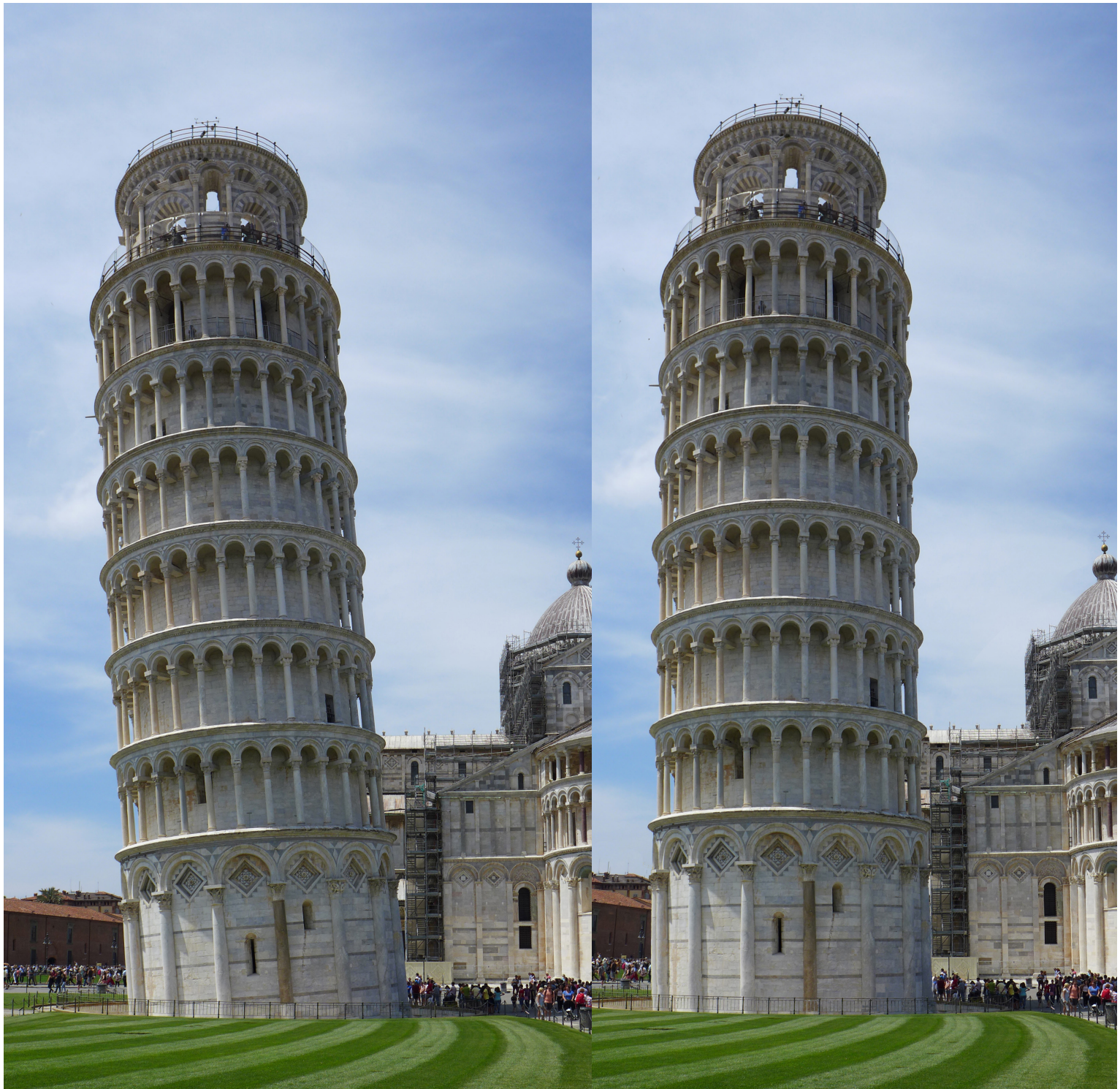




06/2018 Schiefer Turm von Pisa: Ist und Soll

SQL Services

Das weltweit wohl berühmteste Beispiel für einen Fehlschlag mangels tragfähiger Fundamentierung steht in Pisa. Zum Glück für alle Beteiligten kam es nicht zum Einsturz. Auch der Dom neben dem Campanile weist enorme Abweichungen von der Vertikalen auf. Aus heutiger Sicht ist zu bewundern, wie die Baumeister trotz widrigem Baugrund ihre Vorhaben mit den technischen Möglichkeiten des Mittelalters erfolgreich zu Ende führten. Allerdings gibt die Bauzeit von rund 200 Jahren mit extrem langen Unterbrechungen zu denken. Sieben bis acht Generationen haben an dem Turm gebaut oder darauf gewartet, ob er einstürzt.

In solchen historischen Dimensionen ist die Informatik bei weitem noch nicht angekommen, aber Probleme mit schlechten Fundamenten kennt sie ebenfalls reichlich. Die Evolution der Betriebssysteme zeigt zudem, dass es für sie auch nach Jahrzehnten immer noch ein erhebliches Verbesserungspotential gibt. Um so wichtiger ist es für die Anwendungsentwicklung, über längere Zeit und mehrere Betriebssystemgenerationen hinweg Stabilität zu wahren. Gern wird dafür der Weg beschritten, Komplexität und Fluktuation in Software auszulagern, die einerseits als Puffer gegen Veränderungen der Basissoftware dient und andererseits die Programmierung erleichtert. Klassenbibliotheken und Unterprogrammpakete erfüllen auch diese Zwecke. Die im folgenden vorgestellten SQL Services sind zugleich als festes Fundament für die Nutzung von Java SQL wie auch als massive Vereinfachung für den Umgang mit relationalen Tabellen intendiert.

10

Die SQL Services orientieren sich inhaltlich und methodisch an den File Services in den Bänden 1 und 2. Auch hier geht es darum, die Zugriffsdetails vor dem Anwendungsprogramm zu verbergen und sie so zu generalisieren, dass sie für verschiedenste Anwendungszwecke nutzbar sind. Allerdings ist Java SQL dermaßen mächtig, dass die Services nur einen Grundstock bilden können. Dieser kann zwar problemlos erweitert werden, bleibt aber dennoch weit davon entfernt, die enorme Fülle der möglichen RDBMS-Interaktionen vollständig abzubilden. Das ist auch gar nicht nötig, denn der hier gezeigte – durchaus beträchtliche – Funktionsumfang der SQL Services deckt erfahrungsgemäß die Grundformen der Datenbankprogrammierung vollständig ab.

Die Analogie mit den File Services trägt zwar weit und prägt sich auch in den Codestrukturen der SQL Services aus, aber das sollte nicht davon ablenken, dass Datenbankzugriffe sich in vielerlei Hinsicht von sequentiellen Zugriffen massiv unterscheiden. Letztere sind unvergleichlich viel simpler gestrickt, was zur Folge hatte, dass die Phantasie der Designer(innen) und Programmierer(innen) lange Zeit durch die dadurch bedingten Restriktionen gelähmt wurde. Relationale Datenbanken und ihre Vielfalt möglicher Verwendungsformen bieten dagegen ein tendenziell nicht ausschöpfbares Potential für kreative Ideen und elegante Lösungen. Das macht den Umgang damit so interessant.

Es gibt allerdings auch eine Kehrseite: Wer dieses Potential falsch nutzt, kann außerordentlich schlechte Leistungen im inhaltlichen wie im maschinellen Sinn hervorbringen. Das Schlagwort „SQL-Optimierung“ vermag nur unzureichend zu benennen, welchen Herausforderungen sich RDBMS-Experten gegenübersehen, die aus einer vermurksten oder „zu klein gedachten“ Konstruktion dennoch eine halbwegs brauchbare Performance herauskitzeln sollen. So etwas kann auch massiv schief gehen und sogar die Existenz großer Firmen bedrohen. Um so wichtiger ist es, die Chancen, aber auch die Risiken der RDBMS klar zu erkennen und sowohl die Chancen sinnvoll zu nutzen als auch immer die Risiken im Auge zu behalten. Dies erfordert nicht nur eine intensive Ausbildung zur / zum professionellen Anwendungsentwickler(in), sondern auch eine enge Kooperation mit den Datenbankadministratoren, die von beiden Seiten erlernt werden muss. Denn: Oftmals sind beide Seiten nicht nur räumlich und organisatorisch voneinander getrennt, sondern sind auch durch unsinnige Vorgaben seitens des Managements an der nötigen intensiven Kooperation gehindert.

Zurück zum eigentlichen Thema: Wie sind die SQL Services konzipiert?

10.1 Schichtenmodell und Klassendiagramm der SQL Services

Die SQL Services schieben sich zwischen das Anwendungsprogramm und die Java SQL-Klassen, erlauben aber zusätzlich, dass das Anwendungsprogramm sich dem – von den Services erzeugten – Connection Handle sowie den Prepared Statement und Result Set Handles direkt bedient. Die folgende Abbildung zeigt vereinfacht, wie die Schichten bis „hinab“ zu MySQL interagieren. Eine der Vereinfachungen besteht darin, dass die Umsetzung des Java Bytecode in die nativen Befehle des jeweiligen Prozessors mit Hilfe der Java Virtual Machine ausgeblendet wird. Ausserdem besteht TCP/IP aus einer Schichtenarchitektur, und auch MySQL ist eine komplexe Maschine. All diese Details sind jedoch für die Platzierung der SQL Services irrelevant.

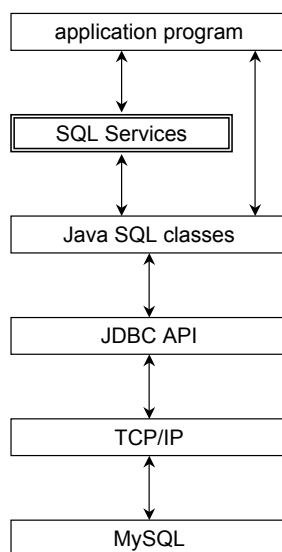


Abbildung 10.1: Schichtenmodell mit den SQL Services

Die „Abbildung 10.2: Klassendiagramm der SQL Services“ auf Seite 286 zeigt, dass auch diese Services intern eine Schichtenstruktur aufweisen. Folgende Aspekte waren für die Entwicklung maßgeblich:

- Das Anwendungsprogramm ruft Methoden von `SQL_Util` auf, dessen Klasse als „`public final class`“ deklariert ist. Somit ist sie als einzige im Package „von außen“ erreichbar, denn die anderen Klassen gelten als „`package private`“.
- `SQL_Util` nimmt sämtliche Zugriffsoperationen mit Hilfe der Java SQL-Klassen vor, meldet die Ergebnisse zurück und behandelt auch Exceptions. Es stützt sich dabei auf die Klasse `SQL_Register`.
- Die Anordnung der Methoden in `SQL_Util` folgt einem Schema zunehmender Detailliertheit:
 - Zu Beginn stehen Methoden zur Eröffnung und Kontrolle von Connections zu MySQL.
 - Die Methode `compileSQL_statement()` macht die vom Programm verwendeten SQL Statements für MySQL nutzbar. Hier entstehen die „Prepared Statement“ Handles.
 - Danach folgen Setter-Methoden, um Parameterwerte in diesen Statements mit Inhalten zu füllen. Diese Parameterwerte sind durch Fragezeichen repräsentiert, wofür im Kapitel „3.8.12 Updates in TS3“ auf Seite 46 ein erstes Beispiel zu finden ist.

- Es sind nur vier Setter-Methoden für die gängigsten Datentypen vorhanden. Wenn weitere Datentypen benötigt werden, können sie anhand dieser Beispielmethode leicht „nachgerüstet“ werden.
 - `executeUpdate()` und `closePreparedStatement()` gehören noch zur Prepared Statement-Ebene. Danach folgen bis `closeCursor()` die Aufrufe der Result Set-Ebene für die Cursor-Verarbeitung.
 - Auf der Zeilenebene operieren die Check-, Get-, Set- und Update-Methoden. Auch hier können weitere Datentypen mittels zusätzlicher Methoden ergänzt werden.
 - Danach folgen interne Methoden sowie eine Kollektion extrem einfacher Getter- und Setter-Methoden.
- Die Komplexität der Java SQL-Schnittstelle bleibt dadurch dem Anwendungsprogramm weitgehend verborgen. Es kommuniziert allein mit der Klasse `SQL_Util` anhand der SQL IDs sowie weiterer Parameter, falls nötig.
- Die Klassen `SQL_Register` und `SQL_Entry` dienen der Verwaltung der Einträge im SQL Array und dem Schutz vor Irrtümern im Anwendungsprogramm.

Nun dürfte deutlich geworden sein, dass die SQL Services inwendig eine deutlich höhere Komplexität als die File Services besitzen. Der Lohn dafür besteht in einer starken Vereinfachung der Anwendungsprogramme.

Bei der Entwicklung solcher Schnittstellen gibt es natürlich immer Optionen, zwischen denen, für die oder gegen die man sich entscheiden muss. So könnte tendenziell ein nicht mehr benötigter Array-Eintrag abschließend entfernt und mit einem anderen SQL Statement neu belegt werden. Damit ist allerdings ein beträchtliches Irrtumspotential verbunden. Deshalb bieten die SQL Services diese Funktionalität nicht an. Erst beim expliziten Schließen der Connection werden auch die Array-Einträge entfernt. Es ist aber möglich, einen Cursor zu schließen und später erneut zu öffnen, um beispielsweise eine andere Ergebnismenge mittels Platzhaltern in der Where-Klausel auszuwählen. Dann wird auch der „row count“ zurückgesetzt, muss also vor dem Schließen des Cursor abgerufen werden.

Beim Vergleich mit den File Services fällt auf, dass es nur einen zentralen SQL Array gibt, aber zwei Arrays für die `BufferedReader`- und `BufferedWriter`-Klassenelemente. Das liegt daran, dass die Java-Containerstrukturen hinsichtlich der Spaltenzuordnungen in Arrays typenrein sein müssen. Die Einträge im SQL Array heißen SQL Entry. Jeder Entry entspricht einer Zeile in der Matrix im folgenden Unterkapitel „10.2 Aufbau und Nutzung des SQL Array“ auf Seite 287 ff. Da der SQL Array „allein“ ist, entfällt auch die Vererbungskonstruktion der File Services mit `NameAndCount_AC`.

Analog zu den File Services gibt jedes Nutzerprogramm beim Aufbauen der Connection vor, wieviele Einträge der SQL Array führen soll. Auch diverse andere Aufrufe erinnern an die File Services und müssen daher hier nicht erläutert werden. Jedes Nutzerprogramm richtet seine Aufrufe an die statische Klasse `SQL_Util`, die keine eigenen Variablen besitzt und sich somit als Kollektion von Zugriffsmethoden darstellt. Nach erfolgreicher Parameterprüfung, sowie gegebenenfalls nach der Ausführung der verlangten MySQL-Interaktionen beauftragen diese Methoden in etlichen Fällen die Klasse `SQL_Register`. Diese Klasse besitzt eigene Variable und führt den SQL Array. Die Methoden von `SQL_Register` sind „package private“, können also von Nutzerprogrammen nicht aufgerufen werden.

Im Gegensatz zu den File Services ist kein Automatismus mit Abbruch nach Erreichen einer Fehlerschranke vorgesehen. Nur das aufrufende Anwendungsprogramm entscheidet über die Fehlerreaktion. In einem Programm, das sowohl die File Services als auch die SQL Services verwendet, käme es sonst zu einer unerwünschten Verflechtung solcher Reaktionen. Wenn ein Nutzerprogramm seinen Lauf abbricht oder abbrechen lässt, können Cursors offen sein, die dann implizit geschlossen werden. Dasselbe passiert mit der Connection zu MySQL. Das ist zwar eine ziemlich ruppige Verabschiedung, wird aber von MySQL toleriert.

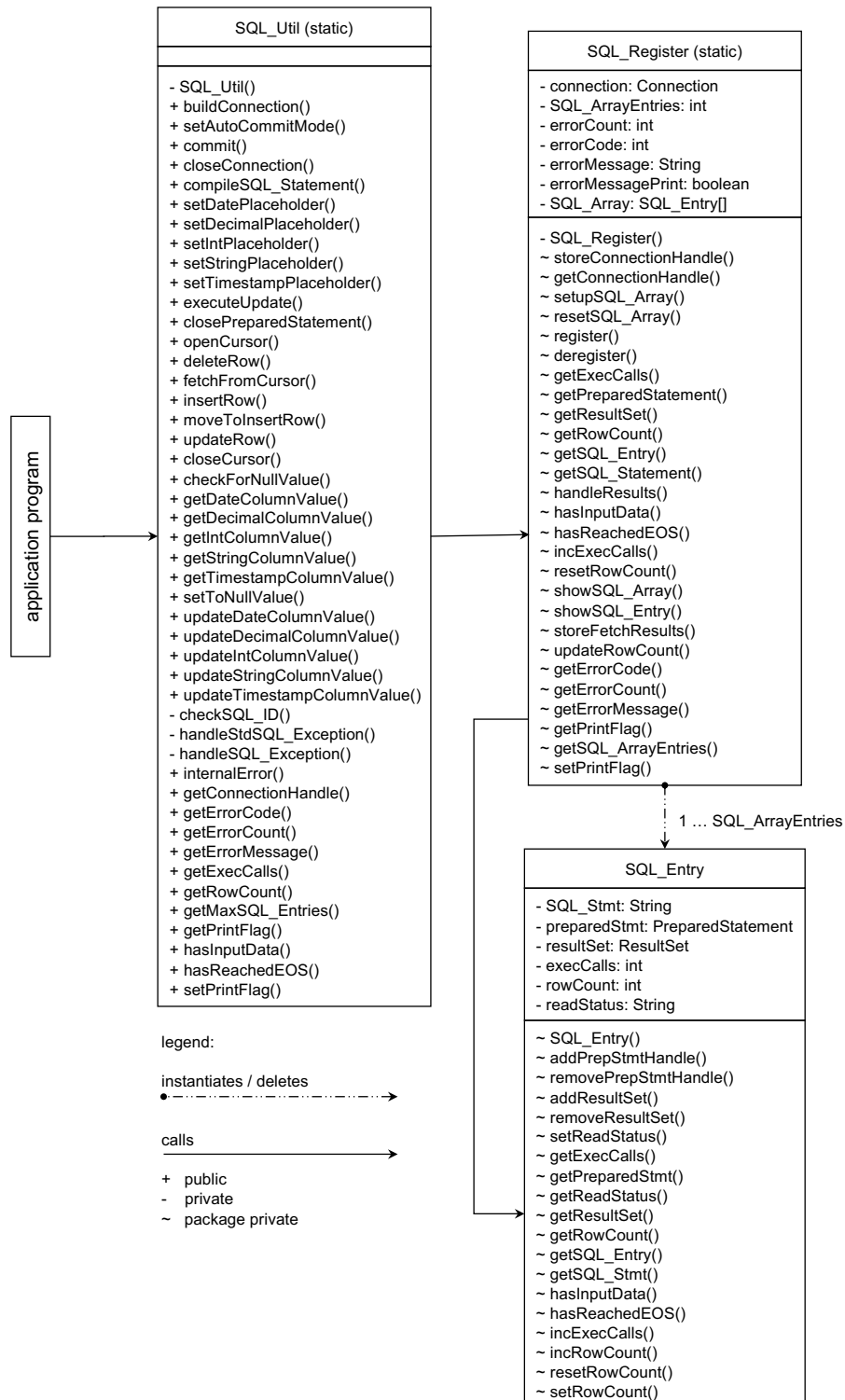


Abbildung 10.2: Klassendiagramm der SQL Services

10.2 Aufbau und Nutzung des SQL Array

Der SQL Array bildet den Kern der SQL Services. Er versammelt alle wichtigen Informationen zu den Interaktionen der Services mit dem jeweiligen RDBMS und ist – am Beispiel von „SQL Example 0“ – wie folgt strukturiert:

SQL ID	SQL Statement	Handle „Prepared Statement“	Handle „Result Set“	exec calls	row count
0	SELECT Tuple_Key, Tuple_Value FROM TS1_0_View	ps handle 0	rs handle 0	0	0
1	SELECT Tuple_Key, Tuple_Value FROM TS2_0_View	ps handle1	rs handle1	0	0
2	INSERT INTO TS3_0_Table (Tuple_Key, Tuple_Value) VALUES (?, ?)	ps handle2		0	0
3	UPDATE TS3_0_Table SET Tuple_Value = Tuple_Value + ? WHERE Tuple_Key = ?	ps handle3		0	0

Die Inhalte des Array zeigen den Stand nach den Op. 27,29,30,31,32,1,2 aus den Kapiteln „3.8.6 Operationen, Kontrollen und Procedure Statements“ auf Seite 40 und „3.8.16 Fertigstellung der Programmstruktur mit Operationen“ auf Seite 48; die Op. 3,4,5,7 ändern daran nichts. Die Insert und Update Statements haben wegen der für sie verwendeten Platzhalter-Methode kein ihnen zugeordnetes „rs handle“. In der Spalte „exec calls“ werden die Aufrufe von `executeUpdate()` und `executeQuery()` gezählt. Die Spalte „row count“ enthält bei Cursor-Zugriffen die Anzahl bislang gelesener Zeilen, andernfalls die vom letzten `executeUpdate()`-Aufruf rückgemeldete Anzahl. Diese ist bei der Op. 18 entweder 1 (Treffer) oder 0 (Zeile existiert nicht). Nach den Op. 5 und 7 wird erst später festgestellt, ob überhaupt eine Zeile eingelesen wurde; daher die obige Bemerkung. Diese Verzögerung erfordert, eine weitere Spalte im Array aufzunehmen, nämlich den „read status“. Dieser erhält anfangs den Wert " ". Nach jedem FETCH wird er entweder auf "D" (für „data“) oder auf "E" (für „end“) gesetzt.

Jeder SQL ID entspricht genau ein SQL Statement, das bei MySQL möglicherweise aus mehreren, durch Semikolon getrennten Anweisungen besteht. Davon wird hier aber kein Gebrauch gemacht. Es kommt nicht darauf an, ob dieses Statement bisher nicht benutzt wurde, noch aktiv ist, oder seinen Zweck bereits erfüllt hat.

Die leeren (!) Array-Einträge werden beim Übersetzen der Statements angelegt und initialisiert. Die Spalte „SQL Statement“ ist daher danach stets belegt. Nach erfolgreichem Compile wird von MySQL ein „Prepared Statement Handle“ rückgemeldet und in den Array in derjenigen Zeile eingetragen, welche der jeweiligen SQL ID zugeordnet ist. Wenn das Statement ein Cursor Select ist, folgt das Öffnen des Cursor und die Rückmeldung eines „Result Set Handle“, das ebenfalls in den Array aufgenommen wird. Das heißt: Jede Zeile der Matrix wird Schritt für Schritt aufgebaut.

Die folgenden Sequenzdiagramme veranschaulichen exemplarisch die Abläufe von der Op. 27 an.

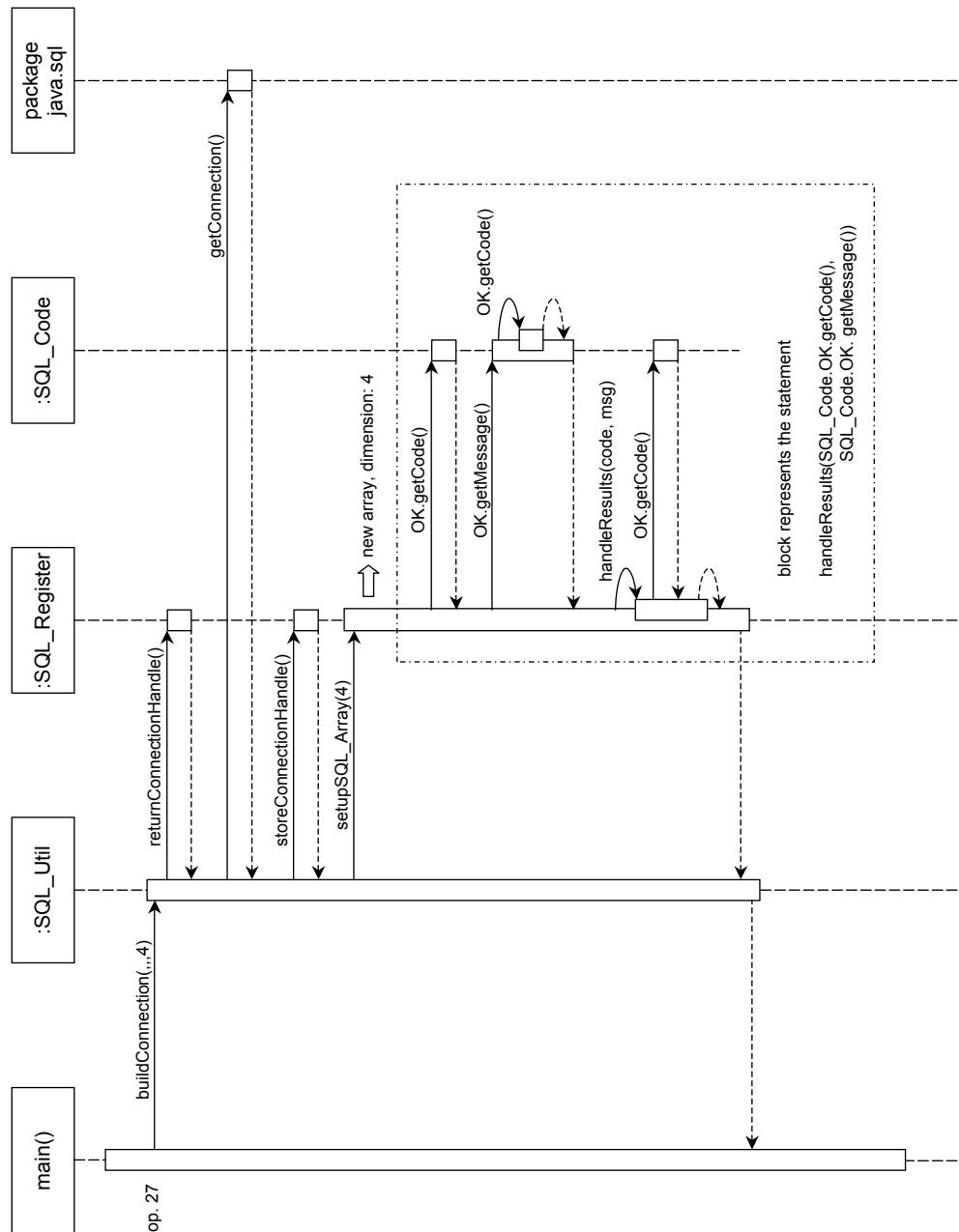
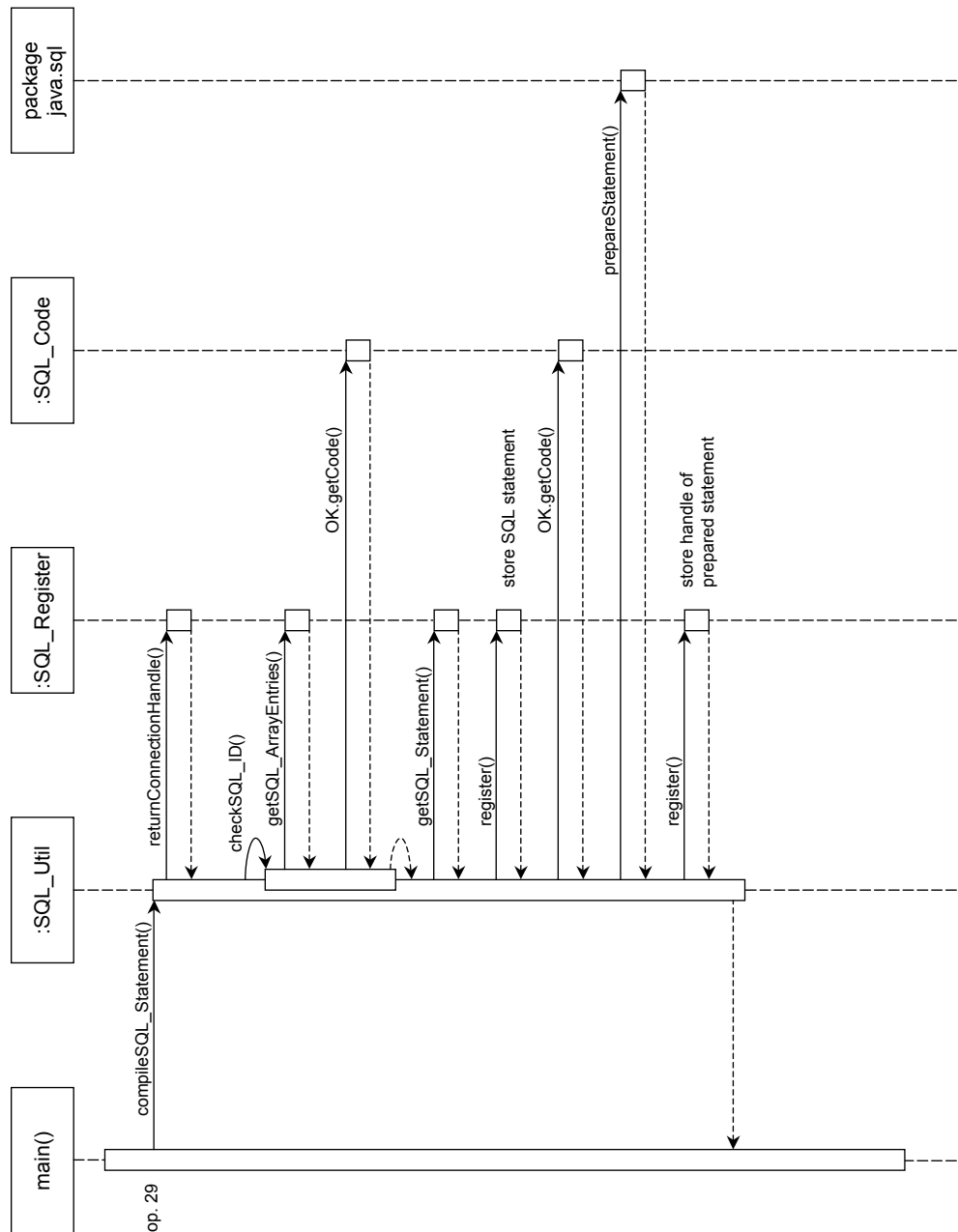


Abbildung 10.3: Verbindungsaufbau zu MySQL und Anlegen des SQL Array

Das Anwendungsprogramm wird durch die SQL Services komplett von der eigentlichen RDBMS-Schnittstelle abgeschirmt, wie auch die nachfolgenden Sequenzdiagramme zeigen. Dem hier gezeigten Verbindungsaufbau entspricht der geordnete Verbindungsabbau am Lauf-Ende.

Nach dem Verbindungsaufbau wird das erste SQL Statement zum Kompilieren übergeben (Op. 29).



10

Abbildung 10.4: Kompilieren des TS1 SELECT

Dieser mehrschrittige Vorgang trägt nach vorbereitenden Prüfungen das SQL Statement in die – davor allokierte – erste Array-Zeile ein, lässt es kompilieren und speichert dort das rückgemeldete Prepared Statement Handle. Die Op. 30,31 und 32 laufen ebenso ab.

Der Cursor für das TS1 SELECT Statement kann nun geöffnet und die erste Zeile gelesen werden.

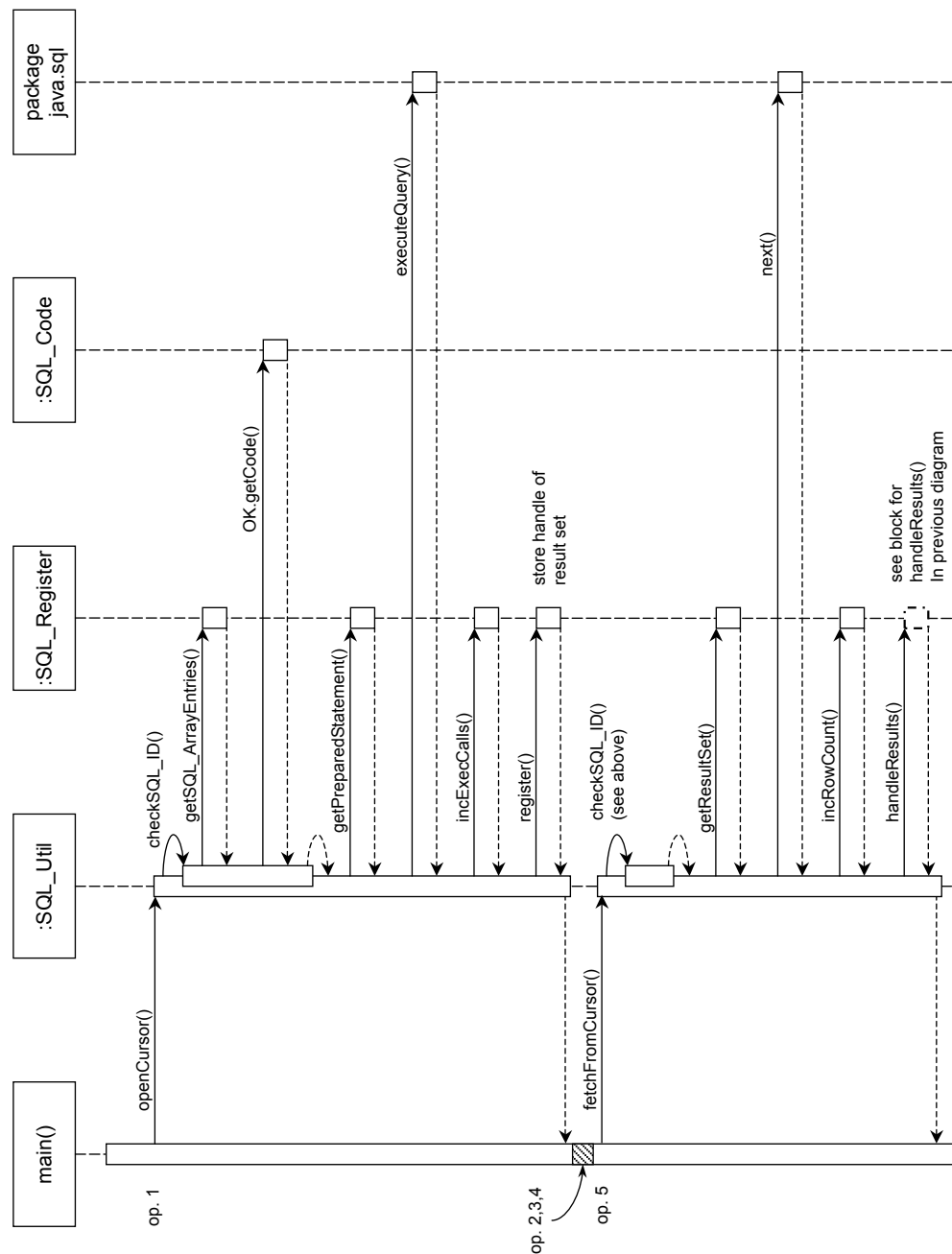


Abbildung 10.5: Öffnen des TS1 Cursor und Lesen der ersten TS1-Zeile

Die Op. 2 läuft völlig analog zur Op. 1 ab; dasselbe gilt für die Op. 7 bezüglich der Op. 5. Die Op. 3 und 4 sind für die Grafik irrelevant.