

SQL PROJECT – DATA CLEANING

Project Overview

Project Name: Data Cleaning in MySQL

Project Description: This project involves cleaning and preparing data related to layoffs from around the world using MySQL.

Prerequisites: MySQL must be installed locally on your machine. If you haven't installed MySQL yet, you can download it from [MySQL Downloads](#) and follow the installation instructions.

Data Source: The dataset used in this project is provided by [Kaggle's dataset repository](#).

About the Dataset

Context:

Tech firms around the globe are facing an economic slowdown. Slow consumer spending, higher interest rates by central banks, and a strong dollar overseas are hinting towards a possible recession, leading tech firms to lay off employees. This economic slowdown recently caused Meta to fire 13% of its workforce, which amounts to more than 11,000 employees.

The data spans from when COVID-19 was declared a pandemic (March 11, 2020) to the present (June 1, 2024). Some data, such as the sources, the list of employees laid off, and the date of addition, has been omitted here.

.

Data Cleaning

Data Cleaning is the process of transforming raw data into a usable format by fixing errors and inconsistencies. This involves identifying and correcting mistakes, removing duplicate records, and ensuring that the data is accurate and complete. Clean data is crucial for creating accurate visualizations, generating reliable insights, and using the data effectively in products and services.

Importance of Data Cleaning

1. **Improves Data Quality:** Ensures the data is accurate, consistent, and free of errors.
2. **Enhances Decision Making:** Provides reliable data for better and more informed business decisions.
3. **Increases Efficiency:** Saves time and effort by reducing the need for later corrections.
4. **Prevents Data Misinterpretation:** Minimizes the risk of drawing incorrect conclusions from the data.
5. **Ensures Compliance:** Helps meet data governance standards and regulatory requirements.
6. **Facilitates Accurate Visualizations:** Ensures that visualizations and models are accurate and useful.
7. **Supports Product Development:** Ensures data used in products and services is reliable, enhancing their quality and performance.

What We Do

In this project, we will create a database, import a real dataset, and clean the data. This process will ensure that the data is accurate and ready for analysis.

Import Data Set into Table

Steps:

1. **Create a New Schema**
 - Click on **New Schema**.
 - Create "**world_layoffs**".
 - Apply → Apply → Finish.
2. **Import Data Set into Table**
 - Go to "**world_layoffs**".
 - Right-click on **Tables**.
 - Select **Table Data Import Wizard**.
 - Select the file to import: "**layoffs**".
 - Select the destination.

- Configure import settings (MySQL will automatically assign data types based on the data in the columns).
- Import data → Next.
- Finish

Cleaning Data

Steps:

1. Remove Duplicates
2. Standardize the Data
3. Handle Null or Blank Values
4. Remove Unnecessary Columns and Rows

```
SELECT *
```

```
FROM world_layoffs.layoffs;
```

-- first thing we want to do is create a staging table. This is the one we will work in and clean the data. We want a table with the raw data in case something happens

```
CREATE TABLE world_layoffs.layoffs_staging
```

```
LIKE world_layoffs.layoffs;
```

```
INSERT layoffs_staging
```

```
SELECT * FROM world_layoffs.layoffs;
```

-- now when we are data cleaning, we usually follow a few steps

-- 1. check for duplicates and remove any

-- 2. standardize data and fix errors

-- 3. Look at null values and see what

-- 4. remove any columns and rows that are not necessary - few ways

-- 1. Remove Duplicates

First let's check for duplicates

```
SELECT *  
FROM world_layoffs.layoffs_staging  
;  
  
SELECT company, industry, total_laid_off, `date`,  
        ROW_NUMBER() OVER (  
            PARTITION BY company, industry, total_laid_off, `date` AS row_num  
        FROM  
            world_layoffs.layoffs_staging;
```

```
SELECT *  
FROM (  
    SELECT company, industry, total_laid_off, `date`,  
           ROW_NUMBER() OVER (  
               PARTITION BY company, industry, total_laid_off, `date`  
           ) AS row_num  
    FROM  
        world_layoffs.layoffs_staging  
    ) duplicates  
WHERE  
    row_num > 1;
```

-- let's just look at oda to confirm

```
SELECT *  
FROM world_layoffs.layoffs_staging
```

```
WHERE company = 'Oda'
```

```
;
```

-- it looks like these are all legitimate entries and shouldn't be deleted. We need to really look at every single row to be accurate

-- these are our real duplicates

```
SELECT *
```

```
FROM (
```

```
    SELECT company, location, industry, total_laid_off,percentage_laid_off,`date`, stage,  
    country, funds_raised_millions,
```

```
        ROW_NUMBER() OVER (
```

```
            PARTITION BY company, location, industry,  
            total_laid_off,percentage_laid_off,`date`, stage, country, funds_raised_millions
```

```
        ) AS row_num
```

```
FROM
```

```
    world_layoffs.layoffs_staging
```

```
) duplicates
```

```
WHERE
```

```
    row_num > 1;
```

-- these are the ones we want to delete where the row number is > 1 or 2 or greater essentially

-- now you may want to write it like this:

```
WITH DELETE_CTE AS
```

```
(
```

```
SELECT *
```

```
FROM (
```

```
    SELECT company, location, industry, total_laid_off,percentage_laid_off,`date`, stage,  
    country, funds_raised_millions,
```

```

        ROW_NUMBER() OVER (
            PARTITION BY company, location, industry,
            total_laid_off,percentage_laid_off,`date`, stage, country, funds_raised_millions
            ) AS row_num
    FROM
        world_layoffs.layoffs_staging
    ) duplicates
WHERE
    row_num > 1
)
DELETE
FROM DELETE_CTE
;

WITH DELETE_CTE AS (
    SELECT company, location, industry, total_laid_off, percentage_laid_off, `date`,
        stage, country, funds_raised_millions,
        ROW_NUMBER() OVER (
            PARTITION BY company, location, industry, total_laid_off,
            percentage_laid_off, `date`, stage, country, funds_raised_millions)
        AS row_num
    FROM world_layoffs.layoffs_staging
)
DELETE FROM world_layoffs.layoffs_staging
WHERE (company, location, industry, total_laid_off, percentage_laid_off, `date`, stage,
country, funds_raised_millions, row_num) IN (
    SELECT company, location, industry, total_laid_off, percentage_laid_off, `date`,
        stage,country, funds_raised_millions, row_num
    FROM DELETE_CTE

```

```
) AND row_num > 1;
```

-- one solution, which I think is a good one. Is to create a new column and add those row numbers in. Then delete where row numbers are over 2, then delete that column

-- so, let's do it!!

```
ALTER TABLE world layoffs. layoffs_staging ADD row_num INT;
```

```
SELECT *
```

```
FROM world layoffs. layoffs_staging
```

```
;
```

```
CREATE TABLE `world layoffs`.` layoffs_staging2` (
```

```
`company` text,
```

```
`location` text,
```

```
`industry` text,
```

```
`total_laid_off` INT,
```

```
`percentage_laid_off` text,
```

```
`date` text,
```

```
`stage` text,
```

```
`country` text,
```

```
`funds_raised_millions` INT,
```

```
row_num INT
```

```
);
```

```
INSERT INTO `world layoffs`.` layoffs_staging2`
```

```
(`company`,
```

```
`location`,
```

```
`industry`,  
`total_laid_off`,  
`percentage_laid_off`,  
`date`,  
`stage`,  
`country`,  
`funds_raised_millions`,  
`row_num`)
```

```
SELECT `company`,
```

```
`location`,
```

```
`industry`,
```

```
`total_laid_off`,
```

```
`percentage_laid_off`,
```

```
`date`,
```

```
`stage`,
```

```
`country`,
```

```
`funds_raised_millions`,
```

```
    ROW_NUMBER() OVER (
```

```
        PARTITION BY company, location, industry, total_laid_off,  
percentage_laid_off, `date`, stage, country, funds_raised_millions
```

```
    ) AS row_num
```

```
FROM
```

```
world_layoffs.layoffs_staging;
```

-- now that we have this we can delete rows where row_num is greater than 2

```
DELETE FROM world_layoffs.layoffs_staging2
```

```
WHERE row_num >= 2;
```


-- 2. Standardize Data

```
SELECT *  
FROM world layoffs layoffs_staging2;
```

-- if we look at industry it looks like we have some null and empty rows, let's take a look at these

```
SELECT DISTINCT industry  
FROM world layoffs layoffs_staging2  
ORDER BY industry;
```

```
SELECT *  
FROM world layoffs layoffs_staging2  
WHERE industry IS NULL  
OR industry = ""  
ORDER BY industry;
```

-- let's take a look at these

```
SELECT *  
FROM world layoffs layoffs_staging2  
WHERE company LIKE 'Bally%';
```

-- nothing wrong here

```
SELECT *  
FROM world layoffs layoffs_staging2  
WHERE company LIKE 'airbnb%';
```

-- it looks like airbnb is a travel, but this one just isn't populated.

-- I'm sure it's the same for the others. What we can do is

-- write a query that if there is another row with the same company name, it will update it to the non-null industry values

-- makes it easy so if there were thousands, we wouldn't have to manually check them all

-- we should set the blanks to nulls since those are typically easier to work with

```
UPDATE world_layoffs.layoffs_staging2
```

```
SET industry = NULL
```

```
WHERE industry = '';
```

-- now if we check those are all null

```
SELECT *
```

```
FROM world_layoffs.layoffs_staging2
```

```
WHERE industry IS NULL
```

```
OR industry = ''
```

```
ORDER BY industry;
```

-- now we need to populate those nulls if possible

```
UPDATE layoffs_staging2 t1
```

```
JOIN layoffs_staging2 t2
```

```
ON t1.company = t2.company
```

```
SET t1.industry = t2.industry
```

```
WHERE t1.industry IS NULL
```

```
AND t2.industry IS NOT NULL;
```

-- and if we check it looks like Bally's was the only one without a populated row to populate this null values

```
SELECT *
```

```
FROM world_layoffs.layoffs_staging2
```

WHERE industry IS NULL

OR industry = ''

ORDER BY industry;

-- I also noticed the Crypto has multiple different variations. We need to standardize that - let's say all to Crypto

SELECT DISTINCT industry

FROM world layoffs. layoffs_staging2

ORDER BY industry;

UPDATE layoffs_staging2

SET industry = 'Crypto'

WHERE industry IN ('Crypto Currency', 'CryptoCurrency');

-- now that's taken care of:

SELECT DISTINCT industry

FROM world layoffs. layoffs_staging2

ORDER BY industry;

-- we also need to look at

SELECT *

FROM world layoffs. layoffs_staging2;

-- everything looks good except apparently we have some "United States" and some "United States." with a period at the end. Let's standardize this.

```
SELECT DISTINCT country
FROM world_layoffs.layoffs_staging2
ORDER BY country;
```

```
UPDATE layoffs_staging2
SET country = TRIM(TRAILING '.' FROM country);
```

-- now if we run this again it is fixed

```
SELECT DISTINCT country
FROM world_layoffs.layoffs_staging2
ORDER BY country;
```

-- Let's also fix the date columns:

```
SELECT *
FROM world_layoffs.layoffs_staging2;
```

-- we can use str to date to update this field

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

-- now we can convert the data type properly

```
ALTER TABLE layoffs_staging2
MODIFY COLUMN `date` DATE;
```

```
SELECT *
FROM world_layoffs.layoffs_staging2;
```

-- 3. Look at Null Values

-- the null values in total_laid_off, percentage_laid_off, and funds_raised_millions all look normal. I don't think I want to change that

-- I like having them null because it makes it easier for calculations during the EDA phase

-- so there isn't anything I want to change with the null values

-- 4. Remove any columns and rows we need to

```
SELECT *  
FROM world_layoffs.layoffs_staging2  
WHERE total_laid_off IS NULL;
```

```
SELECT *  
FROM world_layoffs.layoffs_staging2  
WHERE total_laid_off IS NULL  
AND percentage_laid_off IS NULL;
```

-- Delete Useless data we can't really use

```
DELETE FROM world_layoffs.layoffs_staging2  
WHERE total_laid_off IS NULL  
AND percentage_laid_off IS NULL;
```

```
SELECT *  
FROM world_layoffs.layoffs_staging2;
```

```
ALTER TABLE layoffs_staging2  
DROP COLUMN row_num;
```

```
SELECT *  
FROM world_layoffs.layoffs_staging2;
```

Conclusion

In this project, we successfully performed data cleaning on a dataset related to layoffs from around the world using MySQL. The key steps we took include:

1. **Setting Up the Environment:**
 - Installed MySQL and set up the necessary schema and tables.
2. **Importing the Dataset:**
 - Imported the dataset into a MySQL table for analysis.
3. **Data Cleaning Process:**
 - **Removing Duplicates:** Identified and removed duplicate records to ensure data integrity.
 - **Standardizing Data:** Ensured consistency in data values, such as industry names and country names.
 - **Handling Null Values:** Addressed null values appropriately to maintain data quality.
 - **Removing Unnecessary Columns and Rows:** Removed irrelevant data to streamline the dataset.

Key Outcomes

- The dataset is now clean and ready for further analysis and visualization.
- We have a standardized dataset that can be reliably used for generating insights.

Next Steps

- **Data Analysis:** Perform exploratory data analysis (EDA) to uncover trends and patterns in the cleaned data.
- **Visualization:** Create visualizations to represent the data insights effectively.
- **Reporting:** Compile the findings into a comprehensive report or presentation.