Database Systems:

1. Aquery
2. SQLite

Rules of thumb:

1. Selecting just the needed columns
2. Leveraging clustered index
   1. Clustered index on a column is when neighboring cells have similar column values

Data distributions:

1. Fractal Distribution
2. Uniform Distribution

**Query execution time:**

**Aquery**

|  |  |  |
| --- | --- | --- |
|  | Fractal Distribution | Uniform Distribution |
| Select all columns | 4.96 ms | 2.92 ms |
| Select needed columns | 2.09ms | 1.72 ms |
| Without clustered indexing | 14.4 ms | 15.8 ms |
| With clustered indexing | 14.7 ms | 15.8 ms |

**SQLite**

|  |  |  |
| --- | --- | --- |
|  | Fractal Distribution | Uniform Distribution |
| Select all columns | 1.625 s | 1.655 s |
| Select needed columns | 1.200 s | 1.196 s |
| Without clustered indexing | 1.082 s | 1.082 s |
| With clustered indexing | 1.029 s | 1.030 s |

*Selecting just the needed columns*

The query to select just the needed columns on both distributions in both database systems yielded a shorter execution time than selecting all columns in the table.

Regardless of the distribution of the data, this rule of thumb should be satisfied because it is faster to fetch fewer data since it requires fewer disk accesses.

*Clustered indexing*

Since the query wanted to select stocks with price > 2500, the clustered index was built on the price column.

In aquery, the indexing was achieved by creating a table that is sorted on price and then calling the query

SELECT stock FROM sorted\_table WHERE price > 2500

On the table with indexing.

Similarly, in sqlite a table was created where the data is sorted on price and the same query to fetch stocks with price > 2500 was executed.

In aquery, indexing gave no significant improvement for uniformly distributed data and gave a worse performance for fractally distributed data.

In sqlite, indexing did not yield significant improvement for both uniformly distributed and fractally distributed data.