

O'REILLY®

Mit  
zahlreichen  
Illustrationen

# Daten- architekturen

Modern Data Warehouse, Data Fabric,  
Data Lakehouse und Data Mesh  
richtig einsetzen



James Serra

Übersetzung von Frank Langenau

---

# Das relationale Data Warehouse

Mitte der 2000er-Jahre hatte ich bereits jahrelang mit relationalen Datenbanken gearbeitet, aber ich hatte es noch nie mit relationalen Data Warehouses zu tun gehabt. Ich hatte als Datenbankadministrator (DBA) für ein Unternehmen gearbeitet, das seine finanziellen Transaktionen mithilfe eines Buchhaltungssoftwarepakets abwickelte. Da das Reporting (die Berichtserstellung) des Pakets beschränkt und langsam war, suchte das Unternehmen nach Möglichkeiten, die Performance zu verbessern, Dashboards zu erstellen, um die Daten detailliert zu analysieren, und seine Finanzdaten mit Daten aus einer selbst gestrickten Anwendung zu kombinieren, um die Geschäftsabläufe besser zu verstehen.

Mein Arbeitgeber beauftragte eine Beratungsfirma, ein sogenanntes »relationales Data Warehouse« aufzubauen, und bat mich dabei um Hilfe – eine Entscheidung, die den Verlauf meiner Karriere veränderte. Wir erstellten Dashboards, die den Endbenutzern einen Menge Zeit sparten und ihnen neue Geschäftseinblicke verschafften, die sie vorher nicht hatten. Als ich die Begeisterung auf ihren Gesichtern sah, wusste ich, dass ich meine neue Leidenschaft gefunden hatte. Ich änderte meine berufliche Laufbahn, konzentrierte mich auf Data Warehousing und blickte nie zurück.

## Was ist ein relationales Data Warehouse?

Ein *relationales Data Warehouse* (RDW) speichert und verwaltet zentral große Mengen strukturierter Daten, die aus mehreren Datenquellen kopiert wurden, um sie für das Erstellen von Berichten über Verlaufs- und Trendanalysen zu verwenden, damit ein Unternehmen bessere Geschäftsentscheidungen treffen kann. Die Bezeichnung »relational« geht auf das zugrunde liegende *relationale Modell* zurück, einen weitverbreiteten Ansatz zur Darstellung und Organisation von Daten für Datenbanken. Im relationalen Modell sind die Daten in Tabellen organisiert (die man auch als Relationen bezeichnet, daher der Name). Diese Tabellen bestehen aus Zeilen und Spalten, wobei jede Zeile eine Entität (zum

Beispiel einen Kunden oder ein Produkt) und jede Spalte ein Attribut dieser Entität (wie Name, Preis oder Menge) darstellt. Eine solche Datenbank wird als *Data Warehouse* bezeichnet, weil sie riesige Mengen strukturierter Daten aus verschiedenen Quellen – zum Beispiel aus Transaktionsdatenbanken, Anwendungssystemen und externen Data Feeds – sammelt, speichert und verwaltet.

Nicht alle Data Warehouses basieren auf dem relationalen Modell. Zu den *nicht relationalen Data Warehouses* gehören spaltenorientierte Data Warehouses, auf Graphdatenbanken basierende sowie NoSQL-Data-Warehouses. Allerdings sind relationale Data Warehouses weitaus beliebter und werden häufiger eingesetzt, vor allem weil relationale Datenbanken seit Jahrzehnten das vorherrschende Paradigma für die Datenverwaltung sind. Das relationale Modell eignet sich gut für strukturierte Daten, wie sie in Geschäftsanwendungen häufig vorkommen. Es ist auch deshalb so beliebt, weil SQL sehr verbreitet und seit vielen Jahren die Standardsprache für relationale Data Warehouses ist.

Ein RDW fungiert als zentrales Repository für viele Themenbereiche und enthält die *Single Version of Truth* (SVOT, dt. die einzige Version der Wahrheit). SVOT ist ein entscheidendes Konzept im Data Warehousing und bezieht sich auf die Schaffung einer einheitlichen, konsistenten Sicht auf die Daten eines Unternehmens. Das bedeutet, dass alle Daten innerhalb des Data Warehouse in einem standardisierten, strukturierten Format gespeichert sind und eine einzige genaue Version der Informationen darstellen. Dadurch wird sichergestellt, dass alle Benutzer auf dieselben Informationen zugreifen können, wodurch Diskrepanzen oder Unstimmigkeiten beseitigt und Datensilos vermieden werden. Dies verbessert die Entscheidungsfindung, die Zusammenarbeit und die Effizienz im gesamten Unternehmen. Außerdem verringert sich das Risiko von Fehlern und Missverständnissen, die durch die Arbeit mit unterschiedlichen inkonsistenten Datenquellen entstehen können.

Stellen Sie sich vor, Sie hätten kein Data Warehouse und erstellten Berichte direkt aus mehreren Quellsystemen und vielleicht sogar aus einigen Excel-Dateien. Wenn nun eine Leserin des Berichts die Richtigkeit der Daten infrage stellt, was können Sie ihr dann sagen? Die »Wahrheit« kann über so viele Quellsysteme verstreut sein, dass es schwierig ist, nachzuvollziehen, woher die Daten stammen. Hinzu kommt, dass einige Berichte für dieselben Daten unterschiedliche Ergebnisse liefern – wenn zum Beispiel zwei Berichte auf komplexe Logik zurückgreifen, um Daten aus mehreren Quellen zu beziehen, und die Logik nicht korrekt (oder gar nicht) aktualisiert wird. Alle Daten an einem zentralen Ort zu haben, bedeutet, dass das Data Warehouse die Single Source of Truth ist und alle Fragen zu den Berichtsdaten vom Data Warehouse beantwortet werden können. Eine SVOT zu pflegen, ist unerlässlich für Unternehmen, die das volle Potenzial ihrer Daten ausschöpfen wollen.

Wenn ein *Data Warehouse* (DW) vom gesamten Unternehmen genutzt wird, spricht man oft von einem *Enterprise Data Warehouse* (EDW). Dabei handelt es sich um eine umfassendere und robustere Version eines Data Warehouse, die auf die Bedürfnisse des gesamten Unternehmens zugeschnitten ist. Während ein standardmäßiges Data Warehouse vielleicht nur einige wenige Geschäftsbereiche unterstützt und im gesamten Unternehmen mit vielen Data Warehouses gearbeitet wird, verwendet das EDW eine breitere Palette von Datenquellen und Datentypen, um *sämtliche* Geschäftsbereiche zu unterstützen. Das EDW bietet eine einzige, einheitliche Sicht auf alle Daten des Unternehmens.

Abbildung 4-1 veranschaulicht einen wichtigen Grund, warum ein Data Warehouse eingerichtet werden sollte. Die Zeichnung zeigt auf der linken Seite, wie schwierig es ist, einen Bericht mit Daten aus mehreren Anwendungen zu verwenden, wenn Sie kein Data Warehouse haben. Jede Abteilung erstellt einen Bericht, der Daten aus allen Datenbanken sammelt, die mit den einzelnen Anwendungen verbunden sind. Es sind so viele Abfragen auszuführen, dass es zwangsläufig zu Performanceproblemen und falschen Daten kommt. Es ist ein Durcheinander! Die Zeichnung zeigt auf der rechten Seite, dass es für jede Abteilung sehr einfach ist, einen Bericht zu erstellen, ohne die Performance zu beeinträchtigen, wenn alle Anwendungsdaten in das EDW kopiert werden.

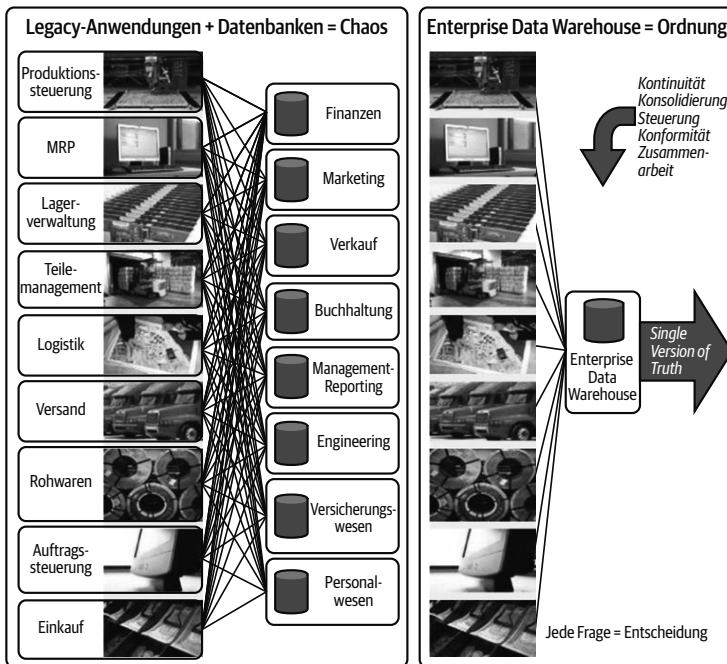


Abbildung 4-1: Situation vor und nach dem Einrichten eines Enterprise Data Warehouse

Um ein Data Warehouse aufzubauen, erstellen Sie in der Regel Datenpipelines, die die Daten in drei Schritten verarbeiten: Extrahieren, Transformieren, Laden (ETL):

1. Die Pipeline extrahiert Daten aus den Quellsystemen, wie zum Beispiel Datenbanken und linearen Dateien.
2. Die extrahierten Daten werden dann transformiert oder manipuliert, um den Anforderungen des Systems (in diesem Fall eines Data Warehouse) zu entsprechen. Hierzu kann das Bereinigen, Filtern, Aggregieren oder Kombinieren von Daten aus mehreren Quellen gehören.
3. Die transformierten Daten werden dann in das Data Warehouse geladen. Ein DBA kann die Namen der Datenbank und der Felder aussagekräftiger gestalten, sodass Endbenutzer einfacher und schneller Berichte erstellen können.

## Was ein Data Warehouse nicht ist

Nachdem Sie nun wissen, was ein Data Warehouse ist, wollen wir seinen Zweck klären, indem wir uns Projekte ansehen, die nicht als Data Warehouse angesehen werden sollten (obwohl ich das schon oft erlebt habe):

### *DW-Präfix*

Ein Data Warehouse ist nicht einfach eine Kopie einer Quelldatenbank aus einem operativen System mit dem Zusatz *DW* im Dateinamen. Nehmen wir an, Sie kopieren eine Datenbank namens *Finance*, die 50 operative Tabellen enthält, und nennen die Kopie *DW\_Finance*. Dann greifen Sie auf diese 50 Tabellen zu, um Ihre Berichte zu erstellen. Das würde zu einem Data Warehouse führen, das für *operative* Daten ausgelegt ist, obwohl Sie es eigentlich für *analytische* Daten benötigen. Mit analytischen Daten erhalten Sie eine bessere Leseperformance und können Datenmodelle erstellen, die es Endbenutzern erleichtern, Berichte zu erstellen. (Mehr dazu im nächsten Abschnitt.)

### *Views mit Unions*

Ein Data Warehouse ist keine Kopie mehrerer Tabellen aus verschiedenen Quellsystemen, die in einer SQL-View (Sicht) miteinander verknüpft sind. (Diese *Verknüpfung* geschieht über die SQL-Anweisung *UNION*, die die Ergebnisse von zwei oder mehr *SELECT*-Anweisungen zu einer einzigen Ergebnismenge zusammenführt.) Wenn Sie beispielsweise Daten aus drei Quellsystemen kopieren, die jeweils Kunden enthalten, würden Sie letztlich im Data Warehouse drei Tabellen namens *CustomerSource1*,

CustomerSource2 und CustomerSource3 haben. Somit müssten Sie eine View namens CustomerView erstellen, d.h. eine SELECT-Anweisung, die die Tabellen CustomerSource1, CustomerSource2 und CustomerSource3 per UNION zusammenführt. Diesen Vorgang würden Sie dann für andere Tabellen wie etwa für Produkte und Aufträge wiederholen.

Stattdessen sollten die Daten aus den drei Tabellen im Data Warehouse in eine Tabelle kopiert werden, was den zusätzlichen Aufwand erfordert, ein Datenmodell zu erstellen, das zu allen drei Tabellen passt. An dieser Stelle werden Sie wahrscheinlich auf die Stammdatenverwaltung (*Master Data Management*, MDM) zurückgreifen, mit der sich Kapitel 6 beschäftigt, um Duplikate zu vermeiden und die Zugänglichkeit und Performance zu verbessern.

### *Müllhalde*

Ein Data Warehouse ist keine Müllhalde für Tabellen. Dieses Phänomen tritt häufig auf, wenn ein Unternehmen kein Data Warehouse hat und ein Endbenutzer einen Bericht aus einer Teilmenge von Daten aus einer Reihe von Quellsystemen erstellen möchte. Um ihm schnell zu helfen, erstellt eine Person aus der IT-Abteilung ohne großes Nachdenken ein Data Warehouse und kopiert die Daten aus diesen beiden Quellsystemen in das Data Warehouse. Andere Endbenutzer bekommen dann Wind von dem Vorteil, den der erste Endbenutzer genießen durfte, und wollen weitere Daten aus denselben Quellsystemen und einigen anderen, um ihre eigenen Berichte zu erstellen. Wieder kopiert der IT-Mitarbeiter die angeforderten Daten schnell in das Data Warehouse. Dieses Vorgehen wiederholt sich immer wieder, bis das Data Warehouse ein Wirrwarr aus Datenbanken und Tabellen geworden ist.

Viele Data Warehouses beginnen deshalb als einmalige Lösung für einige wenige Benutzer und entwickeln sich dann zu ausgewachsenen, aber schlecht konzipierten Data Warehouses für das gesamte Unternehmen. Es gibt einen besseren Weg.

Wenn die erste Anfrage eines Endbenutzers eingeht, sollten Sie stattdessen den Berichtsbedarf Ihres Unternehmens ermitteln. Finden Sie heraus, ob es sich wirklich um eine einmalige Anfrage handelt oder ob dies der Zeitpunkt sein sollte, ein Enterprise Data Warehouse aufzubauen. Wenn das zutrifft, ist dies Ihre Chance, den Führungskräften zu zeigen, warum Ihr Unternehmen ein Data Warehouse braucht. Dann aber sollten Sie auch darauf bestehen, dass Sie im Vorfeld genügend Zeit bekommen, um ein Data Warehouse zu entwickeln, das viele Datenquellen und Endbenutzer unterstützen kann. (Nutzen Sie den Abschnitt »Warum ein relationales Data Warehouse verwenden?« auf Seite 82, um Ihre Argumentation zu untermauern.)

# Der Top-down-Ansatz

In einem relationalen Data Warehouse (RDW) müssen Sie im Vorfeld viel Arbeit leisten, um die Daten so aufzubereiten, dass Sie sie verwenden können, um Berichte zu erstellen. Diese Vorarbeit ist eine Entwurfs- und Implementierungsmethode, die man als *Top-down-Ansatz* bezeichnet. Dieser Ansatz eignet sich gut für Berichte mit historischen Daten, bei denen Sie zu ermitteln versuchen, was passiert ist (*deskriptive Analyse*) und warum es passiert ist (*diagnostische Analyse*). Beim Top-down-Ansatz legen Sie zunächst die Gesamtplanung, das Design und die Architektur des Data Warehouse fest und entwickeln dann die einzelnen Komponenten. Diese Methode betont, wie wichtig es ist, eine unternehmensweite Vision zu definieren und die strategischen Ziele und Informationsanforderungen des Unternehmens zu verstehen, bevor man mit der Entwicklung des Data Warehouse beginnt.

Deskriptive Analyse und diagnostische Analyse sind zwei wichtige Arten der Datenanalyse, die in der Wirtschaft häufig eingesetzt werden. Bei der deskriptiven Analyse analysiert man Daten, um vergangene oder aktuelle Ereignisse zu beschreiben, wobei häufig zusammenfassende Statistiken oder Datenvisualisierungen verwendet werden. Derartige Analysen nutzt man, um zu verstehen, was in der Vergangenheit geschehen ist, und um Muster oder Trends in den Daten zu erkennen, was bei der Entscheidungsfindung helfen kann.

Die *diagnostische Analyse* dient dazu, die Ursachen vergangener Ereignisse zu untersuchen, in der Regel durch Prüfen der Beziehungen zwischen verschiedenen Variablen oder Faktoren. Derartige Analysen können die Ursachen von Problemen ermitteln oder Probleme diagnostizieren, die sich auf die Unternehmensperformance auswirken können.

Angenommen, ein Unternehmen möchte die Verkaufsdaten des vergangenen Jahres analysieren. Bei der deskriptiven Analyse werden zusammenfassende Statistiken wie Gesamtumsatz, durchschnittlicher Umsatz pro Tag und Umsatz nach Produktkategorie berechnet, um zu verstehen, was passiert ist. Dagegen werden bei der diagnostischen Analyse die Beziehungen zwischen Faktoren (zum Beispiel Umsatz und Marketingausgaben oder Saisonalität und Kundendemografie) untersucht, um zu verstehen, warum der Umsatz im Laufe des Jahres schwankte. Indem beide Ansätze kombiniert werden, können Unternehmen ein tieferes Verständnis für ihre Daten gewinnen und sachkundigere Entscheidungen treffen.

Abbildung 4-2 zeigt die Architektur eines typischen RDW. Die Daten aus mehreren Quellen werden mithilfe von ETL in das RDW gebracht, in dem dann Berichte erstellt und andere Analysen durchgeführt werden können.

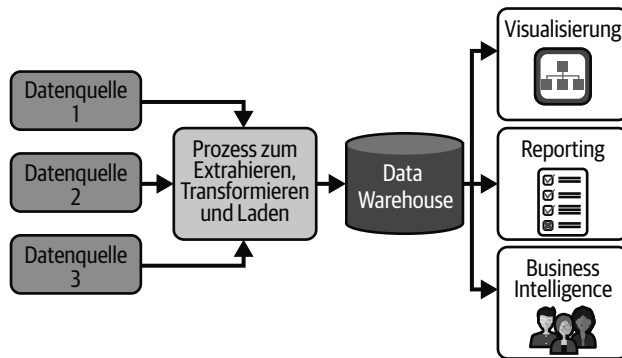


Abbildung 4-2: Architektur eines Data Warehouse

Der Top-down-Ansatz umfasst in der Regel die folgenden Schritte:

### 1. Vorab einige Hypothesen formulieren

Mit einem klaren Verständnis der Unternehmensstrategie beginnen. Machen Sie sich dann klar, welche Fragen Sie bezüglich der Daten stellen wollen.

### 2. Die Unternehmensanforderungen definieren

Identifizieren Sie die Ziele, Vorgaben und Leistungsindikatoren (*Key Performance Indicators*, KPIs) des Unternehmens. Erfassen und analysieren Sie den Informationsbedarf der verschiedenen Abteilungen und Benutzer. Diesen Schritt können Sie auch als Definition Ihrer Berichtsanforderungen betrachten.

### 3. Die Data-Warehouse-Architektur entwerfen

Erstellen Sie auf der Grundlage der Unternehmensanforderungen eine High-Level-Architektur für das Data Warehouse, einschließlich seiner Struktur, der Datenmodelle und der Datenintegrationsprozesse. Dies sind Ihre technischen Anforderungen.

### 4. Das Datenmodell entwickeln

Entwerfen Sie ein detailliertes Datenmodell für das Data Warehouse, wobei Sie die Beziehungen zwischen verschiedenen Datenentitäten und der Granularität der Daten berücksichtigen.

### 5. Die Architektur aufbauen

Entwickeln Sie die passenden Datenbanken, Schemas, Tabellen und Felder für das Data Warehouse. Dies ist der zuvor beschriebene Ansatz, der als *Schema-on-Write* bezeichnet wird.

## 6. ETL entwickeln

Entwickeln Sie die ETL-Prozesse, um Daten aus verschiedenen Quellsystemen zu extrahieren, sie in das gewünschte Format zu transformieren und sie in das Data Warehouse zu laden.

## 7. BI-Tools und -Anwendungen entwickeln und bereitstellen

Implementieren Sie BI-Tools und -Anwendungen, mit denen Benutzer auf die im Data Warehouse gespeicherten Daten zugreifen, sie analysieren und in Berichten verwenden können.

## 8. Das Data Warehouse testen und verfeinern

Führen Sie Tests durch, um Datenqualität, Performance und Zuverlässigkeit sicherzustellen. Nehmen Sie alle notwendigen Anpassungen vor, um das System zu optimieren.

## 9. Das Data Warehouse pflegen und erweitern

Wenn sich die Bedürfnisse des Unternehmens ändern, aktualisieren und erweitern Sie das Data Warehouse entsprechend.

Der Top-down-Ansatz hat einige Vorteile, wie zum Beispiel einen umfassenden Überblick über den Datenbedarf des Unternehmens, bessere Datenkonsistenz und verbesserte Governance. Allerdings kann dieser Ansatz auch zeit- und ressourcenaufwendig sein, sodass es länger dauert, bis er einen Mehrwert liefert, als es bei dem in Kapitel 5 beschriebenen Bottom-up-Ansatz des Data Lake der Fall ist. Die in Kapitel 10 beschriebene Data-Warehouse-Architektur kombiniert die Top-down- und Bottom-up-Ansätze.

# Warum ein relationales Data Warehouse verwenden?

Mit einem RDW lassen sich alle Arten von BI-Projekten viel einfacher erstellen, da BI-Projekte Daten direkt aus dem RDW abrufen können, ohne eine komplexe Logik zum Abrufen von Daten aus mehreren Quellsystemen erstellen zu müssen. Außerdem brauchen Sie die Daten weder zu bereinigen noch zusammenzuführen, da dies bereits das RDW erledigt hat. Das BI-Projekt, das aus dem RDW aufgebaut wird, könnte ein Data Mart sein (der eine Teilmenge der RDW-Daten für eine bestimmte Gruppe von Personen enthält, wie Kapitel 6 erläutert), Daten aggregieren, um Abfragen und Berichte schneller zu erstellen, und sogar in Microsoft Excel verwendbar sein. Fazit: Mit einem RDW haben Sie bereits eine solide Grundlage, auf der Sie aufbauen können.

Sehen wir uns nun einige der wichtigsten Vorteile an, die Sie durch den Einsatz eines RDW erzielen können:

#### *Die Belastung des Produktionssystems verringern*

Dieses Problem haben Sie vielleicht schon einmal erlebt: Ein verärgertes Endbenutzer ruft an und beschwert sich darüber, dass das Einfügen von Aufträgen über die Auftrags erfassungsanwendung ewig dauert. Sie gehen der Sache nach, und es stellt sich heraus, dass ein anderer Endbenutzer über die Auftrags eingabe anwendung einen Bericht ausführt und damit alle Ressourcen auf dem Server, auf dem die Anwendung läuft, in Anspruch nimmt. Diese Situation tritt besonders häufig auf, wenn Endbenutzer Ad-hoc-Abfragen erstellen dürfen und diese noch dazu mit schlecht geschriebenem SQL ausführen können.

Indem Sie die Datenbank für die Auftrags erfassungsanwendung in ein Data Warehouse kopieren und optimieren, können Sie alle Berichte und Ad-hoc-Abfragen auf das DW verlagern und dieses Problem gänzlich vermeiden, insbesondere wenn der Endbenutzer einen Bericht ausführen muss, der auf mehrere Anwendungsdatenbanken zugreift.

#### *Für Lesezugriff optimieren*

Anwendungsdatenbanken werden so optimiert, dass sie alle CRUD-Operationen gleichermaßen unterstützen, sodass das Lesen von Daten nicht so schnell sein wird, wie es sein könnte. Andererseits ist das Data Warehouse ein einmal beschreibbares, mehrfach lesbares System, d. h., wir verwenden es hauptsächlich zum Lesen von Daten. Demzufolge lässt es sich für den Lesezugriff optimieren, insbesondere für die zeitaufwendigen sequenziellen Plattenscans, die häufig bei der Ausführung von Berichten oder Abfragen auftreten. Es gibt viele Datenbanktechniken, mit denen sich der Lesezugriff in einem Data Warehouse beschleunigen lässt – manche zum Nachteil des Schreibzugriffs, um den wir uns hier aber nicht kümmern.

#### *Mehrere Datenquellen integrieren*

Die Möglichkeit, viele Datenquellen zu integrieren, um nützlichere Berichte zu erstellen, ist einer der wichtigsten Gründe, ein Data Warehouse aufzubauen. Alle an einem Ort zu versammeln, anstatt sie über verschiedene Datenbanken zu verteilen, erleichtert es nicht nur, Berichte zu erstellen, sondern verbessert auch die Performance des Reportings (des Erstellens der Berichte) erheblich.

#### *Genaue historische Berichte erstellen*

Ohne ein Data Warehouse führen die Endbenutzer normalerweise alle ihre Berichte jeden Monat an einem bestimmten (in der Regel am letzten) Tag aus. Dann speichern sie diese auf dem Datenträger, damit sie auch in

Zukunft darauf zurückgreifen können. Ein Beispiel: Der Benutzer möchte einen Bericht von vor ein paar Monaten einsehen, in dem die Kundenverkäufe nach Bundesland aufgelistet sind. Nun ist aber ein Kunde vor Kurzem in ein anderes Bundesland umgezogen. Wenn der Benutzer einen aktuellen Bericht ausführt, würde dieser fälschlicherweise die Umsätze dieses Kunden in seinem neuen Bundesland und nicht in seinem alten Bundesland anzeigen (da sein Datensatz in der Datenbank auf sein neues Bundesland aktualisiert wurde). Daher muss der Benutzer auf einen gespeicherten älteren Bericht zurückgreifen, anstatt einen aktuellen Bericht auszuführen. Ein Data Warehouse kann derartige Situationen berücksichtigen, indem es verzeichnet, wann ein Kunde umzieht (über die Verfolgung des Kundenstandortverlaufs mit Anfangs- und Enddatum), und auch andere Felder einbezieht, die überwacht werden müssen (zum Beispiel Arbeitgeber oder Einkommen). Nun kann der Benutzer am aktuellen Tag einen Bericht erstellen, aber verlangen, dass die Daten ab einem bestimmten Datum in der Vergangenheit abzurufen sind – und der Bericht wird korrekt sein. Außerdem ist es nicht mehr erforderlich, die Berichtsdateien jeden Monat zu speichern.

#### *Tabellen umstrukturieren und umbenennen*

Viele Datenbanken von Anwendungen verwenden Tabellen- und Feldnamen, die sehr schwer zu verstehen sind. Das gilt insbesondere für ältere ERP- und CRM-Produkte (denken Sie an Tabellennamen wie T116 und Feldnamen wie RAP16). Im Data Warehouse können Sie die Daten aus diesen Quelltabellen kopieren und dabei mit aussagekräftigeren Namen versehen (zum Beispiel statt T116). Zudem können Sie wahrscheinlich ein besseres Datenmodell für alle diese Tabellen entwerfen. Endbenutzer werden Berichte viel einfacher erstellen können, wenn sie keine kryptischen Tabellen- und Feldnamen interpretieren müssen.

#### *Vor Upgrades von Anwendungen schützen*

Stellen Sie sich vor, Sie hätten kein DW und die Benutzer erstellten stattdessen Berichte mit einer Anwendungsdatenbank. Alles läuft gut, und dann treten plötzlich in vielen Berichten Fehler auf. Es zeigt sich, dass die Anwendung ein Upgrade durchlaufen hat. Bei der neu installierten Version wurden mehrere Tabellen und Felder umbenannt. Nun müssen Sie jeden einzelnen von Hunderten Berichten durchgehen und die geänderten Tabellen und Felder umbenennen. Das kann Monate dauern und eine Menge verärgelter Endbenutzer zur Folge haben. Selbst danach können Berichte, die übersehen wurden, noch Fehler verursachen.

Ein DW kann Sie davor schützen. Nach einem Anwendungsupgrade muss nur der ETL-Prozess aktualisiert werden, der Daten aus den Anwendungsdatenbanken in das DW kopiert – eine schnell zu erledigende Aufgabe. Die Berichte müssen nicht geändert werden. Die Endbenutzer sehen zwar keine neuen Daten, bis die ETL-Pipeline korrigiert ist, aber ihre Berichte sind nicht fehlerhaft.

#### *Sicherheitsbedenken verringern*

Ohne ein Data Warehouse müsste Ihr Team jedem Endbenutzer Sicherheitszugriff auf jede Anwendungsdatenbank gewähren, die er für seine Berichte benötigt. Das könnten Dutzende sein; die Zuteilung von Berechtigungen könnte Wochen dauern, und manchmal hätten sie immer noch keinen Zugriff auf alles, was sie brauchen. Mit einem DW braucht jeder Endbenutzer nur auf die entsprechenden Tabellen zuzugreifen, was viel schneller und einfacher ist.

#### *Historische Daten bewahren*

Viele Produktionssysteme beschränken den Umfang der historischen Daten, die sie aufbewahren (zum Beispiel die Daten der letzten drei Jahre). Das geschieht, um Speicherplatz zu sparen und die Performance zu verbessern, in einigen Fällen aber auch, um Vorschriften einzuhalten. Ältere Daten werden in der Regel jährlich oder monatlich eliminiert. Andererseits kann ein DW die gesamte Historie speichern, sodass Sie sich keine Sorgen machen müssen, wenn Sie einen Bericht für weiter zurückliegende Jahre ausführen sollen, dafür aber keine Daten mehr finden.

#### *Stammdatenverwaltung (Master Data Management, MDM)*

Wenn Sie Daten aus mehreren Quellsystemen sammeln, müssen Sie häufig MDM verwenden, um doppelte Datensätze für Entitäten wie Kunden, Produkte und Güter zu entfernen. (Kapitel 6 geht im Detail auf MDM ein.) Das DW ist der ideale Ort, um MDM durchzuführen. Zudem ist es mit vielen MDM-Tools möglich, Hierarchien zu erstellen (zum Beispiel Unternehmen \_\$pfeil\_ Abteilung \_\$pfeil\_ Mitarbeiter), was das Erstellen von Stammdaten noch wertvoller macht.

#### *Die Datenqualität verbessern, indem Löcher in Quellsystemen geschlossen werden*

Viele Daten, die Sie aus den verschiedenen Quellsystemen erhalten, werden Sie bereinigen müssen, egal was die Besitzer der Anwendungen behaupten. (Ich habe sie oft sagen hören: »Unsere Daten sind sauber«, nur um dann das Gegenteil zu erfahren.) Wenn beispielsweise eine Anwendung zur Auftragseingabe das Geburtsdatum eines Kunden verlangt und die Person, die die Daten eingibt, das Geburtsdatum des Kunden nicht kennt, gibt sie vielleicht ein Datum in der Zukunft oder ein Datum einer

mehr als 100 Jahre alten Person ein, nur um den Auftrag abschließen zu können. Oder die Anwendung unterlässt es, die Richtigkeit des Buchstaben-codes für ein Bundesland zu prüfen (zum Beispiel DE-SA für Sachsen statt DE-SN)<sup>1</sup>. Es gibt immer Dutzende von »Löchern« im Quellsystem. Es ist allerdings nicht damit getan, die Daten im DW zu bereinigen, Sie müssen auch die Verantwortlichen benachrichtigen, damit sie in ihren Systemen die Anwendungen auf solche unvollständigen Datensätze vorbereiten und Fehler beseitigen können. Auf diese Weise tragen Sie dazu bei, die Eingabe fehlerhafter Daten in Zukunft zu verhindern.

#### *Beteiligung der IT beim Erstellen von Berichten eliminieren*

Dies geht zurück auf die in Kapitel 3 erwähnte Self-Service BI: Wird ein geeignetes Data Warehouse aufgebaut, entfällt die Notwendigkeit, die IT-Abteilung in das Erstellen von Berichten einzubeziehen. Diese Aufgabe liegt dann in den Händen der Endbenutzer. Ohne den Engpass der begrenzten IT-Ressourcen können Berichte und Dashboards früher erstellt werden. Und die IT-Abteilung wird dankbar sein, dass sie sich mit interessanteren Projekten beschäftigen kann als mit der Erstellung von Berichten!

## **Nachteile bei Verwendung eines relationalen Data Warehouse**

Es gibt immer Kompromisse, und wenn Sie ein RDW aufbauen, sind folgende Nachteile zu beachten:

#### *Komplexität*

Es kann kompliziert und zeitaufwendig sein, ein Data Warehouse zu entwickeln, zu erstellen und zu warten. Die erforderlichen Fachkenntnisse und Ressourcen können die Kosten hochtreiben.

#### *Hohe Kosten*

Ein Data Warehouse zu implementieren, kann teuer sein und bedeutet erhebliche Investitionen in Hardware, Software und Personal. Laufende Wartung und Upgrades können ebenfalls zu den Kosten beitragen.

#### *Herausforderungen bei der Datenintegration*

Die Integration von Daten aus verschiedenen Quellen kann sich als schwierig erweisen, da sie mit unterschiedlichen Datenformaten, Strukturen und Qualitätsproblemen zu tun haben kann. Infolgedessen müssen

---

<sup>1</sup> Siehe <https://de.wikipedia.org/wiki/Bundesl%C3%A4nderk%C3%BCrzel> für die Codes der deutschen Länder.

Zeit und Mühe für die Datenbereinigung und -vorverarbeitung aufgewendet werden. Darüber hinaus ist es bei bestimmten Daten wie zum Beispiel Streaming-Daten von IoT-Geräten zu schwierig, sie in ein RDW einzubringen, und somit gehen die potenziellen Erkenntnisse aus diesen Informationen verloren.

#### *Zeitaufwendige Datentransformation*

Damit Daten in ein Data Warehouse geladen werden können, müssen sie möglicherweise transformiert werden, um dem Datenmodell des Data Warehouse zu entsprechen. Dieser Prozess kann sehr zeitaufwendig sein, und Fehler bei der Datentransformation können zu ungenauen Analysen führen.

#### *Datenlatenz*

Da Data Warehouses darauf ausgelegt sind, große Datenmengen zu verarbeiten, können sie langsamer sein als andere Arten von Datenbanken. Dies kann zu einer Datenlatenz führen, bei der die Daten im Warehouse nicht mit den jüngsten Änderungen in den Quelldatenbanken übereinstimmen.

#### *Wartungsfenster*

Bei einem RDW ist in der Regel ein Wartungsfenster erforderlich. Das Laden und Bereinigen der Daten ist sehr ressourcenintensiv, und wenn Benutzer dann versuchen, gleichzeitig Berichte auszuführen, werden sie eine sehr geringe Performance spüren. Daher müssen die Benutzer während der Wartungsarbeiten aus dem Warehouse ausgesperrt werden, was einen 24/7-Zugriff verhindert. Treten während des Wartungsfensters Probleme auf, wie zum Beispiel ein fehlgeschlagener ETL-Auftrag, müssen Sie das Wartungsfenster möglicherweise verbreitern. Wenn Benutzer versuchen, Berichte auszuführen, und immer noch ausgesperrt sind, verärgert sie das, weil sie ihre Arbeit nicht erledigen können.

#### *Eingeschränkte Flexibilität*

Data Warehouses sind konzeptionell darauf ausgelegt, bestimmte Arten von Analysen zu unterstützen, was ihre Flexibilität für andere Arten der Verarbeitung oder Analyse von Daten einschränken kann. Unter Umständen müssen Sie zusätzliche Tools oder Systeme in das Warehouse integrieren, um bestimmte Anforderungen zu erfüllen.

#### *Bedenken in Bezug auf Sicherheit und Datenschutz*

Große Mengen vertraulicher Daten an einem zentralen Ort zu speichern, kann das Risiko von Datenschutzverletzungen erhöhen und erfordert daher strenge Sicherheitsmaßnahmen.

# Ein Data Warehouse füllen

Da sich die Quelltabellen, die in ein Data Warehouse eingespeist werden, im Laufe der Zeit ändern, muss das DW diese Änderungen widerspiegeln. Das klingt einfach, verlangt aber, viele Entscheidungen zu treffen: Wie oft sind die Daten zu *extrahieren* (oder abzurufen), welche Extraktionsmethode ist zu verwenden, wie sind die Daten physisch zu extrahieren, und wie wird festgestellt, welche Daten sich seit dem letzten Extrahieren geändert haben? Auf die einzelnen Fragen werde ich im Folgenden eingehen.

## Wie oft sollen die Daten extrahiert werden?

Wie oft Sie das Data Warehouse aktualisieren müssen, hängt weitgehend davon ab, wie oft die Quellsysteme aktualisiert werden und wie zeitnah das Reporting durch den Endbenutzer sein muss. Oftmals möchten die Endbenutzer nicht die Daten des aktuellen Tages sehen, sondern lieber alle Daten bis zum Ende des Vortags. In diesem Fall können Sie Ihre Aufträge zur Extraktion der Daten aus den Quellsystemen über ETL-Tools jede Nacht ausführen, nachdem die Datenbanken der Quellsysteme aktualisiert wurden. Dabei erstellen Sie ein nächtliches Wartungsfenster, in dem der gesamte Datentransfer stattfindet. Wenn Endbenutzer tagsüber Updates benötigen, ist eine häufigere Extraktion erforderlich, beispielsweise stündlich.

Zu berücksichtigen ist auch die Größe der Daten für jeden Extraktionslauf. Wenn der Umfang sehr groß ist, dauert die Aktualisierung des Data Warehouse vielleicht zu lange, sodass Sie das Update in kleinere Blöcke aufteilen und entsprechend häufiger extrahieren und aktualisieren müssen (zum Beispiel stündlich statt täglich). Außerdem kann es zu lange dauern, große Datenmengen von den Quellsystemen in das Data Warehouse zu übertragen, insbesondere wenn die Quelldaten lokal gespeichert sind und Sie keine große Pipeline vom Quellsystem zum Internet haben. Dies ist ein weiterer Grund, von einer großen nächtlichen Übertragung zu kleineren stündlichen Übertragungen im Laufe des Tages überzugehen.

## Extraktionsmethoden

Daten aus Quellsystemen lassen sich prinzipiell nach zwei Methoden extrahieren. Schauen wir uns beide an:

### *Vollständige Extraktion*

Bei einer vollständigen Extraktion werden alle Daten vollständig aus einer oder mehreren Tabellen im Quellsystem extrahiert. Dies funktioniert am

besten bei kleineren Tabellen. Da diese Extraktion alle Daten widerspiegelt, die derzeit im Quellsystem verfügbar sind, ist es nicht notwendig, Änderungen zu verfolgen, sodass sich diese Methode sehr einfach realisieren lässt. Die Quelldaten werden so bereitgestellt, wie sie sind, und Sie benötigen keine zusätzlichen Informationen (z. B. Zeitstempel).

### *Inkrementelle Extraktion*

Bei der inkrementellen Extraktion werden nur die Daten abgerufen, die sich seit einem bestimmten Zeitpunkt (wie zum Beispiel der letzten Extraktion oder dem Ende eines Abrechnungszeitraums) geändert haben, und nicht die gesamte Tabelle. Dies funktioniert am besten bei großen Tabellen und auch nur, wenn es möglich ist, alle geänderten Informationen zu identifizieren (siehe unten).

Für die meisten Quellsysteme werden Sie diese beiden Methoden kombinieren.

Unabhängig davon, ob Sie eine vollständige oder eine inkrementelle Extraktion ausführen, gibt es zwei Modi, die Daten zu extrahieren: online und offline.

Bei der Onlineextraktion kann sich der Extraktionsprozess direkt mit dem Quellsystem verbinden, um auf die Quelltabellen zuzugreifen, oder er kann die Verbindung zu einem Zwischensystem herstellen, das Änderungen an den Daten in einer vorab konfigurierten Art und Weise speichert (zum Beispiel in Transaktionsprotokollen oder Änderungstabellen).

Allerdings ist ein direkter Zugriff auf das Quellsystem nicht immer gegeben. In derartigen Fällen werden die Daten außerhalb des ursprünglichen Quellsystems geparkt und durch eine Extraktionsroutine, die vom Quellsystem ausgeht, erzeugt (zum Beispiel führt ein Mainframe eine Extraktionsroutine auf einer Tabelle aus und legt die Daten in einem Ordner in einem Dateisystem ab). Die extrahierten Daten werden in der Regel in eine einfache Datei geschrieben, und zwar in einem definierten, generischen Format (zum Beispiel CSV oder JSON).

## **Wie man feststellt, welche Daten sich seit der letzten Extraktion geändert haben**

Leider ist es bei vielen Quellsystemen schwierig, die kürzlich modifizierten Daten zu identifizieren, um sie inkrementell zu extrahieren. Die folgenden Techniken zeigen, wie Sie kürzlich modifizierte Daten identifizieren und aus den Quellsystemen extrahieren. Diese Techniken lassen sich in Verbindung mit den diskutierten Methoden zur Datenextraktion einsetzen. Einige Techniken beruhen auf den Eigenschaften der Quellsysteme, andere verlangen möglicher-

weise Änderungen an den Quellsystemen. Die Besitzer des Quellsystems sollten jede Technik vor der Implementierung sorgfältig prüfen:

### *Zeitstempel*

Zeitstempel sind die bevorzugte Option und am einfachsten zu implementieren. Die Tabellen in manchen operativen Systemen enthalten Zeitstempelspalten mit der Uhrzeit und dem Datum der letzten Änderung einer bestimmten Zeile, sodass die neuesten Daten leicht zu erkennen sind. In relationalen Datenbanken hat die Zeitstempelspalte oftmals den Datentyp `timestamp` oder `datetime`, wobei die Spalten mit `Timestamp` oder `Last Modified` benannt sind. Die Quellenanwendung trägt die aktuellen Werte in diese Spalte ein. Wenn das nicht der Fall ist, können Sie die relationale Datenbank so einrichten, dass sie das aktuelle Datum beim Speichern eines Datensatzes aufzeichnet. Eine andere Möglichkeit besteht darin es, der Datenbank einen Trigger hinzuzufügen, um die Spalte zu füllen.

### *Änderungsdaten erfassen*

Die meisten relationalen Datenbanken unterstützen das *Erfassen von Änderungsdaten* (*Change Data Capture*, CDC). Dieses Feature zeichnet auf, wann `INSERT`-, `UPDATE`- und `DELETE`-Anweisungen auf Datenbanktabellen angewendet werden, und stellt einen Tabellendatensatz zur Verfügung, der angibt, was wo wann geändert wurde. Die Grundlage hierfür ist das Transaktionsprotokoll der Datenbank. Wenn Sie Änderungsdaten in Echtzeit benötigen, d. h., wenn die Änderungen am Quellsystem innerhalb weniger Sekunden im Data Warehouse widergespiegelt werden sollen, kann CDC die Schlüsseltechnologie sein.

### *Partitionierung*

Manche Quellsysteme verwenden eine Bereichspartitionierung, bei der die Quelltabellen nach einem Datumsschlüssel aufgeteilt werden, was die Identifizierung neuer Daten erleichtert. Wenn Sie beispielsweise aus einer nach Tagen partitionierten Auftragsstabelle Daten extrahieren, ist es einfach, die Daten des aktuellen oder vorherigen Tags zu identifizieren.

### *Datenbank-Tripper*

Für die Anweisungen `INSERT`, `UPDATE` und `DELETE` können Sie Trigger zu einer einzelnen Tabelle hinzufügen und diese Trigger die Informationen über die Datensatzänderung in eine »Änderungstabelle« schreiben lassen. Dies ähnelt der Erfassung von Änderungsdaten. Verwenden Sie also CDC, wenn Ihr Datenbankprodukt dies unterstützt, und andernfalls Trigger.

### *MERGE-Anweisung*

Die Option, die am wenigsten zu empfehlen ist, besteht darin, eine vollständige Extraktion aus dem Quellsystem in einen Staging-Bereich im

Data Warehouse vorzunehmen und dann diese Tabelle mit einem früheren vollständigen Extrakt aus dem Quellsystem zu vergleichen. Dabei kommt eine MERGE-Anweisung zum Einsatz, um die geänderten Daten zu identifizieren. Bei dieser Methode müssen Sie alle Quellfelder mit allen Zielfeldern vergleichen (oder eine Hashfunktion zu Hilfe nehmen). Auf das Quellsystem hat dieser Ansatz höchstwahrscheinlich kaum einen Einfluss, er kann aber für das Data Warehouse eine beträchtliche Belastung darstellen, insbesondere wenn es sich um große Datenmengen handelt. In der Regel ist diese Option die letzte Möglichkeit, wenn keine anderen Optionen infrage kommen.

## Der Tod des relationalen Data Warehouse wurde übertrieben dargestellt

Anfang der 2010er-Jahre begannen IT-Fachleute zu fragen, ob das relationale Data Warehouse überhaupt noch gebraucht wird: »Ist das relationale Data Warehouse tot?« Viele verstanden dies als Frage, ob Unternehmen überhaupt noch Data Warehouses benötigen. Dass dies der Fall ist, wird in diesem Kapitel dargelegt. Doch in Wirklichkeit bezieht sich die Frage auf die Data-Warehouse-Architektur – kann man einfach einen Data Lake verwenden, oder sollte man sowohl einen Data Lake als auch ein RDW nehmen?

Als Data Lakes auf der Bühne erschienen, basierten sie auf der Apache-Hadoop-Technologie, und es waren vor allem die Hadoop-Anbieter, die das RDW für tot erklärten. »Stellen Sie einfach alle Ihre Daten in den Data Lake und werden Sie Ihr RDW los«, rieten sie. Wie in Kapitel 2 erwähnt, sind alle Projekte gescheitert, die das versucht haben.

Viele Jahre lang war ich der Meinung, dass RDWs immer benötigt werden würden, weil Data Lakes alle auf Hadoop basieren und es einfach zu viele Einschränkungen gibt. Doch als Lösungen wie Delta Lake (siehe Kapitel 12) verfügbar wurden und Data Lakes bessere, einfacher zu verwendende Produkte als Hadoop verwendeten (siehe Kapitel 16), begegneten mir erstmals einige Anwendungsfälle, in denen eine Lösung ohne RDW funktionieren könnte. Diese Art von Lösung ist eine Data-Lakehouse-Architektur, mit der sich Kapitel 12 beschäftigt.

Allerdings gibt es immer noch viele Anwendungsfälle, in denen ein RDW erforderlich ist. Und obwohl sich die Data-Lake-Technologien zweifellos weiter verbessern, was die Bedenken hinsichtlich der Umgehung eines RDW teilweise oder ganz aus dem Weg räumt (siehe Kapitel 12), werden wir RDWs nie ganz

abschaffen können. Dafür gibt es meiner Meinung nach drei Gründe. Erstens ist es immer noch schwieriger, aus einem Data Lake zu berichten als aus einem Data Warehouse. Zweitens erfüllen RDWs weiterhin den Informationsbedarf der Benutzer und bieten einen Mehrwert. Drittens nutzen viele Menschen Data Warehouses, sind von ihnen abhängig, vertrauen ihnen und wollen sie nicht durch Data Lakes ersetzen.

Data Lakes bieten eine reiche Datenquelle für Data Scientists und Self-Service-Datenkonsumenten (*Power User*). Zudem erfüllen sie die Anforderungen für Analysen und Big Data. Aber nicht alle Daten- und Informationsarbeiter wollen Power User werden. Die Mehrheit benötigt weiterhin gut integrierte, systematisch bereinigte, leicht zugängliche relationale Daten, die ein Verlaufsprotokoll enthalten, das die Entwicklung oder den Fortschritt der Dinge über einen bestimmten Zeitraum erfasst. Diese Menschen sind mit einem Data Warehouse am besten bedient.

## Zusammenfassung

Dieses Kapitel hat die erste weithin verwendete Technologielösung behandelt, um Daten von mehreren Quellen zu zentralisieren und Berichte darüber zu schreiben: das relationale Data Warehouse (RDW). Das RDW hat die Art und Weise, wie Unternehmen und Organisationen ihre Daten verwalten, revolutioniert, indem es ein zentrales Repository zum Speichern und Abrufen von Daten bereitgestellt und so eine effizientere Datenverwaltung und -analyse ermöglicht. Durch die Möglichkeit, Daten strukturiert zu speichern und zu organisieren, sind Benutzer eines RDW in der Lage, schnell und einfach komplexe Abfragen und Berichte zu erstellen, die wertvolle Erkenntnisse liefern und die Entscheidungsfindung unterstützen.

Heute ist das relationale Data Warehouse nach wie vor ein grundlegender Bestandteil vieler Datenarchitekturen und findet sich in einem breiten Spektrum von Branchen – vom Finanzwesen über das Gesundheitswesen bis hin zum Einzelhandel und der Fertigung. Das folgende Kapitel erörtert die nächste Technologie, die sich zu einem wichtigen Faktor bei der Zentralisierung von Daten und der Erstellung von Berichten entwickelt: den Data Lake.

<b>Vorwort</b>	<b>19</b>
<b>Einführung</b>	<b>21</b>

## Teil I: Grundlagen

<b>1</b>	<b>Big Data</b>	<b>27</b>
	Was ist Big Data, und wie kann Big Data Ihnen helfen? . . . . .	28
	Data Maturity . . . . .	32
	Stufe 1: Reaktiv . . . . .	33
	Stufe 2: Informativ . . . . .	34
	Stufe 3: Prädiktiv . . . . .	34
	Stufe 4: Transformativ . . . . .	35
	Self-Service Business Intelligence . . . . .	35
	Zusammenfassung . . . . .	36
<b>2</b>	<b>Arten von Datenarchitekturen</b>	<b>39</b>
	Entwicklung von Datenarchitekturen . . . . .	40
	Relationales Data Warehouse . . . . .	43
	Data Lake . . . . .	45
	Modern Data Warehouses . . . . .	48
	Data Fabric . . . . .	48
	Data Lakehouse . . . . .	49
	Data Mesh . . . . .	50
	Zusammenfassung . . . . .	51

<b>3</b>	<b>Die Architektur-Design-Sitzung</b>	<b>53</b>
	Was ist eine ADS? .....	53
	Warum eine ADS abhalten? .....	54
	Vor der ADS .....	55
	Vorbereiten .....	55
	Teilnehmerinnen und Teilnehmer einladen .....	58
	Die ADS leiten .....	60
	Einführungen .....	60
	Erkundung .....	61
	Whiteboarding .....	67
	Nach der ADS .....	68
	Tipps für die Durchführung einer ADS .....	70
	Zusammenfassung .....	72

## Teil II: Allgemeine Datenarchitekturkonzepte

<b>4</b>	<b>Das relationale Data Warehouse</b>	<b>75</b>
	Was ist ein relationales Data Warehouse? .....	75
	Was ein Data Warehouse nicht ist .....	78
	Der Top-down-Ansatz .....	80
	Warum ein relationales Data Warehouse verwenden? .....	82
	Nachteile bei Verwendung eines relationalen Data Warehouse .....	86
	Ein Data Warehouse füllen .....	88
	Wie oft sollen die Daten extrahiert werden? .....	88
	Extraktionsmethoden .....	88
	Wie man feststellt, welche Daten sich seit der letzten Extraktion geändert haben .....	89
	Der Tod des relationalen Data Warehouse wurde übertrieben dargestellt .....	91
	Zusammenfassung .....	92
<b>5</b>	<b>Data Lake</b>	<b>93</b>
	Was ist ein Data Lake? .....	94
	Warum einen Data Lake verwenden? .....	94

Bottom-up-Ansatz . . . . .	97
Best Practices für das Design von Data Lakes . . . . .	98
Mehrere Data Lakes . . . . .	105
Vorteile . . . . .	106
Nachteile . . . . .	109
Zusammenfassung . . . . .	110
<b>6 Lösungen und Prozesse zur Datenspeicherung</b>	<b>111</b>
Datenspeicherlösungen . . . . .	112
Data Marts . . . . .	112
Operational Data Stores . . . . .	113
Data Hubs . . . . .	116
Datenprozesse . . . . .	118
Stammdatenverwaltung . . . . .	119
Datenvirtualisierung und Datenföderierung . . . . .	120
Datenkataloge . . . . .	126
Datenmarktplätze . . . . .	127
Zusammenfassung . . . . .	129
<b>7 Ansätze für das Design</b>	<b>131</b>
OLTP vs. OLAP . . . . .	132
Operative und analytische Daten . . . . .	135
Symmetrisches Multiprocessing und massive Parallelverarbeitung . . . . .	135
Lambda-Architektur . . . . .	137
Kappa-Architektur . . . . .	140
Polyglotte Persistenz und polyglotte Datenspeicher . . . . .	142
Zusammenfassung . . . . .	143
<b>8 Ansätze zur Datenmodellierung</b>	<b>145</b>
Relationale Modellierung . . . . .	145
Schlüssel . . . . .	146
Entity-Relationship-Diagramme . . . . .	146
Normalisierungsregeln und -formen . . . . .	147
Änderungen verfolgen . . . . .	149

Dimensionale Modellierung . . . . .	149
Fakten, Dimensionen und Schlüssel . . . . .	149
Änderungen verfolgen . . . . .	150
Denormalisierung . . . . .	152
Common Data Model . . . . .	153
Data Vault . . . . .	154
Die Methodiken von Kimball und Inmon für das Data Warehousing . . . . .	156
Die Top-down-Methodik von Inmon . . . . .	157
Die Bottom-up-Methodik von Kimball . . . . .	159
Eine Methodik auswählen . . . . .	160
Hybride Modelle . . . . .	161
Mythen über die Methodiken . . . . .	164
Zusammenfassung . . . . .	167
<b>9 Ansätze für die Datenaufnahme</b>	<b>169</b>
ETL vs. ELT . . . . .	169
Reverse ETL . . . . .	172
Stapel- vs. Echtzeitverarbeitung . . . . .	173
Stapelverarbeitung – Vor- und Nachteile . . . . .	175
Echtzeitverarbeitung – Vor- und Nachteile . . . . .	175
Data Governance . . . . .	176
Zusammenfassung . . . . .	177

## Teil III: Datenarchitekturen

<b>10 Das Modern Data Warehouse</b>	<b>181</b>
Die MDW-Architektur . . . . .	181
Die MDW-Architektur – Vor- und Nachteile . . . . .	187
RDW und Data Lake kombinieren . . . . .	189
Data Lake . . . . .	189
Relationales Data Warehouse . . . . .	189
Schritt für Schritt zum MDW . . . . .	190
EDW-Erweiterung . . . . .	191

Temporärer Data Lake plus EDW .....	192
All-in-one .....	193
Fallstudie: Die strategische Umstellung bei Wilson & Gunkerk auf ein MDW .....	194
Herausforderung .....	195
Lösung .....	195
Ergebnis .....	195
Zusammenfassung .....	196
<b>11 Data Fabric</b>	<b>199</b>
Die Data-Fabric-Architektur .....	200
Datenzugriffsrichtlinien .....	201
Metadatenkatalog .....	202
Stammdatenverwaltung .....	203
Datenvirtualisierung .....	203
Echtzeitverarbeitung .....	203
APIs .....	204
Dienste .....	204
Produkte .....	204
Weshalb von einem MDW auf eine Data-Fabric-Architektur umsteigen? .....	204
Potenzielle Nachteile .....	205
Zusammenfassung .....	206
<b>12 Data Lakehouse</b>	<b>207</b>
Delta-Lake-Features .....	208
Performanceverbesserungen .....	211
Die Data-Lakehouse-Architektur .....	212
Was, wenn man das RDW überspringt? .....	214
Relationale Serving-Schicht .....	217
Zusammenfassung .....	217
<b>13 Data-Mesh-Grundlagen</b>	<b>219</b>
Eine dezentralisierte Architektur .....	220
Der Hype um Data Mesh .....	221

Dehghanis vier Prinzipien des Data Mesh .....	223
Prinzip #1: Domain Ownership .....	223
Prinzip #2: Data-as-a-Product .....	224
Prinzip #3: Self-Serve-Infrastructure-as-a-Plattform .....	226
Prinzip #4: Federated Computational Governance .....	228
Das »reine« Data Mesh .....	229
Datendomains .....	231
Logische Data-Mesh-Architektur .....	232
Verschiedene Topologien .....	234
Data Mesh vs. Data Fabric .....	236
Anwendungsfälle .....	237
Zusammenfassung .....	239
<b>14 Data Mesh einführen? – Mythen, Bedenken und die Zukunft</b>	<b>241</b>
Mythen .....	241
Mythos: Data Mesh ist eine Silberkugel, mit der sich alle Datenprobleme schnell lösen lassen .....	242
Mythos: Ein Data Mesh ersetzt Ihren Data Lake und Ihr Data Warehouse .....	242
Mythos: Data-Warehouse-Projekte scheitern alle – ein Data Mesh löst dieses Problem .....	242
Mythos: Ein Data Mesh aufbauen bedeutet, absolut alles zu dezentralisieren .....	243
Mythos: Mit Datenvirtualisierungen lässt sich ein Data Mesh erstellen .....	243
Bedenken .....	244
Philosophische und konzeptionelle Fragen .....	245
Daten in einer dezentralisierten Umgebung kombinieren .....	246
Andere Probleme der Dezentralisierung .....	247
Komplexität .....	249
Duplizierung .....	249
Machbarkeit .....	250
Mitarbeiterinnen und Mitarbeiter .....	253
Hürden auf Domänebene .....	254
Organisatorische Bewertung: Sollten Sie ein Data Mesh einführen? .....	256

Empfehlungen für die Implementierung eines erfolgreichen Data Mesh . . . . .	258
Die Zukunft von Data Mesh . . . . .	259
Blick über den Tellerrand: Datenarchitekturen und ihre Anwendungen . . . . .	260
Zusammenfassung . . . . .	262

## Teil IV: Menschen, Prozesse und Technologien

<b>15 Menschen und Prozesse</b>	<b>265</b>
Teamorganisation: Rollen und Verantwortlichkeiten . . . . .	266
Rollen für MDW, Data Fabric oder Data Lakehouse . . . . .	266
Rollen für Data Mesh . . . . .	268
Warum Projekte scheitern: Fallstricke und Prävention . . . . .	272
Fallstrick: Führungskräfte denken, dass BI »einfach« ist . . . . .	272
Fallstrick: Die falschen Technologien verwenden . . . . .	272
Fallstrick: Zu viele Geschäftsanforderungen sammeln . . . . .	273
Fallstrick: Zu wenige Geschäftsanforderungen sammeln . . . . .	273
Fallstrick: Berichte präsentieren, ohne ihren Inhalt zuvor zu validieren . . . . .	274
Fallstrick: Unerfahrene Berater beauftragen . . . . .	274
Fallstrick: Eine Beratungsfirma beauftragen, die die Entwicklung an Offshore-Arbeiter outsourct . . . . .	274
Fallstrick: Projektbesitz an Berater abgeben . . . . .	275
Fallstrick: Den notwendigen Wissenstransfer zurück in die Organisation vernachlässigen . . . . .	275
Fallstrick: Das Budget auf halbem Weg durch das Projekt kürzen . . . . .	275
Fallstrick: Von einem Enddatum aus rückwärts arbeiten . . . . .	276
Fallstrick: Das Data Warehouse so strukturieren, dass es die Quelldaten und nicht die Geschäftsbedürfnisse widerspiegelt . . . . .	276
Fallstrick: Endbenutzern eine Lösung mit langen Reaktionszeiten oder anderen Performanceproblemen präsentieren . . . . .	277

Fallstrick: Zu viel (oder zu wenig) Design Ihrer Datenarchitektur .....	277
Fallstrick: Mangelnde Kommunikation zwischen IT und Businessdomains .....	277
Tipps für den Erfolg .....	278
Knausern Sie nicht mit Ihren Investitionen .....	278
Benutzer und Benutzerinnen einbeziehen, ihnen Ergebnisse zeigen und sie begeistern .....	279
Mehrwert für neue Berichte und Dashboards .....	280
Die Endbenutzer bitten, einen Prototyp zu erstellen .....	280
Einen Projektchampion/Sponsor finden .....	281
Einen Projektplan erstellen, der auf 80% Effizienz abzielt .....	281
Zusammenfassung .....	282
<b>16 Technologien</b>	<b>285</b>
Eine Plattform auswählen .....	285
Open-Source-Lösungen .....	285
On-Premises-Lösungen .....	288
Cloud-Provider-Lösungen .....	290
Cloud-Service-Modelle .....	293
Große Cloud-Provider .....	295
Multi-Cloud-Lösungen .....	296
Software-Frameworks .....	299
Hadoop .....	300
Databricks .....	304
Snowflake .....	306
Zusammenfassung .....	307
<b>Index</b>	<b>309</b>

## Datenarchitekturen

Data Fabric, Data Lakehouse und Data Mesh sind als praktische Alternativen zum Modern Data Warehouse in den Fokus der Unternehmen gerückt. Diese neuen Architekturen haben solide Vorteile, aber ihre fachliche Einordnung ist auch von Missverständnissen und Übertreibungen geprägt. Dieses praxisorientierte Buch bietet eine gut verständliche Einführung in jeden dieser Architekturansätze und hilft damit Datenexpertinnen und -praktikern, die jeweiligen Vor- und Nachteile zu verstehen.

James Serra erläutert die Konzepte gängiger Datenarchitekturen und zeigt dabei auch, wie sich Data Warehouses weiterentwickeln mussten, um mit Data-Lake-Funktionen arbeiten zu können. Sie erfahren, was Sie mit Data Lakehouses erreichen können und wie Sie Hype und Realität bei Data Meshs unterscheiden. Nach der Lektüre dieses Buchs werden Sie in der Lage sein, die für Ihre Zwecke am besten geeignete Datenarchitektur zu bestimmen.

- Entwickeln Sie ein grundlegendes Verständnis für die verschiedenen Datenarchitekturen
- Informieren Sie sich über die Stärken und Schwächen der einzelnen Ansätze
- Verstehen Sie die Unterschiede zwischen Data Warehouses und Data Lakes
- Profitieren Sie von der langjährigen Erfahrung von James Serra und erfahren Sie, wie Theorie und Praxis der jeweiligen Datenarchitekturen voneinander abweichen
- Wählen Sie die beste Architektur für Ihren Anwendungsfall aus
- Lernen Sie, wie man eine Architektur-Design-Sitzung durchführt, das Team organisiert und was die Erfolgsfaktoren für ein Projekt sind

»Seine Fähigkeit, komplexe technische Konzepte in klare, leicht verständliche Erklärungen zu verwandeln, ist wirklich bemerkenswert.«

– Annie Xu  
Senior Data Customer Engineer,  
Google

»Legen Sie es auf Ihren Schreibtisch – Sie werden es oft zurate ziehen.«

– Sawyer Nyquist  
Autor, Consultant und Inhaber,  
The Data Shop

James Serra arbeitet als Architekt für Big-Data- und Data-Warehousing-Lösungen bei Microsoft. Er ist ein Vordenker für die Nutzung und Anwendung von Big Data und Advanced Analytics, einschließlich Datenarchitekturen wie Modern Data Warehouse, Data Lakehouse, Data Fabric und Data Mesh.



9 783960 092544

[www.dpunkt.de](http://www.dpunkt.de)

Euro 39,90 (D)  
ISBN 978-3-96009-254-4



Gedruckt in Deutschland  
Mineralölfreie Druckfarben  
Zertifiziertes Papier