Assignment, DB architect

Please, find below the 3 tasks that we prepared for you:

1. We are planning a role-based access management system for Virta's admin panel. Access management works in the following way: there are users who have access to the data of certain organizations. In addition to organization-level access, each user has access to different modules in the admin panel. Modules allow you to access certain features, like charging station data, EV driver data, energy management, etc. Both things (organization access and module access) are dynamic, meaning they can change quite often. When a user logs in to the admin panel we need to know both which organizations the user has access to, as well as which modules the user has access to. For example, if you had access to an organization called "Virta Nordics" and a module called "EV Drivers", you could see all the EV driver data in the admin panel from the Virta Nordics -organization. Data size at the moment is approximately 20 k organizations, 10 k users, and 50 modules. We are expecting heavy growth in the numbers in the future.

Task 1: plan a database structure that will support the functionality we described. You can decide how you want to document and present this plan.

2. Virta's charging station data is saved to a DB table called Station. This table has static data like the charging station's name, GPS coordinates, owner, etc. The table also has some dynamic data that changes quite often like the station's online/offline status, and timestamp which is the last time we have received some messages from the station. Station table is linked to a seller-table (seller = company that owns the station). We have quite often a need to query and update the Station-table based on different criteria, for example: get all the stations that are online or offline, get all stations of a certain seller, get stations at certain coordinates, etc. The size of the station table is at the moment about 100 k rows, and we are expecting heavy growth in the future.

Task 2: we are now experiencing performance issues with this table, different DB operations are slow and they may cause big load to the DB. How would you start investigating what causes this slowness, and what would be your first recommendations based on the info provided above on how to improve the database performance?

3. Virta's database has some bigger tables with millions of rows of data. We need to sometimes add a new index to those tables to improve the performance of different queries. However, we have had many incidents where adding a new index can lock database tables for a long time, in the worst case for some hours. When a table is locked, none of the operative systems can use them, and this can cause huge problems throughout the whole system.

Task 3: how would you predict in advance whether or not some operation like adding an index will cause downtime for the system? Or would there be a way to avoid downtime in situations like this?

Tips:

• We appreciate practical, real-life solutions. We also expect you to communicate your solution and the reason behind your choices.

• We do not have a requirement on tools you use to communicate, feel free to use diagrams, drawings, code, presentations, and whatever else you feel comfortable with.

• Please submit your solution via email to our Sr TA Specialist. If you wish to submit code as part of your solution, please use a private git repository provider and send us the invites to acces it.

• Each of those three shouldn't take too much time.

• The deadline is 1 week, if you need more time please, inform our Sr. TA Specialist, Rafaela Tomasi via email.

Good luck and if you have any questions, please let us know!

Thank you in advance.

Task 1: plan a database structure that will support the functionality we described. You can decide how you want to document and present this plan.

Solution: We'll be using MySQL Workbench 8.0.33 Community Edition and we'll design a database structure that consists of the following tables (MySQL lower case notations):

tbl_users: This table would store information about each user, including their username, password, and any other relevant details. Each user would have a unique ID (Primary Key PK) that could be used to link them to the other tables in the database.

tbl_organizations: This table would store information about each organization, including its name and any other relevant details. Each organization would have a unique ID that could be used to link it to the other tables in the database.

tbl_modules: This table would store information about each module in the admin panel, including its name and any other relevant details. Each module would have a unique ID that could be used to link it to the other tables in the database.

tbl_organization_access: This table would store information about which users have access to which organizations. It would have foreign keys linking to the Users and Organizations tables, as well as a boolean value indicating whether the user has access to the organization.

tbl_module_access: This table would store information about which users have access to which modules. It would have foreign keys linking to the Users and Modules tables, as well as a boolean value indicating whether the user has access to the module.

We choose a specific naming convention for naming tables, as a type of database objects and this is using the prefix “tbl_” to easily know to what database objects the names refer to (tables in this case)

Using this database structure, we can easily query the database to determine which organizations and modules a user has access to when they log in to the admin panel.

These are the tables structures we design for this database schema:

tbl_users:

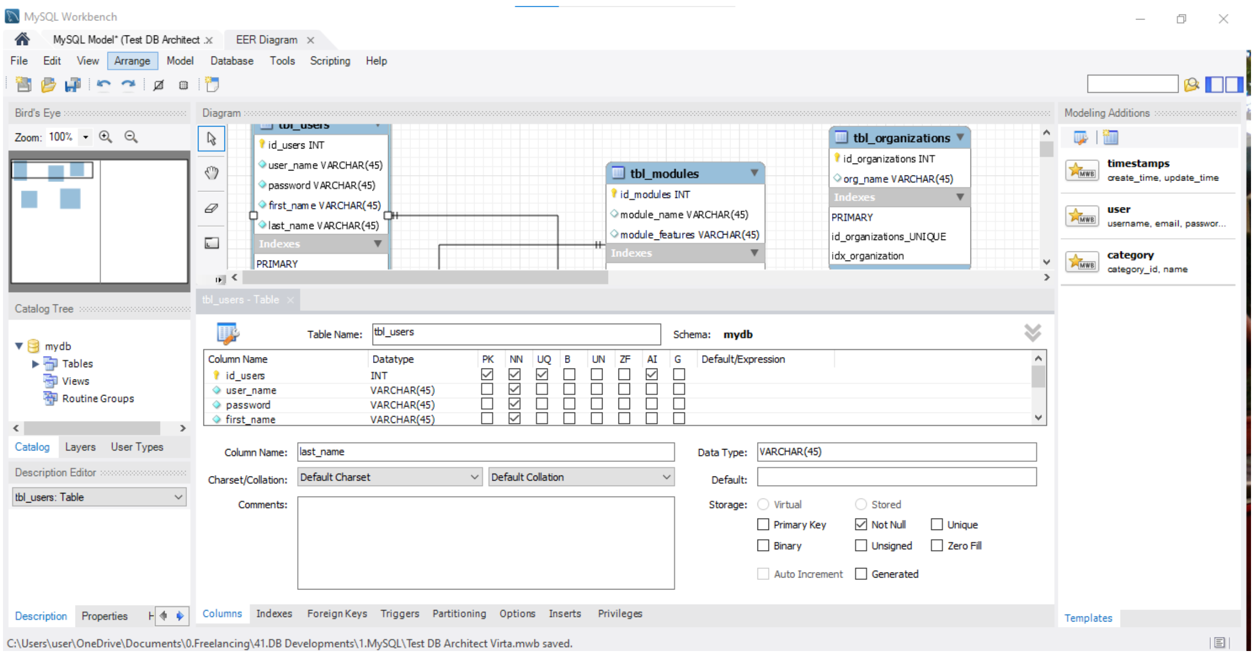


Table Name: tbl_users   Schema: mydb

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id_users	INT	✓	✓	✓				✓		
user_name	VARCHAR(45)		✓							
password	VARCHAR(45)		✓							
first_name	VARCHAR(45)		✓							

Column Name: last_name   Data Type: VARCHAR(45)
Charset/Collation: Default Charset   Default Collation
Default:
Comments:
Storage: Virtual / Stored
[✓ Not Null]

C:\Users\user\OneDrive\Documents\0.Freelancing\41.DB Developments\1.MySQL\Test DB Architect Virta.mwb saved.

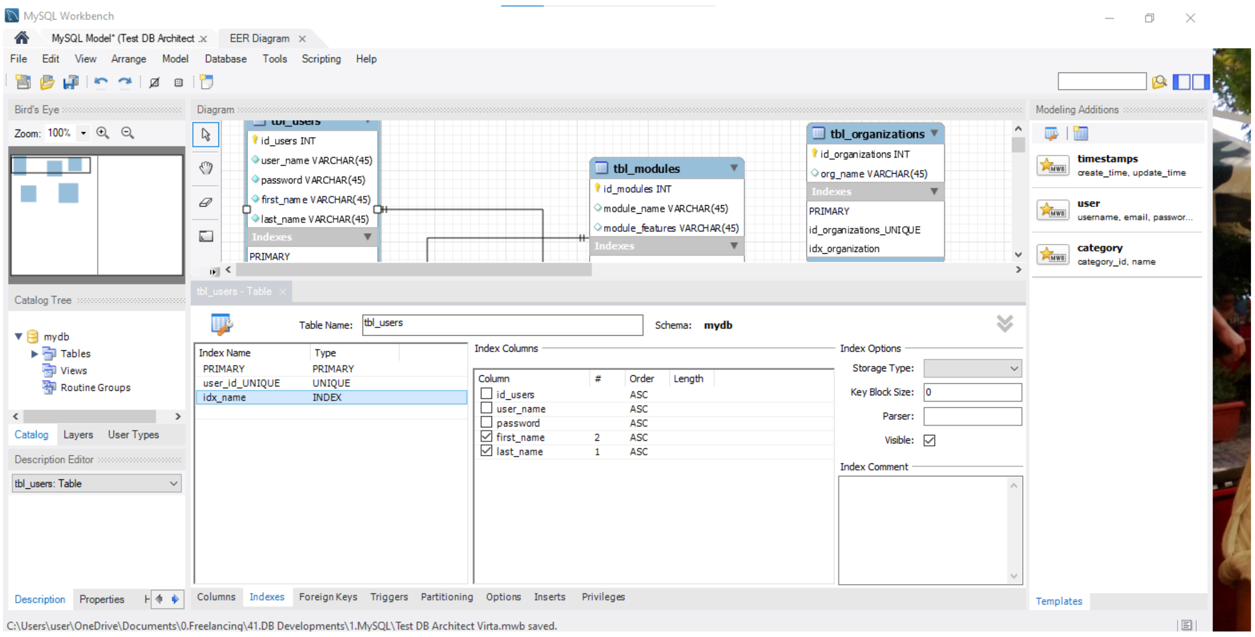


Table Name: tbl_users   Schema: mydb

Index Name	Type
PRIMARY	PRIMARY
user_id_UNIQUE	UNIQUE
idx_name	INDEX

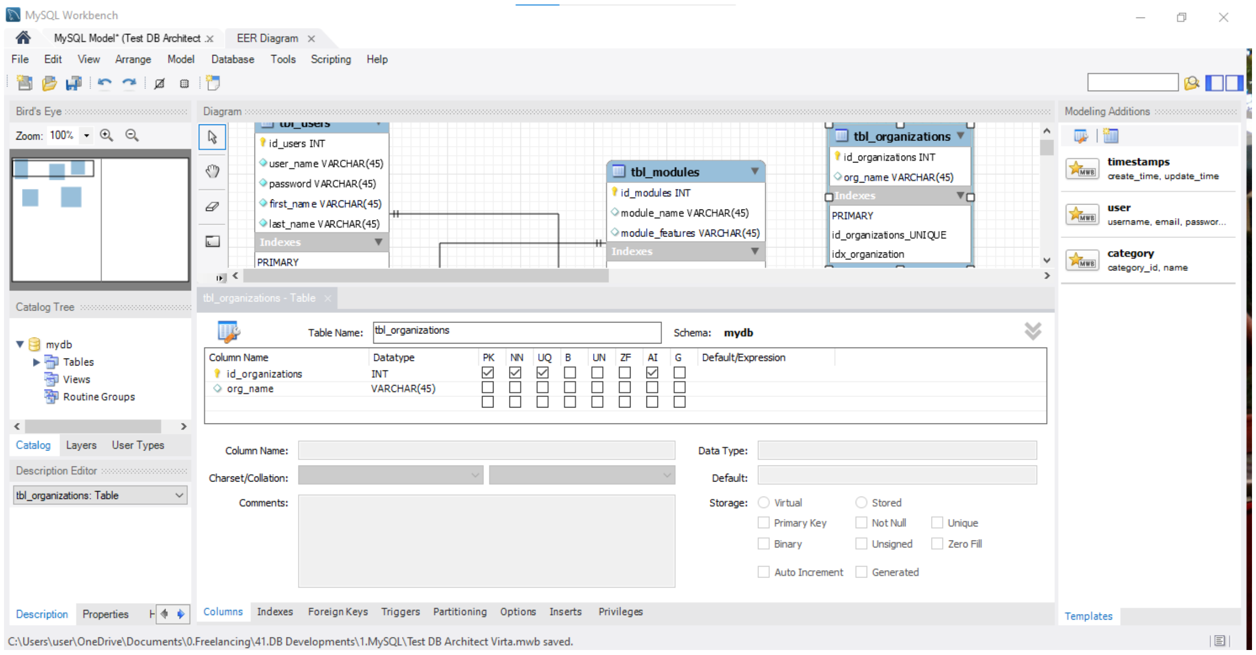
Index Columns

	Column	#	Order	Length
☐	id_users		ASC	
☐	user_name		ASC	
☐	password		ASC	
✓	first_name	2	ASC	
✓	last_name	1	ASC	

Index Options
Storage Type:
Key Block Size: 0
Parser:
Visible: ✓
Index Comment

C:\Users\user\OneDrive\Documents\0.Freelancing\41.DB Developments\1.MySQL\Test DB Architect Virta.mwb saved.

tbl_organizations:


MySQL Workbench
MySQL Model* (Test DB Architect
EER Diagram
File Edit View Arrange Model Database Tools Scripting Help
Bird's Eye
Zoom: 100%
Diagram
Modeling Additions
Catalog Tree
Catalog Layers User Types
Description Editor
Description Properties
Columns Indexes Foreign Keys Triggers Partitioning Options Inserts Privileges
Templates

tbl_users: id_users INT, user_name VARCHAR(45), password VARCHAR(45), first_name VARCHAR(45), last_name VARCHAR(45); Indexes: PRIMARY

tbl_modules: id_modules INT, module_name VARCHAR(45), module_features VARCHAR(45); Indexes

tbl_organizations: id_organizations INT, org_name VARCHAR(45); Indexes: PRIMARY, id_organizations_UNIQUE, idx_organization

timestamps: create_time, update_time
user: username, email, passwor...
category: category_id, name

Catalog Tree: mydb — Tables, Views, Routine Groups

Description Editor: tbl_organizations: Table

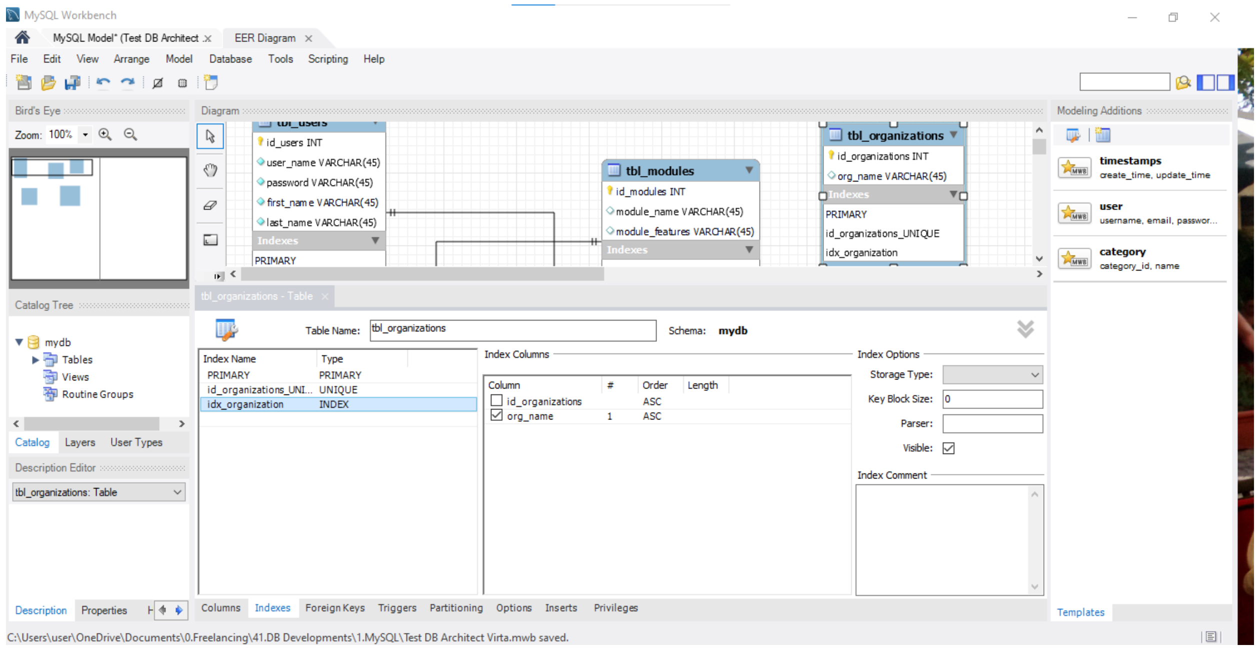
tbl_organizations - Table

Table Name: tbl_organizations   Schema: mydb

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id_organizations	INT	✓	✓	✓				✓		
org_name	VARCHAR(45)									

Column Name: | Data Type: | Charset/Collation: | Default: | Comments: | Storage: Virtual, Stored, Primary Key, Not Null, Unique, Binary, Unsigned, Zero Fill, Auto Increment, Generated

C:\Users\user\OneDrive\Documents\0.Freelancing\41.DB Developments\1.MySQL\Test DB Architect Virta.mwb saved.


MySQL Workbench
MySQL Model* (Test DB Architect
EER Diagram
File Edit View Arrange Model Database Tools Scripting Help
Bird's Eye
Zoom: 100%
Diagram
Modeling Additions
Catalog Tree
Catalog Layers User Types
Description Editor
Description Properties
Columns Indexes Foreign Keys Triggers Partitioning Options Inserts Privileges
Templates

tbl_organizations - Table

Table Name: tbl_organizations   Schema: mydb

Index Name	Type
PRIMARY	PRIMARY
id_organizations_UNI...	UNIQUE
idx_organization	INDEX

Index Columns

Column	#	Order	Length
☐ id_organizations		ASC	
✓ org_name	1	ASC	

Index Options
Storage Type:
Key Block Size: 0
Parser:
Visible: ✓
Index Comment

C:\Users\user\OneDrive\Documents\0.Freelancing\41.DB Developments\1.MySQL\Test DB Architect Virta.mwb saved.

tbl_modules:

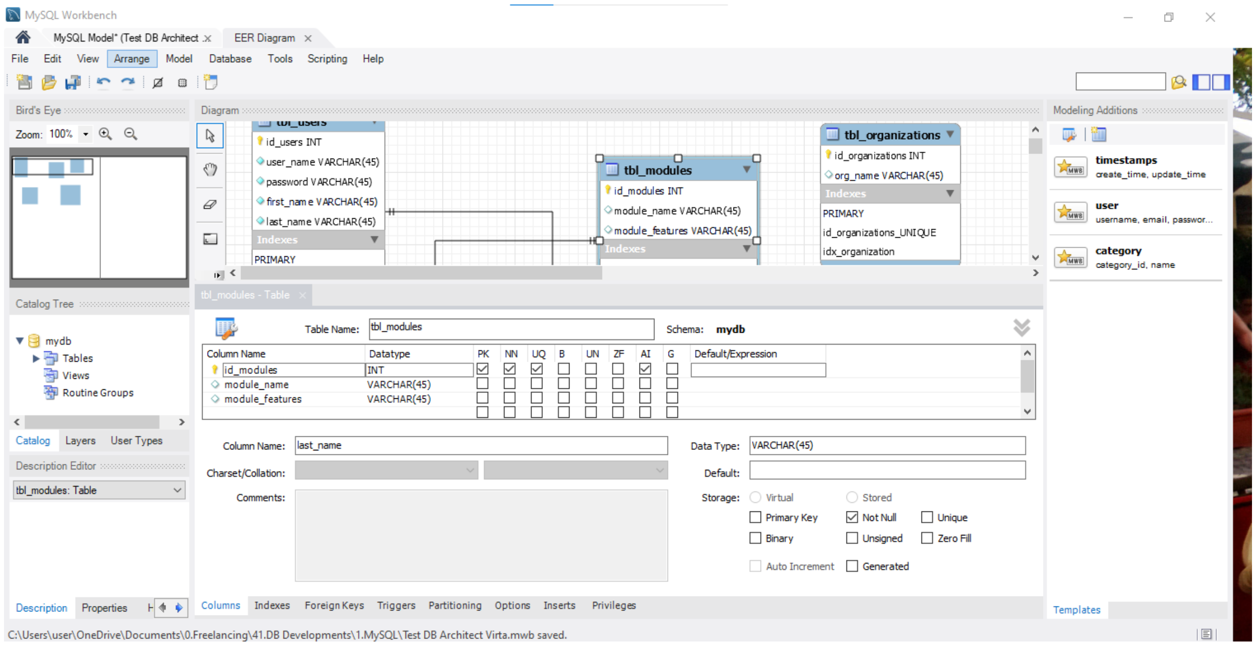


Table Name: tbl_modules Schema: mydb

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id_modules	INT	✓	✓	✓				✓		
module_name	VARCHAR(45)									
module_features	VARCHAR(45)									

Column Name: last_name Data Type: VARCHAR(45)

Storage: Virtual / Stored; Primary Key [ ], Not Null [✓], Unique [ ], Binary [ ], Unsigned [ ], Zero Fill [ ], Auto Increment [ ], Generated [ ]

C:\Users\user\OneDrive\Documents\0.Freelancing\41.DB Developments\1.MySQL\Test DB Architect Virta.mwb saved.

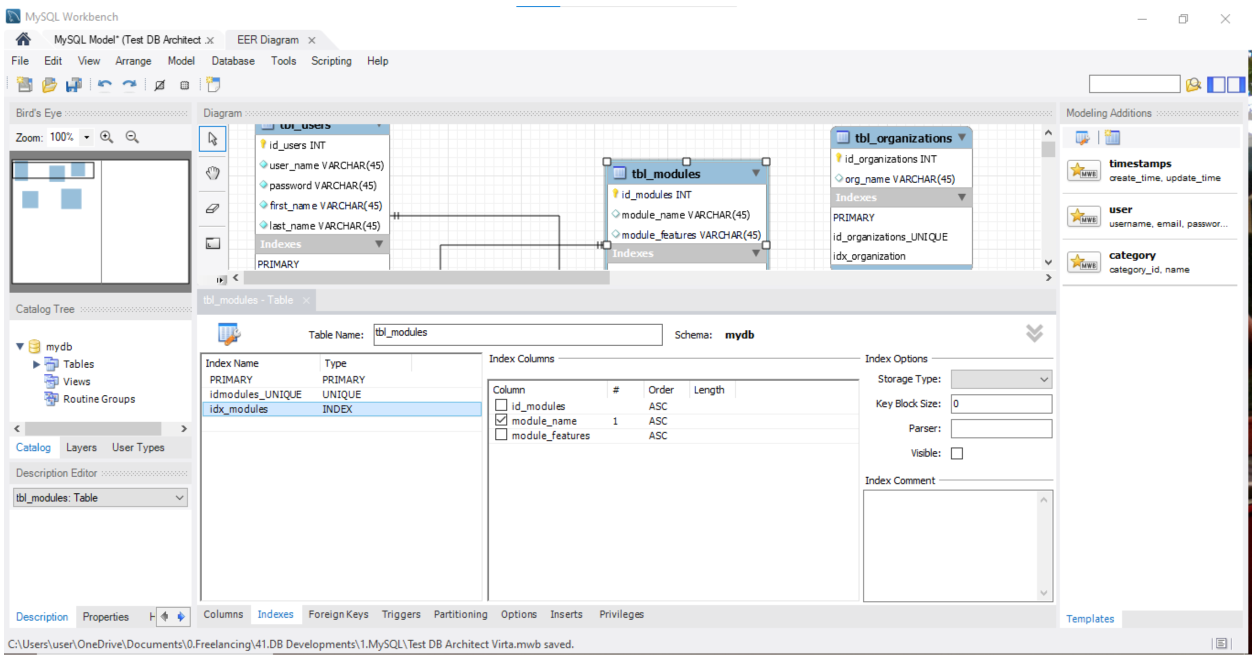


Table Name: tbl_modules Schema: mydb

Index Name	Type
PRIMARY	PRIMARY
idmodules_UNIQUE	UNIQUE
idx_modules	INDEX

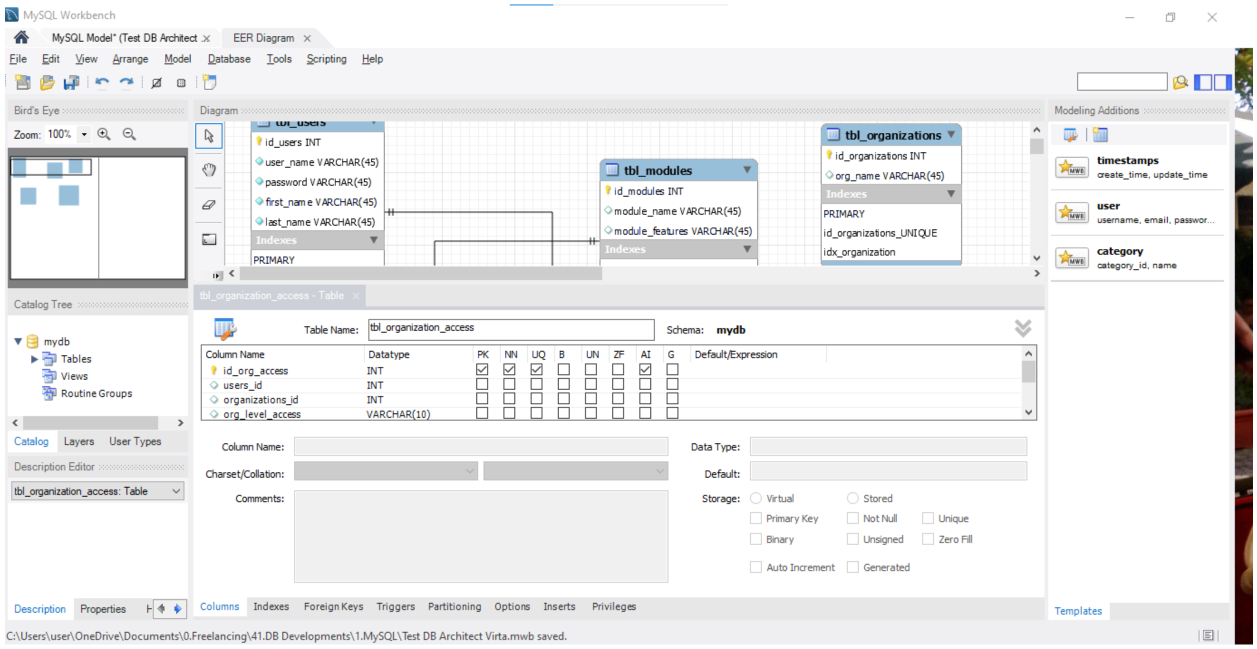
Index Columns

Column	#	Order	Length
[ ] id_modules		ASC	
[✓] module_name	1	ASC	
[ ] module_features		ASC	

Index Options: Storage Type: ; Key Block Size: 0; Parser: ; Visible: [ ]

C:\Users\user\OneDrive\Documents\0.Freelancing\41.DB Developments\1.MySQL\Test DB Architect Virta.mwb saved.

tbl_organization_access:

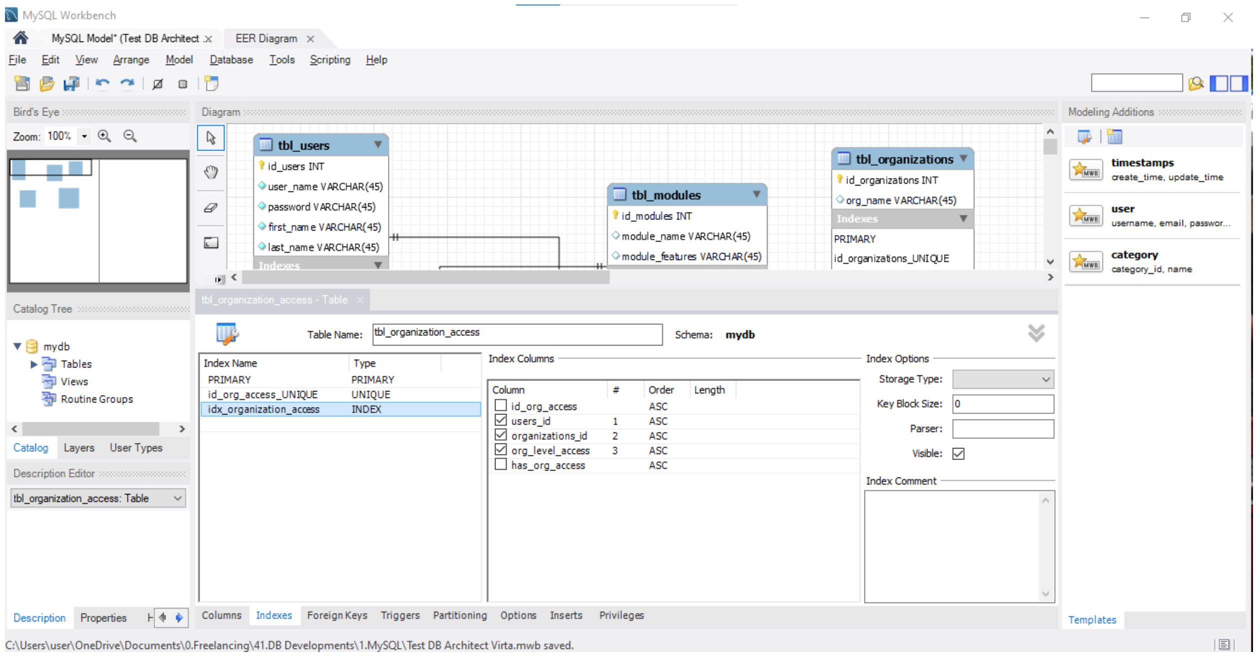


tbl_organization_access - Table

Table Name: tbl_organization_access Schema: **mydb**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id_org_access	INT	✓	✓	✓				✓		
users_id	INT									
organizations_id	INT									
org_level_access	VARCHAR(10)									

C:\Users\user\OneDrive\Documents\0.Freelancing\41.DB Developments\1.MySQL\Test DB Architect Virta.mwb saved.



tbl_organization_access - Table

Table Name: tbl_organization_access Schema: **mydb**

Index Name	Type
PRIMARY	PRIMARY
id_org_access_UNIQUE	UNIQUE
idx_organization_access	INDEX

Index Columns

	Column	#	Order	Length
	id_org_access		ASC	
✓	users_id	1	ASC	
✓	organizations_id	2	ASC	
✓	org_level_access	3	ASC	
	has_org_access		ASC	

Index Options

Storage Type:
Key Block Size: 0
Parser:
Visible: ✓

C:\Users\user\OneDrive\Documents\0.Freelancing\41.DB Developments\1.MySQL\Test DB Architect Virta.mwb saved.

tbl_module_access:

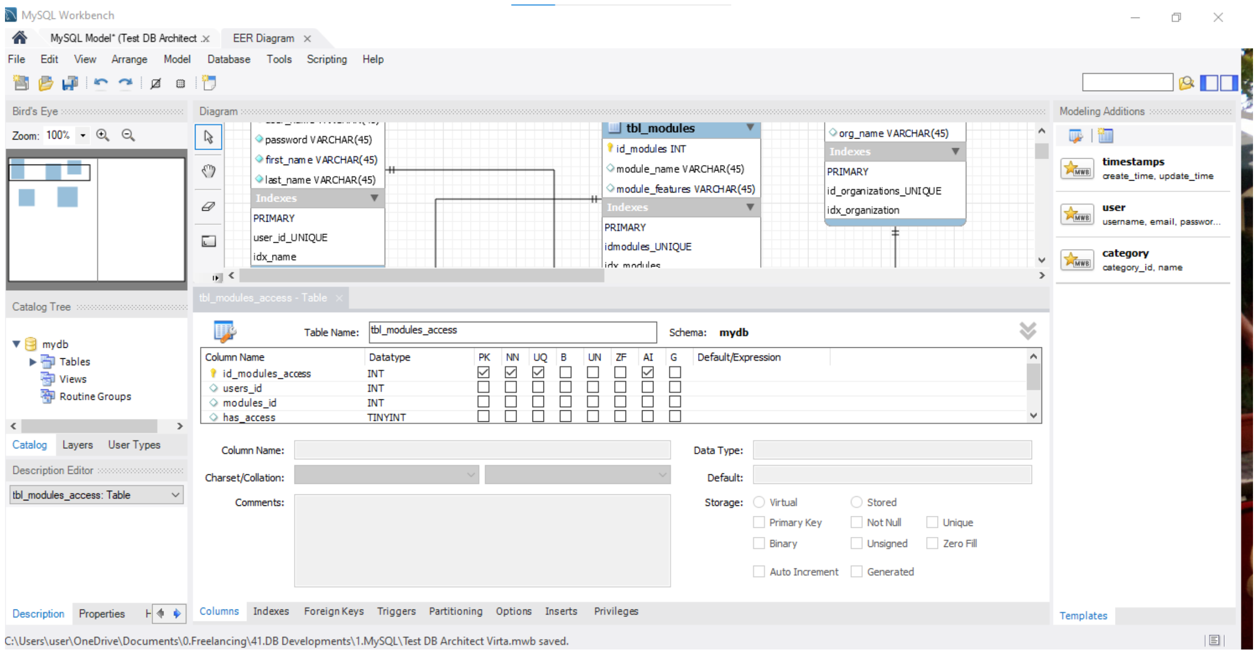


Table Name: tbl_modules_access   Schema: mydb

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id_modules_access	INT	✓	✓	✓				✓		
users_id	INT									
modules_id	INT									
has_access	TINYINT									

C:\Users\user\OneDrive\Documents\0.Freelancing\41.DB Developments\1.MySQL\Test DB Architect Virta.mwb saved.

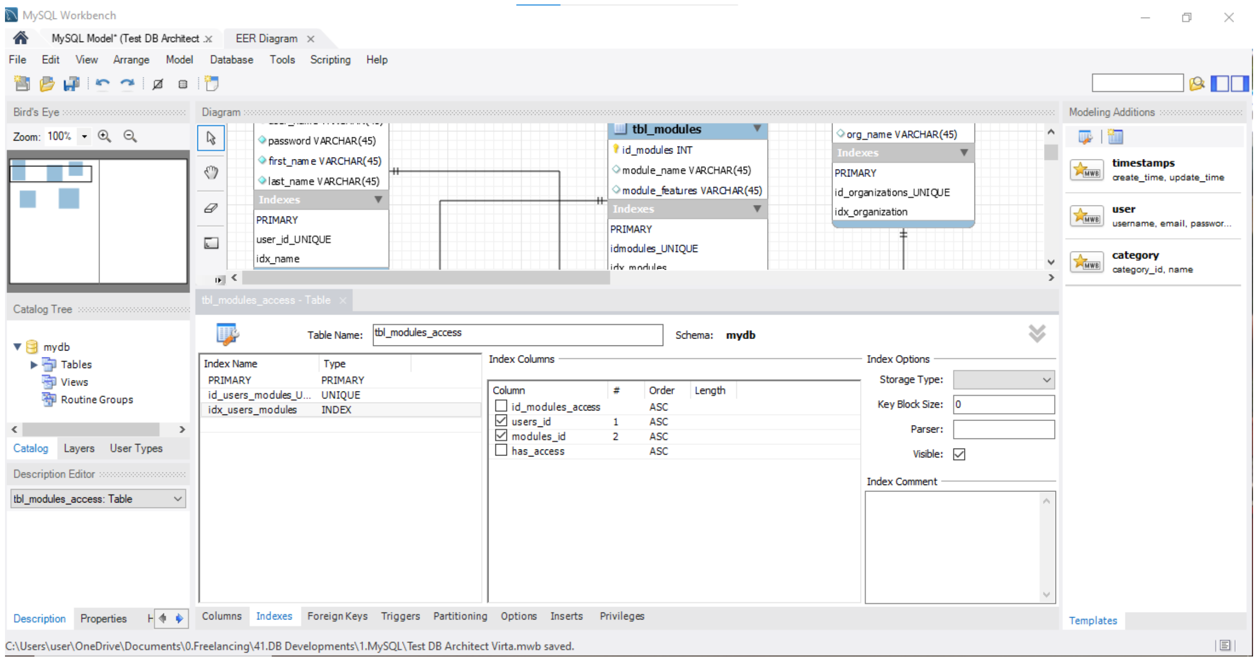


Table Name: tbl_modules_access   Schema: mydb

Index Name	Type
PRIMARY	PRIMARY
id_users_modules_U...	UNIQUE
idx_users_modules	INDEX

Index Columns

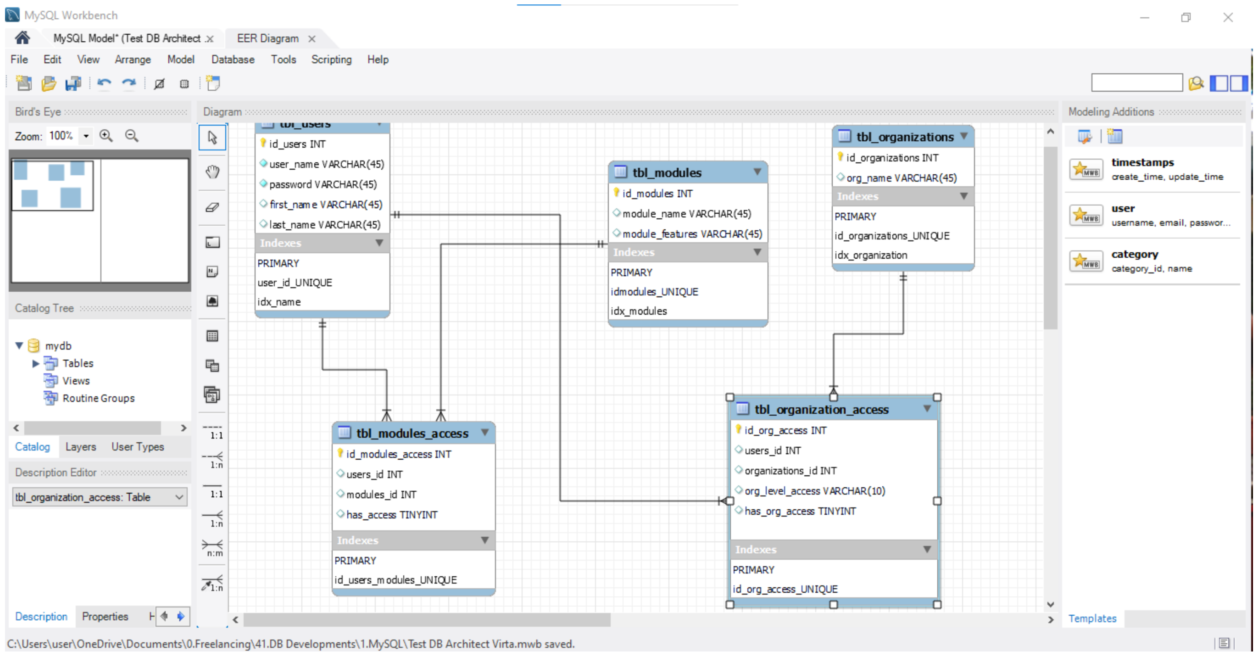
	Column	#	Order	Length
☐	id_modules_access		ASC	
✓	users_id	1	ASC	
✓	modules_id	2	ASC	
☐	has_access		ASC	

Index Options

Storage Type:
Key Block Size: 0
Parser:
Visible: ✓

C:\Users\user\OneDrive\Documents\0.Freelancing\41.DB Developments\1.MySQL\Test DB Architect Virta.mwb saved.

Finally, this is the database schema we propose as a solution for this task:



The tables in this database structure contain added indexes, as it is recommended to add indexes when along with the table...The structure is fully relational, meaning we provisioned foreign keys in related tables, wherever was necessary.

Task 2: we are now experiencing performance issues with this table, different DB operations are slow and they may cause big load to the DB. How would you start investigating what causes this slowness, and what would be your first recommendations based on the info provided above on how to improve the database performance?

Solution: To investigate performance issues with the Station table, we could start by analyzing the queries that are being run against the table and identifying any that are particularly slow or resource intensive. We could also use a tool like Explain Analyze to get more detailed information about the performance of individual queries.

This is how I think we should start investigating regarding the performance issues with the Station table:

1) Analyze query plans; Identify slow performing queries and those that use a full table scan (if any); Use Explain Analyze to get more detailed information and identify the root causes about the poor performance of these individual queries

2) Check if any indexes are used in the Station table; If not, indexes must be designed on critical columns (maybe non-clustered indexes) to speed up the slow queries that use these columns. In general, indexes should be created along with the table, to avoid the risk of locking the table and

experience downtimes because of creating indexes while tables are already in Production and are concurrently accessed and have a large number of records.

3) Investigate how the dynamic data influences the speed of the queries…There may be a large number of records containing the dynamic data showing only that a station is in a On/Off status (the content of the other fields remains the same) and the queries would have to go through these large number of records only to identify which Station is On or Off…These queries should be investigated first and we should analyze the impact of dynamically created data in relation to the slowness of queries.

Improvements I would recommend:

1) Use of proper investigation tools:

--Use database profiling and monitoring tools like MySQL's performance_schema or PostgreSQL's pg_stat_activity to analyze query performance.

--Use an index advisor tool like MySQL's pt-index-usage or PostgreSQL's pg_index_advisor to recommend index creation.

-- Use a load testing tool like Apache JMeter or Locust to simulate heavy traffic and test database performance under load.

2)Analyze query execution plans: Analyze query execution plans to identify any inefficiencies or bottlenecks.

3)Adding indexes to the Station table as suggested by Explain Analyze tool, to speed up slow queries that are run frequently. Consider creating additional indexes to improve query performance. Create indexes on columns frequently used in queries, like the online/offline status and seller ID.

4)Using caching to reduce the number of queries that need to be run against the Station table.

5)Partitioning the Station table horizontally to distribute the data across multiple servers and improve performance (sharding)

6)Using a load balancer to distribute queries across multiple servers and reduce the load on any single server.

7)Query optimization: Check if queries can be optimized to reduce the load on the database. Consider using WHERE clauses to filter the data before running a query.

8)Database maintenance: Check if the database is being properly maintained, such as regular backups and database optimization.

9)Hardware: Check the hardware resources allocated to the DB server and ensure that they are sufficient to handle the current workload and expected growth. Sharding may be necessary, and this is done by adding extra servers.

Task 3: how would you predict in advance whether or not some operation like adding an index will cause downtime for the system? Or would there be a way to avoid downtime in situations like this?

Solution:

1. Adding an index to a table implies using an index creation statement
2. The downtime induced by adding an index to a table is directly linked with the type of storage engine used by MySQL and this is related to the version of MySQL we use. MySQL offers three storage engines used to create database objects and store the associated data for these objects, namely MyISAM, InnoDB and NDB Cluster. The choice of a storage engine depends on the specific needs and requirements of an application or database. InnoDB is a popular and widely used storage engine for its reliability, transaction support, and performance benefits for read-heavy workloads. However, there may be cases where other storage engines are more suitable for specific scenarios.
   For example, MyISAM can be faster for read-heavy workloads that do not require transaction support or foreign key constraints, but it lacks the durability and data consistency guarantees provided by InnoDB.
   Other storage engines like NDB Cluster are designed for distribution, high-availability environments with high concurrency and low latency requirements.
   If we use older versions of MySQL (like 4.x or 5.0 to 5.5) we need to enforce the use of InnoDB storage engine (the best storage engine and the most popular one offered by MySQL) because for these older versions the default storage engine used is MyISAM. If we keep using MyISAM for these older versions, the result will be having a lock on the table and this will cause massive downtime for tables with large number of records. For MySQL versions starting with 5.6, the default storage engine is InnoDB that allows concurrency control and having locks at the row level for read / write operations, which allows the table to remain available during heavy transaction operations, so having downtimes becomes highly unlikely.
   So, the first element used to predict downtimes when adding a index is the version of MySQL used in Production: if this is less than 5.6 we need to enforce using a InnoDB storage engine instead of MyISAM, by using the configuration option "default_storage_engine = InnoDB" in MySQL configuration file.
   For versions starting with 5.6, InnoDB is the default option and the locks are put at the row level…This is maximum we can get because ultimately, having locks during creation of an index, altering a table etc, is a common mechanism of updating records in one of these database objects, these are transactions at the tables level…What is important is to have the locks put at the row level instead of locking the entire table, when having these kind of transactions occurring within a table…

3. The main way to avoid the downtimes is to enforce the usage of InnoDB storage engine instead of MyISAM as we elaborated at 2. Assuming we've already addressed this aspect,

the next steps would be to analyze and optimize the concurrency control aspect. Let's analyze as an example the CREATE INDEX statement for three versions of MySQL.

3.1. A statement used in an earlier version of MySQL (MySQL 5.5 Reference Manual):

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [index_option] ...

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH | RTREE}

index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
```

3.2. A statement to create an index used in a later version of MySQL (MySQL 5.7 Reference Manual):

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (key_part,...)
    [index_option]
    [algorithm_option | lock_option] ...

key_part:
    col_name [(length)] [ASC | DESC]

index_option: {
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
}

index_type:
    USING {BTREE | HASH}

algorithm_option:
    ALGORITHM [=] {DEFAULT | INPLACE | COPY}

lock_option:
    LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

3.3.A statement to create an index used in the actual version of MySQL (MySQL 8.0 Reference Manual):

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (key_part,...)
    [index_option]
    [algorithm_option | lock_option] ...

key_part: {col_name [(length)] | (expr)} [ASC | DESC]

index_option: {
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | {VISIBLE | INVISIBLE}
  | ENGINE_ATTRIBUTE [=] 'string'
  | SECONDARY_ENGINE_ATTRIBUTE [=] 'string'
}

index_type:
    USING {BTREE | HASH}

algorithm_option:
    ALGORITHM [=] {DEFAULT | INPLACE | COPY}

lock_option:
        LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

In the past, for MySQL Server 5.5. and earlier, creating an index on an existing table would lock the entire table for read / write operations for the entire duration of the index creation. Let us notice that in this statement there is no *lock_option* placeholder to specify LOCK options regarding the table on which the index is built. The only option to control the downtime was the enforce the usage of InnoDB storage engine over the MyISAM storage engine, as we showed at 2.

On the other hand, the *lock_option* placeholder is present starting with MySQL 5.6 as it groups the options for a LOCK parameter, like: `LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}`

This means that starting with MySQL 5.6, we have the option to enforce the type of LOCK on the table where we build the index…

So, starting with MySQL 5.6. we can control more thoroughly the concurrency at row level aspect to be able to control the locking aspect due to index creation, we should be using at least a MySQL 5.6 version and we should use one of the LOCK options mentioned above…

LOCK parameter is the means through which we enforce concurrency control, that could be the second way, that involves a control strategy with the objective to minimize or reduce to 0 the downtime due to adding indexes to tables.

More on what we can achieve with this parameter is detailed in MySQL 8.0 Reference Manual, here: https://dev.mysql.com/doc/refman/8.0/en/alter-table.html .