CCS6344 - DATABASE & CLOUD SECURITY
TRIMESTER 2 - 2410, 2023/2024

Assignment 1 Submission

Group name: 10

Lecture Section: TC2L

Tutorial Section: TT3L

Youtube Presentation Link: https://youtu.be/NRkII2YNr_o

Student Name	Student ID
MUHAMMAD NAZHAN HARRAZ BIN KHAIRUL HISHAM	1221301122
SIDDIQ FERHAD BIN KHAIRIL ANUAL	1211103095
SYED DANIAL IMTIAZ BIN SYED ABDUL RAHIM	1221301145

Proposal

Project Objective

To build a Library Management System web application to help with managing the day to day tasks of librarians.

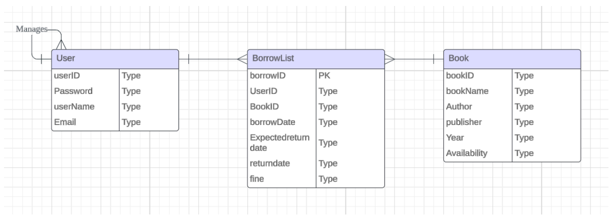
Design & Implementation

The application is a Library Management System where users can browse the books in the library and the staff can use it to log borrowed and returned books as well as automatically calculate fines for late returns. We plan to implement the application using the Python Django framework and to use MSSQL database server for the database.

Hardware and Software

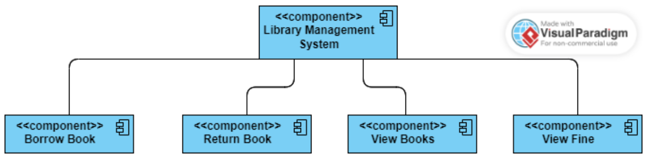
Because we are using the Python Django framework to develop this application, naturally this application will be mainly written in Python. For front-end purposes, we will be using HTML and a bit of CSS. To reduce the complexity, we will only be using Localhost for both Web and Database Servers.

System and Database design



Database

The database will be made up of three main tables. Which is User, BorrowList and Book.



System

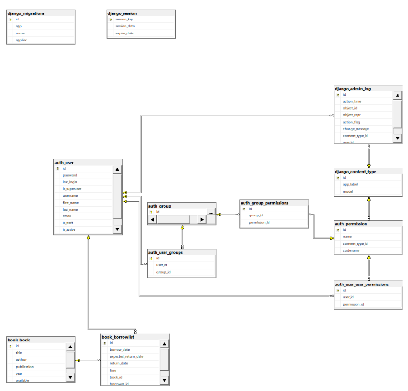
The system will be mainly made up of these 4 components, borrowing books, returning books, viewing books, and viewing fines.

Securing the database

In security, things mostly revolve around a triad. Confidentiality, Integrity, and Availability. In order to preserve it to the best of our abilities, we will implement appropriate security measures. To preserve Confidentiality, we will use encryption and masking on sensitive data. To preserve Integrity, we will use the database server or database roles following the principles of least privilege to mitigate any accidental change on the database. To preserve Availability, we will run constant backups as well as encrypting it to ensure a speedy recovery in case it is needed. There is also the matter of non-repudiation. To counter it, we will set up audits on the server as well as the database. We will also be disabling, renaming and changing the password of the sa account as a precaution.

Implementation of the application using SQL Database

1) Design

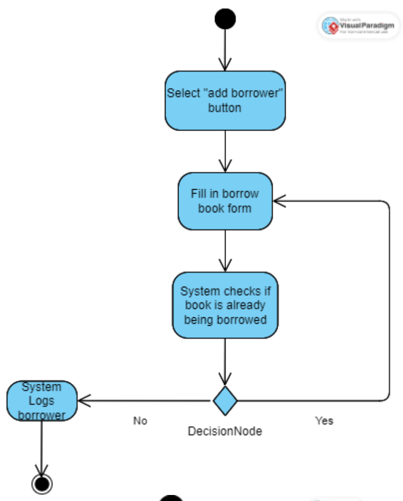


a) Database

The database consists of Django's built in tables as well as our custom tables. Django's built in tables are mainly used for access and authentication, logging etc. We implement two tables of our own which is book and borrowlist. Book holds the data of books in the library and borrowlist is intended to log borrows and returns.

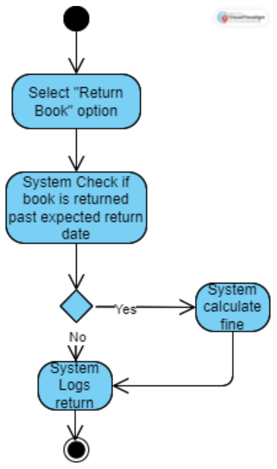
A clearer look: Database.png

b) Components


Select "add borrower" button
Fill in borrow book form
System checks if book is already being borrowed
System Logs borrower
No
DecisionNode
Yes

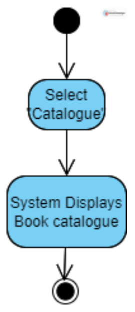
Log Borrowed Books

The Librarian uses the Log Borrowed Books component when a Patron wants to borrow a book. The librarian selects "add borrower" and they will be required to fill in a form with the relevant details. If the book is unavailable, the system will inform the librarian and the form will not be saved and logged. If the book is available, the system will save the form and log the book borrowed.


Select "Return Book" option
System Check if book is returned past expected return date
Yes
System calculate fine
No
System Logs return

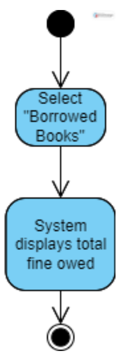
Log Returned Books

The Librarian uses the Log Returned Books component in the event that a Patron returns a borrowed book. The Librarian starts by selecting the "Return Book" option and a form will be required to fill with the relevant details. If the Patron returns the book later than the set return date, the System will calculate their fine amount and save it with the return log.



View Books

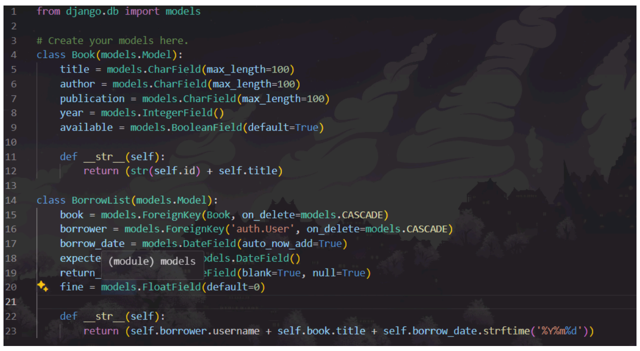
The Librarians and Patrons are able to view the Library's catalogue using the view catalogue component. They start by selecting the "Catalogue" option and the system will display the books currently in the Library's inventory.



View fines

In the event that the user returns a book later than the set return date, they will incur fines. The fine amount is calculated at the date of the book's return. The sum of the fines are displayed in "Borrowed Books" and the user can view it by selecting it.

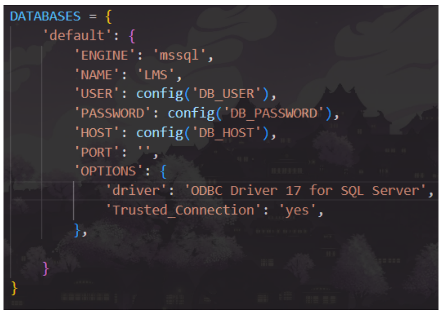
2) Description



```python
from django.db import models

# Create your models here.
class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.CharField(max_length=100)
    publication = models.CharField(max_length=100)
    year = models.IntegerField()
    available = models.BooleanField(default=True)

    def __str__(self):
        return (str(self.id) + self.title)

class BorrowList(models.Model):
    book = models.ForeignKey(Book, on_delete=models.CASCADE)
    borrower = models.ForeignKey('auth.User', on_delete=models.CASCADE)
    borrow_date = models.DateField(auto_now_add=True)
    expecte (module) models dels.DateField()
    return_                 Field(blank=True, null=True)
    fine = models.FloatField(default=0)

    def __str__(self):
        return (self.borrower.username + self.book.title + self.borrow_date.strftime('%Y%m%d'))
```

a) Models

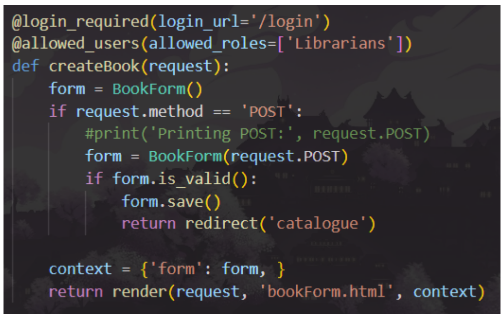
We first start by creating the models for the objects required by the application. In this case, it is Book and BorrowList. We will use Django's default user model for the users.



```python
DATABASES = {
    'default': {
        'ENGINE': 'mssql',
        'NAME': 'LMS',
        'USER': config('DB_USER'),
        'PASSWORD': config('DB_PASSWORD'),
        'HOST': config('DB_HOST'),
        'PORT': '',
        'OPTIONS': {
            'driver': 'ODBC Driver 17 for SQL Server',
            'Trusted_Connection': 'yes',
        },

    }
}
```

b) Database connection

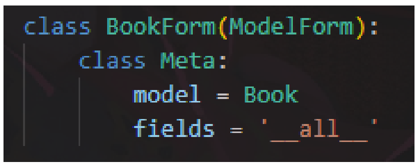
We then connect to the database by filling in the database settings in the **settings.py** file located in the Library Management System folder. After doing this, we are able to **makemigrations** and **migrate** the application. This will create the required tables in the database.



```python
@login_required(login_url='/login')
@allowed_users(allowed_roles=['Librarians'])
def createBook(request):
    form = BookForm()
    if request.method == 'POST':
        #print('Printing POST:', request.POST)
        form = BookForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('catalogue')

    context = {'form': form, }
    return render(request, 'bookForm.html', context)
```

c) Views

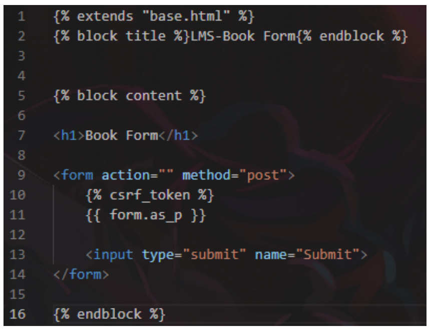
Next, we create our views in the **views.py** file of the book folder. Here is where we implement the back-end of our website. As an example, beside is the view for a page where Librarians can add books into the database. In this view, we pass in a form named BookForm and if the request method is POST, the book is then saved into the database and the librarian will be redirected to the catalogue page. We also pass in a HTML file to be rendered as its front-end.



```python
class BookForm(ModelForm):
    class Meta:
        model = Book
        fields = '__all__'
```

d) Forms

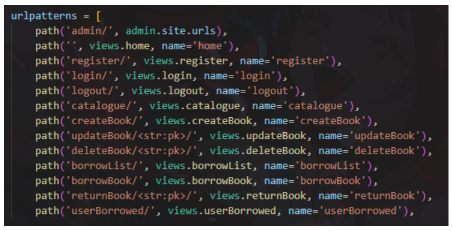
To make the forms we can use, we implement it in a file called **forms.py** in the **book** folder. We do it by assigning what model the form is intended for and specifying the fields we want. In this example, we want to fill in all the properties of the book model.



```
{% extends "base.html" %}
{% block title %}LMS-Book Form{% endblock %}


{% block content %}

<h1>Book Form</h1>

<form action="" method="post">
    {% csrf_token %}
    {{ form.as_p }}

    <input type="submit" name="Submit">
</form>

{% endblock %}
```

e) Templates

We then implement the front-end for the view to render in a folder called **templates** in the root folder. We used basic HTML and Django’s templating engine to construct the web view.



```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name='home'),
    path('register/', views.register, name='register'),
    path('login/', views.login, name='login'),
    path('logout/', views.logout, name='logout'),
    path('catalogue/', views.catalogue, name='catalogue'),
    path('createBook/', views.createBook, name='createBook'),
    path('updateBook/<str:pk>/', views.updateBook, name='updateBook'),
    path('deleteBook/<str:pk>/', views.deleteBook, name='deleteBook'),
    path('borrowList/', views.borrowList, name='borrowList'),
    path('borrowBook/', views.borrowBook, name='borrowBook'),
    path('returnBook/<str:pk>/', views.returnBook, name='returnBook'),
    path('userBorrowed/', views.userBorrowed, name='userBorrowed'),
```

f) Urls

Before a view could be rendered, we need to register its URL in **urls.py** in the **LibraryManagementSystem** folder.

3) Security Measures

a) Parameterised queries

The Django framework uses parameterised queries as its standard query and discourages use of raw queries. This ensures that the application is safe from SQL injections.

b) Decouple database connection

Using the extension **python-decouple**, the database connection settings are saved in a **.env** file instead of being hardcoded into the application’s code in **settings.py**. This ensures that any Login credentials remain hidden from prying eyes.

c) Access and Authorization

Mainly implemented in the **decorators.py** file in the book folder along with Django’s own features.

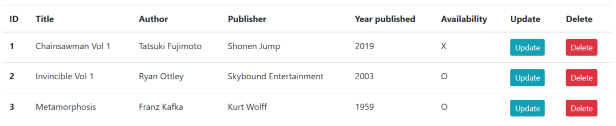
The **unauthenticated_user** functions checks for users that are yet to be authenticated and if true it will allow access, if not the user will be redirected to the home page. Used for the login and registration page.

The **allowed_users** function accepts an array of roles in the Library Management System. The passed role groups will be checked and if true it will grant access. Mainly used for restricting users from librarian components.

4) Tests

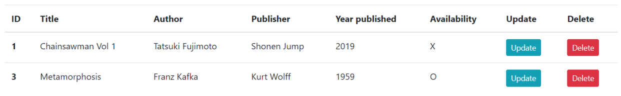
Deleting

Before



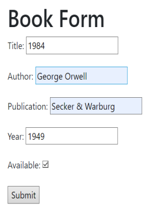
ID	Title	Author	Publisher	Year published	Availability	Update	Delete
1	Chainsawman Vol 1	Tatsuki Fujimoto	Shonen Jump	2019	X	Update	Delete
2	Invincible Vol 1	Ryan Ottley	Skybound Entertainment	2003	O	Update	Delete
3	Metamorphosis	Franz Kafka	Kurt Wolff	1959	O	Update	Delete

After



ID	Title	Author	Publisher	Year published	Availability	Update	Delete
1	Chainsawman Vol 1	Tatsuki Fujimoto	Shonen Jump	2019	X	Update	Delete
3	Metamorphosis	Franz Kafka	Kurt Wolff	1959	O	Update	Delete

Adding



Book Form

Title: 1984

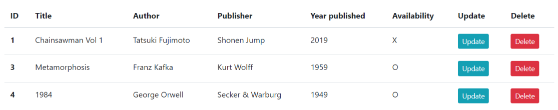
Author: George Orwell

Publication: Secker & Warburg

Year: 1949

Available: ☑

Submit



ID	Title	Author	Publisher	Year published	Availability	Update	Delete
1	Chainsawman Vol 1	Tatsuki Fujimoto	Shonen Jump	2019	X	Update	Delete
3	Metamorphosis	Franz Kafka	Kurt Wolff	1959	O	Update	Delete
4	1984	George Orwell	Secker & Warburg	1949	O	Update	Delete

Threat Modelling

Below is the assessment of our Library Management System (LMS) using the STRIDE and DREAD threat modelling frameworks, with scores assigned to each DREAD category. The final Threat Rating is calculated by averaging the DREAD scores.

STRIDE and DREAD Assessment

Risk	Category (STRIDE)	D	R	E	A	D	Threat Rating
Denial of Service (DoS)	D	7	8	6	9	7	7.4
Stealing SA Account credentials	E	9	8	7	8	9	8.2
DBA Perform malicious actions	R	10	5	5	9	6	7.0
SQL Server Remote Code Execution	S,T	10	9	8	10	9	9.2
Hijacking	S,E	9	8	7	9	8	8.2
Code Injection	S,T,I	8	9	9	10	8	8.8
Spoofing	S	7	7	8	8	7	7.4
Whole Value Substitution Attack	T	8	6	7	7	6	6.8

PDPA 2010

Categorization of Personnel

Under the PDPA 2010, the personnel involved in the LMS can be categorised as follows:

- **Data User:** The organisation (the team managing the LMS) responsible for processing personal data.
- **Data Subject**: The patrons and librarians whose personal data is being processed.
- **Data Processor**: Any third-party service providers managing data on behalf of the LMS organisation.

Data Lifecycle Compliance Mapping

Collection:

- PDPA Requirement: Obtain explicit consent from data subjects and collect only necessary data.
- Compliance Method: Use consent forms and clear privacy policies to inform users. Collect only data essential for library operations.
- Responsible Personnel: Data Protection Officer (DPO) ensures compliance with data collection practices.

Processing:

- PDPA Requirement: Process data fairly and lawfully for specified purposes.
- Compliance Method: Process data according to user consent and for the stated purposes only, such as managing library memberships and book loans.
- Responsible Personnel: IT Manager oversees lawful processing and ensures adherence to policies.

Storage:

- PDPA Requirement: Securely store personal data to prevent unauthorised access, loss, or damage.
- Compliance Method: Implement encryption, access controls, and regular security audits. Store data in secure, access-controlled environments.
- Responsible Personnel: Security Officer manages data security measures and conducts regular audits.

Access:

- PDPA Requirement: Allow data subjects to access and correct their personal data.
- Compliance Method: Provide user-friendly mechanisms for data subjects to request access and correction of their data.
- Responsible Personnel: Customer Support Team facilitates data access requests and corrections.

Disclosure:

- PDPA Requirement: Disclose personal data to third parties only with consent and for legitimate purposes.
- Compliance Method: Establish strict data sharing agreements and obtain explicit user consent before sharing data with third parties.

- Responsible Personnel: Legal Team ensures compliance with data sharing agreements and user consent processes.

Retention:

- PDPA Requirement: Retain data only as long as necessary for the purposes for which it was collected.
- Compliance Method: Implement data retention policies and regular reviews to delete unnecessary data.
- Responsible Personnel: Data Manager oversees data retention schedules and deletion processes.

Destruction:

- PDPA Requirement: Securely delete personal data that is no longer needed.
- Compliance Method: Use secure deletion methods to ensure data cannot be recovered.
- Responsible Personnel: IT Team performs secure data deletion and ensures compliance with destruction policies.

Penalties for Non-Compliance

Non-compliance with PDPA 2010 can result in severe penalties:

- Fines: Up to RM 300,000 depending on the severity of the violation.
- Imprisonment: Up to 2 years for serious breaches of data protection principles.
- Corporate Penalties: Companies may face substantial financial and reputational damage.

Security Measures Implementation

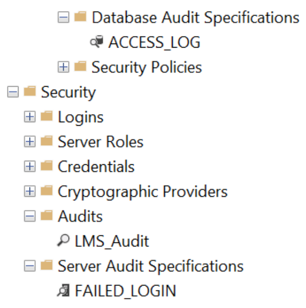
Our team has implemented a total of **8** security measures in the SQL database in order to protect the database from internal and external attacks.

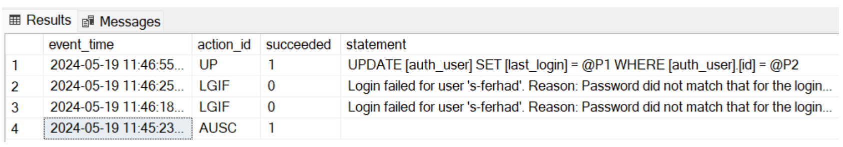
1) SQL Server Audit

➔ We have set up an audit for both the Server Audit Specification and the Database Audit Specification.

➔ Server Audit Specification (**FAILED_LOGIN**) will record all failed logins.

➔ Database Audit Specification (**ACCESS_LOG**) will record any updates to the **dbo.auth_user** table, logging the users who access the website based on changes to the **last_login** column.





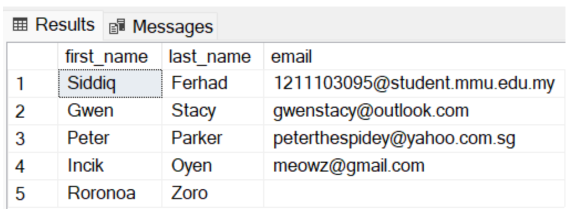
	event_time	action_id	succeeded	statement
1	2024-05-19 11:46:55...	UP	1	UPDATE [auth_user] SET [last_login] = @P1 WHERE [auth_user].[id] = @P2
2	2024-05-19 11:46:25...	LGIF	0	Login failed for user 's-ferhad'. Reason: Password did not match that for the login...
3	2024-05-19 11:46:18...	LGIF	0	Login failed for user 's-ferhad'. Reason: Password did not match that for the login...
4	2024-05-19 11:45:23...	AUSC	1	

2) Dynamic Data Masking

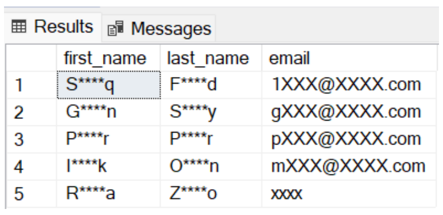
➔ Dynamic data masking (DDM) is used to obfuscate user's first name, last name, and email address in the **dbo.auth_user** table.

➔ **Email** masking method is applied to the **email** column.

➔ **Partial** masking method is applied to the **first_name** and **last_name** column.



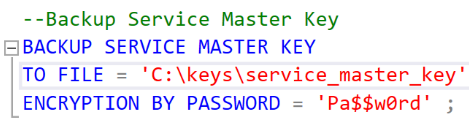
	first_name	last_name	email
1	Siddiq	Ferhad	1211103095@student.mmu.edu.my
2	Gwen	Stacy	gwenstacy@outlook.com
3	Peter	Parker	peterthespidey@yahoo.com.sg
4	Incik	Oyen	meowz@gmail.com
5	Roronoa	Zoro	



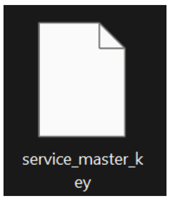
	first_name	last_name	email
1	S****q	F****d	1XXX@XXXX.com
2	G****n	S****y	gXXX@XXXX.com
3	P****r	P****r	pXXX@XXXX.com
4	I****k	O****n	mXXX@XXXX.com
5	R****a	Z****o	xxxx

3) Backup Service Master Key

➔ Since our instance uses encryption features, then backup of the key is required.



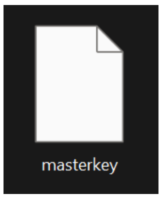
```
--Backup Service Master Key
BACKUP SERVICE MASTER KEY
TO FILE = 'C:\keys\service_master_key'
ENCRYPTION BY PASSWORD = 'Pa$$w0rd' ;
```



4) Backup Database Master Key

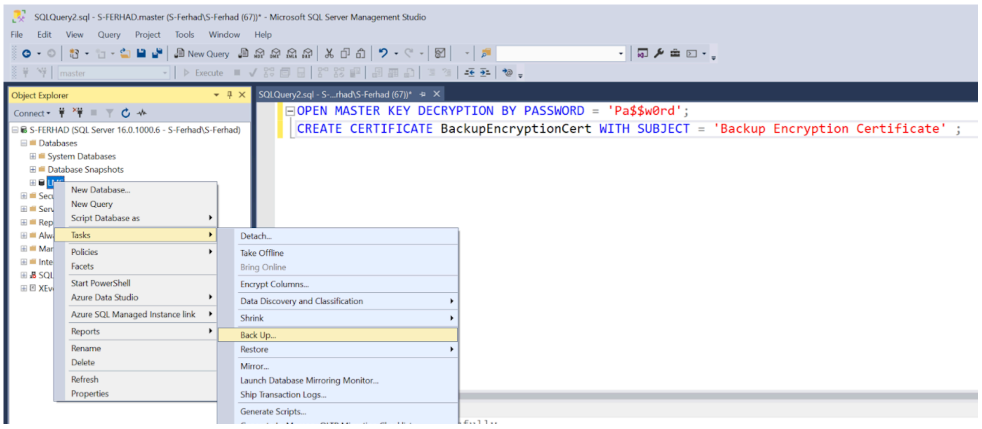
➔ Since Database Master Key is used to encrypt private keys and certificates within the database, then having a backup of the key is important.

```sql
--Backup Database Master Key
USE LMS
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Pa$$w0rd' ;
BACKUP MASTER KEY TO FILE = 'C:\keys\masterkey'
ENCRYPTION BY PASSWORD = 'Pa$$w0rd';
```

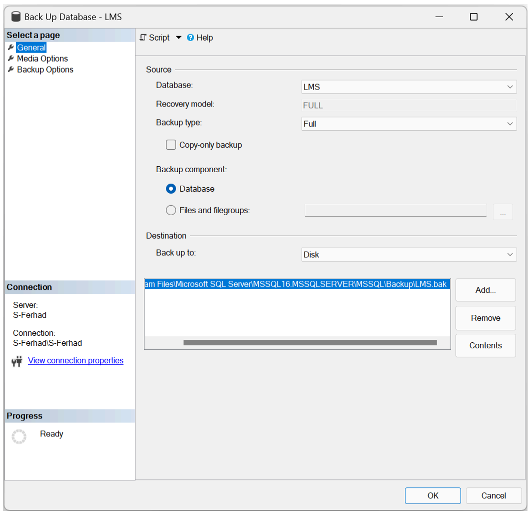


5) Backup Encrypted LMS

➔ We created a backup of the LMS database, in encrypted and compressed configuration.

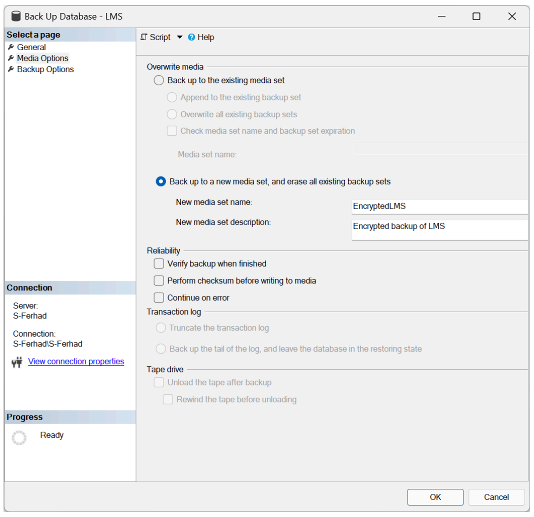


```sql
OPEN MASTER KEY DECRYPTION BY PASSWORD = 'Pa$$w0rd';
CREATE CERTIFICATE BackupEncryptionCert WITH SUBJECT = 'Backup Encryption Certificate' ;
```



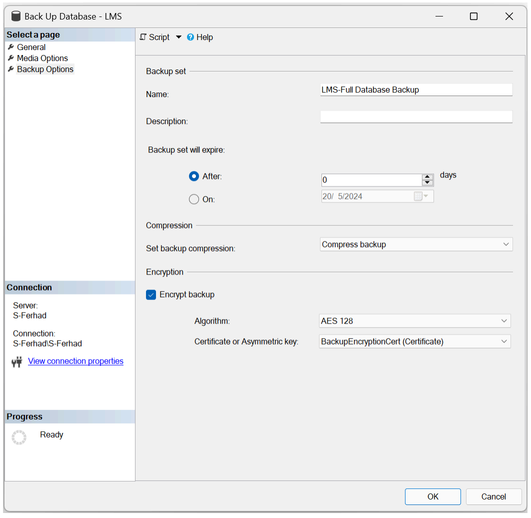
Source
Database: LMS
Recovery model: FULL
Backup type: Full
☐ Copy-only backup
Backup component:
[✓ Database]
Files and filegroups:
Destination
Back up to: Disk
am Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\Backup\LMS.bak

Connection
Server: S-Ferhad
Connection: S-Ferhad\S-Ferhad
Progress
Ready



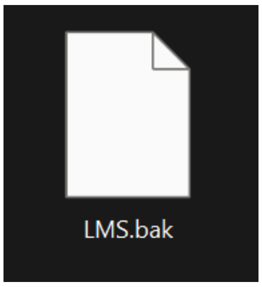
Overwrite media
Back up to the existing media set
Append to the existing backup set
Overwrite all existing backup sets
☐ Check media set name and backup set expiration
Media set name:
[✓ Back up to a new media set, and erase all existing backup sets]
New media set name: EncryptedLMS
New media set description: Encrypted backup of LMS
Reliability
☐ Verify backup when finished
☐ Perform checksum before writing to media
☐ Continue on error
Transaction log
Truncate the transaction log
Back up the tail of the log, and leave the database in the restoring state
Tape drive
☐ Unload the tape after backup
☐ Rewind the tape before unloading

Connection
Server: S-Ferhad
Connection: S-Ferhad\S-Ferhad
Progress
Ready



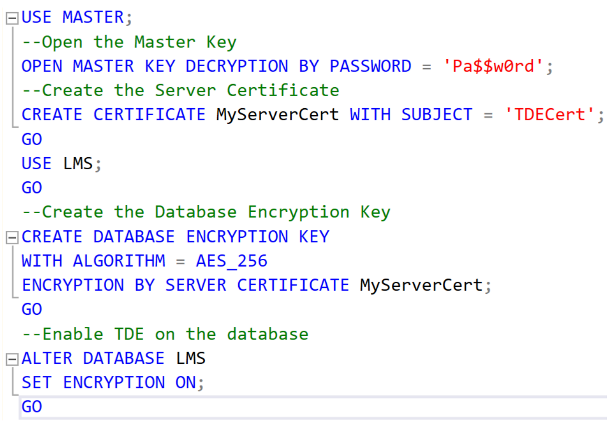
Backup set
Name: LMS-Full Database Backup
Description:
Backup set will expire:
[✓ After:] 0 days
On: 20/ 5/2024
Compression
Set backup compression: Compress backup
Encryption
✓ Encrypt backup
Algorithm: AES 128
Certificate or Asymmetric key: BackupEncryptionCert (Certificate)

Connection
Server: S-Ferhad
Connection: S-Ferhad\S-Ferhad
Progress
Ready

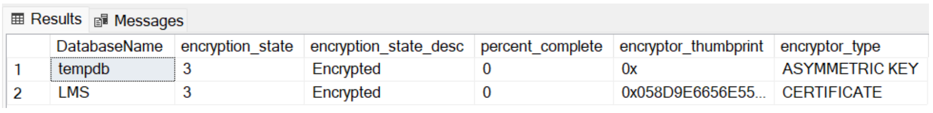


6) Transparent Data Encryption (TDE)

➔ We also implemented TDE that encrypts all data pages and log files of the database using a symmetric key called the **Database Encryption Key**.



```
USE MASTER;
--Open the Master Key
OPEN MASTER KEY DECRYPTION BY PASSWORD = 'Pa$$w0rd';
--Create the Server Certificate
CREATE CERTIFICATE MyServerCert WITH SUBJECT = 'TDECert';
GO
USE LMS;
GO
--Create the Database Encryption Key
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE MyServerCert;
GO
--Enable TDE on the database
ALTER DATABASE LMS
SET ENCRYPTION ON;
GO
```



Results | Messages

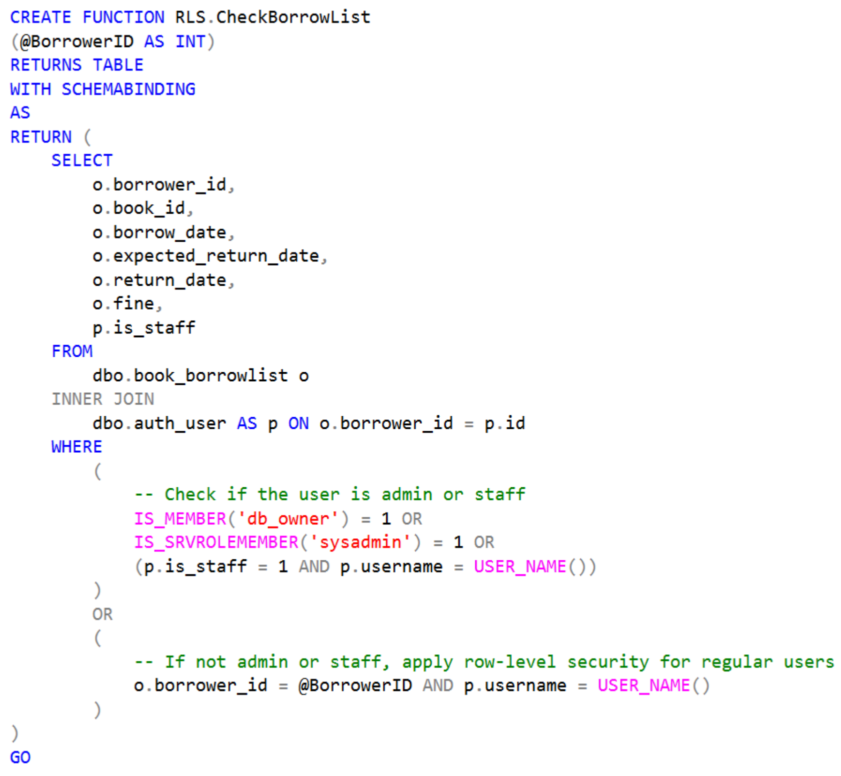
	DatabaseName	encryption_state	encryption_state_desc	percent_complete	encryptor_thumbprint	encryptor_type
1	tempdb	3	Encrypted	0	0x	ASYMMETRIC KEY
2	LMS	3	Encrypted	0	0x058D9E6656E55...	CERTIFICATE

7) Row-Level Security (RLS)

➔ To implement RLS, we need a Security Predicate and a Security Policy.

a) Security Predicate

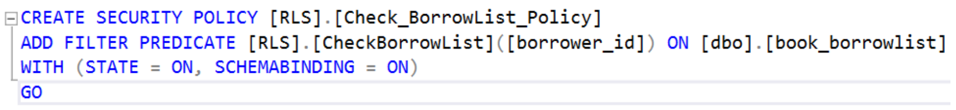
➔ First, we applied a **Filter Predicate** to filter the rows in **dbo.book_borrowlist** based on the user's **borrower_id**, ensuring that only relevant rows are returned when the table or view is queried.



```
CREATE FUNCTION RLS.CheckBorrowList
(@BorrowerID AS INT)
RETURNS TABLE
WITH SCHEMABINDING
AS
RETURN (
    SELECT
        o.borrower_id,
        o.book_id,
        o.borrow_date,
        o.expected_return_date,
        o.return_date,
        o.fine,
        p.is_staff
    FROM
        dbo.book_borrowlist o
    INNER JOIN
        dbo.auth_user AS p ON o.borrower_id = p.id
    WHERE
        (
            -- Check if the user is admin or staff
            IS_MEMBER('db_owner') = 1 OR
            IS_SRVROLEMEMBER('sysadmin') = 1 OR
            (p.is_staff = 1 AND p.username = USER_NAME())
        )
        OR
        (
            -- If not admin or staff, apply row-level security for regular users
            o.borrower_id = @BorrowerID AND p.username = USER_NAME()
        )
)
GO
```

b) Security Policy

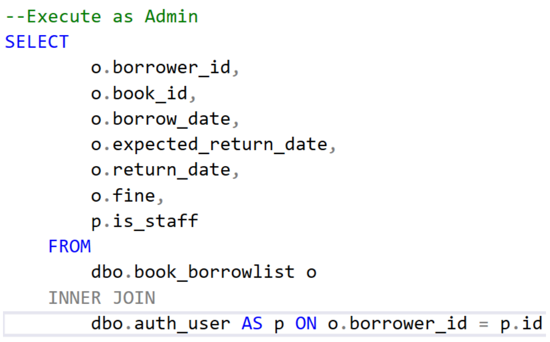
➔ Then, we added a security policy that applies the security predicate we just created as a filter on the **dbo.book_borrowlist** table.



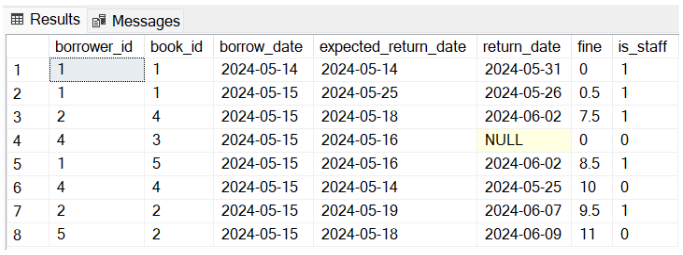
```
CREATE SECURITY POLICY [RLS].[Check_BorrowList_Policy]
ADD FILTER PREDICATE [RLS].[CheckBorrowList]([borrower_id]) ON [dbo].[book_borrowlist]
WITH (STATE = ON, SCHEMABINDING = ON)
GO
```

c) Implementing RLS

➔ Diagrams below show how we tested the result with users in three different roles: Admin, Librarian, and Patron.

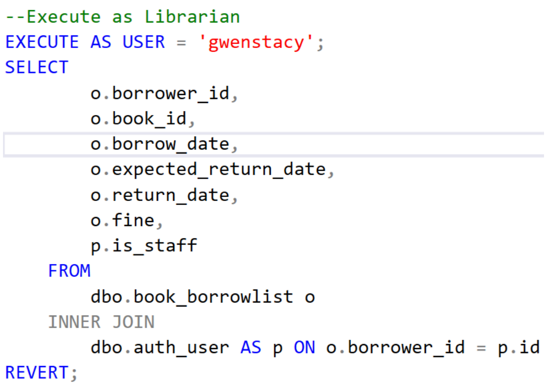


```
--Execute as Admin
SELECT
        o.borrower_id,
        o.book_id,
        o.borrow_date,
        o.expected_return_date,
        o.return_date,
        o.fine,
        p.is_staff
    FROM
        dbo.book_borrowlist o
    INNER JOIN
        dbo.auth_user AS p ON o.borrower_id = p.id
```

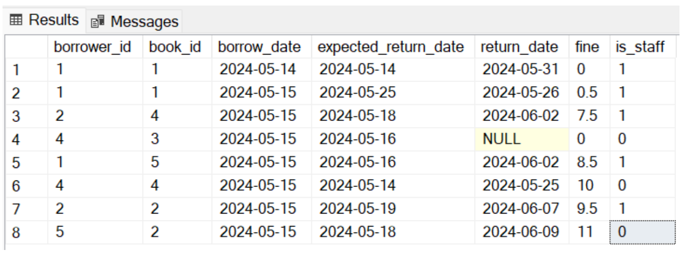


Results Messages

	borrower_id	book_id	borrow_date	expected_return_date	return_date	fine	is_staff
1	1	1	2024-05-14	2024-05-14	2024-05-31	0	1
2	1	1	2024-05-15	2024-05-25	2024-05-26	0.5	1
3	2	4	2024-05-15	2024-05-18	2024-06-02	7.5	1
4	4	3	2024-05-15	2024-05-16	NULL	0	0
5	1	5	2024-05-15	2024-05-16	2024-06-02	8.5	1
6	4	4	2024-05-15	2024-05-14	2024-05-25	10	0
7	2	2	2024-05-15	2024-05-19	2024-06-07	9.5	1
8	5	2	2024-05-15	2024-05-18	2024-06-09	11	0

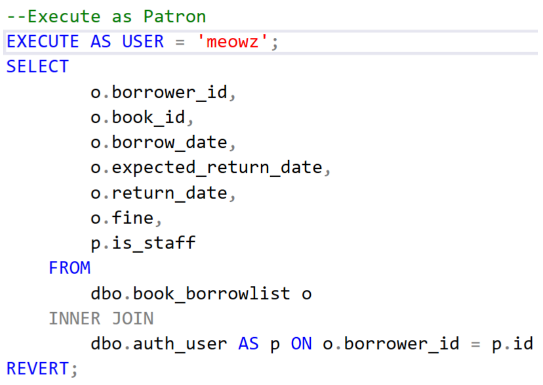


```
--Execute as Librarian
EXECUTE AS USER = 'gwenstacy';
SELECT
        o.borrower_id,
        o.book_id,
        o.borrow_date,
        o.expected_return_date,
        o.return_date,
        o.fine,
        p.is_staff
    FROM
        dbo.book_borrowlist o
    INNER JOIN
        dbo.auth_user AS p ON o.borrower_id = p.id
REVERT;
```

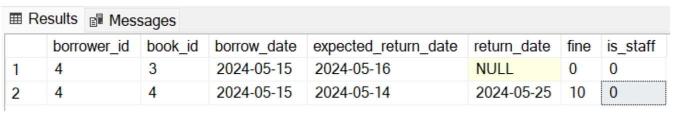


Results Messages

	borrower_id	book_id	borrow_date	expected_return_date	return_date	fine	is_staff
1	1	1	2024-05-14	2024-05-14	2024-05-31	0	1
2	1	1	2024-05-15	2024-05-25	2024-05-26	0.5	1
3	2	4	2024-05-15	2024-05-18	2024-06-02	7.5	1
4	4	3	2024-05-15	2024-05-16	NULL	0	0
5	1	5	2024-05-15	2024-05-16	2024-06-02	8.5	1
6	4	4	2024-05-15	2024-05-14	2024-05-25	10	0
7	2	2	2024-05-15	2024-05-19	2024-06-07	9.5	1
8	5	2	2024-05-15	2024-05-18	2024-06-09	11	0



```
--Execute as Patron
EXECUTE AS USER = 'meowz';
SELECT
        o.borrower_id,
        o.book_id,
        o.borrow_date,
        o.expected_return_date,
        o.return_date,
        o.fine,
        p.is_staff
    FROM
        dbo.book_borrowlist o
    INNER JOIN
        dbo.auth_user AS p ON o.borrower_id = p.id
REVERT;
```



Results Messages

	borrower_id	book_id	borrow_date	expected_return_date	return_date	fine	is_staff
1	4	3	2024-05-15	2024-05-16	NULL	0	0
2	4	4	2024-05-15	2024-05-14	2024-05-25	10	0

8) Protecting the sa Account

➔ In Mixed mode, there are 3 steps that we have taken to mitigate the risk and protect the sa Account:

a) Disabling the sa Account

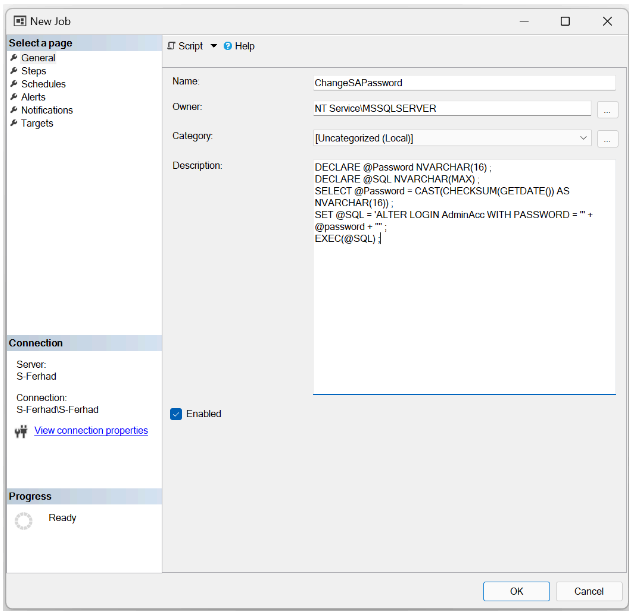
```
--Disable the sa account
ALTER LOGIN sa DISABLE;
GO
```

b) Renaming the sa Account

```
--Rename the sa account
ALTER LOGIN sa WITH NAME = AdminAcc;
GO
```

c) Enforcing password changes

➔ We enforced password rotation using a routine on SQL Server.



Name: ChangeSAPassword

Owner: NT Service\MSSQLSERVER

Category: [Uncategorized (Local)]

Description:

```
DECLARE @Password NVARCHAR(16) ;
DECLARE @SQL NVARCHAR(MAX) ;
SELECT @Password = CAST(CHECKSUM(GETDATE()) AS
NVARCHAR(16)) ;
SET @SQL = 'ALTER LOGIN AdminAcc WITH PASSWORD = ''' +
@password + ''' ;
EXEC(@SQL) ;
```

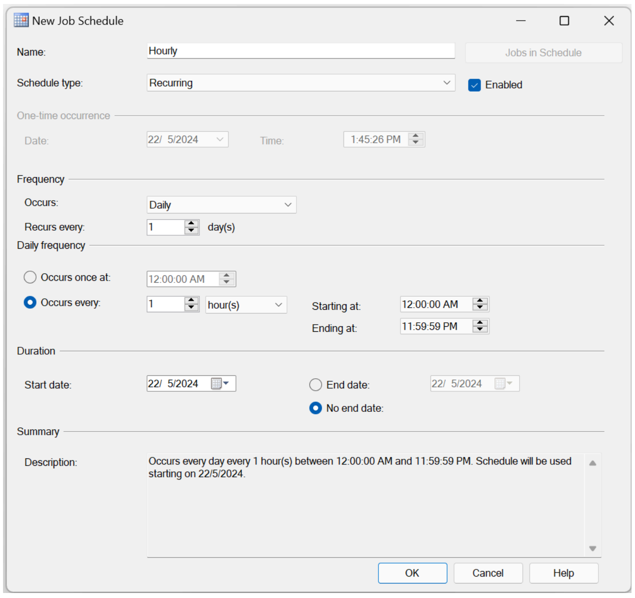
✓ Enabled

Server: S-Ferhad

Connection: S-Ferhad\S-Ferhad

➔ Create an SQL Server Agent job.

➔ Specify the name and enter the T-SQL script.



Name: Hourly

Schedule type: Recurring ✓ Enabled

One-time occurrence

Date: 22/ 5/2024 Time: 1:45:26 PM

Frequency

Occurs: Daily

Recurs every: 1 day(s)

Daily frequency

Occurs once at: 12:00:00 AM [✓ Occurs every: 1 hour(s)] Starting at: 12:00:00 AM Ending at: 11:59:59 PM

Duration

Start date: 22/ 5/2024 End date: 22/ 5/2024 [✓ No end date]

Summary

Description: Occurs every day every 1 hour(s) between 12:00:00 AM and 11:59:59 PM. Schedule will be used starting on 22/5/2024.

➔ Specify the name of the schedule and the frequency.