1 Der SQL Server 2017 stellt sich vor

Der SQL Server 2017 ist da – wieder eine tolle neue Version. Auch wenn diese Version sehr rasch nach dem SQL Server 2016 erschienen ist, der als Hauptrelease bezeichnet wird, ist der SQL Server 2017 als vermeintliche Zwischenversion alles andere als unbedeutend. Während der Entwicklung als SQL Server vNext bezeichnet, ist es doch erstmals in der Geschichte des MS SQL Server soweit, dass dieser neben Windows auch für Linux als Betriebssystem verfügbar ist. Und das ist eine echte Revolution. Da diese Versionen so rasch aufeinanderfolgen, gehe ich in dieser Auflage auch auf Features ein, die mit dem SQL Server 2016 eingeführt worden sind.

Ich bin von den Neuerungen der Versionen 2016 und 2017 begeistert und ertappe mich immer wieder, wenn ich bei einem Kunden noch eine der Vorversionen vorfinde, bei dem Gedanken: „Oje, jetzt muss ich wieder auf dieses und jenes verzichten." Mein persönliches Highlight ist die stark verbesserte Möglichkeit, Tabellen einer Datenbank im Arbeitsspeicher zu halten. Sehr spannend finde ich auch die neuen Sicherheitsfunktionen. Wird die Version 2016 von Microsoft selbst doch als das „Performance und Sicherheits-Release" bezeichnet. Und nun ist dies alles mit dem SQL Server 2017 auch für Linux verfügbar. Viele vermeintliche kleinere Features sind aber groß in der Auswirkung, so ist für Entwickler die native Unterstützung von JSON von großer Bedeutung.

Ich hoffe, auch Sie gehen mit Freude an den SQL Server 2017 und an dieses Buch heran!

Im ersten Kapitel möchte ich Ihnen einen Überblick über das Produkt und seine Komponenten geben. Anschließend stelle ich Ihnen die Editionen vor, in denen der SQL Server 2017 verfügbar ist, und zeige Ihnen, wie Sie bei der Installation vorgehen. Darüber hinaus werden Sie erfahren, wie Sie mit dem SQL Server 2017 arbeiten, um vorhandene Datenbanken und darin enthaltene Datenbankobjekte zu nutzen. Ebenso zeige ich Ihnen, wie eine Integration zu Client-Umgebungen erfolgen kann. Den Abschluss dieses Kapitels bilden die Besonderheiten der freien Express-Version.

1.1 SQL Server – wer ist das?

Eigentlich wollte ich diesen Abschnitt ursprünglich mit „SQL Server – was ist das?“ betiteln. Aber das kam mir dann so plump vor, dass ich das „was“ durch ein „wer“ ersetzt habe. Dies klingt besser, auch wenn ich den SQL Server dadurch nicht personifizieren will.

1.1.1 Der SQL Server im Konzert der Datenbanksysteme

Wenn wir heutzutage von einer Datenbank sprechen, meinen wir in der Regel – ohne dies explizit zu erwähnen – eine relationale Datenbank. Andere Datenbanksysteme, wie zum Beispiel objektorientierte Datenbanken, konnten sich nie wirklich auf breiter Front durchsetzen oder haben ihre beste Zeit bereits hinter sich. Neue moderne Ansätze, die sich unter dem Begriff NoSQL finden, sind für sehr spezielle Anwendungsbereiche ausgerichtet und zielen darauf ab, relationale Datenbanken zu verdrängen oder gar zu ersetzen. Vielmehr wollen sie eine Ergänzung in Nischenbereichen sein, für die relationale Strukturen nicht die ideale Form sind. Daher steht das *No* auch nicht für *kein*, sondern für *not only*. Vielfach kommen diese auch aufs Tapet, wenn von Big Data die Rede ist. Microsoft trägt dem durch die Möglichkeit von hybriden Lösungen mit Hadoop oder R Server Rechnung.

HINWEIS: Relationale Datenbanksysteme entwickeln sich in die Richtung weiter, dass wichtige Features aus dem nicht-relationalen Bereich als Zusätze in die Produkte integriert werden. So bietet der SQL Server 2017 als neues Feature erstmals die Unterstützung von Graph-Daten, eine wichtige Datenart bei NoSQL-Systemen.

Beim „schnellen Einstieg in den SQL Server“ gehen wir von relationalen Datenbanksystemen aus und unterteilen diese in

- *Desktop-Datenbanksysteme* und
- *Server-Datenbanksysteme*.

Eine Datenbankanwendung besteht aus drei Komponenten:

- *Data Layer:* Der Data Layer hat die Aufgabe, Daten zu verwalten und zu speichern. Hier werden außerdem die Strukturen der Datenspeicherung definiert. Diese Aufgabe wird von der Datenbank-Engine wahrgenommen.
- *Program Layer:* Im Program Layer werden die Logiken und Abläufe des Datenzugriffs abgebildet. Hier kommen unterschiedliche Entwicklungsumgebungen zum Einsatz.
- *Presentation Layer:* Aufgabe des Presentation Layers ist es, Ausgaben aus der Datenbank darzustellen. Hierzu gehören insbesondere Benutzeroberflächen und Frontend-Komponenten, mit denen der Benutzer interagiert.

Das Hauptmerkmal eines Desktop-Datenbanksystems besteht darin, dass alle drei Komponenten auf dem Desktop anzutreffen sind. Insbesondere läuft auch die Datenbank-Engine auf dem Desktop. Werden Datenbanken eines desktopbasierten Systems auf dem Server

abgelegt, wird vom Server lediglich der File-Service genutzt, um die Daten remote zur Verfügung zu stellen.

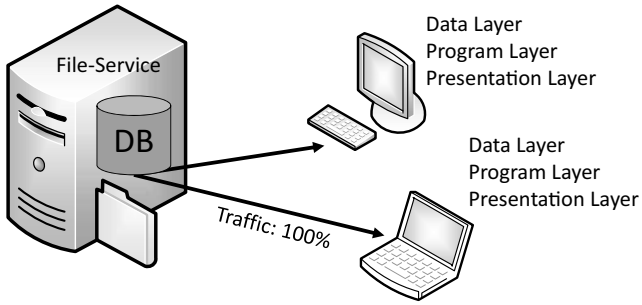

Bild 1.1 Konzept von Desktop-Datenbanken

Ein wesentliches Merkmal eines desktopbasierten Datenbanksystems ist, dass alle datenbankrelevanten Vorgänge auf dem Client ablaufen. Dazu müssen alle Daten vom Server auf den Client transferiert werden, damit die Daten von der lokalen Datenbank-Engine verarbeitet werden können.

Server-Datenbanksysteme hingegen verwenden eine Datenbank-Engine auf dem Server. Von den Clients werden Anfragen an diesen Dienst gestellt, die auf dem Server verarbeitet werden. Dadurch werden nicht alle Rohdaten, sondern nur die Ergebnisse der Anfrage an den Client gesendet. Es findet sozusagen eine Spezialisierung der Aufgaben der Datenverwaltung auf dem Server statt.

In der Abbildung ist der *Program Layer* beiden Komponenten zugeordnet, da Elemente von diesem auch in beiden Komponenten auftreten können. Wir werden später in diesem Buch zwischen serverseitiger und clientseitiger Datenbankprogrammierung unterscheiden.

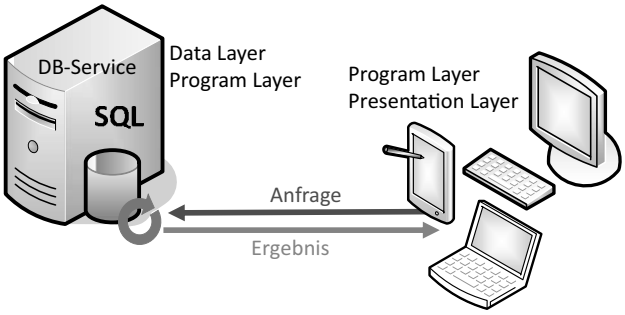

Bild 1.2 Konzept von Server-Datenbanksystemen

In der Kategorie der Desktop-Datenbanksysteme ist vor allem Microsoft Access weit verbreitet. Der SQL Server ist auch als Desktop-Datenbanksystem mit der *LocalDB* genannten Edition vertreten. Diese kann in lokal installierte Anwendungen integriert und weitergegeben werden und ist daher vor allem bei Visual Studio-Entwicklern bekannt. Ebenso in diese Kategorie einzuordnen ist SQLite, die als Embedded-DB ebenso in unzählige Anwendungen lokal integriert ist.

In der Kategorie der Server-Datenbanksysteme sind neben dem Microsoft SQL Server vor allem folgende Produkte von Bedeutung:

- Oracle
- DB2 von IBM
- SAP ASE (früher Adaptive Server Enterprise von Sybase, noch früher Sybase SQL Server)

Als Open-Source-Datenbanksysteme sind zusätzlich von Bedeutung:

- PostgreSQL
- MySQL/MariaDB

Der SQL Server ist das führende serverbasierte Datenbanksystem auf Windows-Plattformen. Bisher sind ja nur die anderen genannten Systeme auch für diverse andere Plattformen verfügbar gewesen.

HINWEIS: Der SQL Server 2017 ist erstmals für Linux verfügbar. Dies läutet einerseits eine neue Ära für den SQL Server ein, andererseits ist das die logische Konsequenz der generell von Microsoft vollzogenen Öffnung der letzten Jahre zu anderen Systemen. Wie weit der SQL Server auch auf diesen Plattformen eine bedeutende Stellung erhalten wird, wird die Zukunft zeigen.

Informationen über den SQL Server 2017 für Linux finden Sie in Kapitel 12.

ACID – das Konsistenzmodell relationaler Datenbanken

Relationale Datenbanken verwenden das Konsistenzmodell ACID. Bei diesem Modell steht die Datenkonsistenz absolut im Vordergrund und ist somit die oberste Maxime. Wenn wir uns die vier Säulen dieses Modells ansehen, werden wir feststellen, dass die Forderungen dieses Modells bei relationalen Desktop-Datenbanken wie Microsoft Access allerdings nicht erfüllt sind. Bei serverbasierten Datenbanken wie dem Microsoft SQL Server sind sie natürlich erfüllt. Die vier Säulen dieses Konsistenzmodells zeigt Bild 1.3.

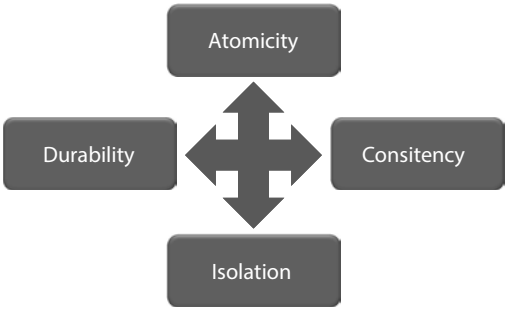

Bild 1.3 Das ACID-Konsistenzmodell

Was bedeuten diese Begriffe im Einzelnen und durch welche Mechanismen werden sie umgesetzt?

- **A – Atomicity:** Zusammenhängende Vorgänge werden entweder zur Gänze oder gar nicht durchgeführt. Gehören mehrere Schreibzugriffe zu einem gemeinsamen Vorgang, werden alle Änderungen erst übernommen, wenn auch der letzte Teilschritt erfolgreich abgeschlossen worden ist. Ist dies aus welchem Grund auch immer nicht möglich, müssen alle bisher vorgenommenen Schritte vollständig wieder rückgängig gemacht werden. Das Werkzeug, um diese Vorgabe zu erreichen, sind *Transaktionen*.
- **C – Consistency:** Die Vorgabe der Konsistenzerhaltung legt fest, dass der Übergang von einem konsistenten Zustand immer nur in einen anderen konsistenten Zustand erfolgen darf. Daten müssen also immer in einem vollständigen Zustand vorliegen, es darf nie Verweise auf nicht vorhandene Daten geben. Die *Referenzielle Integrität* sorgt dafür, dass dieses Ziel erreicht wird.
- **I – Isolation:** Die Forderung der Isolation besagt, dass alle Vorgänge von anderen unbeeinflusst abgegrenzt ablaufen dürfen. Die gleichen Daten können nie zeitgleich von mehreren Personen oder Prozessen geändert werden. Solange Änderungen nicht abgeschlossen sind, sind die betroffenen Daten zumindest für den Schreibzugriff für andere gesperrt. Die Änderungen sind für den Durchführenden sofort sichtbar, für alle anderen erst nach Abschluss des Vorgangs. Auch dafür sind *Transaktionen* zuständig.
- **D – Durability:** Unter der Dauerhaftigkeit versteht man, dass Daten, die einmal festgeschrieben worden sind, dauerhaft verfügbar sind und auch Strom- und andere Systemausfälle überstehen. Dieses Ziel kann durch den Einsatz von *Protokollierung* erfolgen.

Die erwähnten Mechanismen Transaktion, referenzielle Integrität und Protokollierung werden Sie in den entsprechenden Kapiteln dieses Buches im Detail erläutert finden. Besonders interessant ist ACID im Zusammenhang mit den speicheroptimierten Tabellen, die im RAM des Servers gehalten werden. Auf den ersten Blick würde man vermuten, dass diese vor allem im Hinblick auf die Dauerhaftigkeit problematisch sind. Allerdings werden Sie lesen, dass diese Tabellen über eine entsprechende Option bei der Erstellung auch ACID-konform eingesetzt werden können. Dies wird durch das zusätzliche Ablegen der Daten auf den Disks erzielt.

1.1.2 Entscheidungsszenarien für Datenbanksysteme

Wenn Sie vor der Entscheidung stehen, ein Datenbanksystem auszuwählen, gilt es, verschiedene Gesichtspunkte zu berücksichtigen. Ich möchte Ihnen in einem kurzen Überblick die aus meiner Sicht wichtigsten Entscheidungsgründe nennen.

- *Preis (TCO)*: Bei der Betrachtung der Kosten werden häufig fälschlicherweise lediglich die direkten Lizenzkosten angesetzt. Wesentlich zielführender wäre es allerdings, den Ansatz *TCO (Total Cost of Ownership*) zu wählen; denn neben den Lizenzkosten fallen zum Beispiel auch die folgenden Kosten an:
  - Kosten für Hardware
  - Kosten für Schulungen. Hierbei ist auch die Anzahl der zu schulenden Personen zu berücksichtigen. Sollen viele Personen mit einem System umgehen können oder sollen es Spezialisten für Sie erledigen?

  - Kosten aufgrund von Ineffizienz, da Personen, ohne entsprechend geschult zu sein, sich statt mit ihrer eigentlichen Arbeit mit Lösungen im Desktopbereich beschäftigen.

  Man kann hier keine generelle Empfehlung für ein desktop- oder serverbasiertes System aussprechen. Dies muss in der speziellen Situation beurteilt und entschieden werden.

- *Datenmenge*: Serverbasierte Systeme sind in der Lage, wesentlich größere Datenmengen zu speichern und effizienter zu verwalten als desktopbasierte Systeme.
- *Benutzeranzahl*: Nicht nur die theoretische Benutzeranzahl ist bei Serversystemen höher. Können bei Access beispielsweise theoretisch 255 Benutzer gleichzeitig auf eine Datenbank zugreifen, würde ich die tatsächliche Grenze mit 20 bis 30 gleichzeitig angemeldeten Benutzern schon als hoch angesetzt sehen. Dies ist aus der Topologie leicht zu erklären. Stellen Sie sich vor, in einem Lokal würden sich alle Kellner um einen Zapfhahn scharen und versuchen, Bier zu zapfen. Das entspricht der Logik eines Desktopsystems. Wesentlich effizienter wäre es, nur eine Person an den Zapfhahn zu stellen, die Bestellungen bearbeitet und die gezapften Biere dann an alle Kellner verteilt. Dies würde ungefähr einem serverbasierten Datenbanksystem entsprechen. Wahrscheinlich werden bei der zweiten Variante mehr Biere in der gleichen Zeit in durstigen Kehlen landen. Daher sehe ich hier klare Vorteile für ein serverbasiertes System.
- *Portabilität*: Eine Desktop-Datenbank, die oft aus einer einzigen Datei besteht, kann sehr leicht beispielsweise auf ein Notebook transferiert werden. Dies funktioniert bei einem serverbasierten System nicht so ohne Weiteres. Ersetzt man allerdings den Begriff Portabilität durch *Zugriff von überall*, könnte man darunter verstehen, auf eine Datenbank remote über eine Webapplikation zuzugreifen. Dafür wäre wiederum eine Serverdatenbank besser geeignet.
- *Flexibilität*: Eine besondere Stärke eines Desktop-Datenbanksystems liegt in der Flexibilität und Einfachheit der Anwendung. Daher wird es gerne verwendet für:
  - Auswertungen (zum Beispiel werden häufig von großen Server-Datenbanksystemen Daten importiert und danach in einem Desktop-Datenbanksystem ausgewertet),
  - Prototyping oder
  - Klein- und Kleinstlösungen.
- *Transaktionen*: Transaktionen sind für konsistente Daten unerlässlich. In der Regel werden diese nur von serverbasierten Systemen geboten.
- *Sicherheit*: Sicherheit ist unter zwei Gesichtspunkten zu betrachten.
  - Die *Zugriffssicherheit* legt fest, wer mit welchen Daten was tun darf.
  - Die *Datensicherheit* legt fest, wie sicher Daten vor Verlust geschützt sind.

  In beiden Bereichen liegen die Vorteile ganz klar und eindeutig bei Server-Datenbanksystemen, die hierzu spezielle Features anbieten.
- *Backup und Recovery*: Server-Datenbanksysteme ermöglichen Sicherungen im Vollbetrieb und häufig auch das verlustfreie Wiederherstellen exakt bis zum Zustand vor einem Crash. Dies gilt nicht für eine Desktop-Datenbank, bei der diese zunächst alle Anwender verlassen müssen.
- *Netzlast*: Aufgrund der Topologie, dass nur das Ergebnis einer Anfrage vom Server an den Client übertragen wird, der diese Daten dann anzeigt und verarbeitet, können serverba-

sierte Systeme auch über schwächere Leitungen performant betrieben werden. Eine vorgegebene Bandbreite erlaubt eine größere Anzahl an Benutzern.

- *Stabilität und Verfügbarkeit*: Serversysteme verfügen über Mechanismen, welche die Verfügbarkeit der Datenbank nach dem Prinzip 24-7-365 (24 Stunden am Tag, 7 Tage die Woche und 365 Tage im Jahr verfügbar) ermöglichen.
- *Skalierbarkeit*: Durch den Einsatz unterschiedlicher Editionen ermöglichen Server-Datenbanken ein stufenloses Skalieren einer Lösung von einer kleinen Abteilungslösung bis hin zu Konzernlösungen.

Analysieren Sie Ihre Anforderungen an ein Datenbanksystem anhand dieser Kriterien und treffen Sie dann Ihre Entscheidung.

HINWEIS: Der Microsoft SQL Server bietet ein professionelles Server-Datenbanksystem zu einem vergleichsweise günstigen Preis. Mit den Editionen von Express bis Enterprise werden alle Bedürfnisse bedient; daneben erlauben sie ein uneingeschränktes Wachsen der Datenbank. Bereits ab der Express Edition können Sie die Vorteile von Sicherheit, Stabilität, Transaktionen und geringer Netzlast nutzen. Zudem ist Microsoft SQL Server ein Tool, das einfach und flexibel in der Handhabung ist wie kaum ein vergleichbares System. Außerdem sind in den letzten Versionen immer mehr Features, die nur in lizenzierten Editionen verfügbar gewesen sind, auch in die Express Edition integriert worden. Neue Features wie temporale Tabellen sind beispielsweise von Beginn an auch in der Express Edition verfügbar.

Wenn Sie sich nicht mit der Konfiguration und Wartung des SQL Servers selber auseinandersetzen möchten, können Sie den SQL Server auch in der Cloud mit Azure DB nutzen. Achten Sie aber aufgrund der Europäischen Datenschutzgrundverordnung (DSGV) darauf, dass Ihre Daten dabei stets ausschließlich in einem Rechenzentrum innerhalb der EU gehostet werden.

1.1.3 Komponenten einer Datenbankanwendung

In der Praxis benötigen Sie keine Datenbank, sondern eine Datenbankanwendung. Auch wenn die Datenbank als „Motor“ einer Anwendung oft die wichtigste Komponente darstellt, ist ein Motor ohne ein Chassis oft nur wenig von Nutzen. Das Chassis ist die Anwendung, die aus einer Datenbank eine Datenbankanwendung macht. Eine Anwendung wird mit einer Entwicklungsumgebung erstellt und greift über standardisierte Schnittstellen mithilfe von SQL auf ein Datenbanksystem zu. Einen Überblick über einsetzbare Programmiersprachen und Schnittstellen zeigt Bild 1.4.

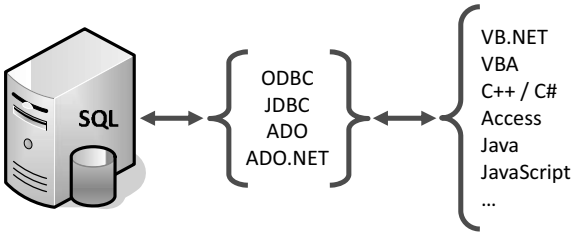

Bild 1.4 Zugriff auf eine Datenbank über Standardschnittstellen

Eine Datenbankanwendung besteht in der Regel aus folgenden Komponenten:

- Datenbankmanagementsystem als Backend für die Verwaltung der Daten
- User-Interface als Frontend für die Bedienung der Anwendung
- Server- und/oder clientseitige Programmierung für die Abbildung von Logiken

Bild 1.5 zeigt eine schematische Darstellung der einzelnen Komponenten und ihr Zusammenspiel.

HINWEIS: Der SQL Server übernimmt in diesem Szenario die Rolle des Datenbankmanagementsystems, auf das mithilfe der Abfragesprache SQL über standardisierte Schnittstellen zugegriffen wird. Für performante Lösungen ergänzt serverseitige Programmierung mittels Transact-SQL und .NET die Datenbankentwicklung mit dem SQL Server.

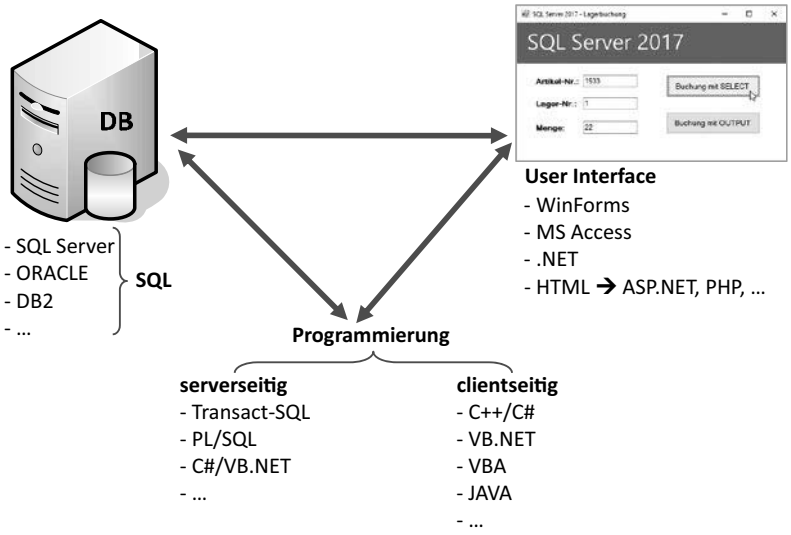

Bild 1.5 Datenbankanwendung und ihre Bestandteile

So lernen Sie den SQL Server in diesem Buch kennen:

- Den SQL Server installieren und konfigurieren
- Datenbanken und Datenbankobjekte mit dem SQL Server erstellen
- Den Zugriff auf Daten mit der Structured Query Language (SQL) vollziehen
- Serverseitig mit Transact-SQL und .NET programmieren
- Die Benutzerverwaltung zur Vergabe von Berechtigungen nutzen
- Sicherung und Wiederherstellung von Datenbanken durchführen
- Erweiterte Features einsetzen

Programmierung im Frontend und Backend

In einer Datenbankanwendung kann sowohl eine Programmierung im Frontend als auch im Backend erfolgen. Im Frontend müssen sämtliche Vorgänge im Zusammenhang mit der Benutzerführung programmiert werden. Manche Vorgänge können aber wahlweise im Frontend oder im Backend programmiert werden. Dies sind vor allem Vorgänge mit Datenbezug.

Die beiden nachfolgenden Abbildungen zeigen die Unterschiede beim Programmablauf von Programmcode, der auf dem Client oder auf dem Server läuft.

Bei clientseitiger Programmierung ist die gesamte Programmlogik im Frontend untergebracht. Werden im Ablauf Informationen aus der Datenbank benötigt oder sind Daten in die Datenbank zu schreiben, werden SQL-Anweisungen zum Datenbankserver geschickt. Mit den Ergebnissen dieser Anweisungen arbeitet der Programmcode anschließend weiter. Ein Programmablauf kann oft aus sehr vielen Einzelschritten bestehen, bei denen mitunter auch sehr viele Datenzugriffe nötig sind.

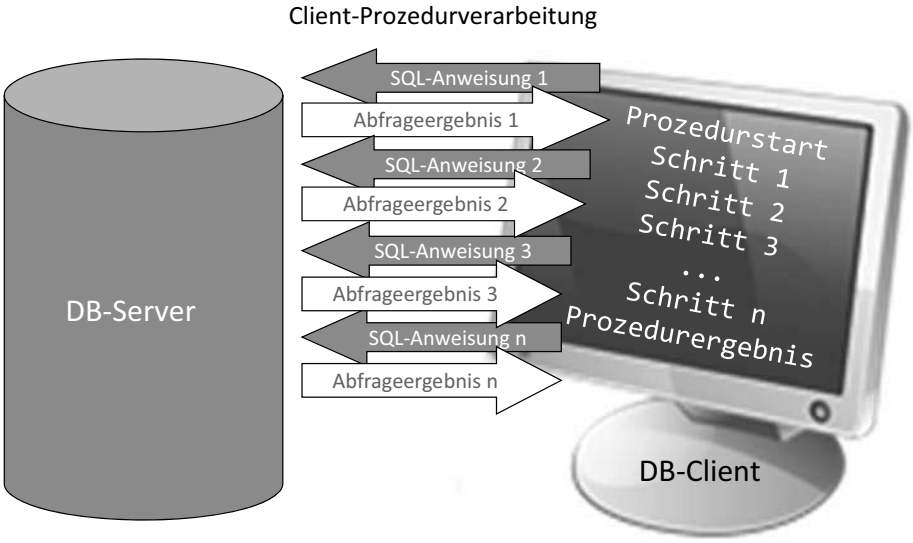

Bild 1.6 Programmlogik im Frontend

Bei serverseitiger Programmierung wird die Programmlogik beispielsweise mithilfe gespeicherter Prozeduren (Stored Procedures) im Backend umgesetzt. Der Vorteil besteht darin, dass das „Hin und Her" zwischen Frontend und Backend entfällt. Im Frontend wird lediglich die am Server hinterlegte Funktionalität aufgerufen und das Ergebnis abgearbeitet.

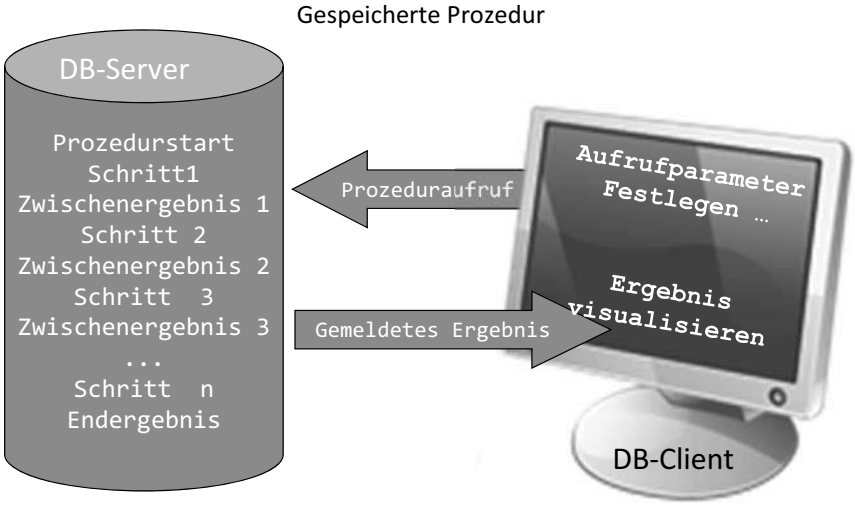

Bild 1.7 Serverseitige Programmierung

In den Kapiteln 4 bis 6 wird das Thema „Serverseitige Datenbankprogrammierung" im Detail behandelt und auf Vor- sowie Nachteile eingegangen. Clientseitige Programmierung ist nicht Thema dieses Buches, da sie nicht vom SQL Server, sondern von der eingesetzten Programmiersprache und Entwicklungsumgebung abhängt. Anhand praktischer Beispiele, die zeigen, wie Programmierelemente des Servers von clientseitigem Code aufgerufen werden, streifen wir jedoch die clientseitige Programmierung.

1.1.4 SQL Server – das Gesamtkonzept

Der SQL Server beschränkt sich keinesfalls auf die Datenbank-Engine. SQL Server ist mittlerweile eine komplette Produktfamilie, die sich um den Kern schart. Damit ist der SQL Server nicht nur ein reines Datenbanksystem. Er bietet auch Lösungen für viele Anwendungen im Datenbankumfeld.

Zur Datenbank-Engine selber zählen folgende Features:

- Volltextsuche
- Datenbankreplikation

Die Zusatzprodukte, oft unter dem Begriff *Business Intelligence (BI)* zusammengefasst, sind folgende Dienste:

- *Integration Services*: Die Integration Services (IS) sind ein umfassendes Werkzeug, um zum Beispiel Daten von A nach B zu transferieren. Dabei sind komplexe Workflows mit Verzweigungen und unzähligen Möglichkeiten realisierbar.

- *Reporting Services*: Aufgabe dieser Services ist es, Berichte, die auf Daten aus der Datenbank basieren, in verschiedenen Formen zur Verfügung zu stellen. Das kann zum Beispiel eine HTML-Seite oder ein PDF-Dokument sein, das per E-Mail verschickt wird. Ziel ist es, das gesamte Berichtswesen eines Unternehmens abbilden zu können. Daher sind diese Berichte auch nicht statisch. Vielmehr erlauben sie es einem Benutzer, durch die Eingabe von Parametern das Ergebnis zu verändern oder über einen definierten Drill-Down immer detailliertere Daten abzurufen. Ein wichtiger Bestandteil der Reporting Services ist neben der Berichtserstellung die Berichtsverteilung. Reporting Services lassen sich sehr gut in Share Point integrieren.
- *Analysis Services*: Diese dienen der Realisierung von Data-Warehouse-Lösungen. Geschäftsleitung, Controller und Marketingmanager benötigen immer anspruchsvollere Analysen und Trendinformationen. Die Basis dafür liegt zu einem Großteil in den bereits auf Servern gespeicherten Unternehmensdaten. In der Praxis werden zur Lösung dieser Aufgabenstellung OLAP-Systeme (Online Analytical Processing; deutsch: analytische Online-Verarbeitung) benötigt, indem auf einfache Weise Informationszusammenstellungen aus OLTP-Daten erstellt werden, die dann für anspruchsvolle Datenanalysen genutzt werden können. Die Analysis Services bieten diese Funktionalität auf einem sehr hohen Niveau und haben den SQL Server in diesem Bereich zu einem der führenden Produkte gemacht.
- *Service Broker*: Dieser Service zielt auf große verteilte Anwendungen ab. Der Service Broker verwaltet Warteschlangen, die mit SQL-Anweisungen „gefüttert“ werden können. Die Inhalte der Warteschlange werden dann der Reihe (englisch: queue) nach abgearbeitet. Diese Warteschlangen können nicht nur am lokalen Server positioniert sein, sondern auch remote abgearbeitet werden.

  Anwendungen, die auf dem Prinzip von Warteschlangen basieren, setzen auf einem anderen Anwendungsverständnis auf, als wir es in der Regel gewohnt sind. Schauen wir uns folgendes Beispiel an: Viele von Ihnen haben sicher schon einmal eine Domänenregistrierung vorgenommen. Wenn Sie eine Domäne registrieren möchten, ist der erste Schritt üblicherweise, dass Sie ermitteln, ob die gewünschte Domäne noch verfügbar ist. In einer Online-Applikation würden Sie eine Schaltfläche anklicken, und die Domäne würde Ihnen gehören. So einfach ist es aber bekanntlich nicht. Sie reichen stattdessen den Antrag bei einer akkreditierten Registrierungsstelle ein. Und hier kommt die Warteschlange ins Spiel. Alle Ihre Eingaben (unter Umständen auch Zusatzinformationen) werden in eine Warteschlange eingereiht. Ihr Antrag steht in der Warteschlange und wird, sobald er an der Reihe ist, bearbeitet. Falls Sie der Erste in der Reihe für diese Domäne gewesen sind, werden Sie die Domäne zugeteilt bekommen.
- *Master Data Services*: Darunter versteht man, wenn Organisationen ihre Stammdaten unternehmensweit zentralisiert vereinheitlichen und für gezielte Analysen bereitstellen.
- *Data Quality Services*: Dies ist ein in dieser Version neues Tool, mit dessen Hilfe die Datenqualität in bestehenden Systemen verbessert werden kann. Lücken in Datenbeständen können damit besser aufgefunden und bereinigt werden. Dies können Fragestellungen sein wie: „Sind alle notwendigen Relationen vorhanden und gesetzt?“

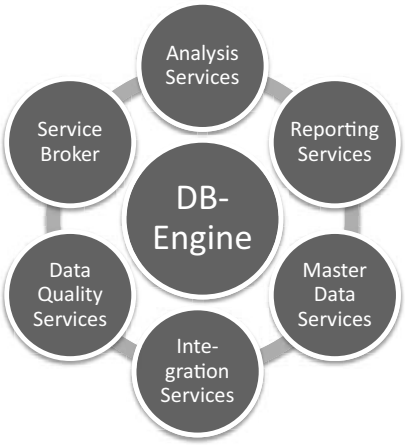

Bild 1.8 SQL Server und seine Zusatzkomponenten

Natürlich stehen diese Features nicht in jeder der verschiedenen SQL Server 2016-Editionen zur Verfügung:

- *Integration Services* stehen ab der Standard Edition zur Verfügung, manche spezielle Formen der Datentransformation erst mit der Enterprise Edition. In allen Editionen ist der SQL Server Import und Export Wizard enthalten.
- *Reporting Services* sind bis zu einem bestimmten Grad bereits ab der Express Edition integriert. Volle Integration findet erst ab der Standard Edition statt.
- *Analysis Services* sind teilweise ab der Standard Edition verfügbar, eine volle Integration ist erst mit der Enterprise Edition gegeben.

1.2 Versionen und Editionen des SQL Servers

Dem Buch liegt die aktuelle Version SQL Server 2017 zugrunde. Diese Version weist gemeinsam mit dem SQL Server 2016 gegenüber deren Vorgängersystem SQL Server 2014 wesentliche Neuerungen auf. Dies betrifft nicht nur die eigentliche relationale Datenbank-Engine, die den Kern des Produkts ausmacht, sondern umfangreiche Erweiterungen der Rahmenprodukte. Diese unter dem Begriff BI (Business Intelligence) zusammengefassten Produkte enthalten beispielsweise die Analysis Services, Integration Services oder Reporting Services.

Erneuerungen gibt es in fast allen Bereichen des SQL Servers. Drei große Schwerpunkte, die ich in meiner täglichen Arbeit nutze, sind:

- *Speicheroptimierte Tabellen*: Ein wahrer Performanceboost sind die mit dem SQL Server 2014 eingeführten speicheroptimierten Tabellen. Hinter diesem Begriff verbirgt sich die Möglichkeit, ganze Tabellen vollständig im RAM zu halten. Da der Zugriff auf Festplatten in der Regel das ist, was eine Datenbank am meisten bremst, bietet diese Möglichkeit

ungeahnte Performancesteigerungen gegenüber herkömmlichen Tabellen. Hat es in der ersten Implementierung noch einige Einschränkungen in der Verwendung gegenüber herkömmlichen Tabellen gegeben – was in der ersten Implementierung eines komplexen und bahnbrechenden Features nichts Ungewöhnliches ist –, sind die meisten dieser Limitationen nun ausgeräumt. Damit ist dies nun ein extrem gutes und praxistaugliches Feature. Und ein weiteres Plus aus meiner Sicht: Es ist nun nicht mehr auf die Enterprise Edition beschränkt.

- *Always Encrypted*: Sensible Daten werden durch Zugriffsrechte auf dem Server vor unbefugtem Zugriff geschützt. Schon bisher hat es die Möglichkeit gegeben, Zugriffsrechte dermaßen einzuschränken, dass Daten vor unbefugten Zugriffen bestmöglich geschützt worden sind. Daten im Backup zu verschlüsseln, ergänzte diese Möglichkeiten, da die Zugriffsrechte, sobald die Datenbank auf einem eigenen Server eingespielt werden kann, selbst definiert werden können. Mit den neuen Möglichkeiten sind sie auch auf dem Server in der Datenbank verschlüsselbar, sodass selbst jemand, der sich der gesamten Datenbank bemächtigt, diese Daten nicht mehr einsehen kann.
- *Temporale Tabellen*: Änderungsprotokollierung ist in der Praxis sehr oft ein Thema. Trotz mehrerer Ansätze in der Vergangenheit, die in der Praxis kein voll befriedigendes Ergebnis geliefert haben, dies einfach vom System her zur Verfügung zu stellen, werden saubere und zufriedenstellende Lösungen mit Triggern ausprogrammiert. Temporale Tabellen liefern aus meiner Sicht erstmals eine sehr brauchbare Lösung, ohne eigene Programmierung eine Lösung für diese Aufgabenstellung zu erhalten. Mit dem SQL Server 2016 eingeführt, unterstützen sie mit dem SQL Server 2017 auch Änderungs- und Löschweitergaben bei Beziehungen (FOREIGN KEY mit DELETE CASCADE und UPDATE CASCADE).
- *Strict Security für Common Language Runtime (CLR)*: Neue Erweiterungen für die Sicherheit von CLR-Code erhöhen mit dem SQL Server 2017 zwar den Aufwand, um CLR-Code auf dem SQL Server ausführen zu können, bieten aber deutlich höhere Sicherheit. Unter CLR-Code versteht man mit C# oder VB.NET programmierte Prozeduren, die mittels der Common Language Runtime (CLR) direkt auf dem SQL Server ausgeführt werden können.

Ein paar weitere Erweiterungen, die teilweise auch in diesem Buch behandelte Themen betreffen, habe ich exemplarisch in der nachfolgenden Tabelle angeführt.

Tabelle 1.1 Einige Neuerungen in SQL Server 2016 und 2017

Thema	Beschreibung
Windows Azure	Daten können, anstelle lokal auf dem Server gehalten zu werden, in Windows Azure-BLOBs abgelegt werden. Das Hosten von SQL Server-Datenbanken auf einem virtuellen Computer in Windows Azure wird über eigene Bereitstellungsassistenten unterstützt. Die Sicherung einer SQL Server-Datenbank kann über eine URL in Windows Azure-BLOBs erfolgen.
Erstellen von Ausführungsplänen	Durch die überarbeitete Logik der Kardinalitätsschätzung werden die Qualität und damit die Effizienz von Ausführungsplänen verbessert. Das wiederum steigert die Abfrageleistung. (Ausführungspläne legen fest, wie der SQL Server intern eine von uns getätigte Abfrage abarbeitet.)
Live-Abfragestatistik	Abfragestatistiken zur Laufzeit bieten Administratoren einen tieferen Einblick in die Verarbeitung von Anweisungen.
Transact-SQL-Erweiterungen	Es gibt zahlreiche Erweiterungen in Form von neuen Funktionen und Anweisungen in der Sprache zur Bearbeitung von Daten. Beispielsweise sind dies die Funktionen DATEDIFF_BIG(), STRING_SPLIT() und STRING_ESCAPE() mit dem SQL Server 2016 und die Funktionen STRING_AGG(), TRANSLATE(), CONCAT_WS() und TRIM() mit dem SQL Server 2017.
Sicherheitserweiterungen	Berechtigungen können nun auf Datensatzebene vergeben werden. Für diese Funktionalität ist bisher ein Workaround über gefilterte Sichten notwendig gewesen.
Temporäre Datenbank *tempdb*	Das Aufteilen der temporären Datenbank ***tempdb*** auf mehrere Datendateien, die auch auf unterschiedlichen Datenträgern verwaltet werden können, ermöglicht eine bessere interne Nutzung und damit eine bessere Performance.
Native JSON-Unterstützung	Für Entwickler interessant ist die native Unterstützung von JSON als Ausgabeformat mit der Klausel FOR JSON, die mit der bisherigen Klausel FOR XML vergleichbar ist.
UTF-8 Unterstützung	Ich bin geneigt zu sagen „endlich“! Die Anweisung BULK INSERT, mit der Textdateien (CSV) einfach und schnell importiert werden können, unterstützt ab dem SQL Server 2016 nun auch UTF-8. Das ist mir in der Vergangenheit sehr oft abgegangen.
GRAPH-Unterstützung	Die Unterstützung von Graph-Datenbanken des SQL Server 2017 ist ein weiterer Schritt zur Integration von NoSQL-Elementen.

Editionen des SQL Server 2017

Microsoft liefert den SQL Server 2017 in einer Reihe unterschiedlicher Editionen aus. Ziel dieser Produktdifferenzierung ist es, dem Kunden ein Angebot zu unterbreiten, das es ermöglicht, den jeweiligen Anforderungen in Hinblick auf Leistungsfähigkeit, Laufzeit und Preise gerecht zu werden. Darüber hinaus werden zahlreiche Zusatzkomponenten angeboten. Welche dieser Komponenten im Einzelfall für eine Installation ausgewählt werden, hängt von den konkreten Anforderungen ab. Die früher verfügbare Business Intelligence Edition ist seit dem SQL Server 2016 nicht mehr verfügbar.

5 Transact-SQL – die Sprache zur Serverprogrammierung

In diesem Kapitel lernen Sie die grundlegenden Konzepte der Datenbankprogrammiersprache Transact-SQL. Sie erfahren, aus welchen Bestandteilen diese Sprache besteht, wie sie eingesetzt wird und welcher Syntax sie sich bedient. Am Ende dieses Kapitels sollten Sie in der Lage sein, die Sprache zu verstehen und anzuwenden. Das Erstellen von Datenbankobjekten, die mittels dieser Sprache programmiert werden, ist Thema des darauffolgenden Kapitels.

Über die SQL Server CLR (Common Language Runtime) kann serverseitige Programmierung auch mit .NET-Programmiersprachen erfolgen. Dies ist eine Ergänzung zu Transact-SQL, um gemeinsam das gesamte Spektrum der Datenbankprogrammierung abzudecken. Sie können auch als .NET-Programmierer nicht auf Transact-SQL verzichten. Spätestens dann, wenn Sie auf Daten zugreifen, werden Sie diese Anweisungen auch innerhalb des .NET-Codes benötigen. Die Einsatzbereiche dieser beiden Sprachen lassen sich wie folgt abgrenzen:

- Transact-SQL sollten Sie immer dann einsetzen, wenn Datenzugriffe im Vordergrund stehen. Dabei spielt es keine Rolle, ob Sie lesend oder schreibend auf Daten zugreifen. Effiziente Formen des Datenzugriffs sind die Stärke von Transact-SQL.
- .NET-Programmierung ist dann zu bevorzugen, wenn es aufwendige Algorithmen umzusetzen gilt. Zusätzlich wird .NET-Programmierung für alles eingesetzt, was nicht direkt datenbankspezifisch ist; zum Beispiel der Zugriff auf das Filesystem, auf einen FTP-Server oder das Bearbeiten einer Grafikdatei, bevor sie in die Datenbank eingefügt wird.

HINWEIS: In diesem Kapitel beschäftigen wir uns ausschließlich mit Transact-SQL. Der .NET-Programmierung mit dem SQL Server widmen wir uns später in einem separaten Kapitel.

Die standardisierte Abfragesprache SQL (ANSI SQL) ist keine prozedurale Sprache. Zur Programmierung reichen die vorhandenen Funktionalitäten nicht aus. Weiterführende Funktionen werden von der prozeduralen Spracherweiterung von SQL geboten: Transact-SQL.

Transact-SQL ist eine prozedurale Spracherweiterung zu SQL, die bestimmte Konstrukte aufweist, die wir von Programmiersprachen der dritten Generation (3GL-Sprachen) her kennen. Genannt seien hier beispielsweise Auswahl- und Wiederholungsstrukturen. Die Erwei-

terung ist rein herstellerbezogen und nur in den Produkten MS SQL Server und Sybase Adaptive Server Enterprise enthalten. Das liegt daran, dass diese beiden Produkte vor vielen Jahren gemeinsame Wurzeln gehabt haben. Gleichartige und vergleichbare Sprachen haben auch andere Hersteller in ihren Datenbankprodukten implementiert. Beispielsweise heißt die prozedurale Spracherweiterung bei Oracle PL/SQL, wobei PL für Procedural Language steht. PL/SQL und Transact-SQL haben jedoch – außer der Grundkonzeption und Funktionalität – hinsichtlich ihrer Sprachsyntax nicht allzu viele Gemeinsamkeiten.

Transact-SQL – auch kurz als T-SQL bezeichnet – wird bei der Entwicklung von Datenbanken unter anderem für die Erstellung *gespeicherter Prozeduren* (Stored Procedures), *benutzerdefinierter Funktionen* (Userdefined Functions) und *Schalter* (Trigger) verwendet. Zunächst zur begrifflichen Abgrenzung:

1. *Stored Procedures* sind Programme, die direkt auf dem Server gespeichert und ausgeführt werden. Der Aufruf erfolgt häufig aber von Client-Anwendungen aus. Um die Wirkungsweise der Prozedur zu steuern, werden einer Stored Procedure beim Aufruf Parameterwerte übergeben. Dies erfolgt wie bei Prozeduren in anderen Programmiersprachen auch.
2. *Userdefined Functions* liefern wie Systemfunktionen einen Wert oder eine Tabelle zurück und können im Gegensatz zu gespeicherten Prozeduren auch in SQL-Anweisungen verwendet werden.
3. *Trigger* sind mit den Ereignisprozeduren in anderen Programmiersprachen vergleichbar. Sie können nicht wie Prozeduren und Funktionen explizit aufgerufen werden. Sie werden vielmehr durch Ereignisse ausgelöst, die in Tabellen auftreten. Diese sind das Einfügen (`INSERT`), Ändern (`UPDATE`) und Löschen (`DELETE`) von Datensätzen.

Auf das Erstellen von gespeicherten Prozeduren, benutzerdefinierten Funktionen und Triggern gehe ich im nächsten Kapitel ein. In diesem Kapitel werden wir uns der Sprache T–SQL widmen.

Für die Eingabe und Erstellung der Beispiele in diesem Kapitel verwenden wir das SQL Server Management Studio. Öffnen Sie dieses bitte, melden Sie sich am Datenbankmodul Ihres Servers an und wählen Sie die Datenbank *wawi* aus.

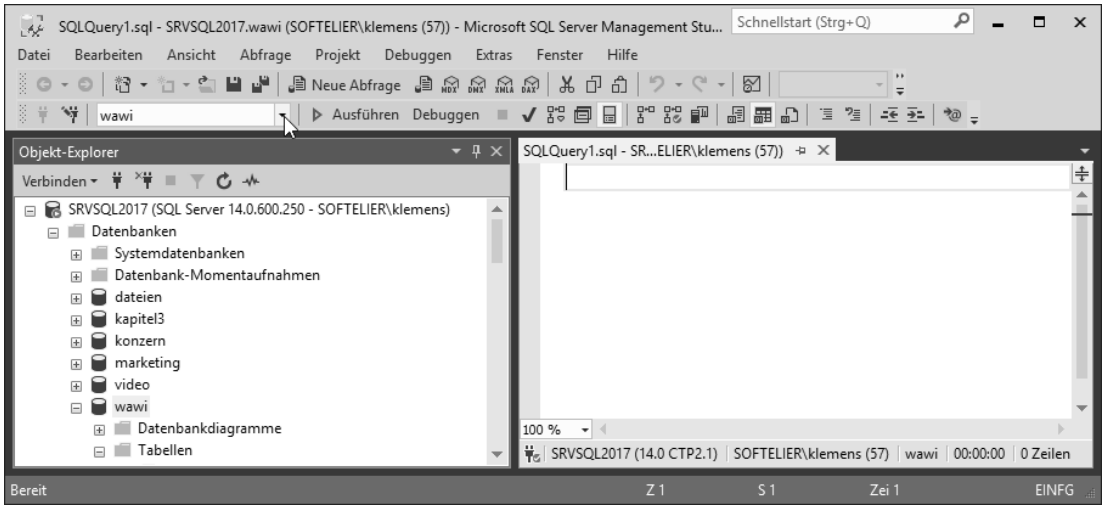

Bild 5.1 Beispieldatenbank für die Arbeit in diesem Kapitel auswählen

5.1 Bestandteile und Funktionalität von Transact-SQL

In diesem ersten Abschnitt gebe ich Ihnen eine Übersicht über die wichtigsten Elemente von T-SQL.

HINWEIS: Damit Sie die nachfolgenden Beispiele verstehen, möchte ich darauf hinweisen, dass in Transact-SQL so wie auch in SQL innerhalb einer Anweisung ein Zeilenumbruch stattfinden darf. Es kann also vorkommen, dass eine Programmzeile nicht unbedingt eine Codezeile sein muss. Solche Zeilenumbrüche werden vor allem dann verwendet, wenn durch sie die Lesbarkeit besonders langer Anweisungen erhöht werden kann. Sehr häufig betrifft dies SQL-Anweisungen, bei denen jede Klausel in eine eigene Zeile geschrieben wird.

5.1.1 Variablen und Datentypen

Wie in jeder anderen Programmiersprache stehen in Transact-SQL auch Variablen zur Verfügung. Es handelt sich dabei um benutzerdefinierte Objekte, die einen Datentyp haben und in denen Werte während der Programmausführung zwischengespeichert und abgerufen werden können. Diese können zum einen Variablen sein, die im Programmcode deklariert werden und bei der Programmausführung Werte zugewiesen bekommen. Es können zum anderen auch Variablen sein, die bereits beim Aufruf der Prozedur mit Übergabewerten gefüllt werden.

Einer Variablen können nicht beliebige Inhalte zugewiesen werden. Diese besitzen – genauso wie Datenfelder einer Tabelle – bestimmte Datentypen. Die für Variablen verwendbaren Datentypen sind dieselben, die innerhalb der Datenbank für Felddatentypen zur Verfügung stehen.

Eine Übersicht entnehmen Sie der nachfolgenden Tabelle.

Tabelle 5.1 Datentypen für Variablen

Kategorie	Datentyp
Character	CHAR(Länge) VARCHAR(Länge) NCHAR (Länge) NVARCHAR(Länge) VARCHAR(MAX) NVARCHAR(MAX)

(Fortsetzung nächste Seite)

Tabelle 5.1 Datentypen für Variablen *(Fortsetzung)*

Kategorie	Datentyp
Datum/Uhrzeit	DATETIME SMALLDATETIME DATE DATETIME2(Länge) DATETIMEOFFSET(Länge) TIME
Zahl	DECIMAL(Genauigkeit, Dezimalstellen) FLOAT(Länge) REAL BIGINT INT SMALLINT TINYINT
Währung	MONEY SMALLMONEY
Boolean	BIT
Binär	BINARY(Länge) VARBINARY(Länge) VARBINARY(MAX)
XML	XML
Variante	SQL_VARIANT
Hierarchie	HIERARCHYID
Räumlich	GEOGRAPHY GEOMETRY

Da Sie vielleicht so manche SQL Server-Version „übersprungen“ haben, möchte ich noch kurz auf den Datentyp *VARCHAR(MAX)* eingehen. Dieser Datentyp vereint die Vorteile der Datentypen *VARCHAR()* und *TEXT*. In einem Feld und in einer Variablen vom Datentyp *TEXT* können mehr als 8000 Zeichen gespeichert werden; allerdings können diese nicht mit Standard-SQL-Anweisungen und Zeichenfolgefunktionen verarbeitet werden. *TEXT* und *IMAGE* – ersetzt durch *VARBINARY(MAX)* – existieren nur aus Gründen der Abwärtskompatibilität und sind bereits auf der Liste der abgekündigten Features.

Folgendes können Sie mit *VARCHAR(MAX)* tun, was mit *TEXT* nicht möglich ist:

- Verwenden als Datentyp für Variablen
- Verwenden von String-Funktionen wie beispielsweise `CHARINDEX()` oder `REPLACE()` zur Bearbeitung
- Inhalte mit anderen Feldern oder Variablen eines *CHARACTER*-Datentyps verketten

Auch *IMAGE* kann nicht für Variablen verwendet werden, der Nachfolger *VARBINARY(MAX)* aber sehr wohl.

Beachten Sie bitte, dass SQL Server zwischen zwei Arten von Variablen unterscheidet:

1. *Benutzerdefinierte Variablen* werden innerhalb eines Transact-SQL-Programms oder einer Benutzersitzung vom Benutzer erzeugt und gelten ausschließlich innerhalb des Pro-

gramms oder der Sitzung, in der sie deklariert wurden. Benutzerdefinierte Variablen werden innerhalb einer Prozedur mit der Anweisung DECLARE erzeugt. Der Name von benutzerdefinierten Variablen beginnt stets mit einem @.

2. *Globale Variablen* sind vom System vordefinierte Variablen, deren Inhalte durch das System zugewiesen werden. Die Inhalte dieser Variablen geben dem Benutzer wertvolle Informationen über das System oder über aktuelle Zustände im Programmcode. Die Namen von globalen Variablen beginnen immer mit @@. Globale Variablen können nur gelesen werden; ihnen kann explizit kein Wert zugewiesen werden. Damit ähneln sie Systemfunktionen in der Verwendung.

HINWEIS: Das System, dass Variablen in Transact-SQL immer mit einem @ beginnen, erleichtert die Arbeit und vor allem die Lesbarkeit von Programmcode enorm. Variablen sind sofort als solche zu erkennen, auch wenn sie denselben Namen haben wie das Feld einer Tabelle. Verwechslungen mit Feldnamen sind daher ausgeschlossen. Nur zum Vergleich: In der Oracle-Programmiersprache PL/SQL werden Variablen nicht extra derart als solche gekennzeichnet. Hier muss sich der Entwickler an Prioritätsregeln halten, um zu bestimmen, ob bei Namensgleichheit die Variable oder der Feldname gemeint ist. In dieser Hinsicht gefällt mir die SQL Server-Implementierung wesentlich besser.

Lokale Variablen deklarieren

Lokale Variablen werden mit der Anweisung DECLARE unter Angabe ihres Datentyps definiert. Dabei kann optional das Schlüsselwort AS verwendet werden.

```
DECLARE @var1 AS int
DECLARE @var2 AS smalldatetime
DECLARE @var3 AS varchar(25)
```

Mit einer DECLARE-Anweisung können Sie auch mehrere Variablen in einer Zeile deklarieren. Dabei müssen alle Variablen mit Komma voneinander getrennt geschrieben werden.

```
DECLARE @var1 int, @var2 smalldatetime, @var3 varchar(25)
DECLARE @var4 int
```

ACHTUNG! Auch wenn Sie mehrere Variablen desselben Datentyps deklarieren, muss bei jeder Variablen der Datentyp separat angegeben werden. Es ist nicht möglich, eine Auswahl für mehrere Variablen gemeinsam zu definieren.

So ist zum Beispiel nachfolgende Deklaration, die drei Variablen vom Typ Integer deklarieren soll, ungültig.

```
DECLARE @var1, @var2, @var3 int
```

Stattdessen muss der Datentyp bei jeder Variablen explizit angegeben werden.

```
DECLARE @var1 int, @var2 int, @var3 int
```

Die Wertzuweisung an eine Variable kann auf zwei Arten erfolgen:

- SET-Anweisung
- SELECT-Anweisung

Die direkte Zuweisung eines Variablenwertes erfolgt mit der Anweisung SET. Die Syntax hierzu lautet:

```
SET @variable = wert
```

Als Wert kann der Variablen ein skalarer Wert, ein Berechnungsausdruck oder das Ergebnis einer Unterabfrage zugewiesen werden. Dabei ist zu berücksichtigen, dass die Unterabfrage in runde Klammern gesetzt werden muss.

```
SET @variable = (SELECT wert FROM ...)
```

Wird ein Wert über eine Unterabfrage zugewiesen, muss diese so ausgelegt sein, dass sie genau eine Spalte und eine Zeile zurückgibt. Liefert die Abfrage mehrere Zeilen – weil beispielsweise die WHERE-Klausel nicht korrekt ist –, führt dies zu einem Fehler.

Sie deklarieren beispielsweise eine Variable und weisen ihr den Namen eines Mitarbeiters zu. Sie vergessen dabei aber die WHERE-Klausel, die sicherstellen sollte, dass die Unterabfrage nur eine Zeile zurückliefert. Also zum Beispiel so:

```
DECLARE @nachname varchar(50)
SET @nachname = (SELECT nachname FROM dbo.personal)
```

Das System meldet Ihnen einen Fehler:

```
Meldung 512, Ebene 16, Status 1, Zeile 2
Die Unterabfrage hat mehr als einen Wert zurückgegeben. Das ist nicht zulässig, wenn
die Unterabfrage auf =, !=, <, <=, > oder >= folgt oder als Ausdruck verwendet wird.
```

Um mehrere Werte aus einer Tabelle abzufragen, müssen Sie daher mehrere SET-Anweisungen verwenden.

Sie möchten beispielsweise den Nachnamen, den Vornamen und das Geburtsdatum des Mitarbeiters mit der Personalnummer 452 in Variablen einlesen:

```
DECLARE @nachname varchar(50), @vorname varchar(50)
DECLARE @gebdatum date

SET @nachname = ( SELECT nachname
                  FROM dbo.personal
                  WHERE persnr = 452 )
SET @vorname = ( SELECT vorname
                 FROM dbo.personal
                 WHERE persnr = 452 )
SET @gebdatum = ( SELECT gebdatum
                  FROM dbo.personal
                  WHERE persnr = 452 )

SELECT @nachname AS NN, @vorname AS VN, @gebdatum AS Geburtsdatum;
```

Ergebnis:

```
NN              VN              Geburtsdatum
--------------  --------------- --------------
Kossegg         Anita           1969-06-20
(1 Zeile(n) betroffen)
```

Die abschließende SELECT-Anweisung dient zur Anzeige der Variableninhalte.

Da Sie mit der SET-Anweisung immer nur einen Wert zuweisen können, müssen Sie mehrere Anweisungen und damit mehrere Abfragen hintereinander verwenden, um drei Werte aus derselben Zeile einer Tabelle auszulesen und in Variablen abzulegen.

In einer solchen Situation ist es effizienter und sinnvoller, die SELECT-Anweisung zur Zuweisung der Variableninhalte zu verwenden. Da mit einer SELECT-Anweisung auch mehrere Variablen mit einer einzigen Anweisung befüllt werden können, benötigen Sie für das vorangegangene Beispiel anstelle von drei separaten Datenzugriffen lediglich einen einzigen.

Die Syntax für die Wertzuweisung über die SELECT-Anweisung lautet:

```
SELECT @var1 = wert1, @var2 = wert2, @var3 = wert3, ...
[FROM ...]
```

Jeder Variablen wird ein Wert zugewiesen. Die einzelnen Zuweisungen werden voneinander mit Komma getrennt. Sofern die Werte aus einer Abfrage stammen, kann diese direkt in die Zuweisung integriert werden. Ergänzen Sie dazu die SELECT-Anweisung mit einer FROM-Klausel und optional mit weiteren Klauseln, die Sie von SELECT-Anweisungen her kennen.

Das obige Beispiel (Name, Vorname und das Geburtsdatum für den Mitarbeiter mit der Personalnummer 452 sollen in einer Variablen gespeichert werden) ist mithilfe der SELECT-Anweisung folgendermaßen zu realisieren:

```
DECLARE @nachname varchar(50), @vorname varchar(50)
DECLARE @gebdatum date

SELECT @nachname = nachname,
       @vorname = vorname,
       @gebdatum = gebdatum
FROM dbo.personal
WHERE persnr = 452;

SELECT @nachname AS NN, @vorname AS VN, @gebdatum AS Geburtsdatum;
```

Ergebnis:

```
NN              VN              Geburtsdatum
--------------  --------------- --------------
Kossegg         Anita           1969-06-20
(1 Zeile(n) betroffen)
```

Auch hier ist darauf zu achten, dass die Anweisung nur eine Zeile zurückliefert. Im Unterschied zur ersten Variante führt es aber zu keinem Fehler, falls mehrere Zeilen geliefert werden. Nach der Anweisung sind jene Werte in den Variablen anzufinden, welche die letzte zurückgegebene Zeile geliefert hat. Da dies oft zu unerwarteten Ergebnissen führen kann,

7 SQL Server CLR-Integration

Mittlerweile schon seit der Version 2005 sind Sie beim Programmieren des SQL Servers nicht mehr nur auf Transact-SQL beschränkt. Dabei arbeiten der SQL Server und das Visual Studio eng zusammen. Das ist auch notwendig, denn der SQL Server enthält eine *Common Language Runtime* (CLR), das heißt, er ist in der Lage, .NET-Code laufen zu lassen. Für die Code-Entwicklung benötigen Sie das Visual Studio. Erst der fertige Code wird in einer Datenbank auf dem SQL Server integriert.

Die *SQL Server Data Tools* (SSDT) sind einerseits in Visual Studio 2015 und 2017 integriert, sind aber andererseits auch in beiden Versionen frei separat verfügbar. Die Integration erstreckt sich auch auf die freie Community Edition des Visual Studio. Somit ist gewährleistet, dass man ohne separat lizenziertes Visual Studio Professional den vollen Umfang der CLR-Programmierung für den SQL Server nutzen kann.

Die Data Tools sind ein umfangreiches Toolset, das alle Bereiche der Datenbankentwicklung mit dem SQL Server abdeckt. Dies ist nicht nur für Programmierer von Vorteil, die lieber in ihrer gewohnten Umgebung bleiben möchten und nicht so gerne mit den SQL Server-Tools arbeiten. Wir konzentrieren uns in diesem Kapitel auf die Programmierung für die SQL Server CLR und werden uns im nächsten Kapitel ausführlicher mit den Data Tools auseinandersetzen.

Sofern Sie noch keine Form der Data Tools auf Ihrem Rechner verfügbar haben, laden Sie diese unter folgender Adresse herunter: *https://docs.microsoft.com/de-de/sql/ssdt/download-sql-server-data-tools-ssdt*

Je nachdem, welche Version Sie bevorzugen, laden Sie sich die SSDT für Visual Studio 2015 oder 2017 herunter oder eine der unterstützten Editionen für Visual Studio in denselben Versionen. Die SQL Server Data Tools integrieren sich in eine bereits installierte Version oder verwenden die abgespeckte Visual Studio Shell. Ich verwende hier das freie Visual Studio 2017 Community Edition. Für Sie sollte es einerlei sein, ob Sie Visual Studio 2015 oder 2017 einsetzen.

ACHTUNG! Frühere Versionen der SQL Server Data Tools werden nicht mehr aktualisiert und sind bereits nicht mehr auf dem letzten Stand. Daher können Sie diese für den SQL Server 2017 nicht mehr verwenden.

Ziel der CLR-Programmierung ist es, Transact-SQL in denjenigen Bereichen zu ergänzen, wo es naturgemäß Schwächen gibt. Dies sind vor allen Aufgabenstellungen, die

- einen sehr komplexen Algorithmus verlangen
- oder einen Bezug außerhalb der Datenbank – wie zum Beispiel Zugriff auf das Dateisystem – aufweisen.

Überall dort, wo Datenzugriffe im Vordergrund stehen, sollte weiterhin T-SQL zum Einsatz kommen. Aufgabenstellungen, für die in älteren SQL Server-Versionen erweiterte Systemprozeduren (`xp_...`) zum Einsatz kamen, werden nun über eine Common Language gelöst. Erweiterte Systemprozeduren werden lediglich aus Gründen der Abwärtskompatibilität noch unterstützt.

HINWEIS: Für die Arbeit mit diesem Kapitel ist es von Vorteil, wenn Sie bereits mit der .NET-Programmierung und dem Visual Studio vertraut sind. Insbesondere benötigen Sie Kenntnisse in ADO.NET. Da eine eingehende Behandlung dieser Themen über den Rahmen dieses Buches hinausginge, verweise ich auf weiterführende Literatur zu diesen Themen.

7.1 Mit im Boot: .NET Framework

Der SQL Server ist durch die Common Language Runtime (CLR) in der Lage, .NET-Code auszuführen. Das Visual Studio dient als Entwicklungswerkzeug für die vom SQL Server ausführbaren Objekte.

Diese sind:

- .NET User-Defined Functions (UDF)
- .NET Stored Procedures
- .NET Trigger
- User-Defined Aggregates (UDA)
- User-Defined Datatypes (UDT)

Benutzerdefinierte Funktionen, gespeicherte Prozeduren und Trigger gleichen in ihrer Funktionalität und ihrem Einsatzbereich ihren Transact-SQL-Pendants.

HINWEIS: Microsoft bedient sich bei der Weiterentwicklung des SQL Servers dieser Funktionalität. So sind die räumlichen Datentypen *geography* und *geometry* sowie der Datentyp *hierarchy_id* als .NET-Datentypen integriert worden.

Aber damit ist noch nicht alles abgedeckt – das .NET Framework spielt auch bei der Verwaltung des SQL Servers mit. Die COM-basierten *Distributed Management Objects* (*SQL-DMO*),

die früher verwendet worden sind, sind mittlerweile durch die .NET-basierten *SQL Server Management Objects* (*SMO*) abgelöst worden.

Mit dem .NET Framework und der Datenbank-Engine prallen aber auch zwei Welten aufeinander, zwischen denen Brücken geschlagen werden müssen. Daher gibt es einen eigenen Satz an SQL-Datentypen, um SQL Server-Datentypen mit den .NET-Datentypen zu verbinden. Eine Übersicht finden Sie in der nachfolgenden Tabelle.

Innerhalb von .NET-Code verwenden Sie die SQL-Typen an den Schnittstellen von und zur Datenbank. Innerhalb des Codes verwenden Sie wie gewohnt die .NET-Datentypen. Sie benutzen übliche Konvertierungen, um Inhalte von Variablen mit .NET-Typen in solche mit SQL-Typen und umgekehrt zu konvertieren.

Tabelle 7.1 Datentypenzuordnung

SQL Server-Datentyp	SqlType	.NET-Datentyp
char varchar nchar nvarchar text ntext	SqlString	String
bigint	SqlInt64	Int64
int	SqlInt32	Int32
smallint	SqlInt16	Int16
tinyint	SqlByte	Byte
numeric decimal	SqlDecimal	Decimal
money smallmoney	SqlMoney	Decimal
real	SqlSingle	Single
float	SqlDouble	Double
datetime smalldatetime	SqlDateTime	Datetime
bit	SqlBoolean	Boolean
binary varbinary image timestamp	SqlBinary SqlBytes	Byte()
uniqueidentifier	SqlGuid	Guid

Sie sollten, bevor Sie mit diesem Kapitel arbeiten, die Kapitel 5 und 6 gelesen haben. In diesen lernen Sie nicht nur die Sprache Transact-SQL, sondern auch die Konzepte hinter dem Einsatz von gespeicherten Prozeduren, Triggern und benutzerdefinierten Funktionen kennen. Diese werden in diesem Kapitel benötigt und nicht nochmals erarbeitet.

7.1.1 Integration mit dem Visual Studio

Die SSDT werden mit dem Setup von SQL Server noch nicht mit installiert. Wie zuvor und in Kapitel 2 beschrieben, müssen Sie die SQL Server Data Tools oder eine entsprechende Visual Studio-Version separat installieren.

HINWEIS: In diesem Kapitel werden wir die Beispiele mit Visual Basic .NET umsetzen. Sie finden bei den Beispieldateien zum Buch allerdings alle Beispiele auch mit C# umgesetzt. Beide Varianten im Text zu behandeln, würde den zur Verfügung stehenden Rahmen sprengen. Daher habe ich mich für das für Einsteiger einfachere Visual Basic entschieden.

Sie finden bei den Buchbeispielen Projektdateien in VB.NET und C# für die Visual Studio-Versionen 2015 und 2017.

Erstellen wir zu Beginn mit den SSDT ein neues Projekt mit der Vorlage *SQL Server-Datenbankprojekt* aus der Gruppe *SQL Server*.

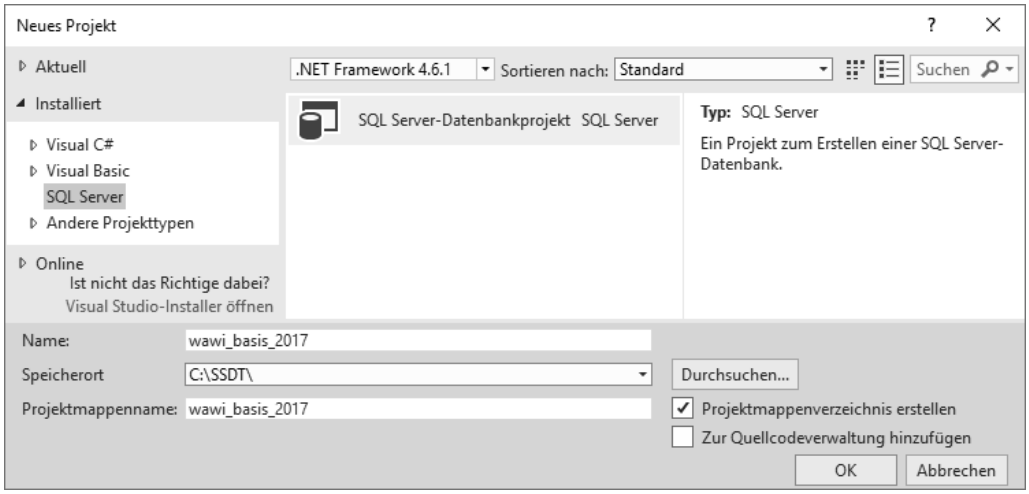

Bild 7.1 Neues SQL Server-Datenbankprojekt in Visual Studio 2017

ACHTUNG! Ändern Sie nach dem Erstellen das Framework von 4.6.1 zum Beispiel auf 4.0, damit das Projekt später auch auf den SQL Server 2012 übertragen werden kann. Verwenden Sie die Version 3.5, wenn Sie auch den SQL Server 2008 R2 unterstützen möchten. In Visual Studio 2017 können Sie dies auch schon direkt im Dialog beim Erstellen erledigen.

Der Name, den Sie diesem Projekt geben, wird später in Ihrer Datenbank als Assembly-Name verwendet, wenn Sie die im Studio erstellten Objekte von Visual Studio automatisch bereitstellen oder veröffentlichen lassen. Das neue Projekt wird wie gewohnt im Projekt-

mappen-Explorer mit dem Projektnamen angezeigt. Dort finden Sie den Eintrag *Projektmappe „wawi_basis_2017“*, den Ordner *wawi_basis_2017* mit den Unterordnern *Properties* und *Verweise*.

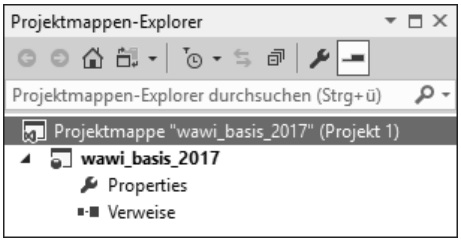

Bild 7.2 Projektmappen-Explorer

In Visual Studio 2015/2017 ist standardmäßig eine Verbindung zu einer LocalDB im SQL Server-Objekt-Explorer eingerichtet. Ist dieser bei Ihnen noch nicht sichtbar, können Sie ihn über das Menü **Ansicht** einblenden. Wenn Sie möchten, richten wir uns eine neue Verbindung zu unserem Server ein, da wir mit unserer Datenbank *wawi* arbeiten möchten. Wir benötigen diese Verbindung zwar nicht unbedingt, um für die CLR zu programmieren, aber wir haben dadurch die Namen von Tabellen und Spalten im Blickfeld, was sicher kein Nachteil ist. Außerdem können wir erstellte Objekte sofort in der Datenbank sehen. Wählen wir dazu im Menü **Extras** den Befehl **SQL Server hinzufügen...** oder klicken auf das entsprechende Symbol im *SQL Server-Objekt-Explorer*. Im anschließenden Anmeldedialog, den wir vom SQL Server Management Studio kennen, melden wir uns an unserem Server an.

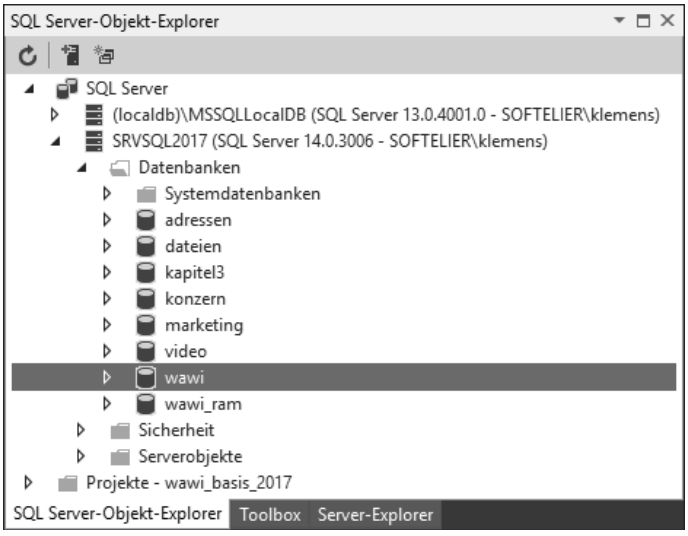

Bild 7.3 SQL Server-Objekt-Explorer

ACHTUNG! Je nachdem, welche Installationsmedien Sie für Ihr Visual Studio verwendet haben, müssen Sie noch ein Update installieren, bevor Sie mit dem SQL Server 2017 arbeiten können. Sie erhalten sonst eine Fehlermeldung beim Versuch, sich mit dem aktuellen SQL Server zu verbinden.

Verwenden Sie den Menübefehl **Extras/Extensions und Updates...**, um verfügbare Updates anzuzeigen.

Bild 7.4 Aktualisierungen installieren

Bei der Installation von Visual Studio 2017 ist bereits eine LocalDB mit der Version 13 (SQL Server 2016) installiert worden. Haben Sie auf Ihrem Rechner zuvor auch andere Versionen des Visual Studio installiert, können weitere Versionen der LocalDB vorhanden sein. Wie auch bei den Vorversionen, wird mit einem zukünftigen Update auch die LocalDB in der aktuellsten Version 14 ergänzt werden. Jedenfalls werden alle verfügbaren Instanzen im SQL Server-Objekt-Explorer automatisch angezeigt. In Bild 7.3 erkennen Sie die als Erstes erwähnte Version im SQL Server-Objekt-Explorer über der gerade ergänzten Verbindung.

Projekteinstellungen können unter den Eigenschaften des Projektordners vorgenommen werden. Klicken Sie dazu im Projektmappen-Explorer den Ordner *Properties* doppelt an. Die Eigenschaften sind, wie in Visual Studio üblich, in verschiedene Kategorien unterteilt. Die für uns im Moment wichtigen Einstellungen finden wir unter *SQLCLR*. Hier werden der Name der Assembly und die Berechtigungsstufe – über beides werden wir später noch sprechen – eingestellt. Auch das Zielframework kann hier konfiguriert werden. Damit kann die diesbezügliche Auswahl, die Sie im Visual Studio 2017 beim Erstellen des Projekts getroffen haben, noch einmal verändert werden. Hier stellen Sie auch ein, ob Sie Visual Basic oder C# als Sprache für dieses Projekt einsetzen möchten.

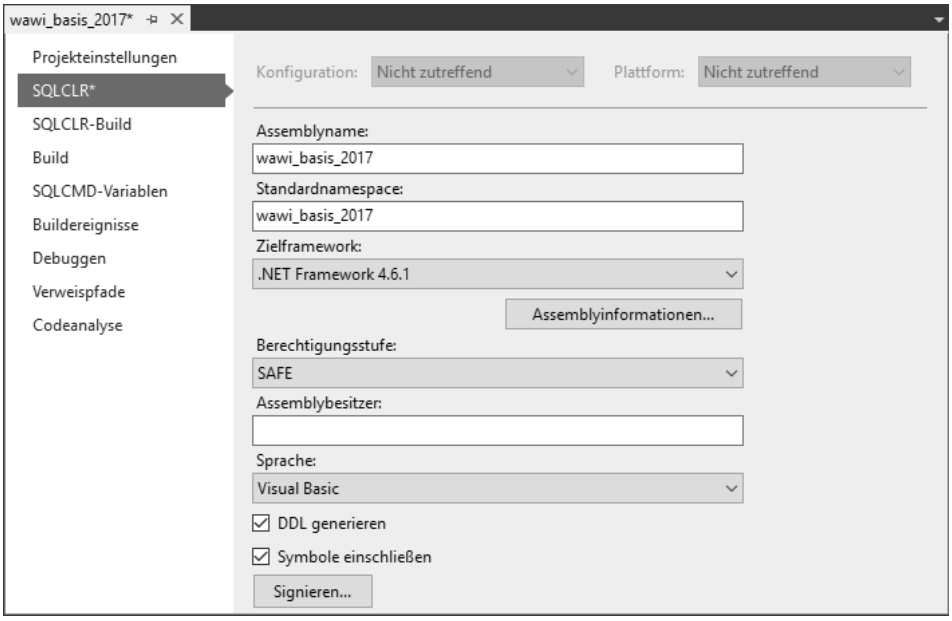

Bild 7.5 Projekteinstellungen

Die Zielplattform wird auf der Seite *Projekteinstellungen* ausgewählt. Dies kann ein SQL Server ab der Version 2005 oder Azure SQL Database sein. Die Einstellung passt aber nicht das verwendete Zielframework mit an, sondern wirkt sich in erster Linie auf das Deployment aus.

Bevor wir mit dem Programmieren für die CLR beginnen, beschäftigen wir uns mit dem Schaffen der nötigen Voraussetzungen auf dem SQL Server.

7.2 CLR-Aktivierung

Nach der Installation ist die CLR bei jeder SQL Server 2017-Edition zunächst deaktiviert. Sie müssen daher die CLR auf Ihrem Server aktivieren, bevor Sie die nachfolgenden Beispiele ausführen können.

PRAXISTIPP: Wenn Sie nicht wissen, ob CLR bei Ihrem Server schon aktiviert ist, können Sie dies aus dem Systemkatalog *sys.configurations* auslesen.

Verwenden Sie zum Beispiel folgende Anweisung:

```
SELECT name, value, description
FROM sys.configurations
WHERE name LIKE 'CLR%';
```

Sie erhalten:

```
name                 value   description
-------------------- ------- -----------------------------------------------
clr enabled          0       CLR user code execution enabled in the server
clr strict security  1       CLR strict security enabled in the server
```

HINWEIS: Mit dem SQL Server 2017 ist eine neue Variante für die Sicherheit von .NET-Code eingeführt worden. Diese hat den Namen *CLR strict security* und sie ist standardmäßig aktiviert. Per Update wird dieses Feature auch nachträglich beim SQL Server 2016 integriert, ist aber per default nicht aktiviert, um die Abwärtskompatibilität zu wahren. Auf dieses neue Feature werden wir etwas später in diesem Kapitel eingehen.

Diese Einstellung ist keine erweiterte Einstellung, kann also ohne vorherige Aktivierung der *Advanced Options* geändert werden.

Für die CLR-Aktivierung führen Sie die Systemprozedur `sp_configure` aus. Mit der Anweisung `RECONFIGURE` setzen Sie die zuvor gemachte Änderung sofort aktiv.

```
EXEC sp_configure 'clr enabled', 1;
GO
```

Ist die CLR-Aktivierung noch nicht erfolgt, erhalten Sie nach der Aktivierung, bevor Sie `RECONFIGURE` ausführen, folgende Meldung:

```
Die Konfigurationsoption 'clr enabled' wurde von 0 in 1 geändert. Führen Sie zum
Installieren die RECONFIGURE-Anweisung aus.
```

Prüfen wir nochmals die Einstellung, sehen wir, dass diese Eigenschaft nun den Wert 1 (wahr) aufweist, der verwendete Wert aber noch 0 (falsch) lautet.

```
SELECT name, value, value_in_use
FROM sys.configurations
WHERE name LIKE 'CLR%';
```

liefert:

```
name                 value           value_in_use
-------------------- --------------- ---------------
clr enabled          1               0
clr strict security  1               1
```

Da es sich bei dieser Einstellung um einen dynamischen Wert handelt, kann er sofort aktiviert werden. Führen Sie dazu bitte die Anweisung `RECONFIGURE` aus.

```
RECONFIGURE;
GO
```

Danach stimmen *value* und *values_in_use* wieder überein.

HINWEIS: Sie können alle nachfolgenden Beispiele erstellen, auch wenn die CLR noch nicht aktiviert ist. Aber spätestens bevor Sie sie ausführen möchten, muss die CLR-Integration aktiv sein.

Offensichtlich gilt die Aktivierung der CLR nur für benutzerdefinierten Code. Denn die in früheren Kapiteln beschriebenen Datentypen *geography*, *geometry* und *hierarchy_id* sind ja als .NET-Datentypen integriert, funktionieren aber auch, wenn die CLR deaktiviert ist.

ACHTUNG! Aufgrund der neuen *CLR strict security* beim SQL Server 2017 gelten höhere Anforderungen, um CLR-Code ausführen zu können. Da ich der Meinung bin, dass diese Neuerungen besser zu verstehen sind, wenn man sich schon mit CLR-Code befasst hat, möchte ich auf diese Neuerungen und ihre Auswirkungen erst am Ende dieses Kapitels eingehen.

Damit die Beispiele (vorerst) funktionieren können, müssen aber unbedingt folgende Voraussetzungen gegeben sein:

- Der Eigentümer der Datenbank muss Mitglied der Serverrolle *sysadmin* sein.
- Die Eigenschaft TRUSTWORTHY muss für die betroffene Datenbank auf ON gesetzt werden.

Der Eigentümer einer Datenbank kann auf der Seite *Dateien* des Dialogs *Datenbankeigenschaften* eingesehen und auch eingestellt werden (Bild 7.6).

Bild 7.6 Datenbankbesitzer anzeigen und ändern

Auslesen können Sie den Datenbankbesitzer auch mit dieser Anweisung:

```
SELECT d.name AS db, l.name AS besitzer
FROM sys.databases d
INNER JOIN sys.syslogins l  ON d.owner_sid = l.sid
WHERE d.name = 'wawi';
```

Einen neuen Datenbankbesitzer können Sie bei Bedarf mit folgender Anweisung festlegen und ihm gegebenenfalls noch die Mitgliedschaft bei den SQL Server-Administratoren erteilen:

```
ALTER AUTHORIZATION ON DATABASE::wawi TO [softelier\alina];
ALTER SERVER ROLE sysadmin ADD MEMBER [softelier\alina];
```

10 Sicherheit und Zugriffsberechtigungen

Meist ist der Grund für den Einsatz einer Client-Server-Datenbank, dass die Anzahl der Benutzer steigt. Auch das Größenwachstum der Datenbank und die zunehmende Wichtigkeit der gespeicherten Daten führen häufig zum Einsatz von Server-Datenbanken. Als Argument hierfür wird oft die Sicherheit der Daten ins Spiel gebracht. Doch Datensicherheit beschränkt sich nicht darauf, dass die Datenbank im Falle eines Ausfalls restlos wiederhergestellt werden kann. Oftmals sind es Benutzer, die erhebliche Schäden verursachen. Dabei sind oft nicht einmal böswillige Absichten der Grund – auch wenn diese nicht außer Acht gelassen werden sollten –, sondern viel öfter ist Fehlbedienung die Ursache für beschädigte Datenbestände.

Um solchen Problemen vorzubeugen, bietet der SQL Server Möglichkeiten, durch gezielte Rechtevergabe den einzelnen Benutzern innerhalb der Datenbank nur diejenigen Möglichkeiten zu eröffnen, die sie für ihre Arbeit benötigen.

HINWEIS: Die in diesem Kapitel beschriebenen Funktionalitäten stehen gleichermaßen bei der Express Edition wie bei den anderen Editionen zur Verfügung.

Sie erfahren im Folgenden, wie das Berechtigungssystem des SQL Servers aufgebaut ist und aus welchen Komponenten es besteht. Sie lernen den Weg von der Anmeldung am Server bis zum Zugriff auf die Daten kennen.

10.1 Authentifizierungsmodi – Anmeldungen und Benutzer

SQL Server unterstützt zwei Authentifizierungsmodi, um einen Zugriff auf den Server zu gewähren:

- Windows-Authentifizierung
- SQL Server-Authentifizierung

Je nach Modus werden dabei Windows-Benutzerkonten oder direkt auf dem SQL Server eingerichtete Konten verwendet.

HINWEIS: Bei SQL Server wird bereits beim Setup festgelegt, ob nur die *Windows-Authentifizierung* oder die *Windows-Authentifizierung und SQL Server-Authentifizierung* verfügbar sein soll. Letztere wird auch als *Gemischter Modus* bezeichnet. Sie können diese Einstellung im Bedarfsfall ändern. Wählen Sie dazu im Objekt-Explorer des Management Studios den gewünschten Server aus. Öffnen Sie im Kontextmenü mittels des Befehls **Eigenschaften** den Dialog *Servereigenschaften*. Auf der Seite *Sicherheit* können Sie dann diese Einstellung ändern.

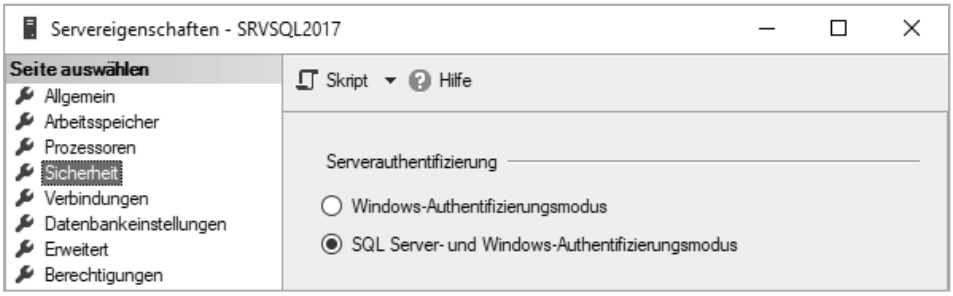

Bild 10.1 Serverauthentifizierung anzeigen und einstellen

ACHTUNG! Beachten Sie, dass bei einer Änderung dieser Einstellung der Serverdienst neu gestartet werden muss. Erst danach ist eine Änderung wirksam.

Diese Einstellung wird in der Registry über die erweiterte Systemprozedur *xp_instance_regwrite* vorgenommen. Der *LoginMode* ist unter dem Registry-Schlüssel *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft SQL Server\INSTANZNAME\MSSQLServer* zu finden. Der Wert 1 steht für die Windows-Authentifizierung; 2 für den gemischten Modus. Der vorletzte Teil des Schlüssels entspricht dem Namen der installierten Instanz. Bei einer SQL Server 2017-Standardinstanz ist zum Beispiel *MSSQL14.MSSQLSERVER* an dieser Position zu finden. Sie können die Einstellung auch direkt hier im Registrierungs-Editor vornehmen. Damit können Sie den Authentifizierungsmodus auf einem Server ändern, wenn einmal kein grafisches Werkzeug zu Verfügung steht. Es kann auch nötig sein, dass Sie den Modus zum Wiedereinrichten eines verlorenen Administratorzugriffs ändern müssen und dies eben mangels Zugriff über die SQL Server Tools nur auf diesem Wege möglich ist. Zu einem diesbezüglichen Beispiel kommen wir noch später in diesem Kapitel.

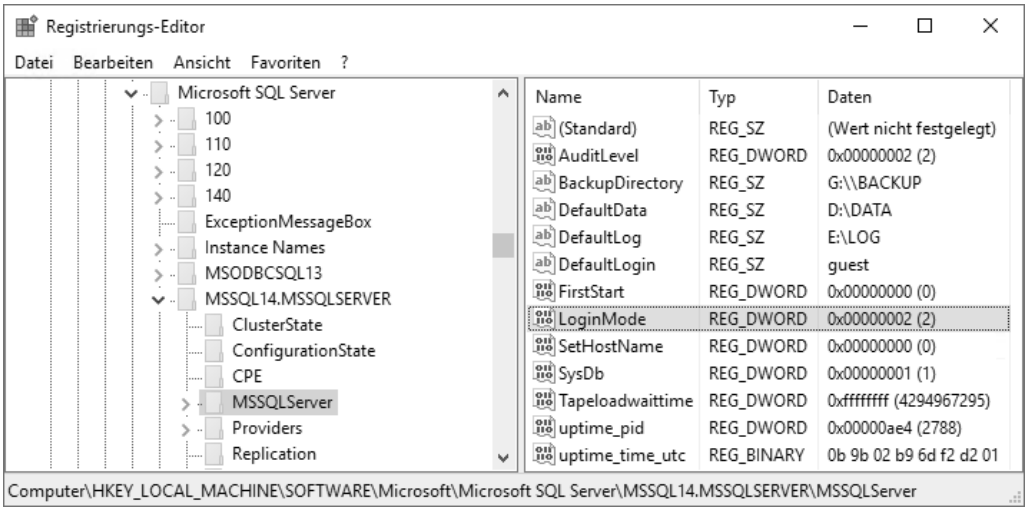

Bild 10.2 Serverauthentifizierung in der Registry

10.1.1 Windows-Authentifizierung

Bei der Windows-Authentifizierung übernimmt der SQL Server den Login von der Domänenanmeldung. Der Anwender muss daher kein separates Kennwort eingeben, wenn er auf den Datenbankserver zugreifen möchte. Die Anmeldung an der Betriebssystemdomäne reicht aus. Dies bedeutet jedoch nicht, dass jeder Benutzer, der sich an der Domäne des Betriebssystems anmelden kann, zugleich auch bereits Zugriff auf den Datenbankserver hat. Der Datenbankadministrator muss einem Betriebssystemkonto explizit das Zugriffsrecht auf den Datenbankserver gewähren.

PRAXISTIPP: Mittels der Windows-Authentifizierung kann nicht nur einem Domänenbenutzerkonto, sondern auch einem Gruppenkonto der Zugriff auf eine Datenbank gewährt werden. Dies kann in einfachen Anwendungsfällen, bei denen keine besondere Differenzierung der Anwender notwendig ist, die Administration vereinfachen. Zugleich ist dies der einzige Fall, bei dem ein Domänenkonto quasi automatisch Zugriffsberechtigungen auf einen SQL Server bekommt. Ist eine Gruppe einmal autorisiert worden, bekommen auch alle später im Active Directory angelegten Konten, welche die Mitgliedschaft in dieser Gruppe erhalten, sofort Zugriff auf den SQL Server.

10.1.2 Gemischter Modus

Der gemischte Modus verwendet sowohl die Windows-Authentifizierung als auch die SQL Server-Authentifizierung. Besitzt ein Anwender aufgrund seiner Betriebssystemanmeldung keine Zugriffsrechte auf den Datenbankserver, kann er sich mithilfe einer SQL Server-Anmeldung anmelden, sofern er eine solche besitzt.

HINWEIS: Clients, die nicht Mitglied der entsprechenden Domäne sind oder sein können, steht nur die SQL Server-Authentifizierung offen. So verwende ich in der Regel SQL Server-Authentifizierung, wenn ich mich von meinem eigenen Notebook auf einem Server bei einem meiner Kunden anmelde.

Dies gilt insbesondere für:

- den Zugriff über einen Webserver,
- andere Betriebssysteme als Windows-Betriebssysteme,
- einen gerouteten Zugriff über ein WAN,
- und auch bei der VNP-Verbindung mittels IPSEC kann in der Regel der Windows-Benutzer nicht verwendet werden.

Sollte einer dieser Fälle bei Ihnen vorliegen, so konfigurieren Sie für Ihren Server bitte unbedingt den gemischten Modus.

10.1.3 Anmeldung und Benutzer

Einer der wichtigsten Punkte beim Sicherheitskonzept des SQL Servers ist die Trennung von Anmeldung und Benutzer:

- **Anmeldung (Login)**: Mittels der Anmeldung erhält man Zugriff auf den Datenbankserver. Die Anmeldung erfolgt mit einer der beiden zuvor beschriebenen Authentifizierungsmethoden. Anmeldungen werden auf Serverebene erstellt und daher in der Systemdatenbank *master* gespeichert, ebenso wie Berechtigungen auf Serverebene. Anmeldungen wurden in früheren Versionen auch als *Systembenutzer* und *Sysuser* bezeichnet. Heute ist auch der Begriff *Server Principal* gebräuchlich.
- **Benutzer (User)**: Jede Anmeldung benötigt einen zugewiesenen Benutzer in einer Datenbank, um auf diese zugreifen zu können und dort Berechtigungen zu erhalten. Benutzer und deren Berechtigungen werden in der jeweiligen Datenbank gespeichert. In früheren Versionen wurden auch die Bezeichnungen *Datenbankbenutzer* und *User* verwendet. Der Begriff *Database Principal* kommt heutzutage auch zum Einsatz.

Die Trennung von Anmeldung und Benutzer hat vor allem zwei Gründe:

1. Durch die Trennung ist eine sinnvolle Integration der Windows-Authentifizierung in den SQL Server erst möglich.
2. Das Berechtigungssystem innerhalb einer Datenbank ist portabel, da es in der Datenbank selber gespeichert ist. Wird eine Datenbank transferiert, „wandern“ alle Benutzer und

Berechtigungen mit. Am Zielsystem müssen lediglich die Verbindungen zwischen den Anmeldungen und den Benutzern neu hergestellt werden.

HINWEIS: Mit der SQL Server-Version 2012 sind als Option sogenannte CONTAINED DATABASES eingeführt worden. Hier wird dieses zweistufige System durchbrochen, indem auf die Anmeldung verzichtet und ausschließlich ein Benutzer in der Datenbank benötigt wird. Ziel ist es, eine Form der Datenbank zu erhalten, die überhaupt vom umgebenden System unabhängig ist und daher ganz einfach von einem Server auf den anderen transferierbar wird. Daher wird auf alles verzichtet, was sich nicht in der Datenbank selber befindet. Und das sind eben auch Anmeldungen beziehungsweise Logins.

Die nachfolgende Abbildung zeigt das Standardschema eines Zugriffs auf Datenbanken.

- Die Anmeldung erfolgt mittels Windows- oder SQL Server-Authentifizierung.
- Einer Anmeldung können Serverrollen für Berechtigungen auf Serverebene zugewiesen werden.
- Einer Anmeldung können in einer Datenbank Benutzer zugewiesen werden. Erst dadurch erlangt ein angemeldeter Benutzer Zugriff auf die Datenbank.
- Innerhalb einer Datenbank werden den Benutzern in der Regel durch Rollenmitgliedschaften Berechtigungen erteilt.

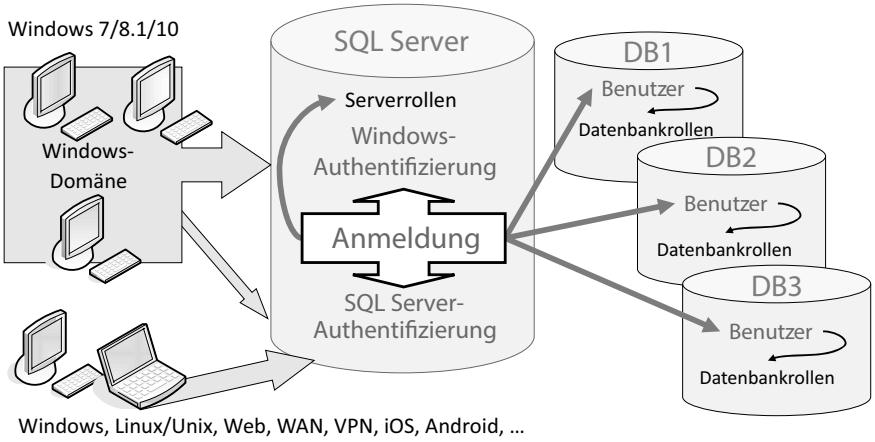

Bild 10.3 Schema eines Datenbankzugriffs

Benutzen Sie eine CONTAINED DATABASE, ändert sich diese Logik. Das Schema zeigt Bild 10.4. Der Zugriff erfolgt direkt auf die Datenbank, ein Zugriff auf andere Serverobjekte ist nicht möglich. Der Server wird quasi wie ein Tunnel passiert und direkt auf eine eigenständige Datenbank – so lautet die deutsche Übersetzung für diese Datenbankart – zugegriffen.

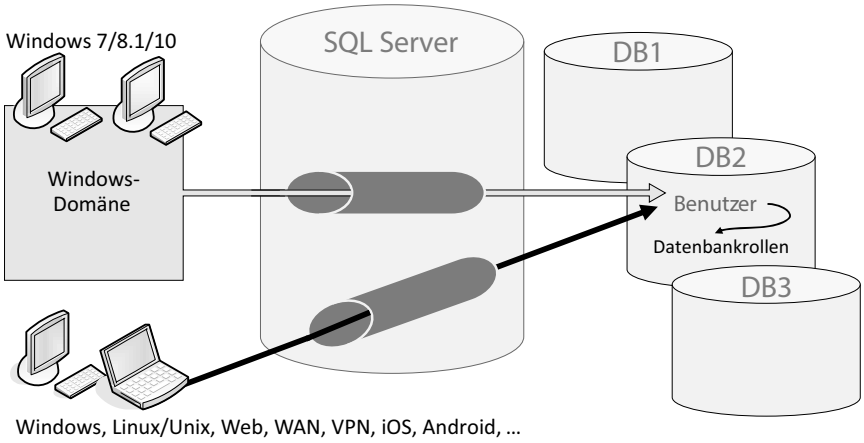

Bild 10.4 Zugriffsschema auf eine eigenständige Datenbank

Die einzelnen Schritte, um Zugriff auf eine Datenbank zu erhalten, werden in den folgenden Abschnitten genauer erläutert.

10.2 Berechtigungen

Jeder Benutzer benötigt Berechtigungen, um innerhalb einer Datenbank etwas tun zu können. Hierbei wird zwischen Objektberechtigungen und Anweisungsberechtigungen unterschieden.

Objektberechtigungen erlauben den Zugriff auf Objekte innerhalb der Datenbank. Diese Objekte sind Tabellen, Spalten einer Tabelle, Sichten, gespeicherte Prozeduren und benutzerdefinierte Funktionen. Für Tabellen und Sichten werden zum Beispiel die Rechte SELECT, INSERT, UPDATE, und DELETE erteilt. Für Tabellen kann auch das Recht REFERENCES (DRI) vergeben werden. Für gespeicherte Prozeduren und benutzerdefinierte Funktionen gibt es das Recht EXECUTE.

Anweisungsberechtigungen werden „gewöhnlichen“ Datenbankbenutzern in der Regel nicht gewährt. Sie beziehen sich nicht auf bestehende Objekte, sondern legen fest, wer Datenbankobjekte erstellen, verwalten und sichern darf. Anweisungsberechtigungen sind beispielsweise CREATE DATABASE, CREATE TABLE, CREATE VIEW und CREATE PROCEDURE. Das Recht BACKUP DATABASE wird benötigt, um eine Sicherung der Datenbank durchführen zu können.

12 SQL Server 2017 auf Linux

Was vor einigen Jahren noch absolut undenkbar gewesen ist, ist mittlerweile Realität. Microsofts Öffnung hin zu anderen Ökosystemen hat es möglich gemacht, dass nun erstmals ein SQL Server auf einer Nicht-Windows-Plattform zur Verfügung steht. Bereits die Ankündigung seitens Microsoft, den SQL Server auch für Linux auf den Markt zu bringen, hat einer kleinen Revolution geglichen. Da diese Ankündigung knapp vor der Veröffentlichung des *SQL Server 2016* gekommen ist, haben seinerzeit alle angenommen, dass diese Version im darauffolgenden Jahr als Linux-Version nachgeschoben werden würde. Erst ein knappes halbes Jahr nach dem Erscheinen des SQL Server 2016 ist mit den ersten Vorabversionen des *SQL Server vNext*, wie die Bezeichnung der neuen Version vor der Vergabe einer fixen Versionsnummer gelautet hat, klar geworden, dass es nochmals eine ganz neue Version des SQL Servers werden wird, die nun als *SQL Server 2017* vorliegt.

Implementierung des SQL Servers auf Linux

Beim SQL Server von Linux handelt es sich im Grundprinzip nicht um ein eigenes parallel entwickeltes Produkt, sondern um die Implementierung derselben Datenbank-Engine in einer anderen Umgebung. Dafür verantwortlich ist der sogenannte *SQL Server Platform Abstraction Layer*, kurz *SQLPAL* genannt. Dieser fungiert als Zwischenschicht zwischen dem SQL Server und dem Betriebssystem, die alle Aufrufe zwischen den beiden Systemen abwickelt. SQLPAL fordert zum Beispiel vom Betriebssystem den Speicher an und kümmert sich um das Lesen und Schreiben der Daten von den Festplatten (IO). In Bild 12.1sehen Sie den schematischen Ablauf dargestellt. In einem isolierten Softwareprozess läuft der SQL Server ident unter Windows und Linux. Dieser sendet seine Windows-Aufrufe an den SQLPAL, der dann ABI-Aufrufe (Application Binary Interface) an die Linux-Gasterweiterungen weiterleitet. SQLPAL ist gleichsam jener Teil, der sich bei der Windows- und der Linux-Version unterscheidet.

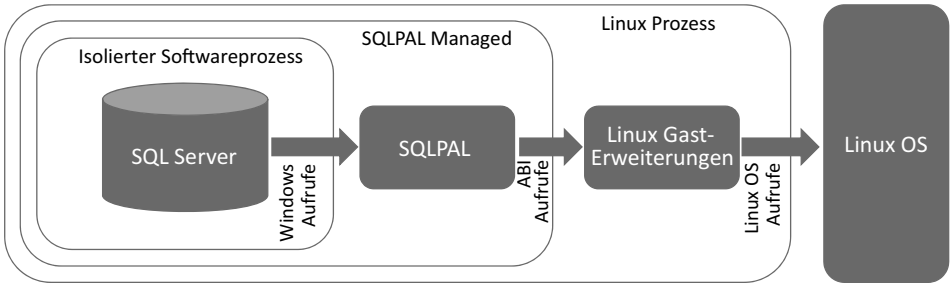

Bild 12.1 Interaktion SQL Server mit Linux über SQLPAL

Unterstützte Plattformen

Im Moment wird der SQL Server für folgende vier Linux-Plattformen unterstützt:

- Ubuntu 16.04 LTS mit EXT4 als Dateisystem.
- Red Hat Enterprise Linux 7.3 oder 7.4 Server, Workstation und Desktop. Als Dateisystem werden XFS und EXT4 unterstützt.
- SUSE Enterprise Linux Server v12 SP2 mit EXT4 als Dateisystem.
- Docker Engine 1.8+ auf Linux, Windows oder MacOS.

HINWEIS: Diese Auflistung entspricht dem momentan aktuellen Stand. Werfen Sie vor der Installation einen Blick auf diese Seite, vielleicht hat es in der Zwischenzeit ja schon eine Aktualisierung oder Erweiterung gegeben:

https://docs.microsoft.com/de-de/sql/linux/sql-server-linux-release-notes

Auf dieser Seite finden Sie zusätzlich eine Liste der unter Linux nicht unterstützen Features. Werfen Sie auch dazu einen Blick auf diese Seite, bevor Sie einen Einsatz planen. Von den in diesem Buch behandelten Features werden folgende (zurzeit noch) nicht unter Linux unterstützt:

- Filetable und Filestream (Kapitel 3)
- CLR-Assemblys, die nicht als SAFE (sicher) markiert sind (Kapitel 7)
- Change Data Capture (Kapitel 9)
- Windows-Authentifizierung für Verbindungsserver (Linked Server, Kapitel 10)
- SQL Server-Browser (Kapitel 9)

Als Clientwerkzeuge kommen neben dem *SQL Server Management Studio* (SSMS) ab der Version 17 auch die *SQL Server Data Tools* (SSDT) ab der Version 17 infrage. Auch wenn dies ein wenig verwirrend ist, trägt die Version 17 das Visual Studio 2015, während das Visual Studio 2017 unter der Version 15 geführt wird und damit neuer und natürlich verwendbar ist. Diese zwei Werkzeuge sind nur unter Windows verwendbar, das wird sich in Zukunft auch nicht ändern.

Wenn Sie ein Clientwerkzeug suchen, das auch unter Linux eingesetzt werden kann, kommen für Sie zusätzlich das *Visual Studio Code* mit den Erweiterungen für den SQL Server sowie das *SQL Operations Studio* infrage. Letzteres ist im Moment erst als Vorabversion verfügbar. Diese beiden sind aber für Windows, Linux und MacOS verfügbar.

Visual Studio Code finden Sie hier:

https://code.visualstudio.com/Download

SQL Operations Studio finden Sie hier:

https://docs.microsoft.com/de-de/sql/sql-operations-studio/download

In dieser automatisch übersetzten Seite wird das Werkzeug als *SQL-Vorgänge Studio* bezeichnet. Ich bezweifle, dass dies der offizielle deutsche Name bei der Veröffentlichung der finalen Version sein wird. Ersetzen Sie in diesem Link de-de durch en-us, gelangen Sie auf die originale englische Seite. Auch wenn ich in diesem deutschsprachigen Buch wenn möglich immer die Links zu deutschen Seiten angebe, verwende ich selber oftmals die originalen englischen Seiten. Dies hilft, wenn die maschinelle Übersetzung der deutschen Seite zu offensichtlich wird und die Gefahr gegeben ist, dass dadurch der Sinn von Aussagen nicht immer klar erkennbar ist.

HINWEIS: Dem *SQL Operations Studio* widme ich mich noch am Ende dieses Kapitels.

ACHTUNG! Begriffe und Vorgehensweisen, die in den vorangegangenen Kapiteln besprochen und erläutert worden sind, werden in diesem Kapitel verwendet, aber nicht nochmals im Detail erläutert. Lesen Sie bitte bei Bedarf die diesbezüglichen Erläuterungen an den entsprechenden Stellen. In diesem Kapitel wird davon ausgegangen, dass die Bestandteile des SQL Servers bekannt sind und Sie zumindest mit den Grundbegriffen von SQL und Zugriffsberechtigungen vertraut sind.

12.1 Installation des SQL Servers

Als eine der am weitesten verbreiteten Distributionen nutze ich Ubuntu für meine Erörterungen in diesem Kapitel. Sie können dazu eine Ubuntu-Maschine in der Azure-Cloud verwenden oder eine lokale Installation vornehmen. Ich installiere für dieses Beispiel Ubuntu in einer virtuellen Maschine auf meinem Server.

Die Installationsdateien für Ubuntu 16.04 LTS finden Sie unter dieser Adresse:

https://www.ubuntu.com/download/server

Wenn Sie Ubuntu installiert haben, geht die Installation von SQL Server in wenigen Schritten sehr einfach von der Hand. Der erste Schritt ist, den GPG Key (GNU Privacy Guard) ins