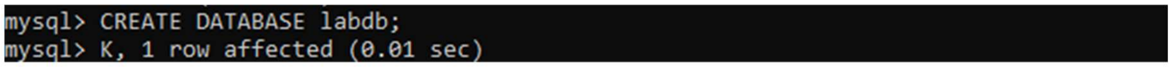
NUID: 001099849

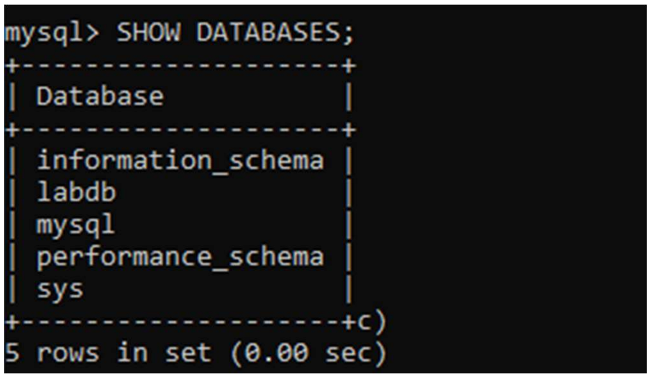
LAB 7: DATABASE SECURITY

PART 1: Basics of MySql

I set up mysql inside the docker container. I then created a database: labdb as seen in figure 1.



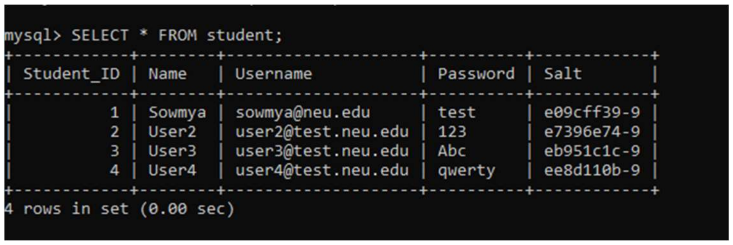
```
mysql> CREATE DATABASE labdb;
mysql> K, 1 row affected (0.01 sec)
```



```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| labdb              |
| mysql              |
| performance_schema |
| sys                |
+--------------------+c)
5 rows in set (0.00 sec)
```

Figure 1: Creating labdb

In this database we create a table student with 5 rows of values inserted. The resulting table looks as seen in figure 2.

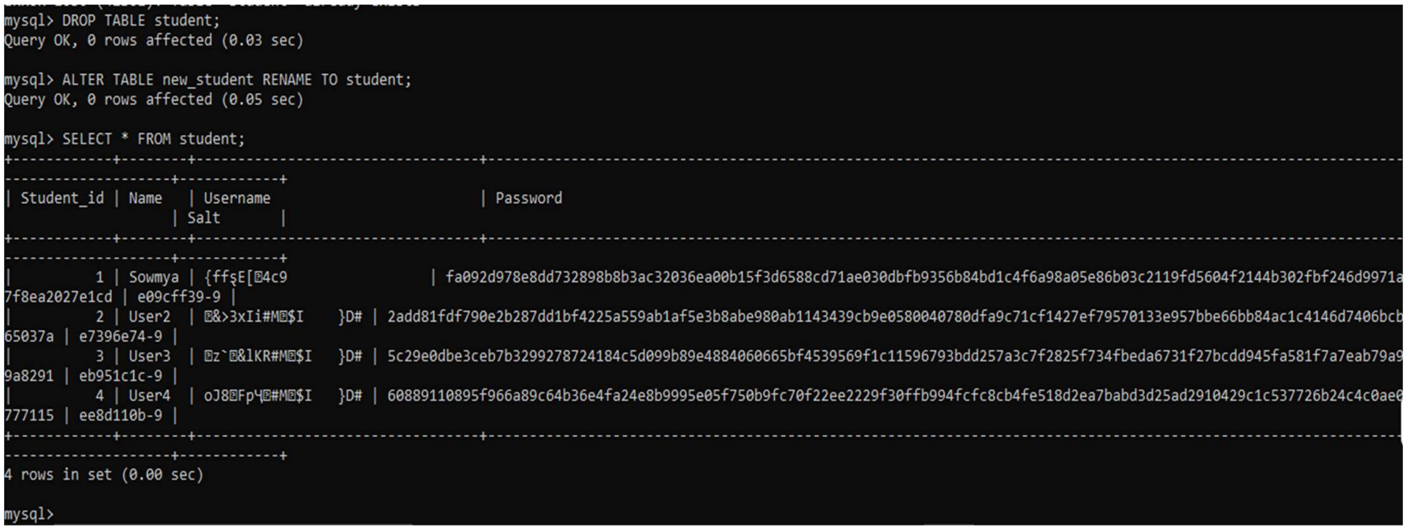


```
mysql> SELECT * FROM student;
+------------+--------+--------------------+----------+------------+
| Student_ID | Name   | Username           | Password | Salt       |
+------------+--------+--------------------+----------+------------+
|          1 | Sowmya | sowmya@neu.edu     | test     | e09cff39-9 |
|          2 | User2  | user2@test.neu.edu | 123      | e7396e74-9 |
|          3 | User3  | user3@test.neu.edu | Abc      | eb951c1c-9 |
|          4 | User4  | user4@test.neu.edu | qwerty   | ee8d110b-9 |
+------------+--------+--------------------+----------+------------+
4 rows in set (0.00 sec)
```

Figure 2: Initial student table

PART 2: Attribute Encryption

Here we protect the stored username and password of the users. Username is encrypted using AES algorithm (128 bit by default) with ‘P@assw0rd!’ as key string and password is first salted and then generated a SHA-2 checksum of length 512 bit. The resultant student table is as shown in figure 3.



```
mysql> DROP TABLE student;
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE new_student RENAME TO student;
Query OK, 0 rows affected (0.05 sec)

mysql> SELECT * FROM student;
+------------+--------+----------------------------------+------------------------------------------------------------------------------------------------------
--------------------+------------+
| Student_id | Name   | Username                         | Password
                    | Salt       |
+------------+--------+----------------------------------+------------------------------------------------------------------------------------------------------
--------------------+------------+
|          1 | Sowmya | {ffşE[?4c9                 | fa092d978e8dd732898b8b3ac32036ea00b15f3d6588cd71ae030dbfb9356b84bd1c4f6a98a05e86b03c2119fd5604f2144b302fbf246d9971a
7f8ea2027e1cd | e09cff39-9 |
|          2 | User2  | ?&>3xIi#M?$I    }D# | 2add81fdf790e2b287dd1bf4225a559ab1af5e3b8abe980ab1143439cb9e0580040780dfa9c71cf1427ef79570133e957bbe66bb84ac1c4146d7406bcb
65037a | e7396e74-9 |
|          3 | User3  | ?z`?&lKR#M?$I   }D# | 5c29e0dbe3ceb7b3299278724184c5d099b89e4884060665bf4539569f1c11596793bdd257a3c7f2825f734fbeda6731f27bcdd945fa581f7a7eab79a9
9a8291 | eb951c1c-9 |
|          4 | User4  | oJ8?FpЧ?#M?$I   }D# | 60889110895f966a89c64b36e4fa24e8b9995e05f750b9fc70f22ee2229f30ffb994fcfc8cb4fe518d2ea7babd3d25ad2910429c1c537726b24c4c0ae0
777115 | ee8d110b-9 |
+------------+--------+----------------------------------+------------------------------------------------------------------------------------------------------
--------------------+------------+
4 rows in set (0.00 sec)

mysql>
```

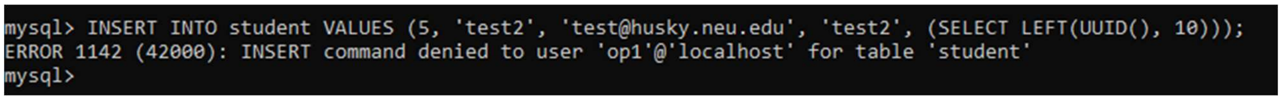
Figure 3: Encrypted Student table

PART 3: Database Access Control

We create total of 3 roles: admin, operator, developer to access student table of labdb database. Admin role has all the permission granted for the table and the user ad1 (password: admin) is created and then assigned to this role.

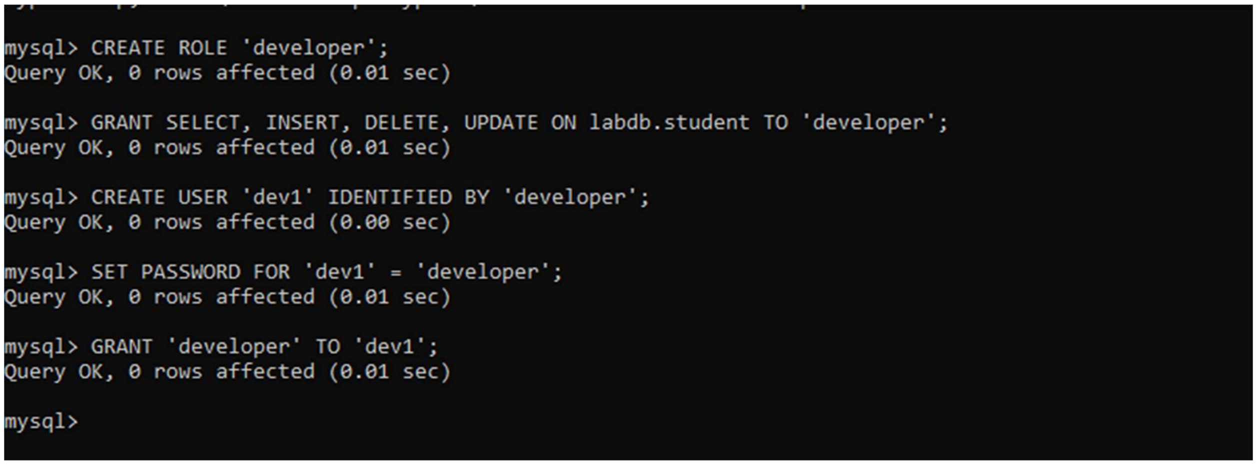
Operator role has only create view and show view permissions granted and the user op1 (password: operator) is then created and assigned to this role. We can verify this by logging into the database as op1 user and try to insert any values into the student table. This action will be denied as seen in figure 4.

Developer role has select, insert, delete, update permissions granted and the user dev1 (password: developer) is then created and assigned to this role as seen in figure 5.



```
mysql> INSERT INTO student VALUES (5, 'test2', 'test@husky.neu.edu', 'test2', (SELECT LEFT(UUID(), 10)));
ERROR 1142 (42000): INSERT command denied to user 'op1'@'localhost' for table 'student'
mysql>
```

Figure 4



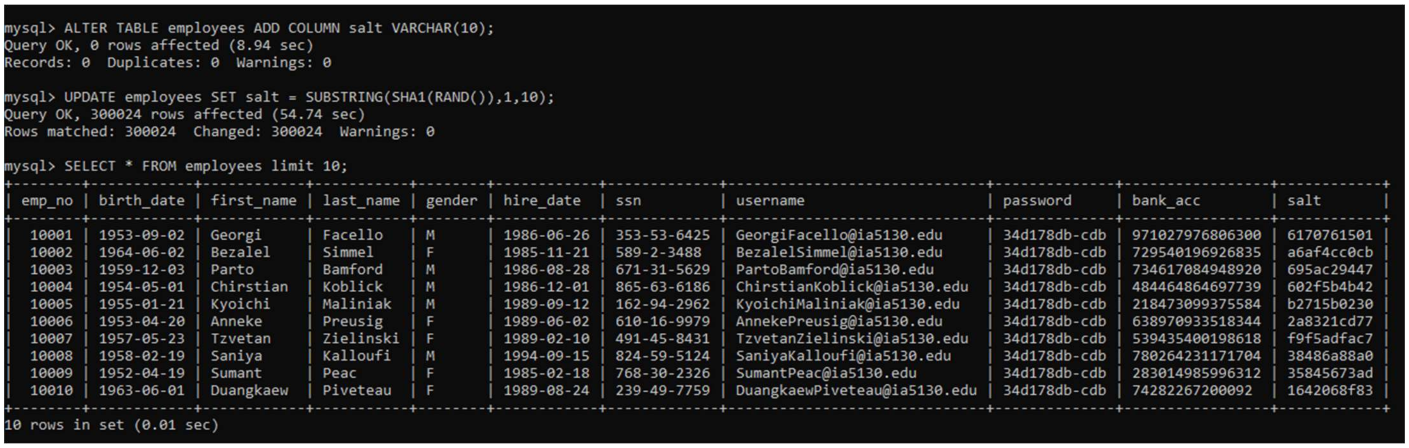
```
mysql> CREATE ROLE 'developer';
Query OK, 0 rows affected (0.01 sec)

mysql> GRANT SELECT, INSERT, DELETE, UPDATE ON labdb.student TO 'developer';
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE USER 'dev1' IDENTIFIED BY 'developer';
Query OK, 0 rows affected (0.00 sec)

mysql> SET PASSWORD FOR 'dev1' = 'developer';
Query OK, 0 rows affected (0.01 sec)

mysql> GRANT 'developer' TO 'dev1';
Query OK, 0 rows affected (0.01 sec)

mysql>
```

Figure 5: Creating developer role

PART 4: Database Hardening

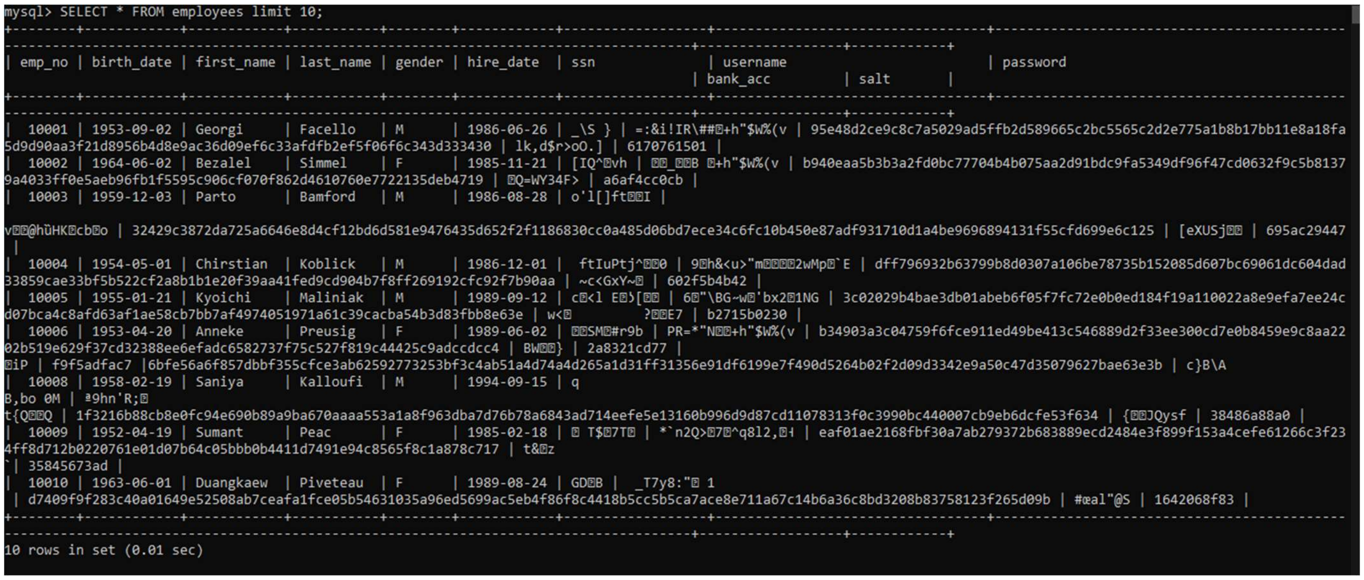
We export the database using the sql file: employees.sql into the empty employeesdb. We then harden the employees table using the same approach used for the student table. We encrypt ssn, username, bank_acc using aes algorithm using the key string: P@ssw0rd! and salted the passwords and generated a SHA-2 (512 bit) checksum. The resulting employees table is as seen in figure 7.

Command used: >UPDATE employees SET ssn = aes_encrypt(ssn, 'P@ssw0rd!'), username = aes_encrypt(username,'P@ssw0rd!'), password = sha2(concat(password,salt), 512), acc_no = aes_encrypt(bank_acc, 'P@ssw0rd!') ;



```
mysql> ALTER TABLE employees ADD COLUMN salt VARCHAR(10);
Query OK, 0 rows affected (8.94 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> UPDATE employees SET salt = SUBSTRING(SHA1(RAND()),1,10);
Query OK, 300024 rows affected (54.74 sec)
Rows matched: 300024  Changed: 300024  Warnings: 0

mysql> SELECT * FROM employees limit 10;
+--------+------------+------------+-----------+--------+------------+-------------+-----------------------------------+--------------+-----------------+------------+
| emp_no | birth_date | first_name | last_name | gender | hire_date  | ssn         | username                          | password     | bank_acc        | salt       |
+--------+------------+------------+-----------+--------+------------+-------------+-----------------------------------+--------------+-----------------+------------+
|  10001 | 1953-09-02 | Georgi     | Facello   | M      | 1986-06-26 | 353-53-6425 | GeorgiFacello@ia5130.edu          | 34d178db-cdb | 971027976806300 | 6170761501 |
|  10002 | 1964-06-02 | Bezalel    | Simmel    | F      | 1985-11-21 | 589-2-3488  | BezalelSimmel@ia5130.edu          | 34d178db-cdb | 729540196926835 | a6af4cc0cb |
|  10003 | 1959-12-03 | Parto      | Bamford   | M      | 1986-08-28 | 671-31-5629 | PartoBamford@ia5130.edu           | 34d178db-cdb | 734617084948920 | 695ac29447 |
|  10004 | 1954-05-01 | Chirstian  | Koblick   | M      | 1986-12-01 | 865-63-6186 | ChirstianKoblick@ia5130.edu       | 34d178db-cdb | 484464864697739 | 602f5b4b42 |
|  10005 | 1955-01-21 | Kyoichi    | Maliniak  | M      | 1989-09-12 | 162-94-2962 | KyoichiMaliniak@ia5130.edu        | 34d178db-cdb | 218473099375584 | b2715b0230 |
|  10006 | 1953-04-20 | Anneke     | Preusig   | F      | 1989-06-02 | 610-16-9979 | AnnekePreusig@ia5130.edu          | 34d178db-cdb | 638970933518344 | 2a8321cd77 |
|  10007 | 1957-05-23 | Tzvetan    | Zielinski | F      | 1989-02-10 | 491-45-8431 | TzvetanZielinski@ia5130.edu       | 34d178db-cdb | 539435400198618 | f9f5adfac7 |
|  10008 | 1958-02-19 | Saniya     | Kalloufi  | M      | 1994-09-15 | 824-59-5124 | SaniyaKalloufi@ia5130.edu         | 34d178db-cdb | 780264231171704 | 38486a88a0 |
|  10009 | 1952-04-19 | Sumant     | Peac      | F      | 1985-02-18 | 768-30-2326 | SumantPeac@ia5130.edu             | 34d178db-cdb | 283014985996312 | 35845673ad |
|  10010 | 1963-06-01 | Duangkaew  | Piveteau  | F      | 1989-08-24 | 239-49-7759 | DuangkaewPiveteau@ia5130.edu      | 34d178db-cdb | 74282267200092  | 1642068f83 |
+--------+------------+------------+-----------+--------+------------+-------------+-----------------------------------+--------------+-----------------+------------+
10 rows in set (0.01 sec)
```

Figure 6: Adding salt column.

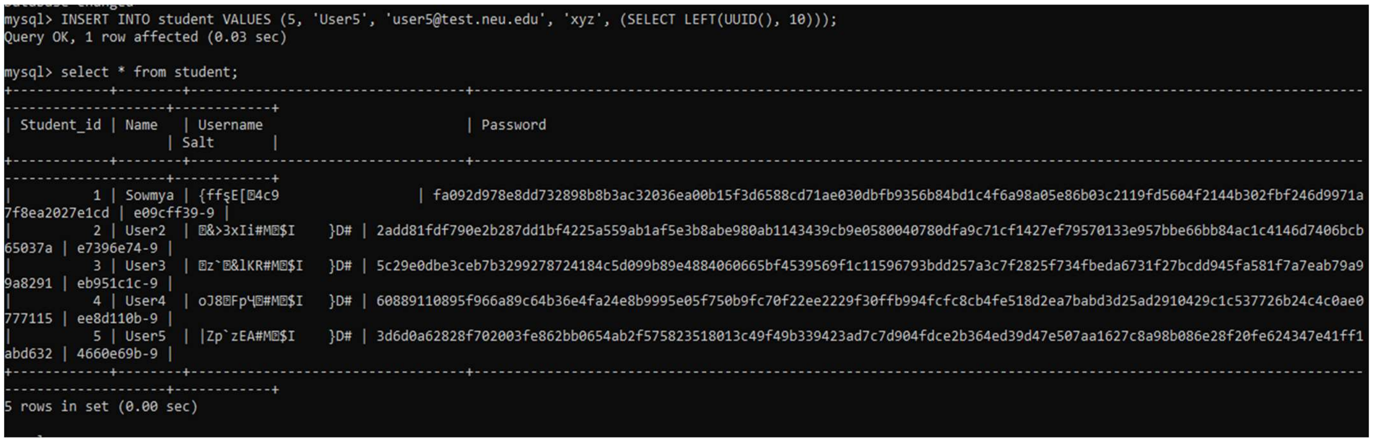


```
mysql> SELECT * FROM employees limit 10;
+--------+------------+------------+-----------+--------+------------+------------------+----------------------------------+--------------------------------------------
-----------------------------------------------------------------------------------------+-----------------+------------+
| emp_no | birth_date | first_name | last_name | gender | hire_date  | ssn              | username                         | password
                                                                                         | bank_acc        | salt       |
+--------+------------+------------+-----------+--------+------------+------------------+----------------------------------+--------------------------------------------
-----------------------------------------------------------------------------------------+-----------------+------------+
|  10001 | 1953-09-02 | Georgi     | Facello   | M      | 1986-06-26 | _\S } | =:&i!IR\##□+h"$W%(v | 95e48d2ce9c8c7a5029ad5ffb2d589665c2bc5565c2d2e775a1b8b17bb11e8a18fa
5d9d90aa3f21d8956b4d8e9ac36d09ef6c33afdfb2ef5f06f6c343d333430 | lk,d$r>oO.] | 6170761501 |
|  10002 | 1964-06-02 | Bezalel    | Simmel    | F      | 1985-11-21 | [IQ^□vh | □□_□□B □+h"$W%(v | b940eaa5b3b3a2fd0bc77704b4b075aa2d91bdc9fa5349df96f47cd0632f9c5b8137
9a4033ff0e5aeb96fb1f5595c906cf070f862d4610760e7722135deb4719 | □Q=WY34F> | a6af4cc0cb |
|  10003 | 1959-12-03 | Parto      | Bamford   | M      | 1986-08-28 | o'l[]ft□□I |

v□□@hũHK□cb□o | 32429c3872da725a6646e8d4cf12bd6d581e9476435d652f2f1186830cc0a485d06bd7ece34c6fc10b450e87adf931710d1a4be9696894131f55cfd699e6c125 | [eXUSj□□ | 695ac29447
 |
|  10004 | 1954-05-01 | Chirstian  | Koblick   | M      | 1986-12-01 |  ftIuPtj^□□0 | 9□h&<u>"m□□□□2wMp□`E | dff796932b63799b8d0307a106be78735b152085d607bc69061dc604dad
33859cae33bf5b522cf2a8b1b1e20f39aa41fed9cd904b7f8ff269192cfc92f7b90aa | ~c<GxY~□ | 602f5b4b42 |
|  10005 | 1955-01-21 | Kyoichi    | Maliniak  | M      | 1989-09-12 | c□<l E□͗[□□ | 6□"\BG~w□'bx2□1NG | 3c02029b4bae3db01abeb6f05f7fc72e0b0ed184f19a110022a8e9efa7ee24c
d07bca4c8afd63af1ae58cb7bb7af4974051971a61c39cacba54b3d83fbb8e63e | w<□         ?□□E7 | b2715b0230 |
|  10006 | 1953-04-20 | Anneke     | Preusig   | F      | 1989-06-02 | □□SM□#r9b | PR=*"N□□+h"$W%(v | b34903a3c04759f6fce911ed49be413c546889d2f33ee300cd7e0b8459e9c8aa22
02b519e629f37cd32388ee6efadc6582737f75c527f819c44425c9adccdcc4 | BW□□} | 2a8321cd77 |
□iP | f9f5adfac7 |6bfe56a6f857dbbf355cfce3ab62592773253bf3c4ab51a4d74a4d265a1d31ff31356e91df6199e7f490d5264b02f2d09d3342e9a50c47d35079627bae63e3b | c}B\A
|  10008 | 1958-02-19 | Saniya     | Kalloufi  | M      | 1994-09-15 | q
B,bo 0M | ª9hn'R;□
t{Q□□Q | 1f3216b88cb8e0fc94e690b89a9ba670aaaa553a1a8f963dba7d76b78a6843ad714eefe5e13160b996d9d87cd11078313f0c3990bc440007cb9eb6dcfe53f634 | {□□JQysf | 38486a88a0 |
|  10009 | 1952-04-19 | Sumant     | Peac      | F      | 1985-02-18 | □ T$□7T□ | *`n2Q>□7□^q8l2,□ɨ | eaf01ae2168fbf30a7ab279372b683889ecd2484e3f899f153a4cefe61266c3f23
4ff8d712b0220761e01d07b64c05bbb0b4411d7491e94c8565f8c1a878c717 | t&□z
`| 35845673ad |
|  10010 | 1963-06-01 | Duangkaew  | Piveteau  | F      | 1989-08-24 | GD□B |  _T7y8:"□ 1
 | d7409f9f283c40a01649e52508ab7ceafa1fce05b54631035a96ed5699ac5eb4f86f8c4418b5cc5b5ca7ace8e711a67c14b6a36c8bd3208b83758123f265d09b | #œal"@S | 1642068f83 |
+--------+------------+------------+-----------+--------+------------+------------------+----------------------------------+--------------------------------------------
-----------------------------------------------------------------------------------------+-----------------+------------+
10 rows in set (0.01 sec)
```

Figure 7: Updated employees table

PART 6: Automatic encryption

To automate the above encryption for student table before insertion, we create a trigger enrypt_data. We can verify this works by inserting another row into the table and check if it is encrypted as seen in figure 8.



```
mysql> INSERT INTO student VALUES (5, 'User5', 'user5@test.neu.edu', 'xyz', (SELECT LEFT(UUID(), 10)));
Query OK, 1 row affected (0.03 sec)

mysql> select * from student;
+------------+--------+----------------------------------+------------------------------------------------------------------------------------------------------------
-------------------+-----------+
| Student_id | Name   | Username                         | Password
                   | Salt      |
+------------+--------+----------------------------------+------------------------------------------------------------------------------------------------------------
-------------------+-----------+
|          1 | Sowmya | {ffşE[□4c9                   | fa092d978e8dd732898b8b3ac32036ea00b15f3d6588cd71ae030dbfb9356b84bd1c4f6a98a05e86b03c2119fd5604f2144b302fbf246d9971a
7f8ea2027e1cd | e09cff39-9 |
|          2 | User2  | □&>3xIi#M□$I    }D# | 2add81fdf790e2b287dd1bf4225a559ab1af5e3b8abe980ab1143439cb9e0580040780dfa9c71cf1427ef79570133e957bbe66bb84ac1c4146d7406bcb
65037a | e7396e74-9 |
|          3 | User3  | □z`□&lKR#M□$I   }D# | 5c29e0dbe3ceb7b3299278724184c5d099b89e4884060665bf4539569f1c11596793bdd257a3c7f2825f734fbeda6731f27bcdd945fa581f7a7eab79a9
9a8291 | eb951c1c-9 |
|          4 | User4  | oJ8□FpЧ□#M□$I   }D# | 60889110895f966a89c64b36e4fa24e8b9995e05f750b9fc70f22ee2229f30ffb994fcfc8cb4fe518d2ea7babd3d25ad2910429c1c537726b24c4c0ae0
777115 | ee8d110b-9 |
|          5 | User5  | |Zp`zEA#M□$I    }D# | 3d6d0a62828f702003fe862bb0654ab2f575823518013c49f49b339423ad7c7d904fdce2b364ed39d47e507aa1627c8a98b086e28f20fe624347e41ff1
abd632 | 4660e69b-9 |
+------------+--------+----------------------------------+------------------------------------------------------------------------------------------------------------
-------------------+-----------+
5 rows in set (0.00 sec)
```

Figure 8: Automated Encryption

Docker container: docker pull sowmyashreebv/db_security:team8

QUESTIONS:

1. What is Role Based Access Control? What are the benefits of RBAC? List other authorization models.

Role Based Access Control is an authorization model where the permission and privileges are granted to a user based on the role that user is assigned to. By using this model, it is possible to provide access at granular level and hence can enforce principle of least-privilege making the system more secure. It is possible to ensure that the access to sensitive info can be tightly controlled hence reducing the risk of breaches. Also, it reduces workload to administer and change permission to the user whenever necessary (ex: change in designation/new hire) by granting/removing roles.

Other authorization models include: Discretionary Access control where the access depends upon the identity of the user (Linux systems), Mandatory Access Control (SELinux) where the access is controlled based on the assigned security labels, Attribute Based Access Control where access depends on the attributes and of the user environmental conditions.

2. What are some attacks on passwords? How to defend against such attacks?

1. Brute Force Attack: Brute force attack takes a trial-and-error approach to guess the password where a computer program is used to try all possible password combination up to a specified length. If performed as an online attack we can defend by enforcing a limit on login retries. But this is usually performed as an offline attack, hence, to defend against it, secure storing of passwords needs to be available. Since this attack is resource intensive, using of strong/long passwords can make the attack difficult.

2. Dictionary Attack: Dictionary attack allows hackers to employ a program which cycles through a list of common words. An easy way to prevent such attacks is ensure stronger passwords and change of passwords regularly and to employ exponential back-off policy.

3. Rainbow Tables: It uses a database of possible passwords and their pre-computed hashes. With the assumption that more than one password can generate the same hash, it is not necessary for the attacker to know what the original password was. Salt, which are random data passed into the hash function along with the plain text, can easily avoid rainbow table attacks.

3. Write a short description on Bcrypt, Scrypt and Argon2 and PBKDF2 algorithms.

Bcrypt: is a hashing algorithm based on blowfish cipher. It uses large salt by default and performs key stretching wherever necessary. Hence it is protected against rainbow table attack. It is memory hardening i.e it employs a cost factor which increases the amount of work/resources necessary to compute the hash exponentially.

Scrypt: is a key derivation function. It is computationally expensive, slow and memory intensive i.e the algorithm requires large amounts of resources to execute the hash. Like Bcrypt it also uses large salt and cost factor. Hence it is immune to brute force attacks and rainbow table attack. It is more commonly used for cryptocurrency mining.

Argon2: Argon2 is a secure password hashing algorithm. It is designed to have both a configurable runtime as well as memory consumption. Therefore, we can determine how long it takes to hash a password and how much memory is required. It consists of 3 parameters: time cost (/ iteration rounds, similar to cost factor of bcrypt), a memory cost, which defines the memory usage and a parallelism degree, which defines the number of parallel threads. It is immune to GPU cracking attacks and its version Argon2i is also resistant to side-channel attacks.

PBKD2: PBKDF2 (Password-Based Key Derivation Function) utilizes a pseudorandom function, such as hash-based message authentication code, to the given text or password along with a salt value this process is repeated several times to produce an encrypted key. The computational work that generates this encrypted key makes it difficult to crack the password. The number of computations needed to generate this key has increased as the processing (CPU) speed has increased. Adding a salt to the password reduces the ability to use precomputed hashes (rainbow tables) for attacks. Generally, a salt length of 68 -128 bits is used in this standard.

4. Explain how passwords are stored in Linux. (location, encryption algorithm, salt, and file permissions).

In Linux, passwords are stored in /etc/passwd file. The file permission is 644 i.e every user has the permission to read this file but only root/ user with root privileges has the permission to right into it. In this file, each row corresponds to one user and each field in a row is delimited by ':'

Format: <Username>:<Encrypted password>:<User ID) >:<Group ID>:<user info>:<User home directory>:<Login shell>

If the Encrypted password has '$' in it then the password hash is stored in /etc/shadow file. This file has the default permission set to 600 i.e it can be accessed only by root/ users with root privilege.

Format: <Username>:<Password hash>:< Last password change>:<min password age >:<max password age>:<warning period >:<inactivity period>:<expiration date>
The password filed in this is of the format $id$salt$hashed where $id is the encryption algorithm used (each hashing algorithm is assigned an id), $salt is a random string appended to password, $hashed is the password hash.