Π18139 ΦΑΙΗ ΣΙΟΣΙΟ
Π20050 ΧΡΥΣΟΥΛΑ ΓΟΥΜΕΝΑΚΗ
Π20069 ΑΓΓΕΛΙΚΗ ΚΑΛΔΙΡΗ

#1 Enrich WHERE statement

For the support of NOT operator we imported the not function from the operator python package we imported it in the misc.py file and used it in the get_op and split_condition functions.The operator not is used in a case sensitive manner,so it makes no difference if it is lower case or upper case.
Example usage: **select * from student where name not zhang**

For the support of the BETWEEN operator we used a similar approach.First we declared the function between because the operator package has no such function.This function checks either if a number is within given range,or a string is within a range.In order for this function to work,we assume all inputs are strings,this modification is done on the _parse_condition function of the Table class.The range is given with the & character and with the lower and upper limit connected with it.The between function performs the stripping process and checks if the value is within range.
Example usage: **select * from student where tot_cred between 50&100**

For the support of the AND operator we altered the way the select command works.If there is no and operator it executes the query plan as before.In the case there is a and word,it changes the way query plans are executed.It execute different queries for each condition.For example
select * from table_1 where cond_1 and cond_2 becomes
select * from table_1 where cond_1 and **select * from table_1 where cond_2**.Then there is code written in the mdb.py file which combines the results produced by the previous operations.The combination is performed on the primary key column and it has to exist on both tables that the query returns.The show function of the table class has been altered in order to be able to return the header and to be printed table so that we can combine them in the mdb.py file.tabulate is still being used to print the output.
Example usage: **select * from student where dept_name=physics and tot_cred >20**

For the support of the OR operator we have used the same approach as in the AND operator,but this time the tables being returned are appended to each other excluding duplicate values.
Example usage: **select * from student where dept_name=physics or tot_cred >20**

Π18139 ΦΑΙΗ ΣΙΟΣΙΟ
Π20050 ΧΡΥΣΟΥΛΑ ΓΟΥΜΕΝΑΚΗ
Π20069 ΑΓΓΕΛΙΚΗ ΚΑΛΔΙΡΗ

#2 Enrich indexing functionality

In order to perform the Btree indexing over unique key we must first introduce the unique constraint to miniDB.In order to include the UNIQUE constraint we have modified the mdb file,but also the Database.py and Table.py.On the mdb.py we have modified the create_query_plan function in which we added the unique constraint,and when this word is found when reading the create table action,the system assigns the unique value of the dictionary to the last column in a manner similar to that of the primary key constraint.Then we had to change the arguments of the create _table function of the Database.py to include the unique key and also the constructor of the Table class. In a similar manner to the way the primary key is shown in the select command a **#U#** is being printed in the column of the table that has the constraint.The way we programmed this functionality allows only for 1 unique constrained column.
Example: **create table u_table (a int primary key,b str unique)**

During this stage of the development it occurred that when selecting a specific column it showed #PK# above it.Therefore in the Table.show() function we added a clause to the if,so that it shows either the #PK# or #U# identifier only if the the column exists and it has the same name.Also we perform a check for tables that have not the attribute unique by using the built in hasattr method.

For the Btree operation we perform the same actions as with the PK but this time we check if the column is a Unique column.In that case it has an index because practicaly a unique key works as a primary key,so we just call the same functions in the same order with the code written for the pk btree index search,just for a differently constrained column.In order for this to occur we have to perform a search on the column that has the Unique constraint.

For the Hash table search we have modified the Table.py file and the Database.py file.It performs a search based on a hash table only on the primary key column.If there is an equality check on the PK column the database object calls the table function _select_hash that creates a hash table with a specific number of bins(5 by default) and then it searches into the hash table based on the modulo function % and return the correct row.Due to some problems with the formatting this function does not follow suit with the others on the printing part,because it does not use tabulate.
Example usage **select * from test_table where a = 3**

For the testing,on the first issue we used the small relations database provided in the project,and for the second issue we created our own database,that you can connect to by pressing **cdb test**