A query consists of one or more conditions and optionally one aggregate function on result set. Conditions can be of various kinds:

a) Requiring linear search
b) Lookup in indexed column
c) Lookup in string enum
d) ListView filter (search only in rows in a given ListView – for sub queries)

conditions

- greater
- less
- equal
- not_equal

Finally we have aggregates

- find first
- find all
- count
- sum
- max
- min
- average

start end range

OR nodes and SubTable.

Assume we have 3 conditions of types b (indexed lookup), d (ListView filter) and a (linear search). Row 0 is left and the dots are matches:

The query engine “travels” from left to right in a single pass, inside a loop belonging to the condition it currently finds is fastest. It jumps frequently between the condition loops.

We must first note that a linear searches and indexed lookups can outperform eachother mutually, depending on match distance and bitwidth. Especially, an indexed search that matches row 1, 11, 12, 14, 25, 30, 54 is outperformed by a linear boolean search that can test row 0 - 63 in one operation.

So we introduce two statistics variables for each condition:

float m_dD:
Average distance (in rowcount) between matches around current position

float m_dT:
Time (in arbitrary units) it takes to test row n + 1 for a match if row n has just been tested. It depends on the condition kind:

Linear search:	`m_dT = (bitwidth == 0 ? 1.0 / MAX_LIST_SIZE : bitwidth / 8.0)`
Index, ListView and Enum:	`m_dT = 0`

We also define a function for each condition:

float cost():	`(16.0 / m_dD) + m_dT`

The **cost** function is an expression of the **average time spent per table row** when inside the given condition loop. Examples:

Kind	**Match distance**	**Bit width**	**cost**
Linear search	64	1	0.1875
Index/ListView/Enum	64	any	0.1875
Linear search	20	4	1.3
Linear search	infinity (no matches)	0	0.001
Index/ListView/Enum	15	any	1.14
Index/ListView/Enum	100	any	0.16