immoGenio

No. 02
Mai 2026

AUDIT-TRAIL

Audit-Trail in der Hausverwaltungs-Software: Append-only, Trigger und der Unterschied zwischen Logging und Revisionssicherheit

Warum ein ehrlicher Audit-Trail Append-only sein muss: GoBD, AO §147, Art. 5 DSGVO Rechenschaftspflicht — und wie PostgreSQL-Trigger die Manipulation auf DB-Ebene verhindern.

AUTOR
ImmoGenio

VERÖFFENTLICHT
1. Mai 2026

ONLINE
www.immogenio.de/blog

— AUF EINEN BLICK

Inhalt

Der Anruf, der alles auslöst

Es ist Mittwochnachmittag, ein Eigentümer ruft an. Er hat seine Hausgeldabrechnung 2024 vor sich liegen und vergleicht sie mit der von 2023. Die Sollstellung ist um 12,40 € pro Monat höher. Er fragt: Wer hat das wann geändert, und auf welcher Grundlage. Der Verwalter öffnet die Software, sieht den aktuellen Wert — und keinen Hinweis darauf, wie er zustande gekommen ist. Kein Eintrag, kein Vorgängerwert, kein Bearbeiter, kein Datum. Nur ein Feld, in dem irgendwann irgendjemand etwas eingetragen hat.

Genau hier scheidet sich Hausverwaltungs-Software, die einer Betriebsprüfung standhält, von Software, die in der Steuerprüfung peinlich wird. Ein revisionssicherer Audit-Trail ist keine optionale Komfortfunktion, sondern die technische Voraussetzung dafür, dass die Verwaltung ihrer Rechenschaftspflicht überhaupt nachkommen kann. In diesem Beitrag zeigen wir, warum lückenloses Mutations-Tracking aus AO §§ 145–147, HGB § 257 und Art. 5 Abs. 2 DSGVO folgt, warum es auf Datenbank-Ebene durchgesetzt werden muss und welche konkreten PostgreSQL-Mechanismen ImmoGenio dafür einsetzt.

Was Audit-Trail bedeutet — und was nicht

Der Begriff wird häufig mit „Logging" verwechselt. Beides hat aber sehr unterschiedliche Aufgaben.

Logging dient der Anwendungs-Diagnostik. Wenn eine Route 500 antwortet, wenn ein Hintergrund-Job hängt, wenn ein OAuth-Callback fehlschlägt — dann hilft das Log dem Entwickler, das Problem zu rekonstruieren. Logs dürfen rotieren, nach 14 Tagen verschwinden, durch Logrotate verkürzt werden. Sie sind Werkzeug, nicht Beweismittel.

Audit-Trail ist das Gegenteil. Er ist die Buchhaltungs-Variante des Tagebuchs: jede fachliche Änderung, jede Mutation an buchhaltungsrelevanten oder personenbezogenen Daten wird protokolliert, und zwar so, dass sie nicht mehr nachträglich verändert werden kann. Wer den Audit-Trail rotiert oder kürzt, hat ihn missverstanden. Aufbewahrungspflichten und Revisionssicherheit greifen — die Details haben wir im Beitrag zu Aufbewahrungsfristen nach HGB, AO, WEG und BetrSichV im Detail beschrieben.

Die regulatorischen Treiber

Ein Audit-Trail ist in der Hausverwaltung kein „Best Practice", sondern eine direkte Folge mehrerer regulatorischer Anforderungen, die sich gegenseitig verstärken.

GoBD (BMF-Schreiben vom 28.11.2019) verlangt für aufzeichnungs- und aufbewahrungspflichtige Daten die Grundsätze der **Nachvollziehbarkeit, Nachprüfbarkeit, Vollständigkeit, Richtigkeit, zeitgerechten Buchungen, Ordnung** und vor allem der **Unveränderbarkeit**. In Randziffer 58 ff. heißt es sinngemäß: Eine Buchung darf nicht so verändert werden, dass der ursprüngliche Inhalt nicht mehr feststellbar ist. Ohne Audit-Trail ist diese Anforderung technisch nicht erfüllbar.

AO § 147 schreibt für aufzeichnungspflichtige Unterlagen eine Aufbewahrungspflicht von zehn Jahren in lesbarer Form vor. Das gilt auch für die Änderungshistorie selbst: Wenn der Audit-Trail Bestandteil der Buchführung ist, fällt er unter dieselbe Frist.

HGB § 257 zieht die handelsrechtliche Linie parallel — Bücher, Handelsbriefe, Buchungsbelege müssen so geführt werden, dass ein sachverständiger Dritter sich in angemessener Zeit einen Überblick verschaffen kann.

DSGVO Art. 5 Abs. 2 verankert die **Rechenschaftspflicht**: Der Verantwortliche muss die Einhaltung der Grundsätze nachweisen können. Bei einer Auskunft nach Art. 15 oder einem Berichtigungsverlangen nach Art. 16 gehört dazu, belegen zu können, wann welcher Wert geändert wurde. **Art. 30 DSGVO** ergänzt das durch das Verzeichnis von Verarbeitungstätigkeiten — auch hier ist nachvollziehbare Veränderung Teil der Doku-Pflicht.

IDW PS 880, der Prüfungsstandard für Software-Bescheinigungen, macht explizit: ohne nachvollziehbare Änderungshistorie keine positive Bescheinigung. Wer sich vom Wirtschaftsprüfer eine GoBD-Konformität bestätigen lassen möchte, muss einen Audit-Trail haben, der den Prüfer überzeugt.

BSI-Grundschutz APP.4.6 für Verwaltungssoftware verlangt darüber hinaus die Protokollierung sicherheitsrelevanter Ereignisse mit Schutz vor unbefugter Veränderung — fachlich identisch zum GoBD-Anspruch, nur aus IT-Sicherheits-Perspektive.

Die Quintessenz: Ein Audit-Trail ist keine Komfort-Funktion, sondern Pflicht aus mindestens vier sich überlappenden Regelwerken.

Was ein revisionssicherer Audit-Trail enthalten muss

Aus den Anforderungen lässt sich konkret ableiten, welche Felder ein Audit-Eintrag enthalten muss.

- **Wer**: Eindeutige User-ID, niemals nur „System". Bei Mutationen durch automatisierte Prozesse (Mahnlauf, Abrechnungslauf) wird der Service-Account explizit markiert, mit eindeutiger Service-Identität. Das ist auch Voraussetzung dafür, dass das RBAC-System mit klaren Rollen und Berechtigungen seine forensische Aussagekraft behält.

- **Wann**: UTC-Timestamp aus der Datenbank, monotone Reihenfolge garantiert. Niemals Client-Zeitstempel — die sind manipulierbar und in verteilten Setups unzuverlässig. Postgres' `clock_timestamp()` oder `statement_timestamp()` liefert die richtige Zeitquelle.
- **Was**: Tabelle, Zeilen-Identifier, alter und neuer Wert auf Feldebene. Diff-Granularität ist der entscheidende Punkt — nicht „Datensatz wurde geändert", sondern „Feld `soll_betrag` wurde von 412,60 auf 425,00 geändert".
- **Warum**: Der Kontext, der die Mutation auslöst. API-Endpunkt, Request-ID, Tenant-ID, optional ein fachlicher Begründungstext, den der Anwender bei sensiblen Operationen aktiv eingibt.
- **Wie**: Die Operation selbst — INSERT, UPDATE, DELETE. Bei UPDATE zusätzlich die Liste der tatsächlich veränderten Felder, nicht der gesamte Datensatz.

Ein Audit-Trail, der nur „User X hat um 14:23 etwas an Tabelle Y gemacht" notiert, ist nutzlos. Erst die Diff-Granularität auf Feldebene macht ihn forensisch belastbar.

Architektur: Trigger statt Anwendungs-Code

Die naheliegende Idee ist, in der Anwendungsschicht zu protokollieren — jede Service-Methode schreibt vor und nach der Mutation einen Eintrag in eine Audit-Tabelle. Diese Architektur ist falsch, und zwar aus genau einem Grund: Sie ist nicht erzwingbar.

Sobald ein neuer API-Endpunkt entsteht, ein Migrations-Skript Daten anpasst, ein Wartungs-Job ein Feld korrigiert oder ein Entwickler unter Druck schnell ein Update einbaut, wird der Audit-Aufruf vergessen. Und genau dann fehlt im Streitfall der Eintrag, den die Verwaltung dringend bräuchte. Anwendungs-Code, der für Compliance verantwortlich ist, ist immer einen Refactor entfernt vom Versagen.

Die richtige Architektur platziert die Protokollierung dort, wo sie nicht umgangen werden kann: auf der **Datenbank** selbst. Postgres-Trigger feuern bei jeder INSERT-, UPDATE- und DELETE-Operation auf den fachlich relevanten Tabellen, unabhängig davon, ob die Mutation aus dem API-Layer, einem psql-Befehl oder einem Wartungs-Skript stammt. Die Trigger-Funktion schreibt den vollständigen Diff in die Tabelle `audit_log`, die selbst durch Datenbank-Mechanismen gegen Manipulation geschützt ist.

In ImmoGenio liegt die Tabelle `audit_log` in der Baseline-Migration (000). Die Trigger-Flush-Funktion folgt in Migration 001, die Permission-Updates für den App-User in 002. Keine Anwendungsroute kann die Protokollierung „vergessen", weil sie auf einer Ebene unter dem API-Layer passiert.

Append-only durchsetzen: Drop-no-update und Drop-no-delete

PostgreSQL kennt keine native „Append-only"-Tabelle. Aber es kennt zwei Mechanismen, die zusammen denselben Effekt erzielen.

Der erste ist die explizite **Rule-basierte Sperre**. Mit `CREATE RULE no_update_audit_log AS ON UPDATE TO audit_log DO INSTEAD NOTHING` lässt sich definieren, dass UPDATE-Statements gegen die Tabelle wirkungslos bleiben. Ergänzt durch eine entsprechende Rule für DELETE entsteht eine Tabelle, die nur Inserts akzeptiert. ImmoGenio setzt diese Rules in den Migrationen 003 (UPDATE) und 054 (DELETE) — der zeitliche Versatz spiegelt die Lernkurve wider, denn die DELETE-Sperre kam später, als klar wurde, dass auch versehentliche Massenlöschungen ausgeschlossen werden müssen.

Der zweite Mechanismus ist das **Privilegien-Hardening**. Der App-User, mit dem die Anwendung gegen die Datenbank spricht, hat auf `audit_log` ausschließlich INSERT-Rechte. UPDATE und DELETE sind nicht gewährt. Selbst wenn ein Angreifer SQL-Injection schaffen sollte, kann er die Tabelle nicht modifizieren. Diese Trennung folgt demselben Least-Privilege-Prinzip, das auch die Multi-Tenancy mit Row-Level-Security absichert.

Beide Mechanismen zusammen — Rules und Privileges — machen die nachträgliche Manipulation des Audit-Logs praktisch unmöglich. Der einzige Weg führt über einen Superuser-Zugriff, und der ist im produktiven Setup organisatorisch und technisch reglementiert.

Diff-Granularität — die unterschätzte Frage

Ein Audit-Trail muss alten und neuen Wert speichern. Die Frage ist: in welcher Form. Die naive Antwort lautet: einfach `OLD.*` und `NEW.*` als JSONB. Das ist auch der richtige Ausgangspunkt — aber nicht das Ende.

Erstens müssen **sensible Felder maskiert** werden. Passwort-Hashes haben im Audit-Log nichts zu suchen, ebenso wenig Token, API-Keys oder verschlüsselte Personalausweis-Daten. Die Trigger-Funktion erhält eine Whitelist oder Blacklist von Feldern und behandelt sie entsprechend.

Zweitens müssen **große Blob-Felder ausgeschlossen** werden. Wenn eine Tabelle PDF-Inhalte oder XML-Dokumente direkt enthält (was nicht ideal ist, aber vorkommt), gehören diese nicht in den Audit. Sonst wächst die Tabelle nicht linear, sondern quadratisch — und mit ihr die Backup-Zeit, die Migration-Dauer, die Restore-Zeit.

Drittens lohnt sich die Unterscheidung zwischen **vollständigem Snapshot** und **Diff**. Bei kleinen Tabellen ist der Snapshot pragmatisch. Bei breiten Tabellen mit 80 Spalten ist nur das Diff sinnvoll — also ein JSONB-Objekt mit ausschließlich den geänderten Feldern. Die Diff-Berechnung übernimmt die Trigger-Funktion zur Laufzeit.

Performance: Wie eine 50-GB-Audit-Tabelle nicht zur Bremse wird

Audit-Logs wachsen schnell. Eine mittlere Hausverwaltung mit 5.000 Wohneinheiten produziert leicht zweistellige Millionen Audit-Einträge pro Jahr. Ohne Architektur-Konzept wird die Tabelle zur Bremse für Backups, Migrations und Reports.

Drei Maßnahmen zusammen halten das im Griff.

Partitionierung nach Monat: Die Tabelle wird über `PARTITION BY RANGE (created_at)` in Monatspartitionen aufgeteilt. Queries, die nach Zeitraum filtern (was bei Audit-Abfragen die Regel ist), berühren nur die relevante Partition. Alte Partitionen können separat gesichert oder ausgelagert werden, ohne den Rest der Tabelle zu beeinflussen.

BRIN-Index auf Timestamp: Block-Range-Indexe sind für append-only-Daten mit monotonen Timestamps ideal. Sie sind um Größenordnungen kleiner als B-Tree-Indexe und liefern für Range-Queries („zeige alle Mutationen zwischen 1.3. und 31.3.") nahezu identische Performance.

Archivierung: Nach 24 Monaten werden Partitionen in einen S3-kompatiblen Object Store ausgelagert. Damit auch dort die Unveränderbarkeit gewahrt bleibt, kommt eine **Hash-Verkettung** zum Einsatz: jede archivierte Partition trägt einen SHA-256-Hash über ihren Inhalt plus den Hash der Vorgänger-Partition. Manipulation an einer alten Partition würde alle nachfolgenden Hashes invalidieren. Dieselbe Idee taucht im GoBD-konformen ZIP-Daten-Export wieder auf, wo der ZIP-Bundle-Hash die Integrität des Exports gegenüber dem Prüfer belegt.

Forensik: Wo der Audit-Trail Geld spart

Audit-Trails klingen abstrakt, bis der erste Streitfall kommt. Drei reale Beispiele aus der Verwaltungspraxis zeigen, wo das System sein Geld einspielt.

Fall 1 — Sollstellungs-Differenz: Ein Eigentümer reklamiert eine erhöhte Sollstellung. Der Audit zeigt, dass der Verwalter sie am 14.3. um 11:42 mit der Begründung „Beschluss ETV 2024 TOP 5" eingetragen hat. Die Begründung verlinkt direkt auf das Beschluss-Protokoll. Der Streitfall ist nach drei Minuten erledigt.

Fall 2 — Heizkosten-Korrektur: Ein Mieter klagt eine vermeintliche nachträgliche Änderung an. Der Audit zeigt, dass der Originalwert nie geändert wurde, sondern dass eine Stornobuchung mit Begründung erfasst wurde. Das ist GoBD-konform und vor Gericht belastbar — eine direkte Korrektur am ursprünglichen Wert wäre es nicht gewesen.

Fall 3 — DSGVO-Auskunft: Ein ehemaliger Mieter verlangt nach Art. 15 Auskunft, welche Daten zu seiner Person gespeichert sind und ob sie verändert wurden. Der Audit-Trail liefert die vollständige Mutationshistorie seiner Stammdaten und die Liste der zugreifenden Mitarbeiter. Ohne Audit wäre die Auskunft nicht erteilbar — und damit selbst ein DSGVO-Verstoß.

Forensik beginnt mit Vollständigkeit. Ein Audit-Trail mit Lücken ist im Streitfall schlechter als gar keiner, weil er den Verdacht erweckt, gezielt geleert worden zu sein.

Wie ImmoGenio das umsetzt

Die konkrete Implementierung in ImmoGenio besteht aus fünf Bausteinen, die zeitlich gestaffelt entstanden sind.

1. **Tabelle** `audit_log` in der Baseline-Migration (000): Schema mit `id`, `tenant_id`, `actor_user_id`, `actor_kind`, `target_table`, `target_id`, `operation`, `field_diff` (JSONB), `request_context` (JSONB), `created_at` (UTC).
2. **Trigger-Flush-Funktion** in Migration 001: generische `pl/pgsql`-Funktion, die je Mutation den Diff berechnet, sensible Felder maskiert und einen Audit-Eintrag schreibt. Pro fachlicher Tabelle wird der Trigger über `CREATE TRIGGER ... AFTER INSERT OR UPDATE OR DELETE` registriert.
3. **Permission-Updates** in Migration 002: Der App-User erhält ausschließlich INSERT-Rechte auf `audit_log`. UPDATE und DELETE bleiben dem DBA-Account vorbehalten.
4. **Drop-no-update-Rule** in Migration 003: Rule, die UPDATEs gegen die Tabelle ins Leere laufen lässt.
5. **Drop-no-delete-Rule** in Migration 054: Analoge Rule für DELETE — kam später hinzu, als wir entschieden haben, das Härtungsmodell konsequent zu Ende zu denken.

Ergänzt wird das durch den Service `auditAlerting.service.ts`, der über das Audit-Log Anomalie-Muster erkennt: ungewöhnlich viele Mutationen pro Sekunde, Mutationen außerhalb der Geschäftszeiten, Service-Account-Aktivität auf personenbezogenen Daten. Die Alarme gehen an den Tenant-Admin und werden selbst wieder geloggt.

Verbindungen zu anderen Bausteinen

Der Audit-Trail steht nicht für sich allein. Er hängt mit mehreren anderen Compliance-Bausteinen zusammen.

- **Mandantentrennung**: Das Audit-Log selbst unterliegt dem RLS-Modell für Multi-Tenancy. Ein Tenant-Admin sieht nur die Mutationen seines Mandanten, nie die anderer Verwaltungen.
- **Aufbewahrungs-Workflow**: Der Audit-Log fällt unter eigene Aufbewahrungsfristen — zehn Jahre nach AO § 147, parallel HGB § 257. Details im Beitrag zu Aufbewahrungsfristen.
- **GoBD-Export**: Der ZIP-Datenexport für die Betriebsprüfung enthält den Audit-Log als Pflichtbestandteil — beschrieben im GoBD-ZIP-Export.
- **DSGVO-Auskunft und Offboarding**: Der Audit-Trail ist die Grundlage dafür, dass User-Offboarding nach Art. 17 DSGVO sauber funktioniert — gelöscht wird der User, sein Audit-Trail bleibt anonymisiert für die Aufbewahrungsfrist erhalten.
- **DATEV-Schnittstelle**: Auch jede Mutation an Buchungssätzen wird auditiert, bevor sie über den DATEV-Export im Format 7 CSV-ZIP-Belegbundle den Steuerberater erreicht.
- **E-Rechnung**: Die In-Flight-Swap-Logik bei ZUGFeRD und XRechnung protokolliert jeden Format-Wechsel im Audit, damit später nachvollziehbar bleibt, welche Variante wann verschickt wurde.

Was bewusst nicht gebaut wird

Drei Dinge gehören explizit nicht in den Audit-Trail, und das ist eine bewusste Entscheidung.

Erstens: kein **Time-Machine-Restore** auf alte Stände. Der Audit dokumentiert, was geändert wurde, aber er ist kein Rollback-Mechanismus. Rollbacks gehören in Stornobuchungen mit eigener Begründung, nicht in „Klick-zurück"-Logik. Eine UI, mit der sich vergangene Zustände einfach wiederherstellen lassen, untergräbt die Beweiskraft des Logs.

Zweitens: kein **Audit-Log für Lese-Operationen** im Standardfall. Jede SELECT-Abfrage zu protokollieren würde die Datenbank um den Faktor 5 bis 10 belasten und das Audit-Volumen unbrauchbar machen. Lese-Audits werden gezielt für besonders sensible Tabellen aktiviert (etwa Personalausweis-Daten), niemals flächendeckend.

Drittens: kein **Audit-Log über die Anwendung**. Die Versuchung, in jedem Service-Aufruf manuell zu loggen, hat ImmoGenio konsequent zurückgewiesen. Der Trigger ist die einzige Quelle, sonst entstehen widersprüchliche Einträge.

Wo wir stehen

Audit-Trail ist in ImmoGenio produktiv. Tabelle, Trigger, Rules und Privilegien sind seit der Baseline aktiv. Anomalie-Alerting läuft. Die Partitionierung wird mit dem nächsten Wartungsfenster auf Monatsschnitt umgestellt, sobald die Datenmenge das rechtfertigt. Die Archivierung in S3 mit Hash-Verkettung ist konzipiert und wird produktiv ausgerollt, sobald die ersten Tenants die 24-Monats-Grenze erreichen.

Was bleibt, ist die kontinuierliche Pflege: jede neue fachliche Tabelle bekommt automatisch ihren Trigger, jede neue Sensibilitäts-Klassifizierung wird in die Maskierungs-Liste übernommen, jede neue Service-Identität wird sauber als `actor_kind = 'service'` markiert. Der Audit-Trail ist nicht „fertig" — er wächst mit der Anwendung mit, aber sein Fundament bleibt unverändert.

Fragen, Rückmeldungen oder eigene Erfahrungen

Wenn Sie in Ihrer Verwaltung mit ähnlichen Anforderungen umgehen, eigene Lösungen entwickelt haben oder konkrete Detailfragen zur PostgreSQL-Umsetzung haben, freuen wir uns über einen Austausch unter kontakt@immogenio.de. Audit-Trail ist ein Thema, bei dem die Praxis viel von der gegenseitigen Erfahrung lebt — und in dem jedes Detail zählt, weil im Streitfall genau dieses Detail über den Ausgang entscheidet.