USE DATABASE Bison\_db;

BISON\_DB.PUBLIC.PROFESSOR\_RATINGS

CREATE TABLE public.professor\_ratings (

id INT IDENTITY(1,1) PRIMARY KEY, -- Auto-incrementing ID

course\_code INT NOT NULL,

course\_name STRING NOT NULL,

instructors STRING NOT NULL, -- Can store multiple names as comma-separated values

rating FLOAT NOT NULL, -- Validate rating range

review\_text STRING, -- Optional: store written feedback

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP()

);

ALTER TABLE professor\_ratings

MODIFY COLUMN ID DROP NOT NULL;

ALTER TABLE professor\_ratings

MODIFY COLUMN RATING DROP NOT NULL;

ALTER TABLE professor\_ratings

MODIFY COLUMN REVIEW\_TEXT DROP NOT NULL;

ALTER TABLE professor\_ratings

MODIFY COLUMN CREATED\_AT DROP NOT NULL;

-- Look at data

SELECT \* FROM public.professor\_ratings;

SELECT \* FROM RMP\_COURSE\_EVALS

SELECT \* FROM INTERNAL\_COURSE\_EVALUATIONS

-- Create Merged dataset

SELECT \*

FROM PROFESSOR\_RATINGS AS db1

FULL OUTER JOIN RMP\_COURSE\_EVALS AS db2

ON CAST(db1.COURSE\_CODE AS TEXT) = CAST(db2.COURSE\_CODE AS TEXT)

AND db1.COURSE\_NAME = db2.COURSE\_NAME

AND db1.INSTRUCTORS = db2.INSTRUCTORS

FULL OUTER JOIN INTERNAL\_COURSE\_EVALUATIONS AS db3

ON CAST(db1.COURSE\_CODE AS TEXT) = CAST(db3.COURSE\_CODE AS TEXT)

AND db1.COURSE\_NAME = db3.COURSE\_NAME;

-- Create a new table with cleaned data

CREATE TABLE clean\_course\_evaluations AS

SELECT

-- Use COALESCE to get the first non-null value from duplicate columns

COALESCE(db1.COURSE\_CODE::TEXT, db2.COURSE\_CODE::TEXT, db3.COURSE\_CODE::TEXT) AS course\_code,

COALESCE(db1.COURSE\_NAME, db2.COURSE\_NAME, db3.COURSE\_NAME) AS course\_name,

-- Professor information

COALESCE(db1.INSTRUCTORS, db2.INSTRUCTORS) AS instructor\_name,

db3.PROFESSOR AS internal\_professor\_name,

-- Course details

db3.SEMESTER,

db3.SECTION,

-- Ratings from different sources

db1.RATING AS professor\_rating,

db2.OVERALL\_RATING AS rmp\_rating,

db3.OVERALL\_RATING AS internal\_rating,

-- Additional evaluation metrics

db2.WOULD\_TAKE\_AGAIN,

db2.DIFFICULTY,

db2.SENTIMENTS,

-- Review information

db1.REVIEW\_TEXT,

db1.CREATED\_AT AS review\_date,

-- Identifiers for joining

db1.ID AS professor\_rating\_id,

db2.RMP\_ID

FROM

PROFESSOR\_RATINGS AS db1

FULL OUTER JOIN

RMP\_COURSE\_EVALS AS db2

ON db1.COURSE\_CODE::TEXT = db2.COURSE\_CODE::TEXT

AND db1.COURSE\_NAME = db2.COURSE\_NAME

AND db1.INSTRUCTORS = db2.INSTRUCTORS

FULL OUTER JOIN

INTERNAL\_COURSE\_EVALUATIONS AS db3

ON COALESCE(db1.COURSE\_CODE::TEXT, db2.COURSE\_CODE::TEXT) = db3.COURSE\_CODE::TEXT

AND COALESCE(db1.COURSE\_NAME, db2.COURSE\_NAME) = db3.COURSE\_NAME;

SELECT \* FROM clean\_course\_evaluations

-- Different Cleaning

CREATE OR REPLACE TABLE clean\_course\_evaluations\_cleaned AS

SELECT

COURSE\_CODE,

COURSE\_NAME,

SECTION,

SEMESTER,

-- Merge professor names

COALESCE(INTERNAL\_PROFESSOR\_NAME, INSTRUCTOR\_NAME) AS PROFESSOR\_NAME,

-- Unified rating

COALESCE(INTERNAL\_RATING, PROFESSOR\_RATING, RMP\_RATING) AS COMBINED\_RATING,

WOULD\_TAKE\_AGAIN,

DIFFICULTY,

SENTIMENTS,

REVIEW\_TEXT,

-- Safer date conversion

CAST(REVIEW\_DATE AS DATE) AS REVIEW\_DATE

FROM clean\_course\_evaluations

WHERE COURSE\_CODE IS NOT NULL;

-- more cleaning

SELECT \* from clean\_course\_evaluations\_cleaned

CREATE OR REPLACE TABLE clean\_course\_evaluations\_cleaned AS

WITH Ranked AS (

SELECT \*,

ROW\_NUMBER() OVER (

PARTITION BY

COURSE\_CODE, SECTION, SEMESTER,

COALESCE(INTERNAL\_PROFESSOR\_NAME, INSTRUCTOR\_NAME),

REVIEW\_TEXT

ORDER BY REVIEW\_DATE DESC NULLS LAST

) AS rn

FROM clean\_course\_evaluations

WHERE COURSE\_CODE IS NOT NULL

)

SELECT

COURSE\_CODE,

COURSE\_NAME,

SECTION,

SEMESTER,

COALESCE(INTERNAL\_PROFESSOR\_NAME, INSTRUCTOR\_NAME) AS PROFESSOR\_NAME,

COALESCE(INTERNAL\_RATING, PROFESSOR\_RATING, RMP\_RATING) AS COMBINED\_RATING,

WOULD\_TAKE\_AGAIN,

DIFFICULTY,

SENTIMENTS,

REVIEW\_TEXT,

CAST(REVIEW\_DATE AS DATE) AS REVIEW\_DATE

FROM Ranked

WHERE rn = 1;

SELECT \* from clean\_course\_evaluations\_cleaned

-- Final cleaning

CREATE OR REPLACE TABLE CLEANED\_COURSE\_REVIEWS AS

WITH transformed AS (

SELECT

COURSE\_CODE,

CASE WHEN COURSE\_NAME = 'Unknown' THEN NULL ELSE COURSE\_NAME END AS COURSE\_NAME,

SECTION,

CASE

WHEN SEMESTER = 'SP 2024' THEN 'Spring 2024'

WHEN SEMESTER = 'FL 2023' THEN 'Fall 2023'

ELSE SEMESTER

END AS SEMESTER,

REGEXP\_REPLACE(PROFESSOR\_NAME, '^(Dr\\.|Professor) ', '') AS PROFESSOR\_NAME,

COMBINED\_RATING,

CASE WHEN WOULD\_TAKE\_AGAIN IS NOT NULL THEN WOULD\_TAKE\_AGAIN/100 ELSE NULL END AS WOULD\_TAKE\_AGAIN\_RATIO,

DIFFICULTY,

SPLIT(SENTIMENTS, ',') AS SENTIMENTS\_ARRAY,

REVIEW\_TEXT,

REVIEW\_DATE

FROM clean\_course\_evaluations\_cleaned

),

deduped AS (

SELECT DISTINCT \*

FROM transformed

)

SELECT

ROW\_NUMBER() OVER (ORDER BY COURSE\_CODE, PROFESSOR\_NAME) AS REVIEW\_ID,

d.\*

FROM deduped d;

-- Step 4: Add the primary key constraint on REVIEW\_ID

ALTER TABLE CLEANED\_COURSE\_REVIEWS ADD PRIMARY KEY (REVIEW\_ID);

-- 3. Create a standardized professor names table

CREATE OR REPLACE TABLE PROFESSOR\_NAME\_MAPPING AS

SELECT

PROFESSOR\_NAME AS ORIGINAL\_NAME,

REGEXP\_REPLACE(PROFESSOR\_NAME, '^(Dr\.|Professor) ', '') AS STANDARDIZED\_NAME

FROM (

SELECT DISTINCT PROFESSOR\_NAME

FROM CLEANED\_COURSE\_REVIEWS

);

-- 4. Apply the standardized professor names

MERGE INTO CLEANED\_COURSE\_REVIEWS target

USING PROFESSOR\_NAME\_MAPPING source

ON target.PROFESSOR\_NAME = source.ORIGINAL\_NAME

WHEN MATCHED THEN UPDATE SET target.PROFESSOR\_NAME = source.STANDARDIZED\_NAME;

-- 5. Fill in missing course names where possible (ensuring one row per COURSE\_CODE)

MERGE INTO CLEANED\_COURSE\_REVIEWS target

USING (

SELECT

COURSE\_CODE,

MAX(COURSE\_NAME) AS COURSE\_NAME

FROM

CLEANED\_COURSE\_REVIEWS

WHERE

COURSE\_NAME IS NOT NULL

GROUP BY

COURSE\_CODE

) source

ON target.COURSE\_CODE = source.COURSE\_CODE

AND target.COURSE\_NAME IS NULL

WHEN MATCHED THEN

UPDATE SET target.COURSE\_NAME = source.COURSE\_NAME;

-- 6. Create a view with course review metrics

CREATE OR REPLACE VIEW COURSE\_REVIEW\_METRICS AS

SELECT

COURSE\_CODE,

COURSE\_NAME,

PROFESSOR\_NAME,

COUNT(\*) AS REVIEW\_COUNT,

AVG(COMBINED\_RATING) AS AVG\_RATING,

AVG(WOULD\_TAKE\_AGAIN\_RATIO) AS AVG\_WOULD\_TAKE\_AGAIN,

AVG(DIFFICULTY) AS AVG\_DIFFICULTY,

ARRAY\_AGG(SENTIMENTS\_ARRAY) WITHIN GROUP (ORDER BY COMBINED\_RATING DESC) AS ALL\_SENTIMENTS

FROM

CLEANED\_COURSE\_REVIEWS

WHERE

COMBINED\_RATING IS NOT NULL

GROUP BY

COURSE\_CODE, COURSE\_NAME, PROFESSOR\_NAME;

-- 7. Create a data quality check view

CREATE OR REPLACE VIEW DATA\_QUALITY\_ISSUES AS

SELECT

'Missing Course Name' AS ISSUE\_TYPE,

COUNT(\*) AS ISSUE\_COUNT

FROM

CLEANED\_COURSE\_REVIEWS

WHERE

COURSE\_NAME IS NULL

UNION ALL

SELECT

'Missing Section' AS ISSUE\_TYPE,

COUNT(\*) AS ISSUE\_COUNT

FROM

CLEANED\_COURSE\_REVIEWS

WHERE

SECTION IS NULL

UNION ALL

SELECT

'Missing Semester' AS ISSUE\_TYPE,

COUNT(\*) AS ISSUE\_COUNT

FROM

CLEANED\_COURSE\_REVIEWS

WHERE

SEMESTER IS NULL

UNION ALL

SELECT

'Missing Ratings' AS ISSUE\_TYPE,

COUNT(\*) AS ISSUE\_COUNT

FROM

CLEANED\_COURSE\_REVIEWS

WHERE

COMBINED\_RATING IS NULL;

-- 8. Create a view for sentiment analysis

CREATE OR REPLACE VIEW SENTIMENT\_ANALYSIS AS

SELECT

COURSE\_CODE,

COURSE\_NAME,

PROFESSOR\_NAME,

f.value AS SENTIMENT,

COUNT(\*) AS SENTIMENT\_COUNT

FROM

CLEANED\_COURSE\_REVIEWS,

LATERAL FLATTEN(input => SENTIMENTS\_ARRAY) f

WHERE

SENTIMENTS\_ARRAY IS NOT NULL

GROUP BY

COURSE\_CODE, COURSE\_NAME, PROFESSOR\_NAME, SENTIMENT

ORDER BY

COURSE\_CODE, PROFESSOR\_NAME, SENTIMENT\_COUNT DESC;

-- 9. Create a view for semester trends

CREATE OR REPLACE VIEW SEMESTER\_TRENDS AS

SELECT

SEMESTER,

COUNT(\*) AS REVIEW\_COUNT,

AVG(COMBINED\_RATING) AS AVG\_RATING,

AVG(DIFFICULTY) AS AVG\_DIFFICULTY,

AVG(WOULD\_TAKE\_AGAIN\_RATIO) AS AVG\_WOULD\_TAKE\_AGAIN

FROM

CLEANED\_COURSE\_REVIEWS

WHERE

SEMESTER IS NOT NULL

AND COMBINED\_RATING IS NOT NULL

GROUP BY

SEMESTER

ORDER BY

SEMESTER;

-- 10. Create a view for professor performance

CREATE OR REPLACE VIEW PROFESSOR\_PERFORMANCE AS

SELECT

PROFESSOR\_NAME,

COUNT(DISTINCT COURSE\_CODE) AS COURSES\_TAUGHT,

COUNT(\*) AS TOTAL\_REVIEWS,

AVG(COMBINED\_RATING) AS AVG\_RATING,

AVG(DIFFICULTY) AS AVG\_DIFFICULTY,

AVG(WOULD\_TAKE\_AGAIN\_RATIO) AS AVG\_WOULD\_TAKE\_AGAIN,

ARRAY\_AGG(DISTINCT COURSE\_NAME) WITHIN GROUP (ORDER BY COURSE\_NAME) AS COURSES

FROM

CLEANED\_COURSE\_REVIEWS

WHERE

COMBINED\_RATING IS NOT NULL

GROUP BY

PROFESSOR\_NAME

ORDER BY

AVG\_RATING DESC;

-- 11. Create a view for course difficulty ranking

CREATE OR REPLACE VIEW COURSE\_DIFFICULTY\_RANKING AS

SELECT

COURSE\_CODE,

COURSE\_NAME,

AVG(DIFFICULTY) AS AVG\_DIFFICULTY,

COUNT(\*) AS REVIEW\_COUNT,

RANK() OVER (ORDER BY AVG(DIFFICULTY) DESC) AS DIFFICULTY\_RANK

FROM

CLEANED\_COURSE\_REVIEWS

WHERE

DIFFICULTY IS NOT NULL

GROUP BY

COURSE\_CODE, COURSE\_NAME

HAVING

COUNT(\*) >= 3 -- Only include courses with at least 3 reviews

ORDER BY

AVG\_DIFFICULTY DESC;

UPDATE CLEANED\_COURSE\_REVIEWS

SET PROFESSOR\_NAME = REGEXP\_SUBSTR(PROFESSOR\_NAME, '[^ ]+$');

--Take a look

SELECT \*

FROM CLEANED\_COURSE\_REVIEWS

LIMIT 100;

Select \* from WASHU\_COURSES\_WITH\_RATINGS

-- Standardize Jerome Data

-- Create a standardized table from WASHU\_COURSES\_WITH\_RATINGS with last name extraction

CREATE OR REPLACE TABLE WASHU\_COURSES\_STANDARDIZED AS

WITH normalized\_data AS (

SELECT

TRIM(PROFESSOR\_NAME) AS PROFESSOR\_NAME,

-- Extract last name from PROFESSOR\_NAME

SPLIT\_PART(TRIM(PROFESSOR\_NAME), ' ', -1) AS PROFESSOR\_LAST\_NAME,

TRIM(COURSE\_CODE) AS COURSE\_CODE,

COALESCE(TRIM(COURSE\_NAME), 'Unknown') AS COURSE\_NAME,

TRIM(SEMESTER) AS SEMESTER,

SECTION,

RMP\_ID,

OVERALL\_RATING,

RMP\_RATING,

-- Normalize RMP\_RATING to 1-7 scale for comparison

CASE

WHEN RMP\_RATING IS NOT NULL THEN (RMP\_RATING - 1) \* (6/4) + 1

ELSE NULL

END AS NORMALIZED\_RMP\_RATING,

-- Use either DIFFICULTY or difficulty column

COALESCE(DIFFICULTY, difficulty) AS STANDARDIZED\_DIFFICULTY,

WOULD\_TAKE\_AGAIN,

-- Convert WOULD\_TAKE\_AGAIN to ratio if it's not already

CASE

WHEN WOULD\_TAKE\_AGAIN > 1 THEN WOULD\_TAKE\_AGAIN/100

ELSE WOULD\_TAKE\_AGAIN

END AS STANDARDIZED\_WOULD\_TAKE\_AGAIN,

TAGS,

SOURCE,

-- Rank entries to remove duplicates based on last name instead of full name

ROW\_NUMBER() OVER (

PARTITION BY

SPLIT\_PART(TRIM(PROFESSOR\_NAME), ' ', -1), -- Use last name

TRIM(COURSE\_CODE),

TRIM(SEMESTER),

COALESCE(SECTION, 0)

ORDER BY

CASE WHEN OVERALL\_RATING IS NOT NULL THEN 0 ELSE 1 END,

CASE WHEN RMP\_RATING IS NOT NULL THEN 0 ELSE 1 END,

CASE WHEN COURSE\_NAME IS NOT NULL THEN 0 ELSE 1 END

) AS ROW\_RANK

FROM WASHU\_COURSES\_WITH\_RATINGS

)

SELECT

PROFESSOR\_NAME,

PROFESSOR\_LAST\_NAME, -- Include last name in the output

COURSE\_CODE,

COURSE\_NAME,

SEMESTER,

SECTION,

RMP\_ID,

OVERALL\_RATING,

RMP\_RATING,

-- Create a combined rating that uses OVERALL\_RATING if available, otherwise use normalized RMP\_RATING

COALESCE(OVERALL\_RATING, NORMALIZED\_RMP\_RATING) AS STANDARDIZED\_RATING,

STANDARDIZED\_WOULD\_TAKE\_AGAIN,

STANDARDIZED\_DIFFICULTY,

TAGS,

SOURCE

FROM normalized\_data

WHERE ROW\_RANK = 1;

-- Create a view for easy querying

CREATE OR REPLACE VIEW V\_WASHU\_COURSES AS

SELECT \* FROM WASHU\_COURSES\_STANDARDIZED;

-- Create a standardized table with professor last names

CREATE OR REPLACE TABLE WASHU\_COURSES\_STANDARDIZED AS

WITH normalized\_data AS (

SELECT

TRIM(PROFESSOR\_NAME) AS ORIGINAL\_PROFESSOR\_NAME,

-- Extract last name from PROFESSOR\_NAME

SPLIT\_PART(TRIM(PROFESSOR\_NAME), ' ', -1) AS PROFESSOR\_NAME,

TRIM(COURSE\_CODE) AS COURSE\_CODE,

COALESCE(TRIM(COURSE\_NAME), 'Unknown') AS COURSE\_NAME,

COALESCE(TRIM(SEMESTER), 'Unknown') AS SEMESTER,

COALESCE(SECTION, 0) AS SECTION,

RMP\_ID,

OVERALL\_RATING,

RMP\_RATING,

-- Normalize RMP\_RATING to 1-7 scale for comparison

CASE

WHEN RMP\_RATING IS NOT NULL THEN (RMP\_RATING - 1) \* (6/4) + 1

ELSE NULL

END AS NORMALIZED\_RMP\_RATING,

-- Use either DIFFICULTY or difficulty column

COALESCE(DIFFICULTY, difficulty) AS STANDARDIZED\_DIFFICULTY,

WOULD\_TAKE\_AGAIN,

-- Convert WOULD\_TAKE\_AGAIN to ratio if it's not already

CASE

WHEN WOULD\_TAKE\_AGAIN > 1 THEN WOULD\_TAKE\_AGAIN/100

ELSE WOULD\_TAKE\_AGAIN

END AS STANDARDIZED\_WOULD\_TAKE\_AGAIN,

TAGS,

SOURCE,

-- Rank entries to remove duplicates based on last name and course

ROW\_NUMBER() OVER (

PARTITION BY

SPLIT\_PART(TRIM(PROFESSOR\_NAME), ' ', -1),

TRIM(COURSE\_CODE),

COALESCE(TRIM(SEMESTER), 'Unknown'),

COALESCE(SECTION, 0)

ORDER BY

CASE WHEN OVERALL\_RATING IS NOT NULL THEN 0 ELSE 1 END,

CASE WHEN RMP\_RATING IS NOT NULL THEN 0 ELSE 1 END,

CASE WHEN COURSE\_NAME IS NOT NULL THEN 0 ELSE 1 END

) AS ROW\_RANK

FROM WASHU\_COURSES\_WITH\_RATINGS

)

SELECT

PROFESSOR\_NAME,

ORIGINAL\_PROFESSOR\_NAME AS FULL\_PROFESSOR\_NAME,

COURSE\_CODE,

COURSE\_NAME,

SEMESTER,

SECTION,

RMP\_ID,

OVERALL\_RATING,

RMP\_RATING,

-- Create a combined rating that uses OVERALL\_RATING if available, otherwise use normalized RMP\_RATING

COALESCE(OVERALL\_RATING, NORMALIZED\_RMP\_RATING) AS STANDARDIZED\_RATING,

STANDARDIZED\_WOULD\_TAKE\_AGAIN,

STANDARDIZED\_DIFFICULTY,

TAGS,

SOURCE

FROM normalized\_data

WHERE ROW\_RANK = 1;

-- Create a view for easy querying

CREATE OR REPLACE VIEW V\_WASHU\_COURSES AS

SELECT \* FROM WASHU\_COURSES\_STANDARDIZED;

SELECT \* FROM V\_WASHU\_COURSES;

DESCRIBE Table V\_WASHU\_COURSES;

DESCRIBE TABLE washu\_courses\_standardized