University Mohammed VI Polytechnic

College of Computing

# Deliverable # 6: Physical Design, Security and Transaction Management

**Data Management Course**

UM6P College of Computing

**Professor:** Karima Echihabi **Program:** Computer Engineering
**Session:** Fall 2025

---

**Team Information**

| | |
|---|---|
| **Team Name** | Leo fl Bernabéu |
| **Member 1** | Yassine SQUALLI-HOUSSAINI |
| **Member 2** | Omar TSOULI |
| **Member 3** | Yahya TALIB |
| **Member 4** | Adam RHYA |
| **Member 5** | Hamza TAALOUCHT |
| **Member 6** | Diaa Eddine ZAINI |
| **Repository Link** | https://github.com/therealzaini/DMG_LAB2_LEO_FL_BERNABEU |

## 1 Index design:

### Indexing strategy:

To optimize query performance on the views UpcomingByHospital, StaffWorkloadThirty, and PatientNextVisit, the following indexing strategies were applied.

*UpcomingByHospital*

• A hash index is created on A.status to speed up equality-based filtering on the status attribute.
• A B+-tree index is created on (C.date, C.CAID), with C.date as the leading column, to efficiently support range queries and ordering on dates.

*StaffWorkloadThirty*

• A B+-tree index on (C.date, C.CAID) is used, with C.date as the leading column, to optimize queries that filter or group clinical activities by date.
• A hash index on A.status is used to accelerate lookups based on activity status.

*PatientNextVisit*

• A hash index on A.status is used for fast equality searches on the status field.
• A B+-tree index on (C.date, C.CAID) supports date-based filtering and ordering.

### Impact on Write operations:

While these indexes significantly improve read performance, they introduce an overhead on insert and update operations. Each insertion requires:

• Additional computations to locate the correct position of C.date in the B+-tree.
• Maintenance of the hash index for A.status.

As a result, write operations are slightly slower due to index maintenance, which is an expected trade-off in physical database design.

### Alternative Indexing Option:

An alternative strategy consists of applying the same indexing scheme uniformly:

• A B+-tree index on (C.date, C.CAID).
• A hash index on A.status.
This option maintains consistency across all views while preserving the same performance trade-offs between read optimization and write overhead.

## 2 Partitions:

**1.** We opted for partitioning by month. Initially, we were concerned that using this type of partitioning might result in a number of partitions exceeding what MySQL can support. After conducting thorough research, we found that the maximum number of partitions supported by MySQL is 8192 (source: https://dev.mysql.com/doc/refman/8.4/en/partitioninglimitations.html ).

**• How this partitioning can speed up queries that filter on recent dates?**

→ Actually with the partition on DATE in the tables stated previously. If our goal is to archive "old" data present in those tables we can directly look in certain partitions that fall into a date_range which we classify as "old" and archive all the rows present there. This will once again prevent the table scans that were necessary to find those rows.

**• Any drawbacks for queries that do not filter on date?**

→ Yes. To illustrate, when we partitioned our tables based on DATE each partition was stored on a different Table space therefore in a different file. So querying data that's not filtered by DATE may include rows from different date ranges equivalently from different partitions so we will add overhead (scanning different files). If we hadn't partitioned our table the worst case would have been a table scan in the same file, now it is multiple scans in different files.

**2.** Let's consider partitioning Stock by HID:

**• Which workloads or queries would benefit from this partitioning?**

→ The queries which will benefit from this partitioning are the ones that filter based on hospitals. To illustrate, any query that searches for rows having "HID = x " for example, will be faster, since it will only go to the partition relevant to the value of the HID.

**• Whether this could create data skew or unbalanced partitions?**

→ After doing some light research on what "data skew" is. Yes, it could create data skew and unbalanced partitions. Suppose, a hospital of HID=x has an overwhelming amount of medications in STOCK, therefore the partition based on HID will be nearly useless if the queries filter mainly on HID = x. Since, that particular partition contains the majority of the rows of STOCK therefore the query will still be expensive scans.

**• How this interacts with joins on HID?**

→ Considering that the joins of other tables (HOSPITAL) on HID will mainly be equi-JOINS, they will be compatible with our portioning, since as stated previously they are HID based partitions so this will speed up the queries *in cases where data skew aren't persistent. Since it avoids table scans.

## 3 Tablespaces/Storage layout:

• Measure the execution time of the query without the index (run it at least three times and report the average time printed by the MySQL client),

– first: 0.8072 ms.
– second: 1.2 ms.
– third: 1.19ms.
– average is: 1.0657ms.

- Create one well chosen secondary index:

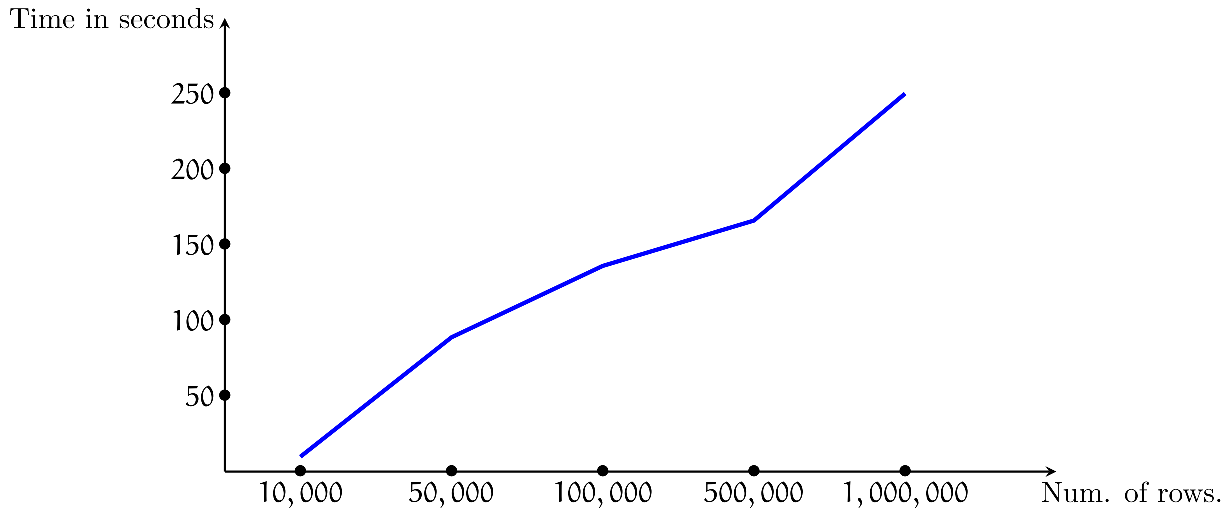
– `CREATE INDEX Clinical_depID ON ClinicalActivity(DEP_ID,DATE);`

- Measure time again using index:

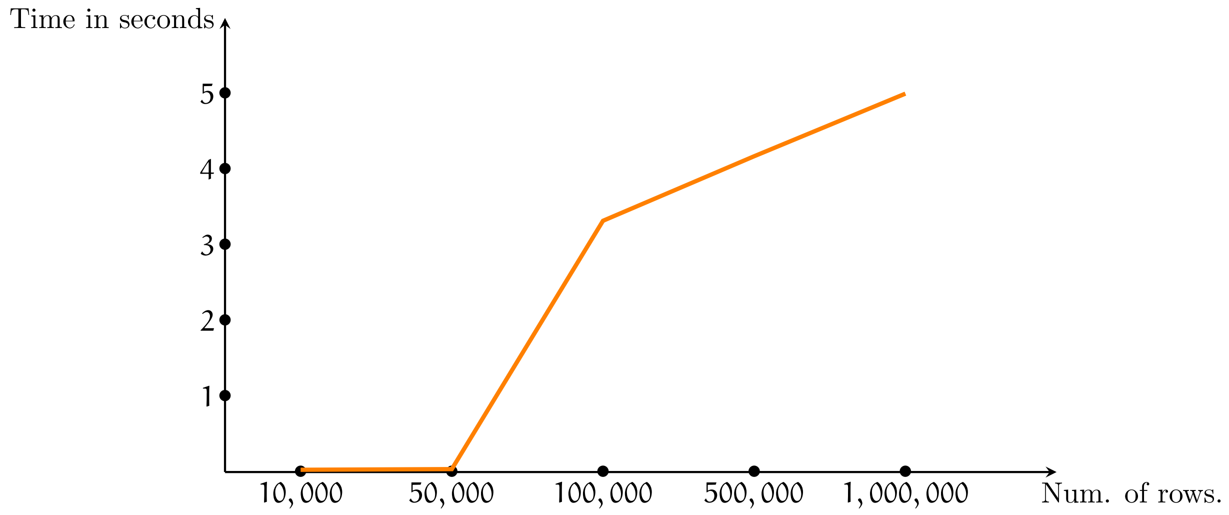
– first: 0.509 ms.
– second: 1.18ms.
– Third: 1.17ms.
– average is : 0.95 ms.

- After some research The query optimizer.

## 4 Tablespaces/Storage layout:



Plot without indexing



Plot with indexing