# Database Security

CT069-3-3


## Database Auditing

# Auditing Objective

We need to know WHAT happened to the database, WHEN it happened and WHO caused it to happen

What ➔ add, modify and remove data

Who ➔ who performed that actions

When➔ when this happened

Purpose ➔ We can determine the next course of action accordingly

# Database Auditing



- Database auditing is the activity of monitoring and recording all actions (at server and database level) by all users (admin, apps, end users etc)
- This is to have a compliance-ready environment with increased visibility on those user actions and any potential risks or breaches
- Goal is eventually to ensure the **security and integrity of the databases and the data** in it.

# What to Audit

- Database structural changes (DDL queries)
  - DDL operations can bring about changes to the database structure such as the database and tables
  - DDL auditing records important information about changes made by database query executions, such as database, table and index creation and deletion
- Data access (DQL) and manipulations (DML queries)
  - DML operations can bring about changes to the actual data
  - DML/DQL auditing records important information about changes made on data such as data being viewed, added, updated or deleted

# What to Audit



- Database permission changes (DCL queries)
  - Grant, Deny and Revoke operations can affect user access
  - DCL auditing records important information about changes made to users' permissions
- User Logins

# MS SQL Server Features for Auditing

- SQL Server Audit
- Temporal Tables
- Triggers – DML, DDL and Logon

# SQL Server Audit

A·P·U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

- SQL Server Auditing is a new feature which allow us to audit everything that happens in the server, from server setting changes all the way down to who modified a value in a specific table in the database.
- This information is then written the Windows security log, the Windows application log or to a flat file.

**FILE | APPLICATION_LOG | SECURITY_LOG | URL | EXTERNAL_MONITOR**

- SQL Server auditing feature encompasses two levels
  - The Server Audit
  - The Audit Specification

# Sample DDL Audit


```
--DDL Audit
USE master ;
GO

CREATE SERVER AUDIT DDLActivities_Audit TO FILE ( FILEPATH =
'C:\Temp' );
GO
-- Enable the server audit.
ALTER SERVER AUDIT DDLActivities_Audit WITH (STATE = ON) ;
Go
```

# Create Audit Specification


```
CREATE SERVER AUDIT SPECIFICATION
[DDLActivities_Audit_Specification ]
FOR SERVER AUDIT [DDLActivities_Audit]
ADD (DATABASE_OBJECT_CHANGE_GROUP),
ADD (DATABASE_OBJECT_PERMISSION_CHANGE_GROUP)
WITH (STATE=ON)
Go
```

# Reading The Audit File


```
DECLARE @AuditFilePath VARCHAR(8000);

Select @AuditFilePath = audit_file_path
From sys.dm_server_audit_status
where name = 'DDLActivities_Audit’

--select * from --
sys.fn_get_audit_file(@AuditFilePath,default,default)

select event_time, database_name, database_principal_name,
object_name, statement
from sys.fn_get_audit_file(@AuditFilePath,default,default)
```

# Auditing DML activities


```
USE master ;
GO
CREATE SERVER AUDIT AllTables_DML  TO FILE ( FILEPATH =
'C:\Temp' );
GO
-- Enable the server audit.
ALTER SERVER AUDIT AllTables_DML  WITH (STATE = ON) ;
Go
CREATE DATABASE AUDIT SPECIFICATION AllTables_DML_Specifications
FOR SERVER AUDIT AllTables_DML
ADD ( INSERT , UPDATE, DELETE, SELECT
ON DATABASE::[HospitalInfoSys] BY public)
WITH (STATE = ON) ;
GO
```

# Reading The Audit File


```
DECLARE @AuditFilePath VARCHAR(8000);

Select @AuditFilePath = audit_file_path
From sys.dm_server_audit_status
where name = 'AllTables_DML’

select event_time, database_name, database_principal_name,
object_name, statement
from sys.fn_get_audit_file(@AuditFilePath,default,default)
```

# System Versioned Temporal Tables



- Temporal tables are a SQL Server feature that brings built-in support for providing information about data stored in the table at any point in time, rather than only the data that is correct at the current moment in time.

- A **system-versioned temporal table** is a type of user table designed to keep a full history of data changes, allowing easy point-in-time analysis.

- This type of temporal table is referred to as a system-versioned temporal table because the period of validity for each row is managed by the system (that is, the database engine).

# System Versioned Temporal Tables

- Every temporal table has two explicitly defined columns, each with a **datetime2** data type. These columns are referred to as *period* columns. These period columns are used exclusively by the system to record the period of validity for each row, whenever a row is modified.

- The main table that stores current data is referred to as the *current table*, or simply as the *temporal table*.

- In addition to these period columns, a temporal table also contains a reference to another table with a mirrored schema, called the *history table*.

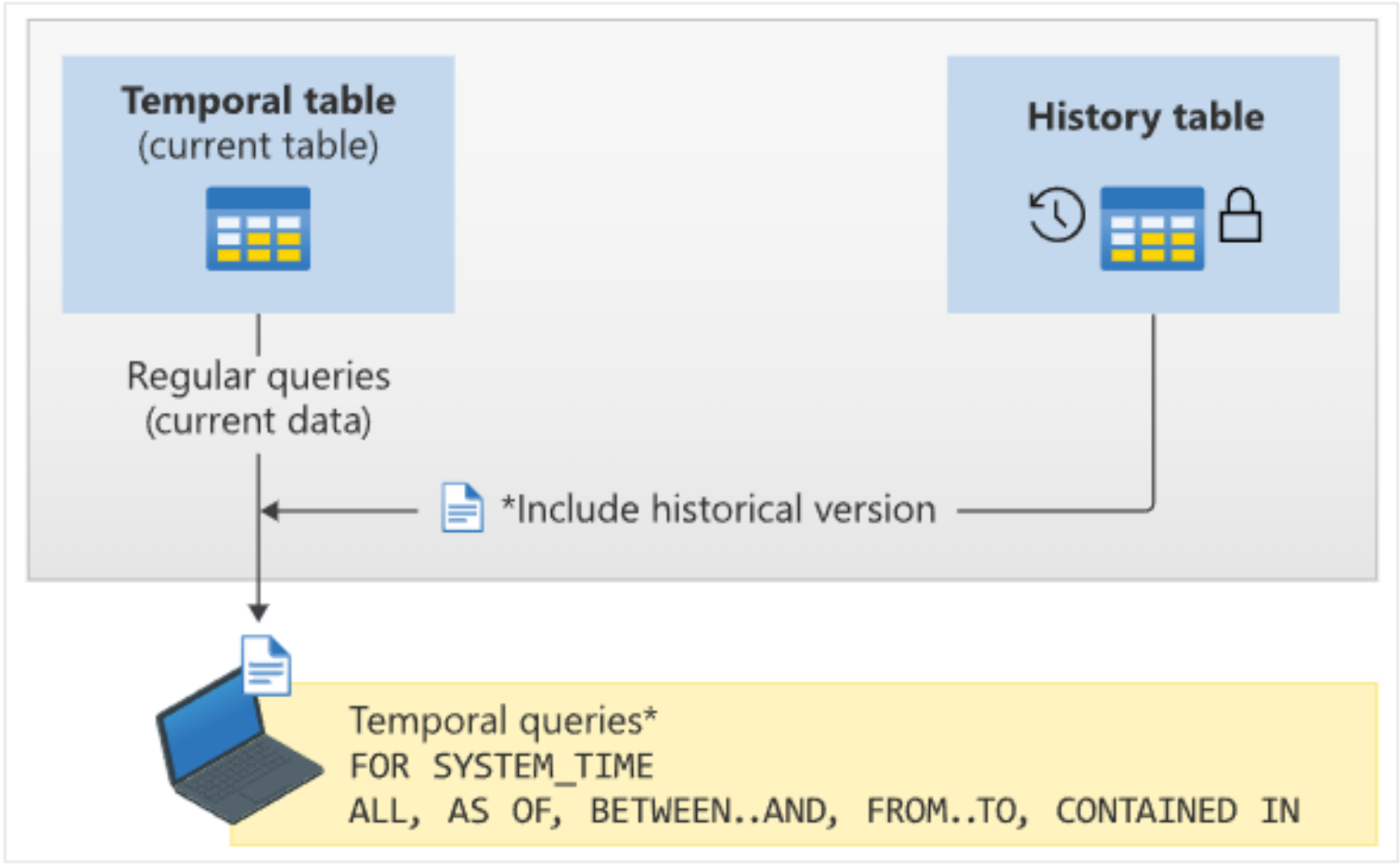
# System Versioned Temporal Tables



- The system uses the history table to automatically store the previous version of the row each time a row in the temporal table gets updated or deleted.

- During temporal table creation users can specify an existing history table (which must be schema compliant) or let the system create a default history table.

# System Versioned Temporal Tables



# System Versioned Temporal Tables


```
USE [HospitalInfoSys]
GO

CREATE TABLE [dbo].[PatientVersioned](
[PatientNo] [varchar](10) primary key,
[PatientName] [varchar](100) NOT NULL,
ValidFrom DATETIME2 GENERATED ALWAYS AS ROW START NOT NULL,
ValidTo DATETIME2 GENERATED ALWAYS AS ROW END NOT NULL,
PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo)
)
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE =
dbo.PatientHistory));
```

# Drop Temporal Tables


```
ALTER TABLE [dbo].[PatientVersioned] SET (
SYSTEM_VERSIONING = OFF)
GO
DROP TABLE [dbo].[PatientVersioned]
GO
DROP TABLE [dbo].[PatientHistory]
GO
```

# Trigger



- Before SQL Audit and System Versioned table features was introduced, audit was done manually using triggers
- Triggers can be useful especially if existing features cannot meet business requirements

# Sample DDL Trigger



Step 1. Create a table to store the audit details

```
CREATE TABLE [AnotherDB].[Audit].[AuditLog_DDL](
    [AuditLogID] [int] Primary Key  IDENTITY(1,1) NOT NULL,
    [LogDate] [datetime] Default GETDATE(),
    [UserName] [sysname] Default USER_NAME(),
    [SQLCmd] [nvarchar](max) NULL
) ;
```

Step 2. Create the trigger to store the audit details

```
CREATE or ALTER TRIGGER TableChanges
ON DATABASE
FOR ALTER_TABLE
AS
-- Detect whether a column was created/altered/dropped.
DECLARE @SQLCmd nvarchar(max)
SELECT @SQLCmd =
EVENTDATA().value('(/EVENT_INSTANCE/TSQLCommand/CommandText)[1]',
'nvarchar(max)')
INSERT INTO [AnotherDB].[Audit].([SQLCmd]) VALUES(@SQLCmd)
```

# Sample DML Trigger

Step 1. Create a table to store the audit details

```
CREATE TABLE [dbo].[Customer_History](
[AuditLogID] [int] IDENTITY(1,1) NOT NULL,
[LogDate] [datetime] Default GETDATE(),
[UserName] [sysname] Default USER_NAME(),
[UserAction] varchar(20),
[MedicineID] [varchar](10) NOT NULL,
[MedicineName] [varchar](100) NULL,
[QuantityInStock] [int] NULL
)
Go
```

# Sample DML Trigger

Step 2. Create the trigger to store the audit details

```
CREATE or ALTER TRIGGER dbo.Medicine_Changes_Trigger
ON [dbo].[Medicine]
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
INSERT INTO [dbo].[Medicine_History]
([UserAction], [MedicineID], [MedicineName], [QuantityInStock])
SELECT 'INSERT', [MedicineID], [MedicineName], [QuantityInStock]
FROM inserted

INSERT INTO [dbo].[Medicine_History]
([UserAction], [MedicineID], [MedicineName], [QuantityInStock])
SELECT 'DELETE', [MedicineID], [MedicineName], [QuantityInStock]
FROM deleted
END;
```


# Q & A