|  |
| --- |
|  |

**LAB 3: Normalization Techniques and Recursive Joins in Telecom Databases**

|  |  |
| --- | --- |
| **Track** | Data & AI |
| **Role** | Software Development - Backend |
| **Level** | Level 2 |
| **Course** | Intermediate SQL |
| **Scope** | * Apply Normalization (1NF to 5NF) * Implement Recursive Joins in SQL |
| **Prerequisite** | * Basic SQL syntax knowledge * Understanding of SQL Joins * Familiarity with SQL Aggregation |

**Objective**

Learn to normalize a telecom database from 1NF to 5NF, ensuring data consistency and eliminating redundancy, and implement recursive joins to explore telecom customer and service relationships.

**Tasks**

1. Normalization techniques:
2. 1st Normal Form (1NF)

* Creating dataset which do not follow any normal rules.
* Creating Telecom\_1F table which separate table that specifically handles call-related data (i.e., Phone and CallMinutes).

1. 2st Normal Form (2NF)
   * Creating Customers\_2NF isolated customer information that is uniquely tied to CustomerID
   * Creating CallDetails table that specifically handles call-related data (i.e., Phone and CallMinutes)
   * Creating PaymentMethod table which manage payment methods.
2. 3st Normal Form (3NF)
   * Creating Plans table stores information about telecom plans, with each PlanID having one PlanName.
3. Boyce-Codd Normal Form (BCNF)
   * Creating CallLogs which links each phone number to the number of call minutes made
4. 4st Normal Form (4NF)
   1. CustomerPhones table that isolate the multi-valued relationship between CustomerID and Phone.
5. 5st Normal Form (5NF)
   1. Creating CustomerPhones Table which manages the relationship between customers and their phone numbers.
   2. Creating CustomerPayments Table which tracks the relationship between customers and their payment methods.
6. Implementing recursive joins:
   1. Find all sub-plans under a specific plan.
   2. Find the names of customers who have purchased a particular service plan and then recursively retrieve additional details about the service plan they purchased.
   3. Finds all calls made by a customer and the customers who received the calls, along with recursively exploring the connections.
   4. Retrieve all customers who are on a certain plan and recursively identify all customers who have upgraded to higher plans.
   5. Retrieve all customers who are on a certain plan and recursively identify all customers who have upgraded to higher plans.

**Outcomes**

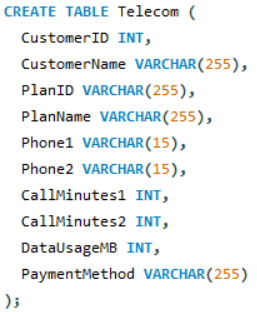
* Efficiently normalize databases from 1NF to 5NF.
* Manage multi-valued dependencies in telecom databases
* Enhance data integrity and minimize redundancy in telecom databases.
* Use recursive joins to query hierarchical telecom data

**Solution**

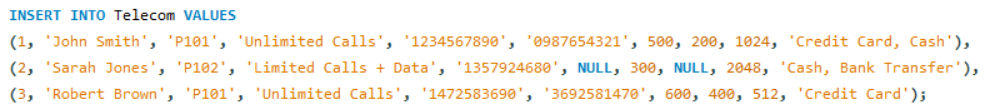
##### **Normalization techniques**:

1. 1st Normal Form (1NF)

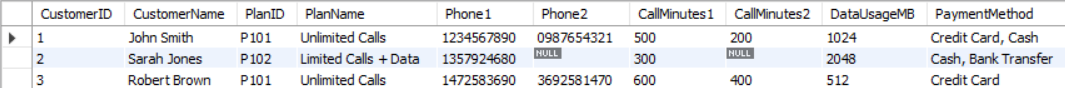
* 1NF ensures that all columns contain atomic (indivisible) values and each record is unique.
* Key Points:
  1. No repeating groups or arrays in columns.
  2. Each record must be unique (identified by a primary key).
  3. All entries in a column must be of the same data type.
* Creating Table



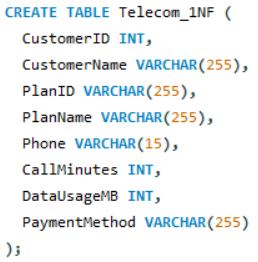
* Inserting data inside table.



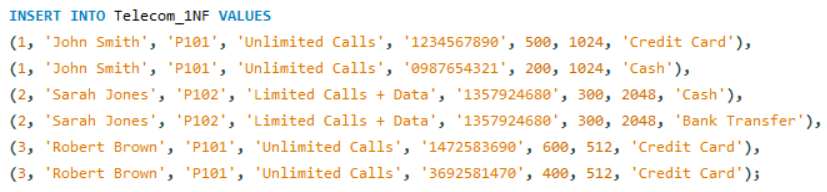
* Visualizing table



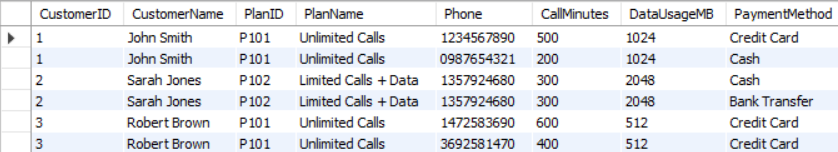
* Problem:
  + Table do not follow 1NF rule because Multi-valued attributes (Phone1, Phone2, PaymentMethod).
* To fix it:
  + We created new rows for each Phone and PaymentMethod, ensuring that each cell contains only one value.
* Recreate table which follow 1NF rule:



* Inserting data inside table:

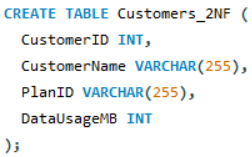


* Visualizing Table: The Table is flat, atomic table, where every cell holds only one piece of information. This makes the table easier to query and manipulate.

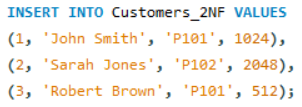


1. 2st Normal Form (2NF)

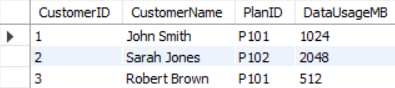
* 2NF ensures that each non-key attribute is fully dependent on the primary key, eliminating partial dependencies.
* Key Points:
  1. Achieved only if the table is in 1NF.
  2. No partial dependency (attributes depending on part of a composite primary key).
  3. Every non-key attribute must be fully dependent on the entire primary key.
* Problem:
  1. The CallMinutes and Phone columns are partially dependent on only part of the primary key (CustomerID, Phone). This violates 2NF, which requires that all non-key attributes depend on the entire primary key.
* Solution:
  + We will create separate tables for Customers, CallDetails, and PaymentMethods.
  1. Customers\_2F table: We isolated customer information that is uniquely tied to CustomerID (e.g., CustomerName, PlanID, DataUsageMB). This data does not depend on the phone number or call details.
     1. Creating Table:



* + 1. Inserting data:

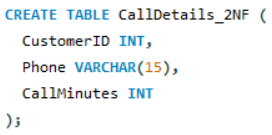


* + 1. Visualizing data: Holds all information related to customers, ensuring that there is no redundant phone or payment data associated with each customer.

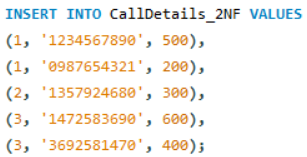


* 1. CallDetails Table: We created a table that specifically handles call-related data (i.e., Phone and CallMinutes). This is because the minutes are specific to the phone number used, so there is a partial dependency between Phone and CallMinutes.

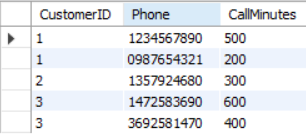
1. Creating table:



1. Inserting data:

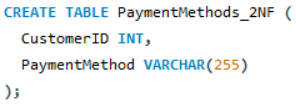


1. Visualizing data: It Tracks the phone numbers used by each customer and the associated call minutes. This eliminates the partial dependency by linking phones directly to call minutes.

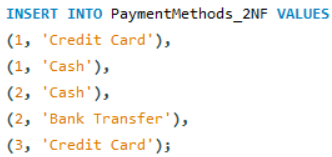


* 1. PaymentMethods Table: We moved payment methods to their own table because these are specific to the customer, but unrelated to calls or phone numbers.

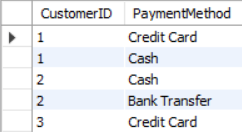
1. Creating table:



1. Inserting data:

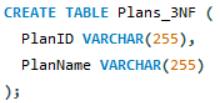


1. Visualizing data: It Tracks how customers make payments, with a separate record for each payment method (e.g., if a customer uses both credit card and cash, these are stored in separate rows).

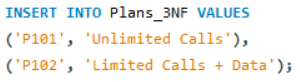


1. 3st Normal Form (3NF)

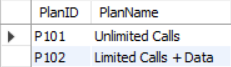
* 3NF ensures that no transitive dependency exists, meaning non-key attributes should depend only on the primary key.
* Key Points:
  + Achieved only if the table is in 2NF.
  + No transitive dependency (non-key attributes depending on other non-key attributes).
  + All attributes should be directly related to the primary key.
* Problem:
  + The PlanName is transitively dependent on PlanID, which is itself dependent on CustomerID. In 3NF, all non-key attributes must depend directly on the primary key and not through other non-key attributes.
* To fix it:
  + We created a new Plans table to hold the PlanID and PlanName values.
* Creating Plan Table:



* Inserting data in it:

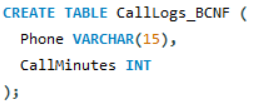


* Visualizing data: The Plans table stores information about telecom plans, with each PlanID having one PlanName. This ensures that plans are described once and can be linked to multiple customers without redundancy

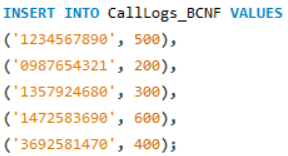


1. Boyce-Codd Normal Form (BCNF)

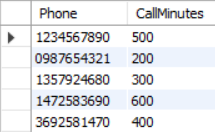
* BCNF is a stricter version of 3NF, ensuring that every functional dependency has a superkey.
* Key Points:
  + Achieved only if the table is in 3NF.
  + Every determinant (attribute that determines others) must be a superkey.
  + Further removes dependencies not covered by 3NF.
* Problem:
  + In the CallDetails table, the phone numbers (Phone) are not unique, and there's a dependency between Phone and CallMinutes that doesn’t involve the primary key (CustomerID).
* To fix it:
  + We created a new CallLogs table to store the relationship between Phone and CallMinutes directly.
* Creating CallLogs Table:



* Inserting data in it:

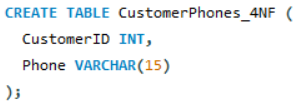


* Visualizing data: It links each phone number to the number of call minutes made. This resolves the non-trivial dependency and ensures that the relationship between phones and call minutes is handled independently from customer information.

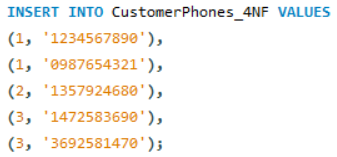


1. 4st Normal Form (4NF)

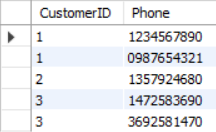
* 4NF removes multivalued dependencies, ensuring that records do not imply multiple independent relationships between attributes.
* Key Points:
  + Achieved only if the table is in BCNF.
  + Eliminate multivalued dependencies.
  + Prevents scenarios where a record contains two or more independent sets of values for a single key.
* Problem:
  + There is a multi-valued dependency between Phone and PaymentMethod. In 4NF, there cannot be multi-valued dependencies unless they are functionally dependent on the primary key.
* To fix it:
  + We created a separate CustomerPhones table to isolate the multi-valued relationship between CustomerID and Phone.
* Creating CustomerPhones Table:



* Inserting data in it:



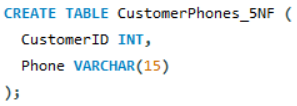
* Visualizing data: It manages the multi-valued dependency between customers and their phone numbers. This table tracks which phones each customer uses, allowing for multiple entries per customer without causing redundancy with other attributes (such as payment methods).



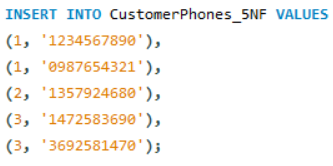
1. 5st Normal Form (5NF)

* 5NF ensures that every join dependency is implied by candidate keys, breaking down data into smaller tables to remove complex dependencies.
* Key Points:
  + Achieved only if the table is in 4NF.
  + Resolves join dependencies.
  + Ensures data can be recombined without redundancy or loss.
* Problem:
  + In the 4NF structure, there were join dependencies between CustomerID, Phone, and PaymentMethod.
* Solution:
  + We decomposed the table further into CustomerPhones and CustomerPayments to isolate the join dependencies.
  1. CustomerPhones Table: This table manages the relationship between customers and their phone numbers.

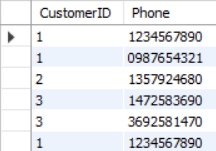
1. Creating CustomerPhones table:



1. Inserting data in it:

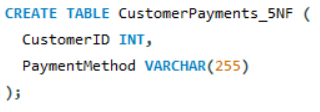


1. Visualizing data: It table eliminates the join dependency between CustomerID and Phone. By keeping only the CustomerID and Phone in this table, we ensure that the phone numbers are correctly linked to the customer without introducing redundant data when the tables are joined.

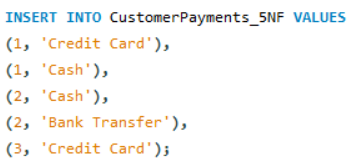


* 1. CustomerPayments Table: This table tracks the relationship between customers and their payment methods.

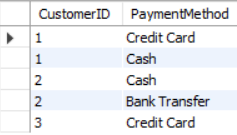
1. Creating CustomerPayments table:



1. Inserting data in it:

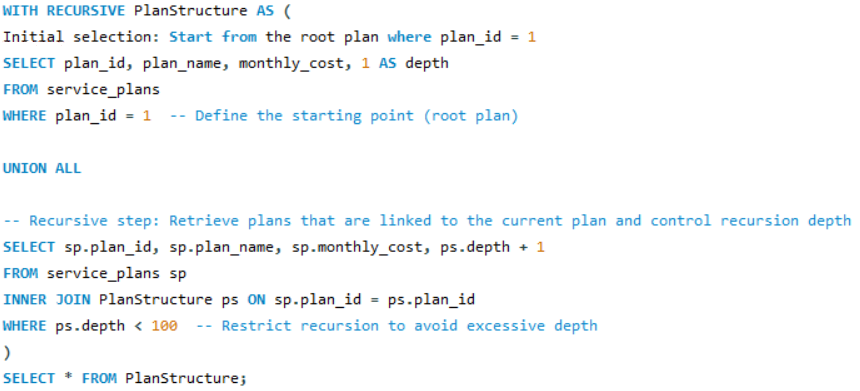


1. Visualizing data: This table eliminates the join dependency between CustomerID and PaymentMethod. By separating payment methods into their own table.

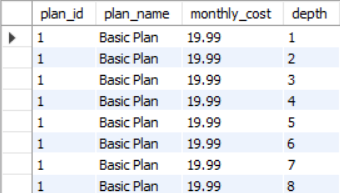


##### **Implementing recursive joins:**

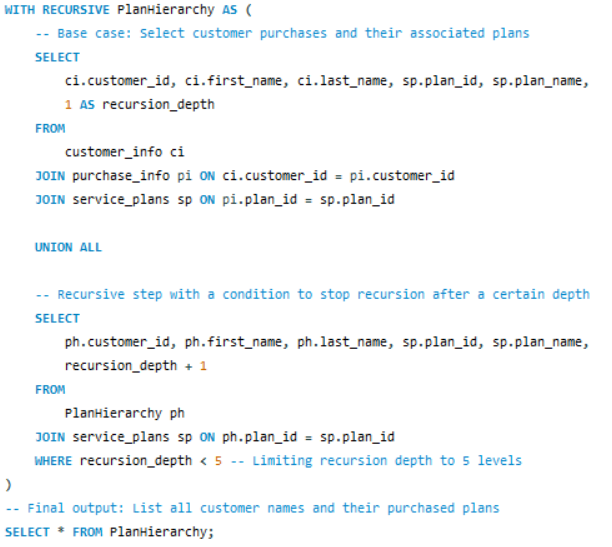
1. Find all sub-plans under a specific plan.



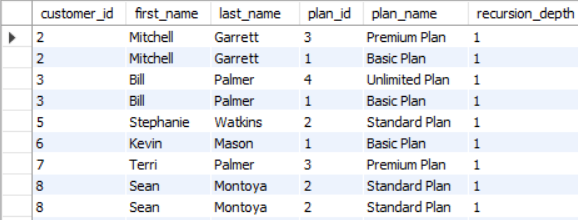
* Output:



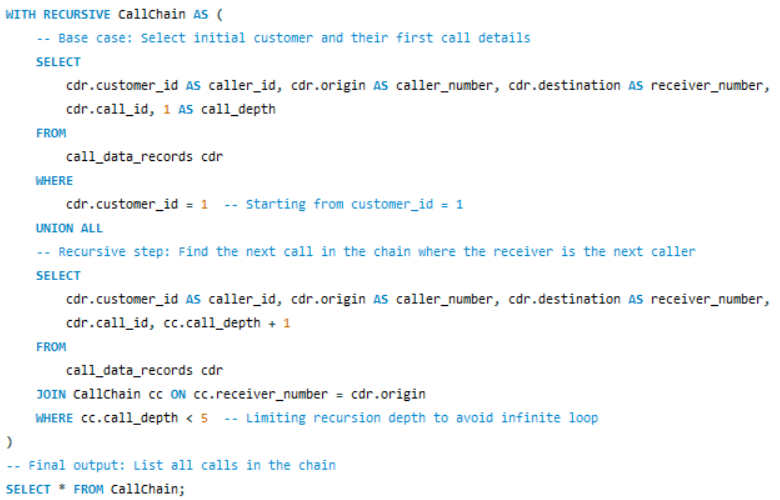
1. Find the names of customers who have purchased a particular service plan and then recursively retrieve additional details about the service plan they purchased.



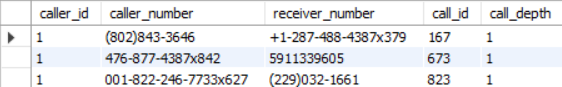
* Output:



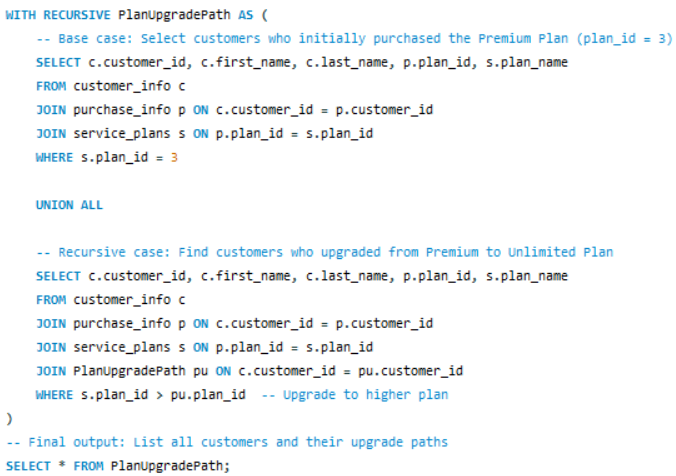
1. Finds all calls made by a customer and the customers who received the calls, along with recursively exploring the connections.



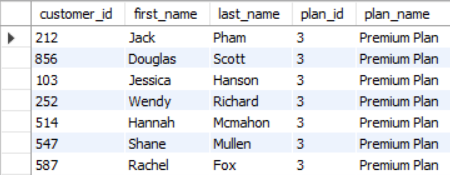
* Output:



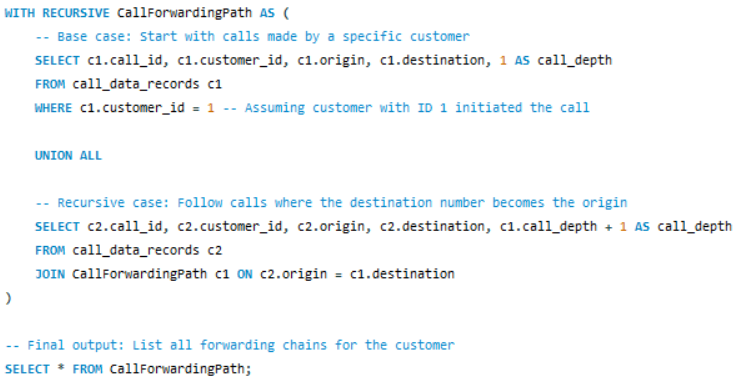
1. Retrieve all customers who are on a certain plan and recursively identify all customers who have upgraded to higher plans.



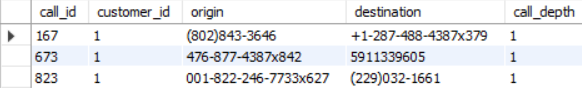
* Output:



1. Retrieve all customers who are on a certain plan and recursively identify all customers who have upgraded to higher plans.



* Output:



# (Additional) Task to do

1. **Apply partial normalization by decomposing** a table into 1NF and 2NF. For example, create a table that violates 1NF by having repeating groups and then normalize it to 1NF. After that, further break the table into 2NF by eliminating partial dependencies.
2. **Create a recursive query** to find the hierarchy of customers who refer each other in a referral system. Implement recursive joins to retrieve all customers who referred others and display the hierarchy in levels.
3. **Create a table that violates 3NF with transitive dependencies**. Normalize the table to 3NF by creating a new table to remove the transitive dependency. Verify that all non-key attributes depend only on the primary key.
4. **Implement a recursive join** to display all calls made within a telecom network where a customer called another customer who in turn called someone else. Use a recursive query to trace the call chain.
5. **Normalize a table to BCNF (Boyce-Codd Normal Form)** by eliminating remaining dependencies that do not involve a superkey. Show the table before and after normalization to BCNF.