Assessment for QA specialist - EHR Rev Transaction Ingestion

Task 1: Data Analysis Using Excel/Google Sheets.

Goal 1: Finding the associated provider for billing data using appointment data.

At first, I am creating new column manually in a **scheduling_data** table of excel sheet.

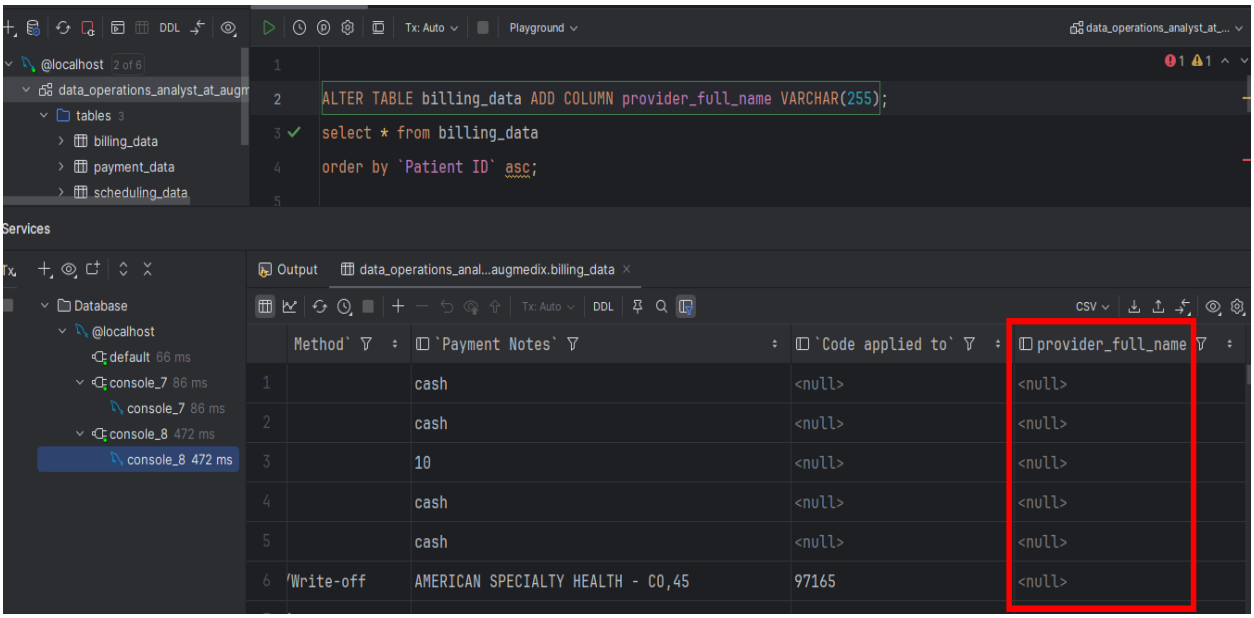
"Staff Full Name" column create using by "Staff First Name" and "Staff Last Name" in Excel Sheet

Staff ID	Staff First Name	Staff Last Name	Staff Full Name
RobertC	Robert	Crochet	Robert Crochet
KatherineT	Katherine	Tyree	Katherine Tyree
KelseaR	Kelsea	Rodriguez	Kelsea Rodriguez
CareyS	Carolyn	Simpson	Carolyn Simpson
KelliC	Kelli	Cruthirds	Kelli Cruthirds
christianl	Christian	Litz	Christian Litz
KatherineT	Katherine	Tyree	Katherine Tyree
RobertC	Robert	Crochet	Robert Crochet
KatherineT	Katherine	Tyree	Katherine Tyree
christianl	Christian	Litz	Christian Litz
RobertC	Robert	Crochet	Robert Crochet
BrianA	James	Allen	James Allen
BrianA	James	Allen	James Allen
KelliC	Kelli	Cruthirds	Kelli Cruthirds
DannaG	D'Anna	Pullen	D'Anna Pullen
VMiley	Victoria	Miley	Victoria Miley
CareyS	Carolyn	Simpson	Carolyn Simpson
KelseaR	Kelsea	Rodriguez	Kelsea Rodriguez
KelliC	Kelli	Cruthirds	Kelli Cruthirds

Then I am adding new column (provider_full_name) in a **billing_data** table using MySQL Database.

SQL Command:

ALTER TABLE billing_data **ADD COLUMN** provider_full_name **VARCHAR(255);**



Then,

- ✓ I am used to MyQL JOIN between the scheduling_data and billing_data table.
- ✓ UPDATE billing_data table.
- ✓ CONCAT using Staff First Name and Staff Last Name.
- ✓ SET value in a provider_full_name

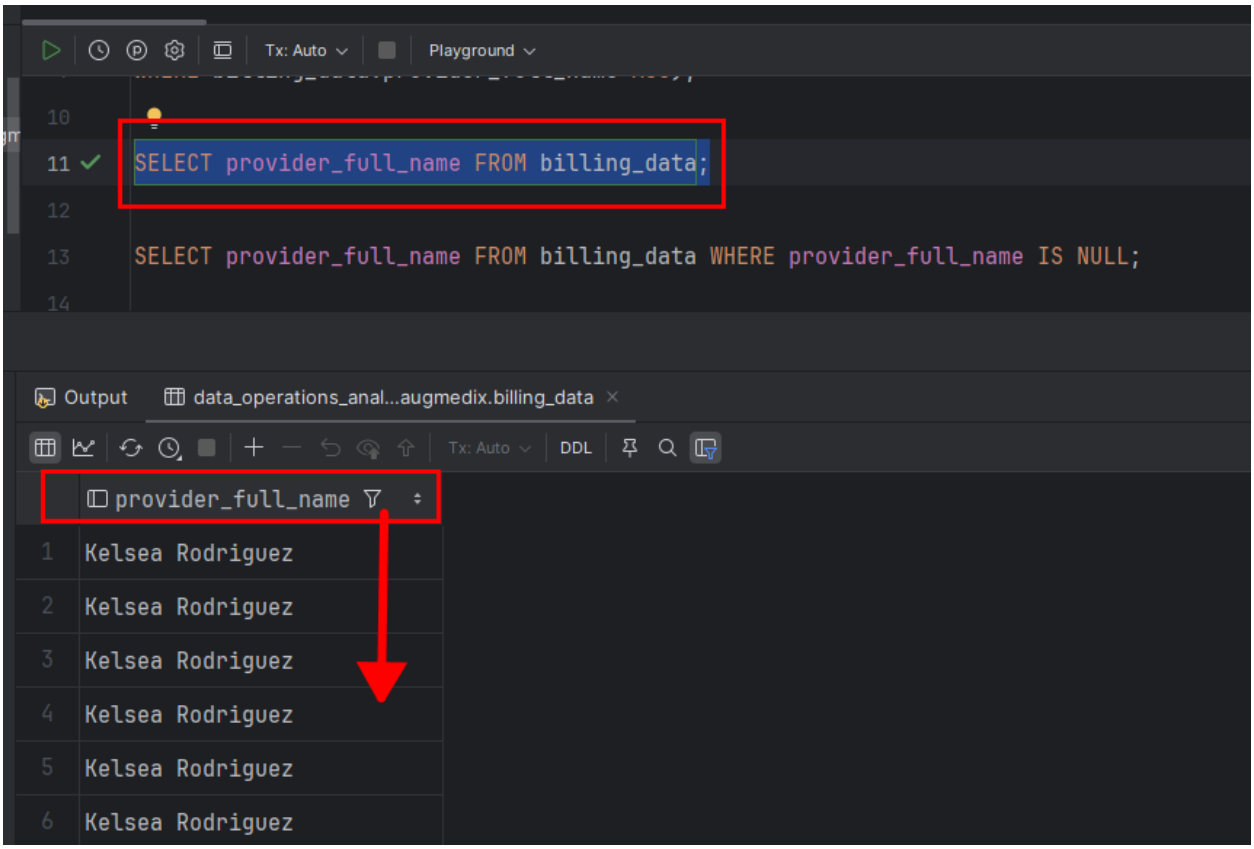
SQL Command:

CONCAT(scheduling_data.`Staff First Name`,' ', scheduling_data.`Staff Last Name`)

(UPDATE billing_data **JOIN** scheduling_data
ON billing_data.`Patient ID` = scheduling_data.`Patient ID`
AND billing_data.DOS = scheduling_data.`Appointment Start Date`
SET billing_data.provider_full_name = **CONCAT**(scheduling_data.`Staff First Name`,' ', scheduling_data.`Staff Last Name`)
WHERE billing_data.provider_full_name ASC);

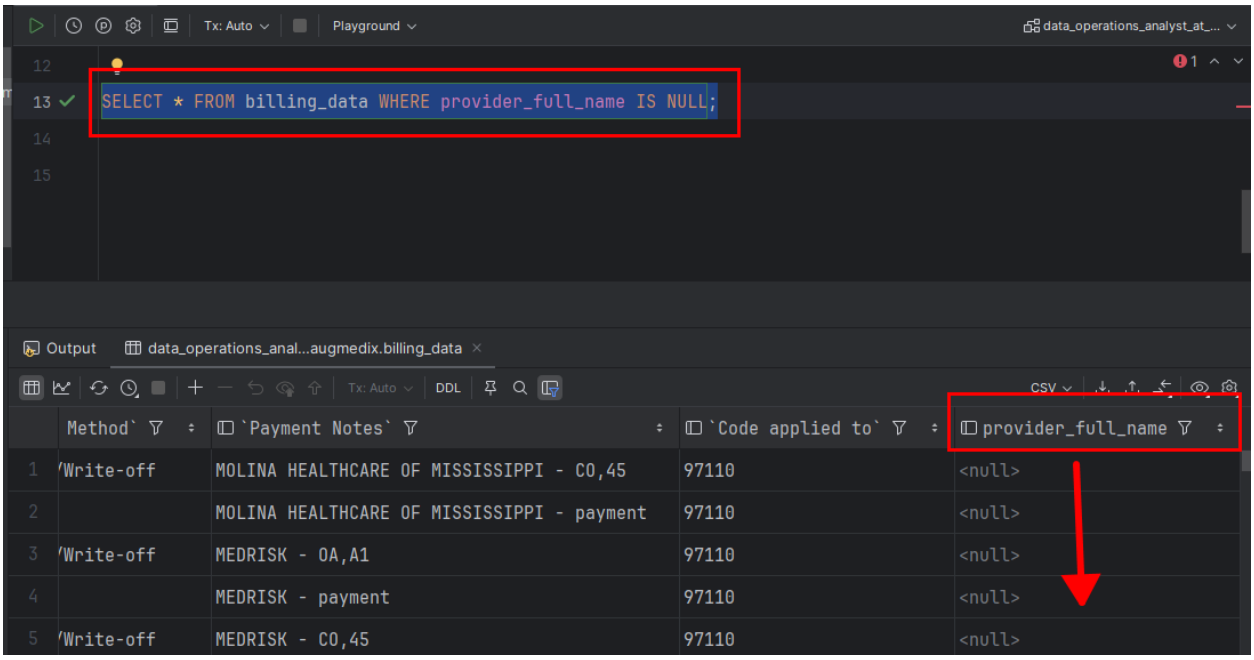
SQL Command:

SELECT provider_full_name **FROM** billing_data**;**



SQL Command:

SELECT * FROM billing_data **WHERE** provider_full_name **IS NULL;**



Goal 2: Calculating Total Amount Collected Per Provider

SQL Command:

SELECT provider_full_name, ***SUM***(Amount) **AS** total_amount_collected **FROM** billing_data
WHERE `Payment Method` **IN** ('Cash', 'Check', 'EFT', 'Credit Card', 'Visa', 'Mastercard', 'AMEX')
GROUP BY provider_full_name
ORDER BY total_amount_collected **DESC;**

```
#Goal 2: Calculating Total Amount Collected Per Provider
SELECT provider_full_name, SUM(Amount) AS total_amount_collected FROM billing_data
WHERE `Payment Method` IN ('Cash', 'Check', 'EFT', 'Credit Card', 'Visa', 'Mastercard', 'AMEX')
GROUP BY provider_full_name
ORDER BY total_amount_collected DESC;
```

provider_full_name	total_amount_collected
Christian Litz	16942.17999999999
Katherine Tyree	14870.76
D'Anna Pullen	13261.95999999999
Victoria Miley	12693.289999999979
James Allen	[illegible]00000016
Jessica Pittman	9693.160000000005

16 rows

Explanation:

1. **Filters out non-revenue payment methods**
   - ✓ Excluded: 'Adjustment/Write-off', 'Check Refund' (since they do not contribute to revenue).
   - ✓ Included: 'Cash', 'Check', 'EFT', 'Credit Card', 'Visa', 'Mastercard', 'AMEX'.
2. **Calculates total revenue for each provider**
   - ✓ Uses SUM(Amount) to compute total revenue for each provider_full_name.
   - ✓ Groups by provider_full_name.
   - ✓ Orders by total revenue in descending order.

Goal 3: Counting the distinct encounters in the billing data

```
#Goal 3: Counting the distinct encounters in the billing data.
SELECT COUNT(DISTINCT
    CASE
        WHEN provider_full_name IS NOT NULL THEN CONCAT(`Patient ID`, '_', `DOS`, '_', provider_full_name)
        ELSE CONCAT(`Patient ID`, '_', `DOS`)
    END
) AS total_unique_encounters FROM billing_data;
```

Output | total_unique_encounters:int

total_unique_encounters
1792

Explanation

Encounters are defined as unique combinations of:

- ✓ Patient ID
- ✓ Date of Service (DOS)
- ✓ provider_full_name.

If provider_full_name is NOT NULL, the encounter is identified by Patient ID, DOS, and provider_full_name.

If provider_full_name is NULL, the encounter is identified by Patient ID and DOS only.

Using DISTINCT and COUNT Functions.

Task 2: RCM Data Analysis Assessment

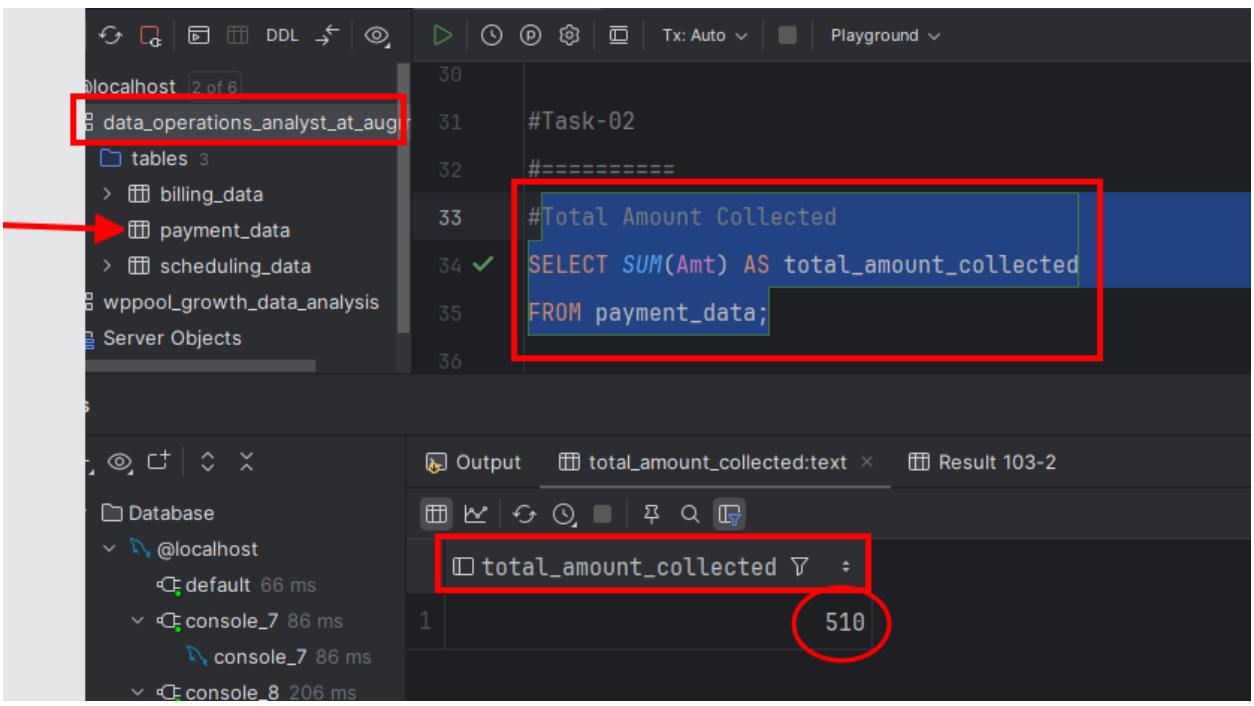
Goal 1: Calculating total collections using claim level dataset provided in payment_data

I did data cleaning and transformation on my payment_data table. Adding billing data and payment data, where the claim ID was matching and the CPT codes were partially matching.

Total Amount Collected:

SQL Command:

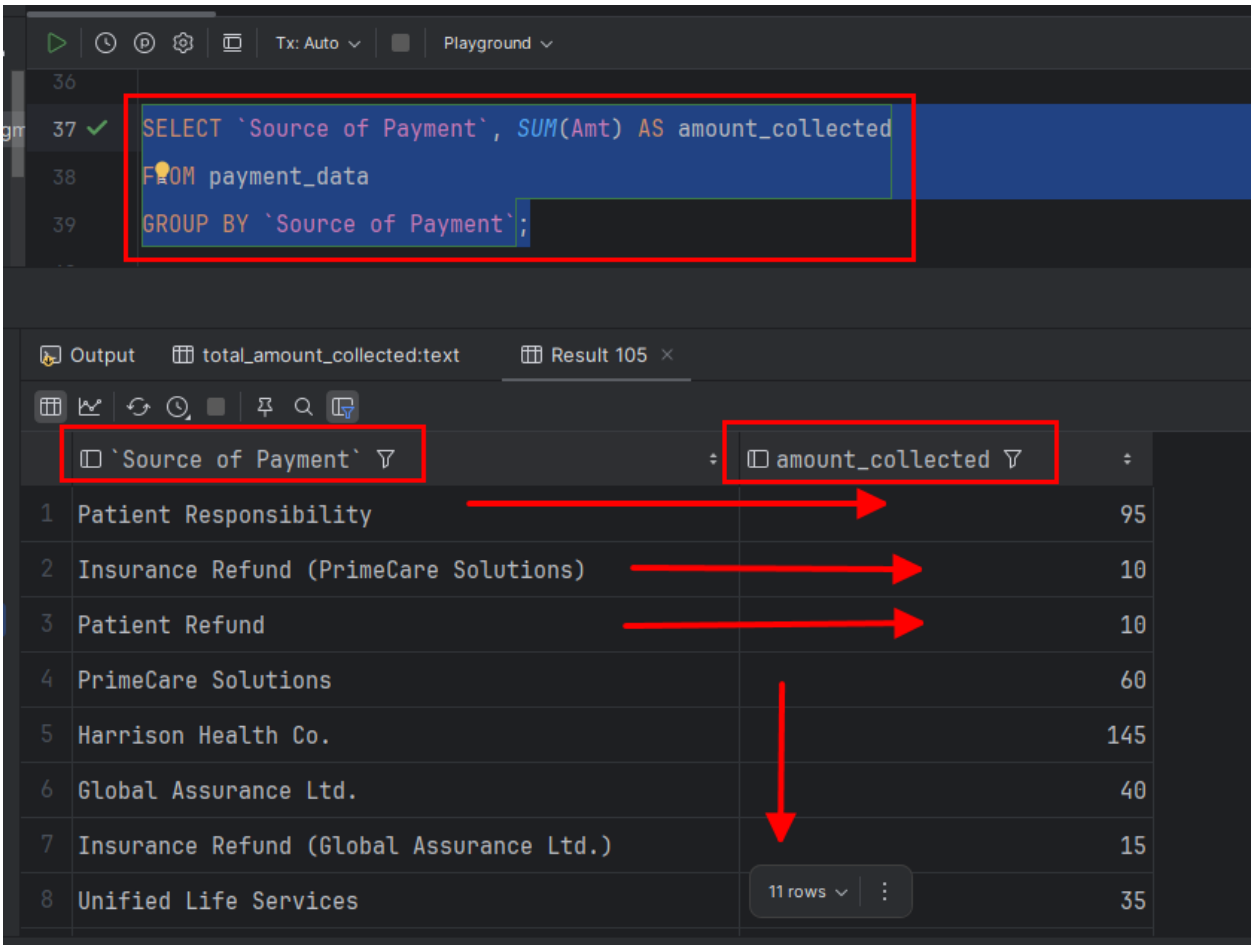
SELECT *SUM*(Amt) AS total_amount_collected **FROM** payment_data;



Breakdown by Payment Source:

SQL Command:

SELECT `Source of Payment`, ***SUM***(Amt) **AS** amount_collected
FROM payment_data
GROUP BY `Source of Payment`;

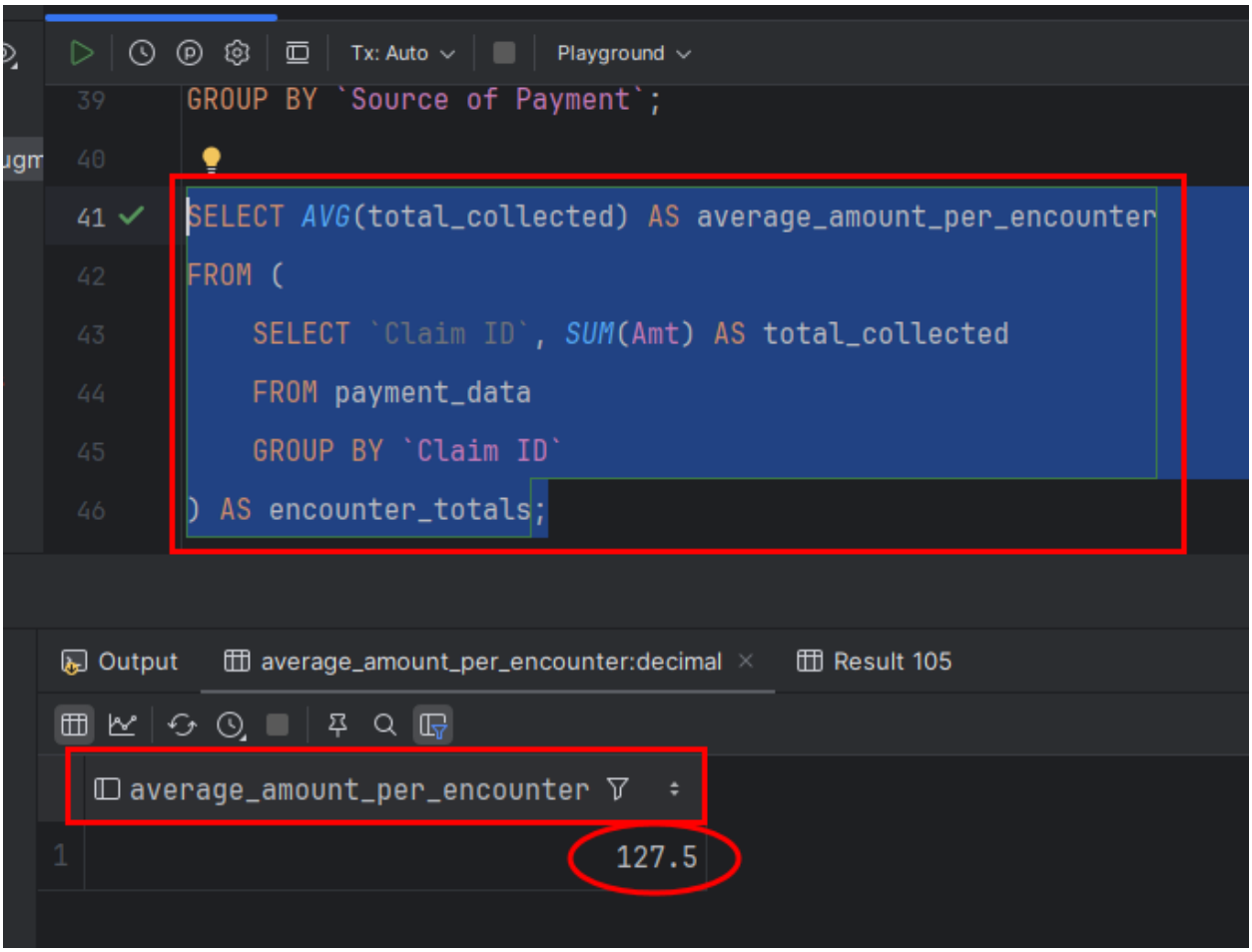


This groups collections by **Primary Insurance, Secondary Insurance, and Patient Responsibility**.

Average Amount Collected Per Encounter:

SQL Command:

```
SELECT AVG(total_collected) AS average_amount_per_encounter
FROM (
   SELECT `Claim ID`, SUM(Amt) AS total_collected
   FROM payment_data
   GROUP BY `Claim ID`
) AS encounter_totals;
```



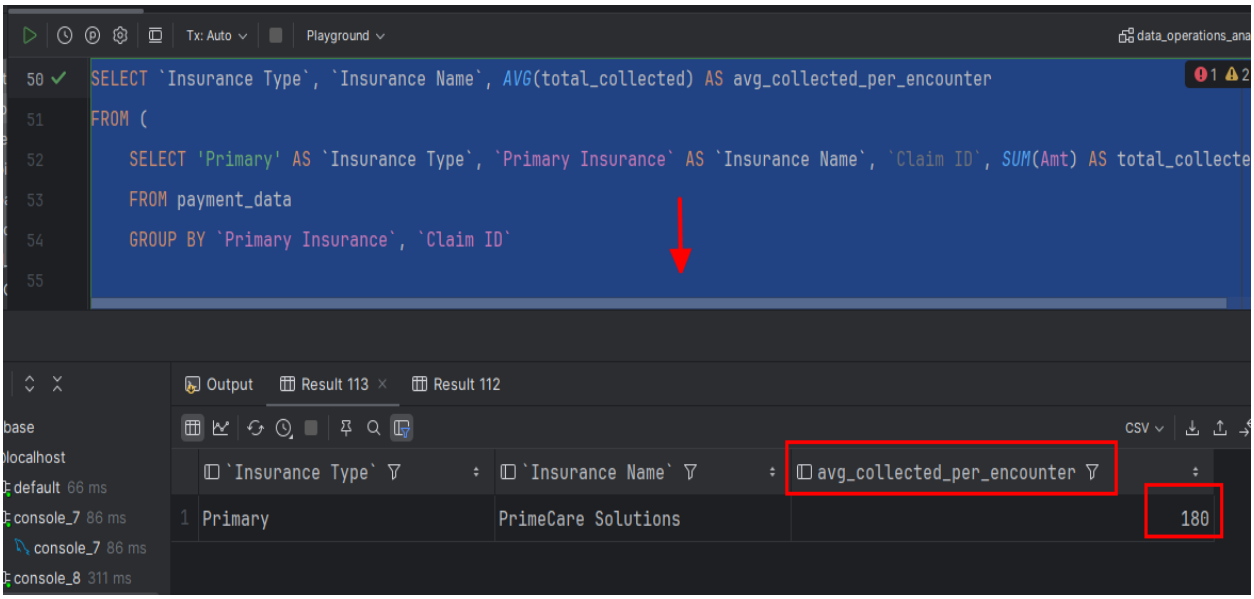
- ✓ First, it groups by Claim ID to get total revenue per encounter.
- ✓ Then, it calculates the average revenue per unique encounter.

Goal 2: Primary and Secondary Insurance Analysis

Average Amount Collected Per Encounter:

SQL Command:

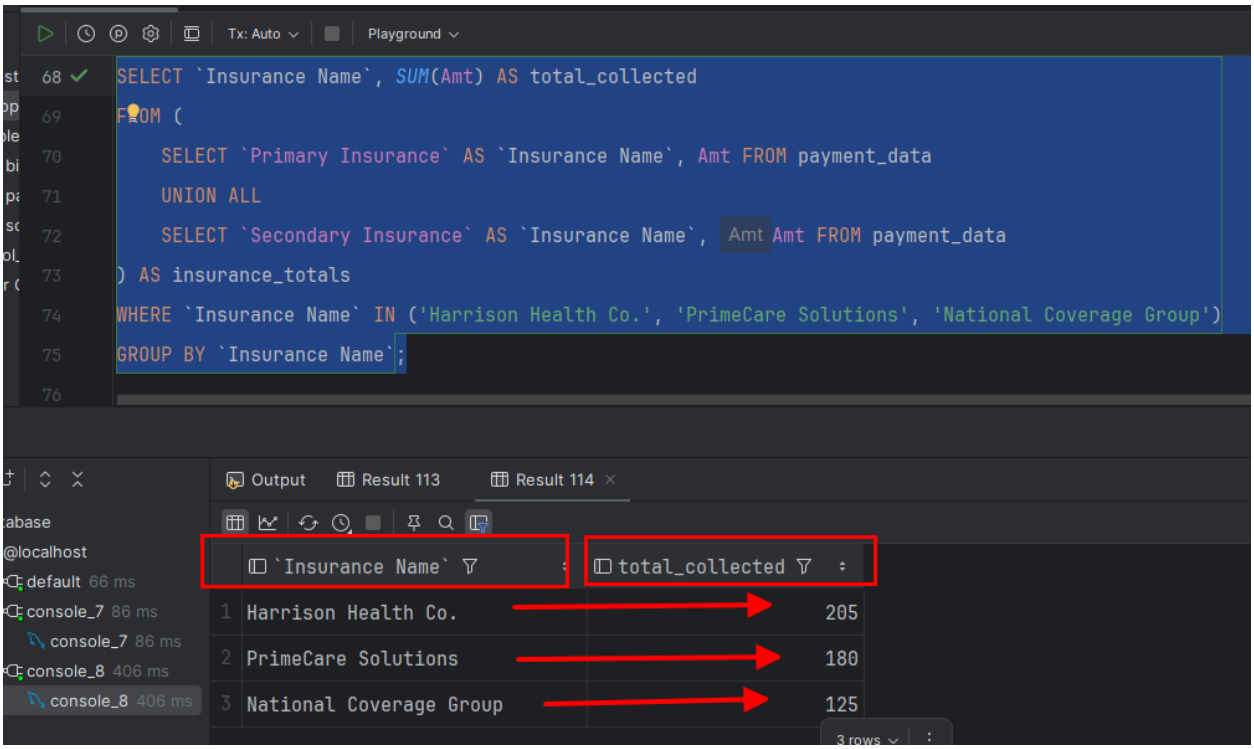
```
SELECT `Insurance Type`, `Insurance Name`, AVG(total_collected) AS
avg_collected_per_encounter
FROM (
   SELECT 'Primary' AS `Insurance Type`, `Primary Insurance` AS
`Insurance Name`, `Claim ID`, SUM(Amt) AS total_collected
   FROM payment_data
   GROUP BY `Primary Insurance`, `Claim ID`

   UNION ALL
   SELECT 'Secondary' AS `Insurance Type`, `Secondary Insurance` AS
`Insurance Name`, `Claim ID`, SUM(Amt) AS total_collected
   FROM payment_data
   GROUP BY `Secondary Insurance`, `Claim ID`
) AS insurance_totals
GROUP BY `Insurance Type`, `Insurance Name`
ORDER BY avg_collected_per_encounter DESC
LIMIT 1;
```



Average Amount Collected Per Encounter:

SQL Command:

SELECT `Insurance Name`, ***SUM***(Amt) **AS** total_collected
FROM (
  SELECT `Primary Insurance` **AS** `Insurance Name`, Amt **FROM** payment_data
  UNION ALL
  SELECT `Secondary Insurance` **AS** `Insurance Name`, Amt **FROM** payment_data
) AS insurance_totals
WHERE `Insurance Name` **IN** ('Harrison Health Co.', 'PrimeCare Solutions', 'National Coverage Group')
GROUP BY `Insurance Name`;



Goal 3: Understanding the basics of revenue cycle management

1. Claim Submissions:

Claim submission is the process of sending healthcare claims to insurance providers for reimbursement of medical services. It ensures that healthcare providers receive payment for services rendered.

Key Steps in Claim Submission:

- **Patient Registration & Insurance Verification**
  - ✓ Collect patient details, insurance coverage, and eligibility.
  - ✓ Ensure accuracy in policy details to prevent denials.

- **Medical Coding & Charge Entry**
  - ✓ Convert diagnoses and procedures into standardized medical codes (CPT, ICD-10, HCPCS).
  - ✓ Ensure that services are correctly billed based on documentation.

📌 **Claim Creation & Review**
✓ Populate claim forms (CMS-1500 for outpatient, UB-04 for inpatient).
✓ Validate against payer-specific rules for compliance.

📌 **Claim Submission to Payers**
✓ Submit claims electronically via clearinghouses (EDI transactions).
✓ Direct submission to insurers when applicable.

📌 **Payer Acknowledgment & Processing**
✓ Insurer receives and validates the claim.
✓ Claim is either approved, denied, or returned for corrections.

2. Claim Reconciliation

Claim reconciliation is the process of matching insurance payments with billed amounts to ensure accurate revenue capture and identify discrepancies.

Importance of Claim Reconciliation:

✓ Prevents revenue leakage due to underpayments or missing payments.
✓ Ensures correct payment posting against claims.
✓ Helps in timely appeals and dispute resolutions for incorrect payments.

Key Steps in Claim Reconciliation:

📌 **Payment Posting**
✓ Match payments received (EFT, checks) with claims in the billing system.
✓ Identify underpayments, overpayments, and unpaid claims.

📌 **Remittance Advice Review**
✓ Analyze Explanation of Benefits (EOB) or Electronic Remittance Advice (ERA).
✓ Verify allowed amount, adjustments, and denials.

- **Discrepancy Identification**
- ✓ Compare expected vs. received payments.
- ✓ Investigate partial payments, missing claims, and incorrect adjustments.

- **Resolution & Resubmission**
- ✓ Contact payers for payment discrepancies.
- ✓ Resubmit claims if errors are found.
- ✓ Apply necessary adjustments or refunds.

3. Claim Denials

Claim denials occur when insurance companies refuse to reimburse a healthcare provider due to errors, missing information, or policy-related reasons.

Common Reasons for Claim Denials:

- ✓ Coding Errors – Incorrect CPT/ICD-10 codes, mismatched procedures and diagnoses.
- ✓ Incomplete Information – Missing patient demographics, authorization numbers, or provider details.
- ✓ Duplicate Claims – Same claim submitted multiple times.
- ✓ Eligibility Issues – Expired insurance, coverage limits, or service exclusions.
- ✓ Timely Filing Limits – Claims submitted after the payer's deadline.

Denial Management Strategies:

- **Denial Tracking & Root Cause Analysis**
- ✓ Categorize denials by type and frequency.
- ✓ Identify patterns to implement preventive measures.
- **Claim Appeal Process**
- ✓ Gather supporting documents (medical records, authorization).
- ✓ Submit corrected claims with justification.
- ✓ Follow up with insurers to ensure resolution.

Process Improvement & Automation

- ✓ Use claim scrubbers to detect errors before submission.
- ✓ Automate eligibility verification and prior authorizations.
- ✓ Train billing staff on compliance and best practices.

Data Visualization using Power BI tools: