

Using a Database in Your Research.

2023-06-19

Simon Colbin

What is a database?

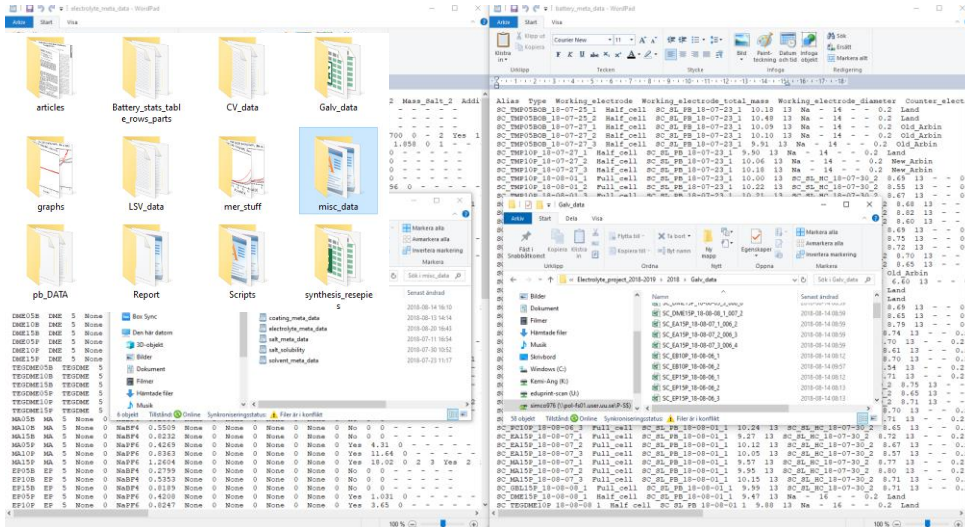
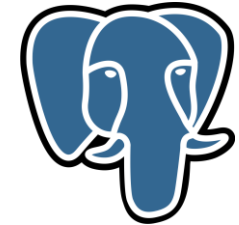
Cambridge Dictionary:

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>



mongoDB®



Why SQLite?

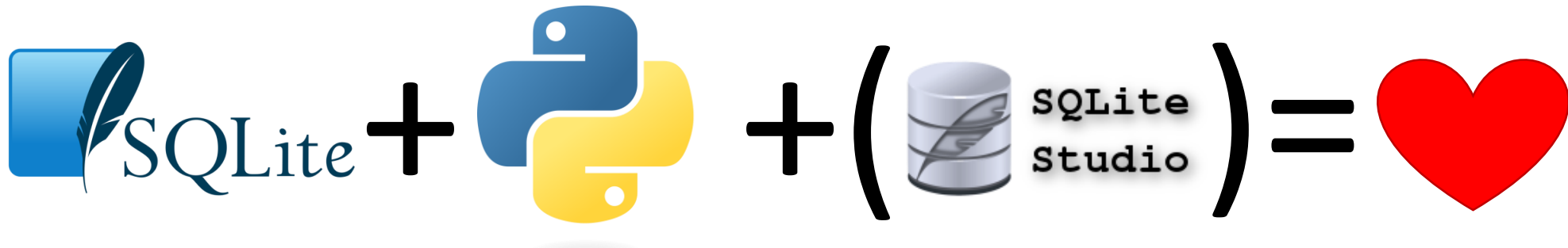
<https://www.sqlite.org/docs.html>

PROS

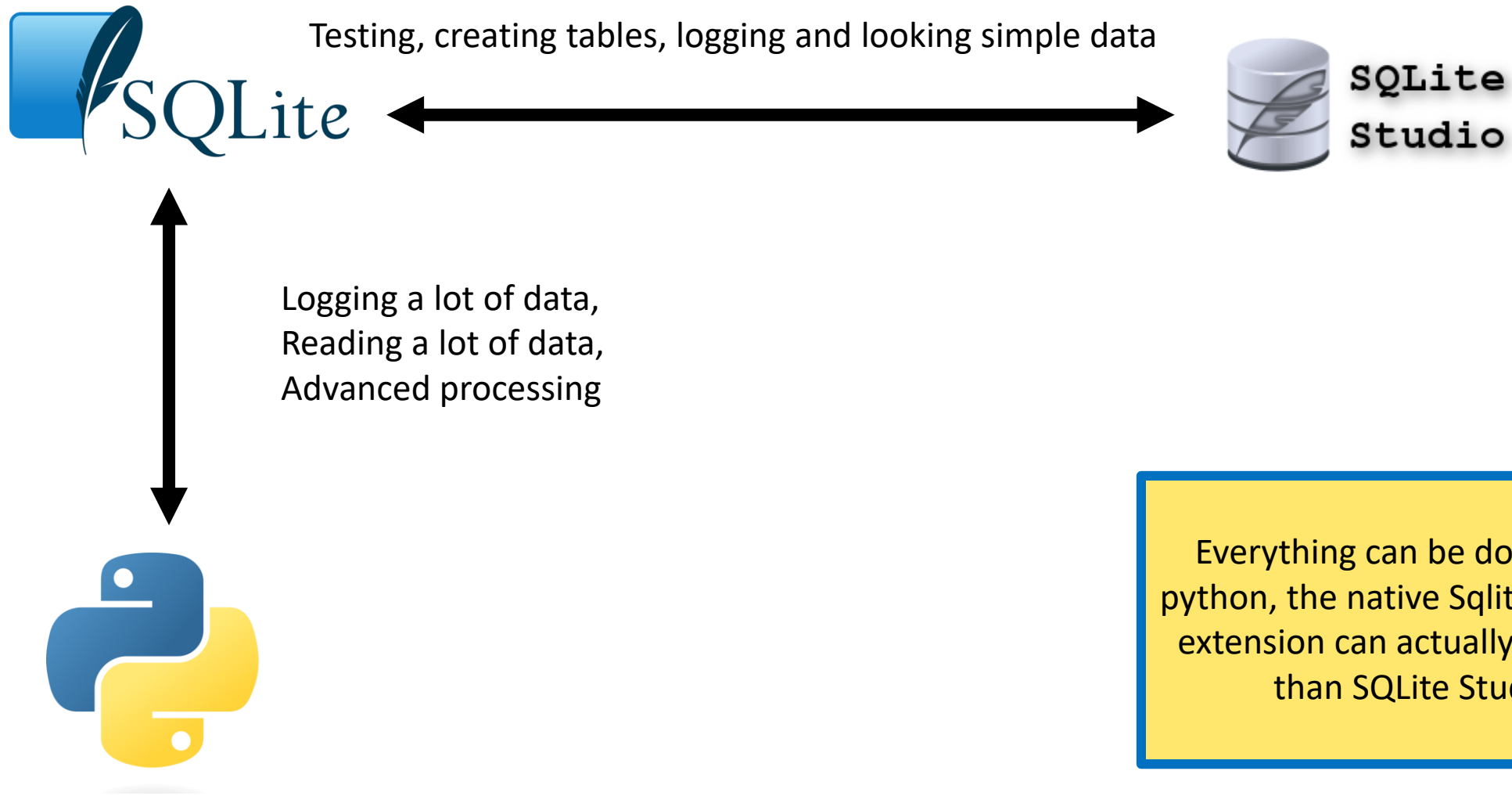
- + Standard format
- + Standard query (search) logic
- + Fast
- + Can store a lot of data (281 TB per db, 2 billion tables, 1.8e19 rows per table)
- + Likely installed on your computer
- + One file (.db) (you can have more)
- + No need for server
- + Works with native python (sqlite3)
- + Easy to set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

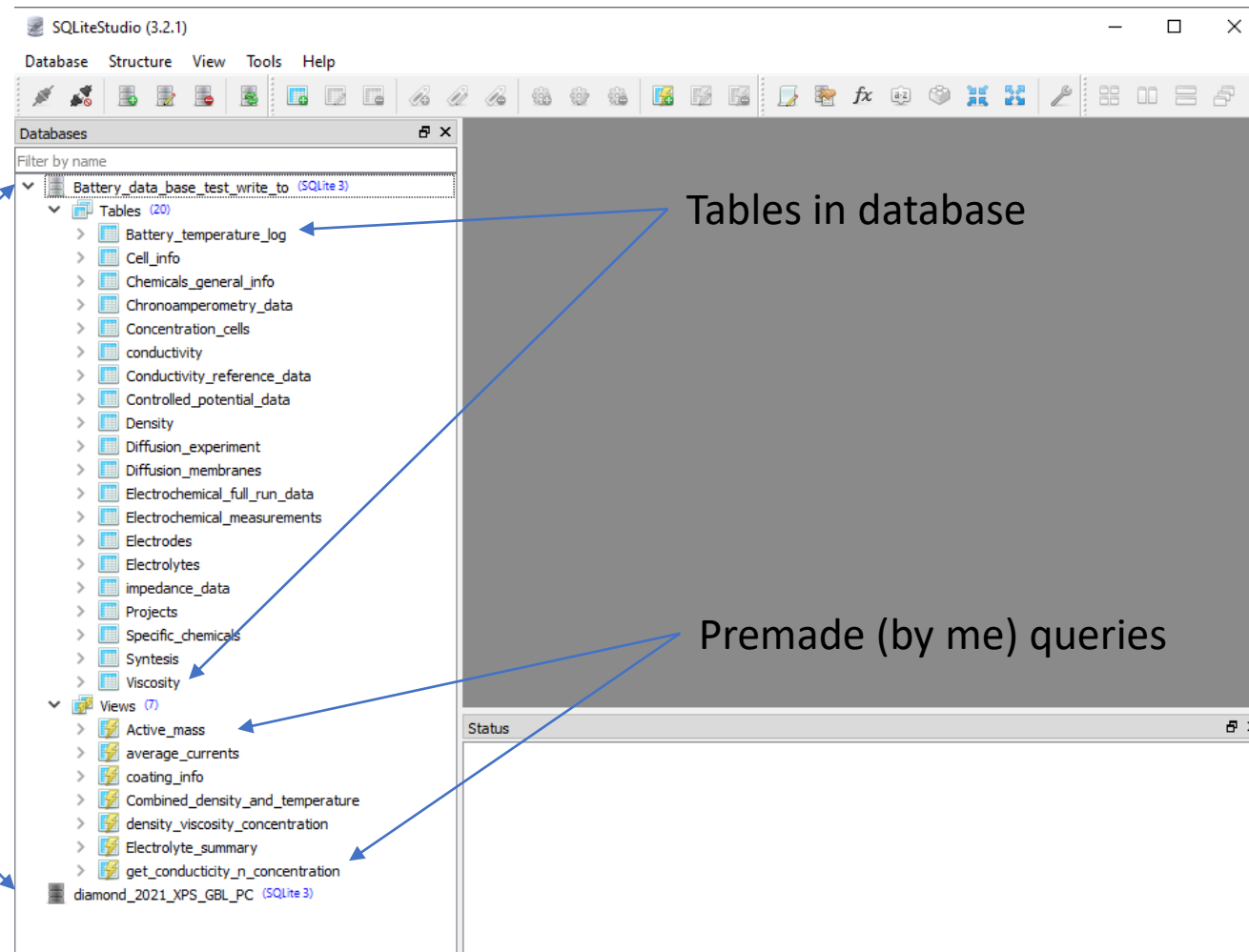


SQLite Studio

<https://sqlitestudio.pl/>

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

Databases

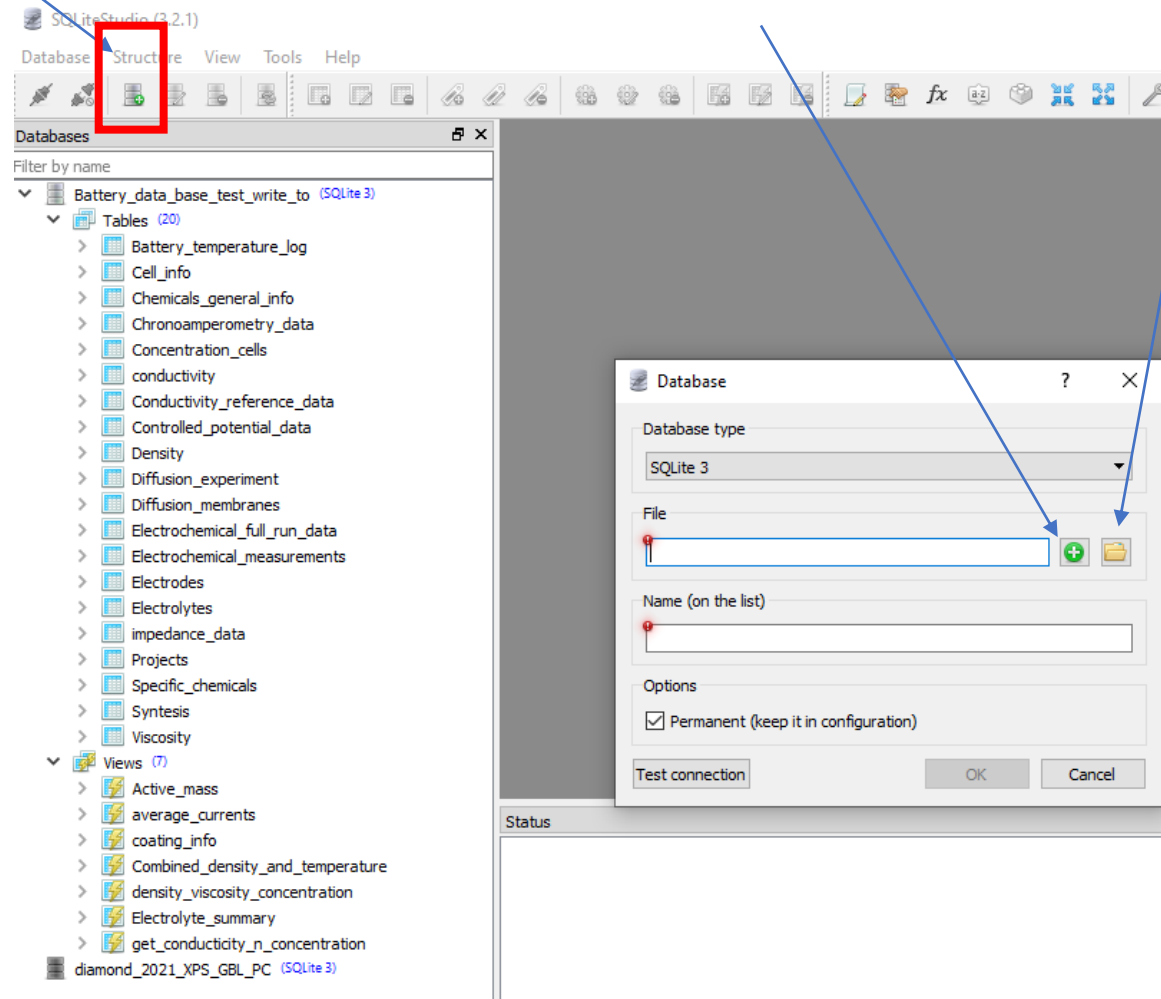


SQLite Studio

Add a database

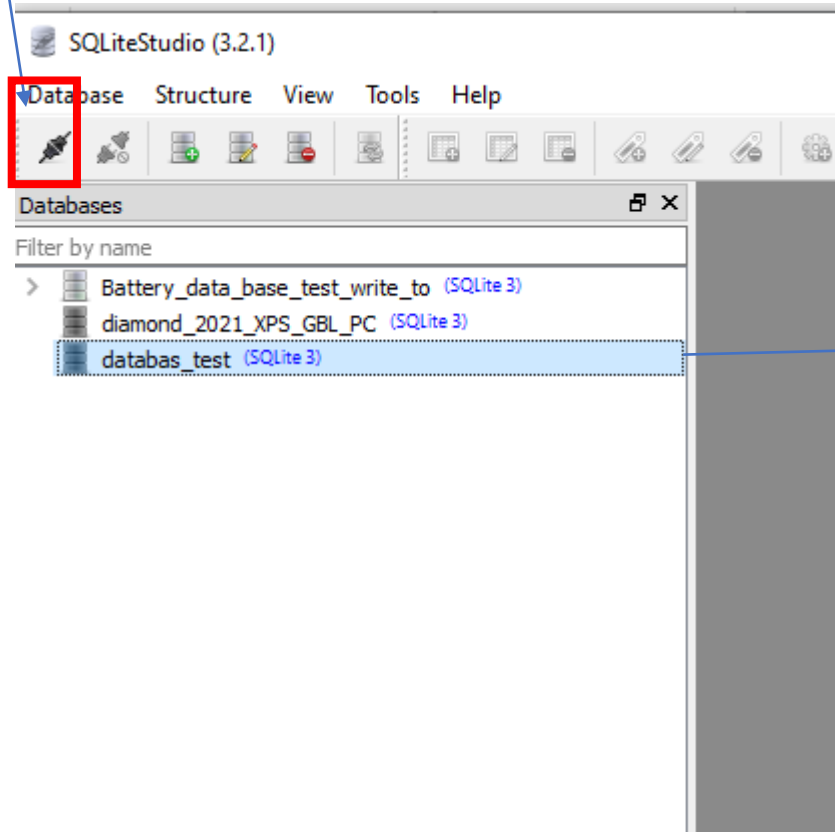
Create a new database

Add existing database

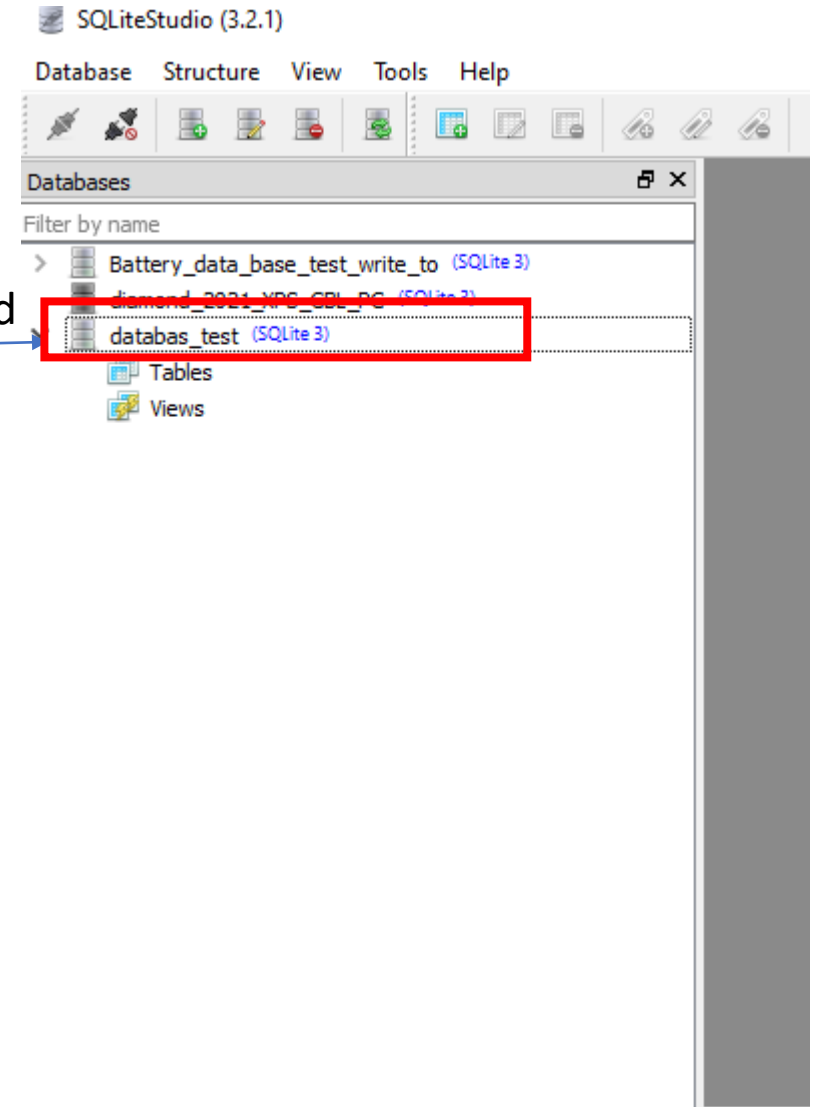


SQLite Studio

Connect to a database



Database connected



Create a table

Each table should be constructed to “maintain integrity”.

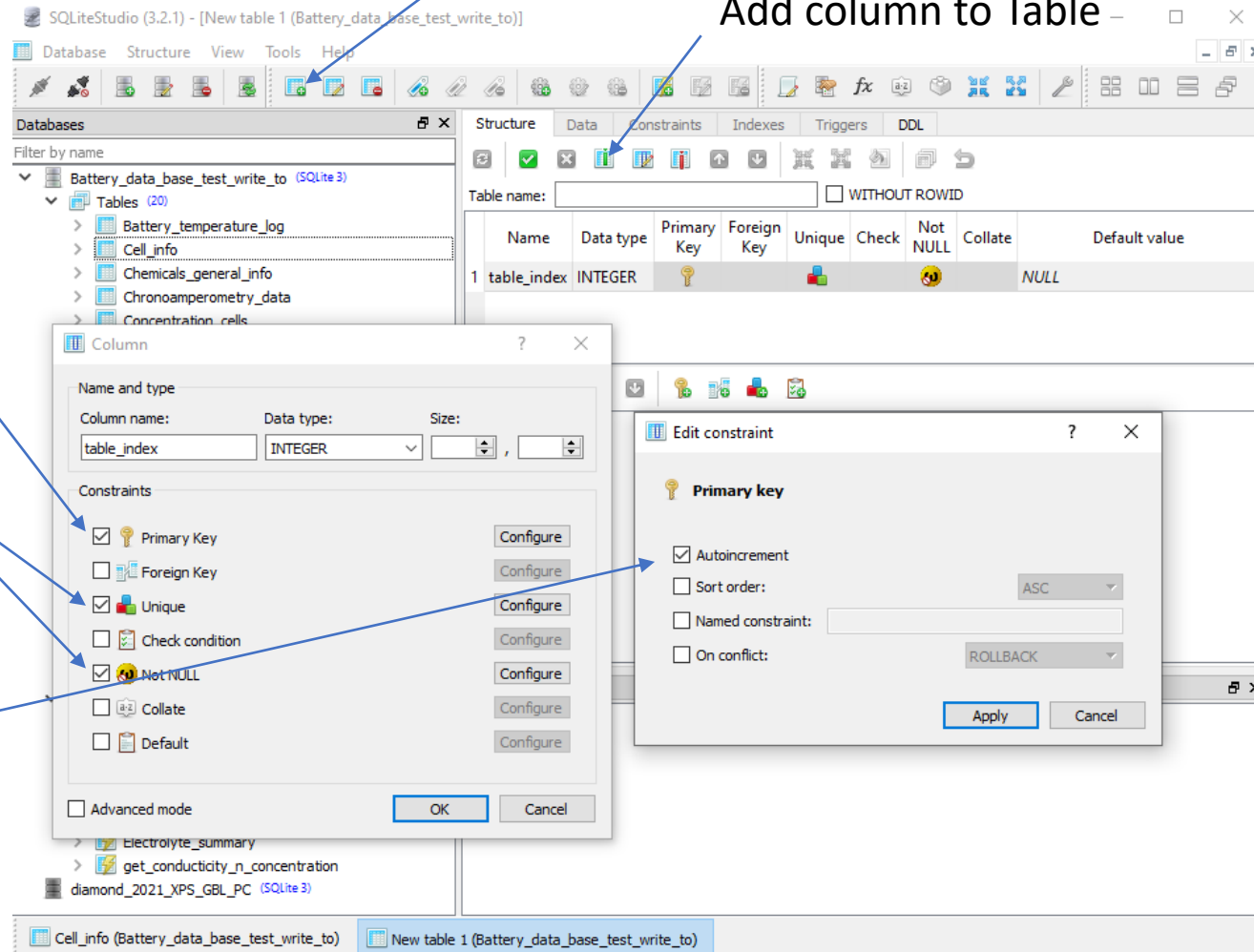
Each table should generally contain one column with a “primary key”.

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).

Add table to selected database

Add column to Table



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

Create a table

Explicitly link tables if a value should be available between tables (maintain integrity).

(When quiring, you can still join tables on non-linked columns)

Add column to Table

The screenshot shows the SQLiteStudio 3.2.1 interface. The 'Databases' pane on the left shows a database named 'Battery_data_base_test_write_to' containing several tables. The 'Structure' pane on the right shows the structure of 'New table 1 (Battery_data_base_test_write_to)', which has two columns: 'table_index' (INTEGER) and 'a column' (DECIMAL). The 'Data' tab is selected, and the 'Table name' field is empty. The 'Table to link' label points to the 'Cell_info' table in the 'Databases' pane. The 'Column' dialog box is open, showing the 'Name and type' section with 'b column' as the column name and 'DECIMAL' as the data type. The 'Constraints' section shows the 'Foreign Key' checkbox checked. The 'Foreign key' dialog box is also open, showing the 'Foreign table' as 'Cell_info' and the 'Foreign column' as 'Cell_ID'. The 'Reactions' section shows 'ON UPDATE' and 'ON DELETE' set to 'NO ACTION' and 'MATCH' set to 'SIMPLE'. The 'Deferred foreign key' section is empty. The 'Named constraint' checkbox is unchecked. The 'Apply' button is highlighted.

Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1 table_index	INTEGER							NULL
2 a column	DECIMAL							

Table to link

Column in other table to link.

Create a table

SQLiteStudio (3.2.1) - [New table 1 (Battery_data_base_test_write_to)]

Database Structure View Tools Help

Databases

Filter by name

Battery_data_base_test_write_to (SQLite 3)

Tables (20)

- Battery_temperature_log
- Cell_info
- Chemicals_general_info
- Chronoamperometry_data
- Concentration_cells
- conductivity
- Conductivity_reference_data
- Controlled_potential_data
- Density
- Diffusion_experiment
- Diffusion_membranes
- Electrochemical_full_run_data
- Electrochemical_measurements
- Electrodes
- Electrolytes
- impedance_data
- Projects
- Specific_chemicals
- Synthesis
- Viscosity

Views (7)

- Active_mass
- average_currents
- coating_info
- Combined_density_and_temperature
- density_viscosity_concentration
- Electrolyte_summary
- get_conductivity_n_concentration

diamond_2021_XPS_GBL_PC (SQLite 3)

Structure Data Constraints Indexes Triggers DDL

Table name: test table ☐ WITHOUT ROWID

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	table_index	INTEGER							NULL
2	a column	DECIMAL							NULL
3	b column	TEXT							NULL

Queries to be executed

```
CREATE TABLE [test table] (  
  table_index INTEGER PRIMARY KEY AUTOINCREMENT  
  UNIQUE  
  NOT NULL,  
  [a column] DECIMAL,  
  [b column] TEXT REFERENCES Cell_info (Cell_ID)  
);
```

☐ Don't show again

Commit 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

0 go to data tab

1 add row

3 commit changes to table

2 fill in data

SQLiteStudio (3.2.1) - [Chemicals_general_info (Battery_data_base_test_write_to)]

Database Structure View Tools Help

Databases

Filter by name

Battery_data_base_test_write_to (SQLite 3)

Tables (20)

- Battery_temperature_log
- Cell_info
- Chemicals_general_info
- Chronoamperometry_data
- Concentration_cells
- conductivity
- Conductivity_reference_data
- Controlled_potential_data
- Density
- Diffusion_experiment
- Diffusion_membranes
- Electrochemical_full_run_data
- Electrochemical_measurements
- Electrodes
- Electrolytes
- impedance_data
- Projects
- Specific_chemicals
- Synthesis
- Viscosity

Views (7)

- Active_mass
- average_currents
- coating_info
- Combined_density_and_temperature
- density_viscosity_concentration

Structure Data Constraints Indexes Triggers DDL

Form view



Filter data Total rows loaded: 30

	Full name	Abbreviat	Molar ma	Density q	Boilinga pc	Point qro	Meltinga 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	Dimethylsulfoxid	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
26	Di(2-methoxyethyl) ether	Diglyme	134.175	0.937	162	NULL	-64	1

Maintaining integrity of database



Make sure that the database remains logical:

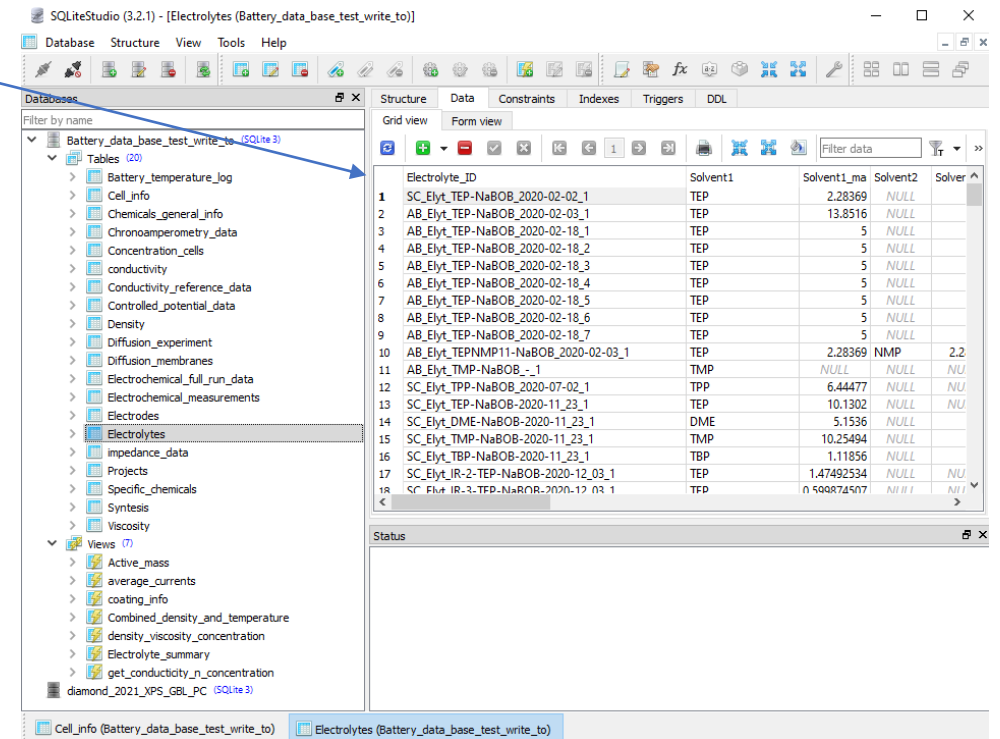
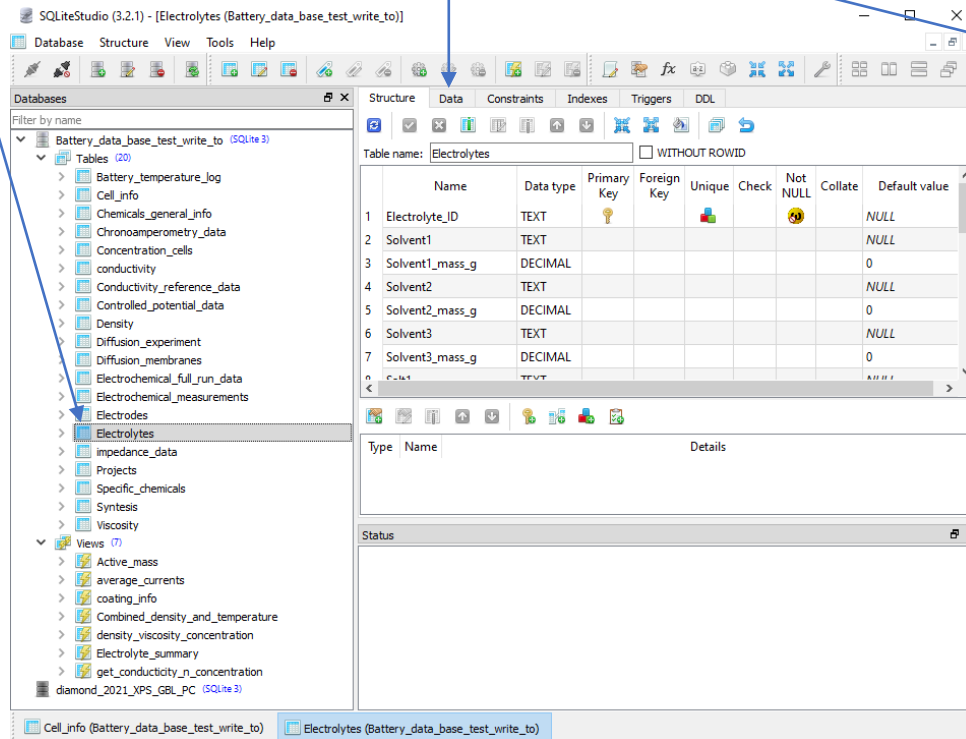
- Rows should to be distinguishable.  Use one (or more) primary key(s) that are (together) unique
- No “dead” ends.  The foreign value should exists in the other table.

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”

SQLiteStudio (3.2.1) - [Electrochemical_full_run_data (Battery_data_base_test_write_to)]

Database Structure View Tools Help

Filter by name

Battery_data_base_test_write_to (SQLite 3)

Tables (20)

- Battery_temperature_log
- Cell_info
- Chemicals_general_info
- Chronoamperometry_data
- Concentration_cells
- conductivity
- Conductivity_reference_data
- Controlled_potential_data
- Density
- Diffusion_experiment
- Diffusion_membranes
- Electrochemical_full_run_data
- Electrochemical_measurements
- Electrodes
- Electrolytes
- Impedance_data
- Projects
- Specific_chemicals
- Synthesis
- Viscosity
- Active_mass
- average_currents
- coating_info
- Combined_density_and_temperature
- density_viscosity_concentration
- Electrolyte_summary
- get_conductivity_concentration
- diamond_2021_IPS_GSL_FC (SQLite 3)

Grid view Form view

Filter data

Total rows loaded: 32811583

table_index	Current_A	Cell_potential	WE_potential	Battery_ID	time_s	Specific_ca	Step_open	Cycle_index	Step_time
1	305679	0	0.5022	NULL	SC_1Chg-DXPS_TEP20_1	0	0	0	0
2	305680	0	0.5044	NULL	SC_1Chg-DXPS_TEP20_1	60	0	0	60
3	305681	0	0.506	NULL	SC_1Chg-DXPS_TEP20_1	120	0	0	120
4	305682	0	0.5078	NULL	SC_1Chg-DXPS_TEP20_1	180	0	0	180
5	305683	0	0.5103	NULL	SC_1Chg-DXPS_TEP20_1	240	0	0	240
6	305684	0	0.5128	NULL	SC_1Chg-DXPS_TEP20_1	300	0	0	300
7	305685	0	0.5143	NULL	SC_1Chg-DXPS_TEP20_1	360	0	0	360
8	305686	0	0.5159	NULL	SC_1Chg-DXPS_TEP20_1	420	0	0	420
9	305687	0	0.5165	NULL	SC_1Chg-DXPS_TEP20_1	480	0	0	480
10	305688	0	0.5187	NULL	SC_1Chg-DXPS_TEP20_1	540	0	0	540
11	305689	0	0.5199	NULL	SC_1Chg-DXPS_TEP20_1	600	0	0	600
12	305690	0	0.5221	NULL	SC_1Chg-DXPS_TEP20_1	660	0	0	660
13	305691	0	0.5236	NULL	SC_1Chg-DXPS_TEP20_1	720	0	0	720
14	305692	0	0.5249	NULL	SC_1Chg-DXPS_TEP20_1	780	0	0	780
15	305693	0	0.5261	NULL	SC_1Chg-DXPS_TEP20_1	840	0	0	840
16	305694	0	0.5273	NULL	SC_1Chg-DXPS_TEP20_1	900	0	0	900
17	305695	0	0.5289	NULL	SC_1Chg-DXPS_TEP20_1	960	0	0	960
18	305696	0	0.5301	NULL	SC_1Chg-DXPS_TEP20_1	1020	0	0	1020
19	305697	0	0.5308	NULL	SC_1Chg-DXPS_TEP20_1	1080	0	0	1080
20	305698	0	0.5332	NULL	SC_1Chg-DXPS_TEP20_1	1140	0	0	1140
21	305699	0	0.5329	NULL	SC_1Chg-DXPS_TEP20_1	1200	0	0	1200
22	305700	0	0.5357	NULL	SC_1Chg-DXPS_TEP20_1	1260	0	0	1260
23	305701	0	0.5363	NULL	SC_1Chg-DXPS_TEP20_1	1320	0	0	1320
24	305702	0	0.537	NULL	SC_1Chg-DXPS_TEP20_1	1380	0	0	1380
25	305703	0	0.5382	NULL	SC_1Chg-DXPS_TEP20_1	1440	0	0	1440
26	305704	0	0.5394	NULL	SC_1Chg-DXPS_TEP20_1	1500	0	0	1500
27	305705	0	0.5413	NULL	SC_1Chg-DXPS_TEP20_1	1560	0	0	1560
28	305706	0	0.5422	NULL	SC_1Chg-DXPS_TEP20_1	1620	0	0	1620
29	305707	0	0.5432	NULL	SC_1Chg-DXPS_TEP20_1	1680	0	0	1680
30	305708	0	0.5444	NULL	SC_1Chg-DXPS_TEP20_1	1740	0	0	1740

SQLiteStudio (3.2.1) - [Electrolytes (Battery_data_base_test_write_to)]

Database Structure View Tools Help

Filter by name

Battery_data_base_test_write_to (SQLite 3)

Tables (20)

- Battery_temperature_log
- Cell_info
- Chemicals_general_info
- Chronoamperometry_data
- Concentration_cells
- conductivity
- Conductivity_reference_data
- Controlled_potential_data
- Density
- Diffusion_experiment
- Diffusion_membranes
- Electrochemical_full_run_data
- Electrochemical_measurements
- Electrodes
- Electrolytes
- Impedance_data
- Projects
- Specific_chemicals
- Synthesis
- Viscosity
- Active_mass
- average_currents
- coating_info
- Combined_density_and_temperature
- density_viscosity_concentration

Grid view Form view

Filter data

Total rows loaded: 405

Electrolyte_ID	Solvent1	Solvent1_ma	Solvent2	Solver
1 SC_Elyt_TEP-NaBOB_2020-02-02_1	TEP	2.28369	NULL	
2 AB_Elyt_TEP-NaBOB_2020-02-03_1	TEP	13.8516	NULL	
3 AB_Elyt_TEP-NaBOB_2020-02-18_1	TEP	5	NULL	
4 AB_Elyt_TEP-NaBOB_2020-02-18_2	TEP	5	NULL	
5 AB_Elyt_TEP-NaBOB_2020-02-18_3	TEP	5	NULL	
6 AB_Elyt_TEP-NaBOB_2020-02-18_4	TEP	5	NULL	
7 AB_Elyt_TEP-NaBOB_2020-02-18_5	TEP	5	NULL	
8 AB_Elyt_TEP-NaBOB_2020-02-18_6	TEP	5	NULL	
9 AB_Elyt_TEP-NaBOB_2020-02-18_7	TEP	5	NULL	
10 AB_Elyt_TEP-NaBOB_2020-02-03_1	TEP	2.28369	NMP	2.2
11 AB_Elyt_TMP-NaBOB_2020-11_23_1	TMP	NULL	NULL	NU
12 SC_Elyt_TPP-NaBOB_2020-07-02_1	TPP	6.44477	NULL	NU
13 SC_Elyt_TPP-NaBOB_2020-11_23_1	TEP	10.1302	NULL	NU
14 SC_Elyt_DME-NaBOB_2020-11_23_1	DME	5.1536	NULL	NU
15 SC_Elyt_TMP-NaBOB_2020-11_23_1	TMP	10.25494	NULL	NU
16 SC_Elyt_TBP-NaBOB_2020-11_23_1	TBP	1.11856	NULL	NU
17 SC_Elyt_IR-2-TEP-NaBOB_2020-12_03_1	TEP	1.47492534	NULL	NU
18 SC_Elyt_IR-2-TEP-NaBOB_2020-12_03_1	TEP	0.98874507	NULL	NU

SQLiteStudio (3.2.1) - [Cell_info (Battery_data_base_test_write_to)]

Database Structure View Tools Help

Filter by name

Battery_data_base_test_write_to (SQLite 3)

Tables (20)

- Battery_temperature_log
- Cell_info
- Chemicals_general_info
- Chronoamperometry_data
- Concentration_cells
- conductivity
- Conductivity_reference_data
- Controlled_potential_data
- Density
- Diffusion_experiment
- Diffusion_membranes
- Electrochemical_full_run_data
- Electrochemical_measurements
- Electrodes

Grid view Form view

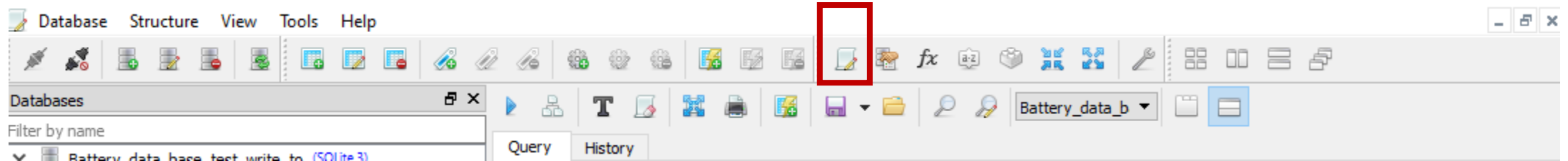
Filter data

Total rows loaded: 405

Cell_ID	Working_Electrode_ID	Working_Electrode_J	Working_E	Counter_Electrode_ID	Counter_Electrode_J	Counter_E	Reference_Electrode	Reference_E	Electrolyte_ID	Electrolyte	Separator	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_1Cyg-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

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

Specify what table
to look in

Limit values to
extract by
constraint

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

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

“*” means that
everything is to be
read

Fast but easy...

The screenshot shows a SQL query interface with a query editor, a query history tab, and a results table. The query editor contains the following SQL query:

```
1 select *
2 from Cell_info
3 where created < '2020 may'
```

The query history tab shows the same query. The results table displays the following data:

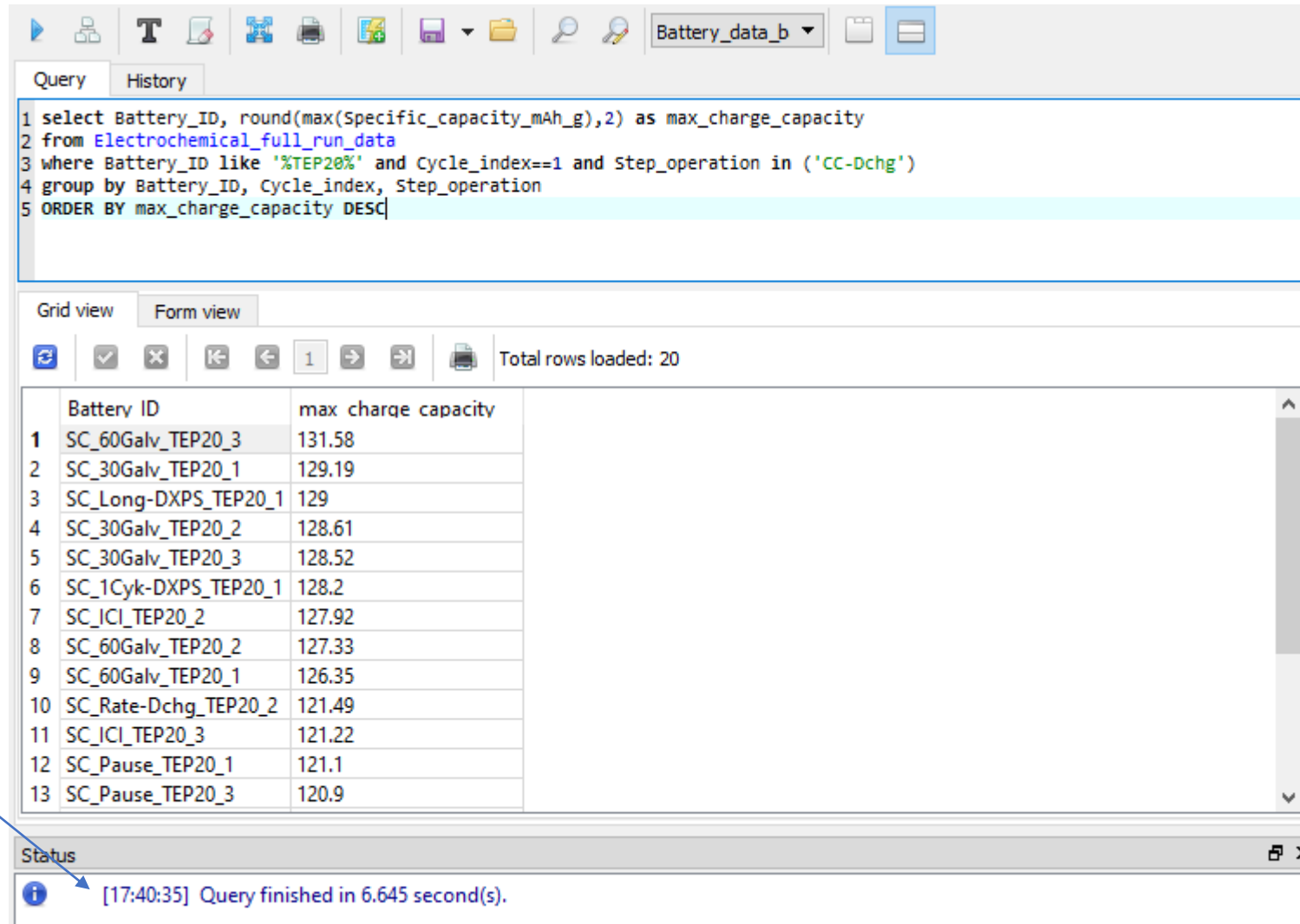
Cell_ID	rode_ID	Counter_Electrode	C
1 LYJ_201912	A	9.37	
2 LYJ_201912	A	9.35	
3 3EL-IC_PC_NONE_SUM19_2	Prussian_blue_SUM19	10.33	13
4 3EL-IC_PC_FEC_SUM19_2	Prussian_blue_SUM19	10.32	13
5 3EL-IC_PC_ES_SUM19_2	Prussian_blue_SUM19	10.41	13
6 3EL-IC_PC_VC_SUM19_2	Prussian_blue_SUM19	10.36	13
7 3EL-IC_PC_TMS_SUM19_2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

The status bar at the bottom indicates: [17:34:14] Query finished in 0.003 second(s).

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 **grouping** and **sorting** the result. Using `pandas.read_excel` this would likely crash your computer or take a very long time...



The screenshot shows a SQL query interface with a toolbar at the top. The query is as follows:

```
1 select Battery_ID, round(max(Specific_capacity_mAh_g),2) as max_charge_capacity
2 from Electrochemical_full_run_data
3 where Battery_ID like '%TEP20%' and Cycle_index==1 and Step_operation in ('CC-Dchg')
4 group by Battery_ID, Cycle_index, Step_operation
5 ORDER BY max_charge_capacity DESC
```

Below the query editor, there are tabs for 'Grid view' and 'Form view'. The 'Grid view' is selected, showing a table with 13 rows and 2 columns: 'Battery ID' and 'max charge capacity'. The table is sorted by 'max charge capacity' in descending order. The status bar at the bottom indicates that the query finished in 6.645 seconds.

	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

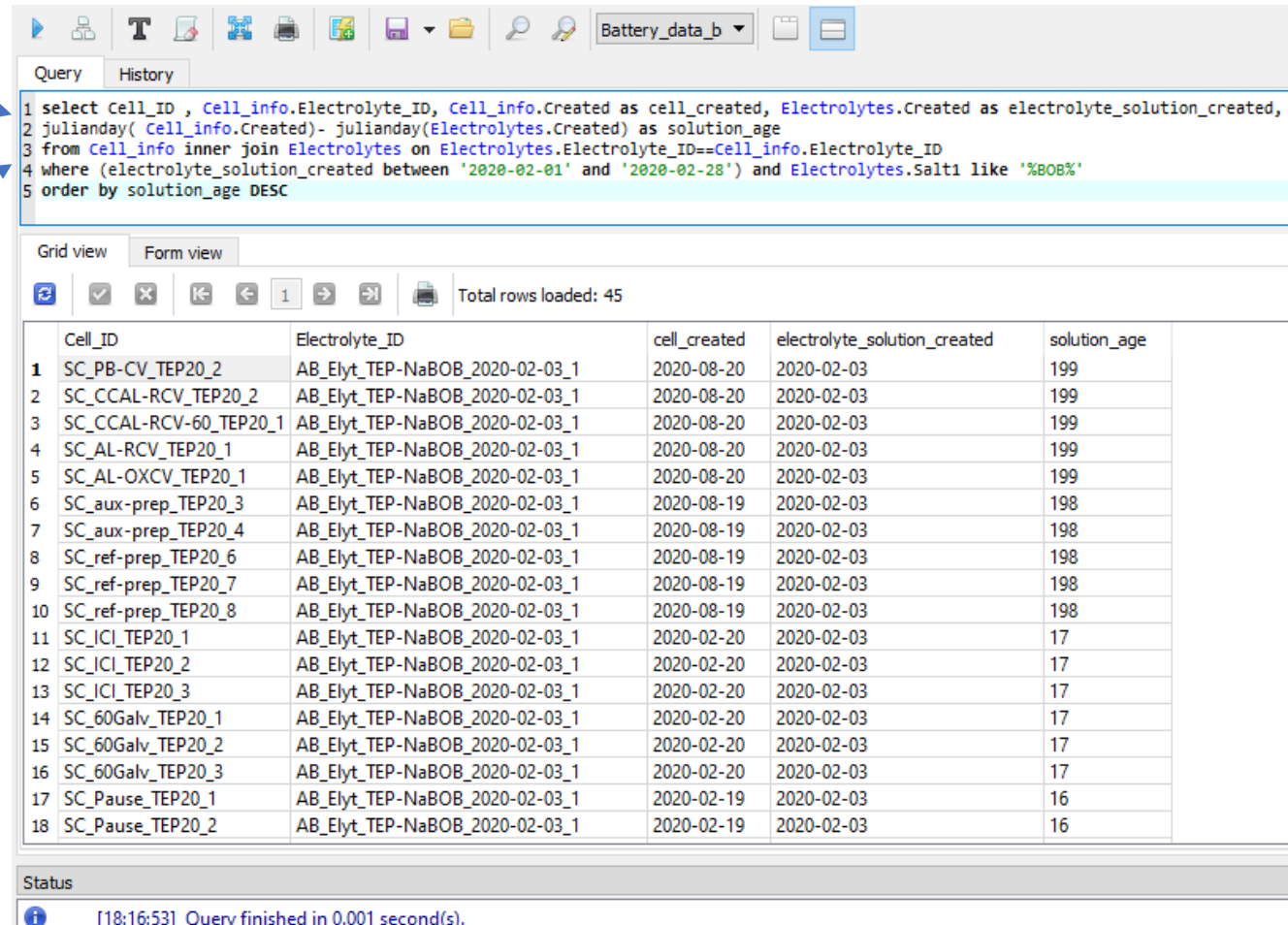
Status: [17:40:35] Query finished in 6.645 second(s).

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
→



The screenshot shows a SQL query editor with a toolbar at the top. The query is as follows:

```
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
```

Below the query editor, there are tabs for 'Grid view' and 'Form view'. The 'Grid view' is selected, showing a table with 5 columns: Cell_ID, Electrolyte_ID, cell_created, electrolyte_solution_created, and solution_age. The table contains 18 rows of data. The status bar at the bottom indicates that the query finished in 0.001 second(s).

	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

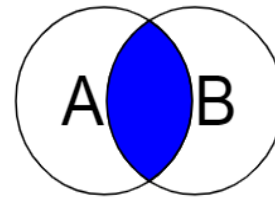
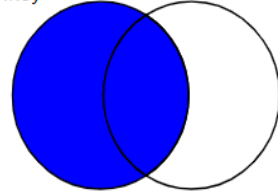
Task:

- 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”.

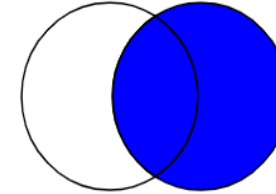
*sqlite3 in python contains more predefined functions than SQLiteStudio

inner join on* → ON A.key = B.key

SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key



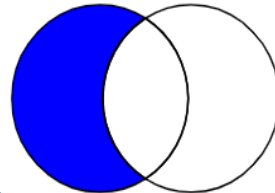
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key



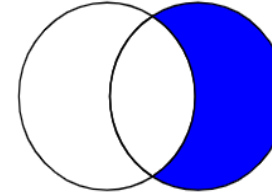
SQL

JOINS

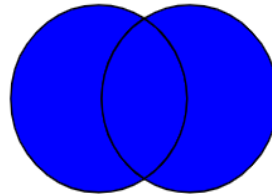
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
WHERE B.key IS NULL



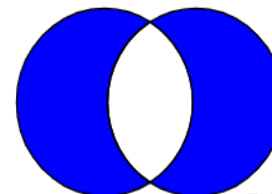
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL



SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key



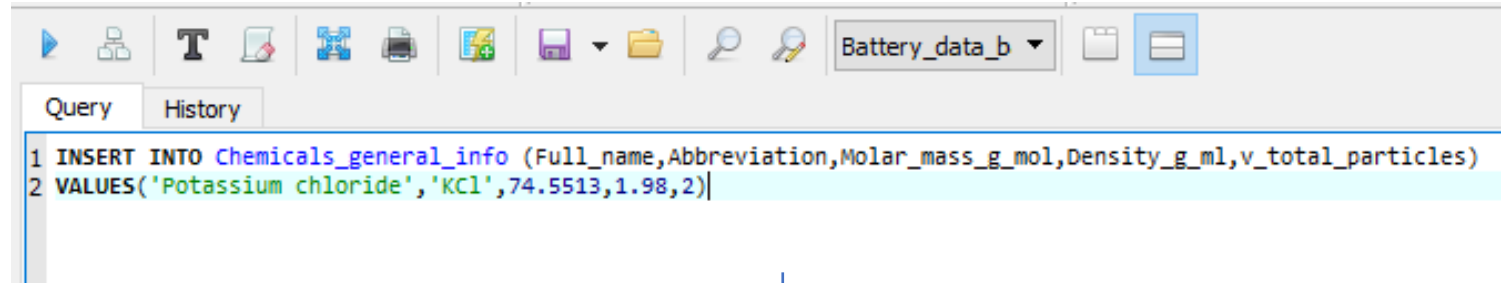
SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL



This work is licensed under a Creative Commons Attribution 3.0 Unported License.
Author: <http://commons.wikimedia.org/wiki/User:Arbeck>

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

Manually writing to SQLite database



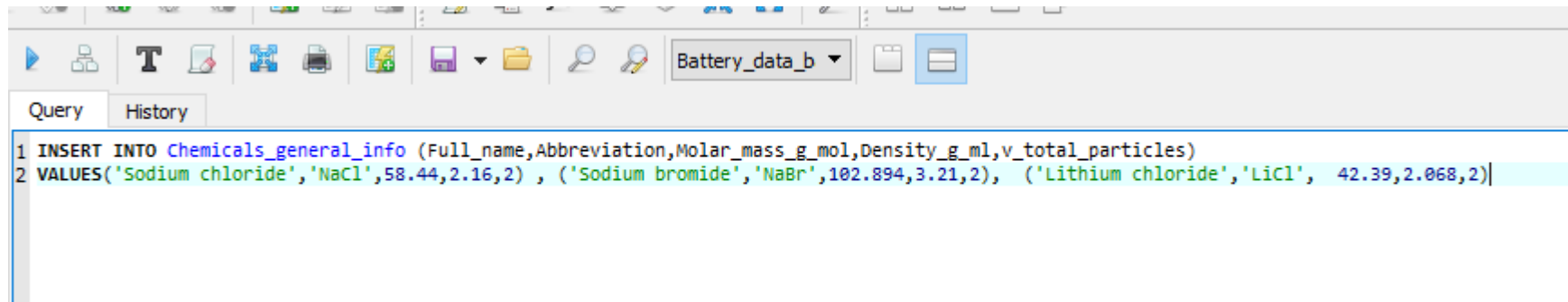
A screenshot of a SQLite query editor interface. The toolbar at the top includes icons for running queries, saving, and other database operations. Below the toolbar, there are tabs for 'Query' and 'History'. The 'Query' tab is active, displaying the following SQL command:

```
1 INSERT INTO Chemicals_general_info (Full_name,Abbreviation,Molar_mass_g_mol,Density_g_ml,v_total_particles)
2 VALUES('Potassium chloride','KCl',74.5513,1.98,2)|
```



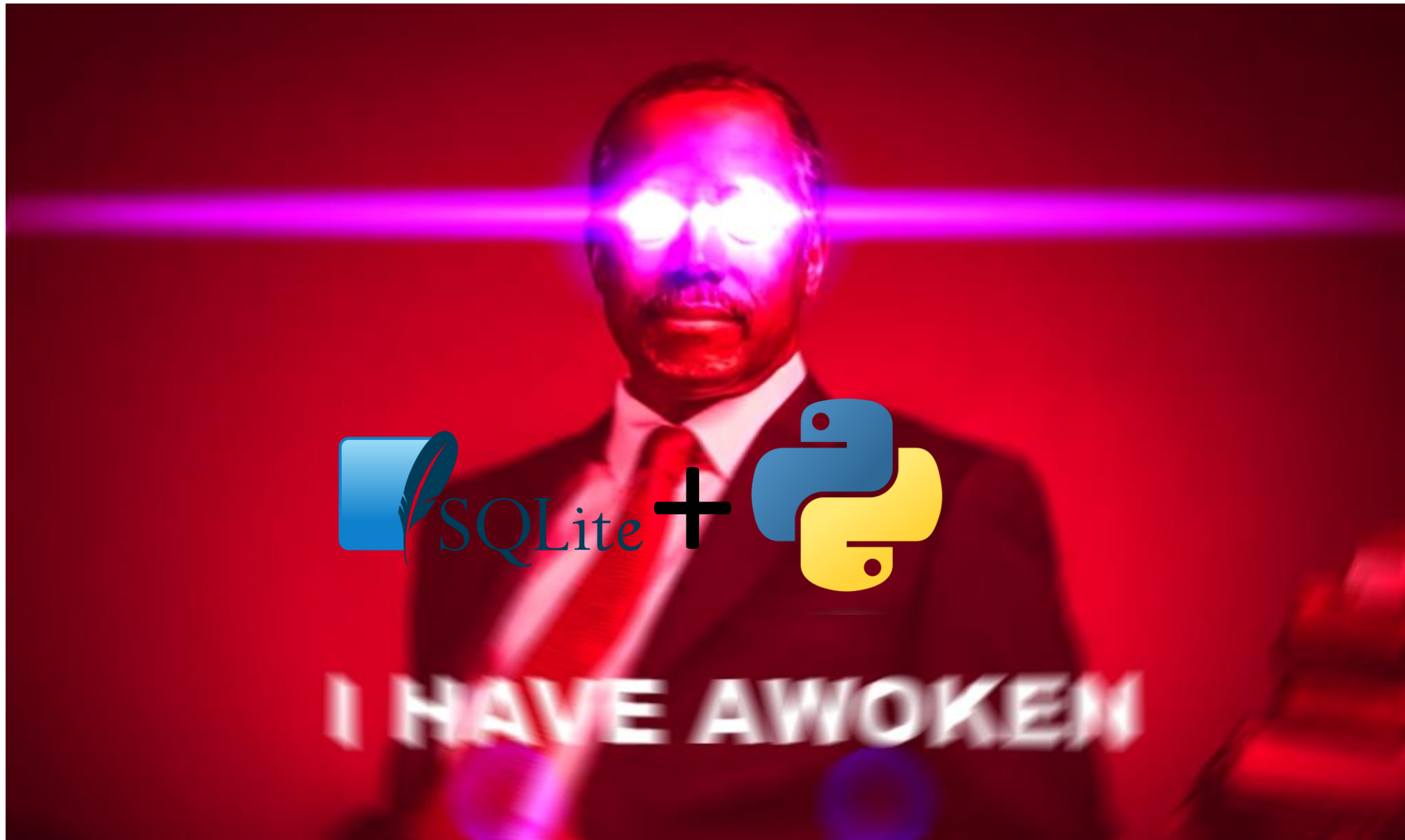
15	Potassium chloride	KCl	74.5513	1.98	NULL	NULL	NULL	2
16	Potassium bis(oxalato)borate	KBOB	225.96	NULL	NULL	NULL	NULL	2
17	N-Methyl-2-pyrrolidone	NMP	99.133	1.028	202	NULL	-24	1

creating multiple rows in one command



A screenshot of a SQLite query editor interface, similar to the one above. The 'Query' tab is active, displaying the following SQL command:

```
1 INSERT INTO Chemicals_general_info (Full_name,Abbreviation,Molar_mass_g_mol,Density_g_ml,v_total_particles)
2 VALUES('Sodium chloride','NaCl',58.44,2.16,2) , ('Sodium bromide','NaBr',102.894,3.21,2), ('Lithium chloride','LiCl', 42.39,2.068,2)|
```



<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))
```

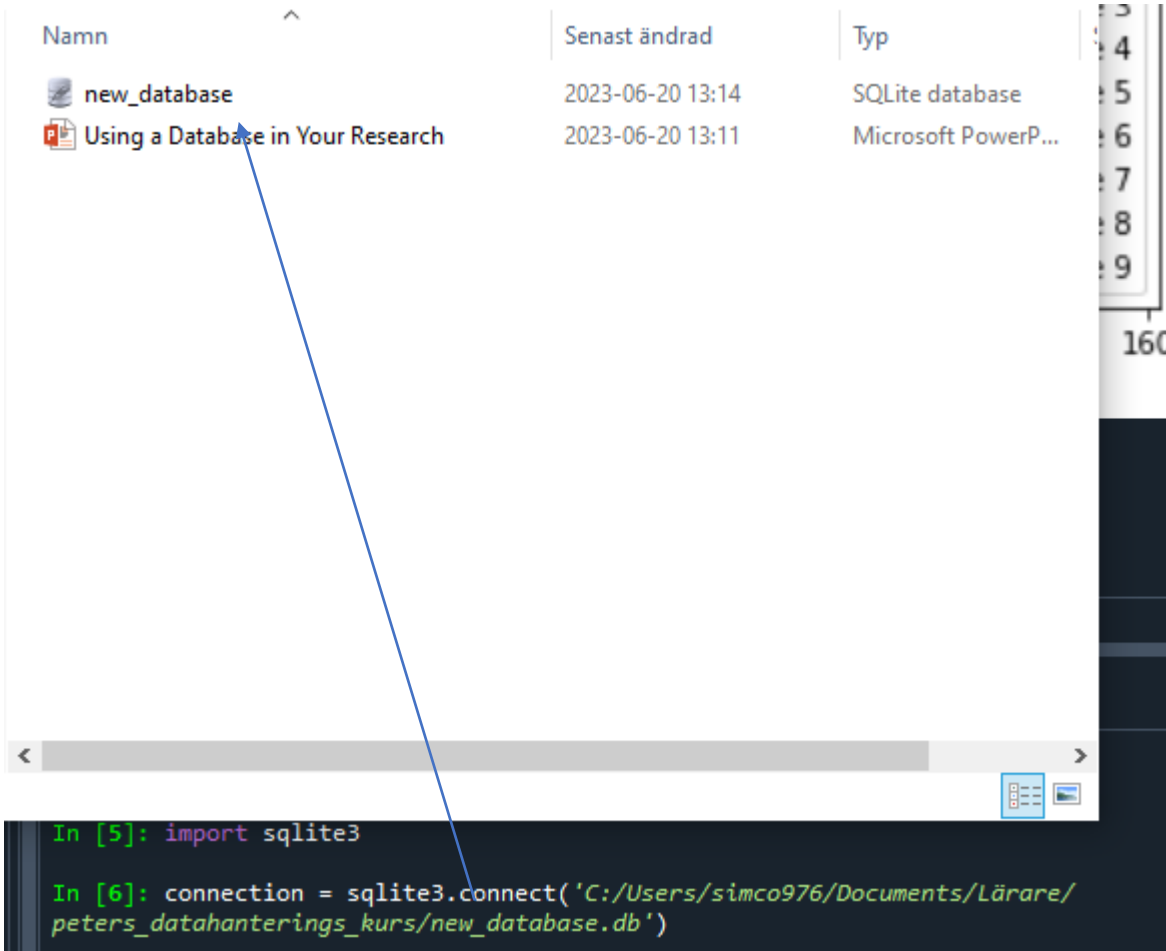
```
select Battery_ID,Cell_potential_V,WE_potential_V,(case when WE_potential_V is not NULL then  
WE_potential_V-Cell_potential_V end) as  
CE_potential_V,Specific_capacity_mAh_g,Cycle_index,Step_operation from  
Electrochemical_full_run_data  
Where (Battery_ID in ('SC_Rate-chg_TEP20_1','SC_Rate-chg_TEP20_2','SC_Rate-  
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 it 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.


```

def get_conductivity_and_viscosity_data(electrolytes=[],solvents=[],additives='NULL'):

    es_ids=get_electrolyte_IDs_with_composition(electrolytes,solvents,additives)

    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]][y]['mole_fraction'] for y in es_info[x[0]]['Electrolytes']])
        molal_c_list+=sum([es_info[x[0]][y]['c_Molal'] for y in es_info[x[0]]['Electrolytes']])
        moles_electrolyte=sum([es_info[x[0]][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','Additives']:
            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'])

```

```
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.

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.

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

```
19
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}
43
```

Live demo?