes in the employee relationship for managerSSN and assistantManagerSSN to keep track of these relationships. A foreign key reference was used, with the employee relation referencing itself (through SSN, managerSSN, and assistantManagerSSN). These attributes were allowed to be null, as an employee in the table could be the manager of the branch, thus not having their own manager. Upon deletion of the manager of assistant manager from the system, we ensured that the behavior was to set these attributes as null, in order to preserve the state of the other employees, who would still be a part of the system even if their supervisors are removed.

The next system requirement was maintaining a record of the accounts in the bank. There are several types of accounts (savings, checking, money market, and loan accounts), each assigned a unique account number, with their own respective balances, access dates, interest rates (depending on the type of account, fixed or variable), and records of overdraft. When choosing to map this entity in the initial entity-relation design phase, we noted that the account was a superclass, and the specific type of account was a defining subclass.

However, since all the subclasses shared mostly the same attributes (except the type of interest rate), and the participation of the superclass was total disjoint, we only created one ACCOUNT entity in the schema design and, thus, the relational database. This is because every account has to either be a savings, checking, money market, and loan account (total participation) and it can only be one of each (disjoint). When mapping this, we created a discriminating attribute (type) to discern the account type in the database, and combined the distinguishing attributes of the subclasses (fixed and variable interest rates) into a single relation, with accountID serving as the primary key, or unqiue identifying attribute for each tuple.

Since we also had to maintain a record of the customers and accounts they manage, which could be many-to-many (in the entity-relation design), as many customers can manage several different accounts, we mapped this as its own relation in the schema and database design. This relation, called MANAGES, maps customers to accounts through the customerSSN and accountNo attributes which, together, serves as the primary key. In the relational database, upon the deletion of either a customer or account from the system, the corresponding record in MANAGES will also be deleted (cascading behavior), as there would no longer be a corresponding customer or account participating in the relationship.

Additionally, we noted that an account could have many overdrafts, noted as a multivalued attribute in the ER diagram. So, when mapping this to the schema, we created a new relation, OVERDRAFT, to record the specific account and amount associated with the overdraft. This relation has the attributes accountNo and amount, serving as a unique identifier combined as the primary key, as we determined that an account could overdraft many times in the real world. Upon deletion of an account, the behavior in the database system is to cascade, as there would no longer be an account in the system associated with the overdrafts.

The next requirement was maintaining information regarding loans in the system. We noted that a loan was a unique entity in the system in the ER diagram, and had its own respective attributes. These include the unique loan number, loan amount, originating branch, and monthly repayment amount; thus, we created a separate relation, LOAN. It has the attributes loanNo, serving as the sole primary key due to its unique nature, amount, branchID, and repaymentAmount. In the physical database design, none of these attributes were allowed to be null, as all the information is required for a valid state. Additionally, the branchID attribute is a foreign key reference in the system, and we decided a cascade behavior was most appropriate, as it would no longer be necessary to maintain information on a loan with no corresponding branch.

As specified in the design requirements, several customers can hold a single loan (noted as an N:1 relationship in the ER diagram), so we created a separate relation TAKES_OUT in the schema and database. This relation keeps track of the necessarily many to one relationship of customer to loan, with the attributes customerSSN (for the corresponding customer) and loanNo (for the corresponding loan) serving as an identifying primary key.

These attributes are also foreign key references to the respective relations in the physical database, and we decided a behavior of cascade was appropriate, as there would no longer be a valid record if the underlying loan or corresponding customer was deleted from the database. Additionally, since loans are associated with specific accounts, we mapped a relation HOLDS to map this relationship in the schema design. It includes the attributes accountNo (for the corresponding account) and loanNo (for the corresponding loan). These two attributes serve as the unique primary key. Upon deletion of the underlying account or loan in the physical database, the behavior is to cascade, similar to the TAKES_OUT relation.

The last requirement was keeping track of the different transactions that an account makes. This includes information about the unique transaction code, the name of the transaction, the date, hour, amount, and whether a charge was registered for the transaction. As a single account can make numerous transactions (noted as a 1:N relationship), we decided to map this as a new relation, TRANSACTION in the schema. This relation has all the corresponding attributes, with the accountNo attribute being a foreign key reference to the ACCOUNT relation. Upon deletion of the account in the physical database, the behavior is to cascade, deleting all the corresponding transactions for the account. The primary key for this relation is the unique transaction code, uniquely identifying each tuple.

During the physical design of the database the most difficult part was choosing appropriate datatypes. This was because there were many ways to go about this, and we wanted to make sure our decisions would be effective when eventually inserting and querying the data. However, we were consistent with our choices throughout all the relations. We generally kept all SSN as CHAR(9) to make sense logically; all codes/ids as VARCHAR(5) to correlate to alphanumeric strings; phone number as a CHAR(10) fixed length string to correspond to US phone numbers, streets and cities as VARCHAR(n) to allow variable inputs; state as CHAR(2) to stay consistent with the implementation in the US; any attributes concerning money or interest rates as a DECIMAL(15,2) to indicate the proper precision and most reasonable amount of significant figures; dates as DATE datatype; and access date as a TIMESTAMP (for increased precision to record both, date and time). For the TRANSACTION relation, since charge was optional, we decided that a BOOLEAN data type (represented as a 0 or 1) would be sufficient for keeping track of whether or not a charge was administrated on a transaction.

RELATION NORMALIZATION

ACCOUNT (ACCOUNTNO, BALANCE, TYPE, FIXEDINTERESTRATE, VARIABLEINTERESTRATE, ACCESSDATE)

PRIMARY KEY: ACCOUNTNO

{ACCOUNTNO} -> {BALANCE, TYPE, ACCESSDATE, FIXEDINTERESTRATE, VARIABLEINTERESTRATE}

THIS RELATION IS IN 3NF BECAUSE ALL ATTRIBUTES ARE FULLY FUNCTIONALLY DEPENDENT ON THE CANDIDATE KEY AND THERE ARE NO TRANSITIVE DEPENDENCIES.

	ACCOUNTNO	BALANCE	TYPE	FIXEDINTERESTRATE	VARIABLEINTERESTRATE	ACCESSDATE
1	A1001	101234.67	Checking Account	(null)	(null)	15-NOV-24
2	A1002	1567.25	Savings Account	0.0025	(null)	15-NOV-24
3	A1003	5999	Money Market	(null)	0.0425	15-NOV-24
4	A1004	567.04	Loan Account	0.023	(null)	15-NOV-24
5	A1005	123.43	Loan Account	0.02	(null)	15-NOV-24
6	A1006	891.56	Loan Account	0.03	(null)	15-NOV-24
7	A1007	200.99	Loan Account	0.004	(null)	15-NOV-24
8	A1008	2000.04	Loan Account	0.005	(null)	15-NOV-24
9	A1009	356.19	Savings Account	0.05	(null)	15-NOV-24
10	A1010	10.5	Checking Account	(null)	(null)	15-NOV-24

ASSET (BRANCHID, ASSETNAME)

PRIMARY KEY: BRANCID, ASSETNAME
FOREIGN KEY: BRANCHID (BRANCH)

{BRANCHID, ASSETNAME} -> {BRANCHID, ASSETNAME}

THIS RELATION IS IN 3NF BECAUSE ALL ATTRIBUTES ARE FULLY FUNCTIONALLY DEPENDENT ON THE CANDIDATE KEY AND THERE ARE NO TRANSITIVE DEPENDENCIES.

	BRANCHID	ASSETNAME
1	B001	ATM Machine
2	B002	Computer
3	B003	Jewels
4	B004	Gold
5	B005	Silver

BRANCH (BRANCHID, STREET, STATE, CITY, ZIPCODE, MANAGERSSN, ASSISTANTMANAGERSSN, PHONENO)

PRIMARY KEY: BRANCH ID

{BRANCHID} -> {STREET, ZIPCODE, MANAGERSSN, ASSISTANTMANAGERSSN, PHONENO}

{ZIPCODE} -> {CITY, STATE}

THIS RELATION IS NOT IN 3NF BECAUSE ZIP CODE CAN DETERMINE THE CITY AND STATE IN THE CASE OF OUR DATABASE (IN THE UNITED STATES). SO, CITY AND STATE ARE TRANSITIVELY DEPENDENT ON BRANCHID. THUS, THE RELATION CAN BE DECOMPOSED TO REDUCE REDUNDANCY AND ANOMALIES. THE NEW LOCATIONS RELATION WILL HAVE ZIP CODE AS ITS PRIMARY KEY

DECOMPOSED RELATIONS

BRANCH (BRANCHID, STREET ZIPCODE, MANAGERSSN, ASSISTANTMANAGERSSN, PHONENO)

LOCATIONS(ZIPCODE, CITY, STATE)

	BRANCHID	STREET	STATE	CITY	ZIPCODE	MANAGERSSN	ASSISTANTMANAGERSSN	PHONE
1	B001	123 Elm St	NY	New York	10001	123456789	234567890	55512345
2	B002	456 Bern St	NJ	New Jersey	90001	456789012	567890123	55598765
3	B003	789 Pine St	TX	Los Angeles	75201	789012345	890123456	55523456
4	B004	101 Maple St	FL	Miami	33101	555666777	888999111	55534567
5	B005	202 Birch St	IL	Chicago	60601	898989899	123123123	55545678

CUSTOMER (SSN, FNAME, MINIT, LNAME, APTNO, STREET, STATE, CITY, ZIP, ESSN, BRANCHID)

PRIMARY KEY: SSN
FOREIGN KEYS: ESSN (EMPLOYEE), BRANCHID (BRANCH)

{SSN} -> {FNAME, MINIT, LNAME, APTNO, STREET, CITY, ESSN}

{ZIPCODE} -> {CITY, STATE}

THIS RELATION IS NOT IN 3NF BECAUSE THE CITY AND STATE CAN BE DETERMINED THROUGH THE UNIQUE ZIP CODE, SO IT IS TRANSITIVELY DEPENDENT ON SSN. IN THE CASE OF OUR DATABASE DESIGN. THUS, A NEW RELATION CAN BE CREATED; ONLY ONE WILL NEED TO BE KEPT FOR THE ENTIRE SCHEMA, AS CREATED FOR BRANCH. WHILE BRANCHID MAY BE THE SAME FOR THE CUSTOMER AND BANKER THEY'RE ASSOCIATED WITH, THIS DOES NOT NECESSARILY NEED TO HOLD BASED ON OUR DETERMINED REQUIREMENTS; THUS, SSN IS NEEDED TO DETERMINE THE BRANCHID (IF THE CUSTOMER IS ASSOCIATED WITH A PARTICULAR BRANCH).

DECOMPOSED RELATIONS

CUSTOMER (SSN, FNAME, MINIT, LNAME, APTNO, STREET, STATE, CITY, ZIP, ESSN, BRANCHID)

LOCATIONS(ZIPCODE, CITY, STATE)

	SSN	FNAME	MINIT	LNAME	APTNO	STREET	STATE	CITY	ZIP	ESSN	BRANCHID
1	444555666	Alice	L	Brown	101	123 Elm St	NY	New York	10001	345678901	B001
2	777888999	Bob	K	Green	202	456 Oak St	CA	Los Angeles	90001	678901234	B002
3	888999666	Charlie	J	White	303	789 Pine St	TX	Dallas	75201	901234567	B003
4	111222333	David	R	Black	404	101 Maple St	FL	Miami	33101	345678901	B004
5	444555777	Eve	T	Pink	505	202 Birch St	IL	Chicago	60601	789012345	B005

DEPENDENT(DEPENDENTNAME, ESSN, SEX, DOB)

{DEPEDENTNAME, ESSN} -> {SEX, DOB}

PRIMARY KEY: DEPENDENTNAME, ESSN
FOREIGN KEY: ESSN (EMPLOYEE)

THIS RELATION IS NOT IN 1NF BECAUSE DEPENDENT NAME CAN BE FURTHER BROKEN DOWN INTO FIRST NAME, MIDDLE INITIAL, AND LAST NAME. THUS, IT IS NOT AN ATOMIC ATTRIBUTE AND VIOLATES 1NF, SO THE RELATION IS NOT IN 3NF. THE ATTRIBUTE CAN BE FURTHER BROKEN DOWN, WITH FIRST NAME AND ESSN AS THE PRIMARY KEY. IT WILL THEN BE IN 3NF BECAUSE NO TRANSITIVE OR PARTIAL DEPENDENCIES WILL OCCUR ON THE NEW PRIMARY KEY (DEPENDENTNAME, ESSN), AND ALL ATTRIBUTES ARE ATOMIC.

NORMALIZED RELATION

DEPENDENT(FNAME, MINIT, LNAME, ESSN, SEX, DOB)

	DEPENDENTNAME	ESSN	SEX	DOB
1	Bill	123456789	M	15-JUN-10
2	Jill	456789012	F	22-AUG-12
3	Emma	789012345	F	30-MAR-15
4	Joey	123456789	M	01-OCT-18
5	Bob	567890123	M	10-NOV-20

EMPLOYEE (SSN, FNAME, MINIT, LNAME, MANAGERSSN, ASSISTANTMANAGERSSN, BRANCHID, STARTDATE, PHONENO, SEX, DOB, SALARY, STREET, STATE, CITY, ZIPCODE)

{SSN} -> {FNAME, MINIT, LNAME, STARTDATE, PHONENO, SEX, DOB, SALARY, STREET, CITY}

{ZIPCODE} -> {CITY, STATE}

PRIMARY KEY: SSN
FOREIGN KEY: MANAGERSSN (EMPLOYEE), ASSISTANTMANAGERSSN (EMPLOYEE), BRANCHID (BRANCH)

THIS RELATION IS NOT IN 3NF BECAUSE THE CITY AND STATE CAN BE DETERMINED THROUGH THE UNIQUE ZIP CODE, SO IT IS TRANSITIVELY DEPENDENT ON SSN IN THE CASE OF OUR DATABASE DESIGN (US ZIP CODES). THUS, A NEW RELATION CAN BE CREATED; ONLY ONE WILL NEED TO BE KEPT FOR THE ENTIRE SCHEMA, AS CREATED FOR BRANCH AND CUSTOMER. THEN, ALL THE RELATIONS WILL BE IN 3NF BECAUSE ALL ATTRIBUTES ARE FULLY FUNCTIONALLY DEPENDENT ON THE PRIMARY KEY.

NORMALIZED RELATION

EMPLOYEE (SSN, FNAME, MINIT, LNAME, MANAGERSSN, ASSISTANTMANAGERSSN, BRANCHID, STARTDATE, PHONENO, SEX, DOB, SALARY, STREET, ZIPCODE)

LOCATIONS(ZIPCODE, CITY, STATE)

	SSN	FNAME	MINIT	LNAME	MANAGERSSN	ASSISTANTMANAGERSSN	BRANCHID	STARTDATE	PHONENO	SEX	DOB	SALARY	STREET	STATE	CITY	ZIPCOD
1	123456789	John	A	Doe	(null)	234567890	B001	15-JAN-20	5551234567	M	12-MAY-85	85000	123 Elm St	NY	New York	10001
2	234567890	Jane	B	Smith	123456789	(null)	B001	10-FEB-21	5555678901	F	22-JUL-90	65000	456 Oak St	NY	New York	10002
3	345678901	Alice	C	Johnson	123456789	234567890	B001	25-MAR-22	5559876543	F	30-OCT-92	55000	789 Pine St	NY	New York	10003
4	456789012	Michael	D	Brown	(null)	345678901	B002	15-JUN-21	5553456789	M	20-FEB-83	90000	321 Birch St	NY	Brooklyn	11201
5	567890123	Laura	E	Taylor	456789012	(null)	B002	01-JUL-21	5558765432	F	17-MAR-87	70000	654 Maple St	NY	Brooklyn	11202
6	678901234	David	F	Wilson	456789012	567890123	B002	30-APR-22	5555432109	M	28-JAN-95	60000	987 Cedar Ave	NY	Brooklyn	11203
7	789012345	William	G	Davis	(null)	678901234	B003	20-NOV-20	5556789123	M	15-AUG-80	95000	789 Pine St	NY	Queens	11301
8	890123456	Olivia	H	Miller	789012345	(null)	B003	10-SEP-21	5552345678	F	03-MAY-92	72000	234 Oak Rd	NY	Queens	11302
9	901234567	Sophia	I	Anderson	789012345	890123456	B003	05-JAN-22	5556543210	F	12-DEC-94	63000	567 Maple Rd	NY	Queens	11303
10	555666777	John	J	Lee	(null)	901234567	B004	25-APR-21	5551234987	M	11-NOV-87	98000	123 Oak St	NY	Manhattan	10004
11	888999111	Evelyn	K	Young	555666777	(null)	B004	01-MAY-21	5555678901	F	10-SEP-90	75000	456 Elm St	NY	Manhattan	10005
12	101010101	James	L	Walker	555666777	888999111	B004	14-JUN-22	5559876543	M	02-MAR-96	68000	789 Birch St	NY	Manhattan	10006
13	898989899	Michael	M	Taylor	(null)	101010101	B005	01-DEC-20	5554321987	M	10-APR-82	97000	987 Cedar St	NY	Staten Island	10301
14	123123123	Grace	N	Jackson	898989899	(null)	B005	15-AUG-21	5558765432	F	25-JUN-94	71000	654 Maple Ave	NY	Staten Island	10302
15	456456456	Liam	O	Scott	898989899	123123123	B005	05-SEP-22	5555432109	M	17-NOV-93	64000	321 Pine Ave	NY	Staten Island	10303

HOLDS(ACCOUNTNO, LOANNO)

{ACCOUNTNO, LOANNO} -> {ACCOUTNO, LOANNO)

PRIMARY KEYS: ACCOUNTNO, LOANNO
FOREIGN KEYS: ACCOUNTNO (ACCOUNT), LOANNO (LOAN)

THIS RELATION IS IN 3NF BECAUSE ALL ATTRIBUTES ARE FULLY FUNCTIONALLY DEPENDENT ON THE PRIMARY KEY (ACCOUNTNO, LOANNO), AND NO ATTRIBUTES ARE TRANSITIVELY DEPENDENT ON THE PRIMARY KEY. BECAUSE THIS IS A TRIVIAL FUNCTIONAL DEPENDENCY SINCE THE RELATION ONLY CONSISTS OF THE PRIMARY KEY, THE RELATION IS AUTOMATICALLY IN 3NF.

	ACCOUN...	LOANNO
1	A1004	L001
2	A1005	L002
3	A1006	L003
4	A1007	L004
5	A1008	L005

LOAN(LOANNO, AMOUNT, BRANCHID, REPAYMENTAMOUNT)

{LOANNO} -> {AMOUNT, BRANCHID, REPAYMENTAMOUNT}

PRIMARY KEY: LOANNO
FOREIGN KEY: BRANCHID

THIS RELATION IS IN 3NF BECAUSE ALL ATTRIBUTES ARE FULLY FUNCTIONALLY DEPENDENT ON THE PRIMARY KEY (LOANNO), AND NO ATTRIBUTES ARE TRANSITIVELY DEPENDENT ON THE PRIMARY KEY.

	LOA...	AMOUNT	BRANCHID	REPAYMENTAMOUNT
1	L001	10000	B001	12000
2	L002	15000	B002	18000
3	L003	20000	B003	24000
4	L004	25000	B004	30000
5	L005	30000	B005	36000

MANAGES(CUSTOMERSSN, ACCOUNTNO)

{CUSTOMERSSN, ACCOUNTNO} -> {CUSTOMERSSN, ACCOUNTNO}

PRIMARY KEYS: CUSTOMERSSN, ACCOUNTNO
FOREIGN KEYS: CUSTOMERSSN (CUSTOMER), ACCOUNTNO (ACCOUNT)

THIS RELATION IS AUTOMATICALLY IN 3NF BECAUSE ALL ATTRIBUTES ARE FULLY FUNCTIONALLY DEPENDENT ON THE PRIMARY KEY (ACCOUNTNO, CUSTOMERSSN), AND NO ATTRIBUTES ARE TRANSITIVELY DEPENDENT ON THE PRIMARY KEY. BECAUSE THIS IS A TRIVIAL FUNCTIONAL DEPENDENCY SINCE THE RELATION ONLY CONSISTS OF THE PRIMARY KEY, THE RELATION IS IN 3NF.

	CUSTOMERSSN	ACCOUNTNO
1	444555666	A1001
2	444555666	A1005
3	777888999	A1002
4	777888999	A1006
5	888999666	A1003
6	888999666	A1007
7	111222333	A1004
8	111222333	A1010
9	444555777	A1008
10	444555777	A1009

OVERDRAFT(ACCOUNTNO, AMOUNT)

{ACCOUTNO, AMOUNT} -> {ACCOUTNO, AMOUNT}
PRIMARY KEY: ACCOUNTNO, AMOUNT
FOREIGN KEY: ACCOUNTNO (ACCOUNT)

THIS RELATION IS AUTOMATICALLY IN 3NF BECAUSE ALL ATTRIBUTES ARE FULLY FUNCTIONALLY DEPENDENT ON THE PRIMARY KEY (ACCOUNTNO, LOANNO), AND NO ATTRIBUTES ARE TRANSITIVELY DEPENDENT ON THE PRIMARY KEY; SINCE THE RELATION HAS NO NON-PRIME ATTRIBUTES (THERE IS ONLY A KEY), THE RELATION IS ALREADY IN 3NF.

Pin	ACCOUNTNO	AMOUNT
1	A1001	500
2	A1002	1000
3	A1003	750
4	A1010	15
5	A1009	1500

TAKES_OUT(CUSTOMERSSN, LOANNO)

{CUSTOMERSSN, LOANNO} -> {CUSTOMERSSN, LOANNO}

PRIMARY KEY: CUSTOMERSSN, LOANNO
FOREIGN KEYS: CUSTOMERSSN (CUSTOMER), LOANNO (LOAN)

THIS RELATION IS AUTOMATICALLY IN 3NF BECAUSE ALL ATTRIBUTES ARE FULLY FUNCTIONALLY DEPENDENT ON THE PRIMARY KEY (CUSTOMERSSN, LOANNO), AND NO ATTRIBUTES ARE TRANSITIVELY DEPENDENT ON THE PRIMARY KEY; SINCE THE RELATION HAS NO NON-PRIME ATTRIBUTES (THERE IS ONLY A KEY), THE RELATION IS ALREADY IN 3NF.

	CUSTOMER...	LOANNO
1	111222333	L001
2	444555666	L002
3	777888999	L003
4	888999666	L004
5	444555777	L005

TRANSACTION(ACCOUNTNO, CODE, NAME, AMOUNT, TRANSACTION_TIME, TRANSACTION_DATE, CHARGE)

{CODE} -> {ACCOUNTNO, NAME, AMOUNT, TRANSACTION_TIME, TRANSACTION_DATE, CHARGE}

PRIMARY KEY: CODE
FOREIGN KEY: ACCOUNTNO (ACCOUNT)

THIS RELATION IS IN 3NF BECAUSE ALL ATTRIBUTES ARE FULLY FUNCTIONALLY DEPENDENT ON THE PRIMARY KEY (CODE), AND NO ATTRIBUTES ARE TRANSITIVELY DEPENDENT ON THE PRIMARY KEY. THE CODE MUST BE KNOWN IN ORDER TO DETERMINE ALL THE OTHER CORRESPONDING NON-PRIME ATTRIBUTES; THUS, ALL ATTRIBUTES ARE FULLY FUNCTIONALLY DEPENDENT ON IT (DEFINITION OF KEY).

	ACCOUNTNO	CODE	NAME	AMOUNT	TRANSACTION_TIME	TRANSACTION_DATE	CHARGE
1	A1001	C001	WD	100	13-NOV-24 09.00.00.000000000 AM	10-JAN-24	1
2	A1002	C002	CD	200	13-NOV-24 10.00.00.000000000 AM	14-MAR-24	0
3	A1003	C003	WD	300	13-NOV-24 11.00.00.000000000 AM	13-AUG-24	1
4	A1009	C004	CD	400	13-NOV-24 12.00.00.000000000 PM	13-NOV-24	0
5	A1010	C005	WD	500	13-NOV-24 01.00.00.000000000 PM	25-JUL-24	1

QUERY #1

CALCULATE THE TOTAL VALUE OF ACCOUNTS ASSOCIATED WITH EACH BRANCH

```
SELECT B.BranchID, SUM(A.BALANCE) AS Total_Account_Value
FROM ACCOUNT A, MANAGES M, CUSTOMER C, BRANCH B
WHERE A.ACCOUNTNO = M.ACCOUNTNO AND M.CUSTOMERSSN = C.SSN AND C.BRANCHID = B.BRANCHID
GROUP BY B.BranchID;
```

Query Result

SQL | All Rows Fetched: 5 in 0.021 seconds

	BRANCHID	TOTAL_ACCOUNT_VALUE
1	B005	2356.23
2	B004	577.54
3	B002	2458.81
4	B003	6199.99
5	B001	101358.1

QUERY #2

ONLY DISPLAY THE BRANCHES WITH EMPLOYEES THAT HAVE MORE THAN 1 DEPENDENT AND THE NUMBER OF DEPENDENTS

```
SELECT B.BranchID, COUNT(*) as Number_of_Dependents
FROM BRANCH B, Employee E, Dependent D
WHERE B.BranchID = E.BranchID AND D.ESSN = E.SSN
GROUP BY B.BranchID
HAVING COUNT(*) > 1;
```

Query Result | Query Result 1

SQL | All Rows Fetched: 2 in 0.01 seconds

	BRANCHID	NUMBER_OF_DEPENDENTS
1	B002	2
2	B001	2

QUERY #3

SELECT THE EMPLOYEES WHOSE SALARIES ARE HIGHER THAN ALL OF THE EMPLOYEES IN BRANCH 1

```
SELECT E.SSN, E.FNAME, E.SALARY, E.BRANCHID
FROM EMPLOYEE E
WHERE E.SALARY > ALL
                 (SELECT SALARY
                  FROM EMPLOYEE
                  WHERE BranchID = 'B001');
```

Query Result

SQL | All Rows Fetched: 6 in 0.031 seconds

	SSN	FNAME	SALARY	BRANCHID
1	567890123	Laura	70000	B002
2	123123123	Grace	71000	B005
3	890123456	Olivia	72000	B003
4	456789012	Michael	90000	B002
5	789012345	William	95000	B003
6	898989899	Michael	97000	B005

QUERY #4

SELECT THE CUSTOMERS THAT HAVE LOAN ACCOUNTS

```
SELECT C.SSN, C.FNAME, C.LNAME, M.ACCOUNTNO
FROM CUSTOMER C, MANAGES M
WHERE C.SSN = M.CUSTOMERSSN AND M.ACCOUNTNO IN
              (SELECT ACCOUNTNO
               FROM ACCOUNT
               WHERE TYPE = 'Loan Account');
```

Query Result | Query Result 1 | Query Result 2 | Query Result 3

SQL | All Rows Fetched: 5 in 0.016 seconds

	SSN	FNAME	LNAME	ACCOUNTNO
1	111222333	David	Black	A1004
2	444555666	Alice	Brown	A1005
3	777888999	Bob	Green	A1006
4	888999666	Charlie	White	A1007
5	444555777	Eve	Pink	A1008

CONCLUDING REMARKS

Overall, this project experience was challenging, comprehensive, and rewarding. Through the design and implementation of the bank database, we cohesively learned the end-to-end steps from conceptual design, mapping schemas, and actually implementing our designs in a physical database. We then went back and revised our initial designs with our knowledge of functional dependencies and normal forms, presenting an even more robust and comprehensive learning experience. Of course, this came with its challenges as we were left with the authority to make several design decisions like mapping subclasses, adding appropriate attributes, and deciding on the best way to map relations in order to stay consistent with the 4 main design guidelines for robust databases. An example of our initial design challenges was that, initially, we made the mistake of considering dependents to be an attribute of employee, when dependents represented their own unique entity, albeit weak. From this, we truly attained a comprehensive understanding of entities in a mini world, and how proper schema database design and management requires them to possess their own table representation, otherwise this would violate higher normal form definitions.

Perhaps the most difficult, yet rewarding, aspect was deciding the differences between multivalued attributes, relationships between entities, and subclasses, and actually mapping this into a schema from the EER diagram, as we were often left with several choices, and we had to selectively choose the best one. While we initially considered entity design to be a simple task, we learned that the boundaries between some of these concepts were unclear at times. However, we were able to effectively create a proper design after deliberating and envisioning what the end product would look like. An important lesson we learned in this entire project was the balance between reducing redundancy, as we eventually learned through normalization, and mapping all the requirements appropriately. We learned this early on, where we were posed with the issue of creating several, unnecessary relations. We were able to consolidate and minimize our schema design by adhering closely to the rules we deemed necessary of our design and, eventually, going back and revising through knowledge of higher normalized forms. If we were to do this again, we would consider the highest normal form when designing the relations, in order to automatically adhere to best practices and prevent having to normalize after the schema design.

What we were most surprised to find, however, was that by adhering to the step by step process of entity-relational mapping, schema design, and physical database implementation, that we adhered to the four key principles of database designs very well, even before normalizing the relations. This truly proved how robust design is strictly correlated to logical real-world applications, as one requires the other. By going through the appropriate steps beforehand, we were not only able to create a robust design that met the requirements, but it was one that naturally was in a higher normal form. What we learned after, however, is that storing redundant location information could be error prone and, although attributes like zipcode, city, and state rarely change independently of each other, it was a wiser design choice to separate location information and generalize this information across relations that had this redundancy (through the created Locations relation).