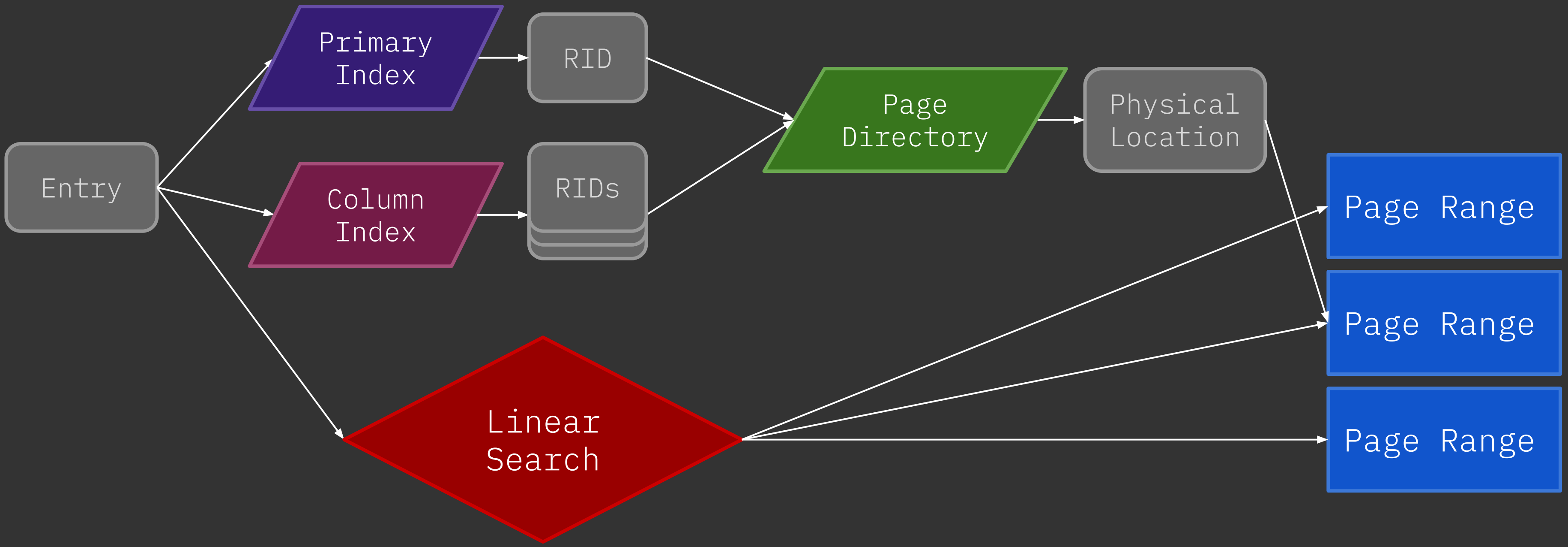
MangoDB

Jared G.
Dominic Q.
Abhi S.
Haskell M.
Noah K.

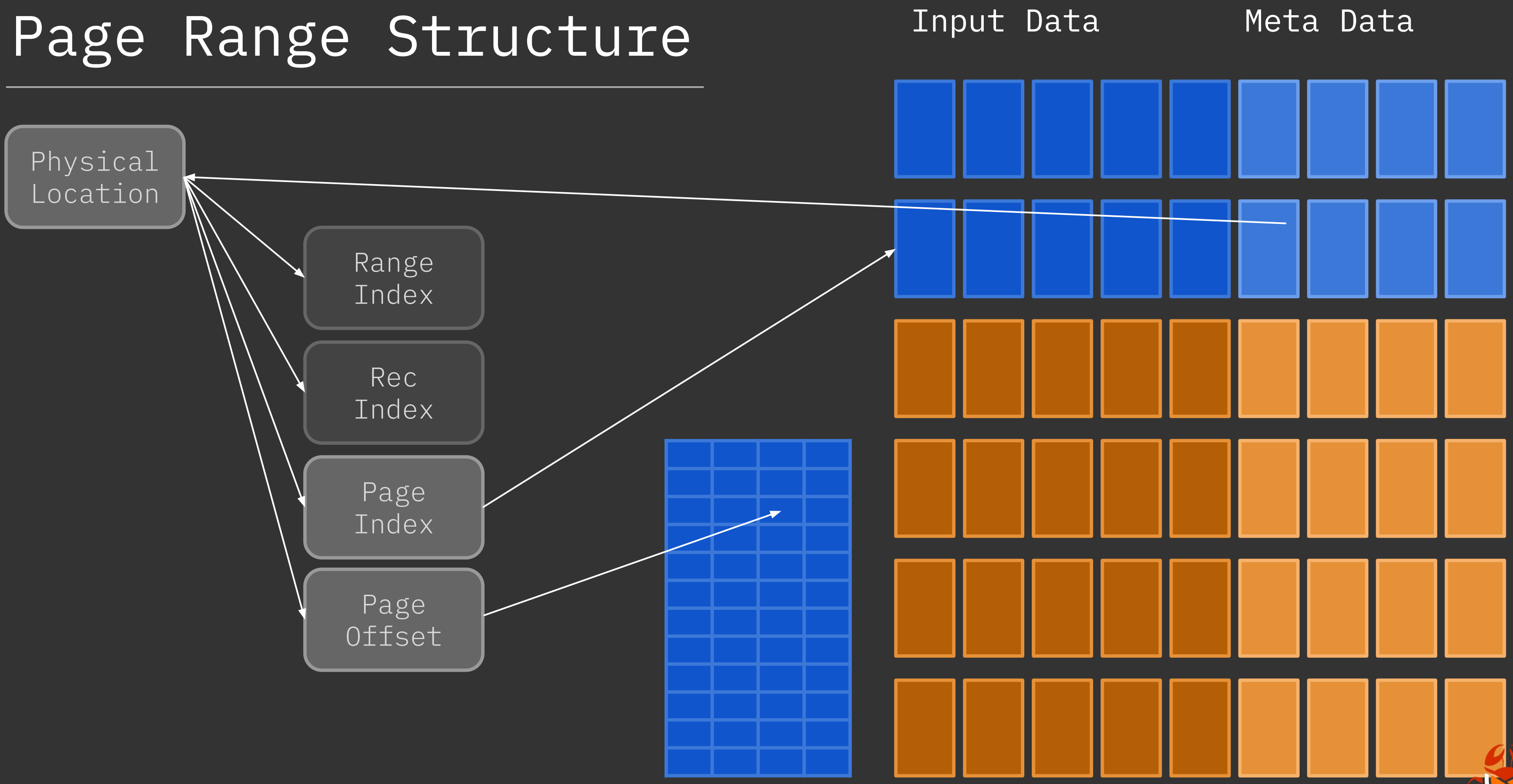
Why Rust?

- Fast interface with CPython
- Memory, thread, and type safety
- Libraries (B-Tree, HashMap, Rayon “parallel iterators”)
- High level features (algebraic data types, Cargo)
- Low level performance (low overhead, no garbage collector)

Table - Lookup Pipeline



Page Range Structure



Input Data

Meta Data

Physical Location

Range Index

Rec Index

Page Index

Page Offset

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 1000 1000 0010 1000

Physical Record Locations

Base Record Location

0000 1000 0000 1000 0000 1000 0000

range index

page index

page offset

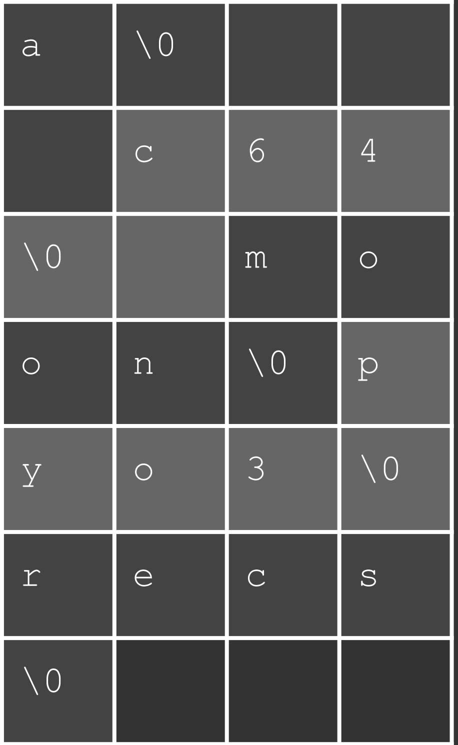
Tail Record Location

0011 0100 1100 0001 0000 0110 1000

page index

page offset

Physical Location in Unaligned Columns



a	\0		
	c	6	4
\0		m	o
o	n	\0	p
y	o	3	\0
r	e	c	s
\0			

record index: record location without range index

column bytes: bytes per attribute in that column

- Uses the record index, the column bytes, and the page size in bytes.
- Calculates the page index and page offset of the unaligned attribute.

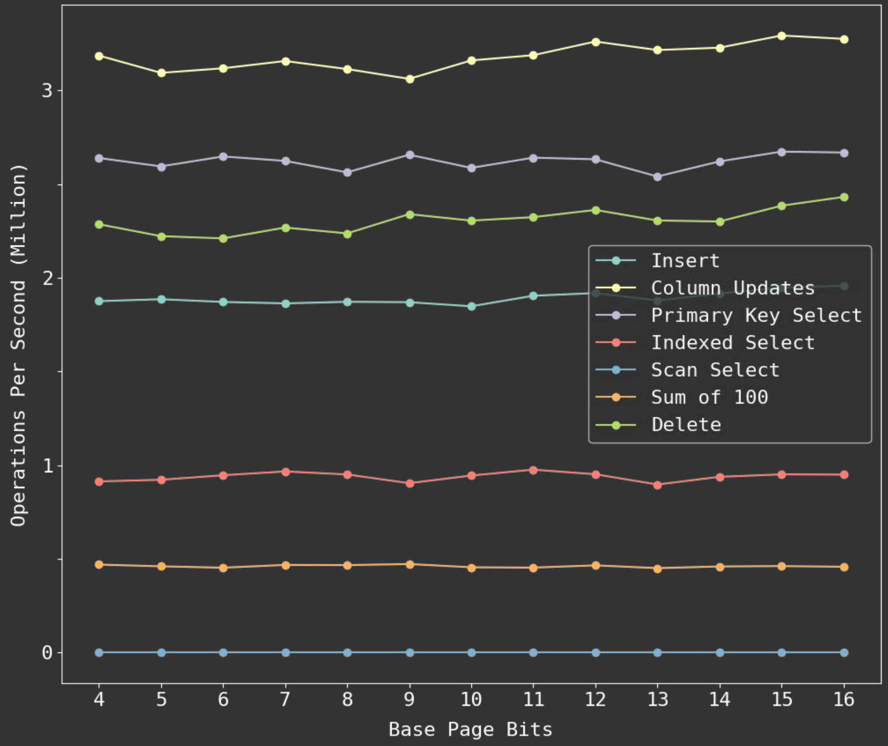
Cumulative updates

LOC	A	B	C	D	Indir	RID	Schema Encoding	Time Stamp	Operation
512			7		8	321	0b0010	00:02	UPDATE 321 (C:7)
.	.	.	.	.	.	.	.	.	...
620		3	7		512	321	0b0110	00:20	UPDATE 321 (B:3)
621	1	3	3	7	620	321	0b1111	01:02	UPDATE 321 (A:1,C:3,D:7)

Later in the Future...

621	1	3	3	7	620	-1	0b1111	01:02	DELETE 321

Benchmarking

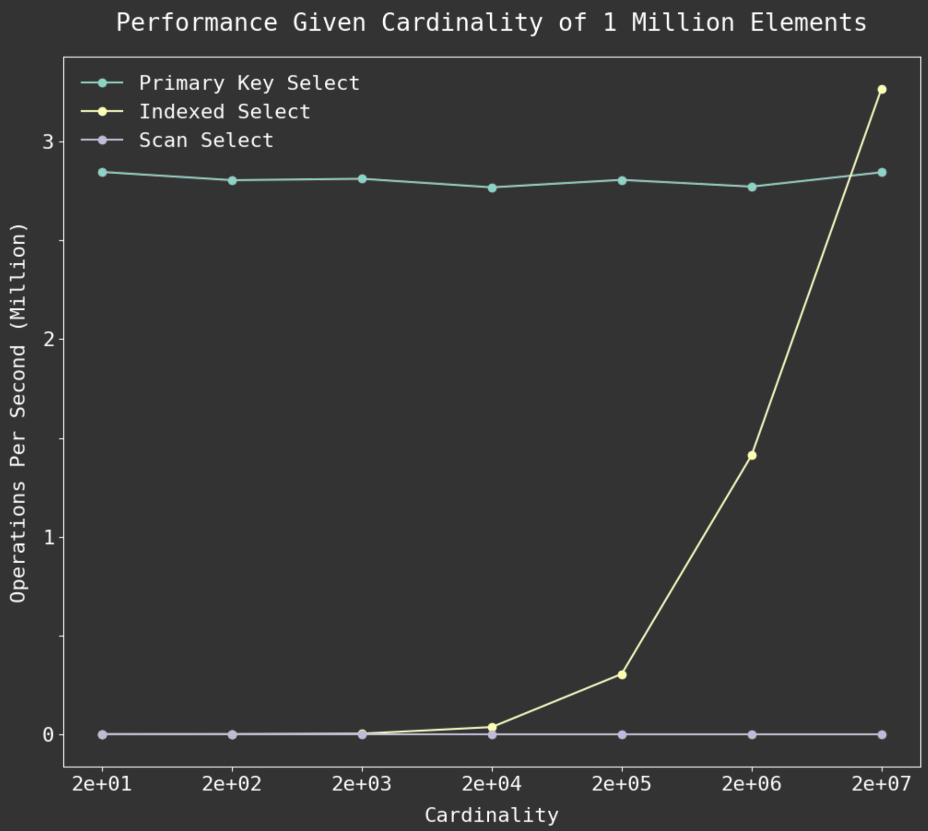


Benchmark DB with varying page range sizes

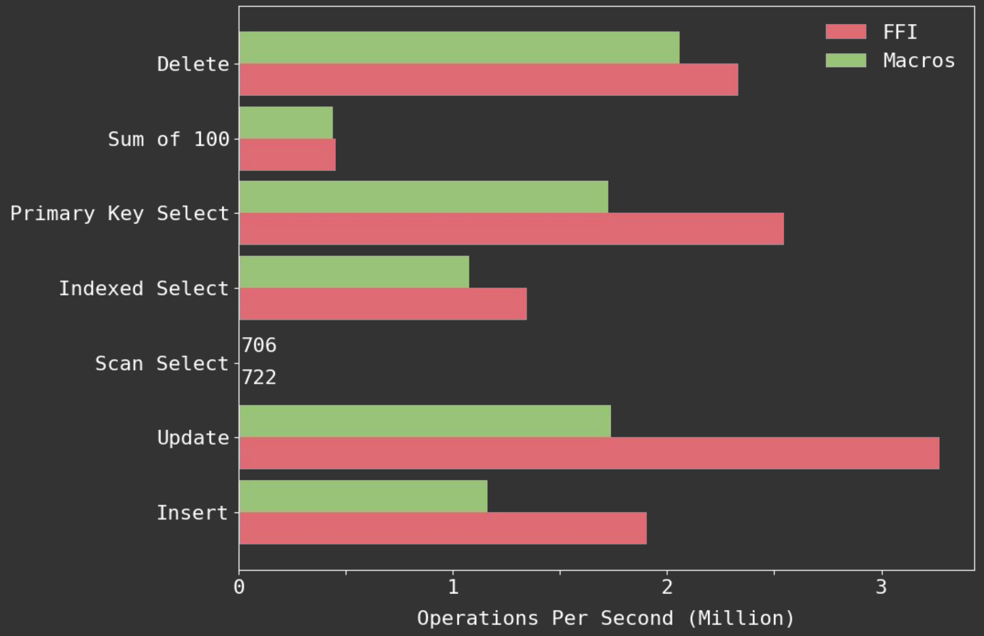
- Graphed performance of the database using different capacities of page range base pages.
- Aggregated the average record/ms for insert, update, delete, and key select.

Indexed Search Performance

- Question: How much does performance improve by increasing index range?
- Generated performance tests to determine the rate of our indexed select operation
- Used hashmap indexing
- Performance gains exponentially
- The outputs depend on the index range size, from 20 to 20 million



Python to Rust PyO3 Macros vs FFI



The PyO3 library provided macro annotations to allow python everything into a dynamic library that could be imported as a Python module.

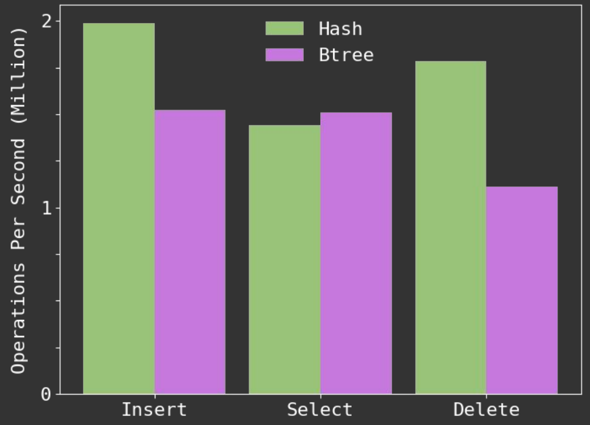
Using the PyO3’s foreign function interface declarations of Python’s C API gave us bindings to the Python Interpreter

B-Trees vs Hash Maps

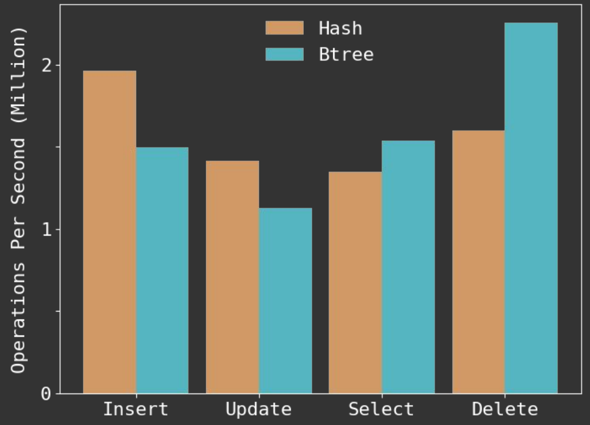
B-Tree for the primary key because the ordering allowed us to easily sum ranges.

HashMap is better for simpler test cases (updating indexed column) it outperformed the B-Tree.

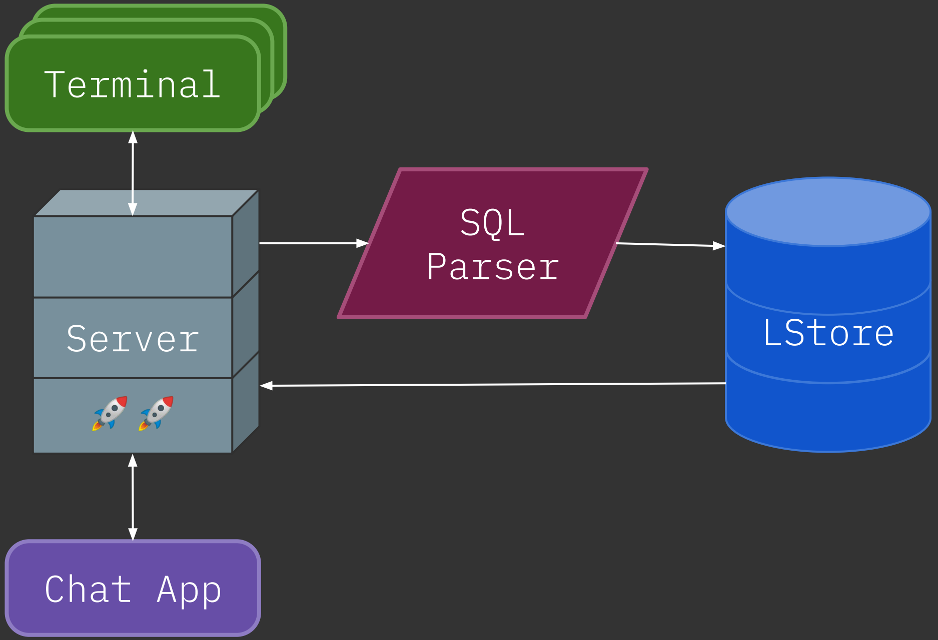
No indexed col updates:



Indexed col updates:



Extended Features



- **Terminal** enables granular troubleshooting
- **Server** provides an HTTP interface to L-store.
- **Chat App** presents practical application of our database

SQL Query Parser

Parses real SQL statements.

Written using parser combinators.

Examples:

```
CREATE TABLE chat (
   _id varchar(1),
   channel varchar(1),
   sender varchar(1),
   message varchar(1)
);
```

```
INSERT INTO chat
    (message_id, channel,
    sender, message)
VALUES
    ('a', 'b', 'c', 'd');
```

```
SELECT * FROM chat
WHERE sender = 'Abhi';
```

Parser Combinators

A parsing technique where you compose complex parsers from simple parsers.

Simple parser: “varchar(12)”

```
let varchar = keyword("varchar")
                .then(int())
                .delimited_by("(",")");
```

Complex parser: Full SQL queries

```
let query = create_query()
              .or(select_query())
              .or(update_query())
              .or(delete_query())
              .or(insert_query())
              ...
```

Parser Combinators - Error Handling

```
CREATE TABLE test_table (message_id varchar(15) channel varchar(25));
> found 'c' but ',' was expected.
```

```
CREATE TABLE test_table [message_id varchar(15), channel varchar(15)];
> found '[' but '(' was expected.
```

```
CREATE TABLE test_table (message_id varchar, channel varchar(25));
> found ',' but ‘(‘ was expected.
```

db-server

```
async function query(query) {
    let req = await fetch("/query/" + encodeURIComponent(query));
    let text = await req.text();
    return { value: text, status: req.status };
}

await query("CREATE TABLE chat (
               _id varchar(1),
               channel varchar(1),
               sender varchar(1),
               text varchar(1)
         );");
await query("INSERT INTO chat
               (message_id, channel, sender, message)
               VALUES
               ('a', 'b', 'c', 'd');");
await query("SELECT * FROM chat WHERE message_id = 'a';")
```

```
{
 "value": "[[“a”,”b”,”c”,"d"]]",
 "status": 200
}
```

Future Optimizations

- Unclustered index results from SUM and SELECT queries could be sorted to reduce repeated page accesses and get closer to the efficiency of a clustered index.
- Page directory structure could be expanded to enable per-column attribute locations, reducing unused space in between valid data.