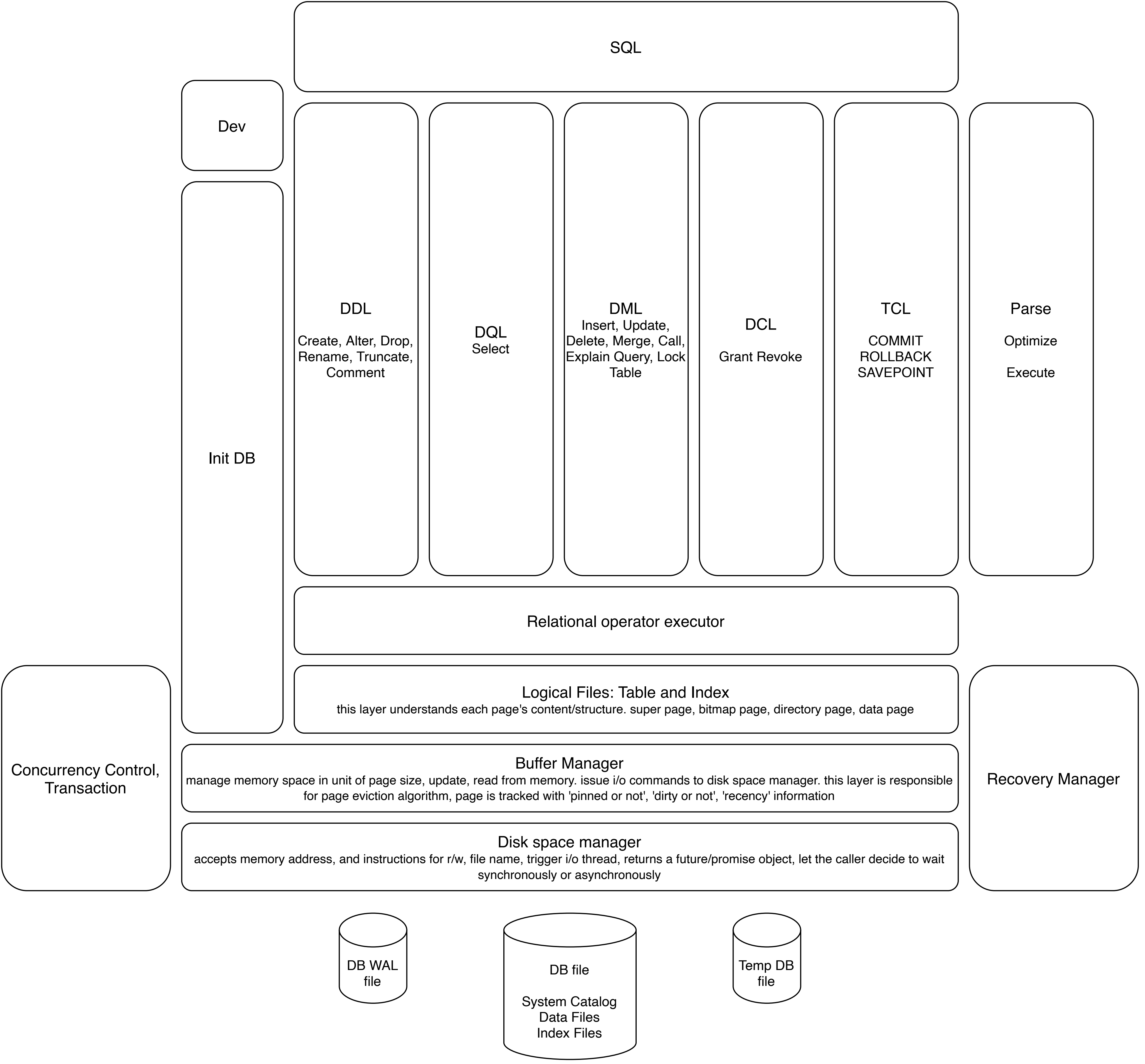

Logical Files: Table and Index
this layer understands each page's content/structure. super page, bitmap page, directory page, data page

Buffer Manager
manage memory space in unit of page size, update, read from memory. issue i/o commands to disk space manager. this layer is responsible for page eviction algorithm, page is tracked with 'pinned or not', 'dirty or not', 'recency' information

Disk space manager
accepts memory address, and instructions for r/w, file name, trigger i/o thread, returns a future/promise object, let the caller decide to wait synchronously or asynchronously

Relational operator executor

This layer exposes iterators to the above customers. Inside iterator, logical_file operation is there. Some operators are streaming, while other operators such as sort are blocking, and might need additional temp db file. The point is: the customer cannot see record, they only see iterators.

This layer accepts a relational tree as input, which comes from query optimizer from the above.

Logical Files

Whenever one entity needs to span across multiple pages, we call it a logical file. So logical files layer needs to set a mapping from logical file page id to physical block id. Buffer Manager will translate from physical block id to frame id so the content is accessible. Known logical files include:

Super file: including the zeroth page, which tells us about the system catalog files' first page, db file's configuration parameters including block size, and bitmap file's first page. Super file is the root to visit all other files. Its first page is a fixed constant number: zero.

Bitmap file: whenever the customer wants a new page, it has to allocate one from bitmap file.

Table file: including system catalog table, and user data table. The only difference between them is: their first page is recorded at different locations. System catalog table's first page is recorded at super file, while the user data table's first page is recorded at system catalog table: sys_table.

Table file's pages are accessed from the Directory page.

Index file's first page is recorded at some system catalog table. We do not need Directory page for index file because it won't be accessed sequentially, and other internal and leaf pages will be accessed from the B+ Tree's first page. However, if it is a hash index, then things might be different.

The interface it provides is: scan_table(), query_table(), modify_table().

BufferManager

BufferManager does not know anything about the managing pages' content. It constructs IO commands and send them downwards.

Internally, buffer manager needs to implement a page eviction algorithm, and dynamically expand the allocated memory to the maximum allowed. Buffer Manager also needs to maintain an in-memory only table so that pageId can be translated to frameId quickly. So buffer manager would have quite some internal data structures.

The interface it provides is: given one page id, it returns the frame address for read/write. The customer never needs to worry about what page to evict because that's internal details

DiskSpaceManager

DiskSpaceManager does not know anything about buffer. It only accepts IO commands from the customer. IO commands including: create a new file, append a block, write a block, read a block. The interface returns a promise, so that the customer can wait synchronously and asynchronously.

Internally, DiskSpaceManager only has one thread, because hard disk is inherently a one-thread device. This IO thread is different from the computation thread. Since CPU and hard disk are parallel devices, separating IO thread and computation thread is a good design.