Think BIGGER

DEEP DIVE FÜR FACHLEUTE

IBM Db2

Relationale Unternehmensdatenbank für transaktionale und analytische Workloads

Autor	Erhard Rainer
Veröffentlicht	11.04.2026
Aktualisiert	06.07.2026
Umfang	ca. 33 Min. Lesezeit
Online	www.thinkbigger.at/wissen/ibm-db2/

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 IBM Db2 die richtige Wahl ist 4
2. Architektur & Kernkonzepte 6
3. Datenmodellierung in der Praxis 9
4. SQL, Utilities und systematische Praxisreferenz 12
5. Transaktionen, Isolation, Recovery und Verfügbarkeit 16
6. Betrieb, Monitoring & Troubleshooting 19
7. Performance, Skalierung & Kostenkontrolle 22
8. Sicherheit, Governance & Compliance 24
9. Weiterführende Links + Uebungsaufgaben 27

0. Überblick (für Eilige)

IBM Db2 ist ein relationales Enterprise-Datenbanksystem mit starkem Fokus auf stabile OLTP-Workloads, saubere Recovery-Pfade, ausgereifte Governance, analytische Zusatzfaehigkeiten und langlaufende Kernsysteme, in denen Betriebssicherheit wichtiger ist als modische Einfachheit. Diese Seite ist eine praxisorientierte Referenz für Architektur, Datenmodellierung, SQL, Utilities, Verfügbarkeit, Betrieb, Performance und Sicherheit. Der Fokus liegt auf `Db2 for Linux, UNIX and Windows (LUW)`, weil dort Themen wie `HADR`, `pureScale`, `BLU`, `RUNSTATS`, `REORG`, `package cache` und `db2pd` zusammenkommen.

> Hinweis zum Versionsstand: Diese Referenz ist fachlich auf Stand **11. April 2026**. Der IBM-Support-Artikel `Download Db2 fix packs by version for Db2 for Linux, UNIX and Windows` war zuletzt am **5. März 2026** geändert und listete zu diesem Zeitpunkt u. a. `Db2 12.1 Mod 4 Fix Pack 0` sowie für `Db2 11.5` das als final markierte `Mod 9 Fix Pack 0`. Auf derselben IBM-Seite war für `Db2 11.5` ein End-of-Support-Datum **30. April 2027** angegeben. Versionssensitive Aussagen sind als `12.1.x` bzw. `11.5.x` markiert.

0.1. Inhaltsverzeichnis (kurz)

- 0. Überblick (für Eilige)
- 1. Einordnung: Wann IBM Db2 die richtige Wahl ist
- 2. Architektur & Kernkonzepte
- 3. Datenmodellierung in der Praxis
- 4. SQL, Utilities und systematische Praxisreferenz
- 5. Transaktionen, Isolation, Recovery und Verfügbarkeit
- 6. Betrieb, Monitoring & Troubleshooting
- 7. Performance, Skalierung & Kostenkontrolle
- 8. Sicherheit, Governance & Compliance
- 9. Weiterführende Links + Uebungsaufgaben

0.2. Quick-Glossar

- **Instance**: Laufzeitumgebung des Db2-Servers mit eigener Konfiguration und Prozessen.
- **Database**: Logische Db2-Datenbank mit eigenem Katalog, Logging und Storage-Kontext.
- **Bufferpool**: Hauptspeicherbereich für Datenseiten einer bestimmten Seitengroesse.
- **Tablespace**: Speichercontainer für Tabellen, Indizes, LOBs oder Temporaerdaten.
- **Package Cache**: Cache für kompilierte SQL-Zugriffsplaene und Statement-Metriken.
- **STMM**: Self-Tuning Memory Manager für automatisches Memory-Tuning.
- **RUNSTATS**: Statistikaktualisierung für den Optimizer.
- **REORG**: Reorganisation von Tabellen oder Indizes zur Wiederherstellung guenstiger physischer Struktur.
- **HADR**: High Availability Disaster Recovery, Primar-/Standby-Replikation für Db2 LUW.
- **pureScale**: Db2-Clusterarchitektur für hohe Verfügbarkeit und horizontale Skalierung.
- **DPF**: Database Partitioning Feature für verteilte, datenpartitionierte Workloads.
- **BLU / column-organized**: Spaltenorientierte Speicherung für analytische oder hybrid-analytische Lasten.
- **Rollforward**: Recovery aus Backup plus archivierten Logs bis zu einem Zeitpunkt oder Log-Ende.
- **WLM**: Workload Management zur Steuerung und Priorisierung von Ressourcen.

0.3. Lernpfad: Vom Einstieg zur Produktion

1. Verstehe zuerst das Betriebsmodell: Instance, Database, Logs, Bufferpools, Tablespaces, Utilities.
2. Trenne dann sauber zwischen OLTP, gemischten Workloads und analytischen Tabellen.
3. Modelliere erst danach Schlüssel, Partitionierung, Indizes und Isolationsniveaus.
4. Baue Runbooks für Backup, Rollforward, HADR, Locking und Performance-Diagnose.
5. Sichere zuletzt das Setup mit Rollen, Auditing, Governance und wiederholbaren Drills ab.

0.4. Mini-Projekt (durchgehend): Auftrags-, Bestands- und Reporting-Plattform

Dieses Mini-Projekt begleitet alle Kapitel. **Szenario:**

- Ein Industrieunternehmen betreibt Auftragsannahme, Lagerbuchungen, Rechnungsstellung und Versandtracking auf einer zentralen Plattform.
- Spitzenlast: bis zu 3.500 transaktionale Änderungen pro Sekunde.
- Verfügbarkeit: 24/7, geplanter Wartungsrahmen klein, Recovery klar nachweisbar.
- Reporting: Operative Dashboards sollen auf Produktionsdaten laufen, Tagesabschluss-Reports müssen vor 06:00 Uhr verfügbar sein.
- Compliance: Änderungen an Preis-, Zahlungs- und Versanddaten müssen nachvollziehbar sein.
- Betrieb: Primarstandort plus Standby; Restore- und Rollforward-Tests müssen reproduzierbar sein.

Wir verwenden dieses Szenario fortlaufend für:

- Datenmodellierung und Schluesseldefinition.
- Isolations- und Locking-Entscheidungen.
- HADR-/Backup-Strategie.
- Performance- und Storage-Design.
- Governance und Auditierbarkeit.

0.5. Profi-Heuristiken

- Db2 ist stark, wenn du Betriebsdisziplin, SQL-Disziplin und Recovery-Disziplin zusammendenkst.
- Nicht jede Tabelle braucht denselben Zugriffspfad, dieselbe Storage-Form oder denselben Isolation-Level.
- `RUNSTATS` und `REORG` sind keine Spaetphase, sondern Teil des Lebenszyklus.
- `HADR` ersetzt kein Backup und `Backup` ersetzt keine Verfuegbarkeitsstrategie.
- Column-organized Tabellen sind maechtige Werkzeuge, aber kein pauschaler Ersatz für OLTP-Tabellen.
- Bufferpools und Tablespaces sind keine Altlast, sondern praktische Hebel für Stabilität und Steuerbarkeit.
- Bei Db2 ist "wir optimieren später" oft teurer als "wir entscheiden heute sauber".

0.6. So setzt du es um - Checkliste

- Definiere 5 bis 10 kritische Endpunkte inklusive Latenz- und Recovery-Ziel.
- Lege pro Tabellenklasse fest: row-organized oder column-organized.
- Trenne Daten-, Index-, LOB- und Temporaerpfade bewusst.
- Plane `RUNSTATS`, `REORG`, Backup und Restore-Tests vor dem Go-Live.
- Dokumentiere HADR-Modus, RPO, RTO und Failover-Verfahren.

0.7. Mentales Modell für den Alltag

Db2 wird im Alltag oft falsch eingeordnet: entweder als "einfach nur ein weiteres SQL-System" oder als "schweres Enterprise-Monster, das man nur mit Spezialwissen anfassen darf". Beides verfehlt die Praxis. Das hilfreiche mentale Modell ist einfacher und gleichzeitig anspruchsvoller: Db2 ist ein relationelles Kernsystem, dessen Stärke aus der Verbindung von sauberem Datenmodell, steuerbarer Speicherarchitektur, reproduzierbarer Recovery und beobachtbarem Laufzeitverhalten entsteht. Genau deshalb sind Design und Betrieb keine getrennten Welten. Ein Tabellendesign ohne Zugriffspfad ist in Db2 nur halbe Arbeit; ein Backup ohne Rollforward-Test ist nur beruhigende Theorie; eine gute SQL-Abfrage ohne aktuelle Statistiken bleibt Zufall; ein HADR-Setup ohne Failover-Drill ist keine belastbare Verfügbarkeit.

Für den Alltag hilft eine einfache Leitfrage pro Entscheidung: "Welche konkrete Laufzeit- oder Recovery-Eigenschaft wird hier besser, messbarer oder beherrschbarer?" Wenn die Antwort unklar bleibt, ist die Entscheidung oft nur Gewohnheit. Das betrifft fast alles: Braucht diese Tabelle wirklich `RR`, oder reicht `CS`? Muss dieses Reporting auf OLTP-Tabellen laufen, oder ist eine column-organized Kopie besser? Ist ein zusätzlicher Index wirklich für einen Kernpfad da, oder nur für eine seltene Komfortabfrage? Wird ein Temp-Space-Problem gerade maskiert, statt behoben? Sobald Teams Db2 so betrachten, sinkt die Zahl der "mystischen" Performance-Spruenge und Recovery-Ueberraschungen deutlich. Das Ziel ist nicht, jede Spezialfunktion auswendig zu kennen, sondern das System so zu betreiben, dass Verhalten unter Last, Fehlern und Änderungen vorhersagbar bleibt.

---

1. Einordnung: Wann IBM Db2 die richtige Wahl ist

1.1. Problemklassen, in denen Db2 typischerweise stark ist

Db2 ist besonders sinnvoll, wenn mehrere Bedingungen gleichzeitig erfüllt sind:

- Stark relationale Kernprozesse mit klaren Integritaetsregeln.
- Hohe Anforderungen an Recovery, Nachvollziehbarkeit und stabile Betriebsfuehrung.
- Lang laufende Enterprise-Workloads mit gemischten OLTP- und Reporting-Anteilen.
- Bedarf an ausgereiften HA-/DR-Mustern ohne proprietaere Eigenbauten.
- Teams, die SQL, Utilities und Datenbankbetrieb bewusst als Kernkompetenz behandeln.

Typische Domaenen:

- ERP-nahe Kernsysteme.
- Auftrags- und Lagersteuerung.
- Finanz- und Abrechnungssysteme.
- Governance-starke Datenplattformen in regulierten Umgebungen.
- Operative Datenhaltung mit analytischer Ergänzung über BLU/column-organized Tabellen.

1.2. Wann ein anderes System oft besser passt

Ein anderes System passt häufig besser, wenn du primär brauchst:

- Ein leichtgewichtiges Open-Source-Setup mit minimalem Betriebsgewicht.
- Maximale Freiheit für lokale Entwicklerumgebungen ohne IBM-spezifische Betriebswege.
- Global verteilte Active-Active-Schreibpfade über viele Regionen.
- Dokumentenorientierte oder schemalose Iteration als Kernansatz.

- Cloud-first-Managed-Database mit möglichst wenig Low-Level-Steuerung. Db2 kann viele Dinge sehr gut, aber es ist nicht dafür da, betriebliche Disziplin zu eliminieren.

1.3. Db2-Familie sauber abgrenzen

Der Name "Db2" beschreibt mehrere Produktlinien. Diese Referenz fokussiert auf `Db2 LUW`, also `Db2 for Linux, UNIX and Windows`. Das ist wichtig, weil Begriffe wie diese dort fachlich zusammengehoeren:

- `HADR`
- `pureScale`
- `DPF`
- `db2pd`
- `RUNSTATS`
- `REORG`

Abgrenzung:

- `Db2 for z/OS` ist technisch und organisatorisch eine andere Welt mit Mainframe-Fokus.
- `Db2 for i` ist eng in IBM i integriert und folgt teils anderen Betriebsmodellen. Wer "Db2" sagt, meint im Alltag oft LUW, aber für Architekturentscheidungen muss diese Abgrenzung explizit sein.

1.4. Harte No-Go-Faelle für Db2

- Das Team will bewusst einen möglichst schlanken, generischen Stack ohne herstellerspezifische Betriebswege.
- Der Hauptnutzen wäre nur "wir kennen es historisch".
- Das Fachmodell ist eigentlich dokumentenorientiert oder stark eventuell-konsistent.
- Es gibt keine Bereitschaft, Backup-, Restore- und Statistikprozesse verbindlich zu betreiben.
- Workloads und Verantwortungen sind so unklar, dass weder Storage- noch Isolation-Entscheidungen sauber getroffen werden können.

> **Anti-Pattern:** "Wir nehmen Db2, weil es Enterprise ist." Ergebnis in der Praxis: zu viel Plattform für zu wenig klares Problem.

1.5. Entscheidungsmatrix (kompakt)

Kriterium	Db2 passt	Db2 passt eher nicht
Workload	relational, transaktional, governance-stark	schemalos, extrem volatil
Recovery	Backup, Rollforward, DR sind kritisch	Recovery ist fachlich weniger relevant
Betrieb	DBA-/SRE-Disziplin vorhanden	kaum Betriebsfokus
SQL-Nutzung	klare Kernpfade, stabile Queries	primär experimentell und lose strukturiert
Verfügbarkeit	HADR/pureScale sinnvoll	verteiltes Active-Active global erwartet

1.6. So setzt du es um - Checkliste

- Formuliere 3 echte Produktionsprobleme, die Db2 lösen soll.
- Vergleiche mindestens eine Alternative unter denselben Last- und Recovery-Annahmen.
- Trenne bei der Bewertung OLTP, Reporting, DR und Governance sauber.

- Dokumentiere explizit, welche Db2-Funktionen ihr wirklich braucht.
- Definiere ein Exit-Kriterium: Wann wäre Db2 für diesen Fall ueberdimensioniert?

1.7. Praxisvergleich: Db2 vs. "wir nehmen das Enterprise-RDBMS aus Gewohnheit"

Viele Einführungen scheitern nicht technisch, sondern organisatorisch. In Konzernen wird Db2 manchmal geerbt, nicht entschieden. Dann entstehen typische Fehlmuster:

- Tabellen werden ohne klares Performance- oder Recovery-Modell angelegt.
- HADR ist "irgendwie aktiviert", aber nie geprobt.
- Reporting läuft auf denselben OLTP-Tabellen, bis Temp-Spills und Locking sichtbar werden.
- `RUNSTATS` und `REORG` werden nur bei Problemen diskutiert.

Die bessere Reihenfolge ist:

1. Fachlich kritische Transaktionen und Reports identifizieren.
2. Recovery-Ziele und Failover-Logik festlegen.
3. Datenmodell, Storage und Isolation daraus ableiten.
4. Erst dann Utilities, Monitoring und Betriebsgrenzen festziehen. Db2 wird stark, wenn es als steuerbares Produktionssystem behandelt wird, nicht als Symbol für Enterprise-Reife.

1.8. Architekturentscheidungen sauber dokumentieren

Für Db2 haben sich kompakte ADR-Fragen bewaehrt:

- Warum Db2 LUW und nicht Alternative X?
- Welche Tabellen sind row-organized, welche column-organized und warum?
- Welche Bufferpools, Tablespaces und Seitengroessen wurden gewählt?
- Welche Isolation-Level gelten für welche fachlichen Pfade?
- Welcher HADR-Modus passt zu RPO/RTO und Netzwerkrealitaet?
- Welche Utilities sind Pflicht und in welcher Frequenz?

Ergaenze immer die Betriebsfolgen:

- Welche Kennzahl zeigt früh, dass die Entscheidung kippt?
- Welches Runbook greift zuerst?
- Welche Rueckfallstrategie gibt es bei Fehlplanung?

---

2. Architektur & Kernkonzepte

2.1. Instance, Database, Schemas, Bufferpools, Tablespaces

Db2 LUW ist logisch relational, operativ aber stark durch seine Speicher- und Laufzeitstruktur gepraegt. Wichtige Ebenen:

- **Instance**: Serverkontext mit Prozessen, Konfiguration und Sicherheitsrahmen.
- **Database**: Enthält Katalog, Logging, Konfiguration und Objekte.
- **Schema**: Logische Namensgrenze für Tabellen, Views, Routinen und Rechte.
- **Bufferpool**: Cacht Seiten einer bestimmten Seitengroesse im Hauptspeicher.
- **Tablespace**: Ordnet Daten physischen Speicherbereichen zu.

Diese Trennung ist operativ relevant, weil sie bestimmt:

- welche Seiten wo liegen,
- wie Temporaerdaten behandelt werden,
- wie I/O-Profile entstehen,
- wie sauber du OLTP, Indizes, LOBs und Temp trennen kannst.

2.2. Automatic Storage und das praktische Speichermodell

Db2 LUW unterstützt `automatic storage`, um Daten über definierte Speicherpfade zu verwalten. Das reduziert Handarbeit, ersetzt aber keine Architekturentscheidung. Bewaehrte Trennung:

- Daten-Tablespaces für Kernobjekte.
- Index-Tablespaces für stark genutzte Indizes.
- LOB-Tablespaces für grosse Objekte.
- Temporary Tablespaces für Sorts, Hashes und Zwischenergebnisse.

Warum das wichtig ist:

- Temp-Spills verhalten sich anders als OLTP-Lesezugriffe.
- LOBs verzerren sonst I/O-Profile normaler Tabellen.
- Recovery- und Reorg-Operationen lassen sich sauberer steuern.

2.3. Logging, Crash Recovery und Rollforward-Grundlogik

Db2 schreibt Änderungen zunächst in Log-Dateien und sichert damit Dauerhaftigkeit vor dem vollständigen Persistieren aller Datenseiten. Praktische Konsequenzen:

- Crash Recovery ist integraler Teil des Systems.
- Point-in-Time oder End-of-Logs Recovery erfordert Archive Logging.
- Ohne saubere Log-Strategie bleibt Recovery auf einfache Crash-Szenarien begrenzt.

Wichtige Unterscheidung:

- **Circular logging**: guenstiger, aber ohne Rollforward-Faehigkeit.
- **Archive logging**: Voraussetzung für ernsthafte Recovery- und DR-Szenarien.

2.4. Optimizer, Statistiken und Package Cache

Db2 trifft Ausfuehrungsplan-Entscheidungen kostenbasiert. Damit der Optimizer gut entscheidet, braucht er:

- aktuelle Tabellen- und Indexstatistiken,
- ein realistisches Datenmodell,
- möglichst klare SQL-Pradikate,
- stabile Zugriffspfade.

Der `package cache` ist dabei praktisch Gold wert:

- Dort landen kompilierte Statements.
- Dort entstehen Metriken für Laufzeitdiagnose.
- Dort siehst du, welche SQLs wirklich teuer sind, statt nur "gefuehlt problematisch".

2.5. Row-organized vs. column-organized

Db2 LUW kann sowohl zeilenorientierte als auch spaltenorientierte Tabellen verarbeiten. Faustregel:

- `row-organized` ist Default für OLTP.
- `column-organized` ist stark für analytische, scan-lastige oder aggregationslastige Pfade.

Wichtig:

- Column-organized ist kein genereller Turbo-Schalter.
- OLTP-Tabellen mit vielen punktuellen Änderungen profitieren meist nicht davon.
- Gemischte Plattformen funktionieren oft am besten, wenn OLTP und Reporting bewusst getrennte Tabellentypen nutzen.

2.6. Locking, Currently Committed und Laufzeitverhalten

Db2 verwaltet Parallelitaet über ein differenziertes Locking-Modell. Für den Alltag wichtig:

- Isolation-Level und Locking gehören zusammen.
- Lange Transaktionen vergroessern Konfliktflaechen ueberproportional.
- `currently committed` reduziert in vielen Fällen unnötige Blockierung zwischen Lesern und Schreibern.
  Wer Locking-Probleme nur als "Datenbank langsam" wahrnimmt, verfehlt oft die eigentliche Ursache.

2.7. Skalierungspfade: Scale-up, DPF und pureScale

Db2 hat mehrere Skalierungsmodelle, die nicht verwechselt werden sollten:

- **Scale-up**: mehr CPU, RAM und I/O auf einer Instanz.
- **DPF**: datenpartitionierte Architektur für grosse verteilte Workloads, häufig eher analytisch.
- **pureScale**: Clusteransatz für hohe Verfügbarkeit und horizontale Skalierung mit gemeinsamen Zugriffsmechanismen.

Praxisregel:

- Starte nicht mit der komplexesten Topologie.
- Waehle den Skalierungspfad nach Workload, nicht nach Marketingbild.

2.8. Mini-Projekt: Architekturentscheidung

Für das Mini-Projekt aus Abschnitt 0 wäre eine solide Startarchitektur:

- Primarsystem auf row-organized OLTP-Tabellen.
- Standby über `HADR`.
- Reporting nicht als ungefilterte Vollscan-Last auf Kerntransaktionstabellen, sondern über gezielt modellierte Objekte.
- Getrennte Tablespaces für Daten, Indizes, LOB und Temp.
- Archive Logging plus regelmäßige Restore-/Rollforward-Proben.

2.9. So setzt du es um - Checkliste

- Trenne Instance-, Database- und Storage-Entscheidungen dokumentiert.
- Lege Seitengroessen und Bufferpools bewusst fest.
- Plane Temporaer-I/O und LOB-Verhalten getrennt von OLTP.
- Aktiviere Archive Logging, wenn Recovery mehr als Crash-Recovery leisten soll.

- Ordne jede größere Tabelle einem klaren Betriebsprofil zu.

2.10. Architektur-Risiken früh erkennen

Architekturprobleme in Db2 zeigen sich oft schleichend:

- Sort- oder Hash-Operationen weichen häufig in Temp aus.
- Statement-Laufzeiten schwanken nach Statistikalter.
- Bestimmte Tabellen brauchen ueberproportional viele Reorgs.
- Log-Verbrauch steigt unter Peak stark an.
- HADR-Lag tritt erst unter Batch-Fenstern sichtbar auf. Das sind keine isolierten Bugs, sondern oft Signale für ein unpassendes Architekturmodell.

2.11. Architektur-Check vor Produktion

Vor produktivem Traffic sollte ein technischer Check zwingend enthalten:

- Archive Logging aktiv und erster erfolgreicher Backup-Zyklus vorhanden.
- Restore auf Testsystem möglich.
- Bufferpools, Tablespaces und Storagepfade bewusst getrennt.
- Temp-Space-Verhalten unter Benchmark gemessen.
- Hauptqueries mit Explain geprüft.
- HADR-Failover oder wenigstens Rollentausch im Test dokumentiert.

2.12. Architektur an SLOs koppeln

Architektur wird belastbar, wenn sie direkt an Serviceziele gekoppelt ist. Beispiel:

- SLO: Auftrag speichern in < 120 ms P95.
- Architekturauswirkung: row-organized Tabelle, schlanke Indizes, kurzer Commit-Pfad.
- Reportingfolge: Tagesreport nicht auf derselben Tabellenform erzwingen. Wenn diese Kopplung fehlt, werden Plattformentscheidungen zu Bauchgefuehl.

---

3. Datenmodellierung in der Praxis

3.1. Das wichtigste Prinzip: Fachregeln und Zugriffspfade gemeinsam modellieren

In Db2 ist gute Modellierung nicht nur Normalisierung. Sie ist die Verbindung aus:

- fachlicher Integrität,
- sinnvollen Schlüsseln,
- realistischen Zugriffspfaden,
- beherrschbarer Wartung. Ein formal schoenes Modell kann im Betrieb trotzdem schlecht sein, wenn Kernabfragen unnötige Sorts, Spills oder Lock-Ketten erzeugen.

3.2. Primärschlüssel: fachlich sauber, technisch stabil

Bewaehrte Regeln:

- Primärschlüssel müssen stabil sein.

- Business Keys sollen dort eindeutig bleiben, wo die Fachlogik es verlangt.
- Surrogate Keys sind sinnvoll, wenn Business Keys zu breit, zu instabil oder zu teuer für FK-Ketten sind.
- Jede Schluesselentscheidung braucht einen Satz dazu, wie sie Indexdesign und Join-Tiefe beeinflusst.

3.3. Fremdschlüssel und referenzielle Integrität bewusst einsetzen

Db2 ist stark, wenn relationale Integrität wirklich genutzt wird. Praxisregeln:

- Fremdschlüssel dort setzen, wo Fachbruch teuer wäre.
- Nicht jede Hilfstabelle braucht denselben Grad an Strenge.
- Löschregeln (`RESTRICT`, `CASCADE`, `SET NULL`) müssen fachlich begründet sein. Referenzielle Integrität ist kein Luxus, aber auch kein Selbstzweck.

3.4. Tabellenformen nach Workload trennen

Für viele Teams ist die wichtigste Modellierungsentscheidung nicht eine Spalte mehr oder weniger, sondern die Trennung der Tabellenformen:

- OLTP-Tabellen: row-organized.
- Analytische Tabellen oder Marts: column-organized.
- Historisierungsobjekte: oft anders indiziert und anders aufbewahrt als operative Kerntabellen. Diese Trennung reduziert Zielkonflikte zwischen kurzen Transaktionen und grossen Aggregationen deutlich.

3.5. Partitionierung für Lebenszyklus, Wartung und Datenvolumen

Partitionierung ist in Db2 kein modisches Feature, sondern oft ein Wartungshebel. Sinnvolle Einsatzzwecke:

- Datenhaltung nach Zeitfenstern.
- Schnellere Archivierungs- oder Purge-Prozesse.
- Lokalisierung grosser Wartungsmassnahmen.
- Bessere Steuerbarkeit grosser Faktentabellen. Gefährlich wird Partitionierung, wenn sie nur "vielleicht später hilfreich" eingefuehrt wird, ohne klaren Purge-, Report- oder Recovery-Nutzen.

3.6. Indexdesign: jeder Index braucht einen legitimen Zweck

Jeder Index verbessert etwas und kostet etwas:

- mehr Schreibaufwand,
- mehr Speicher,
- mehr Reorg-/Statistikaufwand,
- mehr potentielle Fehlpfade für den Optimizer.

Bewaehrte Heuristiken:

- Indizes zuerst für Kernpfade bauen, nicht für Einzelfallkomfort.
- Join-, Filter- und Sortmuster gemeinsam betrachten.
- Ueberlappende Indizes systematisch vermeiden.
- Nach jedem zusätzlichen Index fragen: Welche produktive Query wird dadurch messbar besser?

3.7. Generated Columns, Identity und Sequenzen

Db2 bietet klassische Mittel für stabile Schlüssel- und Ableitungslogik:

- `GENERATED ALWAYS AS IDENTITY`
- Sequenzen
- generierte Spalten für wiederkehrende Ableitungen Der Nutzen ist hoch, wenn dadurch SQL schlanker, Join-Logik stabiler oder Governance konsistenter wird.

3.8. Mini-Projekt: Basisschema

Ein möglicher Ausschnitt für das Mini-Projekt:

```
CREATE SCHEMA core;

CREATE TABLE core.orders (
    order_id            BIGINT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    customer_id         BIGINT NOT NULL,
    order_no            VARCHAR(30) NOT NULL,
    order_status        VARCHAR(20) NOT NULL,
    order_ts            TIMESTAMP NOT NULL DEFAULT CURRENT TIMESTAMP,
    currency_code       CHAR(3) NOT NULL,
    gross_amount        DECIMAL(15,2) NOT NULL,
    CONSTRAINT uq_orders_order_no UNIQUE (order_no)
);

CREATE TABLE core.order_items (
    order_item_id       BIGINT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    order_id            BIGINT NOT NULL,
    line_no             INTEGER NOT NULL,
    sku                 VARCHAR(40) NOT NULL,
    qty                 INTEGER NOT NULL,
    unit_price          DECIMAL(15,2) NOT NULL,
    CONSTRAINT uq_order_items UNIQUE (order_id, line_no),
    CONSTRAINT fk_order_items_orders
        FOREIGN KEY (order_id) REFERENCES core.orders(order_id)
);

CREATE INDEX ix_orders_status_ts
    ON core.orders(order_status, order_ts);

CREATE INDEX ix_order_items_order_id
    ON core.order_items(order_id);
```

Das ist bewusst konservativ:

- row-organized Standard.
- schlanke Unique-Regeln.
- Indizes für realistische Kernzugriffe.
- keine ueberzogene Modellakrobatik.

3.9. Column-organized Beispiel für Reporting

Für scan-lastige Tagesreports kann ein separates Objekt sinnvoll sein:

```
CREATE TABLE mart.daily_sales (
    sales_day           DATE NOT NULL,
    plant_id            INTEGER NOT NULL,
    sku                 VARCHAR(40) NOT NULL,
    revenue_amount      DECIMAL(15,2) NOT NULL,
    sold_qty            BIGINT NOT NULL
)
ORGANIZE BY COLUMN;
```

Der Punkt ist nicht "mehr Technologie", sondern Lasttrennung.

3.10. So setzt du es um - Checkliste

- Definiere pro Tabelle Geschäftszweck, Hauptabfragen und Wartungsprofil.
- Halte Schluesselbreite und FK-Ketten bewusst kontrolliert.
- Trenne OLTP und Reporting früh, wenn deren Lastmuster auseinanderlaufen.
- Führe Partitionierung nur mit klarem Datenlebenszyklus ein.
- Dokumentiere für jeden Index die Kernabfrage, die er absichern soll.

3.11. Typische Anti-Pattern

- Einheitsmodell für OLTP, Reporting und Archiv.
- Zu viele Bequemlichkeitsindizes.
- Sehr breite Schlüssel ohne echten Fachnutzen.
- Fachlich wichtige Integritaetsregeln nur in Anwendungscode.
- Historisierung ohne klaren Purge- oder Archivpfad.

3.12. Datenklassifizierung und Aufbewahrung

Ein pragmatisches Modell:

- **Tier A**: operative Kerndaten mit langer Aufbewahrung und hoher Governance.
- **Tier B**: operative Ereignisse mit mittlerer Haltedauer.
- **Tier C**: abgeleitete Reportingdaten, die rekonstruiert werden können.

Diese Klassifikation hilft bei:

- Tablespace-Strategie,
- Backup-Priorisierung,
- HADR-Erwartung,
- Reorg-/Runstats-Frequenzen,
- Kostenkontrolle.

---

4. SQL, Utilities und systematische Praxisreferenz

4.1. Basismuster für DDL und Deployment

Db2 gewinnt im Alltag durch vorhersehbare Deployments. Grundregeln:

- Schemas bewusst trennen (`core`, `mart`, `stage`, `admin`).
- DDL-Dateien reproduzierbar und versioniert halten.
- Admin-Berechtigungen nicht mit Anwendungskonten mischen.

Kompaktes Startmuster:

```
CREATE ROLE app_read;
CREATE ROLE app_write;

GRANT CONNECT ON DATABASE TO ROLE app_read, ROLE app_write;
GRANT USAGE ON SCHEMA core TO ROLE app_read, ROLE app_write;
GRANT SELECT ON TABLE core.orders TO ROLE app_read;
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE core.orders TO ROLE app_write;
```

4.2. Typische OLTP-SQL-Muster

Ein gutes OLTP-Statement in Db2 ist:

- selektiv,
- parameterisiert,
- gut indexiert,
- klar im Isolation-Verhalten.

Beispiel für einen typischen Kernzugriff:

```
SELECT order_id,
       order_status,
       order_ts,
       gross_amount
FROM core.orders
WHERE order_no = ?
FETCH FIRST 1 ROW ONLY;
```

Das ist bewusst einfach. Genau solche Statements müssen in Db2 langweilig schnell und reproduzierbar sein.

4.3. `MERGE` für Upsert- oder Synchronisationspfade

Ein typisches Muster für Bestandsfortschreibung:

```
MERGE INTO core.inventory_balance AS t
USING (
    VALUES (?, ?, ?)
) AS s(plant_id, sku, delta_qty)
ON t.plant_id = s.plant_id
AND t.sku = s.sku
WHEN MATCHED THEN
    UPDATE SET qty = t.qty + s.delta_qty,
               updated_at = CURRENT TIMESTAMP
WHEN NOT MATCHED THEN
    INSERT (plant_id, sku, qty, updated_at)
    VALUES (s.plant_id, s.sku, s.delta_qty, CURRENT TIMESTAMP);
```

Wichtig:

- `MERGE` ist maechtig, aber nicht automatisch billig.
- Indexe auf Match-Praedikaten sind Pflicht.
- Semantik und Parallelitaet müssen bei hohem Konkurrenzgrad bewusst getestet werden.

4.4. Analytische SQL-Muster

Db2 ist stark in klassischer SQL-Analytik. Beispiel mit Window Functions:

```
SELECT customer_id,
       order_id,
       order_ts,
       gross_amount,
       SUM(gross_amount) OVER (
           PARTITION BY customer_id
           ORDER BY order_ts
           ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
       ) AS running_amount
FROM core.orders
WHERE order_ts >= CURRENT DATE - 30 DAYS;
```

Hier entscheidet nicht nur SQL-Syntax, sondern:

- Datenvolumen,
- Index-/Sortpfad,
- Temp-Nutzung,
- Tabellenform.

4.5. Explain und Access-Path-Diagnose

Kein ernsthaftes Db2-Tuning ohne Explain. Einfaches Muster:

```
EXPLAIN PLAN FOR
SELECT order_id, order_status
FROM core.orders
WHERE order_status = 'OPEN'
AND order_ts >= CURRENT DATE - 7 DAYS;
```

Danach häufiger Praxisweg über `db2exfmt`:

```
db2exfmt -d COREDB -1 -o explain_orders.txt
```

Nicht nur lesen, ob ein Index benutzt wird. Wichtiger ist:

- Wie viele Zeilen schaetzt der Optimizer?
- Gibt es Sorts oder Temp-Spills?
- Sind Statistiken offensichtlich veraltet?
- Ist das Statement für genau diesen Endpunkt akzeptabel?

4.6. Monitoring mit `MON_GET_PKG_CACHE_STMT`

Für SQL-Diagnose ist die Package-Cache-Sicht hochpraktisch:

```
SELECT SUBSTR(stmt_text, 1, 200) AS stmt_text,
       num_executions,
       total_act_time,
       rows_returned
FROM TABLE(MON_GET_PKG_CACHE_STMT(NULL, NULL, NULL, -2)) AS t
ORDER BY total_act_time DESC
FETCH FIRST 20 ROWS ONLY;
```

Das liefert keine komplette Ursachenanalyse, aber sehr schnell:

- teure Statements,
- häufige Statements,
- Kandidaten für Explain und Indexpruefung.

4.7. Administrative SQL und Katalogsicht

Db2 bietet mehrere Beobachtungsflaechen:

- `MON_GET_DATABASE`
- `MON_GET_TABLESPACE`
- `MON_GET_BUFFERPOOL`
- `MON_GET_PKG_CACHE_STMT`
- administrative Views und Katalogtabellen

Der kluegere Weg ist meist:

1. Symptom eingrenzen.
2. Nur die passende Beobachtungsflaeche abfragen.
3. Ergebnis gegen Workload und Zeitfenster interpretieren.

4.8. Zentrale Utilities

Db2-Utilities sind Teil des Normalbetriebs:

- `RUNSTATS`
- `REORG TABLE`
- `REORG INDEXES ALL FOR TABLE`
- `BACKUP DATABASE`
- `RESTORE DATABASE`
- `ROLLFORWARD DATABASE`
- `LOAD`

Beispiele:

```
db2 "RUNSTATS ON TABLE core.orders WITH DISTRIBUTION AND DETAILED INDEXES ALL"
db2 "REORG INDEXES ALL FOR TABLE core.orders ALLOW WRITE ACCESS"
db2 "BACKUP DATABASE coredb ONLINE TO /dbbackup INCLUDE LOGS"
```

Praxisregel:

- Keine Utility "einfach mal" in Produktion ausführen.
- Jede Utility braucht Zeitfenster, Erwartungswerte und Rueckfalllogik.

4.9. `LOAD` und Massenimporte

Massendatenpfade können Produktivverhalten stark beeinflussen. Worauf du achten musst:

- Logging-Auswirkungen.
- Nachgelagerte `RUNSTATS`.
- Eventuelle `REORG`-Notwendigkeit.
- Recovery-Folgen des gewählten Importwegs. Ein schneller Ladevorgang, der später schlechte Plaene oder aufwendige Recovery erzeugt, ist nur scheinbar effizient.

4.10. CLP- und Diagnose-Grundmuster

Typische Kommandos im Alltag:

```
db2 connect to coredb
db2 -tvf deploy.sql
db2pd -db coredb -locks
db2pd -db coredb -utilities
```

`db2pd` ist besonders wertvoll für schnelle Live-Diagnose, wenn du wissen musst:

- Wo hängen Locks?
- Welche Utilities laufen?
- Welche Sessions erzeugen gerade Druck?

4.11. So setzt du es um - Checkliste

- Jede Kernquery hat mindestens einen Explain-Nachweis.
- Package-Cache-Auswertungen sind Teil des Regelbetriebs.
- `RUNSTATS` und `REORG` haben definierte Trigger oder Zeitplaene.
- Admin- und Deploy-SQL sind versioniert und reproduzierbar.
- Massenimporte haben ein eigenes Betriebs- und Recovery-Konzept.

4.12. SQL- und Utility-Fehlmuster

- Explain erst nach dem Incident.
- `RUNSTATS` nur manuell "wenn etwas komisch wirkt".
- `MERGE` auf schlecht indexierte Tabellen.
- Reporting-SQL direkt auf OLTP-Hotpaths ohne Trennung.
- Utility-Lauf ohne Vorabmessung von I/O, Logs und Zeitfenster.

---

5. Transaktionen, Isolation, Recovery und Verfügbarkeit

5.1. Isolation-Level: `UR`, `CS`, `RS`, `RR`

Die vier klassischen Db2-Isolationsniveaus sind keine Theorie, sondern Fachentscheidungen:

- **UR (Uncommitted Read)**: maximal locker, guenstig für gewisse leselastige Diagnose- oder Reportzwecke, aber mit Inkonsistenzrisiko.
- **CS (Cursor Stability)**: häufige Default-Wahl für viele OLTP-Lesepfade.
- **RS (Read Stability)**: stabiler für wiederholtes Lesen relevanter Datenmengen innerhalb einer Transaktion.
- **RR (Repeatable Read)**: am strengsten, aber auch blockierungs- und ressourcenintensiver.

Merksatz:

- Höhere Isolation ist nicht "professioneller".
- Sie ist nur dann richtig, wenn die Fachinvariante sie wirklich braucht.

5.2. Locking und Deadlocks

Db2 reagiert empfindlich auf schlecht begrenzte Transaktionen. Typische Ursachen für Locking-Probleme:

- Lange Business-Transaktionen.
- Falsch gewählte Isolation-Level.
- Schreibkonflikte auf Hot Rows.
- Unnötige Scans auf stark veränderten Tabellen.
- Batch und OLTP im gleichen Zeitfenster ohne Steuerung.

Praxisregeln:

- Transaktionen kurz halten.
- Zugriffsreihenfolgen möglichst konsistent halten.
- Lock-Waits und Deadlocks nicht nur zählen, sondern bis auf SQL und Endpunkt zurueckfuehren.

5.3. Savepoints, Commit-Groessen und Fehlertoleranz

Gute Transaktionsgestaltung in Db2 heisst auch:

- sinnvolle Commit-Groessen,
- Fehlergrenzen in Batch-Pfaden,
- klare Rollback-Semantik,
- keine ueberdimensionierten Alles-oder-nichts-Transaktionen aus Bequemlichkeit. Gerade bei Batch und Synchronisationen sind Savepoints und Commit-Frequenzen oft wichtiger als ein vermeintlich elegantes "ein grosser Lauf".

5.4. HADR-Grundmodell

`HADR` verbindet ein Primarsystem mit einem oder mehreren Standby-Konzepten im Disaster- oder High-Availability-Sinn. Praktisch wichtig:

- HADR ist für Verfügbarkeit und DR.
- HADR ist kein Ersatz für Backup und Recovery-Tests.
- Netzwerklatenz und Sync-Modus bestimmen das reale Verhalten.

Typische Sync-Modi:

- `SYNC`
- `NEARSYNC`
- `ASYNC`
- `SUPERASYNC`

Die Wahl ist immer ein Trade-off aus:

- Datenverlusttoleranz,
- Commit-Latenz,
- Netzwerkqualitaet,
- Standortmodell.

5.5. Backup, Restore und Rollforward

Ein belastbares Db2-System hat nicht nur Backups, sondern nachgewiesene Wiederherstellbarkeit. Mindestbestandteile:

- dokumentierter Backup-Zyklus,
- gültiges Archive Logging,
- Restore-Test,
- Rollforward-Test bis definiertem Punkt.

Wichtige Erkenntnis:

- Ein Backup ist erst dann betriebsreif, wenn du es erfolgreich wiederhergestellt und pruefbar weitergerollt hast.

5.6. Rollforward als Produktionsmechanismus

IBM beschreibt `ROLLFORWARD` explizit als Mechanismus, um nach einem Restore Änderungen aus Logs weiter anzuwenden. Das ist praktisch entscheidend für:

- Point-in-Time-Recovery,
- End-of-Logs-Recovery,
- table-space-bezogene Wiederherstellung,
- ernsthafte DR-Szenarien. Teams, die Rollforward nie ueben, haben oft nur eine theoretische Recovery-Strategie.

5.7. pureScale einordnen

`pureScale` ist kein "einfaches HADR in größer", sondern ein anderes Verfuegbarkeits- und Skalierungsmodell. Es lohnt sich, wenn:

- sehr hohe Verfügbarkeit und Clusterfaehigkeit im Vordergrund stehen,
- das Betriebsmodell die Komplexität rechtfertigt,
- die Anwendung wirklich von diesem Architekturansatz profitiert. Es lohnt sich nicht automatisch, nur weil das System kritisch ist.

5.8. Mini-Projekt: HA-/DR-Profil

Für das Mini-Projekt wäre ein plausibles Profil:

- Primarsystem mit `NEARSYNC` oder `ASYNC` je nach Standort und Latenzbudget.
- Tägliche Online-Backups mit Log-Einbezug.
- Quartalsweise Point-in-Time-Recovery-Drills.
- Klare Trennung zwischen geplanter Umschaltung und ungeplantem Ausfall.

5.9. So setzt du es um - Checkliste

- Waehle Isolation-Level pro Endpunkt, nicht global aus Gewohnheit.
- Begrenze Transaktionsdauer und Commit-Groesse bewusst.
- Dokumentiere HADR-Modus, Ziel-RPO und erwarteten Lag.
- Führe Restore- und Rollforward-Tests regelmäßig auf Testsystemen durch.
- Trenne HA-, Backup- und DR-Diskussionen organisatorisch nicht voneinander.

5.10. Failure-Drills und Nachweispflicht

Bewaehrte Drills:

- Restore einer Produktionssicherung auf isolierter Testumgebung.
- Rollforward bis zu definiertem Zeitpunkt.
- Failover-/Takeover-Probe im HADR-Setup.
- Locking- und Deadlock-Simulation für Hotpaths.

Wichtig:

- Nicht nur "hat irgendwie funktioniert", sondern Dauer, Schritte, Abweichungen und manuelle Eingriffe dokumentieren.

5.11. Verfuegbarkeitsfehler, die oft übersehen werden

- Standby ist formal vorhanden, aber Monitoring auf Replay-Delay fehlt.
- Log-Archiving ist konfiguriert, aber Pfadprobleme bleiben lange unbemerkt.
- Takeover-Verfahren ist nur bei den erfahrensten zwei Personen bekannt.

- Restore-Pfad ist wegen kleiner Umgebungsdifferenzen praktisch nicht einsatzfaehig. Db2 ist robust, aber Robustheit muss operationalisiert werden.

---

6. Betrieb, Monitoring & Troubleshooting

6.1. Betriebsgrundsatz

Db2 wird stabil, wenn Betrieb nicht als Reaktion auf Stoerungen organisiert ist, sondern als laufende Pflege von:

- Statistiken,
- Speicherverhalten,
- Logs,
- Utilities,
- Zugriffspfaden,
- Recovery-Faehigkeit.

6.2. Welche Metriken du wirklich kennen solltest

Eine praxistaugliche Startliste:

- Log-Verbrauch und Log-Wachstum.
- Lock-Waits und Deadlocks.
- HADR-Lag bzw. Replay-Delay.
- Package-Cache-Hotspots.
- Temp-Nutzung und Sort-Spills.
- Tabellen- und Indexwachstum.
- Bufferpool-Verhalten.
- Ueberfaellige `RUNSTATS` oder `REORG`.
- Tablespace-Auslastung.
- Dauer kritischer Utilities.

6.3. Beobachtungsflaechen

Wichtige Werkzeuge:

- `MON_GET_*` Table Functions.
- `db2pd`.
- Explain und `db2exfmt`.
- administrative Views.
- Event Monitors. Kein Tool allein ist "das Monitoring". Die Kunst ist, pro Symptom den schnellsten Pfad zu kennen.

6.4. Typische Diagnosepfade

Wenn SQL langsam wird:

- Statement über Package Cache identifizieren.
- Explain prüfen.

- Statistikalter und Datenveraenderung bewerten.
- Temp-Spills, Sorts und Indexnutzung prüfen.

Wenn Transaktionen hängen:

- `db2pd -db <db> -locks`
- blockierende Sessions identifizieren
- Isolation- und Batch-Muster gegenpruefen

Wenn HADR instabil wirkt:

- Lag und Netzwerkbedingungen prüfen
- Log-Aufkommen pro Zeitfenster analysieren
- Batch-Fenster gegen Commit-Latenz spiegeln

Wenn Storage-Druck steigt:

- Tabellen-/Indexwachstum prüfen
- Reorg-Bedarf identifizieren
- Purge- und Archivierungsmechanismen validieren

6.5. `db2pd` im Alltag

`db2pd` ist besonders wertvoll, wenn du schnelle Live-Sicht brauchst:

- Locks
- Utilities
- HADR
- Speicher- und Sitzungszustand Es ersetzt keine tiefe Analyse, aber es verkuerzt die Zeit bis zur ersten belastbaren Hypothese massiv.

6.6. Statistik- und Reorg-Disziplin

Viele Leistungsprobleme in Db2 sind keine "Engine-Bugs", sondern Folgen von:

- veralteten Statistiken,
- physischer Tabellenfragmentierung,
- gewachsenem Datenvolumen ohne Wartungsanpassung.

Bewaehrte Routine:

- Statistikzyklen pro Tabellenklasse definieren.
- Reorg-Bedarf nicht raten, sondern anhand von Symptomen und Metriken steuern.
- Nach grossen Loads oder starken Datenverschiebungen nicht blind weitermachen.

6.7. Backup- und Restore-Betrieb

Backups brauchen Betriebsdisziplin:

- erfolgreicher Lauf,
- gültige Zielpfade,
- nachvollziehbare Aufbewahrung,
- periodische Wiederherstellungstests. Gefährlich ist nicht nur fehlendes Backup, sondern das "stille Scheitern" rundherum:

- Log-Archive nicht lesbar,
- Restore fällt an Rechten oder Pfaden,
- Testsystem unbrauchbar vorbereitet,
- Rollen und Skripte nicht aktuell.

6.8. Fix-Pack- und Upgrade-Disziplin

Zum Stand **11. April 2026** zeigt IBM parallel eine `12.1`-Linie und weiterhin die `11.5`-Linie, wobei `11.5 Mod 9 Fix Pack 0` auf der IBM-Support-Seite als final markiert ist. Praxisfolgerung:

- Versionsstrategie explizit festlegen.
- Nicht nur Sicherheitsaspekte, sondern auch Kompatibilitaet, Treiber, Utilities und Runbooks mitpruefen.
- Fix Packs zuerst auf repraesentativer Testumgebung erproben.

6.9. So setzt du es um - Checkliste

- Mindestens 10 Betriebsmetriken sind mit Schwellenwerten definiert.
- Package-Cache-Analyse und Explain gehören zum Incident-Standard.
- Statistik- und Reorg-Zyklen sind pro Tabellenklasse dokumentiert.
- Restore- und Rollforward-Tests sind terminiert.
- HADR-Zustand wird nicht nur technisch, sondern gegen RPO interpretiert.

6.10. Incident-Anti-Pattern

- Erst einmal viele Parameter ändern, bevor die Ursache klar ist.
- Locks toeten, ohne die blockierende Geschaeftslogik zu verstehen.
- Langsame SQL nur am Datenbankserver suchen, nicht am Aufrufpfad.
- Reorgs oder Runstats pauschal als Wundermittel verwenden.
- HADR-Probleme auf "Netzwerk halt" reduzieren, ohne Commit- und Batch-Muster anzusehen.

6.11. Runbook-Vorlage pro Kerntabelle

Eine hilfreiche Vorlage:

- Tabellenname und Business-Zweck.
- Kernqueries und erwartete Last.
- Indizes und ihre legitimen Zwecke.
- Statistik- und Reorg-Strategie.
- Recovery-Relevanz.
- Häufige Fehlerbilder.
- Erste Diagnosekommandos. Das senkt Einarbeitungs- und On-Call-Risiken deutlich.

6.12. Post-Incident-Lernen

Nach jedem Incident sollten mindestens diese Fragen beantwortet werden:

1. Was war das erste messbare Signal?
2. Welche Diagnose war schnell hilfreich?
3. Welche Architektur- oder Betriebsannahme war unvollständig?
4. Welche Runbook- oder Monitoring-Aenderung folgt konkret?

5. Wie prüfen wir, dass derselbe Fehler früher erkannt wird?

---

7. Performance, Skalierung & Kostenkontrolle

7.1. Erst Workload-Schnitt, dann Tuning

Db2-Tuning scheitert oft daran, dass Teams alles gleichzeitig optimieren wollen. Die erste sinnvolle Trennung lautet:

- OLTP-Kernpfade
- Reporting- und Scan-Pfade
- Batch- und Ladepfade
- Wartungs- und Utility-Fenster Wenn diese vier Typen nicht unterschieden werden, wird jedes Tuning diffus.

7.2. Bufferpools, Seitengroessen und Temp-Verhalten

Bufferpools sind in Db2 praktische Steuerhebel. Wichtige Punkte:

- Seitengroesse und Objektprofil müssen zusammenpassen.
- Temp-Workloads dürfen OLTP-I/O nicht unbemerkt dominieren.
- Ein "guter Cache-Hit-Wert" allein beweist noch keine gute Performance.

Sinnvoller ist die Frage:

- Ist der Hauptspeicher so eingesetzt, dass kritische Pfade stabil bleiben?

7.3. Statistiken und physische Ordnung als Performancebasis

Ohne brauchbare Statistiken ist selbst gute Hardware oft nur eine teure Nebelmaschine. Db2-Performance hängt stark daran, dass:

- `RUNSTATS` sinnvoll gefahren wird,
- `REORG` nicht verschleppt wird,
- Datenverteilung nicht nur theoretisch, sondern realistisch im Optimizer ankommt.

7.4. Indexkosten konsequent gegenrechnen

Ein neuer Index kann:

- einen Report massiv beschleunigen,
- aber jeden Write-Pfad verlangsamen,
- Speicher vergroessern,
- Wartungsfenster verlängern,
- Access-Path-Stabilitaet verändern.

Darum gehört zu jedem Index:

- welcher SQL-Pfad profitiert,
- wie oft dieser Pfad vorkommt,
- welche Schreibkosten entstehen.

7.5. BLU und column-organized sinnvoll einsetzen

Db2s spaltenorientierte Fähigkeiten sind stark, wenn:

- grosse Datenmengen gescannt werden,
- Aggregationen über viele Zeilen laufen,
- Reporting oder Hybrid-Analytics auf Produktionsnaehe erfolgen soll.

Sie sind weniger stark, wenn:

- sehr häufig kleine Punktupdates dominieren,
- kurze OLTP-Transaktionen der Hauptworkload sind. Der Fehler ist selten "BLU ist schlecht", sondern "BLU wurde für den falschen Workload benutzt".

7.6. Workload Management und Priorisierung

`WLM` ist dort sinnvoll, wo konkurrierende Lastarten aktiv gesteuert werden müssen. Typische Ziele:

- Reporting darf OLTP nicht unkontrolliert verdraengen.
- Batch darf nicht unbemerkt in Kernzeitfenster ausufern.
- Schwere SQLs müssen erkennbare Grenzen bekommen. Ohne solche Steuerung bleibt Performance oft ein politisches Problem statt ein technisches.

7.7. Skalierungsstrategie

Pragmatischer Standardpfad:

1. Zuerst gutes Datenmodell und saubere SQLs.
2. Dann Scale-up und Memory-/I/O-Disziplin.
3. Danach gezielte Workload-Trennung.
4. Erst später komplexere Topologien wie `pureScale` oder `DPF`, falls fachlich wirklich nötig. Die teuerste Skalierung ist häufig die, die ein Modellierungsproblem kaschieren soll.

7.8. Kapazitaetsheuristiken

Plane nicht nur Datenvolumen, sondern auch:

- Peak-Transaktionen pro Sekunde.
- Batch-Fenster und Importwellen.
- Log-Wachstum.
- Temp-Bedarf.
- Indexwachstum.
- Reorg- und Backup-Zeiten.
- HADR-Transfer unter Peak. Realistische Kapazitaetsplanung ist immer systemisch, nie nur "wie gross wird die Tabelle".

7.9. Mini-Projekt: Tuning-Loop

1. Kernpfade im Package Cache identifizieren.
2. Explain und Statistikstatus prüfen.
3. Temp- und Sortverhalten messen.
4. Reporting von OLTP entkoppeln, wenn notwendig.
5. HADR-Verhalten unter Last gegenpruefen.

6. Danach erst Bufferpool- oder Indexaenderungen beurteilen.

7.10. So setzt du es um - Checkliste

- Jede Leistungsmassnahme ist einem Workload-Typ zugeordnet.
- Explain und Package Cache sind für Kernpfade dokumentiert.
- Temp-Spills werden gemessen und nicht nur vermutet.
- BLU/column-organized kommt nur dort zum Einsatz, wo Scan- und Aggregationslogik es rechtfertigt.
- Batch-, Reporting- und OLTP-Fenster sind organisatorisch getrennt oder technisch priorisiert.

7.11. Benchmarking, das wirklich zählt

Ein brauchbarer Benchmark bildet ab:

- echte SQL-Muster,
- realistische Datenverteilung,
- konkurrierende Lastarten,
- sinnvolle Commit-Groessen,
- Recovery- und HADR-Nebeneffekte.

Entscheidend sind nicht nur Durchschnittswerte, sondern:

- P95/P99 kritischer Pfade,
- Verhalten bei Lastwechseln,
- Einfluss auf Logs und Temp,
- Erholung nach Batch-Spitzen.

7.12. Kostenkontrolle durch Datenklassifizierung

Kostenkontrolle funktioniert besser, wenn Datenklassen unterschiedlich behandelt werden:

- Tier A: hoher Schutz, hohe Prioritaet, lange Aufbewahrung.
- Tier B: operative Relevanz, aber kuerzere Haltedauer.
- Tier C: rekonstruierbare oder aggregierte Daten. Damit werden Storage-, Backup- und Reorg-Kosten steuerbar, ohne kritische Pfade zu gefaehrden.

---

8. Sicherheit, Governance & Compliance

8.1. Sicherheitsgrundsatz

Eine funktionierende Db2-Installation ist noch kein produktionsreifes Sicherheitsmodell. Du brauchst mindestens:

- saubere Rollen- und Rechtevergabe,
- abgesicherte Admin-Pfade,
- nachvollziehbare Änderungen,
- geprüfte Backup- und Recovery-Zugaenge.

8.2. Rollen, Rechte und Least Privilege

Bewaehrte Mindestregeln:

- Keine Shared-Admin-Accounts für Anwendungen.
- Lese- und Schreibrollen trennen.
- Deploy-/Admin-Rechte nicht in App-Konten verstecken.
- Rechte möglichst nah an fachlichen Verantwortungen schneiden.

Beispielmuster:

- `app_read_role`
- `app_write_role`
- `ops_role`
- `audit_role`

8.3. Netzwerksicherheit und Verschluesselung

Für produktive Umgebungen gehören Transportverschluesselung und saubere Zertifikatsprozesse dazu. Pragmatischer Grundsatz:

- Client-Kommunikation absichern.
- Replikations- und Betriebswege in dasselbe Sicherheitsmodell einbeziehen.
- Zertifikatsrotation und Secrets nicht nur dokumentieren, sondern ueben.

8.4. Auditing und Nachvollziehbarkeit

Db2 bietet Auditierungsmoeglichkeiten, die in governance-starken Umgebungen zentral sind. Technisch wichtig ist aber weniger das blosse Aktivieren als die saubere Entscheidung:

- Was wird auditiert?
- Wie lange wird es aufbewahrt?
- Wer darf es auswerten?
- Wie bleibt das Auditing selbst nachvollziehbar? Ein Audit, das niemand operational lesen kann, erzeugt mehr Schein als Sicherheit.

8.5. Row and Column Access Control

Db2 bietet mit `RCAC` Mechanismen, um Zugriffe auf Zeilen- und Spaltenebene zu steuern. Das ist besonders sinnvoll, wenn:

- Fachrollen unterschiedliche Sicht auf dieselben Tabellen brauchen,
- Maskierung oder Zugriffseinschraenkung nahe an den Daten liegen soll,
- Anwendungscode nicht jede Regel duplizieren soll.

Aber auch hier gilt:

- RCAC ersetzt kein sauberes Rollenmodell.
- RCAC ersetzt keine Datenklassifikation.

8.6. Trennung von Verantwortungen

Bewaehrte Betriebstrennung:

- Anwendungsteam verantwortet SQL und Fachmodell.
- DBA-/Plattformteam verantwortet Utilities, Recovery, Storage und HADR.
- Security/Governance verantwortet Auditregeln, Rezertifizierung und Ausnahmeprozesse. Wenn diese Verantwortungen verschwimmen, entstehen häufig stille Ueberprivilegierung und unklare Incident-Pfade.

8.7. Compliance-Basis

Für regulierte Umgebungen sollten mindestens vorhanden sein:

- Rollen- und Rechtenachweise,
- dokumentierte Backup-/Restore-Pfade,
- Audit- und Aufbewahrungsregeln,
- rezertifizierte technische Konten,
- nachvollziehbare Änderungen an Kernobjekten und Berechtigungen.

8.8. So setzt du es um - Checkliste

- Rollenmodell folgt Least Privilege.
- Administrative Konten sind getrennt von Anwendungskonten.
- Auditierung ist zielgerichtet konfiguriert.
- RCAC wird dort eingesetzt, wo echte Fachtrennung gebraucht wird.
- Security-Runbooks decken Rechteaenderung, Schluesselmaterial und Incident-Pfade ab.

8.9. Rollenrezertifizierung und Drift-Kontrolle

Die größte Sicherheitsgefahr ist oft nicht die Erstkonfiguration, sondern die Drift über Monate. Bewaehrte Praxis:

- quartalsweise Rechtepruefung,
- technische Konten gegen aktuelle Dienste spiegeln,
- temporäre Ausnahmerechte mit Ablauf versehen,
- Reports über ueberprivilegierte Konten automatisieren.

8.10. Governance-Metriken

Sicherheit wird steuerbar, wenn sie gemessen wird. Sinnvolle Kennzahlen:

- Anteil technischer Konten mit nachgewiesenem Zweck.
- Anzahl offener Rollenrezertifizierungen.
- Anzahl besonders privilegierter Konten.
- Audit-Abdeckung für Kernobjekte.
- Zeit bis zum Entzug nicht mehr benoetigter Rechte.

8.11. Security-Drills

Wie bei Restore-Drills sollten auch Security-Drills erfolgen:

- kompromittiertes Anwendungskonto,
- fehlerhafte Berechtigungsausweitung,
- Audit-Fall mit Datenzugriffsnachweis,

- Zertifikats- oder Secret-Rotation. So wird Sicherheit handlungsfaehig und nicht nur dokumentiert.

8.12. Least Privilege praktisch durchsetzen

Wirksame Mechanismen:

- Rechteaenderungen nur über geprüfte Deploy- oder Admin-Prozesse.
- Keine informellen "nur kurz"-Freischaltungen ohne Ablauf.
- Sichtbare Eigentümer für jede privilegierte Rolle.
- Regelmäßige Reviews von Servicekonten, nicht nur von Personen.

---

9. Weiterführende Links + Uebungsaufgaben

9.1. Offizielle Kernquellen (IBM)

Versions- und Release-Stand:

- Fix Packs nach Version
- Db2 12.1 Doku-Einstieg
- Db2 11.5 Doku-Einstieg

Highlights und neue Funktionen:

- Highlights of Db2 11.5

Recovery:

- Rollforward overview

9.2. Uebungsaufgaben (steigende Schwierigkeit)

1. **Schema-Basis (leicht)**

- Implementiere `core.orders` und `core.order_items` mit passenden Schlüsseln.
- Lege Indizes für zwei Kernqueries an.
- Führe `RUNSTATS` aus und dokumentiere die ersten Explain-Ergebnisse.

2. **Isolation-Drill (leicht bis mittel)**

- Simuliere denselben fachlichen Pfad unter `CS`, `RS` und `RR`.
- Dokumentiere Unterschiede bei Locking und Laufzeit.

3. **Package-Cache-Analyse (mittel)**

- Erzeuge Last mit 5 bis 10 realistischen SQLs.
- Identifiziere über `MON_GET_PKG_CACHE_STMT` die teuersten Statements.
- Begruende für jedes Top-Statement die nächste Tuning-Hypothese.

4. **HADR-Drill (mittel)**

- Richte ein einfaches HADR-Szenario ein.
- Miss Lag und Verhalten unter Peak-Transaktionen.
- Dokumentiere Takeover-Schritte.

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

- Erzeuge Backup plus Logs.
- Restore auf Testumgebung.
- Rollforward bis zu definiertem Zeitpunkt.

6. **BLU-Vergleich (schwer)**

- Modelliere eine scan-lastige Reporting-Tabelle row-organized und column-organized.
- Vergleiche Laufzeit, Storage und Wartungsverhalten.

7. **Governance-Drill (schwer)**

- Implementiere Rollenmodell plus Auditierung für Kernobjekte.
- Prüfe Least Privilege und Nachvollziehbarkeit anhand eines Testfalls.

9.3. Abnahmekriterien für "produktionsreif"

- Kernqueries sind mit Explain und Messwerten abgesichert.
- Statistik- und Reorg-Strategie ist dokumentiert und getestet.
- Backup, Restore und Rollforward sind praktisch nachgewiesen.
- HADR- oder Verfuegbarkeitsmodell ist gegen RPO/RTO bewertet.
- Rollen, Audit und Betriebsverantwortungen sind klar dokumentiert.

9.4. Abschluss: Was "Pro-Level Db2" in der Praxis bedeutet

Pro-Level heisst nicht, jede Spezialoption zu kennen. Pro-Level heisst:

- Du trennst Workload-Typen sauber.
- Du modellierst Schlüssel, Indizes und Tabellenformen bewusst.
- Du nutzt Explain, Package Cache und Utilities systematisch.
- Du kannst Recovery und Verfügbarkeit nachweisen, nicht nur behaupten.
- Du hältst Governance und Rollenmodell betrieblich kontrollierbar. Wenn diese fünf Punkte erfüllt sind, ist Db2 kein schwerfaelliges Alt-System, sondern ein sehr belastbares Fundament für relationale Kernprozesse.

9.5. So setzt du es um - Checkliste

- Offizielle IBM-Quellen sind als Primarreferenz im Team-Wiki hinterlegt.
- Uebungen 1 bis 7 sind mit Ergebnissen dokumentiert.
- Für jede Uebung existieren Lernpunkte und offene Risiken.
- Architektur-, Betriebs- und Security-Review wurden gemeinsam abgeschlossen.
- Ein 90-Tage-Verbesserungsplan ist vereinbart.

9.6. 90-Tage-Umsetzungsfahrplan

Tage 1-30

- Kernqueries, Isolation-Level und Recovery-Ziele finalisieren.
- Tabellenklassen und Indexstrategie festziehen.
- Backup- und Restore-Baseline aufbauen.

Tage 31-60

- Package-Cache- und Explain-Routinen operationalisieren.
- HADR und Monitoring auf reale Last prüfen.
- Statistik- und Reorg-Zyklen stabilisieren.

Tage 61-90

- Restore-, Rollforward- und Failover-Drills fahren.
- Security- und Governance-Metriken etablieren.
- Kosten- und Kapazitaetsmodell gegen echte Datenentwicklung nachziehen.

9.7. Self-Assessment für Teams

Bewerte euer Team ehrlich mit 0 bis 2 Punkten je Aussage:

- Wir haben für alle Kernendpunkte dokumentierte SQL- und Isolation-Profile.
- Wir kennen unsere teuersten Statements im Package Cache.
- Wir wissen, welche Tabellen Reorg- oder Runstats-sensitiv sind.
- Wir haben Restore und Rollforward praktisch geprobt.
- Wir können HADR-Zustand gegen echte RPO-Ziele interpretieren.
- Unsere Rollen folgen nachweisbar Least Privilege.
- Unsere Alerts enthalten konkrete nächste Diagnoseaktionen.

Auswertung:

- 0-5 Punkte: Pionierphase, hohe Betriebsrisiken.
- 6-10 Punkte: solide Basis, aber inkonsistent.
- 11-14 Punkte: gutes Produktionsniveau.

9.8. Nächste Ausbaustufen

Wenn die Basis stabil steht, sind sinnvolle nächste Ausbaustufen:

- Lasttests als Pflichtbestandteil größerer Datenmodell-Aenderungen.
- Performance-Gates im Releaseprozess.
- Regelmäßige Governance- und Restore-Reviews.
- Stufenweise Entkopplung von Reporting- und OLTP-Last.
- Gezielte Erweiterung um BLU, WLM oder komplexere HA-Topologien nur bei echtem Nutzen. Langfristig bewaehrt sich ein quartalsweiser `Db2 Health Review`, der technische Kennzahlen nicht isoliert sammelt, sondern immer auf Fachwirkung zurueckbindet: Welche Endpunkte zeigen den staerksten Latenzdrift, welche Tabellen treiben Reorg- oder Temp-Kosten, wo verändert sich der Log-Druck unter neuen Produktmustern, welche Reports gehören fachlich eigentlich entkoppelt, und welche Security- oder Governance-Massnahmen sind tatsächlich wirksam geworden? Dieser Review sollte immer mit konkreten Folgeentscheidungen enden. Genau so bleibt Db2 ein aktiv gesteuertes Produktionssystem und nicht nur eine historisch gewachsene Datenbankplattform.

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 06.07.2026 wieder; maßgeblich ist die Online-Fassung unter https://www.thinkbigger.at/wissen/ibm-db2/