Think BIGGER

DEEP DIVE FÜR FACHLEUTE

SAP HANA

Architektur, Betrieb, Performance und Sicherheit in der Praxis

Autor	Erhard Rainer
Veröffentlicht	04.04.2026
Aktualisiert	07.07.2026
Umfang	ca. 30 Min. Lesezeit
Online	www.thinkbigger.at/wissen/sap-hana/

thinkbigger.at

Business Intelligence · Machine Learning · KI-Agenten · Schulungen

Erhard Rainer · Schönaugasse 106 · 8020 Graz · info@think-bigger.at

Inhalt

0. Überblick (für Eilige) 2
1. Einordnung: Wann SAP HANA die richtige Wahl ist 4
2. Architektur und Kernkonzepte 6
3. Datenmodellierung in der Praxis 9
4. SQL, SQLScript und Modelling 12
5. Verfügbarkeit, Backup und Recovery 15
6. Betrieb, Monitoring und Troubleshooting 17
7. Performance, Skalierung und Kostenkontrolle 20
8. Sicherheit, Governance und Compliance 22
9. Weiterführende Links und Uebungsaufgaben 25

0. Überblick (für Eilige)

SAP HANA ist eine In-Memory-Datenbankplattform mit starkem column-store-orientiertem Kern, hoher analytischer Leistung und enger Verzahnung mit SAP-nahen Anwendungs- und Datenarchitekturen. Die Plattform ist jedoch nicht nur "schnelles Reporting im RAM", sondern ein vollwertiges Produktionssystem mit Persistenz, Recovery, Hochverfuegbarkeit, Multi-Tenancy, Sicherheitsmechanismen und eigenen Betriebsanforderungen. Diese Seite ist eine praxisorientierte Referenz für Architektur, Datenmodellierung, SQL/SQLScript, Betrieb, Recovery, Performance und Governance. Ziel ist nicht nur "was ist SAP HANA", sondern "wie setze und betreibe ich SAP HANA so, dass das System fachlich schnell und operativ stabil bleibt".

Im Artikelgraphen ist SAP HANA der Datenbank- und Plattform-Layer zum SAP-Hub: Der Hub ordnet SAP als Produktfamilie ein, diese Referenz vertieft HANA als technische Produktionsplattform.

> Hinweis zum Versionsstand: Diese Referenz ist fachlich auf Stand **11. April 2026**. Auf dem SAP Help Portal waren zu diesem Zeitpunkt die Leitdokumente für **SAP HANA Platform 2.0 SPS 08** sichtbar. Die Dokumentation für **SAP HANA Cloud** lief parallel separat und zeigte auf relevanten Seiten bereits **QRC 1/2026**. Versionssensitive Aussagen in dieser Referenz sind deshalb als `2.0 SPS 08` bzw. als Cloud-Hinweis markiert.

0.1. Inhaltsverzeichnis (kurz)

- 0. Überblick (für Eilige)
- 1. Einordnung: Wann SAP HANA die richtige Wahl ist
- 2. Architektur und Kernkonzepte
- 3. Datenmodellierung in der Praxis
- 4. SQL, SQLScript und Modelling
- 5. Verfügbarkeit, Backup und Recovery
- 6. Betrieb, Monitoring und Troubleshooting
- 7. Performance, Skalierung und Kostenkontrolle
- 8. Sicherheit, Governance und Compliance
- 9. Weiterführende Links und Uebungsaufgaben

0.2. Quick-Glossar

- **Column Store**: Spaltenorientierte Speicherung, in HANA für viele Workloads der Standard.
- **Row Store**: Zeilenorientierte Speicherung für spezielle operative oder technische Tabellen.
- **Main Store**: Komprimierter, leseoptimierter Bereich einer Column-Store-Tabelle.
- **Delta Store**: Schreiboptimierter Bereich für neue und geänderte Daten.
- **Delta Merge**: Hintergrundprozess, der Delta-Daten in den Main Store ueberfuehrt.
- **Savepoint**: Regelmäßiger Persistenzpunkt, an dem geänderte Daten auf Storage geschrieben werden.
- **Redo Log**: Transaktionslog für durable Commits und Recovery.
- **MDC**: Multitenant Database Containers, also System Database plus Tenant Databases.
- **NSE**: Native Storage Extension für page-loadable, also "warmes" Datenhalten.
- **System Replication**: Replikationsmechanismus für Hochverfuegbarkeit und Disaster Recovery.
- **Workload Class**: Regelwerk für Prioritaeten, Thread- und Memory-Limits pro Session oder Statement.

- **SQLScript**: HANA-spezifische Erweiterung für prozedurale Logik, Table Functions und Procedures.

0.3. Lernpfad: Vom Einstieg zur Produktion

1. Verstehe zuerst das Betriebsmodell: Column Store, Delta/Main, Savepoints, Logs, Recovery.
2. Modelliere danach bewusst für HANA: Column Store als Default, Partitionierung und Datenalterung mitdenken.
3. Nutze SQL und SQLScript gezielt, aber halte Logik so deklarativ wie möglich.
4. Definiere dann Runbooks für Backup, Recovery, Delta-Merge-Beobachtung, Memory-Engpaesse und Failover.
5. Sichere zuletzt das Setup ab: Rollenmodell, Verschluesselung, Auditing, technische Benutzer und Governance.

0.4. Mini-Projekt (durchgehend): Vertriebs- und Bestandsreporting in Echtzeit

Dieses Mini-Projekt begleitet alle Kapitel.

Szenario:

- Ein SAP-nahes Vertriebssystem liefert Verkaufs-, Kunden- und Bestandsdaten nahezu in Echtzeit.
- Operative Nutzer benötigen aktuelle Auftragsstaende und Lieferstatus.
- Management benötigt schnelle Aggregationen nach Region, Monat, Materialgruppe und Kunde.
- Das System muss auch bei Wartungsfenstern, Host-Ausfall und Recovery-Szenarien kontrollierbar bleiben.
- Ein Teil der Daten ist "heiss", ein Teil nur noch für Rueckfragen und Audits relevant.

Wir verwenden dieses Szenario fortlaufend für:

- Tabellen- und Partitionsdesign.
- Abgrenzung zwischen Column Store, Row Store und warmen Daten.
- SQLScript- und View-Entscheidungen.
- Backup-, Recovery- und System-Replication-Strategien.
- Performance-, Security- und Governance-Entscheidungen.

0.5. Profi-Heuristiken

- HANA ist stark, wenn analytische und operative Anforderungen nah beieinander liegen.
- "In-Memory" bedeutet nicht "ohne Persistenz" und schon gar nicht "ohne Backup".
- Column Store ist in HANA meist der Default, aber nicht jede Tabelle gehört dort hinein.
- Delta Merge ist kein Nebendetail, sondern Teil des Performance- und Betriebsmodells.
- Memory ist in HANA keine beilaufige Ressource, sondern ein Fuehrungsindikator.
- Partitionierung ist kein Selbstzweck, sondern dient Lastverteilung, Wartbarkeit und Scale-out.
- SQLScript hilft, wenn es Datenfluesse erklärt; zu viel imperative Logik macht Systeme schwerer wartbar.
- System Replication ersetzt kein Backup.

0.6. So setzt du es um - Checkliste

- Halte für jede zentrale Tabelle Zweck, Temperaturprofil und Zugriffsmuster schriftlich fest.
- Definiere explizit, welche Daten "heiss", "warm" und "archivnah" sind.
- Lege ein Zielbild für RPO, RTO und zulassige Degradationsmodi fest.

- Messe regelmäßig Delta-Merge-Verhalten, Memory-Druck und teure Statements.
- Dokumentiere pro Anwendung technische Benutzer, Rollen und Audit-Anforderungen.

0.7. Mentales Modell für den Alltag

Wenn du SAP HANA in Projekten sicher einsetzen willst, hilft ein einfaches mentales Modell: HANA ist kein "klassisches RDBMS mit viel RAM", sondern eine Datenplattform, die Lesegeschwindigkeit, Kompression, Persistenz und Betriebssteuerung eng miteinander verzahnt. Genau deshalb können fachlich harmlose Entscheidungen tief in den Betrieb hineinwirken: Eine unpassende Tabellenart verändert nicht nur die Speicherdarstellung, sondern beeinflusst Merge-Verhalten, Memory-Footprint und Recovery-Zeiten; eine schlecht geschnittene Partitionierung ist nicht nur ein Modellierungsproblem, sondern später ein Skalierungs- und Incident-Thema; ein nicht kontrolliertes Rollenmodell ist nicht nur Security-Risiko, sondern erschwert auch Audits, technische Analysen und den Betrieb mit mehreren Teams. In der Praxis bewaehrt sich deshalb eine Leitfrage pro Entscheidung: "Welches messbare Verhalten des Systems wird unter realem Last- oder Stoerungsprofil dadurch besser?" Wenn die Antwort unklar ist, ist die Entscheidung oft nur Gewohnheit oder Tool-Faszination. Produktionsreife HANA-Arbeit verbindet deshalb immer vier Ebenen: Datenstruktur, Laufzeitverhalten, Betriebsfaehigkeit und Governance. Genau auf diese vier Ebenen sind die folgenden Kapitel ausgerichtet.

---

1. Einordnung: Wann SAP HANA die richtige Wahl ist

1.1. Problemklassen, in denen HANA typischerweise stark ist

SAP HANA ist besonders sinnvoll, wenn mehrere Bedingungen gleichzeitig erfüllt sind:

- Hoher Bedarf an schneller analytischer Verarbeitung auf operativ nahen Daten.
- Enge Integration in SAP-Anwendungen, SAP-Datenmodelle oder SAP-nahe Plattformlandschaften.
- Relevanz von Column-Store-Verarbeitung, Kompression und Aggregationsleistung.
- Klare Anforderungen an Recovery, Hochverfuegbarkeit und kontrollierten Plattformbetrieb.
- Bereitschaft, Modellierung und Betrieb an HANA-spezifische Mechanismen anzupassen.

Typische Domaenen:

- Reporting und Analyse auf S/4HANA-nahen Bewegungsdaten.
- Operativ-analytische Anwendungen mit kurzen Reaktionszeiten.
- Planungs-, Forecasting- und Simulationsszenarien.
- Fachanwendungen mit komplexeren Aggregationen auf grossen Tabellen.
- Mehrmandanten- oder Plattform-Szenarien mit klar geregelter Isolation.

1.2. Wann ein klassisches RDBMS oft besser passt

Ein klassisches relationales System ist oft die bessere Wahl, wenn du primär brauchst:

- Kleine bis mittlere operative Workloads ohne besondere analytische Anforderungen.
- Geringe Betriebs- und Lizenzkomplexitaet.
- Standardisierte CRUD-Anwendungen ohne SAP-Naehe.
- Ein Team, das keine HANA-spezifische Plattformkompetenz aufbauen will.

HANA ist stark, aber nicht automatisch wirtschaftlich oder organisatorisch sinnvoll für jeden relationalen Use Case.

1.3. HANA vs. reines Data-Warehouse-Denken

HANA ist nicht einfach "nur ein Warehouse".

HANA-Staerken:

- Sehr schnelle spaltenorientierte Verarbeitung.
- Gute Kombination aus operativen und analytischen Zugriffen.
- Plattformfunktionen für Recovery, Multi-Tenancy, Replication und Security.
- Starke Einbettung in SAP-Werkzeuge und Prozesse.

Reine DWH- oder Lakehouse-Plattformen punkten oft bei:

- Extrem grossen, locker gekoppelten Analyse-Landschaften.
- Preislich sehr elastischen Cloud-Speicherarchitekturen.
- Stark datei- oder batchorientierten Datenpipelines.

Entscheidend ist nicht "modern vs. klassisch", sondern:

- Wie nah Analyse und operative Daten beieinander liegen sollen.
- Wie stark SAP-Integration, Betriebsmodelle und Governance-Anforderungen dominieren.
- Wie hoch die Bereitschaft ist, eine Plattform aktiv zu betreiben.

1.4. Harte No-Go-Faelle für HANA

- Das Projekt braucht nur eine einfache Standarddatenbank ohne Plattformaufwand.
- Fachlich gibt es keinen SAP-Kontext und keinen klaren Nutzen aus HANA-spezifischen Eigenschaften.
- Budget, Plattformbetrieb und Skill-Aufbau sind nicht abgesichert.
- Das Team verwechselt "viel RAM" mit "Tuning ist egal".
- Backup-, Recovery- und Security-Prozesse werden als späteres Thema behandelt.

> **Anti-Pattern:** "Wir nehmen HANA, weil sie schnell ist, und kuemmern uns später um Betrieb und Modellierung." Ergebnis in der Praxis: teure Plattform, unklare Verantwortungen, Memory-Druck und schwierige Incident-Arbeit.

1.5. Entscheidungsmatrix (kompakt)

Kriterium	HANA passt	HANA passt eher nicht
Analytische Last	Hoch, nah am operativen Prozess	Gering oder rein batch-orientiert
SAP-Naehe	Stark	Kaum vorhanden
Plattformbetrieb	Aktiv geplant	Minimalismus ist Ziel
Datenhaltung	Column Store, Kompression, schnelle Aggregation relevant	Standard-OLTP ohne besondere Analyseanforderung
Recovery/HA	Strukturiert und kritisch	Einfaches Single-System reicht

1.6. So setzt du es um - Checkliste

- Dokumentiere, welche fachlichen Endpunkte von HANA-Eigenschaften wirklich profitieren.
- Vergleiche mindestens eine Alternative mit demselben Last- und Betriebsprofil.
- Halte fest, ob der Nutzen aus SAP-Integration, Column Store oder Betriebsfunktionen kommt.
- Definiere ein Exit-Kriterium: Unter welchen Bedingungen wäre HANA die falsche Wahl?

1.7. Architekturentscheidungen sauber dokumentieren

Wenn HANA langfristig stabil laufen soll, braucht jede Kernentscheidung eine pruefbare Begründung:

- Warum HANA und nicht Alternative X?
- Welche Tabellen müssen heiss im Memory bleiben?
- Welche Daten können warm oder page-loadable werden?
- Wo wird mit Views gearbeitet, wo mit persistierten Tabellen?
- Welche RPO- und RTO-Ziele gelten pro Kernprozess?

Ergaenze jede Entscheidung um Betriebsfolgen:

- Welche Metrik zeigt früh, dass die Entscheidung kippt?
- Welcher Runbook-Schritt greift zuerst?
- Welche Migration oder Entlastungsstrategie existiert?

1.8. Plattform vs. Cloud bewusst unterscheiden

Für die Praxis wichtig: SAP HANA Platform und SAP HANA Cloud sind fachlich verwandt, aber betrieblich nicht identisch. Die On-Prem- bzw. Platform-Welt verlangt deutlich mehr Verantwortung für Topologie, Betriebssystem, Storage, Backup-Infrastruktur und teilweise auch Patch-Pfade. Die Cloud-Welt nimmt dir vieles ab, führt aber eigene Betriebs- und Kostenlogiken ein. Für einen Cloud-spezifischen Blick siehe die bestehende Seite SAP HANA Cloud.

---

2. Architektur und Kernkonzepte

2.1. In-Memory ja, aber mit voller Persistenz

HANA hält den Grossteil der verarbeiteten Daten im Memory, persistiert aber trotzdem konsequent. Zwei Kernelemente sind dabei zentral:

- **Savepoints** schreiben geänderte Daten regelmäßig auf die Data Volumes.
- **Redo Logs** machen Commits durable und werden für Restart und Recovery benötigt.

Wichtig für die Praxis:

- Ein Commit wartet nicht darauf, dass komplette Datenstrukturen neu geschrieben werden.
- Nach Stoerungen wird der letzte konsistente Savepoint mit den nachfolgenden Log-Eintraegen zusammengesetzt.
- Genau deshalb bleiben Log-Handling, Storage-Layout und Backup-Strategie produktionskritisch.

2.2. Column Store als Kern, Row Store als Ausnahme

Der Column Store ist in HANA für viele Geschaeftstabellen der Standard. Seine Stärken:

- Hohe Kompression.
- Schnelle Aggregationen.
- Gute Ausnutzung moderner CPU- und Memory-Architekturen.

Der Row Store bleibt sinnvoll für Spezialfaelle:

- Kleine technische Tabellen.
- Konfigurations- und Steuertabellen mit stark punktuellem Zugriff.
- Szenarien, in denen zeilenbasierte Zugriffe klar ueberwiegen und Column-Store-Vorteile kaum greifen.

Faustregel:

- **Column Store zuerst denken.**
- **Row Store bewusst begründen.**

2.3. Main Store, Delta Store und Delta Merge

Der Column Store besteht logisch aus mindestens zwei Zonen:

- **Delta Store** für neue und geänderte Daten.
- **Main Store** für komprimierte, leseoptimierte Daten.

Weil Schreibvorgaenge zunächst im Delta Store landen, ist der **Delta Merge** ein Schluesselprozess. Ohne ihn wachsen Schreibdatenbereiche, Lesepfade werden teurer und die Systemcharakteristik kippt.

Wichtig:

- Der Delta Merge ist normal und kein Fehlerfall.
- Zu viele oder schlecht getimte Merges können Last erzeugen.
- Zu wenige Merges erzeugen ebenfalls Kosten, nur zeitversetzt.

2.4. Kompression, MVCC und Leselogik

HANA kombiniert starke Kompression mit transaktionssicherer Sicht auf Daten. Für die Praxis bedeutet das:

- Grosse Tabellen profitieren massiv von passendem Datentyp- und Spalten-Design.
- Lesepfade sind nicht nur eine Frage des SQL-Textes, sondern auch der Datenorganisation.
- Versionsverwaltung und Transaktionssicht müssen bei langen Laeufen, Audits und Recovery mitgedacht werden.

2.5. MDC: System Database und Tenant Databases

In modernen HANA-Landschaften ist Multi-Tenancy normal. Ein System besteht aus:

- genau **einer System Database** für zentrale Administration,
- und **einer oder mehreren Tenant Databases** für echte Nutzdaten und Anwendungen.

Wichtig für den Alltag:

- System- und Datenbankadministration sind nicht dasselbe.
- Backups, Recovery und viele Sicherheitsentscheidungen müssen tenant-bezogen gedacht werden.
- Verantwortungen zwischen Plattformteam und Anwendungsteams sollten daran ausgerichtet sein.

2.6. Scale-up, Scale-out und Partitionierung

HANA ist traditionell stark im Scale-up, unterstützt aber auch Scale-out. Scale-out bringt Vorteile nur dann, wenn Datenmodell und Betriebsmodell dazu passen:

- grosse Tabellen müssen sauber partitioniert sein,
- Last muss sinnvoll über Hosts verteilt werden,
- Betriebs- und Recovery-Folgen müssen bekannt sein.

Praxisregel:

- Nicht "früh skaliert ist besser", sondern "nur dann verteilen, wenn es fachlich und operativ nachvollziehbar ist".

2.7. Hochverfuegbarkeit mit System Replication

Für Hochverfuegbarkeit und Disaster Recovery ist **SAP HANA System Replication** zentral. Offiziell relevante Modi sind insbesondere:

- Replikationsmodi `sync`, `syncmem` und `async`.
- Betriebsmodi `delta_datashipping`, `logreplay` und `logreplay_readaccess`.

Wichtig:

- System Replication ist keine abstrakte Checkbox, sondern Teil deiner konkreten RPO/RTO-Strategie.
- Ein repliziertes Sekundaersystem muss auch in Runbooks, Tests und Takeover-Prozessen sauber eingebettet sein.

2.8. Native Storage Extension und page-loadable Daten

Nicht jede Tabelle muss permanent voll im DRAM liegen. Mit **SAP HANA Native Storage Extension (NSE)** können page-loadable Daten warm gehalten werden. Das ist besonders nützlich, wenn:

- grosse Tabellen historisch relevant bleiben,
- aber nur ein kleiner Teil aktiv und häufig abgefragt wird,
- und Memory-Budget kontrolliert bleiben muss.

NSE ist kein billiger Ersatz für schlechtes Datenmanagement. Es ist ein Werkzeug, um Datenklassen differenziert zu behandeln.

2.9. Mini-Projekt: Architekturentscheidung

Für das Vertriebs- und Bestandsprojekt bedeutet das:

- Bewegungsdaten in Column Tables.
- Kleine Job- und Steuerungstabellen im Row Store.
- Heisse Fakten im Hauptspeicher, aeltere Monatsbereiche perspektivisch warm/page-loadable.
- Tenant-spezifische Trennung für operative und analytische Workloads nur dann, wenn Governance und Betrieb das verlangen.
- System Replication als expliziter Bestandteil des Betriebsmodells.

2.10. So setzt du es um - Checkliste

- Dokumentiere für jede zentrale Tabelle: Column oder Row und warum.
- Prüfe für grosse Tabellen früh Partitionierungs- und Lebenszyklusstrategien.

- Definiere Savepoint-, Backup- und Replication-Folgen bereits im Architekturentwurf.
- Trenne bewusst zwischen System-Database-Aufgaben und Tenant-Database-Aufgaben.
- Bewerte früh, welche Daten für NSE bzw. page-loadable überhaupt geeignet sind.

2.11. Architektur-Risiken früh erkennen

Architekturbedingte Probleme in HANA zeigen sich oft schleichend:

- wachsender Delta-Anteil,
- ungleich verteilte Partitionen,
- steigender Memory-Druck trotz unveraendertem Datenvolumen,
- lange Recovery-Zeiten, obwohl das System im Alltag "schnell" wirkt.

Frühe Warnsignale:

- immer mehr manuelle Merge-Eingriffe,
- häufige teure Statements auf denselben grossen Objekten,
- Backup-Fenster werden länger,
- Sekundaersysteme und Failover-Prozesse werden kaum noch getestet.

2.12. Architektur an SLO und Recovery-Ziele koppeln

Architektur ist erst dann belastbar, wenn sie mit fachlichen Zielen verbunden ist. Beispiel:

- SLO: Tagesaktuelle Umsatzsicht je Verkaufsorganisation in unter 3 Sekunden.
- Architekturfolge: Column Store, passende Partitionierung, begrenzter Delta-Anteil, stabile Aggregationspfade.
- Recovery-Ziel: Takeover oder Restore darf den Fachprozess nicht für einen halben Arbeitstag blockieren.

Ohne diese Kopplung bleiben Architekturdebatten Geschmacksfragen.

---

3. Datenmodellierung in der Praxis

3.1. Column Store als Modellierungs-Default

In HANA sollte man grosse fachliche Tabellen zuerst als Column-Store-Kandidaten betrachten. Gründe:

- bessere Kompression,
- effizientere Aggregationen,
- oft bessere analytische Performance,
- gute Eignung für gemischte Analyse- und Reporting-Last.

Die Entscheidung ist aber kein Dogma. Sie muss am Zugriffsmuster, an der Größe und an den Betriebsfolgen gemessen werden.

3.2. Row Store nur bewusst einsetzen

Der Row Store passt oft für:

- kleine Lookup- oder Control-Tabellen,

- technische Metadaten,
- sehr zeilenorientierte Punktzugriffe,
- Tabellen, bei denen analytische Verdichtung keine Rolle spielt.

Anti-Pattern:

- grosse Bewegungs- oder Faktentabellen im Row Store "weil Inserts einfacher wirken".

3.3. Schlüssel, Datentypen und Kompression

HANA belohnt saubere Datentypwahl. Praxisregeln:

- Datentypen nicht größer wählen als fachlich nötig.
- Kardinalitaeten, Codes und Statuswerte bewusst modellieren.
- Technische Surrogates nur dort nutzen, wo sie Abfragen, Integrität oder Ladeprozesse wirklich verbessern.
- Lange, schlecht normierte Zeichenketten in grossen Faktentabellen vermeiden, wenn sie selten benötigt werden.

Kompression ist kein Bonus "on top", sondern Teil des Designs.

3.4. Partitionierung: Betriebswerkzeug, kein Selbstzweck

Partitionierung hilft, wenn sie ein echtes Problem löst:

- sehr grosse Tabellen,
- Verteilung über mehrere Hosts,
- gezielte Pflege und Wartung,
- bessere Lastverteilung.

Typische Muster:

- Hash-Partitionierung für Verteilung.
- Range-Partitionierung für Zeit- oder Bereichslogik.
- Mehrstufige Partitionierung für komplexere Lastprofile.

Wichtig:

- Eine schlechte Partitionierung ist oft schlimmer als gar keine.
- Daten- und Abfrageverteilung müssen zu den Partitionsschluesseln passen.

3.5. Datenalterung und Temperaturprofile

Nicht alle Daten sind gleich. Ein robustes HANA-Modell unterscheidet oft:

- **heisse Daten** für tagesaktuelle operative und analytische Nutzung,
- **warme Daten** für Rueckfragen, Audits und seltenere Analysen,
- **archivnahe Daten** ausserhalb der teuren Hauptlastpfade.

Erst wenn dieses Temperaturprofil klar ist, werden Entscheidungen zu NSE, Unload-Verhalten und Aufbewahrung fachlich sauber.

3.6. Semantik in Tabellen, Views oder Calculation Views?

Ein haeufiges Architekturproblem ist nicht fehlende Funktion, sondern unklare Schichtung. Bewaehrte Leitlinien:

- Tabellen für stabile, technisch und fachlich belastbare Datenhaltung.
- Views für klar lesbare Projektion, Filterung und Kombination.
- Calculation Views oder modellierte Semantik nur dort, wo Konsum, Wiederverwendung und Governance davon profitieren.
- Persistierte Zwischenergebnisse nur dann, wenn Laufzeit, Entkopplung oder Recovery das wirklich rechtfertigen.

3.7. SQLScript bewusst dosieren

SQLScript ist maechtig, aber es sollte nicht zum Sammelbecken für jede Fachlogik werden. Gute Einsatzfelder:

- klar gekapselte Table Functions,
- prozedurale Steuerung, wenn deklaratives SQL nicht ausreicht,
- wiederverwendbare Bausteine mit sauberem Vertragsverhalten.

Warnsignal:

- grosse imperative Blobs, die fachliche Regeln, Ladepfade und Sonderfälle unlesbar vermischen.

3.8. Anti-Pattern-Kasten: Drei Klassiker

- **Universal-Faktentabelle ohne Temperaturmodell**: führt später zu Memory-, Merge- und Recovery-Problemen.
- **Partitionierung auf Hoffnung**: "Wir verteilen mal, dann wird es schneller." Ohne Lastmodell endet das oft im Blindflug.
- **Zu viel Logik in Procedures**: erschwert Testbarkeit, Transparenz und Teamarbeit.

3.9. Mini-Projekt: Schema-Grundidee

Im Vertriebsprojekt ist ein sauberes Grundmodell zum Beispiel:

- `SALES_ORDER_ITEM` als zentrale Column Table für Bewegungsdaten.
- `INVENTORY_SNAPSHOT` für periodische Bestandssichten.
- kleine technische Tabellen wie `JOB_CONTROL` im Row Store.
- semantische Views für Reporting-Pfade.
- spätere Auslagerung aelterer Monatsbereiche in warmes/page-loadable Verhalten.

3.10. So setzt du es um - Modellierungsworkflow

1. Liste die 5 bis 10 wichtigsten Abfragepfade und Berichte.
2. Ordne jedem Pfad Frequenz, Aktualitaetsbedarf und Kritikalitaet zu.
3. Leite daraus Tabellenart, Granularitaet und Temperaturprofil ab.
4. Prüfe für grosse Tabellen früh Partitionierungsoptionen.
5. Verteile Semantik bewusst zwischen Tabellen, Views und SQLScript.
6. Miss dann reale Last statt Annahmen aus PowerPoint.

3.11. So setzt du es um - Checkliste

- Jede grosse Tabelle hat eine begründete Store-Entscheidung.
- Datentypen sind auf Fachbedarf und Kompression abgestimmt.
- Partitionierung ist workload-begruendet und dokumentiert.
- Temperaturprofil und Aufbewahrungslogik sind schriftlich festgehalten.
- SQLScript hat einen klaren, begrenzten Zweck.

3.12. Migration aus klassischen SAP- oder AnyDB-Welten

Ein typischer Fehler in Migrationen ist die 1:1-Uebernahme alter Modellierungsgewohnheiten. Was vorher "gut genug" war, muss in HANA nicht automatisch gut sein. Prüfe deshalb bewusst:

- Welche Tabellen profitieren wirklich vom Column Store?
- Welche Altlogik gehört heute in Views statt in persistierte Zwischentabellen?
- Wo entstehen durch alte Schlüssel- oder Partitionsmuster neue Hotspots?
- Welche Betriebsmetriken waren früher irrelevant, sind nun aber fuehrend?

3.13. Datenqualität und Governance im Modell

Ein HANA-Modell ist nur dann robust, wenn auch Herkunft, Aktualitaet und Verantwortungen klar sind. Sinnvolle Fragen pro Objekt:

- Wer ist fachlicher Owner?
- Wie frisch müssen die Daten sein?
- Dürfen sie warm/page-loadable werden?
- Welche Rollen dürfen lesen, schreiben oder administrieren?
- Welche Audits müssen auf diesem Objekt möglich sein?

---

4. SQL, SQLScript und Modelling

4.1. Tabellen-DDL bewusst formulieren

Die Basissyntax ist vertraut, die Betriebsfolgen sind es nicht immer.

```
CREATE COLUMN TABLE SALES_ORDER_ITEM (
  MANDT NVARCHAR(3) NOT NULL,
  SALES_ORG NVARCHAR(4) NOT NULL,
  FISCAL_YEAR SMALLINT NOT NULL,
  FISCAL_MONTH TINYINT NOT NULL,
  ORDER_ID BIGINT NOT NULL,
  ITEM_ID INTEGER NOT NULL,
  CUSTOMER_ID BIGINT NOT NULL,
  MATERIAL_ID BIGINT NOT NULL,
  ORDER_TS TIMESTAMP NOT NULL,
  NET_AMOUNT DECIMAL(15,2) NOT NULL,
  CURRENCY NVARCHAR(3) NOT NULL,
  STATUS NVARCHAR(20) NOT NULL,
  PRIMARY KEY (MANDT, ORDER_ID, ITEM_ID)
);
```

Kleine technische Tabellen können dagegen bewusst als Row Table angelegt werden:

```
CREATE ROW TABLE JOB_CONTROL (
  JOB_NAME NVARCHAR(100) PRIMARY KEY,
  LAST_RUN_TS TIMESTAMP,
  LAST_STATUS NVARCHAR(30)
);
```

4.2. Partitionierung als expliziter Teil der DDL

Wenn du verteilen willst, tue es bewusst und nachvollziehbar:

```
CREATE COLUMN TABLE SALES_ORDER_ITEM_PART (
  MANDT NVARCHAR(3) NOT NULL,
  SALES_ORG NVARCHAR(4) NOT NULL,
  ORDER_ID BIGINT NOT NULL,
  ITEM_ID INTEGER NOT NULL,
  ORDER_TS TIMESTAMP NOT NULL,
  NET_AMOUNT DECIMAL(15,2) NOT NULL,
  PRIMARY KEY (MANDT, ORDER_ID, ITEM_ID)
)
PARTITION BY HASH (SALES_ORG) PARTITIONS 8;
```

Wichtig ist nicht nur die Syntax, sondern die Frage: Passt der Schlüssel zur echten Last- und Datenverteilung?

4.3. SQLScript für wiederverwendbare Logik

Ein sinnvolles Beispiel für deklarationsnahe Logik ist eine Table Function:

```
CREATE FUNCTION TF_MONTHLY_SALES (
  IN I_FISCAL_YEAR SMALLINT
)
RETURNS TABLE (
  SALES_ORG NVARCHAR(4),
  FISCAL_MONTH TINYINT,
  REVENUE DECIMAL(15,2)
)
LANGUAGE SQLSCRIPT
SQL SECURITY INVOKER
AS
BEGIN
  RETURN
    SELECT
      SALES_ORG,
      FISCAL_MONTH,
      SUM(NET_AMOUNT) AS REVENUE
    FROM SALES_ORDER_ITEM
    WHERE FISCAL_YEAR = :I_FISCAL_YEAR
    GROUP BY SALES_ORG, FISCAL_MONTH;
END;
```

Praxisregel:

- lieber kleine, saubere, testbare SQLScript-Bausteine,
- statt monolithischer Procedure-Landschaften.

4.4. Views und semantische Schichten

Nicht jede Fachsicht braucht eine physische Tabelle. Oft reichen:

- SQL Views für Projektion und Filterung,
- semantische Modellierung für Reporting und Self-Service,

- Table Functions für klar begrenzte Berechnungslogik.

Weniger Persistenz bedeutet oft:

- weniger Ladeaufwand,
- weniger Betriebsflaeche,
- weniger Recovery-Komplexitaet.

4.5. Delta Merge nur verstehen, nicht reflexhaft erzwingen

Der Delta Merge kann manuell angestossen werden, aber nur mit guter Begründung:

```
MERGE DELTA OF SALES_ORDER_ITEM;
```

Manuelle Eingriffe sind sinnvoll, wenn:

- ein geplanter Lastwechsel ansteht,
- das Verhalten sauber beobachtet wird,
- und ein Team die Merge-Folgen versteht.

Schlecht ist:

- Merge als Standardreaktion auf jedes Performanceproblem.

4.6. Monitoring-Queries, die im Alltag helfen

Teure Statements:

```
SELECT TOP 20
  START_TIME,
  DURATION_MICROSEC,
  DB_USER,
  OBJECT_NAME,
  STATEMENT_STRING
FROM SYS.M_EXPENSIVE_STATEMENTS
ORDER BY START_TIME DESC;
```

Partitionsuebersicht:

```
SELECT
  SCHEMA_NAME,
  TABLE_NAME,
  PART_ID,
  HOST,
  NODE_ID,
  PARENT_NODE_ID
FROM SYS.M_TABLE_PARTITIONS
WHERE SCHEMA_NAME = 'SALES';
```

Delta-Merge-Historie:

```
SELECT TOP 50
  HOST,
  SCHEMA_NAME,
  TABLE_NAME,
  TYPE
FROM SYS.M_DELTA_MERGE_STATISTICS;
```

4.7. Savepoint und betriebsnahe SQL-Bausteine

Ein Savepoint kann bei Bedarf manuell getriggert werden:

```
ALTER SYSTEM SAVEPOINT;
```

Auch hier gilt:

- nur mit Verständnis für Betriebsfolgen,
- nicht als blindes Ritual.

4.8. Mini-Projekt: Basisbausteine

Für das Vertriebsprojekt sind typische Bausteine:

- Column-Store-DDL für Faktentabellen.
- Row-Store-DDL für technische Steuerung.
- Views für Reporting-Sichten.
- Table Functions für wiederverwendbare Aggregationslogik.
- Monitoring-Queries für teure Statements, Partitionierung und Merge-Verhalten.

4.9. So setzt du es um - Checkliste

- Alle DDL-Entscheidungen sind fachlich und betrieblich begründet.
- SQLScript ist modular, testbar und nicht ueberdehnt.
- Monitoring-Queries für Alltag und Incident sind vorbereitet.
- Manuelle Betriebs-SQL ist dokumentiert und nicht nur Expertenwissen.

4.10. SQL- und Modelling-Review vor Release

Vor jedem größeren Release lohnt sich ein kurzer Review:

- Gibt es neue grosse Tabellen ohne Temperatur- und Partitionierungsentscheidung?
- Ist neue SQLScript-Logik deklarativ genug geblieben?
- Sind neue Reporting-Sichten fachlich sauber geschichtet?
- Sind Monitoring- und Security-Folgen mitgedacht?

---

5. Verfügbarkeit, Backup und Recovery

5.1. Persistenzmodell für Restart und Recovery verstehen

HANA persistiert über Savepoints und Redo Logs. Für die Praxis heisst das:

- Lokaler Restart nach einem Crash ist etwas anderes als Disaster Recovery.
- Schneller Alltagsbetrieb entbindet nicht von sauberer Backup- und Restore-Faehigkeit.
- Recovery-Zeiten hängen auch von Datenvolumen, Backup-Strategie und Testreife ab.

5.2. Backup-Strategie ist Pflicht, nicht Formalie

Eine belastbare Strategie kombiniert typischerweise:

- regelmäßige Datenbackups,

- automatische Logbackups,
- klare Aufbewahrung,
- Pruefmechanismen für Wiederherstellbarkeit,
- gegebenenfalls Storage-Snapshots als zusätzliche Option.

Wichtig:

- Das letzte erfolgreiche Backup ist kein Sicherheitsgefuehl, wenn Restore nie geprobt wurde.

5.3. Log Mode und der erste produktionsreife Zustand

SAP HANA kennt `log mode normal` und `log mode overwrite`. Für produktionsreife Recovery ist `normal` entscheidend. Laut Administrationsdokumentation schaltet HANA nach dem ersten vollen Datenbackup automatisch in den Standardpfad, in dem Logbackups entstehen können.

Praxisregel:

- Kein Go-Live ohne dokumentierten ersten Full Backup und pruefbaren Log-Backup-Pfad.

5.4. Backup-Katalog, Schätzungen und Plausibilisierung

Guter Betrieb schaut nicht nur auf "Backup erfolgreich", sondern auch auf:

- Backup-Katalog-Konsistenz,
- benötigten Speicherplatz,
- Alter der letzten verwertbaren Kette,
- Aufbewahrung und Löschregeln,
- Verschluesselung und Root-Key-Backups.

Gerade in HANA ist die Wiederherstellbarkeit wichtiger als das reine Vorhandensein von Dateien.

5.5. System Replication strategisch einsetzen

System Replication unterstützt schnelle Übernahme im Stoerungsfall und ist ein Kernbaustein für HA/DR. Wichtige Designfragen:

- Welcher Replikationsmodus ist fachlich nötig: `sync`, `syncmem` oder `async`?
- Welcher Betriebsmodus passt: `logreplay`, `delta_datashipping` oder `logreplay_readaccess`?
- Wo liegt der Trade-off zwischen Latenz, Datenverlust-Risiko und Kosten?

Merksatz:

- **System Replication verbessert Verfügbarkeit.**
- **Backups sichern historische Wiederherstellung.**
- Beides wird benötigt.

5.6. RPO- und RTO-Matrix (kompakt)

Ziel	Typische HANA-Massnahme
Nahezu kein Datenverlust	Replication mit strengerem Modus plus saubere Logbackups
Schnelle Übernahme	Geuebter Takeover-Prozess und aktueller Sekundaerstand
Wiederherstellung historischer Zustaende	Datenbackups plus Logbackups

Ziel	Typische HANA-Massnahme
Schutz gegen Bedienfehler oder Datenkorruption	Backup-Kette, Retention, Restore-Drills

5.7. Mini-Projekt: Ausfall-Szenario

Im Vertriebsprojekt wird folgendes Zielbild vereinbart:

- Produktivsystem mit System Replication auf Sekundaerstandort.
- Regelmäßige Datenbackups plus Logbackups.
- Geuebter Takeover-Prozess für Infrastruktur- oder Host-Ausfall.
- Separater Restore-Drill für den Fall fachlicher Datenfehler.

So wird zwischen "schnell weiterarbeiten" und "sauber zurueckholen" sauber unterschieden.

5.8. So setzt du es um - Checkliste

- Der erste Full Backup nach produktiver Inbetriebnahme ist dokumentiert.
- Logbackups werden überwacht und nicht nur vorausgesetzt.
- RPO und RTO sind pro kritischem Prozess festgelegt.
- System Replication ist getestet, nicht nur eingerichtet.
- Restore-Prozeduren sind mit echten Zeiten geprobt.

5.9. Recovery-Governance im Team

Die häufigste Schwachstelle ist nicht Technik, sondern Unklarheit:

- Wer darf ein Takeover freigeben?
- Wer entscheidet zwischen Failover und Restore?
- Welche Systeme haben Prioritaet?
- Welche Kommunikation läuft parallel?

Recovery braucht deshalb nicht nur Technik, sondern klare Entscheidungsrechte.

5.10. Restore-Drills als Reifekriterium

Ein HANA-Team ist erst dann belastbar, wenn es Wiederherstellung praktisch kann. Bewaehrte Drill-Fragen:

- Wie lange dauert ein Restore wirklich?
- Welche Schritte waren nur Expertenwissen?
- Sind Katalog, Medien, Keys und Rollen verfügbar?
- Welche Abhängigkeiten ausserhalb der Datenbank bremsen den Prozess?

---

6. Betrieb, Monitoring und Troubleshooting

6.1. Betriebsgrundsatz: HANA braucht aktives Operieren

SAP HANA ist keine Datenbank, die man nach Installation "einfach laufen lässt". Der Alltag umfasst:

- Memory- und Statement-Beobachtung,

- Delta-Merge-Verhalten,
- Backup- und Log-Ketten,
- Tenant- und Rollenbetrieb,
- gegebenenfalls Replication- und Failover-Bereitschaft.

6.2. Wichtige Werkzeuge

Die zentrale Werkzeuglandschaft umfasst typischerweise:

- **SAP HANA cockpit** für Administration und Monitoring,
- **SAP HANA database explorer** für SQL, Katalog und Analysen,
- **SQL Analyzer** für Plan- und Laufzeitverstehen,
- **hdbsql** und weitere Kommandozeilenwerkzeuge für betrieblichen Zugriff,
- **hdbnsutil** für Replication-nahe Aufgaben.

Wichtig ist nicht nur Tool-Auswahl, sondern reproduzierbare Nutzung im Team.

6.3. Kernmetriken und System Views

Bewaehrte Beobachtungspunkte:

- `M_SERVICE_MEMORY` für dienstbezogenen Memory-Druck,
- `M_EXPENSIVE_STATEMENTS` für lange oder speicherintensive SQLs,
- `M_CS_TABLES` für Column-Store-Sicht auf Tabellenverhalten,
- `M_TABLE_PARTITIONS` für Partitionszustand,
- `M_DELTA_MERGE_STATISTICS` für Merge-Historie,
- Backup-Katalog- und Backup-Groessen-Sichten für Recovery-Faehigkeit,
- Audit-Views für sicherheitsrelevante Aktivitaeten.

Die Kunst ist nicht, möglichst viele Views zu kennen, sondern die richtigen für Standardbetrieb und Incident parat zu haben.

6.4. Typische Fehlerbilder

- **Delta-Merge-Backlog**: steigende Delta-Anteile, schwankende Leselatenzen.
- **Memory-Druck**: Unloads, Verdrangung, schlechtere Reaktionszeiten.
- **Expensive Statements**: einzelne Berichte oder Abfragen dominieren Ressourcen.
- **Partitionschieflage**: ungleiche Last oder ungleicher Datenwuchs.
- **Backup-/Log-Probleme**: Ketten brechen, Aufbewahrung oder Speicherplatz werden kritisch.
- **Security-Drift**: technische Nutzer und Rechte wachsen unkontrolliert.

6.5. Reproduzierbarer Debugging-Workflow

1. Symptom sauber benennen: Latenz, Memory, Merge, Backup, Replikation oder Berechtigung?
2. Betroffene Tenant Database, Host(s) und Zeitfenster eingrenzen.
3. System Views und Cockpit-Daten gegeneinander plausibilisieren.
4. Nicht nur SQL, sondern auch Objektgroesse, Partitionierung und Delta-Zustand prüfen.
5. Wirkung von Massnahmen messen und dokumentieren.

Das verhindert Aktionismus.

6.6. Runbook für Memory-Druck

Wenn Memory knapp wird, sind typische Fragen:

- Ist das Problem global oder tenant-/service-spezifisch?
- Sind neue teure Statements oder neue Objekte beteiligt?
- Wachsen Delta- oder page-loadable Anteile unerwartet?
- Liegt das Problem an Datenwuchs, Query-Pattern, Merge-Verhalten oder Ressourcensteuerung?

Gute Teams reagieren nicht mit "mehr RAM" als erster Antwort, sondern mit Diagnose.

6.7. Backup- und Recovery-Monitoring in den Alltag integrieren

Backups sind kein separater Silo-Prozess. Sie gehören in das tägliche Monitoring:

- Alter der letzten erfolgreichen Daten- und Logbackups.
- Plausibilitaet des Backup-Katalogs.
- Restore-Faehigkeit inklusive Schluesselmaterial.
- Freier Speicher im Backup-Ziel.

6.8. Mini-Projekt: Incident "Monatsabschlussbericht läuft aus dem Ruder"

Im Vertriebsprojekt steigt zum Monatsende die Laufzeit eines Management-Reports stark an. Systematischer Weg:

- `M_EXPENSIVE_STATEMENTS` prüfen.
- betroffene Tabellen in `M_CS_TABLES` und Partitionssicht bewerten.
- Delta-Merge-Historie und Memory-Druck abgleichen.
- prüfen, ob neuer Datenumfang, neues SQL oder veränderte Nutzung ursachlich ist.
- erst dann SQL, Modell oder Betriebsparameter anpassen.

6.9. So setzt du es um - Checkliste

- Ein Kernset an Views und Cockpit-Screens ist als Standard-Runbook dokumentiert.
- Backup- und Recovery-Faehigkeit wird täglich oder mindestens sehr regelmäßig geprüft.
- Expensive Statements werden aktiv verfolgt und nicht erst im Incident angesehen.
- Partitions- und Merge-Entwicklung sind für grosse Tabellen sichtbar gemacht.

6.10. Runbook-Vorlage pro Kerntabelle

Eine einfache, wirksame Vorlage:

- Tabellenname und fachlicher Zweck.
- Column oder Row Store und Begründung.
- Temperaturprofil und Aufbewahrung.
- Partitionierung und Host-Verteilung.
- Typische Abfragen.
- Erwartete Merge- und Memory-Charakteristik.
- Relevante Rollen und Security-Anforderungen.
- Erste 5 Diagnose-Schritte bei Problemen.

6.11. Post-Incident-Lernen

Nach jedem größeren Vorfall lohnt sich eine kurze technische Nachbereitung:

1. Was war der erste messbare Indikator?
2. Welche View oder welches Tool war wirklich hilfreich?
3. Welche Architekturannahme war zu optimistisch?
4. Welche Metrik oder welches Runbook fehlt noch?
5. Welche konkrete Änderung folgt?

---

7. Performance, Skalierung und Kostenkontrolle

7.1. Memory ist die fuehrende Ressource

In HANA ist Memory nicht nur eine Infrastrukturkennzahl, sondern Kern der Architektur. Performancearbeit bedeutet deshalb fast immer auch:

- Datenvolumen verstehen,
- Datenklassen unterscheiden,
- unnötige heisse Daten vermeiden,
- Statement-Spitzen kontrollieren.

7.2. Delta Merge als Performancehebel

Weil Writes im Delta landen, ist das Merge-Verhalten direkt performancewirksam. Zu grosse Delta-Anteile verschlechtern Lesepfade. Zu aggressive Merges belasten CPU und Systemressourcen.

Merksatz:

- Gute HANA-Performance ist oft gutes Merge-Management plus gutes Datenmodell.

7.3. Partitionierung und Platzierung

Skalierung in HANA ist nie nur "mehr Hosts". Fragen vor jedem Scale-out:

- Welche Tabellen treiben Last und Größe?
- Passen Partitionsschluessel zur Verteilung?
- Entsteht echter Parallelismus oder nur mehr Koordination?
- Wie wirken sich neue Hosts auf Backup, Recovery und Betrieb aus?

7.4. Expensive Statements und Plananalyse

Viele Performanceprobleme lassen sich auf wenige SQLs zurueckfuehren. Deshalb:

- teure Statements identifizieren,
- Ausfuehrungsplaene verstehen,
- Statement-Memory und CPU gemeinsam betrachten,
- nicht nur den SQL-Text, sondern auch Objektgroessen und Datenlage einbeziehen.

7.5. Workload Classes als Steuerungsinstrument

Mit Workload Classes lassen sich Prioritaeten sowie Thread- und Memory-Limits auf Session- oder Statement-Ebene setzen. Das ist besonders hilfreich, wenn:

- Fachabfragen und Betriebsprozesse konkurrieren,
- Self-Service-Reporting produktionsnahe Systeme belastet,
- einzelne Anwendungen zu dominant werden.

Workload Classes sind kein Ersatz für saubere Modellierung, aber ein starkes Governance-Werkzeug.

7.6. Warme Daten mit NSE und page-loadable Profilen

Wenn Daten selten, aber nicht nie gebraucht werden, kann page-loadable Verhalten ein sinnvoller Mittelweg sein. Typische Nutzungsfaelle:

- historische Bewegungsdaten,
- Audit- und Rueckfragebereiche,
- grosse Tabellen mit kleinem aktivem Teil.

Wichtig:

- Nicht blind Daten "kalt" machen.
- Erst Nutzung, SLA und Restore-Folgen verstehen.

7.7. Kostenkontrolle ohne Qualitaetsverlust

Kostenkontrolle gelingt oft durch Datenklassifizierung:

- **Tier A**: geschaeftskritisch, heiss, streng überwacht.
- **Tier B**: operativ relevant, aber mit etwas mehr Spielraum.
- **Tier C**: selten genutzte Historie oder audit-naher Bereich, warm/page-loadable denkbar.

Das reduziert Memory-Druck und Plattformkosten, ohne Kernpfade zu gefaehrden.

7.8. Mini-Projekt: Tuning-Loop

1. Ausgangsmessung für Monats- und Tageslast erfassen.
2. Top-Statements, größte Tabellen und Merge-Verhalten ermitteln.
3. Tabellenarten, Partitionierung und Views ueberpruefen.
4. Gegebenenfalls Workload Classes und Temperaturprofil scharfziehen.
5. Erneut messen und nicht nur "gefuehlt schneller" bewerten.

7.9. So setzt du es um - Checkliste

- Memory ist als Fuehrungsmetrik pro Tenant und kritischem Service sichtbar.
- Merge-Verhalten grosser Tabellen wird aktiv beobachtet.
- Teure Statements haben Owner und Follow-up-Prozess.
- Scale-out-Entscheidungen sind workload-begruendet.
- Datenklassifizierung ist technisch und organisatorisch verankert.

7.10. Benchmarking, das wirklich hilft

Gute HANA-Benchmarks bilden nach:

- reale Datenvolumina,
- echte SQL-Muster,
- Peak- und Parallelitaetsphasen,
- Memory- und Statement-Limits,
- Recovery- und Betriebsfolgen größerer Datenmengen.

Schlechte Benchmarks messen nur leere Demos.

7.11. Kapazitaetsplanung für HANA

Kapazitaetsplanung umfasst mehr als Netto-Datenmenge:

- aktives vs. warmes Datenvolumen,
- Delta- und Main-Charakteristik,
- Wachstumsrate,
- Backup-Fenster,
- Replication-Ziele,
- Reserve für Monatsspitzen, Audits und Datenneuladungen.

7.12. Performance-Regressionen früh stoppen

Ein guter Release-Prozess hat klare Stop-Kriterien:

- Statement-Laufzeit verschlechtert sich signifikant.
- Peak-Memory steigt unerklaert.
- Delta-Merge-Verhalten kippt.
- Recovery-Fenster oder Backup-Groessen laufen aus dem Rahmen.

---

8. Sicherheit, Governance und Compliance

8.1. Sicherheitsgrundsatz

Eine funktionierende HANA-Installation ist noch kein sicheres HANA-System. Mindestens nötig sind:

- sauberes Rollen- und Privilegienmodell,
- gesicherte Kommunikation,
- Schutz ruhender Daten,
- Auditing,
- klare Trennung technischer und administrativer Verantwortungen.

8.2. Rollen und Privilegien

Praxisregeln:

- Keine Shared-Admin-Konten für Anwendungen.
- Technische Benutzer pro Dienst oder Integrationspfad.
- Least Privilege auf Datenbank-, Schema- und Objekt-Ebene.
- Administrative Rechte nur bewusst und nachweisbar.

Der `SYSTEM`-Benutzer ist kein Alltagskonto für Teams.

8.3. Authentifizierung und zentrale Identität

Je nach Landschaft sind z. B. folgende Themen relevant:

- lokale Benutzer nur dort, wo sinnvoll,
- LDAP- oder zentrale Verzeichnisanbindung,
- SAML für Single Sign-On,
- technische Konten für Maschinenzugriff.

Wichtig ist nicht die Zahl der Optionen, sondern ein konsistenter Identity-Ansatz.

8.4. Verschluesselung in Transit und at Rest

Für produktionsreife Umgebungen sind typischerweise zu betrachten:

- TLS für Client-Verbindungen,
- sichere interne Kommunikation,
- Datenvolumen-Verschluesselung,
- Redo-Log-Verschluesselung,
- Backup-Verschluesselung.

Wichtig:

- Root Keys gehören in das Betriebs- und Backup-Konzept.
- Verschluesselung ohne Key-Management-Prozess erzeugt Scheinsicherheit.

8.5. Auditing und Audit Trails

Mit Audit Policies lassen sich relevante Aktionen gezielt protokollieren. Praktische Beispiele:

- Benutzer- und Rollenverwaltung,
- Backup-relevante Aktionen,
- sicherheitskritische Konfigurationsaenderungen,
- Zugriffe auf sensible Objekte.

Die Security-Dokumentation nennt für produktive Systeme u. a. Audit-Trails in Datenbanktabelle oder SYSLOG. Die Audit-Views `AUDIT_LOG`, `XSA_AUDIT_LOG` und `ALL_AUDIT_LOG` sind hier operativ relevant.

8.6. Trennung technischer Rollen

Ein robustes Modell trennt:

- Plattformadministration,
- Datenbankadministration,
- Anwendungsbetrieb,
- Entwicklung,
- Auditing bzw. Security Review.

Damit wird Security nicht nur dokumentiert, sondern im Tagesbetrieb durchgesetzt.

8.7. Compliance-Basis

Für regulierte Umgebungen brauchst du mindestens:

- dokumentierte Rollen und Rezertifizierung,

- auditierbare kritische Aktionen,
- Schutz von Backups und Schluesselmaterial,
- nachvollziehbare Restore- und Incident-Prozesse,
- dokumentierte Verantwortungen pro Tenant und System.

8.8. So setzt du es um - Checkliste

- Technische Benutzer sind eindeutig und aufgabenspezifisch.
- Privilegien folgen Least Privilege.
- Verschluesselung für Daten, Log und Backup ist bewertet und umgesetzt, wo erforderlich.
- Audit Policies für kritische Aktionen sind aktiv.
- Root-Key- und Backup-Schutz sind dokumentiert und getestet.

8.9. Rollenrezertifizierung und Drift-Kontrolle

Ein häufiger Langzeitschaden ist Rechtewachstum. Bewaehrte Praxis:

- quartalsweise Rezertifizierung,
- Reports über hochprivilegierte Konten,
- Ablaufdaten für Ausnahmeberechtigungen,
- Review technischer Benutzer bei jeder Architektur- oder Schnittstellenaenderung.

8.10. Security-Incident-Drills

Wie bei Recovery-Drills sollten auch Security-Drills geuebt werden:

- kompromittiertes technisches Konto,
- versehentlich ueberprivilegierte Rolle,
- fehlgeschlagene oder unerwartete Audit-Ereignisse,
- Umgang mit verschluesselten Backups und Key-Rotation.

8.11. Governance-Metriken

Sinnvolle Kennzahlen:

- Anzahl technischer Benutzer mit hohem Privilegiengrad,
- Anteil rezertifizierter Rollen,
- Audit-Abdeckung kritischer Aktionen,
- Anzahl offener Security-Ausnahmen,
- Alter der letzten erfolgreichen Key- und Restore-Pruefung.

8.12. Least Privilege technisch durchsetzen

Least Privilege bleibt nur stabil, wenn es technisch verankert wird:

- Rollenveraenderungen nur über geregelte Freigabe,
- kein Wildwuchs per Direktzugriff,
- automatische Reports über Drift,
- Tests für neue Rollen in Deployments oder Betriebschecks.

---

9. Weiterführende Links und Uebungsaufgaben

9.1. Offizielle Kernquellen (SAP)

Versions- und Einstiegspunkte:

- SAP HANA Administration Guide
- What's New in SAP HANA Platform 2.0 SPS 08
- SAP HANA SQL Reference Guide
- SAP HANA Troubleshooting and Performance Analysis Guide
- SAP HANA Security Guide
- SAP HANA System Replication Guide

Architektur und Betrieb:

- Savepoints and Redo Logs
- The Delta Merge Operation
- System Replication
- Managing Workload with Workload Classes
- SAP HANA Native Storage Extension

Monitoring und Security:

- M_EXPENSIVE_STATEMENTS System View
- M_DELTA_MERGE_STATISTICS System View
- M_TABLE_PARTITIONS System View
- Audit Policies
- Audit Trails

Cloud-Kontext:

- SAP HANA Cloud Administration Guide

Verwandte Seiten auf dieser Website:

- Datenbanken
- SAP HANA Cloud
- SQL Abfragen gegen die HANA Datenbank

9.2. Uebungsaufgaben (steigende Schwierigkeit)

1. **Store-Entscheidung (leicht)**

- Lege für 10 Tabellen fest, ob Column oder Row Store sinnvoll ist.
- Begruende jede Entscheidung technisch und fachlich.

2. **Temperaturprofil (leicht bis mittel)**

- Ordne Datenobjekte in heiss, warm und archivnah ein.
- Leite daraus erste NSE- oder Aufbewahrungsideen ab.

3. **Delta-Merge-Analyse (mittel)**

- Analysiere für eine grosse Column Table das Merge-Verhalten.

- Dokumentiere, wann manuelle Eingriffe sinnvoll wären und wann nicht.

4. **Partitionsreview (mittel)**

- Entwirf eine Hash- oder Range-Partitionierungsstrategie für eine grosse Faktentabelle.
- Prüfe, ob das Modell zu Lastverteilung und Betriebszielen passt.

5. **Recovery-Drill (mittel bis schwer)**

- Definiere RPO und RTO für zwei Fachprozesse.
- Leite Backup-, Log- und System-Replication-Massnahmen ab.

6. **Monitoring-Paket (schwer)**

- Baue ein Kernset aus Views, Cockpit-Sichten und Alarmen für HANA auf.
- Decke Memory, Expensive Statements, Merge, Backup und Security ab.

7. **Security-Hardening (schwer)**

- Entwerfe Rollen für Admin, Entwickler, Reporting und technische Konten.
- Definiere Audit Policies für kritische Aktionen.
- Prüfe Backup- und Key-Schutz mit.

9.3. Abnahmekriterien für "produktionsreif"

- Tabellenarten, Temperaturprofile und Partitionslogik sind dokumentiert.
- Backup, Logbackups und Restore sind nachweisbar funktionsfaehig.
- System Replication ist geprüft, wenn sie als HA-Massnahme vorgesehen ist.
- Memory-, Statement- und Merge-Metriken sind sichtbar und operationalisiert.
- Rollenmodell, Auditing und Schluesselprozesse sind etabliert.

9.4. Abschluss: Was "Pro-Level HANA" in der Praxis bedeutet

Pro-Level HANA heisst nicht, jede Admin-Ansicht auswendig zu kennen. Pro-Level HANA heisst:

- du verstehst Persistenz, Merge und Memory als zusammenhaengendes System,
- du modellierst Daten nicht nur fachlich, sondern auch betrieblich sinnvoll,
- du kannst Performanceprobleme reproduzierbar eingrenzen,
- du trennst Backup, Recovery und Replication sauber,
- du machst Sicherheit und Governance technisch durchsetzbar.

Wenn diese Punkte erfüllt sind, ist SAP HANA nicht nur schnell, sondern auch als Produktionsplattform verlaesslich.

9.5. So setzt du es um - Checkliste

- Lege für die nächsten 90 Tage einen HANA-Verbesserungsplan fest.
- Starte mit Backup/Recovery, Monitoring und Rollenmodell, falls dort Luecken bestehen.
- Ziehe danach Modellierung, Temperaturprofil und Workload-Steuerung nach.
- Wiederhole die Bewertung quartalsweise auf Basis realer Last und echter Incidents.

Über den Autor

Erhard Rainer berät kleine und mittlere Unternehmen in Österreich zu Business Intelligence, Machine Learning und KI-Agenten. Er hat 15 Jahre Erfahrung in Data Analytics, davon 10 Jahre in der Leitung von Teams und Departments in Konzernen, und ist Dozent für betriebswirtschaftliche und Data-Analytics-Fächer.

Sprechen Sie mit mir über Ihre Daten: Das Erstgespräch ist kostenlos.

https://www.thinkbigger.at/contact.html#anfrage

© 2026 Think BIGGER – Erhard Rainer. Dieses PDF gibt den Stand des Online-Artikels vom 07.07.2026 wieder; maßgeblich ist die Online-Fassung unter https://www.thinkbigger.at/wissen/sap-hana/