





Curriculum

Short Specializations 
Average: 80.71% 


0x00. MySQL advanced

Back-end

SQL

MySQL

 Weight: 1

 Project will start Jul 10, 2024 4:00 AM, must end by Jul 12, 2024 4:00 AM

✓ Checker was released at Jul 10, 2024 4:00 PM

☒ An auto review will be launched at the deadline

Concepts

For this project, we expect you to look at this concept:

- [Advanced SQL \(/concepts/555\)](/concepts/555)

Resources

Read or watch:

- [MySQL cheatsheet \(/rltoken/8w9di_hk19DIMSBEV3EayQ\)](/rltoken/8w9di_hk19DIMSBEV3EayQ)
- [MySQL Performance: How To Leverage MySQL Database Indexing \(/rltoken/2GJbZ48zRPA70o2YhTdH7g\)](/rltoken/2GJbZ48zRPA70o2YhTdH7g)
- [Stored Procedure \(/rltoken/K180X2OCzb6gzPngjn-Elg\)](/rltoken/K180X2OCzb6gzPngjn-Elg)
- [Triggers \(/rltoken/cJ1qA4o-rRm4rWlsqYKSZg\)](/rltoken/cJ1qA4o-rRm4rWlsqYKSZg)
- [Views \(/rltoken/vHg1z3UAOcWMvOt8xZHeiA\)](/rltoken/vHg1z3UAOcWMvOt8xZHeiA)
- [Functions and Operators \(/rltoken/g-c1m6iljScpi4LeqxBRqQ\)](/rltoken/g-c1m6iljScpi4LeqxBRqQ)
- [Trigger Syntax and Examples \(/rltoken/gLVwKjQfRL0Jr_nWqAS7VQ\)](/rltoken/gLVwKjQfRL0Jr_nWqAS7VQ)
- [CREATE TABLE Statement \(/rltoken/X789nJ22H6HVh1uCQPI0Ig\)](/rltoken/X789nJ22H6HVh1uCQPI0Ig)
- [CREATE PROCEDURE and CREATE FUNCTION Statements \(/rltoken/mfrWMt1KL3NHXblJykMgZg\)](/rltoken/mfrWMt1KL3NHXblJykMgZg)
- [CREATE INDEX Statement \(/rltoken/oCu8Rg9WfKyF4BhTt8dZGQ\)](/rltoken/oCu8Rg9WfKyF4BhTt8dZGQ)
- [CREATE VIEW Statement \(/rltoken/FEZNIZFKZmD1ISnLiNkCwQ\)](/rltoken/FEZNIZFKZmD1ISnLiNkCwQ)



Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/NEA0Fr7muHfukl5lziVAhg), **without the help of Google**:

General

- How to create tables with constraints
- How to optimize queries by adding indexes
- What is and how to implement stored procedures and functions in MySQL
- What is and how to implement views in MySQL
- What is and how to implement triggers in MySQL

Requirements

General

- All your files will be executed on Ubuntu 18.04 LTS using MySQL 5.7 (version 5.7.30)
- All your files should end with a new line
- All your SQL queries should have a comment just before (i.e. syntax above)
- All your files should start by a comment describing the task
- All SQL keywords should be in uppercase (SELECT , WHERE ...)
- A README.md file, at the root of the folder of the project, is mandatory
- The length of your files will be tested using wc

More Info

Comments for your SQL file:

```
$ cat my_script.sql
-- 3 first students in the Batch ID=3
-- because Batch 3 is the best!
SELECT id, name FROM students WHERE batch_id = 3 ORDER BY created_at DESC LIMIT 3;
$
```

Use "container-on-demand" to run MySQL

- Ask for container Ubuntu 18.04 - Python 3.7
- Connect via SSH
- Or via the WebTerminal
- In the container, you should start MySQL before playing with it:



```
$ service mysql start
* MySQL Community Server 5.7.30 is started

$
$ cat 0-list_databases.sql | mysql -uroot -p my_database
Enter password:
Database
information_schema
mysql
performance_schema
sys
$
```

In the container, credentials are root/root

How to import a SQL dump

```
$ echo "CREATE DATABASE hbtn_0d_tvshows;" | mysql -uroot -p
Enter password:
$ curl "https://s3.amazonaws.com/intranet-projects-files/holbertonschool-higher-level_programming+/274/hbtn_0d_tvshows.sql" -s | mysql -uroot -p hbtn_0d_tvshows
Enter password:
$ echo "SELECT * FROM tv_genres" | mysql -uroot -p hbtn_0d_tvshows
Enter password:
id  name
1   Drama
2   Mystery
3   Adventure
4   Fantasy
5   Comedy
6   Crime
7   Suspense
8   Thriller
$
```

Tasks

0. We are all unique!

mandatory

Write a SQL script that creates a table `users` following these requirements:

- With these attributes:
 - `id` , integer, never null, auto increment and primary key
 - `email` , string (255 characters), never null and unique
 - `name` , string (255 characters)
- If the table already exists, your script should not fail



- Your script can be executed on any database

(//)

Context: Make an attribute unique directly in the table schema will enforced your business rules and avoid bugs in your application


```
bob@dylan:~$ echo "SELECT * FROM users;" | mysql -uroot -p holberton
Enter password:
ERROR 1146 (42S02) at line 1: Table 'holberton.users' doesn't exist
bob@dylan:~$
bob@dylan:~$ cat 0-uniq_users.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ echo 'INSERT INTO users (email, name) VALUES ("bob@dylan.com", "Bob");' | mysql -uroot -p holberton
Enter password:
bob@dylan:~$ echo 'INSERT INTO users (email, name) VALUES ("sylvie@dylan.com", "Sylvie");' | mysql -uroot -p holberton
Enter password:
bob@dylan:~$ echo 'INSERT INTO users (email, name) VALUES ("bob@dylan.com", "Jean");' | mysql -uroot -p holberton
Enter password:
ERROR 1062 (23000) at line 1: Duplicate entry 'bob@dylan.com' for key 'email'
bob@dylan:~$
bob@dylan:~$ echo "SELECT * FROM users;" | mysql -uroot -p holberton
Enter password:
id  email      name
1   bob@dylan.com  Bob
2   sylvie@dylan.com  Sylvie
bob@dylan:~$
```

Repo:

- GitHub repository: alx-backend-storage
- Directory: 0x00-MySQL_Advanced
- File: 0-uniq_users.sql

☐ Done?

Check your code

 Get a sandbox

1. In and not out

mandatory

Write a SQL script that creates a table `users` following these requirements:

- With these attributes:
 - `id` , integer, never null, auto increment and primary key
 - `email` , string (255 characters), never null and unique
 - `name` , string (255 characters)
 - `country` , enumeration of countries: `US` , `CO` and `TN` , never null (= default will be the first element of the enumeration, here `US`)



- If the table already exists, your script should not fail
- (/). Your script can be executed on any database

```
bob@dylan:~$ echo "SELECT * FROM users;" | mysql -uroot -p holberton
Enter password:
ERROR 1146 (42S02) at line 1: Table 'holberton.users' doesn't exist
bob@dylan:~$
bob@dylan:~$ cat 1-country_users.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ echo 'INSERT INTO users (email, name, country) VALUES ("bob@dylan.com",
"Bob", "US");' | mysql -uroot -p holberton
Enter password:
bob@dylan:~$ echo 'INSERT INTO users (email, name, country) VALUES ("sylvie@dylan.co
m", "Sylvie", "CO");' | mysql -uroot -p holberton
Enter password:
bob@dylan:~$ echo 'INSERT INTO users (email, name, country) VALUES ("jean@dylan.co
m", "Jean", "FR");' | mysql -uroot -p holberton
Enter password:
ERROR 1265 (01000) at line 1: Data truncated for column 'country' at row 1
bob@dylan:~$
bob@dylan:~$ echo 'INSERT INTO users (email, name) VALUES ("john@dylan.com", "Joh
n");' | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ echo "SELECT * FROM users;" | mysql -uroot -p holberton
Enter password:
id  email      name      country
1   bob@dylan.com  Bob      US
2   sylvie@dylan.com  Sylvie   CO
3   john@dylan.com  John     US
bob@dylan:~$
```

Repo:

- GitHub repository: alx-backend-storage
- Directory: 0x00-MySQL_Advanced
- File: 1-country_users.sql

☐ Done?

2. Best band ever!

mandatory

Write a SQL script that ranks country origins of bands, ordered by the number of (non-unique) fans

Requirements:

- Import this table dump: metal_bands.sql.zip (/rltoken/uPn947gnZLaa0FJrrAFTGQ)
- Column names must be: origin and nb_fans

- Your script can be executed on any database

(//)

Context: *Calculate/compute something is always power intensive... better to distribute the load!*


```
bob@dylan:~$ cat metal_bands.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ cat 2-fans.sql | mysql -uroot -p holberton > tmp_res ; head tmp_res
Enter password:
origin  nb_fans
USA 99349
Sweden 47169
Finland 32878
United Kingdom 32518
Germany 29486
Norway 22405
Canada 8874
The Netherlands 8819
Italy 7178
bob@dylan:~$
```

Repo:

- GitHub repository: `alx-backend-storage`
- Directory: `0x00-MySQL_Advanced`
- File: `2-fans.sql`

☐ Done?

Check your code

 Get a sandbox

3. Old school band

mandatory

Write a SQL script that lists all bands with `glam rock` as their main style, ranked by their longevity

Requirements:

- Import this table dump: `metal_bands.sql.zip` (`/rltoken/uPn947gnZLaa0FJrrAFTGQ`)
- Column names must be: `band_name` and `lifespan` (in years **until 2022** - please use `2022` instead of `YEAR(CURDATE())`)
- You should use attributes `formed` and `split` for computing the `lifespan`
- Your script can be executed on any database




```
bob@dylan:~$ cat metal_bands.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ cat 3-glam_rock.sql | mysql -uroot -p holberton
Enter password:
band_name    lifespan
Alice Cooper    56
Mötley Crüe    34
Marilyn Manson  31
The 69 Eyes    30
Hardcore Superstar  23
Nasty Idols    0
Hanoi Rocks    0
bob@dylan:~$
```

Repo:

- GitHub repository: alx-backend-storage
- Directory: 0x00-MySQL_Advanced
- File: 3-glam_rock.sql

☐ Done?

Check your code

 Get a sandbox**4. Buy buy buy****mandatory**

Write a SQL script that creates a trigger that decreases the quantity of an item after adding a new order.

Quantity in the table `items` can be negative.

Context: *Updating multiple tables for one action from your application can generate issue: network disconnection, crash, etc... to keep your data in a good shape, let MySQL do it for you!*



```
bob@dylan:~$ cat 4-init.sql
-- Initial

DROP TABLE IF EXISTS items;
DROP TABLE IF EXISTS orders;

CREATE TABLE IF NOT EXISTS items (
    name VARCHAR(255) NOT NULL,
    quantity int NOT NULL DEFAULT 10
);

CREATE TABLE IF NOT EXISTS orders (
    item_name VARCHAR(255) NOT NULL,
    number int NOT NULL
);

INSERT INTO items (name) VALUES ("apple"), ("pineapple"), ("pear");

bob@dylan:~$
bob@dylan:~$ cat 4-init.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ cat 4-store.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ cat 4-main.sql
Enter password:
-- Show and add orders
SELECT * FROM items;
SELECT * FROM orders;

INSERT INTO orders (item_name, number) VALUES ('apple', 1);
INSERT INTO orders (item_name, number) VALUES ('apple', 3);
INSERT INTO orders (item_name, number) VALUES ('pear', 2);

SELECT "--";

SELECT * FROM items;
SELECT * FROM orders;

bob@dylan:~$
bob@dylan:~$ cat 4-main.sql | mysql -uroot -p holberton
Enter password:
name    quantity
apple   10
pineapple 10
pear    10
--
--
name    quantity
apple   6
pineapple 10
pear    8
```





```
item_name  number
apple     1
apple     3
pear      2
bob@dylan:~$
```

Repo:

- GitHub repository: alx-backend-storage
- Directory: 0x00-MySQL_Advanced
- File: 4-store.sql

☐ Done?

Check your code

 Get a sandbox**5. Email validation to sent****mandatory**

Write a SQL script that creates a trigger that resets the attribute `valid_email` only when the `email` has been changed.

Context: *Nothing related to MySQL, but perfect for user email validation - distribute the logic to the database itself!*



```
bob@dylan:~$ cat 5-init.sql
-- Initial
DROP TABLE IF EXISTS users;

CREATE TABLE IF NOT EXISTS users (
    id int not null AUTO_INCREMENT,
    email varchar(255) not null,
    name varchar(255),
    valid_email boolean not null default 0,
    PRIMARY KEY (id)
);

INSERT INTO users (email, name) VALUES ("bob@dylan.com", "Bob");
INSERT INTO users (email, name, valid_email) VALUES ("sylvie@dylan.com", "Sylvie",
1);
INSERT INTO users (email, name, valid_email) VALUES ("jeanne@dylan.com", "Jeanne",
1);

bob@dylan:~$
bob@dylan:~$ cat 5-init.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ cat 5-valid_email.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ cat 5-main.sql
Enter password:
-- Show users and update (or not) email
SELECT * FROM users;

UPDATE users SET valid_email = 1 WHERE email = "bob@dylan.com";
UPDATE users SET email = "sylvie+new@dylan.com" WHERE email = "sylvie@dylan.com";
UPDATE users SET name = "Jannis" WHERE email = "jeanne@dylan.com";

SELECT "--";
SELECT * FROM users;

UPDATE users SET email = "bob@dylan.com" WHERE email = "bob@dylan.com";

SELECT "--";
SELECT * FROM users;

bob@dylan:~$
bob@dylan:~$ cat 5-main.sql | mysql -uroot -p holberton
Enter password:
id  email      name      valid_email
1   bob@dylan.com  Bob      0
2   sylvie@dylan.com  Sylvie   1
3   jeanne@dylan.com  Jeanne   1
--
--
id  email      name      valid_email
```



```
1 bob@dylan.com Bob 1
2) sylvie+new@dylan.com Sylvie 0
3 jeanne@dylan.com Jannis 1

--
--
id email name valid_email
1 bob@dylan.com Bob 1
2 sylvie+new@dylan.com Sylvie 0
3 jeanne@dylan.com Jannis 1
bob@dylan:~$
```

Repo:

- GitHub repository: alx-backend-storage
- Directory: 0x00-MySQL_Advanced
- File: 5-valid_email.sql

☐ Done?☐ Check your code☐ > Get a sandbox**6. Add bonus****mandatory**

Write a SQL script that creates a stored procedure `AddBonus` that adds a new correction for a student.

Requirements:

- Procedure `AddBonus` is taking 3 inputs (in this order):
 - `user_id`, a `users.id` value (you can assume `user_id` is linked to an existing `users`)
 - `project_name`, a new or already exists `projects` - if no `projects.name` found in the table, you should create it
 - `score`, the score value for the correction

Context: Write code in SQL is a nice level up!



```
bob@dylan:~$ cat 6-init.sql
```

```
-- Initial
```

```
DROP TABLE IF EXISTS corrections;
```

```
DROP TABLE IF EXISTS users;
```

```
DROP TABLE IF EXISTS projects;
```

```
CREATE TABLE IF NOT EXISTS users (  
    id int not null AUTO_INCREMENT,  
    name varchar(255) not null,  
    average_score float default 0,  
    PRIMARY KEY (id)  
);
```

```
CREATE TABLE IF NOT EXISTS projects (  
    id int not null AUTO_INCREMENT,  
    name varchar(255) not null,  
    PRIMARY KEY (id)  
);
```

```
CREATE TABLE IF NOT EXISTS corrections (  
    user_id int not null,  
    project_id int not null,  
    score int default 0,  
    KEY `user_id` (`user_id`),  
    KEY `project_id` (`project_id`),  
    CONSTRAINT fk_user_id FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE,  
    CONSTRAINT fk_project_id FOREIGN KEY (`project_id`) REFERENCES `projects` (`id`) ON DELETE CASCADE  
);
```

```
INSERT INTO users (name) VALUES ("Bob");  
SET @user_bob = LAST_INSERT_ID();
```

```
INSERT INTO users (name) VALUES ("Jeanne");  
SET @user_jeanne = LAST_INSERT_ID();
```

```
INSERT INTO projects (name) VALUES ("C is fun");  
SET @project_c = LAST_INSERT_ID();
```

```
INSERT INTO projects (name) VALUES ("Python is cool");  
SET @project_py = LAST_INSERT_ID();
```

```
INSERT INTO corrections (user_id, project_id, score) VALUES (@user_bob, @project_c,  
80);
```

```
INSERT INTO corrections (user_id, project_id, score) VALUES (@user_bob, @project_py,  
96);
```

```
INSERT INTO corrections (user_id, project_id, score) VALUES (@user_jeanne, @project_c,  
91);
```

```
INSERT INTO corrections (user_id, project_id, score) VALUES (@user_jeanne, @project_
```

```
py, 73);
(/)
bob@dylan:~$
bob@dylan:~$ cat 6-init.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ cat 6-bonus.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ cat 6-main.sql
Enter password:
-- Show and add bonus correction
SELECT * FROM projects;
SELECT * FROM corrections;

SELECT "--";

CALL AddBonus((SELECT id FROM users WHERE name = "Jeanne"), "Python is cool", 100);

CALL AddBonus((SELECT id FROM users WHERE name = "Jeanne"), "Bonus project", 100);
CALL AddBonus((SELECT id FROM users WHERE name = "Bob"), "Bonus project", 10);

CALL AddBonus((SELECT id FROM users WHERE name = "Jeanne"), "New bonus", 90);

SELECT "--";

SELECT * FROM projects;
SELECT * FROM corrections;

bob@dylan:~$
bob@dylan:~$ cat 6-main.sql | mysql -uroot -p holberton
Enter password:
id  name
1   C is fun
2   Python is cool
user_id project_id  score
1    1      80
1    2      96
2    1      91
2    2      73
--
--
--
--
id  name
1   C is fun
2   Python is cool
3   Bonus project
4   New bonus
user_id project_id  score
1    1      80
1    2      96
```




```
2 1 91
(2) 2 73
2 2 100
2 3 100
1 3 10
2 4 90
bob@dylan:~$
```

Repo:

- GitHub repository: alx-backend-storage
- Directory: 0x00-MySQL_Advanced
- File: 6-bonus.sql

☐ Done?

Check your code

 Get a sandbox**7. Average score**

mandatory

Write a SQL script that creates a stored procedure `ComputeAverageScoreForUser` that computes and store the average score for a student. Note: An average score can be a decimal

Requirements:

- Procedure `ComputeAverageScoreForUser` is taking 1 input:
 - `user_id`, a `users.id` value (you can assume `user_id` is linked to an existing `users`)



```
bob@dylan:~$ cat 7-init.sql
```

```
-- Initial
```

```
DROP TABLE IF EXISTS corrections;
```

```
DROP TABLE IF EXISTS users;
```

```
DROP TABLE IF EXISTS projects;
```

```
CREATE TABLE IF NOT EXISTS users (  
    id int not null AUTO_INCREMENT,  
    name varchar(255) not null,  
    average_score float default 0,  
    PRIMARY KEY (id)  
);
```

```
CREATE TABLE IF NOT EXISTS projects (  
    id int not null AUTO_INCREMENT,  
    name varchar(255) not null,  
    PRIMARY KEY (id)  
);
```

```
CREATE TABLE IF NOT EXISTS corrections (  
    user_id int not null,  
    project_id int not null,  
    score int default 0,  
    KEY `user_id` (`user_id`),  
    KEY `project_id` (`project_id`),  
    CONSTRAINT fk_user_id FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE,  
    CONSTRAINT fk_project_id FOREIGN KEY (`project_id`) REFERENCES `projects` (`id`) ON DELETE CASCADE  
);
```

```
INSERT INTO users (name) VALUES ("Bob");  
SET @user_bob = LAST_INSERT_ID();
```

```
INSERT INTO users (name) VALUES ("Jeanne");  
SET @user_jeanne = LAST_INSERT_ID();
```

```
INSERT INTO projects (name) VALUES ("C is fun");  
SET @project_c = LAST_INSERT_ID();
```

```
INSERT INTO projects (name) VALUES ("Python is cool");  
SET @project_py = LAST_INSERT_ID();
```

```
INSERT INTO corrections (user_id, project_id, score) VALUES (@user_bob, @project_c,  
80);
```

```
INSERT INTO corrections (user_id, project_id, score) VALUES (@user_bob, @project_py,  
96);
```

```
INSERT INTO corrections (user_id, project_id, score) VALUES (@user_jeanne, @project_  
c, 91);
```

```
INSERT INTO corrections (user_id, project_id, score) VALUES (@user_jeanne, @project_
```

```
py, 73);
(/)
bob@dylan:~$
bob@dylan:~$ cat 7-init.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ cat 7-average_score.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ cat 7-main.sql
-- Show and compute average score
SELECT * FROM users;
SELECT * FROM corrections;

SELECT "--";
CALL ComputeAverageScoreForUser((SELECT id FROM users WHERE name = "Jeanne"));

SELECT "--";
SELECT * FROM users;

bob@dylan:~$
bob@dylan:~$ cat 7-main.sql | mysql -uroot -p holberton
Enter password:
id  name  average_score
1   Bob   0
2   Jeanne 0
user_id project_id score
1   1     80
1   2     96
2   1     91
2   2     73
--
--
--
--
id  name  average_score
1   Bob   0
2   Jeanne 82
bob@dylan:~$
```

Repo:

- GitHub repository: alx-backend-storage
- Directory: 0x00-MySQL_Advanced
- File: 7-average_score.sql

☐ Done?

Check your code

Get a sandbox

8. Optimize simple search

mandatory

Write a SQL script that creates an index `idx_name_first` on the table `names` and the first letter of `name` .

Requirements:

- Import this table dump: `names.sql.zip` (/rltoken/BluyCCllfw0NqcjqUiUdEw)
- Only the first letter of `name` must be indexed

Context: *Index is not the solution for any performance issue, but well used, it's really powerful!*



```

bob@dylan:~$ cat names.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ mysql -uroot -p holberton
Enter password:
mysql> SELECT COUNT(name) FROM names WHERE name LIKE 'a%';
+-----+
| COUNT(name) |
+-----+
|      302936 |
+-----+
1 row in set (2.19 sec)
mysql>
mysql> exit
bye
bob@dylan:~$
bob@dylan:~$ cat 8-index_my_names.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ mysql -uroot -p holberton
Enter password:
mysql> SHOW index FROM names;
+-----+-----+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name      | Seq_in_index | Column_name | Collation | Car-
|       | Sub_part  | Packed | Null | Index_type | Comment | Index_comment |
|       |          |        |      |            |          |          |
+-----+-----+-----+-----+-----+-----+-----+
| names |          1 | idx_name_first |          1 | name        | A        |
25 |          1 | NULL    | YES  | BTREE      |          |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
mysql>
mysql> SELECT COUNT(name) FROM names WHERE name LIKE 'a%';
+-----+
| COUNT(name) |
+-----+
|      302936 |
+-----+
1 row in set (0.82 sec)
mysql>
mysql> exit
bye
bob@dylan:~$

```




Repo:

- GitHub repository: alx-backend-storage
- Directory: 0x00-MySQL_Advanced

- File: 8-index_my_names.sql (/)

☐ Done?

Check your code

 Get a sandbox

9. Optimize search and score

mandatory

Write a SQL script that creates an index `idx_name_first_score` on the table `names` and the first letter of `name` and the `score` .

Requirements:

- Import this table dump: names.sql.zip (/rltoken/BluyCCIlfw0NqcjqUiUdEw)
- Only the first letter of `name` AND `score` must be indexed



```

bob@dylan:~$ cat names.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ mysql -uroot -p holberton
Enter password:
mysql> SELECT COUNT(name) FROM names WHERE name LIKE 'a%' AND score < 80;
+-----+
| count(name) |
+-----+
|          60717 |
+-----+
1 row in set (2.40 sec)
mysql>
mysql> exit
bye
bob@dylan:~$
bob@dylan:~$ cat 9-index_name_score.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ mysql -uroot -p holberton
Enter password:
mysql> SHOW index FROM names;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name          | Seq_in_index | Column_name | Collation |
| Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| names |          1 | idx_name_first_score |          1 | name        | A        |
|          25 |          1 | NULL    | YES  | BTREE      |          |
| names |          1 | idx_name_first_score |          2 | score       | A        |
|          3901 | NULL | NULL    | YES  | BTREE      |          |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
mysql>
mysql> SELECT COUNT(name) FROM names WHERE name LIKE 'a%' AND score < 80;
+-----+
| COUNT(name) |
+-----+
|          60717 |
+-----+
1 row in set (0.48 sec)
mysql>
mysql> exit
bye
bob@dylan:~$

```




Repo:

- GitHub repository: `alx-backend-storage`
- (/). Directory: `0x00-MySQL_Advanced`
- File: `9-index_name_score.sql`

☐ Done?

Check your code

 Get a sandbox

10. Safe divide

mandatory

Write a SQL script that creates a function `SafeDiv` that divides (and returns) the first by the second number or returns 0 if the second number is equal to 0.

Requirements:

- You must create a function
- The function `SafeDiv` takes 2 arguments:
 - `a, INT`
 - `b, INT`
- And returns `a / b` or 0 if `b == 0`



```
bob@dylan:~$ cat 10-init.sql
-- Initial
DROP TABLE IF EXISTS numbers;

CREATE TABLE IF NOT EXISTS numbers (
    a int default 0,
    b int default 0
);

INSERT INTO numbers (a, b) VALUES (10, 2);
INSERT INTO numbers (a, b) VALUES (4, 5);
INSERT INTO numbers (a, b) VALUES (2, 3);
INSERT INTO numbers (a, b) VALUES (6, 3);
INSERT INTO numbers (a, b) VALUES (7, 0);
INSERT INTO numbers (a, b) VALUES (6, 8);

bob@dylan:~$ cat 10-init.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ cat 10-div.sql | mysql -uroot -p holberton
Enter password:
bob@dylan:~$
bob@dylan:~$ echo "SELECT (a / b) FROM numbers;" | mysql -uroot -p holberton
Enter password:
(a / b)
5.0000
0.8000
0.6667
2.0000
NULL
0.7500
bob@dylan:~$
bob@dylan:~$ echo "SELECT SafeDiv(a, b) FROM numbers;" | mysql -uroot -p holberton
Enter password:
SafeDiv(a, b)
5
0.8000000011920929
0.66666666865348816
2
0
0.75
bob@dylan:~$
```

Repo:

- GitHub repository: alx-backend-storage
- Directory: 0x00-MySQL_Advanced
- File: 10-div.sql



[\(A\) Done?](#)[Check your code](#)[>_ Get a sandbox](#)

11. No table for a meeting

mandatory

Write a SQL script that creates a view `need_meeting` that lists all students that have a score under 80 (strict) and no `last_meeting` or more than 1 month.

Requirements:

- The view `need_meeting` should return all students name when:
 - They score are under (strict) to 80
 - **AND** no `last_meeting` date **OR** more than a month



```
bob@dylan:~$ cat 11-init.sql
```

```
-- Initial
```

```
DROP TABLE IF EXISTS students;
```

```
CREATE TABLE IF NOT EXISTS students (  
    name VARCHAR(255) NOT NULL,  
    score INT default 0,  
    last_meeting DATE NULL  
);
```

```
INSERT INTO students (name, score) VALUES ("Bob", 80);  
INSERT INTO students (name, score) VALUES ("Sylvia", 120);  
INSERT INTO students (name, score) VALUES ("Jean", 60);  
INSERT INTO students (name, score) VALUES ("Steeve", 50);  
INSERT INTO students (name, score) VALUES ("Camilia", 80);  
INSERT INTO students (name, score) VALUES ("Alexa", 130);
```

```
bob@dylan:~$ cat 11-init.sql | mysql -uroot -p holberton
```

```
Enter password:
```

```
bob@dylan:~$
```

```
bob@dylan:~$ cat 11-need_meeting.sql | mysql -uroot -p holberton
```

```
Enter password:
```

```
bob@dylan:~$
```

```
bob@dylan:~$ cat 11-main.sql
```

```
-- Test view
```

```
SELECT * FROM need_meeting;
```

```
SELECT "--";
```

```
UPDATE students SET score = 40 WHERE name = 'Bob';
```

```
SELECT * FROM need_meeting;
```

```
SELECT "--";
```

```
UPDATE students SET score = 80 WHERE name = 'Steeve';
```

```
SELECT * FROM need_meeting;
```

```
SELECT "--";
```

```
UPDATE students SET last_meeting = CURDATE() WHERE name = 'Jean';
```

```
SELECT * FROM need_meeting;
```

```
SELECT "--";
```

```
UPDATE students SET last_meeting = ADDDATE(CURDATE(), INTERVAL -2 MONTH) WHERE name  
= 'Jean';
```

```
SELECT * FROM need_meeting;
```

```
SELECT "--";
```

```
SHOW CREATE TABLE need_meeting;
```




```
SELECT "--";  
(/)  
SHOW CREATE TABLE students;
```

```
bob@dylan:~$  
bob@dylan:~$ cat 11-main.sql | mysql -uroot -p holberton  
Enter password:  
name  
Jean  
Steeve  
--  
--  
name  
Bob  
Jean  
Steeve  
--  
--  
name  
Bob  
Jean  
--  
--  
name  
Bob  
--  
--  
name  
Bob  
Jean  
--  
--  
View      Create View character_set_client      collation_connection  
XXXXXX<yes, here it will display the View SQL statement :->XXXXXX  
--  
--  
Table      Create Table  
students   CREATE TABLE `students` (\n  `name` varchar(255) NOT NULL,\n  `score` in  
t(11) DEFAULT '0',\n  `last_meeting` date DEFAULT NULL\n) ENGINE=InnoDB DEFAULT CHAR  
SET=latin1  
bob@dylan:~$
```

Repo:

- GitHub repository: alx-backend-storage
- Directory: 0x00-MySQL_Advanced
- File: 11-need_meeting.sql

☐ Done?

(/)

Done with the mandatory tasks? Unlock 2 advanced tasks now!

Copyright © 2024 ALX, All rights reserved.

