Database I Lab/ 3rd Grade

[Second Semester] and [2026]

[Lab 1]

[3/2/ 2026] [8.30+10.30 am]

Instructor Information

Instructor
Dr. Rasool Hisham
Dr. Zainab Namh
Assist. Prof. Zahraa Jaaz
Dr. Azhar Flaih

Email
[rasool.hisham@nahrainuniv.edu.iq]

Hours
[2 Hrs]

Indexes tab

The primary objective of this lab is to explore how **indexes** enhance database efficiency. By the end of the lab, you should be able to: **Understand the Concept of Indexes**: Learn how indexes speed up data search.

1. Indexes Tab

Database indexing plays a significant role in this by providing powerful tools to optimize operations and improving query speed in MySQL. Without an index, MySQL must perform a full table scan, reading every row to find the desired data, which becomes increasingly inefficient as the table grows larger.

By creating an index on one or more columns, MySQL can quickly locate the relevant rows, significantly reducing the amount of data that needs to be scanned.

Most MySQL indexes (PRIMARY KEY, UNIQUE, INDEX, and FULLTEXT) are stored in B-trees. A tree data structure that is popular for use in database indexes. The structure is kept sorted at all times, enabling fast lookup. Because B-tree nodes can have many children, a B-tree is not the same as a binary tree, which is limited to 2 children per node.

Here's an example to illustrate the difference between a non-indexed column and an indexed column when retrieving information from the friends table for the name "ZACK":

```
SELECT * FROM friends WHERE name = 'Zack';
```

friends		
id	**name**	**city**
1	Matt	San Francisco
2	Dave	Oakland
3	Andrew	Blacksburg
4	Todd	Chicago
5	Blake	Atlanta
6	Evan	Detroit
7	Nick	New York City
8	Zack	Seattle

If the table was ordered alphabetically, searching for a name could happen a lot faster because we could skip looking for the data in certain rows. If we wanted to search for "Zack" and we know the data is in alphabetical order we could jump down to halfway

Instructor Information

Instructor	Email	Hours
Dr. Rasool Hisham Dr. Zainab Namh Assist. Prof. Zahraa Jaaz Dr. Azhar Flaih	[rasool.hisham@nahrainuniv.edu.iq]	[2 Hrs]

through the data to see if Zack comes before or after that row. We could then half the remaining rows and make the same comparison.

SELECT * FROM friends WHERE name = 'Zack';

friends_name_asc	
Name	**Index**
Andrew	3
Blake	5
Dave	2
Evan	6
Matt	1
Nick	7
Todd	4
Zack	8

This took 3 comparisons to find the right answer instead of 8 in the unindexed data.

Let's look at the index from the previous example and see how it maps back to the original Friends table:

Index

friends_name_asc	
name	**id**
Andrew	3
Blake	5
Dave	2
Evan	6
Matt	1
Nick	7
Todd	4
Zack	8

points to

Table

Friends		
id	**name**	**city**
1	Matt	San Francisco
2	Dave	Oakland
3	Andrew	Blacksburg
4	Todd	Chicago
5	Blake	Atlanta
6	Evan	Detroit
7	Nick	New York City
8	Zack	Seattle

We can see here that the table has the data stored ordered by an incrementing id based on the order in which the data was added. And the Index has the names stored in alphabetical order.

Database I Lab/ 3rd Grade

[Second Semester] and [2026]

[Lab 1]

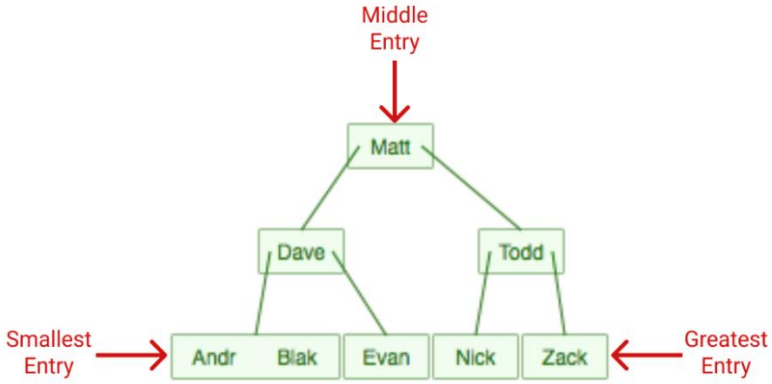
[3/2/ 2026] [8.30+10.30 am]

Instructor Information

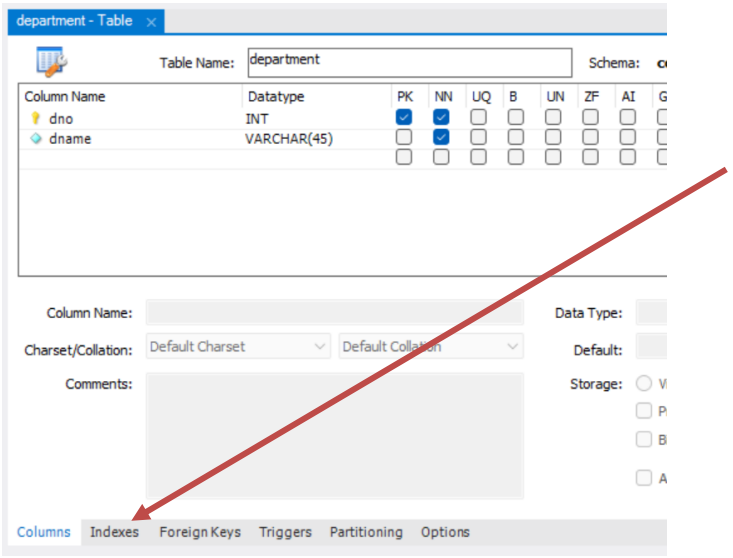
Instructor
Dr. Rasool Hisham
Dr. Zainab Namh
Assist. Prof. Zahraa Jaaz
Dr. Azhar Flaih

Email
[rasool.hisham@nahrainuniv.edu.iq]

Hours
[2 Hrs]

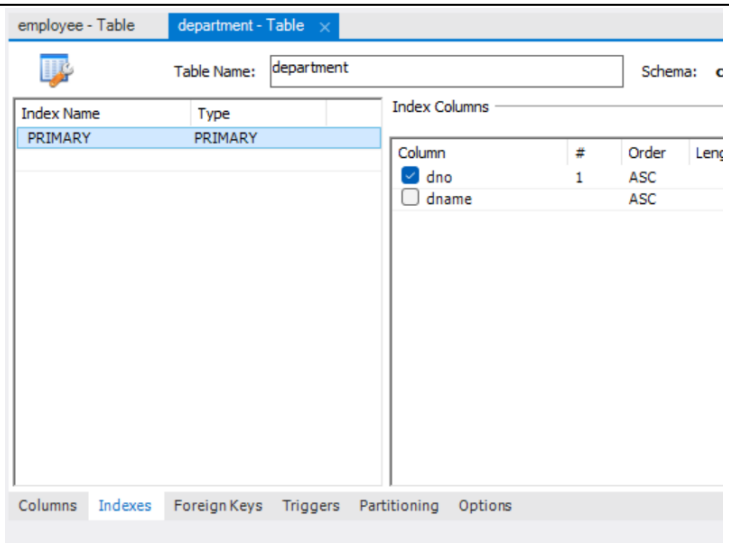


In workbench, the **Indexes** subtab contains all of the index information for your table. Use this subtab to add, drop, and modify indexes. The following figure shows an example of the layout with the **PRIMARY** index of the department table selected and both the index columns and index options shown.



Instructor Information

Instructor	Email	Hours
Dr. Rasool Hisham Dr. Zainab Namh Assist. Prof. Zahraa Jaaz Dr. Azhar Flaih	[rasool.hisham@nahrainuniv.edu.iq]	[2 Hrs]



2. Indexes Types

MySQL offers several types of indexes, each designed to optimize query performance for different use cases. Understanding these MySQL indexes and their purposes can help you select the best one for your needs.

- **Primary key**

A primary key index ensures each row in a table can be uniquely identified, which is crucial for database integrity. This index enforces the uniqueness of the column(s) it covers and prevents duplicate entries.

- **Unique**

A unique index is similar to a primary key but allows for NULL values while ensuring non-NULL values are unique. This type of index helps maintain data uniqueness without the strict constraints of a primary key.

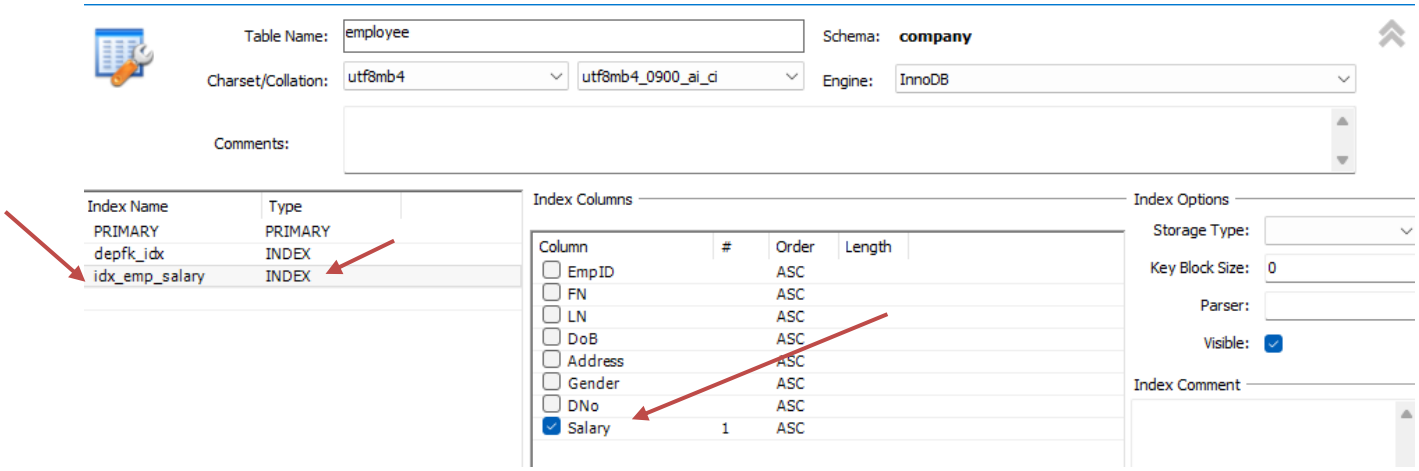
- **Index**

A standard index, also known simply as an index, speeds up searches on frequently queried columns. This type of index can be applied to any column and is beneficial for improving the performance of SELECT queries. By default, the foreign key is set as an index.

Instructor Information

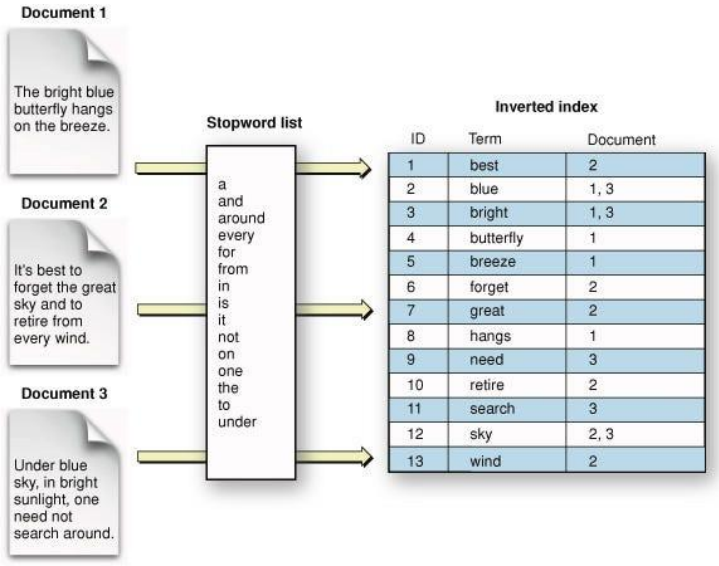
Instructor	Email	Hours
Dr. Rasool Hisham Dr. Zainab Namh Assist. Prof. Zahraa Jaaz Dr. Azhar Flaih	[rasool.hisham@nahrainuniv.edu.iq]	[2 Hrs]

Now set the **salary** column in the **employee** table as an index:



- **Full-text**

Full-text indexes are specialized for full-text search functionalities, allowing efficient text search and retrieval operations. Full-text indexes are created on text-based columns (CHAR, VARCHAR, or TEXT columns) to speed up queries. InnoDB full-text indexes have an inverted index design. Inverted indexes store a list of words, and for each word, a list of documents that the word appears in.



Database I Lab/ 3rd Grade

[Second Semester] and [2026]

[Lab 1]

[3/2/ 2026] [8.30+10.30 am]

Instructor Information

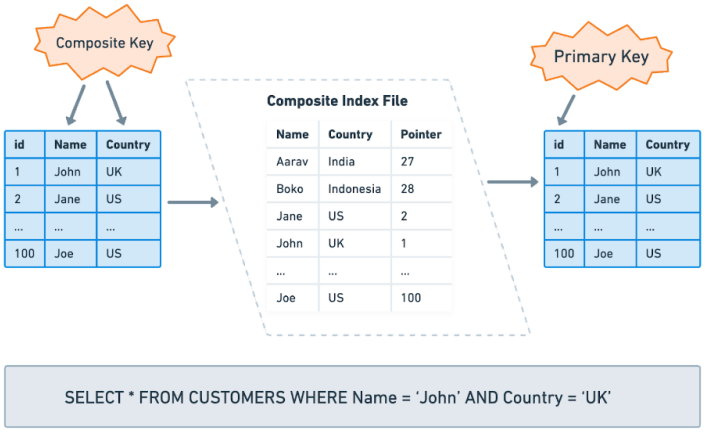
Instructor
Dr. Rasool Hisham
Dr. Zainab Namh
Assist. Prof. Zahraa Jaaz
Dr. Azhar Flaih

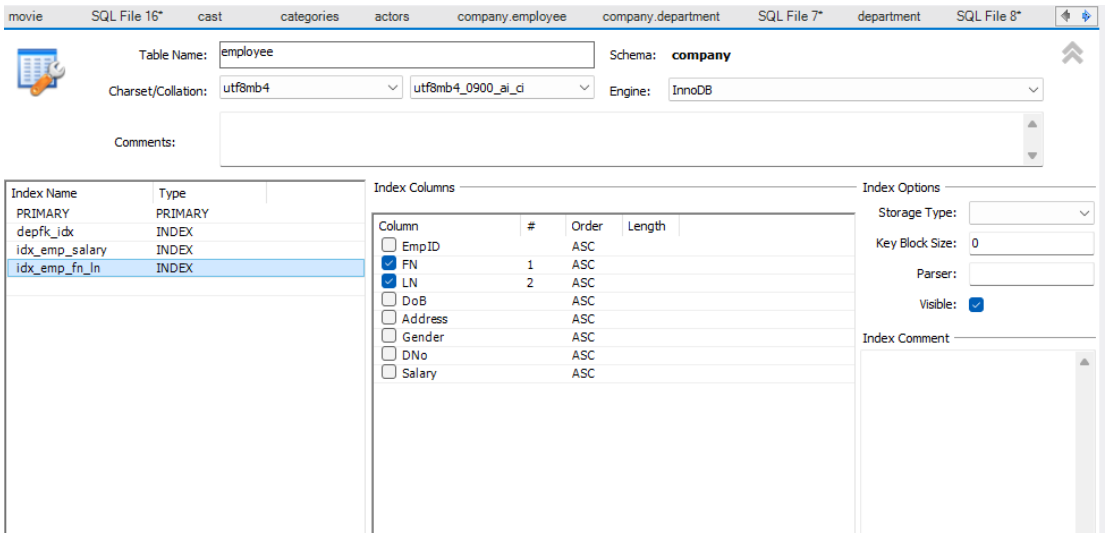
Email
[rasool.hisham@nahrainuniv.edu.iq]

Hours
[2 Hrs]

Composite indexes

Composite indexes cover multiple columns, optimizing queries that filter or sort by more than one column. They provide quick access to combined data and are useful for complex query scenarios. This can be achieved by selecting both columns when creating the index.






```
ALTER TABLE `employee`
ADD INDEX `idx_emp_fn_ln` (`FN`, `LN`) VISIBLE;
```

Index Naming Convention: idx_tablename_columnname