SQL Automated Data Cleaning

(Complete Code: Github Link)

This documentation outlines a project with the goal of automating the data cleaning. The project aims to enhance data quality by leveraging stored procedure and event to remove duplicates, correct data inconsistencies and standardize data fields.

The project has two key objectives:

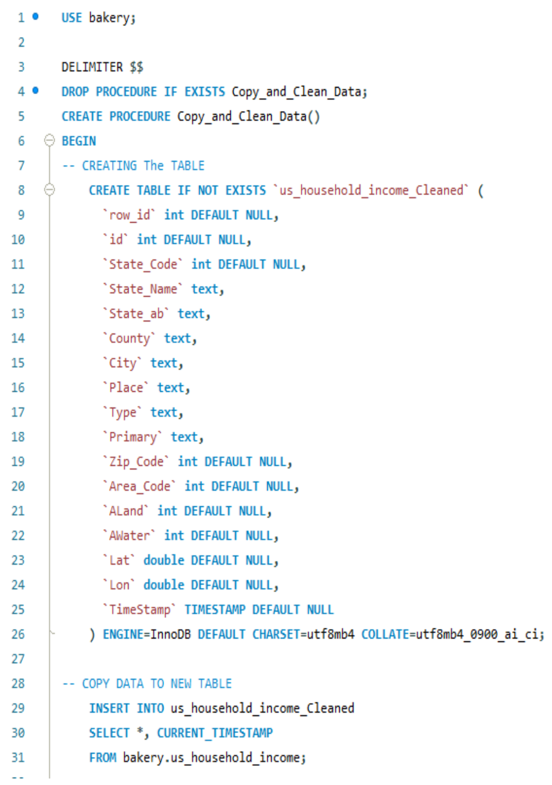
- To create a cleaned version of the data table.
- To automate the data cleaning process using stored procedure, event and trigger.

The project involves creating a new cleaned data table, developing a stored procedure for data cleaning, setting up automated events to schedule regular data cleaning and implementing triggers to clean data immediately after insertion.

In this project, the following key stages are undertaken:

- Creating the Cleaned Data Table: Defining and creating a new table for cleaned data.

(The best practice is to have a copy of the raw data and then carry out the data cleaning process)



```sql
USE bakery;

DELIMITER $$
DROP PROCEDURE IF EXISTS Copy_and_Clean_Data;
CREATE PROCEDURE Copy_and_Clean_Data()
BEGIN
-- CREATING The TABLE
    CREATE TABLE IF NOT EXISTS `us_household_income_Cleaned` (
      `row_id` int DEFAULT NULL,
      `id` int DEFAULT NULL,
      `State_Code` int DEFAULT NULL,
      `State_Name` text,
      `State_ab` text,
      `County` text,
      `City` text,
      `Place` text,
      `Type` text,
      `Primary` text,
      `Zip_Code` int DEFAULT NULL,
      `Area_Code` int DEFAULT NULL,
      `ALand` int DEFAULT NULL,
      `AWater` int DEFAULT NULL,
      `Lat` double DEFAULT NULL,
      `Lon` double DEFAULT NULL,
      `TimeStamp` TIMESTAMP DEFAULT NULL
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-- COPY DATA TO NEW TABLE
    INSERT INTO us_household_income_Cleaned
    SELECT *, CURRENT_TIMESTAMP
    FROM bakery.us_household_income;
```

Adding a ‘TimeStamp’ column is important as it aids in debugging. With an automated process, data may change over time and ‘TimeStamp’ would make a significant difference when identifying and correcting errors.

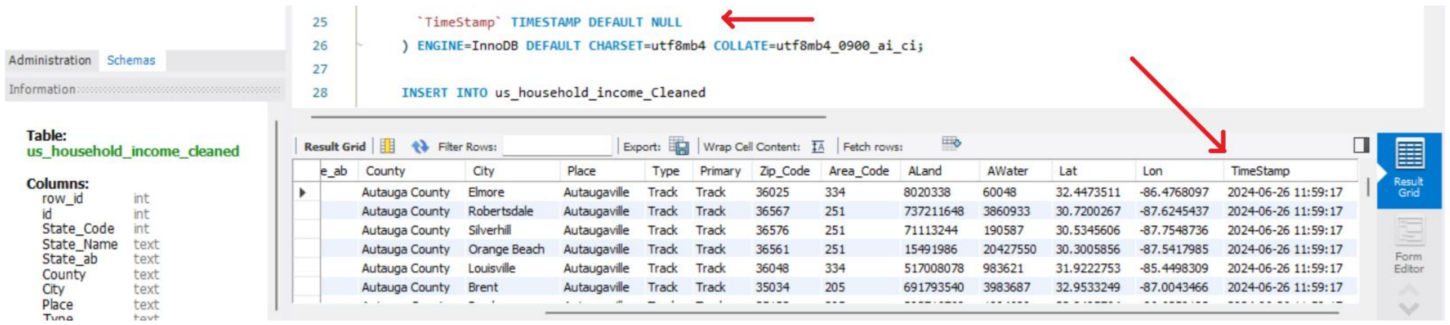


Table: us_household_income_cleaned

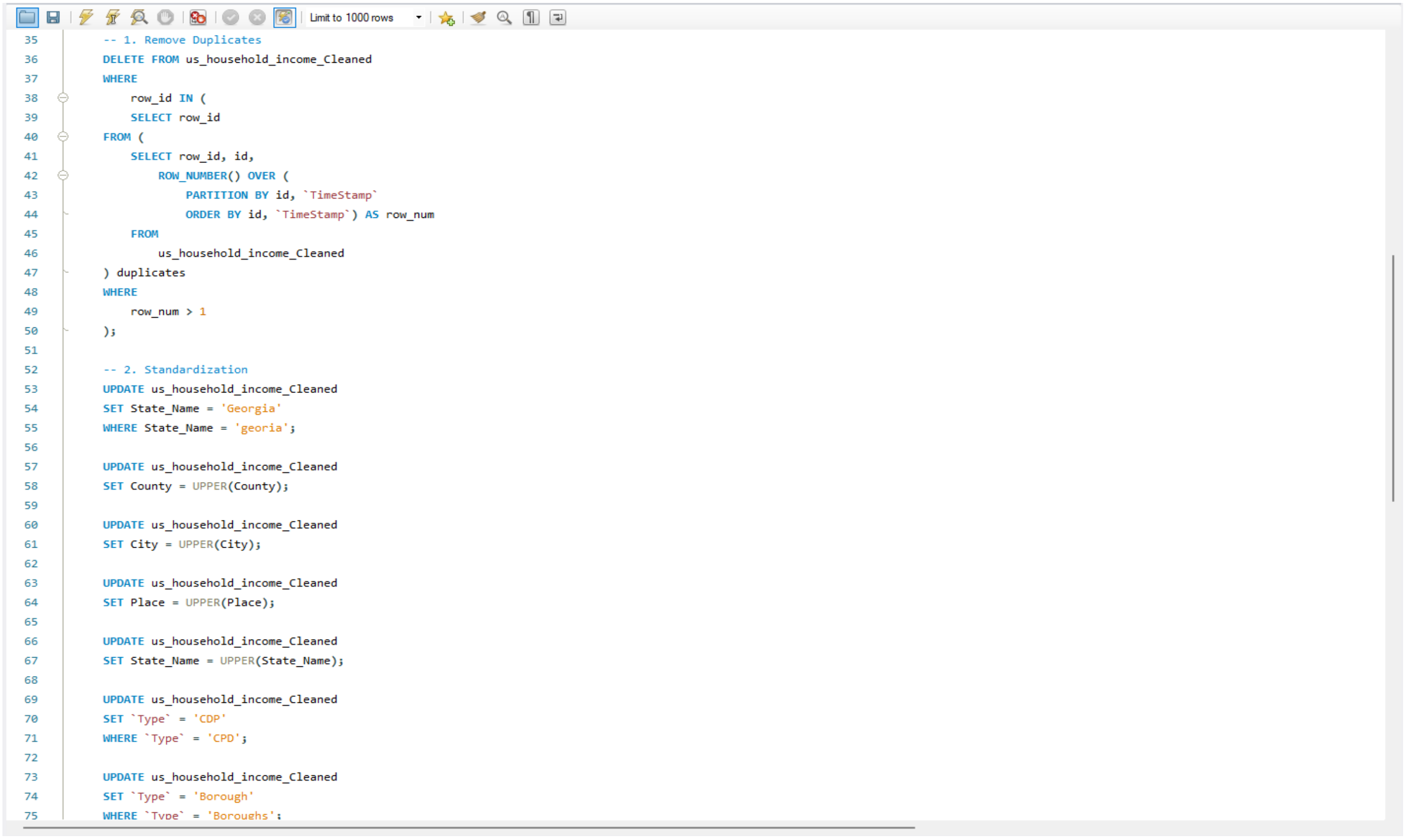
Columns:

row_id	int
id	int
State_Code	int
State_Name	text
State_ab	text
County	text
City	text
Place	text

```sql
    `TimeStamp` TIMESTAMP DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

INSERT INTO us_household_income_Cleaned
```

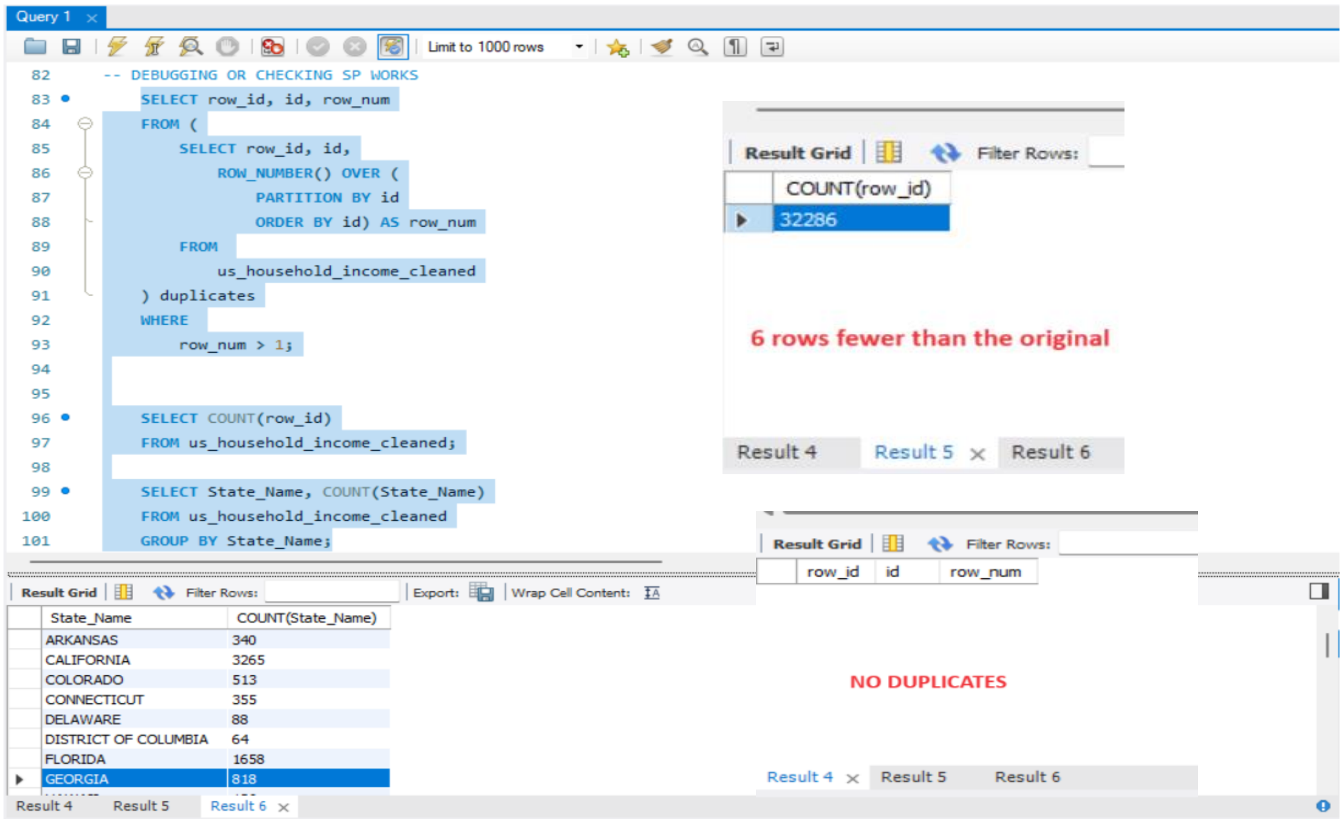
e_ab	County	City	Place	Type	Primary	Zip_Code	Area_Code	ALand	AWater	Lat	Lon	TimeStamp
	Autauga County	Elmore	Autaugaville	Track	Track	36025	334	8020338	60048	32.4473511	-86.4768097	2024-06-26 11:59:17
	Autauga County	Robertsdale	Autaugaville	Track	Track	36567	251	737211648	3860933	30.7200267	-87.6245437	2024-06-26 11:59:17
	Autauga County	Silverhill	Autaugaville	Track	Track	36576	251	71113244	190587	30.5345606	-87.7548736	2024-06-26 11:59:17
	Autauga County	Orange Beach	Autaugaville	Track	Track	36561	251	15491986	20427550	30.3005856	-87.5417985	2024-06-26 11:59:17
	Autauga County	Louisville	Autaugaville	Track	Track	36048	334	517008078	983621	31.9222753	-85.4498309	2024-06-26 11:59:17
	Autauga County	Brent	Autaugaville	Track	Track	35034	205	691793540	3983687	32.9533249	-87.0043466	2024-06-26 11:59:17

- Procedure for Data Cleaning: Developing a stored procedure to copy data, remove duplicates and standardize fields.



```sql
    -- 1. Remove Duplicates
    DELETE FROM us_household_income_Cleaned
    WHERE
        row_id IN (
        SELECT row_id
    FROM (
        SELECT row_id, id,
            ROW_NUMBER() OVER (
                PARTITION BY id, `TimeStamp`
                ORDER BY id, `TimeStamp`) AS row_num
        FROM
            us_household_income_Cleaned
    ) duplicates
    WHERE
        row_num > 1
    );

    -- 2. Standardization
    UPDATE us_household_income_Cleaned
    SET State_Name = 'Georgia'
    WHERE State_Name = 'georia';

    UPDATE us_household_income_Cleaned
    SET County = UPPER(County);

    UPDATE us_household_income_Cleaned
    SET City = UPPER(City);

    UPDATE us_household_income_Cleaned
    SET Place = UPPER(Place);

    UPDATE us_household_income_Cleaned
    SET State_Name = UPPER(State_Name);

    UPDATE us_household_income_Cleaned
    SET `Type` = 'CDP'
    WHERE `Type` = 'CPD';

    UPDATE us_household_income_Cleaned
    SET `Type` = 'Borough'
    WHERE `Type` = 'Boroughs';
```

- Testing the Procedure: Manually executing the procedure to validate its functionality.



```sql
-- DEBUGGING OR CHECKING SP WORKS
	SELECT row_id, id, row_num
	FROM (
		SELECT row_id, id,
			ROW_NUMBER() OVER (
				PARTITION BY id
				ORDER BY id) AS row_num
		FROM
			us_household_income_cleaned
	) duplicates
	WHERE
		row_num > 1;


	SELECT COUNT(row_id)
	FROM us_household_income_cleaned;

	SELECT State_Name, COUNT(State_Name)
	FROM us_household_income_cleaned
	GROUP BY State_Name;
```

COUNT(row_id)
32286

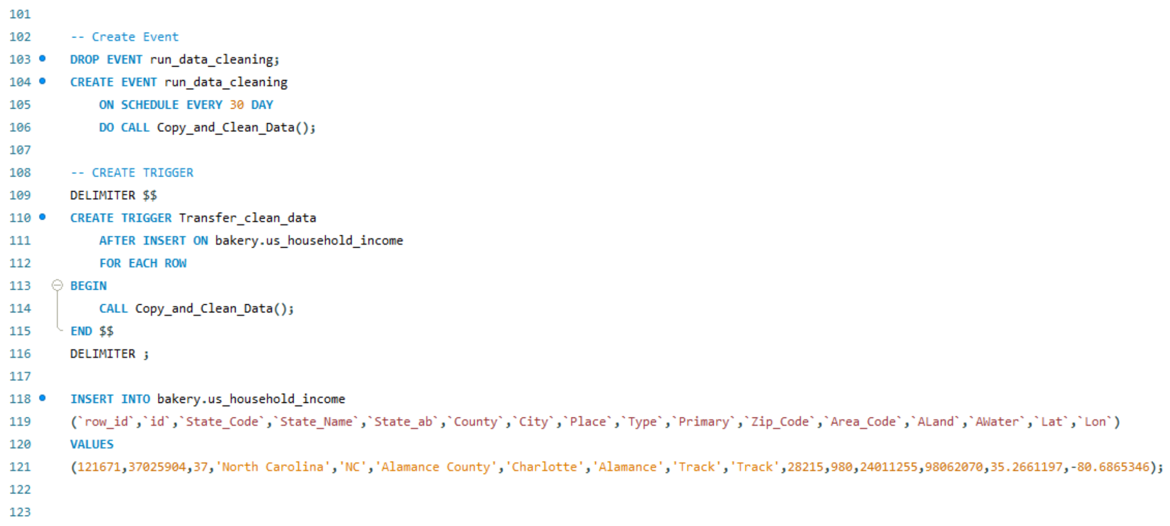
6 rows fewer than the original

row_id	id	row_num

NO DUPLICATES

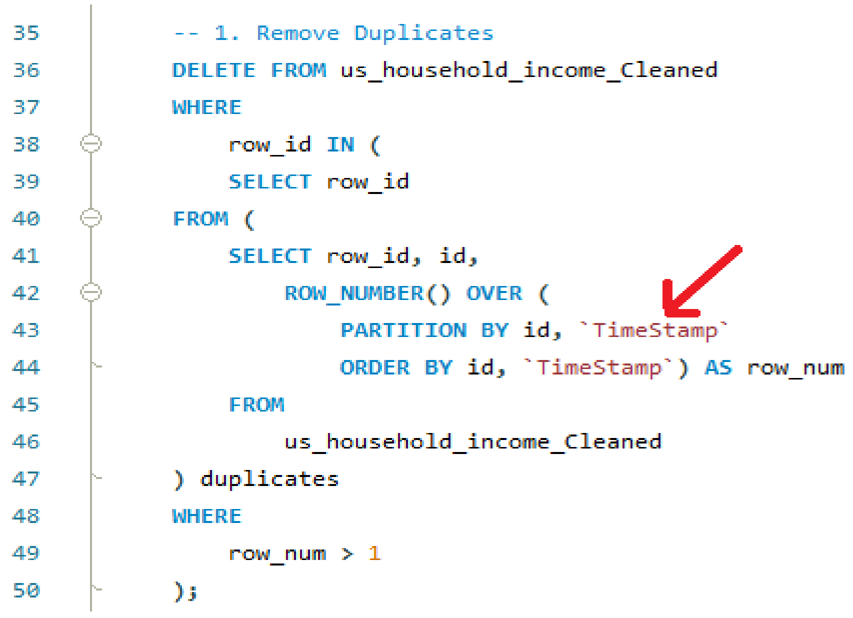
State_Name	COUNT(State_Name)
ARKANSAS	340
CALIFORNIA	3265
COLORADO	513
CONNECTICUT	355
DELAWARE	88
DISTRICT OF COLUMBIA	64
FLORIDA	1658
GEORGIA	818

- Automating the Cleaning Process: Setting up a MySQL event to run the cleaning procedure periodically and creating triggers for real-time data cleaning.



```sql

-- Create Event
DROP EVENT run_data_cleaning;
CREATE EVENT run_data_cleaning
	ON SCHEDULE EVERY 30 DAY
	DO CALL Copy_and_Clean_Data();

-- CREATE TRIGGER
DELIMITER $$
CREATE TRIGGER Transfer_clean_data
	AFTER INSERT ON bakery.us_household_income
	FOR EACH ROW
BEGIN
	CALL Copy_and_Clean_Data();
END $$
DELIMITER ;

INSERT INTO bakery.us_household_income
(`row_id`,`id`,`State_Code`,`State_Name`,`State_ab`,`County`,`City`,`Place`,`Type`,`Primary`,`Zip_Code`,`Area_Code`,`ALand`,`AWater`,`Lat`,`Lon`)
VALUES
(121671,37025904,37,'North Carolina','NC','Alamance County','Charlotte','Alamance','Track','Track',28215,980,24011255,98062070,35.2661197,-80.6865346);
```

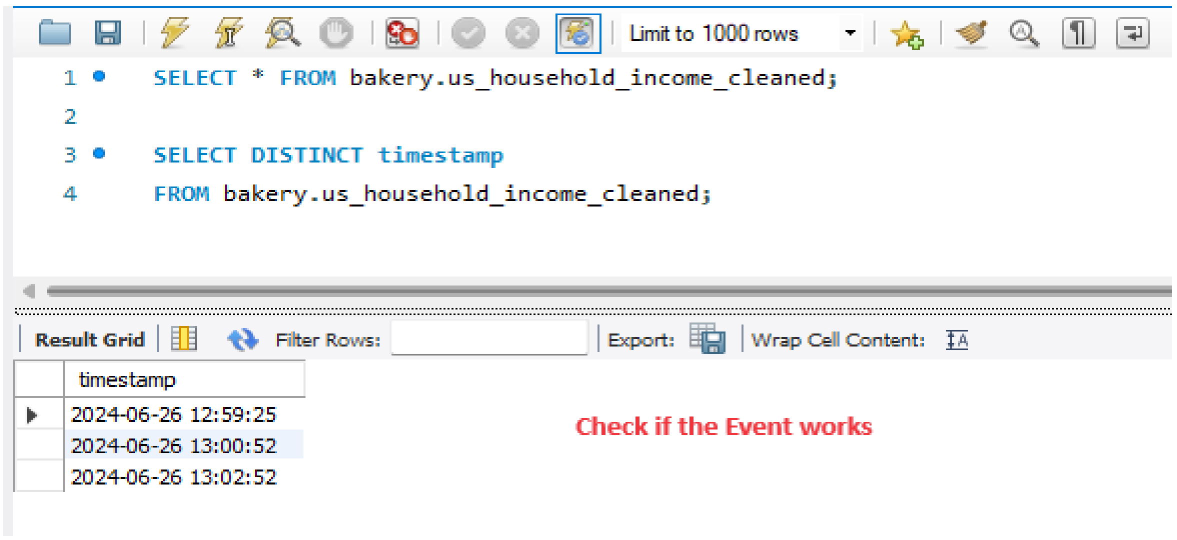
- We might encounter the error of getting the table empty when running the event. That's when 'TimeStamp' comes in. It is unique to the time when the stored procedure runs. So, by including 'TimeStamp' when querying for duplicates, only the timestamped data will be removed when the event runs.



```sql
-- 1. Remove Duplicates
DELETE FROM us_household_income_Cleaned
WHERE
    row_id IN (
    SELECT row_id
FROM (
    SELECT row_id, id,
        ROW_NUMBER() OVER (
            PARTITION BY id, `TimeStamp`
            ORDER BY id, `TimeStamp`) AS row_num
    FROM
        us_household_income_Cleaned
) duplicates
WHERE
    row_num > 1
);
```

- Verification and Validation: Performing queries to check data integrity and consistency in the cleaned data table.

(Initially, I tested with a 2-minute scheduled event, but later changed it to 30 days.)



```sql
SELECT * FROM bakery.us_household_income_cleaned;

SELECT DISTINCT timestamp
FROM bakery.us_household_income_cleaned;
```

timestamp
2024-06-26 12:59:25
2024-06-26 13:00:52
2024-06-26 13:02:52

Check if the Event works

TIPS

When dealing with large datasets, it's essential to optimize SQL queries and processes to ensure efficient performance. Here are a few techniques to consider:

INDEXES:

- Create indexes on columns that are frequently used in WHERE clauses, joins, and sorting operations to speed up query performance.

Common Table Expressions (CTEs):

- Use CTEs for better readability and to break down complex queries into simpler parts.

Batch Processing:

- For large data migrations or updates, consider processing data in smaller batches to avoid long-running transactions and reduce locking issues.