Storage and Indexes:
Step-by-step Guide from Official Docs

CSCI3170 Tutorial

TA: YANG Yitao, ytyang@cse.cuhk.edu.hk

Content

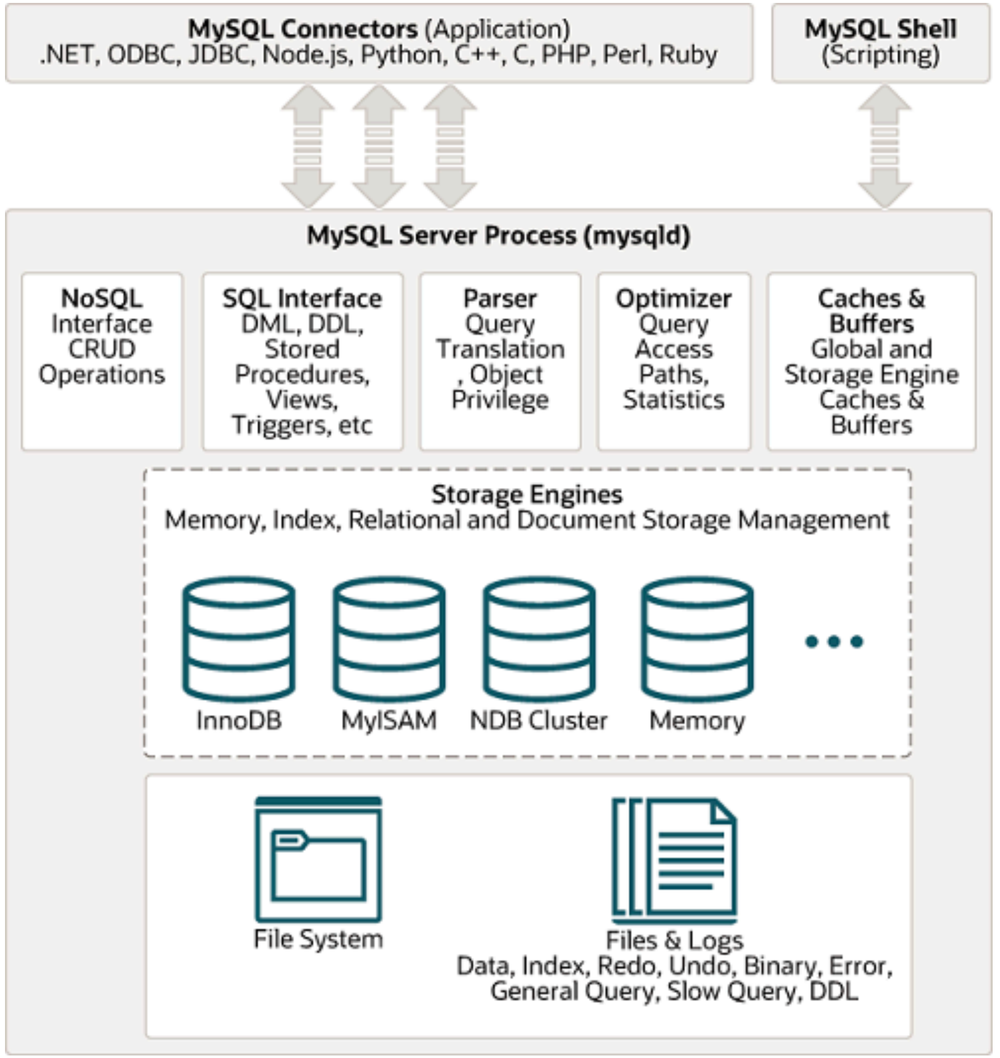
- MySQL Storage Engines
- MySQL Performance Optimization - Indexes
- Hand on Practices - Revisit LeetCode 175

Storage Engines

Reference: https://dev.mysql.com/doc/refman/8.0/en/

Storage Engines: Introduction

- Storage engines are MySQL components that handle the SQL operations for different table types.
- InnoDB is the default and most general-purpose storage engine, and Oracle recommends using it for tables except for specialized use cases. (The CREATE TABLE statement in MySQL 8.0 creates InnoDB tables by default.)



Manual 16: Alternative Storage Engines

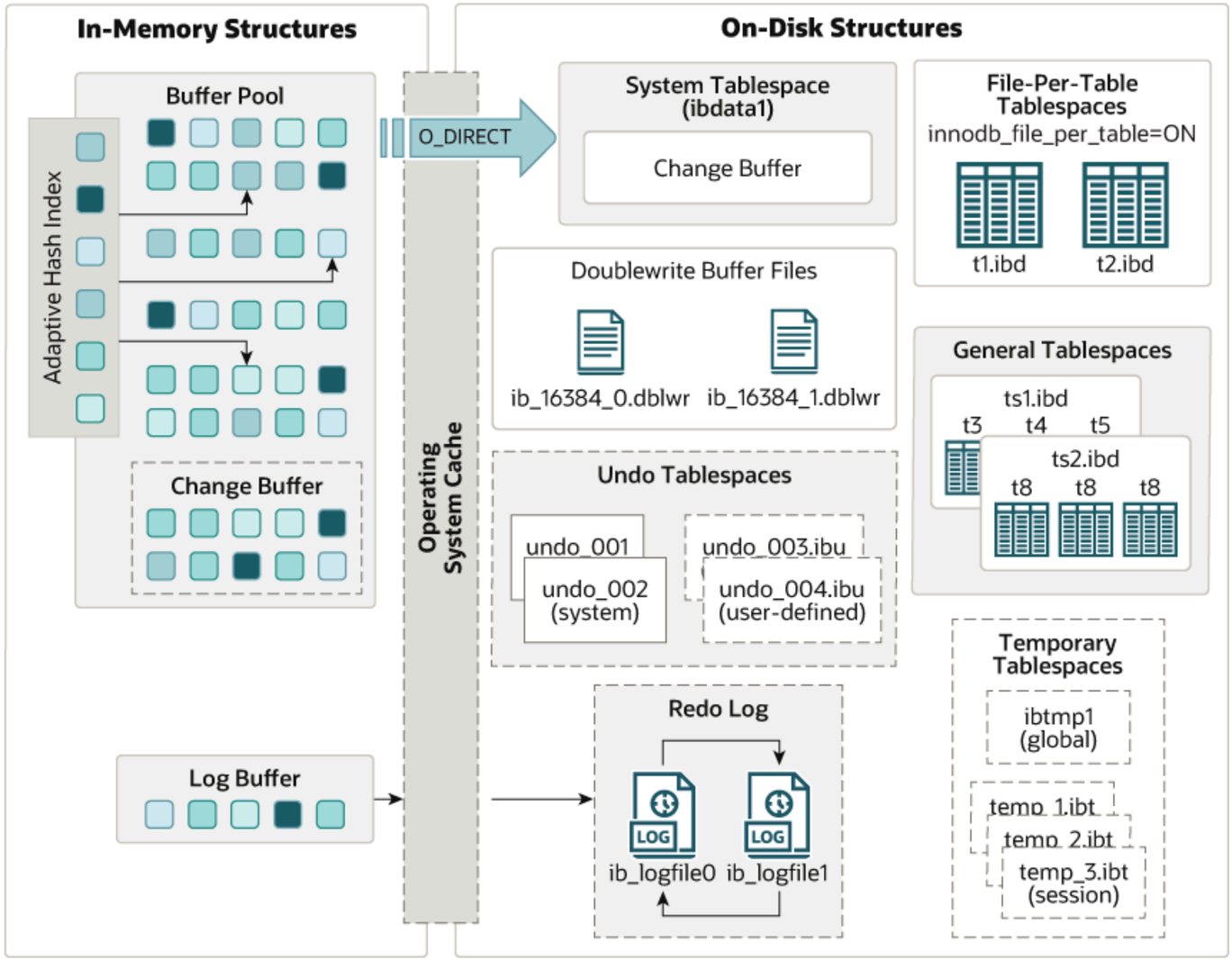
- To determine which storage engines your server supports, use the SHOW ENGINES statement.

```
mysql> show engines;
+--------------------+---------+----------------------------------------------------------------+
| Engine             | Support | Comment                                                        |
+--------------------+---------+----------------------------------------------------------------+
| ARCHIVE            | YES     | Archive storage engine                                         |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears) |
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables                          |
| FEDERATED          | NO      | Federated MySQL storage engine                                 |
| MyISAM             | YES     | MyISAM storage engine                                          |
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                             |
| InnoDB             | DEFAULT | Supports transactions, row-level locking, and foreign keys     |
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables      |
| CSV                | YES     | CSV storage engine                                             |
+--------------------+---------+----------------------------------------------------------------+
```

Manual 16: MySQL 8.0 Supported Storage Engines

- To determine which storage engines your server supports, use the SHOW ENGINES statement.
- InnoDB: The default storage engine in MySQL 8.0. InnoDB is a transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data.
- MyISAM: These tables have a small footprint. Table-level locking limits the performance in read/write workloads, so it is often used in read-only or read-mostly workloads in Web and data warehousing configurations.
- Memory: Stores all data in RAM, for fast access in environments that require quick lookups of non-critical data. This engine was formerly known as the HEAP engine.
- CSV: Its tables are really text files with comma-separated values. CSV tables let you import or dump data in CSV format, to exchange data with scripts and applications that read and write that same format.

Manual 15.4: InnoDB Architecture




```
~ % ls /opt/homebrew/var/mysql
 ibdata1
 ibtmp1
 mysql
 mysql.ibd
 mysql_upgrade_info
 performance_schema
 private_key.pem
 public_key.pem
 server-cert.pem
 server-key.pem
 sys
 undo_001
 undo_002
```

Manual 15.1-5: Introduction to InnoDB and Buffer Pool

- Key Advantages of InnoDB:
    - Its DML operations follow the ACID model, with transactions featuring commit, rollback, and crash-recovery capabilities to protect user data.
    - Row-level locking and Oracle-style consistent reads increase multi-user concurrency and performance.
    - InnoDB tables arrange your data on disk to optimize queries based on primary keys.
- Buffer Pool of InnoDB:
    - The buffer pool is an area in main memory where InnoDB caches table and index data as it is accessed.
    - The buffer pool permits frequently used data to be accessed directly from memory, which speeds up processing. On dedicated servers, up to 80% of physical memory is often assigned to the buffer pool.

Optimization and Indexes

Reference: https://dev.mysql.com/doc/refman/8.0/en/optimization-indexes.html

Manual 8.3: Optimization and Indexes

- Why:
    - The best way to **improve the performance** of SELECT operations is to create indexes on one or more of the columns that are tested in the query.
    - The index entries act like pointers to the table rows, allowing the query to **quickly determine which rows match a condition** in the WHERE clause, and retrieve the other column values for those rows. All MySQL data types can be indexed.
- Why Not:
    - Although it can be tempting to create an indexes for every possible column used in a query, **unnecessary indexes waste space and waste time** for MySQL to determine which indexes to use.
    - Indexes also **add to the cost of inserts, updates, and deletes because each index must be updated**. You must find the right balance to achieve fast queries using the optimal set of indexes.

Manual 8.3.1: How MySQL Uses Indexes

- Most MySQL indexes (PRIMARY KEY, UNIQUE, INDEX, and FULLTEXT) are stored in B-trees.
- MEMORY tables also support hash indexes; InnoDB uses inverted lists for FULLTEXT indexes.
- Indexes are less important for queries on small tables, or big tables where report queries process most or all of the rows. When a query needs to access most of the rows, reading sequentially is faster than working through an index.

Manual 8.3.9: Why B-Tree Indexes instead of Hash

- B/B+ tree is costly to maintain since it needs to be updated after every insertion and deletion.
- Hash based indexing is efficient in equity queries whereas B+ trees are efficient in range queries.
- Efficiency of hash based index is low in case of large no. of repeated key values because of hash collision problems.
- Hash index is not efficient in sorting.
- Hash index is effective in insertion and deletion whereas B+ tree isn't.

Hand on Practices

LeetCode 175, Combine Two Tables

- Write an SQL query to report the *first name, last name, city, and state* of each person in the *Person* table.
- If the *address* of a *personId* is not present in the *Address* table, report *null* instead.

Input

Person =

personId	lastName	firstName
1	Wang	Allen
2	Alice	Bob

Address =

addressId	personId	city	state
1	2	New York City	New York
2	3	Leetcode	California

Expected

firstName	lastName	city	state
Allen	Wang	null	null
Bob	Alice	New York City	New York

Solution: LEFT JOIN

- A join clause in SQL, corresponding to a join operation in relational algebra, combines columns from one or more tables **into a new table**.
- Informally, a join stitches two tables and puts on the same row records with **matching fields**.

```
3  SELECT
4      t1.firstName,
5      t1.lastName,
6      t2.city, t2.state
7  FROM
8  Person t1 LEFT JOIN Address t2
9  ON t1.personId = t2.personId
```