**CRM Database Documentation**

This document provides a comprehensive overview of the crm database schema, including its tables, relationships, stored procedures, and functions. The database is designed to manage customer relationships, track leads, customers, campaigns, products, and employee interactions.

**1. Database Structure (Tables)**

The crm database is built with several interconnected tables, each serving a specific purpose in managing CRM data.

**person Table**

* **Purpose:** Stores generic personal information that can be linked to employees, leads, or customers.
* **Columns:**
  + personID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for each person.
  + name (VARCHAR(255)): Full name of the person.
  + email (VARCHAR(255)): Email address of the person.
  + phone (VARCHAR(50)): Phone number of the person.
  + dateCreated (DATETIME): Timestamp when the record was created.
  + dateUpdated (DATETIME): Timestamp when the record was last updated.

**employee Table**

* **Purpose:** Stores details about employees, linked to a person record.
* **Columns:**
  + empID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for each employee.
  + personID (INT, FOREIGN KEY): Links to the person table.
  + role (VARCHAR(100)): The employee's role (e.g., 'Sales Manager', 'Support Specialist').
  + dateHired (DATETIME): Date the employee was hired.
  + dateTerminated (DATETIME): Date the employee was terminated (NULL if active).
* **Foreign Keys:**
  + personID references person(personID) (ON DELETE CASCADE).

**addressType Table**

* **Purpose:** Defines types of addresses (e.g., Home, Work, Billing).
* **Columns:**
  + typeID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for address type.
  + typeName (VARCHAR(100)): Name of the address type.

**address Table**

* **Purpose:** Stores address details, linked to a person and an addressType.
* **Columns:**
  + addressID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for each address.
  + personID (INT, FOREIGN KEY): Links to the person table.
  + zip (VARCHAR(20)): Zip code.
  + city (VARCHAR(100)): City.
  + state (VARCHAR(100)): State.
  + country (VARCHAR(100)): Country.
  + typeID (INT, FOREIGN KEY): Links to the addressType table.
* **Foreign Keys:**
  + personID references person(personID) (ON DELETE CASCADE).
  + typeID references addressType(typeID) (ON DELETE CASCADE).

**campaign Table**

* **Purpose:** Stores information about marketing campaigns.
* **Columns:**
  + campaignID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for each campaign.
  + name (VARCHAR(255)): Name of the campaign.
  + startDate (DATETIME): Start date of the campaign.
  + endDate (DATETIME): End date of the campaign.
  + budget (DECIMAL(10,2)): Budget allocated for the campaign.

**source Table**

* **Purpose:** Defines the origin of leads, linked to a campaign.
* **Columns:**
  + sourceID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for each source.
  + campaignID (INT, FOREIGN KEY): Links to the campaign table.
  + sourceName (VARCHAR(255)): Name of the source (e.g., 'Google Ads', 'Facebook Ads').
  + sourceDesc (VARCHAR(1000)): Description of the source.
* **Foreign Keys:**
  + campaignID references campaign(campaignID) (ON DELETE CASCADE).

**status Table**

* **Purpose:** Defines the possible statuses for leads (e.g., 'New Lead', 'Contacted', 'Converted').
* **Columns:**
  + statusID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for each status.
  + statusName (VARCHAR(100)): Name of the status.
  + statusDesc (VARCHAR(1000)): Description of the status.

**leads Table**

* **Purpose:** Stores information about potential customers (leads).
* **Columns:**
  + leadID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for each lead.
  + sourceID (INT, FOREIGN KEY): Links to the source table (origin of the lead).
  + statusID (INT, FOREIGN KEY): Links to the status table (current status of the lead).
  + empID (INT, FOREIGN KEY): Links to the employee table (assigned employee).
  + personID (INT, FOREIGN KEY): Links to the person table (personal details of the lead).
  + dateCreated (DATETIME): Date the lead was created.
  + notes (VARCHAR(1000)): Any relevant notes about the lead.
* **Foreign Keys:**
  + sourceID references source(sourceID) (ON DELETE CASCADE).
  + statusID references status(statusID) (ON DELETE CASCADE).
  + empID references employee(empID) (ON DELETE CASCADE).
  + personID references person(personID) (ON DELETE CASCADE).

**customer Table**

* **Purpose:** Stores information about converted leads who are now customers.
* **Columns:**
  + custID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for each customer.
  + leadID (INT, UNIQUE, FOREIGN KEY): Links to the leads table (the lead from which this customer was converted). UNIQUE ensures a lead can only become one customer record.
  + personID (INT, FOREIGN KEY): Links to the person table (personal details of the customer).
  + dateCreated (DATETIME): Date the customer record was created.
  + dateUpdated (DATETIME): Date the customer record was last updated.
* **Foreign Keys:**
  + leadID references leads(leadID) (ON DELETE CASCADE).
  + personID references person(personID) (ON DELETE CASCADE).

**contactMethod Table**

* **Purpose:** Defines methods of contact (e.g., Email, Phone, SMS).
* **Columns:**
  + methodID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for contact method.
  + methodName (VARCHAR(100)): Name of the contact method.

**customerPreferences Table**

* **Purpose:** Stores specific contact and product preferences for a customer.
* **Columns:**
  + custPrefID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for customer preference.
  + custID (INT, FOREIGN KEY): Links to the customer table.
  + contactMethodID (INT, FOREIGN KEY): Links to the contactMethod table (preferred contact method).
  + pLanguage (VARCHAR(50)): Preferred language.
  + pContactTime (VARCHAR(100)): Preferred time for contact.
  + pBudget (DECIMAL(10,2)): Preferred budget for products/services.
  + dateUpdated (DATETIME): Timestamp when preferences were last updated.
* **Foreign Keys:**
  + custID references customer(custID) (ON DELETE CASCADE).
  + contactMethodID references contactMethod(methodID) (ON DELETE CASCADE).

**product Table**

* **Purpose:** Stores information about products or services offered.
* **Columns:**
  + productID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for each product.
  + productName (VARCHAR(255)): Name of the product.
  + productDesc (VARCHAR(1000)): Description of the product.

**customerProductInterest Table**

* **Purpose:** Links customers to products they are interested in.
* **Columns:**
  + custID (INT, FOREIGN KEY): Links to the customer table.
  + productID (INT, FOREIGN KEY): Links to the product table.
* **Primary Key:**
  + (custID, productID): Composite primary key ensuring a customer can only have one entry per product.
* **Foreign Keys:**
  + custID references customer(custID) (ON DELETE CASCADE).
  + productID references product(productID) (ON DELETE CASCADE).

**interactionChannel Table**

* **Purpose:** Defines channels through which interactions occur (e.g., Email, Phone Call, Chat).
* **Columns:**
  + channelID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for interaction channel.
  + channelName (VARCHAR(100)): Name of the channel.

**interaction Table**

* **Purpose:** Logs interactions between employees and customers.
* **Columns:**
  + interactionID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for each interaction.
  + empID (INT, FOREIGN KEY): Links to the employee table (employee who logged the interaction).
  + custID (INT, FOREIGN KEY): Links to the customer table (customer involved in the interaction).
  + notes (VARCHAR(1000)): Details of the interaction.
  + channelID (INT, FOREIGN KEY): Links to the interactionChannel table (channel used for interaction).
  + interactionDate (DATETIME): Date and time of the interaction.
* **Foreign Keys:**
  + empID references employee(empID) (ON DELETE CASCADE).
  + custID references customer(custID) (ON DELETE CASCADE).
  + channelID references interactionChannel(channelID) (ON DELETE CASCADE).

**followUp Table**

* **Purpose:** Tracks scheduled follow-up activities for leads.
* **Columns:**
  + followUpID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for each follow-up.
  + leadID (INT, FOREIGN KEY): Links to the leads table.
  + empID (INT, FOREIGN KEY): Links to the employee table (employee assigned the follow-up).
  + notes (VARCHAR(1000)): Notes about the follow-up.
  + followUpDate (DATETIME): Scheduled date and time for the follow-up.
  + completed (BOOLEAN, DEFAULT FALSE): Indicates if the follow-up has been completed.
* **Foreign Keys:**
  + leadID references leads(leadID) (ON DELETE CASCADE).
  + empID references employee(empID) (ON DELETE CASCADE).

**statusHistory Table**

* **Purpose:** Logs changes in a lead's status over time.
* **Columns:**
  + statusHistoryID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for each status change record.
  + leadID (INT, FOREIGN KEY): Links to the leads table.
  + statusID (INT, FOREIGN KEY): Links to the status table (new status).
  + changedByEmpID (INT, FOREIGN KEY): Links to the employee table (employee who changed the status).
  + changeDate (DATETIME): Date and time of the status change.
  + notes (VARCHAR(1000)): Any notes related to the status change.
* **Foreign Keys:**
  + leadID references leads(leadID) (ON DELETE CASCADE).
  + statusID references status(statusID) (ON DELETE CASCADE).
  + changedByEmpID references employee(empID) (ON DELETE CASCADE).

**admin Table**

* **Purpose:** Stores administrator credentials for application login.
* **Columns:**
  + adminID (INT, PRIMARY KEY, AUTO\_INCREMENT): Unique identifier for each admin user.
  + username (VARCHAR(255), NOT NULL, UNIQUE): Unique username for login.
  + password (VARCHAR(255), NOT NULL): Password for the admin user.
    - **SECURITY NOTE:** In a production environment, passwords should always be hashed (e.g., using bcrypt) and never stored in plain text.

**2. Relationships (Joins) and Common Queries**

The database tables are interconnected using foreign keys, allowing for complex queries that retrieve data from multiple tables.

**Retrieving Lead Information with Associated Details**

This query combines leads, person, employee, and status tables to get a comprehensive view of leads.

SELECT

l.leadID,

p.name AS LeadName,

p.email AS LeadEmail,

p.phone AS LeadPhone,

emp\_p.name AS AssignedEmployee,

s.statusName AS CurrentStatus,

l.dateCreated,

l.notes,

src.sourceName AS LeadSource,

camp.name AS CampaignName

FROM leads l

JOIN person p ON l.personID = p.personID

LEFT JOIN employee emp ON l.empID = emp.empID

LEFT JOIN person emp\_p ON emp.personID = emp\_p.personID -- Join to get employee's person details

JOIN status s ON l.statusID = s.statusID

LEFT JOIN source src ON l.sourceID = src.sourceID

LEFT JOIN campaign camp ON src.campaignID = camp.campaignID;

* **JOIN**: Used when you want to retrieve records only if there's a match in both tables (e.g., a lead must have a person record).
* **LEFT JOIN**: Used when you want to retrieve all records from the left table (e.g., leads) and matching records from the right table. If no match, the right side columns will be NULL. This is useful for optional relationships (e.g., a lead might not always have an assigned employee, or a source might not be linked to a campaign).

**Retrieving Customer Profile with Preferences and Interests**

This query combines customer, person, customerPreferences, contactMethod, customerProductInterest, and product tables.

SELECT

c.custID,

p.name AS CustomerName,

p.email AS CustomerEmail,

p.phone AS CustomerPhone,

cp.pLanguage AS PreferredLanguage,

cp.pContactTime AS PreferredContactTime,

cp.pBudget AS PreferredBudget,

cm.methodName AS PreferredContactMethod,

GROUP\_CONCAT(prod.productName SEPARATOR ', ') AS ProductInterests

FROM customer c

JOIN person p ON c.personID = p.personID

LEFT JOIN customerPreferences cp ON c.custID = cp.custID

LEFT JOIN contactMethod cm ON cp.contactMethodID = cm.methodID

LEFT JOIN customerProductInterest cpi ON c.custID = cpi.custID

LEFT JOIN product prod ON cpi.productID = prod.productID

GROUP BY c.custID, p.name, p.email, p.phone, cp.pLanguage, cp.pContactTime, cp.pBudget, cm.methodName;

* **GROUP\_CONCAT()**: An aggregate function used here to combine multiple product interests into a single comma-separated string for each customer.

**Subquery Example: Customers interested in 'Product A'**

This query uses a subquery to find customers who have an interest in a specific product.

SELECT

c.custID,

p.name AS CustomerName,

p.email

FROM customer c

JOIN person p ON c.personID = p.personID

WHERE c.custID IN (

SELECT custID

FROM customerProductInterest cpi

JOIN product prod ON cpi.productID = prod.productID

WHERE prod.productName = 'Product A'

);

* **IN operator with Subquery**: The inner query (SELECT custID FROM customerProductInterest cpi JOIN product prod ON cpi.productID = prod.productID WHERE prod.productName = 'Product A') returns a list of custIDs. The outer query then selects customers whose custID is present in this list.

**3. Stored Procedures**

Stored procedures encapsulate complex SQL logic, improve performance, and enhance security by abstracting direct table access. Below are examples of key procedures:

**authenticateAdmin**

* **Purpose:** Authenticates an administrator by checking their username and password.
* **Parameters:** p\_username (VARCHAR), p\_password (VARCHAR)
* **Logic:** Selects the username from the admin table where both username and password match.
* **Returns:** A row with adminID and username if valid.

**createLead (CREATE Example)**

* **Purpose:** Inserts a new lead record.
* **Parameters:** IN\_personID, IN\_sourceID, IN\_empID, IN\_statusID, IN\_notes
* **Logic:** Inserts a new row into the leads table with provided details and current timestamp.

**getLeads (GET Example)**

* **Purpose:** Retrieves lead records with optional filtering.
* **Parameters:** IN\_empID (Optional), IN\_statusID (Optional), IN\_startDate (Optional), IN\_endDate (Optional)
* **Logic:** Selects lead details, including person name, assigned employee name, status name, creation date, and notes, using LEFT JOIN and WHERE clauses for filtering.
* **Returns:** A result set of lead records.

**updateLead (UPDATE Example)**

* **Purpose:** Updates an existing lead's source, status, assigned employee, and notes.
* **Parameters:** IN\_leadID, IN\_newSourceID, IN\_newStatusID, IN\_newEmpID, IN\_newNotes
* **Logic:** Updates the specified columns for the given leadID.

**deleteLead (DELETE Example)**

* **Purpose:** Deletes a lead record and cascades deletions to related statusHistory and followUp records.
* **Parameters:** p\_leadID
* **Logic:** Deletes the row from the leads table.

**convertLeadToCustomer (Business Logic Example)**

* **Purpose:** Atomically converts a lead into a customer, updating lead status and logging the change.
* **Parameters:** p\_leadID, p\_changedByEmpID
* **Logic:**
  1. Starts a transaction.
  2. Gets the 'Converted' statusID.
  3. Retrieves personID from the lead.
  4. Updates lead's status to 'Converted'.
  5. Logs the status change in statusHistory.
  6. Creates a new record in the customer table.
  7. Commits the transaction.
* **Returns:** The newCustomerID of the newly created customer.

**4. Functions**

Functions in MySQL are routines that return a single value and can be used within SQL statements.

**getTotalLeads**

* **Purpose:** Returns the total count of leads in the database.
* **Parameters:** None.
* **Returns:** INT (total number of leads).
* **Logic:** Counts all rows in the leads table.

**getTotalCustomers**

* **Purpose:** Returns the total count of customers in the database.
* **Parameters:** None.
* **Returns:** INT (total number of customers).
* **Logic:** Counts all rows in the customer table.

**5. Python Application Overview**

The CRM application is built using Python with PySide6 for the graphical user interface and mysql.connector for database interaction.

**5.1 GUI Library: PySide6**

* **Role:** PySide6 is a Python binding for the Qt cross-platform application framework. It provides a comprehensive set of widgets for building desktop applications.
* **Key Concepts:**
  + **Widgets:** Fundamental building blocks of the UI (e.g., QMainWindow, QWidget, QLabel, QPushButton, QLineEdit, QTableWidget, QComboBox).
  + **Layouts:** Organize widgets within a window (QVBoxLayout, QHBoxLayout, QFormLayout). They handle the positioning and resizing of widgets.
  + **Signals and Slots:** Qt's powerful mechanism for communication between objects. A "signal" is emitted when an event occurs (e.g., a button is clicked), and a "slot" is a function that responds to that signal.
  + **QSS (Qt Style Sheets):** Used for styling the application's appearance, similar to CSS for web pages. This allows for a modern and consistent look and feel across the UI elements.

**5.2 Database Connector: mysql.connector**

* **Role:** This Python library provides the interface to connect to and interact with MySQL databases.
* **Key Functionality:**
  + **Connection Management:** Establishes and closes connections to the MySQL server.
  + **Cursor Objects:** Used to execute SQL queries and stored procedures.
  + **Data Fetching:** Retrieves results from queries, often in dictionary format for easy access by column name.
  + **Transaction Control:** Supports committing (saving changes) or rolling back (undoing changes) database transactions to maintain data integrity.

**5.3 Application Structure (main.py)**

The Python application follows a modular design, separating concerns into different classes:

* **CRMApp(QMainWindow):**
  + **Purpose:** The main application window.
  + **Structure:** Contains a sidebar for navigation and a QStackedWidget to display different content pages. It's the central hub that manages the overall application flow.
  + **Initialization:** Handles the initial database connection and displays the LoginDialog before showing the main dashboard.
* **LoginDialog(QDialog):**
  + **Purpose:** Provides the user authentication interface.
  + **Structure:** A simple dialog with QLineEdit for username/password and a QPushButton for login.
  + **Interaction:** Communicates with DatabaseManager to verify credentials. If successful, it accept()s, allowing CRMApp to proceed.
* **DatabaseManager:**
  + **Purpose:** Centralizes all database interaction logic. This class acts as an abstraction layer between the UI and the MySQL database.
  + **Methods:** Contains wrapper methods (e.g., getLeads(), createLead(), authenticateAdmin()) that call the corresponding MySQL stored procedures. This keeps database-specific code out of the UI classes.
  + **Error Handling:** Includes try-except blocks to catch database errors and display user-friendly messages.
* **Page Classes (e.g., DashboardPage, LeadsPage, CustomersPage)**:
  + **Purpose:** Represent different sections or modules of the CRM application.
  + **Structure:** Each page typically inherits from QWidget and uses a QVBoxLayout or QHBoxLayout to arrange its content. They often contain QTableWidget to display lists of data, QLineEdit and QComboBox for filters, and QPushButton for actions.
  + **Data Loading:** Implement methods (e.g., load\_data(), load\_leads()) that call DatabaseManager methods to fetch and display relevant information.
  + **Event Handling:** Connects UI element signals (e.g., button clicks, combo box selections) to specific slots (methods) within the page class to handle user interactions.
* **Dialog Classes (e.g., LeadFormDialog, CustomerProfileDialog, CustomerProductInterestDialog)**:
  + **Purpose:** Provide dedicated forms or detailed views for specific data entry or display tasks.
  + **Structure:** Inherit from QDialog and often use QFormLayout for structured input fields.
  + **Interaction:** Typically opened modally (blocking interaction with the parent window) and return QDialog.Accepted or QDialog.Rejected based on user action (e.g., Save or Cancel). They also interact with DatabaseManager to save or retrieve data.

This modular approach ensures that the application is maintainable, scalable, and that the database logic is clearly separated from the user interface.