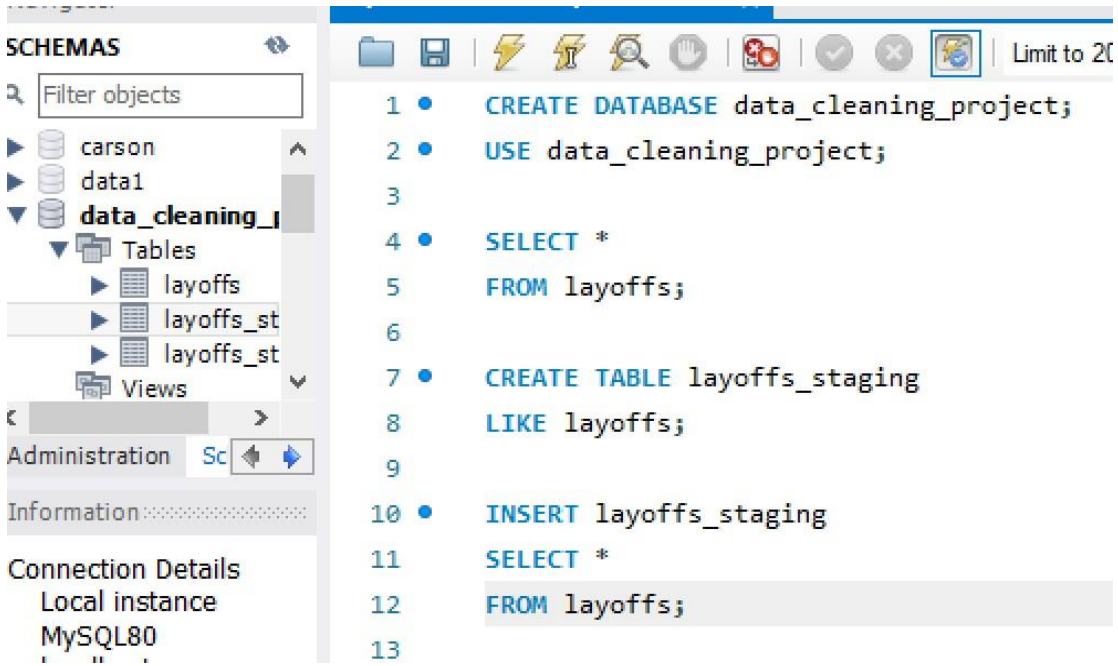
# DATA CLEANING PROJECT

## Project Overview

This project systematically, using MySQL, cleans and standardizes a layoff dataset by applying analytical techniques such as de-duplication using window functions, data normalization, null handling through conditional joins, and type conversion. These steps enhance data quality, consistency, and integrity, enabling more accurate downstream analysis and reliable business intelligence insights.

This SQL code is designed to **clean and standardize** a dataset related to company layoffs.
It begins by duplicating the original layoffs table into a staging environment (layoffs_staging and layoffs_staging2) to perform cleaning without altering the original data.



This SQL code checks for **duplicate records** in the layoffs_staging table. It uses the ROW_NUMBER() window function to assign a sequential number (row_num) to rows that share the same values for company , location, industry, total_laid_off, percentage_laid_off, and date. By partitioning this way, duplicate rows will receive row_num > 1 making it easy to identify and remove.

```sql
SELECT *
FROM layoffs_staging;

# 1. Removing Duplicates

# Checking for duplicates
SELECT *,
ROW_NUMBER() OVER(
PARTITION BY company,location,industry, total_laid_off, percentage_laid_off, `date`) AS row_num
FROM layoffs_staging;
```

```sql
WITH duplicate_cte AS
(
SELECT *,
ROW_NUMBER() OVER(
PARTITION BY company, location, industry, total_laid_off, percentage_laid_off, `date`,
stage, country, funds_raised_millions) AS row_num
FROM layoffs_staging
)
SELECT *
FROM duplicate_cte
WHERE row_num > 1;
```

This code creates a duplicate of the original table with row_num added to it so that any row with row_num greater than one can be filtered and removed.

```sql
# Create new table to add the row_num column
CREATE TABLE `layoffs_staging2` (
  `company` text,
  `location` text,
  `industry` text,
  `total_laid_off` int DEFAULT NULL,
  `percentage_laid_off` text,
  `date` text,
  `stage` text,
  `country` text,
  `funds_raised_millions` int DEFAULT NULL,
  `row_num` INT
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```sql
INSERT INTO layoffs_staging2
SELECT *,
ROW_NUMBER() OVER(
PARTITION BY company, location, industry, total_laid_off, percentage_laid_off, `date`,
stage, country, funds_raised_millions) AS row_num
FROM layoffs_staging;

SELECT *
FROM layoffs_staging2;

# Delete duplicates
DELETE
FROM layoffs_staging2
WHERE row_num > 1;
```

# Standardizing the data:

This section of the code focuses on **standardizing and cleaning specific columns** in the layoffs_staging2 table to ensure data consistency. It begins by trimming whitespaces of the country column. For the industry column, it identifies inconsistencies like “Cryptocurrency” and standardizes them to “Crypto”. The location and country column are reviewed for irregularities including trailing periods in country names like “United States.”, which are then removed. The script also processes date to consistent date format using STR_TO_DATE, and finally alters the column data type to ensure proper date handling. These transformations enhance data uniformity, enabling more reliable analysis.

```sql
# 2. Standardizing Data

# Checking company column
UPDATE layoffs_staging2
SET company = TRIM(company);

# Checking industry column
SELECT DISTINCT(industry)
FROM layoffs_staging2
ORDER BY 1;

SELECT *
FROM layoffs_staging2
WHERE industry LIKE 'Crypto%';
```

```sql
SELECT *
FROM layoffs_staging2
WHERE industry LIKE 'Crypto%';

# Replacing Cryptocurrency with Crypto
UPDATE layoffs_staging2
SET industry = 'Crypto'
WHERE industry LIKE 'Crypto%';

SELECT DISTINCT(industry)
FROM layoffs_staging2;

# Checking LOcations
SELECT DISTINCT(location)
FROM layoffs_staging2
ORDER BY 1;
```

```sql
# Checking country column
SELECT DISTINCT(country), TRIM(TRAILING '.' FROM country)
FROM layoffs_staging2
ORDER BY 1;

SELECT *
FROM layoffs_staging2
WHERE country LIKE 'United States%';

UPDATE layoffs_staging2
SET country = TRIM(TRAILING '.' FROM country)
WHERE country LIKE 'United States%';
```

```sql
# Checking country column
SELECT DISTINCT(country), TRIM(TRAILING '.' FROM country)
FROM layoffs_staging2
ORDER BY 1;

SELECT *
FROM layoffs_staging2
WHERE country LIKE 'United States%';

UPDATE layoffs_staging2
SET country = TRIM(TRAILING '.' FROM country)
WHERE country LIKE 'United States%';
```

```sql
# Working with dates

SELECT `date`,
STR_TO_DATE(`date`, '%m/%d/%Y') AS new_date
FROM layoffs_staging2;

UPDATE layoffs_staging2
SET `date` = STR_TO_DATE(`date`, '%m/%d/%Y');

SELECT *
FROM layoffs_staging2;

# changing data type to date
ALTER TABLE layoffs_staging2
MODIFY COLUMN `date` DATE;
```

**NULL VALUES**

This code handles missing values in the layoffs_staging2 table. It identifies rows with null total_laid_off and percentage_laid_off, and fills missing industry data by referencing other entries from the same company.

```
# Working with NULLS
SELECT *
FROM layoffs_staging2
WHERE total_laid_off IS NULL
AND percentage_laid_off IS NULL;

SELECT t1.industry, t2.industry
FROM layoffs_staging2 t1
JOIN layoffs_staging2 t2
     ON t1.company = t2.company
WHERE (t1.industry IS NULL OR t1.industry = '')
AND t2.industry IS NOT NULL;


UPDATE layoffs_staging2
SET industry = NULL
WHERE industry = '';
```

# Removing unwanted columns

```
# Removing columns and rows we dont need
DELETE
FROM layoffs_staging2
WHERE total_laid_off IS NULL
AND percentage_laid_off IS NULL;

SELECT *
FROM layoffs_staging2;

ALTER TABLE layoffs_staging2
DROP COLUMN row_num;
```

| company | location | industry | total_laid_off | percentage_laid_off | date | stage | country | funds_raised_millions |
|---|---|---|---|---|---|---|---|---|
| Included Health | SF Bay Area | Healthcare | NULL | 0.06 | 2022-07-25 | Series E | United States | 272 |
| &Open | Dublin | Marketing | 9 | 0.09 | 2022-11-17 | Series A | Ireland | 35 |
| #Paid | Toronto | Marketing | 19 | 0.17 | 2023-01-27 | Series B | Canada | 21 |
| 100 Thieves | Los Angeles | Consumer | 12 | NULL | 2022-07-13 | Series C | United States | 120 |
| 10X Genomics | SF Bay Area | Healthcare | 100 | 0.08 | 2022-08-04 | Post-IPO | United States | 242 |
| 1stdibs | New York City | Retail | 70 | 0.17 | 2020-04-02 | Series D | United States | 253 |
| 2TM | Sao Paulo | Crypto | 90 | 0.12 | 2022-06-01 | Unknown | Brazil | 250 |
| 2TM | Sao Paulo | Crypto | 100 | 0.15 | 2022-09-01 | Unknown | Brazil | 250 |
| 2U | Washington D.C. | Education | NULL | 0.2 | 2022-07-28 | 2022-09-01 | United States | 426 |
| 54gene | Washington D.C. | Healthcare | 95 | 0.3 | 2022-08-29 | Series B | United States | 44 |
| 5B Solar | Sydney | Energy | NULL | 0.25 | 2022-06-03 | Series A | Australia | 12 |
| 6sense | SF Bay Area | Sales | 150 | 0.1 | 2022-10-12 | Series E | United States | 426 |
| 80 Acres Farms | Cincinnati | Food | NULL | 0.1 | 2023-01-18 | Unknown | United States | 275 |
| 8x8 | SF Bay Area | Support | 155 | 0.07 | 2023-01-18 | Post-IPO | United States | 253 |
| 8x8 | SF Bay Area | Support | 200 | 0.09 | 2022-10-04 | Post-IPO | United States | 253 |
| 98point6 | Seattle | Healthcare | NULL | 0.1 | 2022-07-21 | Series E | United States | 247 |
| 99 | Sao Paulo | Transportation | 75 | 0.02 | 2022-09-20 | Acquired | Brazil | 244 |
| Abra | SF Bay Area | Crypto | 12 | 0.05 | 2022-06-30 | Series C | United States | 106 |
| Absci | Vancouver | Healthcare | 40 | NULL | 2022-08-09 | Post-IPO | United States | 237 |
| Acast | Stockholm | Media | 70 | 0.15 | 2022-09-15 | Post-IPO | Sweden | 126 |

ayoffs_staging2 54 ×