

Dieses Kapitel behandelt kurz Grundlagen zu Datenbanksystemen und Rechnernetzen bzw. verteilten Systemen, die für das weitere Verständnis des Buchs benötigt werden. Das im größten Teil dieses Buchs unterstellte Datenmodell ist das relationale Datenmodell, da die relationale Datenbanktechnologie weiterhin die in der Praxis dominierende Form des Datenmanagements darstellt. Jedoch sind viele Prinzipien der verteilten und parallelen Datenbankverarbeitung auch auf nichtrelationale Systemansätze wie NoSQL-Systeme übertragbar, wie an verschiedenen Stellen verdeutlicht werden wird. Ein wesentlicher Vorteil des relationalen Modells zur verteilten und parallelen Datenbankverarbeitung liegt darin, dass die Anfragen mengenorientiert sind und somit ein weit größeres Optimierungspotenzial besteht als für satzorientierte Zugriffsoperationen und Anfragesprachen [9]. Die Mehrzahl Verteilter und Paralleler Datenbanksysteme basiert daher auch auf dem relationalen Datenmodell.

Wir diskutieren zunächst Grundlagen des relationalen Datenmodells und seiner Operationen. Es folgt die Beschreibung des internen Aufbaus von Datenbanksystemen (DBS), welche aus einem Datenbankverwaltungssystem (Database-Management-System, DBMS) und der Datenbank selbst bestehen. Die zur Transaktionsverwaltung benötigten Funktionen werden mit dem Transaktionskonzept in Abschn. 2.3 erläutert. Nach einer Diskussion wesentlicher DBS-Einsatzformen (OLTP, OLAP) wenden wir uns sogenannten *NoSQL-Systemen* zu, die zunehmend Verbreitung finden und auch meist eine verteilte Datenverwaltung unterstützen. Abschließend diskutieren wir noch Grundlagen von Rechnernetzen, die für das weitere Verständnis hilfreich sind. Mehrere Lehrbücher bieten weitergehende Informationen zur Einführung in Datenbanksysteme, u. a. [18, 26], zu DBS-Implementierungskonzepten für den zentralen Fall [14, 25] sowie zu Rechnernetzen [29, 21].

2.1 Relationales Datenmodell

In relationalen Datenbanken werden die Daten in Tabellen oder Relationen gespeichert. Jede Relation besteht aus einer bestimmten Anzahl von Spalten oder Attributen sowie einer Menge von Zeilen oder Tupeln (Sätzen). Die Anzahl der Attribute bestimmt den *Grad*, die Anzahl der Tupel die *Kardinalität* einer Relation. Abbildung 2.1 zeigt ein Beispiel zweier Relationen vom Grad 3 und Kardinalität 6 bzw. 5.

Jedem Attribut ist eine sogenannte *Domain* (Definitionsereich) zugeordnet, die die zulässigen Werte festlegt. Die Beschreibung einer Relation umfasst den Namen der Relation und deren Attribute sowie die zugehörigen Domains. Die Ausprägung der Relation ist durch die Menge der Tupel gegeben und entspricht einer Teilmenge des kartesischen Produktes über den Attribut-Domains. Die Mengeneigenschaft von Relationen bedeutet, dass kein Tupel mehrfach vorkommen darf und dass keine vorgegebene Reihenfolge unter den Tupeln besteht.

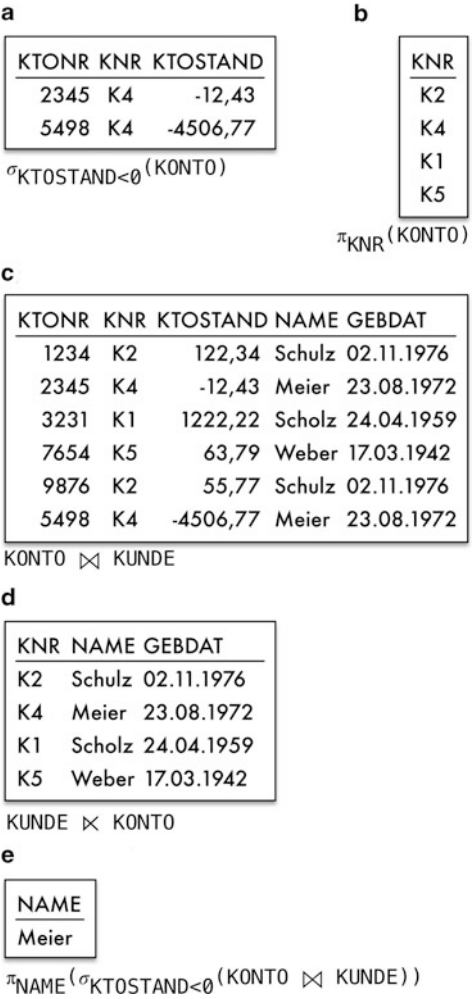
Das Relationenmodell schreibt zwei modellinhärente Integritätsbedingungen vor, die sogenannten *relationalen Invarianten*. Die Primärschlüsselbedingung verlangt, dass zu jeder Relation ein Attribut bzw. eine Kombination von Attributen als *Primärschlüssel* fungiert, mit dem Tupel eindeutig identifiziert werden können. In Abb. 2.1 bilden z. B. die Attribute KTONR (Relation KONTO) und KNR (Relation KUNDE) geeignete Primärschlüssel. Die zweite modellinhärente Integritätsbedingung betrifft sogenannte *Fremdschlüssel*, mit denen Beziehungen zwischen Relationen realisiert werden können. In Abb. 2.1 ist das Attribut KNR (Kundennummer) in Relation KONTO ein solcher Fremdschlüssel, mit dem der Inhaber eines Kontos durch einen Verweis auf den Primärschlüssel der Relation KUNDE repräsentiert wird. Die Fremdschlüsselbedingung verlangt, dass das durch einen Fremdschlüsselwert referenzierte Tupel in der Datenbank existieren, d. h. dass in der referenzierten Relation ein entsprechender Primärschlüsselwert definiert sein muss. Dies nennt man referenzielle Integrität.

Zur Beschreibung relationaler Anfragen greifen wir (neben SQL) vor allem auf die *Relationenalgebra* zurück. Die Relationenalgebra erlaubt eine kompakte Darstellung von Operationen und spezifiziert explizit die Basisoperatoren, die bei der Anfragebearbeitung auszuführen sind. Die Operatoren der Relationenalgebra sind mengenorientiert, da sie aus

Abb. 2.1 Beispiele von Relationen

KONTO			KUNDE		
KTONR	KNR	KTOSTAND	KNR	NAME	GEBDAT
1234	K2	122,34	K2	Schulz	02.11.1976
2345	K4	-12,43	K4	Meier	23.08.1972
3231	K1	1222,22	K3	Müller	04.07.1987
7654	K5	63,79	K1	Scholz	24.04.1959
9876	K2	55,77	K5	Weber	17.03.1942
5498	K4	-4506,77			

Abb. 2.2 Beispiele relationaler Operatoren **a** Selektion, **b** Projektion, **c** natürlicher Verbund, **d** Semi-Join, **e** Anfrage mit mehreren Operatoren (bezogen auf die Relationen aus Abb. 2.1)



einer oder zwei Eingaberelationen wiederum eine Relation erzeugen. Neben allgemeinen mengentheoretischen Operatoren wie Vereinigung ($\cup$), Durchschnitt ($\cap$), kartesisches Produkt ($\times$) oder Differenz ($-$) interessieren vor allem die relationalen Operatoren Selektion, Projektion und Verbund (Join).

Mit der *Selektion* $\sigma_P(R)$ wird eine zeilenweise (horizontale) Teilmenge einer Relation R bestimmt, die alle Tupel enthält, die das Selektionsprädikat P erfüllen. Ein Beispiel dazu findet sich in Abb. 2.2a. Die *Projektion* $\pi_{Attributliste}(R)$ bildet eine spaltenweise (vertikale) Teilmenge der Eingaberelation, wobei alle nicht spezifizierten Attribute „weggefiltert“ werden (Beispiel: Abb. 2.2b). Dabei werden zudem Duplikate eliminiert, um

die Mengeneigenschaft der Ergebnisrelation zu wahren. Der *Verbund* ($\bowtie$) schließlich erlaubt die Verknüpfung zweier Relationen, die Attribute mit übereinstimmenden Domains besitzen. Der mit Abstand wichtigste Verbundtyp ist der *Gleichverbund* (*Equi-Join*), bei dem die Verknüpfung durch eine Gleichheitsbedingung auf den Verbundattributen definiert ist. Der Gleichverbund zwischen Relation R mit Verbundattribut r und Relation S mit Verbundattribut s kann als Selektion auf dem kartesischen Produkt zwischen R und S definiert werden:

$$R \bowtie_{r=s} S = \sigma_{r=s}(R \times S)$$

Der *natürliche Verbund* zwischen zwei Relationen, bei denen die Verbundattribute den gleichen Namen besitzen, ist definiert als Gleichverbund zwischen den Relationen, wobei die Verbundattribute nur einmal im Ergebnis vorkommen. Aufgrund der Namenskonvention kann beim natürlichen Verbund die Join-Bedingung weggelassen werden ($R \bowtie S$). Abbildung 2.2c zeigt ein Beispiel für den natürlichen Verbund.

Der *Semi-Join* zwischen zwei Relationen R und S ($R \ltimes S$) ist definiert als ein Verbund, in dessen Ergebnis nur Attribute der Relation R enthalten sind:

$$R \ltimes_{r=s} S = \pi_{R\text{-Attribute}}(R \bowtie_{r=s} S)$$

Abbildung 2.2d zeigt ein Beispiel für einen (natürlichen) Semi-Join, bei dem die Verbundbeziehung sich auf gleichnamige Attribute in den Eingangstabellen bezieht. Semi-Joins haben vor allem im Kontext von Verteilten Datenbanksystemen Bedeutung erlangt. Sie erlauben eine Verkleinerung der Ergebnismenge gegenüber dem normalen Join, was zu einer Reduzierung der zu übertragenden Datenmengen genutzt werden kann (s. Kap. 8).

Da die Operatoren Relationen als Eingabe verwenden und wiederum Relationen erzeugen, können sie miteinander kombiniert werden, um komplexere Anfragen zu formulieren. Ein Beispiel dazu ist in Abb. 2.2e gezeigt.

In der Praxis werden DB-Anfragen an relationale Datenbanksysteme in der standardisierten und weitverbreiteten Sprache *SQL* formuliert, die wesentlich nutzerfreundlicher als die Relationenalgebra ist. Im Gegensatz zur Relationenalgebra unterstützt SQL auch die Berechnung sogenannter *Aggregatfunktionen* (z. B. **MIN**, **MAX**, **SUM**, **COUNT**, **AVG**) sowie die Sortierung von Ergebnismengen. In diesem Buch werden nur vereinzelt Beispiele in SQL angegeben, die zudem weitgehend selbsterklärend sein dürften. Die Anfrage aus Abb. 2.2e kann in SQL z. B. folgendermaßen formuliert werden:

```
SELECT NAME {Projektion auf Attribut Name}
FROM KONTO NATURAL JOIN KUNDE
      {Angabe der Tabellen und Join-Ausdruck}
WHERE KTOSTAND < 0 {Selektionsbedingung}
```

2.2 Aufbau von Datenbanksystemen

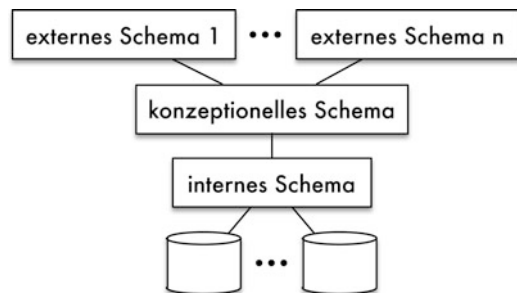
Wir diskutieren zunächst die sogenannte Schemaarchitektur zentralisierter DBS, mit der unterschiedliche Sichtweisen auf eine Datenbank unterstützt werden. Anschließend beschreiben wir den internen DBS-Aufbau anhand eines dreistufigen Schichtenmodells.

2.2.1 Schemaarchitektur

Ein wesentlicher Vorteil moderner Datenbanksysteme liegt in der Bereitstellung einer hohen Datenunabhängigkeit sowie der Möglichkeit, verschiedenen Benutzern unterschiedliche Sichtweisen auf eine Datenbank zu ermöglichen. Dies wird durch eine dreistufige Schemaarchitektur unterstützt (Abb. 2.3), welche vom *ANSI/SPARC-Komitee* vorgeschlagen wurde und in vielen DBS realisiert ist. Im Einzelnen sind dabei drei Arten von Schemata zur Beschreibung des Datenbankaufbaus zu unterscheiden:

- Im Mittelpunkt steht dabei das *konzeptionelle Schema*, das den logischen Aufbau der Datenbank vollständig beschreibt. Im Falle relationaler Datenbanken enthält das konzeptionelle Schema die Definition sämtlicher Relationen, Attribute, Domains, Integritätsbedingungen etc.
- Benutzer bzw. Anwendungsprogramme greifen i. Allg. nicht direkt über das konzeptionelle Schema auf die Datenbank zu, sondern über auf ihre Anforderungen und Zugriffsrechte ausgerichtete *externe Schemata*. Ein externes Schema enthält i. Allg. nur eine Teilmenge der Objekte des konzeptionellen Schemas, wodurch sich für die betreffende Benutzergruppe eine einfachere DB-Benutzung ergibt sowie eine Zugriffskontrolle unterstützt wird. Es wird damit auch ein höherer Grad an Datenunabhängigkeit als mit dem konzeptionellen Schema erreicht, da Änderungen im logischen DB-Aufbau nur dann Rückwirkungen auf Anwendungen haben, falls sie Objekte des jeweiligen externen Schemas betreffen (logische Datenunabhängigkeit).
- Das *interne Schema* beschreibt, wie logische Objekte des konzeptionellen Schemas physisch gespeichert werden sollen (Clusterbildung, Komprimierung etc.) und welche

Abb. 2.3 DB-Schemaarchitektur nach ANSI/SPARC



Indexstrukturen vom DBMS zu warten sind. Diese Festlegungen werden automatisch vom DBMS oder vom Datenbankadministrator getroffen und sind für den DB-Benutzer vollkommen transparent (physische Datenunabhängigkeit).

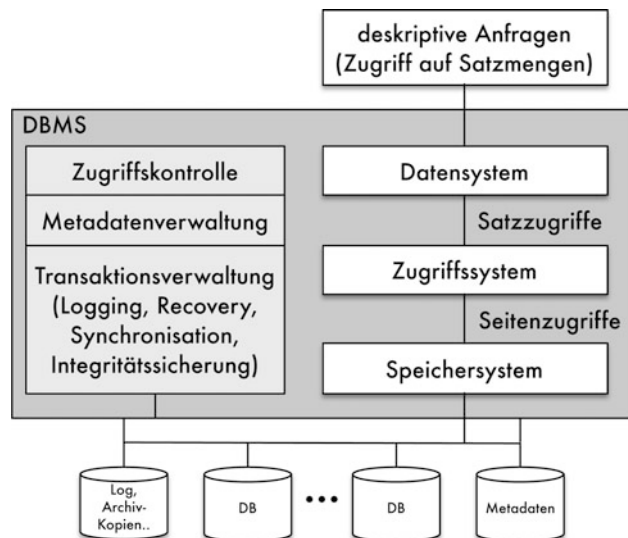
Die Schemaangaben repräsentieren Beschreibungsinformationen oder *Metadaten* über den Datenbankaufbau, die innerhalb eines *Katalogs* (Data Dictionary) geführt werden, ebenso wie benutzerspezifische Angaben (Zugriffsrechte etc.). Die Daten selbst werden physisch gemäß der Spezifikation des internen Schemas gespeichert. Beim Datenzugriff sind durch das DBMS gemäß der Schemaarchitektur entsprechende Abbildungen zwischen externer und interner Repräsentation vorzunehmen.

2.2.2 Funktioneller Aufbau relationaler Datenbanksysteme

Der interne Aufbau eines Datenbankverwaltungssystems kann durch mehrstufige *Schichtenmodelle* beschrieben werden [14]. Dabei werden die Funktionen des DBMS aufeinander aufbauenden Schichten zugeordnet, welche Operationen und Objekte der DBMS-Schnittstelle schrittweise auf interne Strukturen abbilden bis hinunter auf die Bitebene der Externspeicher. Abbildung 2.4 verdeutlicht im mittleren Teil diesen Abbildungsprozess anhand eines dreistufigen Schichtenmodells bestehend aus einem Daten-, Zugriffs- und Speichersystem.

Das *Datensystem* bildet in diesem Modell die oberste Schicht, welche die mengenorientierten DB-Operationen einer deskriptiven Anfragesprache auf Objekten eines externen Schemas zu verarbeiten hat. Diese Operationen werden entweder im Rahmen vordefinier-

Abb. 2.4 Aufbau eines zentralisierten Datenbanksystems



ter Anwendungsprogramme gestellt oder aber interaktiv als Ad-hoc-Anfragen abgesetzt. Aufgabe des Datensystems ist es, zu den DB-Operationen entsprechende Ausführungspläne zu erstellen (Query-Übersetzung und -Optimierung) und zu bearbeiten. Dabei werden u. a. externe Objektbezeichnungen auf interne Namen abgebildet und die Verwendung von Basisoperatoren wie Selektion (Scan), Projektion oder Join festgelegt. Für jeden der Basisoperatoren können mehrere Implementierungsalternativen bestehen, unter denen in Abhängigkeit der DB-Operation, vorhandener Indexstrukturen sowie anderer Faktoren eine Auswahl erfolgt. Die Ausführung der Operatoren erfolgt mit den satzorientierten Operationen des Zugriffssystems.

Das *Zugriffssystem* verwaltet die DB-Sätze innerhalb der Seiten und unterstützt entsprechende Satzoperationen. Daneben werden dort für den schnellen Zugriff Zugriffspfade und Indexstrukturen für ausgewählte Attribute geführt, meist auf Basis von B*-Bäumen. Clusterbildung (Clustering) von häufig zusammen referenzierten Sätzen ist ebenfalls eine wirkungsvolle Technik, um die Anzahl von Seiten- und damit Externspeicherzugriffen zu reduzieren. Alternativ bzw. in Ergänzung zur traditionellen Speicherung ganzer Sätze innerhalb der Seiten (Row-Store-Ansatz) unterstützen relationale DBS zunehmend eine spaltenweise Zerlegung und Speicherung von Sätzen im Rahmen sogenannter *Column Stores* [28, 1]. Damit werden die Attributwerte von Spalten benachbart in Seiten gespeichert. Damit sind für Anfragen auf einzelnen Attributen signifikant weniger Seiten zu verarbeiten, sodass sich die Ausführungszeiten im Vergleich zu Row Stores oft um ein Vielfaches beschleunigen lassen. Durch den Einsatz von Komprimierungstechniken auf den Attributwerten lassen sich diese Effekte weiter steigern [2].

Das *Speichersystem* schließlich ist für die Verwaltung von DB-Seiten zuständig, wobei insbesondere ein DB-Puffer (Systempuffer) im Hauptspeicher verwaltet wird, um Lokalität im Referenzverhalten zur Einsparung physischer E/A-Vorgänge zu nutzen. Die Zugriffe auf Externspeicher erfolgen üblicherweise über die Dateiverwaltung des Betriebssystems, welche eine Abstraktion von physischen Geräteeigenschaften bietet.

Als Externspeicher dienen traditionell Magnetplattenspeicher, die eine sehr hohe Speicherkapazität zu geringen Kosten anbieten, jedoch verbunden mit langsamen Zugriffszeiten von mehreren Millisekunden pro Block (Seite). Für leistungskritische Anwendungen werden daher zunehmend auch Flash-Speicher als *Solid-State-Disk (SSD)* eingesetzt, die im Vergleich zu Magnetplatten wesentlich schnellere Zugriffszeiten und E/A-Raten (bei rückläufigen relativen Zusatzkosten) ermöglichen [22, 11]. Ein weiterer Trend besteht in der zunehmenden Verwendung von *Hauptspeicher-* oder *In-Memory-Datenbanken*, um Externspeicherzugriffe für den Datenbankzugriff komplett auszuschließen [24, 19]. In diesem Fall sind die Dauerhaftigkeit der Daten und die Transaktionseigenschaften (Abschn. 2.3) durch eine Replikation der Daten in anderen Rechnern zu gewährleisten, was zu verteilten Systemarchitekturen führt. Daneben erhofft man sich künftig von neuen nicht-flüchtigen Halbleiterspeichertechnologien wie PCM (Phase Change Memory) signifikante Vorteile für den DBS-Einsatz [7]. Wir werden in diesem Buch jedoch meist die klassische und weiterhin dominierende Nutzung von Plattenspeichern zur dauerhaften Datenspeicherung voraussetzen.

Ergänzend zu den diskutierten Schichten umfasst ein DBMS Funktionen zur Zugriffskontrolle, zur Verwaltung von Metadaten (die in unterschiedlicher Ausprägung in allen Schichten benötigt werden) sowie zur Transaktionsverwaltung, um die ACID-Eigenschaften sicherzustellen.

2.3 Das Transaktionskonzept (ACID)

Die DB-Verarbeitung erfolgt in der Regel durch Transaktionen, welche aus einer oder mehreren Operationen bestehen. Bezüglich der Ausführung von Transaktionen garantieren Datenbanksysteme die Einhaltung des sogenannten *Transaktionskonzepts* [15]. Dies betrifft die automatische Gewährleistung der folgenden vier kennzeichnenden ACID-Eigenschaften (Atomicity, Consistency, Isolation, Durability) von Transaktionen:

1. *Atomarität* („*Alles oder nichts*“): Änderungen einer Transaktion werden entweder vollkommen oder gar nicht in die Datenbank eingebracht. Diese Eigenschaft ermöglicht eine erhebliche Vereinfachung der Anwendungsprogrammierung, da Fehlersituationen während der Programmausführung (z. B. Rechnerausfall) nicht im Programm abgefangen werden müssen. Das DBMS sorgt dafür, dass die Transaktion in einem solchen Fall vollständig zurückgesetzt wird, sodass keine ihrer Änderungen in der Datenbank verbleibt. Der Programmierer kann somit bei der Realisierung von Anwendungen von einer fehlerfreien Umgebung ausgehen.
2. *Konsistenz*: Die Transaktion ist die Einheit der Datenbankkonsistenz. Dies bedeutet, dass bei Beginn und nach Ende einer Transaktion sämtliche Integritätsbedingungen erfüllt sind.
3. *Isolation*: Datenbanksysteme unterstützen typischerweise eine große Anzahl von Benutzern, die gleichzeitig auf die Datenbank zugreifen können. Trotz dieses *Mehrbenutzerbetriebes* wird garantiert, dass dadurch keine unerwünschten Nebenwirkungen eintreten (z. B. gegenseitiges Überschreiben derselben Datenbankobjekte). Vielmehr bietet das DBMS jedem Benutzer bzw. Programm einen „logischen Einbenutzerbetrieb“, sodass parallele Datenbankzugriffe anderer Benutzer unsichtbar bleiben. Auch hierdurch ergibt sich eine erhebliche Vereinfachung der Programmierung.
4. *Dauerhaftigkeit*: Die Dauerhaftigkeit von erfolgreich beendeten Transaktionen wird garantiert. Dies bedeutet, dass Änderungen dieser Transaktionen alle erwarteten Fehler (insbesondere Rechnerausfälle, Externspeicherfehler und Nachrichtenverlust) überleben.

Die Konsistenzüberwachung (Eigenschaft 2) erfordert die Überprüfung aller anwendungsseitig definierten Integritätsbedingungen sowie der modellinhärenten Bedingungen (relationale Invarianten) bei der Ausführung von Änderungstransaktionen. Eigenschaften 1 und 4 werden vom Datenbanksystem durch geeignete *Logging- und Recovery*-Maßnahmen eingehalten. Nach einem Rechnerausfall wird insbesondere der jüngste transaktionskon-

sistente Datenbankzustand hergestellt. Dazu erfolgt ein Zurücksetzen aller Transaktionen, die aufgrund des Rechnerausfalles nicht zu Ende gekommen sind (Undo-Recovery); für erfolgreiche Transaktionen wird eine Redo-Recovery vorgenommen, um deren Änderungen in die Datenbank einzubringen (falls erforderlich).

Zur Gewährung der Isolation im Mehrbenutzerbetrieb (Eigenschaft 3) sind geeignete Verfahren zur *Synchronisation* (Concurrency Control) bereitzustellen. Das allgemein akzeptierte Korrektheitskriterium ist dabei die *Serialisierbarkeit*. Obwohl ein großes Spektrum von Synchronisationsverfahren zur Wahrung der Serialisierbarkeit vorgeschlagen wurde, verwenden existierende Datenbanksysteme vorrangig Sperrverfahren zur Synchronisation. Lese- und Schreibsperrern werden dabei üblicherweise bis zum Transaktionsende gehalten, um Serialisierbarkeit zu gewährleisten und Änderungen einer Transaktion nicht vor deren Commit (erfolgreichem Ende) anderen Transaktionen zugänglich zu machen (striktes Zwei-Phasen-Sperrprotokoll). Für eine hohe Leistungsfähigkeit ist es wesentlich, eine Synchronisation mit möglichst wenig Sperrkonflikten und Rücksetzungen zu erreichen. Dies verlangt die Unterstützung feiner Sperrgranulate (z. B. einzelne DB-Sätze) innerhalb eines hierarchischen Verfahrens sowie ggf. Spezialprotokolle für Datenobjekte wie Indexstrukturen [12].

Die Realisierung des Transaktionskonzepts für verteilte DMS wird im Teil IV dieses Buchs ausführlich erläutert. Dabei wird auch auf die Basistechniken aus zentralisierten DBS eingegangen, z. B. zur Synchronisation (Kap. 12).

2.4 DBS-Einsatz: OLTP vs. OLAP

Der operationale Betrieb mit lokalen Datenmodifikationen und die Analyse großer Datenbestände sind zwei verbreitete Szenarien für den Datenbankeinsatz, die stark unterschiedliche Anforderungen aufweisen. Wir werden beide Szenarien im Folgenden kurz skizzieren.

2.4.1 Online Transaction Processing

Traditionellerweise werden DBS intensiv zur effizienten Verarbeitung von Transaktionen im Rahmen sogenannter *OLTP-Anwendungen* (*Online Transaction Processing*) eingesetzt. Die Anwendungsfunktionen werden durch eine Reihe vordefinierter Anwendungsprogramme realisiert, welche auf die Datenbank für Lese- und Änderungsoperationen zugreifen. Typische OLTP-Anwendungen sind Onlinebanking oder Auskunfts- und Buchungssysteme, deren Dienste meist über Webformulare aufgerufen und mit aktuellen Parametern (z. B. Kontonummer für eine Kontostandsabfrage) versorgt werden. Dem Nutzer bleiben alle Implementierungsdetails der Anwendungsprogramme sowie der indirekt involvierten Datenbanken vollständig verborgen. Die Transaktionen sind meist sehr einfach und umfassen nur wenige Datenbankoperationen. Sie müssen für die interaktive

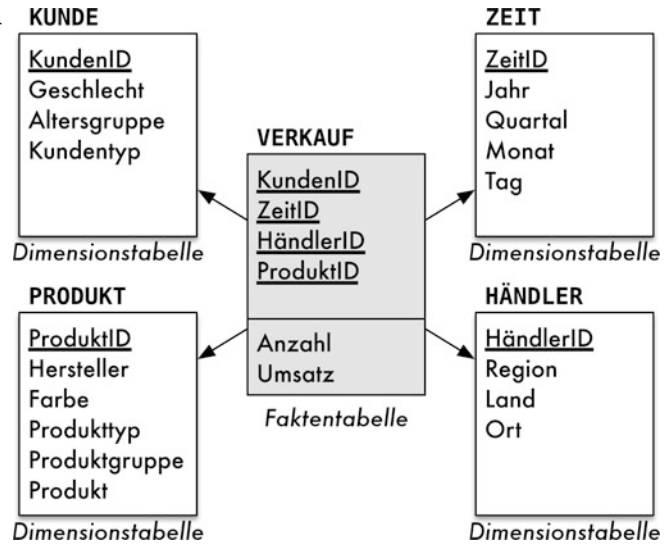
Ausführung in sehr kurzer Zeit (maximal einige Sekunden) bearbeitet werden. Zudem müssen sehr hohe Transaktionsraten unterstützt werden, um viele Nutzer gleichzeitig zu bedienen. Wesentlich für OLTP ist die Gewährleistung der ACID-Eigenschaften, um alle Datenbankänderungen sicher und korrekt durchzuführen.

OLTP-Anwendungen sind ein klassisches Einsatzgebiet etablierter relationaler DBMS, wie z. B. Oracle, IBM DB2, Microsoft SQL-Server oder MySQL. Jedoch wurde in den letzten Jahren eine Reihe von neueren relationalen DBMS mit Fokus auf OLTP-Anwendungen entwickelt, die auch als *NewSQL*-Systeme bezeichnet werden [13]. Sie versuchen durch den OLTP-Fokus u. a. die Leistungsfähigkeit signifikant zu erhöhen, z. B. um sehr viele Nutzer von Webanwendungen gleichzeitig zu unterstützen. Dies erfolgt zum Teil durch eine parallele und verteilte Transaktionsausführung, wie wir sie in diesem Buch behandeln werden. Zudem werden in Systemen wie *VoltDB* (sowie dem zugehörigen Forschungsprototypen *H-Store* [17]) und *MemSQL* In-Memory-Datenbanken zum weitgehenden Wegfall von I/O-Verzögerungen vorausgesetzt und darauf zugeschnittene Synchronisations- und Recovery-Mechanismen verwendet. Ein aktueller Einsatzschwerpunkt liegt in der schnellen Realzeitverarbeitung von Ereignissen und Sensordaten, wie Aktienkursen, Twitter-Nachrichten, Websiteinteraktionen u. Ä. [5]. Das System *Google Spanner* kann aufgrund seiner Unterstützung von relationenähnlichen Datenstrukturen, von SQL und ACID (s. Kap. 12 und 15.3) auch als ein NewSQL-System angesehen werden, das sich durch eine massive Skalierbarkeit auf viele Tausende Rechner auszeichnet [8].

2.4.2 OLAP und Data Warehouses

Neben OLTP-Transaktionen sind auf Datenbanken auch häufig Analysen durchzuführen, in denen zur Unterstützung von Unternehmensentscheidungen (Decision Support) große Datenmengen gelesen und ausgewertet werden. Diese Auswertungen sollten möglichst auch interaktiv in relativ kurzer Zeit ausführbar sein, wobei man hier zur Abgrenzung zu OLTP von *OLAP* (*Online Analytical Processing*) spricht. Die gleichzeitige Ausführung von OLAP-Analysen und OLTP-Transaktionen auf demselben Datenbestand kann jedoch zu gegenseitigen Behinderungen führen, die insbesondere die Einhaltung der OLTP-Leistungsziele bezüglich Antwortzeiten und Durchsatz gefährden. Dies ist einer der Gründe dafür, dass Analyseanwendungen häufig auf dedizierten *Data-Warehouse*-Datenbanken [3, 20] durchgeführt werden, die Daten aus den OLTP-Systemen redundant führen. Zudem werden in Data Warehouses meist Daten aus mehreren Datenquellen zusammengeführt und konsolidiert, um übergreifende Auswertungen zu ermöglichen (siehe Abschn. 3.5). Dabei werden die Daten für Analysezwecke gezielt strukturiert, um eine große Flexibilität und schnelle Berechnung von Auswertungen zu erreichen. Insbesondere gilt es für OLAP eine Vielzahl mehrdimensionaler Auswertungen sowie vielfältige Aggregationsmöglichkeiten von Daten zu unterstützen, z. B. um die Umsatz- oder Gewinnentwicklung eines Unternehmens für unterschiedliche Zeiträume, Regionen oder Produktarten auswerten und vergleichen zu können.

Abb. 2.5 Beispiel eines Data-Warehouse-Schemas (Star Schema)



Für Data Warehouses auf Basis von relationalen Datenbanken werden für diese mehrdimensionalen Auswertungen die Daten meist gemäß eines *Stern-Schemas* (*Star Schema*) oder einer seiner Varianten wie Schneeflocken- (Snowflake-) oder Galaxie-Schema strukturiert. Ein einfaches Beispiel für ein derartiges Star Schema zeigt Abb. 2.5. Dabei steht eine sogenannte Faktentabelle im Zentrum, welche alle Einzelsätze zu den auszuwertenden Kenngrößen enthält, im Beispiel jedes Verkaufsereignis mit der Anzahl verkaufter Produkte und dem damit erzielten Umsatz. Durch Fremdschlüsselverweise auf Dimensionstabellen werden die einzelnen Fakten semantisch beschrieben, im Beispiel bezüglich des Verkaufszeitpunkts, des kaufenden Kunden, des verkaufenden Händlers und des verkauften Produkts. Die Masse der Daten ist typischerweise in der Faktentabelle angesiedelt, da die meisten Dimensionstabellen bis auf Ausnahmen wie für Produkte oder Kunden eher klein sind.

Alle Attribute der Dimensionstabellen können für die Einschränkung der auszuwertenden Fakten herangezogen werden, wobei die Wertebereiche der Dimensionen oft hierarchisch strukturiert sind, um unterschiedliche Auswertungsgranularitäten zu unterstützen. So ist im Beispiel bezüglich der Zeitdimension die Auswertung auf der Ebene von Tagen, Monaten, Quartalen und Jahren möglich; Händler lassen sich nach Ort, Bundesland und Region eingrenzen und produktbezogene Auswertungen können für einzelne Produkte, Produktgruppen oder Produkttypen erfolgen. Die (mehrdimensionalen) Auswertungen resultieren in sogenannte *Star Joins*, in denen die Faktentabelle mit den Dimensionstabellen verknüpft werden und auf den selektierten Fakten in der Regel mehrere Gruppierungen und Aggregationen erfolgen. Der folgende Star Join gruppiert und aggregiert die Anzahl verkaufter Produkte und die erzielten Umsätze für die beiden Hersteller Apple und

Samsung nach Quartal und Produkttyp, um Vergleiche bezüglich der genannten drei Dimensionen zu unterstützen.

```
SELECT Quartal, Hersteller, Produkttyp, SUM(Anzahl), SUM(Umsatz)
FROM Verkauf V, Zeit Z, Produkt P
WHERE Hersteller IN ('Apple','Samsung') AND Jahr=2013 AND
      V.ProduktID=P.ProduktID AND V.ZID=Z.ZID {Joinbedingungen}
GROUP BY Quartal, Hersteller, Produkttyp
```

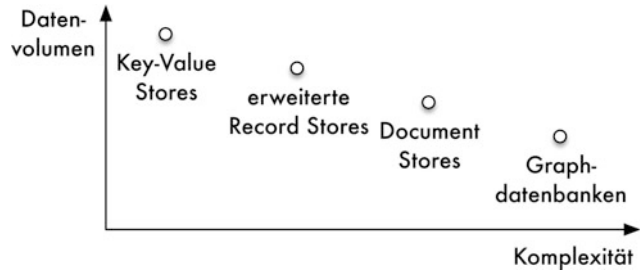
Die effiziente OLAP-Unterstützung stellt aufgrund der großen Datenmengen und der Vielzahl möglicher, komplexer Auswertungen hohe Leistungsanforderungen, die u. a. durch den Einsatz Paralleler DBS abgedeckt werden können. Daneben wurden Techniken wie die Nutzung von Bitlisten-Indizes und materialisierte Sichten mit vorberechneten Aggregationsergebnissen entwickelt und in zahlreichen DBMS unterstützt [3, 20]. Eine weitere signifikante Leistungssteigerung wird durch die Nutzung von In-Memory-Datenbanken und Column Stores ermöglicht [1]. Ähnlich wie mit den NewSQL-Systemen im OLTP-Bereich entstand auch im OLAP-Umfeld eine Vielzahl von neuen Systementwicklungen, die u. a. auf Technologien wie Column Stores und In-Memory-Verarbeitung setzen. Beispielhaft seien genannt *SAP HANA*, *Exasol*, *Greenplum* oder *HP Vertica*. Einige In-Memory-Ansätze wie SAP HANA oder der Prototyp *HyPer* versuchen sowohl OLTP- als auch OLAP-Anwendungen effizient zu unterstützen [24, 19].

2.5 NoSQL-Systeme

Datenbanksysteme fokussieren auf die effiziente und sichere Verarbeitung großer Mengen strukturierter Daten. Der Großteil erzeugter Daten, insbesondere im Web, in sozialen Netzen oder auf mobilen Geräten wie Smartphones, ist jedoch nur teilstrukturiert (E-Mail-Nachrichten, Webseiten, Benutzerprofile, Produktangebote etc.) oder unstrukturiert (z. B. Fotos, Videos). Die Verarbeitung solcher Daten (bzw. generell von Big Data) mit relationalen DBS hat sich vielfach als zu ineffizient und unflexibel erwiesen, was zur Entwicklung von sogenannten NoSQL-Systemen zum Datenmanagement geführt hat. Unter diese Bezeichnung wird eine Vielzahl von Systemen gerechnet, die meist durch folgende Eigenschaften gekennzeichnet sind:

- nichtrelationale Datenorganisation,
- schemafrei,
- verteilte Realisierung,
- hohe Skalierbarkeit auf große Datenmengen,
- replizierte Datenspeicherung für hohe Verfügbarkeit,
- kein SQL,

Abb. 2.6 Grobeinteilung von NoSQL-Systemen (nach [10])



- kein oder nur eingeschränktes ACID,
- Verfügbarkeit einer Open-Source-Version.

Durch die Abkehr vom relationalen Datenmodell und der damit verbundenen Notwendigkeit, den Aufbau einer Datenbank vorab in einem Schema zu definieren, ergibt sich eine hohe Flexibilität zur notwendigen Verwaltung vieler teil- und unstrukturierter Daten mit hoher Variabilität. Zur hohen Skalierbarkeit auf große Datenmengen bzw. eine sehr große Anzahl von Nutzern sind verteilte Lösungen erforderlich. Dafür werden Abstriche im Funktionsumfang in Kauf genommen, insbesondere bei den Transaktionseigenschaften sowie den Zugriffs- und Abfragemöglichkeiten (keine SQL-Unterstützung). Anstelle der ACID-Eigenschaften werden dabei teilweise die schwächeren BASE-Eigenschaften (Basically Available, Soft state, Eventually consistent) verfolgt (siehe Kap. 15).

Die meisten NoSQL-Systeme lassen sich einer der folgenden Gruppen zuordnen [4, 16, 13]:

- Key Value Stores oder
- erweiterbare Record Stores (Column Family Stores) oder
- Document Stores oder
- Graph-Datenbanksysteme.

Wie in Abb. 2.6 gezeigt, nimmt für die Systeme in der genannten Reihenfolge die Komplexität der Daten und Zugriffsoperationen zu, während die typischerweise unterstützten Datenvolumina abnehmen. Somit wird von Key Value Stores die größte Skalierbarkeit hinsichtlich des Datenumfangs unterstützt, jedoch für einfach strukturierte Daten und einfache Zugriffsoperationen. Umgekehrt sind Graph-Datenbanken für die komplexesten Daten und Operationen ausgelegt.

Key Value Stores sind daher von den genannten Kategorien am einfachsten bezüglich ihrer Datenstrukturierung und den unterstützten Operationen ausgelegt. Ein Datensatz wird dabei über einen systemweit eindeutigen Schlüssel (Key) identifiziert und besitzt daneben einen für das System meist nicht weiter interpretierbaren Zusatzwert, z. B. zur Speicherung von Zahlen, Strings, Arrays, Webseiten, Texten, Fotos. Somit wird eine schemafreie Speicherung und Verwaltung beliebiger Daten unterstützt. Als Operationen

werden nur einfache Lese- und Schreibzugriffe über den Schlüssel unterstützt; die Verarbeitung der den Schlüsseln zugeordneten Werte ist durch die Anwendungen zu realisieren. Die Verteilung auf viele Knoten und eine hohe Skalierbarkeit sind damit sehr einfach zu erreichen. Key Value Stores können noch danach unterschieden werden, ob sie die verteilte Speicherung der Daten im Hauptspeicher (In-Memory Key Value Stores) oder auf Externspeicher vornehmen (persistente Key Value Stores) [13]. Beispiele von In-Memory-Systemen sind Redis und Memcached, zu den Vertretern persistenter Key Value Stores zählen Voldemort und Riak.

Erweiterbare Record Stores gehen auf *Google BigTable* [6] zurück, welches eine flexible Tabellenstruktur mit sogenannten Spaltenfamilien (Column Families) verwendet. Es gibt dabei ein vordefiniertes Tabellenschema, jedoch lassen sich im Rahmen der Spaltenfamilien jederzeit neue Attribute ergänzen. Zudem können unterschiedliche Sätze verschiedene Spaltenfamilien aufweisen, wodurch eine hohe Flexibilität entsteht. Der Zugriff erfolgt wiederum primär über den eindeutigen Key der Sätze. Als Nachbildung von BigTable ist *Apache HBase* ein bekannter Vertreter dieser Kategorie, der Teil des Big-Data-Software-Ökosystems *Hadoop* ist. Verwandte Implementierungen sind *Cassandra* und *DynamoDB*, wenngleich sie eine abgewandelte Tabellenstruktur verwenden.

Document Stores wie *MongoDB*, *CouchDB* oder *MarkLogic* fokussieren auf die schemalose und skalierbare Verwaltung teilstrukturierter Dokumente, z. B. im JSON-Format (JavaScript Object Notation). Damit können Daten für Anwendungen im Web oder in sozialen Netzwerken flexibel verwaltet werden, z. B. Benutzer mit ihren Fotos, Nachrichten. Jedes Dokument hat einen eindeutigen Schlüssel, kann jedoch ansonsten unterschiedlich strukturiert sein. Für den Zugriff auf die Dokumente werden einfache, mengenorientierte Anfragesprachen bereitgestellt, worüber Bedingungen auf den Attributen der Dokumente ausgewertet werden können. Attribute können zur beschleunigten Auswertung von Anfragen indexiert werden.

Graph-Datenbanksysteme wie *Neo4J* oder *HypergraphDB* verwalten die Datenobjekte und die Beziehungen zwischen ihnen gemäß eines Graph-Datenmodells [27]. Besonders populär ist das auch in Neo4J genutzte *Property-Graph-Datenmodell*, bei dem Knoten und Kanten beliebig viele Attribute (Properties) aufweisen können. Graph-DBS eignen sich besonders zur Verwaltung und Auswertung komplexer Beziehungen und Abhängigkeiten, z. B. in sozialen Netzwerken oder für wissenschaftliche Anwendungen mit großen Netzstrukturen wie Proteinnetzwerke. Die Systeme unterstützen dafür i. Allg. komplexe Anfragesprachen, die zum Teil über die Fähigkeiten von SQL hinausgehen. Allerdings stehen hier die Skalierbarkeit und Verteilung oft nicht im Vordergrund. Wir werden diese Systeme daher in diesem Buch auch nicht weiter betrachten.

Die bereits im vorigen Abschnitt diskutierten NewSQL-Systeme verstehen sich zum Teil als Alternative zu den NoSQL-Systemen, da sie ebenfalls eine hohe Skalierbarkeit und Verfügbarkeit anstreben, jedoch unter Beibehaltung von SQL und den ACID-Zusicherungen. Aufgrund des Festhaltens am Relationenmodell bleiben jedoch die Einschränkungen hinsichtlich der Flexibilität und Unterstützung teilstrukturierter Daten.

2.6 Rechnernetze und verteilte Systeme

Ein *Rechnernetz* verbindet mehrere Rechner miteinander und erlaubt den Austausch von Informationen (Nachrichten) zwischen ihnen. Wir diskutieren zunächst unterschiedliche Typen von Rechnernetzen und gehen danach auf Kommunikationsprotokolle ein. Für eine detaillierte Behandlung von Rechnernetzen und Kommunikationsprotokollen muss auf die reichlich vorhandene Literatur verwiesen werden, z. B. [29, 21].

2.6.1 Typen von Rechnernetzen

Rechnernetze lassen sich nach vielerlei Gesichtspunkten klassifizieren, z. B. nach dem verwendeten Übertragungsmedium (Kupferkabel, Glasfaserkabel oder drahtlose Übertragung in WLAN- und Sensornetzen oder über Satelliten), der Netzwerktopologie (Ring, Bus, Sternstruktur, vollvermaschtes Netz etc.), Punkt-zu-Punkt- vs. Broadcast-Verbindung, Leitungs- vs. Paketvermittlung oder nach räumlicher Entfernung [29]. Für unsere Zwecke ist vor allem die Einteilung nach räumlicher Entfernung von Interesse, wobei man grob drei Kategorien unterscheiden kann:

- *Cluster*,
- *Lokales Netzwerk* (Local Area Network, LAN),
- *Weitverkehrsnetz* (Wide Area Network, WAN).

Ein *Cluster* besteht aus mehreren räumlich benachbarten Rechnern, die typischerweise nur wenige Meter voneinander getrennt sind. Die Kommunikation erfolgt über ein Hochgeschwindigkeitsnetz mit hoher Übertragungsbandbreite und kurzen Nachrichtenlaufzeiten. Lokale Netze erlauben größere Entfernungen zwischen den Rechnern bis zu wenigen Kilometern. Darüber hinaus spricht man von Weitverkehrsnetzen, die auch mehrere Kontinente umfassen können. Die Kommunikation im WAN-Bereich erfolgt primär über das öffentliche Internet. Alternativ kann auf langsamere Telefonnetzwerke oder privat betriebene Verbindungen zurückgegriffen werden.

Die genannten Netzwerktypen weisen erhebliche Unterschiede bezüglich Leistungs- und Verfügbarkeitsmerkmalen auf. Grob gesagt nehmen sowohl Leistungsfähigkeit als auch Fehlertoleranz (gegenüber Leitungsausfall, Nachrichtenverlust, Nachrichtenduplizierung etc.) mit zunehmender geografischer Verteilung stark ab. Während innerhalb eines Clusters derzeit Übertragungsbandbreiten von ca. 10–100 Gbit/s erreicht werden, operieren typische LANs auf Ethernet-Basis zwischen 100–1000 Mbit/s. Die Übertragungsgeschwindigkeit von Weitverkehrsnetzen lag um 1990 noch meist zwischen 10 und 100 Kbit/s, hat sich jedoch im Internetkontext deutlich auf Werte im Bereich von 100 Mbit/s gesteigert. Die Übertragungsdauer im Weitverkehrsbereich ist jedoch nicht nur durch die Bandbreite beeinflusst, sondern auch durch die aufgrund der großen geografischen Entfernungen beträchtlichen Signallaufzeiten, die letztlich durch die Licht-

Tab. 2.1 Typische Leistungsmerkmale von Rechnernetzen (1990 vs. 2010)

	Durchmesser (km)	Latenzzeit (ms)	Bandbreite (Mbit/s)		Übertragung 10 KB (ms)	
			1990	2010	1990	2010
Cluster	0,02	0,001	1000	10.000	0,1	0,01
LAN	1	0,01	10	1000	10	0,1
WAN	10.000	100	0,05	100	1700	101

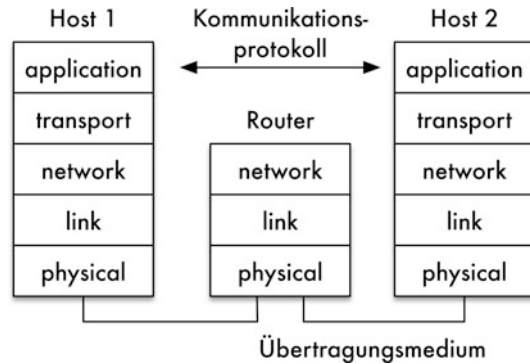
geschwindigkeit begrenzt sind. Die Übertragungsdauer einer Nachricht ist im Internet zudem noch stark von lastbedingten Verzögerungen bei der Weiterleitung der Nachrichten über Vermittlungsknoten beeinflusst. Hinzu kommt der CPU-seitige Aufwand zur Abwicklung des Kommunikationsprotokolls.

Zur Verdeutlichung der Leistungsunterschiede sind in der Tab. 2.1 Schätzwerte für die Latenzzeit (Signallaufzeit), Bandbreite sowie Übertragungsdauer einer Nachricht von 10 KB für die Jahre 1990 und 2010 zusammengestellt. Die Übertragungsdauer enthält dabei die durch die geografische Entfernung bestimmte Signallaufzeit sowie die durch die Bandbreite begrenzte Übertragungszeit als auch CPU-Belegungszeiten zur Abwicklung des Kommunikationsprotokolls. Man erkennt, dass die Übertragungsbandbreite im Weitverkehrsbereich sich besonders gesteigert hat und damit keinen leistungsbegrenzenden Faktor mehr darstellt. Dennoch bleiben Weitverkehrsnetze signifikant langsamer als lokale Netze, da die Übertragungszeiten zunehmend durch die Signallaufzeiten (Lichtgeschwindigkeit) dominiert werden.

2.6.2 Kommunikationsprotokolle

Der Nachrichtenaustausch zwischen zwei Kommunikationspartnern erfolgt nach festgelegten Kommunikationsprotokollen, welche regeln, wie die einzelnen Nachrichten aufgebaut sind und in welcher Reihenfolge sie verschickt werden können. Üblicherweise kommt eine Hierarchie von Protokollen zum Einsatz, die innerhalb einer *Kommunikationsarchitektur* angeordnet sind. Solche Architekturen wurden sowohl herstellerneutral (ISO OSI, TCP/IP etc.) als auch von einzelnen Herstellern für ihre Plattformen definiert. Den Kommunikationsarchitekturen ist gemein, dass sie als Schichtenmodelle realisiert sind. Die einzelnen Kommunikationsprotokolle regeln dabei jeweils die Kommunikation zwischen Instanzen derselben Schicht. Zur Realisierung eines Protokolls der Schicht $i + 1$ werden die Dienste der Schicht i genutzt, ohne die Realisierungseinzelheiten der darunterliegenden Schichten kennen zu müssen.

Die in der Praxis für LANs und WANs bedeutsamste Kommunikationsarchitektur basiert auf dem Internetprotokoll *TCP/IP* und umfasst inklusive einer Anwendungsschicht die fünf in Abb. 2.7 gezeigten Schichten. In Vermittlungsknoten wie Switches und Router sind lediglich die unteren zwei bzw. drei Schichten repräsentiert. Die Kommunikation verläuft bei dem in der Transportschicht realisierten TCP-Protokoll „verbindungsorientiert“,

Abb. 2.7 Schichten der Internetprotokollarchitektur

d. h., es sind zunächst logische Verbindungen zwischen den Kommunikationspartnern aufzubauen, über die dann die Nachrichten ausgetauscht werden. Nach Beendigung der Datenübertragung werden die Verbindungen wieder abgebaut. Für „verbindungslose“ Protokolle wie UCP mit Kommunikation über sogenannte Datagramme entfällt der Aufwand der Verbindungsverwaltung, dafür besteht eine reduzierte Fehlertoleranz z. B. hinsichtlich Nachrichtenverlust oder Überholvorgängen. Auf der Anwendungsebene sind zahlreiche Protokolle angesiedelt, insbesondere zur Unterstützung von klassischen Internetdiensten wie Web (HTTP), E-Mail (SMTP, IMAP, POP3), Adressauflösung (DNS) oder Dateitransfers (FTP). Auf eine genauere Beschreibung der einzelnen Schichten soll hier verzichtet werden. Wir verweisen dazu auf Lehrbücher wie [29, 21].

2.6.3 Kommunikation in Clustern und Datencentern

Die Kommunikationstechnologien für LAN können im Prinzip auch zur Kommunikation zwischen Rechnern in Clustern und Datencentern eingesetzt werden, z. B. zur Realisierung von Parallelen DBS vom Typ Shared Nothing. Der Vorteil liegt dabei in der Nutzung einer preiswerten Standardtechnologie, z. B. auf Ethernet-Basis. Dabei werden die einzelnen Rechner üblicherweise über einen Switch bzw. bei größerer Knotenzahl eine Hierarchie von Switches miteinander gekoppelt. Abbildung 2.8 [21] zeigt dazu eine mögliche Realisierung für den Einsatz in einem Datacenter. Dabei besteht das Datacenter aus einer größeren Zahl von Racks mit typischerweise jeweils 20–40 modular austauschbaren Knoten („blades“), bestehend aus Prozessoren, lokalem Hauptspeicher und Festplattenspeicher. Die Rechner eines Racks sind direkt mit einem „Top-of-Rack“- (TOR-) Switch gekoppelt, über den innerhalb eines Racks schnell kommuniziert werden kann. Die TOR-Switches sind untereinander über weitere Switches gekoppelt und letztlich über Router mit dem Internet und externen Nutzern verbunden. Die Kommunikation zwischen Rechnern verschiedener Racks ist damit i. Allg. deutlich langsamer als innerhalb eines Racks, da weitere Switches an der Kommunikation beteiligt sind und es zu gegenseitigen Behinderungen auf den gemeinsam genutzten Switch-Verbindungen kommen kann. Aus diesem

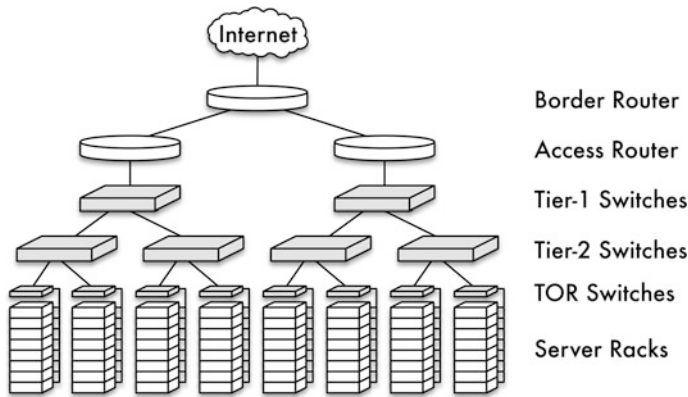


Abb. 2.8 Kommunikationsarchitektur in einem Datacenter (nach [21])

Grund kommen verstärkt leistungsfähigere Verbindungsnetzwerke für Datacenter zum Einsatz, u. a. mit zusätzlichen Verbindungen zwischen den Switches [21].

Im Hochleistungsbereich werden auch für kleinere Cluster, z. B. zur Realisierung von Shared-Disk-Systemen, häufig schnellere Kommunikationsnetzwerke bzw. Spezialprotokolle eingesetzt. So wird der Zugriff von Servern auf gemeinsam genutzte Speichernetzwerke (Storage Area Networks, SANs) meist über Glasfaserverbindungen und Protokolle wie Fiber Channel (FC) mit Übertragungsraten von mehreren Gbit/s realisiert [30]. Ein Vorteil dabei ist, dass die Externspeicher bis zu mehreren Kilometern von den Rechnern entfernt untergebracht werden können, was die Fehlertoleranz des Gesamtsystems erhöht. Zudem werden oft serielle Hochleistungsnetzwerke wie Infiniband-Switches oder herstellerspezifische Technologien (z. B. von Cluster-Herstellern wie HP oder Oracle/Sun) mit hohen Gbit/s-Übertragungsraten unterstützt. Ein weiterer Trend liegt in der Nutzung spezieller Kommunikationsprotokolle, die den Overhead im Betriebssystem und von vielen Schichten eines Protokoll-Stacks umgehen. So wird mit Technologien wie *Remote Direct Memory Access* (RDMA) ein schneller Nachrichten- und Datenaustausch unterstützt, bei dem direkt auf vordefinierte Speicherbereiche anderer Knoten zugegriffen werden kann. Gemäß [23] wird damit eine Datentransferzeit (round trip latency) von 3 Mikrosekunden bei im Vergleich zur Nachrichtenkommunikation wesentlich reduziertem Verarbeitungs-Overhead erreicht.

2.7 Übungsaufgaben

Übung 2.1 (No versus New)

Stellen Sie die Anforderungen an NoSQL denen von NewSQL gegenüber. Geben Sie für beide Systemklassen je eine Anwendungsklasse an, die besonders gut zu diesen Anforderungen passt, und diskutieren Sie, welche Nachteile die Nutzung der jeweils anderen Systemvariante hätte.

Literatur

1. Abadi, D., Boncz, P.A., Harizopoulos, S., Idreos, S., Madden, S.: The design and implementation of modern column-oriented database systems. *Foundations and Trends in Databases* **5**(3), 197–280 (2013)
2. Abadi, D.J., Madden, S., Ferreira, M.: Integrating compression and execution in column-oriented database systems. In: *Proc. ACM SIGMOD Conf.*, S. 671–682 (2006)
3. Bauer, A., Günzel, H. (eds.): *Data-Warehouse-Systeme: Architektur, Entwicklung, Anwendung*. 4. Auflage, dpunkt (2013)
4. Cattell, R.: Scalable SQL and nosql data stores. *SIGMOD Record* **39**(4), 12–27 (2010)
5. Cetintemel, U., Du, J., Kraska, T., Madden, S., Maier, D., Meehan, J., Pavlo, A., Stonebraker, M., Sutherland, E., Tatbul, N., et al.: S-Store: A streaming NewSQL system for big velocity applications. *Proc.s of the VLDB Endowment (PVLDB)* **7**(13) (2014)
6. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)* **26**(2), 4 (2008)
7. Chen, S., Gibbons, P.B., Nath, S.: Rethinking database algorithms for phase change memory. In: *Proc. CIDR*, S. 21–31 (2011)
8. Corbett, J.C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J.J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., Hsieh, W., Kanthak, S., Kogan, E., Li, H., Lloyd, A., Melnik, S., Mwaura, D., Nagle, D., Quinlan, S., Rao, R., Rolig, L., Saito, Y., Szymaniak, M., Taylor, C., Wang, R., Woodford, D.: Spanner: Google’s globally distributed database. *ACM Trans. Comput. Syst.* **31**(3), 8:1–8:22 (2013)
9. DeWitt, D.J., Gray, J.: Parallel database systems: The future of high performance database systems. *Commun. ACM* **35**(6), 85–98 (1992)
10. Eifrem, E.: Nosql: scaling to size and scaling to complexity. *Neotechnology Blog* (2009)
11. Graefe, G., Harizopoulos, S., Kuno, H.A., Shah, M.A., Tsirogiannis, D., Wiener, J.L.: Designing database operators for flash-enabled memory hierarchies. *IEEE Data Eng. Bull.* **33**(4), 21–27 (2010)
12. Gray, J., Reuter, A.: *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann (1993)
13. Grolinger, K., Higashino, W.A., Tiwari, A., Capretz, M.A.: Data management in cloud environments: Nosql and newsql data stores. *Journal of Cloud Computing: Advances, Systems and Applications* **2**(1), 22 (2013)
14. Härder, T., Rahm, E.: *Datenbanksysteme – Konzepte und Techniken der Implementierung*. 2. Auflage, Springer-Verlag (2001)
15. Härder, T., Reuter, A.: Principles of Transaction-Oriented Database Recovery. *ACM Comput. Surv.* **15**(4), 287–317 (1983)
16. Hecht, R., Jablonski, S.: Nosql evaluation. In: *International Conference on Cloud and Service Computing* (2011)
17. Kallman, R., Kimura, H., Natkins, J., Pavlo, A., Rasin, A., Zdonik, S., Jones, E.P., Madden, S., Stonebraker, M., Zhang, Y., et al.: H-store: a high-performance, distributed main memory transaction processing system. *Proc. of the VLDB Endowment (PVLDB)* **1**(2), 1496–1499 (2008)

18. Kemper, A., Eickler, A.: Datenbanksysteme – Eine Einführung. 8. Auflage, Oldenbourg Wissenschaftsverlag (2011)
19. Kemper, A., Neumann, T.: Hyper: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In: Proc. Int. Conf. Data Engineering (ICDE), S. 195–206 (2011)
20. Köppen, V., Saake, G., Sattler, K.U.: Data Warehouse Technologien. 2. Auflage, MITP (2014)
21. Kurose, J., Ross, K.: Computer Networking (6. Auflage). Pearson (2012)
22. Lee, S.W., Moon, B., Park, C., Kim, J.M., Kim, S.W.: A case for flash memory ssd in enterprise database applications. In: Proc. ACM SIGMOD Conf., S. 1075–1086 (2008)
23. Mitchell, C., Geng, Y., Li, J.: Using one-sided rdma reads to build a fast, cpu-efficient key-value store. In: USENIX Annual Technical Conference, S. 103–114 (2013)
24. Plattner, H., Zeier, A.: In-memory data management: an inflection point for enterprise applications. Springer (2011)
25. Saake, G., Sattler, K., Heuer, A.: Datenbanken: Implementierungstechniken. 3. Auflage, mitp (2011)
26. Saake, G., Sattler, K., Heuer, A.: Datenbanken – Konzepte und Sprachen. 5. Auflage, mitp (2013)
27. Sakr, S., Pardede, E.: Graph Data Management: Techniques and Applications. Information Science Reference (2011)
28. Stonebraker, M., Abadi, D.J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O’Neil, E.J., O’Neil, P.E., Rasin, A., Tran, N., Zdonik, S.B.: C-store: A column-oriented DBMS. In: Proc. 31st Int. Conf. on Very Large Data Bases (VLDB), S. 553–564 (2005)
29. Tanenbaum, A., Wetherall, D.: Computer Networks (5. Auflage). Pearson (2011)
30. Tate, J., Beck, P., Ibarra, H.H., Kumaravel, S., Miklas, L.: Introduction to Storage Area Networks and System Networking. IBM Red Book (2012)

Verteiltes und Paralleles Datenmanagement
Von verteilten Datenbanken zu Big Data und Cloud
Rahm, E.; Saake, G.; Sattler, K.-U.
2015, XIII, 379 S. 149 Abb., Softcover
ISBN: 978-3-642-45241-3