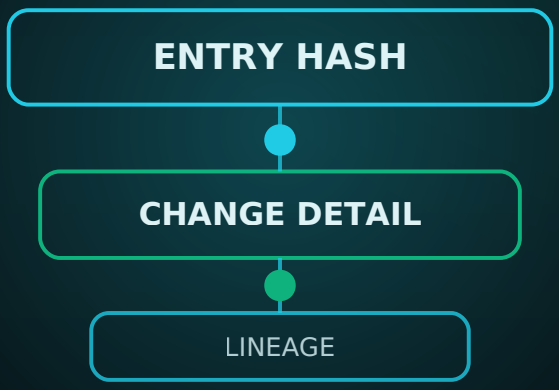
# CHAPTER 22
# AUDIT CONFIGURATIONS

Mapping immutable entries, entity changes, property changes, lineage, access checks, exports, signatures, retention, and integration trace into explicit EF Core configuration files.




CONFIGURATION ATLAS STEP

## One audit chain - every change traceable - fully mapped

SQL Server / EF Core friendly mapping reference for the NEXUS-1 audit sector.

GRIGORIOS AGATHANGELIDIS

DRAFT BUILD - 2026

# Chapter 22 - Audit Configurations

From Entity to Context
The EF Core Configuration Atlas of the NEXUS-1 Digital Twin

---

The Audit schema is the final mapped sector. It does not model a plant component or a report output; it models accountability. The EF Core configuration layer keeps the audit chain append-only in spirit, explicit in naming, and stable under migration.

> **Configuration principle.** Audit mappings favour restrictive deletes, stable target identity, explicit hashes, typed lookups, and named indexes. Nothing should disappear because a parent business row was removed.

# 22.1 Purpose of the Audit configuration layer

The SQL schema defines the tables. The EF Core configuration layer defines how C# respects them: table names, key names, lookup patterns, relationship delete behaviour, target indexes, and hash fields. In a large system this is not decoration; it is how the audit record remains trustworthy.

**Immutable record**
Audit entries, changes, and property details are mapped so history is appended and reviewed, not casually overwritten.

**Traceable action**
Commands, access checks, exports, signatures, and retention runs all carry actor, status, and timing context.

**Lineage**
Data lineage and evidence links connect derived outputs back to source entities and evidence records.

> **Boundary.** EF Core can enforce mapping discipline, but it cannot by itself make an audit regime compliant. Compliance also needs policy, access control, review process, and operational governance.

# 22.2 Configuration file catalogue

One entity, one configuration file. This is especially important for Audit because many tables are shared by every other sector.

**Lookup**
**11 configurations**
AuditActionType, AuditEntityType, AuditSeverity, ChangeSourceType, CommandStatus ...

**Audit Chain**
**4 configurations**
AuditBatch, AuditEntry, EntityChange, PropertyChange

**Commands and Integration**
**3 configurations**
CommandLog, IntegrationMessage, IntegrationMessageAttempt

**Lineage and Evidence**
**3 configurations**
DataLineage, LineageLink, AuditEvidenceLink

**Access, Export, Signatures**
**3 configurations**
AccessAudit, ExportAudit, SignatureRecord

**Retention and Temporal**
**3 configurations**
RetentionPolicy, RetentionExecution, TemporalTableRegistry

| GROUP | TABLE | CONFIGURATION FILE | MAPPING PURPOSE |
|---|---|---|---|
| Lookup | `Audit.AuditActionType` | `AuditActionTypeConfiguration.cs` | Classifies the action recorded: insert, update, delete, sign, export, login, retention, or system action. |
| Lookup | `Audit.AuditEntityType` | `AuditEntityTypeConfiguration.cs` | Classifies what kind of target was changed or observed: business entity, document, signal, report, policy, or configuration. |
| Lookup | `Audit.AuditSeverity` | `AuditSeverityConfiguration.cs` | Severity band for audit events that need attention or escalation. |
| Lookup | `Audit.ChangeSourceType` | `ChangeSourceTypeConfiguration.cs` | Origin of the change: user, API, job, import, system, simulation, or integration. |
| Lookup | `Audit.CommandStatus` | `CommandStatusConfiguration.cs` | Execution status for commands and retention runs. |
| Lookup | `Audit.IntegrationMessageStatus` | `IntegrationMessageStatusConfiguration.cs` | Lifecycle state for inbound or outbound integration messages. |
| Lookup | `Audit.RetentionActionType` | `RetentionActionTypeConfiguration.cs` | Retention action: archive, delete, anonymize, seal, or review. |
| Lookup | `Audit.RetentionRuleStatus` | `RetentionRuleStatusConfiguration.cs` | Lifecycle status for retention rules and policies. |
| Lookup | `Audit.SignatureStatus` | `SignatureStatusConfiguration.cs` | Validation state for electronic signatures. |
| Lookup | `Audit.AccessResult` | `AccessResultConfiguration.cs` | Allowed, denied, challenged, expired, or not applicable access decisions. |
| Lookup | `Audit.ExportPurpose` | `ExportPurposeConfiguration.cs` | Reason for a data export: operational, compliance, investigation, training, or support. |
| Audit Chain | `Audit.AuditBatch` | `AuditBatchConfiguration.cs` | Groups related audit entries into one transaction, command, job, import, or service operation. |
| Audit Chain | `Audit.AuditEntry` | `AuditEntryConfiguration.cs` | The central immutable audit event: actor, action, target identity, correlation, hashes, and timestamp. |
| Audit Chain | `Audit.EntityChange` | `EntityChangeConfiguration.cs` | Row-level change record associated with an audit entry. |

| GROUP | TABLE | CONFIGURATION FILE | MAPPING PURPOSE |
|---|---|---|---|
| Audit Chain | `Audit.PropertyChange` | `PropertyChangeConfiguration.cs` | Column/property-level before-and-after details associated with an entity change. |
| Commands and Integration | `Audit.CommandLog` | `CommandLogConfiguration.cs` | Records application commands with input/output hashes, status, timing, and actor context. |
| Lineage and Evidence | `Audit.DataLineage` | `DataLineageConfiguration.cs` | Captures source-to-target lineage for derived rows, reports, snapshots, and integrations. |
| Lineage and Evidence | `Audit.LineageLink` | `LineageLinkConfiguration.cs` | Links lineage records into chains so a derived result can be traced backward. |
| Commands and Integration | `Audit.IntegrationMessage` | `IntegrationMessageConfiguration.cs` | Inbound or outbound integration message envelope with correlation and status. |
| Commands and Integration | `Audit.IntegrationMessageAttempt` | `IntegrationMessageAttemptConfiguration.cs` | Delivery/processing attempt history for one integration message. |
| Access, Export, Signatures | `Audit.AccessAudit` | `AccessAuditConfiguration.cs` | Records authorization checks, allowed/denied outcomes, client context, and target resource identity. |
| Access, Export, Signatures | `Audit.ExportAudit` | `ExportAuditConfiguration.cs` | Records data export requests, purpose, actor, filters, row counts, and file hashes. |
| Access, Export, Signatures | `Audit.SignatureRecord` | `SignatureRecordConfiguration.cs` | Records a signature over a target business object, evidence item, report, or approval. |
| Retention and Temporal | `Audit.RetentionPolicy` | `RetentionPolicyConfiguration.cs` | Defines retention rules for schema/table targets and the action to take. |
| Retention and Temporal | `Audit.RetentionExecution` | `RetentionExecutionConfiguration.cs` | One execution of a retention policy with status, timing, actor, and rows affected. |
| Retention and Temporal | `Audit.TemporalTableRegistry` | `TemporalTableRegistryConfiguration.cs` | Registry of system-versioned tables and their history table mapping. |
| Lineage and Evidence | `Audit.AuditEvidenceLink` | `AuditEvidenceLinkConfiguration.cs` | Links audit entries to evidence records without pretending every possible target has the same PK type. |

# 22.3 EF Core dependency diagram

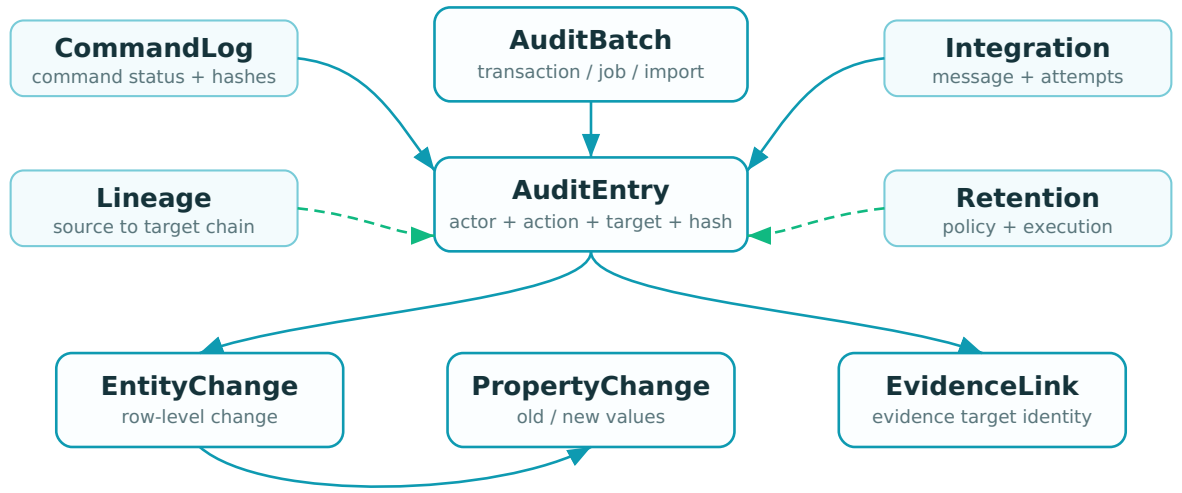



*Figure 22.1 - Audit mapping centre. Batch and entry form the chain; entity and property changes record detail; evidence, commands, integration, lineage, and retention surround it.*

## Configuration folders

```
Nexus.Data/
  Configurations/
    Audit/
      Lookups/
        AuditActionTypeConfiguration.cs
        AuditEntityTypeConfiguration.cs
        AuditSeverityConfiguration.cs
        ChangeSourceTypeConfiguration.cs
        ...
      AuditChain/
        AuditBatchConfiguration.cs
        AuditEntryConfiguration.cs
        EntityChangeConfiguration.cs
        PropertyChangeConfiguration.cs
      CommandsAndIntegration/
        CommandLogConfiguration.cs
        IntegrationMessageConfiguration.cs
        IntegrationMessageAttemptConfiguration.cs
      LineageAndEvidence/
        DataLineageConfiguration.cs
        LineageLinkConfiguration.cs
        AuditEvidenceLinkConfiguration.cs
      AccessExportSignature/
        AccessAuditConfiguration.cs
        ExportAuditConfiguration.cs
        SignatureRecordConfiguration.cs
      RetentionAndTemporal/
        RetentionPolicyConfiguration.cs
        RetentionExecutionConfiguration.cs
        TemporalTableRegistryConfiguration.cs
```

# 22.4 Cross-schema passport map

The Audit sector references users and signers strongly where the lifecycle requires it, but it does not foreign-key every audited business row. For audited targets it stores stable identity: schema name, table name, and entity key. That choice preserves history even when source rows are archived, retained, or deleted.

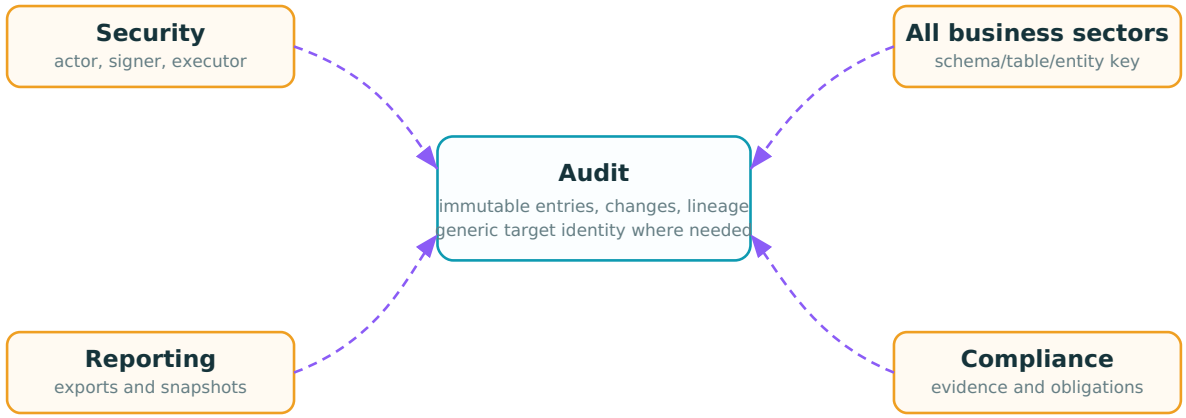



*Figure 22.2 - Audit passports. Actor references are strong; target references are often stable textual identities so historical trace survives source lifecycle changes.*

# 22.5 Shared lookup configuration

Audit lookups define action, severity, source, command status, integration status, retention status, signature status, access result, and export purpose. The base configuration keeps the lookup shape uniform.

```
public abstract class AuditLookupConfiguration<TEntity>
    : IEntityTypeConfiguration<TEntity>
    where TEntity : AuditLookup
{
    protected abstract string TableName { get; }
    protected abstract string KeyName { get; }

    public virtual void Configure(EntityTypeBuilder<TEntity> b)
    {
        b.ToTable(TableName, "Audit");
        b.HasKey(e => e.Id).HasName($"PK_Audit_{TableName}");
        b.Property(e => e.Id).HasColumnName(KeyName);

        b.Property(e => e.Code).HasMaxLength(60).IsRequired();
        b.Property(e => e.Name).HasMaxLength(160).IsRequired();
        b.Property(e => e.Description).HasMaxLength(500);
        b.Property(e => e.IsActive).HasDefaultValue(true);
        b.Property(e => e.DisplayOrder).HasDefaultValue(0);

        b.HasIndex(e => e.Code)
            .IsUnique()
            .HasDatabaseName($"UQ_Audit_{TableName}_Code");
    }
}
```

## Lookup catalogue

| LOOKUP TABLE | REPRESENTATIVE SEED CODES | USED FOR |
|---|---|---|
| `Audit.AuditActionType` | INSERT, UPDATE, DELETE, LOGIN, LOGOUT, ACKNOWLEDGE, APPROVE, EXPORT, SIGN, RETENTION | Classifies the action recorded: insert, update, delete, sign, export, login, retention, or system action. |
| `Audit.AuditEntityType` | BUSINESS_ENTITY, EVENT, MEASUREMENT, DOCUMENT, REPORT, CONFIGURATION, SECURITY_OBJECT | Classifies what kind of target was changed or observed: business entity, document, signal, report, policy, or configuration. |
| `Audit.AuditSeverity` | INFO, NOTICE, WARNING, CRITICAL | Severity band for audit events that need attention or escalation. |
| `Audit.ChangeSourceType` | USER, API, JOB, IMPORT, SYSTEM, SIMULATION, INTEGRATION | Origin of the change: user, API, job, import, system, simulation, or integration. |
| `Audit.CommandStatus` | STARTED, SUCCEEDED, FAILED, CANCELLED, TIMED_OUT | Execution status for commands and retention runs. |
| `Audit.IntegrationMessageStatus` | PENDING, SENT, RECEIVED, FAILED, DEAD_LETTERED | Lifecycle state for inbound or outbound integration messages. |
| `Audit.RetentionActionType` | ARCHIVE, DELETE, ANONYMIZE, SEAL, REVIEW | Retention action: archive, delete, anonymize, seal, or review. |
| `Audit.RetentionRuleStatus` | DRAFT, ACTIVE, SUSPENDED, RETIRED | Lifecycle status for retention rules and policies. |
| `Audit.SignatureStatus` | PENDING, VALID, INVALID, REVOKED, EXPIRED | Validation state for electronic signatures. |
| `Audit.AccessResult` | ALLOWED, DENIED, CHALLENGED, EXPIRED, NOT_APPLICABLE | Allowed, denied, challenged, expired, or not applicable access decisions. |

| LOOKUP TABLE | REPRESENTATIVE SEED CODES | USED FOR |
| --- | --- | --- |
| `Audit.ExportPurpose` | OPERATIONAL, COMPLIANCE, INVESTIGATION, TRAINING, SUPPORT | Reason for a data export: operational, compliance, investigation, training, or support. |

# 22.6 Audit entry mapping

`AuditEntryConfiguration` is the centre of the chapter. It maps action, actor, target identity, correlation, request metadata, and hash fields.

```
public sealed class AuditEntryConfiguration
    : IEntityTypeConfiguration<AuditEntry>
{
    public void Configure(EntityTypeBuilder<AuditEntry> b)
    {
        b.ToTable("AuditEntry", "Audit");
        b.HasKey(e => e.AuditEntryId)
            .HasName("PK_Audit_AuditEntry");

        b.Property(e => e.SchemaName).HasMaxLength(128).IsRequired();
        b.Property(e => e.TableName).HasMaxLength(128).IsRequired();
        b.Property(e => e.EntityKey).HasMaxLength(200).IsRequired();
        b.Property(e => e.Summary).HasMaxLength(1000);
        b.Property(e => e.OccurredAtUtc).HasDefaultValueSql("SYSUTCDATETIME()");
        b.Property(e => e.HashAlgorithm).HasMaxLength(30).HasDefaultValue("SHA2-256");
        b.Property(e => e.PreviousEntryHash).HasMaxLength(32);
        b.Property(e => e.EntryHash).HasMaxLength(32);

        b.HasOne(e => e.AuditBatch)
            .WithMany(e => e.Entries)
            .HasForeignKey(e => e.AuditBatchId)
            .OnDelete(DeleteBehavior.Restrict)
            .HasConstraintName("FK_AuditEntry_AuditBatch");

        b.HasOne(e => e.ActionType)
            .WithMany()
            .HasForeignKey(e => e.AuditActionTypeId)
            .OnDelete(DeleteBehavior.Restrict)
            .HasConstraintName("FK_AuditEntry_ActionType");

        b.HasIndex(e => new { e.SchemaName, e.TableName, e.EntityKey, e.OccurredAtUtc })
            .HasDatabaseName("IX_Audit_AuditEntry_Target_Time");

        b.HasIndex(e => e.CorrelationId)
            .HasDatabaseName("IX_Audit_AuditEntry_CorrelationId");
    }
}
```

# 22.7 Entity and property change mapping

Row-level and property-level changes are separated so a system can answer both questions: which row changed, and exactly which property changed?

```
public sealed class EntityChangeConfiguration
    : IEntityTypeConfiguration<EntityChange>
{
    public void Configure(EntityTypeBuilder<EntityChange> b)
    {
        b.ToTable("EntityChange", "Audit");
        b.HasKey(e => e.EntityChangeId)
            .HasName("PK_Audit_EntityChange");

        b.Property(e => e.SchemaName).HasMaxLength(128).IsRequired();
        b.Property(e => e.TableName).HasMaxLength(128).IsRequired();
        b.Property(e => e.PrimaryKeyName).HasMaxLength(128).IsRequired();
        b.Property(e => e.PrimaryKeyValue).HasMaxLength(200).IsRequired();
        b.Property(e => e.Operation).HasMaxLength(20).IsRequired();
        b.Property(e => e.BeforeJson);
        b.Property(e => e.AfterJson);

        b.HasOne(e => e.AuditEntry)
            .WithMany(e => e.EntityChanges)
            .HasForeignKey(e => e.AuditEntryId)
            .OnDelete(DeleteBehavior.Restrict)
            .HasConstraintName("FK_EntityChange_AuditEntry");

        b.HasIndex(e => new { e.SchemaName, e.TableName, e.PrimaryKeyValue })
            .HasDatabaseName("IX_Audit_EntityChange_Target");
    }
}
```

```
public sealed class PropertyChangeConfiguration
    : IEntityTypeConfiguration<PropertyChange>
{
    public void Configure(EntityTypeBuilder<PropertyChange> b)
    {
        b.ToTable("PropertyChange", "Audit");
        b.HasKey(e => e.PropertyChangeId)
            .HasName("PK_Audit_PropertyChange");

        b.Property(e => e.PropertyName).HasMaxLength(128).IsRequired();
        b.Property(e => e.OldValue).HasMaxLength(4000);
        b.Property(e => e.NewValue).HasMaxLength(4000);
        b.Property(e => e.ValueType).HasMaxLength(80);
        b.Property(e => e.IsSensitive).HasDefaultValue(false);

        b.HasOne(e => e.EntityChange)
            .WithMany(e => e.PropertyChanges)
            .HasForeignKey(e => e.EntityChangeId)
            .OnDelete(DeleteBehavior.Restrict)
            .HasConstraintName("FK_PropertyChange_EntityChange");

        b.HasIndex(e => new { e.EntityChangeId, e.PropertyName })
            .HasDatabaseName("IX_Audit_PropertyChange_Entity_Property");
    }
}
```

# 22.8 Command and integration mapping

Commands describe what the application tried to do. Integration messages describe what crossed a boundary. Both are audit facts, and both deserve named mappings.

```
public sealed class CommandLogConfiguration
    : IEntityTypeConfiguration<CommandLog>
{
    public void Configure(EntityTypeBuilder<CommandLog> b)
    {
        b.ToTable("CommandLog", "Audit");
        b.HasKey(e => e.CommandLogId)
            .HasName("PK_Audit_CommandLog");

        b.Property(e => e.CommandName).HasMaxLength(180).IsRequired();
        b.Property(e => e.HandlerName).HasMaxLength(220);
        b.Property(e => e.InputHash).HasMaxLength(128);
        b.Property(e => e.OutputHash).HasMaxLength(128);
        b.Property(e => e.ErrorMessage).HasMaxLength(2000);
        b.Property(e => e.StartedAtUtc).HasDefaultValueSql("SYSUTCDATETIME()");

        b.HasOne(e => e.CommandStatus)
            .WithMany()
            .HasForeignKey(e => e.CommandStatusId)
            .OnDelete(DeleteBehavior.Restrict)
            .HasConstraintName("FK_CommandLog_CommandStatus");

        b.HasIndex(e => new { e.CommandName, e.StartedAtUtc })
            .HasDatabaseName("IX_Audit_CommandLog_Command_Time");
    }
}
```

```
public sealed class IntegrationMessageConfiguration
    : IEntityTypeConfiguration<IntegrationMessage>
{
    public void Configure(EntityTypeBuilder<IntegrationMessage> b)
    {
        b.ToTable("IntegrationMessage", "Audit");
        b.HasKey(e => e.IntegrationMessageId)
            .HasName("PK_Audit_IntegrationMessage");

        b.Property(e => e.ExternalSystem).HasMaxLength(120).IsRequired();
        b.Property(e => e.MessageType).HasMaxLength(120).IsRequired();
        b.Property(e => e.Direction).HasMaxLength(20).IsRequired();
        b.Property(e => e.PayloadHash).HasMaxLength(128);
        b.Property(e => e.CreatedAtUtc).HasDefaultValueSql("SYSUTCDATETIME()");

        b.HasOne(e => e.Status)
            .WithMany()
            .HasForeignKey(e => e.IntegrationMessageStatusId)
            .OnDelete(DeleteBehavior.Restrict)
            .HasConstraintName("FK_IntegrationMessage_Status");

        b.HasIndex(e => new { e.ExternalSystem, e.MessageType, e.CreatedAtUtc })
            .HasDatabaseName("IX_Audit_IntegrationMessage_System_Type_Time");
    }
}
```

# 22.9 Access, export, and signature mapping

Access checks, exports, and signatures often become the records people ask for after an incident. Their mappings prioritize actor, target, status, timestamp, and hashes.

```
public sealed class AccessAuditConfiguration
    : IEntityTypeConfiguration<AccessAudit>
{
    public void Configure(EntityTypeBuilder<AccessAudit> b)
    {
        b.ToTable("AccessAudit", "Audit");
        b.HasKey(e => e.AccessAuditId)
            .HasName("PK_Audit_AccessAudit");

        b.Property(e => e.ResourceSchema).HasMaxLength(128).IsRequired();
        b.Property(e => e.ResourceTable).HasMaxLength(128).IsRequired();
        b.Property(e => e.ResourceKey).HasMaxLength(200).IsRequired();
        b.Property(e => e.PermissionCode).HasMaxLength(120).IsRequired();
        b.Property(e => e.ClientIp).HasMaxLength(64);
        b.Property(e => e.OccurredAtUtc).HasDefaultValueSql("SYSUTCDATETIME()");

        b.HasOne(e => e.Result)
            .WithMany()
            .HasForeignKey(e => e.AccessResultId)
            .OnDelete(DeleteBehavior.Restrict)
            .HasConstraintName("FK_AccessAudit_AccessResult");

        b.HasIndex(e => new { e.ActorUserId, e.OccurredAtUtc })
            .HasDatabaseName("IX_Audit_AccessAudit_Actor_Time");
    }
}
```

```
public sealed class ExportAuditConfiguration
    : IEntityTypeConfiguration<ExportAudit>
{
    public void Configure(EntityTypeBuilder<ExportAudit> b)
    {
        b.ToTable("ExportAudit", "Audit");
        b.HasKey(e => e.ExportAuditId)
            .HasName("PK_Audit_ExportAudit");

        b.Property(e => e.ExportName).HasMaxLength(220).IsRequired();
        b.Property(e => e.TargetSchema).HasMaxLength(128);
        b.Property(e => e.TargetTable).HasMaxLength(128);
        b.Property(e => e.FilterJson);
        b.Property(e => e.FileHash).HasMaxLength(128);
        b.Property(e => e.RequestedAtUtc).HasDefaultValueSql("SYSUTCDATETIME()");

        b.HasOne(e => e.Purpose)
            .WithMany()
            .HasForeignKey(e => e.ExportPurposeId)
            .OnDelete(DeleteBehavior.Restrict)
            .HasConstraintName("FK_ExportAudit_ExportPurpose");

        b.HasIndex(e => new { e.RequestedByUserId, e.RequestedAtUtc })
            .HasDatabaseName("IX_Audit_ExportAudit_User_Time");
    }
}
```

```
public sealed class SignatureRecordConfiguration
    : IEntityTypeConfiguration<SignatureRecord>
{
    public void Configure(EntityTypeBuilder<SignatureRecord> b)
    {
        b.ToTable("SignatureRecord", "Audit");
        b.HasKey(e => e.SignatureRecordId)
```

```
            .HasName("PK_Audit_SignatureRecord");

        b.Property(e => e.TargetSchema).HasMaxLength(128).IsRequired();
        b.Property(e => e.TargetTable).HasMaxLength(128).IsRequired();
        b.Property(e => e.TargetKey).HasMaxLength(200).IsRequired();
        b.Property(e => e.SignaturePurpose).HasMaxLength(160).IsRequired();
        b.Property(e => e.SignatureHash).HasMaxLength(128).IsRequired();
        b.Property(e => e.CertificateThumbprint).HasMaxLength(120);
        b.Property(e => e.SignedAtUtc).HasDefaultValueSql("SYSUTCDATETIME()");

        b.HasOne(e => e.SignatureStatus)
            .WithMany()
            .HasForeignKey(e => e.SignatureStatusId)
            .OnDelete(DeleteBehavior.Restrict)
            .HasConstraintName("FK_SignatureRecord_Status");

        b.HasIndex(e => new { e.TargetSchema, e.TargetTable, e.TargetKey })
            .HasDatabaseName("IX_Audit_SignatureRecord_Target");
    }
}
```

# 22.10 Retention mapping

Retention is itself auditable. A retention policy is a governed rule; a retention execution is a historical event. The mapping keeps both visible.

```
public sealed class RetentionPolicyConfiguration
    : IEntityTypeConfiguration<RetentionPolicy>
{
    public void Configure(EntityTypeBuilder<RetentionPolicy> b)
    {
        b.ToTable("RetentionPolicy", "Audit");
        b.HasKey(e => e.RetentionPolicyId)
            .HasName("PK_Audit_RetentionPolicy");

        b.Property(e => e.Code).HasMaxLength(80).IsRequired();
        b.Property(e => e.SchemaName).HasMaxLength(128).IsRequired();
        b.Property(e => e.TableName).HasMaxLength(128).IsRequired();
        b.Property(e => e.RetentionDays).IsRequired();
        b.Property(e => e.Description).HasMaxLength(1000);
        b.Property(e => e.RowVersion).IsRowVersion();

        b.HasOne(e => e.ActionType)
            .WithMany()
            .HasForeignKey(e => e.RetentionActionTypeId)
            .OnDelete(DeleteBehavior.Restrict)
            .HasConstraintName("FK_RetentionPolicy_ActionType");

        b.HasIndex(e => new { e.SchemaName, e.TableName })
            .HasDatabaseName("IX_Audit_RetentionPolicy_Target");
    }
}
```

# 22.11 Context discovery

The audit configurations are discovered like every other sector. The `DbContext` remains clean even though Audit contains cross-cutting infrastructure.

```
public sealed class NexusContext : DbContext
{
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.ApplyConfigurationsFromAssembly(
            typeof(NexusContext).Assembly);
    }
}
```

> **Why this matters.** Audit is the easiest sector to turn into a giant special case. Keeping one configuration file per entity prevents that.

# 22.12 Foreign-key mapping

The table below lists the main configuration targets. Internal relationships stay inside `Audit`; actor and signer relationships cross into `Security`.

| ENTITY | FK PROPERTY | TARGET | KIND |
|---|---|---|---|
| `Audit.AuditBatch` | `ChangeSourceTypeId` | `Audit.ChangeSourceType(ChangeSourceTypeId)` | internal |
| `Audit.AuditBatch` | `StartedByUserId` | `Security.ApplicationUser(ApplicationUserId)` | passport |
| `Audit.AuditEntry` | `AuditBatchId` | `Audit.AuditBatch(AuditBatchId)` | internal |
| `Audit.AuditEntry` | `ActionTypeId` | `Audit.AuditActionType(AuditActionTypeId)` | internal |
| `Audit.AuditEntry` | `EntityTypeId` | `Audit.AuditEntityType(AuditEntityTypeId)` | internal |
| `Audit.AuditEntry` | `SeverityId` | `Audit.AuditSeverity(AuditSeverityId)` | internal |
| `Audit.AuditEntry` | `ChangeSourceTypeId` | `Audit.ChangeSourceType(ChangeSourceTypeId)` | internal |
| `Audit.AuditEntry` | `PerformedByUserId` | `Security.ApplicationUser(ApplicationUserId)` | passport |
| `Audit.EntityChange` | `AuditEntryId` | `Audit.AuditEntry(AuditEntryId)` | internal |
| `Audit.PropertyChange` | `EntityChangeId` | `Audit.EntityChange(EntityChangeId)` | internal |
| `Audit.CommandLog` | `CommandStatusId` | `Audit.CommandStatus(CommandStatusId)` | internal |
| `Audit.CommandLog` | `SourceTypeId` | `Audit.ChangeSourceType(ChangeSourceTypeId)` | internal |
| `Audit.CommandLog` | `RequestedByUserId` | `Security.ApplicationUser(ApplicationUserId)` | passport |
| `Audit.DataLineage` | `CreatedByUserId` | `Security.ApplicationUser(ApplicationUserId)` | passport |
| `Audit.LineageLink` | `DataLineageId` | `Audit.DataLineage(DataLineageId)` | internal |
| `Audit.IntegrationMessageAttempt` | `IntegrationMessageId` | `Audit.IntegrationMessage(IntegrationMessageId)` | internal |
| `Audit.IntegrationMessageAttempt` | `IntegrationMessageStatusId` | `Audit.IntegrationMessageStatus(IntegrationMessageStatusId)` | internal |
| `Audit.AccessAudit` | `ApplicationUserId` | `Security.ApplicationUser(ApplicationUserId)` | passport |
| `Audit.AccessAudit` | `AccessResultId` | `Audit.AccessResult(AccessResultId)` | internal |
| `Audit.AccessAudit` | `SourceTypeId` | `Audit.ChangeSourceType(ChangeSourceTypeId)` | internal |
| `Audit.ExportAudit` | `ReportExportId` | `Reporting.ReportExport(ReportExportId)` | passport |
| `Audit.ExportAudit` | `ExportedByUserId` | `Security.ApplicationUser(ApplicationUserId)` | passport |
| `Audit.ExportAudit` | `ExportPurposeId` | `Audit.ExportPurpose(ExportPurposeId)` | internal |

| ENTITY | FK PROPERTY | TARGET | KIND |
|---|---|---|---|
| `Audit.SignatureRecord` | `SignatureStatusId` | `Audit.SignatureStatus(SignatureStatusId)` | internal |
| `Audit.SignatureRecord` | `SignedByUserId` | `Security.ApplicationUser(ApplicationUserId)` | passport |
| `Audit.RetentionPolicy` | `RetentionActionTypeId` | `Audit.RetentionActionType(RetentionActionTypeId)` | internal |
| `Audit.RetentionPolicy` | `RetentionRuleStatusId` | `Audit.RetentionRuleStatus(RetentionRuleStatusId)` | internal |
| `Audit.RetentionExecution` | `RetentionPolicyId` | `Audit.RetentionPolicy(RetentionPolicyId)` | internal |
| `Audit.RetentionExecution` | `RetentionActionTypeId` | `Audit.RetentionActionType(RetentionActionTypeId)` | internal |
| `Audit.RetentionExecution` | `ExecutedByUserId` | `Security.ApplicationUser(ApplicationUserId)` | passport |
| `Audit.TemporalTableRegistry` | `RetentionPolicyId` | `Audit.RetentionPolicy(RetentionPolicyId)` | internal |
| `Audit.AuditEvidenceLink` | `AuditEntryId` | `Audit.AuditEntry(AuditEntryId)` | internal |
| `Audit.AuditEvidenceLink` | `ComplianceEvidenceId` | `Compliance.Evidence(EvidenceId)` | passport |

# 22.13 Migration and verification notes

**Restrictive deletes**
Use `DeleteBehavior.Restrict` on audit relationships. Audit history must not be cascade-deleted by business operations.

**Stable target identity**
Use `SchemaName`, `TableName`, and `EntityKey` for generic targets so archived or deleted source records remain traceable.

**Hashes**
Map `EntryHash`, `PreviousEntryHash`, payload hashes, file hashes, and signature hashes explicitly.

**Time indexes**
Most audit investigations start from actor, target, correlation ID, or time window. Index accordingly.

```
// Verification query idea after migration
SELECT name
FROM sys.indexes
WHERE object_id = OBJECT_ID('Audit.AuditEntry')
ORDER BY name;

// EF Core model check idea
var auditTypes = context.Model.GetEntityTypes()
    .Where(e => e.GetSchema() == "Audit")
    .Select(e => e.GetTableName())
    .OrderBy(n => n)
    .ToList();

// Every Audit configuration should be discovered by ApplyConfigurationsFromAssembly.
```

# 22.14 Configuration file index

| | |
|---|---|
| **AuditActionTypeConfiguration.cs** | Lookup |
| **AuditEntityTypeConfiguration.cs** | Lookup |
| **AuditSeverityConfiguration.cs** | Lookup |
| **ChangeSourceTypeConfiguration.cs** | Lookup |
| **CommandStatusConfiguration.cs** | Lookup |
| **IntegrationMessageStatusConfiguration.cs** | Lookup |
| **RetentionActionTypeConfiguration.cs** | Lookup |
| **RetentionRuleStatusConfiguration.cs** | Lookup |
| **SignatureStatusConfiguration.cs** | Lookup |
| **AccessResultConfiguration.cs** | Lookup |
| **ExportPurposeConfiguration.cs** | Lookup |
| **AuditBatchConfiguration.cs** | Audit Chain |
| **AuditEntryConfiguration.cs** | Audit Chain |
| **EntityChangeConfiguration.cs** | Audit Chain |
| **PropertyChangeConfiguration.cs** | Audit Chain |
| **CommandLogConfiguration.cs** | Commands and Integration |
| **DataLineageConfiguration.cs** | Lineage and Evidence |
| **LineageLinkConfiguration.cs** | Lineage and Evidence |
| **IntegrationMessageConfiguration.cs** | Commands and Integration |
| **IntegrationMessageAttemptConfiguration.cs** | Commands and Integration |
| **AccessAuditConfiguration.cs** | Access, Export, Signatures |
| **ExportAuditConfiguration.cs** | Access, Export, Signatures |
| **SignatureRecordConfiguration.cs** | Access, Export, Signatures |
| **RetentionPolicyConfiguration.cs** | Retention and Temporal |
| **RetentionExecutionConfiguration.cs** | Retention and Temporal |
| **TemporalTableRegistryConfiguration.cs** | Retention and Temporal |
| **AuditEvidenceLinkConfiguration.cs** | Lineage and Evidence |

**End of schema atlas boundary.** This chapter completes the schema-by-schema EF Core configuration atlas. The next natural step is a combined master PDF with front matter, all chapters, consolidated indexes, and a cross-schema configuration catalogue.