**Term Project Final Report**

**INFO 5707.401 – Data Modelling for Information Professionals**

**University of North Texas**

**Department of Information Science**

**Instructor: Dr. Tozammel Hossain**

**Title:**

**Insurance Claim Management System**

Adithya Suresh – 11817067

Contents

[Project Description: 3](#_Toc184577463)

[Scope: 3](#_Toc184577464)

[Project Requirements: 4](#_Toc184577465)

[Database Requirements: 4](#_Toc184577466)

[User Requirements: 5](#_Toc184577467)

[Business Rules: 5](#_Toc184577468)

[Entity Relationship Diagram (ERD): 6](#_Toc184577469)

[Data Dictionary: 7](#_Toc184577470)

[Basic Concepts of the Data Dictionary: 8](#_Toc184577471)

[1) Entity Generation and Data Entry for Table Claim: 10](#_Toc184577472)

[2) Entity Generation and Data Entry for Table Policy: 12](#_Toc184577473)

[3) Entity Generation and Data Entry for Table PolicyCoverage: 14](#_Toc184577474)

[4) Entity Generation and Data Entry for Table CoveredPeople: 15](#_Toc184577475)

[5) Entity Generation and Data Entry for Table VehicleDetails: 17](#_Toc184577476)

[6) Entity Generation and Data Entry for Table History: 18](#_Toc184577477)

[7) Entity Generation and Data Entry for Table Employer: 20](#_Toc184577478)

[8) Entity Generation and Data Entry for Table Exposure: 21](#_Toc184577479)

[9) Entity Generation and Data Entry for Table FNOL: 23](#_Toc184577480)

[10) Entity Generation and Data Entry for Table Financial: 24](#_Toc184577481)

[11) Entity Generation and Data Entry for Table LossType: 26](#_Toc184577482)

[Data Retrieval and Simple Reports: 28](#_Toc184577483)

[Data Analysis: 28](#_Toc184577484)

[Analysis -1 : 28](#_Toc184577485)

[Analysis – 2: 29](#_Toc184577486)

[Analysis -3: 30](#_Toc184577487)

[Analysis 4: 31](#_Toc184577488)

[Analysis – 5: 32](#_Toc184577489)

[Analysis – 6: 33](#_Toc184577490)

[Analysis – 7: 34](#_Toc184577491)

[Analysis – 8: 35](#_Toc184577492)

[Analysis-9: 36](#_Toc184577493)

[Analysis – 10: 37](#_Toc184577494)

## **Project Description:**

This Insurance claim management system is a sophisticated and comprehensive solution aimed at optimizing the way insurance companies manage their claims. It is designed while keeping in mind the scalability factor as well as maintaining efficiency for larger inventories. Claim management system addresses critical problems that insurance providers face in their day-to-day work. By utilizing modern database technologies, the management system effortlessly integrates the claim data with the transactional data and policy details.

A unified database containing all required data at one place would drastically improve the efficiency of the database while maintaining integrity. This project uses that method to eliminate any data silos and make sure that the data being worked upon is consistent with accurate data. From the initial phase of the claim which is the First notice of the loss (FNOL) to the end final settlement of the loss, this CMS provides a robust framework to analyze and manage the claims.

**Objective:**

The Claim Management System represents a sophisticated and all-encompassing solution aimed at transforming the methods employed by insurance firms in managing claims. Through the incorporation of processes for claim initiation, monitoring, and resolution, the system intends to eradicate inefficiencies while improving operational precision. It enables customer service representatives to document claim information with accuracy, thereby lessening errors and diminishing processing duration. Automation and integrated data consolidate client satisfaction, faster claims settlement, and allow transparency in all respects. The system integrates various aspects relating to the claims management process for policy information, financial transactions involved, and loss-related information so as to provide a single-window framework enabling efficiency as well as consistency.

It also enables teamwork within the personnel in an easy-to-use interface, with real-time data access and controlled function-based access, thereby enhancing the overall processing efficiency and reliability. Precisely, it is targeted at ensuring the delivery of high-quality service, easing administrative tasks, and ensuring the growth of any concerned organization.

## **Scope:**

The Claim Management System aims at changing the trend in the insurance claim process, handling all aspects of claim management right from origination to settlement. The system will allow the creation of unique claim numbers assigned for every incident reported, usually referred to as First Notice of Loss-FNOL. This would be a very important identifier in relationship to a policy table that holds everything about the customer's policy and coverage. Each documented loss gets classified by certain placeholders, known as exposures, that define the coverage type applied on the loss. These exposures then link with the financials table, which captures the payment information to allow for correct and timely reimbursements. This system also provides communication between the customer and the employees. If a loss occurs, the customer calls a service representative. The service representative builds the claim details in the system. The system keeps track of everything the representative does to create, modify, or remove records of the actions to the claim history table so that there is accountability and auditing. The system shall also have role-based access-for example, only relevant persons shall have access to confidential information.

Besides, the Claims Management System makes the process of insurance smooth by allowing real-time status tracking, detailed reporting, and automation of workflow processes. It improves coordination between various departments due to the integration of policy information, claim documents, and financial records on a single platform. It also helps in efficient resource allocation by giving the necessary insight into claim trends and other payment patterns. In enabling faster and more accurate processing, therefore, the system satisfies regulatory requirements by enhancing the totality of customer experiences and positions insurance companies to customers as reliable yet prompt responders to customer needs.

## **Project Requirements:**

* **Operating System:** Windows
* **Database:** MySQL
* **Applications:** MYSQL Workbench, draw.io, Microsoft Word, Microsoft Excel

## **Database Requirements:**

The following is the list of the various data tables that will be used.

* **Claim Table**
* **Policy Table**
* **PolicyCoverage Table**
* **CoveredPeople Table**
* **VehicleDetails Table**
* **History Table**
* **Employer Table**
* **Exposure Table**
* **FNOL Table**
* **Financial Table**
* **LossType Table**
* **LossState Table**

## **User Requirements:**

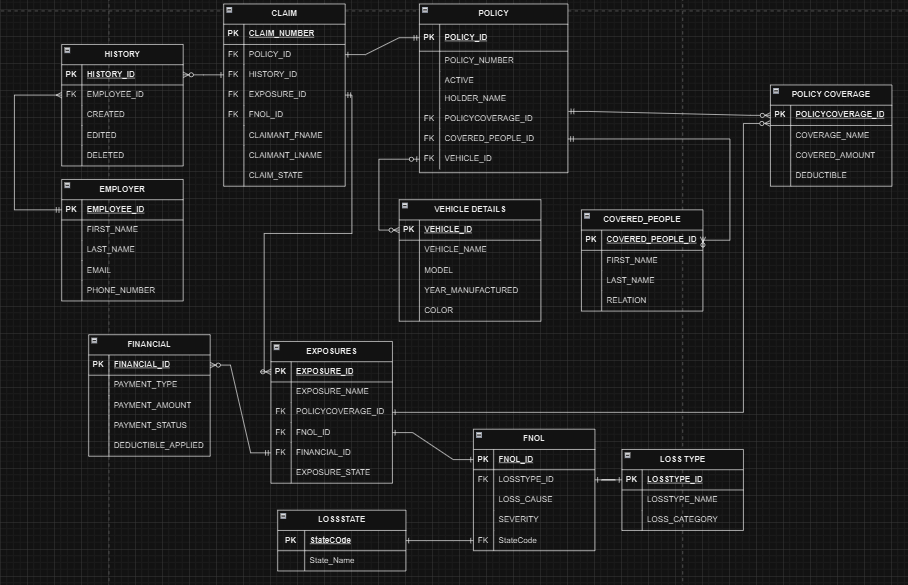
1. **Claim Registration:** To be able to open up a new claim in the customer service representative, which automatically flags it as the First Notice of Loss-unique number-identified for each claim.
2. **FNOL Information:** Comprehensive details, encompassing the date of loss and the corresponding policy number, must be submitted when initiating a claim.
3. **Vehicle Damages and Casualties Information:** Vehicle damage claims require recording at least one injury detail or one vehicle damage detail.
4. **Making Exposures:** The exposures should be calculated based on the amount of insurance carried when a policy was underwritten.
5. **Editing Exposure:** Agents should be able to fill in the system the following underwriting exposure information: amount claimed, coverage type, and details of loss.
6. **Payment Logging:** All transactions, including all changes, need to be tracked in the system with no loss of data, but also with a connection to related claims.
7. **Payment Constraining:** The compensation can be made only for the exposure created, while the amount compensated must be less than or equal to the compensation amount.
8. **Insurance Coverage Verification:** Agents should be allowed to add insurance directly from the policy while creating exposures.
9. **History Event Limitations:** The users cannot change all the past events recorded in the system because they are kept for future reference.
10. **Active policy requirement:** A claim can only be filed when the loss arises under an active policy.
11. **Claim Summary and Status:** Agents must be given access to summaries of claims and exposure regarding the status that could be open, closed, and pending payments.

## **Business Rules:**

1. **Identifying Distinctive Claims:** A different number identifies each claim for efficient tracking and management.
2. **Compulsory Claim Information:** It requires basic information about the policy number, claim amount, and incident date to file the claim.
3. **Policy Coverage Verification:** First, each claim must be verified to determine if the claim is within policy coverage.
4. **Claims Status Tracking:** The system will show different claim statuses at different stages, like "Pending," "In Review," "Approved," or "Rejected," depending on the processing stage.
5. **Role-Based Access Control:** Access to view or edit a claim is based on the user's role, be it a customer service representative or a manager.
6. **Auditing Logging:** The system keeps a record of all edits to the claims information, including the editor and time of edit.
7. **Duplicate Claims Avoidance:** Mechanisms are in place to avoid duplicate claims filing for the same incidence.
8. **Document Storage and Linking:** The relevant documents for incident reports and medical treatments are kept on secured files coupled with the case.
9. **Regulatory Compliance:** The system follows the data protection policy, hence the safety and privacy of customer information.
10. **Payment Authorization Control:** Payments require approval from designated staff members, ensuring they align with policy details before disbursement.

## **Entity Relationship Diagram (ERD):**

The following diagram represents the entity relationship in the database.



The ERD for the Claim Management System includes those 12 entities, capturing various aspects of the claim process, including but not limited to claims, policies, exposures, financial transactions, history, and FNOL-first notice of loss-which would be involved in providing functional claim tracking and processing within the insurance context.

**Components of the ERM:**

|  |  |
| --- | --- |
| TABLES | TABLE RELATIONSHIPS |
| Employer ↔ History | One to Many (1:M) |
| Claim ↔ History | One to Many (1:M) |
| Claim ↔ Policy | One to One (1:1) |
| Claim ↔ Exposure | One to Many (1:M) |
| Policy ↔ Policy Coverage | One to Many (1:M) |
| Policy ↔Vehicle Details | One to Many (1:M) |
| Policy ↔ Covered People | One to Many (1:M) |
| Exposure ↔ Policy Coverage | One to Many (1:M) |
| Exposure ↔ Financial | One to Many (1:M) |
| Exposure ↔ FNOL | One to One (1:1) |
| FNOL ↔ Loss State | One to One (1:1) |
| FNOL ↔ Loss Type | One to One (1:1) |

## **Data Dictionary:**

The data dictionary plays a vital role in the design of the database: it acts as a central store whereby information related to data elements within a system is stored. It enhances uniformity, clarity, and standardization of all data assets by defining important attributes like data names, types, formats, and ranges. The mapping of tables and their relationships enhances traceability with the data dictionary and helps in the interaction whereby good communication among team members is established.

### **Basic Concepts of the Data Dictionary:**

* Ensures there is no duplication of data within the warehouse data management system to avoid errors and inconsistencies.
* Performs complete documentation of naming convention, data formats, and their usage within the system for consistency in interpretation.
* Aligns team members regarding data collection, management, and interpretation for effective collaboration and avoidance of misunderstandings.
* Enforces standard rules in creating, changing, and utilizing data, making sure quality and integrity are maintained in the system.
* It can enhance the analysis process in defined definitions and relationships, thus aiding in the quicker discovery of business insights.

**(Note: Please double-click on the below file to view the data dictionary.)**

****

**Entity Generation and Data Entry:**

**Overview:**

The Claim Management System data model has been designed with the following tables, related through the relationships, namely: Claim, Policy, PolicyCoverage, CoveredPeople, VehicleDetails, History, Employer, Exposure, FNOL, Financial, and LossType. While implementing such tables, foreign key constraints should be defined to ensure that the data integrity is maintained and wrong data cannot be inserted in the tables. A foreign key value in the child table must match a primary key value of the associated parent table. This ensures consistency and reliability of information in the system. Resultantly, postings on the tables are done in an organized manner such that errors are avoided and successful execution of the SQL queries are allowed.

#### **To create a database:**

CREATE DATABASE IF NOT EXISTS CLAIM\_MANAGEMENT\_SYSTEM;

### **1) Entity Generation and Data Entry for Table Claim:**

**Query:**

CREATE TABLE Claim

(

claim\_Number VARCHAR(30),

policy\_id VARCHAR(30),

history\_id VARCHAR(30),

exposure\_id VARCHAR(30),

FNOL\_ID VARCHAR(30),

claimant\_Fname VARCHAR(255),

claimant\_Lname VARCHAR(255),

claim\_State VARCHAR(10),

PRIMARY KEY(claim\_Number),

FOREIGN KEY(policy\_id) REFERENCES Policy(policy\_id) ON DELETE CASCADE,

FOREIGN KEY(history\_id) REFERENCES History(history\_id) ON DELETE CASCADE,

FOREIGN KEY(exposure\_id) REFERENCES Exposure(exposure\_id) ON DELETE CASCADE,

FOREIGN KEY(FNOL\_ID) REFERENCES FNOL(FNOL\_ID) ON DELETE CASCADE

);

**Explanation:**

The Claim table will store all the information about the insurance claim and provides the required association among a policy, exposure, history, and First Notice of Loss record. This table picks up all the basic information of the claim, like the claim number, which is the unique identifier of the claim, the first and last name of the claimant, and the current status, whether the case is open or closed. The Claim table follows a very organized structure, which can enforce tracking every aspect of a claim by ensuring data integrity and hassle-free access through foreign keys referencing other tables that will be used for the comprehensive claim processing.

**Command Used to Load the data from csv file to Claim table**

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ Claim.csv'

INTO TABLE Claim

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

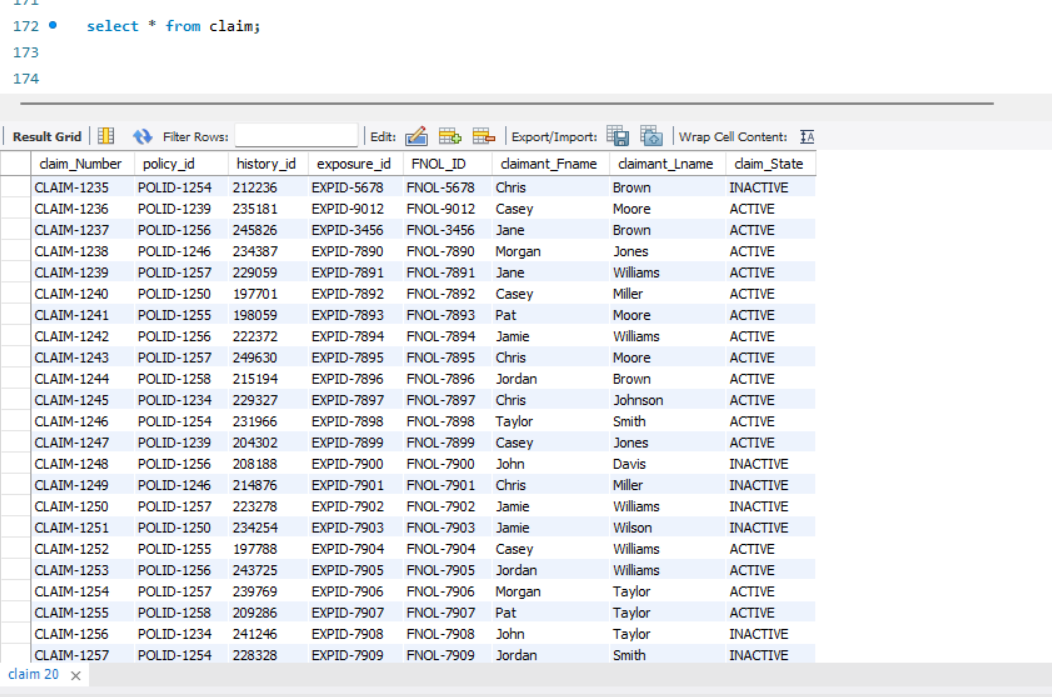
LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

**Data Ingestion:**

The claim table data will be loaded from the CSV file using the LOAD DATA INFILE command, which introduces speed and accuracy in claiming data.

**Screenshot:**



### **2) Entity Generation and Data Entry for Table Policy:**

**Query:**

CREATE TABLE Policy

(

policy\_id VARCHAR(30),

policyNumber VARCHAR(30),

active varchar(10),

policyHolderName VARCHAR(255),

policyCoverage\_id VARCHAR(30),

coveredPeople\_id VARCHAR(30),

vehicle\_id VARCHAR(30),

PRIMARY KEY(policy\_id),

FOREIGN KEY(policyCoverage\_id) REFERENCES policyCoverage(policyCoverage\_id) ON DELETE CASCADE,

FOREIGN KEY(coveredPeople\_id) REFERENCES coveredPeople(coveredPeople\_id) ON DELETE CASCADE,

FOREIGN KEY(vehicle\_id) REFERENCES vehicleDetails(vehicle\_id) ON DELETE CASCADE

);

**Explanation:**

The Policy table is an inclusive table that comprises all data regarding insurance policies, having a unique policy ID, policy number, policyholder's name, and the status of the policy. It connects further, through foreign key relations, to other related entities like policy coverage, covered individuals, and vehicles covered under that policy.

The policyholder can strongly represent his or her insured property and persons by maintaining these relationships enabled through the Policy table. This will help in accurate identification of eligibility and verification of coverage of events of claims and thus is indispensable in determining the range of protection granted under each policy.

**Command Used to Load the data from csv file to Policy table**

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ Policy.csv'

INTO TABLE Policy

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

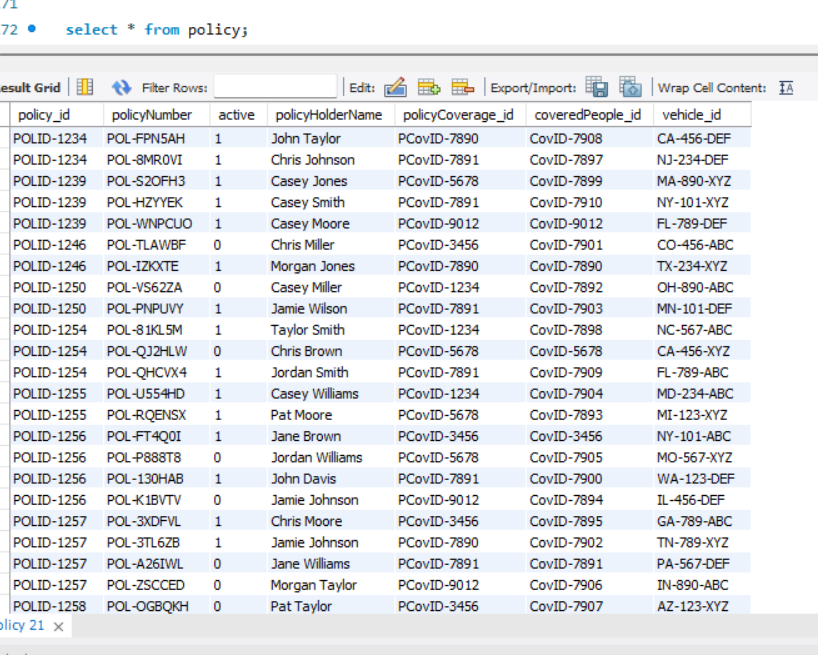
LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

**Data Ingestion:**

The CSV file will import all the details about the policy directly into the database via a bulk data entry LOAD DATA INFILE command for the Policy table without manual errors.

**Screenshot:**



### **3) Entity Generation and Data Entry for Table PolicyCoverage:**

**Query:**

CREATE TABLE PolicyCoverage

(

policyCoverage\_id VARCHAR(30),

coverageName VARCHAR(30),

coveredAmount FLOAT NOT NULL,

deductible FLOAT,

PRIMARY KEY(policyCoverage\_id)

);

**Explanation:**

PolicyCoverage table focuses on the detailed coverage entailed in an insurance policy. It captures a unique identifier for each category of coverage, the naming of coverage type-say, collision or liability-the monetary amount covered under the policy, and any applicable deductibles. This table will make sure the monetary aspect of the policy is well recorded for clarity on what is included in the cover and allows the claim assessment to be precisely made.

It plays a critical role in the claims process in that it sets the financial limit of policy, payment.

**Command Used to Load the data from csv file to PolicyCoverage table**

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ PolicyCoverage.csv'

INTO TABLE PolicyCoverage

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

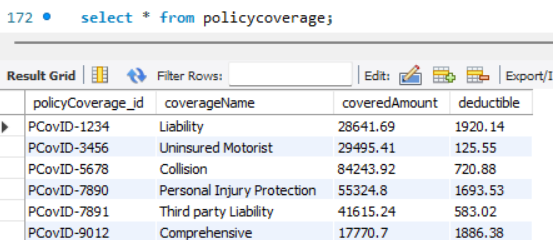
LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

**Data Ingestion:**

This command uses the LOAD DATA INFILE command, which imports the information of coverages for the PolicyCoverage table efficiently via a CSV file.

**Screenshot:**



### **4) Entity Generation and Data Entry for Table CoveredPeople:**

**Query:**

CREATE TABLE CoveredPeople

(

coveredPeople\_id VARCHAR(30),

firstName VARCHAR(255),

lastName VARCHAR(255),

relation VARCHAR(255),

PRIMARY KEY(coveredPeople\_id)

);

**Explanation:**

The CoveredPeople table contains information about the people covered under a specific policy, first and last name, and their relation to the policyholder, like spouse or dependent. Each record has a unique key that identifies the covered peoples, 'Covered People ID', which links to the 'Policy' table. This helps the insurance company pinpoint exactly the people that have been covered under one policy and manage the claims related to these people.

**Command Used to Load the data from csv file to CoveredPeople table**

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ CoveredPeople.csv'

INTO TABLE CoveredPeople

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

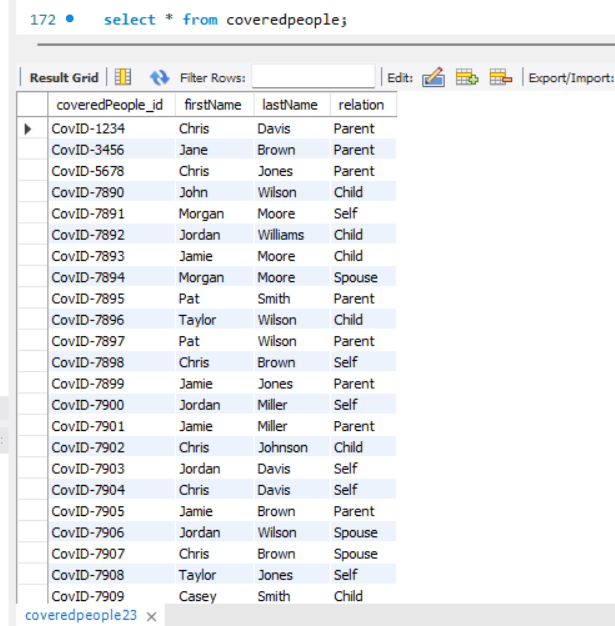
LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

**Data Ingestion:**

Importing data into the CoveredPeople table is achieved using a CSV file through the LOAD DATA INFILE command, which allows the structured inputting of people's data.

**Screenshot:**



### **5) Entity Generation and Data Entry for Table VehicleDetails:**

**Query:**

CREATE TABLE VehicleDetails

(

vehicle\_id VARCHAR(30),

vehicleName VARCHAR(30),

model VARCHAR(20),

yearManufactured DATE,

color VARCHAR(20),

PRIMARY KEY(vehicle\_id)

);

**Explanation:**

The VehicleDetails table summarizes the information on the vehicle covered under a particular policy. It links up the vehicle name, model, and year of production with its color through the vehicle identification number. It is essential for any claim about a policy that covers automotive, as it allows for proper identification and valuation of the vehicle.

**Command Used to Load the data from csv file to VehicleDetails table**

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ VehicleDetails.csv'

INTO TABLE VehicleDetails

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

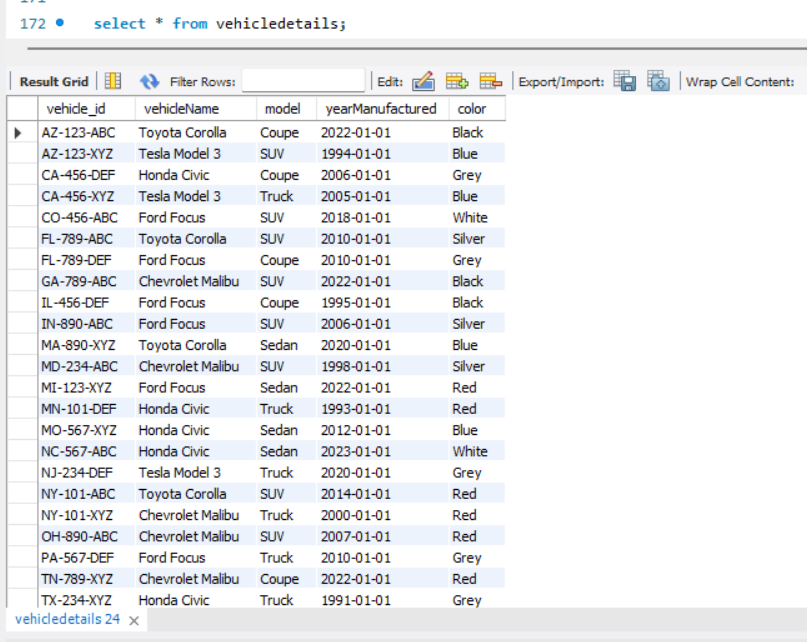
LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

**Data Ingestion:**

The data for vehicles are populated to the VehicleDetails table using the import of comma-separated values via the LOAD DATA INFILE command.

**Screenshot:**



### **6) Entity Generation and Data Entry for Table History:**

**Query:**

CREATE TABLE History

(

history\_id VARCHAR(30),

employee\_id VARCHAR(30),

created DATE,

edited DATE,

deleted DATE,

PRIMARY KEY(history\_id),

FOREIGN KEY(employee\_id) REFERENCES employer(employee\_id) ON DELETE CASCADE

);

**Explanation:**

The History table will track all significant changes made to either claims or policies, including creating, editing, and delete date/time stamps. It will also identify what employee has made a certain change by referring back to the Employer table. This provides full audit trail capabilities that can be very useful when managing accountability regarding policies and claims.

**Command Used to Load the data from csv file to History table**

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ History.csv'

INTO TABLE History

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

**Data Ingestion:**

Records in the History table are populated from a CSV file using the LOAD DATA INFILE command.

**Screenshot:**



### **7) Entity Generation and Data Entry for Table Employer:**

**Query:**

CREATE TABLE Employer

(

employee\_id VARCHAR(30),

firstName VARCHAR(255),

lastName VARCHAR(255),

email VARCHAR(255),

phoneNumber INT,

PRIMARY KEY(employee\_id),

UNIQUE KEY(email),

UNIQUE KEY(phoneNumber)

);

**Explanation:**

The Employer table stores much information about the employees who contribute to the claim management process. It contains Employee ID, Firstname, Lastname, Email address, and Telephone number. Unique constraints for both email and phone numbers prevent duplication entries. This table allows this application to implement the required role-based access and identify which employee is responsible for various functions throughout this application.

**Command Used to Load the data from csv file to Employer table**

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ Employer.csv'

INTO TABLE Employer

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

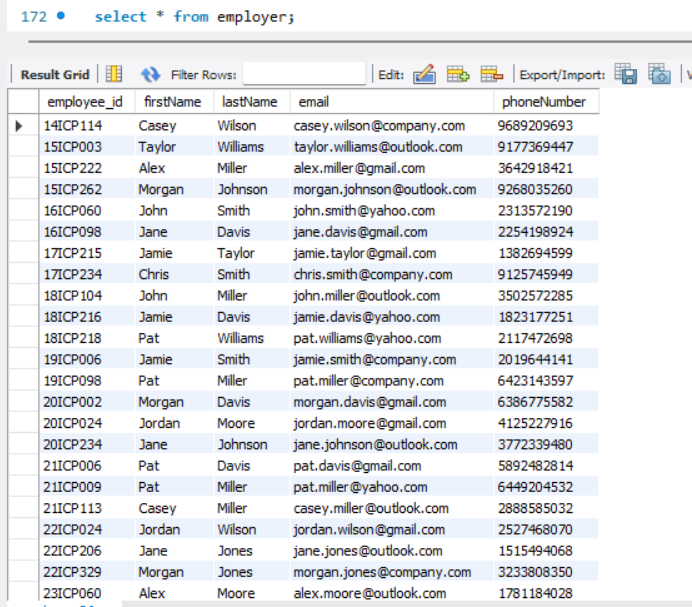
LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

**Data Ingestion:**

The LOAD DATA INFILE command will import the data about employees in the Employer table from a CSV.

**Screenshot:**



### **8) Entity Generation and Data Entry for Table Exposure:**

**Query:**

CREATE TABLE Exposure

(

exposure\_id VARCHAR(30),

exposureName VARCHAR(255),

policyCoverage\_id VARCHAR(30),

FNOL\_id VARCHAR(30),

financials\_id VARCHAR(30),

exposureState VARCHAR(10),

PRIMARY KEY(exposure\_id),

FOREIGN KEY(policyCoverage\_id) REFERENCES PolicyCoverage(policyCoverage\_id) ON DELETE CASCADE,

FOREIGN KEY(FNOL\_id) REFERENCES FNOL(FNOL\_id) ON DELETE CASCADE,

FOREIGN KEY(financials\_id) REFERENCES Financial(financial\_id) ON DELETE CASCADE

);

**Explanation:**

The Exposure table holds particular risk or losses to be insured under a policy; examples include the name of the exposure, current state of exposure, and links to financials, FNOL, and policy coverage. It helps in monitoring each exposure involved in claims to make sure all possible risks are mitigated and assessed properly during the course of a claim.

**Command Used to Load the data from csv file to Exposure table**

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ Exposure.csv'

INTO TABLE Exposure

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

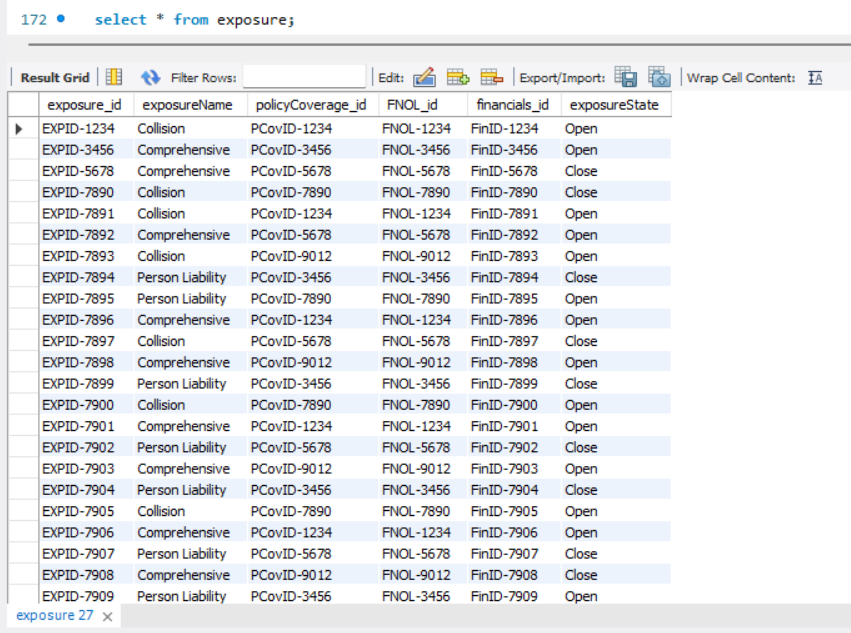
LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

**Data Ingestion:**

The information in the Exposure table will be added by importing the data from a CSV ﬁle using the LOAD DATA INFILE command.

**Screenshot:**



### **9) Entity Generation and Data Entry for Table FNOL:**

**Query:**

CREATE TABLE FNOL

(

FNOL\_id VARCHAR(30),

lossType\_id VARCHAR(30),

lossCause VARCHAR(255),

severity VARCHAR(10),

PRIMARY KEY(FNOL\_id),

FOREIGN KEY(lossType\_id) REFERENCES LossType(lossType\_id) ON DELETE CASCADE

);

**Explanation:**

First Notice of Loss Table First reporting of a loss or incident that may involve a claim includes type of loss, cause of loss, and the degree of severity. The table would relate back to the LossType table as it would categorize incidents so that claims could begin processing as quickly as possible.

**Command Used to Load the data from csv file to FNOL table**

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ FNOL.csv'

INTO TABLE FNOL

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

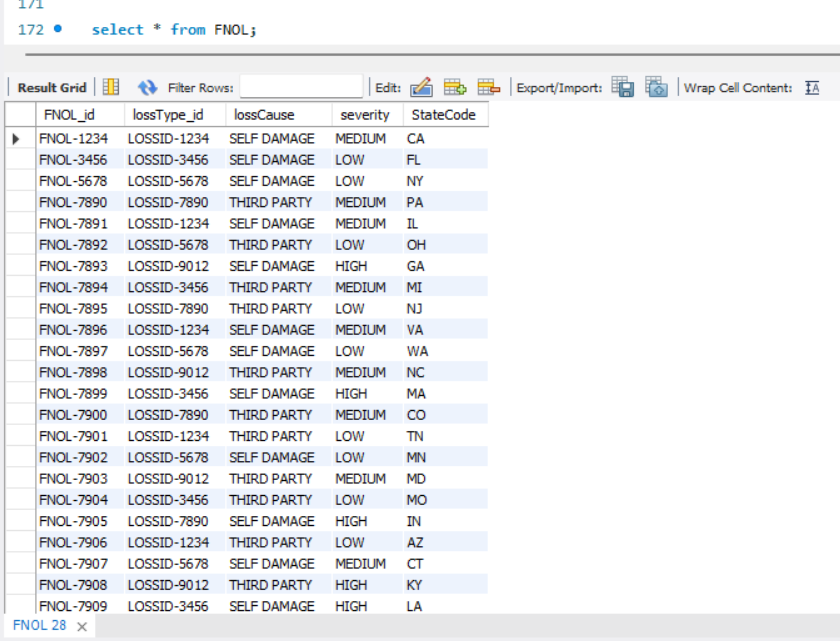
LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

**Data Ingestion:**

The information for the FNOL table is fed to the database through the LOAD DATA INFILE command via the CSV file.

**Screenshot:**



### **10) Entity Generation and Data Entry for Table Financial:**

**Query:**

CREATE TABLE Financial

(

financial\_id VARCHAR(30),

paymentType VARCHAR(255),

paymentAmmount FLOAT NOT NULL,

paymentStatus VARCHAR(10),

deductibleApplied FLOAT,

PRIMARY KEY(financial\_id)

);

**Explanation:**

The Financial table oversees all payment-related information linked to claims. It monitors the method of payment (for instance, cash or check), the amount disbursed, the status of the payment, and any deductibles that have been applied. This table guarantees clarity regarding the financial dimensions of claims processing and offers a comprehensive analysis of monetary transactions.

**Command Used to Load the data from csv file to Financial table**

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ Financial.csv'

INTO TABLE Financial

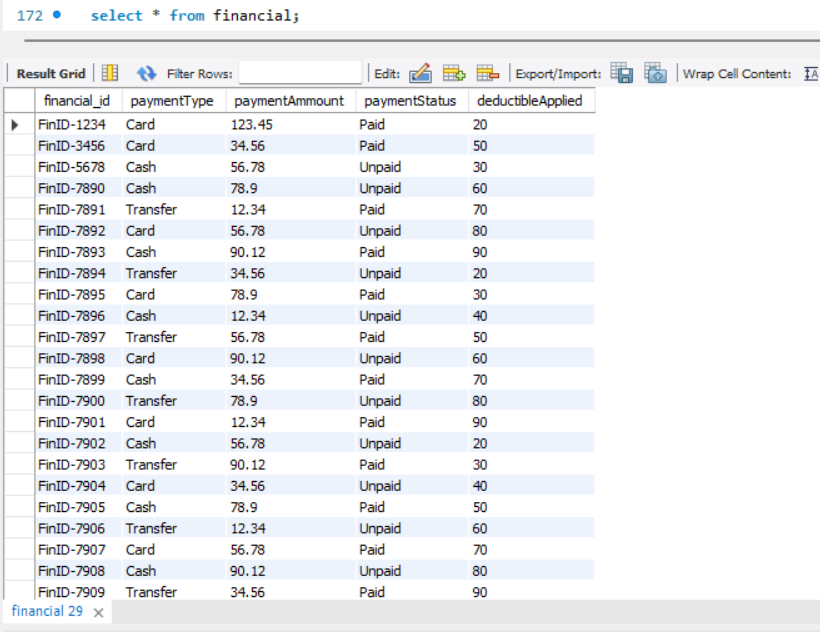
FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

**Data Ingestion:**

Loading Data into the financial table is populated with financial information from the CSV file by using a command called LOAD DATA INFILE.  
  
**Screenshot:**

### **11) Entity Generation and Data Entry for Table LossType:**

**Query:**

CREATE TABLE LossType

(

lossType\_id VARCHAR(30),

lossTypeName VARCHAR(255),

lossCategory VARCHAR(255),

PRIMARY KEY(lossType\_id)

);

**Explanation:**

The LossType table categorizes various types of losses that an insurance policy might cover. The table will hold fields for the loss type name and category, making the classification and analysis of claims more orderly. The table will provide uniformity in loss classification and facilitate detailed reporting and analytics.

**Command Used to Load the data from csv file to LossType table**

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ LossType.csv'

INTO TABLE LossType

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

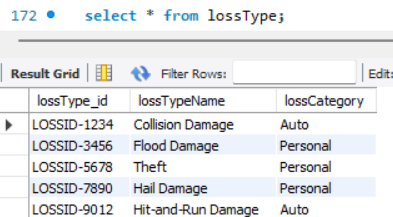
LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

**Data Ingestion:**

Importing data into a database from the CSV file will be accomplished through using the LOAD DATA INFILE command.

**Screenshot:**



**12) Entity Generation and Data Entry for Table LossType:**

**Query**:  
CREATE TABLE LossState

(

StateCode VARCHAR(5),

StateName VARCHAR(40),

PRIMARY KEY(StateCode)

);

**Explanation:**

The loss state table contains the state code and their respective names, this will used as a foreign key in the FNOL table which helps the user to understand where the loss happened.  
  
**Command Used to Load the data from csv file to LossType table**

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ LossState.csv'

INTO TABLE LossState

FIELDS TERMINATED BY ','

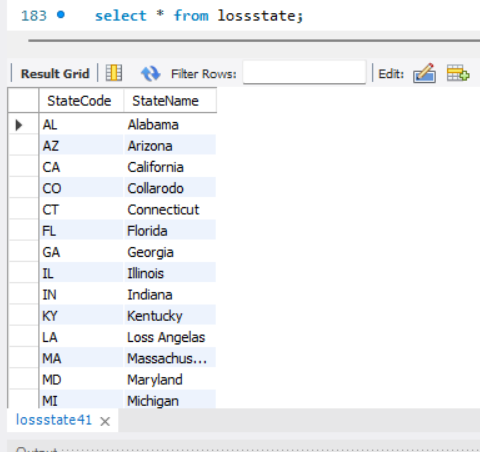
ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 ROWS;

**Data Ingestion:**

Importing data into a database from the CSV file will be accomplished through using the LOAD DATA INFILE command.

**Screenshot:**

## 

## **Data Retrieval and Simple Reports:**

**Overview:**

Information extraction from raw data into insightful information is an essential part of doing business today, as it enables the organization to use the extracted information toward making informed decisions. This project involves formulating queries with the intent of stakeholders deriving necessary insights to improve processes and optimize all operations. Results from the analysis will allow management to decrease operation costs, increase profitability, and ultimately drive high customer satisfaction. This will involve explaining what the purpose of every query is, breaking down each simple SQL query, and the complete SQL SELECT statement. Attached also are screenshots of the result set of the queries to show the findings and help interpret data.

## **Data Analysis:**

The advanced analysis gives more information on your database by explaining the relation, trends, and irregularities within the tables. The type of policies that have the highest claims per occurrence, top employees who handle claims, and causes of loss by state. Loss costs are reviewed by type of loss and state, whereas claims are viewed for duplication, severity, and lack of related exposures or policy information. The analysis also identifies the relative unused resources including the orphan policies with zero corresponding claims or financial values. Incorporating vehicle and policyholder data, the queries give ideas of operation performance and recommendations for improvement.

### **Analysis -1 :**

This analysis helps us to learn the total number of the claims done by the policy holders. Which in turn helps us to identify the pattern and reassess the policy premium.

**Query:**

SELECT p.policy\_id, COUNT(c.policy\_id) AS claim\_count

FROM policy p

LEFT JOIN claim c ON p.policy\_id = c.policy\_id

GROUP BY p.policy\_id

LIMIT 0, 1000;

**Explanation:**

This query identifies the policies with more than one claim associated with them and displays them.

**Result and screenshot**:

A screenshot of a computer

Description automatically generated

### **Analysis – 2:**

To find out the employees who processed most number of claims. This helps us to evaluate the employees and reward the best.

**Query:**

SELECT e.employee\_id, e.firstName, e.lastName, COUNT(c.history\_id) AS claims\_processed

FROM employer e

JOIN history h ON e.employee\_id = h.employee\_id

JOIN claim c ON h.history\_id = c.history\_id

GROUP BY e.employee\_id

ORDER BY claims\_processed DESC

LIMIT 5;

**Explanation:**

This query joins the tables of employers, claim and history. It identifies the employees while grouping them based on the employee.id. The order by helps us to determine the employee’s in a descending order.

**Result and Screenshot:**

A screenshot of a computer

Description automatically generated

### **Analysis -3:**

Understanding the claims which have missing exposure details are key to assess where the data is being entered with negligence. This helps us to resolve any problems that might arise in the future.

**Query:**

SELECT c.claim\_Number, c.policy\_id, c.exposure\_id

FROM claim c

LEFT JOIN exposure e ON c.exposure\_id = e.exposure\_id

WHERE e.exposure\_id IS NULL;

**Explanation:**

This query selects the claim number, policy ID, and associated exposure ID. This is possible by left joining the exposure onto the claims table. While also keeping a conditional statement WHERE. Which helps us to identify null IDs.

**Result and Screenshot:**

A screenshot of a computer

Description automatically generated

### **Analysis 4:**

To analyze the distribution of claims based on the severity of the incident.

**Query:**

SELECT f.severity, COUNT(c.claim\_Number) AS claim\_count

FROM fnol f

JOIN claim c ON f.FNOL\_id = c.FNOL\_ID

GROUP BY f.severity

ORDER BY claim\_count DESC;

**Explanation:**  
The query helps to determine the total count of claims in order to compare their distribution by the level of claimed severities. It initially, extracts severity levels from the ‘fnol’ table and tallys up related claims from the ‘claim’ table. It employs the `FNOL\_id” to link the tables so that related records can be easily identified. After that, the results are partitioned with key `f.severity` to guarantee that each severity level will be aggregated with its claims. Lastly, it provides an interesting quantization as the severity level is arranged in descending order of the count of the claim. These identified criteria enable the prioritization of the claims to be addressed depending on their level of harm.

**Result and Screenshot:**

A screenshot of a computer

Description automatically generated

### **Analysis – 5:**

To find active policies and associated claims for them.

**Query:**

SELECT p.policyNumber,

p.policyHolderName,

COUNT(c.claim\_Number) AS claim\_count

FROM policy p

LEFT JOIN claim c ON p.policy\_id = c.policy\_id

WHERE p.active = 1

GROUP BY p.policyNumber, p.policyHolderName

ORDER BY claim\_count DESC;

**Explanation:**

This query treats active policies i.e., policies whose status is shown by `p.active = 1` and counts for claims. Using the policy and claim tables it connects them using `policy\_id` and select results grouped by policy details such as `policyNumber` and `policyHolderName` Most policies have been ordered depending on the number of claims a specific policy has received to display those with high activity in claims.

**Result and Screenshot:**A screenshot of a computer

Description automatically generated

### **Analysis – 6:**

This query describes the inactive policies that still have some claims yet to be resolved. Which helps the user to keep up with the stuff and solve the issues.

**Query:**

SELECT p.policy\_id,

p.policyNumber,

COUNT(c.claim\_Number) AS claim\_count

FROM policy p

JOIN claim c ON p.policy\_id = c.policy\_id

WHERE p.active = 0

GROUP BY p.policy\_id, p.policyNumber

ORDER BY claim\_count DESC;

**Explanation:**

This query involves policies that still have claims and are inactive (`p.active = 0`). It combines occurrences of `policy` and `claim` based on the policy ID, aggregates findings based on policy information (policy ID, policy number), quantifies accompanying claims, and ranks results based on the claim count, sorting for policies that experienced the greatest number of claims.

**Result and Screenshot:**

A screenshot of a computer

Description automatically generated

### **Analysis – 7:**

Retrieving specific policies and claims by specific policyholders.

**Query:**

SELECT c.claim\_Number,

p.policyHolderName,

c.claimant\_Fname,

c.claimant\_Lname,

pc.coveredAmount

FROM claim c

JOIN policy p ON c.policy\_id = p.policy\_id

JOIN policycoverage pc ON p.policyCoverage\_id = pc.policyCoverage\_id

WHERE p.policyHolderName IN ('John Doe', 'Jane Smith');

**Explanation:**

In this query, the user will get related claims that are associated with particular policyholders (policyHolderName). Combining policy and policycoverage tables by the policy\_id and policyCoverage\_id, the table is provided with the claim informations along with the claimant name and the coveredAmount.

**Result and Screenshot:**

A screenshot of a computer

Description automatically generated

### **Analysis – 8:**

Find the most used payment type of the policyholders.

**Query:**

SELECT paymentType,

COUNT(financial\_id) AS transaction\_count

FROM financial

GROUP BY paymentType

ORDER BY transaction\_count DESC

LIMIT 5;

**Explanation:**

This query returns the most frequent payment types in the ‘financial’ table in the data base. It sorts data based on ‘paymentType’, calculates number of transactions as ‘financial\_id’, sorts result in descending order of transaction count and limits it to 5 records.

**Result and Screenshot:**

A screenshot of a computer

Description automatically generated

### **Analysis-9:**

Examining the trends between the payment status and the payment types.

**Query:**

SELECT f.paymentStatus,

f.paymentType,

COUNT(f.financial\_id) AS total\_transactions,

SUM(CAST(f.paymentAmmount AS DECIMAL(10, 2))) AS total\_amount

FROM financial f

GROUP BY f.paymentStatus, f.paymentType

ORDER BY f.paymentStatus, total\_transactions DESC;

**Explanation:**

The payment trends are evaluated by categorizing payment status and payment type from the `financial` table. It uses the concept of counts, sum, ordering of the result based on `paymentStatus` and transaction count to show trends on different locations based on Payment type and payment status. Responsive of the payment types and statuses discovered by trends in the transaction number and overall payment amounts.

**Result and Screenshot:**

A screenshot of a computer

Description automatically generated

### **Analysis – 10:**

To analyze and keep note of the actions done in months and years.

**Query:**

SELECT YEAR(created) AS activity\_year,

MONTH(created) AS activity\_month,

COUNT(history\_id) AS total\_activities

FROM history

WHERE created IS NOT NULL

GROUP BY activity\_year, activity\_month

ORDER BY activity\_year DESC, activity\_month DESC;

**Explanation:**

This query generates activity trends by the number of records developed yearly and monthly within the history table. They are arranged based on the year and month of action and filtered for the null creation date, arranged in descending order by year and then by month.

**Result and Screenshot:**

A screenshot of a computer

Description automatically generated