To find duplicate rows in the `WorldLifeExpectancy` dataset based on the `Country` and `Year` columns using MySQL, you can use the following SQL query. This query identifies duplicate `Country` and `Year` combinations and shows how many times each appears.

```sql

SELECT Country, Year, COUNT(\*) as count

FROM WorldLifeExpectancy

GROUP BY Country, Year

HAVING COUNT(\*) > 1;

```

### How It Works in MySQL:

1. \*\*SELECT Country, Year, COUNT(\*) as count\*\*:

- Selects the `Country` and `Year` columns and counts occurrences of each combination using `COUNT(\*)`.

2. \*\*FROM WorldLifeExpectancy\*\*:

- Specifies the `WorldLifeExpectancy` table as the data source.

3. \*\*GROUP BY Country, Year\*\*:

- Groups rows by unique `Country` and `Year` combinations.

4. \*\*HAVING COUNT(\*) > 1\*\*:

- Filters for groups with more than one occurrence, indicating duplicates.

### Expected Results:

Based on the provided dataset, running this query in MySQL would return:

- \*\*Afghanistan, 2018\*\*: Count = 2 (rows with `Row\_ID` 5 and 6).

- \*\*Albania, 2016\*\*: Count = 2 (rows with `Row\_ID` 23 and 24).

### To View Full Duplicate Rows:

If you want to see all columns for the duplicate rows (not just `Country` and `Year`), you can use a subquery in MySQL to retrieve the complete rows:

```sql

SELECT \*

FROM WorldLifeExpectancy

WHERE (Country, Year) IN (

SELECT Country, Year

FROM WorldLifeExpectancy

GROUP BY Country, Year

HAVING COUNT(\*) > 1

)

ORDER BY Country, Year;

```

### How the Second Query Works:

- The subquery identifies `Country` and `Year` combinations with duplicates (same as the first query).

- The outer query selects all columns (`\*`) for rows where `Country` and `Year` match the subquery results.

- `ORDER BY Country, Year` sorts the output for clarity.

### Results from the Second Query:

This would return the full rows for:

- \*\*Afghanistan, 2018\*\*:

- Row 5: `Afghanistan, 2018, Developing, , 275, 71, 7.1, 3013, 17.2, 97, 68, 68, 0.1, 64, 18.2, 18.2, 9.5, 5`

- Row 6: `Afghanistan, 2018, Developing, 58.8, 279, 74, 79.7, 1989, 16.7, 102, 66, 66, 0.1, 553, 18.4, 18.4, 9.2, 6`

- \*\*Albania, 2016\*\*:

- Row 23: `Albania, 2016, Developing, 76.1, 91, 1, 348.1, 0, 53.5, 1, 98, 98, 0.1, 4114, 1.5, 1.6, 12.2, 23`

- Row 24: `Albania, 2016, Developing, 75.3, 1, 1, 36.6, 0, 52.6, 1, 99, 99, 0.1, 438, 1.6, 1.6, 12, 24`

### MySQL-Specific Notes:

- \*\*Table Setup\*\*: Ensure the `WorldLifeExpectancy` table is created in your MySQL database with the correct schema (columns: `Country`, `Year`, `Status`, `Life expectancy`, etc.). You can load the provided Excel data into MySQL using tools like MySQL Workbench or a script to import CSV/Excel data.

- \*\*Case Sensitivity\*\*: MySQL is generally case-insensitive for column names, but ensure the table and column names match exactly as defined in your database.

- \*\*Performance\*\*: For small datasets like this (64 rows), performance is not an issue. For larger datasets, ensure indexes on `Country` and `Year` if you run similar queries frequently:

```sql

CREATE INDEX idx\_country\_year ON WorldLifeExpectancy (Country, Year);

```

- \*\*Handling NULLs\*\*: The dataset has some missing values (e.g., `Life expectancy` in row 5). The queries above will still work, as they only check `Country` and `Year`, which are populated.

### Steps to Execute in MySQL:

1. \*\*Load the Data\*\*:

- Convert the Excel file to a CSV or use MySQL Workbench to import the `WorldLifeExpectancy.xlsx` file.

- Create a table in MySQL with the appropriate schema. Example:

```sql

CREATE TABLE WorldLifeExpectancy (

Country VARCHAR(100),

Year INT,

Status VARCHAR(50),

Life\_expectancy FLOAT,

Adult\_Mortality INT,

infant\_deaths INT,

percentage\_expenditure FLOAT,

Measles INT,

BMI FLOAT,

under\_five\_deaths INT,

Polio INT,

Diphtheria INT,

HIV\_AIDS FLOAT,

GDP FLOAT,

thinness\_1\_19\_years FLOAT,

thinness\_5\_9\_years FLOAT,

Schooling FLOAT,

Row\_ID INT

);

```

- Import the data using a tool or a `LOAD DATA INFILE` command for CSV.

2. \*\*Run the Query\*\*:

- Open your MySQL client (e.g., MySQL Workbench, phpMyAdmin, or command line).

- Execute either the first query (to see duplicate counts) or the second query (to see full duplicate rows).

3. \*\*Review Results\*\*:

- Check the output to confirm the duplicates (Afghanistan 2018, Albania 2016).

- Inspect the full rows to identify differences (e.g., `Life expectancy` or `GDP` variations).

### Additional Notes:

- If you notice discrepancies in the duplicate rows (e.g., row 24 for Albania has an unusually low `Adult\_Mortality` of 1, which may be a data error), you may need to clean the data or decide which row to keep.

- If you want to remove duplicates, you can use a `DELETE` query with a self-join or row numbering, but let me know if you need help with that.

Let me know if you need assistance with setting up the table, importing the data, or further MySQL queries (e.g., deleting duplicates or analyzing differences)!

To delete duplicate rows in the `WorldLifeExpectancy` table based on the `Country` and `Year` columns in MySQL, you need to keep one instance of each duplicate combination and remove the others. Since the dataset includes a unique `Row\_ID` column, we can use it to decide which row to keep (e.g., the row with the lowest `Row\_ID`). Below are two approaches to delete duplicates in MySQL, tailored to your dataset.

### Approach 1: Delete Duplicates Using a Self-Join

This method identifies duplicates and deletes rows with higher `Row\_ID` values for each `Country` and `Year` combination.

```sql

DELETE t1 FROM WorldLifeExpectancy t1

INNER JOIN WorldLifeExpectancy t2

WHERE

t1.Country = t2.Country

AND t1.Year = t2.Year

AND t1.Row\_ID > t2.Row\_ID;

```

#### How It Works:

- \*\*Self-Join\*\*: The table `WorldLifeExpectancy` is joined with itself (`t1` and `t2`) on matching `Country` and `Year`.

- \*\*Condition `t1.Row\_ID > t2.Row\_ID`\*\*: Ensures that for each duplicate pair, the row with the higher `Row\_ID` is deleted, keeping the one with the lower `Row\_ID`.

- \*\*DELETE t1\*\*: Removes the rows from `t1` that meet the join condition.

#### Effect on Your Data:

- \*\*Afghanistan, 2018\*\*:

- Row 5 (`Row\_ID = 5`) will be kept.

- Row 6 (`Row\_ID = 6`) will be deleted.

- \*\*Albania, 2016\*\*:

- Row 23 (`Row\_ID = 23`) will be kept.

- Row 24 (`Row\_ID = 24`) will be deleted.

### Approach 2: Delete Duplicates Using a Subquery with ROW\_NUMBER

If you prefer a more flexible approach or want to control which row to keep based on other criteria (e.g., non-missing `Life expectancy`), you can use the `ROW\_NUMBER()` function (available in MySQL 8.0+).

```sql

WITH RankedRows AS (

SELECT \*,

ROW\_NUMBER() OVER (PARTITION BY Country, Year ORDER BY Row\_ID) AS rn

FROM WorldLifeExpectancy

)

DELETE FROM WorldLifeExpectancy

WHERE Row\_ID IN (

SELECT Row\_ID

FROM RankedRows

WHERE rn > 1

);

```

#### How It Works:

- \*\*ROW\_NUMBER()\*\*: Assigns a unique number (`rn`) to each row within each `Country` and `Year` group, ordered by `Row\_ID`.

- \*\*PARTITION BY Country, Year\*\*: Groups rows by `Country` and `Year`.

- \*\*ORDER BY Row\_ID\*\*: Ensures the row with the lowest `Row\_ID` gets `rn = 1`.

- \*\*DELETE\*\*: Removes rows where `rn > 1`, keeping only the first row per duplicate group.

#### Effect on Your Data:

- Same as Approach 1: Keeps `Row\_ID` 5 for Afghanistan 2018 and `Row\_ID` 23 for Albania 2016, deletes the others.

### Choosing Which Row to Keep:

In your dataset, duplicates have differences (e.g., Afghanistan 2018 has missing `Life expectancy` in row 5 but not in row 6). If you want to keep rows based on specific criteria (e.g., non-missing `Life expectancy`), modify the `ORDER BY` in Approach 2. For example, to prioritize rows with non-NULL `Life expectancy`:

```sql

WITH RankedRows AS (

SELECT \*,

ROW\_NUMBER() OVER (PARTITION BY Country, Year ORDER BY Life\_expectancy IS NOT NULL DESC, Row\_ID) AS rn

FROM WorldLifeExpectancy

)

DELETE FROM WorldLifeExpectancy

WHERE Row\_ID IN (

SELECT Row\_ID

FROM RankedRows

WHERE rn > 1

);

```

- \*\*ORDER BY Life\_expectancy IS NOT NULL DESC, Row\_ID\*\*: Prioritizes rows with non-NULL `Life expectancy`, and for ties, uses the lowest `Row\_ID`.

#### Effect with This Criteria:

- \*\*Afghanistan, 2018\*\*: Keeps row 6 (`Row\_ID = 6`, `Life expectancy = 58.8`) and deletes row 5 (missing `Life expectancy`).

- \*\*Albania, 2016\*\*: Keeps row 23 (`Row\_ID = 23`, `Life expectancy = 76.1`) and deletes row 24 (`Life expectancy = 75.3`), assuming you prefer the higher `Life expectancy` or lower `Row\_ID`.

### Steps to Execute in MySQL:

1. \*\*Backup Your Data\*\*:

- Before deleting, create a backup of your table to avoid accidental data loss:

```sql

CREATE TABLE WorldLifeExpectancy\_backup AS

SELECT \* FROM WorldLifeExpectancy;

```

2. \*\*Verify Duplicates First\*\*:

- Run the following to confirm which rows will be affected:

```sql

SELECT \*

FROM WorldLifeExpectancy

WHERE (Country, Year) IN (

SELECT Country, Year

FROM WorldLifeExpectancy

GROUP BY Country, Year

HAVING COUNT(\*) > 1

)

ORDER BY Country, Year;

```

3. \*\*Execute the Delete Query\*\*:

- Choose one of the above queries (Approach 1 or Approach 2) based on your MySQL version and preference.

- Run the query in your MySQL client (e.g., MySQL Workbench, phpMyAdmin, or command line).

4. \*\*Verify Results\*\*:

- Check that duplicates are removed:

```sql

SELECT Country, Year, COUNT(\*) as count

FROM WorldLifeExpectancy

GROUP BY Country, Year

HAVING COUNT(\*) > 1;

```

This should return no rows if duplicates are successfully deleted.

### MySQL-Specific Notes:

- \*\*MySQL Version\*\*: Approach 2 requires MySQL 8.0+ for `ROW\_NUMBER()`. If you're using an older version (e.g., 5.7), use Approach 1.

- \*\*Permissions\*\*: Ensure you have `DELETE` privileges on the database.

- \*\*Transaction Safety\*\*: Wrap the delete in a transaction to allow rollback if needed:

```sql

START TRANSACTION;

-- Run the DELETE query (e.g., Approach 1 or 2)

-- Verify results

SELECT \* FROM WorldLifeExpectancy WHERE Country = 'Afghanistan' AND Year = 2018;

SELECT \* FROM WorldLifeExpectancy WHERE Country = 'Albania' AND Year = 2016;

-- If satisfied, commit

COMMIT;

-- If not, rollback

-- ROLLBACK;

```

- \*\*Data Integrity\*\*: Inspect the duplicates before deletion. For example, row 24 for Albania 2016 has an `Adult\_Mortality` of 1, which seems like a data error. You might prefer to keep row 23 or manually correct the data.

### Recommendations:

- If data quality matters (e.g., keeping rows with valid `Life expectancy`), use the modified `ROW\_NUMBER()` query with custom `ORDER BY`.

- If you just want the simplest approach, use the self-join (Approach 1) to keep the lowest `Row\_ID`.

- After deletion, you may want to reindex `Row\_ID` or update it to be sequential if needed:

```sql

SET @row\_number = 0;

UPDATE WorldLifeExpectancy

SET Row\_ID = (@row\_number := @row\_number + 1)

ORDER BY Row\_ID;

```

Let me know if you need help executing these queries, choosing specific criteria for keeping rows, or further assistance with MySQL!