Using a Database in Your Research.

2023-06-19

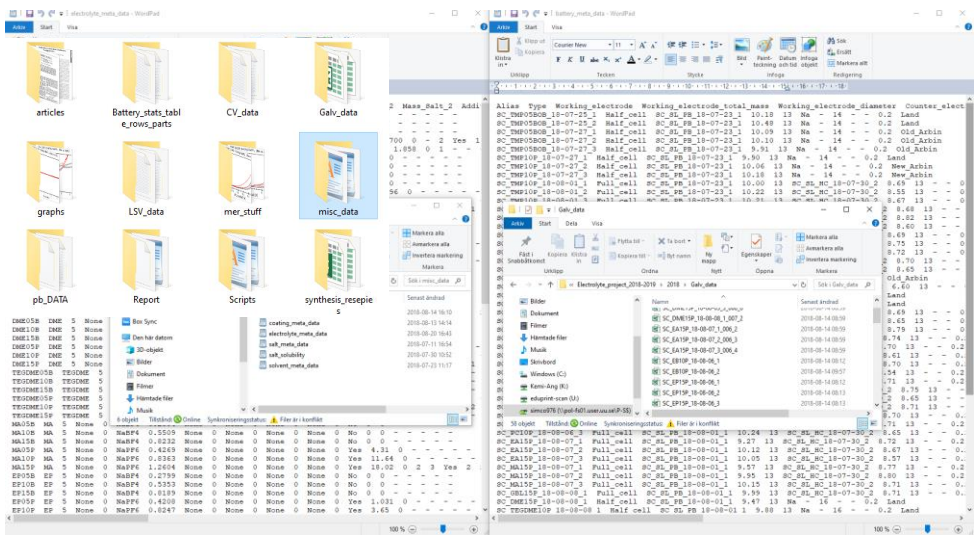
Simon Colbin

What is a database?

Cambridge Dictionary:

a large amount of information stored in a computer system in such a way that it can be easily looked at or changed

https://dictionary.cambridge.org/dictionary/english/database



SQLite

MySQL

Why SQLite?

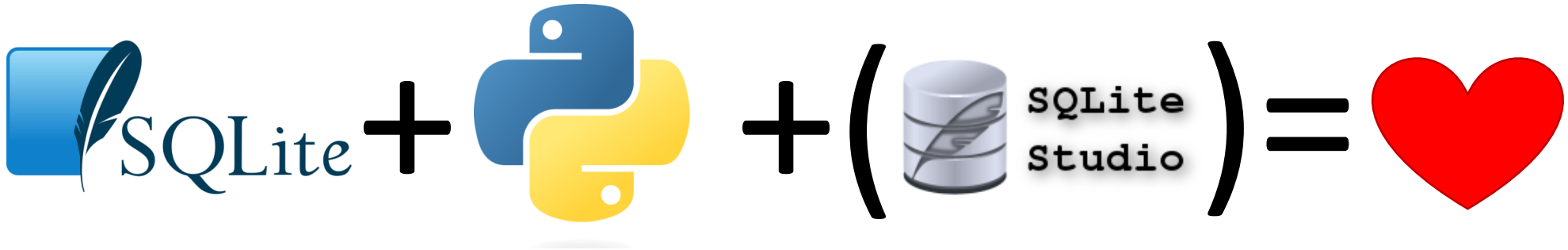
https://www.sqlite.org/docs.html

PROS

+ Standard format
+ Standard query (search) logic
+ Fast
+ Can store a lot of data (281 TB per db, 2 billion tables, 1.8e19 rows per table)
+ Likely installed on you computer
+ One file (.db) (you can have more)
+ No need for server
+ Works with native python (sqlite3)
+ Easy to set up a relational data structure
+ Allegedly easy to convert to more advanced format

CONS

- Offline
- One user at a time
- Not optimal for continuous data
- Limited functionally



How I approach sqlite?

Testing, creating tables, logging and looking simple data

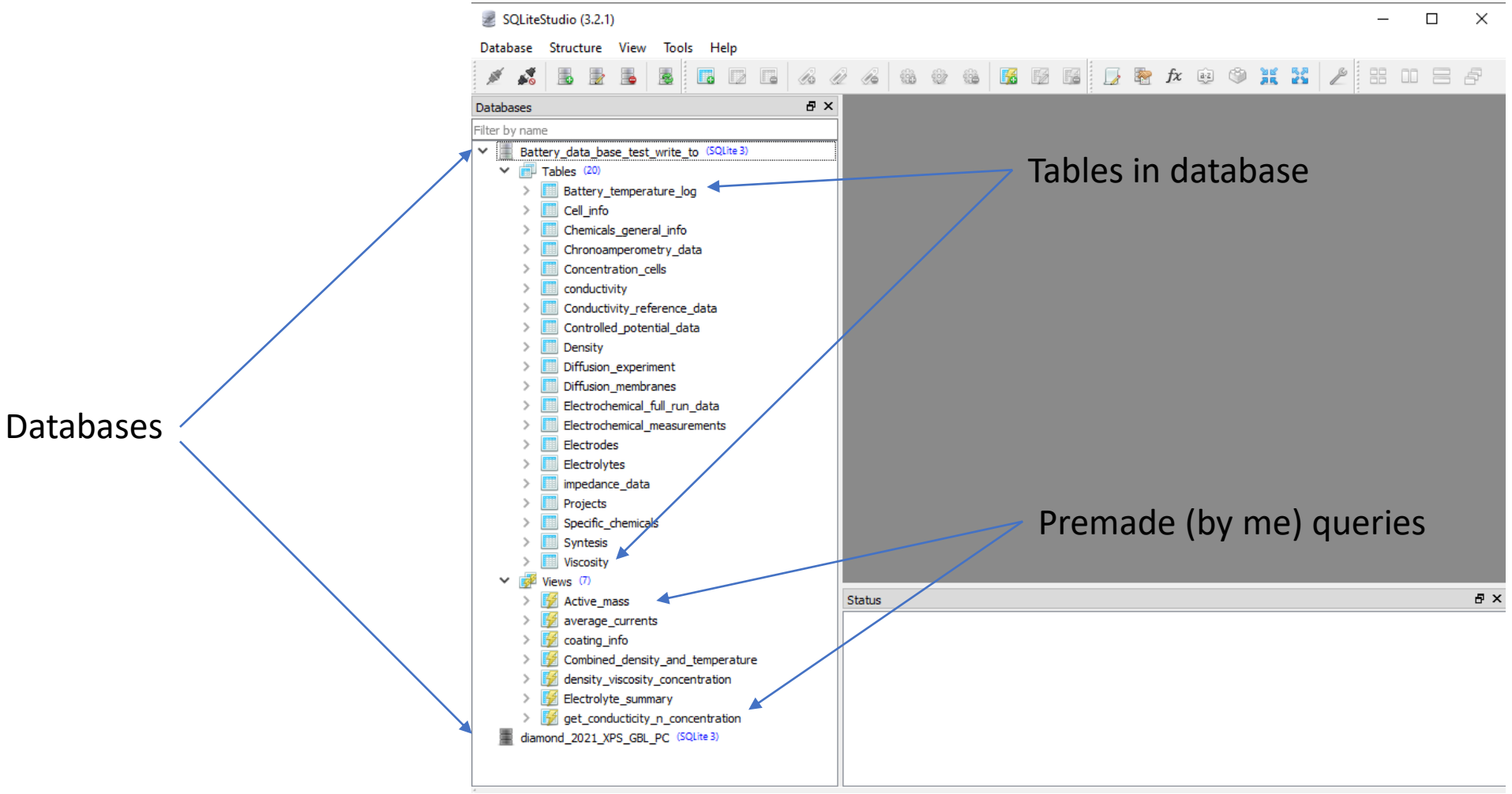
Logging a lot of data,
Reading a lot of data,
Advanced processing

Everything can be done from python, the native Sqlite3 python extension can actually do more than SQLite Studio

SQLite Studio

https://sqlitestudio.pl/

https://github.com/pawelsalawa/sqlitestudio/wiki/User_Manual

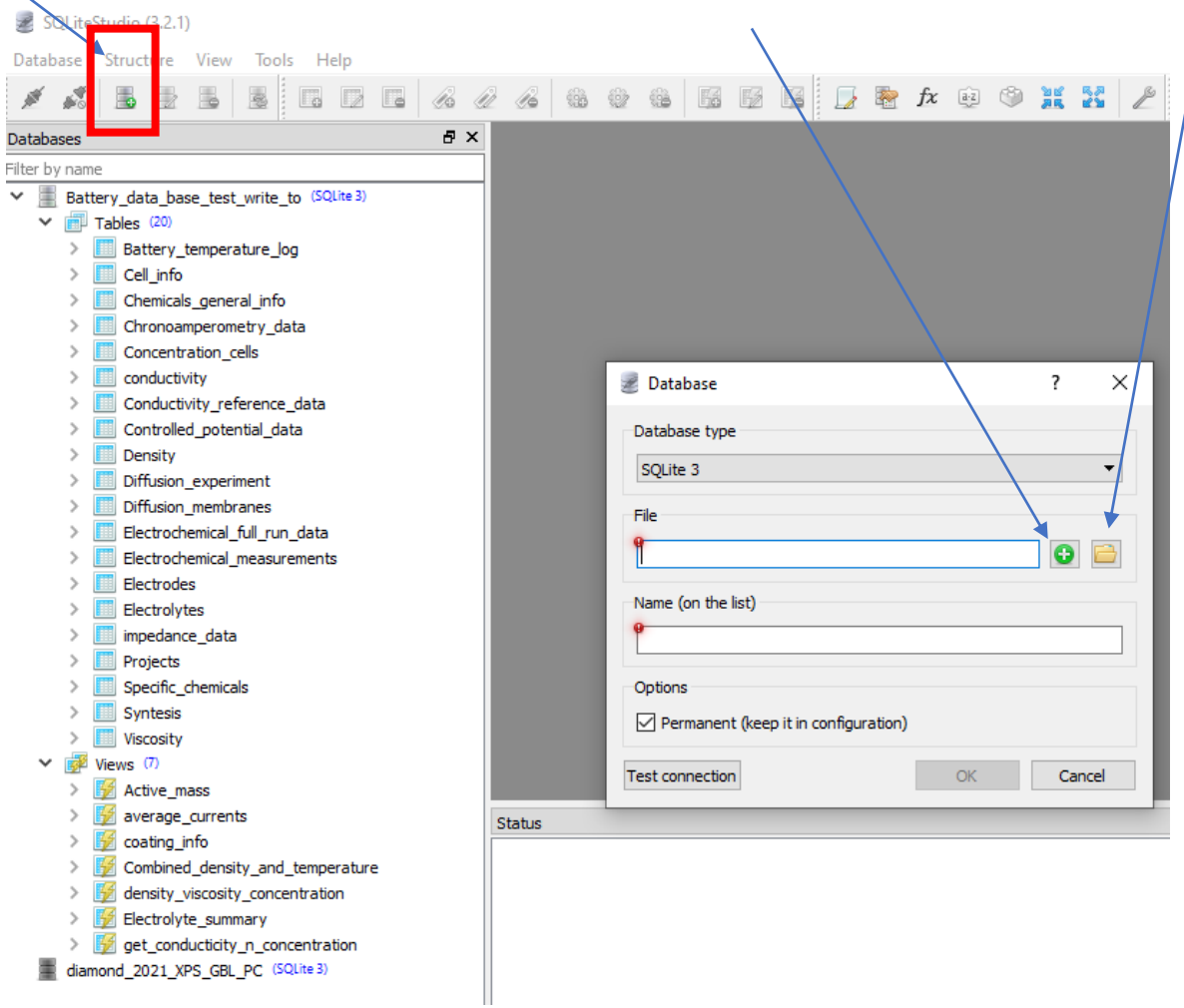


SQLite Studio

Add a database

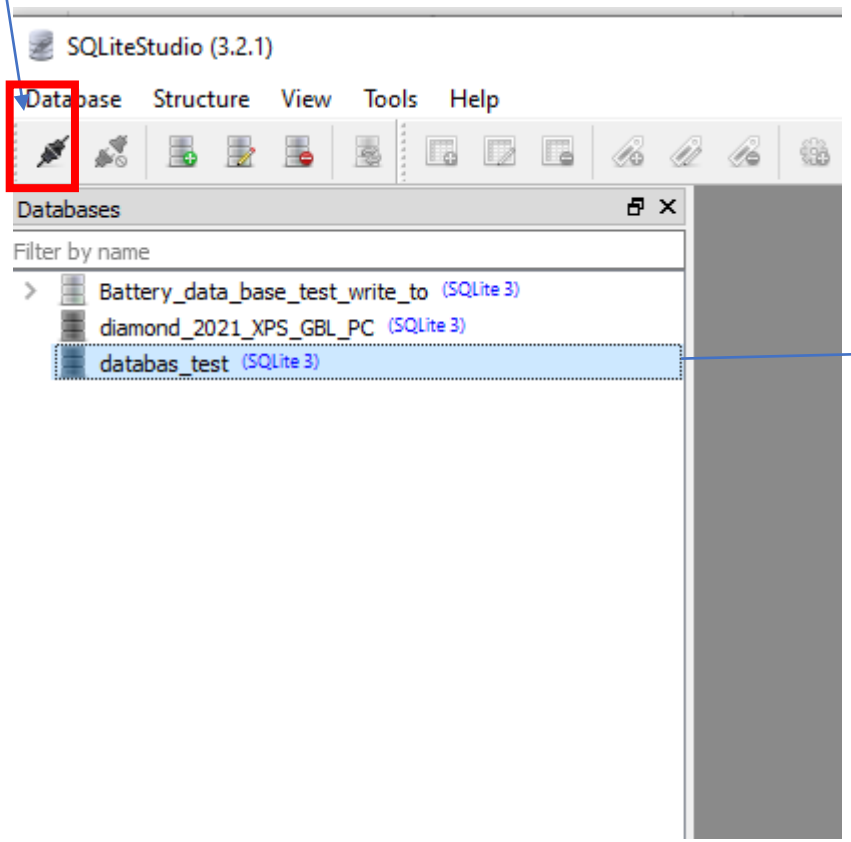
Create a new database

Add existing database

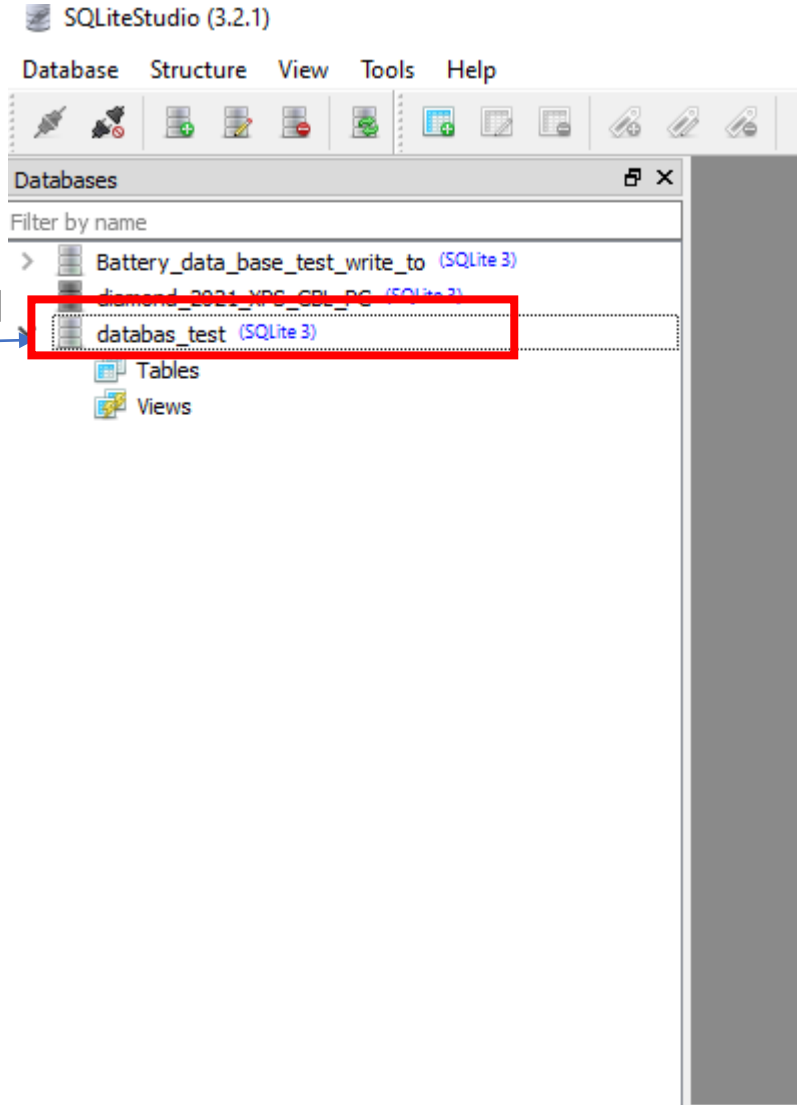


SQLite Studio

Connect to a database



Database connected



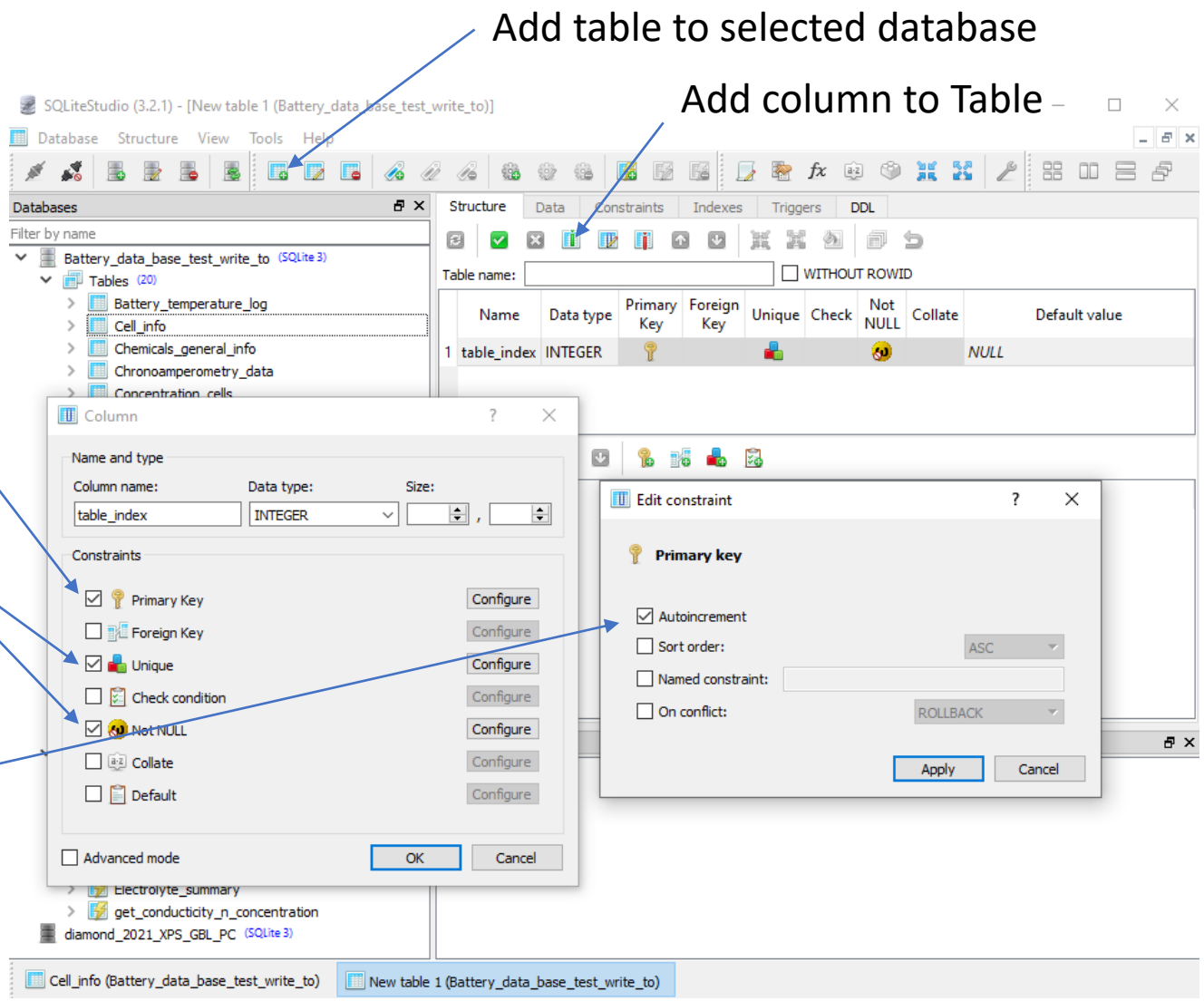
Create a table

Each table should be constructed to “maintain integrity”.

Each table should generally contain one column with a “primary key”.

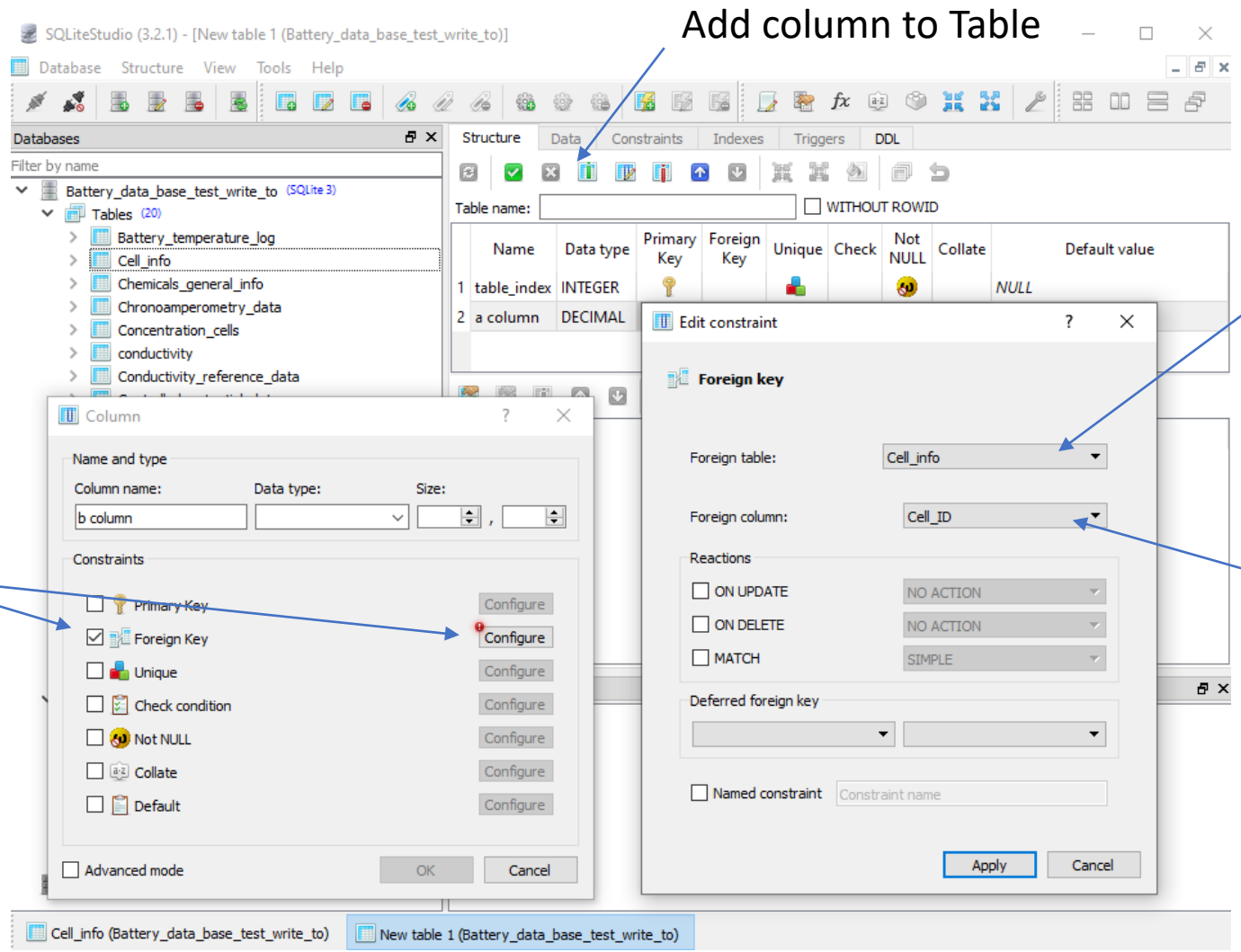
The values should be unique for each row of this column.

The value can be an automatically incremented integer, or a unique time, or a string (sample name).

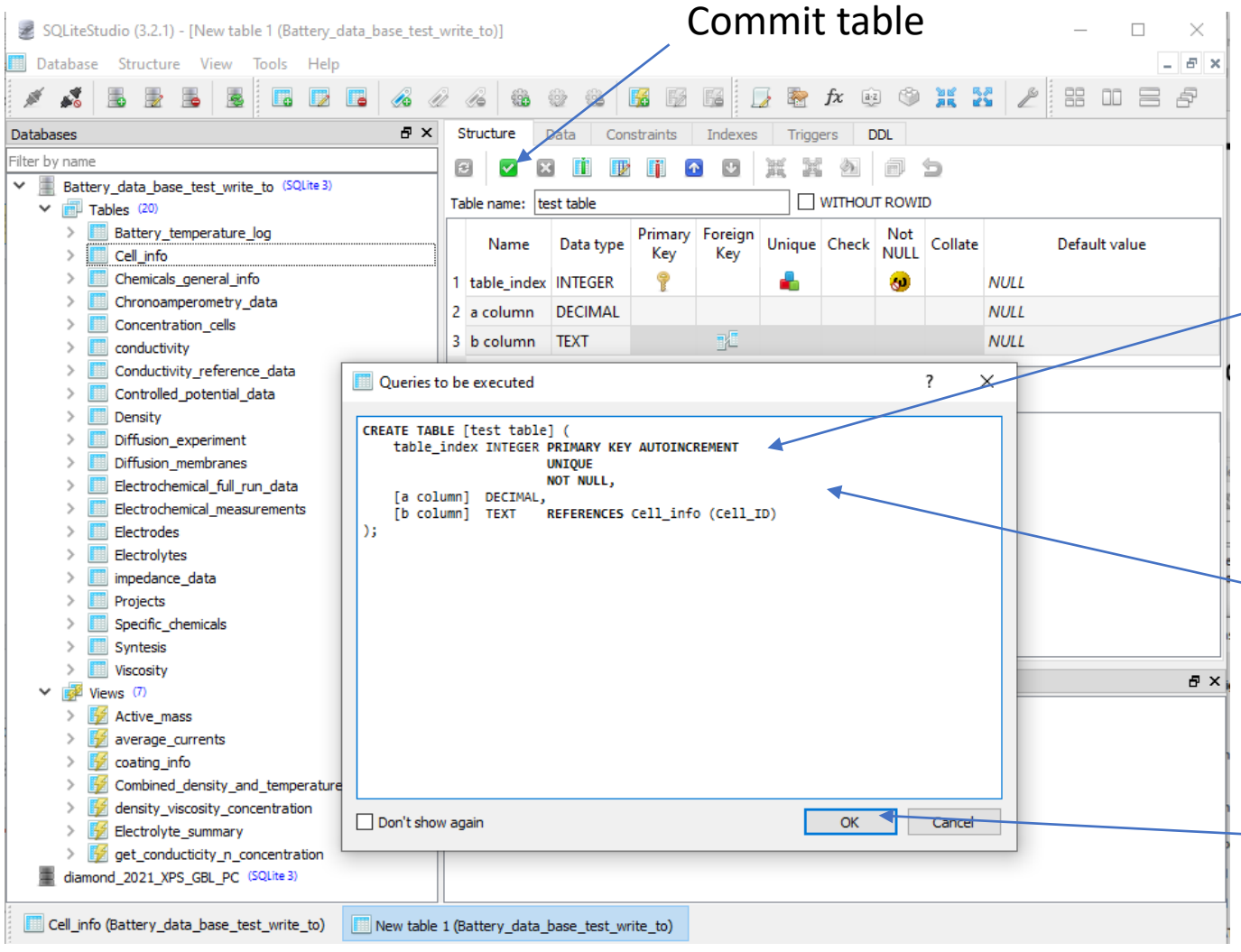


In my experience: It is possible to add columns to an existing operational table ☺

Create a table



Create a table

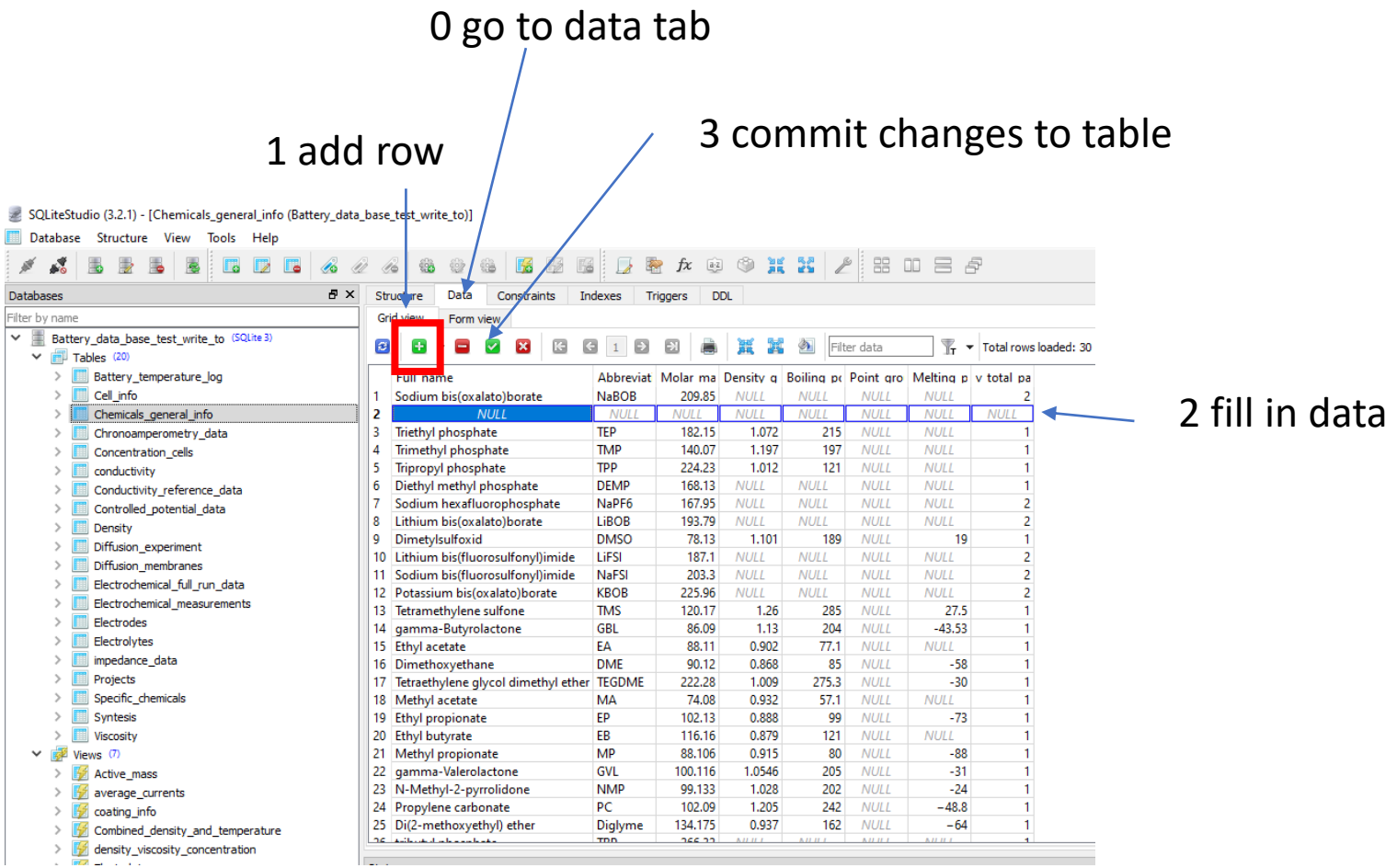


An sql command will be generated

This code could be used in python as a string

Execute the command, this is where the creation actually happens

Add data to table



	Full name	Abbreviat	Molar ma	Density g	Boiling pc	Point gro	Melting p	v total pa
1	Sodium bis(oxalato)borate	NaBOB	209.85	NULL	NULL	NULL	NULL	2
2	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
3	Triethyl phosphate	TEP	182.15	1.072	215	NULL	NULL	1
4	Trimethyl phosphate	TMP	140.07	1.197	197	NULL	NULL	1
5	Tripropyl phosphate	TPP	224.23	1.012	121	NULL	NULL	1
6	Diethyl methyl phosphate	DEMP	168.13	NULL	NULL	NULL	NULL	1
7	Sodium hexafluorophosphate	NaPF6	167.95	NULL	NULL	NULL	NULL	2
8	Lithium bis(oxalato)borate	LiBOB	193.79	NULL	NULL	NULL	NULL	2
9	Dimetylsulfoxid	DMSO	78.13	1.101	189	NULL	19	1
10	Lithium bis(fluorosulfonyl)imide	LiFSI	187.1	NULL	NULL	NULL	NULL	2
11	Sodium bis(fluorosulfonyl)imide	NaFSI	203.3	NULL	NULL	NULL	NULL	2
12	Potassium bis(oxalato)borate	KBOB	225.96	NULL	NULL	NULL	NULL	2
13	Tetramethylene sulfone	TMS	120.17	1.26	285	NULL	27.5	1
14	gamma-Butyrolactone	GBL	86.09	1.13	204	NULL	-43.53	1
15	Ethyl acetate	EA	88.11	0.902	77.1	NULL	NULL	1
16	Dimethoxyethane	DME	90.12	0.868	85	NULL	-58	1
17	Tetraethylene glycol dimethyl ether	TEGDME	222.28	1.009	275.3	NULL	-30	1
18	Methyl acetate	MA	74.08	0.932	57.1	NULL	NULL	1
19	Ethyl propionate	EP	102.13	0.888	99	NULL	-73	1
20	Ethyl butyrate	EB	116.16	0.879	121	NULL	NULL	1
21	Methyl propionate	MP	88.106	0.915	80	NULL	-88	1
22	gamma-Valerolactone	GVL	100.116	1.0546	205	NULL	-31	1
23	N-Methyl-2-pyrrolidone	NMP	99.133	1.028	202	NULL	-24	1
24	Propylene carbonate	PC	102.09	1.205	242	NULL	−48.8	1
25	Di(2-methoxyethyl) ether	Diglyme	134.175	0.937	162	NULL	−64	1

Maintaining integrity of database

Make sure that the database remains logical:

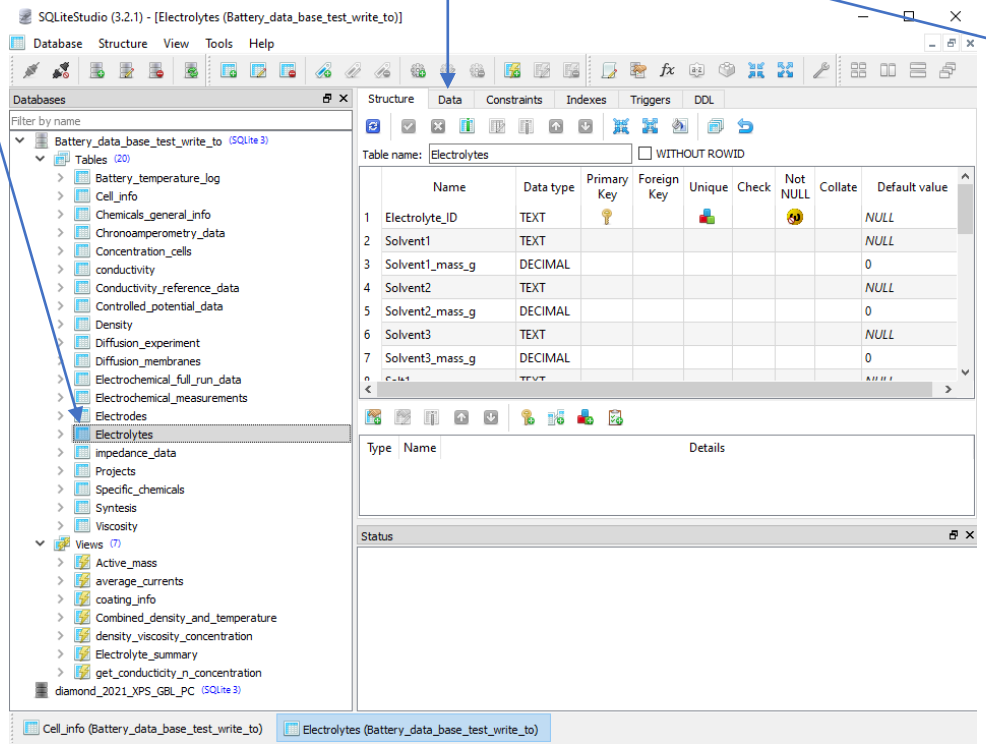
- Rows should to be distinguishable. → Use one (or more) primary key(s) that are (together) unique
- No “dead” ends. → The foreign value should exists in the other table.

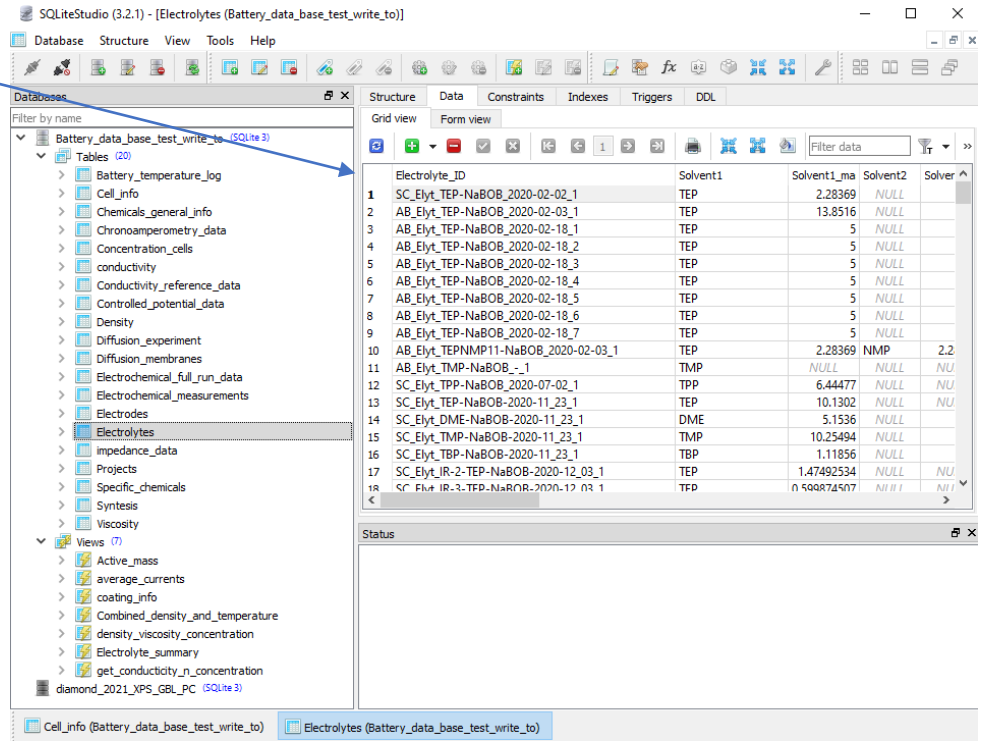
This is both helpful for you when thinking about your data, and experiments. It will also likely save you a time when handling the data.

Looking at my database in SQLite studio

Select table

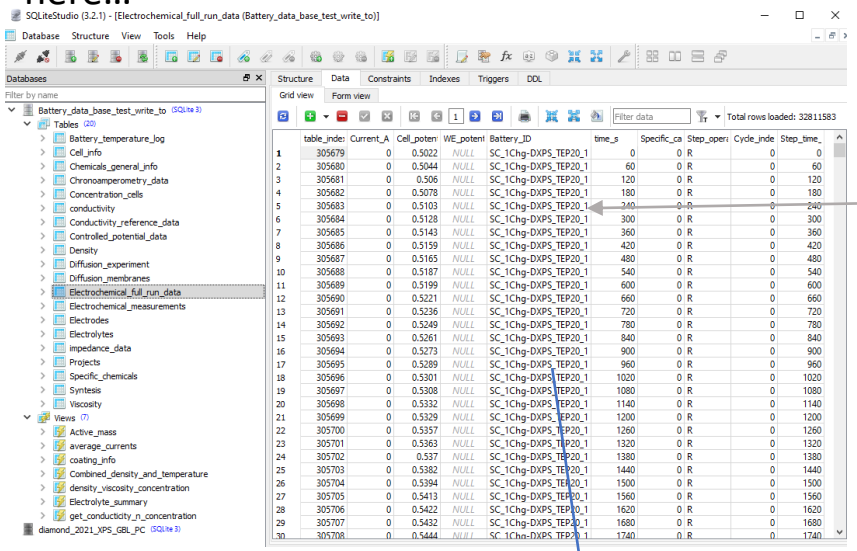
Select data

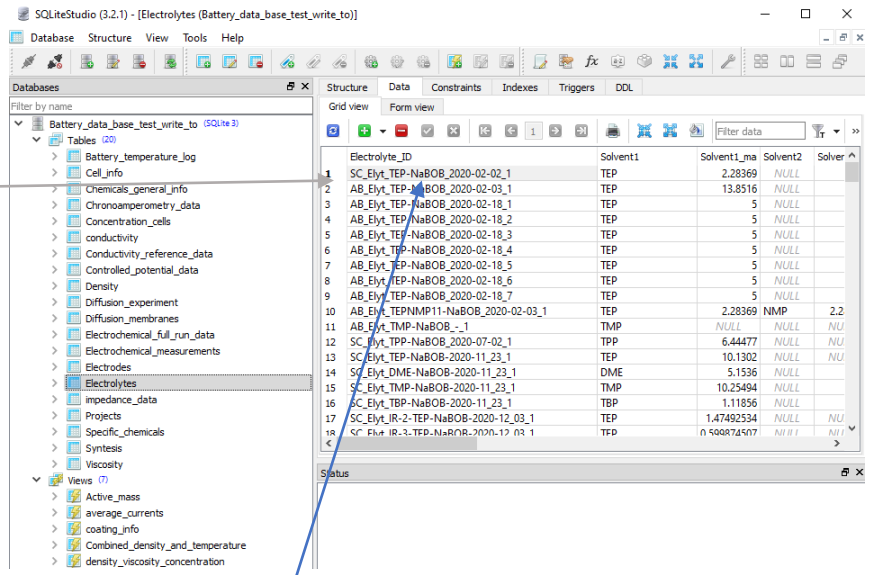




Cell cycling can be incorporated with less redundancy than shown here…

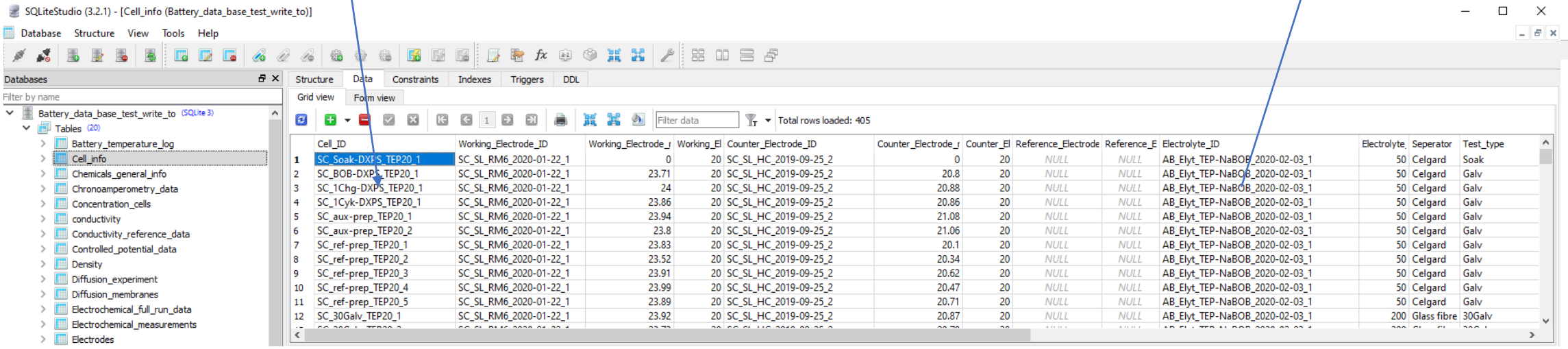
Looking at my database in SQLite studio





Everything becomes linked and can in principle be joined

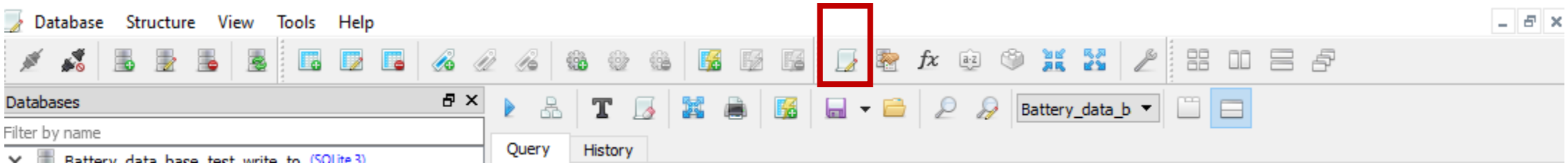
“Show second charge of all cells that contained a specific electrolyte solution“



	Cell_ID	Working_Electrode_ID	Working_Electrode_r	Working_El	Counter_Electrode_ID	Counter_Electrode_r	Counter_El	Reference_Electrode	Reference_E	Electrolyte_ID	Electrolyte	Seperator	Test_type
1	SC_Soak-DXPS_TEP20_1	SC_SL_RM6_2020-01-22_1	0	20	SC_SL_HC_2019-09-25_2	0	20	NULL	NULL	AB_Elyt_TEP-NaBOB_2020-02-03_1	50	Celgard	Soak
2	SC_BOB-DXPS_TEP20_1	SC_SL_RM6_2020-01-22_1	23.71	20	SC_SL_HC_2019-09-25_2	20.8	20	NULL	NULL	AB_Elyt_TEP-NaBOB_2020-02-03_1	50	Celgard	Galv
3	SC_1Chg-DXPS_TEP20_1	SC_SL_RM6_2020-01-22_1	24	20	SC_SL_HC_2019-09-25_2	20.88	20	NULL	NULL	AB_Elyt_TEP-NaBOB_2020-02-03_1	50	Celgard	Galv
4	SC_1Cyk-DXPS_TEP20_1	SC_SL_RM6_2020-01-22_1	23.86	20	SC_SL_HC_2019-09-25_2	20.86	20	NULL	NULL	AB_Elyt_TEP-NaBOB_2020-02-03_1	50	Celgard	Galv
5	SC_aux-prep_TEP20_1	SC_SL_RM6_2020-01-22_1	23.94	20	SC_SL_HC_2019-09-25_2	21.08	20	NULL	NULL	AB_Elyt_TEP-NaBOB_2020-02-03_1	50	Celgard	Galv
6	SC_aux-prep_TEP20_2	SC_SL_RM6_2020-01-22_1	23.8	20	SC_SL_HC_2019-09-25_2	21.06	20	NULL	NULL	AB_Elyt_TEP-NaBOB_2020-02-03_1	50	Celgard	Galv
7	SC_ref-prep_TEP20_1	SC_SL_RM6_2020-01-22_1	23.83	20	SC_SL_HC_2019-09-25_2	20.1	20	NULL	NULL	AB_Elyt_TEP-NaBOB_2020-02-03_1	50	Celgard	Galv
8	SC_ref-prep_TEP20_2	SC_SL_RM6_2020-01-22_1	23.52	20	SC_SL_HC_2019-09-25_2	20.34	20	NULL	NULL	AB_Elyt_TEP-NaBOB_2020-02-03_1	50	Celgard	Galv
9	SC_ref-prep_TEP20_3	SC_SL_RM6_2020-01-22_1	23.91	20	SC_SL_HC_2019-09-25_2	20.62	20	NULL	NULL	AB_Elyt_TEP-NaBOB_2020-02-03_1	50	Celgard	Galv
10	SC_ref-prep_TEP20_4	SC_SL_RM6_2020-01-22_1	23.99	20	SC_SL_HC_2019-09-25_2	20.47	20	NULL	NULL	AB_Elyt_TEP-NaBOB_2020-02-03_1	50	Celgard	Galv
11	SC_ref-prep_TEP20_5	SC_SL_RM6_2020-01-22_1	23.89	20	SC_SL_HC_2019-09-25_2	20.71	20	NULL	NULL	AB_Elyt_TEP-NaBOB_2020-02-03_1	50	Celgard	Galv
12	SC_30Galv_TEP20_1	SC_SL_RM6_2020-01-22_1	23.92	20	SC_SL_HC_2019-09-25_2	20.87	20	NULL	NULL	AB_Elyt_TEP-NaBOB_2020-02-03_1	200	Glass fibre	30Galv

Basic SQL query

Everything can be done as a manual SQL commands in SQLite studio. This is exactly the same syntax that would be used in the sqlite3 python package.



Basic SQL query

"get all columns and rows from cell info for cells created before may 2020"

Specify that data is to be read, and what data to extract

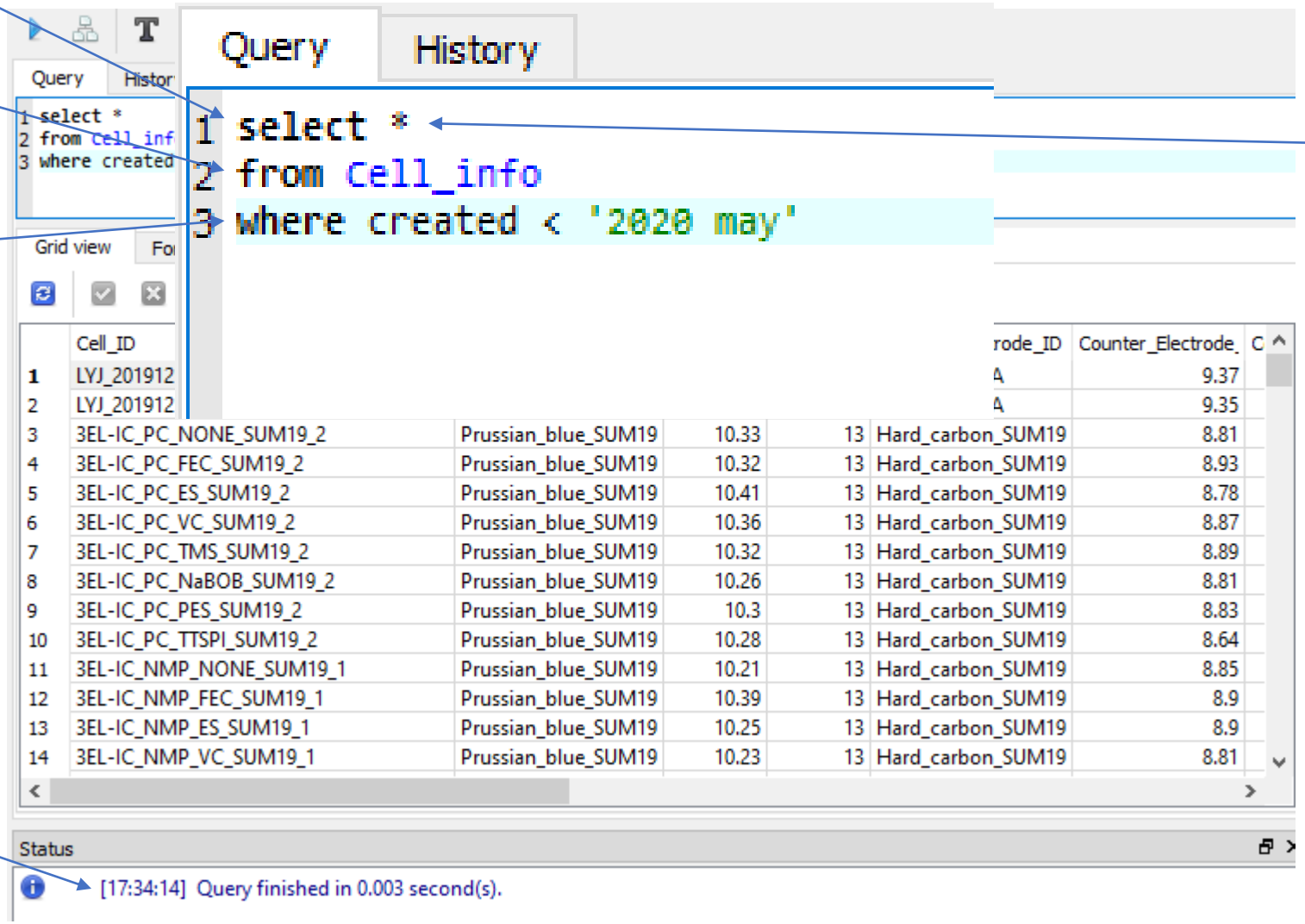
Specify what table to look in

Limit values to extract by constraint

Fast but easy…

The syntax is exactly the same when querying using sqlite3 in python!

'*' means that everything is to be read

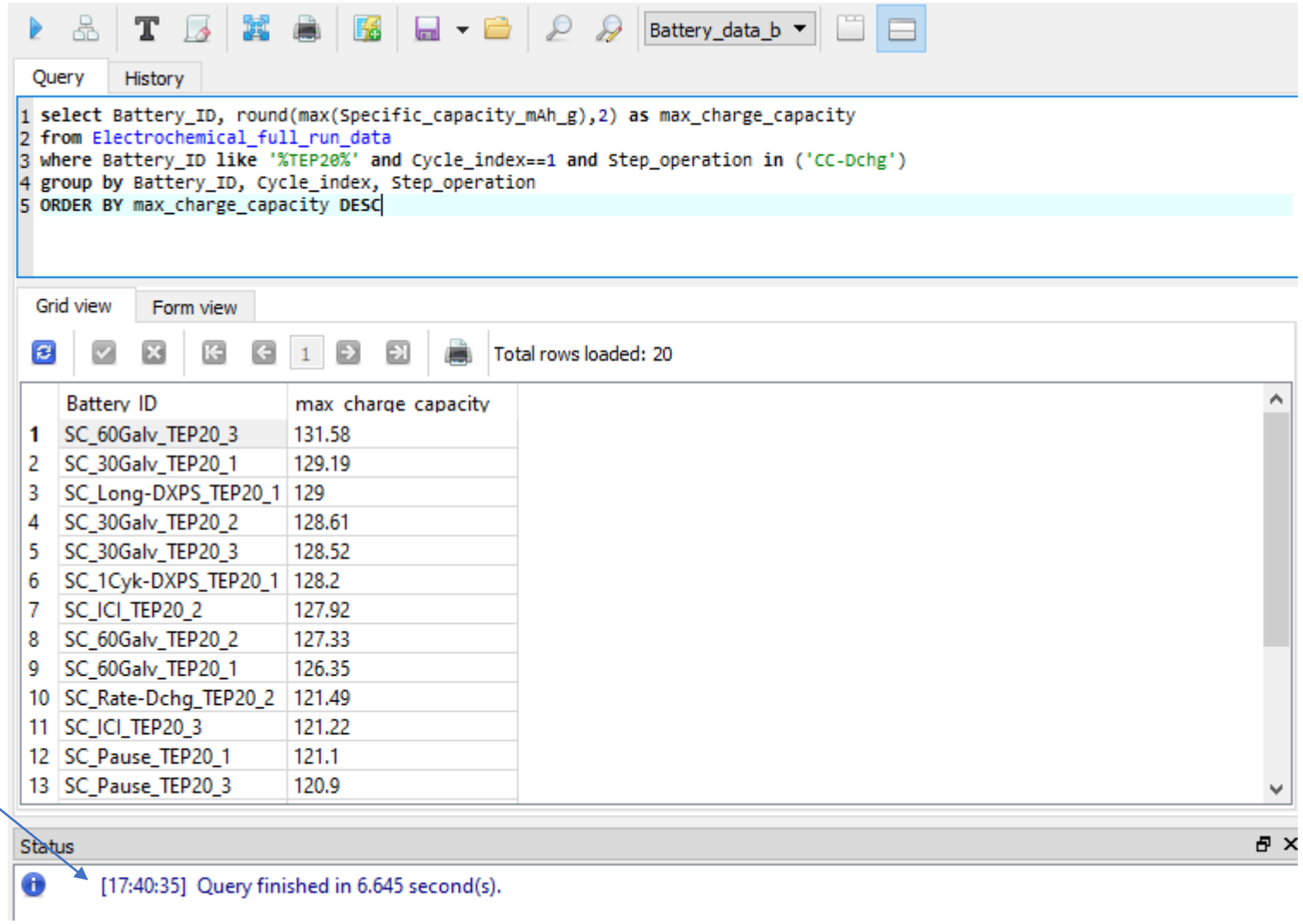


	Cell_ID					rode_ID	Counter_Electrode	C
1	LYJ_201912					A	9.37	
2	LYJ_201912					A	9.35	
3	3EL-IC_PC_NONE_SUM19_2	Prussian_blue_SUM19	10.33	13	Hard_carbon_SUM19		8.81	
4	3EL-IC_PC_FEC_SUM19_2	Prussian_blue_SUM19	10.32	13	Hard_carbon_SUM19		8.93	
5	3EL-IC_PC_ES_SUM19_2	Prussian_blue_SUM19	10.41	13	Hard_carbon_SUM19		8.78	
6	3EL-IC_PC_VC_SUM19_2	Prussian_blue_SUM19	10.36	13	Hard_carbon_SUM19		8.87	
7	3EL-IC_PC_TMS_SUM19_2	Prussian_blue_SUM19	10.32	13	Hard_carbon_SUM19		8.89	
8	3EL-IC_PC_NaBOB_SUM19_2	Prussian_blue_SUM19	10.26	13	Hard_carbon_SUM19		8.81	
9	3EL-IC_PC_PES_SUM19_2	Prussian_blue_SUM19	10.3	13	Hard_carbon_SUM19		8.83	
10	3EL-IC_PC_TTSPI_SUM19_2	Prussian_blue_SUM19	10.28	13	Hard_carbon_SUM19		8.64	
11	3EL-IC_NMP_NONE_SUM19_1	Prussian_blue_SUM19	10.21	13	Hard_carbon_SUM19		8.85	
12	3EL-IC_NMP_FEC_SUM19_1	Prussian_blue_SUM19	10.39	13	Hard_carbon_SUM19		8.9	
13	3EL-IC_NMP_ES_SUM19_1	Prussian_blue_SUM19	10.25	13	Hard_carbon_SUM19		8.9	
14	3EL-IC_NMP_VC_SUM19_1	Prussian_blue_SUM19	10.23	13	Hard_carbon_SUM19		8.81	

Basic SQL query

"get first cycle discharge capacity of every cell that contains 'TEP20' in the cell name"

This task means looking at **32 million rows** of data and **evaluating** based on the **where** statement and **group**ing and **sort**ing the result.
Using pandas.read_excel this would likely crash your computer or take a very long time…



	Battery_ID	max_charge_capacity
1	SC_60Galv_TEP20_3	131.58
2	SC_30Galv_TEP20_1	129.19
3	SC_Long-DXPS_TEP20_1	129
4	SC_30Galv_TEP20_2	128.61
5	SC_30Galv_TEP20_3	128.52
6	SC_1Cyk-DXPS_TEP20_1	128.2
7	SC_ICI_TEP20_2	127.92
8	SC_60Galv_TEP20_2	127.33
9	SC_60Galv_TEP20_1	126.35
10	SC_Rate-Dchg_TEP20_2	121.49
11	SC_ICI_TEP20_3	121.22
12	SC_Pause_TEP20_1	121.1
13	SC_Pause_TEP20_3	120.9

Basic SQL query

"get all cells using an electrolyte solution containing BOB anion, where the solution was created between 2020-02-01 and 2020-02-28, also show the days between when the solution was created and when the cell was made"

Predefined function: julianday()*

Query from a joint table of Cell_info and Electrolytes where the joining point is the Electrolyte_ID in each table

Battery_data_b

Query | History

```
1 select Cell_ID , Cell_info.Electrolyte_ID, Cell_info.Created as cell_created, Electrolytes.Created as electrolyte_solution_created,
2 julianday( Cell_info.Created)- julianday(Electrolytes.Created) as solution_age
3 from Cell_info inner join Electrolytes on Electrolytes.Electrolyte_ID==Cell_info.Electrolyte_ID
4 where (electrolyte_solution_created between '2020-02-01' and '2020-02-28') and Electrolytes.Salt1 like '%BOB%'
5 order by solution_age DESC
```

Grid view | Form view

Total rows loaded: 45

	Cell_ID	Electrolyte_ID	cell_created	electrolyte_solution_created	solution_age
1	SC_PB-CV_TEP20_2	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-08-20	2020-02-03	199
2	SC_CCAL-RCV_TEP20_2	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-08-20	2020-02-03	199
3	SC_CCAL-RCV-60_TEP20_1	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-08-20	2020-02-03	199
4	SC_AL-RCV_TEP20_1	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-08-20	2020-02-03	199
5	SC_AL-OXCV_TEP20_1	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-08-20	2020-02-03	199
6	SC_aux-prep_TEP20_3	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-08-19	2020-02-03	198
7	SC_aux-prep_TEP20_4	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-08-19	2020-02-03	198
8	SC_ref-prep_TEP20_6	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-08-19	2020-02-03	198
9	SC_ref-prep_TEP20_7	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-08-19	2020-02-03	198
10	SC_ref-prep_TEP20_8	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-08-19	2020-02-03	198
11	SC_ICI_TEP20_1	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-02-20	2020-02-03	17
12	SC_ICI_TEP20_2	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-02-20	2020-02-03	17
13	SC_ICI_TEP20_3	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-02-20	2020-02-03	17
14	SC_60Galv_TEP20_1	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-02-20	2020-02-03	17
15	SC_60Galv_TEP20_2	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-02-20	2020-02-03	17
16	SC_60Galv_TEP20_3	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-02-20	2020-02-03	17
17	SC_Pause_TEP20_1	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-02-19	2020-02-03	16
18	SC_Pause_TEP20_2	AB_Elyt_TEP-NaBOB_2020-02-03_1	2020-02-19	2020-02-03	16

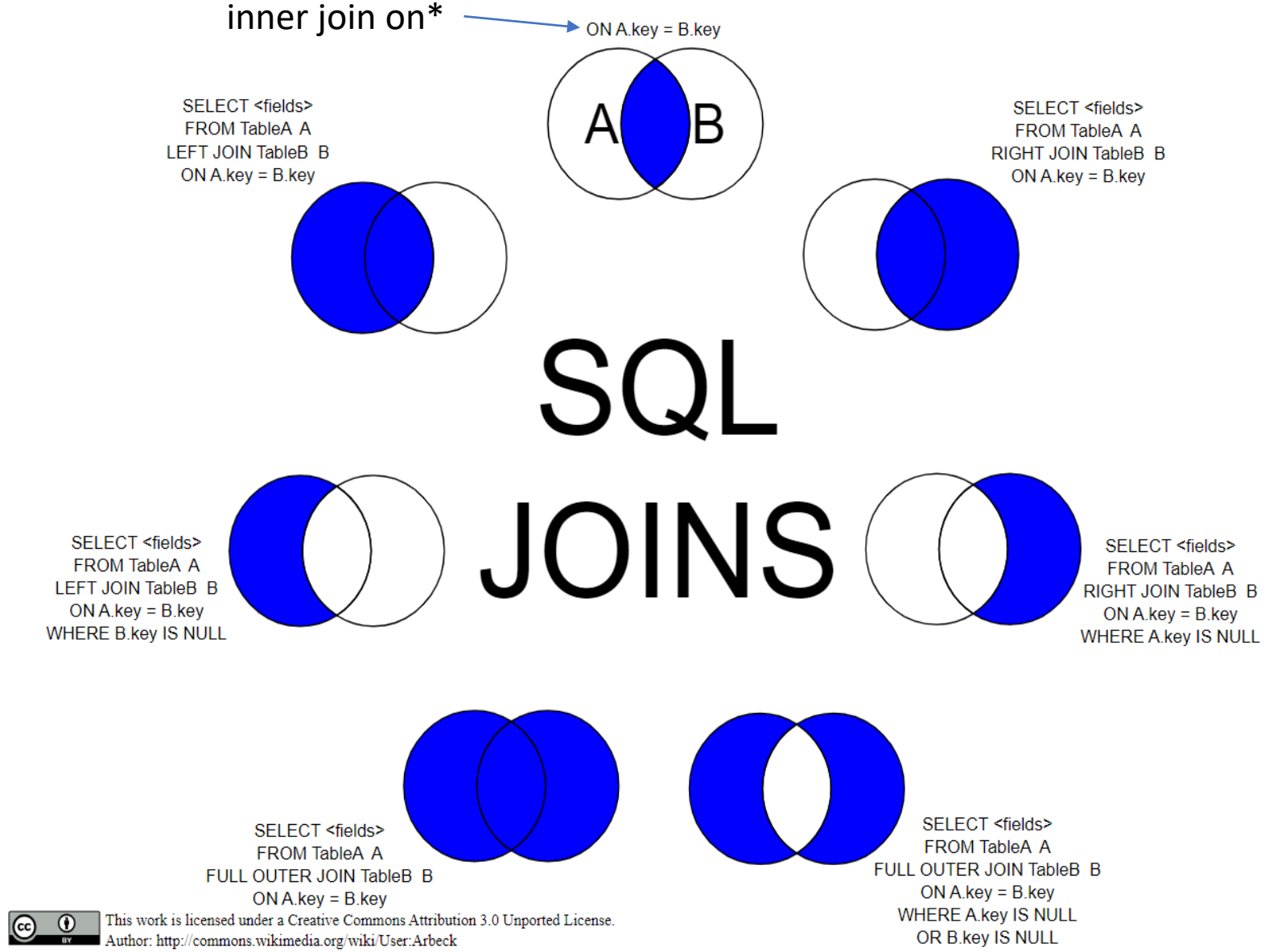
Status

[18:16:53] Query finished in 0.001 second(s).

Task:

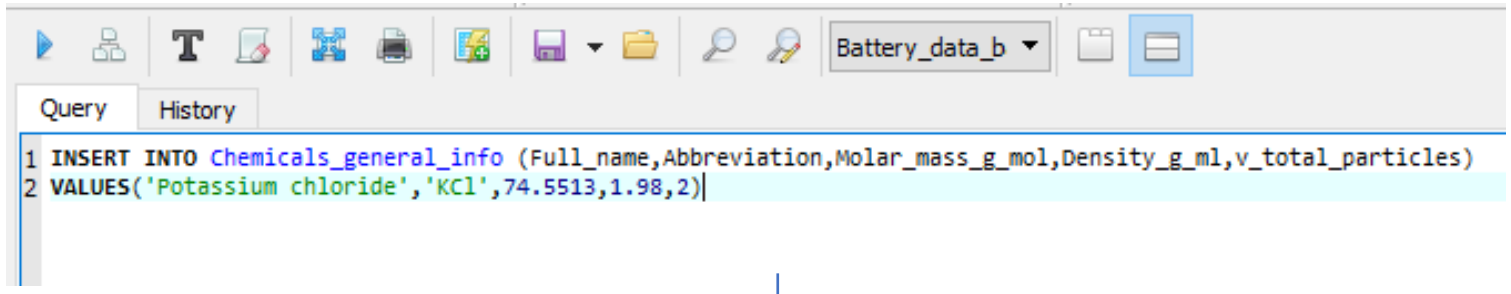
- Join two tables Cell_info and Electrolytes.
- Specify which type of join is meant and on which column the tables should be joint on.
- Some columns have the same name, and you need to specify which column is aim for "table.column".

*sqlite3 in python contains more predefined functions than SQLiteStudio



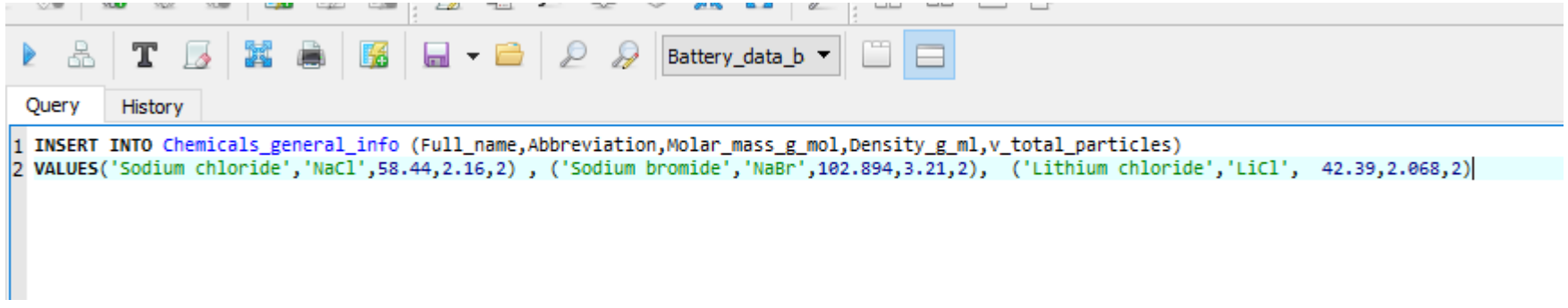
Not all operations are available in SQLite, sqlite3 in python have more options than SQLiteStudio.

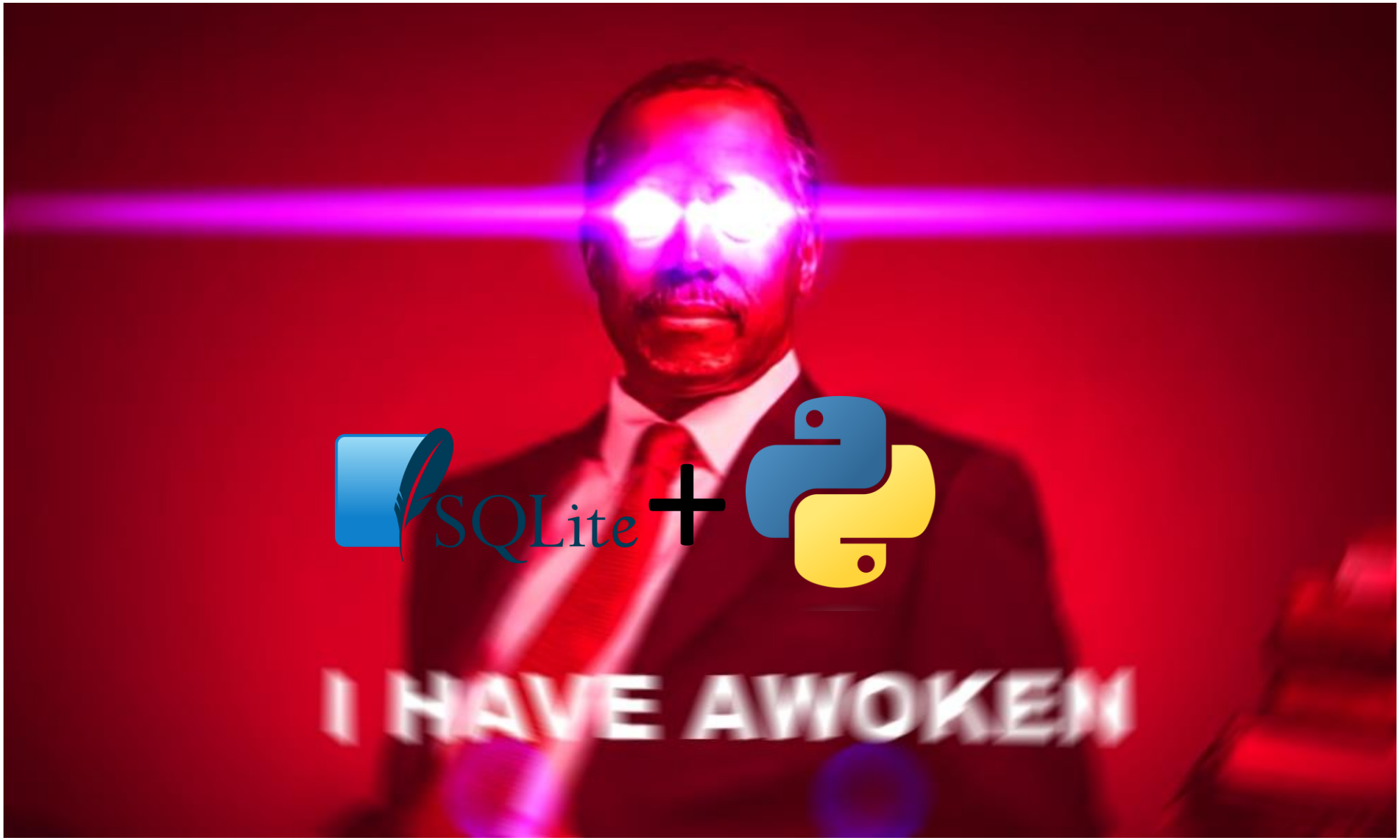
Manually writing to SQLite database



15	Potassium chloride	KCl	74.5513	1.98	NULL	NULL	NULL	2
16	Potassium bis(oxalato)borate	KBOB	225.96	NULL	NULL	NULL	NULL	2
17	N-Methyl-2-pyrrolidone	NMP	99.133	1.028	202	NULL	-24	1

creating multiple rows in one command





https://docs.python.org/3/library/sqlite3.html

Basic SQL query in python

```
db_dir="X:/data_all_projects/Battery_data_base_test_write_to.db"


def get_molar_masses_from_table():
    q_string="""select Abbreviation, Molar_mass_g_mol,v_total_particles from Chemicals_general_info"""
    connection = sqlite3.connect(db_dir)
    cursor = connection.cursor()
    cursor.execute(q_string)
    records = cursor.fetchall()
    connection.close()
    return {x[0]:x[1] for x in records}, {x[0]:x[2] for x in records}
```

1 Connect to a database
2 Create a cursor object
3 Execute an SQLquery as a sting
4 Fetch the result

The possibilities are endless when combining python and SQLite*

- Use sting formatting to make the table selection variable:

```
"""Select * from {var1}""".format(**{'var1':"Cell_info"})
'Select * from Cell_info'
```

- Create query builder.

```
plot_capacity_VS_voltage(get_battery_names_from_cell_info('TEP20',' and Cell_ID like "%Rate%"'),(1,4))
```

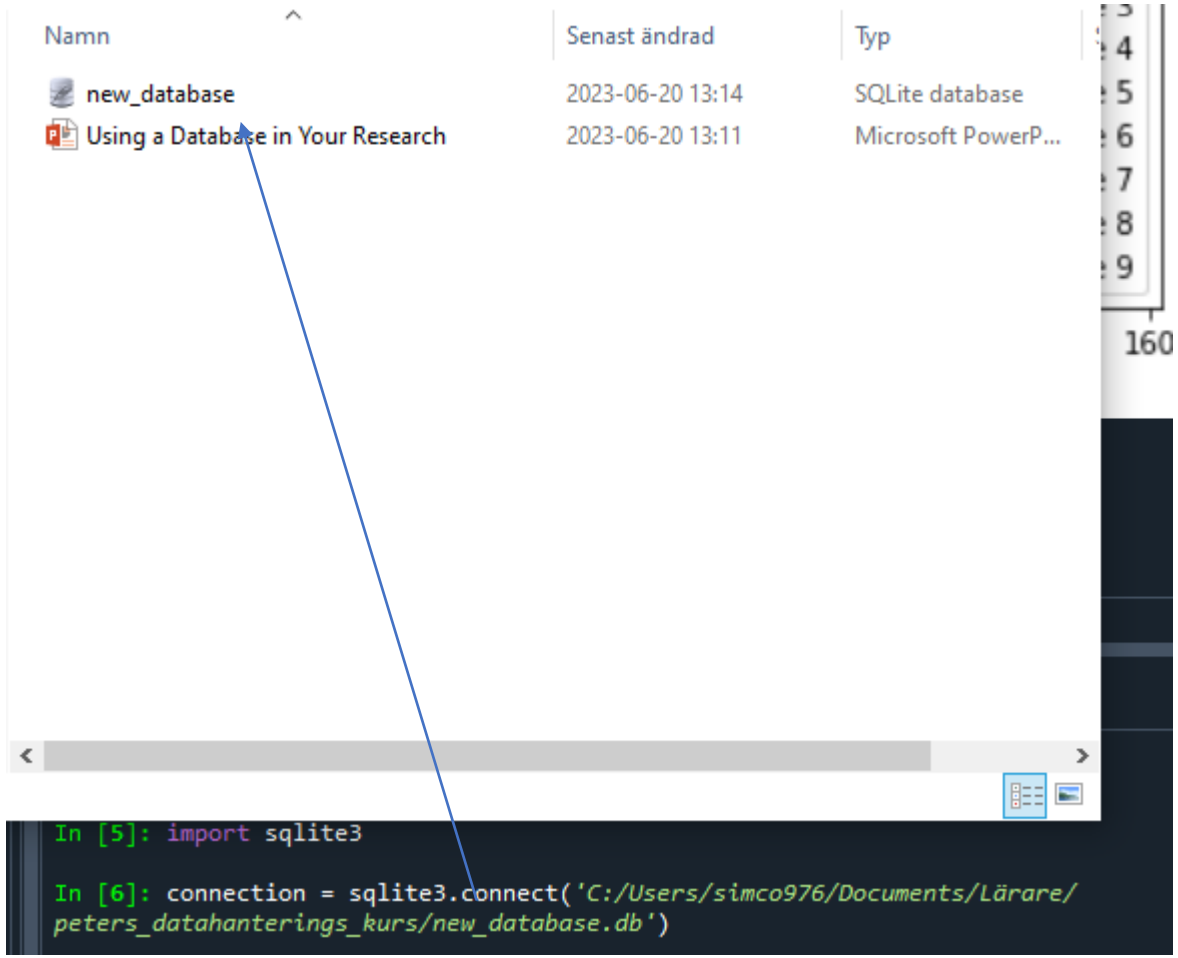
```
select Battery_ID,Cell_potential_V,WE_potential_V,(case when WE_potential_V is not NULL then
WE_potential_V-Cell_potential_V end) as
CE_potential_V,Specific_capacity_mAh_g,Cycle_index,Step_operation from
Electrochemical_full_run_data
Where (Battery_ID in ('SC_Rate-chg_TEP20_1','SC_Rate-chg_TEP20_2','SC_Rate-
chg_TEP20_3','SC_Rate-Dchg_TEP20_1','SC_Rate-Dchg_TEP20_2','SC_Rate-Dchg_TEP20_3')) and
(cycle_index BETWEEN 1 and 4)
```

- Write data

```
def write_df_to_db(df):

    does_df_exists=look_for_df_in_db(df)
    if does_df_exists==True:
        print('df exists, delete old data before adding')
        return

    if look_for_df_in_cell_info(df):
        conn = sqlite3.connect(db_path)

        df.to_sql('Chronoamperometry_data', con=conn, if_exists='append',index=False)

        conn.close()
    else:
        print('cell not registered in cell info, do this first')
```

*A lot of the things one can do is frowned upon, because if means a safety risk if a hacker accesses the database, they might ruin you! Still there is a difference between having a private database and running a public database for a company.

Creating a database from python



Simply connect to a database file that does not already exist.

```python
def get_conductivity_and_viscosity_data(electrolytes=[],solvents=[],addetives='NULL'):

    es_ids=get_electrolyte_IDs_with_composition(electrolytes,solvents,addetives)

    es_info=get_electrolyte_solutions_from_table(es_ids)


    q_string="""Select Electrolyte_ID, Conductivity_mS_cm,conductivity.Temperature_C, Density_g_cm, Dynamic_viscosity_mPAS
    from conductivity inner join
    (Select Density.Sample_ID, Density.Temperature_C, Dynamic_viscosity_mPAS, Density_g_cm from Viscosity inner join Density on
    (Density.Sample_ID==Viscosity.Sample_ID and Density.Temperature_C==Viscosity.Temperature_C )) as temp1
    on (temp1.Sample_ID == conductivity.Electrolyte_ID and temp1.Temperature_C==conductivity.Temperature_C)
    where electrolyte_ID in ({})
    group by conductivity.Electrolyte_ID,conductivity.Temperature_C
    order by Electrolyte_ID, conductivity.Temperature_C""".format(','.join(["'"+x+"'" for x in es_ids]))


    connection = sqlite3.connect(db_dir)
    cursor = connection.cursor()
    cursor.execute(q_string)
    records = cursor.fetchall()

    headers = [description[0] for description in cursor.description]
    connection.close()


    molar_c_list=[]
    mole_fraction_c_list=[]
    molal_c_list=[]
    solvent_list=[]
    electrolyte_list=[]

    for x in records:
        mole_fraction_c_list+=[sum([es_info[x[0]]['Electrolytes'][y]['mole_fraction'] for y in es_info[x[0]]['Electrolytes']])]
        molal_c_list+=[sum([es_info[x[0]]['Electrolytes'][y]['c_Molal'] for y in es_info[x[0]]['Electrolytes']])]
        moles_electrolyte=sum([es_info[x[0]]['Electrolytes'][y]['moles'] for y in es_info[x[0]]['Electrolytes']])
        total_mass=0
        solvent_list+=[ ', '.join(es_info[x[0]]['Solvents'].keys()) ]
        electrolyte_list+=[ ', '.join(es_info[x[0]]['Electrolytes'].keys())]
        for y in ['Electrolytes','Solvents','Addetives']:
            if len(es_info[x[0]][y].keys())>0:
                for z in es_info[x[0]][y]:
                    total_mass+=es_info[x[0]][y][z]['mass']


        molar_c_list+=[1000*moles_electrolyte/(total_mass/x[3])]




    out_dict={'Electrolyte': electrolyte_list,'Solvent':solvent_list,'c mol/L': molar_c_list,'c mol/kg': molal_c_list, 'x (mole fraction)': mole_fraction_c_list}
    records=np.array(records).T

    table_dict={x[1]:records[x[0]].astype(np.float64) for x in enumerate(headers) if x[1] != 'Electrolyte_ID'}

    out_dict.update(table_dict)

    df=pd.DataFrame(out_dict)


    return df.sort_values(['x (mole fraction)','Temperature_C'])
```

```python
df=get_conductivity_and_viscosity_data(['NaBOB'],['TEP'])
```

Note of proper file/database interactions in python

https://docs.python.org/3/reference/compound_stmts.html

https://docs.python.org/3/reference/datamodel.html#context-managers

https://www.youtube.com/watch?v=-aKFBoZpiqA

On failure or exit indent, the file will automatically be closed.

On failure between these points may make it necessary to restart computer before accessing the file again. (Can possible damage the file?)

There are cases when it is more convenient to not use a “context manager” (with statement). But, then, there are probably better ways to handle the file connation anyway.

```
20
21  db_dir="X:/data_all_projects/Battery_data_base_test_write_to.db"
22
23
24  def good_get_molar_masses_from_table():
25      q_string="""select Abbreviation, Molar_mass_g_mol,v_total_particles from Chemicals_general_info"""
26      with sqlite3.connect(db_dir) as connection:
27          cursor = connection.cursor()
28          cursor.execute(q_string)
29          records = cursor.fetchall()
30
31      return {x[0]:x[1] for x in records}, {x[0]:x[2] for x in records}
32
33
34
35  def bad_get_molar_masses_from_table():
36      q_string="""select Abbreviation, Molar_mass_g_mol,v_total_particles from Chemicals_general_info"""
37      connection = sqlite3.connect(db_dir)
38      cursor = connection.cursor()
39      cursor.execute(q_string)
40      records = cursor.fetchall()
41      connection.close()
42      return {x[0]:x[1] for x in records}, {x[0]:x[2] for x in records}
```

Live demo?