SQL functionality in Albion Dependent Software

Sequel Query Language (SQL) is a database is a standard language for accessing and manipulating database and the data in database tables. In Albion we have exposed the power of SQL language to the end user, so that they can more readily and easily manipulated data in tables (layers). SQL allows for user to do reporting, cross correlate data across multiple tables for example: vlookup - find records found in or not in another table. SQL language in Albion can be used across all existing support data sources, shape files, sqlite db, avdb, css

The SQL language specification that is implemented in Albion is sqlite3 the documentation for this language can be found here at: http://www.sqlite.org/lang.html. The sections of the language that Albion supports are the following:

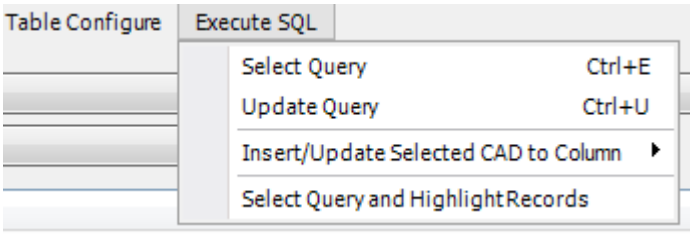
- SELECT - Create a new table with a copy of the information your query has requested.
- UPDATE - Update an original table
- INSERT (INSERT INTO) – allows appending or records from on table to another
- DELETE
- Expression – logical if statements, calculations, math, string manipulation, sub query's
- Aggregate functions - ability to do sum, total average, reporting.

Many SQL examples can be found on the web, but we would refer you to the following source to learn the whole language that is reasonable simple and straight forward and a tool.

- http://www.w3schools.com/sql/default.asp

Different Modes that SQL can be used for in Albion

These modes can be found on the Execute SQL menu.



- Select Query – Creates a new table with a copy of the data that has been retrieved and manipulated through SQL from existing tables.
    - Reporting Sum, AVG, COUNT
    - Retrieve unique (distinct) list of items in a set of columns.
- Update Query – Make Changes to existing tables data using SQL
    - Update …
    - INSERT INTO ..
    - DELETE..

- Select Query and Highlight – User SQL statement to retrieve a set of records to highlight.
    - This is similar to the existing SQL found at the bottom of the grid in image below.
    - The difference is that it allows you the ability to write full SQL statement and not just the part after a 'WHERE' clauses
    - This means the SQL statement can be based on multiple tables of data. The most common form and use is to highlight records, using what excel refers to as a vlookup, where the columns value of the current table is found or not found in another table column (list).

Pointers:

The equivalent 'if' statement in excel is:

CASE col WHEN value THEN 1 WHEN value THEN 2 ELSE 3 END

Or

CASE WHEN condition THEN 1 WHEN condition THEN 2 ELSE 3 END

Additional External Functions

acos, asin, atan, atn2, atan2, acosh, asinh, atanh, difference, degrees, radians, cos, sin, tan, cot, cosh, sinh, tanh, coth, exp, log, log10, power, sign, sqrt, square, ceil, floor, pi. String: replicate, charindex, leftstr, rightstr, ltrim, rtrim, trim, replace, reverse, proper, padl, padr, padc, strfilter.

Aggregate:

stdev, variance, , median, lower_quartile, upper_quartile

Examples of SQL Statements

These are a set of the more common statements that will be used by Albion operators.

Reporting features:

Reporting feature make use of a select query that features aggregate functions and group by , having clauses. The following queries are based on the data set in test_diameters.sqlite. These exact SQL query statements can be found in the SQL_QUERY table in the database. They can be executed thought the 'Select Query' menu.

Select [TYPE], Count(*) as number_of_pipes, SUM([diameter]) as total_length, avg([diameter]) as average_length From [Pipes]

PVC 411 84666 206

Select [TYPE], Count(*) as number_of_pipes, SUM([diameter]) as total_length, avg([diameter]) as average_length From [Pipes] GROUP BY [TYPE]

Type	Number of Pipes	Total Length	Average Length
ABS	82	16933	206.5
CEMT	82	16851	205.5
HPVC	82	17015	207.5
POLY	82	16769	204.5
PVC	83	17098	206

Select [TYPE], Count(*) as number_of_pipes, SUM([diameter]) as total_length, avg([diameter]) as average_length From [Pipes] GROUP BY [TYPE]

HAVING total_length < 17000

Type	Number of Pipes	Total Length	Average Length
ABS	82	16933	206.5
CEMT	82	16851	205.5
POLY	82	16769	204.5

Having clauses can be used to filter the results of the aggregate function columns, those columns that feature Count, Sum, AVG. This is clearly visible in the difference between the queries.

Select d.*, Count(*) as number_of_pipes, SUM([diameter]) as total_length, avg([diameter]) as average_length From Diameters as d inner join Pipes as p on (p.diameter <= d.less and p.diameter > d.more) group by d.[Field1]

Cat	Less	More	Number of Pipes	Total Length	Average Length
Diameter Cat 100	100	0	100	5050	50.5
Diameter Cat 200	200	100	100	15050	150.5
Diameter Cat 300	300	200	100	25050	250.5
Diameter Cat 400	400	300	100	35050	350.5
Diameter Cat 500	500	400	11	4466	406

Select d.*,[TYPE], Count(*) as number_of_pipes, SUM([diameter]) as total_length, avg([diameter]) as average_length From Diameters as d inner join Pipes as p on (p.diameter <= d.less and p.diameter > d.more) group by d.[Field1], [TYPE]

Cat	Less	More	TYPE	Number of Pipes	Total Length	Average Length
Diameter Cat 100	100	0	ABS	20	1030	51.5
Diameter Cat 100	100	0	CEMT	20	1010	50.5
Diameter Cat 100	100	0	HPVC	20	1050	52.5
Diameter Cat 100	100	0	POLY	20	990	49.5
Diameter Cat 100	100	0	PVC	20	970	48.5
Diameter Cat 200	200	100	ABS	20	3030	151.5
Diameter Cat 200	200	100	CEMT	20	3010	150.5
Diameter Cat 200	200	100	HPVC	20	3050	152.5
Diameter Cat 200	200	100	POLY	20	2990	149.5
Diameter Cat 200	200	100	PVC	20	2970	148.5
Diameter Cat 300	300	200	ABS	20	5030	251.5
Diameter Cat 300	300	200	CEMT	20	5010	250.5
Diameter Cat 300	300	200	HPVC	20	5050	252.5
Diameter Cat 300	300	200	POLY	20	4990	249.5
Diameter Cat 300	300	200	PVC	20	4970	248.5
Diameter Cat 400	400	300	ABS	20	7030	351.5
Diameter Cat 400	400	300	CEMT	20	7010	350.5
Diameter Cat 400	400	300	HPVC	20	7050	352.5
Diameter Cat 400	400	300	POLY	20	6990	349.5
Diameter Cat 400	400	300	PVC	20	6970	348.5
Diameter Cat 500	500	400	ABS	2	813	406.5
Diameter Cat 500	500	400	CEMT	2	811	405.5
Diameter Cat 500	500	400	HPVC	2	815	407.5
Diameter Cat 500	500	400	POLY	2	809	404.5
Diameter Cat 500	500	400	PVC	3	1218	406

Alternative to quickly getting evenly grouped summaries reports.

Say for instant you want to generate a report just like above, but you need a quick method to

get the count of how many diameters that are form an evenly spaced ranged, there is simple solution which is very quick. So say we wanted categories starting at 0 an being group by every 100 then this is how you would write the report.

Select (ceil([Diameters]/100) * 100) as diam_cat_Less_equal, count(*) as num_of_pipes , avg([diameter]) as average_length from [Pipes] group by (ceil([Diameters]/100) * 100)

Select Query and Highlight Records

Make sure that you have the layer on which you want records, to be selected on from a resulting SQL query that is open in the grid. Then click the menu item from the 'Select Query and Highlight Records' found under the 'Execute SQL' menu.

For this to work and to highlight the right records the SQL statement must return the

'rowid' column from the current table you are viewing in the grid, as can be seen in the statement below.

To run these two examples the data set can be found in 'Select_where_in_example.sqlite'

Open the Grid and set the layer to 'BackColors'

- **Lookup columns found in another table:**

Select rowid From BackColors where col1 in (Select col_in From table_in where col_in is not null)

- **Lookup columns not found in another table.**

Select rowid From BackColors where col1 not in (Select col_in From table_in ***where col_in is not null***)

> Very import in the sub-query the SQL statement in bold, italics, red is the part where the column on which the lookup is to be done, must exclude nulls. Reason a NULL will match anything, thus it will return no records to highlight.
>
> If you imagine null as a set and null means the set is empty (nothing) then what is the not or opposite NULL then? Not of empty is full and the not of nothing is everything. This is the reason why you have to exclude nulls from the subquery as you noting the result which in tern then is everything.

This is a better method to use to save a sub selection of data, than to create a new table using Select Query.