Indexes

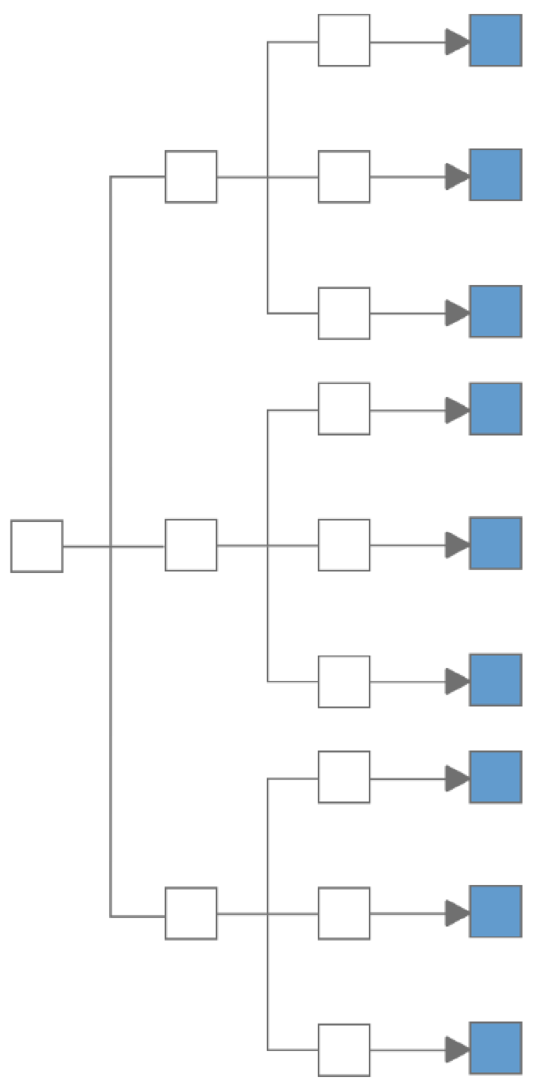
Indexing is the process of creating indexes on one or more columns in a database table.

These indexes allow queries to quickly locate the rows that match a particular condition or set of conditions, without having to scan the entire table.

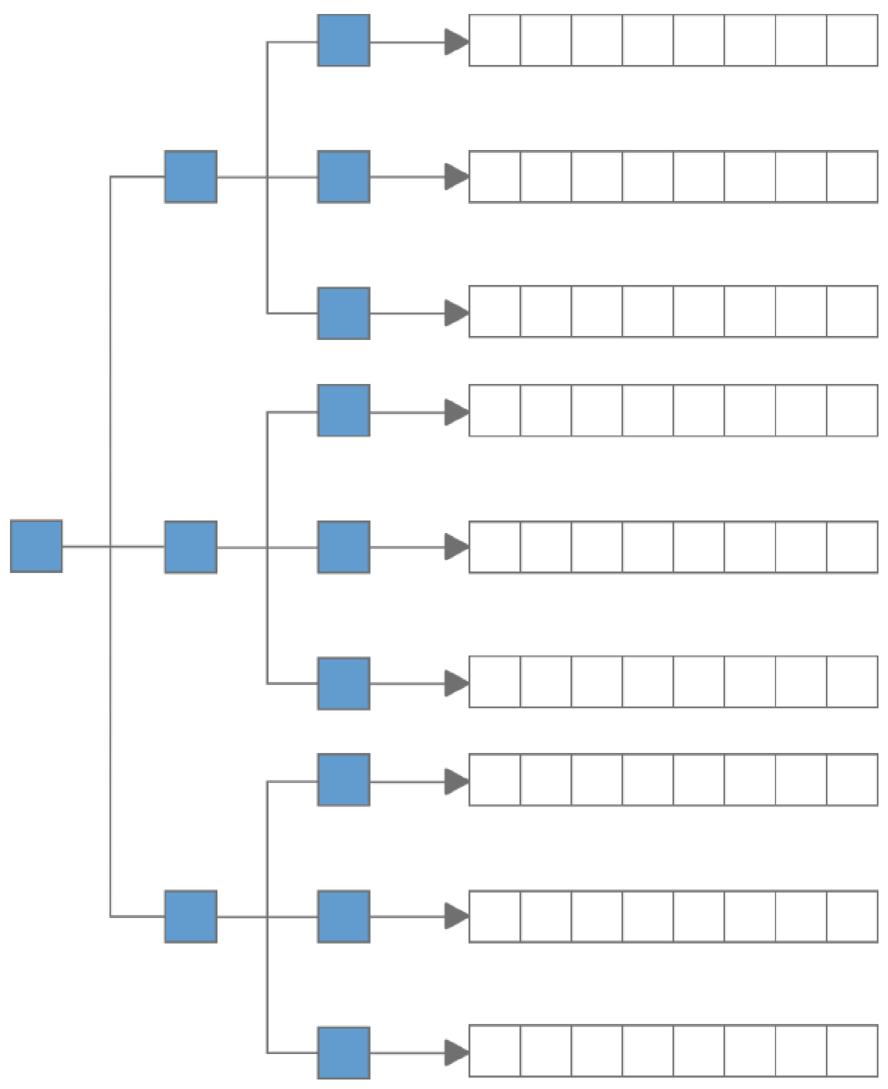
Here are a few key points to keep in mind when using indexing:

There are a few scenarios where MySQL automatically creates an index

- **Primary key:** when you define a primary key on a column, MySQL automatically creates an index on that column.
- **Unique constraints:** When you define a unique constraint on a column or a combination of columns, MySQL automatically creates a unique index on those columns.
- **Foreign keys:** When you define a foreign key constraint on a column or a combination of columns, MySQL automatically creates an index on the referenced column(s) to optimize join operations.
- **Full-text search:** When you define a full-text index on a column, MySQL automatically creates a full-text search index that allows you to search for words or phrases within the indexed column(s).



Secondary indexes



Primary indexes

Indexing Best Practice and Query Optimization

Always have primary key in each table

- In MySQL will create a hidden one if not created explicitly
- Fastest lookup is PK.

Single index with multiple columns

- Format (Left most first and each additional field in a composite key)
- Composite indexes better.
- Because many PK will require more storage.
- And one index scan is faster than two.

```
alter table employee add index idx_id_name_email(id, name, email);
select name, email from employee where id = 2514;
```

- This query more efficient because we index all things that we want.
- As we search by **(id)**, then the first column in the composite index must be **(id)**.
- Once the database filters records based on the **(id)**, the columns we want to select will be retrieved directly from this **(id)** as this **(id)** is attached with the name and the email.

Avoid duplicated indexes (first, second) (second, first) are same

- But take care about the order of the query to use this index in the correct way.
- As if we search based on the (first) column, then the first index is the correct one.
- But if we search based on the (second) column, then the second index is the correct one.

Data types is matter

- only numeric for numbers (don’t store it in string)

Negative clauses and subqueries aren’t as good as positive clauses.

- Not like
- Is not
- Is not null
- Not in

Negative clauses and subqueries can be more difficult to optimize with indexes.

For example, if you have a subquery with a **NOT IN** clause that references a column that is not indexed, MySQL may need to perform a full table scan to evaluate the subquery.

This can be slow if the table is large.

Use INNER join instead of LEFT join as you can.

- As in the **left join**, the database will read and load the full left table and match it with the right table
- But the inner join returns records that have a matching value in the join condition. This can reduce the number of rows that need to be processed and can improve query performance.

We can use index hint.

```
Select * from table use index (ind1) ignore index for order by (ind2) order by a;
```

- By using an index hint, you can ensure that the query optimizer selects the most efficient index for a given query. This can improve query performance and reduce execution time.
- If you have a query with complex conditions or multiple join clauses, the MySQL query optimizer may not choose the optimal index. In these cases, an index hint can help to ensure that the query is executed efficiently.

```
select * from sys.schema_redundant_indexes;
```

- Use this query to know the redundant indexes in the scheme.
- You must remove it to improve the performance.

Primary keys

- Must be as smaller size as possible. **(Smaller in size)**
- Must be as sequential as possible. **(1, 2, 3, ...)**

Indexes are useful in these cases

- Reduce the rows examined.
- Sort data.
- Validate data.
- Enforce that the data are unique
- Find min or max value.

```
select * from sys.schema_unused_indexes;
```

- Use this query to know the unused indexes so that you may delete it.

DBMS decides to use indexes, or sequential scan based on the following

- If needed rows > **30%** from the total rows, then sequential scan is preferred.

★ We can use (**EXPLAIN**) before any statement to view the process of executing this statement.

★ We can use (explain, explain analyze, explain format=json)

Use **LIMIT** and **OFFSET**

- If you are retrieving a large number of rows from a table, consider using the **LIMIT** and **OFFSET** clauses to limit the number of rows returned.
- This can significantly improve query performance, especially when working with large datasets.

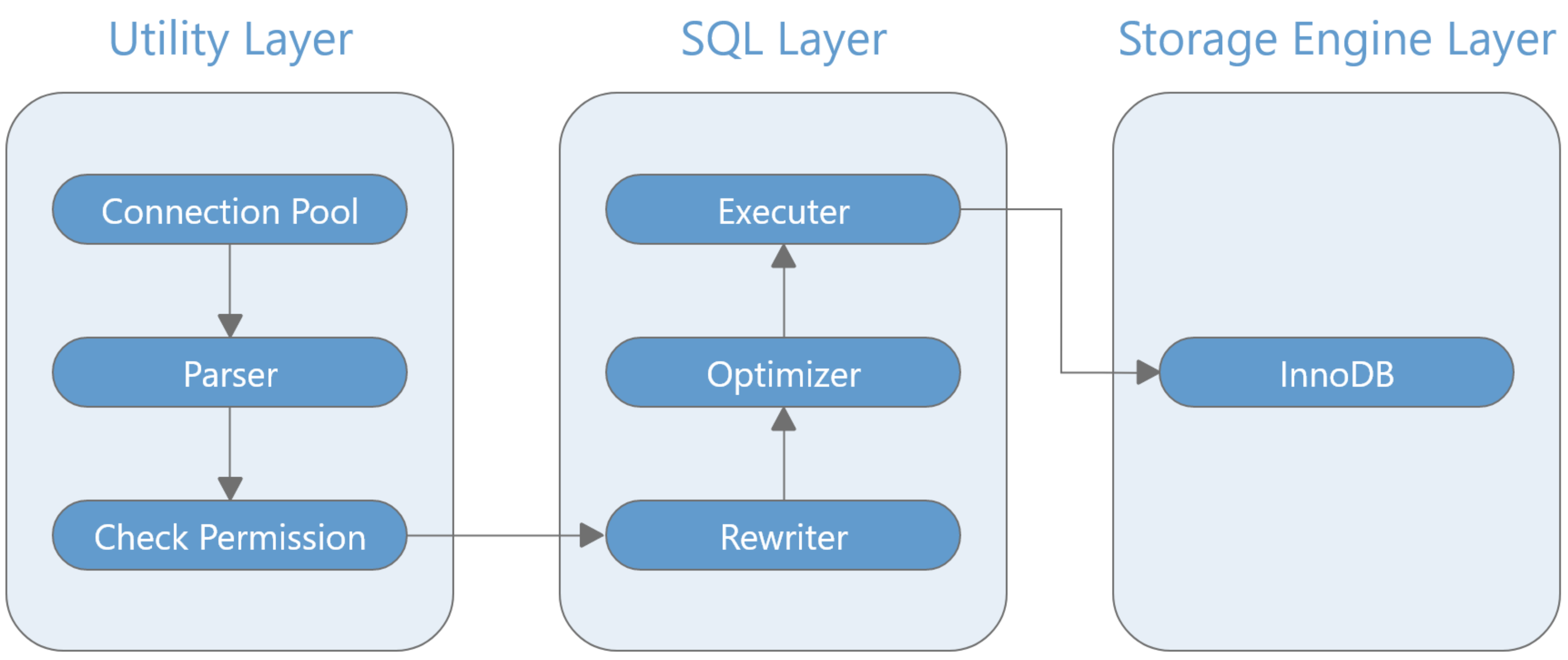
Use **UNION ALL** instead of **UNION**

- If you need to combine the results of multiple queries, consider using **UNION ALL** instead of **UNION**.
- **UNION ALL** is generally faster than **UNION**, as it doesn't remove duplicated rows.

Avoid using **SELECT ***

- When writing queries, try to avoid using the **SELECT *** syntax, as this can be less efficient than specifying the exact columns you need.
- By only selecting the columns you need, you can reduce the amount of data that needs to be transferred and processed, which can improve query performance.

MySQL Logical Architecture



Consists of three groups:

1- Utility layers

Connection pool

- It is recommended to send the related queries in one batch to reduce the load on the database.

Parser

- It validates the query and check for the mistakes.
- It takes the query and parses it to tokens in a token tree.

Example:

```
select name from users where id = 2514;
```

- fields: name
- view: users
- constant: 2514
- It checks permissions.
- It checks the privileges.

2- SQL layer

Rewriter

- It rewrites again the query based on some rules in the MySQL.

Optimizer

Its job is to find the best way to execute the query using some statistics:

- Use the full table scan.
- Use temp table (cached).
- Use index lookup.

- These statistics may be wrong.
- We can help the optimizer to decide the best way using hints.

Executer

- Execute the query received from the optimizer and send it to the storage engine layer

3- Storage Engine Layer

- Is responsible for storing and retrieving all data
- Examples: InnoDB

InnoDB

- InnoDB has a default buffer pool = 128MB
- It acts as cache
- If a request is stored in this buffer, then the buffer returns it directly without retrieve it again from the database
- The buffer stores the new queries if doesn’t exist in this buffer.
- If a new request wants to be stored but the buffer is full, then the InnoDB will remove the least recently used requests.

```
set global innodb_buffer_pool_size = newSize;
```

<> Use this query to change the size of the above buffer.