NoSQL Databases - Weekly 7 Final Project FINAL

INFO 579 Week 7 Final Project Report
Course: INFO 579: SQL/NoSQL Databases for Data and Information Sciences
Module/Week: 7 - Week of August 18, 2025
Topic: Week 7 Final Project Report

NOTE: The assignment document must provide the following information. Up to 5 points may be deducted due to the lack of the below information.

Student's Full Name: Matthew Qi Lan Thompson
Course Title: INFO 579: SQL/NoSQL Databases for Data and Information Sciences
Term name and year: Summer 2025
Submission Week: Week 7 Final Project Report
Instructor's Name: Dr. Nayem Rahman
Date of Submission: Aug 22, 2025

NOTE: A 20% deduction of points will be applied for any late submission.

Week 2 Final Project Update 1: Total Points - 25

Week 2 Final Project Update 1:

1. Come up with a business plan for which you need to collect and store data
2. Come up with the data about your business. Example: like the one I gave you for the assignment.

Requirements: As part of the Final Project Update 1, submit a single-spaced one-page report:

1. A brief description of your project.
2. The data you will use and the source of that data.
3. Upload the data file (like the one I gave you to do the assignments).

Business Descriptive Report

1. Business Plan: Hospital Performance & Patient Outcome Analytics System

I am interested in working with data that allows me to assess the following:

- **patient outcomes**,
- **provider performance**, and
- **procedure effectiveness**

I thought about the kind of data I want to work with in the future and I decided to try healthcare-based data and would like to gain experiences with improving healthcare services overall.

---

2. Data Collection & Storage Plan

Data Sources:

I'll be using open-sourced data from **Synthea** (https://synthetichealth.github.io/synthea/), which represents realistic hospital EHR data settings. The listed datasets I plan to use are shown below:

Data Tables	Brief Description	Objective
`patients.csv`	Basic demographic info	Identified patients, stratify risk by age/gender
`encounters.csv`	Visits per patient	Core fact table; basis for all events
`observations.csv`	Vitals, labs, metrics	Outcome indicators (e.g., BP, glucose)
`conditions.csv`	Diagnoses assigned	Track chronic illness and other diagnoses
`procedures.csv`	Surgeries/treatments	Evaluate procedure volume & quality
`providers.csv`	Doctors/facilities	Analyze workload, quality, profitability

With these data tables in csv, I'll be able to easily import them to MySQL and use SQL queries to obtain valuable insights about these imported data in a relational schema altogether.

---

3. Business Objectives (Driven by Data)

When working with these data using SQL queries, I propose some improvements for the healthcare service quality in a virtual simulation setting using the following objectives I can think of below:

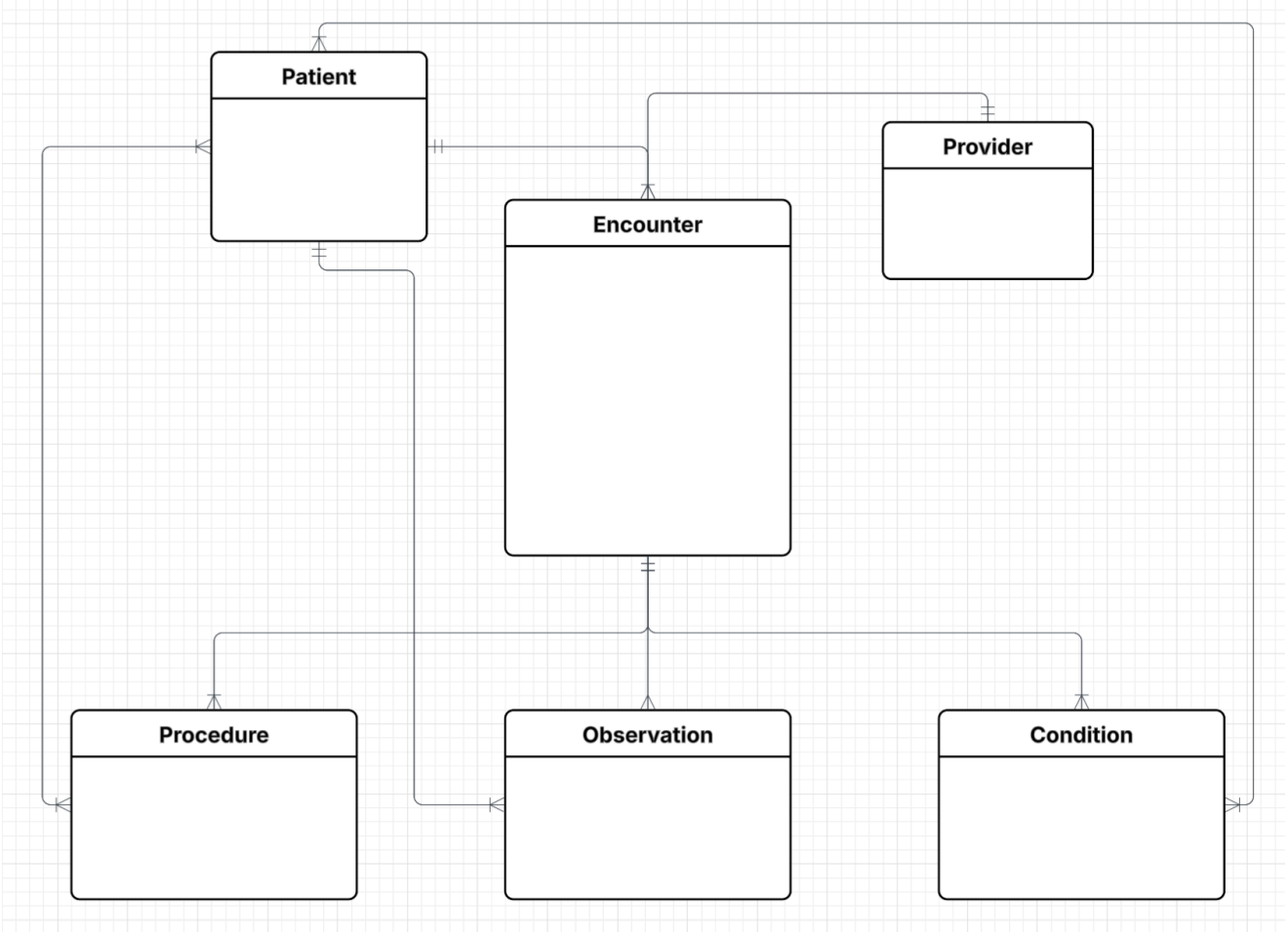
1. **Increase Profitability**
   I'll look at procedure outcomes and compare them to the costs. If a procedure has poorer quality but higher cost, that's something worth reviewing and possibly improving.
2. **Improve Clinical Quality**
   I'll check the frequency of conditions and assess whether they're being well managed, based on the number of visits for the same condition.
3. **Optimize Provider Utilization**
   I can look at providers and see how well they're performing, indicated by the number of encounters they receive under their care.
4. **Reduce Readmissions**
   I can assess a patient's health risk based on how often they return for the same condition and how much time passes between each visit. If the time gap is short and the frequency is high, they could be flagged as higher risk, which can help streamline follow-up care.
5. **Identify Care Strategic Expansion**
   I can observe the frequency of certain conditions. If some conditions show up very often, I could consider proposing improvements or expansions in that area of care, depending on other factors like treatment outcomes and how often patients return.

However, I may continue to add more business objectives or adjust them as I work with the Synthea data, since its size is enormous and it's designed to mimic a realistic healthcare setting. That gives me plenty of opportunities to practice working with data analytics from different angles beyond what I've proposed so far.

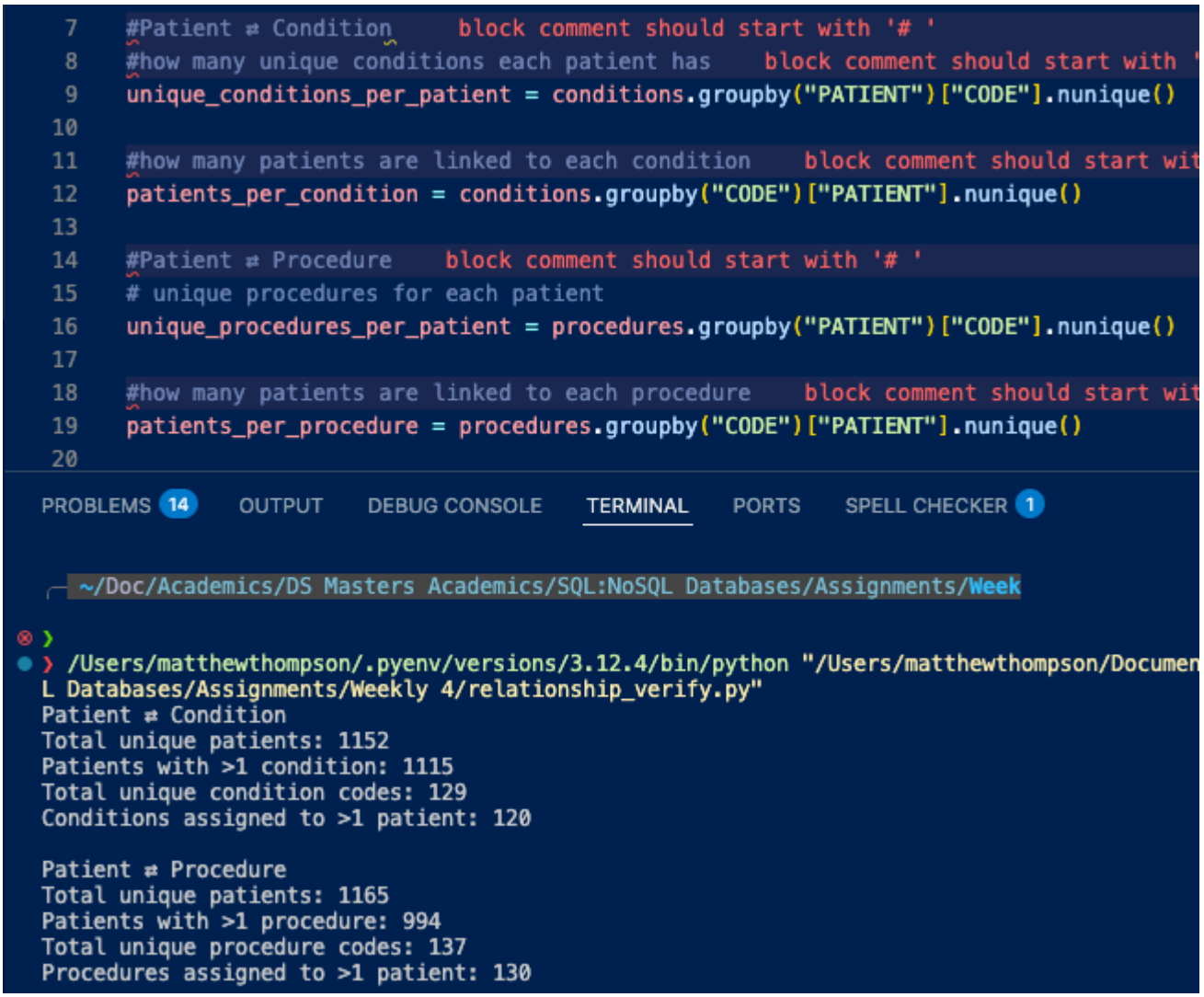
I'll upload the data files I plan to use (and mentioned in data collection & storage plan) along with this brief project description in PDF.

Week 4 Final Project Update 2: Total Points - 25

3. Develop a Conceptual Model. Consider 4 or 5 entities. Make sure you have at least one many-to-many relationship. Explain with data why it's a many-to-many relationship.

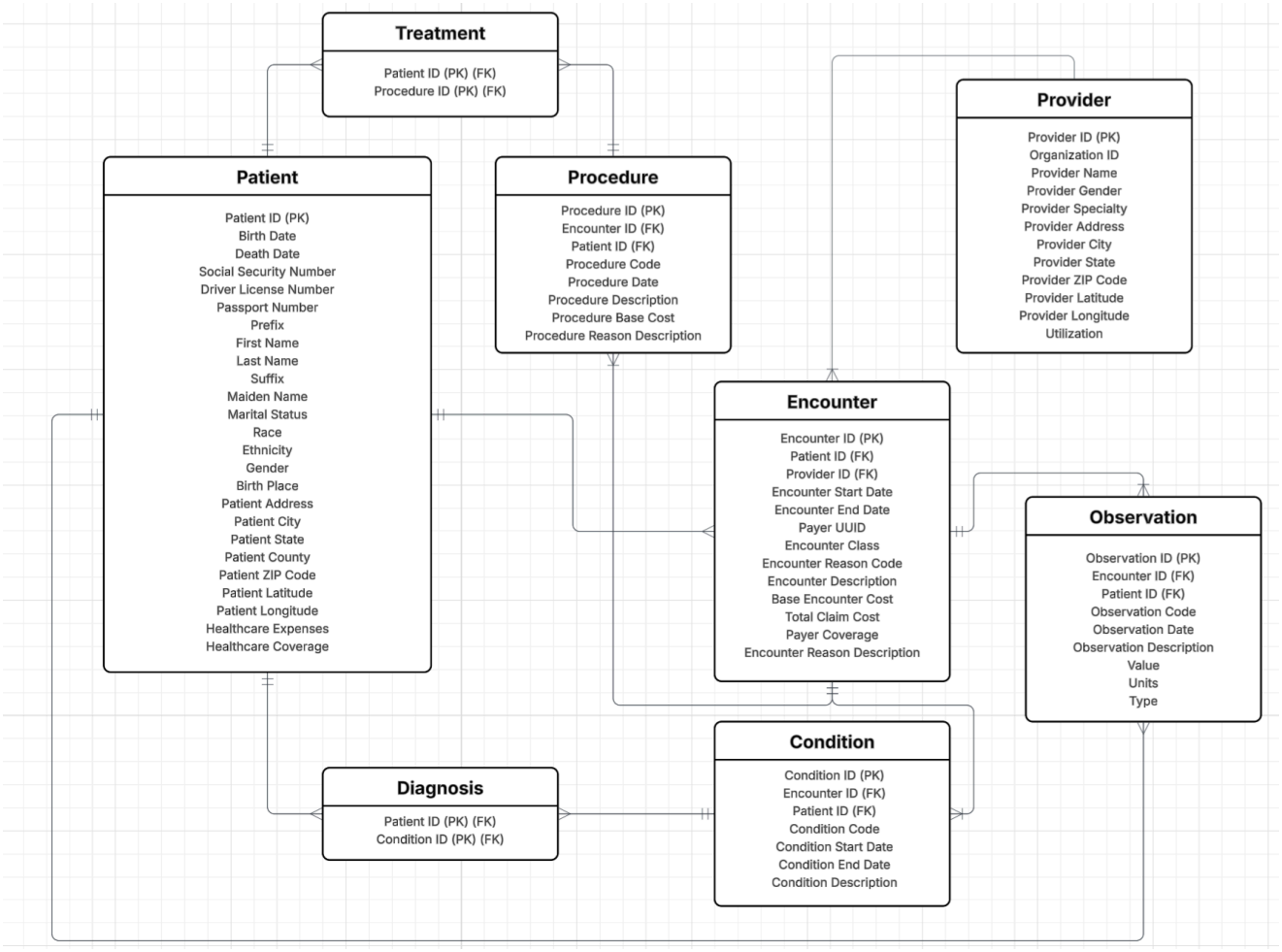


Two entities has many-to-many relationship: Patient and Procedures & Patient and Condition.



According to the result:
I checked for unique patients and conditions first to justify many-to-many relationships. Because, it shows that conditions are assigned to more than 1 patients, while each patient also has more than one condition. Same is said for the patient & procedures.

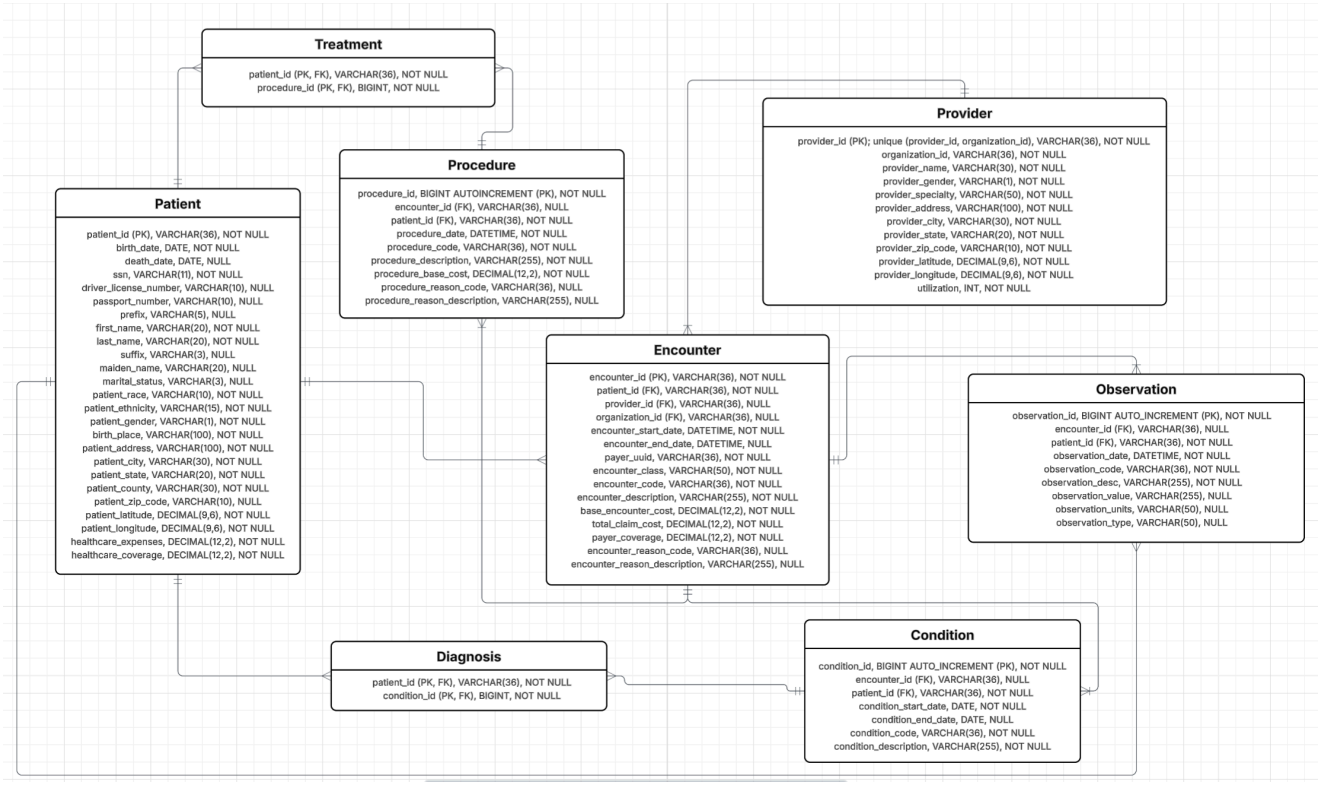
4. Develop a Logical Model using the Conceptual Model. Make sure you come up with a junction entity to resolve the many-to-many relationship.



Week 7 Final Project Report: Total Points - 100

Physical Model

5. Develop the physical model based on the Logical Model



6. Create tables using a database system. Insert data into the database tables. You must provide the DDL (CREATE TABLE statements), INSERT statements, and SELECT statements.

Details: Create the tables that you have come up with (the table must be based on the Physical Model).

- (a) Columns, Primary Key (PK), Data Type, and length, and NULL/NOT NULL need to be implemented, per the Physical Model.
- (b) Show the table definition (DDL) that you implemented (not in a graphical view).
- (c) Insert the complete set of data that you have come up with and show the insert statements used.

Created Tables

Patient

IMPORTANT NOTE: Instead of INSERTING one at a time, I've found a way to import csv files directly into SQL after creating the tables. So, I can bulk insert every single rows and columns by using **LOAD DATA INFILE**. Thankfully, it went well after a few tweaks.

```
CREATE TABLE patient (
  patient_id              VARCHAR(36)    NOT NULL PRI
  birth_date              DATE           NOT NULL,
  death_date              DATE           NULL,
  ssn                     VARCHAR(11)    NOT NULL,
  driver_license_number   VARCHAR(10)    NULL,
  passport_number         VARCHAR(10)    NULL,
  prefix                  VARCHAR(5)     NULL,
  first_name              VARCHAR(20)    NOT NULL,
  last_name               VARCHAR(20)    NOT NULL,
  suffix                  VARCHAR(3)     NULL,
  maiden_name             VARCHAR(20)    NULL,
  marital_status          VARCHAR(3)     NULL,
  patient_race            VARCHAR(10)    NOT NULL,
  patient_ethnicity       VARCHAR(15)    NOT NULL,
  patient_gender          VARCHAR(1)     NOT NULL,
  birth_place             VARCHAR(100)   NOT NULL,
  patient_address         VARCHAR(100)   NOT NULL,
  patient_city            VARCHAR(30)    NOT NULL,
  patient_state           VARCHAR(20)    NOT NULL,
  patient_county          VARCHAR(30)    NOT NULL,
  patient_zip_code        VARCHAR(10)    NULL,
  patient_latitude        DECIMAL(9,6)   NOT NULL,
  patient_longitude       DECIMAL(9,6)   NOT NULL,
  healthcare_expenses     DECIMAL(12,2)  NOT NULL,
  healthcare_coverage     DECIMAL(12,2)   NOT NULL
);
```

```
LOAD DATA INFILE '/var/lib/mysql-files/patients.csv'
INTO TABLE patient
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(@id, @birthdate, @deathdate, @ssn, @drivers, @passport, @prefix, @first, @last, @suffix,
 @maiden, @marital, @race, @ethnicity, @gender, @birthplace, @address, @city, @state, @county,
 @zip, @lat, @lon, @expenses, @coverage)
SET
  patient_id           = @id,
  birth_date           = STR_TO_DATE(@birthdate, '%Y-%m-%d'),
  death_date = NULLIF(STR_TO_DATE(NULLIF(@deathdate,''), '%Y-%m-%d'), NULL),
  ssn                  = @ssn,
  driver_license_number = @drivers,
  passport_number      = @passport,
  prefix               = @prefix,
  first_name           = @first,
  last_name            = @last,
  suffix               = @suffix,
  maiden_name          = @maiden,
  marital_status       = @marital,
  patient_race         = @race,
  patient_ethnicity    = @ethnicity,
  patient_gender       = @gender,
  birth_place          = @birthplace,
  patient_address      = @address,
  patient_city         = @city,
  patient_state        = @state,
  patient_county       = @county,
  patient_zip_code     = @zip,
  patient_latitude     = @lat,
  patient_longitude    = @lon,
  healthcare_expenses  = @expenses,
  healthcare_coverage  = @coverage;
```

```
SELECT * FROM patient LIMIT 100
```

Search Results | Export | Cost: 53ms | 1 2 3 4 ··· 12 | Total 1171

patient_id varchar(36)	birth_date date	death_date date	ssn varchar(11)	driver_license_numb varchar(10)	passport_number varchar(10)	prefix varchar(5)	first_name varchar(20)	last_na varchar(2
00185faa-2760-4218-9bf5-c	2003-11-18	(NULL)	999-50-8531	S99964760			Eusebio566	Wyman9
0042862c-9889-4a2e-b782-	2009-11-26	(NULL)	999-20-4613				Dewitt635	Feest10
0047123f-12e7-486c-82df-8	1960-01-20	(NULL)	999-92-5264	S99959789	X2594715X	Mr.	Jordon466	Harber2
010d4a3a-2316-45ed-ae15-	1998-05-31	(NULL)	999-21-2604	S99974819	X34193837X	Mr.	Patrick786	Hettinge
01207ecd-9dff-4754-8887-4	2019-05-15	(NULL)	999-81-4349				Karyn217	Mueller8

Provider

```sql
CREATE TABLE provider (
  provider_id          VARCHAR(36)  NOT NULL PRIMARY KEY,
  organization_id      VARCHAR(36)  NOT NULL,
  provider_name        VARCHAR(30)  NOT NULL,
  provider_gender      VARCHAR(1)   NOT NULL,
  provider_specialty   VARCHAR(50)  NOT NULL,
  provider_address     VARCHAR(100) NOT NULL,
  provider_city        VARCHAR(30)  NOT NULL,
  provider_state       VARCHAR(20)  NOT NULL,
  provider_zip_code    VARCHAR(10)  NOT NULL,
  provider_latitude    DECIMAL(9,6) NOT NULL,
  provider_longitude   DECIMAL(9,6) NOT NULL,
  utilization          INT NOT NULL
);

▷Run
ALTER TABLE provider
  ADD UNIQUE KEY uq_provider_org (provider_id, organization_id);
```

Note: I created the unique key, because each provider_id is unique while there are some duplicates of organization_id. By making an unique key with the paired provider_id and organization_id, I would be able to use both provider_id and organization_id as foreign keys.

```sql
▷Run | Select
LOAD DATA INFILE '/var/lib/mysql-files/providers.csv'
INTO TABLE provider
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(@id, @org, @name, @gender, @speciality, @address, @city, @state, @zip, @lat, @lon, @util)
SET
  provider_id          = @id,
  organization_id      = @org,
  provider_name        = @name,
  provider_gender      = LEFT(@gender,1),          -- enforce single char
  provider_specialty   = @speciality,              -- CSV header is "SPECIALITY"    "specialı
  provider_address     = @address,
  provider_city        = @city,
  provider_state       = @state,
  provider_zip_code    = @zip,
  provider_latitude    = @lat,
  provider_longitude   = @lon,
  utilization          = @util;
```

Encounter

```
CREATE TABLE encounter (
  encounter_id                VARCHAR(36)   NOT NULL,
  patient_id                  VARCHAR(36)   NOT NULL,
  provider_id                 VARCHAR(36)   NULL,
  organization_id             VARCHAR(36)   NULL,
  encounter_start_date        DATETIME          NOT NULL,
  encounter_end_date          DATETIME          NULL,
  payer_uuid                  VARCHAR(36)   NULL,
  encounter_class             VARCHAR(50)   NOT NULL,
  encounter_code              VARCHAR(36)   NOT NULL,
  encounter_description       VARCHAR(255)  NOT NULL,
  base_encounter_cost         DECIMAL(12,2) NOT NULL,
  total_claim_cost            DECIMAL(12,2) NOT NULL,
  payer_coverage              DECIMAL(12,2) NOT NULL,
  encounter_reason_code       VARCHAR(36)   NULL,
  encounter_reason_description  VARCHAR(255)  NULL,

  CONSTRAINT pk_encounter PRIMARY KEY (encounter_id),

  CONSTRAINT fk_encounter_patient
    FOREIGN KEY (patient_id) REFERENCES patient(patient_id)
    ON DELETE RESTRICT ON UPDATE CASCADE,

  CONSTRAINT fk_encounter_provider_org
    FOREIGN KEY (provider_id, organization_id)
    REFERENCES provider (provider_id, organization_id)
    ON DELETE SET NULL ON UPDATE CASCADE
);
```

```
▷Run | Select
LOAD DATA INFILE '/var/lib/mysql-files/encounters.csv'
INTO TABLE encounter
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(@id, @start, @stop, @patient, @org, @provider, @payer,
 @class, @code, @desc, @base_cost, @total_cost, @coverage, @reason_code, @reason_desc)
SET
  encounter_id                = @id,
  patient_id                  = @patient,
  provider_id                 = NULLIF(@provider,''),
  organization_id             = NULLIF(@org,''),
  encounter_start_date        = STR_TO_DATE(REPLACE(REPLACE(@start,'T',' '),'Z',''), '%Y-%m-%d %H:%i:%s'),
  encounter_end_date          = STR_TO_DATE(REPLACE(REPLACE(NULLIF(@stop,''),'T',' '),'Z',''), '%Y-%m-%d %H:%i:%s'),
  payer_uuid                  = NULLIF(@payer,''),
  encounter_class             = @class,
  encounter_code              = @code,
  encounter_description       = @desc,
  base_encounter_cost         = @base_cost,
  total_claim_cost            = @total_cost,
  payer_coverage              = @coverage,
  encounter_reason_code       = NULLIF(@reason_code,''),
  encounter_reason_description  = NULLIF(@reason_desc,'');
```

Observation

```
▷Run | Select
CREATE TABLE observation (
  observation_id       BIGINT AUTO_INCREMENT PRIMARY KEY,
  observation_date     DATETIME          NOT NULL,
  patient_id           VARCHAR(36)   NOT NULL,
  encounter_id         VARCHAR(36)   NULL,
  observation_code     VARCHAR(36)   NOT NULL,
  observation_desc     VARCHAR(255)  NOT NULL,
  observation_value    VARCHAR(255)  NULL,
  observation_units    VARCHAR(50)   NULL,
  observation_type     VARCHAR(50)   NULL,

  FOREIGN KEY (patient_id) REFERENCES patient(patient_id)
    ON DELETE RESTRICT ON UPDATE CASCADE,

  FOREIGN KEY (encounter_id) REFERENCES encounter(encounter_id)
    ON DELETE SET NULL ON UPDATE CASCADE
);
```

```
▷Run | Select
CREATE TABLE st_observation (
  `DATE`        VARCHAR(30),
  `PATIENT`     VARCHAR(36),
  `ENCOUNTER`   VARCHAR(36),
  `CODE`        VARCHAR(36),
  `DESCRIPTION` VARCHAR(255),
  `VALUE`       VARCHAR(255),
  `UNITS`       VARCHAR(50),
  `TYPE`        VARCHAR(50)
);

▷Run | Select
LOAD DATA INFILE '/var/lib/mysql-files/observations.csv'
INTO TABLE st_observation
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES;

▷Run | Select
INSERT INTO observation
  (observation_date, patient_id, encounter_id,
   observation_code, observation_desc, observation_value,
   observation_units, observation_type)
SELECT
  STR_TO_DATE(REPLACE(REPLACE(s.`DATE`,'T',' '),'Z',''), '%Y-%m-%d %H:%i:%s')     AS observation_date,
  s.`PATIENT`                                                                     AS patient_id,
  CASE WHEN e.encounter_id IS NULL THEN NULL ELSE s.`ENCOUNTER` END               AS encounter_id,
  s.`CODE`                                                                        AS observation_code,
  s.`DESCRIPTION`                                                                 AS observation_desc,
  NULLIF(s.`VALUE`,'')                                                            AS observation_value,
  NULLIF(s.`UNITS`,'')                                                            AS observation_units,
  NULLIF(s.`TYPE`,'')                                                             AS observation_type
FROM st_observation s
LEFT JOIN encounter e ON e.encounter_id = s.`ENCOUNTER`;

▷Run
DROP TABLE st_observation;
```

Note: I created this table st_observation, because it allows me to load the whole data into that staging table. Then, I INSERT and SELECT staged observation table into observation (intended) table WHILE LEFT JOIN on encounter_id from encounter table. Any found mismatched encounter_id will be inserted as NULL into observation table. Then I dropped st observation, because it is not needed after. Apparently, I did a bit of investigation and there are around 10.13% encounter_id in observation csv file not found in encounter csv file. I decided to keep these NULL values for future insights later if interested.

```
239   --verifying if this works
      ▷Run | +Tab | JSON | Select
240   SELECT
241       COUNT(*) AS total_rows,
242       SUM(encounter_id IS NULL) AS null_encounter_rows,
243       ROUND(SUM(encounter_id IS NULL) * 100.0 / COUNT(*), 2) AS pct_null_encounter
244   FROM observation;  86ms
```

observation

Search Results | Export | Cost: 94ms | 1

total_rows	null_encounter_rows	pct_null_encounter
299697	30363	10.13

Condition

```
▷ Run | Select
CREATE TABLE medical_condition (
  condition_id              BIGINT AUTO_INCREMENT PRIMARY KEY,
  condition_start_date      DATE          NOT NULL,
  condition_end_date        DATE          NULL,
  patient_id                VARCHAR(36)   NOT NULL,
  encounter_id              VARCHAR(36)   NULL,
  condition_code            VARCHAR(36)   NOT NULL,
  condition_description     VARCHAR(255)  NOT NULL,
  FOREIGN KEY (patient_id)   REFERENCES patient(patient_id)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  FOREIGN KEY (encounter_id) REFERENCES encounter(encounter_id)
    ON DELETE SET NULL ON UPDATE CASCADE
);
```

```
▷ Run | Select
LOAD DATA INFILE '/var/lib/mysql-files/conditions.csv'
INTO TABLE medical_condition
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(@start, @stop, @patient, @encounter, @code, @desc)
SET
  condition_start_date  = STR_TO_DATE(@start, '%Y-%m-%d'),
  condition_end_date    = STR_TO_DATE(NULLIF(@stop,''), '%Y-%m-%d'),
  patient_id            = @patient,
  encounter_id          = NULLIF(@encounter,''),
  condition_code        = @code,
  condition_description = @desc;
```

Procedure

```sql
CREATE TABLE procedures (
  procedure_id                BIGINT AUTO_INCREMENT PRIMARY KEY,
  procedure_date              DATETIME      NOT NULL,
  patient_id                  VARCHAR(36)   NOT NULL,
  encounter_id                VARCHAR(36)   NULL,
  procedure_code              VARCHAR(36)   NOT NULL,
  procedure_description       VARCHAR(255)  NOT NULL,
  procedure_base_cost         DECIMAL(12,2) NOT NULL,
  procedure_reason_code       VARCHAR(36)   NULL,
  procedure_reason_description VARCHAR(255)  NULL,
  FOREIGN KEY (patient_id)   REFERENCES patient(patient_id)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  FOREIGN KEY (encounter_id) REFERENCES encounter(encounter_id)
    ON DELETE SET NULL ON UPDATE CASCADE
);
```

```sql
▷Run | Select
LOAD DATA INFILE '/var/lib/mysql-files/procedures.csv'
INTO TABLE procedures
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(@date, @patient, @encounter, @code, @desc, @base_cost, @reason_code, @reason_desc)
SET
  procedure_date              = STR_TO_DATE(REPLACE(REPLACE(@date,'T',' '),'Z',''), '%Y-%m-%d %H:%i:%s'),
  patient_id                  = @patient,
  encounter_id                = NULLIF(@encounter,''),
  procedure_code              = @code,
  procedure_description       = @desc,
  procedure_base_cost         = CAST(NULLIF(@base_cost,'') AS DECIMAL(12,2)),
  procedure_reason_code       = NULLIF(@reason_code,''),
  procedure_reason_description = NULLIF(@reason_desc,'');
```

Junction Tables

```sql
▷Run | Select
CREATE TABLE diagnosis (
  patient_id   VARCHAR(36) NOT NULL,
  condition_id BIGINT      NOT NULL,
  PRIMARY KEY (patient_id, condition_id),
  FOREIGN KEY (patient_id)   REFERENCES patient(patient_id)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  FOREIGN KEY (condition_id) REFERENCES medical_condition(condition_id)
    ON DELETE CASCADE  ON UPDATE CASCADE
);
```

```
CREATE TABLE treatment (
  patient_id   VARCHAR(36) NOT NULL,
  procedure_id BIGINT      NOT NULL,
  PRIMARY KEY (patient_id, procedure_id),
  FOREIGN KEY (patient_id)   REFERENCES patient(patient_id)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  FOREIGN KEY (procedure_id) REFERENCES procedures(procedure_id)
    ON DELETE CASCADE  ON UPDATE CASCADE
);
```

Retrieving Data with SQL Queries

SQL Queries for Business Questions (Mostly)

7. Create a variety of SQL queries to retrieve data from one or many tables:

7.1 Provider utilization (encounters per provider & specialty)

```
1  CREATE TABLE rpt_provider_utilization AS
2  SELECT pr.provider_id,
3  pr.organization_id,
4  pr.provider_name,
5  pr.provider_specialty,
6  COUNT(e.encounter_id) AS n_encounters
7  FROM provider pr
8  LEFT JOIN encounter e
9  ON e.provider_id = pr.provider_id
10 AND e.organization_id = pr.organization_id
11 GROUP BY pr.provider_id, pr.organization_id, pr.provider_name,
   pr.provider_specialty;
12
13 SELECT * FROM rpt_provider_utilization
14 ORDER BY n_encounters DESC, provider_id;
```

Search Results | Export | Cost: 43ms | 1 2 3 4 ··· 59 | Total 5855

provider_id varchar(36)	organization_id varchar(36)	provider_name varchar(30)	provider_specialty varchar(50)	n_encounters bigint
8f9aea5b-fd01-37c0-8931-18b6d64bdae6	be4c63f3-8d38-3fa9-a183-62045b5c85f6	Gaynell126 Streich926	GENERAL PRACTICE	3217
793c18cd-8269-387d-b998-c135459e4248	d311e70d-86e7-3c03-b115-53892bcf7ef1	Gertrudis163 Schaden604	GENERAL PRACTICE	1972
f4eb93d1-9187-3cfb-83a4-6d9cd77f7df6	d692e283-0833-3201-8e55-4f868a9c0736	Vern731 Powlowski563	GENERAL PRACTICE	1658
e6283e46-fd81-3611-9459-0edb1c3da357	e002090d-4e92-300e-b41e-7d1f21dee4c6	Jeanmarie510 Beatty507	GENERAL PRACTICE	1028
4b04cd2f-3f27-35bc-8069-f4ca6339529f	3d10019f-c88e-3de5-9916-6107b9c0263d	Maile198 Frami345	GENERAL PRACTICE	955
40d7ff6c-5040-39aa-a876-19f1ccc6740c	3bd5eda0-16da-3ba5-8500-4dfd6ae118b8	Luke971 Rath779	GENERAL PRACTICE	859
0a8a9359-7b33-3256-a068-b5a7d18ebe4b	465de31f-3098-365c-af70-48a071e1f5aa	Keri25 Schmidt332	GENERAL PRACTICE	857
7bd4e666-a82d-3ad1-bc7c-b49eb726577b	24cb4eab-6166-3530-bddc-a5a8a14a4fc1	Lonna614 Dietrich576	GENERAL PRACTICE	824
af01a385-31d3-3c77-8fdb-2867fe88df2f	5d4b9df1-93ae-3bc9-b680-03249990e558	Garth972 Wyman904	GENERAL PRACTICE	783
0359f968-d1a6-30eb-b1cc-e6cc0b4d3513	5e765f2b-e908-3888-9fc7-df2cb87beb58	Gaynell126 Streich926	GENERAL PRACTICE	763

7.2 Inpatient length of stay (LOS) per provider

```sql
CREATE TABLE rpt_inpatient_los_provider AS
SELECT pr.provider_id,
pr.organization_id,
pr.provider_name,

ROUND(AVG(TIMESTAMPDIFF(HOUR, e.encounter_start_date,
e.encounter_end_date))/24, 2) AS avg_los_days,
COUNT(*) AS n_inpatient_encounters
FROM encounter e
JOIN provider pr
ON pr.provider_id = e.provider_id
AND pr.organization_id = e.organization_id
WHERE LOWER(e.encounter_class) = 'inpatient'
AND e.encounter_end_date IS NOT NULL
GROUP BY pr.provider_id, pr.organization_id, pr.provider_name;

SELECT * FROM rpt_inpatient_los_provider
ORDER BY avg_los_days DESC, provider_id;
```

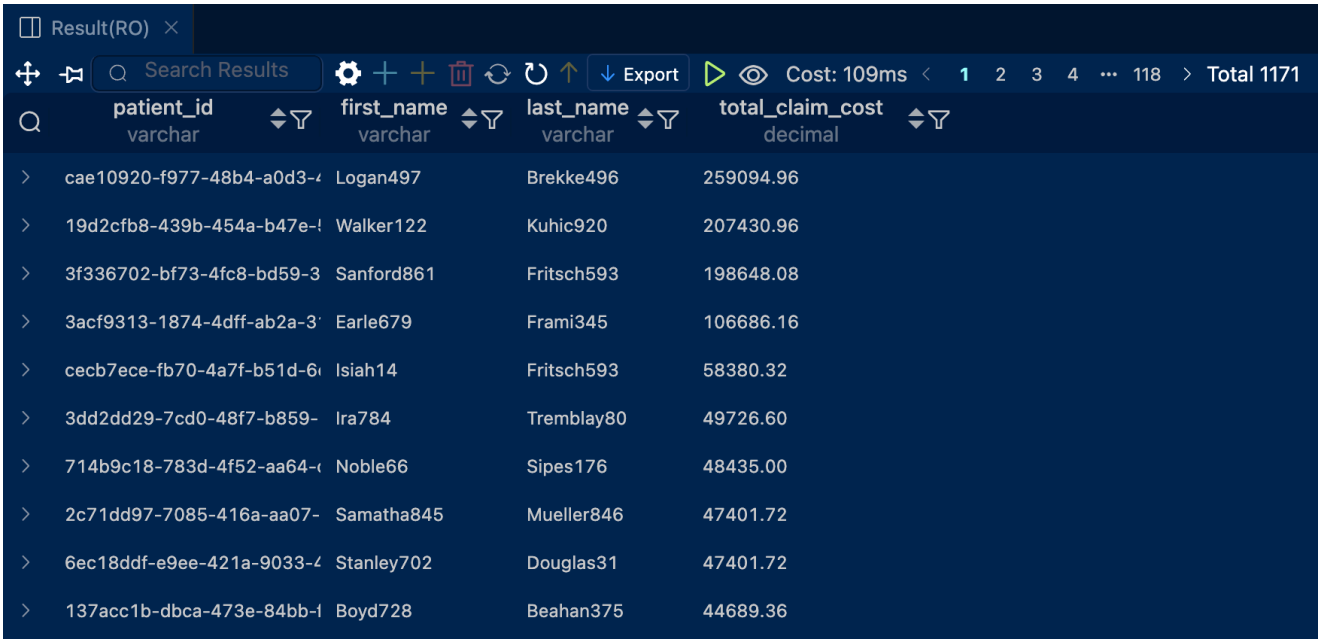
rpt_inpatient_los_provider

Search Results | Export | Cost: 30ms | 1 | Total 72

provider_id varchar(36)	organization_id varchar(36)	provider_name varchar(30)	avg_los_days decimal(23,2)	n_inpatient_encour bigint
1045937e-4c7c-3dee-81de-	f71ce1a9-cca7-3295-9a1e-c	Jc393 Borer986	578.66	35
40d7ff6c-5040-39aa-a876-1	3bd5eda0-16da-3ba5-8500-	Luke971 Rath779	232.84	46
77a7881d-6dd5-32e1-9e18-	ef6ab57c-ed94-3dbe-9861-8	Phillip440 McCullough561	8.38	30
1595224e-648a-3615-aae6-	161a799c-4894-36f6-8ca7-;	Quincy153 Heathcote539	2.54	2
8f9aea5b-fd01-37c0-8931-1	be4c63f3-8d38-3fa9-a183-6	Gaynell126 Streich926	2.50	2
b6c72690-131c-355f-91d0-	8881e3ee-e915-31b3-867f-8	Lynwood354 Monahan736	2.42	52
1b75d7a2-3964-340e-b025-	7fb56531-86bd-3e4a-8619-	Jeramy610 Littel644	2.04	6
8ee28b4a-9018-3065-9f6b-(	331f4c11-d298-308b-aaa1-(	Malinda718 Cassin499	2.04	1
ea53dda3-3bf7-30c9-804b-a	a9f20dc1-5147-3789-bcef-b	Judith460 Johns824	2.02	2
69e585d1-1d1f-39ed-a9e8-b	1bd44727-a8e3-322d-9c14-	Suzette512 Monahan736	2.00	1

7.3 Top patients by total claim cost

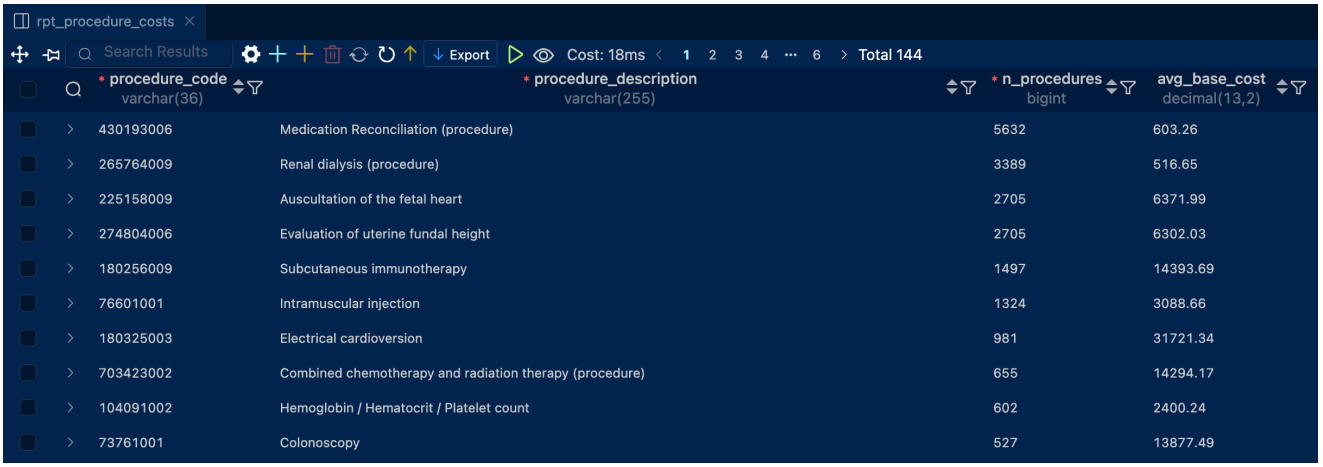
```sql
SELECT p.patient_id,
p.first_name, p.last_name,
SUM(e.total_claim_cost) AS total_claim_cost
FROM patient p
JOIN encounter e ON e.patient_id = p.patient_id
GROUP BY p.patient_id, p.first_name, p.last_name
ORDER BY total_claim_cost DESC, p.patient_id
LIMIT 10;
```



patient_id varchar	first_name varchar	last_name varchar	total_claim_cost decimal
cae10920-f977-48b4-a0d3-4	Logan497	Brekke496	259094.96
19d2cfb8-439b-454a-b47e-!	Walker122	Kuhic920	207430.96
3f336702-bf73-4fc8-bd59-3	Sanford861	Fritsch593	198648.08
3acf9313-1874-4dff-ab2a-3'	Earle679	Frami345	106686.16
cecb7ece-fb70-4a7f-b51d-6(	Isiah14	Fritsch593	58380.32
3dd2dd29-7cd0-48f7-b859-	Ira784	Tremblay80	49726.60
714b9c18-783d-4f52-aa64-(	Noble66	Sipes176	48435.00
2c71dd97-7085-416a-aa07-	Samatha845	Mueller846	47401.72
6ec18ddf-e9ee-421a-9033-4	Stanley702	Douglas31	47401.72
137acc1b-dbca-473e-84bb-(	Boyd728	Beahan375	44689.36

7.4 Procedure volume & average base cost

```sql
CREATE TABLE rpt_procedure_costs AS
SELECT prc.procedure_code,
prc.procedure_description,
COUNT(*) AS n_procedures,
ROUND(AVG(prc.procedure_base_cost), 2) AS avg_base_cost
FROM procedures prc
GROUP BY prc.procedure_code, prc.procedure_description;

SELECT * FROM rpt_procedure_costs
ORDER BY n_procedures DESC, procedure_code
LIMIT 25;
```



* procedure_code varchar(36)	* procedure_description varchar(255)	* n_procedures bigint	avg_base_cost decimal(13,2)
430193006	Medication Reconciliation (procedure)	5632	603.26
265764009	Renal dialysis (procedure)	3389	516.65
225158009	Auscultation of the fetal heart	2705	6371.99
274804006	Evaluation of uterine fundal height	2705	6302.03
180256009	Subcutaneous immunotherapy	1497	14393.69
76601001	Intramuscular injection	1324	3088.66
180325003	Electrical cardioversion	981	31721.34
703423002	Combined chemotherapy and radiation therapy (procedure)	655	14294.17
104091002	Hemoglobin / Hematocrit / Platelet count	602	2400.24
73761001	Colonoscopy	527	13877.49

7.5 Post-procedure follow-up within 14 days (rate)

```sql
CREATE TABLE rpt_followup_14d_by_proc_code AS
WITH proc_flags AS (
SELECT prc.procedure_id,
prc.procedure_code,
```

```
prc.procedure_description,
CASE
WHEN EXISTS (
SELECT 1
FROM observation o
WHERE o.patient_id = prc.patient_id
AND o.observation_date > prc.procedure_date
AND o.observation_date <= DATE_ADD(prc.procedure_date, INTERVAL 14 DAY)
) THEN 1 ELSE 0
END AS has_followup_14d
FROM procedures prc
)
SELECT procedure_code,
procedure_description,
COUNT(*) AS n_procedures,
SUM(has_followup_14d) AS n_with_followup_14d,
ROUND(100 * SUM(has_followup_14d) / NULLIF(COUNT(*),0), 2) AS pct_with_followup_14d
FROM proc_flags
GROUP BY procedure_code, procedure_description;

SELECT * FROM rpt_followup_14d_by_proc_code
ORDER BY pct_with_followup_14d ASC, n_procedures DESC
LIMIT 25;
```

rpt_followup_14d_by_proc_code

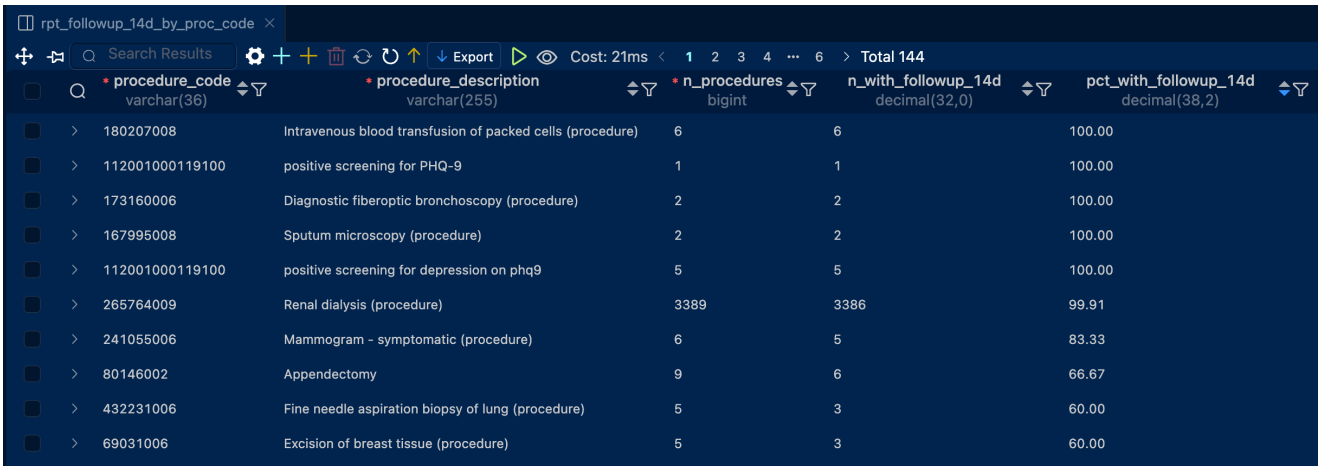
Search Results | Export | Cost: 21ms | 1 2 3 4 ... 6 | Total 144

* procedure_code varchar(36)	* procedure_description varchar(255)	* n_procedures bigint	n_with_followup_14d decimal(32,0)	pct_with_followup_14d decimal(38,2)
180207008	Intravenous blood transfusion of packed cells (procedure)	6	6	100.00
112001000119100	positive screening for PHQ-9	1	1	100.00
173160006	Diagnostic fiberoptic bronchoscopy (procedure)	2	2	100.00
167995008	Sputum microscopy (procedure)	2	2	100.00
112001000119100	positive screening for depression on phq9	5	5	100.00
265764009	Renal dialysis (procedure)	3389	3386	99.91
241055006	Mammogram - symptomatic (procedure)	6	5	83.33
80146002	Appendectomy	9	6	66.67
432231006	Fine needle aspiration biopsy of lung (procedure)	5	3	60.00
69031006	Excision of breast tissue (procedure)	5	3	60.00

7.6 High-risk signal: frequent emergency users (>=3 ER visits)

```
CREATE TABLE rpt_er_frequenters AS
SELECT
p.patient_id,
p.first_name,
p.last_name,
COUNT(*) AS er_visits
FROM encounter e
JOIN patient p
```

```
ON p.patient_id = e.patient_id
WHERE LOWER(e.encounter_class) = 'emergency'
GROUP BY p.patient_id, p.first_name, p.last_name
HAVING COUNT(*) >= 3;

SELECT *
FROM rpt_er_frequenters
ORDER BY er_visits DESC, last_name, first_name, patient_id;
```



rpt_followup_14d_by_proc_code

Search Results | Export | Cost: 21ms | 1 2 3 4 ... 6 | Total 144

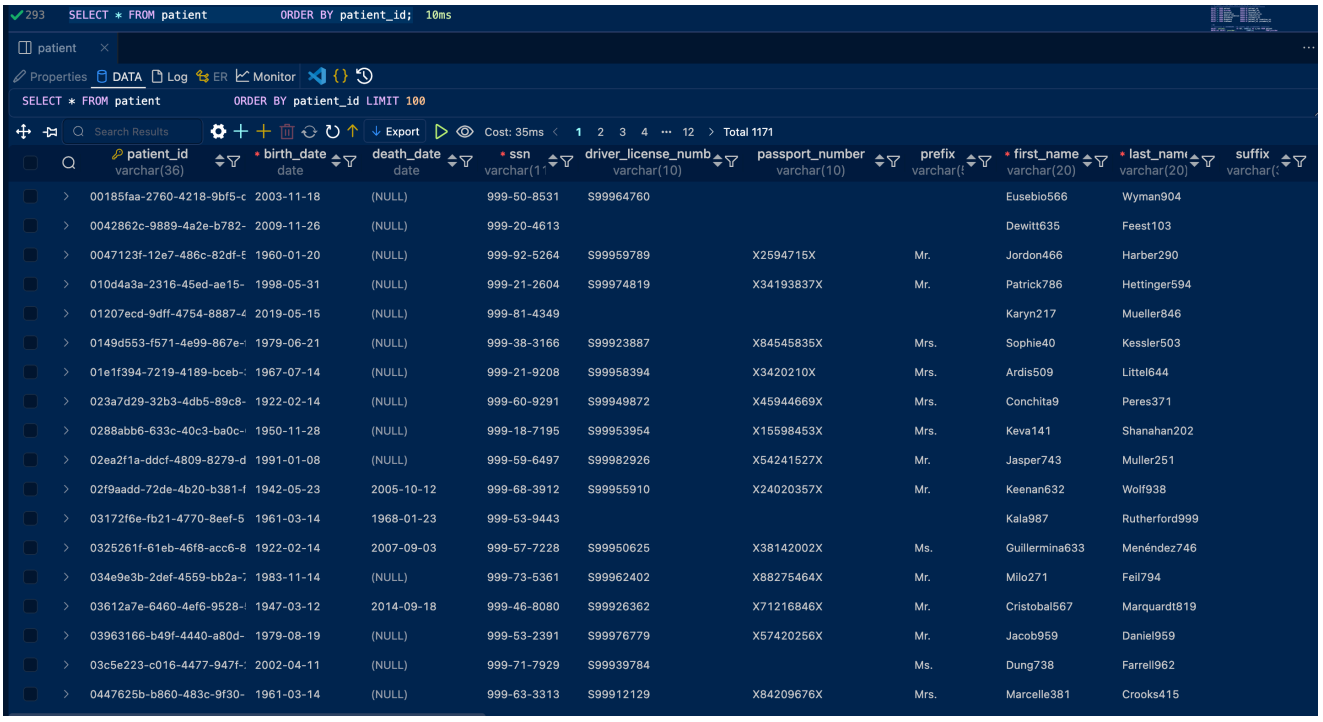
procedure_code varchar(36)	procedure_description varchar(255)	n_procedures bigint	n_with_followup_14d decimal(32,0)	pct_with_followup_14d decimal(38,2)
180207008	Intravenous blood transfusion of packed cells (procedure)	6	6	100.00
112001000119100	positive screening for PHQ-9	1	1	100.00
173160006	Diagnostic fiberoptic bronchoscopy (procedure)	2	2	100.00
167995008	Sputum microscopy (procedure)	2	2	100.00
112001000119100	positive screening for depression on phq9	5	5	100.00
265764009	Renal dialysis (procedure)	3389	3386	99.91
241055006	Mammogram - symptomatic (procedure)	6	5	83.33
80146002	Appendectomy	9	6	66.67
432231006	Fine needle aspiration biopsy of lung (procedure)	5	3	60.00
69031006	Excision of breast tissue (procedure)	5	3	60.00

Each Retrieved Tables

8. Retrieve the data from each table by using the SELECT * statement and order by PK column(s).
    - Show the output. Make sure you show the print screen of the complete set of rows and columns. The rows must be ordered by the PK column(s).

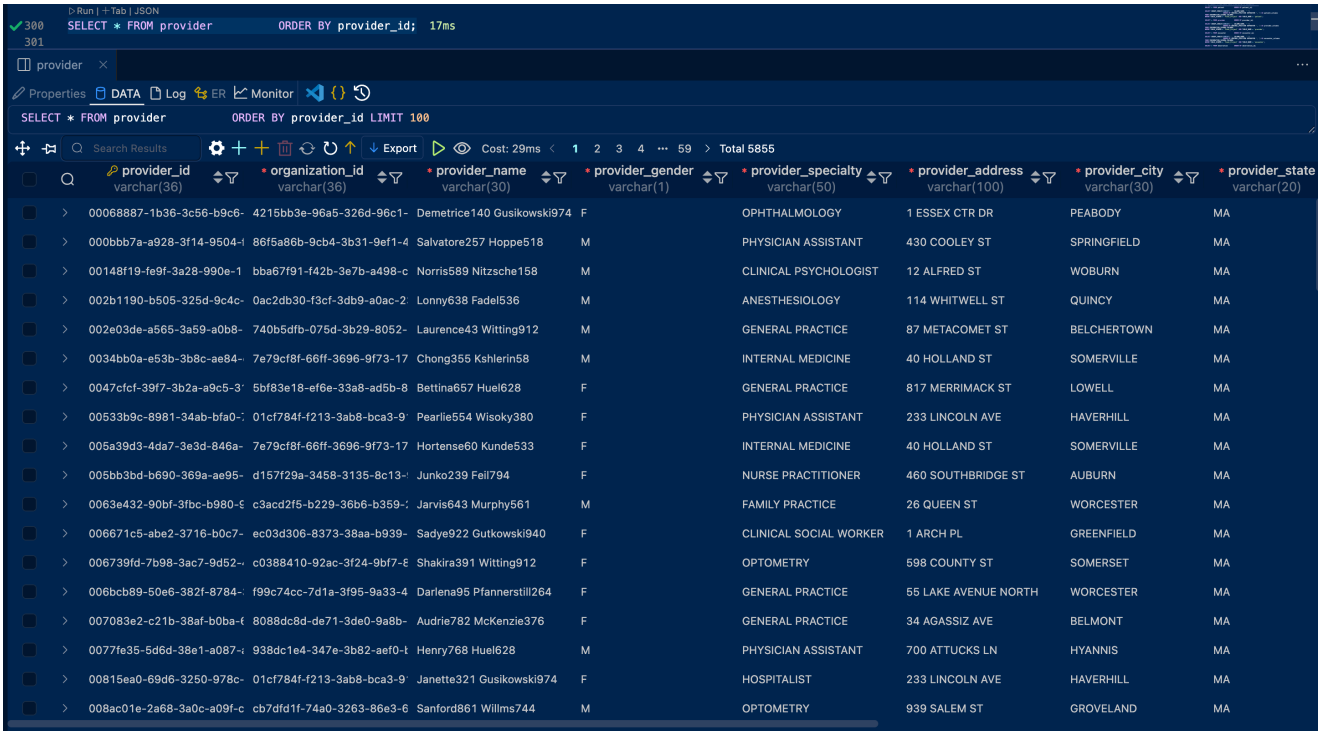
```
SELECT * FROM patient ORDER BY patient_id;

SELECT * FROM provider ORDER BY provider_id;

SELECT * FROM encounter ORDER BY encounter_id;

SELECT * FROM observation ORDER BY observation_id;

SELECT * FROM medical_condition ORDER BY condition_id;

SELECT * FROM procedures ORDER BY procedure_id;

SELECT * FROM diagnosis ORDER BY patient_id, condition_id;

SELECT * FROM treatment ORDER BY patient_id, procedure_id;
```

Patient Table



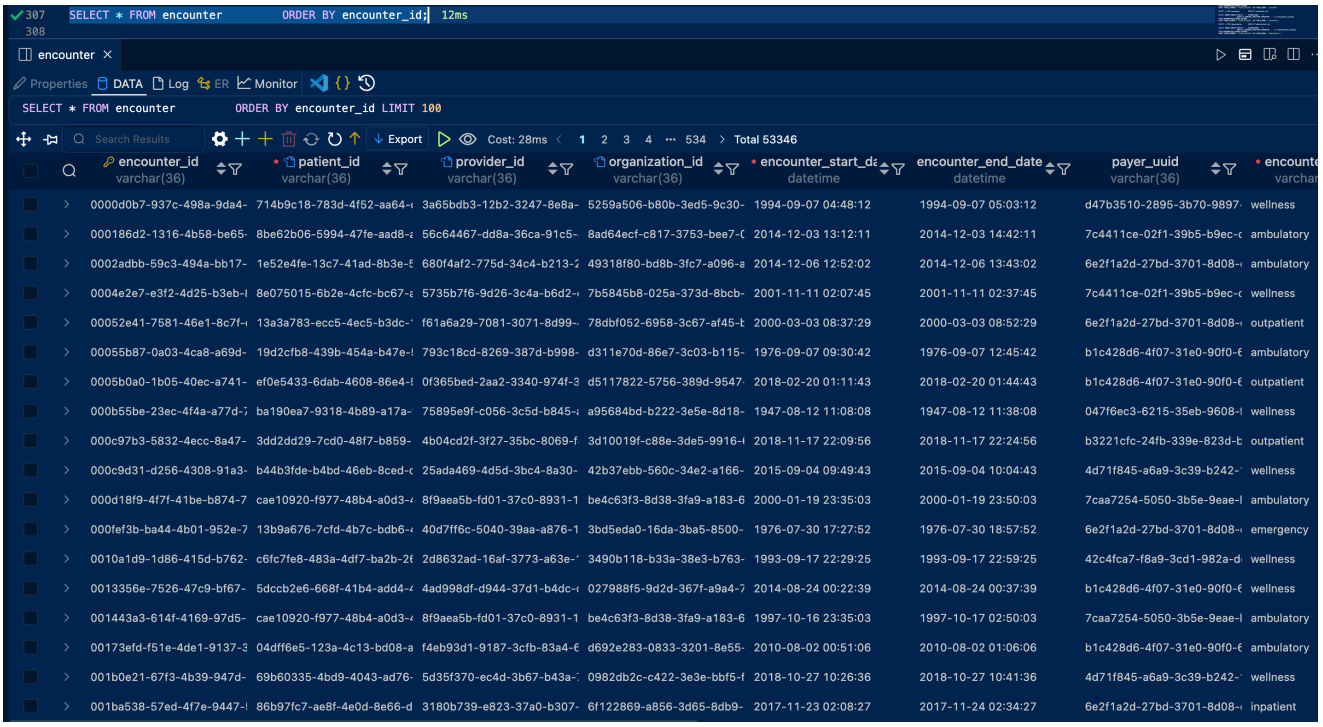
patient_columns	`patient_id`, `birth_date`, `death_date`, `ssn`, `driver_license_number`, `passport_number`, `prefix`, `first_name`, `last_name`, `suffix`, `maiden_name`, `marital_status`, `patient_race`, `patient_ethnicity`, `patient_gender`, `birth_place`, `patient_address`, `patient_city`, `patient_state`, `patient_county`, `patient_zip_code`, `patient_latitude`, `patient_longitude`, `healthcare_expenses`, `healthcare_coverage`

Provider Table



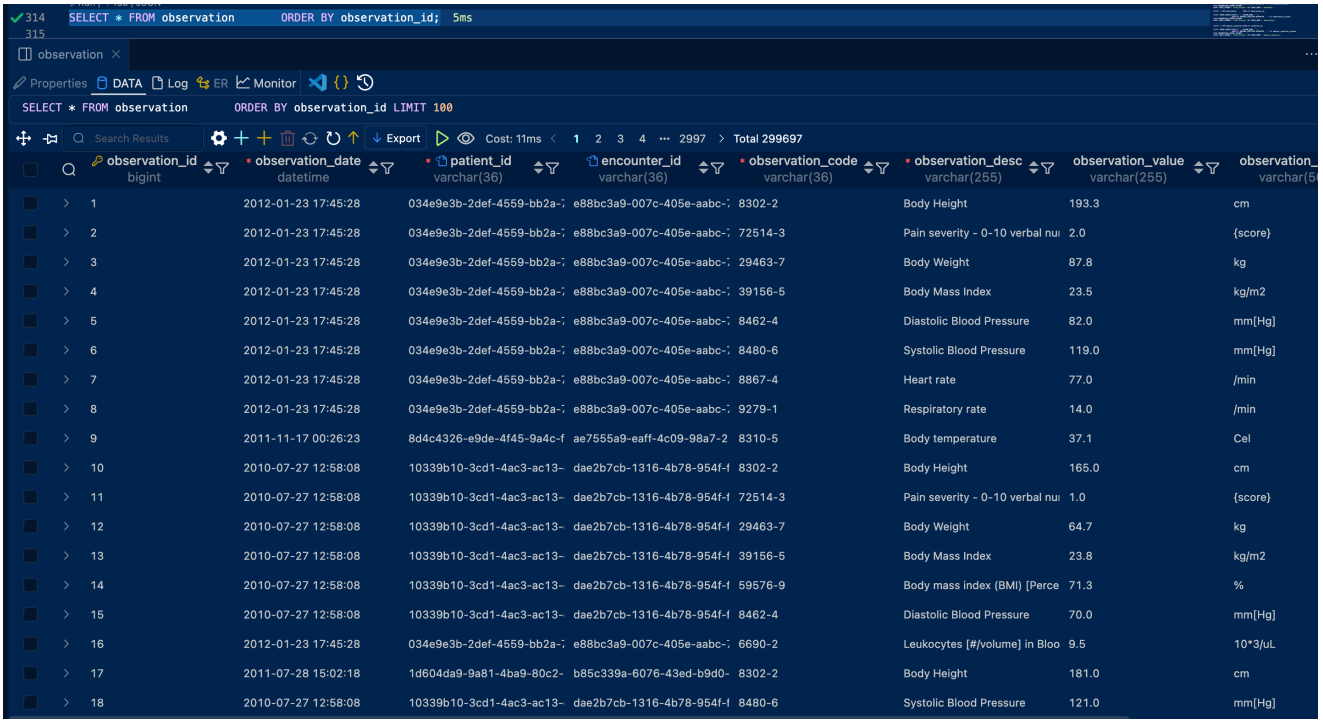
provider_columns	`provider_id`, `organization_id`, `provider_name`, `provider_gender`, `provider_specialty`, `provider_address`, `provider_city`, `provider_state`, `provider_zip_code`, `provider_latitude`, `provider_longitude`, `utilization`

Encounter Table



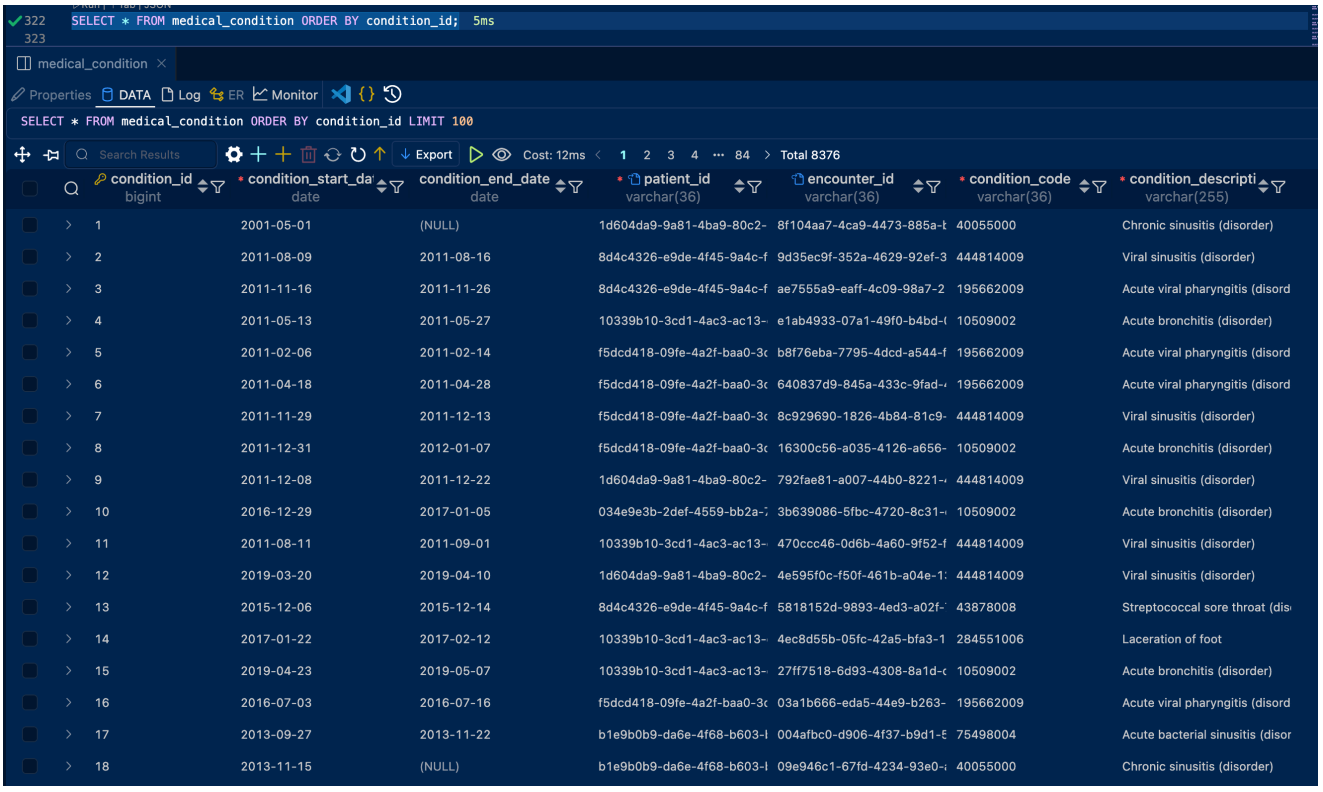
encounter_columns	`encounter_id`, `patient_id`, `provider_id`, `organization_id`, `encounter_start_date`, `encounter_end_date`, `payer_uuid`, `encounter_class`, `encounter_code`, `encounter_description`, `base_encounter_cost`, `total_claim_cost`, `payer_coverage`, `encounter_reason_code`, `encounter_reason_description`

Observation Table



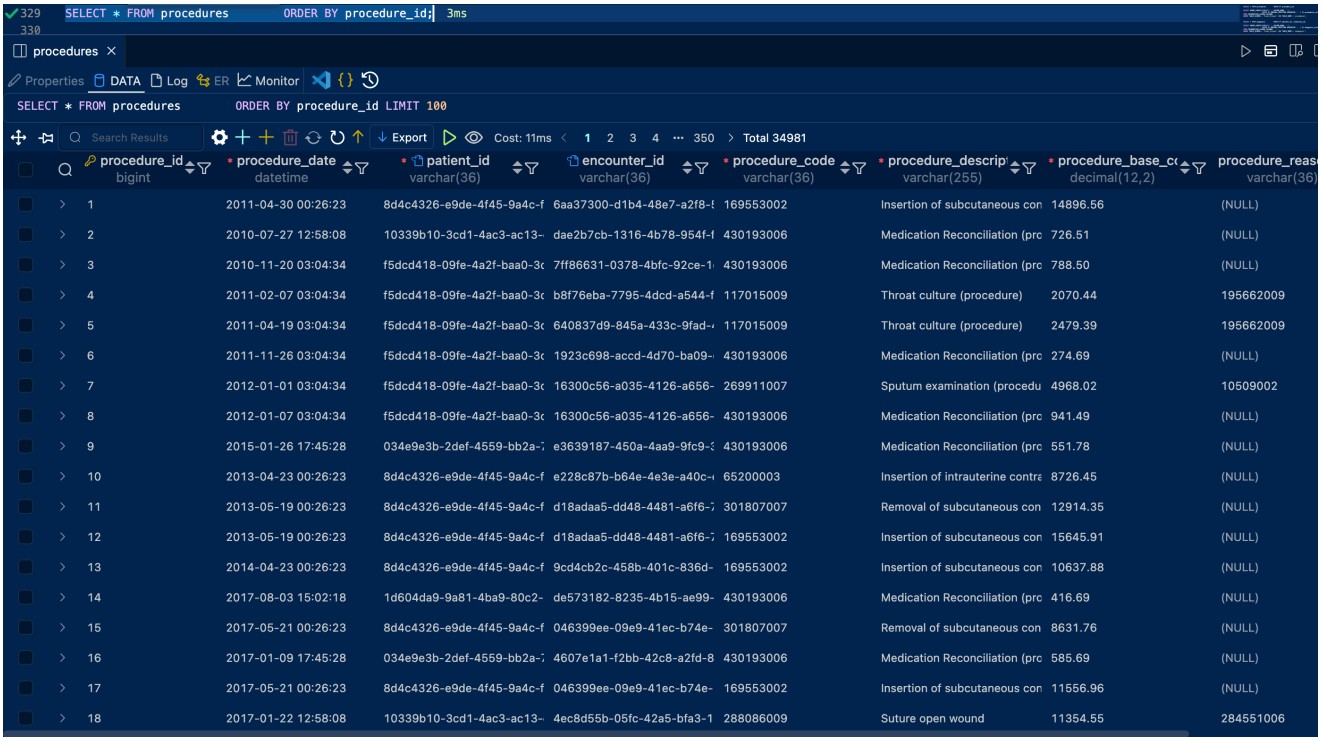
observation_columns	`observation_id`, `observation_date`, `patient_id`, `encounter_id`, `observation_code`, `observation_desc`, `observation_value`, `observation_units`, `observation_type`

Medical Conditions Table



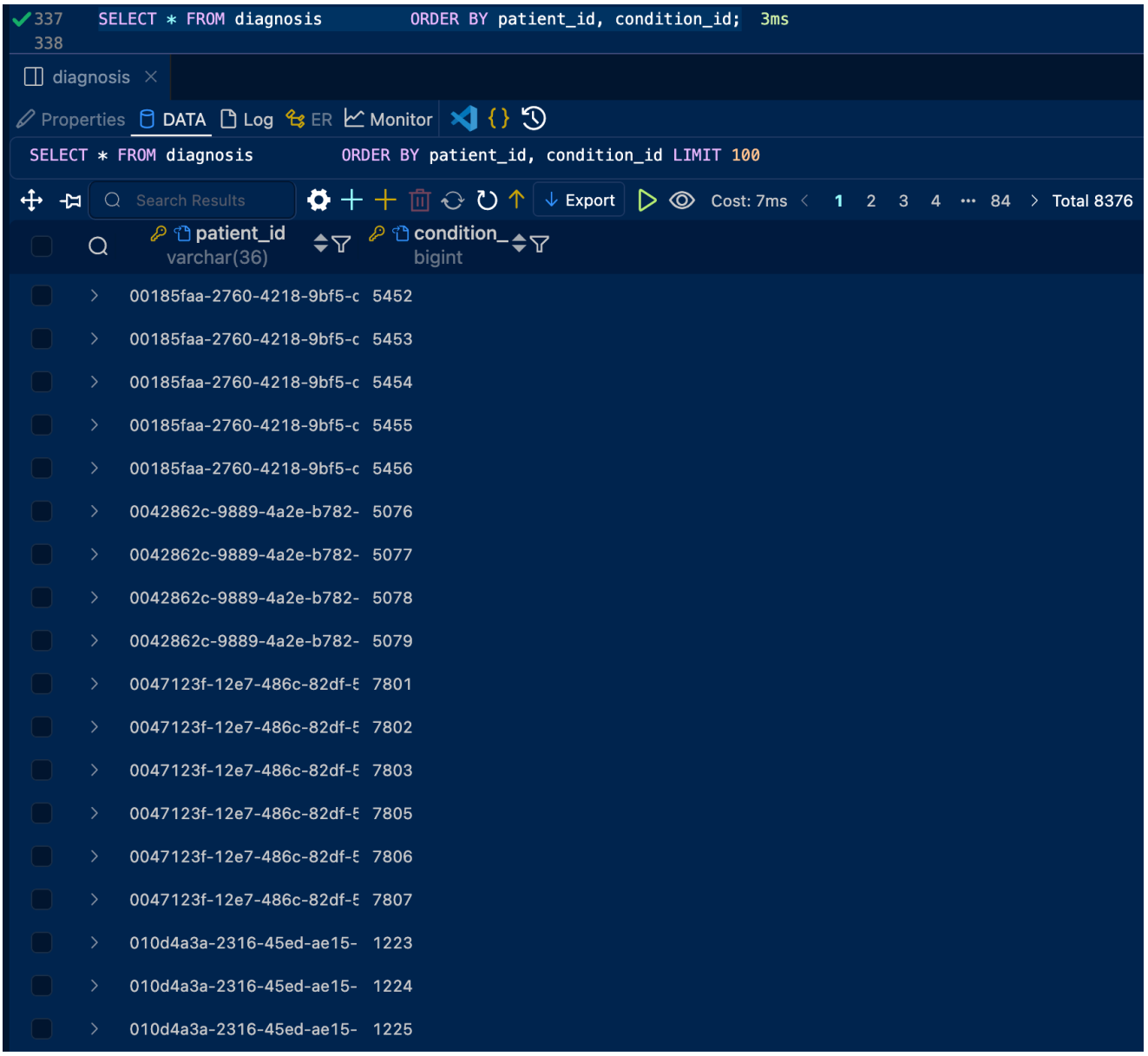
medical_condition_columns	`condition_id`, `condition_start_date`, `condition_end_date`, `patient_id`, `encounter_id`, `condition_code`, `condition_description`

Procedures Table



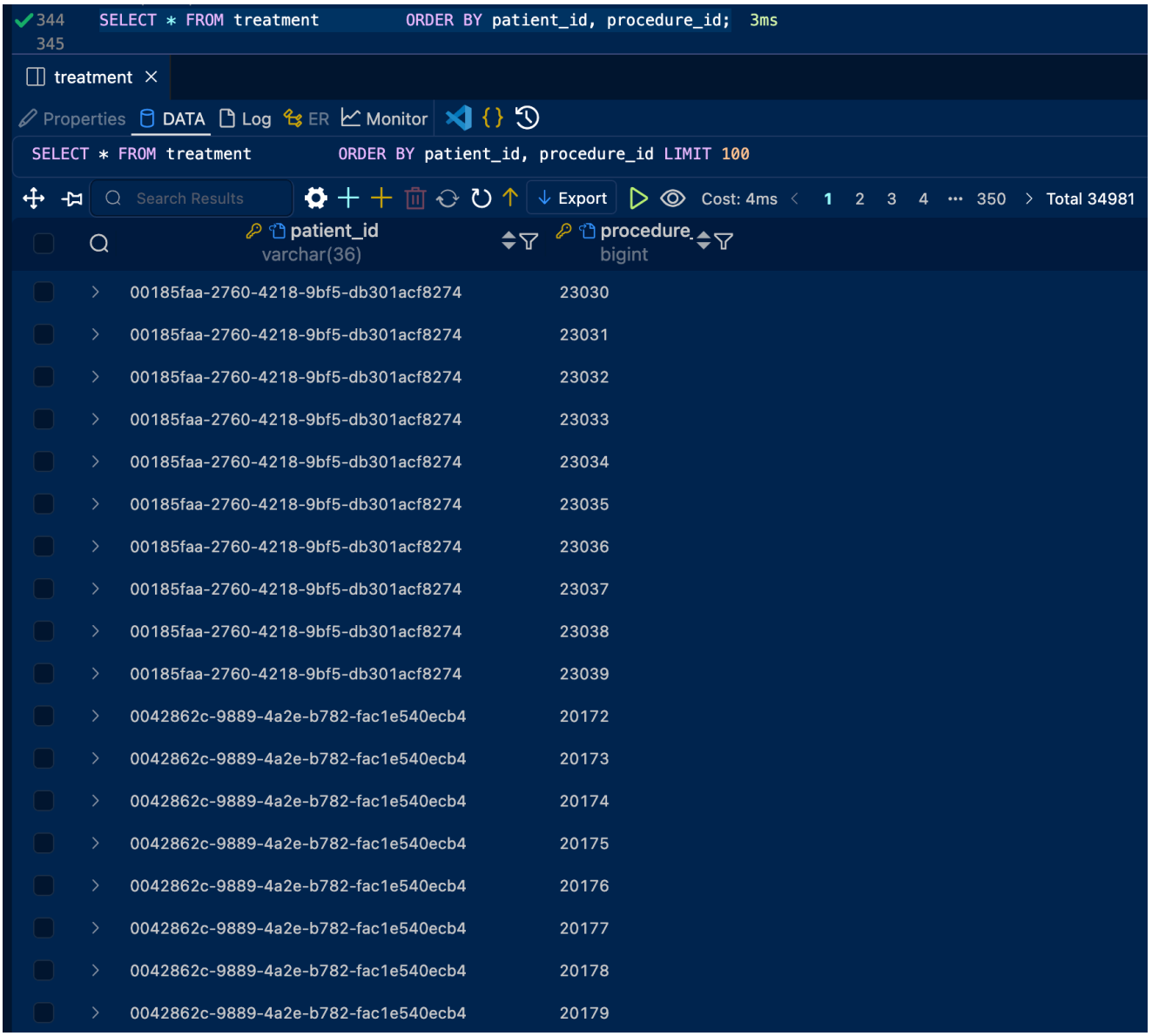
procedures_columns	`procedure_id`, `procedure_date`, `patient_id`, `encounter_id`, `procedure_code`, `procedure_description`, `procedure_base_cost`, `procedure_reason_code`, `procedure_reason_description`

Diagnosis Table



diagnosis_columns	`patient_id`, `condition_id`

Treatment Table



```
344  SELECT * FROM treatment        ORDER BY patient_id, procedure_id;  3ms
345
```

treatment

Properties | DATA | Log | ER | Monitor

```
SELECT * FROM treatment        ORDER BY patient_id, procedure_id LIMIT 100
```

Search Results | Export | Cost: 4ms | 1 2 3 4 ··· 350 | Total 34981

patient_id varchar(36)	procedure_ bigint
00185faa-2760-4218-9bf5-db301acf8274	23030
00185faa-2760-4218-9bf5-db301acf8274	23031
00185faa-2760-4218-9bf5-db301acf8274	23032
00185faa-2760-4218-9bf5-db301acf8274	23033
00185faa-2760-4218-9bf5-db301acf8274	23034
00185faa-2760-4218-9bf5-db301acf8274	23035
00185faa-2760-4218-9bf5-db301acf8274	23036
00185faa-2760-4218-9bf5-db301acf8274	23037
00185faa-2760-4218-9bf5-db301acf8274	23038
00185faa-2760-4218-9bf5-db301acf8274	23039
0042862c-9889-4a2e-b782-fac1e540ecb4	20172
0042862c-9889-4a2e-b782-fac1e540ecb4	20173
0042862c-9889-4a2e-b782-fac1e540ecb4	20174
0042862c-9889-4a2e-b782-fac1e540ecb4	20175
0042862c-9889-4a2e-b782-fac1e540ecb4	20176
0042862c-9889-4a2e-b782-fac1e540ecb4	20177
0042862c-9889-4a2e-b782-fac1e540ecb4	20178
0042862c-9889-4a2e-b782-fac1e540ecb4	20179

treatment_columns	`patient_id`, `procedure_id`

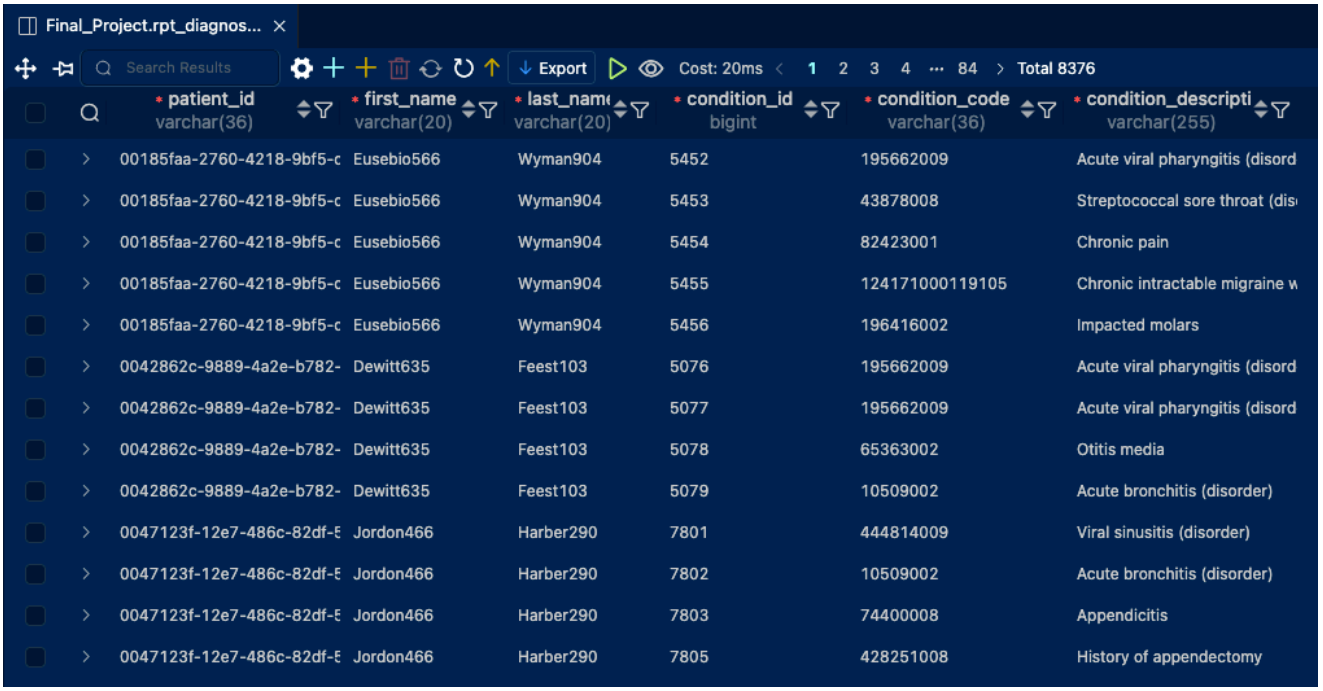
SQL Queries

2) INNER JOIN

2. Write an SQL involving the junction table and two other related tables. You must use the INNER JOIN to connect with all three tables. The database that you created must be included in your SQL queries.

```
CREATE TABLE Final_Project.rpt_diagnosis_patient_condition AS
SELECT
  d.patient_id,
  p.first_name,
  p.last_name,
  mc.condition_id,
  mc.condition_code,
```

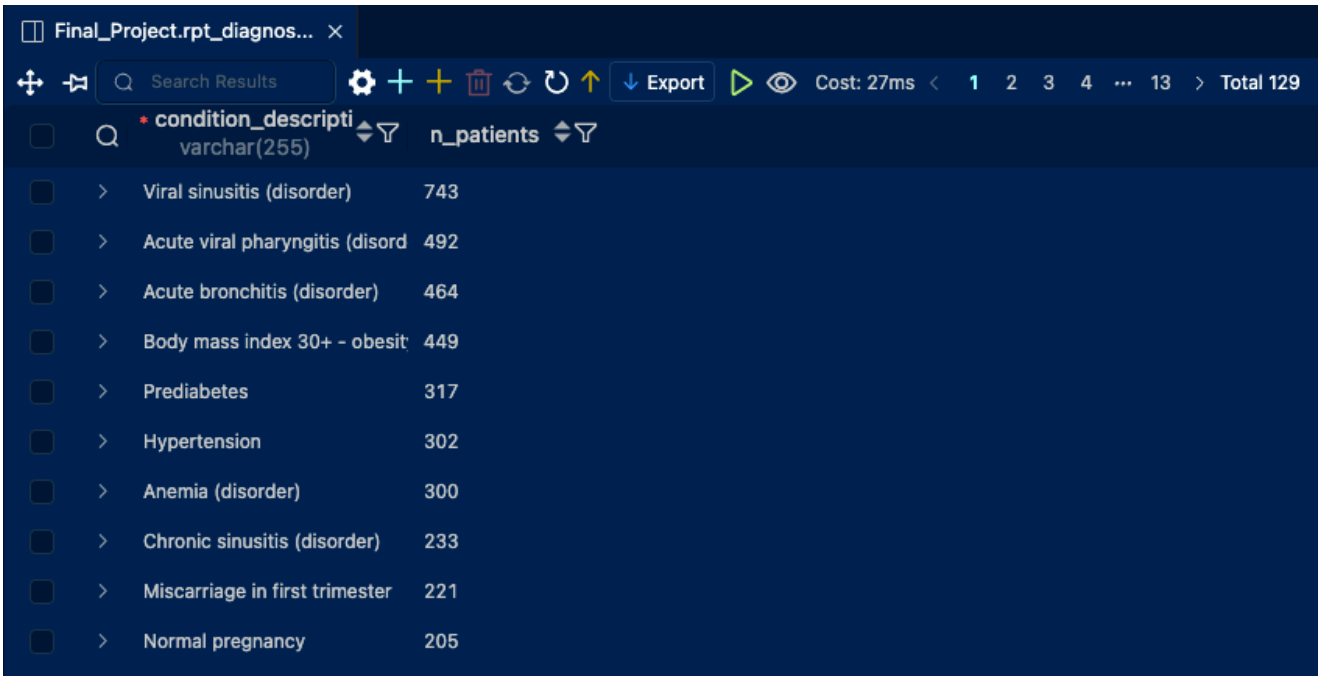
```
    mc.condition_description
FROM Final_Project.diagnosis AS d
INNER JOIN Final_Project.patient AS p
    ON p.patient_id = d.patient_id
INNER JOIN Final_Project.medical_condition AS mc
    ON mc.condition_id = d.condition_id;

SELECT *
FROM Final_Project.rpt_diagnosis_patient_condition
ORDER BY patient_id, condition_id;

-- top 10 conditions by DISTINCT patients (prevalence)
SELECT
    condition_description,
    COUNT(DISTINCT patient_id) AS n_patients
FROM Final_Project.rpt_diagnosis_patient_condition
GROUP BY condition_description
ORDER BY n_patients DESC, condition_description
LIMIT 10;
```



Final_Project.rpt_diagnos...

Search Results | Export | Cost: 20ms | 1 2 3 4 ··· 84 | Total 8376

patient_id varchar(36)	first_name varchar(20)	last_nam varchar(20)	condition_id bigint	condition_code varchar(36)	condition_descripti varchar(255)
00185faa-2760-4218-9bf5-c	Eusebio566	Wyman904	5452	195662009	Acute viral pharyngitis (disord
00185faa-2760-4218-9bf5-c	Eusebio566	Wyman904	5453	43878008	Streptococcal sore throat (dis
00185faa-2760-4218-9bf5-c	Eusebio566	Wyman904	5454	82423001	Chronic pain
00185faa-2760-4218-9bf5-c	Eusebio566	Wyman904	5455	124171000119105	Chronic intractable migraine w
00185faa-2760-4218-9bf5-c	Eusebio566	Wyman904	5456	196416002	Impacted molars
0042862c-9889-4a2e-b782-	Dewitt635	Feest103	5076	195662009	Acute viral pharyngitis (disord
0042862c-9889-4a2e-b782-	Dewitt635	Feest103	5077	195662009	Acute viral pharyngitis (disord
0042862c-9889-4a2e-b782-	Dewitt635	Feest103	5078	65363002	Otitis media
0042862c-9889-4a2e-b782-	Dewitt635	Feest103	5079	10509002	Acute bronchitis (disorder)
0047123f-12e7-486c-82df-E	Jordon466	Harber290	7801	444814009	Viral sinusitis (disorder)
0047123f-12e7-486c-82df-E	Jordon466	Harber290	7802	10509002	Acute bronchitis (disorder)
0047123f-12e7-486c-82df-E	Jordon466	Harber290	7803	74400008	Appendicitis
0047123f-12e7-486c-82df-E	Jordon466	Harber290	7805	428251008	History of appendectomy



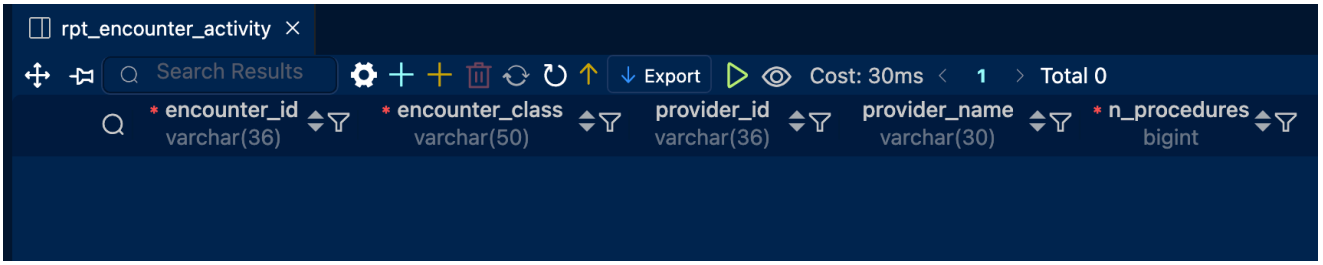
Output: I've created the table and INNER JOIN on patient_id from two tables: medical conditions and patient leaving out NULL values (if applicable), then I can show the number of patients diagnosed with each conditions shown above.

3) LEFT OUTER JOIN

3. Write an SQL statement by including two or more tables and using the LEFT OUTER JOIN. Show the results and sort the results by key field(s). Interpret the results compared to what an INNER JOIN does.

```
CREATE TABLE rpt_encounter_activity AS
SELECT
e.encounter_id,
e.patient_id,
e.encounter_start_date,
e.encounter_class,
pr.provider_id,
pr.organization_id,
pr.provider_name,
COUNT(pc.procedure_id) AS n_procedures
FROM encounter e
LEFT JOIN provider pr
ON pr.provider_id = e.provider_id
AND pr.organization_id = e.organization_id
LEFT JOIN procedures pc
ON pc.encounter_id = e.encounter_id
GROUP BY
e.encounter_id, e.patient_id, e.encounter_start_date,
e.encounter_class,
pr.provider_id, pr.organization_id, pr.provider_name;

```

```
SELECT
encounter_id,
encounter_class,
provider_id,
provider_name,
n_procedures
FROM rpt_encounter_activity
WHERE provider_id IS NULL
ORDER BY encounter_id;

```

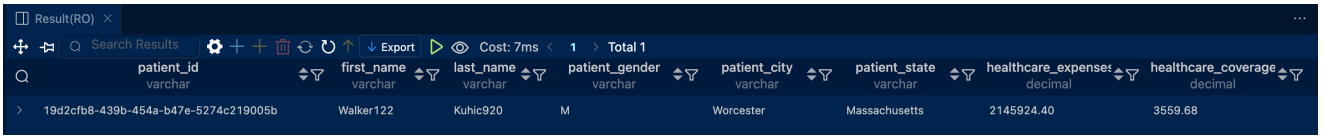


Output: When creating a new table, I LEFT JOIN on provider_id, organization_id (unique key), and encounter_id to find out if there are any encounters that has no recorded providers. Apparently, every encounter always have providers recorded which is good to know.

4) SINGLE-ROW SUBQUERY

4. Write a single-row subquery. Show the results and sort the results by key field(s). Interpret the output.

```
SELECT patient_id, first_name, last_name, patient_gender,
patient_city, patient_state, healthcare_expenses, healthcare_coverage
FROM patient
WHERE healthcare_expenses = (SELECT MAX(healthcare_expenses) FROM
patient)
ORDER BY patient_id;
```

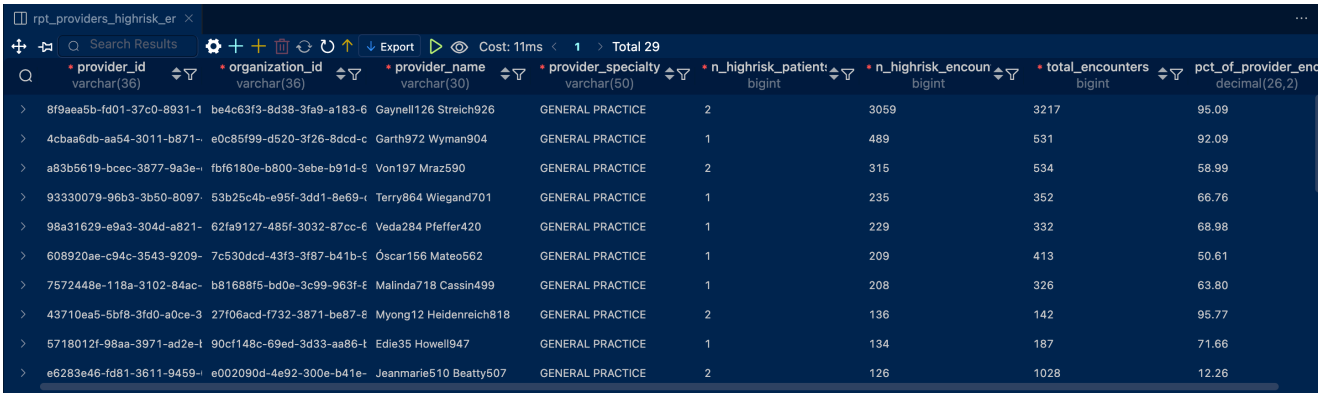


Output: I select few necessary columns associated with patient records and find a patient with the highest healthcare expense cost using SELECT MAX. The result is shown above.

5) MULTIPLE-ROW SUBQUERY

5. Write a multiple-row subquery. Show the results and sort the results by key field(s). Interpret the output.

```
CREATE TABLE rpt_providers_highrisk_er AS
WITH provider_tot AS (
SELECT provider_id, organization_id, COUNT(*) AS total_encounters
FROM encounter
GROUP BY provider_id, organization_id
),
highrisk_patients AS (
SELECT e.patient_id
FROM encounter e
WHERE LOWER(e.encounter_class) = 'emergency'
GROUP BY e.patient_id
HAVING COUNT(*) >= 10
)
SELECT
pr.provider_id,
pr.organization_id,
pr.provider_name,
pr.provider_specialty,
COUNT(DISTINCT e.patient_id) AS n_highrisk_patients,
COUNT(*) AS n_highrisk_encounters,
pt.total_encounters,
ROUND(100 * COUNT(*) / NULLIF(pt.total_encounters, 0), 2) AS
pct_of_provider_encounters
FROM provider pr
JOIN encounter e
ON e.provider_id = pr.provider_id
AND e.organization_id = pr.organization_id
JOIN provider_tot pt
ON pt.provider_id = pr.provider_id
AND pt.organization_id = pr.organization_id
WHERE e.patient_id IN (SELECT patient_id FROM highrisk_patients) --
multi-row subquery use
GROUP BY
pr.provider_id, pr.organization_id, pr.provider_name,
pr.provider_specialty, pt.total_encounters;

SELECT *
FROM rpt_providers_highrisk_er
ORDER BY n_highrisk_encounters DESC;
```

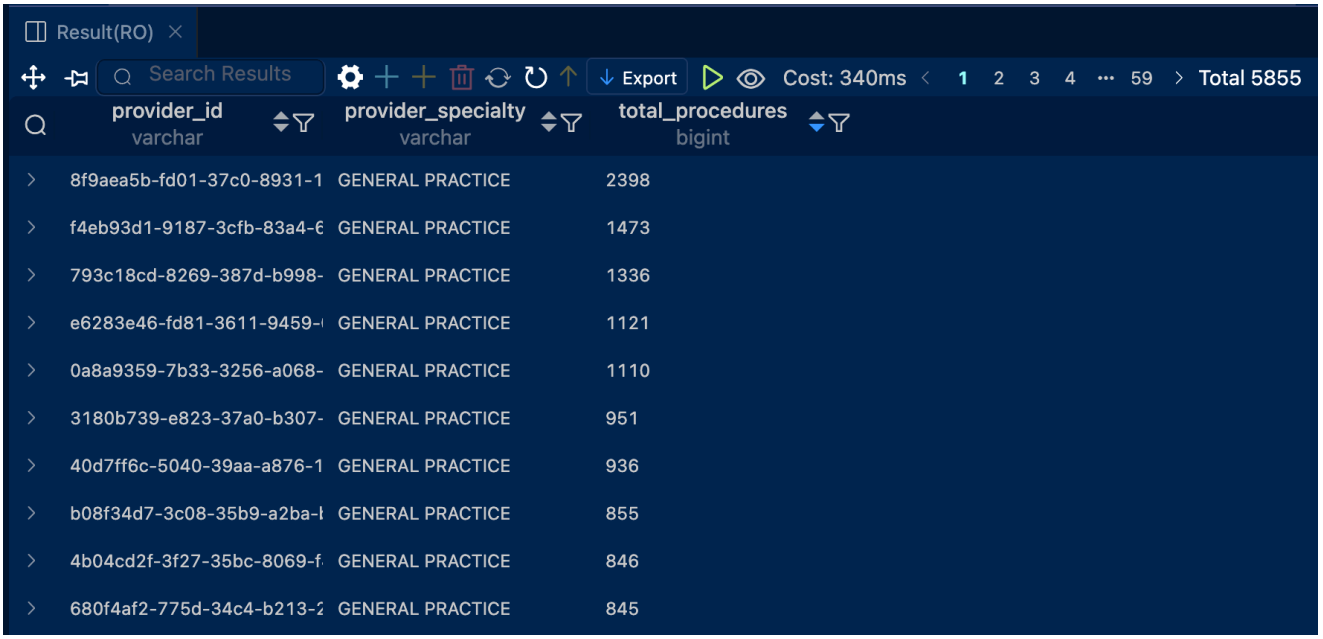


Output: I created a new table that counts the number of encounters (distinct patients) categorized as emergency indicating high risk for each provider and organization ID. Thus, this is a multi-row query with the result shown above ordered by the number of high risk encounters.

6) AGGREGATION

6. Write an SQL to aggregate the results by using multiple columns in the SELECT clause. Interpret the output.

```sql
SELECT pr.provider_id,
pr.provider_specialty,
COUNT(pc.procedure_id) AS total_procedures
FROM provider pr
LEFT JOIN encounter e
ON pr.provider_id = e.provider_id
AND pr.organization_id = e.organization_id
LEFT JOIN procedures pc
ON pc.encounter_id = e.encounter_id
GROUP BY pr.provider_id, pr.provider_specialty
ORDER BY total_procedures DESC;
```

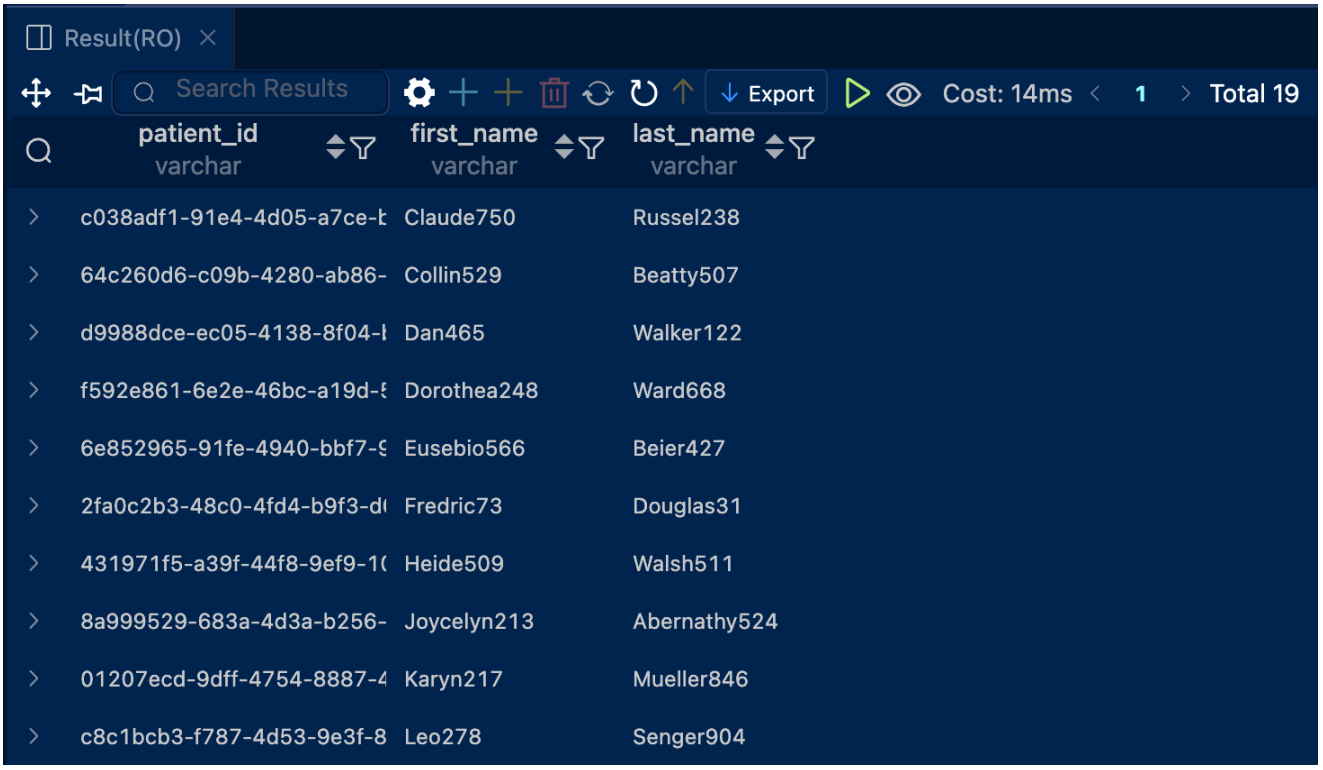


Output: Here, I LEFT JOIN encounter_id (provider -> encounter) allowing me to also LEFT JOIN procedures on encounter_ID (encounter to procedures). That way, I can group by provider_id and specialty to find out what specialties are the most common on total procedures performed. Turns out, general practice is very common. I guess good doctors gotta be prepared for any situation, eh?

7) NOT IN OPERATOR SUBQUERY

7. Write a subquery using the NOT IN operator. Show the results and sort the results by key field(s). Interpret the output.

```
SELECT patient_id, first_name, last_name
FROM patient
WHERE patient_id NOT IN (SELECT DISTINCT patient_id FROM diagnosis)
ORDER BY first_name, last_name;
```



patient_id varchar	first_name varchar	last_name varchar
c038adf1-91e4-4d05-a7ce-b	Claude750	Russel238
64c260d6-c09b-4280-ab86-	Collin529	Beatty507
d9988dce-ec05-4138-8f04-b	Dan465	Walker122
f592e861-6e2e-46bc-a19d-5	Dorothea248	Ward668
6e852965-91fe-4940-bbf7-9	Eusebio566	Beier427
2fa0c2b3-48c0-4fd4-b9f3-d(	Fredric73	Douglas31
431971f5-a39f-44f8-9ef9-1(	Heide509	Walsh511
8a999529-683a-4d3a-b256-	Joycelyn213	Abernathy524
01207ecd-9dff-4754-8887-4	Karyn217	Mueller846
c8c1bcb3-f787-4d53-9e3f-8	Leo278	Senger904

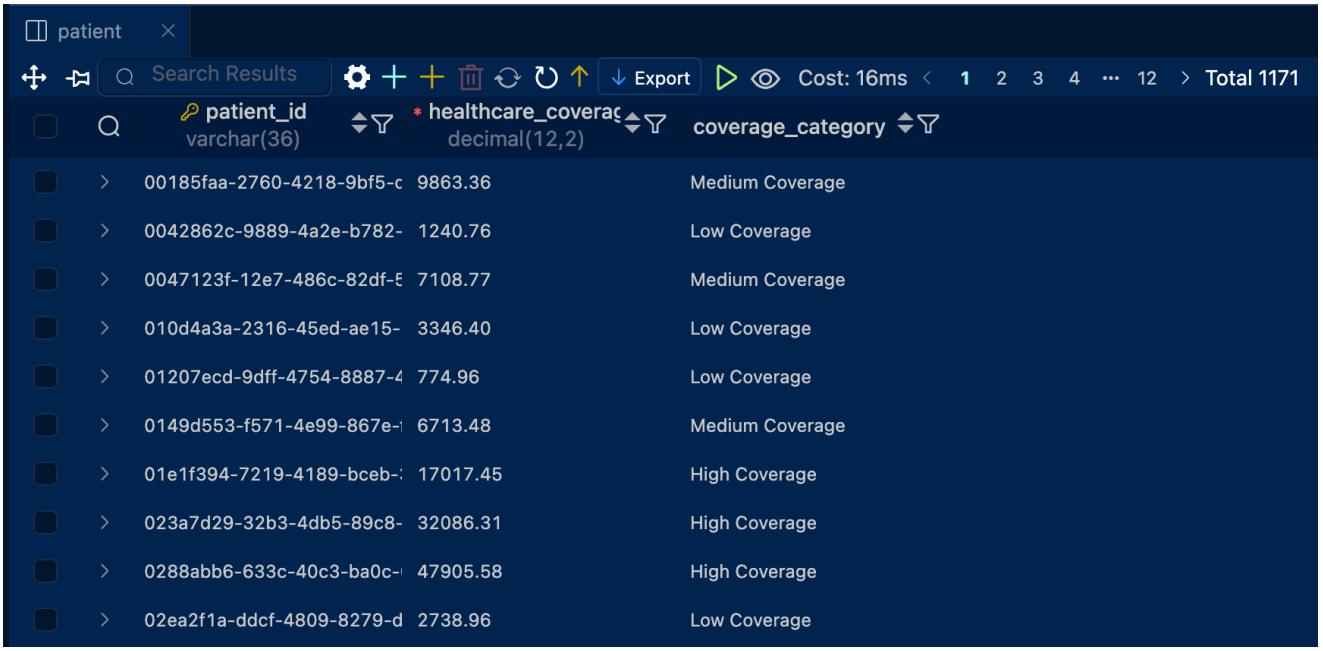
Output: This is pretty straightforward. I find patients that are not found in diagnosis list using patient table as a reference with NOT IN clause. There are 19 patients who do not have recorded diagnoses.

8) CASE STATEMENT QUERY

8. Write a query using a CASE statement. Show the results and sort the results by key field(s). Interpret the output.

```
SELECT patient_id,
healthcare_coverage,
CASE
```

```
WHEN healthcare_coverage >= 10000 THEN 'High Coverage'
WHEN healthcare_coverage >= 5000 THEN 'Medium Coverage'
ELSE 'Low Coverage'
END AS coverage_category
FROM patient
ORDER BY patient_id;

--printed table of summed counts of each coverage
SELECT
SUM(CASE WHEN healthcare_coverage >= 10000 THEN 1 ELSE 0 END) AS highcoverage_count,
SUM(CASE WHEN healthcare_coverage >= 5000 AND healthcare_coverage < 10000 THEN 1 ELSE 0 END) AS mediumcoverage_count,
SUM(CASE WHEN healthcare_coverage < 5000 THEN 1 ELSE 0 END) AS lowcoverage_count
FROM patient;
```



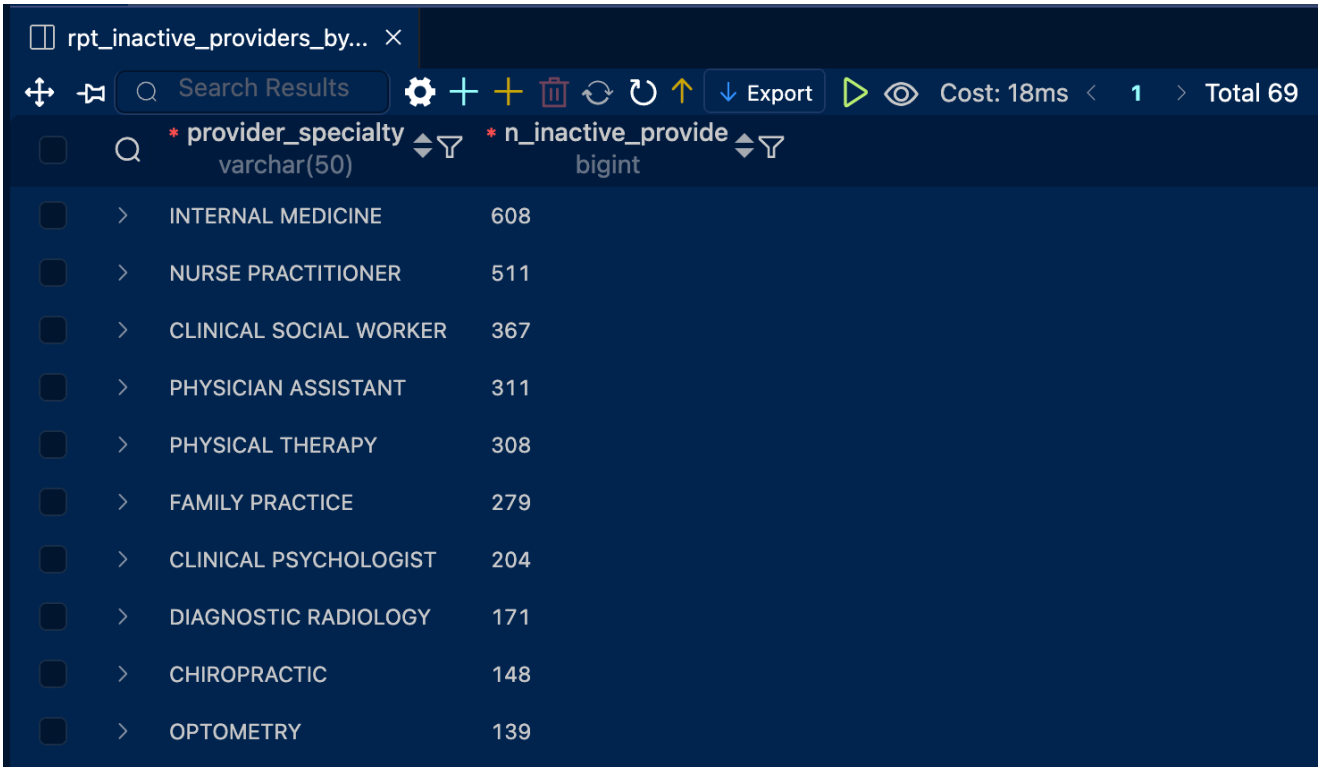
highcoverage_count	326
mediumcoverage_count	309
lowcoverage_count	536

Output: Using CASE WHEN clause, I categorize each healthcare coverage amount based on the criteria as low coverage (less than $5000), medium coverage (greater than or equal to $5000), or high coverage (greater than $10,000) when going through each patient. I also performed a useful SQL query that prints the aggregated count of each coverage shown in the small table above.

9) NOT EXISTS OPERATOR QUERY

9. Write a query using the NOT EXISTS operator. Show the results and sort the results by key field(s). Interpret the output.

```
CREATE TABLE rpt_inactive_providers_by_specialty AS
SELECT
pr.provider_specialty,
COUNT(*) AS n_inactive_providers
FROM provider pr
WHERE NOT EXISTS (
SELECT 1
FROM encounter e
WHERE e.provider_id = pr.provider_id
AND e.organization_id = pr.organization_id
)
GROUP BY pr.provider_specialty;

SELECT *
FROM rpt_inactive_providers_by_specialty
ORDER BY n_inactive_providers DESC, provider_specialty;
```



Output: I created a new table where it sums the count of inactive providers by specialty to find out most frequent specialties that have no recorded encounters (which are no matching row in encounter for the provider & organization ID).

10) NOT NULL OPERATOR SUBQUERY

10. Write a subquery using the NOT NULL operator in the inner query. Show the results and sort the results by key field(s). Interpret the output.

```sql
SELECT
e.encounter_class,
COUNT(*) AS n_encounters_deceased
FROM encounter e
WHERE e.patient_id IN (
SELECT patient_id
FROM patient
WHERE death_date IS NOT NULL
)
GROUP BY e.encounter_class
ORDER BY n_encounters_deceased DESC, e.encounter_class;
```

Output: Here, I sum the count of dead patients (by death_date that is NOT NULL) by encounter class. The result is shown above.

Answered Business Questions

1) Increase Profitability

Business question: Who/what drives cost, and where might we tighten up?

- **7.3 Top patients by total claim cost** → flags highest cost.
- **7.4 Procedure volume & average base cost** → identifies costly/high-volume procedures.
- **3) Encounters with zero procedures (LEFT JOIN)**.

I've been able to find out who has really high claim costs by joining patient and encounter on patient_id then summed them up. This will serve as a recommended system to use for SQL queries to flag patients with high costs and consider strategies to assist them with covering cost, thus less risk of financial issues for both parties.

Procedures performed over 1,000 times are the following:

- Medication Reconciliation (procedure)
- Renal dialysis (procedure)
- Auscultation of the fetal heart
- Evaluation of uterine fundal height
- Subcutaneous immunotherapy
- Intramuscular injection

What they have in common seems to be some kind of interventions through medications, therapies, or evaluations. But they're also non-surgical. Thus the quality of services should focus on the streamlined interventions of these methods. This would give providers sufficient time to cover uncommon procedures. Lastly, encounters with zero procedures showed that every encounter had a recorded provider, which means the services were always provided.

2) Improve Clinical Quality

Business question: Are patients getting timely follow-up and are inpatient stays efficient?

- **7.5 Post-procedure 14-day follow-up rate** → timely follow-up (quality of care).
- **7.2 Inpatient length of stay (LOS) per provider**.
- **10) Encounters for deceased patients (by class)**.

The table `rpt_inpatient_los_provider` shows the number of encounters that are "inpatient" with averaged length of stay (LOS) by each provider who has inpatient encounters. There are a few who have really high encounter numbers and also longer LOS. Either the provider specializes in lengthy procedures or perhaps there is a reason behind longer stay. Ideally, providers want less averaged LOS for any number of encounters. Higher encounters might be good in this context where they're a popular choice for patients when receiving treatments. Either way, this table serves as a recommended system for tracking providers who may need interventions and improve their skills, or allocate them when necessary to best alleviate workloads, or anything associated with providers' ability to provide quality service where it is demanded.

Regarding conditions, procedures, etc. and other causes leading to treatments, top 5 conditions seem to be:

- Viral sinusitis (disorder)
- Acute viral pharyngitis (disorder)
- Acute bronchitis (disorder)
- Body mass index 30+ — obesity (finding)
- Prediabetes

Conditions mainly consist of acute upper-respiratory infections and cardiometabolic risk conditions. The most common after that are chronic conditions, obstetric events (like normal

pregnancy, miscarriages, etc.), and other acute infections. Understanding the most frequent conditions can allow providers or organizations to improve care qualities focusing on these areas.

Finally, deaths by encounter class originally showed the top three as ambulatory, wellness, and outpatient. These are typically scheduled visits, so I re-checked using “death within 30 days of a visit” as a rate per 1,000 encounters.

```
SELECT
  LOWER(e.encounter_class) AS encounter_class,
  SUM(p.death_date IS NOT NULL
      AND p.death_date BETWEEN e.encounter_start_date AND
DATE_ADD(e.encounter_start_date, INTERVAL 30 DAY)) AS
deaths_within_30d,
  COUNT(*) AS total_encounters,
  ROUND(1000 * AVG(p.death_date IS NOT NULL
      AND p.death_date BETWEEN e.encounter_start_date AND
DATE_ADD(e.encounter_start_date, INTERVAL 30 DAY)), 2) AS
rate_per_1000
FROM encounter e
JOIN patient p ON p.patient_id = e.patient_id
GROUP BY LOWER(e.encounter_class)
ORDER BY rate_per_1000 DESC, encounter_class;
```

The result shows as follows:

encounter_class varchar	deaths_within_30d decimal	total_encounters bigint	rate_per_1000 decimal
inpatient	73	1838	39.72
emergency	57	2090	27.27
urgentcare	9	2373	3.79
ambulatory	36	18936	1.90
outpatient	11	9003	1.22
wellness	12	19106	0.63

The result shows very low rates for those classes and higher for emergency/inpatient. This is a good example of testing whether encounter class (X) really correlates with death rates (Y). Knowing common encounter types, reasons behind them, and correlations like death rates helps organizations prepare ahead of time and provide timely care. This serves as another recommendation system to track care by encounter class and related correlations to tackle clinical quality improvements.

3) Optimize Provider Utilization

Business question: Are workloads balanced, and where is capacity underused?

- **7.1 Provider utilization (encounters per provider & specialty)** → workload/volume.
- **9) Inactive providers by specialty (NOT EXISTS)**
- **6) Zero-procedure specialties** → specialties with no procedure activity.

Here, I identified the most popular providers (first page top list):

- Gaynell126 Streich926
- Gertrudis163 Schaden604
- Vern/31 Powlowski563
- Jeanmarie510 Beatty507
- Maile 198 Frami345
- Luke971 Rath779

On the same first page, the workload quickly became unbalanced where Gaynell had over 3,000 encounters while every other provider had less than 2,000 and rapidly decreasing when descending (all of them are under general practice specialty). This indicates very unbalanced allocation and would demand better allocation for workload between providers under similar specialty. This also serves as another recommended system to track workload allocations indicated by the number of encounters for each provider.

Most inactive specialties in the top 5 are:

- INTERNAL MEDICINE
- NURSE PRACTITIONER
- CLINICAL SOCIAL WORKER
- PHYSICIAN ASSISTANT
- PHYSICAL THERAPY

The table `rpt_inactive_providers_by_specialty` showed that providers with zero encounters fall under the most frequent specialties listed above. It looks like they mainly function in ambulatory settings where patients are not admitted overnight (same-day care, outpatient environments). This serves as one other system where it tracks encounters by specialty and may prompt minimizing or removing these specialties to better allocate budgeting into other much-needed care. Less is sometimes more such as faster workflow, less busy lineups, etc.

4) Reduce Readmissions

Business question: Who is at risk of coming back soon, and what can we do about it?

- **7.6 High-risk signal: frequent emergency users (≥3 ER visits)** → likely to revisit the ED.
- **7.5 Post-procedure 14-day follow-up rate** → improving follow-up.

These two tables serve as a recommended system where organizations/providers can track procedures and the quality of their service by the number of encounters under 14 days as well as patients with high emergency visits. If the encounter rate is high under 14 days, it would indicate poorer service quality and may need interventions or investigations for further improvements in that area. These patients with high ER visits will be flagged for more streamlined check-ins allowing for quick interventions and less waiting time when necessary.

5) Identify Care Strategic Expansion

Business question: Where should we grow services or redesign for demand and cost?

- **7.4 Procedure volume & average base cost** → where services are heavily used (and expensive).
- **2) Top conditions by distinct patients** (`rpt_diagnosis_patient_condition`)
- **9) Inactive providers by specialty** + **6) Zero-procedure specialties**.

It looks like therapies, evaluations, and medications are commonly used, but procedures in any form (e.g., therapies, evaluations) seem to be very expensive, usually over $10,000+. I suppose these service areas would need some extra investments for improved care and higher efficiency at a reduced cost due to better equipment and other factors. Same for other conditions like acute upper-respiratory infections, cardiometabolic risk conditions, chronic conditions, obstetric events (like normal pregnancy, miscarriages, etc.), and other acute infections. To reiterate, inactive providers and zero procedures by specialty allow the organization to either maximize budgeting in the most frequent specialties or increase care services in certain lacking specialties only if the circumstance requires it.

In summary, I laid out practical recommendations for improving care services and paired them with other SQL queries tailored for ongoing tracking.

Sources:
“A Review of the Cardiometabolic Risk Experience Among Canadian Métis Populations.” *Canadian Journal of Cardiology*, vol. 29, no. 8, Aug. 2013, pp. 1006–13. *www.sciencedirect.com*, https://doi.org/10.1016/j.cjca.2012.11.029.

Cost Reduction Strategies for Health Systems. https://www.compassonehealthcare.com/blog/cost-reduction-strategies-health-systems/. Accessed 14 Aug. 2025.

Drozdz, Dorota, et al. “Obesity and Cardiometabolic Risk Factors: From Childhood to Adulthood.” *Nutrients*, vol. 13, no. 11, Nov. 2021, p. 4176. *PubMed Central*, https://doi.org/10.3390/nu13114176.

“Health Care Costs: What’s the Problem?” *AAMC*, https://doi.org/10.15766/rai_dozyvvh2. Accessed 14 Aug. 2025.

“Take a Listen: What Auscultation Can Say About Your Health.” *Cleveland Clinic*, https://my.clevelandclinic.org/health/diagnostics/23080-auscultation. Accessed 14 Aug. 2025.

“What Is Dialysis?” *Cleveland Clinic*, https://my.clevelandclinic.org/health/treatments/14618-dialysis. Accessed 14 Aug. 2025.

“What’s an Upper Respiratory Infection?” *Cleveland Clinic*, https://my.clevelandclinic.org/health/diseases/4022-upper-respiratory-infection. Accessed 14 Aug. 2025.

The End