**IST 722: TEAM 1 - PROJECT**

**SNOWFLAKE CODE TO CREATE DATA WAREHOUSE, DATABASE, DIM TABLES AND FACT TABLES**

-- Create Database

CREATE OR REPLACE DATABASE TRANSIT\_DWH;

-- Set context

USE DATABASE TRANSIT\_DWH;

-- Create Schemas

CREATE OR REPLACE SCHEMA STAGING;

CREATE OR REPLACE SCHEMA ODS;

CREATE OR REPLACE SCHEMA DWH;

USE SCHEMA STAGING;

-- Create File Format

CREATE OR REPLACE FILE FORMAT TRANSIT\_DWH.STAGING.csv\_format

TYPE = 'CSV'

FIELD\_DELIMITER = ','

SKIP\_HEADER = 1

NULL\_IF = ('NULL', 'null')

EMPTY\_FIELD\_AS\_NULL = TRUE;

-- Create Stage

CREATE OR REPLACE STAGE TRANSIT\_DWH.STAGING.transit\_stage

FILE\_FORMAT = TRANSIT\_DWH.STAGING.csv\_format;

USE SCHEMA DWH;

-- Switch context

USE DATABASE TRANSIT\_DWH;

USE SCHEMA DWH;

-- DIMENSION TABLES

-- 1. Dim\_Time

CREATE OR REPLACE TABLE DWH.DIM\_TIME (

time\_key INTEGER IDENTITY(1,1) PRIMARY KEY,

datetime TIMESTAMP,

hour INTEGER,

day VARCHAR(10),

month INTEGER,

year INTEGER,

is\_weekend BOOLEAN,

is\_peak\_hour BOOLEAN,

day\_type VARCHAR(20),

created\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP(),

updated\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP()

);

-- 2. Dim\_Stops

CREATE OR REPLACE TABLE DWH.DIM\_STOPS (

stop\_id INTEGER PRIMARY KEY,

stop\_name VARCHAR(255),

division VARCHAR(50),

north\_direction\_label VARCHAR(50),

south\_direction\_label VARCHAR(50),

near\_commercial\_area BOOLEAN,

near\_university BOOLEAN,

created\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP(),

updated\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP()

);

-- 3. Dim\_Lines

CREATE OR REPLACE TABLE DWH.DIM\_LINES (

line\_id INTEGER PRIMARY KEY,

line VARCHAR(255),

division VARCHAR(50),

created\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP(),

updated\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP()

);

-- 4. Dim\_Remote\_Units

CREATE OR REPLACE TABLE DWH.DIM\_REMOTE\_UNITS (

remote\_unit\_id INTEGER PRIMARY KEY,

remote\_unit VARCHAR(50),

created\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP(),

updated\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP()

);

-- BRIDGE TABLES

-- 1. Connecting\_Lines (Stops-to-Lines bridge)

CREATE OR REPLACE TABLE DWH.CONNECTING\_LINES (

connection\_id INTEGER IDENTITY(1,1) PRIMARY KEY,

stop\_id INTEGER NOT NULL REFERENCES DIM\_STOPS(stop\_id),

line\_id INTEGER NOT NULL REFERENCES DIM\_LINES(line\_id),

created\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP(),

UNIQUE(stop\_id, line\_id)

);

-- 2. Daytime\_Routes (Stops-to-Lines bridge for daytime routes)

CREATE OR REPLACE TABLE DWH.DAYTIME\_ROUTES (

route\_id INTEGER IDENTITY(1,1) PRIMARY KEY,

stop\_id INTEGER NOT NULL REFERENCES DIM\_STOPS(stop\_id),

line\_id INTEGER NOT NULL REFERENCES DIM\_LINES(line\_id),

created\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP(),

UNIQUE(stop\_id, line\_id)

);

-- 3. Unit\_Locations (Stops-to-Remote Units bridge)

CREATE OR REPLACE TABLE DWH.UNIT\_LOCATIONS (

location\_id INTEGER IDENTITY(1,1) PRIMARY KEY,

stop\_id INTEGER NOT NULL REFERENCES DIM\_STOPS(stop\_id),

remote\_unit\_id INTEGER NOT NULL REFERENCES DIM\_REMOTE\_UNITS(remote\_unit\_id),

created\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP(),

UNIQUE(stop\_id, remote\_unit\_id)

);

-- Create corresponding STAGING tables

CREATE OR REPLACE TABLE STAGING.DIM\_TIME LIKE DWH.DIM\_TIME;

CREATE OR REPLACE TABLE STAGING.DIM\_STOPS LIKE DWH.DIM\_STOPS;

CREATE OR REPLACE TABLE STAGING.DIM\_LINES LIKE DWH.DIM\_LINES;

CREATE OR REPLACE TABLE STAGING.DIM\_REMOTE\_UNITS LIKE DWH.DIM\_REMOTE\_UNITS;

CREATE OR REPLACE TABLE STAGING.CONNECTING\_LINES LIKE DWH.CONNECTING\_LINES;

CREATE OR REPLACE TABLE STAGING.DAYTIME\_ROUTES LIKE DWH.DAYTIME\_ROUTES;

CREATE OR REPLACE TABLE STAGING.UNIT\_LOCATIONS LIKE DWH.UNIT\_LOCATIONS;

-- Create corresponding ODS tables

CREATE OR REPLACE TABLE ODS.DIM\_TIME LIKE DWH.DIM\_TIME;

CREATE OR REPLACE TABLE ODS.DIM\_STOPS LIKE DWH.DIM\_STOPS;

CREATE OR REPLACE TABLE ODS.DIM\_LINES LIKE DWH.DIM\_LINES;

CREATE OR REPLACE TABLE ODS.DIM\_REMOTE\_UNITS LIKE DWH.DIM\_REMOTE\_UNITS;

CREATE OR REPLACE TABLE ODS.CONNECTING\_LINES LIKE DWH.CONNECTING\_LINES;

CREATE OR REPLACE TABLE ODS.DAYTIME\_ROUTES LIKE DWH.DAYTIME\_ROUTES;

CREATE OR REPLACE TABLE ODS.UNIT\_LOCATIONS LIKE DWH.UNIT\_LOCATIONS;

-- Switch context

USE DATABASE TRANSIT\_DWH;

USE SCHEMA DWH;

-- 1. Fact\_Rider\_Volume

CREATE OR REPLACE TABLE DWH.FACT\_RIDER\_VOLUME (

fact\_id INTEGER IDENTITY(1,1) PRIMARY KEY,

time\_key INTEGER NOT NULL REFERENCES DIM\_TIME(time\_key),

stop\_id INTEGER NOT NULL REFERENCES DIM\_STOPS(stop\_id),

line\_id INTEGER NOT NULL REFERENCES DIM\_LINES(line\_id),

remote\_unit\_id INTEGER NOT NULL REFERENCES DIM\_REMOTE\_UNITS(remote\_unit\_id),

rider\_count INTEGER,

entries INTEGER,

exits INTEGER,

capacity INTEGER,

utilization\_rate FLOAT,

created\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP()

);

-- 2. Fact\_Stop\_Utilization

CREATE OR REPLACE TABLE DWH.FACT\_STOP\_UTILIZATION (

fact\_id INTEGER IDENTITY(1,1) PRIMARY KEY,

time\_key INTEGER NOT NULL REFERENCES DIM\_TIME(time\_key),

stop\_id INTEGER NOT NULL REFERENCES DIM\_STOPS(stop\_id),

utilization\_rate FLOAT,

scheduled\_capacity INTEGER,

actual\_usage INTEGER,

is\_peak BOOLEAN,

created\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP()

);

-- 3. Fact\_Line\_Performance

CREATE OR REPLACE TABLE DWH.FACT\_LINE\_PERFORMANCE (

fact\_id INTEGER IDENTITY(1,1) PRIMARY KEY,

time\_key INTEGER NOT NULL REFERENCES DIM\_TIME(time\_key),

line\_id INTEGER NOT NULL REFERENCES DIM\_LINES(line\_id),

schedule\_status VARCHAR(50),

trip\_timing FLOAT,

delay\_minutes INTEGER,

on\_time\_rate FLOAT,

estimated\_capacity INTEGER,

created\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP()

);

-- 4. Fact\_Remote\_Monitor

CREATE OR REPLACE TABLE DWH.FACT\_REMOTE\_MONITOR (

fact\_id INTEGER IDENTITY(1,1) PRIMARY KEY,

time\_key INTEGER NOT NULL REFERENCES DIM\_TIME(time\_key),

remote\_unit\_id INTEGER NOT NULL REFERENCES DIM\_REMOTE\_UNITS(remote\_unit\_id),

event\_type VARCHAR(50),

status VARCHAR(50),

battery\_health FLOAT,

location VARCHAR(50),

created\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP()

);

-- 5. Fact\_Rider\_Segments

CREATE OR REPLACE TABLE DWH.FACT\_RIDER\_SEGMENTS (

fact\_id INTEGER IDENTITY(1,1) PRIMARY KEY,

time\_key INTEGER NOT NULL REFERENCES DIM\_TIME(time\_key),

stop\_id INTEGER NOT NULL REFERENCES DIM\_STOPS(stop\_id),

demographic VARCHAR(50),

travel\_pattern VARCHAR(50),

segment\_count INTEGER,

location\_type VARCHAR(50),

created\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP()

);

-- Create corresponding STAGING tables

CREATE OR REPLACE TABLE STAGING.FACT\_RIDER\_VOLUME LIKE DWH.FACT\_RIDER\_VOLUME;

CREATE OR REPLACE TABLE STAGING.FACT\_STOP\_UTILIZATION LIKE DWH.FACT\_STOP\_UTILIZATION;

CREATE OR REPLACE TABLE STAGING.FACT\_LINE\_PERFORMANCE LIKE DWH.FACT\_LINE\_PERFORMANCE;

CREATE OR REPLACE TABLE STAGING.FACT\_REMOTE\_MONITOR LIKE DWH.FACT\_REMOTE\_MONITOR;

CREATE OR REPLACE TABLE STAGING.FACT\_RIDER\_SEGMENTS LIKE DWH.FACT\_RIDER\_SEGMENTS;

-- Create corresponding ODS tables

CREATE OR REPLACE TABLE ODS.FACT\_RIDER\_VOLUME LIKE DWH.FACT\_RIDER\_VOLUME;

CREATE OR REPLACE TABLE ODS.FACT\_STOP\_UTILIZATION LIKE DWH.FACT\_STOP\_UTILIZATION;

CREATE OR REPLACE TABLE ODS.FACT\_LINE\_PERFORMANCE LIKE DWH.FACT\_LINE\_PERFORMANCE;

CREATE OR REPLACE TABLE ODS.FACT\_REMOTE\_MONITOR LIKE DWH.FACT\_REMOTE\_MONITOR;

CREATE OR REPLACE TABLE ODS.FACT\_RIDER\_SEGMENTS LIKE DWH.FACT\_RIDER\_SEGMENTS;

-- Switch to staging schema

USE DATABASE TRANSIT\_DWH;

USE SCHEMA STAGING;

-- Create raw staging table to exactly match input CSV

CREATE OR REPLACE TABLE STAGING.RAW\_TRANSIT\_DATA (

Unnamed INTEGER,

Fact\_ID INTEGER,

Datetime TIMESTAMP,

Stop\_ID INTEGER,

Remote\_Unit\_ID INTEGER,

Line\_ID INTEGER,

Rider\_Count INTEGER,

Stop\_Name VARCHAR(255),

North\_Direction\_Label VARCHAR(50),

South\_Direction\_Label VARCHAR(50),

Division VARCHAR(50),

Line VARCHAR(255),

Connecting\_Lines VARCHAR(255),

Daytime\_Routes VARCHAR(255),

Remote\_Unit VARCHAR(50),

Hour INTEGER,

Day VARCHAR(10),

Month INTEGER,

Year INTEGER,

Capacity INTEGER,

Peak\_Off\_Peak VARCHAR(10),

Schedule\_On\_Time VARCHAR(50),

Trip\_Timing FLOAT,

Event\_Type VARCHAR(50),

Status VARCHAR(50),

Battery\_Health FLOAT,

Demographic VARCHAR(50),

Travel\_Pattern VARCHAR(50),

Entries INTEGER,

Exits INTEGER,

Near\_Commercial\_Area BOOLEAN,

Near\_University BOOLEAN

);

-- Copy data from staged files

COPY INTO STAGING.RAW\_TRANSIT\_DATA

FROM @transit\_stage/transit\_data.csv

FILE\_FORMAT = (FORMAT\_NAME = 'csv\_format')

ON\_ERROR = 'CONTINUE';

-- Transform and load dimensions

-- 1. Load DIM\_TIME

INSERT INTO ODS.DIM\_TIME (datetime, hour, day, month, year, is\_weekend, is\_peak\_hour)

SELECT DISTINCT

Datetime,

Hour,

Day,

Month,

Year,

CASE WHEN Day IN ('Saturday', 'Sunday') THEN TRUE ELSE FALSE END as is\_weekend,

CASE WHEN Peak\_Off\_Peak = 'Peak' THEN TRUE ELSE FALSE END as is\_peak\_hour

FROM STAGING.RAW\_TRANSIT\_DATA;

-- 2. Load DIM\_STOPS

INSERT INTO ODS.DIM\_STOPS (stop\_id, stop\_name, division, north\_direction\_label, south\_direction\_label, near\_commercial\_area, near\_university)

SELECT DISTINCT

Stop\_ID,

Stop\_Name,

Division,

North\_Direction\_Label,

South\_Direction\_Label,

Near\_Commercial\_Area,

Near\_University

FROM STAGING.RAW\_TRANSIT\_DATA;

-- 3. Load DIM\_LINES

INSERT INTO ODS.DIM\_LINES (line\_id, line, division)

SELECT DISTINCT

Line\_ID,

Line,

Division

FROM STAGING.RAW\_TRANSIT\_DATA;

-- 4. Load DIM\_REMOTE\_UNITS

INSERT INTO ODS.DIM\_REMOTE\_UNITS (remote\_unit\_id, remote\_unit)

SELECT DISTINCT

Remote\_Unit\_ID,

Remote\_Unit

FROM STAGING.RAW\_TRANSIT\_DATA;

-- Create stored procedure for bridge tables loading

CREATE OR REPLACE PROCEDURE LOAD\_BRIDGE\_TABLES()

RETURNS STRING

LANGUAGE SQL

AS

$$

BEGIN

-- Load CONNECTING\_LINES

INSERT INTO ODS.CONNECTING\_LINES (stop\_id, line\_id)

WITH SPLIT\_CONNECTING\_LINES AS (

SELECT DISTINCT

Stop\_ID,

TRIM(value) as connected\_line

FROM STAGING.RAW\_TRANSIT\_DATA,

LATERAL SPLIT\_TO\_TABLE(Connecting\_Lines, ',')

)

SELECT DISTINCT

s.Stop\_ID,

l.line\_id

FROM SPLIT\_CONNECTING\_LINES s

JOIN DIM\_LINES l ON TRIM(l.line) = TRIM(s.connected\_line);

-- Load DAYTIME\_ROUTES similar process...

RETURN 'Bridge tables loaded successfully';

END;

$$;

-- Switch context

USE DATABASE TRANSIT\_DWH;

-- Procedure to load dimensions

CREATE OR REPLACE PROCEDURE LOAD\_DIMENSIONS()

RETURNS STRING

LANGUAGE SQL

AS

$$

BEGIN

-- Load DIM\_TIME

INSERT INTO ODS.DIM\_TIME (datetime, hour, day, month, year, is\_weekend, is\_peak\_hour, day\_type)

SELECT DISTINCT

Datetime,

Hour,

Day,

Month,

Year,

CASE WHEN Day IN ('Saturday', 'Sunday') THEN TRUE ELSE FALSE END,

CASE WHEN Peak\_Off\_Peak = 'Peak' THEN TRUE ELSE FALSE END,

CASE

WHEN Day IN ('Saturday', 'Sunday') THEN 'Weekend'

ELSE 'Weekday'

END

FROM STAGING.RAW\_TRANSIT\_DATA

WHERE Datetime NOT IN (SELECT datetime FROM ODS.DIM\_TIME);

-- Load DIM\_STOPS

INSERT INTO ODS.DIM\_STOPS

(stop\_id, stop\_name, division, north\_direction\_label, south\_direction\_label, near\_commercial\_area, near\_university)

SELECT DISTINCT

Stop\_ID,

Stop\_Name,

Division,

North\_Direction\_Label,

South\_Direction\_Label,

Near\_Commercial\_Area,

Near\_University

FROM STAGING.RAW\_TRANSIT\_DATA

WHERE Stop\_ID NOT IN (SELECT stop\_id FROM ODS.DIM\_STOPS);

-- Load DIM\_LINES

INSERT INTO ODS.DIM\_LINES (line\_id, line, division)

SELECT DISTINCT

Line\_ID,

Line,

Division

FROM STAGING.RAW\_TRANSIT\_DATA

WHERE Line\_ID NOT IN (SELECT line\_id FROM ODS.DIM\_LINES);

-- Load DIM\_REMOTE\_UNITS

INSERT INTO ODS.DIM\_REMOTE\_UNITS (remote\_unit\_id, remote\_unit)

SELECT DISTINCT

Remote\_Unit\_ID,

Remote\_Unit

FROM STAGING.RAW\_TRANSIT\_DATA

WHERE Remote\_Unit\_ID NOT IN (SELECT remote\_unit\_id FROM ODS.DIM\_REMOTE\_UNITS);

RETURN 'Dimensions loaded successfully';

END;

$$;

-- Procedure to load bridge tables

CREATE OR REPLACE PROCEDURE LOAD\_BRIDGE\_TABLES()

RETURNS STRING

LANGUAGE SQL

AS

$$

BEGIN

-- Load CONNECTING\_LINES

INSERT INTO ODS.CONNECTING\_LINES (stop\_id, line\_id)

WITH SPLIT\_LINES AS (

SELECT DISTINCT

Stop\_ID,

TRIM(value) as connected\_line

FROM STAGING.RAW\_TRANSIT\_DATA,

LATERAL SPLIT\_TO\_TABLE(Connecting\_Lines, ',')

)

SELECT DISTINCT

s.Stop\_ID,

l.line\_id

FROM SPLIT\_LINES s

JOIN ODS.DIM\_LINES l ON TRIM(l.line) = TRIM(s.connected\_line)

WHERE NOT EXISTS (

SELECT 1 FROM ODS.CONNECTING\_LINES cl

WHERE cl.stop\_id = s.Stop\_ID AND cl.line\_id = l.line\_id

);

-- Load DAYTIME\_ROUTES

INSERT INTO ODS.DAYTIME\_ROUTES (stop\_id, line\_id)

WITH SPLIT\_ROUTES AS (

SELECT DISTINCT

Stop\_ID,

TRIM(value) as daytime\_route

FROM STAGING.RAW\_TRANSIT\_DATA,

LATERAL SPLIT\_TO\_TABLE(Daytime\_Routes, ',')

)

SELECT DISTINCT

s.Stop\_ID,

l.line\_id

FROM SPLIT\_ROUTES s

JOIN ODS.DIM\_LINES l ON TRIM(l.line) = TRIM(s.daytime\_route)

WHERE NOT EXISTS (

SELECT 1 FROM ODS.DAYTIME\_ROUTES dr

WHERE dr.stop\_id = s.Stop\_ID AND dr.line\_id = l.line\_id

);

RETURN 'Bridge tables loaded successfully';

END;

$$;

-- Move data from ODS to DWH

CREATE OR REPLACE PROCEDURE LOAD\_DWH()

RETURNS STRING

LANGUAGE SQL

AS

$$

BEGIN

-- Load dimensions from ODS to DWH

INSERT INTO DWH.DIM\_TIME SELECT \* FROM ODS.DIM\_TIME WHERE time\_key NOT IN (SELECT time\_key FROM DWH.DIM\_TIME);

INSERT INTO DWH.DIM\_STOPS SELECT \* FROM ODS.DIM\_STOPS WHERE stop\_id NOT IN (SELECT stop\_id FROM DWH.DIM\_STOPS);

INSERT INTO DWH.DIM\_LINES SELECT \* FROM ODS.DIM\_LINES WHERE line\_id NOT IN (SELECT line\_id FROM DWH.DIM\_LINES);

INSERT INTO DWH.DIM\_REMOTE\_UNITS SELECT \* FROM ODS.DIM\_REMOTE\_UNITS WHERE remote\_unit\_id NOT IN (SELECT remote\_unit\_id FROM DWH.DIM\_REMOTE\_UNITS);

-- Load bridge tables from ODS to DWH

INSERT INTO DWH.CONNECTING\_LINES SELECT \* FROM ODS.CONNECTING\_LINES;

INSERT INTO DWH.DAYTIME\_ROUTES SELECT \* FROM ODS.DAYTIME\_ROUTES;

RETURN 'DWH loaded successfully';

END;

$$;

-- Switch context

USE DATABASE TRANSIT\_DWH;

-- Procedure to load fact tables

CREATE OR REPLACE PROCEDURE LOAD\_FACT\_TABLES()

RETURNS STRING

LANGUAGE SQL

AS

$$

BEGIN

-- Load FACT\_RIDER\_VOLUME

INSERT INTO ODS.FACT\_RIDER\_VOLUME (

time\_key, stop\_id, line\_id, remote\_unit\_id,

rider\_count, entries, exits, capacity, utilization\_rate

)

SELECT

t.time\_key,

r.Stop\_ID,

r.Line\_ID,

r.Remote\_Unit\_ID,

r.Rider\_Count,

r.Entries,

r.Exits,

r.Capacity,

CASE

WHEN r.Capacity > 0 THEN (r.Rider\_Count::FLOAT / r.Capacity)

ELSE NULL

END as utilization\_rate

FROM STAGING.RAW\_TRANSIT\_DATA r

JOIN DWH.DIM\_TIME t ON r.Datetime = t.datetime;

-- Load FACT\_STOP\_UTILIZATION

INSERT INTO ODS.FACT\_STOP\_UTILIZATION (

time\_key, stop\_id, utilization\_rate,

scheduled\_capacity, actual\_usage, is\_peak

)

SELECT

t.time\_key,

r.Stop\_ID,

CASE

WHEN r.Capacity > 0 THEN (r.Rider\_Count::FLOAT / r.Capacity)

ELSE NULL

END as utilization\_rate,

r.Capacity as scheduled\_capacity,

r.Rider\_Count as actual\_usage,

CASE WHEN r.Peak\_Off\_Peak = 'Peak' THEN TRUE ELSE FALSE END as is\_peak

FROM STAGING.RAW\_TRANSIT\_DATA r

JOIN DWH.DIM\_TIME t ON r.Datetime = t.datetime;

-- Load FACT\_LINE\_PERFORMANCE

INSERT INTO ODS.FACT\_LINE\_PERFORMANCE (

time\_key, line\_id, schedule\_status,

trip\_timing, delay\_minutes, on\_time\_rate, estimated\_capacity

)

SELECT

t.time\_key,

r.Line\_ID,

r.Schedule\_On\_Time as schedule\_status,

r.Trip\_Timing,

CASE

WHEN r.Schedule\_On\_Time = 'Delayed' THEN r.Trip\_Timing::INTEGER

ELSE 0

END as delay\_minutes,

CASE

WHEN r.Schedule\_On\_Time = 'On-Time' THEN 1.0

ELSE 0.0

END as on\_time\_rate,

r.Capacity as estimated\_capacity

FROM STAGING.RAW\_TRANSIT\_DATA r

JOIN DWH.DIM\_TIME t ON r.Datetime = t.datetime;

-- Load FACT\_REMOTE\_MONITOR

INSERT INTO ODS.FACT\_REMOTE\_MONITOR (

time\_key, remote\_unit\_id, event\_type,

status, battery\_health, location

)

SELECT

t.time\_key,

r.Remote\_Unit\_ID,

r.Event\_Type,

r.Status,

r.Battery\_Health,

r.Stop\_Name as location

FROM STAGING.RAW\_TRANSIT\_DATA r

JOIN DWH.DIM\_TIME t ON r.Datetime = t.datetime;

-- Load FACT\_RIDER\_SEGMENTS

INSERT INTO ODS.FACT\_RIDER\_SEGMENTS (

time\_key, stop\_id, demographic,

travel\_pattern, segment\_count, location\_type

)

SELECT

t.time\_key,

r.Stop\_ID,

r.Demographic,

r.Travel\_Pattern,

r.Rider\_Count as segment\_count,

CASE

WHEN r.Near\_Commercial\_Area THEN 'Commercial'

WHEN r.Near\_University THEN 'University'

ELSE 'Regular'

END as location\_type

FROM STAGING.RAW\_TRANSIT\_DATA r

JOIN DWH.DIM\_TIME t ON r.Datetime = t.datetime;

RETURN 'Fact tables loaded successfully';

END;

$$;

-- Master procedure to orchestrate all loads

CREATE OR REPLACE PROCEDURE MASTER\_LOAD()

RETURNS STRING

LANGUAGE SQL

AS

$$

DECLARE

load\_status STRING;

BEGIN

-- Load dimensions

CALL LOAD\_DIMENSIONS();

-- Load bridge tables

CALL LOAD\_BRIDGE\_TABLES();

-- Load fact tables

CALL LOAD\_FACT\_TABLES();

-- Move to DWH

CALL LOAD\_DWH();

RETURN 'Full load completed successfully';

END;

$$;

-- Switch context

USE DATABASE TRANSIT\_DWH;

-- Create logging table

CREATE OR REPLACE TABLE DWH.ETL\_LOG (

log\_id INTEGER IDENTITY(1,1),

procedure\_name VARCHAR(100),

status VARCHAR(20),

records\_processed INTEGER,

error\_message VARCHAR(1000),

start\_time TIMESTAMP\_LTZ,

end\_time TIMESTAMP\_LTZ,

duration\_seconds FLOAT

);

-- Enhanced Master Load Procedure with Error Handling

CREATE OR REPLACE PROCEDURE MASTER\_LOAD()

RETURNS STRING

LANGUAGE JAVASCRIPT

AS

$$

try {

// Start logging

var start\_time = new Date();

// Execute dimension load

var dim\_result = snowflake.execute({

sqlText: "CALL LOAD\_DIMENSIONS()"

});

// Log dimension load

snowflake.execute({

sqlText: `INSERT INTO DWH.ETL\_LOG (

procedure\_name, status, start\_time, end\_time, duration\_seconds

) VALUES (?, ?, ?, ?, ?)`,

binds: ['LOAD\_DIMENSIONS', 'SUCCESS', start\_time, new Date(),

(new Date() - start\_time)/1000]

});

// Execute bridge tables load

var bridge\_result = snowflake.execute({

sqlText: "CALL LOAD\_BRIDGE\_TABLES()"

});

// Log bridge load

snowflake.execute({

sqlText: `INSERT INTO DWH.ETL\_LOG (

procedure\_name, status, start\_time, end\_time, duration\_seconds

) VALUES (?, ?, ?, ?, ?)`,

binds: ['LOAD\_BRIDGE\_TABLES', 'SUCCESS', start\_time, new Date(),

(new Date() - start\_time)/1000]

});

// Execute fact tables load

var fact\_result = snowflake.execute({

sqlText: "CALL LOAD\_FACT\_TABLES()"

});

// Log fact load

snowflake.execute({

sqlText: `INSERT INTO DWH.ETL\_LOG (

procedure\_name, status, start\_time, end\_time, duration\_seconds

) VALUES (?, ?, ?, ?, ?)`,

binds: ['LOAD\_FACT\_TABLES', 'SUCCESS', start\_time, new Date(),

(new Date() - start\_time)/1000]

});

// Execute DWH load

var dwh\_result = snowflake.execute({

sqlText: "CALL LOAD\_DWH()"

});

// Log final DWH load

snowflake.execute({

sqlText: `INSERT INTO DWH.ETL\_LOG (

procedure\_name, status, start\_time, end\_time, duration\_seconds

) VALUES (?, ?, ?, ?, ?)`,

binds: ['LOAD\_DWH', 'SUCCESS', start\_time, new Date(),

(new Date() - start\_time)/1000]

});

return "ETL process completed successfully";

} catch (err) {

// Log error

snowflake.execute({

sqlText: `INSERT INTO DWH.ETL\_LOG (

procedure\_name, status, error\_message, start\_time, end\_time, duration\_seconds

) VALUES (?, ?, ?, ?, ?, ?)`,

binds: ['MASTER\_LOAD', 'ERROR', err.message, start\_time, new Date(),

(new Date() - start\_time)/1000]

});

throw err.message;

}

$$;

-- Create validation procedure

-- Create validation procedure

CREATE OR REPLACE PROCEDURE VALIDATE\_LOAD()

RETURNS VARCHAR

LANGUAGE SQL

AS

$$

BEGIN

-- Create temporary table for validation results

CREATE OR REPLACE TEMPORARY TABLE validation\_results (

validation\_check VARCHAR,

status VARCHAR,

detail VARCHAR

);

-- Check for orphaned foreign keys

INSERT INTO validation\_results

SELECT

'Foreign Key Check',

CASE WHEN count(\*) = 0 THEN 'PASS' ELSE 'FAIL' END,

count(\*) || ' orphaned records found'

FROM DWH.FACT\_RIDER\_VOLUME f

LEFT JOIN DWH.DIM\_TIME t ON f.time\_key = t.time\_key

WHERE t.time\_key IS NULL;

-- Check for duplicate records

INSERT INTO validation\_results

SELECT

'Duplicate Check',

CASE WHEN count(\*) = 0 THEN 'PASS' ELSE 'FAIL' END,

count(\*) || ' duplicate records found'

FROM (

SELECT time\_key, stop\_id, line\_id, COUNT(\*)

FROM DWH.FACT\_RIDER\_VOLUME

GROUP BY time\_key, stop\_id, line\_id

HAVING COUNT(\*) > 1

);

-- Check for null values in critical columns

INSERT INTO validation\_results

SELECT

'Null Check',

CASE WHEN count(\*) = 0 THEN 'PASS' ELSE 'FAIL' END,

count(\*) || ' records with null values found'

FROM DWH.FACT\_RIDER\_VOLUME

WHERE rider\_count IS NULL OR capacity IS NULL;

RETURN 'Validation completed. Check validation\_results table for details.';

END;

$$;

-- Data as loaded through the Data Wizard --

--Verifying the loading of data

-- Check total number of rows

SELECT COUNT(\*) as total\_rows FROM RAW\_TRANSIT\_DATA;

-- Check for any NULL values in key columns

SELECT

COUNT(\*) as total\_rows,

COUNT(CASE WHEN Fact\_ID IS NULL THEN 1 END) as null\_fact\_ids,

COUNT(CASE WHEN Stop\_ID IS NULL THEN 1 END) as null\_stop\_ids,

COUNT(CASE WHEN Line\_ID IS NULL THEN 1 END) as null\_line\_ids,

COUNT(CASE WHEN Datetime IS NULL THEN 1 END) as null\_datetimes

FROM RAW\_TRANSIT\_DATA;

-- Look at sample data

SELECT \* FROM RAW\_TRANSIT\_DATA LIMIT 5;

-- Check distinct values for categorical columns

SELECT

COUNT(DISTINCT Stop\_ID) as unique\_stops,

COUNT(DISTINCT Line\_ID) as unique\_lines,

COUNT(DISTINCT Remote\_Unit\_ID) as unique\_remote\_units

FROM RAW\_TRANSIT\_DATA;

-- Verify date range

SELECT

MIN(Datetime) as earliest\_date,

MAX(Datetime) as latest\_date

FROM RAW\_TRANSIT\_DATA;

**DBT CODE TO LOAD DATA INTO TABLES AND CREATE VIEWS**

Stage:

WITH source AS (

SELECT \*

FROM {{ source('staging', 'raw\_transit\_data') }}

)

SELECT

FACT\_ID as fact\_id,

DATETIME as datetime,

STOP\_ID as stop\_id,

REMOTE\_UNIT\_ID as remote\_unit\_id,

LINE\_ID as line\_id,

RIDER\_COUNT as rider\_count,

STOP\_NAME as stop\_name,

NORTH\_DIRECTION\_LABEL as north\_direction\_label,

SOUTH\_DIRECTION\_LABEL as south\_direction\_label,

DIVISION as division,

LINE as line,

CONNECTING\_LINES as connecting\_lines,

DAYTIME\_ROUTES as daytime\_routes,

REMOTE\_UNIT as remote\_unit,

HOUR as hour,

DAY as day,

MONTH as month,

YEAR as year,

CAPACITY as capacity,

PEAK\_OFF\_PEAK as peak\_off\_peak,

SCHEDULE\_ON\_TIME as schedule\_on\_time,

TRIP\_TIMING as trip\_timing,

EVENT\_TYPE as event\_type,

STATUS as status,

BATTERY\_HEALTH as battery\_health,

DEMOGRAPHIC as demographic,

TRAVEL\_PATTERN as travel\_pattern,

ENTRIES as entries,

EXITS as exits,

NEAR\_COMMERCIAL\_AREA as near\_commercial\_area,

NEAR\_UNIVERSITY as near\_university

FROM source

**DIMENSION TABLES**

WITH line\_dimension AS (

SELECT DISTINCT -- ensure unique combinations

line\_id,

FIRST\_VALUE(line) OVER (PARTITION BY line\_id ORDER BY datetime) as line,

FIRST\_VALUE(division) OVER (PARTITION BY line\_id ORDER BY datetime) as division

FROM {{ ref('stg\_raw\_transit') }}

)

SELECT \* FROM line\_dimension

WHERE line\_id IS NOT NULL -- ensure no nulls

WITH remote\_unit\_dimension AS (

SELECT DISTINCT

remote\_unit\_id,

remote\_unit

FROM {{ ref('stg\_raw\_transit') }}

)

SELECT \*

FROM remote\_unit\_dimension

WITH stop\_dimension AS (

SELECT DISTINCT -- ensure unique combinations

stop\_id,

FIRST\_VALUE(stop\_name) OVER (PARTITION BY stop\_id ORDER BY datetime) as stop\_name,

FIRST\_VALUE(division) OVER (PARTITION BY stop\_id ORDER BY datetime) as division,

FIRST\_VALUE(north\_direction\_label) OVER (PARTITION BY stop\_id ORDER BY datetime) as north\_direction\_label,

FIRST\_VALUE(south\_direction\_label) OVER (PARTITION BY stop\_id ORDER BY datetime) as south\_direction\_label,

FIRST\_VALUE(near\_commercial\_area) OVER (PARTITION BY stop\_id ORDER BY datetime) as near\_commercial\_area,

FIRST\_VALUE(near\_university) OVER (PARTITION BY stop\_id ORDER BY datetime) as near\_university

FROM {{ ref('stg\_raw\_transit') }}

)

SELECT \* FROM stop\_dimension

WHERE stop\_id IS NOT NULL -- ensure no nulls

WITH time\_dimension AS (

SELECT DISTINCT -- Get unique time records

datetime,

hour,

day,

month,

year,

CASE

WHEN day IN ('Saturday', 'Sunday') THEN TRUE

ELSE FALSE

END as is\_weekend,

CASE

WHEN Peak\_Off\_Peak = 'Peak' THEN TRUE

ELSE FALSE

END as is\_peak\_hour,

CASE

WHEN day IN ('Saturday', 'Sunday') THEN 'Weekend'

ELSE 'Weekday'

END as day\_type

FROM {{ ref('stg\_raw\_transit') }}

),

numbered AS (

-- Add a row number to create unique keys

SELECT

ROW\_NUMBER() OVER (ORDER BY datetime) as time\_key,

\*

FROM time\_dimension

)

SELECT \* FROM numbered

**SCHEMA FOR DIM TABLES DATA LOAD**

version: 2

models:

- name: dim\_time

description: Time dimension table

columns:

- name: time\_key

description: Surrogate key for time dimension

tests:

- unique

- not\_null

- name: datetime

description: Full timestamp

tests:

- not\_null

- name: dim\_stops

description: Stops dimension table

columns:

- name: stop\_id

description: Primary key for stops

tests:

- unique

- not\_null

- name: stop\_name

description: Name of the stop

tests:

- not\_null

- name: dim\_lines

description: Lines dimension table

columns:

- name: line\_id

description: Primary key for lines

tests:

- unique

- not\_null

- name: line

description: Line name/code

tests:

- not\_null

- name: dim\_remote\_units

description: Remote units dimension table

columns:

- name: remote\_unit\_id

description: Primary key for remote units

tests:

- unique

- not\_null

- name: remote\_unit

description: Remote unit code

tests:

- not\_null

**FACT TABLES**

-- models/facts/fact\_line\_performance.sql

WITH fact\_line\_performance AS (

SELECT

t.time\_key,

l.line\_id,

r.schedule\_on\_time as schedule\_status,

r.trip\_timing,

CASE

WHEN r.schedule\_on\_time = 'Delayed'

THEN FLOOR(r.trip\_timing)

ELSE 0

END as delay\_minutes,

CASE

WHEN r.schedule\_on\_time = 'On-Time' THEN 1.0

ELSE 0.0

END as on\_time\_rate,

r.capacity as estimated\_capacity

FROM {{ ref('stg\_raw\_transit') }} r

JOIN {{ ref('dim\_time') }} t

ON r.datetime = t.datetime

JOIN {{ ref('dim\_lines') }} l

ON r.line\_id = l.line\_id

)

SELECT

ROW\_NUMBER() OVER (ORDER BY time\_key, line\_id) as fact\_id,

\*

FROM fact\_line\_performance

-- models/facts/fact\_remote\_monitor.sql

WITH fact\_remote\_monitor AS (

SELECT

t.time\_key,

ru.remote\_unit\_id,

r.event\_type,

r.status,

r.battery\_health,

s.stop\_name as location

FROM {{ ref('stg\_raw\_transit') }} r

JOIN {{ ref('dim\_time') }} t

ON r.datetime = t.datetime

JOIN {{ ref('dim\_remote\_units') }} ru

ON r.remote\_unit\_id = ru.remote\_unit\_id

JOIN {{ ref('dim\_stops') }} s

ON r.stop\_id = s.stop\_id

)

SELECT

ROW\_NUMBER() OVER (ORDER BY time\_key, remote\_unit\_id) as fact\_id,

\*

FROM fact\_remote\_monitor

-- models/facts/fact\_rider\_segments.sql

WITH fact\_rider\_segments AS (

SELECT

t.time\_key,

s.stop\_id,

r.demographic,

r.travel\_pattern,

r.rider\_count as segment\_count,

CASE

WHEN r.near\_commercial\_area THEN 'Commercial'

WHEN r.near\_university THEN 'University'

ELSE 'Regular'

END as location\_type

FROM {{ ref('stg\_raw\_transit') }} r

JOIN {{ ref('dim\_time') }} t

ON r.datetime = t.datetime

JOIN {{ ref('dim\_stops') }} s

ON r.stop\_id = s.stop\_id

)

SELECT

ROW\_NUMBER() OVER (ORDER BY time\_key, stop\_id, demographic, travel\_pattern) as fact\_id,

\*

FROM fact\_rider\_segments

-- models/facts/fact\_rider\_volume.sql

WITH fact\_rider\_volume AS (

SELECT

t.time\_key,

s.stop\_id,

l.line\_id,

r.remote\_unit\_id,

r.rider\_count,

r.entries,

r.exits,

r.capacity,

CASE

WHEN r.capacity > 0 THEN (r.rider\_count::FLOAT / r.capacity)

ELSE NULL

END as utilization\_rate

FROM {{ ref('stg\_raw\_transit') }} r

JOIN {{ ref('dim\_time') }} t

ON r.datetime = t.datetime

JOIN {{ ref('dim\_stops') }} s

ON r.stop\_id = s.stop\_id

JOIN {{ ref('dim\_lines') }} l

ON r.line\_id = l.line\_id

JOIN {{ ref('dim\_remote\_units') }} ru

ON r.remote\_unit\_id = ru.remote\_unit\_id

)

SELECT

ROW\_NUMBER() OVER (ORDER BY time\_key, stop\_id, line\_id) as fact\_id,

\*

FROM fact\_rider\_volume

-- models/facts/fact\_stop\_utilization.sql

WITH fact\_stop\_utilization AS (

SELECT

t.time\_key,

s.stop\_id,

CASE

WHEN r.capacity > 0 THEN (r.rider\_count::FLOAT / r.capacity)

ELSE NULL

END as utilization\_rate,

r.capacity as scheduled\_capacity,

r.rider\_count as actual\_usage,

CASE

WHEN r.peak\_off\_peak = 'Peak' THEN TRUE

ELSE FALSE

END as is\_peak

FROM {{ ref('stg\_raw\_transit') }} r

JOIN {{ ref('dim\_time') }} t

ON r.datetime = t.datetime

JOIN {{ ref('dim\_stops') }} s

ON r.stop\_id = s.stop\_id

)

SELECT

ROW\_NUMBER() OVER (ORDER BY time\_key, stop\_id) as fact\_id,

\*

FROM fact\_stop\_utilization

**SCHEMA FOR FACT TABLES**

version: 2

models:

- name: fact\_rider\_volume

description: Fact table recording rider volumes with details per stop and time

columns:

- name: fact\_id

description: Primary key for fact table

tests:

- unique

- not\_null

- name: time\_key

description: Foreign key to dim\_time

tests:

- not\_null

- relationships:

to: ref('dim\_time')

field: time\_key

- name: stop\_id

description: Foreign key to dim\_stops

tests:

- not\_null

- relationships:

to: ref('dim\_stops')

field: stop\_id

- name: line\_id

description: Foreign key to dim\_lines

tests:

- not\_null

- relationships:

to: ref('dim\_lines')

field: line\_id

- name: remote\_unit\_id

description: Foreign key to dim\_remote\_units

- name: fact\_stop\_utilization

description: Fact table for stop utilization metrics

columns:

- name: fact\_id

description: Primary key for fact table

tests:

- unique

- not\_null

- name: time\_key

description: Foreign key to dim\_time

tests:

- not\_null

- relationships:

to: ref('dim\_time')

field: time\_key

- name: stop\_id

description: Foreign key to dim\_stops

tests:

- not\_null

- relationships:

to: ref('dim\_stops')

field: stop\_id

- name: utilization\_rate

description: Calculated utilization rate (actual\_usage/scheduled\_capacity)

- name: scheduled\_capacity

description: Planned capacity of the stop

tests:

- not\_null

- name: actual\_usage

description: Actual number of riders using the stop

tests:

- not\_null

- name: is\_peak

description: Flag indicating peak hour status

tests:

- not\_null

- name: fact\_line\_performance

description: Fact table for line performance metrics

columns:

- name: fact\_id

description: Primary key for fact table

tests:

- unique

- not\_null

- name: time\_key

description: Foreign key to dim\_time

tests:

- not\_null

- relationships:

to: ref('dim\_time')

field: time\_key

- name: line\_id

description: Foreign key to dim\_lines

tests:

- not\_null

- relationships:

to: ref('dim\_lines')

field: line\_id

- name: schedule\_status

description: Status of schedule (On-Time/Delayed)

tests:

- not\_null

- name: trip\_timing

description: Actual trip time

tests:

- not\_null

- name: delay\_minutes

description: Calculated delay in minutes

- name: on\_time\_rate

description: Binary indicator for on-time status

tests:

- not\_null

- name: estimated\_capacity

description: Estimated line capacity

tests:

- not\_null

- name: fact\_remote\_monitor

description: Fact table for remote unit monitoring events

columns:

- name: fact\_id

description: Primary key for fact table

tests:

- unique

- not\_null

- name: time\_key

description: Foreign key to dim\_time

tests:

- not\_null

- relationships:

to: ref('dim\_time')

field: time\_key

- name: remote\_unit\_id

description: Foreign key to dim\_remote\_units

tests:

- not\_null

- relationships:

to: ref('dim\_remote\_units')

field: remote\_unit\_id

- name: event\_type

description: Type of monitoring event

tests:

- not\_null

- name: status

description: Operational status of unit

tests:

- not\_null

- name: battery\_health

description: Battery health percentage

tests:

- not\_null

- name: location

description: Current location of remote unit

tests:

- not\_null

- name: fact\_rider\_segments

description: Fact table for rider segment analysis

columns:

- name: fact\_id

description: Primary key for fact table

tests:

- unique

- not\_null

- name: time\_key

description: Foreign key to dim\_time

tests:

- not\_null

- relationships:

to: ref('dim\_time')

field: time\_key

- name: stop\_id

description: Foreign key to dim\_stops

tests:

- not\_null

- relationships:

to: ref('dim\_stops')

field: stop\_id

- name: demographic

description: Demographic category of riders

tests:

- not\_null

- name: travel\_pattern

description: Travel pattern category

tests:

- not\_null

- name: segment\_count

description: Count of riders in this segment

tests:

- not\_null

- name: location\_type

description: Type of location (Commercial/University/Regular)

tests:

- not\_null

DBT CODE TO CREATE VIEWS FOR DASHBOARD

WITH demographic\_metrics AS (

SELECT

t.datetime::DATE as report\_date,

s.stop\_name,

rs.demographic,

rs.travel\_pattern,

rs.location\_type,

SUM(rs.segment\_count) as total\_riders,

COUNT(DISTINCT t.hour) as active\_hours

FROM {{ ref('fact\_rider\_segments') }} rs

JOIN {{ ref('dim\_time') }} t ON rs.time\_key = t.time\_key

JOIN {{ ref('dim\_stops') }} s ON rs.stop\_id = s.stop\_id

GROUP BY

t.datetime::DATE,

s.stop\_name,

rs.demographic,

rs.travel\_pattern,

rs.location\_type

)

SELECT \* FROM demographic\_metrics

WITH hourly\_ridership AS (

SELECT

t.datetime::DATE as report\_date,

t.hour,

s.stop\_name,

l.line,

SUM(rv.rider\_count) as total\_riders,

AVG(rv.rider\_count) as avg\_riders,

AVG(rv.entries) as avg\_entries,

AVG(rv.exits) as avg\_exits,

SUM(rv.entries) as total\_entries,

SUM(rv.exits) as total\_exits,

AVG(rv.utilization\_rate) as avg\_utilization

FROM {{ ref('fact\_rider\_volume') }} rv

JOIN {{ ref('dim\_time') }} t ON rv.time\_key = t.time\_key

JOIN {{ ref('dim\_stops') }} s ON rv.stop\_id = s.stop\_id

JOIN {{ ref('dim\_lines') }} l ON rv.line\_id = l.line\_id

GROUP BY

t.datetime::DATE,

t.hour,

s.stop\_name,

l.line

)

SELECT \* FROM hourly\_ridership

WITH kpi\_metrics AS (

SELECT

t.datetime::DATE as report\_date,

SUM(rv.rider\_count) as total\_ridership,

AVG(rv.rider\_count) as avg\_ridership,

AVG(rv.utilization\_rate) as avg\_utilization,

AVG(CASE WHEN lp.schedule\_status = 'On-Time' THEN 1 ELSE 0 END) as on\_time\_rate,

AVG(rm.battery\_health) as avg\_battery\_health,

COUNT(DISTINCT CASE WHEN rm.status = 'Degraded' OR rm.status = 'Offline' THEN rm.remote\_unit\_id END) as alert\_count

FROM {{ ref('fact\_rider\_volume') }} rv

JOIN {{ ref('dim\_time') }} t ON rv.time\_key = t.time\_key

LEFT JOIN {{ ref('fact\_line\_performance') }} lp ON t.time\_key = lp.time\_key

LEFT JOIN {{ ref('fact\_remote\_monitor') }} rm ON t.time\_key = rm.time\_key

GROUP BY t.datetime::DATE

)

SELECT \* FROM kpi\_metrics

WITH performance\_metrics AS (

SELECT

t.datetime::DATE as report\_date,

t.hour,

rm.status,

rm.event\_type,

rm.battery\_health,

lp.schedule\_status,

lp.delay\_minutes,

s.stop\_name,

COUNT(\*) as event\_count

FROM {{ ref('fact\_remote\_monitor') }} rm

JOIN {{ ref('dim\_time') }} t ON rm.time\_key = t.time\_key

JOIN {{ ref('fact\_line\_performance') }} lp ON t.time\_key = lp.time\_key

JOIN {{ ref('dim\_stops') }} s ON s.stop\_id = s.stop\_id

GROUP BY

t.datetime::DATE,

t.hour,

rm.status,

rm.event\_type,

rm.battery\_health,

lp.schedule\_status,

lp.delay\_minutes,

s.stop\_name

)

SELECT \* FROM performance\_metrics

SCHEMA FOR VIEWS

version: 2

models:

- name: dash\_kpi\_metrics

description: Top-level KPIs for transit dashboard

- name: dash\_hourly\_ridership

description: Hourly ridership analysis with drill-down capabilities

- name: dash\_performance\_monitor

description: System performance and equipment monitoring metrics

- name: dash\_demographic\_analysis

description: Demographic and travel pattern analysis