SBD MySQL Report

Table of Contents

Introduction **3**

Storage and File Structure **4**
- MySQL and its Storage Engine 4
- Buffer pool 6
- Partitioning 7
- File System 9

Indexing and Hashing **10**
- Clustered and Secondary indexes 10
- LSM-tree and MyRocks in MySQL 12

Query Processing and Optimization **14**
- Main Operations 14
- Complex Operations 15

Transaction Management and Concurrency Control **17**
- ACID Principles 17
- Isolation Levels 18
- Consistency Levels 19
- Granularity 20

Distributed Database Support **21**
- Replication 21
- Fragmentation 22

Other Features **24**
- Cloud deployment 24
- MyISAM (Alternative to InnoDB) 24
- MEMORY (Alternative to InnoDB) 24

References **25**

Introduction

This paper was made in the scope of the discipline of database systems and aims to study and analyze a database management system (DBMS) and compare it with the Oracle 18c database management system studied during classes.

The DBMS chosen was MySQL. This is an open-source relational database management system. A relational database organizes data into one or more tables in which data may be related to each other, these relations help structure the data.

MySQL is written in c,c++ and its SQL parser is written in yacc, but it uses a home-brewed lexical analyser(is the process of converting a sequence of characters, such as in a computer program or web page, into a sequence of tokens). It is able to run in a wide variety of operating systems.

It is a free and open-source software under the terms of the GNU General Public License. The original author is a Swedish company MySQL AB but now is owned and developed by oracle Corporation.

MySQL has received positive feedback like, it "performs extremely well in the average case", "developer interfaces are there, and the documentation (not to mention feedback in the real world via Web sites and the like) is very, very good". It has also been tested to be a "fast, stable and true multi-user, multi-threaded SQL database server".

During this paper we approach various characteristics of mysql and in the end a comparison to the characteristics of oracle which was studied in class.

Storage and File Structure

MySQL and its Storage Engine

Storage engines are MySQL components that can handle SQL operations for different table types to store and manage information in a database. There are many storage engines available in MySQL and they are used for different purposes but MySQL uses InnoDB as its default engine since its 5.5 version.
When creating a table with MySQL we can use the ENGINE option to choose which engine we want to use and if we want to change a table from one engine to another we can do it with an ALTER TABLE statement and an ENGINE option.

Examples:

```
CREATE TABLE t1 (i int); using InnoDB as default option.
CREATE TABLE t2 (i int) ENGINE = CVS;
ALTER TABLE t2 ENGINE = InnoDB;
```

InnoDB

InnoDB is a general-purpose storage engine that balances high reliability and high performance.
Features of InnoDB storage engine:

Storage limits	64TB	Transactions	Yes	Locking granularity	Row
MVCC (Multiversion concurrency control)	Yes	Geospatial data type support	Yes	Geospatial indexing support	No
B-tree indexes	Yes	T-tree indexes	No	Hash indexes	No
Full-text search indexes	Yes	Clustered indexes	Yes	Data caches	Yes
Index caches	Yes	Compressed data	Yes	Encrypted data	Yes
Cluster database support	No	Replication support	Yes	Foreign key support	Yes
Backup / point-in-time recovery	Yes	Query cache support	Yes	Update statistics for data dictionary	Yes

Advantages:

- Has maximum performance when processing large data volumes.
- DML(Data manipulation language) operations are compatible with ACID properties.
- Row-level locking increases multi-user concurrency and performance.
- Tables arrange data on disk to optimize queries based on primary keys.
- Supports FOREIGN KEY constraints to maintain data integrity.
- It is possible to mix InnoDB tables with tables from other MySQL storage engines within the same statement.

Limitation of InnoDB table:

- Maximum 1017 columns are allowed in a table.
- Maximum 64 secondary indexes are allowed in a table.
- The maximum row length except for variable-length columns(VARBINARY, VARCHAR, BLOB AND TEXT) is about 8000 bytes for the default page size of 16KB.
- Maximum table space size is 64TB and the minimum is slightly larger than 10MB.

InnoDB, Variable-Length Records and Slotted pages

Slotted page structure is a very widely used data format especially in MySQL. One of its biggest strengths is its ability to store variable length efficiently without wasting too much space.
Pages have a linked list feature, as in page nr7 has prev = page nr6 and next=page nr8.

Page Merging

When a page has received enough deletes to match the MERGE_THRESHOLD(50% of the page size by default), InnoDB starts to look to the closest pages, next or previous, to see if the two pages can merge.

What InnoDB will do is:

1. If page nr6 is utilizing less than 50%.
2. Page nr5 received many deletes and is also now less than 50% used.
3. The merge operation results in page nr5 containing its previous data plus the data from page nr6.
4. Page nr6 becomes an empty page usable for new data.

Page Splits

Pages can be filled up to 100% and when this happens the next page takes new records.
But if page nr7 doesn't have enough space for a new or updated record and page nr8 is also full, data cannot be inserted out of order so the following happens.

What InnoDB will do is:

1. Create a new page, page nr9 is created.
2. Split the records first half in page nr7 and second half in page nr9 plus the new or updated record.
3. Redefine the page relationships
    - Page nr7 will have prev = 6 and next = 9
    - Page nr9 will have prev = 7 and next = 8
    - Page nr8 will have prev = 9 and next = 10

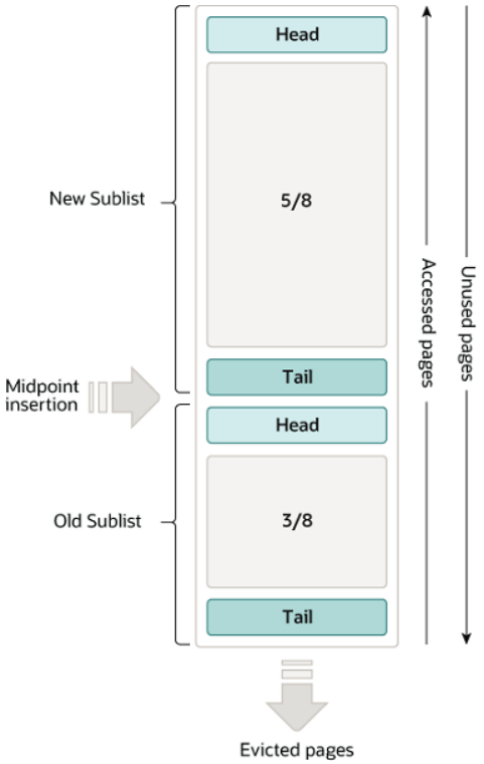
Page splits happen on insert or update and cause page dislocation. Once the split page is created, one way to move it back is to have the created page drop below the merge threshold. Then this happens, innoDB moves the data from the split page with a merge operation.
The other way is to reorganize the data by optimizing the table. This can be a very heavy and long process, but to recover from a situation where too many pages are located in sparse extents.

Buffer pool

The buffer is an area in main memory where InnoDB caches table and index data as it is accessed. The buffer pool permits used data to be accessed directly from memory, which speeds up processing. In dedicated servers, up to 80% of the physical memory is often assigned to the buffer pool.
For efficiency of high-volume read operations, the buffer poll is divided into pages that can hold multiple rows. The buffer pool is implemented as a linked list of pages, data that is rarely used is aged out of the cache using a variation of the LRU algorithm.
Like most DBMS, knowing how to take advantage of the buffer pool is an important aspect of tuning.

Buffer Pool LRU Algorithm

When room is needed to add a new page, the least recently used page is evicted and a new page is added to the middle of the list. This midpoint insertion strategy uses the list as two sublists. At the head, a sublist of new pages that were accessed recently and at the tail, a sublist of old pages that were accessed less recently.



By default the algorithm operates as follows:

- ⅜ of the buffer pool is devoted to the old sublist.
- The midpoint of the list is the boundary where the tail of the new sublist meets the head of the old sublist.
- If the page was read because it was required by a user-initiated operation, the access occurs immediately and the page is made young. If the page was read due to a read-ahead the access might not occur at all before the page is evicted.
- Eventually, a page that remains unused reaches the tail of the old sublist and is evicted.

Buffer Pool configuration

- Ideally the size of the buffer should be as large as possible, leaving enough memory for other processes on the server. The larger the buffer pool, the more InnoDB acts like an in-memory database.
- On 64-bit systems with sufficient memory, you can split the buffer pool into multiple parts to minimize contention for memory structures among concurrent operations.
- We can control how and when to perform read-ahead to prefetch pages into the buffer pool asynchronously.
- We can configure how InnoDB preserves the current buffer pool state to avoid lengthy warmup after a server restart.

Partitioning

In MySQL 8.0, partitioning is provided by the InnoDB and NDB (Nucleic Acid Database) storage engines. The MySQL server, even prior to the introduction of the partitioning, could be configured to employ different physical directories for storing different databases.
Partitioning takes this notion a step further, by enabling distribution of portions of individual tables across a file system according to rules which we can set.

There's two ways to do partitioning, we can do it using horizontal or vertical partitioning. In horizontal partitioning different rows of a table may be assigned to different physical partitions this is supported by MySQL but vertical partitioning is not supported in which different columns of a table are assigned to different physical partitions. The official documentation of MySQL clearly states it won't support vertical partitions any time soon: "There are no plans at this time to introduce vertical partitioning into MySQL.".

Advantages of partitioning:

- Possible to store more data in one table than can be held on a single disk or file system partition.
- Data that loses its usefulness can often be easily removed from a partitioned table.
- Some queries can be greatly optimized because of the fact that data satisfying a where clause can be stored only on one or more partitions, this technique is often called partition pruning.
- Partitions can be altered after being created that allows reorganizing the data to new frequent queries.

Partitioning types that are available in MySQL:

- Range - assigns rows to partitions based on column values falling within the range.
- List - similar to range but partition is selected based on columns matching one of a set of discrete values.
- Hash - a partition is selected based on the number returned by a user-defined expression on column values.
- Key - similar to hash partitioning but one or more columns can be evaluated and MySQL server provides its own hashing function that guarantees an integer result regardless of the column data type.

The benefit of partitioning by hash is that the data can be written to these partitions more concurrently.

Restrictions and Limitations on Partitioning:

- Arithmetic and logical operators are permitted. However, the result must be an integer value or a null.
- Div is supported but the operator is not permitted.
- Bit operators |, &, ^, <<, >> and ~ are not allowed in partitioning expressions.
- Changing the SQL mode at any time after the creation of partitioned tables may lead to changes in the behavior of tables and to corruption or loss of data.
- The maximum number of partitions for a given table not using the NBD storage engine is 8192 this includes subpartitions.
- Partitioned tables using InnoDB do not support foreign keys.
- An `ALTER TABLE … ORDER BY COLUMN` statements on a partitioned table causes ordering of rows only within each partition.
- Partitioned tables do not support fulltext indexes or searches.
- Columns with spatial data types such as POINT or GEOMETRY cannot be used in partitioned tables.
- Temporary tables cannot be partitioned.
- Expressions employing ENUM columns cannot be used.
- and others…

How to use partitions:

```
CREATE TABLE members (
    firstname VARCHAR(25) NOT NULL,
    lastname VARCHAR(25) NOT NULL,
    username VARCHAR(16) NOT NULL,
    email VARCHAR(35),
    joined DATE NOT NULL
)

PARTITION BY KEY(joined)
PARTITIONS 6;

PARTITION BY RANGE( YEAR(joined) ) (
    PARTITION p0 VALUES LESS THAN (1960),
    PARTITION p1 VALUES LESS THAN (1970),
    PARTITION p2 VALUES LESS THAN (1980),
    PARTITION p3 VALUES LESS THAN (1990),
    PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

Oracle partitioning

Unlike MySQL, Oracle supports both vertical and horizontal partitioning and offers a lot of partitioning methods: by range, list, hash, multi-column range, interval, composite, reference, virtual column-based, and interval reference. All these methods can be used with any application.

File System

MySQL uses the native file system of the host volume on the host operating system with a variety of I/O techniques including memory mapping, scatter/gather and ordinary read and writes system calls all integrated with the file system journaling features. The exact techniques used depend on the kind of query, the access method and the state of the caches.

Compared to Oracle 18c which implements its own Database File System(DBFS). DBFS creates a standard file system interface on top of files and directories that are stored in database tables. With these techniques the Oracle Database provides much better security, availability, robustness, transactional capability and scalability than traditional files systems.

Indexing and Hashing

Clustered and Secondary indexes

Each InnoDb table has a special index called the clustered index that stores row data. Typically the cluster index is synonymous with the primary key.

How the Clustered Index speeds up queries

Accessing a row through the clustered index is fast because the index search leads directly to the page that contains the row data. When a table is large, the clustered index architecture often saves a disk I/O operation when compared to a data store using a different page from the index record.

How Secondary Indexes relate to the Clustered Index

In InnoDB, each record in a secondary index contains the primary key columns for the row, as well as the columns for the secondary index. InnoDB uses this primary key value to search for the row in the clustered index. If the primary key is long, the secondary indexes use more space, so it is advantageous to have a short primary key.

The Physical Structure of an InnoDB Index

InnoDB indexes are B-tree data structures with the exception of spatial indexes that use R-trees, which are specialized data structures for indexing multidimensional data with one or more columns.

If index records are inserted in a sequential order the resulting index pages are about 15/16 full, if they are inserted in a random order, the pages are from ½ to 15/16 full because InnoDB tries to leave 1/16 of the page free for future insertions and updates.

Sorted index builds

InnoDB performs a bulk load instead of inserting one index record at a time when creating or rebuilding indexes. This is not supported for spatial indexes.

There are three phases to an index build.

1. The clustered index is scanned, and index entries are generated and added to the sort buffer. When the buffer becomes full, entries are sorted and written to a temporary file. This process is also called a “run”.
2. With one or more runs written to the temporary file, a merge sort is performed on all entries in a file.
3. The sorted entries are inserted into the B-tree.

To set aside space for future index growth, you can use the innodb_fill_factor to reserve a percentage of the B-tree space setting it to 80 reserves 20% of the space of the B-tree pages during a sorted index build.

InnoDB Full-text Indexes

Full-text indexes are created on text-based columns to speed up queries and DML on data contained within those columns. Full-text search is performed using MATCH() … AGAINST syntax.

InnoDB Full-Text Index Design and Cache

InnoDB full-text has an inverted index design, they store a list of words, and for each word, a list of documents that the words appear in.

Inserting a document even if small can result in numerous small insertions to the auxiliary index table, to avoid this problem innoDB uses an index cache to temporarily cache index table insertions for recently inserted rows. This in-memory cache structure holds insertions until the cache is full then flushes them to disk.
This technique allows for less frequent updates to auxiliary index tables, avoids multiple insertions for the same word and minimizes duplicated entries, insertions for the same word are merged and flushed to disk as a single entry.

MEMORY Storage Engine

When using a MEMORY storage engine we can choose B-tree or hash-indexe(default), understanding the differences of those data structures can help predict how different queries perform. Hash index is only available in the MEMORY storage engine.

A B-tree index can be used for column comparisons that use the =, <, <=, >, >=, or BETWEEN operators, and can be for LIKE comparisons if the argument to LIKE is a constant string that does not start with a wildcard character.

Hash indexes on the other hand can only be used for equality comparisons that use the = or <=> operators but are very fast.
The optimizer cannot use a hash index to speed up ORDER BY operations.
Only whole keys can be used to search for a row but with a B-tree index, any leftmost prefix of the key can be used to find rows.

Adaptive Hash Index

Although InnoDB doesn't use a hash index, MySQL monitors index searches for InnoDB tables and if queries could benefit from a hash index , it builds one automatically based on an existing B-tree index. MySQL can build a hash index on a prefix of any length of the key defined for the B-tree, depending on the pattern of searches.
A hash index can be partial and the whole B-tree index does not need to be cached in the buffer poll

LSM-tree and MyRocks in MySQL

The standard storage engines don´t support LSM-tree data structures, but facebook created MyRocks, a new storage engine for MySQL based on RocksDB, a high performance embedded database. It's a fork of Google LevelDB optimized to make efficient use of fast storage such as ssd and its base on LSM-tree data structure.
The idea of MyRocks is to use the features of LSM-tree data structure combined with MySQL features.

MyRocks efficiency focuses on better space, write and read efficiency.

- Better space efficiency means using less SSD storage.
- Better write efficiency means SSD endurance.
- Better read efficiency comes from more available I/O capacity for handling queries.

Advantages of MyRocks:

- Better compression and storage efficiency compared to other MySQL engines.
- Random operation reduction, because the access patterns are sequential. This allows us to use almost the full I/O capacity of the storage.

Disadvantages of MyRocks:

- Not as many features as InnoDB.
- LSM-tree uses more CPU per query and might increase the number of CPU cache misses.
- LSM-trees suffer from the range read penalty because they have to check several files to get data for a range query.

Usage Example

Run the following code:

```
CREATE TABLE city
(Name VARCHAR(35),
 Country VARCHAR(4),
 Province VARCHAR(35),
 Population INT,
 Longitude FLOAT,
 Latitude FLOAT,
 CONSTRAINT CityKey PRIMARY KEY (Name, Country, Province),
 CONSTRAINT CityPop CHECK (Population >= 0),
 CONSTRAINT CityLon CHECK ((Longitude >= -180) AND (Longitude <=
180)),
 CONSTRAINT CityLat CHECK ((Latitude >= -90) AND (Latitude <= 90)));
```

This will create a B-tree index by default when using InnoDB or hash index when using MEMORY.

If we run the following query after creating a table with InnoDB :
“`SHOW INDEX FROM city;`” the output will be the following.

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
city	0	PRIMARY	1	Name	A	NULL	NULL	NULL		BTREE			YES	NULL
city	0	PRIMARY	2	Country	A	NULL	NULL	NULL		BTREE			YES	NULL
city	0	PRIMARY	3	Province	A	3111	NULL	NULL		BTREE			YES	NULL

If we alter the engine with the query:
“`ALTER TABLE city ENGINE=MEMORY;`” then the output of the show index query will be different.

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
city	0	PRIMARY	1	Name	NULL	NULL	NULL	NULL		HASH			YES	NULL
city	0	PRIMARY	2	Country	NULL	NULL	NULL	NULL		HASH			YES	NULL
city	0	PRIMARY	3	Province	NULL	3111	NULL	NULL		HASH			YES	NULL

Now lets create a B-tree and hash indexes with the two following queries:

```
CREATE INDEX cityNAME ON city (name);
CREATE INDEX cityNP USING BTREE ON city (name, population);
```

When they are finished we have the following index:

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
city	0	PRIMARY	1	Name	NULL	NULL	NULL	NULL		HASH			YES	NULL
city	0	PRIMARY	2	Country	NULL	NULL	NULL	NULL		HASH			YES	NULL
city	0	PRIMARY	3	Province	NULL	3111	NULL	NULL		HASH			YES	NULL
city	1	cityNAME	1	Name	NULL	NULL	NULL	NULL		HASH			YES	NULL
city	1	cityNP	1	Name	A	3044	NULL	NULL		BTREE			YES	NULL
city	1	cityNP	2	Population	A	3107	NULL	NULL	YES	BTREE			YES	NULL

Query Processing and Optimization

Main Operations

The Main Query Operations are implemented in the following manner in MySQL:

- SELECT- Starts at the FROM clause and then proceeds to the SELECT clause by performing a full table scan if no Index is available. If there is an Index available, it will be used instead.
- JOIN - This operation supports *Nested Loops* and *Block Nested Loops*. The only type of Hash Join it supports is the *Inner Hash Join* Algorithm. It does not support *Merge Join* at all. If an equi-join operation is being performed, the system will use *Hash Join* by default, it will use *Block Nested Loops* for other situations (like outer joins).
- ORDER BY- If an Index is available for sorting, it will be used. If not, the *Filesort* operation will be used instead. If the sorting can not be done fully in memory then the operation will use temporary disk files as necessary. This is different from Oracle 18c which uses several sorting algorithms. Here is the breakdown of the *Filesort* algorithm:
    1. Read the rows that match the WHERE clause.
    2. For each row, record a tuple of values consisting of the sort key value and the additional fields referenced by the query.
    3. When the sort buffer becomes full, sort the tuples by sort key value in memory and write it to a temporary file.
    4. After merge-sorting the temporary file, retrieve the rows in sorted order, read the required columns directly from the sorted tuples

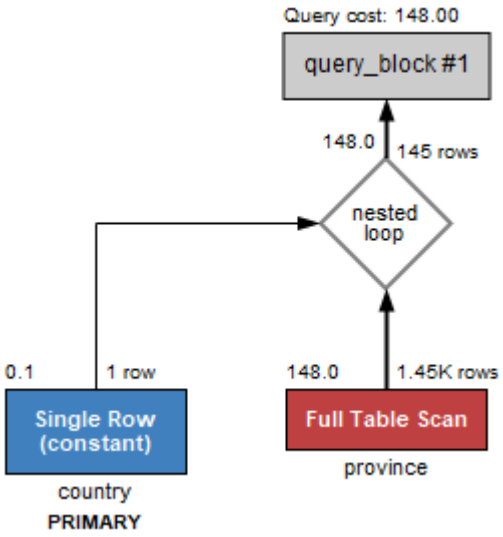
In regards to Query Processing, the intermediate language that is utilized is *yacc* (Yet Another Compiler Compiler). What *yacc* does, is that it translates from a textual format to binary structures, which are easier to manipulate by the optimizer.

As for MySQL's optimizer, it is a very important tool that completes several tasks:

- Determines the join order, when more than 2 tables must be joined.
- Removes unused tables from the join.
- Rewrites the WHERE clause to reduce unnecessary computations.
- Determines whether keys or indexes can be used for ORDER BY and GROUP BY clauses.
- Tries to replace OUTER JOIN with an equivalent INNER JOIN.
- Attempts to simplify subqueries and determines if their results can be cached.

As of now, MySQL performs limited subquery optimizations. An example is when a subquery returns only one row, it replaces this value with a constant, as in:

```
SELECT*
FROM country INNER JOIN province ON country.Code = province.Country
WHERE country.Code = 'BR';
```



The optimizer generates possible query solutions and attributes a cost estimation to each one. This cost is the reference used to pick the best execution plan, and usually, the lower the cost, the faster the execution plan is. This cost metric is based on several things, like the size of the tables involved in the query and the expected time that certain operations may take (like JOIN or ORDER BY).

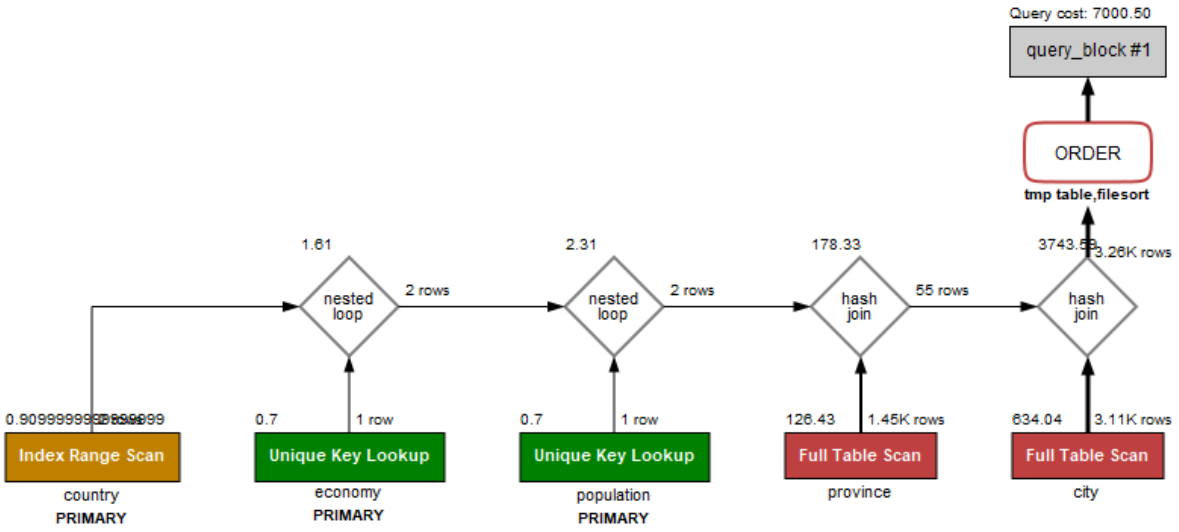
Complex Operations

For more complex operations the optimizer uses *materialization* for more efficient subquery processing. This is, creating a temporary table from a subquery, that is stored usually in memory. It also features Semi-Join transformations, which can be used when we only want to know if there is a match in a query, instead of how many matches, this lowers execution time. Currently it's parallel query support is very limited, only applying to index reading and queries involving *count(*)* and *check table* statements. These options are only controlled by the system and the user cannot choose when they are used. If we compare this to Oracle 18c, Oracle's system is more complex and features parallel queries for more operations.

However, there are some options that can be chosen. When computing join operations, the user can choose to use *Block Nested Loops* or not with the BNL and NO_BNL optimizer hints. Older versions may use *hash_join=on* or *hash_join=off* as part of the setting for the *optimizer_switch* server system variable. The Oracle system however, also has a *Merge Join* algorithm which is useful when the tables are already sorted by the join key. Oracle also lets us choose the order of the join operations while MySQL does not.

MySQL features a very useful analysis tool, these are diagrams that are called EXPLAIN statements or *Execution Plans*. They show a detailed description of the manner in which the query will be processed. It tells us the order of the operations, which algorithms were used and also the cost of processing every step. Hovering the mouse over any of these operations will provide more details about them, and it even provides an explanation when the costs are very high, as to why this might be. Comparing this analysis tool, to the one featured in Oracle 18c, this one looks simpler and easier to understand because it features diagrams instead of hierarchies of text, and because it offers some explanations, a feature that is lacking in Oracle's equivalent tool.
An example of a query to see the analysis tool:

```
SELECT *
FROM country INNER JOIN province ON country.Code = province.Country
INNER JOIN city ON country.Code = city.Country
INNER JOIN economy ON country.Code = economy.Country INNER JOIN
population ON country.Code = population.Country
WHERE country.Code = 'BR' OR country.Code = 'A' ORDER BY
country.Code;
```



Transaction Management and Concurrency Control

ACID Principles

MySQL features statements that provide control over *Transactions*. *Transactions* are a set of operations that if any of them fail, the system rolls back to a previous state and no changes are committed. If all operations succeed, all changes are committed. This means that a transaction allows for a database to never store the results of partial operations. It does this by respecting the ACID properties:

- *Atomicity* - Either all statements in a transaction are executed, or none of them are.
- *Durability* - Once the system announces that the transaction is complete, the changes committed must persist even if there are hardware or software failures.
- *Consistency* - After the transaction is complete, all consistency requirements must be satisfied.
- *Isolation* - If multiple transactions are executed simultaneously, they all must execute as if they were executing completely independently. This means that the resulting state must be the same no matter the order of transactions that was executed.

Here is an example of what a Transaction may look like:

```
START TRANSACTION;
SELECT @mort:= MAX(Infant_Mortality) FROM population;

INSERT INTO country
VALUES ('TestMort','AAT','test','test',10,10);

INSERT INTO population
VALUES ('AAT', 10, 10);

COMMIT;
```

The MySQL system does not support nested *Transactions* because when you issue a START TRANSACTION command it implicitly commits any current transaction.
It does support long duration transactions, which only assure the consistency and durability principles. This is similar in Oracle 18c.

Isolation Levels

It supports the 4 standard SQL Isolation levels: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, and SERIALIZABLE. The user can set a transaction to the needed level with the SET TRANSACTION command. Also, the server settings can be configured to change the default isolation level with the *--transaction-isolation* option.

1. The default Isolation level is REPEATABLE READ. This level returns consistent reads in relation to the same transaction, that are based on a snapshot that was created by the first read. This means that several SELECT statements will be consistent with each other.

2. The level with highest security is SERIALIZABLE, in this level all transactions must be executed in some order (no concurrency) but in MySQL, this level gives the same results as REPEATABLE READ.

3. READ COMMITTED makes it so that each read operation sets and reads its own snapshot. This means that phantom row problems may occur, as other transactions can insert new rows into the table, which will be present if they are committed.

4. The level with least security is READ UNCOMMITTED, which lets the system read changes that are currently being made and haven't even been committed.

The only difference between MySQL's isolation levels and Oracle's is that Oracle also supports SNAPSHOT ISOLATION, which allows writing on the copy of the data, allowing other operations to continue without locking the table.

These Isolation levels are implemented using lock-based protocols supported by *InnoDB*, the storage engine used by MySQL. It implements shared and exclusive locks to manage concurrent operations. These locks may introduce a deadlock situation, which happens when different transactions hold a lock that the other one needs, staying stuck and not progressing. The documentation recommends using transactions instead of LOCK TABLE statements and to make transactions small, so they don't hold locks for a very long time. However, if things fail, the system has an option called *innodb_deadlock_detect* (which is enabled by default). When *InnoDB* detects a deadlock it will rollback one of the transactions. If the deadlock detection in disabled the system relies on the *innodb_lock_wait_timeout* setting to roll back a transaction anyway. This means that the user must also handle the cases where a rollback can happen after a transaction.

In *InnoDb*, all user activity must end either with a COMMIT or a ROLLBACK. A commit is keeping the changes, but a ROLLBACK is reverting the database to the state before the transaction began. To accomplish this, MySQL stores a copy of the original state, in an area of the database called the *rollback segment*. Then, if a rollback is requested, the system can retrieve the original state and revert the changes that have been made.

Related to user control MySQL *InnoDB* provides support for the statements SAVEPOINT, ROLLBACK TO SAVEPOINT and RELEASE SAVEPOINT. These statements are to be used inside transactions, the user can create a savepoint by giving it a name inside the transaction and can then return to this state with the ROLLBACK TO SAVEPOINT command. The savepoint can also be deleted with RELEASE SAVEPOINT. The same savepoint feature is present in Oracle 18c.

Here is an example of rolling back in a query, only the first insert will actually be committed:

```
START TRANSACTION;

INSERT INTO country
VALUES ('Rollback','AAR','roll','roll',10,10);

SAVEPOINT mysave;

INSERT INTO population
VALUES ('AAR', 10, 10);

ROLLBACK TO SAVEPOINT mysave;

COMMIT;
```

Consistency Levels

MySQL offers the option to configure transaction consistency levels. There are 5 levels available:

- EVENTUAL - Neither Read-Only nor Read-Write transactions wait for preceding transactions to be applied before executing.
- BEFORE_ON_PRIMARY_FAILOVER - Read-Only and Read-Write transactions are held until any backlog has been applied.
- BEFORE - Both Read-Only and Read-Write transactions wait for all preceding transactions to be processed.
- AFTER - Read-Write transactions wait until their changes have been applied to all members. This makes sure that any subsequent transactions reads the most recent value.
- BEFORE_AND_AFTER - Waits for all preceding transactions to complete and waits until Read-Write changes have been applied to other members.

The user can change this setting by attributing to group_replication_consistency one of possible consistency levels. Oracle 18c offers the option of choosing a transaction-level read consistency. If a transaction runs in the serializable level, its data accesses will reflect the state of the database when the transaction started. The only changes seen are the ones made by its own transaction. Transaction-level read consistency provides repeatable reads and doesn't allow for phantom reads.

Granularity

MySQL supports 3 different levels of granularity for its locks. These locks can be:

- *Page-Level* - Deprecated lock that would be associated with the contents of one particular page.
- *Table-Level* - Simple lock, requires little memory, but may make parallel tasks very slow.
- *Row-Level* - Locks particular rows so it uses more memory but since it blocks less content, it will better handle parallelism.

But not all of these can be used with the default storage engine, *InnoDB* only allows for Row-Level locks. *InnoDB's* lock granularity is different from the other storage engines available to MySQL because only one session can update the tables when using table-level locking. When we use row-level locking, several sessions can write to the same table, making this option the best one in most cases. Oracle 18c performs what is called lock conversion which converts a table lock of lower restrictiveness to one of higher restrictiveness if necessary.

Distributed Database Support

MySQL doesn't support distributed databases because it is designed as a centralized database, but it can be used as a distributed database when using MySQL Cluster, which is a distributed database that combines linear scalability and high availability.

Replication

In distributed databases, replication enables data from one MySQL database server to be copied to another MySQL database server. This allows replication of all databases, selected databases or just selected tables within a database.

There are 2 core types of replication format, Statement Based Replication (SBR), which replicates the entire SQL statement. Row Based replication (RBR), which replicates only the changed rows.

Replication in MySQL has several advantages like:

- Scale-out solutions - spreading the load among multiple replicas to improve performance.
- Data security - Run backup services on replica without corrupting the source data.
- Analytics - It is possible to analyze the live data without affecting the performance of the source.
- Long-distance data distribution - Possibility of creating local copy of data for remote site, to use without accessing the source.

There are several types of replication:

- Master-slave replication - Single Master server that accepts reads and writes and one or more read-only Slave servers. Slave servers asynchronoysly replicate data from master server.
    - Advantages:
        - Very fast.
        - It is possible to split read and write requests to different servers.
    - Disadvantages:
        - Because it is asynchronous replication, it is not very reliable.
        - Write requests can hardly be scaled.
- Master-master replication - Two or more masters that accept reads and writes and each one has multiple slave nodes. Replication between master nodes is asynchronous.
    - Advantages:
        - Possible to scale write requests via adding additional master nodes.
        - Low chance of all master nodes failing simultaneously.
    - Disadvantages:
        - If master nodes fail, some transactions may be lost due to asynchronous replication.

        - It is not possible to ensure that each master node's backups contain the same data.
    - Group replication - Creates fault-tolerant systems with redundancy that ensure the cluster is still available even if the server fails.
        - Advantages:
            - In case the Master node fails, a new Master is elected in their group.
            - Possible to scale both reads and writes by adding new Master and Slave nodes.
        - Disadvantages:
            - Each group is limited to 9 nodes.
    - Multi-master cluster - Synchronous cluster of databases with several master nodes based on synchronous replication.
        - Advantages:
            - Guarantees data safety using "quorum" protocol.
            - Very fast read request and can be scaled efficiently.

In Oracle, replication supports Multi-master Replication,which we saw before, and Snapshot Replication, a snapshot(read-only or updatable) that contains a complete or partial copy of a target master table that is locally accessible by the client.

Fragmentation

Fragmentation happens when the database has many deletes and consequently leaves holes in tables or blocks which are fully read when scanning the table leaving it less efficient. To optimize it 's possible to use the `optimize table TABLE_NAME` operation, this is efficient if it is a small database, in case of big databases, MySQL has Master-Master replication which has 2 separate databases, one active and another passive, both identical. To perform the operation it optimizes the passive database first. Once the operation has been completed, switch the passive database to active, and the active to passive and start optimizing the new passive database.

How to connect with another database

To connect another database, it is needed to set up a remote server to possibly accept connections and use `mysql -u username -h mysql_server_ip -p` .

In Oracle it is possible to connect another database using a database link (pointer that defines a one-way communication path from an Oracle database server to another database server) using `CREATE DATABASE LINK … .`

Global Transition

Statement transactions are started for each statement that accesses transactional tables. If it succeeds, it is committed, if it fails, it is rolled back. A statement transaction can be viewed as a transaction which starts with a savepoint which MySQL maintains automatically, making the effects of one statement atomic. It encloses all statement transactions that are issued between its beginning and its end. MySQL supports pluggable storage engine architecture (PSEA) and maintains a transactional state independent for each engine. To commit a transaction, MySQL employs a two-phase commit protocol, same for Oracle.

Global Locks

Databases need to control resource access when concurrent access occurs. And the lock is designed to solve these problems when it happens. Global locks is one of three categories of locks (Global lock, Table-lock and Row-level lock) and the functionality is to lock the entire database instance. MySQL provides a global read lock method, with the command Flush tables With Read Lock. It puts the entire database in a read-only state; This means all the addition, deletion and modification operations of other threads will be blocked, including creating tables and modifying table structures.

It is typically used for a full database logical backup to ensure that no other threads operate on the data.But there is business shutdown risk. The solution is use mysqldump, logical backup toll of the MySQL database, it uses parameters -single-transaction, a transaction with Repeatable reading isolation level to ensure consistent view and possible to back up data without affecting the business, but it requires InnoDB as database engine.

Weak level of isolation

The weakest level of isolation is READ UNCOMMITTED as we saw before. It's not safe to use the Read Uncommitted level of isolation on Statement Replication, because it's possible to get errors on some simple update/delete statements that wouldn't be seen in the default isolation.

Other Features

Cloud deployment

MySQL can also be run on cloud computing platforms such as Microsoft Azure, Amazon EC2, Oracle Cloud Infrastructure.

Virtual machine image

In this implementation, cloud users can upload a machine image of their own with MySQL installed, or use a ready-made machine image with an optimized installation of MySQL on it, such as the one provided by Amazon EC2.

MySQL as a service

In this configuration, application owners do not have to install and maintain the MySQL database on their own. Instead, the service provider takes responsibility for installing and maintaining the database. In this model the database provider takes responsibility for maintaining the host and database.

MyISAM (Alternative to InnoDB)

Storage limits	256TB	Transactions	No	Locking granularity	Table
MVCC (Multiversion concurrency control)	No	Geospatial data type support	Yes	Geospatial indexing support	Yes
B-tree indexes	Yes	T-tree indexes	No	Hash indexes	No
Full-text search indexes	Yes	Clustered indexes	No	Data caches	No
Index caches	Yes	Compressed data	Yes	Encrypted data	Yes
Cluster database support	No	Replication support	Yes	Foreign key support	No
Backup / point-in-time recovery	Yes	Query cache support	Yes	Update statistics for data dictionary	Yes

MEMORY (Alternative to InnoDB)

Storage limits	RAM	Transactions	No	Locking granularity	Table
MVCC	No	Geospatial data type support	No	Geospatial indexing support	No
B-tree indexes	Yes	T-tree indexes	No	Hash indexes	Yes
Full-text search indexes	No	Clustered indexes	No	Data caches	N/A
Index caches	N/A	Compressed data	No	Encrypted data	Yes
Cluster database support	No	Replication support	Yes	Foreign key support	No
Backup / point-in-time recover	Yes	Query cache support	Yes	Update statistics for data dictionary	Yes

References

- Introduction

"MySQL Internals Manual". Dev.mysql.com. 4 March 2009. Retrieved 8 June 2009
Review of MySQL Server 5.0, Techworld.com, November 2005
MySQL Server Review, LinuxMint.com

-Storage and File Structure

https://dev.mysql.com/doc/internals/en/innodb-page-overview.html
https://www.percona.com/blog/2017/04/10/innodb-page-merging-and-page-splitting/
https://duynguyen-ori75.github.io/slotted-page-format/
https://www.w3resource.com/mysql/mysql-storage-engines.php
https://dev.mysql.com/doc/refman/8.0/en/innodb-on-disk-structures.html
https://dev.mysql.com/doc/refman/8.0/en/innodb-buffer-pool.html

-Indexing and Hashing

https://en.wikipedia.org/wiki/RocksDB
https://en.wikipedia.org/wiki/MyRocks
https://blog.toadworld.com/2017/11/15/an-lsm-tree-engine-for-mysql
http://smalldatum.blogspot.com/2016/11/myrocks-use-less-io-on-writes-to-have.html
http://jetware.io/blog/redmine-performance-on-myrocks
https://dev.mysql.com/doc/refman/5.6/en/index-btree-hash.html
https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_hash_index
https://blog.toadworld.com/2017/11/15/an-lsm-tree-engine-for-mysql
https://www.liquidweb.com/kb/mysql-optimization-how-to-leverage-mysql-database-indexing/
https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html

-Query Processing and Optimization

Book: **Understanding MySQL Internals by Sasha Pache**
https://dev.mysql.com/blog-archive/filesort-optimization-in-5-7-3-pack-values-in-the-sort-buffer/
https://dev.mysql.com/doc/refman/8.0/en/optimization.html
https://www.oreilly.com/library/view/understanding-mysql-internals/0596009577/ch09.html

-Transaction Management and Concurrency Control

https://dev.mysql.com/doc/refman/5.7/en/implicit-commit.html
https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html
https://dev.mysql.com/doc/refman/8.0/en/innodb-deadlocks.html
https://dev.mysql.com/doc/refman/8.0/en/group-replication-configuring-consistency-guarantees.html
https://severalnines.com/database-blog/understanding-lock-granularity-mysql
https://docs.oracle.com/en/database/oracle/oracle-database/21/cncpt/data-concurrency-and-consistency.html#GUID-FCE31DFC-474C-4145-9561-29344C02BB3E

-Distributed Database Support

https://dev.mysql.com/doc/mysql-replication-excerpt/8.0/en/replication.html
https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html#isolevel_read-uncommitted
https://hidora.io/resources/pros-and-cons-of-mysql-replication-types-and-how-to-run-them-in-the-cloud/
https://www.databasejournal.com/mysql/mysql-data-fragmentation-what-when-and-how/
https://dev.mysql.com/doc/refman/8.0/en/replication-features-transactions.html
https://en.wikipedia.org/wiki/Quorum_(distributed_computing)
https://developpaper.com/mysql-global-lock-and-table-lock/
https://docs.oracle.com/en/database/oracle/oracle-database/18/admin/distributed-database-concepts.html

-Other Features

https://en.wikipedia.org/wiki/MySQL
https://tech.co/news/math-can-business-benefit-cloud-database-2015-08
https://aws.amazon.com/pt/articles/running-mysql-on-amazon-ec2-with-ebs-elastic-block-store/