

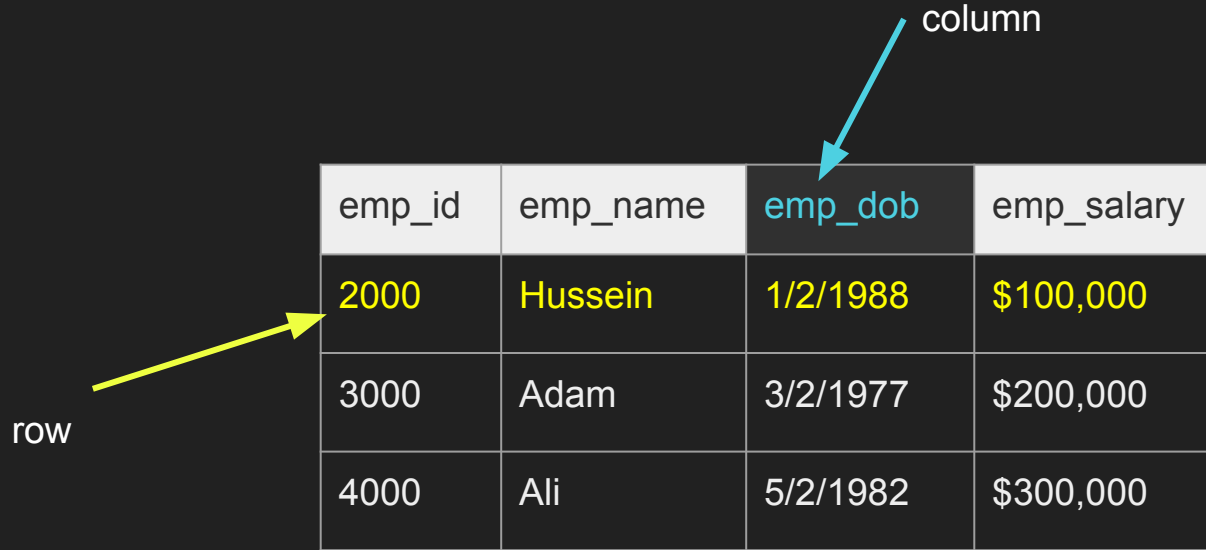
How tables and indexes are stored on disk

And how they are queried

Storage concepts

- Table
- Row_id
- Page
- IO
- Heap data structure
- Index data structure b-tree
- Example of a query

Logical Table



The diagram shows a table with four columns and three rows. A yellow arrow labeled 'row' points to the first row of data. A cyan arrow labeled 'column' points to the 'emp_dob' column header.

emp_id	emp_name	emp_dob	emp_salary
2000	Hussein	1/2/1988	\$100,000
3000	Adam	3/2/1977	\$200,000
4000	Ali	5/2/1982	\$300,000

Row_ID

- Internal and system maintained
- In certain databases (mysql -InnoDB) it is the same as the primary key but other databases like Postgres have a system column row_id (tuple_id)

row_id	emp_id	emp_name	emp_dob	emp_salary
1	2000	Hussein	1/2/1988	\$100,000
2	3000	Adam	3/2/1977	\$200,000
3	4000	Ali	5/2/1982	\$300,000

Page

- Depending on the storage model (row vs column store), the rows are stored and read in logical pages.
- The database doesn't read a single row, it reads a page or more in a single IO and we get a lot of rows in that IO.
- Each page has a size (e.g. 8KB in postgres, 16KB in MySQL)
- Assume each page holds 3 rows in this example, with 1001 rows you will have $1001/3 = 333\sim$ pages

row_id	emp_id	emp_name	emp_dob	emp_salary
1	10	Hussein	1/2/1988	\$100,000
2	20	Adam	3/2/1977	\$200,000
3	30	Ali	5/2/1982	\$300,000
...
1000	10000	Eddard	1/27/1999	\$250,000

Page 0

1,10,Hussein,1/2/1988,\$100,000|2,20,Adam,3/2/1977|3,30,Ali,5/2/1982,\$300,000

Page 1

(Rows 4,5,6)

Page 2

(Rows 7,8,9)

.....

Page 333

More rows...1000,10000,Eddard,1/27/1999,\$250,000

IO

- IO operation (input/output) is a read request to the disk
- We try to minimize this as much as possible
- An IO can fetch 1 page or more depending on the disk partitions and other factors
- An IO cannot read a single row, its a page with many rows in them, you get them for free.
- You want to minimize the number of IOs as they are expensive.
- Some IOs in operating systems goes to the operating system cache and not disk

Page 0

1,10,Hussein,1/2/1988,\$100,000|2,20,Adam,3/2/1977|3,30,Ali,5/2/1982,\$300,000

Page 1

(Rows 4,5,6)

Page 2

(Rows 7,8,9)

.....

Page 333

More rows....1000,10000,Eddard,1/27/1999,\$250,000

Heap

- The Heap is data structure where the table is stored with all its pages one after another.
- This is where the actual data is stored including everything
- Traversing the heap is expensive as we need to read so may data to find what we want
- That is why we need indexes that help tell us exactly what part of the heap we need to read. What page(s) of the heap we need to pull

Page 0

```
1,10,Hussein,1/2/1
988,$100,000|2,
20,Adam,3/2/1977|
3,30,Ali,5/2/1982,$
300,000
```

Page 1

```
( Rows 4,5,6 ) .....
```

Page 2

```
( Rows 7,8,9 ) .....
```

.....

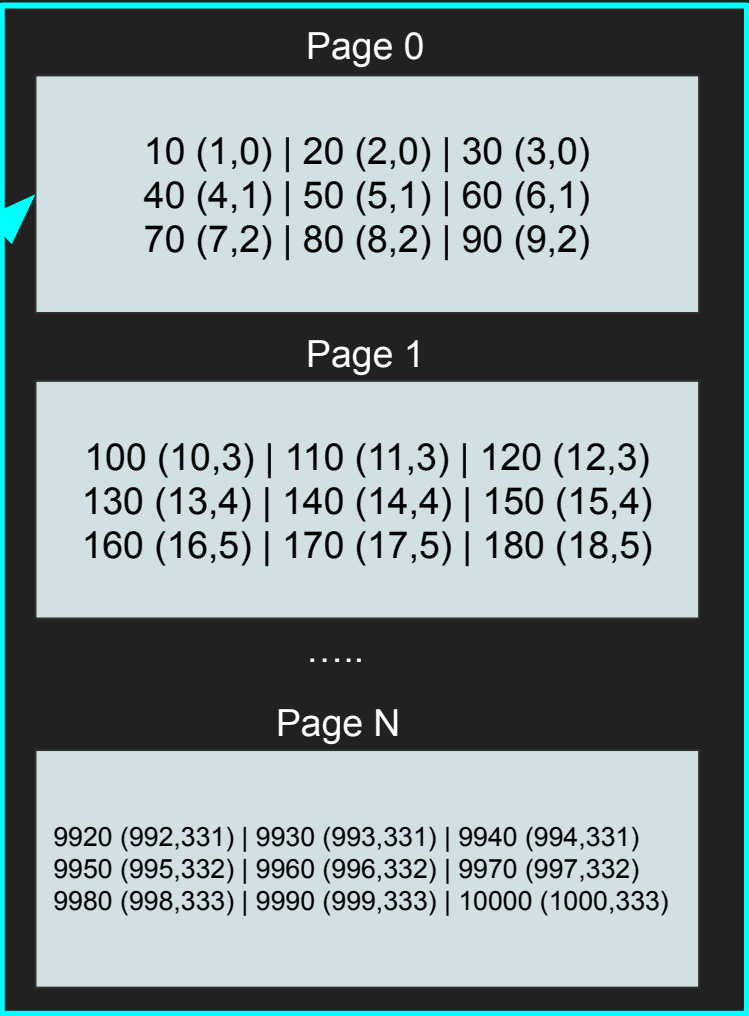
Page 333

```
More
rows....1000,10000
,Eddard,1/27/1999,
$250,000
```

Index

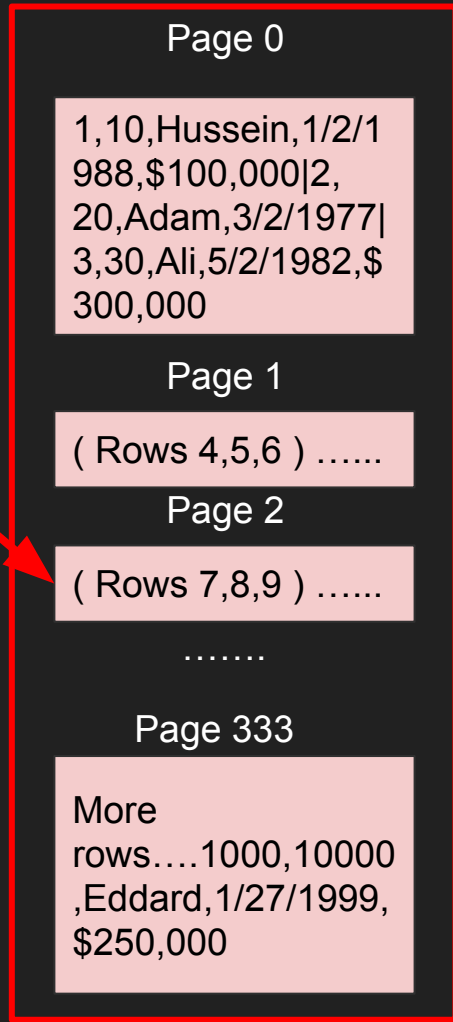
- An index is another data structure separate from the heap that has “pointers” to the heap
- It has part of the data and used to quickly search for something
- You can index on one column or more.
- Once you find a value of the index, you go to the heap to fetch more information where everything is there
- Index tells you EXACTLY which page to fetch in the heap instead of taking the hit to scan every page in the heap
- The index is also stored as pages and cost IO to pull the entries of the index.
- The smaller the index, the more it can fit in memory the faster the search
- Popular data structure for index is b-trees, learn more on that in the b-tree section

Index on
EMP_ID



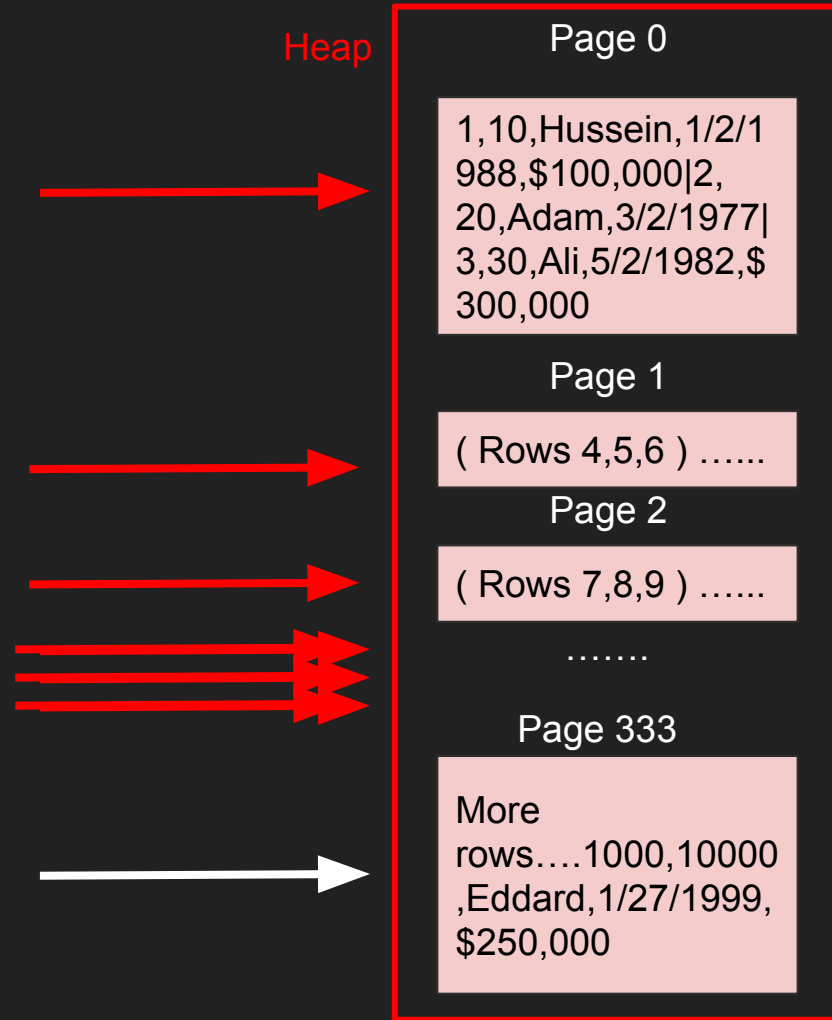
IO1 on
the index
to find the
page/row

Heap



IO2 on
the heap
to pull
exactly
the
page(s)
we found
in the
index

No Index -
SELECT * FROM EMP
WHERE EMP_ID =
10000;



Index on
EMP_ID



Page 0

10 (1,0) | 20 (2,0) | 30 (3,0)
40 (4,1) | 50 (5,1) | 60 (6,1)
70 (7,2) | 80 (8,2) | 90 (9,2)

Page 1

100 (10,3) | 110 (11,3) | 120 (12,3)
130 (13,4) | 140 (14,4) | 150 (15,4)
160 (16,5) | 170 (17,5) | 180 (18,5)

.....

Page N

9920 (992,331) | 9930 (993,331) | 9940 (994,331)
9950 (995,332) | 9960 (996,332) | 9970 (997,332)
9980 (998,333) | 9990 (999,333) | 10000 (1000,333)



With Index -
`SELECT * FROM EMP
WHERE EMP_ID =
10000;`

10000 (1000,333)

10000 (1000,333)

Fetch page 333, and pull row
10000

With Index -

```
SELECT * FROM EMP  
WHERE EMP_ID =  
10000;
```

Heap

Page 0

1,10,Hussein,1/2/1
988,\$100,000|2,
20,Adam,3/2/1977|
3,30,Ali,5/2/1982,\$
300,000

Page 1

(Rows 4,5,6)

Page 2

(Rows 7,8,9)

.....

Page 333

More
rows....1000,10000
,Eddard,1/27/1999,
\$250,000



Notes

- Sometimes the heap table can be organized around a single index. This is called a clustered index or an Index Organized Table.
- Primary key is usually a clustered index unless otherwise specified.
- MySQL InnoDB always have a primary key (clustered index) other indexes point to the primary key “value”
- Postgres only have secondary indexes and all indexes point directly to the row_id which lives in the heap.

Storage concepts - Summary

- Table
- Row_id
- Page
- IO
- Heap data structure
- Index data structure b-tree
- Example of a query