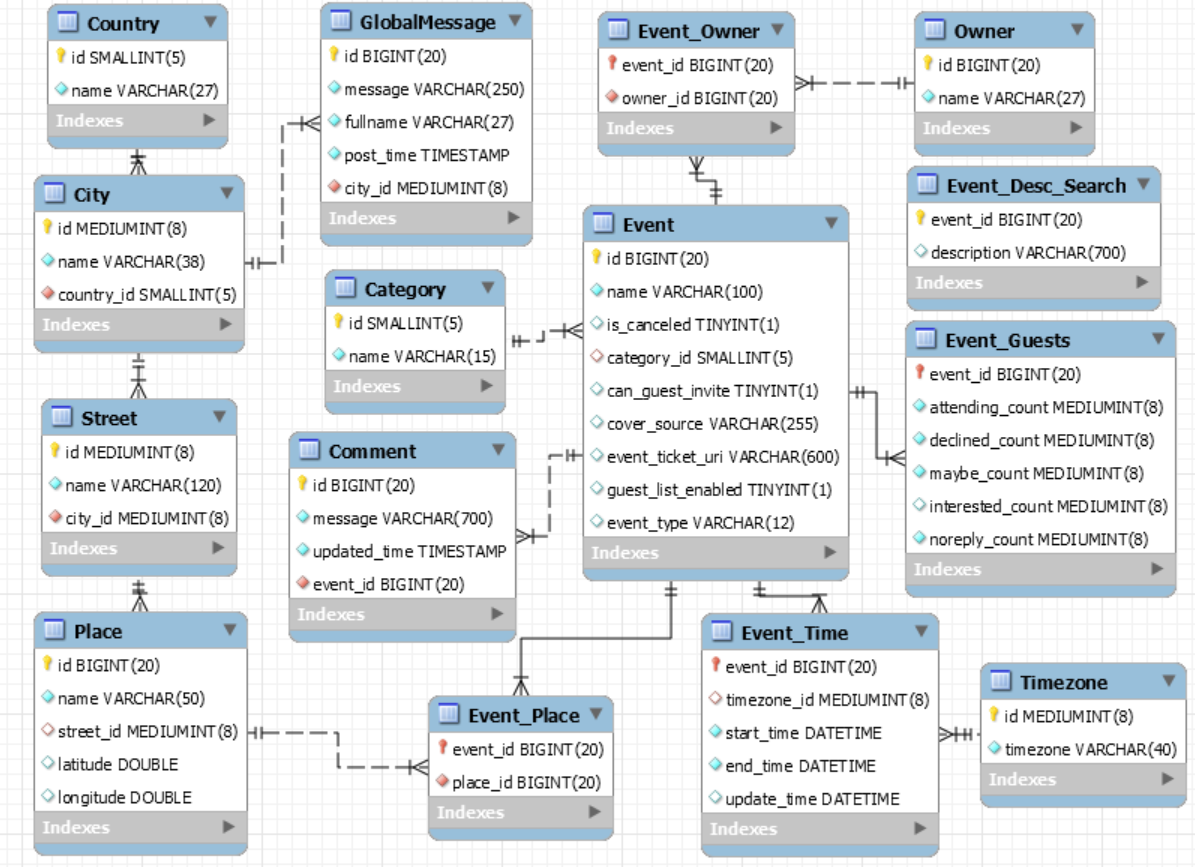
**Village Mouse Documentation**

## **Software Documentation**

## DB scheme structure



**The Tables:**

1. Event – a table that holds the basic information about an event, like its name and type. This is the central entity we base our database around, so to make the database more efficient we aimed to keep this table as small as possible

Primary key: id.

Foreign Keys: category\_id to "Category" table.

1. Category – a table that holds event's possible categories.

Primary key: id.

1. Comment – a table that holds comments of events

Primary key: id.

Foreign Keys: event\_id to "Event" table.

1. Owner – a table that holds information about owners of events

Primary key: id.

1. Event\_Owner – an associative table that connects between Event and it's Owner.

Primary key: event\_id.

Foreign Keys: event\_id to "Event" Table

owner\_id to "Owner" table

1. Place – a table that holds information about places of events, like it's name and it's latitude and longitude.

Primary key: id.

Foreign Keys: street\_id to "Street" Table

1. Street – a table that hold information about streets.

Primary key: id.

Foreign Keys: city\_id to "City" table.

1. City – a table that hold information about cities.

Primary key: id.

Foreign Keys: country\_id to "Country" table.

1. Country – a table that hold information about countries.

Primary key: id.

1. Event\_Place – an associative table that connects between Event and it's Place.

Primary key: event\_id.

Foreign Keys: event\_id to "Event" Table

Place\_id to "Place" table

1. Timezone – a table that hold information about time zones in the world .

Primary key: id.

1. Event\_Time – an associative table that connects between Event and it's times information.

Primary key: event\_id.

Foreign Keys: event\_id to "Event" Table

timezone\_id to "Timezone" table

1. Event\_Guests – a table that holds information about event's guests, like it's attending\_count.

Primary key: event\_id.

Foreign Keys: event\_id to "Event" Table

1. Event\_Desc\_Search – a table that holds descriptions of events and used in order to execute a Full Text Search Query.

Primary key: event\_id.

1. GlobalMessage – a table that hold messages of users of our application, like the name of the user and the message itself.

Primary key: id.

Foreign Keys: city\_id to "City" Table.

## DB Optimization

We used some indexes for our database. We created an index for every field that take a part of a search over the database in an execution of one of our queries.

Our indexed are:

1. Index for 'country\_id' field on City table.
2. Index for 'name' field on City table.
3. Index for 'city\_id' field on Street table.
4. Index for 'street\_id' field on Place table.
5. Index for 'category\_id' field on Event table.
6. Full Text Index for 'description' field on Event\_Desc\_Search table.
7. Index for 'attending\_count' field on Event\_Guests.
8. Index for 'start\_time' field on Event\_Time table.
9. Index for 'event\_id' field on Comment table.
10. Index for 'place\_id' field on Event\_Place table.
11. Index for 'owner\_id' field on Event\_Owner table.
12. Index for 'post\_time' field on GlobalMessage table.

And also the indexes for our table's primary keys that were created automatically.

We also know that in "InnoDB" Engine indexes for foreign keys are being created automatically but still we want to add a description of those indexes to our project documentation.

For our table's storage engines, we used "InnoDB" for all our table except Event\_Desc\_Search table that should support Full Text Search Index and Query, so for that table we used "MyISAM" storage engines.

More than that, we split some tables into parts in order to keep small tables instead of one big table. For example, in our original database, the "Event" table contained fields about the event's owner, event's place, event's description, event's times and event's counts of guests. This table was, from our point of view, too big and so we split it into "Event" Table and 5 more sub tables with foreign keys and indexes of their own.

## DB Complex Queries

|  |  |  |
| --- | --- | --- |
| **תיאור השאילתא** | **Query Description** | **Output** |
| האירוע הכי חדש עבור 8 הקטגוריות בהן האירועים הכי חדשים (חדשים מבחינת זמן התחלה עתידי) | Mosaic Query : The newest event for the 8 categories that have the newest events (newest in start\_time). | Event\_name  Event\_cover  Event\_description |
| "העיר החמה" : מחזיר את 10 האירועים עם התאריך הקרוב ביותר ליום הנוכחי בעיר שבה יש הכי הרבה אירועים | Hottest City – 10 events with the closest start date to the current date (in the future) in the city that has the highest number of events. | City\_name  Event\_name  Event\_description |
| "העונה החמה בעיר עם האירוע הכי גדול" : מחזיר את האירועים עם הכי הרבה משתתפים מהעיר שבה יש את האירוע עם הכי הרבה משתתפים, ובעיר יש לפחות 10 אירועים בסה"כ. האירועים שיחזרו חייבים להתקיים באותה עונה כמו האירוע בעל המס' הגדול ביותר של המשתתפים: בטווח של חודשיים לפני או אחרי האירוע (אך יתכן שהאירועים מתקיימים בשנים שונות). | All events in the hottest season in the city hosting the biggest event:  Returns all events with the most attendants in the city that hosts the event with the largest amount of attending people, and the city hosts at least 10 events total. The events returned must also occur within the same season as the event with the maximal amount of attendees )meaning 2 months before or after the maximal event, can occur on different years) | Event\_name  City\_name  Event\_description  Event\_start\_time  Event\_end\_time  Event\_attending\_count |
| הקטגוריות של עשרת האירועים בעלי מספר התגובות הגבוה ביותר | The categories of the 10 events with the highest number of comments | Category\_name |
| הבעלים הכי פופולארי – עשרת הבעלים שלאירועים שלהם יש בסך הכל את מספר התגובות הכי גבוה מבין עשרים הבעלים שלאירועים שלהם יש בסך הכל את מספר מאשרי ההגעה הכי גבוה. | Most Popular Owners – 10 owners that their events has the highest number of comments together among the 20 owners that their events has the highest number of attendings together. | Owner\_name  Owner\_event\_attendings Number\_of\_comments |
| עשרת המקומות שבהם מתקיים המספר הגדול ביותר של אירועים מהקטגוריה שמספר מאשרי ההגעה לאירועים שלה הוא בסך הכל הגדול ביותר | 10 places that got the highest number of events from the category with the highest number of attendings. | Place\_name  Country  City  Street  Category\_with\_highest\_attendings  Number\_of\_events\_from\_category |
| **Full text search query**  הבעלים הכי רגישים – עשרת הבעלים שהם בעלים של המספר הגדול ביותר של אירועים שבתיאור שלהם מופיעה המילה "Love." | Full Text Search Query – Most Sentimental Owners – 10 owners that got the highest number of events that has the word "Love" in their description. | Owner\_name  Number Of Events with the word 'Love' |
| מבין כל הרחובות שבארה"ב ובבריטניה, מי אלה חמש הרחובות שבהם מתקיימים הכי הרבה אירועים. | Among all the streets in USA and UK, who are the 5 streets that got the highest number of events. | Street.name  City.name  Country.name  Number Of Events |

## Special Queries

1. COMPLEX Location query – we used the latitudes and longitude of place of event in order to find the nearest events and present them to the user with Google Maps.
2. INSERT Query – Add new comment to a specific event. The comment will be inserted to "Comment" table with an appropriate event\_id.
3. INSERT Query – Send a "Message To The World" through our website. The message will be inserted to "GlobalMessage" table.
4. UPDATE Query – Using the API of facebook in order to update some of the guests counters of one specific event. The counters that will be updated are "interested\_count" and "noreply\_count". We chose not to update the "attendings\_count" field because there is an index on this field and the update process would be slow because of the needed update of the index.

We also decided not to update "declined\_count" and "maybe\_count" because those fields of event can be read correctly only by using Facebook API User Key because those fields are fields of private Facebook events. If we will update them, we will set both to 0 for any event because we only use Facebook API APP key in our application. And so, in order to not lose data we decided not to update those fields.

1. Input Query – Search event by it's id, when the user chooses an event, the query will be executed and returns event's details.
2. Input Query – Search events by input of a word in their description.
3. Input Query – Count Event By City – returns the number of event that occurs in the city input.

## Query Optimization

As mentioned before, we created an index for every field that take a part in a search process in at least one of our queries. Those index helped us to optimize the computation time of our queries because they store those fields in a data structure that support efficient search.

We also mentioned that we split our tables, and so, because of the complexity of our queries we needed to execute quite of few join operations between tables, and because our new tables were smaller we were able to do those joins operations almost every time between smaller tables and so, according to our understanding we were managed to improve our database performance in the meaning of computation time of our queries.

## Code Structure

Pre-Production Actions

1. We created the DB scheme and all required fields and keys. All data fields were set to encoding – utf8. ;
2. Run locally the script: **fb\_extract.py**

**The script was run once, and download all the required data from FacebookAPI**.

* MySQLdb code structure

In this project we used MySqlDB, in order to connect to DB and execute queries via Python, we have used, *‘import MySQLdb’,* python library.

Connect to DB:

*Con = MySQLdb.connect (database\_hostname, 'DbMysql08', 'DbMysql08', 'DbMysql08', charset='utf8')*

We used ‘utf-8’ coding in order to support characters in many languages.

The next stage is to create cursor for executing queries.

*Cur = con.cursor (MySQLdb.cursors.DictCursor)*

In order to prevent from auto commit, in order to be able handle transaction:

|  |
| --- |
|  |
|  |
|  |

*Con.autocommit (false)*

The queries was executed in the following way:

*Cur.execute([Query], [Parameters])*

The parameters wasn’t empty in insert or update. In order to commit the changes we used Con.Commit() or in case of failure Con.rollback(). We tried to keep the data base in consistent and stable state. Any DB exception was handled and printed to log message without stack the program.

## Description of API

* Facebook API Code

In order to communicate with facebook data base we used Facebook Graph API via python. The main python lib the implement the connection is ‘import facebook’.

API Usages

1. Create graph API object:

*graph = facebook.GraphAPI(access\_token=app\_token, version='2.2')*

1. Get events by keyword:

*graph.request('search',{'access\_token': user\_token,'q': key,'type': 'event'})['data']*

1. Get Event details by EventID:

graph.get\_objects(ids=[eventID], fields= [Required Fields])

1. Get All posts on specific event by event id:

graph.get\_connections(id=event\_id, connection\_name='feed')

All the data from facebook API in JSON format. In the next section JSON parsing will be explained. The data was reorganized in order to match to our DB scheme.

Facebook API Error Handling

We ignored facebook.GraphAPIError, in order to create stable facebook API usage without crashing due to third party exceptions. In addition, there are some facebook API restrictions for the using of API. If we exceed maximum retry count ConnectionError will be thrown. The code catch the exception, holds for 60 seconds and retry again. Any other exception is caught and handled in the following way: Log description is printed, and the program continues, without crashing.

**Script flow:**

1. Open Keyword.txt file and get all keywords. After it get Events ID’s from Facebook DB using keywords.
2. For each Event ID get event details from Facebook DB in JSON format.
3. Save all events details to data.json file, using json.dump.
4. For each Event ID get all posts to this event as JSON format.
5. Save posts in (Event\_ID)\_comments.json file

## External packages/libraries

1. Facebook – a python library for the usage of Facebook API.
2. MySQLdb – for the work in from of our database from our web.
3. Flask – for the programming of our web server.
4. Angular – in order to connect between our website's GUI and the data that returns from our database.
5. Bootstrap – for the display of our website.