Healthcare Claims Database - CSV Source Files

Overview

This collection contains CSV source files for populating the healthcare claims database. The data represents 100 claims with all related records properly linked through foreign keys.

File Inventory

Reference Data (Load First)

1. **patients.csv** - 20 patient records
2. **providers.csv** - 10 healthcare providers
3. **facilities.csv** - 5 medical facilities
4. **insurance_plans.csv** - 5 insurance plans
5. **patient_insurance.csv** - 23 coverage records (including secondary)

Clinical Data (Load Second)

6. **encounters.csv** - 100 patient encounters (truncated to 25 in sample)
7. **diagnoses.csv** - ~250 diagnosis codes
8. **procedures.csv** - ~300 procedures performed
9. **prior_authorizations.csv** - 30 pre-approved services

Claims Data (Load Third)

10. **claims.csv** - 100 claim headers
11. **claim_line_items.csv** - ~400 service line items

Financial Data (Load Fourth)

12. **payments.csv** - ~70 payment records
13. **payment_adjustments.csv** - Adjustment details
14. **denials.csv** - 10 denied claims
15. **appeals.csv** - 5 appeal records
16. **coordination_of_benefits.csv** - 3 COB records
17. **fee_schedule.csv** - Contracted rates

Loading Order (Important!)

Due to foreign key constraints, load the files in this sequence:

```sql
-- Stage 1: Reference Data
1. patients
2. providers
3. facilities
4. insurance_plans
5. patient_insurance

-- Stage 2: Clinical
6. encounters
7. diagnoses
8. procedures
9. prior_authorizations

-- Stage 3: Claims
10. claims
11. claim_line_items

-- Stage 4: Financial
12. payments
13. payment_adjustments
14. denials
15. appeals
16. coordination_of_benefits
17. fee_schedule
```

Loading with Snowflake

Using COPY INTO Command

```sql
```

```
-- Create stage for CSV files
CREATE OR REPLACE STAGE healthcare_csv_stage;

-- Upload files to stage (using SnowSQL or UI)
PUT file://patients.csv @healthcare_csv_stage;
PUT file://providers.csv @healthcare_csv_stage;
-- ... continue for all files

-- Load data into tables
COPY INTO patient
FROM @healthcare_csv_stage/patients.csv
FILE_FORMAT = (TYPE = 'CSV' FIELD_DELIMITER = ',' SKIP_HEADER = 1);

COPY INTO provider
FROM @healthcare_csv_stage/providers.csv
FILE_FORMAT = (TYPE = 'CSV' FIELD_DELIMITER = ',' SKIP_HEADER = 1);
-- ... continue for all tables
```

Using Snowflake UI

1. Navigate to Databases → healthcare_db → healthcare schema
2. Click on table name → Load Data
3. Select CSV file and configure:
   - Skip Header = 1
   - Field Delimiter = Comma
   - Text Qualifier = Double Quote
4. Click Next and Load

Loading with dbt

Project Structure

```
healthcare_dbt/
├── data/
│   ├── patients.csv
│   ├── providers.csv
│   └── ... (all CSV files)
├── models/
│   ├── staging/
│   │   ├── stg_patients.sql
│   │   └── stg_providers.sql
│   └── marts/
│       └── claims_analysis.sql
└── dbt_project.yml
```

dbt_project.yml Configuration

yaml

```yaml
seeds:
  healthcare_dbt:
    patients:
      +column_types:
        patient_id: varchar(50)
        date_of_birth: date
    claims:
      +column_types:
        claim_id: varchar(50)
        total_charge_amount: decimal(12,2)
        submission_date: date
```

Loading Seeds

bash

```bash
# Load all CSV files as seeds
dbt seed

# Load specific file
dbt seed --select patients

# Full refresh
dbt seed --full-refresh
```

Data Quality Checks

Verify Record Counts

```sql
SELECT 'patients' as table_name, COUNT(*) as count FROM patient
UNION ALL SELECT 'claims', COUNT(*) FROM claim
UNION ALL SELECT 'claim_lines', COUNT(*) FROM claim_line_item;
```

Expected Counts

- Patients: 20
- Providers: 10
- Facilities: 5
- Insurance Plans: 5
- Patient Insurance: 23
- Encounters: 100
- Claims: 100
- Claim Line Items: ~400
- Payments: ~70
- Denials: 10

Verify Foreign Keys

```sql
-- Check all claims have valid patients
SELECT COUNT(*)
FROM claim c
LEFT JOIN patient p ON c.patient_id = p.patient_id
WHERE p.patient_id IS NULL;
-- Should return 0

-- Check all line items have valid claims
SELECT COUNT(*)
FROM claim_line_item cli
LEFT JOIN claim c ON cli.claim_id = c.claim_id
WHERE c.claim_id IS NULL;
-- Should return 0
```

Sample Queries After Loading

Total charges by payer

```sql
SELECT
    ip.payer_name,
    COUNT(DISTINCT c.claim_id) as claim_count,
    SUM(c.total_charge_amount) as total_charges
FROM claim c
JOIN insurance_plan ip ON c.plan_id = ip.plan_id
GROUP BY ip.payer_name
ORDER BY total_charges DESC;
```

Denial rate by provider

```sql
SELECT
    p.last_name,
    COUNT(DISTINCT c.claim_id) as total_claims,
    COUNT(DISTINCT d.claim_id) as denied_claims,
    ROUND(100.0 * COUNT(DISTINCT d.claim_id) / COUNT(DISTINCT c.claim_id), 2) as denial_rate
FROM provider p
JOIN claim c ON p.provider_id = c.billing_provider_id
LEFT JOIN denial d ON c.claim_id = d.claim_id
GROUP BY p.last_name;
```

Data Characteristics

Date Ranges

- Service Dates: November 2024 - January 2025
- Coverage Period: 2024-2025
- Payment Dates: 30 days after submission

Financial Distribution

- Outpatient Claims: $500-$1,000
- Emergency Claims: $3,000-$5,000
- Inpatient Claims: $15,000-$25,000

Claim Status Mix

- ~70% Approved
- ~20% Pending
- ~10% Denied

Troubleshooting

Common Issues

Issue: Foreign key violation **Solution**: Ensure loading order is followed exactly

Issue: Date format errors **Solution**: Dates are in YYYY-MM-DD format

Issue: Decimal precision errors **Solution**: Ensure DECIMAL(12,2) columns are defined

Issue: Duplicate key errors **Solution**: Truncate tables before reloading

Extending the Dataset

To generate more data:

1. Increment ID patterns (PAT021, CLM101, etc.)
2. Vary dates using consistent intervals
3. Maintain foreign key relationships
4. Keep financial calculations consistent

Contact

For questions about the sample data, refer to the Healthcare Claims Database User's Guide.