

# INTELLIGENT CHATBOT

**TEAM MEMBERS:** Anyesha ghosh, Namrata Gupta, Reema Kumari, Swarnima and Kumari Rashmi Bala

**MENTORS:** Anurag Dwivedi and Shubham Gupta

**PROJECT AIM:** To make a computer program designed to simulate an intelligent conversation with one or more human users.

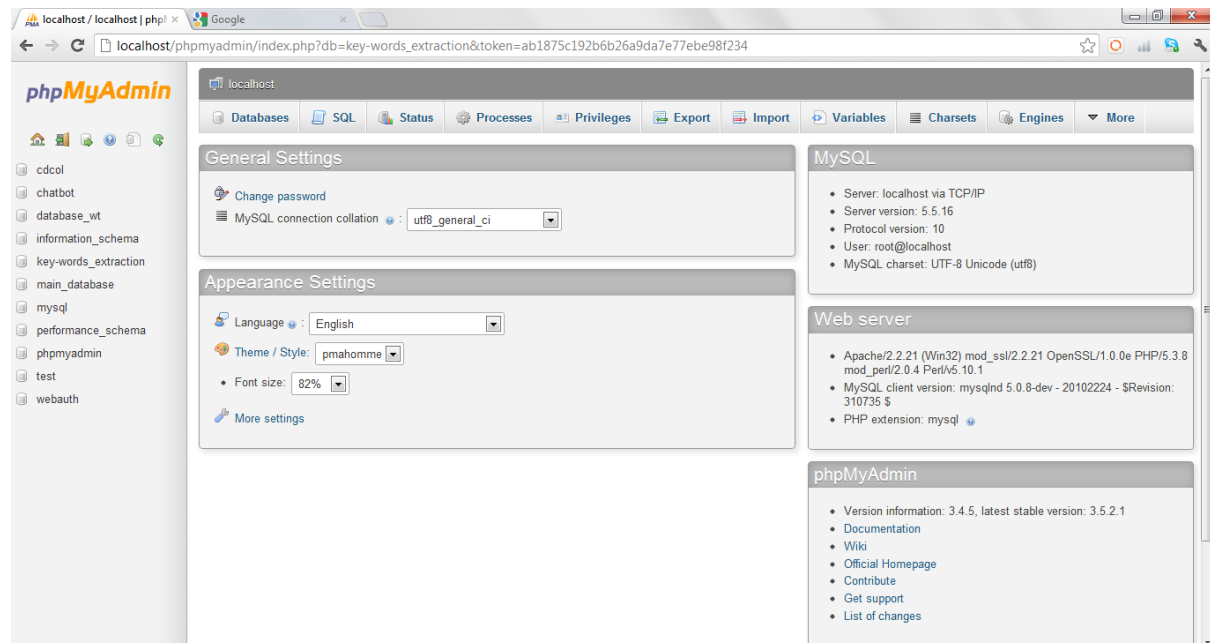
**MOTIVATION FOR THE IDEA:** We sometimes pass our time by chatting with different chatterboxes available on internet, so to make one of them was indeed an interesting idea. We also had thought of integrating it with speech-to-text and text-to-speech softwares and equipping it with face recognition technique. Here is one link of a video on You Tube, which motivated us a lot :

[http://www.youtube.com/watch?v=JF\\_HXTQ7Quo](http://www.youtube.com/watch?v=JF_HXTQ7Quo)

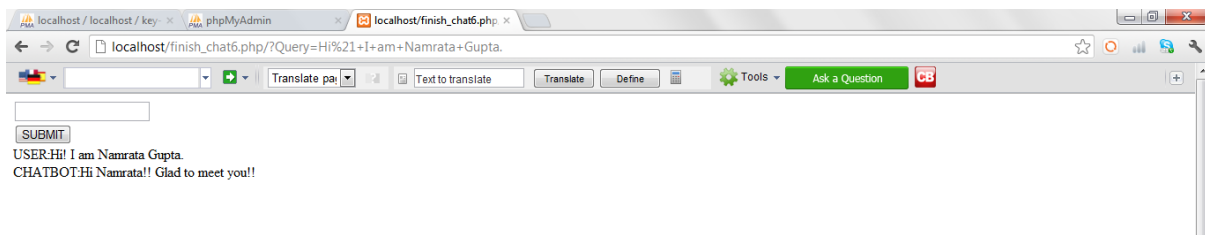
**THINGS WE USED:** We used MySQL to build our database and used PHP and SQL programming languages to access this database. Most of the part of the code is written in PHP, only the commands used to interact with the database are in SQL. We installed XAMPP server, which is a free and open source cross-platform web server solution stalk package, consisting mainly of the Apache HTTP server, MySQL database and interpreters for scripts written in PHP and PERL programming languages.

**ABOUT MYSQL:** It is an open source relational database management system (RDBMS) that runs as a server providing multi-user access to a number of databases. MySQL is a popular choice of database for use in web applications, and is a central component of the widely used LAMP open source web application software stack—LAMP is an acronym for “Linux, Apache, MySQL, Perl/PHP/Python”. We interact with our MySQL databases or the Xampp server through phpMyAdmin. It is a free software tool written in [PHP](#), intended to handle the administration of [MySQL](#) over the World Wide Web. It supports a wide range of operations with MySQL. The most frequently used operations are supported by the user interface (managing databases, tables, fields, relations, indexes, users, permissions, etc), while you still have the ability to directly execute any SQL statement. phpMyAdmin is accessed from localhost. To access it, just open a browser and type <http://localhost/phpmyadmin/>.

Here is a snapshot of how does it look like when opened (after providing our password for it): Here the words ‘cdcol’, ‘chatbot’, ‘database\_wt’ etc. (written in the left) are the names of the databases currently present in the server. By clicking any of those we can see and manipulate the tables present inside it.



**THEORY:** We basically have a database containing different tables. These different tables store key-words, answer-sentences, mutual weight of each key-word corresponding to each answer-sentence and the lists of words to be filtered out. Then we have codes to search through these tables and come up with the most appropriate reply corresponding to a particular user’s query. E.g.



It has got a feedback system also which after knowing that a query is a positive or a negative feedback, increases or decreases accordingly the mutual weights of the key-words of the previous user's query corresponding to the previously presented answer. Moreover, we don't have to fill our database manually. We can easily make our bot learn from the chats provided to it. A code has been written for that purpose.

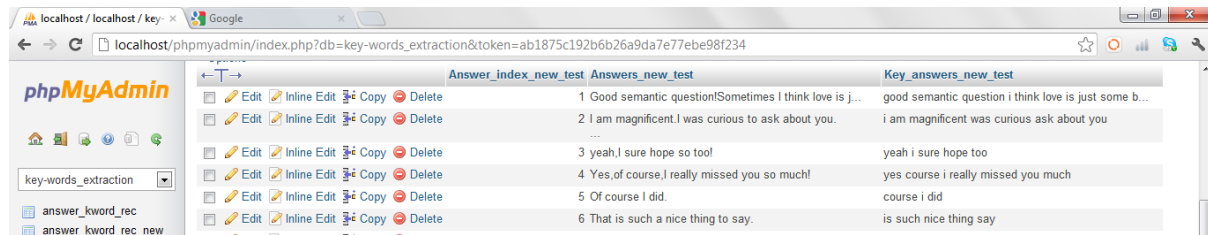
**ARRANGEMENT AND TYPES OF TABLES IN THE DATABASE:** This database is a collection of tables. It has three main tables. Here the things written in the left side are the names of the table contained in the database named 'key-words\_extraction'. The last three tables as shown in the snapshot, named 'test\_table1', 'test\_table\_answer' and 'test\_table\_main' are playing the key roles here.

Table	Action	Rows	Type	Collation	Size	Overhead
answer_kword_rec	Browse Structure Search Insert Empty Drop	12	InnoDB	latin1_swedish_ci	16.0 K B	-
answer_kword_rec_new	Browse Structure Search Insert Empty Drop	99	InnoDB	latin1_swedish_ci	16.0 K B	-
auxiliary_verbs	Browse Structure Search Insert Empty Drop	2	InnoDB	latin1_swedish_ci	16.0 K B	-
conjunctions	Browse Structure Search Insert Empty Drop	47	InnoDB	latin1_swedish_ci	16.0 K B	-
feedback	Browse Structure Search Insert Empty Drop	2	InnoDB	latin1_swedish_ci	16.0 K B	-
feedback_neg	Browse Structure Search Insert Empty Drop	3	InnoDB	latin1_swedish_ci	16.0 K B	-
interjections	Browse Structure Search Insert Empty Drop	54	InnoDB	latin1_swedish_ci	16.0 K B	-
main_table	Browse Structure Search Insert Empty Drop	516	InnoDB	latin1_swedish_ci	48.0 K B	-
main_table_new	Browse Structure Search Insert Empty Drop	13,812	InnoDB	latin1_swedish_ci	1.3 M B	-
prepositions	Browse Structure Search Insert Empty Drop	69	InnoDB	latin1_swedish_ci	16.0 K B	-
pronouns	Browse Structure Search Insert Empty Drop	76	InnoDB	latin1_swedish_ci	16.0 K B	-
query_kword_rec	Browse Structure Search Insert Empty Drop	43	InnoDB	latin1_swedish_ci	16.0 K B	-
query_kword_rec_new	Browse Structure Search Insert Empty Drop	212	InnoDB	latin1_swedish_ci	16.0 K B	-
side_table1	Browse Structure Search Insert Empty Drop	13	InnoDB	latin1_swedish_ci	16.0 K B	-
side_table2	Browse Structure Search Insert Empty Drop	4	InnoDB	latin1_swedish_ci	16.0 K B	-
starting_words	Browse Structure Search Insert Empty Drop	190	InnoDB	latin1_swedish_ci	16.0 K B	-
test_table1	Browse Structure Search Insert Empty Drop	91	InnoDB	latin1_swedish_ci	16.0 K B	-
test_table_answer	Browse Structure Search Insert Empty Drop	31	InnoDB	latin1_swedish_ci	16.0 K B	-
test_table_main	Browse Structure Search Insert Empty Drop	16,588	InnoDB	latin1_swedish_ci	1.5 M B	-
19 tables	Sum	31,864	InnoDB	latin1_swedish_ci	3.2 M B	0 B

The first one has two columns and it stores all key-words (available in the database at a particular time) with indices assigned to them as shown in the following snapshot. The snapshot is showing the words only up to the index 19 but a table contains several pages like this.

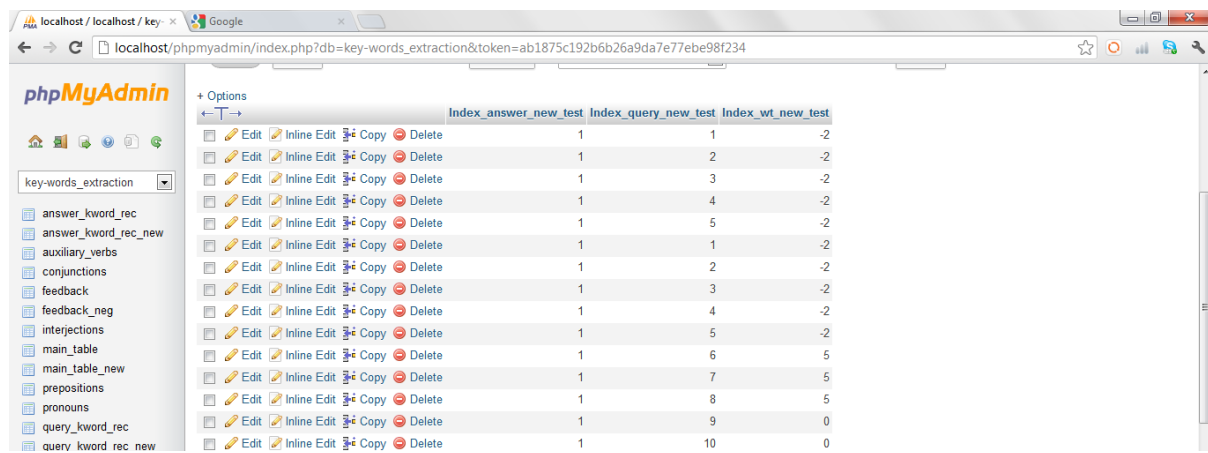
Index_kword_new_test	Complete_kword_new_test
1	you
2	are
3	not
4	intelligent
5	chatbot
6	what
7	is
8	love
9	how
10	i
11	am
12	also
13	fine
14	hope
16	rain
17	today
17	do
18	want
19	talk

The second table has three columns and it contains all answer-sentences, their indices and their key-words in these three columns as shown in the following snapshot:



Answer_index_new_test	Answers_new_test	Key_answers_new_test
1	Good semantic question! Sometimes I think love is j...	good semantic question i think love is just some b...
2	I am magnificent. I was curious to ask about you.	i am magnificent was curious ask about you
3	yeah, I sure hope so too!	yeah i sure hope too
4	Yes, of course, I really missed you so much!	yes course i really missed you much
5	Of course I did.	course i did
6	That is such a nice thing to say.	is such nice thing say

Third table is most important. It has also got three columns. In each row, first column contains indices of answers, second contains indices of key-words and the third contains their mutual weight (means mutual weight of that particular answer with respect to that key-word) as shown in the following snapshot:



Index_answer_new_test	Index_query_new_test	Index_wt_new_test
1	1	-2
1	2	-2
1	3	-2
1	4	-2
1	5	-2
1	1	-2
1	2	-2
1	3	-2
1	4	-2
1	5	-2
1	6	5
1	7	5
1	8	5
1	9	0
1	10	0

Some other tables contain the words which need to be eliminated from a user's query (filtering). There is one table which contains all possible one word queries e.g. 'hi', 'hello', 'thanks' etc. and their responses, because if user puts any of those things, we want our bot to answer directly from this table rather than pulling a reply using any other tedious method. Two of these tables are for taking feedback from the user. Basically, they are for recognition of the fact that user is giving a feedback and after that to differentiate between a positive feedback and a negative one. Basically, these tables contain set of key-words which might be present in a feedback query, so they contain all (almost) such possible sets both for positive and negative feedbacks.

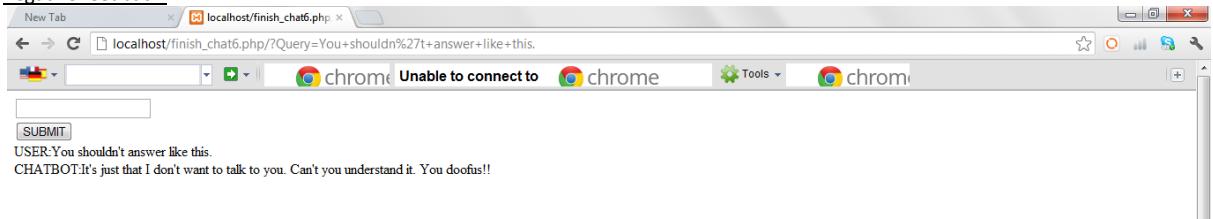
**FINAL ALGORITHM:** Finally we have a database and two codes used for:

1. To get a reply from the database and keeping the mutual weights correct and logical using feedback system.  
Here is the link for the google doc containing this code:  
<https://docs.google.com/document/d/1mZWitOBkqWXCv24dXLI7ZlvcKFY8OyhxiB8pdVabkkM/edit>
2. To increase the size of database, means inserting new key-words and answer-sentences by providing a series of chat, in a text document, to this code.  
Here is the link for the google doc containing this code:  
[https://docs.google.com/document/d/1DsY\\_vev9t\\_KARqBffzHKZ1ciZq6y8s8et7-R7Suw2fg/edit](https://docs.google.com/document/d/1DsY_vev9t_KARqBffzHKZ1ciZq6y8s8et7-R7Suw2fg/edit)

For the first code (which is for the case when someone is chatting with the bot and providing feedback if he/she wants) the steps for what it does, are written below:

1. It stores the user's query in an array(\$array1) word by word, does the required filtering of prepositions, conjunctions, auxiliary verbs, interjections etc. and stores rest of them in an another array(\$array2).
2. It finds out all the proper nouns and stores them in an array \$proper\_nouns.
3. It filters out \$proper\_nouns from \$array2 and stores the rest of them in \$array3, so now every word stored in \$array3 is a key-word.
4. It goes to the table containing key-words, find out the matching ones (with those stored in \$array3) and stores their indices in \$array\_in.
5. It then stores the rest of the words (the new ones which need to be inserted) in \$array4.
6. If number of key-words is only one, it goes to the table storing all possible one word queries (as described above), tries to find a match. In case it matches with any of them, it gives the corresponding reply (stored in that table). If it doesn't find a match, it puts a reply sort of "well, I am not getting your point. Probably, you can tell me more." In this way, it basically provokes the user to come up with a query with more number of matching key-words.

7. If number of key-words is greater than one, it goes to the feedback tables. There it tries to find a match for the key-words. If it finds a match, it displays the answer stored there as shown in the following example. This example is demonstrating the case of negative feedback.



8. Then it goes to the table containing mutual weights and decreases or increases (depending upon type of the feedback- positive or negative) the weights of key-words of the previous query corresponding to the answer displayed previously, by some decided amount.
9. If it doesn't find a match in the feedback tables, then also it puts an answer to provoke the user to come up with a query with more number of matching key-words.
10. If number of key-words is greater than one and the number of matched key-words is also greater than one, it goes to the table containing the mutual weights of the available key-words and the answer-sentences. It sums up the mutual weights all the key-words corresponding to each answer and then displays the answer with the highest summation. It then write this answer and the key-words of this query into a text file for future use means if user's next query is some kind of feedback then it reads this text file and corrects their mutual weights.
11. Now it inserts the new key-words (which we have earlier stored in \$array4) into all required tables and gives a mutual weight of zero to them corresponding to all answer-sentences except the displayed one. It assigns a non-zero mutual weight to them corresponding to the displayed answer.

For the second code (which is for the case when the bot is learning from a series of chat provided to it) the steps for what it does, are written below:

It basically processes the conversation written in the text document sentence by sentence.

1. It examines the first word of a particular sentence. If it is 'User', it goes through a code written for the processing of a query-sentence and if the first word is 'Bot', it goes through a different code written for the processing of an answer-sentence.
2. If it's a user's query then it goes through the same filtering process and find out the key-words. Then it finds out the matching ones and thus the new ones, which need to be inserted. It inserts these new key-words in required tables at required places giving a mutual weight of zero to them corresponding to each answer available in the database till that point.
3. Then it finds the answer in the next sentence. It applies filtering on this sentence also and finds out its key-words, finds out the new key-words and inserts them in the required tables giving a mutual weight of zero corresponding to each answer available in the database. Then it finds out if key-words of this answer are matching with any of those of the previously stored answers. If it finds a match, it increases the mutual weights of the key-words of the query sentence with respect to this matched answer.
4. If it doesn't find a match, it inserts this new answer with its key-words in the required tables and assigns a non-zero mutual weight to the key-words of the query sentence with respect to this answer and gives zero mutual weight to all other key-words corresponding to this new answer.

**TASKS COMPLETED:** Everything in the theory and algorithm explained above, has been completed.

**TASKS LEFT:** We couldn't integrate our chatbot with text-to-speech and speech-to-text softwares. Also, we couldn't give it the technique of face-recognition. We would like to complete these tasks.

**FUTURE PROSPECT:** After having a lot of chatting with different users and receiving feedbacks, it will be able to answer quite reasonably. Also its database will continue to grow in size automatically. Equipping it with face-recognition techniques will give it the ability of recognising its usual users. Moreover, integration of speech-recognition software will give relief to a user from the pain of typing. It can also be integrated into dialogue systems for various practical purposes such as personalized service or information-acquisition. So, it can also be useful apart from being entertaining.

## REFERENCES:

To download and install XAMPP server: [http://download.cnet.com/XAMPP/3000-10248\\_4-10703782.html](http://download.cnet.com/XAMPP/3000-10248_4-10703782.html)

To learn about PHP and MySQL: <http://www.homeandlearn.co.uk/php/php.html>

To solve any problem related to PHP: <http://w3schools.com/>

<http://php.net/>

**ACKNOWLEDGEMENT:** We are thankful to E-Club for providing us with this opportunity of thinking and implementing our idea. We would also like to thank our mentors Anurag Dwivedi and Shubham Gupta for guiding and motivating us. We really enjoyed doing this project and it was a very useful experience.