HANSER


Leseprobe

Dani Schnider, Claus Jordan, Peter Welker, Joachim Wehner

Data Warehouse Blueprints

Business Intelligence in der Praxis

ISBN (Buch): 978-3-446-45075-2

ISBN (E-Book): 978-3-446-45111-7

Weitere Informationen oder Bestellungen unter

http://www.hanser-fachbuch.de/978-3-446-45075-2

sowie im Buchhandel.

# Inhalt

# Geleitwort

*Von Dr. Carsten Bange, Gründer und Geschäftsführer des Business Application Research Centers (BARC), Teil des europäischen Analystenhauses CXP Group.*

Noch ein Buch über Data Warehousing? Ist darüber in den vergangenen 25 Jahren nicht genug geschrieben worden? Ich gebe zu, ich war skeptisch als die Autoren mich baten, ein Vorwort zu verfassen. Insbesondere auch, da wir in unserer täglichen Praxis als Marktanalysten eine deutlich wachsende Kritik vieler Unternehmen an ihrem Data Warehouse wahrnehmen. Insbesondere die Anwender verlangen nach Änderungen, um ihren veränderten Anforderungen Rechnung zu tragen. Die letzte BARC-Anwenderbefragung zu diesem Thema[1] zeigt deutlich, was den Veränderungsbedarf treibt: 62 % der 323 befragten BI- und Data-Warehouse-Verantwortlichen sehen sich mit deutlich erhöhten Erwartungen in den Fachbereichen konfrontiert, 51 % verstehen dabei die schnellere Veränderung von Geschäftsprozessen als wesentlichen Treiber für Anpassungen an Datenmanagement-Konzepten und 45 % erfahren eine Unzufriedenheit mit der benötigten Zeit, um neue Anforderungen im Data Warehouse umzusetzen.

Vielen Unternehmen wird also immer klarer, dass sie die etablierten Data-Warehouse-Systeme so nicht mehr weiterbetreiben können, sondern hinsichtlich der Prozesse und der Organisation, IT-Architektur und eingesetzte Technologien und Werkzeuge komplett überdenken müssen.

Das vorliegende Buch liefert hierzu einen guten Beitrag und legt seinen Fokus dabei auf Methodik und Technologie. Es trägt eine große Menge von Erfahrungen zu „Best Practice"-Anleitungen zusammen, die helfen, das eigene Projekt auf eine solide Basis zu stellen und typische Fehler zu vermeiden. Es behandelt dabei auch neue Technologien, z. B. aus dem Hadoop- und NoSQL-Umfeld, die eine interessante Ergänzung der etablierten und ausgereiften Datenbank- und Datenintegrationstechnologien sein können. Die Autoren bieten damit Entwicklern, BI- und Data-Warehouse-Verantwortlichen ein solides methodisches Fundament, um die Informationsversorgung zur Entscheidungsfindung in Unternehmen erfolgreich aufzubauen.

Es bleibt dann im Unternehmen die wichtige Aufgabe, die verfügbaren Technologien in eine anforderungsgerechte Organisation einzubetten. Gerade Agilität und Flexibilität sind hier

[1] s. BARC-Anwenderbefragung „Modernes Datenmanagement für die Analytik" (BARC 2015), Ergebnisstudie kostenfrei verfügbar unter *www.barc.de* im Bereich Research.

die wesentlichen Anforderungen, die in den letzten Jahren beispielsweise den Trend zu „Self Service BI“ angefeuert haben, also der Bereitstellung weitgehender Möglichkeiten zur Zusammenstellung, Aufbereitung und Visualisierung von Daten für Fachanwender. Da dies häufig auch mit einer „Self Service-Datenintegration“ verbunden ist, ergibt sich schnell die Kehrseite der Medaille solcher Initiativen: Die Konsistenz von Daten kann in einer dezentralisierten Welt individueller Datenaufbereitung – wenn überhaupt – nur mit erheblichen Anstrengungen einer Data Governance sichergestellt werden. Die ersten Unternehmen kehren demnach auch schon wieder zu stärker zentralistisch ausgerichteten Konzepten zurück, um dem Daten-Wildwuchs Einhalt zu gebieten.

Dieser Spagat zwischen der Bereitstellung qualitätsgesicherter Daten unter übergreifender Kontrolle auf der einen sowie Flexibilität und Individualität in Datenzusammenstellung und -auswertung auf der anderen Seite ist aus unserer Sicht die momentan größte Herausforderung für Betreiber entscheidungsunterstützender Informationssysteme.

Das Buch zeigt, dass viele Methoden und Technologien hierfür zur Verfügung stehen. Werden sie richtig eingesetzt, sind dem Data Warehouse auch weitere 25 Jahre erfolgreichen Einsatzes beschieden, denn Entscheidungsträger im Unternehmen werden auch in Zukunft nicht auf konsistente und qualitätsgesicherte Daten zur Entscheidungsfindung verzichten.

Würzburg, den 14.7.2016

*Dr. Carsten Bange*

# Über dieses Buch

Das vorliegende Buch ist eine Weiterentwicklung des Buches „Data Warehousing mit Oracle – Business Intelligence in der Praxis“, das 2011 beim Carl Hanser Verlag erschienen und mittlerweile vergriffen ist. Im Vergleich zur vorherigen Version wurden hier die allgemeinen Konzepte, Architekturvorschläge und Vorgehensweisen stark ausgebaut und aktualisiert. Oracle-spezifische Informationen wurden – bis auf die Verwendung in Beispielen – weitgehend verallgemeinert, sodass die vorliegenden Blueprints auch für andere Datenbanktechnologien eingesetzt werden können.

Die Data Warehouse Blueprints wurden vorerst als interner Leitfaden für die BI-Consultants bei Trivadis zur Verfügung gestellt, bevor sie öffentlich publiziert wurden. Während dieser Zeit haben verschiedene Trivadis-Kollegen die einzelnen Kapitel überprüft und zahlreiche Korrekturen, Änderungsvorschläge und Ergänzungen zur nun vorliegenden Ausgabe beigetragen.

# Die Autoren

### Dani Schnider

Dani Schnider ist seit seinem abgeschlossenen Informatikstudium an der ETH Zürich (1990) in der Informatik tätig. Seit 1997 arbeitet er vorwiegend in DWH-Projekten. Konzeption, Design, Aufbau und Weiterentwicklung von Data Warehouses, logische und physische Datenmodellierung im DWH-Umfeld sowie Reviews, Architekturberatungen und Schulungen bilden seine Aufgabenschwerpunkte in diesem Bereich. Präsentationen an verschiedenen Konferenzen und Publikationen von Fachartikeln und Blog-Posts runden seine Tätigkeiten ab. (Kontakt: dani.schnider@trivadis.com)

### Claus Jordan

Seit seinem Abschluss des Studiums der Wirtschaftsinformatik 1993 ist Claus Jordan im Umfeld Data Warehouse und Business Intelligence aktiv. Seit 2003 bringt er seine Erfahrung in diesen Bereichen für die Trivadis GmbH in zahlreichen Kundenprojekten, als Trainer und als Autor ein. Seine Schwerpunkte liegen dabei im Design unterschiedlicher Datenmodellierungsmethoden eines Data Warehouse, sowie in der Implementierung von ETL-Prozessen und deren Standardisierung. (Kontakt: claus.jordan@trivadis.com)

### Peter Welker

Peter Welker arbeitete bereits vor dem Abschluss seines Studiums der Medizininformatik 1996 als Entwickler für Anwendungssoftware. 1998 wechselte er ins Data Warehousing und ist seitdem hauptsächlich in Projekten mit dem Fokus ETL, DWH-Lösungsarchitektur, Review und Performance aktiv. In den letzten Jahren beschäftigt er sich intensiv mit den neuen Technologien. Er präsentiert an Konferenzen, publiziert Fachartikel und verantwortet bei der Deutschen Oracle-Anwendergruppe (DOAG) das Thema „Big Data“. (Kontakt: peter.welker@trivadis.com)

### Joachim Wehner

Seit seiner Diplomarbeit „Werkzeuge zum Aufbau eines Data Warehouses“ aus dem Jahre 1996 lässt ihn dieses Thema nicht mehr los. Als Berater und Trainer arbeitet Joachim Wehner über die Jahre primär in BI-/DWH-Kundenprojekten. Im Mittelpunkt stehen dabei fast immer die Architektur, das Design sowie Reviews solcher Data-Warehouse-Umgebungen. Inzwischen hat sich sein Verantwortungsbereich von der Technik auf die Managementseite verlagert. (Kontakt: joachim.wehner@trivadis.com)

# Danksagung

Die Kapitel dieses Buches wurden von verschiedenen Trivadis-Consultants geprüft, korrigiert und mit wertvollen Ergänzungen und Änderungsvorschlägen angereichert. Der Dank für diese Reviewarbeit gilt folgenden Personen: Adrian Abegglen, Aron Hennerdal, Beat Flühmann, Christoph Hisserich, Kamilla Reichardt, Maurice Müller, Peter Denk, Stanislav Lando, Thomas Brunner und Willfried Färber. Die gute und konstruktive Zusammenarbeit mit Frau Hasselbach und Frau Weilhart vom Hanser Verlag ist an dieser Stelle ebenfalls dankend zu erwähnen wie das passende Geleitwort zu diesem Buch von Herrn Dr. Carsten Bange vom Business Application Research Center (BARC).

Ein besonderer Dank gilt natürlich der Trivadis AG für die Unterstützung des Buchprojekts während der Erstellung und dafür, dass sie es ermöglicht hat, dass das Buch wiederum öffentlich publiziert werden kann.

# 2 Architektur

Eine gut strukturierte Architektur ist eine wichtige Voraussetzung für den erfolgreichen Einsatz von Data Warehousing und Business Intelligence. Die wichtigsten Grundsätze zur Architektur von DWH-Systemen und BI-Anwendungen sowie verschiedene Möglichkeiten zur Datenhaltung in einem Data Warehouse sind in diesem Kapitel zusammengefasst.

- Abschnitt 2.1 beschreibt die grundlegende Architektur eines Data Warehouse und stellt die verschiedenen Schichten einer DWH-Architektur vor.
- In Abschnitt 2.2 werden die wichtigsten Grundsätze zur Architektur und zum Aufbau von BI-Anwendungen erläutert.
- Abschnitt 2.3 gibt einen Überblick über verschiedene Konzepte zur Datenhaltung, wie sie für Data Warehouses zum Einsatz kommen.

## 2.1 Data Warehouse-Architektur

Ein Data Warehouse (DWH) stellt die technische Infrastruktur zur Verfügung, die benötigt wird, um Business Intelligence betreiben zu können. Sein Zweck ist es, Daten aus unterschiedlichen Datenquellen zu integrieren und eine historisierte Datenbasis zur Verfügung zu stellen, welche für Standard- und Ad-hoc-Reporting, OLAP[1]-Analysen, Balanced Scorecards, BI-Dashboards und weitere BI-Anwendungen eingesetzt werden kann. Ein DWH ist ein abfrageoptimiertes System, mit welchem auf eine Sammlung von historisierten Daten über einen längeren Zeitpunkt zugegriffen werden kann.

Durch diese Ausgangslage ergeben sich einige Unterschiede zwischen einem operativen System (auch OLTP[2]-System genannt) und einem Data Warehouse. Während in einem OLTP-System mehrere bis viele Anwender gleichzeitig Daten einfügen, ändern und löschen, ist dies bei einem DWH-System in der Regel nicht der Fall. Die einzigen „Anwender“, die in ein Data Warehouse schreiben, sind die ETL-Prozesse, welche Daten von den Quellsystemen ins DWH laden. Auch die Art der Abfragen ist unterschiedlich. In operativen Systemen werden typi-

[1] OLAP = Online Analytical Processing
[2] OLTP = Online Transaction Processing

scherweise spezifische Informationen in einem großen Datenbestand gesucht, beispielsweise die letzten zehn Banktransaktionen eines bestimmten Kunden. In einem Data Warehouse hingegen werden meistens Auswertungen über große Datenmengen ausgeführt und aggregiert, zum Beispiel die Summe über alle Verkäufe an alle Kunden einer bestimmten Region.

Um diesen unterschiedlichen Bedürfnissen gerecht zu werden, werden DWH-Datenbanken anders aufgebaut als OLTP-Datenbanken. Architektur, Design und Datenmodellierung funktionieren im DWH-Umfeld nicht auf die gleiche Weise, wie es viele erfahrene Architekten, Datenmodellierer und Entwickler gewohnt sind, die hauptsächlich im Bereich von OLTP-Datenbanken tätig sind. Auf die spezifischen Bedürfnisse von DWH-Systemen wird deshalb nachfolgend eingegangen.

Die Komplexität und Erweiterbarkeit eines Data Warehouse ist weitgehend abhängig von der verwendeten Architektur. Deshalb ist es in jedem DWH-Projekt von Anfang an wichtig, dass eine saubere Architektur definiert und implementiert wird. In der Regel bedeutet das, dass die Architektur aus unterschiedlichen Schichten besteht. Diese Schichten decken jeweils unterschiedliche Anforderungen ab. Auch wenn dies zum Beginn des Projektes nach Mehraufwand aussieht, zahlt sich eine konsequente Aufteilung in verschiedene DWH-Schichten im späteren Projektverlauf und im operativen Betrieb des Systems aus.

Leider wird oft der Fehler gemacht, dass aufgrund von knappen Terminvorgaben wesentliche Architekturgrundsätze missachtet und „Abkürzungen“ bzw. „Schnellschüsse“ implementiert werden. Diese Ad-hoc-Lösungen können früher oder später zu Problemen führen. Der Aufwand, diese wiederum zu beheben, ist oft größer als der Aufwand, von Anfang an eine saubere Lösung zu realisieren.

### 2.1.1 Aufbau eines Data Warehouse

Ein Data Warehouse besteht typischerweise aus verschiedenen Schichten (auch Layers, Bereiche oder Komponenten genannt) und Datenflüssen zwischen diesen Schichten. Auch wenn nicht jedes DWH-System alle Schichten umfassen muss, lässt sich jedes Data Warehouse auf eine Grundarchitektur, wie sie in Bild 2.1 dargestellt ist, zurückführen.

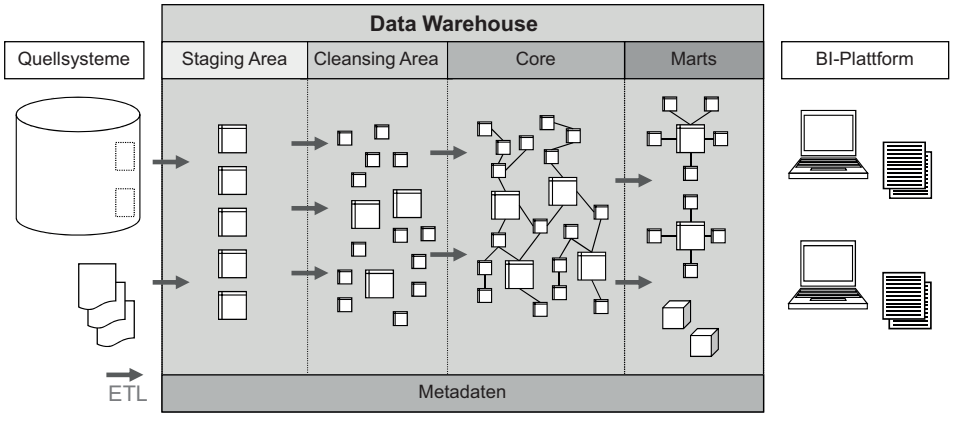


**Bild 2.1** Grundarchitektur eines Data Warehouse

*Um den Zweck der einzelnen Schichten in einer DWH-Architektur zu erklären, werden nachfolgend Beispiele aus dem „realen Leben" gezeigt. Nehmen wir an, das DWH sei ein großes Lebensmittelgeschäft. Auch dort gibt es verschiedene Bereiche, die jeweils einem bestimmten Zweck dienen.*

Folgende Schichten oder Bereiche gehören zu einer vollständigen DWH-Architektur:

- *Staging Area:* Daten aus unterschiedlichen Quellsystemen werden zuerst in die Staging Area geladen. In diesem ersten Bereich des DWH werden die Daten so gespeichert, wie sie angeliefert werden. Die Struktur der Stage-Tabellen entspricht deshalb der Schnittstelle zum Quellsystem[3]. Beziehungen zwischen den einzelnen Tabellen bestehen keine. Jede Tabelle enthält die Daten der letzten Lieferung, welche vor der nächsten Lieferung gelöscht werden.

  *In einem Lebensmittelgeschäft entspricht die Staging Area der Laderampe, an der die Lieferanten (Quellsysteme) ihre Waren (Daten) abliefern. Auch dort werden immer nur die neuesten Lieferungen zwischengelagert, bevor sie in den nächsten Bereich überführt werden.*

- *Cleansing*[4] *Area:* Bevor die gelieferten Daten ins Core geladen werden, müssen sie bereinigt werden. Fehlerhafte Daten müssen entweder ausgefiltert, korrigiert oder durch Singletons (Defaultwerte) ergänzt werden. Daten aus unterschiedlichen Quellsystemen müssen in eine vereinheitlichte Form transformiert und integriert werden. Die meisten dieser Bereinigungsschritte werden in der Cleansing Area durchgeführt. Auch diese Schicht enthält nur die Daten der letzten Lieferung.

  *Im Lebensmittelgeschäft kann die Cleansing Area mit dem Bereich verglichen werden, in dem die Waren für den Verkauf kommissioniert werden. Die Waren werden ausgepackt, Gemüse und Salat werden gewaschen, das Fleisch portioniert, ggf. mehrere Produkte zusammengefasst und alles mit Preisetiketten versehen. Die Qualitätskontrolle der angelieferten Ware gehört ebenfalls in diesen Bereich.*

- *Core*: Die Daten aus den verschiedenen Quellsystemen werden über die Staging und Cleansing Area in einem zentralen Bereich, dem Core, zusammengeführt und dort über einen längeren Zeitraum, oft mehrere Jahre, gespeichert. Eine Hauptaufgabe des Core ist es, die Daten aus den unterschiedlichen Quellen zu integrieren und nicht mehr getrennt nach Herkunft, sondern themenspezifisch strukturiert zu speichern. Oft spricht man bei thematischen Teilbereichen im Core von „Subject Areas". Die Daten werden im Core so abgelegt, dass historische Daten zu jedem späteren Zeitpunkt ermittelt werden können. Das Core sollte die einzige Datenquelle für die Data Marts sein. Direkte Zugriffe von Benutzern auf das Core sollten möglichst vermieden werden.

  *Das Core kann mit einem Hochregallager verglichen werden. Waren werden so abgelegt, dass sie jederzeit auffindbar sind, aber der Zugriff darauf ist nur internen Mitarbeitern möglich. Kunden haben im Lager nichts zu suchen – außer vielleicht bei IKEA. Im Gegensatz zu einem Hochregallager bleiben die Daten aber auch dann im Core erhalten, nachdem sie an die Data Marts übertragen wurden.*

---

[3] Oft werden den Stage-Tabellen zusätzliche Attribute für Auditinformationen zugefügt, die im Quellsystem nicht vorhanden sind.

[4] Der Begriff „Cleansing" wird englisch „klensing" ausgesprochen und nicht „kliinsing".

- *Marts:* In den Data Marts werden Teilmengen der Daten aus dem Core so aufbereitet abgespeichert, *dass* sie in einer für die Benutzerabfragen geeigneten Form zur Verfügung stehen. Jeder Data Mart sollte nur die für die jeweilige Anwendung relevanten Daten bzw. eine spezielle Sicht auf die Daten enthalten. Das bedeutet, dass typischerweise mehrere Data Marts für unterschiedliche Benutzergruppen und BI-Anwendungen definiert werden. Dadurch kann die Komplexität der Abfragen reduziert werden. Das erhöht die Akzeptanz des DWH-Systems bei den Benutzern.

  *Die Data Marts sind die Marktstände oder Verkaufsgestelle im Lebensmittelgeschäft. Jeder Marktstand bietet eine bestimmte Auswahl von Waren an, z. B. Gemüse, Fleisch oder Käse. Die Waren werden so präsentiert, dass sie von der jeweiligen Kundengruppe akzeptiert, also gekauft werden.*

- *ETL-Prozesse:* Die Daten, die von den Quellsystemen als Files, Schnittstellentabellen oder über einen View Layer zur Verfügung gestellt werden, werden in die Staging Area geladen, in der Cleansing Area bereinigt und dann im Core integriert und historisiert. Vom Core werden aufgrund von fachlichen Anforderungen Teilmengen oder oft auch nur Aggregate in die verschiedenen Data Marts geladen.

  All diese Datenflüsse werden unter dem Begriff ETL (Extraction, Transformation, Loading) zusammengefasst. Die Extraktion der Daten aus den Quellsystemen findet in der Regel außerhalb des DWH-Systems statt, nämlich in den Quellsystemen selbst. Als Transformationen werden alle Datenumformungen, Bereinigungen, Anreicherungen mit Zusatzinformationen und Aggregationen bezeichnet. Schließlich werden die Daten in die Zieltabellen der nächsten Schicht geladen.

  *Die ETL-Prozesse sind die Mitarbeiter des Lebensmittelgeschäfts, die unterschiedliche Arbeiten verrichten müssen, damit die Lebensmittel vom Lieferanten bis hin zum Kunden gelangen.*

- *Metadaten*: Für den reibungsfreien Betrieb des Data Warehouse werden unterschiedliche Arten von Metadaten benötigt. Fachliche Metadaten enthalten fachliche Beschreibungen aller Attribute, Drill-Pfade und Aggregationsregeln für die Frontend-Applikationen und Codebezeichnungen. Technische Metadaten beschreiben z. B. Datenstrukturen, Mapping-Regeln und Parameter zur ETL-Steuerung. Operative Metadaten beinhalten alle Log-Tabellen, Fehlermeldungen, Protokollierungen der ETL-Prozesse und vieles mehr. Die Metadaten bilden die Infrastruktur eines DWH-Systems und werden als „Daten über Daten“ beschrieben.

  *Auch in unserem Lebensmittelgeschäft braucht es eine funktionierende Infrastruktur – von Wegweisern zur Kasse bis hin zur Klimaüberwachung der Frischwaren.*

Nicht jedes Data Warehouse hat genau diesen Aufbau. Teilweise werden einzelne Bereiche zusammengefasst – zum Beispiel Staging Area und Cleansing Area – oder anders bezeichnet. So wird zum Teil das Core als „Integration Layer“ oder als „(Core) Data Warehouse“ bezeichnet. Wichtig ist jedoch, dass das Gesamtsystem in verschiedene Bereiche unterteilt wird, um die unterschiedlichen Aufgabenbereiche wie Datenbereinigung, Integration, Historisierung und Benutzerabfragen zu entkoppeln. Auf diese Weise kann die Komplexität der Transformationsschritte zwischen den einzelnen Schichten reduziert werden.

### 2.1.2 Transformationsschritte

Mithilfe der ETL-Prozesse werden die Daten von den Quellsystemen ins Data Warehouse geladen. In jeder DWH-Schicht werden dabei unterschiedliche Transformationsschritte durchgeführt, wie in Bild 2.2 dargestellt.

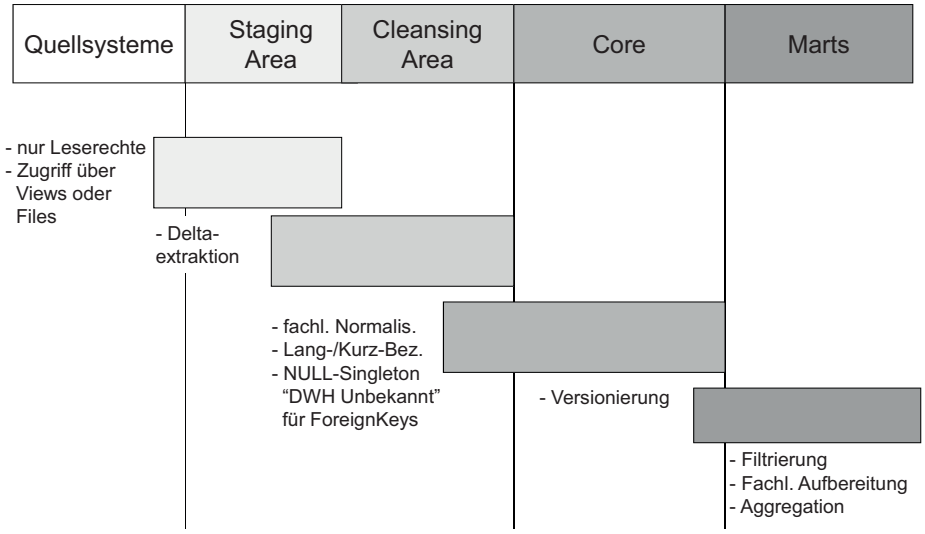


**Bild 2.2** Transformationsschritte im Data Warehouse

- *Quellsystem → Staging Area*
  Wird direkt auf ein relationales Quellsystem zugegriffen, sollte als Schnittstelle zwischen Quellsystem und DWH ein View Layer als definierte Zugriffsschicht implementiert werden. Der Zugriff der ETL-Prozesse auf das Quellsystem erfolgt dann ausschließlich über diese Views. Auf diese Weise kann eine gewisse Unabhängigkeit der ETL-Prozesse gegenüber Strukturänderungen auf dem Quellsystem erreicht werden. Die Views können außerdem für die Delta-Extraktion verwendet werden, indem sie so implementiert werden, dass nur die jeweils relevante Teilmenge der Daten in den Views zur Verfügung steht. Dieses Verfahren wird teilweise für Change Data Capture (CDC) verwendet.

  Als Alternative zum direkten Zugriff auf das Quellsystem werden häufig Dateien als Schnittstelle zwischen Quellsystem und Data Warehouse verwendet. Die Extraktion der Daten in die Dateien erfolgt auf dem Quellsystem und wird meistens außerhalb des DWH-Projektes realisiert.

  Die Daten, ob über Views oder Files geliefert, werden unverändert in die Staging Area geladen und ggf. mit Auditinformationen angereichert.
- *Staging Area → Cleansing Area*
  Beim Laden in die Cleansing Area werden die Daten geprüft, bereinigt und mit zusätzlichen Attributen angereichert. Dazu gehört zum Beispiel die Ermittlung von Lang- und Kurztexten aus den fachlichen Attributen der Quellsysteme. Fehlende oder fehlerhafte Attribute und Foreign Keys werden durch Singletons ersetzt.

Fehlerhafte Datensätze können je nach Anforderungen ignoriert, aufgrund von fixen Regeln korrigiert, durch Singletons ersetzt oder in Fehlertabellen geschrieben werden. Fehlertabellen können als Basis für Fehlerprotokolle oder manuelle Korrekturen verwendet werden. Bei solchen aufwendigen Varianten der Fehlerbehandlung muss allerdings organisatorisch geklärt werden, wer für die Fehlerkorrekturen verantwortlich ist.

- *Cleansing Area → Core*
  Nachdem die Daten in der Cleansing Area in die benötigte Form aufbereitet wurden, werden sie ins Core geladen. In diesem Schritt findet die Versionierung der Stammdaten[5] statt, d. h., es wird für jeden Datensatz geprüft, ob sich etwas geändert hat und somit eine neue Version erstellt werden muss. Je nach Historisierungsanforderungen und Core-Datenmodell gibt es verschiedene Varianten der Versionierung von Stammdaten.

  Die Bewegungsdaten[6] werden historisiert. Weil sich Bewegungsdaten nachträglich nicht mehr ändern, heißt das, dass laufend neue Daten eingefügt und über einen längeren Zeitraum gespeichert werden. Oft besteht die Anforderung, dass Bewegungsdaten nach einer gewissen Zeit – in der Regel nach mehreren Jahren – aus dem Core gelöscht werden.

  Aggregationen werden im Core nicht durchgeführt. Die Bewegungsdaten werden auf der Detaillierungsstufe, die geliefert wird, gespeichert.
- *Core → Marts*
  Die Transformationen vom Core in die Data Marts bestehen aus der Filtrierung der Daten auf die für jeden Data Mart erforderliche Teilmenge, der fachlichen Aufbereitung der Dimensionen[7] in die gewünschten Hierarchiestufen sowie – falls erforderlich – der Aggregation der Bewegungsdaten auf die Granularität der Faktentabellen.

### 2.1.3 Architekturgrundsätze

Obwohl sich die Architektur vieler DWH-Systeme in Details unterscheidet und oft auch unterschiedliche Namen für die einzelnen Bereiche verwendet werden, gibt es ein paar wichtige Architekturgrundsätze, die auf jeden Fall berücksichtigt werden sollten. Vereinfachungen der Architektur sind erlaubt, aber die wichtigsten Schichten sollten auf keinen Fall weggelassen werden.

---

[5] Stammdaten (oder Referenzdaten) sind zustandsorientierte Daten, die sich im Laufe der Zeit ändern können. Um die Änderungen im Core nachvollziehbar abspeichern zu können, werden die Daten versioniert. Das heißt, dass für jede Datenänderung ein neuer Datensatz im Core eingefügt wird. Die Versionierung von Stammdaten wird in Kapitel 3 (Datenmodellierung) genau erklärt.

[6] Bewegungsdaten (oder Transaktionsdaten) sind ereignisorientierte Daten, die aufgrund eines bestimmten Ereignisses (z. B. Transaktion, Messung) entstehen und nachträglich nicht mehr geändert werden. Sie sind immer mit einem Ereigniszeitpunkt (z. B. Transaktionsdatum) verbunden. Die Historisierung von Bewegungsdaten wird in Kapitel 3 (Datenmodellierung) beschrieben.

[7] Die Begriffe Dimensionen, Hierarchien und Fakten sind Elemente der dimensionalen Modellierung und werden in Kapitel 3 (Datenmodellierung) erläutert.

**Jedes Data Warehouse besitzt ein Core**

Data Marts werden nie direkt aus den Quellsystemen geladen. Einzige Datenquelle für die Data Marts ist das Core, welches als „Single Source of Truth“ (Inmon 2005) verwendet wird. Gemeint ist damit, dass sämtliche Daten in allen Data Marts aus dem Core geladen werden, um Inkonsistenzen zu vermeiden.

Das Core dient als zentrale Integrationsplattform innerhalb des Data Warehouse. Daten aus verschiedenen Quellsystemen müssen in eine vergleichbare Form transformiert werden. Dieser Aufwand soll für jedes Quellsystem nur einmal gemacht werden und nicht für jeden Data Mart separat.

Bei mehreren Quellsystemen und mehreren Data Marts wird schnell ersichtlich, dass es keine Lösung sein kann, für jede gewünschte Kombination separate ETL-Prozesse zu definieren. Die Komplexität der ETL-Prozesse wird rasch sehr hoch (siehe Bild 2.3 links). Durch die mehrfache Implementation kann nicht mehr garantiert werden, dass die Business Rules in allen Data Marts gleich umgesetzt werden. Nur durch ein zentrales Core kann sichergestellt werden, dass die Daten in den unterschiedlichen Data Marts konsistent und somit vergleichbar sind (siehe Bild 2.3 rechts).

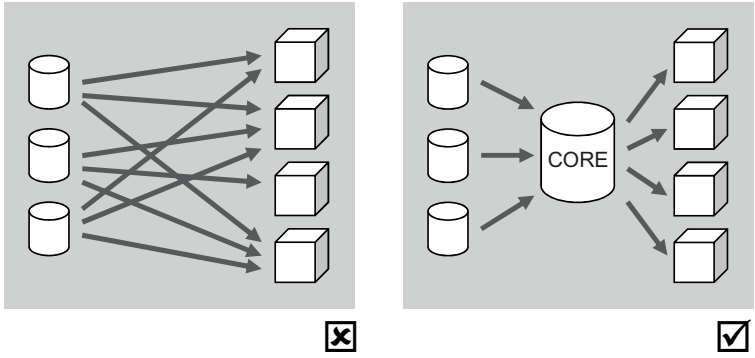


**Bild 2.3** ETL-Prozesse mit und ohne Core

Eine Ausnahme ist es, wenn nur ein Quellsystem und ein Data Mart vorhanden sind. In diesem Fall kann auf ein Core verzichtet werden. Trotzdem ist es zu empfehlen, diese Schicht zumindest architektonisch vorzusehen, denn typischerweise werden an ein bestehendes Data Warehouse immer neue Anforderungen gestellt, sodass zu einem späteren Zeitpunkt weitere Data Marts dazukommen. Ein pragmatischer Weg ist es deshalb, zu Beginn ein Core zu realisieren und den ersten Data Mart als „virtuellen Data Mart“ (d. h. View Layer) zu implementieren.

Ein weiterer Grund für das Core ist die Historisierung der Daten. Im Core werden alle Bewegungsdaten über eine längere Zeitperiode – oft mehrere Jahre – gespeichert, und die Stammdaten werden versioniert. Dies erlaubt, bei Bedarf einen Data Mart neu laden zu können. Existiert kein Core, so können die historischen Daten nicht mehr rekonstruiert werden, da sie in der Regel in den Quellsystemen nicht mehr vorhanden sind.

**Benutzer greifen nie direkt aufs Core zu**

Für Abfragen werden Data Marts zur Verfügung gestellt. Das Core hat einen anderen Zweck und ist nicht für direkte Benutzerabfragen optimiert. Falls für bestimmte Anwendungen direkt auf Informationen im Core zugegriffen werden muss, kann dies durch entsprechende Views auf die Core-Tabellen implementiert werden. Aus Architektursicht ist ein solcher View Layer ein nicht-persistenter Data Mart.

Die Datenstrukturen im Core sind darauf ausgelegt, Rohdaten in historisierter Form zu archivieren. Abfragen auf die Core-Tabellen können deshalb – je nach Datenvolumen, Datenmodell und verwendeter Historisierungslogik – sehr komplex werden und somit zu langen Antwortzeiten führen. Das Core ist nicht für Abfragen optimiert.

Für die Benutzer (und für viele BI-Tools) ist es wichtig, dass die Abfragen möglichst einfach und immer nach einem ähnlichen Muster erfolgen können. Deshalb werden die Daten in geeigneter Form in Data Marts zur Verfügung gestellt.

Wenn die Anforderung besteht, dass bestimmte Benutzergruppen direkt auf das Core zugreifen dürfen, sollte dies so gelöst werden, dass ein View Layer („virtueller Data Mart“) zur Verfügung gestellt wird. Die Views enthalten dann beispielsweise die Historisierungslogik, welche die jeweils gültige Version zu einem bestimmten Zeitpunkt zurückliefert. Durch den View Layer können die Abfragen vereinfacht werden, und es besteht eine gewisse Unabhängigkeit der Abfragen von der physischen Speicherung der Daten im Core.

**Pro Anwendungsbereich wird ein Data Mart erstellt**

Ein universeller Data Mart, der alle Bedürfnisse abdeckt, führt zu hoher Komplexität und langen Antwortzeiten. Dies entspricht nicht den Wünschen der Anwender.

Je nach Anwendungsbereich, Benutzergruppe und verwendetem Frontend-Tool bestehen unterschiedliche Anforderungen an die Datenstrukturen in den Data Marts. Granularität der Daten, historisches Zeitfenster, Häufigkeit der Aktualisierung, aber auch die geforderten Antwortzeiten können sehr unterschiedlich sein. In Bild 2.4 wird dies anhand von zwei Data Marts – einen für OLAP-Analysen und einen für ein Decision Support System (DSS) – illustriert.

Für Reporting-Anwendungen werden typischerweise Detaildaten mit feiner Granularität benötigt, während mit einem OLAP-Tool teilweise nur auf aggregierte zugegriffen wird, wenn die Detaildaten für die entsprechenden Benutzeranforderungen zu umfangreich sind. Für Standardreports, die einmal täglich berechnet und als PDF-Dateien den Benutzern zur Verfügung gestellt werden, gelten andere Performanceanforderungen als für Online-Analysen, die auf einem Data Mart ausgeführt werden.

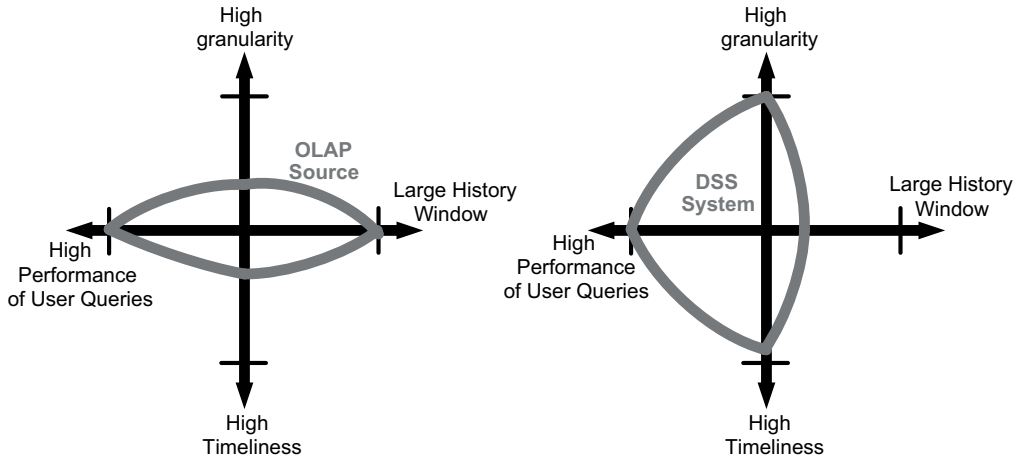


**Bild 2.4** Unterschiedliche Anforderungen an Data Marts

Außerdem sollte jeder Data Mart auf die wesentlichen Daten reduziert werden. Nicht jede Benutzergruppe oder Abteilung benötigt sämtliche Dimensionen und Hierarchiestufen für ihre Abfragen. Ein Data Mart mit Dutzenden von Dimensionen ist für die Endbenutzer oft verwirrend. Es empfiehlt sich deshalb, spezifische Data Marts mit den relevanten Dimensionen und Verdichtungsstufen zur Verfügung zu stellen. Diese können viel besser auf die jeweiligen Bedürfnisse der Benutzer optimiert werden.

## 2.2 Architektur BI-Anwendungen

Unternehmen nutzen Business-Intelligence-Systeme in vielfältigster Weise. BI-Anwendungen oder BI-Tools sind deshalb sehr speziell auf die Anforderungen der Mitarbeiter in den Fachabteilungen zugeschnitten. In größeren Unternehmen sind häufig mehrere voneinander unabhängige BI-Anwendungen (von unterschiedlichen Anbietern oder individuell programmiert) im Einsatz. Auch Tabellenkalkulationsprogramme sind aufgrund ihrer Flexibilität bei den BI-Anwendern sehr verbreitet. Die meisten BI-Anwendungen bieten neben den spezifischen Funktionalitäten solche für Präsentation und Ad-hoc-Analyse. Die Präsentation anwendungsspezifischer Informationen erfolgt in Form von individuellen Berichten und Listen oder modernen, interaktiven Management Cockpits bzw. Dashboards. Die Ad-hoc-Analysen dagegen dienen der individuellen Analyse des Datenmaterials.

**Benutzerberechtigungen in BI-Anwendungen**

Ebenfalls zum Funktionsumfang von professionellen BI-Anwendungen gehört die Unterstützung von individuellen Benutzerberechtigungen. Pro Benutzer oder Benutzergruppe kann definiert werden, welche Berichte und Funktionen ausgeführt (Funktionsberechtigungen) und welche Daten angezeigt werden dürfen (Datenberechtigungen). So dürfen beispielsweise zwei Benutzer den gleichen Bericht ausführen, sehen aber nur jeweils die Umsatzzahlen ihrer eigenen Niederlassung.

Die Zugriffsberechtigungen auf die Daten sollten aber nicht nur in den BI-Anwendungen konfiguriert werden, sondern auch auf der Datenbank. Sonst könnte ein Benutzer über ein anderes Tool oder direkt mittels SQL-Abfragen auf nicht berechtigte Daten zugreifen.

Neben diesen eben genannten generellen Funktionalitäten ist allen BI-Anwendungen gemein, dass sie auf *Daten* zugreifen. Die bevorzugte Datenquelle für BI-Anwendungen ist ein solides Data Warehouse, weil dort die Daten aus verschiedenen operativen Systemen integriert und historisiert vorliegen (siehe Abschnitt 2.1). Außerdem gewährleistet das dimensionale Datenmodell, das im nächsten Kapitel erläutert wird, in der DWH-Schicht *Marts* eine sehr gute Abfrageperformance, die wiederum für die Akzeptanz von BI-Anwendungen, und damit für deren erfolgreichen Einsatz, entscheidend ist.

Eine weitere Implementierungsmöglichkeit für dimensionale Datenmodelle sind spezielle OLAP-Datenbanken[8], die eine noch bessere Abfrageperformance und zusätzliche analytische Möglichkeiten bieten. Aus diesen Gründen sind OLAP-Datenbanken ebenfalls geeignete Datenquellen für BI-Anwendungen. Der direkte Zugriff auf operative Systeme (z. B. Finanzbuchhaltung) und auf externe Daten (z. B. Marktdaten) sollte von den BI-Anwendungen in Ausnahmefällen möglich sein, ist jedoch nicht zu empfehlen.

Die zugrunde liegenden Daten können und sollen prinzipiell unabhängig von der speziellen BI-Anwendung genutzt werden. Um auf diese verschiedenen Datenquellen zugreifen zu können, müssen in heterogenen Systemumgebungen die BI-Anwendungen den Zugriff auf unterschiedliche Technologien von unterschiedlichen Anbietern erlauben. Das Bild 2.5 zeigt den Zugriff von zwei BI-Anwendungen unterschiedlicher Anbieter auf verschiedene Datenquellen. In diesem Beispiel kommen die Funktionalitäten *Standardberichtswesen* und *Ad-hoc-Analysen* in beiden BI-Anwendungen zum Einsatz. Solche allgemeinen, anwendungsübergreifenden Funktionalitäten sind typischerweise in den meisten BI-Anwendungen enthalten. Kommen für unterschiedliche BI-Anwendungen verschiedene Anbieter zum Einsatz, so unterscheiden sich meist die verwendete Technologie, deren Administration und Handhabung grundlegend voneinander.

[8] OLAP-Datenbanken werden auch multidimensionale Datenbanken genannt, welche deshalb auch als MOLAP-Datenbanken bezeichnet werden. Die Speicherung der Daten in MOLAP-Datenbanken erfolgt dabei nicht in Tabellen, wie in relationalen Datenbanken, sondern in n-dimensionalen Feldern (Arrays), die häufig als Cubes bezeichnet werden.

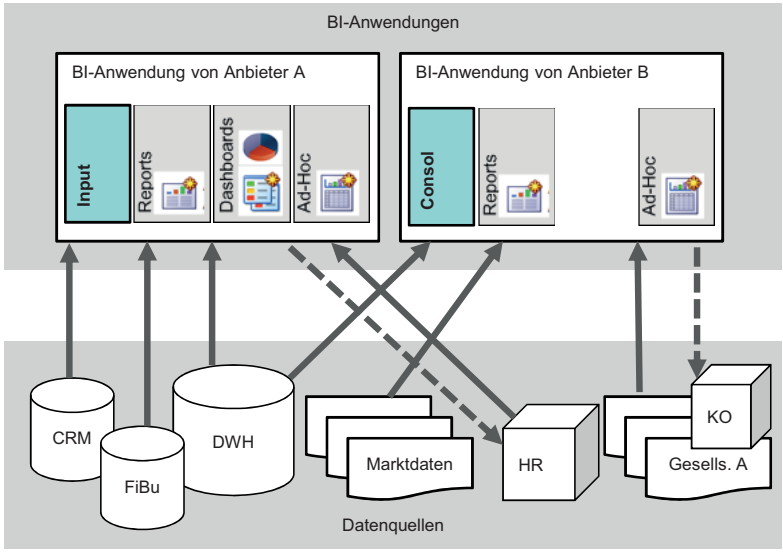


**Bild 2.5** BI-Anwendungen/BI-Tools teilen Datenquellen und Funktionalitäten.

Die Nachteile dieser Architektur liegen auf der Hand:

- Unterschiedliche Handhabung der einzelnen Werkzeuge. Der Schulungsbedarf steigt.
- Unterschiedliche Technologien kommen zum Einsatz. Der Schulungsbedarf und Administrationsaufwand steigen. Spezifische und separate Hardware muss beschafft und betrieben werden.
- Eine Kombination von Daten, welche verschiedene BI-Anwendungen betreffen, ist nicht ohne Weiteres möglich, weil die einzelnen Werkzeuge oft nicht mit allen Datenquellen umgehen können.
- Abfragen müssen in jeder BI-Anwendung separat definiert und gespeichert werden. Dies bedeutet einen Mehraufwand bei der Definition und bei nachträglichen Änderungen. Daraus resultiert eine höhere Fehleranfälligkeit.
- Zugriffsberechtigungen müssen für jede BI-Anwendung und Datenquelle separat definiert werden.

Bei der Architektur von BI-Anwendungen geht es also um zweierlei. Zum einen sollen verschiedene Datenquellen von möglichst vielen BI-Anwendungen gemeinsam genutzt werden können. Zum anderen ist zu vermeiden, dass redundante Funktionalitäten in parallel eingesetzten BI-Anwendungen eingesetzt werden. Die folgenden Abschnitte behandeln diese Aspekte separat voneinander.

### 2.2.1 Die BI-Plattform zur Integration von Datenquellen

Wie bereits angesprochen, werden für eine BI-Anwendung in der Regel Informationen aus Data-Warehouse-Systemen oder MOLAP-Datenbanken verwendet. Zusätzlich werden Informationen aus anderen Datenquellen benötigt. Gleichzeitig werden dieselben Datenquellen

für verschiedene BI-Anwendungen herangezogen. So werden beispielsweise für die BI-Anwendung *Budgetierung* zur Darstellung der Kennzahl *durchschnittlicher Umsatz pro Außendienstmitarbeiter* Daten aus dem Data Warehouse (DWH) und dem Personalsystem (HR) benötigt. Die zweite BI-Anwendung *Konsolidierung* stellt die Umsatzzahlen aus dem Data Warehouse (DWH) den Umsätzen des Gesamtmarktes (bereitgestellt durch ein CSV-File) gegenüber. Die Abfrage der Umsatzzahlen an das DWH muss nun zweimal definiert werden. Aus diesem einfachen Beispiel wird ersichtlich, dass die mehrfache Abfragedefinition von Informationen für mehrere BI-Anwendungen einen erheblichen Mehraufwand darstellen kann. Außerdem steigt mit jeder erneuten Bereitstellung derselben Information die Fehlerwahrscheinlichkeit, insbesondere wenn nachträgliche Änderungen nicht in allen Abfragen nachgepflegt werden. Die sogenannte *BI-Plattform* dient als Schnittstelle zwischen Datenquellen und BI-Anwendungen. Bild 2.6 zeigt die Architektur und damit die Vorteile einer BI-Plattform deutlich. Die BI-Plattform stellt keine physische Datenhaltung dar, sondern basiert auf Metadaten. So werden dort einmalig beispielsweise die Kennzahlen *Umsatz* und *Anzahl Außendienstmitarbeiter* definiert. Als Datenquelle ist für die erste Kennzahl *DWH* (z. B. Tabelle *SALES*, Spalte *TURNOVER*) und für die zweite *HR* (z. B. Cube *SALESREP*, Spalte *EMPNO*) angegeben. Die BI-Anwendungen referenzieren lediglich die BI-Plattform und können auf die dort definierten Kennzahlen lesend zugreifen und diese nach Belieben verwenden. Beim schreibenden Zugriff spezieller BI-Anwendungen kann möglicherweise die BI-Plattform umgangen werden (gestrichelter Pfeil).

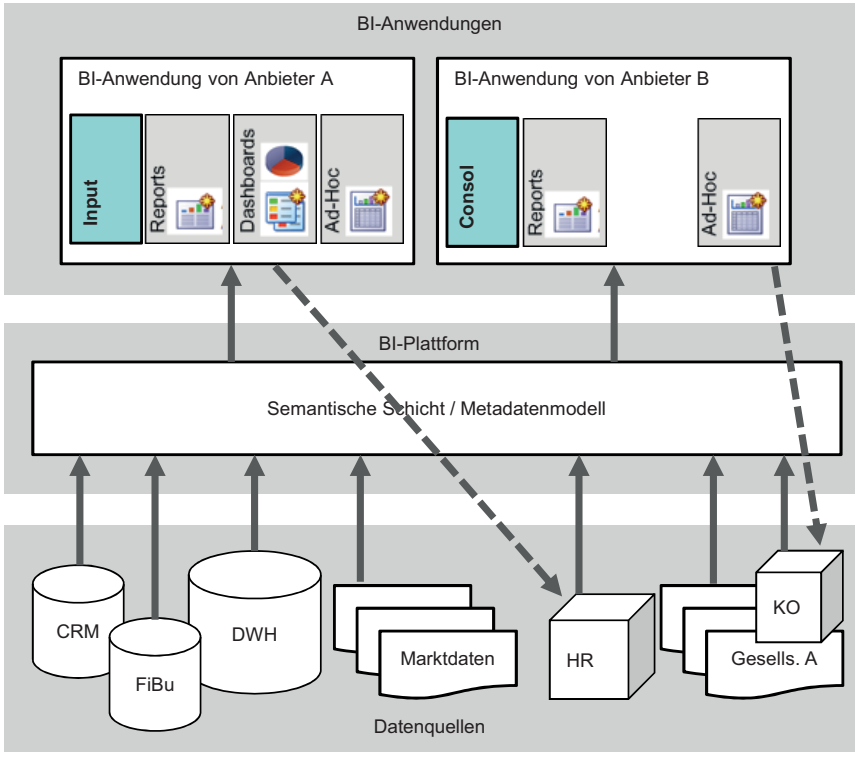


**Bild 2.6** Die BI-Plattform als Schnittstelle zwischen BI-Anwendungen/BI-Tools und Data Warehouse und weiteren Datenquellen

Durch diese Entkopplung von BI-Anwendung und Datenquellen mithilfe einer BI-Plattform können Änderungen sowohl in den Datenquellen als auch in Reports bzw. Analysen von BI-Anwendungen wesentlich einfacher administriert werden.

### 2.2.2 Die BI-Plattform zur Vereinheitlichung der Frontends

Neben den spezifischen Aufgaben der eingesetzten BI-Anwendungen bzw. BI-Tools (z. B. Konzernkonsolidierung) bieten diese in fast allen Fällen die Möglichkeit der Datenvisualisierung. Es besteht also die Möglichkeit, Standardberichte, Ad-hoc-Analysen mit OLAP-Funktionalität (Slice and Dice) und Dashboards zu erstellen bzw. aufzurufen. Meist jedoch sind diese Frontends speziell für die jeweilige BI-Anwendung entwickelt worden, sie sind also proprietär. Daraus resultiert, dass die Benutzeroberfläche über alle BI-Anwendungen hinweg nicht einheitlich zu bedienen ist. Dies kann natürlich nur dann als Nachteil gewertet werden, wenn in einem Unternehmen überhaupt unterschiedliche BI-Anwendungen eingesetzt werden. Dass mit solchen proprietären Frontends keine anwendungsübergreifenden Auswertungen möglich sind, versteht sich von selbst.

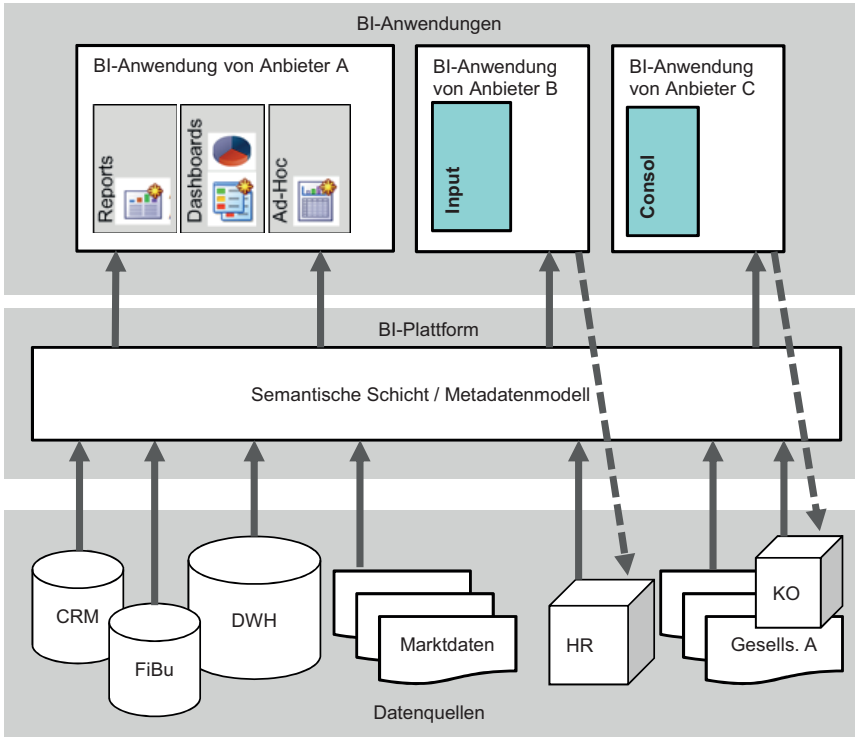


**Bild 2.7** Die BI-Plattform als Schnittstelle zwischen BI-Anwendungen und Datenquellen

Die Frontend-Funktionalität der einzelnen BI-Anwendungen ist meist sehr ähnlich – von Ausnahmen natürlich abgesehen. Gegen ein einheitliches Werkzeug für Standard-Reporting, Ad-hoc-Analyse und Dashboards sprechen nur die womöglich unterschiedlichen Schnittstellen zu

den Datenquellen. Diese sind jedoch beim Einsatz einer BI-Plattform für die Datenintegration einheitlich, wie zuvor beschrieben. Es muss nur sichergestellt werden, dass die Benutzer der einzelnen BI-Anwendungen nur diejenigen Informationen sehen dürfen, die für die jeweilige Anwendung bzw. für den jeweiligen Benutzer relevant sind. Hier hilft eine entsprechende Benutzerverwaltung innerhalb der BI-Plattform. Bild 2.7 zeigt die Zusammenfassung von generellen BI-Anwendungen wie Standard-Reporting, Ad-hoc-Analysen und Dashboards. Voraussetzung ist, dass alle BI-Anwendungen lesenden Zugriff auf die BI-Plattform haben.

**Hinweis**

Das Konzept der BI-Plattform vereinheitlicht nicht nur den Zugriff auf die Datenquellen, sondern auch die Bedienung und Funktionalität der Benutzeroberfläche selbst. Anwendungsübergreifende Auswertungen stellen so kein Problem dar.

## 2.3 Datenhaltung

Daten eines Data Warehouse werden üblicherweise in einer relationalen Datenbank gespeichert und mittels SQL gelesen und verarbeitet. Teilweise werden für Data Marts auch MOLAP-Datenbanken eingesetzt. Diese beiden Technologien sind für viele typische Anwendungsfälle eines Data Warehouse wie betriebswirtschaftliches Standard-Reporting oder Controlling gut geeignet.

Relationale Datenbanken bieten die Möglichkeit, jederzeit Daten über Joins miteinander zu verbinden. Sie bieten die nötige Flexibilität für Ad-hoc-Abfragen sowie die Integration von neuen Anforderungen. Die Abfragesprache SQL ist leicht zu erlernen und wird von jeder etablierten BI- und Reporting-Software, die auf relationale Datenbanken zugreift, direkt unterstützt. Relationale Datenbanken gewährleisten Datenkonsistenz, Hochverfügbarkeit sowie eine gute Verarbeitungsgeschwindigkeit auch für sehr große strukturierte Datenmengen.

Multidimensionale OLAP-Datenbanken strukturieren die Daten aus fachlicher Sicht noch stärker und gehen somit intensiv auf Geschäftsanforderungen ein. Sie unterscheiden Kennzahlen von Dimensionen, bilden Hierarchien auf Stammdaten und stellen Informationen über viele Stufen vorverdichtet und auf hohe Abfragegeschwindigkeit optimiert bereit. Viele Benutzer aus den Fachbereichen kommen mit dieser Strukturierung sehr gut zurecht.

Beide gängigen Technologien präsentieren Daten in einem vereinheitlichten und stabilen Modell, was den Anwendern eine jederzeit verständliche und verlässliche Datenbasis für fachliche Fragestellungen bietet.

**Geeignete Technologien für Data Warehousing**

Der Einsatz relationaler Datenbanken ist für die meisten heute gebräuchlichen Data-Warehouse-Anwendungsfälle die richtige Wahl. Für die fachliche Arbeit in Data Marts sind multidimensionale OLAP-Datenbanken eine gute Ergänzung.

### 2.3.1 Grenzen gängiger DWH/BI-Technologien

Die gängigen Lösungen haben auch Einschränkungen. Der folgende Überblick fasst typische Schwachstellen kurz zusammen.

- Schwierig wird es für relationale und MOLAP-Datenbanken, wenn es um *unstrukturierte Daten* wie Dokumente, Filme und Audiodateien[9] oder um Daten mit sich häufig ändernder oder nicht vordefinierter Struktur geht. Zu letzteren gehören auch Daten, die ihre Struktur implizit mitbringen, wie zum Beispiel XML-Dateien.
- Relationale oder MOLAP-Datenbanken sind nicht darauf optimiert, Zigtausende oder gar *Millionen einzelner Transaktionen pro Sekunde* zu bewältigen, wie sie zum Beispiel bei der zeitnahen Verarbeitung von Maschinen-, Sensor- oder Webdaten anfallen können. Dabei ist hier nicht der schiere Durchsatz an Datensätzen problematisch: Selbst auf einem modernen Notebook mit SSD-Laufwerk können heute mehr als hunderttausend kleine Datensätze pro Sekunde in ein RDBMS geladen und dabei noch mit anderen Daten verbunden werden. Schwierig ist es vielmehr, jeden Datensatz separat in einer eigenen Transaktion zu verarbeiten. Der durch diese Vereinzelung entstehende zusätzliche Ressourcenverbrauch und die durch Konsistenzregeln bedingten „Flaschenhälse“ sind bei relationalen Datenbanken naturgemäß besonders ausgeprägt.
- Auch die *Antwortzeiten* bei Benutzerabfragen sind auf gängigen relationalen Datenbanken und großen Datenmengen oft eingeschränkt, falls dazu riesige Datenmengen ausgewertet werden müssen. Manchmal ist auch mittels OLAP-Lösungen aufgrund besonders hoher Datenmengen feinster Granularität (bspw. „Call-Data-Records“ bei Telefondienstleistern) keine Beschleunigung mehr möglich. Es kommt auch vor, dass die Zeitnähe von Abfragen eine Vorverdichtung verhindert oder neue Anforderungen auf großen Beständen ad hoc umgesetzt werden müssen. Dann werden andere technische Ansätze benötigt, um sonst stundenlange Antwortzeiten auf Minuten oder Sekunden zu reduzieren.
- Die reine *Menge an Daten* ist nur selten eine unüberwindliche Grenze für relationale Datenbanksysteme respektive große Datenbankcluster. Allerdings führen Datenmengen im hohen Terabyte- oder gar Petabyte-Bereich dazu, dass gängige Verfahren (bspw. Backup oder komplexere Migrationen von Datenstrukturen) nicht mehr einfach nach gewohntem Muster durchgeführt werden können, ohne die Betriebsfähigkeit spürbar einzuschränken.
- Nicht zuletzt spielen die *Lizenz- und Hardwarekosten* bei moderner, leistungsstarker RDBMS- oder OLAP-Software für den Data-Warehouse-Einsatz eine erhebliche Rolle. Hier werden gerade bei besonders großen Datenmengen Möglichkeiten zur Kostenersparnis zunehmend attraktiver.

[9] Diese Informationen sind natürlich schon strukturiert, jedoch nicht in einer Form, die herkömmliche Auswertungen und Abfragen auf die gespeicherten Daten erlaubt.

**Grenzen herkömmlicher relationaler und multidimensionaler Technologien**

Für Business-Intelligence-Anforderungen rund um unstrukturierte Daten, mit extrem hohen Transaktionsraten, besonderer Zeitnähe (Latenz), kurzer Antwortzeit bei Ad-hoc-Abfragen, besonders großen Datenmengen oder erforderlicher Kostenersparnis kommen weitere Technologien ins Spiel.

Diese immer stärker aufkommenden Anforderungen wurden bereits im Jahre 2001 vom META-Group-Analysten Doug Langley (heute Gartner) durch die bekannten drei „V" (High **V**olume, High **V**elocity, High **V**ariety) benannt. Erst später wurden diese Begriffe als Definition für *„Big Data"* proklamiert und dafür kostengünstige und innovative Methoden für eine neue und bessere Informationsverarbeitung gefordert.

Die aus heutiger Sicht interessantesten Vertreter solcher alternativen, im BI-Umfeld nutzbaren Big-Data-Technologien speichern und verarbeiten Daten entweder direkt in sehr frei definierbaren Strukturen (oder Dateien) oder möglichst komplett im Hauptspeicher.

Große Freiheiten sowohl bei Datenstrukturen als auch Datenmengen bieten manche *NoSQL-Datenbanken*. Solche Systeme fokussieren dazu auf geringe Latenz bei der Verarbeitung von hohen Transaktionsmengen. Die Verarbeitung von sehr großen bzw. sehr vielen Dateien wird dagegen von der Software *Apache Hadoop* dominiert. Der Zusammenschluss zahlreicher, auf Hadoop aufsetzender Softwareprodukte bildet das „Hadoop-Ecosystem", für das inzwischen zahlreiche, zum Teil auch kommerzielle Distributionen existieren. Solch eine Plattform beinhaltet neben Hadoop auch NoSQL-Datenbanken, Orchestrierungs-, Scripting-, Programmier- und Datenintegrationswerkzeuge und noch vieles mehr.

Dass die einzelnen Komponenten oft separiert entwickelt werden, hat allerdings nicht nur Vorteile: Die Flexibilität der Lösungen ist zwar hoch und es gibt eine große Auswahl spezialisierter Tools für jeden Einsatzbereich. Aber deren Zusammenspiel funktioniert heute noch nicht nahtlos. Außerdem sind viele Tools gegenwärtig in einem frühen Entwicklungsstadium. Die für einen hohen Stabilitäts- und Reifegrad erforderliche Konsolidierung ist daher noch lange nicht abgeschlossen.

Wir wollen Hadoop & Co. in diesem Buch nicht allzu sehr technisch vertiefen. Dafür gibt es zahlreiche ausgezeichnete Einführungen, Beschreibungen und Dokumentationen. Wir müssen aber einen Blick auf die besonderen Vor- und Nachteile werfen, um zu verstehen, warum in manchen Fällen der Einsatz dieser „Big-Data"-Technologien im BI-Umfeld eine wichtige Option sein kann.

### 2.3.2 Datenhaltung im Hadoop-Ecosystem

Apache Hadoop ist ein Framework, um auf sehr vielen einfachen, autonomen Rechnern in einem Netzwerk Programme auszuführen, die verteilt an gemeinsamen Aufgaben arbeiten. Hauptziel ist die möglichst lineare *Skalierbarkeit* auf praktisch unbegrenzt vielen, *kostengünstigen* Rechenknoten, die mit einfacher PC-Technik aufgebaut sind. Hadoop basiert auf einem speziellen Dateisystem namens HDFS (Hadoop Distributed File System). Mit HDFS

werden Dateien auf den einzelnen Knoten verteilt gespeichert, also jeweils nur ein Teil einer Datei auf einem[10] Rechenknoten abgelegt. Programme, die nun diese beliebig strukturierten Daten verarbeiten, tun dies oft nach dem sogenannten *MapReduce*-Programmiermodell. Solche MapReduce Implementierungen existieren in Form von Softwarebibliotheken für zahlreiche Programmier- und Skriptsprachen wie Java, C/C++ oder Python. Sie werden darüber hinaus auch von weiteren Softwareprodukten wie beispielsweise dem Statistikpaket *„R“* oder dem SQL-Konverter *Hive* genutzt.

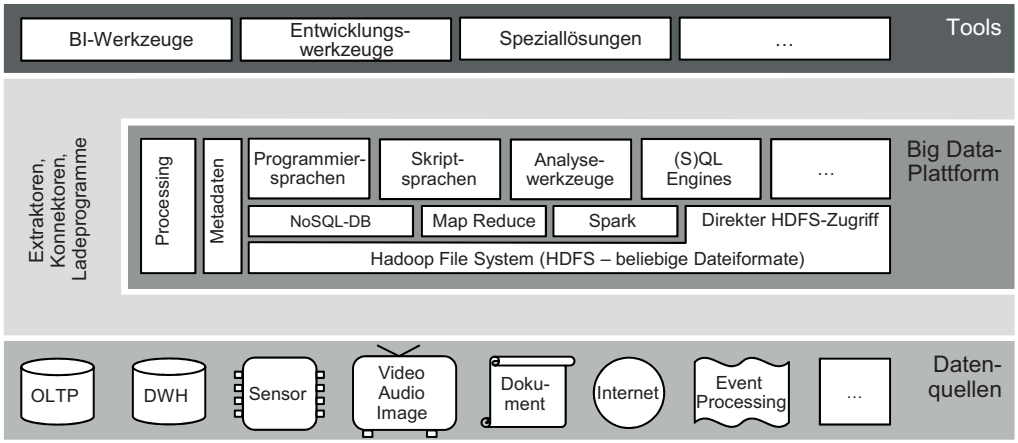


**Bild 2.8** Datenhaltung und Verarbeitung mittels Big Data-Technologien aus Sicht des Hadoop-Ecosystems

MapReduce gibt vor, wie Daten in einem Rechencluster möglichst effektiv gescannt, interpretiert, verdichtet und zusammengeführt werden. Während in relationalen Datenbankmanagementsystemen der optimale Ausführungsplan für die Verarbeitung von SQL-Befehlen durch sogenannte *„Optimizer“*-Module automatisch errechnet und ausgeführt werden, bleibt die Zusammenstellung der Verarbeitungsmethodik und deren Reihenfolge bei MapReduce dem Programmierer überlassen. Einige Tools wie Hive oder die Scripting-Engine PIG generieren zwar bereits optimierte MapReduce-Programme, sind aber bei Weitem noch nicht so effizient wie gängige Datenbank-Optimizer.

Besonders die quantitative Berücksichtigung von Datenmenge und -verteilung, die sogenannte kostenbasierte Optimierung, ist in solchen Werkzeugen entweder noch nicht vorhanden oder in einem relativ frühen Entwicklungsstadium. MapReduce-Programme sind zudem reine Batch-Programme, die üblicherweise im Minuten- oder Stundenbereich arbeiten und nicht für interaktive Arbeit gedacht sind.

Manche NoSQL-Datenbanken wie *HBase* oder neuere MapReduce-Alternativen wie Apache *Spark* arbeiten ohne MapReduce direkt mit HDFS-Daten und optimieren die Antwortzeiten für bestimmte Anforderungen durch spezielle Indexierung oder Caching-Mechanismen.

[10] Tatsächlich werden alle Dateien aus Sicherheitsgründen auf mehreren (standardmäßig auf drei Rechnern) gespeichert.

**Das Hadoop-Ecosystem**

... besteht aus einer über fast beliebig viele kostengünstige Rechner verteilten Datenhaltung und -verarbeitung und einer großen Menge von Tools für unterschiedliche Anwendungszwecke.

- *Unstrukturierte Daten* wie Dokumente oder implizit strukturierte Daten wie XML können praktisch mit allen Programmier- und Skriptsprachen verarbeitet werden. Die über die Komponente HCatalog definierbaren Metadaten erlauben darüber hinaus beliebig viele, auch mehrfache und unterschiedliche relationale Sichten auf jede Art von Daten, da die Zugriffsmechanismen und die Transformation in die relationale Darstellung frei implementierbar sind.

  *Nachteil:* Bis auf wenige vordefinierte Formate müssen die Zugriffsmechanismen allerdings auch tatsächlich selbst implementiert werden. Immerhin steigt die Zahl neuer, allgemein verfügbarer Formatimplementierungen inzwischen stetig an.

- Zur zeitnahen Verarbeitung *extrem vieler einzelner Transaktionen* werden diverse Komponenten benötigt. Die Möglichkeiten sind vielfältig. Beispielsweise kann ein verteiltes Messaging-System wie Apache *Kafka* zum „Einfangen" der Transaktionen dienen. Mittels einer „Distributed Computing Engine" wie Apache Storm werden die Daten dann über viele Rechenknoten vorverarbeitet, klassifiziert und in eine NoSQL-Datenbank wie HBase geschrieben. Die Verarbeitung einzelner Transaktion kann so innerhalb von Millisekunden durchgeführt werden. Durch die extreme Skalierbarkeit über sehr viele Rechenknoten sind damit auch Hunderttausende oder Millionen von Transaktionen pro Sekunde zu bewältigen.

  *Nachteil:* Architektur und Implementierung sind aufwendig und die in vielen NoSQL-Datenbanken abgelegten Daten sind nicht beliebig verknüpfbar (Joins gibt es dort nicht immer). Zudem ist die Sicht auf die Daten oft nicht auf jedem Knoten identisch, denn die Konsistenz ist je nach Werkzeug nicht zu jeder Zeit durchgängig gewährleistet. Und schließlich sind NoSQL-Datenbanken nicht ohne Weiteres durch gängige BI-FrontEnd-Werkzeuge nutzbar.

- Auch extrem große *Mengen an Daten*, zum Beispiel Hunderte von Petabytes[11], können in einem Hadoop-Cluster gespeichert und verarbeitet werden. Backup und Recovery bzw. Hochverfügbarkeit werden hier anders gelöst als in relationalen Datenbanken, unterliegen dabei allerdings auch anderen Einschränkungen. Ein vollständig konsistentes Online-Backup ist teilweise nicht möglich. Dafür sind aber auch örtlich weit verteilte Infrastrukturen (Stretch-Cluster) noch effizient möglich.

- Bei *Lizenz- und Hardwarekosten* sind Lösungen rund um Hadoop sehr günstig, wenn man die Investitionskosten pro Terabyte Nutzdaten betrachtet. Da vor allem weit verbreitete und günstige Hardware verwendet wird, kann man heute von weniger als 1000 € pro Terabyte unkomprimierter Rohdaten für Speicherung und Verarbeitung ausgehen. Bei der Anmiete in einer Cloud-Lösung können diese Preise sogar auf Tage, Stunden oder gar Minuten heruntergebrochen werden. Dazu kommen keine oder geringe Kosten für Soft-

[11] Yahoo! hatte bereits 2011 einen Hadoop-Cluster mit 42000 Knoten mit mehr als 180 Petabyte Rohdaten in Betrieb.

warelizenzen, da die meisten Tools unter einer Open-Source-Lizenz nutzbar sind. TCO[12]-Betrachtungen kommen allerdings zu anderen Ergebnissen, wenn Support, Know-how, Aufbau und Pflege, Administration, Standkosten, Backup und alle weiteren üblichen Kostenblöcke eingerechnet werden. Dann schrumpft der Vorteil bzw. kann bei unsachgemäßer Nutzung auch deutlich schlechter sein als bei kommerziellen RDBMS mit lokalem oder netzbasiertem Storage (SAN/NAS).

**Schema versus Schemalos**

Freiheit in der Datenstruktur, also der Verzicht auf ein festes „Schema", hat den wesentlichen Vorteil, viele verschiedene Sichten auf dieselben Grunddaten bereitstellen und diese jederzeit ändern zu können. Dafür ist diese Freiheit immer mit der Notwendigkeit verbunden, die Daten bei jedem Zugriff wiederholt zu interpretieren. Will man diesen ressourcenintensiven Aufwand nur einmalig beim Befüllen einer Struktur mit Daten betreiben, verliert man diese Freiheit, gewinnt dafür aber deutlich an Effektivität und Effizienz bei mehrfachen Zugriffen. Verfechter eines festen Schemas („Schema on write") verweisen gerne auf die fachliche Orientierung und die Erfordernis, Anforderungen schon in der Planung intensiv durchzudenken. Außerdem muss die fachliche Interpretation von Daten ohnehin spätestens beim Analysieren erfolgen, also zumindest ein „Schema-On-Read"-Ansatz verfolgt werden. Wir empfehlen den ganz pragmatischen Ansatz, beide Verfahren ihrer Eignung gemäß einzusetzen, also Vor- und Nachteile gegenüberzustellen und die Aufwände und Risiken eines ggf. nötigen Technologiewechsels mit einzukalkulieren.

Was in unserer Betrachtung noch fehlt, sind Lösungen, um *Antwortzeiten* bei beliebigen Ad-hoc-Benutzerabfragen minimal zu halten, ohne deren Vorverarbeitungszeit zu erhöhen. Damit beschäftigt sich der folgende Abschnitt.

### 2.3.3 In-Memory-Datenbanken

Es gibt zahlreiche Datenbanken sowohl im SQL- als auch im NoSQL-Umfeld, die ihre Daten ganz oder vorwiegend im Hauptspeicher – und das genau bedeutet „In-Memory"[13] wörtlich – halten und verarbeiten. Im Rahmen unserer oben beschriebenen Anforderung, Daten ad hoc frei miteinander zu kombinieren, sind relationale Datenbanken nach wie vor die wichtigsten Vertreter. Gerade in diesem Markt halten In-Memory-Techniken heute vermehrt Einzug. Wir beschränken uns im Folgenden auf diese Spezies und betrachten daran Funktionsweisen, Vor- und Nachteile etwas genauer.

[12] TCO = Total Cost of Ownership (Gesamtkosten über einen bestimmten Zeitraum).
Siehe bspw. *http://rainstor.com/how-much-is-that-hadoop-cluster-really-costing-you/*

[13] Neben „In-Memory Databases" werden auch Begriffe wie „Main-Memory Databases" oder „Memory-Resident Databases" verwendet.

**In-Memory & das Hadoop-Ecosystem**

Auch im Hadoop-Umfeld gibt es hauptspeicheroptimierte und SQL-fähige Werkzeuge wie Impala. Sie nutzen im Wesentlichen dieselben oder ähnliche Methoden zur Optimierung der Antwortzeit wie die im Folgenden vorgestellten Lösungen.

Ein Kriterium für Geschwindigkeit in der Datenverarbeitung ist die *Latenz*, also die Zeit für einen einfachen Zugriff auf einen Wert. Für einen einzelnen Datenzugriff liegt diese Zeit bei einer modernen mechanischen Festplatte bei ca. zehn Millisekunden, bei einer schnellen Solid State Disk (SSD) bei ca. zehn Mikrosekunden und bei einem gängigen Server-Hauptspeicherbaustein bei ca. zehn Nanosekunden. Letzteres ist eine Million mal schneller als der Zugriff auf eine mechanische Festplatte!

Ein weiteres wichtiges Maß in der Datenverarbeitung ist der *Durchsatz*. Er wird in GB pro Sekunde gemessen. Hier ist der Unterschied nicht ganz so eklatant: Eine mechanische Festplatte schafft ungefähr 200 MB in der Sekunde. Eine SSD oder Flash Disk ermöglicht bis zu 2 GB pro Sekunde. Ein moderner Speicherriegel dagegen liegt bei ca. 15 GB pro Sekunde. Das sind bereits sehr gute Gründe, möglichst viel Verarbeitung direkt im Hauptspeicher zu erledigen und nicht erst Daten aus viel langsameren Medien ins RAM zu laden.

Noch viel schneller sind *CPU-Caches*, spezielle Speicherbausteine innerhalb eines Prozessors. Diese in mehreren Ebenen angeordneten Bausteine sind allerdings extrem teuer und üblicherweise auf wenige MB limitiert. Wenn Daten hier liegen, können sie am schnellsten verarbeitet werden.

Dann liegt die Lösung aller Performance-Probleme also darin, einfach alle Daten in den Hauptspeicher zu laden und nur noch dort zu verarbeiten? So einfach ist es leider nicht, und zwar aus den folgenden Gründen:

- Erstens werden Daten von Festplatte schon innerhalb der Festplatte, im Storage-Subsystem, auf Betriebssystemebene und im Hauptspeicherbereich der Datenbank „gecached“. So erfolgen Zugriffe auf Daten in Datenbanken schon heute oft zu 50 %, 90 % oder gar über 99 % in Caches oder im Hauptspeicher.
- Zweitens sind viele Verarbeitungsalgorithmen und Speicherstrukturen von Datenbanken schon lange auf eine Minimierung der Plattenzugriffe ausgelegt. Zum Beispiel optimieren die seit 40 Jahren üblichen B[14]-Baum-Indexe den Zugriff auf Festplatten auf Kosten der Effizienz bei der Verarbeitung rein im Hauptspeicher. Solche heute in den meisten Datenbanken eingesetzten Verfahren sind also auf den Datentransfer zwischen Platte und Hauptspeicher optimiert und nicht auf die Verarbeitungsgeschwindigkeit von Daten innerhalb des Hauptspeichers.
- Drittens wird nicht nur gelesen, sondern auch geschrieben. Schreiben muss auch dann die Konsistenz sicherstellen, wenn am Datenbankserver direkt nach dem Commit der Stecker gezogen wird. Dieses „Durability“-Konzept ist aber nur einzuhalten, wenn die Daten irgendwo dauerhaft gespeichert werden. Mit aktuellen Hauptspeicherbausteinen

[14] „B“ steht dabei für Prof. Rudolf **B**ayer aus München, der diese Strukturen und deren Mechanik in den 70er-Jahren erarbeitet hat.

ist dies nicht möglich. Darum müssen zumindest die „Transaktionslogs“[15], das sind Dateien, die einfach nur die Änderungen in einer langen Liste enthalten, spätestens beim „Commit“ nach Änderungen diese physisch auf Festplatte oder SSD geschrieben werden.[16]

**„In-Memory“-Datenbanken heute**

In-Memory Datenbanken nutzen Hauptspeicher und CPU so effektiv wie nur möglich. Dafür verwenden sie aber andere Algorithmen und andere interne Datenstrukturen als die „klassischen“ Datenbanken, die viel Augenmerk auf die Ausgewogenheit zwischen Festplatten und Hauptspeicher legen. In-Memory-Lösungen profitieren damit überproportional von viel Hauptspeicher.

Natürlich sind vor allem die Kosten für Hauptspeicher ausschlaggebend. Die werden in Relation zur Hauptspeicherkapazität zwar immer kleiner, dasselbe gilt allerdings auch für den Preis je Terabyte bei Festplatten. Entscheidend ist hierbei, dass die Datenmenge in zahlreichen Einsatzgebieten zwar gleichzeitig auch steigt, aber bei Weitem nicht im gleichen Maße, wie die Kosten für Hauptspeicher sinken. Dadurch wird Hardware mit viel Hauptspeicher für die meisten Einsatzgebiete erst attraktiv. So sind heute Rechner mit mehr als einem Terabyte Hauptspeicher keine Seltenheit mehr.

Zunächst muss man zwischen unterschiedlichen Einsatzgebieten von In-Memory-Datenbanken unterscheiden. Es gibt analyseoptimierte und transaktionsoptimierte Ansätze. Erstere haben ihr Augenmerk auf der Beschleunigung von Lesevorgängen, insbesondere bei komplexeren, analytischen Abfragen. Letztere beschleunigen hauptsächlich Applikationen mit sehr vielen einzelnen Schreiboperationen auf Daten.

Schreiboptimierte Verfahren versuchen meist, den Flaschenhals der Konkurrenzmechanismen vieler schreibender Transaktionen in den Griff zu bekommen. Analyseoptimierte In-Memory-Datenbanken verwalten ihre Daten vorzugsweise **spaltenorientiert** und nicht zeilenorientiert. Das heißt, in einer Tabelle liegen zumindest im Hauptspeicher zunächst die Daten der ersten Spalte hintereinander, dann die der zweiten Spalte und so weiter. Das bringt einen erheblichen Vorteil, wenn man beispielsweise nur zwei oder drei Spalten einer Tabelle mit 100 Spalten lesen möchte, wie es bei analytischen Abfragen oft der Fall ist (es ist allerdings nachteilig, wenn alle hundert Spalten gelesen werden sollen). Ein weiterer Vorteil der spaltenweisen Speicherung ist die Möglichkeit, die Inhalte einer Spalte zu sortieren und dann mit möglicherweise erheblichem Platzgewinn zu komprimieren. Das sogenannte Run Length Encoding (RLE) soll hier als Beispiel genannt werden, wie für Spalten mit wenigen unterschiedlichen Werten im Extremfall sogar Gigabytes an (stark redundanten) Daten in wenige Bytes verdichtet werden können.

Nicht der In-Memory-Verarbeitung vorbehalten, aber oft damit einher geht die Nutzung moderner CPU-Funktionen wie beispielsweise **SIMD**. SIMD steht für „Single Instruction Multiple Data“ und erlaubt modernen Prozessoren mit sehr vielen logischen Recheneinheiten (ALU) innerhalb eines Rechenkerns, pro Takt zahlreiche Einzeldaten zu verarbeiten. So

[15] Je nach Datenbankprodukt auch „Redologs“ o. Ä. genannt

[16] Nicht volatile Speicherbausteine, sogenannte NVRAM, befinden sich im Status der Forschung und könnten in Zukunft den langsamen Abgleich mit externen Speichermedien obsolet machen.

kann ein großes 128-Bit-CPU-Register in einem Schritt auch mit vier 32-Bit-Werten operieren. Das ermöglicht zum Beispiel das Lesen von mehr Datensätzen pro Sekunde und CPU-Core, als die CPU-Frequenz eigentlich zulassen sollte.

Typische Vertreter der In-Memory-Spezies im RDBMS-Umfeld sind beispielsweise SAP HANA, MS SQL Server In-Memory Tables, die Oracle In-Memory-Option, Exasol oder die IBM DB2-Erweiterung BLU Acceleration. Allerdings verbergen sich dahinter teils sehr unterschiedliche Ansätze. So ist SAP HANA eine reine In-Memory-Datenbank, während die Oracle In-Memory-Option aus zwei separaten Erweiterungen – einer für schnelles Schreiben, einer für schnelles Lesen – der gängigen Datenbank besteht.

# Index

## E



## F



## G

## H



## I



## J



## K



## L



## M



## N



## O