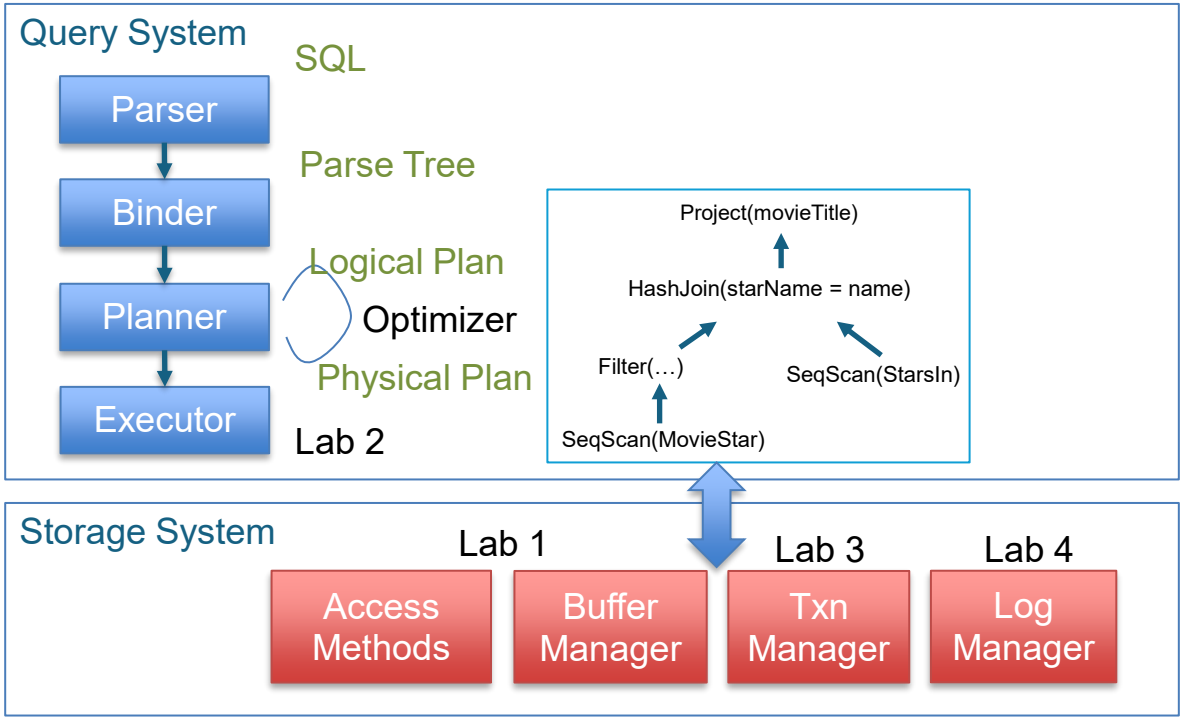
6.5830 / 6.5831 – Lab 1 Bootcamp

Storage & Buffer Management

Query Execution: Overview



What are we building?

- Bitmaps
- Memory Pages
- Buffer Pool
- Tables

Mental Model

- Disk → Pages → Buffer Pool → Tables → Executors

Disk is the lowest level.

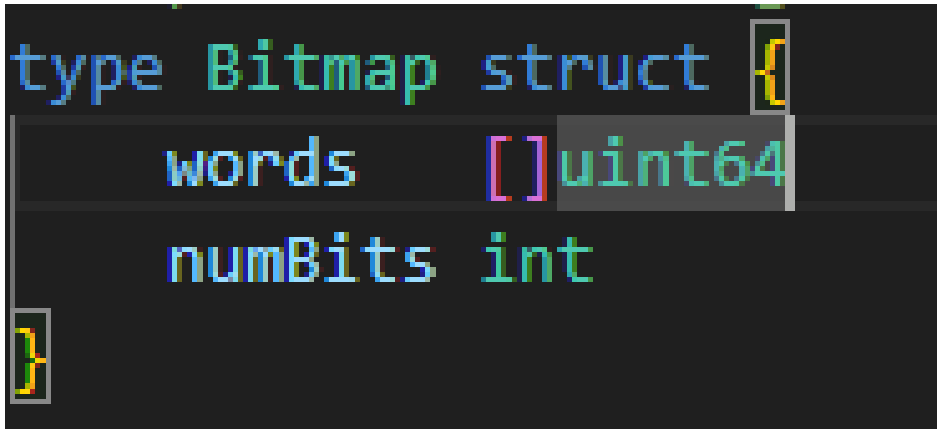
Pages work directly with raw bytes.

Buffer Pool manages pages.

Tables use the buffer pool.

Executors operate on tables.

Part 0 - Bitmaps



The implementation should be optimized for performance by performing word-level operations (uint64)

The main task is to locate which bit within which word is the target.

Bit Basics

- **Bitwise logical operations**: &, |, ^
- **Bitwise shifts**: <<, >>

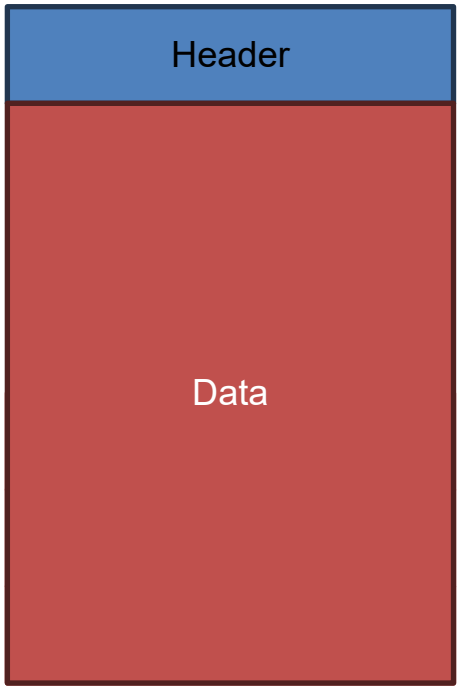
- ***types.go*** might be helpful

Common Mistakes

- Type casting (uint, int, uint64)
- Boundary Checks

Part 1 – Heap Pages

- Fixed-Length Tuples
- No need for slot array
- Allocation Bitmap
- Deletion Bitmap



PageFrame vs HeapPage

```
// HeapPage Layout:
// LSN (8) | RowSize (2) | NumSlots (2) |  NumUsed (2) | Padding (2)
// | allocation Bitmap | deleted Bitmap | rows
type HeapPage struct {
    *PageFrame
}
```

```
// PageFrame represents a physical page of data in memory.
// It holds the raw bytes of the page and acts as the container for Buffer Pool management.
type PageFrame struct {
    // Bytes holds the raw physical data of the page.
    Bytes [common.PageSize]byte
    // PageLatch protects the content of the page from concurrent access.
    PageLatch sync.RWMutex
    // Hint: You will need to add fields and synchronization structures here to track the state of this page.
}
```

A **HeapPage** interprets the raw data stored in a **PageFrame.**
Logical view vs Physical view

RawTuple and RawTupleDesc

- A **Tuple** is basically a single row.
- **RawTuple** is the physical on-disk representation.
- **RawTupleDesc** is a schema for a **RawTuple**.
- **HeapPages** are initialized by the layout/**RawTupleDesc**

Common Mistakes

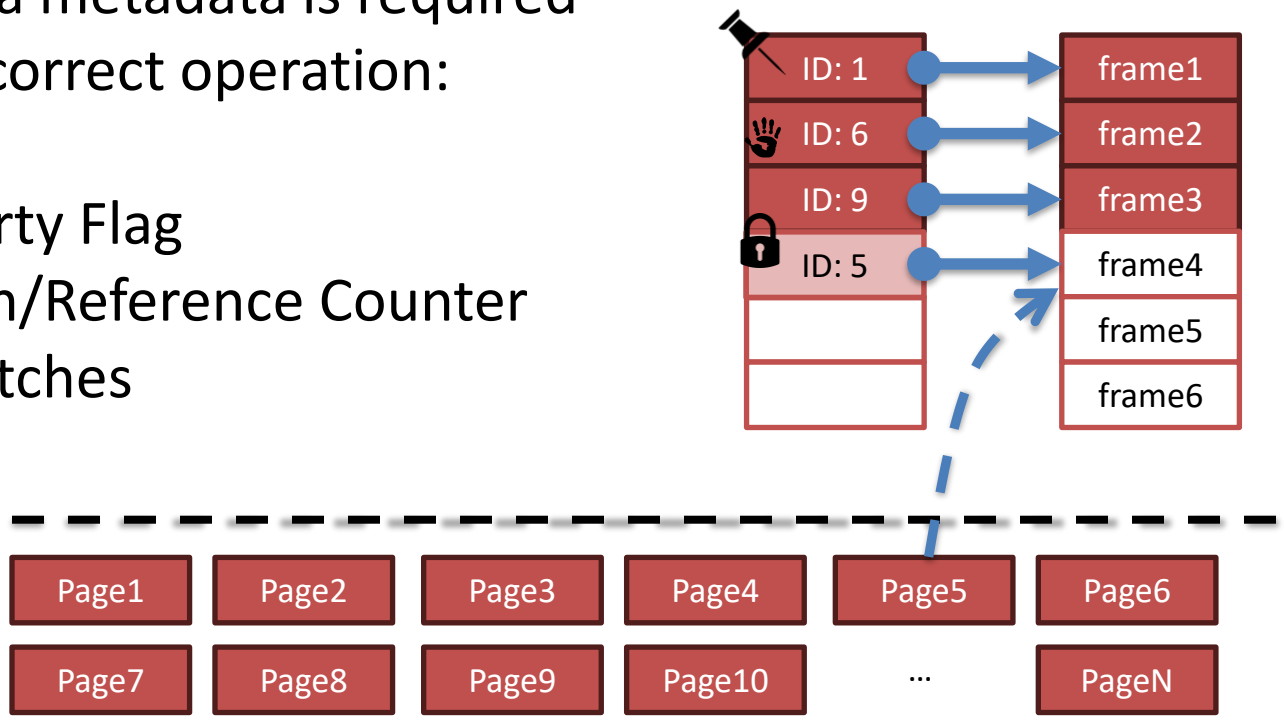
- **InitializeHeapPage()**
  - Do not forget that you decide the size of **Allocation** and **Deleted** Bitmaps, so you need to make sure that they are 8-aligned as well
  - Header is already aligned
  - **utils.go** helper functions

Part 2 – Buffer Pool

A cache for pages.

Extra metadata is required for correct operation:

- Dirty Flag
- Pin/Reference Counter
- Latches



Responsibilities

- fetch
- pin
- unpin
- evict
- flush if dirty

Vocabulary

- **Pin** → cannot evict
- **Unpin** → may evict
- **Dirty** → must write before eviction

Two GetPage paths

- Hit → increment pin
- Miss → find victim → maybe flush → read

Clock Eviction Policy

- **Purpose**: Decide which page to evict when buffer pool is full
- **Mechanism**:
  - reference bit
  - clock hand
- “Second Chance” eviction algorithm

Concurrency Crash-Course

- **Atomic Values**: atomic.Adduint64(&counter, 1)
- **Locks** and **RWLocks**
  - TryLock()
- **xsync** / **sync**
  - xsync.MapOf[K, V], sync.Map[K, V]
  - Store(k, v), Load(k, v) LoadOrStore(k, v)
- **runtime.GoSched()**

Common Mistakes

- One lock for the whole buffer pool
- **GetPage()**
  - Duplicate page loading
  - Pin/Unpin races
  - Dirty page flushes
  - Lock Contention
- **FlushAllPages()**
  - Deadlocks

Debugging advice

- go test –race
- go test -count=100

Part 3 – TableHeap

- Wrapper around HeapPages, a table abstraction
- **Insertion:** Try last page, else allocate new
- **Reads / updates / deletes:** Fetch page → operate → unpin
- Uses Iterator.

Iterator

- Standard way to traverse a collection without exposing its internal structure.
- Similar to generators in python
- **State:** Keep track of current position
- **Next()**
  - Advance to next valid element
  - Return true if a next element exists, false if at end
- **Current():** Return the current element without advancing
- **Error():** Optionally track errors
- **Close():** Release any resources help by the iterator

Common Mistakes

- Reuse buffers. Avoid allocations in loops.
- Usually InsertTuple() is the place for deadlocks or lock contention

Time expectation

- BufferPool will take most of your time.
- If you are stuck or have already spent too much time on debugging come to office hours.

Questions?