Logo, company name

Description automatically generated

**GROUP ASSIGNMENT**

**TECHNOLOGY PARK MALAYSIA**

**CT069-3-3 DBS**

**DATABASE SECURITY**

**APU3F2411CS(DA)**

|  |  |
| --- | --- |
| **Lecturer** | **: DR. HONG SIAW SWIN** |
| **Weightage** | **: 65%** |
| **Hand-out Date** | **: 22nd July 2025** |
| **Hand-in Date** | **: 17th August 2025** |

**Group 28**

|  |  |  |
| --- | --- | --- |
| **NO.** | **STUDENT NAME** | **TP NUMBER** |
| **1.** | **KOO KAH HENG** | **TP068681** |
| **2.** | **LIM KHAI XUAN** | **TP072200** |
| **3.** | **LIM ZHI JUN** | **TP069330** |
| **4.** | **LOW VI VIAN** | **TP068628** |

**Table of Contents**

[1.0 Introduction 4](#_Toc206228605)

[1.1 Data Dictionary 4](#_Toc206228606)

[1.2 Workload Matrix 11](#_Toc206228607)

[2.0 Permission Management 12](#_Toc206228608)

[2.1 Importance of Permission Management in Database Security 12](#_Toc206228609)

[2.2 Role Based Access Control (RBAC) 14](#_Toc206228610)

[2.3 Object Level Permission 16](#_Toc206228611)

[2.4 Row Level Security (RLS) 17](#_Toc206228612)

[2.5 Authorization Matrix 18](#_Toc206228613)

[2.6 Final Implementation Solution 25](#_Toc206228614)

[2.6.1 Create Roles 25](#_Toc206228615)

[2.6.2 Grant Tables Permission 26](#_Toc206228616)

[2.6.3 Manage User Access 30](#_Toc206228617)

[2.6.4 Implement RLS 37](#_Toc206228618)

[2.6.5 Restricted Column View 43](#_Toc206228619)

[2.6.6 Check Granted Access 50](#_Toc206228620)

[3.0 Data Protection 51](#_Toc206228621)

[3.1 Data Obfuscation 52](#_Toc206228622)

[3.2 Data Encryption 53](#_Toc206228623)

[3.3 Data Backup 54](#_Toc206228624)

[3.4 Data Classification Matrix 55](#_Toc206228625)

[3.5 Final Implementation Solution 58](#_Toc206228626)

[3.5.1 Database TDE 58](#_Toc206228627)

[3.5.2 Masking 59](#_Toc206228628)

[3.5.3 Column Level Encryption 60](#_Toc206228629)

[3.5.4 Backup Process 61](#_Toc206228630)

[4.0 Auditing 68](#_Toc206228631)

[4.1 Concept of Database Auditing 68](#_Toc206228632)

[4.2 Auditing Sources 69](#_Toc206228633)

[4.3 Auditing Techniques 70](#_Toc206228634)

[4.4 Database Security Audit Matrix 71](#_Toc206228635)

[4.5 Final Implementation Solution 74](#_Toc206228636)

[4.5.1 Agents Audit Log 74](#_Toc206228637)

[4.5.2 Sales Audit Log 76](#_Toc206228638)

[4.5.3 Commission Audit Log 77](#_Toc206228639)

[4.5.4 Tracking DCL Change 78](#_Toc206228640)

[4.5.5 Tracking DDL Change 79](#_Toc206228641)

[4.5.6 Tracking DML Change 80](#_Toc206228642)

[4.5.7 Tracking Login Activity 81](#_Toc206228643)

[4.5.8 Audit Logs Viewing 82](#_Toc206228644)

[4.5.9 Logon Triggers 85](#_Toc206228645)

[4.5.10 Number of Sessions Limitation on Specific Login 86](#_Toc206228646)

[4.5.11 Hours of SQL Server Access Limitation 87](#_Toc206228647)

[4.5.12 Temporal Tables 89](#_Toc206228648)

[5.0 Summary 91](#_Toc206228649)

[References 92](#_Toc206228650)

# **1.0 Introduction**

This project aims to redesign the Distributor Agent Management System (DAMS) used internally by Healthy Plants Organic Sdn Bhd to address its original database security issues. The company mainly focuses on organic vegetable distribution, and the original DAMS built by a full-stack development team, lacked many key security measures, including access control, data encryption, and auditing. As the company expanded and new IT departments are established, the system needs stronger security measures and different levels of data access permissions for each department.

The objective of this project is to address the lack of permission management, data protection, and auditing in the original system, ensuring that the system supports cross-departmental collaboration and meets database security requirements. This system will support multiple roles, including database administrator (DBA), agent (AGT), marketing staff (MKT), user portal developer (UPD), analyst (ANL), auditor (ADT), and end user (USR), and ensure each role can only access and manage data relevant to their responsibilities through permission assigned. In addition, this system will also implement data encryption, backup, and audit logging features.

This solution will meet Healthy Plants Organic Sdn Bhd’s database security needs, ensuring the system supports cross-departmental collaboration while maintaining the three principles of data confidentiality, integrity, and availability. Through these improvements, DAMS will provide restrict access control, data encryption, and clear audit trails, thereby effectively mitigating database security issues such as data leakage and misuse.

## 1.1 Data Dictionary

Table 1: Data Dictionary

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Table Name** | **Attribute Name** | **Contents** | **Type** | **Format** | **Range** | **PK/FK** | **FK Reference Table** |
| **Users** | UserID | Unique User ID | INT | 1 |  | PK |  |
| IdentificationNo | User’s Identification No | VARBINARY(MAX) | 900101145678 |  |  |  |
| Username | User’s Name | NVARCHAR(50) | admin1 |  |  |  |
| Password | User’s Password | VARBINARY(32) | admin123 |  |  |  |
| Role | Role of User in the System | NVARCHAR(50) | USR |  |  |  |
| CreatedAt | Date User was Created | DATETIME | 2025-08-02 19:30:03.940 |  |  |  |
| **Agents** | AgentID | Unique Agent ID | INT | 1 |  | PK |  |
| IdentificationNo | Agent’s Identification No | VARBINARY(MAX) | 900101145678 |  |  |  |
| Name | Agent’s Name | NVARCHAR(100) | Jessica |  |  |  |
| Email | Agent’s Email | NVARCHAR(100) | jessica@gmail.com |  |  |  |
| Phone | Agent’s Phone Number | NVARCHAR(30) | 0113456767 |  |  |  |
| Address | Agent’s Address | NVARCHAR(255) | 45 Jalan Teratai, Penang |  |  |  |
| Status | Employment Status | NVARCHAR(20) | Active | Active / Inactive |  |  |
| CreatedAt | Record Creation Time | DATETIME | 2025-08-02 19:30:10.290 |  |  |  |
| **Products** | ProductID | Unique Product ID | INT | 1 |  | PK |  |
| Name | Product Name | NVARCHAR(100) | Organic Spinach |  |  |  |
| Description | Product Description | NVARCHAR(255) | Fresh organic spinach 250g pack |  |  |  |
| Price | Product Price | DECIMAL(10,2) | 4.50 |  |  |  |
| CreatedAt | Record Creation Time | DATETIME | 2025-08-02 19:25:06.853 |  |  |  |
| **Sales** | SaleID | Unique Sale ID | INT | 1 |  | PK |  |
| AgentID | Foreign Key to Agent | INT | 1 |  | FK | Agents |
| ProductID | Foreign Key to Product | INT | 1 |  | FK | Products |
| Quantity | Units Sold | INT | 10 |  |  |  |
| TotalAmount | Total Sale Value | DECIMAL(10,2) | 50.00 |  |  |  |
| SaleDate | Date of Sale | DATETIME | 2025-08-02 19:30:10.287 |  |  |  |
| **Commission** | CommissionID | Unique Commission ID | INT | 1 |  | PK |  |
| AgentID | Foreign Key to Agent | INT | 1 |  | FK | Agents |
| SaleID | Foreign Key to Sale | INT | 1 |  | FK | Sales |
| CommissionRate | Rate of Commission | DECIMAL(5,2) | 5.00 |  |  |  |
| CommissionAmount | Amount Paid | DECIMAL(10,2) | 5.00 |  |  |  |
| CreatedAt | Record Creation Time | DATETIME | 2025-08-02 19:30:10.290 |  |  |  |
| **Notifications** | NotificationID | Unique Notification ID | INT | 1 |  | PK |  |
| Title | Header Title of Notification | NVARCHAR(100) | System Maintenance |  |  |  |
| Message | Body Content of Notification | NVARCHAR(255) | Check out the new agent profile dashboard! |  |  |  |
| Target | Role the Notification is Sent to | NVARCHAR(50) | All |  |  |  |
| CreatedAt | Notification Creation DateTime | DATETIME | 2025-08-02 19:25:06.853 |  |  |  |
| CreatedBy | Notification Created by UPD user | NVARCHAR(100) | admin1 |  |  |  |
| **MKT\_campaigns** | CampaignID | Unique Campaign ID | INT | 1 |  | PK |  |
| Title | Title of Campaign | NVARCHAR(100) | Eat Green Campaign |  |  |  |
| Description | Description of the Campaign | NVARCHAR(255) | Promote leafy greens to urban areas |  |  |  |
| StartDate | Campaign’s Starting Date | DATE | 2025-08-01 |  |  |  |
| EndDate | Campaign’s Ending Date | DATE | 2025-08-01 |  |  |  |
| Budget | Campaign’s Budget | DECIMAL(10,2) | 15000.00 |  |  |  |
| Status | Status of Campaign | NVARCHAR(20) | Planned | Planned / Completed / Implementing |  |  |
| CreatedBy | Marketing campaigns Created by MKT user | NVARCHAR(100) | admin1 |  |  |  |
| CreatedAt | Marketing Campaign Creation DateTime | DATETIME | 2025-08-02 19:25:06.853 |  |  |  |
| **Users\_AuditLog** | U\_AuditLogID | Unique Users Audit Log ID | INT | 1 |  | PK |  |
| UserID | Unique User ID | INT | 1 |  |  |  |
| Action | Action performed | NVARCHAR(10) | INSERT | INSERT / UPDATE / DELETE |  |  |
| Username | User’s Name | NVARCHAR(50) | user1 |  |  |  |
| Password | User’s Password | NVARCHAR(255) | user456 |  |  |  |
| Role | User’s Role | NVARCHAR(50) | USR |  |  |  |
| CreatedAt | Date User was Created | DATETIME | 2025-08-02 19:25:06.853 |  |  |  |
| ActionDate | Date and time of action | DATETIME | 2025-08-02 19:25:06.853 |  |  |  |
| PerformedBy | User that performing the action | NVARCHAR(100) | admin1 |  |  |  |
| **Agents\_AuditLog** | A\_AuditLogID | Unique Agents Audit Log ID | INT | 1 |  | PK |  |
| AgentID | Unique Agent ID | INT | 1 |  |  |  |
| Action | Action performed | NVARCHAR(10) | INSERT | INSERT / UPDATE / DELETE |  |  |
| IdentificationNo | Agent’s Identification No | VARBINARY | 900101145678 |  |  |  |
| Name | User’s Name | NVARCHAR(100) | admin1 |  |  |  |
| Email | Agent’s Email | NVARCHAR(100) | jessica@mail.com |  |  |  |
| Phone | Agent’s Phone Number | NVARCHAR(20) | 0111234556 |  |  |  |
| Address | Agent’s Address | NVARCHAR(255) | 45 Jalan Teratai, Penang |  |  |  |
| Status | Employment Status | NVARCHAR(20) | Active | Active / Inactive |  |  |
| CreatedAt | Record Creation Time | DATETIME | 2025-08-02 19:30:10.290 |  |  |  |
| ActionDate | Date and time of action | DATETIME | 2025-08-02 19:30:10.290 |  |  |  |
| PerformedBy | User that performing the action | NVARCHAR(100) | admin1 |  |  |  |
| **Sales\_AuditLog** | S\_AuditLogID | Unique Sales Audit Log ID | INT | 1 |  | PK |  |
| SaleID | Unique Sale ID | INT | 1 |  |  |  |
| AgentID | Unique Agent ID | INT | 1 |  | FK | Agents |
| ProductID | Foreign Key to Product | INT | 1 |  | FK | Products |
| Action | Action performed | NVARCHAR(20) | INSERT | INSERT / UPDATE / DELETE |  |  |
| Quantity | Units Sold | INT | 1 |  |  |  |
| TotalAmount | Total Sale Value | DECIMAL(10,2) |  |  |  |  |
| SaleDate | Date of Sale | DATETIME | 2025-08-02 19:30:10.290 |  |  |  |
| ActionDate | Date and time of action | DATETIME | 2025-08-02 19:30:10.290 |  |  |  |
| PerformedBy | User or role performing the action | NVARCHAR(100) | user1 |  |  |  |
| **Commission\_AuditLog** | C\_AuditLogID | Unique Commission Audit Log ID | INT | 1 |  | PK |  |
| CommissionID | Foreign Key to Sale | INT | 1 |  |  |  |
| AgentID | Foreign Key to Agent | INT | 1 |  | FK | Agents |
| SaleID | Foreign Key to Sale | INT | 1 |  | FK | Sales |
| Action | Action performed | NVARCHAR(10) | INSERT | INSERT / UPDATE / DELETE |  |  |
| CommissionRate | Rate of Commission | DECIMAL(5,2) | 5.00 |  |  |  |
| CommissionAmount | Amount Paid | DECIMAL(10,2) | 15.00 |  |  |  |
| CreatedAt | Record Creation Time | DATETIME | 2025-08-02 19:30:10.290 |  |  |  |
| ActionDate | Date and time of action | DATETIME | 2025-08-02 19:30:10.290 |  |  |  |
| PerformedBy | User or role performing the action | NVARCHAR(100) | admin1 |  |  |  |

# 

As shown in Table 1, the DAMS we designed consists of 11 tables: Users, Agents, Products, Sales, Commission, Notifications, MKT\_campaigns, Users\_AuditLog, Agents\_AuditLog, Sales\_AuditLog, and Commission\_AuditLog.

The Users table records information for the all the end users. The Agents table records personal information for all agents. This project also implements row level security to ensure that agents can only view their own records. All sensitive information, such as IdentificationNo and Password will be encrypted using Symmetric Key Encryption (AES-256) or Hash Encryption (SHA2-256). IdentificationNo is encrypted with AES-256, while Password is hashed with SHA2-256. For Email, Phone, and Address, dynamic masking will be applied to prevent unauthorized users from viewing the full information.

The Products table records detailed information and prices of products sold by Healthy Plants Organic Sdn Bhd. The Sales and Commission tables record the sales and commission details for each agent, so these two tables are linked to the Agents table via the AgentID. The Sales table also links to the Products table via ProductID, and the Commission table links to the Sales table via SaleID.

The Notifications table records all user notifications and implements restrict column view to ensure that users can only view notifications relevant to them. The MKT\_campaigns table records detailed information about all marketing campaigns.

The Users\_AuditLog, Agents\_AuditLog, Sales\_AuditLog, and Commission\_AuditLog tables are automatically populated by triggers. These four tables are used for auditing purposes and store all historical modification records to ensure traceability of operations.

## 1.2 Workload Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Components** | **Koo Kah Heng**  **(TP068681)** | **Lim Khai Xuan**  **(TP072200)** | **Low Vi Vian**  **(TP068628)** | **Lim Zhi Jun**  **(TP069330)** |
| **Introduction** | - | - | 50% | 50% |
| **Permission Management** | 25% | 25% | 25% | 25% |
| **Data Protection** | 25% | 25% | 25% | 25% |
| **Auditing** | 25% | 25% | 25% | 25% |
| **Summary** | 50% | 50% | - | - |
| Signature |  | A black background with a black square  Description automatically generated with medium confidence | A close-up of a name  Description automatically generated with medium confidence |  |

# **2.0 Permission Management**

## 2.1 Importance of Permission Management in Database Security

Permission management is the process of controlling access to data and resources in a database to ensure that only authorized users can view, modify, or manage this data, thereby protecting sensitive information from unauthorized access (Alooba, n.d.). It also ensures the integrity of the data and that the organization follows regulatory standards for strict access control. In the context of database, permission management determines which users can access which data, ensuring that only authorized users can perform specific operations. Permission management is very important and effective way in preventing the database being malicious access and data leakage.

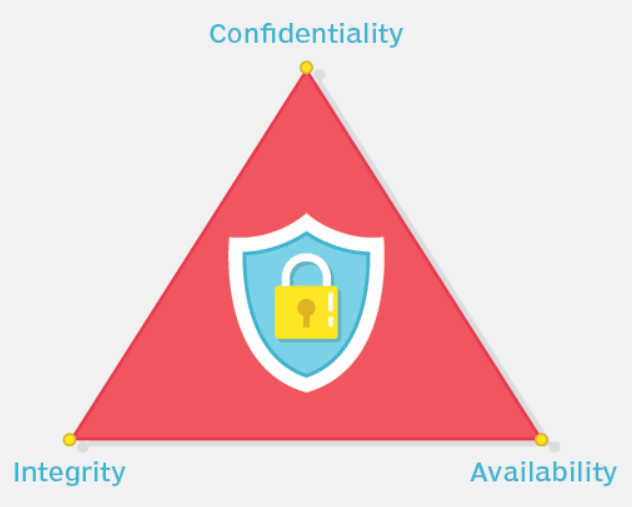


Figure 1: CIA Triad (Hashemi-Pour & Chai, 2023)

The CIA triad is a model designed to guide an organization’s information security policies, which highlights the three principles of Confidentiality, Integrity, and Availability (Hashemi-Pour & Chai, 2023). Confidentiality ensures that only authorized users can access specific data, integrity ensures that data is trustworthy and cannot be tampered with, and availability ensures that authorized individuals can access the information when needed (Hashemi-Pour & Chai, 2023). Permission management must satisfy all these three principles to ensure database security.

For the Distributor Agent Management System (DAMS) of Healthy Plants Organic Sdn Bhd, it is essential to guarantee that employees in various departments have different levels of access permissions, ensuring they can only access or modify the information related to their work, and that necessary information remains accessible. In addition, the changes made by each user should be recorded, allowing management team can track the activities of employees in each different department, holding individuals accountable for their actions.

Unauthorized access is the greatest threat to database security. Without effective permission management, users may gain access to information out of the range of their responsibilities. This will lead to serious security risks in the data, including data misuse, accidental deletion, or unauthorized modification. For example, if an agent is able to view the commission details of other agents, it could lead to data misuse. If a marketer can freely modify product pricing, it would result in malicious tampering with DAMS data, which would affect the business operations and customer trust of Healthy Plants Organic Sdn Bhd.

To address these security issues, DAMS need to implement Role-Based Access Control (RBAC), Object Level Permission, and Row-Level Security (RLS) to restrict data access by different roles and users. These measures ensure that only authorized users can access or modify specific data, maintaining data security and providing transparent auditing and accountability tracking for database management.

## 2.2 Role Based Access Control (RBAC)

As the number of users increases, directly assigning permissions to each user becomes more difficult and time consuming. This will also lead to higher security vulnerabilities caused by incorrect permissions allocated. The organization will also be challenged to track specific users’ access permissions, that difficult to quickly determine what permissions are assigned for each individual user. However, Role-Based Access Control (RBAC) provide a perfect method to solve this problem.

Role-Based Access Control (RBAC) is a method of granting or restricting user access to resources based on their assigned roles (Data Sunrise, n.d.). RBAC assigns permissions to the different roles first, and then users are assigned to their specific roles, rather than assigning permissions to each user individually (Data Sunrise, n.d.). This means that each role corresponds to a set of permissions, and users inherit these permissions through their roles directly, simplifying the process of permission management.

In addition, the core principle of RBAC is the Principle of Least Privilege (MongoDB, n.d.). This indicates that each user is granted only the minimum permissions required to perform their duties, thereby reducing the risk of permission abuse. For the Distributor Agent Management System (DAMS) of Healthy Plants Organic Sdn Bhd, RBAC provides more precise permission control, ensuring that permissions are reasonably allocated to various users according to the work needs of each department in the system. In DAMS, we created seven roles, including Database Administrator (DBA), Agent (AGT), Marketer (MKT), User Portal Developer (UPD), Analyst (ANL), Auditor (ADT), and End User (USR). We assigned appropriate permissions and set access restrictions for each role using GRANT and DENY commands, and then assigned each user to their respective roles. The detailed code implementation will be discussed in Section 2.6.

Moreover, the principle of least privilege in RBAC ensures that each user can only access or modify data relevant to their responsibilities. For example, the AGT role is only allowed to update their own information but is restricted from deleting data, while the ANL role only allowed to query marketing campaigns but not make any changes to marketing data. This implementation effectively enhances database security by preventing accidental data tampering or deletion.

The implementation of RBAC also help database administrators reduce their workload of assigning permissions. This is because administrators do not need to configure permissions for each user individually, that they can simply assign users to the corresponding role, and the system automatically grants those permissions for that role. This flexible permission configuration method enables organizations to more efficiently configure permissions for new employees onboarding and revoke access permissions when employees departure.

## 2.3 Object Level Permission

Object level permissions control the access and operations permissions on individual database objects, including tables, stored procedures, and views (OTI, 2023). These permissions explicitly define the SELECT, INSERT, UPDATE, and DELETE operations that different roles or users can perform on these objects.

For the Distributor Agent Management System (DAMS) of Healthy Plants Organic Sdn Bhd, object-level permissions are assigned strictly according to the principle of least privilege, ensuring that each role has only the minimum permissions required to perform their responsibilities. For example, the Analyst (ANL) role needs to analyze product pricing distribution, so they are granted the SELECT permission on the Products table. This allows them to query product information, but restricted them from performing any update, insert, or delete operations. This permission allocation method satisfies the analyst’s data needs while also preventing the risk of accidental or malicious data tampering, thereby ensuring more secure database access. Object-level permissions are used complementary with RBAC. Roles are first assigned different permissions and restrictions on different database objects through GRANT and DENY, and then users are assigned to the corresponding roles to inherit these permissions.

## 2.4 Row Level Security (RLS)

Row Level Security (RLS) is used to control users access to rows in a database table based on their identity (Microsoft, 2024). This means it allows administrators to restrict database access to specific rows of data rather than the entire tables. (Snowflake, n.d.). By implementing RLS, different users can see different data rows in the same table, depending on their identity, role, or other conditions. In DAMS, RLS can effectively address the permission gaps found in the implementations of RBAC and Object Level Permission, such as agents can query sales data of other agents, agents can check each other’s commissions, and end users can see notifications that do not belong to them. RLS implementation to solve these permission gaps will be discussed in Section 2.6.

RLS is implemented using filter predicates to determine whether a user has permission to access a specific row of data based on their role and identity information (Microsoft, 2024). Each time a user queries the data, RLS automatically applies these filtering rules based on the query condition and user identity, silently filtering and returning only the rows that the user is authorized to access. This further strengthens permission control in the database, ensuring that each user can only see the data they are authorized to view, preventing data leakage and permission abuse. Due to RLS is applied directly to the data, it simplifies the management of data access permissions (Teradata, n.d.). This means that by binding permissions to data rows rather than directly managing permissions for each user, thereby ensuring the complexity of manual configuration and permission management is effectively reduced.

## 2.5 Authorization Matrix

The following table shows the authorization matrix that defines the access permissions for different roles in this DAMS based on the principle of least privilege. This DAMS includes:

* **11 Tables**: Users, Agents, Products, Commission, Notifications, MKT\_campaigns, Users\_AuditLog, Agents\_AuditLog, Sales\_AuditLog, and Commission\_AuditLog
* **7 Roles**: DBA, AGT, MKT, UPD, ANL, ADT, and USR
* **4 Stored Procedures**: CreateUserAndAssignRole, DeleteUserAndLogin, usp\_GetDecryptedUsers, and CheckRoleMembership.
* **7 Restricted Column Views**: vw\_Notifications\_User, vw\_Notifications\_Limited, vw\_MKTCampaigns\_User, vw\_MKTCampaigns\_Limited, vw\_Sales\_Restricted, vw\_MonthlyProductSales, and vw\_Product\_User.

The Database Administrator (DBA) role has full access and execution permissions for all tables and stored procedures, except for the audit logs.

All roles are restricted to modify or delete the four audit log tables (Users\_AuditLog, Agents\_AuditLog, Sales\_AuditLog, and Commission\_AuditLog). Only the Database Administrator (DBA) and Auditor (ADT) roles are allowed to read these logs, thereby preventing malicious tampering with the historical records. The User Portal Developer (UPD) role also has read access to the Users\_AuditLog table to facilitate monitoring the changes of user accounts.

The Agent (AGT) role has restricted access permissions, meaning they can only operate on their own data. For example, they have SELECT and UPDATE permissions on the Agents table, SELECT, INSERT, and UPDATE permissions on the Sales table, and SELECT permission on the Commission table, but all of these permissions are set as restricted access. This indicates that agent can only update their personal information, insert and update their own sales data, and view their own commissions.

The End User (USR) role can only access notifications, marketing campaigns, and product information through secure views (vw\_Notifications\_User, vw\_MKTCampaigns\_User, vw\_Product\_User) and cannot directly query any table other than the Users table. For the Users table, they can only view their own personal information, but are not allowed to perform any INSERT, UPDATE, or DELETE operations.

In addition, the Marketing (MKT) role is responsible for managing marketing campaigns. Therefore, they have full access to the ducts and MKT\_campaigns tables, but they cannot access or modify any other tables.

The Analyst (ANL) role relies on views (vw\_Sales\_Restricted, vw\_MonthlyProductSales) to perform analysis tasks. This indicates that DAMS does not allow them to modify any tables and they only have SELECT permissions on the Products and MKT\_campaigns tables, which provides them with sufficient data for analysis tasks. The detailed explanation of all these restricted column views for each role will be provided in Section 2.6.5.

DAMS also uses stored procedures (CreateUserAndAssignRole, DeleteUserAndLogin, CheckRoleMembership) to manage the process of adding and deleting users. These three stored procedures can only be executed by the DBA, UPD, and ADT roles to prevent unauthorized users from creating or deleting accounts, thereby ensuring database security. In addition, another stored procedure (usp\_GetDecryptedUsers) can only be executed by ADT role to prevent end user’s sensitive information leakage. The more detailed explanation of these four stored procedures will be provided in Section 2.6.3.

Table 2: Authorization Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| **Role** | **Permission Type** | **Object** | **Privilege(s)** |
| DBA | Grant | TABLE **Agents** | SELECT, INSERT, UPDATE, DELETE |
| DBA | Grant | TABLE **Products** | SELECT, INSERT, UPDATE, DELETE |
| DBA | Grant | TABLE **Sales** | SELECT, INSERT, UPDATE, DELETE |
| DBA | Grant | TABLE **Commission** | SELECT, INSERT, UPDATE, DELETE |
| DBA | Grant | TABLE  **Users\_AuditLog**  **Agents\_AuditLog**  **Sales\_AuditLog**  **Commission\_AuditLog** | SELECT |
| DBA | Deny | TABLE  **Users\_AuditLog**  **Agents\_AuditLog**  **Sales\_AuditLog**  **Commission\_AuditLog** | INSERT, UPDATE, DELETE |
| DBA | Grant | TABLE **Users** | SELECT, INSERT, UPDATE, DELETE |
| DBA | Grant | TABLE **Notifications** | SELECT, INSERT, UPDATE, DELETE |
| DBA | Grant | TABLE **MKT\_campaigns** | SELECT, INSERT, UPDATE, DELETE |
| DBA | Grant | S.P. Procedure  **CreateUserAndAssignRole**  **DeleteUserAndLogin**  **CheckRoleMembership** | EXECUTE |
| AGT | Grant | TABLE **Agents** | SELECT, UPDATE (SELF) |
| AGT | Deny | TABLE **Agents** | INSERT, DELETE |
| AGT | Grant | TABLE **Products** | SELECT |
| AGT | Deny | TABLE **Products** | INSERT, UPDATE, DELETE |
| AGT | Grant | TABLE **Sales** | SELECT, INSERT, UPDATE (SELF) |
| AGT | Deny | TABLE **Sales** | DELETE |
| AGT | Grant | TABLE **Commission** | SELECT (SELF) |
| AGT | Deny | TABLE **Commission** | INSERT, UPDATE, DELETE |
| AGT | Deny | TABLE  **Users\_AuditLog**  **Agents\_AuditLog**  **Sales\_AuditLog**  **Commission\_AuditLog** | SELECT, INSERT, UPDATE, DELETE |
| AGT | Deny | TABLE **Users** | SELECT, INSERT, UPDATE, DELETE |
| AGT | Deny | TABLE **Notifications** | SELECT, INSERT, UPDATE, DELETE |
| AGT | Deny | TABLE **MKT\_campaigns** | SELECT, INSERT, UPDATE, DELETE |
| AGT | Grant | View  **vw\_MKTCampaigns\_Limited** | SELECT |
| MKT | Deny | TABLE **Agents** | SELECT, INSERT, UPDATE, DELETE |
| MKT | Grant | TABLE **Products** | SELECT, INSERT, UPDATE, DELETE |
| MKT | Deny | TABLE **Sales** | SELECT, INSERT, UPDATE, DELETE |
| MKT | Deny | TABLE **Commission** | SELECT, INSERT, UPDATE, DELETE |
| MKT | Deny | TABLE  **Users\_AuditLog**  **Agents\_AuditLog**  **Sales\_AuditLog**  **Commission\_AuditLog** | SELECT, INSERT, UPDATE, DELETE |
| MKT | Deny | TABLE **Users** | SELECT, INSERT, UPDATE, DELETE |
| MKT | Deny | TABLE **Notifications** | SELECT, INSERT, UPDATE, DELETE |
| MKT | Grant | TABLE **MKT\_campaigns** | SELECT, INSERT, UPDATE, DELETE |
| MKT | Grant | View  **vw\_Sales\_Restricted** | SELECT |
| UPD | Deny | TABLE **Agents** | SELECT, INSERT, UPDATE, DELETE |
| UPD | Grant | TABLE **Products** | SELECT |
| UPD | Deny | TABLE **Products** | INSERT, UPDATE, DELETE |
| UPD | Deny | TABLE **Sales** | SELECT, INSERT, UPDATE, DELETE |
| UPD | Deny | TABLE **Commission** | SELECT, INSERT, UPDATE, DELETE |
| UPD | Grant | TABLE **Users\_AuditLog** | SELECT |
| UPD | Deny | TABLE **Users\_AuditLog** | INSERT, UPDATE, DELETE |
| UPD | Deny | TABLE  **Agents\_AuditLog**  **Sales\_AuditLog**  **Commission\_AuditLog** | SELECT, INSERT, UPDATE, DELETE |
| UPD | Grant | TABLE **Users** | SELECT, INSERT, UPDATE, DELETE |
| UPD | Grant | TABLE **Notifications** | SELECT, INSERT, UPDATE, DELETE |
| UPD | Deny | TABLE **MKT\_campaigns** | SELECT, INSERT, UPDATE, DELETE |
| UPD | Grant | View  **vw\_MKTCampaigns\_Limited** | SELECT |
| UPD | Grant | S.P. Procedure  **CreateUserAndAssignRole**  **DeleteUserAndLogin**  **CheckRoleMembership** | EXECUTE |
| ANL | Deny | TABLE **Agents** | SELECT, INSERT, UPDATE, DELETE |
| ANL | Grant | TABLE **Products** | SELECT |
| ANL | Deny | TABLE **Products** | INSERT, UPDATE, DELETE |
| ANL | Deny | TABLE **Sales** | SELECT, INSERT, UPDATE, DELETE |
| ANL | Deny | TABLE **Commission** | SELECT, INSERT, UPDATE, DELETE |
| ANL | Deny | TABLE  **Users\_AuditLog**  **Agents\_AuditLog**  **Sales\_AuditLog**  **Commission\_AuditLog** | SELECT, INSERT, UPDATE, DELETE |
| ANL | Deny | TABLE **Users** | SELECT, INSERT, UPDATE, DELETE |
| ANL | Deny | TABLE **Notifications** | SELECT, INSERT, UPDATE, DELETE |
| ANL | Grant | TABLE **MKT\_campaigns** | SELECT |
| ANL | Deny | TABLE **MKT\_campaigns** | INSERT, UPDATE, DELETE |
| ANL | Grant | View  **vw\_Notifications\_Limited**  **vw\_Sales\_Restricted**  **vw\_MonthlyProductSales** | SELECT |
| ADT | Grant | TABLE **Agents** | SELECT, INSERT, UPDATE, DELETE |
| ADT | Grant | TABLE **Products** | SELECT |
| ADT | Deny | TABLE **Products** | INSERT, UPDATE, DELETE |
| ADT | Grant | TABLE **Sales** | SELECT, INSERT, UPDATE, DELETE |
| ADT | Grant | TABLE **Commission** | SELECT, INSERT, UPDATE, DELETE |
| ADT | Grant | TABLE  **Users\_AuditLog**  **Agents\_AuditLog**  **Sales\_AuditLog**  **Commission\_AuditLog** | SELECT |
| ADT | Deny | TABLE  **Users\_AuditLog**  **Agents\_AuditLog**  **Sales\_AuditLog**  **Commission\_AuditLog** | INSERT, UPDATE, DELETE |
| ADT | Grant | TABLE **Users** | SELECT, INSERT, UPDATE, DELETE |
| ADT | Grant | TABLE **Notifications** | SELECT |
| ADT | Deny | TABLE **Notifications** | INSERT, UPDATE, DELETE |
| ADT | Grant | TABLE **MKT\_campaigns** | SELECT |
| ADT | Deny | TABLE **MKT\_campaigns** | INSERT, UPDATE, DELETE |
| ADT | Grant | S.P. Procedure  **CreateUserAndAssignRole**  **DeleteUserAndLogin**  **usp\_GetDecryptedUsers**  **CheckRoleMembership** | EXECUTE |
| ADT | Grant | View  **vw\_Notifications\_Limited** | SELECT |
| USR | Deny | TABLE **Agents** | SELECT, INSERT, UPDATE, DELETE |
| USR | Deny | TABLE **Products** | SELECT, INSERT, UPDATE, DELETE |
| USR | Deny | TABLE **Sales** | SELECT, INSERT, UPDATE, DELETE |
| USR | Deny | TABLE **Commission** | SELECT, INSERT, UPDATE, DELETE |
| USR | Deny | TABLE  **Users\_AuditLog**  **Agents\_AuditLog**  **Sales\_AuditLog**  **Commission\_AuditLog** | SELECT, INSERT, UPDATE, DELETE |
| USR | Grant | TABLE **Users** | SELECT (SELF) |
| USR | Deny | TABLE **Users** | INSERT, UPDATE, DELETE |
| USR | Deny | TABLE **Notifications** | SELECT, INSERT, UPDATE, DELETE |
| USR | Deny | TABLE **MKT\_campaigns** | SELECT, INSERT, UPDATE, DELETE |
| USR | Grant | View  **vw\_Notifications\_User**  **vw\_MKTCampaigns\_User**  **vw\_Product\_User** | SELECT |

## 2.6 Final Implementation Solution

### **2.6.1 Create Roles**

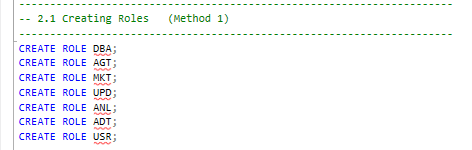


Figure 2: Create Roles

This SQL code creates seven different roles, including database administrator (DBA), agent (AGT), marketing staff (MKT), user portal developer (UPD), analyst (ANL), auditor (ADT), and end user (USR). By creating these roles, different permissions can be assigned based on the specific responsibilities of each role, ensuring the principle of least privilege is followed. This implementation follows the Role Based Access Control method for permission allocation, which different roles are created first, and these roles are granted different permissions. Users are then assigned to these roles to inherit the corresponding permissions.

### **2.6.2 Grant Tables Permission**

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 3: Grant Tables Permission to Each Role (1)

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 4: Grant Tables Permission to Each Role (2)

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 5: Grant Tables Permission to Each Role (3)

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 6: Grant Tables Permission to Each Role (4)

The SQL code shown in Figures 3 to 6 are used to grant the access permission for different roles to the 11 tables in this DAMS, including the permission of SELECT, INSERT, UPDATE, and DELETE. These permission allocation is based on the Authorization Matrix in Table 2. Permissions are granted to the roles using the GRANT command, while the DENY command strictly limits the permissions that a role cannot obtain. The configuration of these permissions follows the principle of least privilege, ensuring that each role can only access and operate the data necessary to fulfil their responsibilities, thereby reducing potential security risks and data misuse.

The **DBA** role has full access to all tables, including SELECT, INSERT, UPDATE, and DELETE, except for the four audit log tables. This role also has SELECT permission on the four audit log tables (Users\_AuditLog, Agents\_AuditLog, Sales\_AuditLog, and Commission\_AuditLog), but cannot perform INSERT, UPDATE, or DELETE operations on these tables to ensure audit logs cannot be tampered.

The permissions of **AGT** role are configured as limited access, meaning although they have the permissions on updating data, they can only operate on their own data. This will be implemented using Row Level Security on Section 2.6.4. For example, they have SELECT and UPDATE permission on their own records in the Agents table, SELECT, INSERT, and UPDATE permission on their own sales data in the Sales table, and SELECT permission on their own commission information in the Commission table. However, they cannot perform any SELECT, INSERT, UPDATE, or DELETE operations on other agents’ data. For other tables, such as the audit logs and Users, Notifications, and MKT\_campaigns, they have no any permissions, meaning they cannot view or modify any information in these tables.

The **MKT** role has full access only to the Products and MKT\_campaigns tables, but cannot access or modify any other tables. This permission configuration aligns with their responsibilities, ensuring they can only access and modify data related to marketing campaigns, such as product information and campaign details.

The **UPD** role has full access to the Users and Notifications tables. They can also view the Products and Users\_AuditLog tables, but cannot perform any INSERT, UPDATE, or DELETE operations. This gives them sufficient permissions to manage users and notifications, while also monitoring changes in user accounts. By restricting their ability to modify data in the Products and Users\_AuditLog tables, ensuring unauthorized data changes are effectively prevented, minimizing the risk of data loss or confusion due to accidental modifications.

The **ANL** role is not allowed to modify any tables and only has SELECT permission on the Products and MKT\_campaigns tables. By viewing the data in these tables, they have enough information to perform analysis tasks.

The **ADT** role has full access to the Agents, Sales, Commission, and Users tables. This role also has SELECT permission on the four audit log tables, but cannot perform INSERT, UPDATE, or DELETE operations on these tables. For the Products, Notifications, and MKT\_campaigns tables, they can only view the data and are also limited from making any changes any changes.

The **USR** role can only access the data in the Users table and view their own personal information. They are not allowed to perform INSERT, UPDATE, or DELETE operations.

### **2.6.3 Manage User Access**

#### **2.6.3.1 Stored Procedure 1: Create User (CreateUserAndAssignRole)**

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 7: Create CreateUserAndAssignRole Stored Procedure

The SQL code shown above creates a stored procedure named CreateUserAndAssignRole. This stored procedure is used to create login accounts for users and assign a role to the user in this DAMS database. First, this stored procedure checks whether the username already exists as a login account in SQL Server. If not, it creates a new login account and sets the password. Next, this stored procedure will checks whether the user exists in the database. If the user has not been created yet, this stored procedure will creates the user and associates it with the previously created login account.

In addition, the ALTER ROLE command is used to add the specified user to the corresponding role. This stored procedure also ensures that each user can only appears once in each role. If the user is already a member of certain roles, this stored procedure will removes the user from the previous duplicate roles and then assigns this user to the current new role.

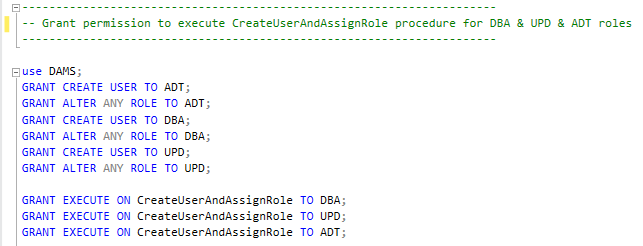


Figure 8: Grant Execute Permission

This SQL code grants the DBA, UPD, and ADT roles the permissions to create new users and modify user roles. This indicates that these three roles can create new user accounts and modify the membership of other roles in DBMS database. In addition, this code also grants the DBA, UPD, and ADT roles the permission to execute the CreateUserAndAssignRole stored procedure, allowing the user who as the roles of DBA, UPD, or ADT to use this stored procedure to create new users and assign roles.

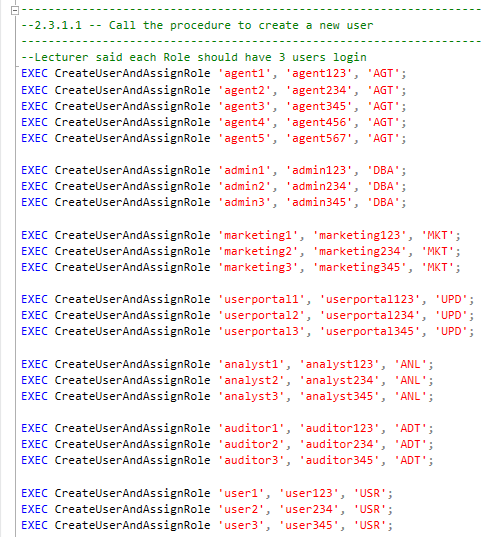


Figure 9: Execute Stored Procedure to Create New User

This SQL code executes CreateUserAndAssignRole stored procedure to create new users for different roles and assign the corresponding roles to these users.

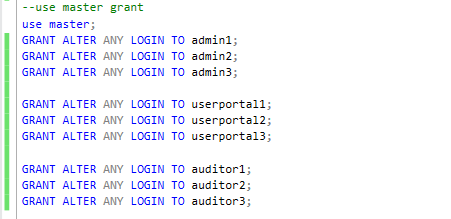


Figure 10: Grant Login Permission to Admin & User Portal & Auditor Using Master Database

Although the DBA, UPD, and ADT roles already have permission to execute the CreateUserAndAssignRole stored procedure (refer Figure 8), and the users inheriting these roles also have permission to execute CreateUserAndAssignRole at the database level. However, this stored procedure includes the operation of creating a server login (CREATE LOGIN), and this permission cannot be inherited through database roles. It requires the additional ALTER ANY LOGIN permission at the server level.

Therefore, this SQL code grants all the admins, user portal developers, and auditors the ALTER ANY LOGIN server-level permission to ensure these authorized users have the permissions to create new users and modify user roles, allowing them to successfully execute the CreateUserAndAssignRole stored procedure.

#### **2.6.3.2 Stored Procedure 2: Drop User (DeleteUserAndLogin)**

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 11: Create DeleteUserAndLogin Stored Procedure

The SQL code shown above creates a stored procedure named DeleteUserAndLogin. This stored procedure is used to delete the user’s record from the Users table, delete the user’s information from the database, and delete the user’s login account on SQL Server.

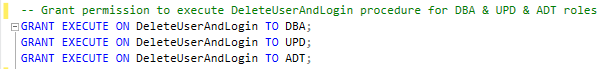


Figure 12: Grant Execute Permission

This SQL code grants the DBA, UPD, and ADT roles the permission to execute the DeleteUserAndLogin stored procedure. This indicates that users who as the roles of DBA, UPD, or ADT are allowed to use this stored procedure to delete users.

#### **2.6.3.3 Stored Procedure 3: Check User Information (usp\_GetDecryptedUsers)**

A screen shot of a computer code

AI-generated content may be incorrect.

Figure 13: Create usp\_GetDecryptedUsers Stored Procedure & Grant Execute Permission

This SQL code creates a stored procedure named usp\_GetDecryptedUsers. This stored procedure helps auditors to check all the decrypted IdentificationNo for all end users (customers), ensuring their identities can be quickly traced if they commit illegal activities. However, to prevent end user identities from being leaked, only the ADT role is granted the permission to execute the usp\_GetDecryptedUsers stored procedure. This indicates that only users who as the role of ADT is allowed to use this stored procedure to access end users’ sensitive information.

#### **2.6.3.4 Stored Procedure 4: Check Membership (CheckRoleMembership)**

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 14: Create CheckRoleMembership Stored Procedure

The SQL code creates a stored procedure named CheckRoleMembership that used to check all the members of a specific role. This stored procedure helps database administrators, user portal developers, and auditors quickly view the members of a particular role when managing roles and permissions in the database.

A group of black text

AI-generated content may be incorrect.

Figure 15: Grant Execute Permission

This SQL code grants the DBA, UPD, and ADT roles the permission to execute the CheckRoleMembership stored procedure. This indicates that users who as the roles of DBA, UPD, or ADT are allowed to use this stored procedure to view all the members of the specified role. When viewing, the system will display the usernames of all the users that belong to that specific role.

### **2.6.4 Implement RLS**

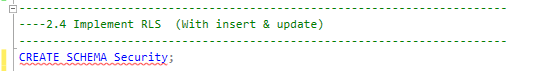


Figure 16: Create Security Schema

This SQL code creates a new database schema named Security, which is used to store tables, functions, and policies related to row level security.

#### **2.6.4.1 Users Table Predicate**

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 17: Create Users Table Predicate Function

The SQL code creates a row level security predicate function for the Users table named fn\_UsersSelfPredicate. This predicate function used to checks which rows of data the user can view for the Users table. As defined, SQL admin (including sysadmin and db\_owner), DBA, UPD, and ADT roles have permission to view all rows of the user data. By comparing the username of the user to determine whether the current user owns the row. Therefore, each user can only view their own data in the Users table.

#### **2.6.4.2 Sales Table Predicate**

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 18: Create Sales Table Predicate Function

The SQL code creates a row level security predicate function for the Sales table named fn\_SalesSelfPredicate. This predicate function used to implement access control for the sales data, ensuring the agents can only view their own sales data. As defined, SQL admin (including sysadmin and db\_owner), DBA, UPD, ANL, ADT, and MKT roles have permission to view all rows of the sales data. By comparing the username of the agent to determine whether the current agent owns the row. Therefore, each agent can only view their own sales data in the Sales table.

#### **2.6.4.3 Agents Table Predicate**

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 19: Create Agents Table Predicate Function

The SQL code creates a row level security predicate function for the Agents table named fn\_AgentsSelfPredicate. This predicate function used to ensures that agents can only view their own personal information. As defined, SQL admin (including sysadmin and db\_owner), DBA, UPD, and ADT roles have permission to view all rows of the agent data. By comparing the username of the agent to determine whether the current agent owns the row. Therefore, each agent can only view their own data in the Agents table.

#### **2.6.4.4 Commission Table Predicate**

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 20: Create Commission Table Predicate Function

The SQL code creates a row level security predicate function for the Commission table named fn\_CommissionSelfPredicate. This predicate function used to ensures that agents can only view their own commission information. As defined, SQL admin (including sysadmin and db\_owner), DBA, UPD, ANL, and ADT roles have permission to view commission information for all agents. However, agents can only view their own commission information in the Commission table. This is implemented by comparing the username of the agent to determine whether the current agent owns the row.

#### **2.6.4.5 Notifications Table Predicate**

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 21: Create Notifications Table Predicate Function

The SQL code creates a row level security predicate function for the Notifications table named fn\_NotificationSelfPredicate. This predicate function used to ensures that end users can only view notifications they have received. As defined, SQL admin (including sysadmin and db\_owner), DBA, UPD, ANL, and ADT roles have permission to view all notifications. However, end users can only view notifications targeted to them or notifications without a specific target.

#### **2.6.4.6 Create Security Policies for Each Table**

A screenshot of a computer security policy

AI-generated content may be incorrect.

Figure 22: Create Security Policies

This SQL code creates the security policies for each and adds the previously defined filtering predicate functions to these security policies, thereby implementing row level security. This ensures that users can only see the rows of data they are authorized to view, preventing data leakage and permission abuse.

In addition, the GRANT ALTER ANY SECURITY POLICY command is used to provide the permission to the DBA, ADT, and UPD roles, allowing them to modify the security policies in this DAMS database. The GRANT CONTROL ON SCHEMA::Security command also used to grants these three roles with full access control to all objects in the Security schema, including both predicate functions and security policies.

### **2.6.5 Restricted Column View**

#### **2.6.5.1 Notifications**

##### **View 1: End User (vw\_Notifications\_User)**

A screen shot of a computer

AI-generated content may be incorrect.

Figure 23: Create View vw\_Notifications\_User for End User

This SQL code creates a view named vw\_Notifications\_User to implement the restricted column view functionality. As shown in the figure above, this view will only displays the Title, Message, and CreatedAt columns from the Notifications table. Next, the SELECT permission for this view is granted to the USR role, allowing all end users to view the contents of this view.

Due to row level security has been implemented for the Notifications table in Section 2.6.4.5. Therefore, by combining the row level security and this restricted column view vw\_Notifications\_User, end users can only view their own notifications, and the notifications will displaying only the Title, Message, and CreatedAt columns.

##### **View 2: Analyst & Auditor (vw\_Notifications\_Limited)**

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 24: Create View vw\_Notifications\_Limited for Analyst & Auditor

This SQL code creates a view named vw\_Notifications\_Limited. This view will displays the Title, Message, TargetRole, CreatedAt, and CreatedBy columns from the Notifications table, and also masked the sensitive information. As shown in the figure above, all TargetRole columns are displayed as ‘Restricted’ to prevent analysts and auditors from knowing which end user the notification was sent to. If the CreatedBy column value is ‘Admin’, this view will also make this column value displayed as ‘System’ to protect administrator information. auditors to view its contents. Next, grant the SELECT permissions on this view to the ANL and ADT roles, allowing all analysts and auditors to view the contents of this view.

#### **2.6.5.2 MKT\_campaigns**

##### **View 1: User (vw\_MKTCampaigns\_User)**

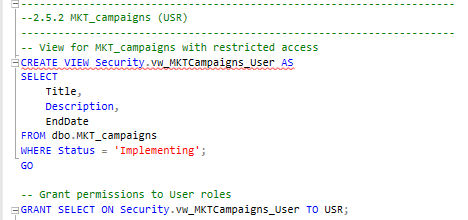


Figure 25: Create View vw\_MKTCampaigns\_User for User

This SQL code creates a view named vw\_MKTCampaigns\_User. This view will displays only the Title, Description, and EndDate columns from the MKT\_campaigns table. This view also using WHERE statement to filter only the campaigns with Status of ‘Implementing’. Next, the SELECT privilege on this view is granted to the USR role, allowing all end users to view the contents of this view. This indicat4es that the view will only displays the ongoing campaigns to end users, allowing them to know more about the marketing campaigns and encourage them to join. This implementation also ensures that internal organizational information, such as who created the marketing campaigns and when created, is not available to end users.

##### **View 2: Agent & User Portal Developer (vw\_MKTCampaigns\_Limited)**

A screenshot of a computer

AI-generated content may be incorrect.

Figure 26: Create View vw\_MKTCampaigns\_Limited for Agent & User Portal Developer

This SQL code creates a view named vw\_MKTCampaigns\_Limited. This view will displays only the CampaignID, Title, Description, StartDate, EndDate, Budget, Status, CreatedBy, and CreatedAt columns from the MKT\_campaigns table. This view also using WHERE statement to filter only the campaigns with Status of ‘Implementing’ and also masking all the sensitive information for these marketing campaigns.

As shown in the figure above, all StartDate and EndDate columns display as ‘XXXX-XX-XX’, the Budget column displays as the default value of 0.00, the Status column displays ‘REDACTED’, the CreatedBy column displays ‘Anonymous’, and the CreatedAt column displays ‘1900-01-01’. These masking operations prevent all agents and user portal developers from viewing information such as actual campaign dates and budgets, thereby preventing data leakage. Next, the SELECT permission for this view is granted to the AGT and UPD roles, allowing all the agents and user portal developers to view the contents of this view.

#### **2.6.5.3 Sales**

##### **View 1: Marketing & Analyst (vw\_Sales\_Restricted)**

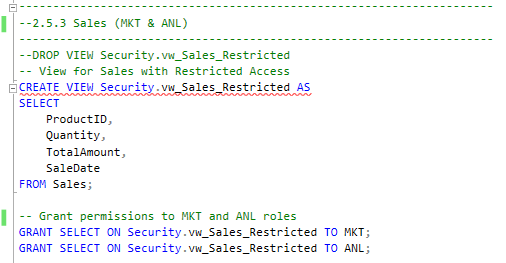


Figure 27: Create View vw\_Sales\_Restricted for Marketing & Analyst

This SQL code creates a view named vw\_Sales\_Restricted. This view will displays the ProductID, Quantity, TotalAmount, and SaleDate columns from the Sales table. Next, the SELECT permission for this view is granted to the MKT and ANL roles, allowing all marketing and analysts to view this sales information. This implementation ensures that marketing and analysts have sufficient sales information to complete their work, without providing excessive detail.

##### **View 2: Analyst (vw\_MonthlyProductSales)**

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 28: Create View vw\_MonthlyProductSales for Analyst

This SQL code creates a view named vw\_MonthlyProductSales for analyzing monthly sales data. This view joins the Sales table with the Products table using a JOIN statement to calculate the total number of sales, total number of products, and total amount for each product sold each month. Next, grant the SELECT privilege on this view to ANL role, ensuring that all analysts can view monthly sales summary data for each product.

#### **2.6.5.4 Products**

A screen shot of a computer

AI-generated content may be incorrect.

Figure 29: Create View vw\_Product\_User for User

This SQL code creates a view named vw\_Product\_User. This view will displays the Name, Description, and Price columns from the Products table. Next, grant the SELECT permission on this view to the USR role, allowing that all end users can view basic product information that is enough for them to understand the product and decide whether to purchase it. This implementation ensures that sensitive information within the organization, such as who entered the product information into the database will not show to the end users.

### **2.6.6 Check Granted Access**

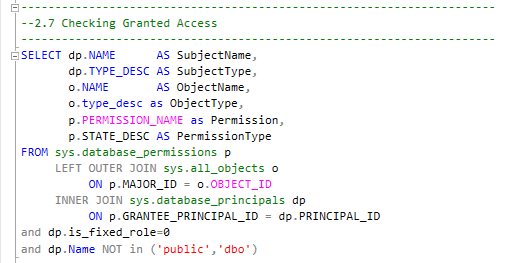


Figure 30: Check Granted Access

This code is used to query the detailed information of the permissions granted or denied for each role and user in this database. This implementation helps database administrators to check which roles and users are granted which permissions, thereby ensuring the permission allocation meet the security policies and the principle of least privilege.

# **3.0 Data Protection**

Data protection means putting in place rules, technology, and processes that keep data safe from the time it is created until it is sent, backed up, and finally thrown away. In the context of databases, it is meant to protect confidentiality, integrity, and availability (CIA) by stopping unauthorized access, finding tampering, and making sure that data can be reliably recovered after failures or security incidents. This means using many layers of security, like encryption, masking, access control, and strong backup plans, to protect against both outside cyber threats and misuse by employees (Aldossari & Sidorova, 2022). Data protection is very important because today’s businesses work in complicated environments with more rules and changing cyber threats. A single breach can cost a company money, hurt its brand, and lead to fines for not following the rules (Alam et al., 2023).

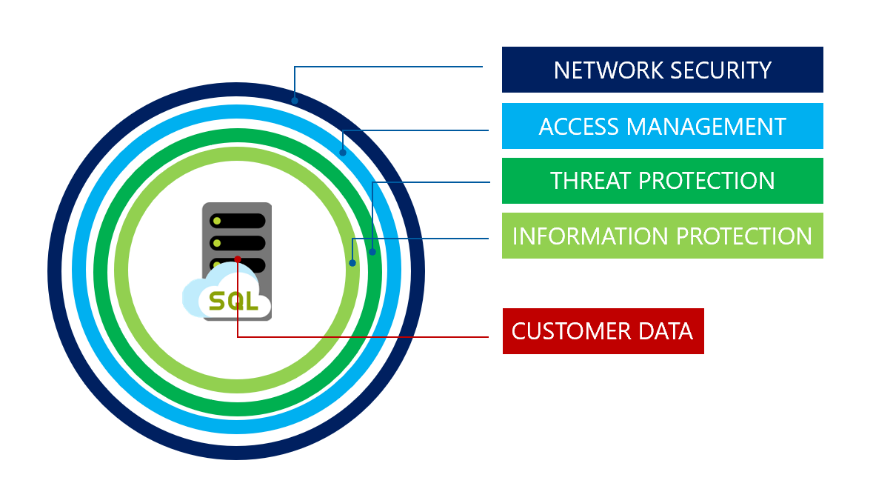


Figure 31: Data Protection Layers (Azure, 2025)

Data protection is important for more than just technical resilience, it also makes sure that you follow the rules set by the law and your business, like the General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act (HIPAA). As digital services evolve quickly, data breaches happen more often and in increasingly advanced ways. To keep businesses running and customers’ trust, businesses need to take steps to secure themselves (Subramani et al., 2021). This project will look at data protection in three main areas which are data obfuscation, which protects sensitive information in environments that aren’t used for production. Data encryption, which protects data while it is being sent and while it’s at rest using both symmetric and asymmetric methods and data backup, which makes sure that data can be recovered if it is lost, corrupted, or destroyed. These steps will be the basis for the ultimate implementation of our security plan. They will work with permission management and auditing systems to protect the entire database environment.

## 3.1 Data Obfuscation

Data obfuscation is the act of intentionally changing or hiding data so that unauthorized users cannot understand it, while nevertheless keeping its structure and function intact for testing or operational purposes. This method is very useful in places where genuine datasets may be needed but revealing sensitive information would put security and compliance at danger, like in development or quality assurance (Kumar & Kumar, 2022). Data categorization is the first stage in putting data obfuscation into action. It means putting data into groups based on how sensitive and important it is, including public, internal, confidential, or restricted. This classification tells you how much obfuscation or masking is needed. For example, personal identifiers, financial records, and authentication credentials are very sensitive fields that may need irreversible anonymization or tokenization. Less sensitive fields can be partially masked (Subramani et al., 2021).

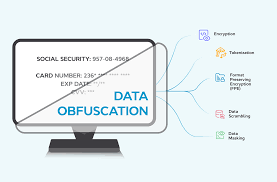


Figure 32: Data Obfuscation Methods (OvalEdge, 2024)

Permission management and effective data obfuscation go hand in hand. This makes sure that only authorized roles with a real business requirement can see obfuscated or masked data. This is in line with the idea of least privilege, which limits user access to only what they need to do their jobs. This lowers the chance of insider abuse and unauthorized data exposure (Alam et al., 2023). Permission management enforces either role-based or row-level access controls. This means that users with higher rights cannot get around masked values until they get permission. Organizations can create a tiered security model by adding data classification and permission management to the obfuscation process. This reduces data exposure, helps with compliance with privacy laws like GDPR, and keeps testing and analytics running smoothly.

## 3.2 Data Encryption

Data encryption is the process of changing plain text into an unreadable format (ciphertext) to keep it safe while it is being stored and sent. This makes sure that only people who are allowed to see the original information can do so if they have the right decryption key. Symmetric encryption employs the same key to encrypt and decrypt data. This makes it fast and efficient for large datasets, but the secret must be shared securely to keep it safe. As opposed to symmetric encryption, which uses a single key to encrypt and decode data, asymmetric encryption uses a pair of keys which is a public key to encrypt data and a private key to decrypt it. Asymmetric methods are better for managing keys in contexts that are not trusted, but they are slower to compute. This makes them great for protecting little but sensitive data or for encrypting symmetric keys in hybrid encryption schemes (Jain & Sharma, 2022). In database systems, both methods are commonly used together. Symmetric encryption for protecting large amounts of data and asymmetric encryption for secure key exchange. This makes sure that both performance and security are good (Choudhary et al., 2023).

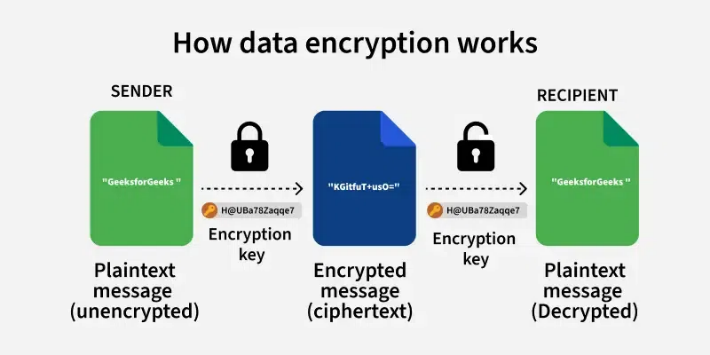


Figure 33: Data Encryption Workflow (GeeksforGeek, 2022)

Column-level encryption is a focused encryption solution that protects some critical fields, like credit card information or personal identifiers, instead of the whole database. This method follows the data minimization principle by only encrypting high-risk data elements and leaving less critical fields unprotected to speed up queries (Alotaibi et al., 2021). It is quite helpful in companies that must follow rules like GDPR or HIPAA, where only certain types of data need to be encrypted. But for it to work well, you need to have strong key management policies in place, such as key rotation, access logging, and separating roles to stop anyone from decrypting data without permission. Organizations can make a layered encryption approach that combines performance, granular protection, and compliance with rules by using symmetric, asymmetric, and column-level encryption. This is an important part of a complete data security architecture.

## 3.3 Data Backup

Backing up data is making duplicates of database data so that it can be restored in case of corruption, unintentional deletion, hardware failure, or security breaches like ransomware attacks. The major reason for backups is to keep the business running and satisfy recovery goals, such as the Recovery Point Objective (RPO) and Recovery Time Objective (RTO), by getting data back to a consistent and useful condition (Alam et al., 2023). A full backup takes a copy of the entire database, including all its data and schema. It is usually done on a regular basis, such every week. A differential backup only saves the changes that have been made since the last complete backup. This is a good mix between speed and efficiency while restoring. The transactional log backup keeps all transaction log records from the last log backup. This makes it possible to recover data at a specific moment in time for systems that need to be available all the time, which is why it is so important for mission-critical environments (Ahmed et al., 2022).

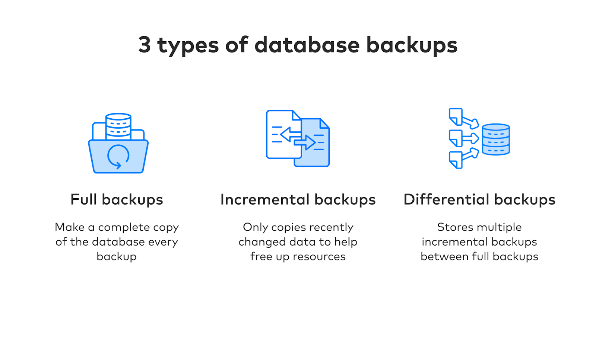


Figure 34: Types of Database Backup (FiveTran, 2023)

Other ways to back up your data have their own advantages. A copy-only backup makes a separate copy of the database without changing the existing backup sequence. This makes it perfect for one-time backups before upgrades or tests. Database mirroring is not a backup in the strictest sense, but it does allow real-time replication to a backup server, which means that the server is always available and there is little downtime if the primary server goes down (Chakraborty & Saha, 2021). The 3-2-1 backup rule is a common way for businesses to make sure their data is safe. It says to keep three copies of the data on two different types of media, with one copy kept offsite or in storage that cannot be changed. Always encrypt backups, check their integrity, and test them from time to time to make sure they can be restored. A strong backup plan may defend against both operational problems and big disasters by using several types of backups and making sure they meet the organization’s RPO and RTO needs. This will make sure that the database is reliable and meets industry standards.

## 3.4 Data Classification Matrix

The data classification matrix details the categories of each of the columns in the company-related tables for the entire database. The classifications range from the most exclusive which is private class to the most exposed public class. Given the fact that this project is solely meant to assist in facilitating the expansion and migration of the company’s internal system however, there are no columns with the public classification since none of the data should be available to unauthorized external parties.

The significance of the matrix helps in identifying crucial data types that explicitly needs to be protected by some form of data obfuscation techniques. As such, Table 3 details each table name and column for identification, along with each column’s classification and level.

For the scope of this project, the user’s passwords in the User table are classified as private which is the most sensitive of all. As the exact password should only be privy to the owner, hashing is applied for this data type since it is not expected to be able to be recovered at any point throughout the system.

Level 2, labelled as confidential, is designated to the other columns that are more accessible but only to a specific group or department within the internal context. These columns include the agent’s email, phone number, address, and identification numbers for agents and users. From these, the agent’s email, phone number, and address are hidden with the MASKED function in SSMS from everyone other than the agents themselves. This will allow the agent’s information to remain confidential within the system and protect their privacy. Furthermore, these masked data are further handled with RLS as detailed in the earlier section. It is not encrypted because active data such as these may still be required by the company for various purposes. Masking provides a secure way to hide the person’s details from others, while still allowing it to be used by the company for production or testing environments (Goel, 2020). Moreover, masking is chosen for active data due to it being faster than encryption (Sims, 2022). This will reduce the strain on the overall system and prevent a cumbersome user experience altogether.

Meanwhile, encryption is applied to the identification numbers of the agents and users. This is based on the assumption that the identification numbers are only meant to serve as personal identifiers for the people, and are considered as data at rest, encrypted only for storage and transit purposes.

Table 3: Data Classification Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| **Table Name** | **Column** | **Classification** | **Level** |
| **Users** | UserID | Internal | 3 |
| IdentificationNo | Confidential | 2 |
| Username | Internal | 3 |
| Password | Private | 0 |
| Role | Internal | 3 |
| CreatedAt | Internal | 3 |
| **Agents** | AgentID | Internal | 3 |
| IdentificationNo | Confidential | 2 |
| Name | Internal | 3 |
| Email | Confidential | 2 |
| Phone | Confidential | 2 |
| Address | Confidential | 2 |
| Status | Internal | 3 |
| CreatedAt | Internal | 3 |
| **Products** | ProductID | Internal | 3 |
| Name | Internal | 3 |
| Description | Internal | 3 |
| Price | Internal | 3 |
| CreatedAt | Internal | 3 |
| **Sales** | SaleID | Internal | 3 |
| AgentID | Internal | 3 |
| ProductID | Internal | 3 |
| Quantity | Internal | 3 |
| TotalAmount | Internal | 3 |
| SaleDate | Internal | 3 |
| **Commission** | CommissionID | Internal | 3 |
| AgentID | Internal | 3 |
| SaleID | Internal | 3 |
| CommissionRate | Internal | 3 |
| CommissionAmount | Internal | 3 |
| CreatedAt | Internal | 3 |
| **Notifications** | NotificationID | Internal | 3 |
| Title | Internal | 3 |
| Message | Internal | 3 |
| Target | Internal | 3 |
| CreatedAt | Internal | 3 |
| CreatedBy | Internal | 3 |
| **MKT\_campaigns** | CampaignID | Internal | 3 |
| Title | Internal | 3 |
| Description | Internal | 3 |
| StartDate | Internal | 3 |
| EndDate | Internal | 3 |
| Budget | Internal | 3 |
| Status | Internal | 3 |
| CreatedBy | Internal | 3 |

## 3.5 Final Implementation Solution

### **3.5.1 Database TDE**

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 35: Database TDE SQL Code

The SQL script given here uses Transparent Data Encryption (TDE) in SQL Server. This technology protects data at rest by encrypting the database and its transaction logs. To start, use DROP MASTER KEY to get rid of any existing master key. This makes sure that the setup is clean. Then, CREATE MASTER KEY ENCRYPTION BY PASSWORD makes a new Database Master Key (DMK), which is the main encryption key for the database. After this, a certificate called HPO\_TDE is made with CREATE CERTIFICATE. This certificate acts as a middleman to safeguard the Database Encryption Key (DEK). Then, using CREATE DATABASE ENCRYPTION KEY, the DEK is made using the AES-256 encryption technique, which is a powerful encryption standard that is widely used. The certificate encrypts the DEK. The last step is to run the ALTER DATABASE DAMS SET ENCRYPTION ON command, which turns on TDE for the DAMS database. This makes sure that all database and transaction log files are encrypted on disc.

This implementation immediately addresses the confidentiality part of the CIA Triad by stopping people from accessing database files without permission when they are not connected to the internet. Even if someone steals or breaks into storage media, the encrypted files can’t be read without the right keys and certificates. TDE also helps us follow data protection laws like HIPAA and GDPR since it protects sensitive data without requiring any changes to the application level. This setup uses a layered encryption hierarchy from Master Key to Certificate till Database Encryption Key to handle keys well and follow the best practices for database security.

### **3.5.2 Masking**

A group of blue and red text

AI-generated content may be incorrect.

Figure 36: Masking Implementation Code

The SQL script uses Dynamic Data Masking (DDM) in SQL Server to keep unauthorized users from seeing sensitive values in the Agents table. DDM changes the results of a query so that sensitive data looks disguised for users who do not have the UNMASK privilege. This does not change the real data in the database. The partial() function is used in this script to hide the Phone column, showing just a masked placeholder in the format XXX-XXX except for the first 3 digits. The built-in email() masking method is used in the Email column. This masks part of the address while keeping the format of the address. The Address column is also covered up with a partial mask that substitutes the actual data with [REDACTED]. These changes make sure that data may still be used for operational queries while reducing the risk of exposure to people who should not have access to it.

The last command, GRANT UNMASK TO AGT, gives the AGT role the right to see unmasked data. This makes sure that only users who have been given permission can see full, sensitive information. This is in line with the principle of least privilege, which says that only certain roles, like administrators or compliance officials, should be able to see unmasked information. Dynamic masking adds an extra layer of protection to permission management by stopping sensitive data from being accidentally revealed in front-end apps, reports, or ad-hoc queries. The implementation facilitates compliance with privacy standards like GDPR and HIPAA by combining DDM with role-based access control. This keeps personally identifiable information (PII) safe while still letting non-privileged users do their jobs without revealing raw sensitive values.

### **3.5.3 Column Level Encryption**

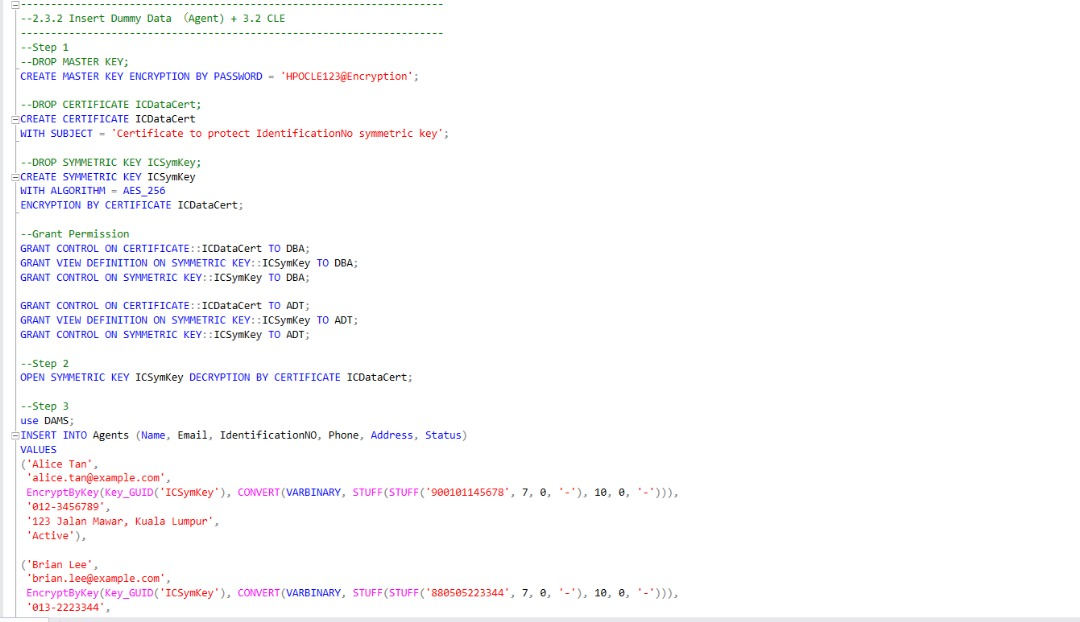


Figure 37: Column Level Encryption Implementation

This approach protects critical fields in the Agents table, notably the IdentificationNo column, by encrypting them at the column level. Column-level encryption, on the other hand, only encrypts certain data fields. This means that even database administrators cannot see unencrypted values without the right rights. The first step is to make a Database Master Key (DMK), and then a certificate (ICDataCert) to protect a symmetric key (ICSymKey) using the AES-256 encryption algorithm. The principle of least privilege is followed by giving specific roles (DBA and ADT) permission to control and inspect the symmetric key.

When the symmetric key is opened using DECRYPTION BY CERTIFICATE, dummy agent data is added, and the IdentificationNo values are encrypted with EncryptByKey. Before encryption, the STUFF function formats these values such that they all have the same disguised structure. This method makes sure that even if unauthorized users get access to the table, important identifiers stay unreadable without the right decryption key. This encryption layer, together with TDE and dynamic data masking, gives the database a multi-layered security approach that makes it easier to follow privacy laws like GDPR and Malaysia’s PDPA while still being easy to use for business.

### **3.5.4 Backup Process**

Copy Only Backup

A screen shot of a computer

AI-generated content may be incorrect.

Figure 38: Copy Only Backup Implementation

The SQL script makes a Copy-Only Backup of the DAMS database. This is a particular kind of backup in SQL Server that does not get in the way of the standard backup process. This is especially helpful when you need to do ad-hoc backups for certain upgrades, migrations, or compliance needs without messing with planned differential or transaction log backups. The script tells the backup where to go, utilizes the COPY\_ONLY option to keep the backup separate from the main backup chain, and includes INIT to overwrite the file if it is already there. CHECKSUM checks the backup for corruption during the process to make sure data integrity. COMPRESSION is used to save space.

The backup has a NAME and a MEDIANAME on it so that it may be easily found when it needs to be restored. This method helps with disaster recovery and keeping the business running by giving you a verified, recoverable copy of the database that does not depend on regular backup schedules. It also follows best practices for managing database backups, making sure that special-purpose backups do not get in the way of the organization’s recovery point objectives (RPO) and recovery time objectives (RTO). By adding copy-only backups to the backup strategy, administrators can quickly respond to urgent backup requests, like before a major patch or structural change, without compromising the main backup plan.

System Backups

A screenshot of a computer error

AI-generated content may be incorrect.

Figure 39: System Backups Implementation

The script does system database backups for SQL Server’s master, msdb, and model databases. These backups are important for restoring the SQL Server instance’s setup and metadata if something goes wrong or a disaster happens. The master database holds important system-level information like login accounts, system settings, and the locations of all the other databases. The msdb database keeps track of SQL Server Agent jobs, backup history, and alert settings. This makes it very important for recovering scheduled automation. The model database is used as a template for all new databases that are made on the server. This means that having a backup that is up to date makes sure that future databases are always set up the same way.  
  
The TO DISC command tells the backup where to save the files, the INIT option lets you replace any existing backup file, and the COMPRESSION option makes the files smaller. The CHECKSUM option is included to check the backup’s integrity while it is being made, which lowers the chance of restoring files that are broken. The backups are easy to find for recovery operations because they have names like Master DB Backup, MSDB Backup, and Model DB Backup. These backups of the system are usually done after making big changes to the settings, installing important upgrades to the software, or before doing maintenance work that could be dangerous. Administrators may restore not only user data but also the operating environment of SQL Server by keeping up-to-date backups of these system databases. This means less downtime and faster disaster recovery.

SQL Agent System Backup

A computer screen shot

Description automatically generated

Figure 40: Full System Backup (1)

A screenshot of a computer

Description automatically generated

Figure 41: Full System Backup (2)

The first part of the script sets up a Full Backup procedure for the DAMS database using SQL Server Agent automation. A task called DAMS\_Full\_T is set up with a separate step to make a full backup of the database. The result is saved in both a primary path (D:\APU Assignments\Y3S2\DBS\Backups\DAMS\_Full.bak) and a mirrored secondary path (C:\Backups\DAMS\_Full\_Mirror.bak) for safety. The backup operation is improved with choices like FORMAT, INIT, COMPRESSION, and CHECKSUM to make sure the data is safe, consume less storage space, and get the backup media ready for use again. The job is set to execute every day at 2:00 AM from August 4 to August 8, 2025. This will make sure that the full database backups are always the same and may be used as the starting point for recovery operations.

A screenshot of a computer

Description automatically generated

Figure 42: Differential Backup (1)

A screenshot of a computer

Description automatically generated

Figure 43: Differential Backup (2)

The second part of the script sets up a Differential Backup process, which is called DAMS\_Differential\_T in the task name. This backup method only saves the changes to the data that have happened since the last complete backup. This makes it faster and uses less storage space than conducting full backups all the time. The output file (DAMS\_Diff.bak) is saved in the backup directory that was set up for it, with compression and checksum checking turned on to keep storage space and data safe. The differential backup job is set to run every four hours, every day. This makes sure that recent changes to transactions are saved often, but the most recent full backup is still used for full restoration.

A screenshot of a computer

Description automatically generated

Figure 44: Transactional Backup (1)

A screenshot of a computer

Description automatically generated

Figure 45: Transactional Backup (2)

A screenshot of a computer

Description automatically generated

Figure 46: Transactional Backup (3)

The third part uses a job called DAMS\_Transactional\_Log to set up a Transactional Log Backup plan. This procedure keeps track of the order of all transactions in the database, which makes it possible to recover data exactly as it was at a certain moment in time if something goes wrong or data becomes corrupted. With the FORMAT, INIT, COMPRESSION, and CHECKSUM options, the transactional log backup file (DAMS\_Log.trn) is made to make sure it is accurate and does not take up too much space. There are two schedules: one runs every 15 minutes from 12:00 AM to 1:59 AM and the other runs from 2:15 AM until 11:59 PM. This high-frequency logging makes sure that there is very little chance of data loss, which is important for mission-critical recovery situations where restoration needs to happen right away.

A screenshot of a computer

Description automatically generated

Figure 47: Weekly System Backup (1)

A screenshot of a computer program

Description automatically generated

Figure 48: Weekly System Backup (2)

A screenshot of a computer program

Description automatically generated

Figure 49: Weekly System Backup (3)

The last part sets up a Weekly System Database Backup process to protect SQL Server’s system databases (master, model, and msdb). There is a job called Weekly\_System\_DB\_Backup that has three phases, each of which backs up one of the system databases. To make sure that the backups are safe and can be checked, the output is saved in the System Backups folder with FORMAT, INIT, COMPRESSION, and CHECKSUM turned on. This job runs every Sunday at 3:00 AM and keeps important system-level metadata, configurations, and job scheduling information. This makes sure that the database server’s operational environment can be restored if there is a hardware failure, a system misconfiguration, or a catastrophic loss.

# **4.0 Auditing**

## 4.1 Concept of Database Auditing

Database auditing is the planned way of keeping track of and looking at database activities to make sure they are safe, follow the rules, and work properly. It entails monitoring user behaviors, access trends, and modifications to data or database frameworks to identify unauthorized acts, examine incidents, and uphold responsibility (Jain & Singh, 2022). Auditing upholds the concepts of Confidentiality, Integrity, and Availability (CIA) by guaranteeing that sensitive information is accessible solely to authorized personnel, modifications are executed in a regulated and verifiable manner, and systems maintain uninterrupted availability. Auditing checks that roles, privileges, and access controls are being enforced correctly in the context of permission management. It also finds cases where permissions may have been misused or given out incorrectly.

Database auditing includes keeping an eye on changes to data, structure, and operational procedures both before and after change-control actions. This involves keeping track of Data Manipulation Language statements like INSERT, UPDATE, and DELETE, as well as Data Definition Language actions that change the structure of schemas. Auditing is also very important for showing that you are following the rules set by the law and the industry, such GDPR, HIPAA, and ISO 27001. This is done by keeping logs of access and changes that cannot be changed (Alam et al., 2023). These logs can be used as evidence in court if there are disputes or investigations. In the end, database auditing is not just a way to keep data safe; it is also a way to regulate it by making things clear and building trust amongst system administrators, regulatory authorities, and stakeholders.

## 4.2 Auditing Sources

Auditing sources are the reference points and structured documents that serve as the starting point for keeping an eye on and checking database activity. The data dictionary is a central place where you may find metadata about the database structure, such as tables, columns, indexes, constraints, and relationships. It is important to know how data is organized and to check that modifications fit with the approved schema design (Kaur & Singh, 2021). The authorization matrix shows which users or roles can access certain database objects. This lets auditors check that rights follow security rules. This matrix is very important for finding too many privileges that could lead to insider threats or data breaches.



Figure 50: Server Auditing Ideal Practices (IDNS, 2020)

The audit matrix connects security and compliance needs to the database actions that need to be watched, making sure that no important tasks are missed. The data categorization matrix sorts of data into groups based on how sensitive it is, such as public, internal, confidential, or restricted. This let’s audit policies focus on the riskiest data items. Finally, SQL objects like stored procedures, triggers, and functions are a significant source for auditing since modifications to these objects can have a direct effect on system logic, data integrity, and security. By bringing these sources together, organizations may keep a complete and trustworthy audit framework that covers all important parts of database security and compliance (Saxena & Rani, 2022).

## 4.3 Auditing Techniques

Database systems employ many methods to record and keep track of changes to data, user actions, and system events. A feature of SQL Server called temporal tables keeps a full history of modifications to data, so auditors can look at any table’s past states at any time. This is especially helpful for forensic analysis, compliance reporting, and keeping track of small changes to data without having to write specialized logging code (Soni & Sharma, 2022). Trigger-based auditing means making database triggers, which are special stored procedures that run automatically when certain events happen, like INSERT, UPDATE, or DELETE. These triggers can record detailed before-and-after values of data changes, save them in audit tables, and link them to the user and time of the operation.

Enterprise auditing features are database capabilities that are built-in or can be added on to work in big, compliance-driven organizations. SQL Server has capabilities like SQL Server Audit that let you choose exactly which actions are logged. This makes logs that cannot be changed and can be kept in safe places for a long time. These technologies generally work with centralized monitoring systems to automatically provide alerts and reports when they see something suspect (Alam et al., 2023). Organizations can create a whole auditing strategy that meets both operational and regulatory needs by using temporal tables to maintain history, triggers to collect detailed events, and enterprise-grade auditing tools to report on compliance.

## 4.4 Database Security Audit Matrix

Table 4 is the database security audit matrix, responsible for identifying the necessary SQL objects and its associated actions that need to be audited. These objects include tables, database, and server level instances that all have their own significance and priority which necessitate different forms and levels of auditing.

Based on the requirements and scope of actions, this project employs a plethora of mechanisms to be able to capture these changes to ensure data integrity, compliance, and security. If present, the table also states the location and format of the audit log created from the mechanism to enable convenient access as well as expedite auditing processes for the relevant parties.

Table 4: Database Security Audit Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SQL Object Name | SQL Object Type | What to Capture | Mechanism | Audit Log Location |
| Agents | Table | **Insert, Update, Delete** actions | SQL AFTER Trigger (Agents\_AuditTrigger) | SQL Table  (Agents\_AuditLog) |
| Sales | Table | **Insert, Update, Delete** actions | SQL AFTER Trigger (Sales\_AuditTrigger) | SQL Table  (Sales\_AuditLog) |
| Commission | Table | **Insert, Update, Delete** actions | SQL AFTER Trigger (Commission\_AuditTrigger) | SQL Table (Commission\_AuditLog) |
| DAMS | Database | DCL changes (permissions, role changes) | SQL Server Audit Feature | SQLAUDIT File:  TrackDCLChanges\_E4D5C953-A140-4BA6-BF15-8DBE3CDB3B82\_0\_133988596402740000.sqlaudit |
| DAMS | Database | DDL changes (create, alter, drop objects) | SQL Server Audit Feature | SQLAUDIT File:  TrackDDLChanges\_F6DF9E1E-1720-4240-ABF3-25F72242EEA0\_0\_133988596315280000.sqlaudit |
| DAMS | Database | DML changes (insert, update, delete, select) | SQL Server Audit Feature | SQLAUDIT File:  TrackDMLChanges\_2318B8AA-4708-4C57-BCBF-465AB5FC17A3\_0\_133988597121460000.sqlaudit |
| SQL Server Instance | Server | Login and Logout activities | SQL Server Audit Feature | SQLAUDIT File:  TrackLogin\_C29E7ECC-3B04-4B61-ADA1-C493AA4B7471\_0\_133988597402330000.sqlaudit |
| Users, Agents, Products | Table | Historical versions of rows | System-Versioned Temporal Tables | History database tables:  UsersHistory, AgentsHistory, and ProductsHistory |
| All Server Logons | Server | Limit concurrent sessions for a login | Logon Trigger (Logon\_Limit\_Sessions) | No audit log generated |
| All Server Logons | Server | Restrict login times by hour | Logon Trigger (Logon\_Limit\_Hours) | No audit log generated |

## 4.5 Final Implementation Solution

### **4.5.1 Agents Audit Log**

A screenshot of a computer screen

AI-generated content may be incorrect.

Figure 51: Agents Audit Log Implementation (1)

The SQL script given makes a trigger called Agents\_AuditTrigger on the Agents table that logs all INSERT, UPDATE, and DELETE actions into the Agents\_AuditLog table. The trigger saves the new agent’s information, like their ID number, name, email, phone number, address, status, and timestamps, as well as the SYSTEM\_USER who did the action. The WHERE NOT EXISTS condition makes sure that an INSERT log is only made if the record did not come from a deletion event. This stops duplicate entries. For DELETE actions, equivalent data fields are taken from the deleted pseudo-table, and information is only logged if the record is not in the inserted set. This method makes sure that there is a clear audit trail for when records are added or deleted. This is important for checking the integrity of the data and for security investigations.

A computer screen shot of a program

AI-generated content may be incorrect.

Figure 52: Agents Audit Log Implementation (2)

When you conduct an UPDATE action, the trigger compares the data in the inserted and deleted pseudo-tables. It checks for changes in important fields such identification number, name, email, phone, address, status, and creation date using ISNULL(). It only logs an entry if it finds a change. This selective logging stops audit records from being made for fields that have not changed, which saves space and makes the audit log easier to read. The action type, the data that was changed, the date and time of the modification GETDATE(), and the user who made the change (SYSTEM\_USER) are all included in each log entry. By using this auditing approach, the system makes sure that any changes to the Agents table can be tracked. This meets the standards for accountability, integrity, and compliance with data protection laws.

### **4.5.2 Sales Audit Log**

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 53: Sales Audit Log Implementation

The Sales\_AuditTrigger keeps watch of all INSERT, UPDATE, and DELETE activities on the Sales table, making sure that there is a full record of all sales transactions. The trigger saves the sale ID, agent ID, product ID, quantity sold, total amount, sale date, action type (INSERT), timestamp (GETDATE()), and user (SYSTEM\_USER) who did the action for INSERT operations. This makes sure that every new sales record uploaded to the system is recorded so that people can be held accountable. In the same way, for DELETE operations, the trigger keeps track of the same values from the deleted pseudo-table. This keeps track of sales records that were deleted from the system in case there are any disputes or compliance checks.  
  
The UPDATE part of the trigger tracks changes only when important fields like quantity, total amount, or sale date have been changed. Using conditional checks in the WHERE clause, this is done by comparing data from the pseudo-tables that were added and removed. The trigger will only add an UPDATE action to the Sales\_AuditLog table if it finds differences. It will also provide the information and the user who did the action. This selective logging method cuts down on needless log entries while still making sure that important changes are recorded. This audit trigger makes sales activities more open, helps keep data safe under the CIA Triad, and gives proof of following the rules and keeping an eye on operations.

### **4.5.3 Commission Audit Log**

A screenshot of a computer

AI-generated content may be incorrect.

Figure 54: Commission Audit Log Implementation

The Commission\_AuditTrigger keeps an eye on all INSERT, UPDATE, and DELETE actions on the Commission table. This makes sure that commission-related transactions are clear and can be traced back. The trigger keeps track of important information for INSERT actions, like the commission ID, agent ID, sale ID, commission rate, commission amount, creation date, the type of action (INSERT), the timestamp (GETDATE()), and the person who did the operation (SYSTEM\_USER). This makes sure that all new commission records are kept track of for compliance and auditing reasons. The same fields are taken from the deleted pseudo-table for DELETE operations to keep track of erased commission items. This is important proof in case of disputes or financial investigation.

The UPDATE part of the trigger is all about keeping track of changes to commission rates, quantities, or creation dates. The trigger checks the inserted and deleted pseudo-tables against each other to make sure that only important changes are documented. This keeps the audit log from getting too full. This selective logging method makes storage more efficient and makes it easier to spot important changes in commission data. The recorded information, such as the type of activity, the data values, the time of the modification, and the user who did it, makes a strong audit trail that improves integrity and accountability under the CIA Triad. This logging system also meets the norms for financial record-keeping set by the industry and corporate governance, making sure that commission transactions are still clear and can be verified.

### **4.5.4 Tracking DCL Change**

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 55: DCL Changes Implementation

The TrackDCLChanges audit configuration is meant to keep a watch on Data Control Language (DCL) activities, which are changes to access control in the SQL Server environment, such as giving and revoking permissions, assigning roles, and other changes. The first thing the script does is create a server-level audit (CREATE SERVER AUDIT) and set up a file directory to hold the audit logs. This makes sure that all access-control events are recorded in one location and cannot be modified. Then, you can turn on the audit by using ALTER SERVER AUDIT (STATE = ON), which starts logging significant events in real time. The setting keeps the audit trail safe by delivering the logs to a safe file location. This is useful for both operational evaluations and forensic investigations.

The CREATE SERVER AUDIT SPECIFICATION connects the audit to certain change groups. These groups cover actions like adding or removing users, modifying who is in a role, changing schema permissions, and changing passwords. It’s crucial to keep a watch on these changes to uphold the principle of least privilege and discover unauthorized privilege escalations. Administrators can make sure that security settings stay in line with policy, discover problems quickly, and follow requirements like ISO 27001 and GDPR by keeping a watch on DCL activities. This audit at the server level shows all the modifications to access control. This minimizes the risk of insider attacks and makes database governance better overall.

### **4.5.5 Tracking DDL Change**

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 56: DDL Changes Implementation

The TrackDDLChanges setting makes a server-level audit that writes to a file target and then turns on with STATE = ON. The associated SERVER AUDIT SPECIFICATION signs up the audit for several event groups. These together record schema-changing operations such creating, changing, or deleting tables, views, procedures, indexes, transferring ownership, changing permissions on objects, and changing server objects. When you save these events to an external file, you make a tamper-evident trail that helps with forensic reviews and keeps operational logs separate from transactional data.

Monitoring DDL is essential for maintaining integrity and implementing change control. Changes to the schema that are not authorized or verified can damage applications, modify data assumptions, or subtly degrade security (for example, by removing a foreign key or giving access to an object-level grant). Auditing DDL gives you proof of who performed what and when, helps with release retrospectives, and helps meet ISO/IEC 27001 and other governance standards for documented, reviewable modifications. This server audit, along with your DCL and DML auditing, closes the loop: it keeps an eye on permissions, records schema changes, and makes data changes easy to find, giving you full-stack accountability.

### **4.5.6 Tracking DML Change**

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 57: Track DML Implementation

The TrackDMLChanges setting sets up a server-level audit that publishes logs to a safe file directory. After that, it turns it on with STATE = ON. Then, a database-specific audit definition (DAMS\_DMLChanges) is produced for the DAMS database and linked to the server audit. This standard subscribes to pertinent event groups and mandates explicit oversight of INSERT, UPDATE, DELETE, and SELECT actions executed on the database by the public role. The audit makes sure that all read and write actions are recorded by recording these Data Manipulation Language (DML) activities. This creates a complete trail of all data access and modification events.

DML auditing is necessary to keep transactional data safe and honest. It helps administrators find unauthorized changes, keep an eye on who has access to critical documents, and help with forensic investigation in case of a breach. Adding SELECT auditing, especially for the public role, can assist find any data leaks through read-only queries, which is a mistake that many systems make. Logging DML modifications also helps meet compliance requirements for standards like ISO 27001 and rules like GDPR, which require verification of who accessed or changed personal or sensitive data. This setup, along with DDL and DCL audits, makes up a full-spectrum monitoring strategy that lets you see changes in permissions, schema evolution, and daily data transactions.

### **4.5.7 Tracking Login Activity**

A screenshot of a computer

AI-generated content may be incorrect.

Figure 58: Track Login Activity Implementation

The TrackLogin setting sets up an audit at the server level to record all login-related events in SQL Server, with the audit data stored in a secure file path. To turn on the audit once it has been made, use ALTER SERVER AUDIT WITH (STATE = ON). There is a SERVER AUDIT SPECIFICATION called TrackLoginAttempt that lets you subscribe to event groups. This combination lets administrators keep an eye on all login attempts, whether they were successful or not, as well as password changes and logout occurrences. The system makes sure that login patterns may be looked at for unusual behavior, possible brute-force attempts, or unauthorized access by keeping an unchangeable log of these actions.

Keeping track of login activity directly supports the CIA Triad’s pillars of privacy and availability. Failed login attempts can mean that someone is trying to break in, which means that the incident response team needs to act quickly. Successful logins and password changes, on the other hand, help keep a record of who has access that can be checked. Keeping an eye on logouts is just as critical as keeping an eye on logins to find unusual session terminations that could be tied to network or account compromise. This method is in line with security standards like ISO 27001, which say that businesses must keep track of authentication events for forensic analysis and to follow the rules. When used with DML, DDL, and DCL auditing, login tracking gives a complete picture of how users behave in the database environment, which lets you find security problems before they happen.

### **4.5.8 Audit Logs Viewing**

A screenshot of a computer

AI-generated content may be incorrect.

Figure 59: Audit Log Viewing Implementation (1)

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 60: Audit Log Viewing Implementation (2)

A computer screen shot of a program

AI-generated content may be incorrect.

Figure 61: Audit Log Viewing Implementation (3)

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 62: Audit Log Viewing Implementation (4)

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 63: Audit Log Viewing Implementation (5)

The Audit Logs Viewing implementation uses SQL Server views to give people structured, easy-to-read access to raw audit files that were made by earlier DCL, DDL, DML, and logon audits. A CREATE VIEW statement is set up for each category to use sys.fn\_get\_audit\_file() to get the right .sqlaudit files from the secure audit directory. The views vw\_Audit\_Login\_Activity, vw\_Audit\_DCL\_Changes, vw\_Audit\_DDL\_Changes, and vw\_Audit\_DML\_Changes change raw event data into a format that is easy to understand. The scripts use DATEADD to change timestamps to match the local time zone, rename columns to make them clearer, and use CASE statements to turn action codes into more descriptive names. Filtering by action\_id makes ensuring that each view only shows events that are relevant, like permission modifications for DCL, creating or changing objects for DDL, and data manipulation activities for DML.

Then, for each view, the ADT role is given permission (GRANT SELECT) to query audit data without having direct access to the raw audit files. This helps keep the data private and stops anybody from changing it. Along with the views, sys.fn\_get\_audit\_file() lets you run direct SELECT queries on the raw data, which makes forensic investigation more versatile when necessary. This method makes sure that audit logs are safe and easy to get to for monitoring, compliance checks, and incident investigation by using secure storage, filtered views, and role-based access control. This solution meets the standards of ISO 27001 for safe log management and the GDPR’s rules for limiting access to sensitive operational data.

### **4.5.9 Logon Triggers**

A white rectangle with green and yellow lines

AI-generated content may be incorrect.

Figure 64: Logon Triggers Implementation

The script starts by utilizing the CREATE SERVER ROLE statement to construct a server-level role called DBA. This job will work with logon triggers to limit access and keep an eye on logins to the database for certain privileged accounts. When a user logs on to SQL Server, a specific stored procedure called a logon trigger runs automatically. This lets administrators manage how sessions are set up based on things like the user’s role, the time of access, the client’s IP address, or other contextual attributes. By linking the DBA role to a logon trigger, the system can make sure that only authorized administrative users can start privileged sessions. It can also log all these logins for auditing purposes if needed.

This method adds an extra layer of permission authorization on top of normal role-based access control by letting security regulations be enforced in real time when users log in. For instance, the trigger may stop anyone from getting to the DBA position outside of maintenance times or from networks that are not secure. This would lower the chances of credential theft or insider attacks. Also, keeping track of these occurrences helps meet ISO 27001 criteria for keeping track of administrative access and finding unusual use of privileged accounts. Logon triggers improve both preventative and detective controls over high-risk accounts when used with the login auditing setup from 4.5.9. This makes the database environment safer overall.

### **4.5.10 Number of Sessions Limitation on Specific Login**

A screenshot of a computer

AI-generated content may be incorrect.

Figure 65: Login Session Limitation Implementation

At the server level, the Logon\_Limit\_Sessions trigger is used to limit the amount of sessions that a single login can have at the same time. The trigger runs with the permissions of the sa account and happens every time someone logs in. It gets the current login name with ORIGINAL\_LOGIN() and only imposes the restriction if the login is not on the list of exempted logins (like admin1, admin2, sa, and some service accounts). The core logic checks the sys.dm\_exec\_sessions dynamic management view to see how many active user processes there are for that login. If the number is higher than a certain level, in this case, more than one active session, the login attempt is blocked with a PRINT message and a ROLLBACK action.

This setup is a security measure that stops people from abusing shared or generic accounts by restricting the number of connections at once. This lowers the chance of unauthorized parallel activity. These kinds of limits are helpful for making sure people follow the rules, keeping track of how much server space is being used, and stopping brute-force or automated attacks. Administrators can keep an eye on blocked attempts and investigate strange behavior by using the trigger along with auditing. This is in line with ISO 27001’s recommended practices for access control, which say that both technological limits and monitoring should be used to protect privileged and high-risk accounts.

### **4.5.11 Hours of SQL Server Access Limitation**

A screenshot of a computer

AI-generated content may be incorrect.

Figure 66: Hour Usage Limitations Implementation (1)

The Logon\_Limit\_Hours trigger is a server-level control that limits database logins to a certain time frame, in this case, from 9:00 AM to 6:00 PM. It runs with higher permissions (EXECUTE AS ‘sa’) so that it can implement limitations on all relevant logins. The trigger stops logins that happen outside of the allowed time frame by sending a PRINT message and rolling back the connection attempt. It does this by using ORIGINAL\_LOGIN() to get the real login name and CAST(GETDATE() AS TIME) to check the current time. Some administrative and service accounts are not subject to the restriction. This kind of time-based access control can help keep people from getting in after hours, which lowers the danger of insider misuse or external compromise when monitoring staff may not be accessible.

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 67: Hour Usage Limitations Implementation (2)

There are other diagnostic enquiries that help manage and keep an eye on active sessions. These include getting trigger definitions, verifying the number of concurrent sessions per login, and using the KILL command to forcefully end sessions. These scripts let administrators impose access rules ahead of time and rapidly get rid of connections that are not authorized or are not being used. This method is in line with ISO 27001 standards for access management, which say that both limiting and monitoring privileged access is a good idea. The system makes database access more secure and ready for compliance by combining this restriction with the login tracking techniques from 4.5.9 and the concurrent session controls from 4.5.12.

### **4.5.12 Temporal Tables**

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 68: Temporal Table Implementation

Temporal tables in SQL Server are a way to automatically keep track of changes to data over time. This lets users query both current and former states without having to do human audits. In this version, the Users, Agents, and Products tables are changed to add ValidFrom and ValidTo datetime columns. These columns show when each row is valid. The start date is set to GETDATE(), while the end date is set to a timestamp far in the future. When you turn on system versioning with the HISTORY\_TABLE option, SQL Server automatically saves old row versions in separate history tables called UsersHistory, AgentsHistory, and ProductsHistory. This makes sure that all changes and deletions are recorded. This makes it possible to do point-in-time analysis, compliance reporting, and forensic investigation without having to log in manually.

But not all the datasets in this project had historical tables. This is because temporal tables can make storage needs much bigger and cause version history to grow exponentially if they are not managed appropriately. The design strikes a balance between auditing capability and storage efficiency by only using temporal tracking on important entities like Users, Agents, and Products. When used with the auditing methodologies in Sections 4.3 and 4.5, this focused usage of temporal tables makes the entire data accountability framework better by making sure that important changes to data can be traced while keeping system overhead to a minimum.

# **5.0 Summary**

This project has given us a complete and well-organized study of database security, starting with the basics, and moving on to more advanced procedures for implementing and auditing. Chapter 1 set the stage for the project by talking about its background, goals, and scope. It also talked about how important it is to protect data in a time when cyber risks, regulatory requirements, and digital transformation are all on the rise. It stressed the importance of having a well-organized database security framework that meets both the needs of the business and the law. Chapter 2 came next and included a thorough overview of the literature, bringing together the latest research, industry standards, and best practices. It looked at current security problems, current ways to protect against them, and new ways to audit, which set the stage for the actual remedies that would be discussed in coming chapters.

Chapter 3 turned these ideas into real-world data protection plans, including data obfuscation, encryption, masking, and strong backup methods. Practical applications like Transparent Data Encryption (TDE) to protect data at rest, Dynamic Data Masking (DDM) to limit access to sensitive fields, and different backup methods, such as copy-only and system database backups, showed how layered defenses can meet regulatory requirements like GDPR and HIPAA while still being able to keep the business running. Chapter 4 included database auditing to the security model. It talked about goals, key audit sources, and how to use techniques like temporal tables, trigger-based logging, and enterprise auditing features. These solutions make sure that monitoring is done in advance, that historical records are kept accurately, and that the company is ready to follow the rules.

As conclusion, this project shows that the best way to keep a database safe is to use an integrated approach that includes strategic planning, theoretical basis, proactive protective measures, and constant monitoring. By connecting academic research with real-world use, our study creates a complete security framework that can protect important data assets, keep businesses running, and help with compliance in today’s complicated and changing threat landscape.

# **References**

Ahmed, M., Khan, M., & Rahman, S. (2022). Enhancing database availability through optimized backup and recovery strategies. International Journal of Advanced Computer Science and Applications, 13(5), 412–420. [https://thesai.org/Publications/ViewPaper?Volume=13&Issue=5&Code=IJACSA&SerialNo=54](https://thesai.org/Publications/ViewPaper?Volume=13&Issue=5&Code=IJACSA&SerialNo=54%20)

Alam, M., Islam, S., & Karim, M. (2023). A framework for secure data storage and sharing in cloud environments. Journal of Information Security and Applications, 73, 103513. <https://www.sciencedirect.com/science/article/pii/S2214212623000513>

Aldossari, M., & Sidorova, A. (2022). Data governance for privacy and security: A framework for digital transformation. Computers & Security, 118, 102746. <https://www.sciencedirect.com/science/article/pii/S0167404822000884>

Alooba. (n.d.). Permission Management. <https://www.alooba.com/skills/concepts/database-security-424/permission-management/>

Alotaibi, Y., Subramani, R., & Alghamdi, S. (2021). Secure and efficient database encryption techniques for cloud computing. International Journal of Advanced Computer Science and Applications, 12(8), 243–251. [https://thesai.org/Publications/ViewPaper?Volume=12&Issue=8&Code=IJACSA&SerialNo=33](https://thesai.org/Publications/ViewPaper?Volume=12&Issue=8&Code=IJACSA&SerialNo=33%20)

Chakraborty, S., & Saha, S. (2021). High availability database design using mirroring and failover clustering. International Journal of Computer Applications, 183(33), 25–31. [https://www.ijcaonline.org/archives/volume183/number33/32174-2021921946](https://www.ijcaonline.org/archives/volume183/number33/32174-2021921946%20)

Choudhary, V., Singh, A., & Sinha, A. (2023). Hybrid encryption framework for secure database management systems. Journal of Information Security and Applications, 75, 103525. <https://www.sciencedirect.com/science/article/pii/S221421262300063X>

Data Sunrise. (n.d.). Role-Based Access Control (RBAC). <https://www.datasunrise.com/knowledge-center/role-based-access-control-rbac/>

Goel, A. (2020, September 23). What is the difference between Encryption and Masking? Which is better for data security? Retrieved from encryptionconsulting: <https://www.encryptionconsulting.com/education-center/encryption-vs-masking>

Hashemi-Pour, C., & Chai, W. (2023, December 21). What is the CIA triad (confidentiality, integrity and availability)?. TechTarget. <https://www.techtarget.com/whatis/definition/Confidentiality-integrity-and-availability-CIA>

Jain, R., & Sharma, A. (2022). Performance evaluation of symmetric and asymmetric encryption algorithms in securing data. International Journal of Computer Applications, 184(10), 15–22. [https://www.ijcaonline.org/archives/volume184/number10/32406-2022922537](https://www.ijcaonline.org/archives/volume184/number10/32406-2022922537%20)

Jain, R., & Singh, P. (2022). A comprehensive review on database auditing and monitoring approaches. International Journal of Computer Applications, 184(25), 28–35. [https://www.ijcaonline.org/archives/volume184/number25/32506-2022922498](https://www.ijcaonline.org/archives/volume184/number25/32506-2022922498%20)

Kaur, P., & Singh, M. (2021). Database security and auditing: An overview of best practices. International Journal of Computer Applications, 183(29), 15–22. [https://www.ijcaonline.org/archives/volume183/number29/32094-2021921651](https://www.ijcaonline.org/archives/volume183/number29/32094-2021921651%20)

Kumar, S., & Kumar, A. (2022). Data masking techniques for secure database management systems. International Journal of Information Security Science, 11(2), 134–146. [https://www.ijiss.org/ijiss/index.php/ijiss/article/view/653](https://www.ijiss.org/ijiss/index.php/ijiss/article/view/653%20)

Microsoft. (2024, November 23). Row-level security. Microsoft Learn. <https://learn.microsoft.com/en-us/sql/relational-databases/security/row-level-security?view=sql-server-ver17>

MongoDB. (n.d.). Role-based Access Control (RBAC). <https://www.mongodb.com/resources/basics/role-based-access-control>

OTI. (2023, July 14). Find Object-Level Permissions in SQL Server with T-SQL. Our Tech Ideas. <https://ourtechideas.com/find-object-level-permissions-in-sql-server-with-t-sql/>

Saxena, A., & Rani, R. (2022). Role-based auditing framework for relational databases. Journal of Information Security and Applications, 66, 103151. <https://www.sciencedirect.com/science/article/pii/S2214212622000601>

Sims, A. (2022, October 24). Data masking vs encryption: what’s the best way to protect my data? Retrieved from Tonic: <https://www.tonic.ai/blog/masking-vs-encryption-whats-the-best-way-to-protect-my-data>

Snowflake. (n.d.). What Is Row-Level Security (RLS)? Benefits and Use Cases. <https://www.snowflake.com/en/fundamentals/row-level-security-tying-data-access-to-user-identity/>

Soni, A., & Sharma, N. (2022). Implementation of temporal tables for auditing in relational databases. International Journal of Computer Science and Mobile Computing, 11(6), 120–127. [https://www.ijcsmc.com/docs/papers/June2022/V11I6202213.pdf](https://www.ijcsmc.com/docs/papers/June2022/V11I6202213.pdf%20)

Subramani, R., Alotaibi, Y., & Alghamdi, S. (2021). Enhanced encryption and masking techniques for secure database systems. International Journal of Computer Applications, 183(16), 30–37. [https://www.ijcaonline.org/archives/volume183/number16/32021-2021921434](https://www.ijcaonline.org/archives/volume183/number16/32021-2021921434%20)

Teradata. (n.d.). What Is Row-Level Security? A Guide to Fine-Tuned Data Access. <https://www.teradata.com/insights/data-security/what-is-row-level-security>