מבוא

תיעוד המערכת – סדנה במסדי נתונים (89679)

מגישים: נועם רוט (212450431), טל סיגמן (322888389), יאיר לוי (322884834)

רעיון המערכת

אנשים רבים מתעניינים בשינויים היומיים והמגמות השונות של התפשטות מגפת הקורונה בכל העולם. עם זאת, קיים קושי בהצגת כלל המידע על ההתפשטות המושפעת ממאפיינים שונים בכל מדינה (גיל חציוני, צפיפות אוכלוסין, גודל האוכלוסיה מצב כלכלי ועוד), באופן ברור ונגיש.

לכן, קיים הצורך בהעברת המידע באופן נגיש, ויזואלי (ע"י גרפים, מפות ועוד) וידידותי למשתמש, שיציג את התמונה המלאה על המגפה במקומות שונים. המערכת תציג את הנתונים ותאפשר השוואה של נתוני הקורונה בין מדינות, יבשות שונות ואף עבור אותה מדינה בפרקי זמן שונים, וכן הצגה עפ"י מאפיינים רלוונטיים של המדינות.

המידע שעליו בנויה המערכת

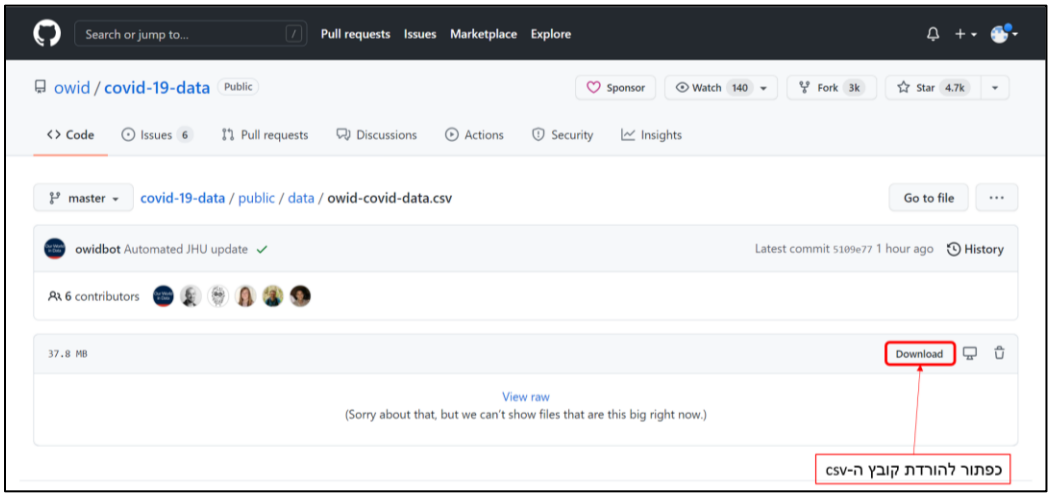
הנתונים שיוצגו במערכת הם נתונים הנאספו במשך שנה וחצי (החל מ-22.01.2020) על מגפת הקורונה, ומתעדכנים מספר פעמים ביום (מספרי נדבקים ומתים – יומיים ומצטברים) לפי מדינות שונות ונתונים שלהן (יבשת, צפיפות אוכלוסין, גיל חציוני וכו').

ה-dataset שמהנתונים שלו תתבסס המערכת נלקח מהאתר OurWorldInData:

https://ourworldindata.org/coronavirus-source-data

שמקור המידע שלו הוא Johns Hopkins University, וממנו נבחר קובץ ה-csv שמתוכו נבחרו העמודות הרלוונטיות למימוש הפרויקט שלנו:

https://github.com/owid/covid-19-data/blob/master/public/data/owid-covid-data.csv

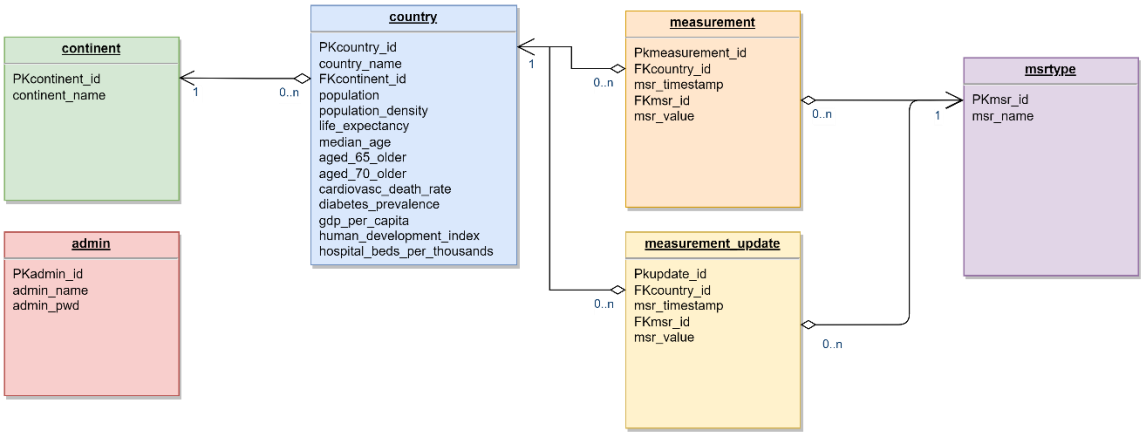


ב-dataset זה ישנן למעלה מ-100K רשומות עם יותר מ-25 עמודות שונות בתחומים הרלוונטיים.

כל המידע המופיע בפרויקט נבחר מטבלה זו (ששונתה במבנה, אך לא בתוכן בהתאם לרעיון הפרויקט).

סכמת בסיס הנתונים של המערכת

מבנה מסד הנתונים של המערכת ייבנה באופן המאפשר הרחבת מודל הנתונים בעתיד ותחזוקתיות קלה של המערכת (החיצים שבין הטבלאות מקשרים ביניהן לפי הקשרים בין הישויות אותן מייצגות: one-to-many):



כעת, נתאר את העמודות בכל טבלה ואת סוג המידע (מחרוזת, ערך נומרי וכו') שמכילה כל עמודה, ואת הקשרים שבין הטבלאות (וסוג הקשר ביניהן).

טבלת continent

טבלה המכילה מידע על היבשות השונות:

עמודה	שם עמודה	טיפוס	הערות
.1	**PKcontinent_id**	TINYINT	מספר הזיהוי של היבשת
.2	continent_name	VARCHAR(45)	שם היבשת

טבלת country

טבלה המכילה מידע סטטי (לא מדידה תלוית זמן) על המדינות השונות:

עמודה	שם עמודה	טיפוס	הערות
.1	**PKcountry_id**	INT	מספר הזיהוי של המדינה
.2	country_name	VARCHAR(45)	שם המדינה
.3	FKcontinent_id	TINYINT	מספר הזיהוי של היבשת בה נמצאת המדינה – foreign key
.4	population	INT	גודל האוכלוסייה
.5	population_density	FLOAT	צפיפות האוכלוסייה
.6	life_expectancy	FLOAT	תוחלת חיים ממוצעת
.7	median_age	FLOAT	גיל חציוני
.8	aged_65_older	FLOAT	אחוז מהאוכלוסייה מעל גיל 65
.9	aged_70_older	FLOAT	אחוז מהאוכלוסייה מעל גיל 70
.10	cardiovasc_death_rate	FLOAT	אחוז מהאוכלוסייה עם מחלות לב
.11	diabetes_prevalence	FLOAT	אחוז מהאוכלוסייה עם סוכרת
.12	gdp_per_capita	DOUBLE	התמ"ג של המדינה
.13	human_development_index	FLOAT	מדד הפיתוח של המדינה
.14	hospital_beds_per_thousand	FLOAT	מספר מיטות בבתי חולים ביחס לגודל האוכלוסייה

סכמת בסיס הנתונים של המערכת

טבלת measurement

טבלת הנתונים המרכזית - טבלה המכילה מידע על המדידות השונות (מדידות של נתוני קורונה):

עמודה	שם עמודה	טיפוס	הערות
1.	**PKmeasurement_id**	INT	מספר הזיהוי של המדידה
2.	FKcountry_id	INT	מספר הזיהוי של המדינה שבה נערכה המדידה – foreign key
3.	msr_timestamp	DATE	זמן קיום המדידה
4.	FKmsr_id	INT	מספר הזיהוי של סוג המדידה – foreign key
5.	msr_value	DOUBLE	ערך המדידה

מבנה נתונים זה מאפשר:

1. הוספה עתידית קלה של מדדים נוספים
2. מאפשר קליטת מדדים חלקיים ממדינות ובתקופות מסויימות
3. שונות בין המדדים הנאספים מכל מדינה

טבלת msrtype

טבלה המכילה מידע על **סוגי** המדידות השונות (מדידות של נתוני קורונה):

עמודה	שם עמודה	טיפוס	הערות
1.	**PKmsr_id**	INT	מספר הזיהוי של סוג המדידה
2.	msr_name	VARCHAR(45)	שם קטגוריית המדידה

טבלת measurement_update

טבלת עדכון הנתונים - טבלה המכילה מידע על עדכונים של מדידות שונות של נתוני קורונה:

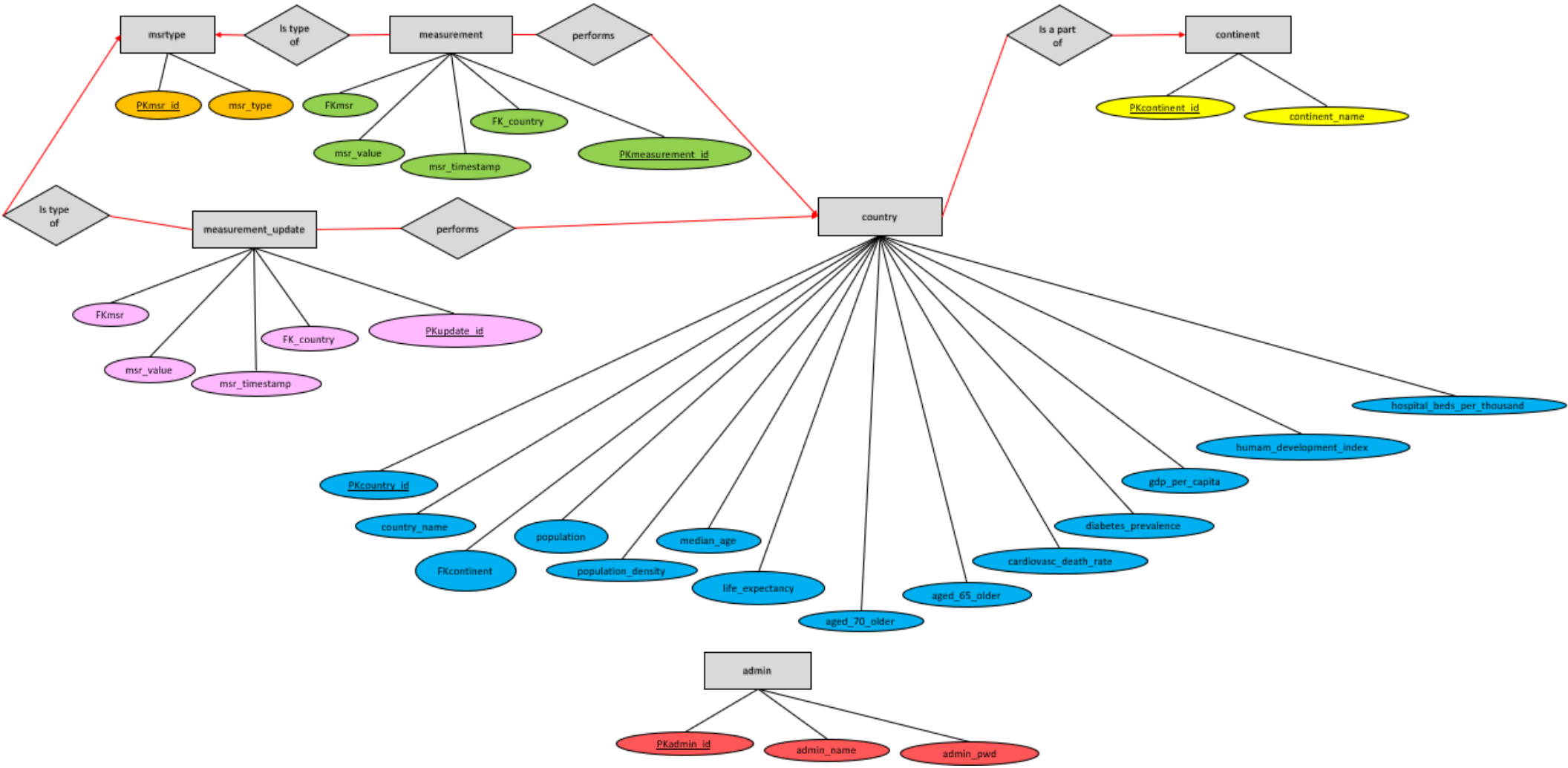
עמודה	שם עמודה	טיפוס	הערות
1.	**PKupdate_id**	INT	מספר הזיהוי של המדידה המעודכנת (המתווספת)
2.	FKcountry_id	INT	מספר הזיהוי של המדינה שבה נערכה המדידה המעודכנת – foreign key
3.	msr_timestamp	DATE	זמן קיום המדידה המעודכנת
4.	FKmsr_id	INT	מספר הזיהוי של סוג המדידה המעודכנת – foreign key
5.	msr_value	DOUBLE	ערך המדידה המעודכנת

טבלת admin

טבלה המכילה מידע על מנהלי המערכת (שיכולים לאשר/ לדחות עדכונים המוצעים ע"י משתמשים):

עמודה	שם עמודה	טיפוס	הערות
1.	**PKadmin_id**	INT	מספר הזיהוי של מנהל המערכת
2.	admin_name	VARCHAR(45)	שם המנהל
3.	admin_pwd	VARCHAR(45)	סיסמת המנהל

תרשים הישויות והקשרים ביניהן



שאילתות פשוטות המופיעות בפרויקט (בקובץ oop_queries.py)

קריאת הנתונים הבסיסיים (יבשת ואוכלוסייה) של כל מדינה להצגה במפה (בפונקציה get_countries):

```
SELECT continent.continent_name, country.country_name, country.population
FROM continent,country
WHERE continent.PKcontinent_id = country.FKcontinent_id
```

רשימת המשתנים הדינאמיים האפשריים למדינה (בפונקציה get_variables):

```
SELECT msr_name FROM msrtype
```

קריאת התאריך המאוחר ביותר בו נאספו נתונים (בפונקציה get_dates):

```
SELECT MAX(msr_timestamp) FROM measurement
```

קריאת התאריך המוקדם ביותר בו נאספו נתונים (בפונקציה get_dates):

```
SELECT MIN(msr_timestamp) FROM measurement
```

בהינתן משתנה מבין (new_cases, new_deaths), ותאריך מסוים, השאילתא תחזיר את סכום ערכי המשתנה שנמדד בכל מדינות העולם בדיוק תאריך שקיבלנו עבור המשתנה הזה (ולא בתאריך הכי מאוחר שקרוב לתאריך שקיבלנו)

(get_info_of_variable_in_specific_date):

```
SELECT sum(msr_value)
FROM measurement USE INDEX(searchIndex)
WHERE msr_timestamp = '{}'
AND FKmsr_id = (select PKmsr_id FROM msrtype WHERE msr_name = '{}')
```

קריאת הנתונים בכל תאריך עבור מדינה נתונה וסוג מדידה נתון (בפונקציה get_data_for_scatter_line_graph):

```
SELECT msr_timestamp, msr_value
FROM measurement
WHERE FKcountry_id = (SELECT PKcountry_id FROM country WHERE country_name LIKE "{0}")
AND FKmsr_id = (ELECT PKmsr_id FROM msrtype WHERE msr_name LIKE "{1}")
```

החזרת ערך משתנה סטטי של מדינה נתונה (בפונקציה get_static_data):

```
SELECT {0}
FROM country
WHERE PKcountry_id = (SELECT PKcountry_id FROM country WHERE country_name LIKE "{1}")
```

רשימת כל המדינות כדי שהמשתמש יוכל לבחור מדינה לעדכן בה נתונים (בפונקציה user_update):

```
SELECT PKcountry_id FROM country WHERE country_name LIKE '{0}'
```

רשימת כל המדידות כדי שהמשתמש יוכל לבחור מדידה שהנתונים שלה יעודכנו (בפונקציה user_update):

```
SELECT PKmsr_id FROM msrtype WHERE msr_name = '{0}'
```

הכנסת הבקשה לעדכון לרשימת העדכונים [השדות והערכים נתונים בשאילתא]

(בפונקציות user_update | update_measurements_table | check_for_adding_another_measurement):

```
INSERT INTO measurement_update ({0}) VALUES (%s, %s, %s, %s)
```

בדיקה האם המשתמש והסיסמה [הנתונים] של ה-admin נכונים (בפונקציה check_admin):

```
SELECT EXISTS(
        SELECT admin_name, admin_pwd FROM admin
        WHERE admin_name = '{0}' AND admin_pwd = '{1}')
```

יצירת סוג מדידה דינאמית חדש (בפונקציה add_new_measurement_type):

```
INSERT INTO msrtype ({0}) VALUES (%s)
```

בהינתן פרטי העדכון, נבדוק האם העדכון קיים בטבלת העדכונים, ואם כן נחזיר 1 (בפונקציה confirm_user_update):

```
SELECT EXISTS(select 1
                from measurement_update where FKcountry_id = {0}
                 and msr_timestamp = '{1}'
                 and FKmsr_id = {2}
                 and msr_value = {3})
```

קבלת שם סוג המדידה בהינתן הקוד שלה (בפונקציה update_measurements_table):

```
SELECT msr_name FROM msrtype WHERE PKmsr_id = {}
```

בהינתן תאריך, מזהה מדינה, ומזהה מדידה, השאילתא בודקת האם קיימת רשומה שאלו הנתונים שלה, בטבלת המדידות (בפונקציה update_measurements_table):

```
SELECT EXISTS (
        SELECT 1
        FROM measurement
        WHERE msr_timestamp = '{0}' AND FKcountry_id = {1} AND FKmsr_id = {2})
```

קבלת ערך המדידה עבור המדידה עם המאפיינים הנתונים (בפונקציה update_measurements_table):

```
SELECT msr_value
FROM measurement
WHERE FKcountry_id = {0} AND FKmsr_id = {1} AND msr_timestamp = '{2}'
```

עדכון ערך המדידה הקיימת לערך מטבלת העדכונים (בפונקציה update_measurements_table):

```
UPDATE measurement SET msr_value = {0}

WHERE msr_timestamp = '{1}' AND FKcountry_id = {2} AND FKmsr_id = {3}
```

קבלת קוד סוג המדידה עבור סוג מדידה "total_cases" (בפונקציה update_measurements_table):

```
SELECT PKmsr_id FROM msrtype WHERE msr_name = 'total_cases'
```

קבלת קוד סוג המדידה עבור סוג מדידה "total_deaths" (בפונקציה update_measurements_table):

```
SELECT PKmsr_id FROM msrtype WHERE msr_name = 'total_deaths'
```

מציאת התאריך הבא ביחס לתאריך הנתון (בפונקציה update_measurements_table):

```
SELECT DATE_ADD('{}', INTERVAL 1 DAY)
```

עדכון ערך במדידות העוקבות למדידה נתונה [מדינה וסוג מדידה] (בפונקציה update_next_measurements):

```
UPDATE measurement SET msr_value = {0}

WHERE FKcountry_id = {1} AND FKmsr_id = {2} AND msr_timestamp BETWEEN '{3}' AND '{4}'
```

ספירת מספר העדכונים הקיימים (בפונקציה get_updates_for_display):

```
SELECT count(*) FROM measurement_update
```

החזרת העדכונים מטבלת העדכונים (נחזיר מס' מוגדר מראש של רשומות). נשים לב, כי טבלת העדכונים מכילה את מזהה מדינה – ונרצה להחזיר את שם המדינה, וכנ"ל לגבי סוג המדידה – נרצה להחזיר את השם שלה, ולא את המזהה שלה, ולכן יש צורך במכפלה הקרטזית.

(בפונקציה get_updates_for_display):

```
SELECT DISTINCT country.country_name ,measurement_update.msr_timestamp, msrtype.msr_name,
measurement_update.msr_value

FROM measurement_update, country, msrtype

WHERE country.PKcountry_id = measurement_update.FKcountry_id

AND measurement_update.FKmsr_id = msrtype.PKmsr_id

LIMIT {0}
```

קבלת התאריך המאוחר ביותר שבו משתנה מסוים נמדד במדינה מסוימת (בפונקציה check_for_adding_another_measurement):

```
SELECT MAX(msr_timestamp) FROM measurement WHERE FKcountry_id = {0} AND FKmsr_id = {1}
```

קבלת ערך המדידה עבור הנתונים שניתנים (בפונקציה check_for_adding_another_measurement):

```
SELECT msr_value FROM measurement

WHERE FKcountry_id = {0} AND FKmsr_id = {1} AND msr_timestamp = '{2}'
```

בהינתן פרטי העדכון, נבדוק האם העדכון קיים בטבלת העדכונים, ואם כן נחזיר 1 (בפונקציה reject_user_update):

```
SELECT EXISTS(select 1
                from measurement_update where FKcountry_id = {0}
                 and msr_timestamp = '{1}'
                  and FKmsr_id = {2}
                   and msr_value = {3})
```

מחיקת עדכון נתון מטבלת העדכונים (בפונקציה reject_user_update):

```
DELETE FROM measurement_update
WHERE FKcountry_id = {0} AND msr_timestamp = '{1}' AND FKmsr_id = {2} AND msr_value = {3}
```

<u>שאילתות מורכבות המופיעות בפרויקט (בקובץ oop_queries.py)</u>

נרצה לסכום את מספר המאומתים בכל יבשת (סכימת ה-total_cases של כל המדינות המשתייכות לאותה היבשת) בתאריך הכי מאוחר בטווח שבו כל מדינה מדדה את המדידה של מספר המאומתים המצטבר. נשים לב כי לא מובטח שבכל תאריך בהכרח כל המדינות ערכו את כל הבדיקות (ונבחר לכל מדינה תאריך משלה).

נרצה גם לסכום את גודל האוכלוסייה בכל יבשת (סכימת האוכלוסייה של כל המדינות המשתייכות לאותה היבשת) כדי לחשב את האחוזים המתאימים מהאוכלוסייה שאומתו.

כלומר: נסכום לכל יבשת את הערכים המאוחרים ביותר של מדידות במדינות המתאימות ונסכום את האוכלוסייה לכל יבשת (ערך סטטי) ונחלק את התוצאות אלו באלו.

<u>אחוז המאומתים המצטבר בכל יבשת ביחס לגודל האוכלוסיה בתאריך המאוחר ביותר</u> (בפונקציה percentage_cases_out_of_total_population_in_each_continent):

```
SELECT continent_name, SUM(msr_value) AS total_cases, continental_population AS total_population,

100 * SUM(msr_value) / continental_population AS cases_percentage

FROM (WITH last_msr AS (

    SELECT *, ROW_NUMBER () OVER (PARTITION BY FKcountry_id ORDER BY msr_timestamp DESC) rn

    FROM measurement  WHERE FKmsr_id = (SELECT PKmsr_id

    FROM msrtype WHERE msr_name = 'total_cases')

     ) SELECT last_msr.*, country.FKcontinent_id FROM last_msr, country

       WHERE rn = 1 AND PKcountry_id = FKcountry_id) AS m, continent, (

     SELECT PKcontinent_id, SUM(population) AS continental_population

     FROM country, continent

     WHERE continent.PKcontinent_id = country.FKcontinent_id

     GROUP BY PKcontinent_id) AS continent_population

WHERE continent.PKcontinent_id = m.FKcontinent_id

AND continent.PKcontinent_id = continent_population.PKcontinent_id

GROUP BY m.FKcontinent_id

ORDER BY cases_percentage DESC
```

נרצה למצוא את מספר המתים בכל אחת מהמדינות בתאריך הכי מאוחר בטווח שבו כל מדינה מדדה את המדידה של מספר המתים המצטבר. נשים לב כי לא מובטח שבכל תאריך בהכרח כל המדינות ערכו בדיקה זו (ונבחר לכל מדינה תאריך משלה).

נרצה גם למצוא את מספר המאומתים בכל אחת מהמדינות בתאריך הכי מאוחר בטווח שבו כל מדינה מדדה את המדידה של מספר המאומתים המצטבר. נשים לב כי לא מובטח שבכל תאריך בהכרח כל המדינות ערכו בדיקה זו (ונבחר לכל מדינה תאריך משלה).

כלומר: נסכום לכל אחת מ-5 המדינות את הערכים המאוחרים ביותר של מדידות המתים והמאומתים ונחלק את התוצאות אלו באלו.

<u>אחוז המתים מכלל המאומתים בכל מדינה מבין 5 המדינות עם אחוז האוכלוסיה מעל גיל 70 הגבוה ביותר בתאריך המאוחר ביותר</u> (בפונקציה percentage_of_verified_deaths_out_of_total_cases):

```
SELECT DISTINCT country_name, aged.aged_70_older, total_deaths, population,
total_cases, 100*total_deaths/total_cases AS deaths_precentage
FROM (
	SELECT FKcountry_id, aged_70_older, msr_value AS total_deaths
	FROM (WITH last_msr AS (
		SELECT *, ROW_NUMBER () OVER (
			PARTITION BY FKcountry_id ORDER BY msr_timestamp DESC) rn
		FROM measurement
		 WHERE FKmsr_id = (
			SELECT PKmsr_id FROM msrtype WHERE msr_name = 'total_deaths')
	) SELECT last_msr.*, country.aged_70_older FROM last_msr, country
	WHERE rn = 1 AND PKcountry_id = FKcountry_id) AS m
	ORDER BY aged_70_older DESC
	LIMIT 5) AS aged, (
	SELECT FKcountry_id, msr_value AS total_cases
	FROM (WITH last_msr AS (
	SELECT *, ROW_NUMBER () OVER (
		PARTITION BY FKcountry_id ORDER BY msr_timestamp DESC) rn
	FROM measurement
	WHERE FKmsr_id = (SELECT PKmsr_id FROM msrtype WHERE msr_name = 'total_cases')
	) SELECT last_msr.* FROM last_msr WHERE rn = 1) AS m) AS cases, country
WHERE aged.FKcountry_id = cases.FKcountry_id AND aged.FKcountry_id = country.PKcountry_id
ORDER BY deaths_precentage DESC
```

נרצה למצוא את מספר המאומתים בכל יבשת בתאריך הכי מאוחר בטווח שבו כל מדינה ביבשת מדדה את המדידה של מספר המאומתים המצטבר. נשים לב כי לא מובטח שבכל תאריך בהכרח כל המדינות ערכו את כל הבדיקות (ונבחר לכל מדינה תאריך משלה).

נרצה למצוא את מספר המאומתים העולמי הכולל בתאריך הכי מאוחר בטווח שבו כל מדינה מדדה את המדידה של מספר המאומתים המצטבר. נשים לב כי לא מובטח שבכל תאריך בהכרח כל המדינות ערכו את כל הבדיקות (ונבחר לכל מדינה תאריך משלה). לבסוף נסכום את כל הערכים הסופיים לכדי המספר העולמי הכולל.

כלומר: נסכום לכל אחת מהיבשות את הערכים המאוחרים ביותר של מדידות המאומתים ונחלק את התוצאות בערך העולמי הכולל שחושב.

<u>אחוז המאומתים בכל יבשת מכלל המאומתים הגלובלי בתאריך המאוחר ביותר</u> (בפונקציה percentage_of_verified_cases_out_of_all_global_verified_cases_for_each_continent):

```
SELECT cases_per_continent.continent_name, 100*total_cases_continent/global_total_cases AS
percentage

FROM (

        SELECT continent_name, SUM(msr_value) AS total_cases_continent

        FROM (WITH last_msr AS (

                        SELECT *, ROW_NUMBER () OVER (

                                PARTITION BY FKcountry_id ORDER BY msr_timestamp DESC) rn

                        FROM measurement

                        WHERE FKmsr_id = (

                                SELECT PKmsr_id FROM msrtype WHERE msr_name = 'total_cases')

                ) SELECT last_msr.*, country.FKcontinent_id FROM last_msr, country

                WHERE rn = 1 AND PKcountry_id = FKcountry_id) AS m, continent

        WHERE continent.PKcontinent_id = m.FKcontinent_id

        GROUP BY continent_name) AS cases_per_continent, (

        SELECT SUM(msr_value) AS global_total_cases

        FROM (WITH last_msr AS (

                SELECT *, ROW_NUMBER () OVER (

                        PARTITION BY FKcountry_id ORDER BY msr_timestamp DESC) rn

                FROM measurement

                WHERE FKmsr_id = (SELECT PKmsr_id FROM msrtype WHERE msr_name = 'total_cases')

        ) SELECT last_msr.*, country.FKcontinent_id FROM last_msr, country

        WHERE rn = 1 AND PKcountry_id = FKcountry_id) AS m) AS total_continental_cases

ORDER BY percentage DESC
```

בהינתן תאריך, ושם משתנה מבין (total_cases, total_deaths), קבלת הערך המצטבר של המשתנה שנמדד בכל מדינות העולם, בתאריך הכי מאוחר שבטווח: [מהתאריך המוקדם ביותר ב-DB עד התאריך שקיבלנו] שבו כל מדינה מדדה את המדידה של המשתנה הרלוונטי. נשים לב כי לא מובטח שבכל תאריך בהכרח כל המדינות ערכו את כל הבדיקות.

למשל: בהינתן תאריך x ומשתנה total_cases, השאילתא תחזיר את סכום ערכי המדידות של משתנה זה בכל המדינות בעולם, בתאריך הכי מאוחר שבו כל מדינה מדדה אותו (התאריך הכי מאוחר שהכי קרוב לתאריך x).

(בפונקציה get_info_of_variable):

```
SELECT SUM(msr_value)

FROM measurement AS m1 USE INDEX(mapIndex),

        SELECT FKcountry_id AS 'country_id', MAX(msr_timestamp) AS 'max_timestamp'

        FROM measurement USE INDEX(searchIndex)

        WHERE msr_timestamp BETWEEN '{0}' AND '{1}'

        AND FKmsr_id = (SELECT PKmsr_id FROM msrtype WHERE msr_name = '{2}')

        GROUP BY FKcountry_id) AS m2

WHERE m1.msr_timestamp = m2.max_timestamp

AND m1.FKmsr_id = (SELECT PKmsr_id FROM msrtype WHERE msr_name = '{3}')

AND m1.FKcountry_id = m2.country_id
```

שימוש ב-trigger וב-index

טבלת measurement

הוספנו אל הטבלה הזו trigger כדי שברגע העלאת הנתונים לטבלה, במידה ואחד הערכים הוא שלילי, אז ה-trigger יהפוך אותו אל 0. כמו כן, הוספנו שני אינדקסים (בעת יצירת הטבלה) כדי לייעל את החיפוש:

```
def add_indices_to_measurements_table(my_cursor, my_connection):
    my_cursor.execute("create index searchIndex on measurement(msr_timestamp, FKmsr_id);")
    my_cursor.execute("""create index mapIndex on measurement(msr_timestamp, FKmsr_id, FKcountry_id);""")
    my_connection.commit()
```

המטרה של שני האינדקסים היא לייעל את זמן השאילתה שנמצאת בפונקציה get_info_of_variable בקובץ oop_queries.py. מטרת השאילתה היא שבהינתן שני תאריכים, ומשתנה, נוכל למצוא לכל מדינה את ערך המדידה של המשתנה שקיבלנו, שנמדד בתאריך הכי מאוחר (בטווח שבין שני התאריכים שקיבלנו) שבו כל מדינה מדדה (לא מובטח שהמדידה האחרונה של המשתנה שקיבלנו בהכרח תהיה באותה התאריך לכל המדינות).
מקטע מהשאילתא הנ"ל:

```
SELECT SUM(msr_value)
FROM measurement AS m1 USE INDEX(mapIndex),
        SELECT FKcountry_id AS 'country_id', MAX(msr_timestamp) AS 'max_timestamp'
        FROM measurement USE INDEX(searchIndex)
        WHERE msr_timestamp BETWEEN '{0}' AND '{1}'
        AND FKmsr_id = (SELECT PKmsr_id FROM msrtype WHERE msr_name = '{2}')
        GROUP BY FKcountry_id) AS m2
WHERE m1.msr_timestamp = m2.max_timestamp
AND m1.FKmsr_id = (SELECT PKmsr_id FROM msrtype WHERE msr_name = '{3}')
AND m1.FKcountry_id = m2.country_id
```

מטרת searchIndex - שיפור זמן ריצת חיפוש הרשומות לפי msr_timestamp ואז לפי FKmsr_id. לעומת זאת בשאילתה החיצונית, נשתמש ב-mapIndex לחיפוש לפי FKmsr_id, msr_timestamp ולפי FKcountry_id.

טבלת country

הוספנו אל הטבלה הזו trigger כדי שברגע העלאת הנתונים לטבלה, במידה ואחד הערכים הוא NULL, אז ה-trigger יהפוך אותו אל 0.

מבנה התיקיות של הפרויקט

תיקייה – db_creation: מכילה את כל קבצי ה data וכל קבצי ה-python הנדרשים לצורך יצירת ה DB.

- קובץ ה-data המקורי - covid_data_set_OWID.csv
- קבצי ה-data שחולצו מקובץ ה-data המקורי - continents.csv, countries.csv
  continent_to_country.csv, measurements.csv, admins.csv, msrType.csv
- קבצי קוד שנועדו לצורך יצירת ה DB:
  - csvSplit.csv – מסנן את הקובץ covid_data_set_OWID.csv אל קבצי ה-csv המוצגים לעיל.
  - connection.py – בהינתן קבצי ה-csv, הוא יוצר את ה-DB, ואת הטבלאות עצמן. כמו כן, הוא מכניס אליהן נתונים ויוצר את האינדקסים והטריגרים, ולבסוף מעלה הכל לשרת.
  - פעולות מרכזיות בקובץ connection.py :

```
def create_table(my_cursor, db_table_name, query):

def insert_into_table(csv_name, table_name, my_cursor, my_connection):
```

תיקיות של קבצי הקוד של ה-UI:

- assets
- css
- js

קבצי UI נוספים שנמצאים ב root:

- index.html
- package-lock.json
- package.json

קבצי שאילתות – נמצא ב root:

- oop_queries.py – מחלקה המכילה את כל השאילתות ש-UI צריך בכדי לעבוד.
  - פעולות מרכזיות הקשורות לעדכונים:
    user_update
    check_admin
    add_new_measurement_type
    confirm_user_update
    reject_user_update
  - פעולות מרכזיות הקשורות לנתונים שיוצגו ב UI:
    השאילתות שהוצגו בעמודים הקודמים בתור – "שאילתות מורכבות המופיעות בפרויקט"

קובץ תלויות - נמצא ב root

dependencies.bat – מוריד את כל הספריות הנדרשות עבור הפרויקט.

יש להריץ אותו לפני יצירת ה DB, ולפני ריצת השרת בכדי לוודא שכל הספריות מותקנות.

קובץ השרת - נמצא ב root

server.py – הקובץ שמריצים בכדי להפעיל את האתר, המכיל מופע של oop_queries.