```sql
# World Life Expectancy Project (Data Cleaning)

# Look at the data after having imported them into MySQL Workbench.
SELECT *
FROM world_life_expectancy
;

# Even though Alex separates data cleaning from exploratory data analysis,
# he says that in the real world both are likely to be performed at the same time.

# Each row has a row ID, so duplicates can be determined by combining values from two different
columns,
# kinda like composite primary keys in the context of database administration. In the case of this project,
# the combination of country and year can help identify unique rows since there should be one row
# (i.e., a combination of values) for each combination of country and year.

SELECT Country, Year, CONCAT(Country, Year)
FROM world_life_expectancy
;

# Everything in the column CONCAT(Country, Year) should be unique; however,
# Alex quickly found a duplicate (two instances of Zimbabwe2019).

# Run the same query as before but with a count of instances of CONCAT(Country, Year).
# Reminder: Aggregate functions in the SELECT statement require a GROUP BY statment
# later on in the same query.
SELECT Country, Year, CONCAT(Country, Year), COUNT(CONCAT(Country, Year))
FROM world_life_expectancy
GROUP BY Country, Year, CONCAT(Country, Year)
;

# Run the same query as before but with a HAVING statement this time
# to idintify counts under COUNT(CONCAT(Country, Year)) greater than 1.
# Reminder: Aggregate functions in the SELECT statement require a GROUP BY statment
# later on in the same query.
SELECT Country, Year, CONCAT(Country, Year), COUNT(CONCAT(Country, Year))
FROM world_life_expectancy
GROUP BY Country, Year, CONCAT(Country, Year)
HAVING 1 < COUNT(CONCAT(Country, Year))
;

# Look at the data once again and take note of the column Row_ID.
SELECT *
FROM world_life_expectancy
;

# Generate row numbers over partitions.
SELECT Row_ID,
        CONCAT(Country, Year),
   ROW_NUMBER() OVER(PARTITION BY CONCAT(Country, Year) ORDER BY CONCAT(Country, Year)) AS
Row_Num
FROM world_life_expectancy
;

# In order to filter on the result of the above query,
# the above query needs to be used as a subquery.
SELECT *
FROM (
        SELECT Row_ID,
        CONCAT(Country, Year),
   ROW_NUMBER() OVER(PARTITION BY CONCAT(Country, Year) ORDER BY CONCAT(Country, Year)) AS
Row_Num
        FROM world_life_expectancy
) AS row_table
WHERE 1 < Row_Num
;

# Row IDs that appear in the above query will become the basis
# for deleting duplicates in the world life expectancy table.

# Alex goes through the steps of creating a backup of the world life expectancy table
# (named world_life_expectancy_backup) in order to demonstrate best practices
# when doing data cleaning in the real world.

# The above query will be used in the WHERE statement of the next query
# in order to identify which rows to delete. However, instead of SELECT * like
# in the first subquery, SELECT Row_ID will be used instead.
# I'm going to first use a SELECT * instead of going straight to a DELETE statement
# just to confirm that I'm about to delete what I want to delete.
# (This is just a me thing. Sometimes bad luck does happen)
SELECT *
FROM world_life_expectancy
WHERE Row_ID IN (
        SELECT Row_ID
        FROM (
                SELECT Row_ID,
                CONCAT(Country, Year),
                ROW_NUMBER() OVER(PARTITION BY CONCAT(Country, Year) ORDER BY
CONCAT(Country, Year)) AS Row_Num
                FROM world_life_expectancy
        ) AS row_table
        WHERE 1 < Row_Num
)
;

# The results of the above query look good, so I'll go back to following along
# with what Alex is doing in the video.
DELETE
FROM world_life_expectancy
WHERE Row_ID IN (
    SELECT Row_ID
    FROM (
        SELECT Row_ID,
        CONCAT(Country, Year),
        ROW_NUMBER() OVER(PARTITION BY CONCAT(Country, Year) ORDER BY
CONCAT(Country, Year)) AS Row_Num
        FROM world_life_expectancy
    ) AS row_table
    WHERE 1 < Row_Num
)
;

# Run the following query. Nothing should show up.
SELECT *
    FROM (
        SELECT Row_ID,
        CONCAT(Country, Year),
        ROW_NUMBER() OVER(PARTITION BY CONCAT(Country, Year) ORDER BY
CONCAT(Country, Year)) AS Row_Num
        FROM world_life_expectancy
    ) AS row_table
    WHERE 1 < Row_Num
;

# Alex points out that figuring out how to remove duplicates in a dataset may not be intuitive.

# Look at the data once again to prepare for the next phase of the data cleaning process.
SELECT *
FROM world_life_expectancy
;

# Look for the rows where the status of the country is blank.
# The word NULL doesn't appear in queries for that column,
# so the value is likely just a blank/empty string
SELECT *
FROM world_life_expectancy
WHERE Status = ''
;

# Look at the statuses among the countries that contain at least one blank
# for their development status
# Alex uses the <> combination for inequality; I choose the != combination.
# The outcome should be the same.
SELECT DISTINCT Status
```

```
FROM world_life_expectancy
WHERE Status != ''
;

# Look at countries with the 'Developing' status
SELECT DISTINCT country
FROM world_life_expectancy
WHERE Status = 'Developing'
;

# Attempt to update the world life expectancy table
# so that 'Developing' countries are set
# to 'Developing' in order to fill any blanks in status column
# for developing countries
UPDATE world_life_expectancy
SET status = 'Developing'
WHERE country IN (
        SELECT DISTINCT country
        FROM world_life_expectancy
        WHERE Status = 'Developing'
)
;

 # The above query failed to execute becuase the table being subjected
 # to the UPDATE statement can't show up in the FROM statement (of the subquery
 # in this case).

 # Take a differne approach - perform a self-join on the world life expectancy table,
 # and then try to update the table from there
 # Again, Alex uses <> for inequality, while I use != insetad. The outcome
 # should still be the same.
 UPDATE world_life_expectancy AS t1
 INNER JOIN world_life_expectancy AS t2
        ON t1.Country = t2.Country
SET t1.Status = 'Developing'
WHERE t1.Status = ''
        AND t2.Status != ''
   AND t2.Status = 'Developing'
;

# Thinking about it, the statement AND t2.Status != '' in the above query feels unnecessary
# because if t2.Status = 'Developing', then it is also the case that t2.Status != '', ya know?
# However, I'll still follow along with what Alex has so that I don't in advertently
# become confused later on.

# Run a query to confirm that the change had gone through
SELECT *
FROM world_life_expectancy
```

```
WHERE Status = ''
;

# Run a query to look at the development status
# of the United States of America
SELECT *
FROM world_life_expectancy
WHERE Country = 'United States of America'
;

 # Perform a self-join on the world life expectancy table again,
 # and then try to update the table from there once more

# Prior to attempting the same update as Alex,
# I'm going to just run an INNER JOIN
# of the world life expectancy with itself on country
# and then just look at the rows for the United States of America
# in order to understand how the previous update worked.
SELECT *
FROM world_life_expectancy AS t1
 INNER JOIN world_life_expectancy AS t2
              ON t1.Country = t2.Country
WHERE t1.Country = 'United States of America'
;

# So it looks like performing such an INNER JOIN produced
# what looks like the result of a CROSS JOIN. The output window
# said that 256 row(s) had been returned, and the square root of 256 is 16
# according to a Google search.

SELECT Country, COUNT(Country)
FROM world_life_expectancy
WHERE Country = 'United States of America'
;

# The above query returns 16 for the number of rows for the United States of America,
# which matches the square root of 256, so indeed it does look like performing an INNER JOIN
# of a table with itself along a value that repeats itself across all relevant rows
# does result in a CROSS JOIN. (Good to know, eh?)
# Therefore, it's possible
# for a country-year combination that's missing a development status
# to be paired with country-year combination that isn't missing a development status.

 # Again, Alex uses <> for inequality, while I use != insetad. The outcome
 # should still be the same.
 # Also, Alex opts to just copy-and-paste the value 'Developed'
 # instead of typing it out like I did in order to safeguard against
 # spelling something incorrectly. That makes a lot of sense, so going forward
 # I'll try to copy-and-paste values from the table(s) when applicable.
UPDATE world_life_expectancy AS t1
INNER JOIN world_life_expectancy AS t2
        ON t1.Country = t2.Country
SET t1.Status = 'Developed'
WHERE t1.Status = ''
        AND t2.Status != ''
   AND t2.Status = 'Developed'
;

# Run a query to look at the development status
# of the United States of America.
SELECT *
FROM world_life_expectancy
WHERE Country = 'United States of America'
;

# Look for any other rows with a blank development status.
SELECT *
FROM world_life_expectancy
WHERE Status = ''
;

# Also check for any other rows with a NULL value in the Status column.
# (These NULL values should've been addressed earlier,
# but it doesn't hurt to check again.)
SELECT *
FROM world_life_expectancy
WHERE Status = NULL
;

# Look at the table once more.
# Alex says that he likes to keep copies of queries in the same order that he's thinking
# through the problem, even if such copies can become redudnant.
# I'm inclined to do the same since doing so helps maintain the thought process,
# especially when comments are added in to explicitly explain things.
SELECT *
FROM world_life_expectancy
;

# A concern that I have here is that up to this point Alex didn't explicitly factor in
# the possibility of a country's development status changing from 'Developing' to 'Developed'
# which then introduces the possibility that a blank status could appear
# between a row that that has 'Developing' and a row that says 'Developed"
# when ordering those rows by year.
# I suppose such a concern could be something that I myself could keep track of for this guided project
# or take into consideration when working to clean data in the real world.
# Isolate just the rows with blank values under the life expectancy column
SELECT *
FROM world_life_expectancy
WHERE `Life expectancy` = ''
;

# The above query returns a row for Afghanistan and a row for Albania.

# Review the full table once again.
SELECT *
FROM world_life_expectancy
;

# Alex briefly looks over the data for Afghanistan in the full table, and
# he notices that the numbers in the life expectancy column for that country are
# gradually increasing, so he recommends populating the blank for that country
# with the average of two numbers - the number of the year for the country before that blank
# and the number of the year for the country after that blank, provided that the rows are ordered
# by year for that country.
# Moreover, he says that later on when he goes through exploratory data analysis (EDA), it would be a
good idea
# to have each blank space populated with some a data point.
# Trying to populate the blank with the average of the values above and below it when the rows are
seroted
# can be tricky, but Alex says that that's just how things go for data analysts.

# Focus on a few columns of the world life expectancy table.
SELECT Country, Year, `Life expectancy`
FROM world_life_expectancy
;

# Attempt two self-joins between the world life expectancy table and itself.
# Remember to specify the table for the column names in the SELECT statement.
SELECT t1.Country, t1.Year, t1.`Life expectancy`,
        t2.Country, t2.Year, t2.`Life expectancy`,
  t3.Country, t3.Year, t3.`Life expectancy`
FROM world_life_expectancy AS t1
INNER JOIN world_life_expectancy AS t2
        ON t1.Country = t2.Country
  AND t1.Year = (t2.Year - 1)
INNER JOIN world_life_expectancy AS t3
        ON t1.Country = t3.Country
  AND t1.Year = (t3.Year + 1)
;

# Now perform the above query again, but with focus placed
# on that one row with the blank value under life expectancy.
SELECT t1.Country, t1.Year, t1.`Life expectancy`,
            t2.Country, t2.Year, t2.`Life expectancy`,
   t3.Country, t3.Year, t3.`Life expectancy`
FROM world_life_expectancy AS t1
INNER JOIN world_life_expectancy AS t2
            ON t1.Country = t2.Country
   AND t1.Year = (t2.Year - 1)
INNER JOIN world_life_expectancy AS t3
            ON t1.Country = t3.Country
   AND t1.Year = (t3.Year + 1)
WHERE t1.`Life expectancy` = ''
;

# Now perform the above query again, but add a column
# with the average of the second and third life expectancy columns.
SELECT t1.Country, t1.Year, t1.`Life expectancy`,
            t2.Country, t2.Year, t2.`Life expectancy`,
   t3.Country, t3.Year, t3.`Life expectancy`,
   ROUND((t2.`Life expectancy` + t3.`Life expectancy`)/2, 1)
FROM world_life_expectancy AS t1
INNER JOIN world_life_expectancy AS t2
            ON t1.Country = t2.Country
   AND t1.Year = (t2.Year - 1)
INNER JOIN world_life_expectancy AS t3
            ON t1.Country = t3.Country
   AND t1.Year = (t3.Year + 1)
WHERE t1.`Life expectancy` = ''
;

# Update the world life expectancy table using what had been found with the previous query
# as well as what had been discovered/learned during the previous update.
# Reminder: The table selected for updating can't show up in the FROM statement.
# The "double self-join" will go in the UPDATE statement.
# The calculated life expectancy value should only be applied to the rows
# where the life expectancy is blank.
UPDATE (world_life_expectancy AS t1
INNER JOIN world_life_expectancy AS t2
            ON t1.Country = t2.Country
   AND t1.Year = (t2.Year - 1)
INNER JOIN world_life_expectancy AS t3
            ON t1.Country = t3.Country
   AND t1.Year = (t3.Year + 1)
)
SET t1.`Life expectancy` = ROUND((t2.`Life expectancy` + t3.`Life expectancy`)/2, 1)
WHERE t1.`Life expectancy` = ''
;

# Look at the "double self-join" again to confirm that the update went through as intended.
SELECT t1.Country, t1.Year, t1.`Life expectancy`,
            t2.Country, t2.Year, t2.`Life expectancy`,
   t3.Country, t3.Year, t3.`Life expectancy`,
   ROUND((t2.`Life expectancy` + t3.`Life expectancy`)/2, 1)
FROM world_life_expectancy AS t1
INNER JOIN world_life_expectancy AS t2
            ON t1.Country = t2.Country
   AND t1.Year = (t2.Year - 1)
INNER JOIN world_life_expectancy AS t3
            ON t1.Country = t3.Country
   AND t1.Year = (t3.Year + 1)
WHERE t1.`Life expectancy` = ''
;

# No rows showed up in the result of the above query, so that's neat.

# Look at the world life expectancy table again
# with a focus on just a few columns
SELECT Country, Year, `Life expectancy`
FROM world_life_expectancy
;

# Check for any rows with blanks in the life expectancy column.
SELECT Country, Year, `Life expectancy`
FROM world_life_expectancy
WHERE `Life expectancy` = ''
;

# Nothing shows up - neat.

# According to Alex, figuring out the calculations to replace the blank values
# is not easy, but going through the steps does help with future data cleaning procedures.

# I do have a conern here, though. When looking at the trend of life expectancy values
# to determine how to replace a blank value, Alex only looked at the numbers for Afghanistan.
# He didn't look at the values for Albania. And yet, for both Afghanistan and Albania,
# he decided to calculate the average of "flanking" life expectancy values for each country to replace
their
# blank values. In the grand scheme of it all, such a decision
# might not negatively impact the exploratory data analysis too much later on; however, I feel like
# both the values for Afghanistan and Albania should've been observed separately before deciding on
# how to replace the blank values for each country. Maybe all of this is something I can keep in mind
# when doing my own data cleaning in the future.

# Bring up the whole life expectancy table one more time.
SELECT *
FROM world_life_expectancy
;
```

```
# It doesn't look like there's anything else to address at this point. However, Alex points out
# that it's possilbe that more deficiencies in the data can be found when doing EDA later on,
# so those can be addressed then. Also, he points out that all of these steps taken to clean this dataset
# is basically how things go in the real world when it comes to cleaning any dataset.
```