**SQL**

**In**

**Claims**

**Processing System**

**for**

**Insurance**

In the realm of insurance, data analysis plays a pivotal role in optimizing operations, enhancing risk management, and ensuring financial stability. The following SQL queries have been designed to provide actionable insights into insurance claims processing and policyholder management.

**QUERY 1**

Applicants Table

CREATE TABLE applicants (

applicant\_id INT PRIMARY KEY,

name VARCHAR(100),

age INT,

occupation VARCHAR(50),

annual\_income DECIMAL(10, 2),

status VARCHAR(20),

join\_date DATE

);

The applicants table stores information about individuals applying for a program or service. Each row represents an applicant, with key details captured such as:

* applicant\_id: A unique identifier for each applicant (primary key).
* name: The applicant's full name.
* age: Applicant's age in years.
* occupation: Current job title or profession of the applicant.
* annual\_income: The annual income of the applicant, stored as a decimal value.
* status: The current status of the applicant in the process (e.g., "Pending", "Approved").
* join\_date: The date the applicant submitted their application.

This table is designed for easy tracking and management of applicants, with all key details efficiently structured.

-- Policies Table

CREATE TABLE policies (

policy\_id INT PRIMARY KEY,

policyholder\_id INT,

policy\_type VARCHAR(50),

premium\_amount DECIMAL(10, 2),

status VARCHAR(20),

start\_date DATE,

renewal\_date DATE,

FOREIGN KEY (policyholder\_id) REFERENCES applicants(applicant\_id)

);

The policies table stores information about insurance policies held by applicants. Each row represents an individual policy with relevant details, and it's linked to the applicants table through a foreign key (policyholder\_id), ensuring that every policy is associated with a valid applicant.

* **policy\_id**: A unique identifier for each policy (primary key).
* **policyholder\_id**: Refers to the applicant holding the policy, linking to the applicants table.
* **policy\_type**: The type of policy (e.g., life insurance, health insurance).
* **premium\_amount**: The premium to be paid for the policy, stored as a decimal value.
* **status**: The current status of the policy (e.g., "Active", "Expired").
* **start\_date**: The date when the policy coverage began.
* **renewal\_date**: The date when the policy is set for renewal.

This table allows efficient management of insurance policies and ensures each policy is correctly assigned to a policyholder.

-- Claims Table

CREATE TABLE claims (

claim\_id INT PRIMARY KEY,

policy\_id INT,

claim\_amount DECIMAL(10, 2),

claim\_date DATE,

claim\_status VARCHAR(20),

rejection\_reason VARCHAR(255),

payment\_date DATE,

FOREIGN KEY (policy\_id) REFERENCES policies(policy\_id)

);

The claims table tracks insurance claims made by policyholders. Each row represents an individual claim linked to a policy via the foreign key policy\_id, which references the policies table. This structure ensures that each claim is associated with a valid policy.

* **claim\_id**: A unique identifier for each claim (primary key).
* **policy\_id**: Refers to the policy associated with the claim, linking to the policies table.
* **claim\_amount**: The monetary value of the claim.
* **claim\_date**: The date when the claim was submitted.
* **claim\_status**: The current status of the claim (e.g., "Pending", "Approved", "Rejected").
* **rejection\_reason**: If the claim is rejected, this field stores the reason for rejection.
* **payment\_date**: The date when the claim payment was processed, if applicable.

This table helps efficiently manage and track insurance claims, their status, and outcomes.

-- Underwriters Table

CREATE TABLE underwriters (

underwriter\_id INT PRIMARY KEY,

name VARCHAR(100)

);

The underwriters table stores basic information about underwriters involved in managing insurance policies. Each row represents a unique underwriter.

* **underwriter\_id**: A unique identifier for each underwriter (primary key).
* **name**: The full name of the underwriter (up to 100 characters).

This table provides a simple structure for keeping track of underwriters, which can later be linked to policies or claims if needed.

**QUERY 2**

Each row contains basic details such as the applicant's name, occupation, income, status, and the date they joined.

-- Insert Sample Applicants

INSERT INTO applicants VALUES

(1, 'John Doe', 45, 'Engineer', 75000, 'Active', '2020-04-15'),

(2, 'Jane Smith', 30, 'Teacher', 50000, 'Active', '2021-06-20'),

(3, 'Bob Johnson', 65, 'Retired', 30000, 'Inactive', '2019-03-10');

-- Insert Sample Policies

INSERT INTO policies VALUES

(101, 1, 'Health', 5000, 'Active', '2021-01-01', '2022-01-01'),

(102, 2, 'Auto', 3000, 'Active', '2021-06-01', '2022-06-01'),

(103, 3, 'Life', 7000, 'Inactive', '2018-05-01', '2019-05-01');

-- Insert Sample Claims

INSERT INTO claims VALUES

(1001, 101, 1500, '2023-07-15', 'Approved', NULL, '2023-07-20'),

(1002, 102, 2000, '2023-08-10', 'Rejected', 'Insufficient documentation', NULL),

(1003, 101, 500, '2023-09-05', 'Pending', NULL, NULL);

-- Insert Sample Underwriters

INSERT INTO underwriters VALUES

(1, 'Alice Brown'),

(2, 'David Wilson');

**QUERY 3 -** Risk Assessment Report

SELECT

applicant\_id,

name,

age,

occupation,

annual\_income,

CASE

WHEN age > 60 THEN 'High Risk'

WHEN occupation IN ('Construction Worker', 'Pilot') THEN 'High Risk'

WHEN annual\_income < 30000 THEN 'Medium Risk'

ELSE 'Low Risk'

END AS risk\_category

FROM applicants

WHERE status = 'Active';

Risk Assessment Report - **Explanation:**  
This query categorizes each active applicant into 'High Risk', 'Medium Risk', or 'Low Risk' based on predefined criteria.

This SQL query retrieves details of active applicants and categorizes them into different risk categories based on specific conditions:

1. **Risk Categorization Logic**:
   * Applicants older than 60 are labeled as "High Risk."
   * Applicants with risky occupations such as 'Construction Worker' or 'Pilot' are also marked as "High Risk."
   * Applicants with an annual income below 30,000 are assigned "Medium Risk."
   * Everyone else is categorized as "Low Risk."
2. **Filtering**:
   * Only applicants with an Active status are included in the result.

**QUERY 4 -** Policy Renewal Report

**Objective:** Track the renewal rates of different policy types.

SELECT

policy\_type,

COUNT(\*) AS total\_policies,

COUNT(CASE WHEN renewal\_date > start\_date THEN 1 END) AS renewed\_policies,

ROUND((COUNT(CASE WHEN renewal\_date > start\_date THEN 1 END) \* 100.0) / COUNT(\*), 2) AS renewal\_rate\_percentage

FROM policies

GROUP BY policy\_type

ORDER BY renewal\_rate\_percentage DESC;

**Explanation:**  
This query calculates the renewal rate for each policy type by comparing renewal dates to start dates.

1. **SELECT Statement**:
   * **policy\_type**: The type of insurance policy (e.g., health, life, auto).
   * **COUNT(\*) AS total\_policies**: Counts all policies for each policy type.
   * **COUNT(CASE WHEN renewal\_date > start\_date THEN 1 END) AS renewed\_policies**: Counts only the policies that have been renewed, where the renewal date is after the start date.
   * **ROUND(...) AS renewal\_rate\_percentage**: Calculates the renewal rate as a percentage of renewed policies out of total policies, rounded to two decimal places.
2. **FROM policies**: Indicates that the data is being selected from the policies table.
3. **GROUP BY policy\_type**: Groups the results by the policy\_type column, so each row in the result set corresponds to a unique policy type.
4. **ORDER BY renewal\_rate\_percentage DESC**: Sorts the results in descending order based on the calculated renewal rate percentage, allowing you to easily see which policy types have the highest renewal rates.

**QUERY 5 - Fraud Detection Report**

**Objective:** Identify potentially fraudulent claims based on claim patterns.

SELECT

c.claim\_id,

a.name AS policyholder\_name,

c.claim\_amount,

c.claim\_date,

c.rejection\_reason

FROM claims c

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

JOIN applicants a ON p.policyholder\_id = a.applicant\_id

WHERE c.claim\_amount > 10000

OR (c.claim\_status = 'Rejected' AND c.rejection\_reason IS NOT NULL);

**Explanation:**  
This query flags high-value claims and rejected claims with reasons, which may require further investigation for fraud.

1. **Selecting Relevant Information**:
   * **c.claim\_id**: Retrieves the unique identifier for each claim.
   * **a.name AS policyholder\_name**: Retrieves the name of the policyholder associated with the claim.
   * **c.claim\_amount**: Retrieves the amount claimed.
   * **c.claim\_date**: Retrieves the date when the claim was submitted.
   * **c.rejection\_reason**: Retrieves the reason for rejection if the claim was denied.
2. **Joining Tables**:
   * The query joins three tables:
     + **claims c**: The main table holding claims data.
     + **policies p**: To associate each claim with its corresponding policy.
     + **applicants a**: To link policies with their policyholders.
3. **Filtering Conditions**:
   * The WHERE clause flags potentially suspicious claims:
     + **c.claim\_amount > 10000**: Identifies high-value claims that may warrant closer scrutiny.
     + **OR (c.claim\_status = 'Rejected' AND c.rejection\_reason IS NOT NULL)**: Flags rejected claims that have specified reasons, which may indicate patterns of potential fraud.

**Purpose**

This query serves as an initial filter for identifying claims that may require further investigation for fraudulent activity. High-value claims and rejected claims with clear reasons are often areas of concern for insurers, making this report a useful tool for risk assessment and fraud detection efforts.

**QUERY 6 - Loss Ratio Calculation**

**Objective:** Calculate the loss ratio (claims paid vs. premiums earned) for each policy type.

SELECT

p.policy\_type,

SUM(c.claim\_amount) AS total\_claims\_paid,

SUM(p.premium\_amount) AS total\_premiums\_earned,

ROUND((SUM(c.claim\_amount) / SUM(p.premium\_amount)) \* 100, 2) AS loss\_ratio\_percentage

FROM policies p

LEFT JOIN claims c ON p.policy\_id = c.policy\_id AND c.claim\_status = 'Approved'

WHERE p.status = 'Active'

GROUP BY p.policy\_type

ORDER BY loss\_ratio\_percentage DESC;

**Explanation:**  
This query calculates the loss ratio by comparing the total approved claims to the total premiums earned for each policy type.

* **Purpose**: This query calculates the loss ratio for each policy type by comparing total approved claims to total earned premiums.
* **Key Columns**:
  + **policy\_type**: Type of insurance policy.
  + **total\_claims\_paid**: Sum of approved claim amounts.
  + **total\_premiums\_earned**: Sum of premiums for active policies.
  + **loss\_ratio\_percentage**: Percentage of claims paid relative to premiums earned, rounded to two decimal places.
* **Join Type**: Uses a left join to include all active policies, counting only approved claims.
* **Output**: Results are grouped by policy type and ordered by loss ratio percentage in descending order, highlighting riskier policies.