

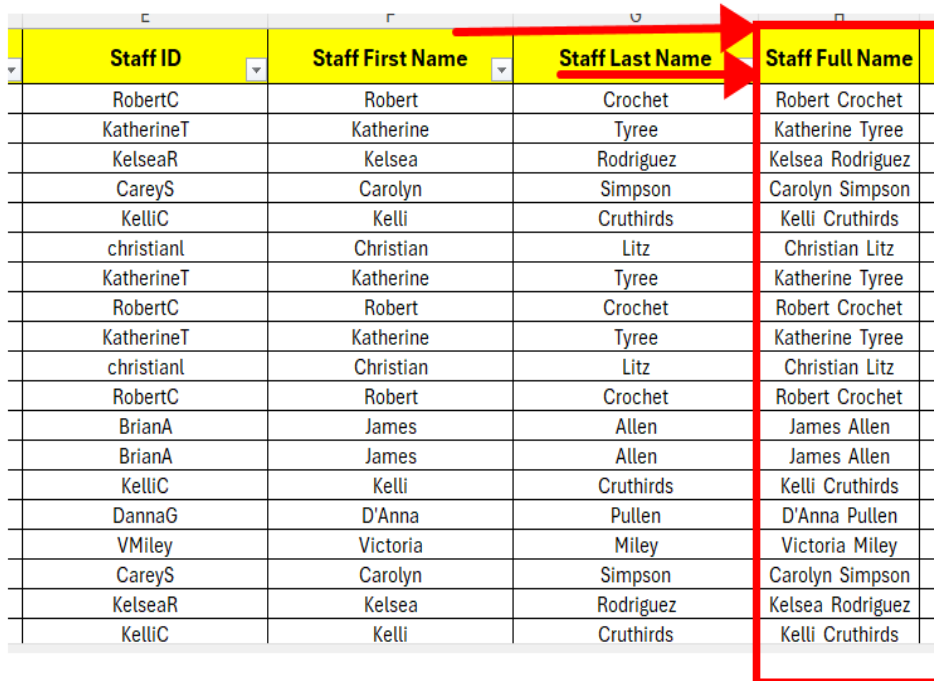
## Assessment for QA specialist - EHR Rev Transaction Ingestion

### **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**

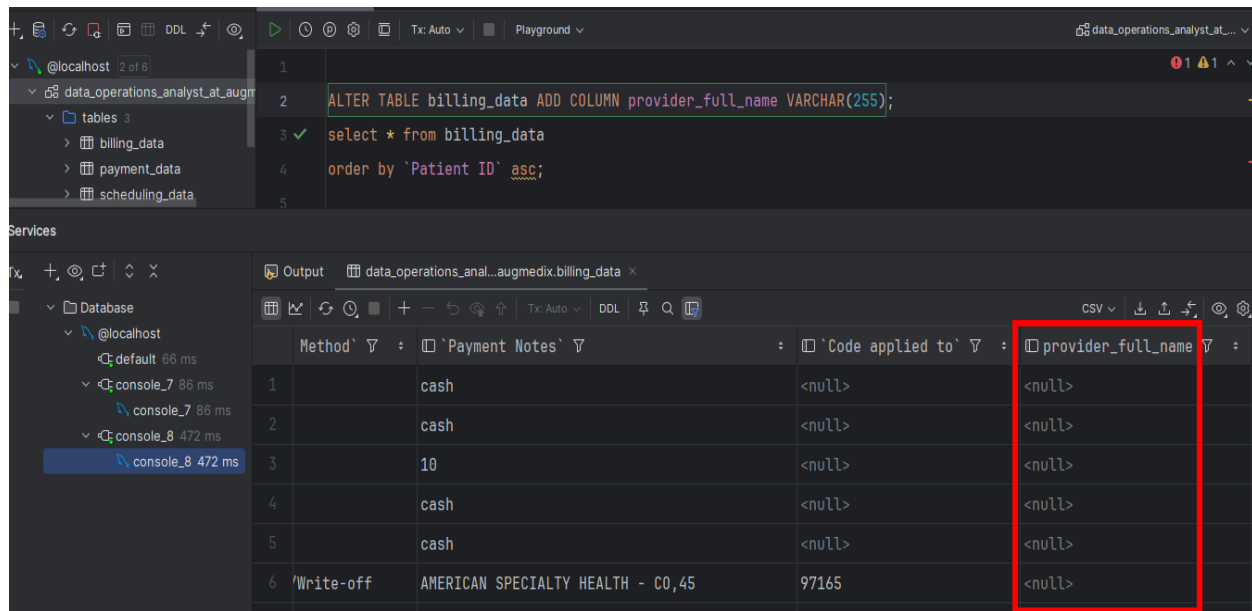


Staff ID	Staff First Name	Staff Last Name	Staff Full Name
RobertC	Robert	Crochet	Robert Crochet
KatherineT	Katherine	Tyree	Katherine Tyree
KelseaR	Kelsea	Rodriguez	Kelsea Rodriguez
CareyS	Carolyn	Simpson	Carolyn Simpson
KelliC	Kelli	Cruthirds	Kelli Cruthirds
christianl	Christian	Litz	Christian Litz
KatherineT	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);**



The screenshot shows a MySQL IDE interface. The top panel displays the SQL command: `ALTER TABLE billing_data ADD COLUMN provider_full_name VARCHAR(255);`. Below the command, a table view shows the results of a query. The table has four columns: 'Method', 'Payment Notes', 'Code applied to', and 'provider\_full\_name'. The 'provider\_full\_name' column is highlighted in red. The table contains six rows of data, with the last row showing a 'Write-off' method and a provider name.

Method	Payment Notes	Code applied to	provider_full_name
1	cash	<null>	<null>
2	cash	<null>	<null>
3	10	<null>	<null>
4	cash	<null>	<null>
5	cash	<null>	<null>
6	'Write-off	AMERICAN SPECIALTY HEALTH - CO,45	97165

Then,

- ✓ I am used to MySQL 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;

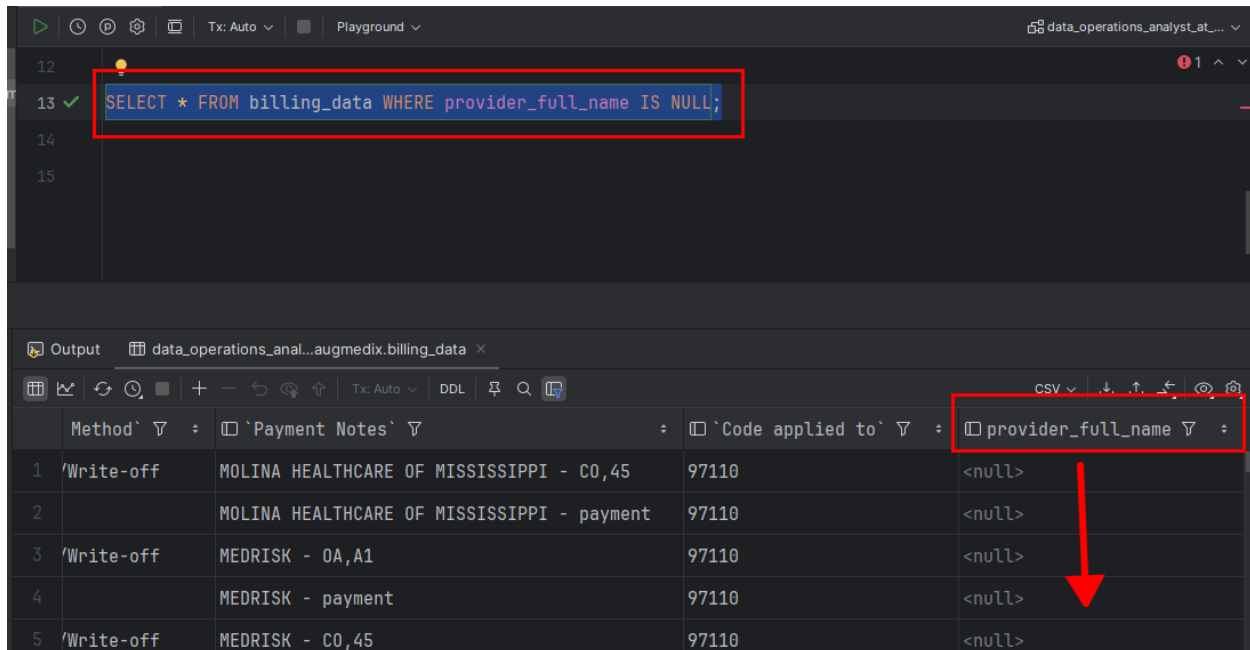
The screenshot shows a SQL playground interface. The top section displays a list of SQL queries. Query 11, `SELECT provider_full_name FROM billing_data;`, is highlighted with a red box and a green checkmark, indicating it was executed successfully. Below the queries, the 'Output' tab is active, showing the results of the selected query. The results are displayed in a table with a single column, `provider_full_name`, which is also highlighted with a red box. The table contains six rows, all with the value 'Kelsea Rodriguez'. A red arrow points from the column header to the first row of data.

```
10
11 ✓ SELECT provider_full_name FROM billing_data;
12
13 SELECT provider_full_name FROM billing_data WHERE provider_full_name IS NULL;
14
```

	provider_full_name
1	Kelsea Rodriguez
2	Kelsea Rodriguez
3	Kelsea Rodriguez
4	Kelsea Rodriguez
5	Kelsea Rodriguez
6	Kelsea Rodriguez

### SQL Command:

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



The screenshot shows a SQL playground interface. The top panel displays the SQL query: `SELECT * FROM billing_data WHERE provider_full_name IS NULL;`. The bottom panel shows the results of the query in a table format. The table has five columns: Method, Payment Notes, Code applied to, and provider\_full\_name. The results show five rows of data, all with null values in the provider\_full\_name column. A red box highlights the provider\_full\_name column header, and a red arrow points to the first row of data.

	Method	Payment Notes	Code applied to	provider_full_name
1	Write-off	MOLINA HEALTHCARE OF MISSISSIPPI - C0,45	97110	<null>
2		MOLINA HEALTHCARE OF MISSISSIPPI - payment	97110	<null>
3	Write-off	MEDRISK - 0A,A1	97110	<null>
4		MEDRISK - payment	97110	<null>
5	Write-off	MEDRISK - C0,45	97110	<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;
```

15

16 #Goal 2: Calculating Total Amount Collected Per Provider

17 `SELECT provider_full_name, SUM(Amount) AS total_amount_collected FROM billing_data`

18 `WHERE 'Payment Method' IN ('Cash', 'Check', 'EFT', 'Credit Card', 'Visa', 'Mastercard', 'AMEX')`

19 `GROUP BY provider_full_name`

20 `ORDER BY total_amount_collected DESC;`

21

Output Result 96

provider\_full\_name total\_amount\_collected

1	Christian Litz	16942.17999999999
2	Katherine Tyree	14870.76
3	D'Anna Pullen	13261.95999999999
4	Victoria Miley	12693.289999999979
5	James Allen	9693.160000000016
6	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

The screenshot shows a SQL playground interface. At the top, there's a toolbar with icons for running, saving, and other functions. Below the toolbar, a text area contains a SQL query. The query is as follows:

```
#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;
```

Below the query editor, there's an 'Output' section. It shows a table with one column named 'total\_unique\_encounters' and one row with the value '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;
```

The screenshot displays a SQL IDE interface. On the left, a database explorer shows a table named `payment_data` under the `data_operations_analyst_at_aug` schema. A red arrow points to this table. The main editor shows the following SQL query:

```
#Task-02
#=====
#Total Amount Collected
SELECT SUM(Amt) AS total_amount_collected
FROM payment_data;
```

The query is highlighted with a blue background. Below the editor, the 'Output' pane shows the result of the query. The column `total_amount_collected` is highlighted, and the value `510` is displayed in a red circle.

total_amount_collected
510

## Breakdown by Payment Source:

### SQL Command:

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

Output total\_amount\_collected:text Result 105 x

	Source of Payment	amount_collected
1	Patient Responsibility	95
2	Insurance Refund (PrimeCare Solutions)	10
3	Patient Refund	10
4	PrimeCare Solutions	60
5	Harrison Health Co.	145
6	Global Assurance Ltd.	40
7	Insurance Refund (Global Assurance Ltd.)	15
8	Unified Life Services	35

11 rows

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;
```

The screenshot displays a SQL playground interface. The top section shows the SQL query being executed, which is the same query as provided in the text. The query is highlighted with a red box. Below the query editor, the 'Output' tab is selected, showing the result of the query. The result is a single row with the value 127.5, which is circled in red. The column name 'average\_amount\_per\_encounter' is also highlighted with a red box.

average_amount_per_encounter
127.5

- ✓ 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;
```

```

50 ✓ SELECT `Insurance Type`, `Insurance Name`, AVG(total_collected) AS avg_collected_per_encounter
51 FROM (
52     SELECT 'Primary' AS `Insurance Type`, `Primary Insurance` AS `Insurance Name`, `Claim ID`, SUM(Amt) AS total_collected
53     FROM payment_data
54     GROUP BY `Primary Insurance`, `Claim ID`
55 )

```

Insurance Type	Insurance Name	avg_collected_per_encounter
Primary	PrimeCare Solutions	180

**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`;

```

```
68 ✓ SELECT `Insurance Name`, SUM(Amt) AS total_collected
69 FROM (
70     SELECT `Primary Insurance` AS `Insurance Name`, Amt FROM payment_data
71     UNION ALL
72     SELECT `Secondary Insurance` AS `Insurance Name`, Amt FROM payment_data
73 ) AS insurance_totals
74 WHERE `Insurance Name` IN ('Harrison Health Co.', 'PrimeCare Solutions', 'National Coverage Group')
75 GROUP BY `Insurance Name`;
76
```

Insurance Name	total_collected
Harrison Health Co.	205
PrimeCare Solutions	180
National Coverage Group	125

## 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:

