HOCHSCHULE MERSEBURG
University of Applied Sciences

Bachelorarbeit

Entwurf und Bewertung eines SQL-Tutoriums zur Verwendung im Bachelor-Studium

Verfasser: Bianca Mayer
Matrikelnummer: 24644

Erstgutachter/in: Prof. Dr. Ronny Weinkauf
Zweitgutachter/in: Ulrich Borchert

Eingereicht am: 25.06.2025

Inhaltsverzeichnis

Abbildungsverzeichnis III

Kurzzusammenfassung (en) IV

Kurzzusammenfassung (de) V

I Einleitung und Problemstellung 1

1 Einleitung 1

2 Ziele der Arbeit 2

3 Aufbau der Arbeit 2

II Bestandsaufnahme 3

4 Relationale Datenbanken und SQL 3

4.1 Historischer Hintergrund 3

4.2 Gegenwärtige Relevanz 5

4.3 Grundlagen des relationalen Modells 7

4.4 Sprachaufbau und Funktionsweise 10

4.4.1 Data Definition Language 12

4.4.2 Data Manipulation Language 15

4.4.3 Data Control Language 19

5 Integration von SQL in der Hochschulausbildung 20

5.1 Allgemeine Form der SQL-Ausbildung 20

5.2 Häufige Schwierigkeiten im Erlernen von SQL 22

5.3 Rahmenbedingungen des betrachteten SQL Tutorials 23

III Entwurf des neuen Tutorials 25

6 Analyse des bestehenden Tutorials 25

7 Identifizierung der Schwachstellen 40

8 Entwicklung der neuen Tutorialversion 40

IV Evaluierung 49

9 Beobachtungen aus dem Praxis-Test 49

10 Analyse des studentischen Feedbacks 50

V Fazit und zukünftige Möglichkeiten 53

11 Zukünftige Beobachtungsstellen 53

12 Weitere Verbesserungsansätze 53

13 Zusammenfassung 54

Literaturverzeichnis 56

A Anhang 1 59

Abbildungsverzeichnis

Abbildung 1: Das Logo von Oracle SQL Developer, das IBM-Symbol für Magnetspeicher und ein Dysan CE Pack....3
Abbildung 2: Stackoverflow Developer Survey 2024, Abschnitt Databases (Ausschnitt)....6
Abbildung 3: DB-Engines Ranking (Ausschnitt)....7
Abbildung 4: Eine Übung auf SQLBolt mit dynamischem Feedback....21
Abbildung 5: Die Auflistung der Datentypen im Ilias-Kurs....26
Abbildung 6: Die Format-Erklärung für den DATE-Datentyp....28
Abbildung 7: Die Operatoren-Tabelle....29
Abbildung 8: Die Folie zur Dual Tabelle....29
Abbildung 9: Handschriftliches ERD der Beispiels-Datenbank....30
Abbildung 10: Das auf Folie 5.2.9 verwendete ERD....32
Abbildung 11: Folie 5.2.25 Die Datendefinitionssprache DDL....33
Abbildung 12: Folie 5.2.37 DROP-Anweisung....34
Abbildung 13: Folie 5.3.3 - DML: INSERT....35
Abbildung 14: Folie 5.3.13 Übung SELECT Bibliothek....36
Abbildung 15: Folie 5.4.4 Beispiel Pool-User....38
Abbildung 16: Grafische Darstellung eines "Dirty Reads", zu sehen auf Folie 6.1.5....39
Abbildung 17: Ursprüngliche Version des "dirty read" Diagramms....41
Abbildung 18: Vereinfachte und höher auflösende Version des "dirty read" Diagramms....41
Abbildung 19: Das alte zwei-Phasen-Sperrprotokoll Diagramm....41
Abbildung 20: Die neue Version der selben Grafik....42
Abbildung 21: Die alte Folie "RENAME TABLE Anweisung"....43
Abbildung 22: Die angepasste Version der "RENAME TABLE Anweisung" Folie....43
Abbildung 23: ERD der Datenbank "Bibliothek"....44
Abbildung 24: ERD der Datenbank "Mitarbeiter"....45
Abbildung 25: Das erweiterte ERD der Mitarbeiter-Datenbank....47
Abbildung 26: Bewertung der Aussagen in Frage 5....51

Kurzzusammenfassung (en)

A robust understanding of the Structured Query Language SQL is, due to the widespread adoption and powerful capabilities of the language, an important aspect of many developers' toolkits. Accordingly, it has a central place in database curriculums across higher education.

This thesis examines one such course currently in use at the Hochschule Merseburg in Saxony-Anhalt and aims to improve its overall quality. The course in question aims to introduce students to a foundational knowledge of SQL that will be built on in further practical exercises. To guarantee the quality of this education, the SQL tutorial (a three hour long segment of the overall course) will be revised and tested.

After an introduction to both the relational model and its practical application through SQL, the course material is critically examined and analysed to identify key weakpoints. While the tutorial proves more then adequate contentwise and structurally, several problems are found regarding the presentation which cause unnecessary complications for the understanding of these contents. After remedying these issues, the tutorial is tested in the summer semester of 2025 to evaluate the practicality of the revised material and gather important feedback from students.

Finally, the results of the practical test are analysed and further possibilities for future examination and improvements to the course are presented.

Kurzzusammenfassung (de)

Ein robustes Verständnis der Structured Query Language SQL ist, aufgrund der weiten Verbreitung und Leistungsfähigkeit der Sprache ein wichtiger Anteil im Werkzeugkasten vieler Developer. In den Lehrplänen der Hochschulausbildung nimmt es einen entsprechend wichtigen Platz ein.

Diese Arbeit betrachtet einen solchen Kurs, der gegenwärtig an der Hochschule Merseburg in Sachsen-Anhalt eingesetzt wird. Das Ziel dieses Kurses ist es, Studierenden ein grundlegendes Verständnis von SQL zu vermitteln, auf dem weitere praktische Übungen aufbauen können. Um eine möglichst hohe Qualität dieser Ausbildung zu gewährleisten, wird das SQL-Tutorial (ein dreistündiges Segment des übergreifenden Datenbank-Kurses) überarbeitet.

Nach einer Einführung in sowohl die Theorie des relationale Modell als auch seine praktische Verwendung mittels SQL wird das Kursmaterial kritisch betrachtet, um Schwachstellen zu identifizieren. Wärend sich das Tutorial inhaltlich und strukturell als mehr als adequat herausstellt, gibt es einige Probleme in deren Präsentation, die das Verständnis der Inhalte unnötig erschweren. Nach Behebung dieser Kritikpunkte wird das Tutorial im Sommersemester 2025 getestet, um die Praxistauglichkeit des erarbeiteten Materials zu bewerten und wichtige Rückmeldungen von Seiten der Studierenden zu erhalten.

Schließlich werden die Ergebnisse des praktischen Tests analysiert und weitere Möglichkeiten für zukünftige Beobachtungen und Verbesserungen des Kurses präsentiert.

I Einleitung und Problemstellung

1 Einleitung

Die Rolle, die Daten in der Organisation und Verwaltung der modernen Welt spielen, ist schwer zu überschätzen. Selbst eine so geringfügige Geschäftsinteraktion wie der Kauf eines Paars Schuhe erzeugt Daten, die gespeichert und verwaltet werden müssen. Ein Online-Handel muss einen Datensatz mit dem Namen, Preis und der Beschreibung seiner Produkte verwalten und dem Kunden zugänglich machen, damit dieser sich für den Kauf entscheiden kann, und für den Kaufprozess werden die Adresse und Bankdaten des Kunden benötigt, private Daten, die sicher verwahrt werden müssen. Selbst, wenn der Kauf nicht online, sondern vor Ort im Einzelhandel stattfindet, ist gegebenenfalls ein Kundenkonto vorhanden, und bei der bargeldlosen Zahlung bleibt die Verbindung mit den Bankdaten des Kunden nicht aus. Selbst wenn der Kunde kein Kundenkonto hat und bar bezahlt, muss die Transaktion zur Abrechnung im Kassensystems des Geschäfts gespeichert werden.

Die Patientendaten einer Klinik, die Abfahrtszeiten des öffentlichen Nahverkehrs, der Stundenplan einer Hochschule – im Laufe des Alltags kommen wir immer wieder mit Daten in Berührung, die realitätsgetreu und schnell abrufbar sein müssen, und nicht wenige dieser Daten sind darüber hinaus streng vertraulich, wie Bankverbindungen oder die eben erwähnten Patientendaten.

Entsprechend wichtig ist ein angemessener Umgang mit diesen Daten, ein Thema, mit dem sich die Informatik seit ihrer Anfangszeit beschäftigt, und das auch in der Zukunft nicht an Bedeutung verlieren wird. Deshalb ist es notwendig, dass dieses Thema auch in der Informatikausbildung ausreichend behandelt wird.

An der Hochschule Merseburg wird das Thema in der Vorlesung „Datenbanken“ behandelt, die für Studiengänge Angewandte Informatik und Wirtschaftsinformatik eine Pflichtveranstaltung ist. Im Rahmen dieser Vorlesung wird ein SQL-Tutorial durchgeführt, das eine Einführung in die Structured Query Language bietet und damit die Grundlagen für die praktische Verwendung SQL-basierter Datenbanken schafft. Wie in jedem Feld ist auch hier die Schaffung einer soliden Wissensbasis entscheidend für das weitere Lernen, weshalb dieses Tutorial eine Schlüsselrolle im Ablauf der Vorlesung spielt.

2 Ziele der Arbeit

Um die Qualität der Vorlesung zu verbessern, soll das SQL Tutorial im Zuge dieser Arbeit auf Schwachstellen getestet und angepasst werden. Zu diesem Zweck wird zuerst das existierende Tutorial analysiert, um Verbesserungsmöglichkeiten zu identifizieren und diese umzusetzen. Im Vordergrund steht hierbei, den Studierenden eine fundierte Grundlage zu bieten, um erfolgreich SQL Queries zu schreiben, auf der weiterführende Veranstaltungen aufbauen können. Da das Tutorial zeitlich eingegrenzt ist, ist es wichtig, dass es in dieser Zeit die behandelten Inhalte anschaulich und verständlich vermittelt.
Besonderes Augenmerk liegt hierbei darauf, die verwendeten Beispiele und Übungsaufgaben zu vereinheitlichen und verständlicher zu gestalten.

Um dies zu testen, wird das Tutorial im Sommersemester praktischen Einsatz finden. Dadurch können Einschätzungen der Studierenden gesammelt werden, um die vorher eingesetzten Verbesserungen zu evaluieren und darüber hinaus neue Pfade zur Weiterentwicklung des Lehrmaterials aufzuzeigen.

3 Aufbau der Arbeit

Die Grundlage dieser Arbeit ist eine ausführliche Bestandsaufnahme in Kapitel 2. Darin werden zuerst die historischen und technischen Grundlagen des relationalen Datenbankmodells und von SQL betrachtet, bevor der zweite Abschnitt des Kapitels sich mit dem Erlernen dieser Grundlagen beschäftigt, besonders im Rahmen der Hochschulausbildung. Dafür werden sowohl eine Literaturrecherche über den Stand des Wissens zum Lernen und Lehren von SQL im allgemeinen, als auch eine Einschätzung der Rahmenbedingungen der speziellen Situation dieser Arbeit durchgeführt.

In Kapitel 3 wird schließlich die erste Runde an Verbesserungen durchgeführt, wobei besonders die Analyse des vorhandenen Tutorials hier einen großen Anteil des Kapitels ausmacht.

Nach dem praktischen Einsatz der überarbeiteten Version wird in Kapitel 4 das studentische Feedback und die Erfahrungen aus dem Praxiseinsatz dokumentiert und analysiert, und in Kapitel 5 werden schließlich die Ergebnisse zusammengefasst und ein Ausblick auf mögliche Ansätze für eine zukünftige Weiterentwicklung des Lehrmaterials gestellt.

II Bestandsaufnahme

4 Relationale Datenbanken und SQL

4.1 Historischer Hintergrund

Das Speichern von persistenten Daten, die verfügbar sind, bis sie absichtlich gelöscht werden und einen Neustart des Systems überdauern, ist eine Funktion, die aus modernen Computern nicht mehr wegzudenken ist und über die Endnutzer selten nachdenken – es erfordert nicht mehr Aufwand als einen Mausklick oder die Tastenkombination Strg+S. In der 60er und 70er Jahren des zwanzigsten Jahrhunderts war die Handhabung von persistenten Speichern aber eines der wichtigsten Probleme, die es für die heute alltägliche Verbreitung von Computersystem zu lösen galt.

In dieser Zeit gab es wichtige Fortschritte sowohl im Hardware- als auch im Softwarebereich. Einige Spuren dieser Entwicklungen sind auch noch heute sichtbar, zum Beispiel in dem geläufigen Zylindersymbol für Datenbanken. Dieses beruht auf einem in den 60ern von IBM kodifizierten Symbol für die Darstellung von Magnetscheibenspeicher in Programmablaufsdiagrammen. Im Gegensatz zu den wesentlich kleineren Festplattenlaufwerken, die später in persönlichen Computern verbaut werden, sind an den zu dieser Zeit in Großrechnern eingesetzten Speichermedien die einzelnen Magnetscheiben auch von außen deutlich sichtbar.

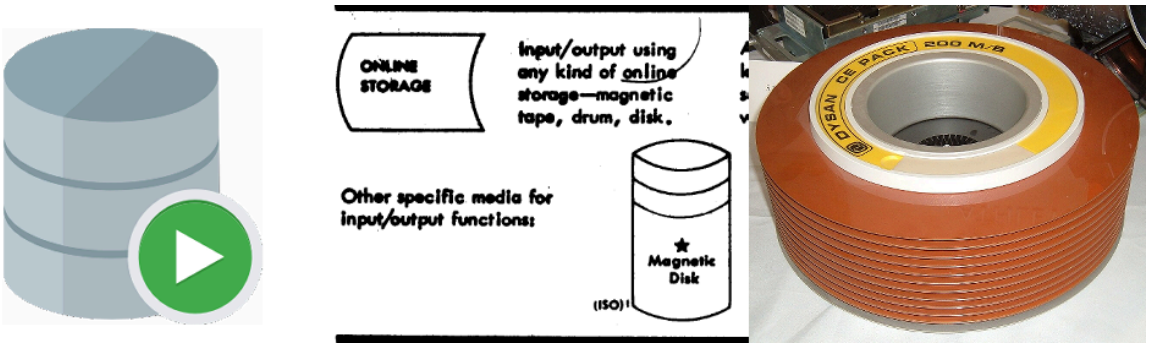


Abbildung 1: Das Logo von Oracle SQL Developer[1], das IBM-Symbol für Magnetspeicher[2] und ein Dysan CE Pack[3]

Die Entwicklung des Magnetscheibenspeichers machte es auf Hardwareebene möglich, bestimmte Speicheradressen gezielt zu adressieren, im Gegensatz zu den vorher geläufigen Magnetbändern, bei denen das gesamte Band durchlaufen werden muss. Auf der Softwareseite finden die ersten Schritte hin zu relationalen Datenbanken am Ende der 60er Jahre am IBM Research Institute in San Jose statt.

Zu dieser Zeit werden die Grundlagen des relationalen Datenbankmodells von Edgar F. Codd entwickelt und in „A relational model of data for large shared data banks“, publiziert in CACM im Jahr 1970, ausführlich vorgestellt. Codds Ziel mit diesem Modell war es, die Nutzer zukünftiger Datenbanken von den physikalischen Speicherstrukturen unabhängig zu machen.[4] Diese Unabhängigkeit bietet mehrere Vorteile, die letztendlich dazu führen, dass das relationale Modell das zeitgleich entwickelte Netzwerkmodell (auch CODASYL-Modell genannt) bei den 80er Jahren verdrängt.

Am wichtigsten für den Inhalt dieser Arbeit ist, dass durch die Unabhängigkeit der Organisation der Daten von ihrer Anordnung auf Hardware-Ebene die Grundlage für eine Abfrage-Sprache geschaffen wird, durch die Programme unabhängig von der physischen Struktur der Daten auf diese zugreifen können. Aus E.F. Codds Hintergrund als Mathematiker heraus entwickelte er die beiden Abfrage Sprachen Relational Algebra und Relational Calculus, die Symbole der formalen Logik verwenden, um Datenbankabfragen darzustellen. Als Ray Boyce und Don Chamberlin in 1972 an einem Symposium von ihnen erfuhren, bieteten diese Query Sprachen ihnen den Anstoß, eine neue Sprache zu entwickeln. Boyce und Chamberlin waren von den Möglichkeiten der Sprachen überzeugt, betrachteten ihre enge Verknüpfung mit formaler Logik jedoch als eine Hürde zu ihrer Verbreitung.[5]

In 1973 begannen sie im Zuge des IBM-Projektes System R mit der Arbeit an der neuen Query Sprache Sequel. Der Vorteil, den Sequel über Relational Algebra bietet, ist, dass die Abfragen ohne mathematische Symbole dargestellt werden können.

$\pi(Spaltenname)(\sigma(Spaltenname > 10)Tabellenname)$

ist ohne Kenntnisse der mathematischen Symbole nicht verständlich, der Ausdruck der selben Abfrage in Sequel,

```
SELECT Spaltenname FROM Tabellenname WHERE Tabellenname.Spaltenname > 10;
```

kann ein fachfremder Benutzer, der zumindest mit der englischen Sprache und dem Größer-als-Zeichen vertraut ist, mit einer minimalen Erklärung intuitiv interpretieren – alle Werte einer bestimmten Tabellenspalte, die größer als 10 sind, werden ausgewählt.

Sequel wird zuerst experimentell von IBM eingesetzt und weiter verbessert, und der Name wird von Sequel (Structured English QUEry Language) zu SQL (Structured Query Language) geändert. International sind sowohl die Aussprache als „Sequel“ als auch als „Es-Quu-El“ in Gebrauch. In den späten 70ern treten die ersten kommerziellen SQL Implementationen auf, und in 1986 wird mit ANSI X3.135-1986 der erste SQL Standard veröffentlicht. Im folgenden Jahr werden die selben technischen Spezifikationen im ISO Standard ISO 9075-1987 auf internationaler Ebene adoptiert. Der zum Zeitpunkt des Schreibens neueste SQL-Standard wurde in 2023 in ISO/IEC 9075-1:2023 „Information Technology – Database languages SQL Part 1: Framework (SQL/Framework)“ veröffentlicht. Wie “Part 1” schon vermuten lässt, ist der SQL Standard über die Jahre auf nicht weniger als 16 Dokumente angewachsen.

4.2 Gegenwärtige Relevanz

Heute gibt es viele relationale Datenbankmanagementsysteme, die auf dem SQL-Standard beruhen, unter anderem Oracle, bereitgestellt von dem gleichnamigen Entwickler, IBM DB2 oder Microsoft SQL Server, aber auch Open Source Lösungen wie PostgreSQL. Obwohl diese Anbieter den selben SQL-Standard implementieren, gibt es in der Umsetzung trotzdem Unterschiede, weshalb sich die verwendete SQL-Sprache von verschiedenen DBMS auch voneinander unterscheidet. Wie in einem gesprochenen Sprache werden diese unterschiedlichen SQL Varianten als Dialekte bezeichnet.

Seit der Jahrtausendwende hört man häufiger das Argument, dass SQL-Datenbanken den Bedürfnissen des modernen Internets nicht mehr gewachsen ist. Das relationale Datenbankmodell ist aufgebaut, um die Konsistenz der Daten zu garantieren. Konsistenz ist in vielen Anwendungsbereichen eine sehr hohe Priorität (das beste Beispiel hier ist jeder Prozess, der Geld und Zahlungsabläufe behandelt), aber sie kommt auf Kosten der Effizienz, besonders im Mehrbenutzerbetrieb.

Viele moderne Webanwendungen, wie beispielsweise Social Media, produzieren eine sehr große Menge an Daten, sind grundlegend skalierbar aufgebaut, und inkonsistente Daten zwischen zwei Knoten sind nicht der katastrophale Fehler, den sie in der Datenbank einer Bank darstellen würden. NoSQL Datenbanken sollen diese Probleme beheben, indem sie nicht-relationale Datenmodelle und Systeme verwenden, die auf horizontale Skalierbarkeit optimiert sind. Außerdem sind die Schemata in NoSQL Datenbanken flexibler, anders als in relationalen Datenbanken müssen sich einzelne Datensätze nicht immer an die Form einer übergeordneten Tabelle halten.[6]

NoSQL heißt aber nicht kein SQL, offiziell steht der Name für „Not only SQL“, denn viele NoSQL Datenbanken implementieren trotz ihres nicht-relationalen Aufbaus SQL-Dialekte. Einer der frühesten Befürworter für NoSQL-Datenbanken war Google, die schon in 2006 ihr hauseigenes NoSQL DBMS, Bigtable, der Öffentlichkeit vorstellten.[7] In 2024 erhielt Bigtable trotzdem eine eingeschränkte SQL-Implementation, und in der Bekanntmachung dieser wird SQL als der „the de-facto data management standard for decades“ bezeichnet.[8]

Auch reine SQL Datenbanken sind immer noch weit verbreitet. Laut der Stackoverflow Developer Survey waren in 2024 die vier am häufigsten verwendeten Datenbanken alle SQL-basiert. Die Umfrage stellte Entwicklern mit verschiedenen Erfahrungsleveln die Frage “Which database environments have you done extensive development work in over the past year, and which do you want to work in over the next year?” und aus den zehn am häufigsten genannten DBMS der Umfrage sind MongoDB, Redis, Elasticsearch und DynamoDB NoSQL Datenbanken und PostgreSQL, MySQL, SQLite, Microsoft SQL Server, MariaDB und Oracle sind SQL-basiert.

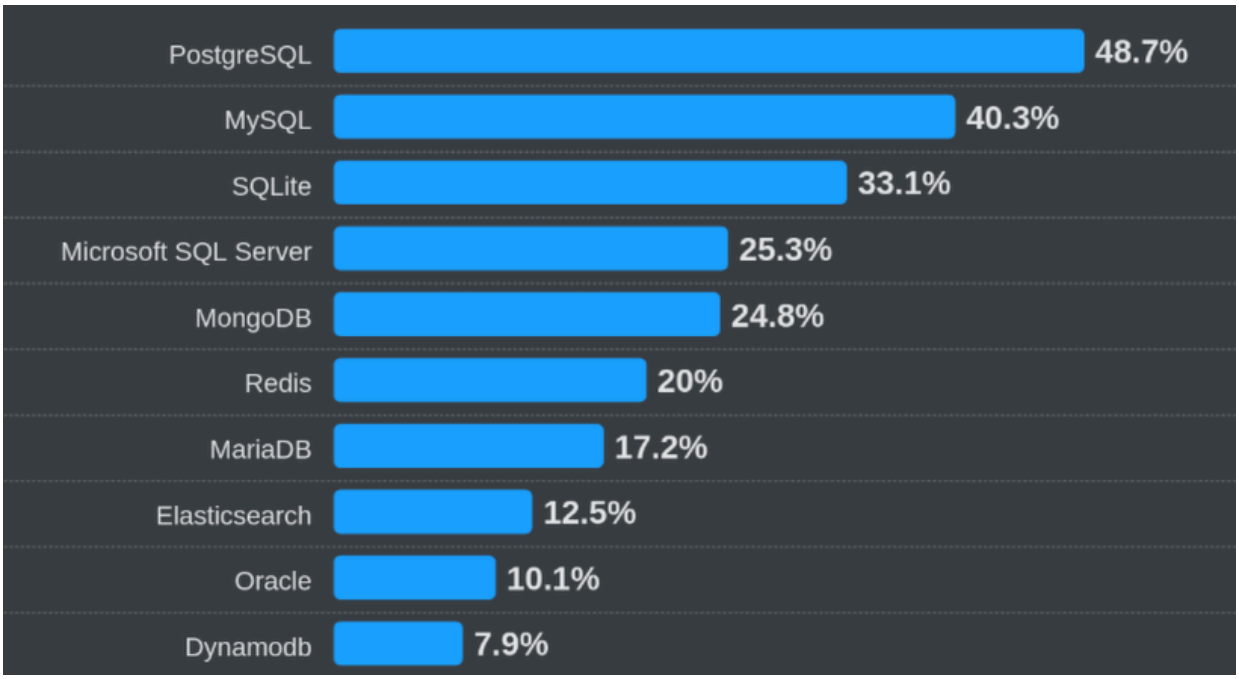


Abbildung 2: Stackoverflow Developer Survey 2024, Abschnitt Databases (Ausschnitt)[9]

In dem von Redgate Software bereitgestellten DB-Engines Ranking, das monatlich kalkuliert wird und sich aus mehreren Faktoren wie dem Erscheinen in Google Trends, technischen Diskussionen online oder Job Angeboten zusammensetzt, sieht es ähnlich aus. Relationale Datenbanken machen hier einen noch größeren Anteil der ersten zehn Positionen aus, mit Oracle an erster Stelle und dem dreifachen Punktwert der ersten NoSQL-Datenbank, MongoDB.

Rank			DBMS	Database Model	Score		
May 2025	Apr 2025	May 2024			May 2025	Apr 2025	May 2024
1.	1.	1.	Oracle	Relational, Multi-model	1226.57	-4.49	-9.72
2.	2.	2.	MySQL	Relational, Multi-model	964.98	-22.12	-118.76
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	774.89	-10.12	-49.41
4.	4.	4.	PostgreSQL	Relational, Multi-model	674.32	+7.07	+28.77
5.	5.	5.	MongoDB	Document, Multi-model	402.51	+2.46	-19.14
6.	6.	↑ 9.	Snowflake	Relational	172.01	+3.92	+50.67
7.	7.	↓ 6.	Redis	Key-value, Multi-model	152.19	-1.92	-5.61
8.	↑ 9.	8.	IBM Db2	Relational, Multi-model	126.40	+0.36	-2.06
9.	↓ 8.	↓ 7.	Elasticsearch	Multi-model	123.81	-4.27	-11.54
10.	10.	10.	SQLite	Relational	117.77	+3.68	+3.45

Abbildung 3: DB-Engines Ranking (Ausschnitt)[10]

Nach nunmehr einem halben Jahrhundert seit ihrer Entwicklung ist die Dominanz der SQL-Sprache im Feld der Datenbanken nicht mehr unangefochten, aber in Angesicht dieser Statistiken wirken Nachrichten über ihr Ableben doch etwas voreilig. Die Zukunft der Technologie vorauszusagen, will ich mir nicht anmaßen, aber für Anwendungen, bei denen die Konsistenz der Daten von höherer Priorität ist, werden relationale DBMS weiterhin einen Vorteil gegenüber ihren NoSQL Gegenstücken haben, und zumindest in der Gegenwart und nahen Zukunft ist SQL weiterhin der Industriestandard, weshalb sie auch aus der Informatikausbildung noch nicht wegzudenken ist.

4.3 Grundlagen des relationalen Modells

Das von E. F. Codd angestrebte Ziel war es, Benutzern den Zugriff auf Daten zu ermöglichen, ohne von ihnen Wissen zu verlangen, dass über die gesuchten Daten hinaus geht. Um dies zu ermöglichen, sind die Daten in sogenannten Relationen, von denen sich der Name des Modells ableitet, gespeichert. Umgangssprachlich werden Relationen auch als Tabellen bezeichnet, was ihren Aufbau anschaulicher macht, und diese Bezeichnung wird auch in der SQL-Sprache selbst verwendet. Jede Tabelle ist eindeutig benannt, und ebenso sind die Bezeichnungen der Spalten innerhalb einer Tabelle eindeutig. Eine Datenbank kann also zwei Tabellen (beispielsweise *Kunden* und *Mitarbeiter*) enthalten, die beide eine Spalte *Name* haben, und die Spalten sind trotzdem als *Kunden.Name* und *Mitarbeiter.Name* eindeutig identifizierbar.

Jeder einzelne Kunde oder Mitarbeiter wird durch eine Zeile in der Tabelle abgebildet, diese Zeilen werden in relationalen Datenbanken auch Tupel genannt. Ebenso wie die Spalten muss auch jedes Tupel eindeutig identifizierbar sein, dafür dient der sogenannte Primary Key oder Primärschlüssel. Dabei handelt es typischerweise um eine Spalte, in der jeder Wert nur einmal vorkommt, beispielsweise einer Kundennum-

mer. Ein Primärschlüssel kann sich aber auch aus mehreren Spalten zusammensetzen, wenn dies für die eindeutige Bezeichnung notwendig ist. Nehmen wir als Beispiel eine Tabelle, speichert, welche Personen an welchen Veranstaltungen teilnehmen. Diese Tabelle hat mindestens die Spalten *Veranstaltung* und *Teilnehmer*. Keine dieser Spalten ist ausreichend, um die Zeilen zu identifizieren, da ein Teilnehmer mehrere Veranstaltungen besuchen kann, und Veranstaltungen mehrere Teilnehmer haben. Die Kombination von Veranstaltung und Teilnehmer ist jedoch in jeder Zeile einzigartig, weshalb sich die beiden Spalten als zusammengesetzter Primärschlüssel eignen.

Eine Tabelle kann mehrere Spalten enthalten, die als Primärschlüssel in Frage kommen. Dies kommt beispielsweise in Produkttabellen häufig vor, in denen jedes Tupel eine Artikelnummer und einen Namen hat. In diesem Fall wird nur eine der beiden Spalten als Primärschlüssel ausgewählt.

Primärschlüssel sind auch dafür wichtig, zwei Tabellen miteinander zu verwinden, zum Beispiel um einzelne Bestellungen Kunden zuzuordnen. Hierbei handelt es sich um eine 1:n Beziehung: Ein Kunde kann beliebig viele Bestellungen haben, aber jede Bestellung ist genau einem Kunden zugeordnet. Es liegen zwei Tabellen vor: *Kunden* und *Bestellungen*. Um die Tupel der Tabelle *Bestellungen* einzelnen Kunden zuzuordnen, wird ihr eine Spalte hinzugefügt, deren Werte in der Menge der Werte des Primärschlüssels der Tabelle *Kunden* enthalten ist. Diese Spalte wird als Foreign Key oder Fremdschlüssel bezeichnet, da sich ihre Werte auf den Primary Key einer fremden Tabelle beziehen.

Das Gegenstück zu 1:n Beziehungen sind 1:1 Beziehungen, in denen jedem Tupel genau ein Wert einer anderen Tabelle zugewiesen ist. Häufig kann eine solche Beziehung innerhalb von einer Tabelle zusammengefasst werden, beispielsweise wäre es möglich, in einer Personentabelle die Kontaktinformationen der aufgelisteten Personen in eine zweite Tabelle auszulagern, die durch einen Fremdschlüssel mit der Personentabelle verbunden ist, aber die Telefonnummern und Adressen können auch als Spalte in der Personentabelle selbst angelegt werden. Trotzdem können 1:1 Beziehungen aus praktischen Gründen sinnvoll sein, um Zugriffberechtigungen an den Tabellen klar voneinander zu trennen, oder auch schlicht zum Zweck der Übersichtlichkeit. In manchen Fällen kann es auch vorkommen, dass eine 1:1 Beziehung notwendig ist, wenn beide Seiten der Beziehung unabhängig voneinander existieren können. Betrachten wir als Beispiel eine Datenbank, die die Mitarbeiter und Fahrzeuge eines Unternehmens enthält und die Firmenfahrzeuge den Mitarbeitern zuweisen soll. Ein Mitarbeiter kann existieren, ohne eine zugehöriges Fahrzeug zu haben, und ebenso kann ein Fahrzeug gegenwärtig niemandem zugewiesen sein.

Aus diesen Punkten lässt es sich vermuten, dass es auch eine n:m Beziehung gibt, in der Werte auf beiden Seiten der Beziehung beliebig oft vorkommen können, und es fällt auch nicht schwer, sich ein Beispiel hierfür auszudenken. Ein Schauspieler tritt in mehreren Filmen auf, aber die meisten Filme enthalten auch mehr als einen Schauspieler. Da

Beziehungen in relationalen Datenbanken aber durch Verweise auf die eindeutigen Primärschlüssel der referenzierten Tabelle erfolgen, ist dies praktisch nicht direkt umsetzbar. Um solche Beziehungen trotzdem darzustellen, muss man eine dritte Tabelle zur Verknüpfung verwenden, die sich auf die Primärschlüssel beider Tabellen in der n:m Beziehung beruft. Ein Beispiel dafür wäre die vorher erwähnte Tabelle, die die Teilnahme an verschiedenen Veranstaltungen speichert, indem sie die Primärschlüssel der Tabellen *Veranstaltungen* und *Personen* in einem Tupel verbindet.

Um einen Datenbestand in einer relationalen Datenbank zu verwalten, ist es notwendig, dass die Daten in der sogenannten Normalform vorliegen. Das entscheidende Merkmal dieser Form ist, dass die einzelnen Zellen der Tabelle nur atomare Elemente enthalten, also Daten, die keine Unterelemente enthalten. Nehmen wir wieder die Kundentabelle als Beispiel: Um die Bestellungen eines Kunden direkt in der Kundentabelle selbst aufzulisten, müsste diese eine Spalte haben, deren Felder eine Datenliste oder sogar eine zweite Untertabelle enthalten, da ein Kunde beliebig viele Bestellungen haben kann. Dies ist in relationalen Datenbanken nicht möglich, weshalb die Bestellungen in einer eigenen Tabelle geführt und durch einen Fremdschlüssel mit der Kundentabelle verbunden werden. Der Vorgang, ein Schema diesen Anforderungen anzupassen, heißt Normalisierung.

Von E. F. Codd wird dieser Zustand ursprünglich schlicht als „Normalform“ bezeichnet, wurde von ihm aber schon in 1971 in „Further Normalization of the Data Base Relational Model“ um die zweite und dritte Normalform erweitert. Die Normalformen bauen aufeinander auf, das heißt, dass eine Relation, die der 2. Normalform entspricht, grundlegend auch der 1. Normalform entsprechen muss.[11] Seitdem wurden von anderen weitere Normalformen definiert, aber allgemein wir eine Datenbank als normalisiert betrachtet, wenn sie der 3. Normalform entspricht.

Ziel der zweiten und dritten Normalform ist es, die Datenbank gegen Anomalien bei der Änderung oder dem Einfügen von Daten abzusichern, indem doppelt gehaltene Daten reduziert werden. Für die zweite Normalform werden alle Felder, die nur von einer Teilmenge des Primärschlüssels, aber nicht von dem gesamten Schlüssel abhängig sind, in eigene Tabellen ausgelagert, die nur von diesem Schlüssel bestimmt sind. Die 2NF ist nur für Tabellen mit zusammengesetzten Primärschlüsseln relevant, da sie sonst automatisch erfüllt ist.

In der dritten Normalform werden Abhängigkeiten von Nichtschlüssel-Spalten voneinander eliminiert. Ist Spalte B von Spalte A abhängig, aber keine der beiden Spalten ist der Schlüssel oder Schlüsselkandidat (eine Spalte oder Kombination von Spalten, die technisch als Primärschlüssel in Frage kommt, aber nicht als Primärschlüssel gewählt wurde) der Tabelle, so wird eine neue Tabelle mit dem Primärschlüssel A erstellt, und Spalte B wird aus der ursprünglichen Tabelle entfernt und in diese neue Tabelle eingefügt.

Eine Relation befindet sich also in der dritten Normalform, wenn jede Tabelle nur atomare Werte enthält, jede Spalte nur von dem gesamten Primärschlüssel oder Schlüsselkandidaten abhängig ist und keine weiteren Abhängigkeiten innerhalb einer Tabelle bestehen.

Im SQL-1999 Standard wurde das Modell der relationalen Datenbanken noch einmal um ein wichtiges Konzept erweitert. Einerseits ist es häufig notwendig, Dateien von außerhalb des Datenbankkonzeptes zu verwalten, beispielsweise Medieninhalte wie Bilder oder auch Pläne oder Geodaten. Andererseits werden Datenbanken häufig eingebettet in objektorientierte Programmiersprachen verwendet. Die Objekte, die der objektorientierten Programmierung ihren Namen geben, können sehr komplexe Beziehungen und Abhängigkeiten zu- und voneinander haben.[12]

Um eine Liste von Personen mit Namen und Adresse in einer relationalen Datenbank abzubilden, würde man mehrere Spalten aneinander reihen, Vorname, Nachname, Postleitzahl, Wohnort, Straße, Hausnummer. In einem objektorientierten Ansatz wäre es dagegen möglich, die Adresse als ein Objekt zu formulieren, das die Datenfelder PLZ, Ort, Straße und Hausnummer umfasst. Das Objekt *Person* hat dann die Datenfelder Vorname, Nachname und Adresse, wobei das Adressen-Feld ein Objekt des neu definierten Typs *Adresse* enthält.

In SQL-1999 wurde durch die Aufnahme diese Objekt-Konzepts das relationale Datenbankmodell zum objektrelationalen Datenbankmodell erweitert, in dem es möglich ist, neben den elementaren Datentypen auch nutzerdefinierte Datenobjekte zu verwenden und Dateien aus fremden Anwendungen, die nicht dem relationalen Modell entsprechen, zu verwalten.

4.4 Sprachaufbau und Funktionsweise

SQL ist eine deklarative Sprache. Im Gegensatz zu imperativen Sprachen wie C oder Java beschreibt sie, was das Skript erreichen soll, aber nicht wie. Der präzise Ablauf wird durch den Compiler des DBMS ermittelt. SQL Abfragen werden häufig nicht direkt von einem menschlichen Benutzer geschrieben, sondern von einem Programm (beispielsweise einer Java-Anwendung) nach vorprogrammierten Regeln formuliert.

Zwar wird SQL als eine Sprache bezeichnet, sie setzt sich aber tatsächlich aus mindestens drei verschiedenen Sprachbestandteilen zusammen, die jeweils unterschiedlichen Zwecken dienen. Dabei handelt es sich um

- Data Definition Language (DDL): Behandelt das Schema der Datenbank, indem der Aufbau der enthaltenen Tabellen beschrieben wird. Dazu gehört die Definition der Tabellenspalten und die Beziehungen verschiedener Tabellen zueinander.

- Data Manipulation Language (DML): Dient dazu, die in der Datenbank enthalten Daten zu verändern, neue Daten einzufügen und vorhandene Daten zu löschen. Eine Teilmenge der DML, die manchmal auch als eigenständiger Sprachanteil betrachtet wird, ist die Data Query Language (DQL), die Daten abfragt und bereitstellt.
- Data Control Language (DCL): Verwaltet die Rechte der Datenbanknutzer und weißt Nutzern Rechte an bestimmten Tabellen zu oder entzieht sie.

Die Informationen, die für die Verwaltung der Daten benötigt werden, sind ihrerseits ebenfalls tabellarisch in dem sogenannten Data Dictionary gespeichert. Das Data Dictionary wird beim Erstellen einer Datenbank sofort mit angelegt und enthält wichtige Informationen über die einzelnen Objekte der Datenbank und ihre Benutzer. Der Aufbau des Data Dictionarys ist stark anbieterabhängig, aber grundsätzlich bedeutet das Speichern in Tabellenform, dass das Data Dictionary selbst ebenfalls durch SQL abfragbar ist.

Da die Aufgabe einer Datenbank das Speichern von Daten ist, ist das Format dieser Daten natürlich sehr wichtig, aber gerade hier kann es große Abweichungen zwischen zwei verschiedenen DBMS geben, trotz der Definition einiger Datentypen im SQL Standard.

Allgemein kann man davon ausgehen, dass die numerischen Datentypen INTEGER (auch als INT abgekürzt) für Ganzzahlen und DECIMAL oder NUMERIC für Kommazahlen vorhanden sind, aber in ihrer Implementierung kann es Unterschiede geben. Der maximale Wertebereich kann sich zwischen verschiedenen DBMS unterscheiden, und bestimmte Details ihrer Handhabung (beispielsweise die Frage, ob das Ergebnis einer Integer-Division immer eine weitere Ganzzahl ist) sind ebenfalls anwendungsabhängig. die Stellenzahl wird beim Festlegen des Datentyps mit definiert. Der Aufbau dafür sieht folgendermaßen aus: NUMERIC(p,s) wobei p die Gesamtanzahl der Stellen der Dezimalzahl festlegt und s die Anzahl dieser Stellen, die nach dem Komma stehen. NUMERIC(9,2) zum Beispiel bezeichnet eine Zahl mit insgesamt neun Stellen, sieben vor dem Komma und zwei Nachkommastellen.

Daneben gibt es noch weitere numerischen Typen wie BIGINT (einer Ganzzahl mit einer höheren Speichergröße als der INT Datentyp) oder FLOAT für Gleitkommazahlen, auf die hier nicht näher eingegangen werden soll. In jedem Fall ist es wichtig, die Spezifikationen für das jeweils verwendete DBMS zu konsultieren, um zu erfahren, welche Datentypen verfügbar sind und welche Regeln sie befolgen.

Für Zeichenketten gibt es vor allem zwei Datentypen, CHAR für eine Zeichenkette fester Länge und VARCHAR für eine Zeichenkette mit variabler Länge. CHAR-Zeichenketten werden immer mit Leerzeichen auf die maximale Länge aufgefüllt, weshalb sie immer den selben Platz im Hardwarespeicher beanspruchen. Aus diesem Grund sollte für Zeichenketten, deren Länge stark variiert, VARCHAR verwendet werden, um Speicherplatz einzusparen.

Ebenfalls wichtig sind Zeit- oder Datumsangaben, wofür DATE und TIME zur Verfügung stehen. In der Regel gibt DATE nur ein Datum ohne Zeitangabe an und TIME gibt eine Uhrzeit auf die Sekunde genau an, TIMESTAMP oder DATETIME enthalten sowohl Datum als auch Uhrzeit, und gehen in der Regel über die Sekunden-Genauigkeit des TIME-Datentyps hinaus. Gerade für Datumsangaben ist es wichtig, das verwendete Datumsformat zu beachten. Dieses hängt wieder vom DBMS ab, aber es sind häufig verschiedene Formate verfügbar, und im Fall von Oracle ist das Standardformat sogar über das DBMS hinaus von den geografischen Einstellungen abhängig, in Deutschland wird DD.MM.YY verwendet.

Zwei weitere Datentypen sollen hier noch erwähnt werden: BLOB und CLOB (Binary Large Object und Character Large Object). BLOB dient dazu, größere Binärdaten abzuspeichern (die Größe hängt wieder von der Implementation ab), und CLOB speichert Zeichenketten, die über die maximale Länge von VARCHAR hinaus gehen.

4.4.1 Data Definition Language

Wenn man von Grund auf eine neue Datenbank erstellen will, ist der erste Schritt, mit der Data Definition Language die Schemas der Tabellen zu erstellen und damit den Rahmen zu schaffen, in den die Daten eingefügt werden sollen. Dafür wird der Befehl CREATE TABLE verwendet. Es ist in SQL üblich, Schlüsselwörter der Sprache in Großbuchstaben zu schreiben, auch wenn die Syntax der Sprache selbst die Groß- und Kleinschreibung in der Regel nicht beachtet. Abhängig vom verwendeten DBMS und teilweise auch vom Betriebssystem können Objektnamen trotzdem „case sensitive“ sein, weshalb es wünschenswert ist, Regeln für die Groß- und Kleinschreibung in einer Datenbank festzulegen und diese einzuhalten, auch, um die Lesbarkeit zu verbessern.

Die Syntax für CREATE TABLE sieht folgendermaßen aus:

```
CREATE TABLE Tabellenname
(Spaltenname Datentyp
[,…]);
```

Nach dem Namen der Tabellen werden innerhalb von Klammern und getrennt durch Kommas die einzelnen Spalten definiert. Jede Spaltendefinition umfasst zumindest den Namen der Spalte und ihren Datentyp, und kann zusätzlich einen Default-Wert, auf den die Werte gesetzt werden, wenn beim Einfügen eines Tupels kein Wert angegeben wird, und sogenannte Constraints, die die möglichen Werte der Spalte begrenzen.

Ein Beispiel für Constraints ist die Markierung des Primärschlüssels mit dem Schlüsselwort PRIMARY KEY. Um beispielsweise eine Kundentabelle mit einer ganzzahligen Kundennummer als Primärschlüssel und einer Spalte *Name*, die Zeichenketten enthält, zu erstellen, würde folgendes Skript verwendet werden:

```
CREATE TABLE Kunden
(Kundennummer INT PRIMARY KEY,
Name CHAR(30));
```

Auch Fremdschlüssel werden durch Constraints realisiert. Andere Beispiele für mögliche Constraints sind die Beschränkung einer Spalte auf eine vordefinierte Liste von Werten oder eine Regel, die festlegt, dass die Werte einer numerischen Spalte kleiner als 100 sein müssen.

```
CREATE TABLE Beispiel
(Primaerschluessel INT PRIMARY KEY,
Bsp_Fremdschluessel INT REFERENCES Fremdtabelle(Schluessel),
Bsp_Zahl INT CHECK (Bsp_Zahl < 100));
```

Weitere wichtige Constraint-Typen sind NOT NULL, das festlegt, dass eine Spalte keine leeren Felder enthalten kann, und UNIQUE, bei der sich alle Werte in einer Spalte voneinander unterscheiden müssen. Bei Primärschlüsseln sind diese beiden Bedingungen impliziert und müssen nicht zusätzlich angegeben werden.

Constraints können in zwei Kategorien eingeteilt werden: Column-Constraints, die sich nur auf eine Spalte beziehen, und Table-Constraints, die mehr als eine Spalte umfassen können. In der SQL-Anweisung können diese Kategorien auseinandergehalten werden, da Column-Constraints wie in dem obigen Beispiel in der Definition der Spalte festgelegt werden, während Table Constraints am Ende der Tabellendefinition stehen.

Wenn alle Constraints erfüllt sind, sind die Daten konsistent, und es ist Aufgabe des DBMS, sicherzustellen, dass inkonsistente Daten nicht gespeichert werden, so dass in der Datenbank nach Abschluss einer Transaktion immer ein konsistenter Zustand vorliegt. Diese Überprüfung durch das DBMS durchführen zu lassen, hat in der Praxis den Vorteil, dass nicht jedes Programm, das Daten in die Datenbank einfließen lässt, eigene Integritätsprüfungen durchführen muss. Trotzdem kann es in einigen Situationen nötig sein, externe Mittel zur Integritätsprüfung zu verwenden, wenn die in SQL enthaltenen Regeln für komplexere Tests nicht ausreichend sind.

Nachdem Tabellen erstellt wurden, können sie auch geändert oder gelöscht werden, mit ALTER TABLE und DROP TABLE. Dabei muss allerdings darauf geachtet werden, dass die Integrität des gesamten Datenbestandes nicht verletzt wird. Das Löschen einer Tabelle, die von einer anderen Tabelle über einen Fremdschlüssel referenziert wird, kann je nach Einstellung der Datenbank blockiert werden, um die Integrität der zweiten Tabelle zu sichern. ALTER TABLE kann auch Spalten verändern, und auch hier gibt es Einschränkungen. Kurz gesagt kann durch die ALTER TABLE Anweisung der Speicherumfang einer Spalte immer erhöht, aber nur unter bestimmten Umständen reduziert werden. Beispielsweise kann die Anzahl der Zeichen in einer Zeichenketten-Spalte unabhängig von ihrem Inhalt erhöht werden, aber eine Reduktion ist nur möglich, wenn die Spalte nur NULL Werte enthält. Auch der Datentyp einer Spalte kann nur geändert werden, wenn diese leer ist.

Mit RENAME TABLE ist es möglich, Tabellen umzubenennen, für Spalten existiert kein äquivalenter Befehl.

Andere Datenbankobjekte, die mit der CREATE-Anweisung erstellt werden können, sind Benutzer (CREATE USER), Indexes (CREATE INDEX) und Views (CREATE VIEWS).

Ein Index wird erstellt mit

```
CREATE INDEX Indexname ON Tabellenname
(Spaltenname
[, Spaltenname … ]);
```

und dient dazu, den Zugriff auf Daten der Tabelle über diese Spalte zu beschleunigen. Der Primärschlüssel wird automatisch indiziert. Anders als der Primärschlüssel muss ein Index nicht eindeutig sein, eine hohe Selektivität (viele verschiedene Werte) ist jedoch wünschenswert.

Views oder Sichten sind bestimmte Ansichten existierender Tabellen und dienen dazu, die Abfrage dieser Konfigurationen zu beschleunigen. Deshalb sollten sie für Anfragen eingesetzt werden, die häufig verwendet werden. Wird der Inhalt einer Sicht im Speicher der Datenbank dauerhaft abgelegt, spricht man von einer materialisierten Sicht, ein Konzept das im reinen SQL Standard nicht vorhanden ist, aber von verschiedenen Datenbankanbietern unterstützt wird. Da eine materialisierte Sicht nicht bei jeder Abfrage neu erstellt werden muss, verbessert sie die Zugriffsgeschwindigkeit auf die Daten, aber da diese Daten jetzt doppelt gehalten werden (in der ursprünglichen Tabelle und in der Sicht) kommt es hier zu neuen Bedenken zur Datenkonsistenz. Die Sicht muss aktualisiert werden, wenn zugrundeliegenden Tabellen verändert werden, aber diese Aktualisierung bei jeder Änderung durchzuführen, ist arbeitsaufwendig, weshalb es möglich ist, Sichten nur periodisch zu aktualisieren – was zu inkonsistenten Daten führen kann.

Das Gegenstück zu materialisierten Sichten sind virtuelle Sichten, die nicht im Speicher hinterlegt werden, sondern bei jeder Anfrage aufs neue erstellt werden. Dadurch ist der Zugriff langsamer, aber die abgerufenen Daten sind immer aktuell und es wird kein zusätzlicher Speicher in Anspruch genommen.

4.4.2 Data Manipulation Language

Nachdem die Tabellen erstellt wurden, lautet der nächste Schritt natürlich, diese mit Daten zu füllen. Dafür wird die INSERT Anweisung verwendet, die jeweils eine Spalte in die Datenbank einfügt. Die INSERT-Anweisung kann auf zwei verschiedene Arten strukturiert werden, die hier am Beispiel einer Tabelle namens *Kunden* mit den Spalten *Kundennummer* (INT), *Vorname* (VARCHAR) und *Nachname* (VARCHAR) demonstriert werden sollen.

Die erste Variante ist, die Werte für jedes Feld der neuen Zeile aufzulisten.

```
INSERT INTO Kunden
VALUES (12345, ‚Max‘, ‚Mustermann‘);
```

Damit die Daten den Spalten richtig zugeordnet werden, ist die selbe Spaltenreihenfolge hier unbedingt einzuhalten, und um einzelne Felder leer zu lassen, muss für diese Felder NULL definiert werden, beispielsweise

```
INSERT INTO Kunden
```

```
VALUES (12345, NULL, ‚Mustermann‘);
```

Es ist auch möglich, zuerst die zu füllenden Spalten aufzulisten. Dadurch können Zellen, die nicht gefüllt werden sollen, ausgelassen werden.

```
INSERT INTO Kunden (Kundennummer, Nachname)
VALUES (12345, ‚Mustermann‘);
```

Mit DELETE und UPDATE können Tupel gelöscht, beziehungsweise geändert, werden. Werden diese Befehle ohne Einschränkungen benutzt – beispielsweise

```
DELETE FROM Kunden;
```

werden sämtliche Zeilen in einer Tabelle gelöscht, was natürlich in den meisten Fällen nicht das gewünschte Resultat ist. Deshalb ist es wichtig, die ausgewählten Zeilen einschränken zu können. In SQL wird dafür das WHERE Schlüsselwort verwendet.

Mit WHERE können Bedingungen formuliert werden, die bestimmen, welche Zeilen selektiert werden. Häufig wird es in Kombination mit dem Primärschlüssel benutzt, da dieser am besten geeignet ist, um einzelne Tupel zu identifizieren:

```
DELETE FROM Kunden
WHERE Kundennummer = 12345;
```

Es können allerdings auch andere Bedingungen verwendet werden, und die WHERE Bedingung kann beliebig viele Zeilen auswählen.

```
DELETE FROM Kunden
WHERE Registrierung_Datum < DATE ‚2010-01-01‘;
```

Die WHERE Klausel findet außerdem auch in dem wohl am häufigsten verwendeten SQL-Befehl Anwendung: SELECT, dem Schlüsselwort, das verwendet wird, um Daten aus der Datenbank auszuwählen.

Die Syntax hierfür ist:

```
SELECT Spaltenname [, Spaltenname …]
FROM Tabellenname
WHERE [Bedingung]
```

Bedingungen können auch durch AND, OR und NOT kombiniert oder negiert werden.

```
SELECT * FROM Kunden
WHERE Ort = ‚Stuttgart‘
AND Registrierung_Datum >= DATE ‚2020-01-01‘;
```

gibt beispielsweise alle Kunden zurück, deren Adresse in Stuttgart liegt und die 2020 oder später registriert wurden. Der Sternchen anstellen des Spaltennamens steht dafür, dass alle Spalten ausgewählt werden. Um nur bestimmte Spalten anzuzeigen, wäre es möglich, diese Spalten namentlich anzugeben, beispielsweise

```
SELECT Nachname, Vorname FROM Kunden;
```

Ohne WHERE Bedingung werden alle Zeilen in der Tabelle ausgegeben.

Außerdem können auch mehrere Tabellen verbunden werden. Sollen beispielsweise alle Bestellungen mit den Namen des Kunden angegeben werden, wenn die Tabelle *Bestellungen* nur die Kundennummer enthält, muss der Name aus der separaten Kundentabelle bezogen werden. Dafür kann ein sogenannter Inner Join durchgeführt werden, der die Zeilen beider Tabellen kombiniert, wenn ein Schlüsselwert (in diesem Fall die Kundennummer) zwischen beiden Tabellen übereinstimmt.

```
SELECT * FROM Kunden, Bestellungen
WHERE Kunden.Kundennummer = Bestellungen.Kundennummer;
```

Ohne die WHERE Bedingung würde hier jede Zeile der Tabelle Kunden mit jeder Zeile der Tabelle Bestellungen verbunden werden, auch wenn diese beiden Zeilen inhaltlich keinen Bezug zueinander haben. Dies wird als Cross Join bezeichnet und entspricht aus mathematischer Sicht dem kartesischen Produkt der beiden Mengen. In der Praxis wird der Cross Join im Vergleich zum Inner Join seltener benötigt.

Eine weitere Variante ist der Outer Join. Ein Outer Join gibt wie ein Inner Join alle Zeilen aus, die zwischen den Tabellen durch einen gemeinsamen Wert zusammen gepaart werden können, enthält aber zusätzlich auch alle Zeilen, für die keine passenden Partner in der anderen Tabelle gefunden werden. In diesen Zeilen werden die fehlenden Spalten mit Nullwerten aufgefüllt. Sind die einzelnstehenden Zeilen beider Tabellen in dem Join enthalten, spricht man von einem Full Outer Join, werden nur die Zeilen einer Tabelle aufgefüllt, handelt es sich um einen Left Outer Join oder Right Outer Join, je nachdem, welche Tabelle vollständig verwendet wird.

Die Syntax für Outer Joins unterscheidet sich zwischen verschiedenen SQL-Anbietern, im SQL 92 Standard, in dem Outer Joins zum ersten Mal definiert wurden, lautet sie

```
SELECT * FROM Tabelle1 [FULL/LEFT/RIGHT] OUTER JOIN Tabelle2;
```

Abfragen können auch durch sogenannte Subqueries (Queries, die Teil einer übergeordneten Abfrage sind) modifiziert werden. Um beispielsweise die Bestellungen zu ermitteln, deren Wert über dem Durchschnitt aller Bestellungen liegt, muss zuerst der Durchschnitt ermittelt werden.

```
SELECT * FROM Bestellungen
WHERE Bestellwert > (SELECT AVG(Bestellwert) FROM Bestellungen);
```

Dieses Beispiel zeigt auch, wie mit AVG der Durchschnitt bestimmt werden kann. Analog dazu gibt es auch den Befehl SUM zur Summenbildung, und diese Ausdrücke werden häufig in Kombination mit GROUP BY eingesetzt, um den Durchschnitt, beziehungsweise die Summe, für Teilmengen der Auswahl zu bestimmen.

```
SELECT Kundennummer, SUM(Bestellwert)
FROM Bestellungen
GROUP BY Kundennummer;
```

würde die Zeilen der Tabelle *Bestellungen* nach der Kundennummer gruppieren und dann die Summe der Bestellwerte für jede einzelne Kundennummer bilden.

Auch nach Zeilen mit leeren Feldern kann mit der Bedingung

```
WHERE Spaltenname IS NULL
```

gefiltert werden.

Diese Auflistung der möglichen Formulierungen in SELECT-Ausdrücken ist nicht vollständig, und weitere Funktionen können anbieterabhängig sein. Trotzdem sollte dieser Abschnitt einen Eindruck davon vermitteln, wie flexibel und leistungsstark die Datenaufbereitung in SQL-basierten System ist, was einer der Gründe für SQLs Verbreitung als Standardwerkzeug ist.

4.4.3 Data Control Language

Der dritte und letzte Teil der SQL-Sprache ist die DCL (Data Control Language). Damit werden die Benutzer der Datenbank und ihre Rechte verwaltet. Prinzipiell verläuft die Nutzerverwaltung folgendermaßen: Mit

```
CREATE USER Benutzer IDENTIFIED BY Passwort;
```

wird ein Benutzer-Account und ein assoziiertes Passwort angelegt. Danach werden dem Nutzer Rechte an bestimmten Tabellen eingeräumt:

```
GRANT Privileg ON Tabelle TO USER Benutzer;
```

Bei diesem Privileg kann es sich zum Beispiel um SELECT oder UPDATE handeln. Mit GRANT ALL werden einem Nutzer alle Rechte an einer Tabelle gleichzeitig eingeräumt, ohne sie einzeln auflisten zu müssen, und der „ON Tabelle“ Abschnitt des Befehls kann entfallen. Die Recht, neue Tabellen zu erstellen, ist logischerweise nicht auf bestimmte existierende Tabellen beschränkt, und deshalb enthält die Anweisung diesen Teil nicht:

```
GRANT CREATE TABLE TO Benutzer;
```

Das Gegenstück zu GRANT ist REVOKE, mit dem analog Rechte wieder entzogen werden können.

Die DML und die Handhabung von Nutzerrechten allgemein unterscheiden sich zwischen verschiedenen DBMS. MySQL beispielsweise hat innerhalb des CREATE USER Befehls eine optionalen Parameter „resource_option“, mit dem die Häufigkeit der Anfragen des Benutzers begrenzt werden kann, das im Oracle-SQL Gegenstück nicht existiert, dafür hat Oracle einen QUOTA Parameter, der die Größe des durch diesen Nutzer zuweisbaren Speicherplatz limitiert.

5 Integration von SQL in der Hochschulausbildung

5.1 Allgemeine Form der SQL-Ausbildung

Wie in dem ersten Abschnitt bereits dargelegt, stellt SQL eine wichtige Qualifikation im Informatikbereich dar und sollte deshalb in der Ausbildung nicht vernachlässigt werden. Auch wenn es eines der Ziele in der Entwicklung von SQL war, eine Querysprache zu schaffen, die auch für Laien leicht verwendbar ist, hat sich dies in der Praxis leider nicht bewahrheitet. Eine effektive Nutzung von SQL erfordert zumindest ein gutes Verständnis der Grundkonzepte relationaler Datenbanken, weshalb SQL selten als einzelnen Fach unterrichtet wird, sondern meistens innerhalb eines übergeordneten Kurses über Datenbanken behandelt wird. Umso wichtiger ist es, dass der zeitliche Anteil dieser Veranstaltungen, der SQL zugewiesen wird, effektiv genutzt wird und wichtige Schlüsselkonzepte in dieser Zeit ausreichend vermittelt werden. Datenbankkurse finden in der Regel in der zweiten Hälfte des Studiums statt und sind in ihrem Aufbau größtenteils vergleichbar.[13] Es kann also davon ausgegangen werden, dass die Studierenden schon aus anderen Kursen mit Grundprinzipien der Programmierung, wie Datentypen oder physischer Speicherverwaltung, vertraut sind und das Einbetten von SQL in eine Programmiersprache, wie beispielsweise Java, behandelt werden kann.

Inhaltlich liegt der größte Schwerpunkt auf der DML, was durch die Häufigkeit ihrer Verwendung auch angebracht ist. DCL wird vergleichsweise nur sehr kurz behandelt, was zum Teil daran liegen wird, dass dieser Sprachanteil nicht sehr umfangreich ist. Andere Gründe, warum DCL in der Ausbildung eher eine kleine Rolle spielt, könnten

sein, dass Operationen zur Nutzerverwaltung im Gegensatz zu Datenabfragen nur selten vorkommen und die Implementation dieser stark vom verwendeten DBMS abhängt, weshalb spezifischeres Wissen, das über die Grundlagen der GRANT und REVOKE Anweisungen hinaus geht, nur begrenzt übertragbar ist.

Auch Lehrbücher sind ähnlich aufgebaut und konzentrieren sich in erster Linie auf DML und DDL, so wird in Michael Laube's *Einstieg in SQL*[14] trotz dem Umfang des Buches Data Control Language überhaupt nicht behandelt. Dieses Buch stellt die Praxis in den Vordergrund, während andere Lehrbücher (beispielsweise *Datenbanken und SQL* von Edwin Schicker[15] oder Frank Roeings *Oracle7 Datenbanken erfolgreich realisieren*[16]) häufig mit einer theoretischen Einführung in das relationale Modell beginnen, stellt Einstieg in SQL die praktische Anwendung an erster Stelle und behandelt die Modellierung von Datenbanken erst in Kapiteln 6 und 7.

Der Inhalt der für diese Arbeit betrachteten Lehrbücher ist dem Aufbau der Lehreinheit in den Grundzügen ähnlich. Konsistent ausführlich behandelte Themen sind das relationale Datenmodell, DML und DDL, darüber hinaus weichen die Schwerpunkte aber voneinander ab. Die Einbettung von SQL in Programmiersprachen wird beispielsweise nur unregelmäßig behandelt. Grob können Lernmaterialien in zwei Kategorien eingeteilt werden: Solche, die eine fundierte Grundlage des relationalen Modells und Strategien zur Datenmodellierung vermitteln, und solche, die sich in erster Linie auf die praktische Anwendung konzentrieren, um die Lernenden möglichst zeiteffizient in die Lage zu versetzen, SQL-Queries selbst zu schreiben.

Online verfügbare Tutorials wie beispielsweise SQLBolt[17], gehören überwiegend zu der zweiten Kategorie, haben gegenüber Lehrbüchern aber den Vorteil der Interaktivität, wodurch das Schreiben von Queries direkt an den verwendeten Beispielen erprobt werden kann.

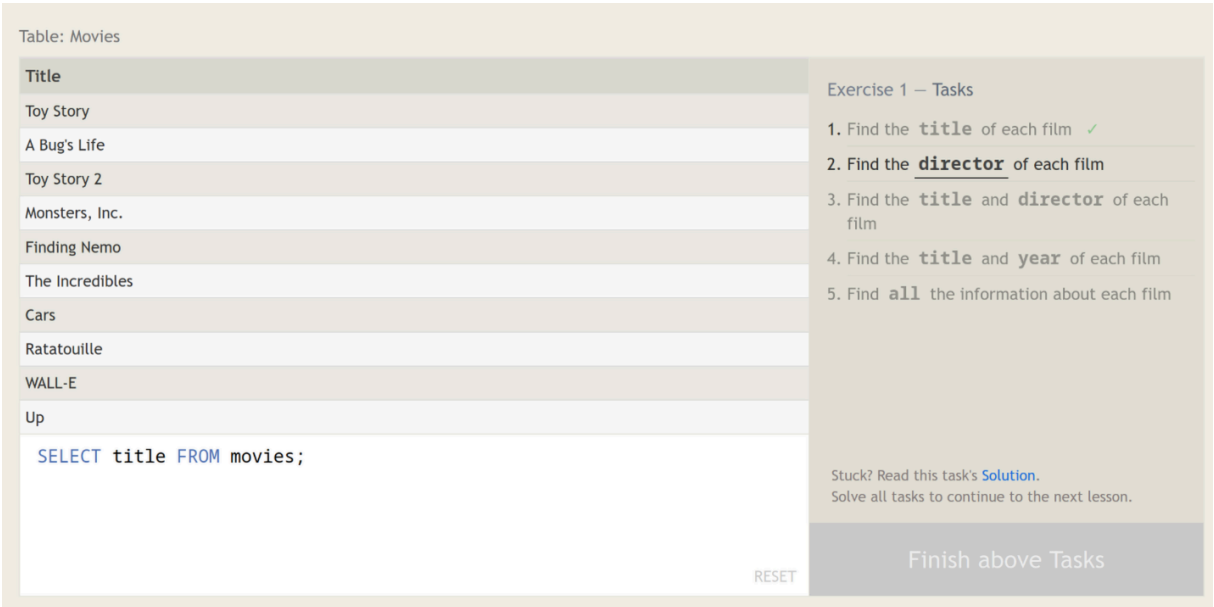


Abbildung 4: Eine Übung auf SQLBolt mit dynamischem Feedback

5.2 Häufige Schwierigkeiten im Erlernen von SQL

Die Grundlagen von SQL zu erlernen, scheint vermeintlich einfach zu sein, da die Syntax einer natürlichen Sprache ähnelt und zielführende SELECT-Anfragen häufig nur eine überschaubare Anzahl von Schlüsselworten verwenden. Trotzdem haben SQL-Lernende häufig auch mit grundlegenden Konzepten der Sprache wie Joins Schwierigkeiten.[18]

Es gibt verschiedene Faktoren, die zu diesen Problem beitragen. Während einfache SQL-Anweisungen durch ihre relative Nähe zur englischen Sprache leicht zu interpretieren sind, werden Befehle mit wachsender Komplexität sehr schnell unübersichtlich, selbst, wenn sie eine in natürlicher Sprache einfach formulierte Frage abbilden.[19] Eine Studie aus 1981, durchgeführt von Charles Welty und David W. Stemple, in der die Testergebnisse von Studierenden nach einem Kurs in SQL und einem Kurs in der oberflächlich ähnlichen, aber strikt prozedural aufgebauten Query-Sprache TABLET verglichen werden, stellt fest, dass die beiden Gruppen bei einfachen Aufgabestellungen ähnliche Ergebnisse erzielten, die TABLET-Gruppe bei komplexeren Abfragen jedoch eine höhere Erfolgsquote hatte.[20] Dieses Ergebnis suggeriert, dass schrittweise Vorgehensweise in prozeduralen Sprachen für komplexe Aufgaben leichter zu kontrollieren ist als deklarative Formulierungen.

Eine weitere Konsequenz der deklarativen Natur von SQL ist, das zur Identifikation und Korrektur von Fehlern nicht die selben Methoden wie in prozeduralen Sprachen verwendet können.[21] Der typische Ablauf zum debuggen eines Programms in einer prozeduralen Sprache wie Java besteht darin, festzustellen, in welchem Schritt das Programm zuerst vom erwarteten Verhalten abweicht. In SQL ist die nur schwer möglich. Die Reihenfolge, in der die Teilanweisungen auf Maschinenebene durchgeführt werden, ist aus ihrer Abfolge in dem geschriebenen Ausdruck nicht direkt zu erkennen.

Diese Schwierigkeiten bei der Korrektur werden durch die verfügbaren DBMS zusätzlich verstärkt. Üblicherweise wird bei einem fehlerhaften Skript nur eine Fehlermeldung angezeigt, und die Verbesserungsvorschläge sind oft falsch oder irreführend.[22] Für syntaktische korrekte Anfragen, die aber semantische Fehler aufweisen und dadurch nicht zielführend sind, beispielsweise Queries mit zwei widersprüchlichen Bedingungen, geben viele DBMS, unter anderem auch die in dem Tutorial verwendete Software, keine Warnungen aus, auch wenn das Ergebnis einer solchen Query immer eine leere Menge ist. Ein Beispiel hierfür wäre

```
SELECT * FROM Kunden
```

```
WHERE Kundennummer = 10000 AND Kundennummer = 20000;
```

Es gibt keine Zeile, in der die Kundennummer sowohl 10000 als auch 20000 ist, weshalb aus der Zeile selbst erkenntlich ist, dass die Abfrage fehlerhaft ist, auch wenn die Syntax selbst einwandfrei ist. Besonders häufig sind Fehler, die Join-Bedingungen betreffen – in einer von Stefan Brass und Christian Goldberg durchgeführten Studie machten fehlende Join-Bedingungen und unnötige Join-Bedingungen gemeinsam 24,4% der von Studierenden in Prüfungen begangenen Fehlern,[23] was suggeriert, dass das Konzept von Joins für Studierende schwerer anzuwenden ist, als andere Themen. Andere häufige Fehler in dieser Studie sind inkonsistente Bedingungen (12,5%) und Lösungen, die viele Duplikate (15,2%) ausgeben, wobei es bei letzterer zu beachten gilt, dass es bei kontextlosen Anfragen, wie sie in Testsituationen vorliegen, schwierig sein kann, zu bestimmen, ob eine Query unnötige Duplikate ausgibt, oder ob Duplikate möglicherweise für die Weiterverarbeitung wichtig sind.

5.3 Rahmenbedingungen des betrachteten SQL Tutorials

Das Modul Datenbanken wird an der Hochschule Merseburg in zwei Veranstaltungen durchgeführt: Einer Vorlesung, in der die theoretischen Voraussetzungen behandelt werden, und eine Übung, in der sie umgesetzt werden. Das SQL-Tutorial, das in dieser Arbeit behandelt wird, ist Teil der Vorlesung. Verpflichtend ist das Modul für die Studiengänge Angewandte Informatik und Wirtschaftsinformatik. Das überarbeitete Tutorial wird im Sommersemester 2025 im Studiengang Wirtschaftsinformatik verwendet, und die Studierenden waren aus vorhergehenden Semestern mit Grundkonzepten der Informatik wie Speicherverwaltung, booleschen Operatoren oder Datentypen schon vertraut, was den Einstieg in das Thema Datenbanken erleichtert.

Das SQL Tutorial ist in der Mitte der Vorlesung angeordnet. Davor werden, nach einer allgemeinen Einführung in Datenbanken und ihre verschiedenen Funktionsweisen, Werkzeuge für die Konzipierung von Datenbanken wie das Phasenmodell und Entity Relationship Modelle behandelt und danach folgen komplexere Themen – ein Kapitel über Transaktionen und Speicherstrukturen, gefolgt von einem Einblick in die Programmierung von Datenbankanwendungen (speziell in Java mit Hilfe der Java DataBase Connectivity) und schließlich ein Überblick über die erweiterten Möglichkeiten Objektrelationaler Datenbanken, die im SQL 1999 Standard definiert wurden.

Idealerweise sollte das Tutorial anbieterunabhängig sein, aber durch die Unterschiede in den verschiedenen SQL-Dialekten ist das leider nicht möglich. Das in dem Tutorial verwendete DBMS ist Oracle Database 11g, eine ältere und getestete Version des Programms, und Oracle SQL Developer, da diese Software den Studenten an der Hochschule zur Verfügung steht. Da das Tutorial sich in erster Linie mit grundlegenden SQL-Anweisungen beschäftigt, die größtenteils schon in früheren Versionen von SQL verfügbar waren, ist es wahrscheinlich vollständig mit neueren Version der Oracle Database, dies wurde jedoch nicht getestet.

In diesem Semester gibt es einige jedoch einige besondere Umstände, die es für die Auswertung zu beachten gilt. Das Tutorial wird nicht, wie üblich, in der Mitte des Semesters durchgeführt, sondern aus praktischen Gründen für diese Arbeit am Anfang der Vorlesung nach einer Einführungsveranstaltung von Professor Weinkauf durchgeführt. Dadurch kann auch getestet werden, ob eine früherer praktischer Einstieg eventuell Vorteile beim Verständnis der theoretischen Kursinhalte bieten kann. Wie in Kapitel 5.1. besprochen, weichen gängige Lehrmaterialien in dieser Hinsicht voneinander ab und beide Abfolgen haben erfolgreiche Beispiele.

III Entwurf des neuen Tutorials

6 Analyse des bestehenden Tutorials

Das vorhergehende Tutorial behandelt die praktische Anwendung von SQL in einem Zeitrahmen von zwei neunzigminütigen Veranstaltungen, in denen die Studenten durch die Beispiele und Übungen geführt werden, und da es sich um einen Abschnitt in einem übergeordneten Modul handelt, muss auch die überarbeitete Version an diese Rahmenbedingungen angepasst sein.

Diese Arbeit wird sich nicht mit der gesamten Vorlesung befassen, sondern nur mit dem Ausschnitt, der die praktische Einführung zu SQL2003 beinhaltet, und dem Kapitel über Transaktionen. Der relevante Abschnitt ist wie folgt aufgebaut:

- [...]
- 5. Einführung in SQL2003
  - 5.1. Geschichte und Sprachbestandteile
  - 5.2. DDL
  - 5.3. DML
  - 5.4. DCL
  - 5.5. Data Dictionary
- 6. Transaktionen und Speicherstrukturen
  - 6.1. Transaktionen
  - [...]
- [...]

Die Abschnitte 5.1. Geschichte und Sprachbestandteile, 5.5. Data Dictionary und 6.1. Transaktionen sind überwiegend theoretischer Natur, während sich die Abschnitte 5.2.-5.4. damit befassen, die Anwendung von SQL-Befehlen an Hand von Beispielen und Übungsaufgaben zu erklären.

5.1. beginnt mit einer kurzen Einführung in die Geschichte der SQL-Sprache und stellt danach die grundlegenden Sprachbestandteile vor. Dabei handelt es sich um Konventionen zur Schreibweise, Datentypen und Operatoren. Es ist unwahrscheinlich, dass sie die Studierenden die hier enthaltenen Informationen sofort einprägen werden, besonders, da es in diesem Bereich noch keine Anwendungsbeispiele gibt, die das gelernte festigen könnten. Besonders die Liste der Datentypen ist zu lang, als das sich die Studierenden sie von hier merken könnten, besonders, da sowohl die ANSI- als auch die Oracle-Datentypen aufgelistet werden. Als Referenz und Ausblick für zukünftige Lektionen ist diese Folie dennoch nützlich.

Abbildung 5: Die Auflistung der Datentypen im Ilias-Kurs

Auffällig ist jedoch, dass nur einige der Oracle-spezifischen Datentypen kurze Erklärungen haben, die ANSI-Datentypen jedoch nicht. Auch eine Erklärung dazu, wie sich die Datentypen zueinander verhalten fehlt, beispielsweise stellen der ANSI Datentyp DECIMAL und NUMBER in der Oracle-Implementation beide Dezimalzahlen mit einer bestimmten Anzahl Vor- und Nachkommastellen dar. NUMBER ist auch fälschlicherweise in beiden Listen vorhanden, obwohl das Schlüsselwort NUMBER als solches im ANSI-Standard nicht verwendet wird. Auf die Unterschiede in den Datentypen näher einzugehen, könnte auch als anschauliches Beispiel für die Unterschiede zwischen verschiedenen SQL-Implementationen nützlich sein und die Studierenden darauf vorbereiten, dass sie bei eigener Recherche in externen Quellen auf unterschiedliche Ausdrücke stoßen können. Auch für einige Datentypen in der Oracle-Liste fehlen Erklärungen. INTEGER als Format für Ganzzahlen ist den Studierenden bereits aus anderen Veranstaltungen vertraut, SMALLINT hingegen wird in aller Wahrscheinlichkeit einen neuen Begriff darstellen, da der äquivalente Datentyp in geläufigen Programmiersprachen wie C++ oder Java als short bezeichnet wird. Auch wenn es möglich ist, sich den Unterschied zwischen INTEGER und SMALLINT aus den Namen herzuleiten (INTEGER ist 4 Byte groß, SMALLINT misst nur 2 Byte), könnte es an dieser Stelle doch sinnvoll sein, die Speichergröße der beiden Datentypen explizit zu erwähnen.

5.1.10 Formatelement für Date-Datentyp

Element	Beschreibung
YY bzw. RR	steht für 2-stellige Jahresangabe
YYYY bzw. RRRR	steht für 4-stellige Jahresangabe
MM	steht für Monatsangabe in Ziffern
DD	steht für 2-stellige Tagesangabe
MONTH	voller englischer Monatsname
DAY	voller englischer Wochentag
YEAR	ausgeschriebenes Jahr

Hinweis
RR bzw. RRRR stehen für Anfangswert Jahr 2000.

Beispiel:
RRRR+ Eingabe von 02 wird gespeichert als 2002
YYYY+ Eingabe von 02 wird gespeichert als 1902

Abbildung 6: Die Format-Erklärung für den DATE-Datentyp

Die Datentypen NUMBER und DATE erhalten zusätzlich ausführlichere Erklärungen auf den darauf folgenden Folien, besonders für DATE ist dies auch notwendig, um die verschiedenen Formate zu erklären. Was hier jedoch fehlt, ist ein Beispiel für ein vollständiges Datumsformat. Die Folie listet nur die einzelnen Bestandteile eines Datumsformats, aber die einzigen Beispiele sind einzelne Jahreszahlen. Es wäre an dieser Stelle wichtig zu wissen, dass sich das Datumsformat zwischen verschiedenen DBMS unterscheidet, und besonders im Fall von Oracle auch von den geographischen Einstellungen des Systems abhängt, speziell durch den Parameter NLS_DATE_FORMAT.[24]

Die im Tutorial verwendete Datenbank ist auf Deutschland eingestellt, weshalb das NLS_DATE_FORMAT DD-MM-YYYY ist, was von dem Format YYYY-MM-DD abweicht, dass von anderen DBMS wie MySQL[25] oder Microsoft SQL Server[26] (oder auch Oracle mit anderen NLS-Einstellungen) verwendet wird.

Die folgenden Abschnitte 5.1.11 Operatoren, 5.1.12 Funktionen und 5.1.13 Systemvariablen und Wildcards haben tabellarische Erklärungen und Anwendungsbeispiele zu allen aufgeführten Elementen, was diese Folien aussagekräftiger als die Folien zu den Datentypen macht.

5.1.11 Operatoren

Operator	Beschreibung	Beispiel
+ -	positiver Ausdruck negativer Ausdruck	SELECT * FROM Teile Where -Bestand<0;
* /	Multiplikation Division	UPDATE Angestellte SET Gehalt = Gehalt * 1.1;
+ -	Addition Subtraktion	SELECT Gehalt - Netto FROM Angestellte;
||	Konkatenation	SELECT Vorname ||''||Nachname FROM Angestellte;

Abbildung 7: Die Operatoren-Tabelle

Folie 5.1.14 bezieht sich auf die DUAL Tabelle in Oracle, wobei der Text auf dieser Folie im Gegensatz zum Rest der Präsentation auf Englisch ist, und nur den Aufbau, aber nicht den Nutzen der DUAL Tabelle beschreibt.

5.1.14 DUAL Tabelle in Oracle

dual is a table which is created by oracle along with the data dictionary. It consists of exactly one column whose name is dummy and one record. The value of that record is X.

D
-
X

Abbildung 8: Die Folie zur Dual Tabelle

Darauf folgt mit 5.1.15 „Zusammenfassung Sprachumfang“ die letzte Folie in diesem Einführungskapitel. Sie bietet einen kurzen Überblick über die drei folgenden Abschnitte DDL, DCL und DML, und die Befehle, die in diesen Kapiteln behandelt werden.

Das Unterkapitel DDL beginnt damit, eine Mitarbeiter Datenbank als Beispiel vorzustellen, die die Tabellen *Mitarbeiter*, *Abteilung*, *Projekt* und *arbeitet_an* enthält.

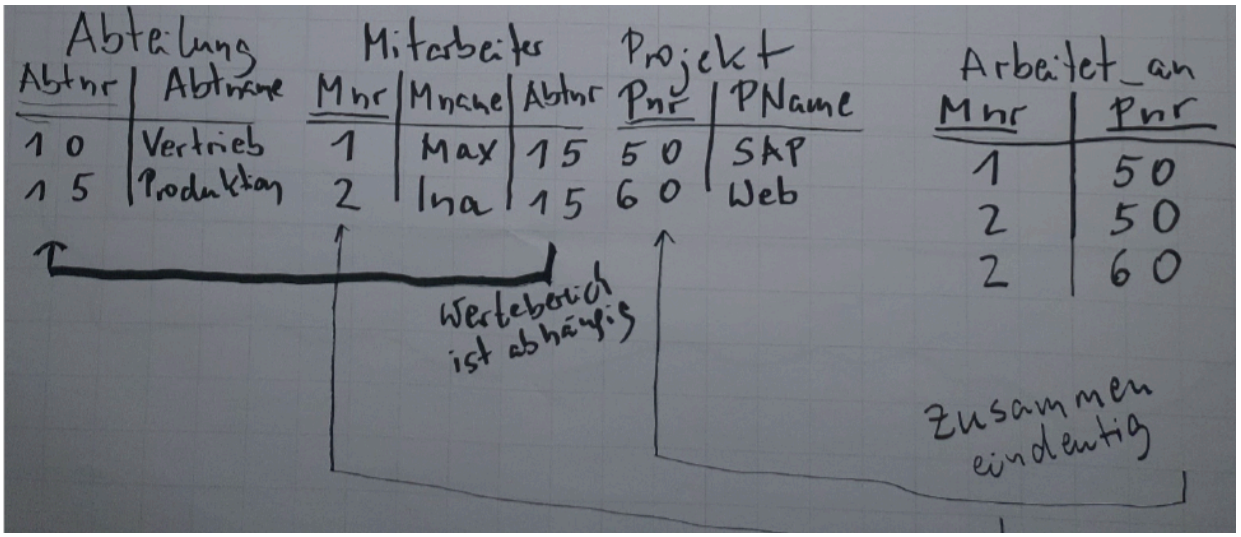


Abbildung 9: Handschriftliches ERD der Beispiels-Datenbank

Die erste Folie enthält neben dem oben stehenden handschriftlichen Diagramm eine Abfolge von SQL-Befehlen, die diese Tabellen erstellen, mit Daten füllen und schließlich löschen. Diese Befehle können überwältigend wirken, sie dienen aber nur dazu, einen Ausblick über den Inhalt des Kapitels zu bieten, indem die einzelnen Begriffe schrittweise genauer erklärt werden.

Nach dieser Einführung wird als erstes CREATE TABLE behandelt, die Syntax des Befehls wird anhand der Abteilungstabelle der obenstehenden Mitarbeiter-Datenbank erklärt. Im Zuge dieser Erklärung werden auch Spaltendefinitionen und Constraints behandelt.

Um die Constraints zu erklären, werden zwei neue Tabellen eingeführt: *Teile* und *Struktur*. Diese Tabellen sind von der vorher verwendeten Mitarbeiter-Datenbank vollkommen unabhängig, und ihre Beziehung zueinander wird erst durch ein Diagramm erklärt, nachdem die Definitionen der Constraints gezeigt werden (im Gegensatz zu der ersten Folie, die mit der bildlichen Darstellung der Mitarbeiter-Datenbank beginnt), wodurch sie etwas schwerer verständlich sind. Die Teile selbst bleiben abstrakt, ihr Aussehen und Einsatzbereich wird nicht erläutert, und über die Strukturen ist nur bekannt, dass sie aus zwei Teilen (einem OTeil und UTeil, die auch identisch sein können) bestehen.

Die Verwendung der Teile ist für die Funktionen der Datenbank aus technischer Hinsicht irrelevant, auch mit vollkommen abstrakten Spaltennamen und beliebigen Datensätzen würde die Datenbank identisch funktionieren. Aus pädagogischer Hinsicht dagegen ist ein abstraktes Beispiel schwerer zu verstehen, und ein neues Beispiel wiederum schwerer als ein bekanntes. Ein neues Beispiel einzuführen, wäre sinnvoll, wenn es umständlich oder unmöglich wäre, die Lerninhalte anhand der existierenden Datenbank zu erklären, aber das ist für die Erklärung der Constraints nicht der Fall. Die drei Constraints der Tabelle *Teile* (PRIMARY KEY, NOT NULL und ein CHECK-Constraint, der sicherstellt, dass ein Zahlenwert größer als 0 ist) sind problemlos an einer anderen Tabelle umsetzbar. Die Constraints der Struktur-Tabelle sind etwas komplexer: Ihr Primary Key setzt sich aus zwei Fremdschlüsseln zusammen, die beide auf die Tabelle Teile verweisen, da die Strukturen jeweils aus zwei Teilen zusammengesetzt sind. Um den selben Lerninhalt des zusammengesetzten Primary Keys darzustellen, ist eine Tabelle notwendig, die ebenfalls über zwei Fremdschlüssel verfügt, aber die Mitarbeiter-Datenbank enthält schon eine solche Tabelle: *arbeite_an*, die Mitarbeiter mit Projekten verknüpft. Zwar beziehen sich ihre Fremdschlüssel auf zwei verschiedene Tabellen, aber das ist für dieses Beispiel ein unerheblicher Unterschied.

Erst nachdem die Constraints in 5.2.7 und 5.2.8 behandelt wurden, wird auf Folie 5.2.9 „FOREIGN KEY + ERD“ die Beziehung der beiden Tabellen anhand eines Diagramms erklärt. Die Lesbarkeit des Textes in diesem Bild ist durch die mangelnde Auflösung beeinträchtigt, außerdem hat das Bild keinen Alternativtext, was zwecks der Barrierefreiheit wünschenswert wäre. Die selben Probleme bestehen für den überwiegenden Anteil der im Modul enthaltenen Grafiken, in diesem Fall ist es jedoch auch fraglich, warum der Create-Table-Befehl überhaupt Teil der Grafik ist, wenn es ebenso als Textelement in dem Tutorial eingebunden sein könnte. Das ERD selbst ist der einzige Bestandteil dieser Grafik, der ein Bildelement erfordert, der erklärende Text und das SQL-Snippet wären als Textelemente lesbarer.

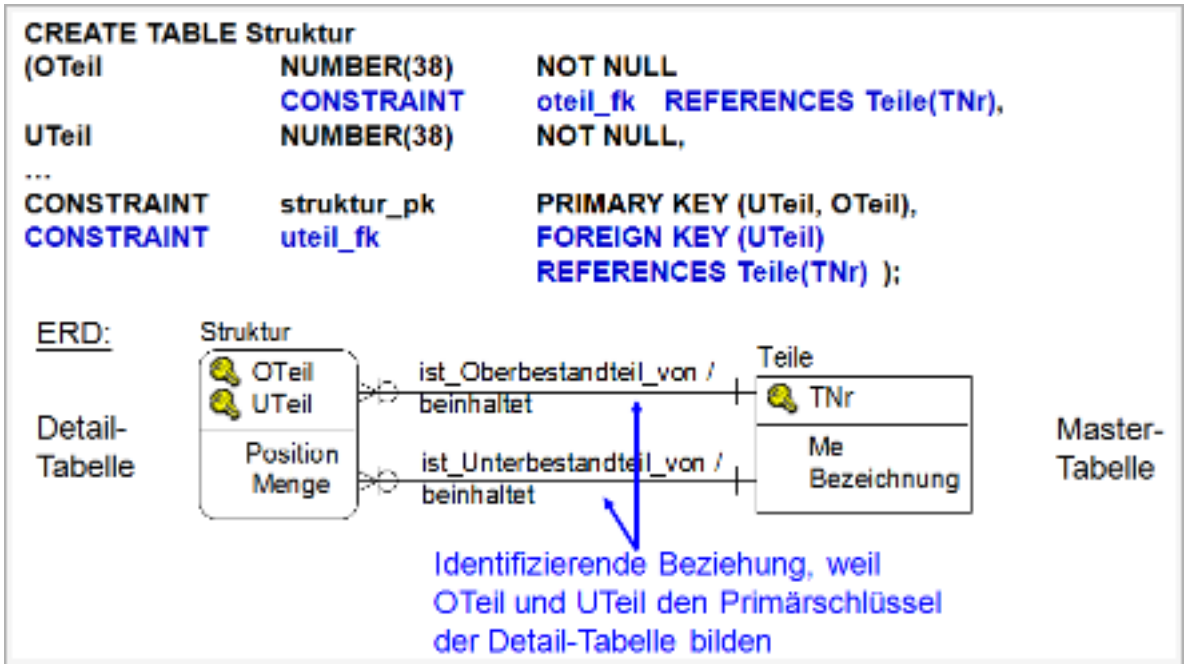


Abbildung 10: Das auf Folie 5.2.9 verwendete ERD

Auf den folgenden Folien wird die technische Umsetzung der Constraints genauer behandelt, insbesondere die Grenzen der in kommerziellen Datenbankmanagementsystemen enthaltenen Methoden zur Integritätsprüfung und das Verhalten des DBMS, wenn ein Constraint verletzt wird, bevor das Thema schließlich mit zusätzlichen Beispielen abgeschlossen wird. In diesen Beispielen wird eine neue Tabelle *Angestellte* verwendet, die den selben Inhalt wie die Tabelle *Mitarbeiter* abbildet, allerdings andere Spaltennamen verwendet.

Insgesamt nimmt der Abschnitt über Constraints 16 der 43 Folien in Kapitel 5.2 ein, was auch dadurch zu begründen ist, dass hier auch allgemeine Datenbank-Konzepte wie Konsistenz oder die genaue Definition einer Transaktion erklärt werden. Die Textinhalte dieser Folien sind überwiegend in Stichpunkten gehalten, und da sie als Begleitmaterial für eine Vorlesung verwendet werden und nicht als eigenständiges Lernmaterial, ist dies auch ausreichend.

Mit den Constraints ist auch das Thema „CREATE TABLE“ abgeschlossen, und das Tutorial kann sich den weiteren CREATE Befehlen widmen. Die CREATE INDEX Anweisung verwendet ebenfalls das Beispiel der Struktur-Tabelle, wird aber nur auf den zwei Folien 5.2.23 und 5.2.24 behandelt, da der Befehl weniger komplex ist als CREATE TABLE. In 5.2.25 „Die Datendefinitionsspache DDL“ folgt ein kurzer Überblick über alle Objekte, deren Erstellung explizit behandelt wird. Hierbei handelt es sich um eine Kopie der Folie 5.2.2, die am Beginn des Kapitels steht, mit dem einzigen Unterschied, dass das Objekt „Table“ in 5.2.25 nicht wie die anderen Objekte in Großbuchsta-

ben geschrieben ist. Die Wiederholung ist an dieser Stelle, nach dem langen Abschnitt zu CREATE TABLE und den Constraints, durchaus sinnvoll, allerdings erfolgt sie erst nachdem der CREATE INDEX Befehl eingeführt wird, für die die Definition und der Nutzen eines Index nützliche Informationen wären.

5.2.25 Die Datendefinitionssprache DDL

Objekt	Beschreibung
Tabelle	Basisrelation als Tabelle
INDEX	Verbessert den Datenzugriff durch die Vereinbarung von Schlüsseln
VIEW	Abfrage an die Datenbank, die als virtuelle Tabelle abgelegt ist
SEQUENCE	Erzeugt fortlaufende Nummer (ORACLE spezifisch)

DDL-Anweisungen:

- CREATE
- DROP
- ALTER
- RENAME

Abbildung 11: Folie 5.2.25 Die Datendefinitionssprache DDL

Die nächste Create-Anweisung, CREATE VIEWS, verwendet wieder die Tabelle *Angestellte* zur Demonstration. Für die aufgeführten Beispiele macht die Wahl der Tabelle wenige Unterschiede, aber gerade aus diesem Grund sollte ein Wechsel möglichst vermieden werden, um das Tutorial konsistent zu gestalten.

Das letzte Objekt in diesem Abschnitt ist die Oracle-spezifische Sequence, die der fortlaufenden Nummerierung von Datensätzen dient. Hierfür wird eine weitere neue Tabelle *Kunden* eingeführt, aber die Erklärung ist kurz gehalten, da die Anwendung der Sequence einfach zu verstehen ist. Darauf folgt eine Übersicht über Objekte, die hier nicht näher behandelt werden, beispielsweise User, Cluster oder Snapshot, bevor auf den letzten 7 Folien die SQL-Befehle DROP, RENAME und ALTER behandelt werden.

5.2.37 „DROP-Anweisung“ und 5.2.38 „RENAME-TABLE-Anweisung“ enthalten beide lediglich einen Dummy-Befehl, der die Syntax des Befehls zeigt, und ein einzelnes Beispiel, keinerlei Erklärungen.

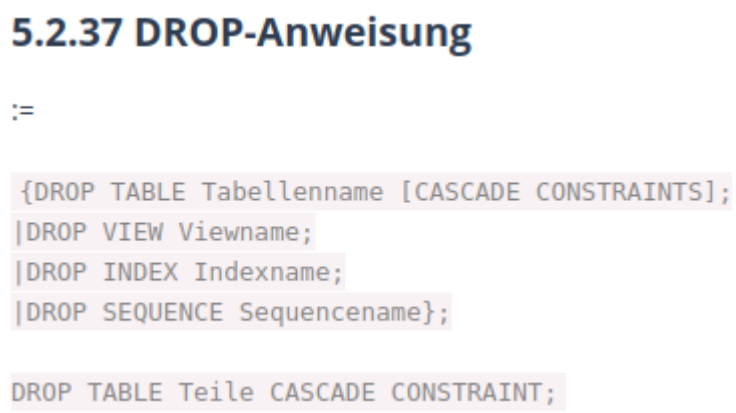


Abbildung 12: Folie 5.2.37 DROP-Anweisung

5.2.38 enthält eine zusätzliche Tabelle zur Verfügbarkeit der Anweisung in verschiedenen Datenbankmanagementsystemen. Wie bereits erwähnt handelt es sich bei den Folien um Begleitmaterial für eine Veranstaltung, in der die Inhalte von einer lehrenden Person erklärt werden, weshalb der Mangel an erläuterndem Text hier keinen schwerwiegenden Mangel darstellt, aber ein schriftlicher Hinweis und eine deutlichere Kennzeichnung der Beispiele wäre dennoch von Nutzen, auch, um die Wiederholung des Vorlesungsmaterials zu erleichtern. Bei ALTER TABLE verhält es sich ähnlich, allerdings sind die Parameter der ALTER TABLE Anweisung, wie MODIFY oder DISABLE, mit Grundkenntnissen der englischen Sprache selbsterklärend, was für CASCADE CONSTRAINTS in der DROP-Anweisung nicht der Fall ist.

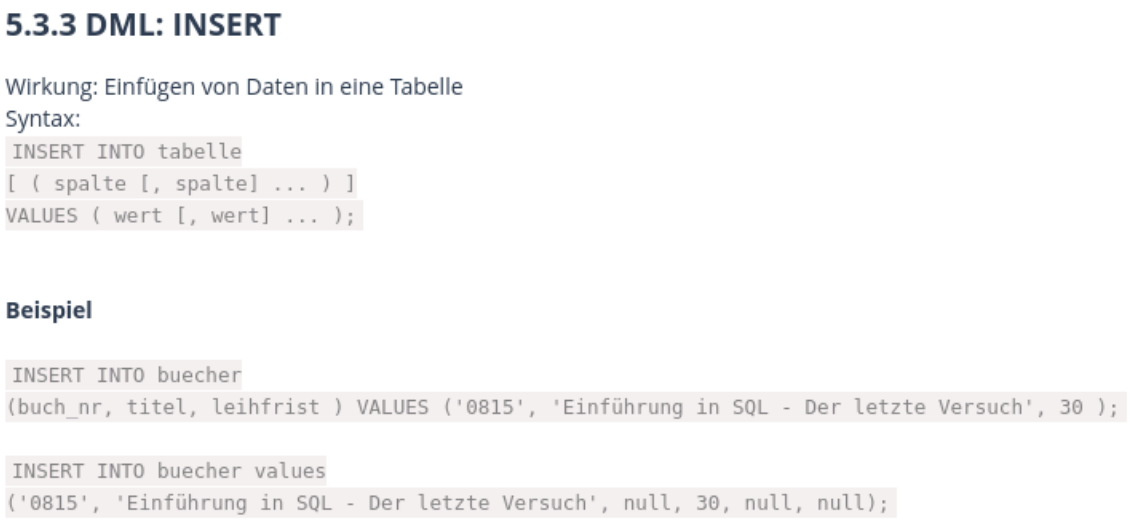
Damit ist Kapitel 5.2 abgeschlossen.

Kapitel 5.3 widmet sich der Data Manipulation Language. Dieses Kapitel enthält nur halb so viele Folien wie das vorhergehende, enthält aber die zwei Übungen des Tutorials. Nach einer kurzen Wiederholung der SQL-Sprachbestandteile wird auf der Folie 5.3.2 eine neue Datenbank vorgestellt, die das Leihsystem einer fiktiven Bibliothek abbildet. Für ihren Zweck als Beispiel in einem Lernsystem ist diese Datenbank gut konzipiert. Sie enthält nur fünf Tabellen, die jeweils nicht mehr als sechs Spalten haben, und ist durch diese Übersichtlichkeit und das anschauliche Beispiel des Leihsystems einfach zu verstehen. Trotzdem enthalten diese Tabellen zumindest die geläufigsten Datentypen (NUMBER, CHAR, VARCHAR, DATE) und haben verschiedene Beziehungen zueinander, wodurch unterschiedliche Abfragen an die Datenbank möglich sind.

Fraglich ist jedoch, ob eine weitere Datenbank für diesen Abschnitt nötig war, oder ob die Einführung von neuen Tabellen und Beziehungen eine unnötige kognitive Last für die Studierenden darstellt. Auch wenn die Datenbank nicht schwer zu verstehen ist, nimmt es trotzdem Zeit und mentale Kapazitäten in Anspruch, sich mit diesem neuen Beispiel zu befassen, die für das eigentliche Ziel des Tutorials, die DML, verwendet werden könnten.

Selbst kleine Inkonsistenzen zwischen den Datenbanken können eine unnötige Ablenkung darstellen. Beispielsweise verwendet die Mitarbeiter-Datenbank Tabellennamen im Singular („Abteilung“, „Projekt“), während die Spalten in der Bibliothekstabelle im Plural benannt sind („Buecher“, „Strafen“). Unabhängig voneinander betrachtet, sind beide Namensgebungen vollkommen legitim, aber als Teile eines zusammenhängenden Tutorials sind sie inkonsistent und können zu Verwirrung und Fehlern führen, die nicht auf einem mangelnden Verständnis von SQL beruhen.

Folie 5.3.3 „DML: INSERT“ hat die doppelten Nutzen, den Insert-Befehl zu zeigen, und die Tabellen aus der letzten Folie mit Daten zu füllen, die für die folgenden Aufgaben verwendet werden können. Diese Folie ist inhaltlich ähnlich zu den DROP und RENAME Tabellen aus dem letzten Kapitel, bietet aber auch ein Gegenbeispiel zu dem dort identifizierten Problem, denn die INSERT-Folie hat sowohl eine Kurzerklärung des Befehls als auch Beschriftungen für die Beispiele, und in einer Gegenüberstellung wird deutlich, dass diese geringfügigen Änderungen die Folie wesentlich verständlicher machen.



5.3.3 DML: INSERT

Wirkung: Einfügen von Daten in eine Tabelle
Syntax:

```
INSERT INTO tabelle
[ ( spalte [, spalte] ... ) ]
VALUES ( wert [, wert] ... );
```

Beispiel

```
INSERT INTO buecher
(buch_nr, titel, leihfrist ) VALUES ('0815', 'Einführung in SQL - Der letzte Versuch', 30 );

INSERT INTO buecher values
('0815', 'Einführung in SQL - Der letzte Versuch', null, 30, null, null);
```

Abbildung 13: Folie 5.3.3 - DML: INSERT

Die Delete- und Update-Anweisungen werden auf den folgenden zwei ähnlich aufgebauten Folien erklärt. Die Select-Anweisung erhält eine längere Erklärung über insgesamt sieben Seiten, die auf die verschiedenen Schlüsselwörter eingehen, die in Verbindung mit SELECT verwendet werden können, um Anfragen zu präzisieren.

Bei der folgenden Folie handelt es sich um eine Übung, die aus neun Aufgaben besteht. Jede dieser Aufgaben besteht darin, eine SELECT-Anweisung an die Bibliotheks-Datenbank zu formulieren. Die Aufgaben sind ungefähr nach steigender Komplexität angeordert, bei Aufgabe 2 handelt es sich zum Beispiel um eine erweiterte Version der ersten Aufgabe: In Aufgabe 1 wird Summe der Ausleihzahlen pro Autor ermittelt, und in Aufgabe 2 werden nur die Autoren mit einer Ausleihsumme größer als 7 ausgegeben. Die einzige Ausnahme zu dieser Regel ist, dass Aufgabe 6, „Wie sieht die prozentuale Verteilung der Bücher auf die einzelnen Gruppen aus? Bitte legen Sie dazu mit CREATE TABLE name AS die Tabellen gruppen_anzahl und gesamt_anzahl an und löschen diese anschließend!“, komplexer als die folgenden drei Aufgaben ist. Diese leichte Inkonsistenz im Schwierigkeitsgrad sollte jedoch kein Problem darstellen, und die Aufgaben bieten eine Gelegenheit, die bisher behandelten Begriffe und Schlüsselwörter praktisch zu verwenden und sollte dazu beitragen, ihr Verständnis zu verbessern und im Gedächtnis zu festigen.

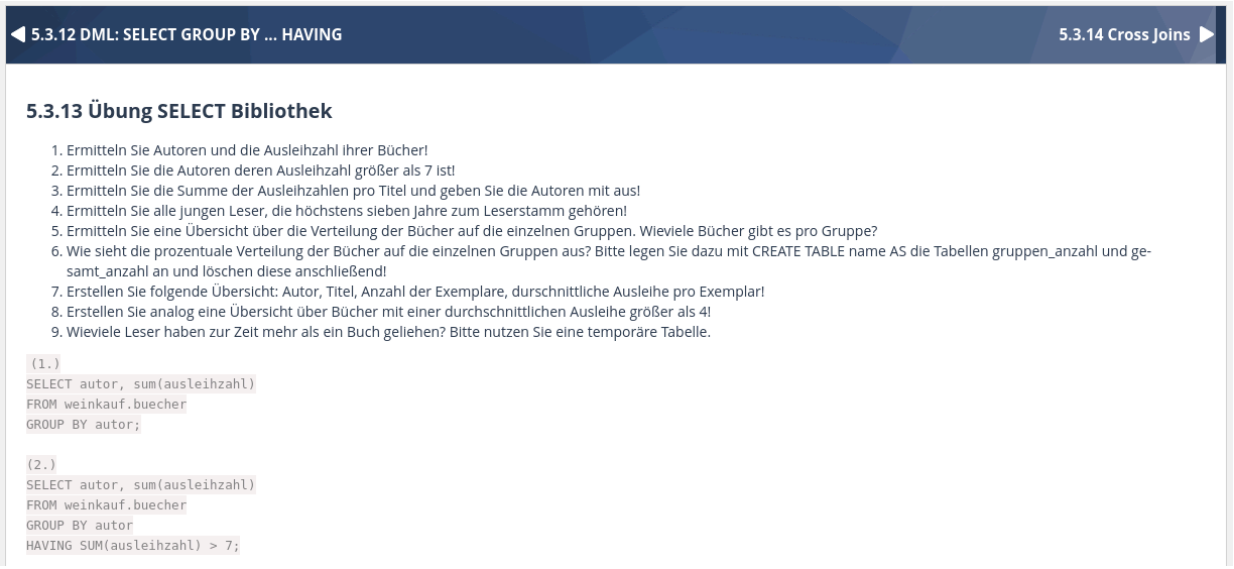


Abbildung 14: Folie 5.3.13 Übung SELECT Bibliothek

Allerdings ist es nicht ideal, dass die Fragen und Antworten sich auf der selben Seite befinden. Die Antworten für die ersten Aufgaben sind sofort sichtbar, wenn diese Folie angezeigt wird, weshalb es leicht vorkommen kann, dass diese unabsichtlich gelesen werden, wodurch das lernen gehindert wird. Außerdem müssen die Nummern der Aufgaben konsultiert werden, um die Lösungen den Fragen zu zu ordnen, und wenn der Benutzer nach unten scrollt, um die späteren Lösungen anzuzeigen, sind die dazugehörigen Aufgaben nicht mehr sichtbar.

Nachdem auf den folgenden Seiten Joins, Subqueries und die Schlüsselwörter UNION, INTERSECT und EXCEPT in sehr ausführlichen Folien behandelt werden, folgt eine zweite Übung, die diese neuen Elemente behandelt. Sie ist wie die erste Übung aufgebaut, nur die visuelle Formatierung ist inkonsistent. Wie bei der ersten Übung findet

sich auch hier die komplexeste Aufgabe in der Mitte der Übung, in diesem Fall ist es Aufgabe 4, „Welche Leser haben zur Zeit aus allen Buchgruppen entliehen?“, und auch die Lösungen befinden sich wieder auf der selben Seite. Da die SQL-Anweisungen für diese Aufgaben länger sind als in der ersten Übung, ist es zumindest unwahrscheinlicher (wenn auch nicht ausgeschlossen), dass ein Betrachter die Lösung unfreiwillig aufnimmt, und die Lösungen sind in einem unabhängig scrollbaren Block enthalten, wodurch die Aufgaben sichtbar bleiben, wenn die Lösungen betrachtet werden.

Die Abschnitte DCL und Data Dictionary sind kurz gehalten und überwiegend theoretischer Natur, die Anweisungen der DCL Sprache werden nur kurz an Beispielen gezeigt. Die Folien zu den einzelnen Befehlen sind hier etwas inkonsistent, die Anweisungen GRANT und REVOKE haben keine Erklärungen, sondern nur eine Beschreibung der Syntax, die Befehle COMMIT und ROLLBACK teilen sich eine Folie, haben aber beide einzelne Erklärungen ihrer Funktion.

Folie 5.4.4 „Beispiel Pool User“ ist ohne zusätzliche verbale Erklärung schwer zu durchblicken. Sie besteht aus einem beispielhaften SQL-Skript, dass einen neuen Datenbanknutzer erstellt, diesem Nutzer Rechte zuweist und den Nutzer wieder löscht. Das Skript ist jedoch nicht kommentiert und besonders der erste Befehl,

```
CREATE USER “GAST01”
IDENTIFIED BY “gast000”
DEFAULT TABLESPACE “USER_DATA”
TEMPORARY TABLESPACE “TEMPORARY_DATA”
PROFILE DEFAULT ACCOUNT UNLOCK;
```

hat auch keine Erklärung für seine relativ komplexe Syntax.

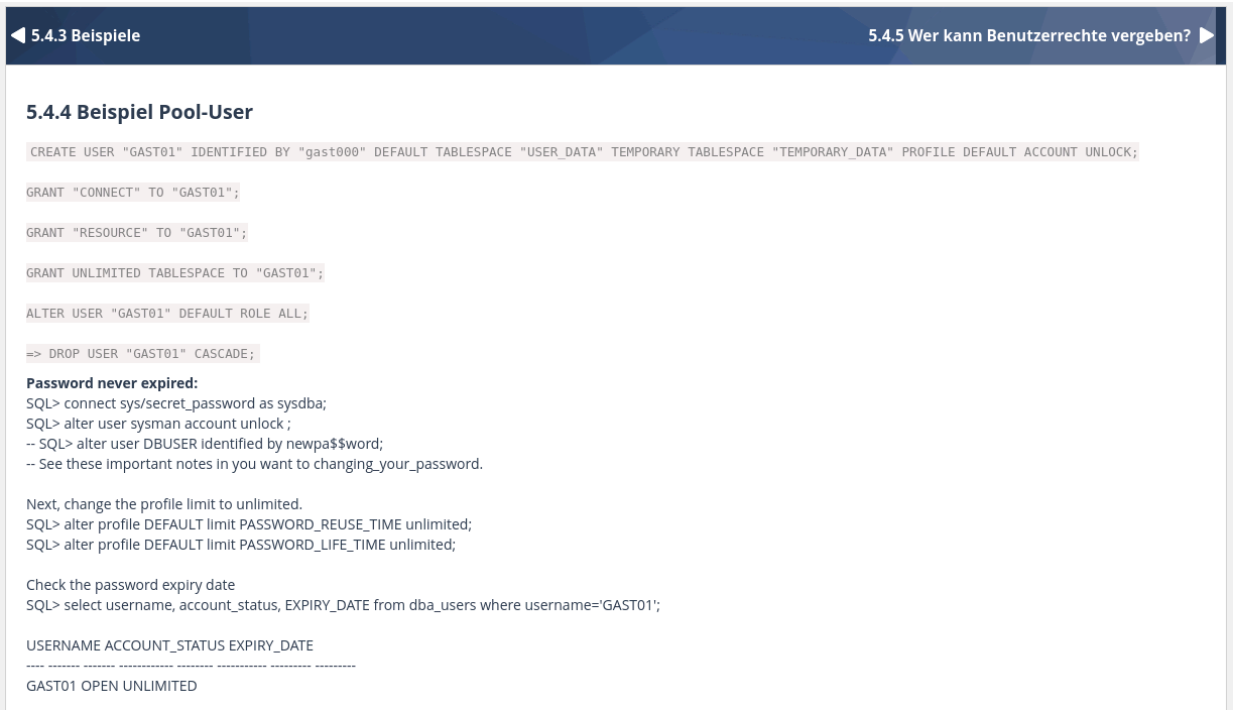


Abbildung 15: Folie 5.4.4 Beispiel Pool-User

Darauf folgt ein Abschnitt in englischer Sprache. Er beschreibt den Prozess, um die Einstellungen eines Benutzeraccounts so zu ändern, dass das Passwort dieses Accounts für eine unbegrenzte Zeit gültig ist, aber dieses Ziel wird nicht direkt erklärt. Die Überschrift lautet “Password never expired”, aber da diese Überschrift in der Vergangenheitsform formuliert ist, ist sie auf ersten Blick nicht als Zielsetzung erkennbar und wirkt eher wie eine Beschreibung des Ausgangszustands. Das ist nicht die einzige grammatikalische Eigenheit hier, in der Beschreibung selbst findet sich auch der Satz “See these important notes in you want to changing_your_password”. Nicht nur ist die Grammatik dieses Satzes falsch, es ist auch nicht klar, auf welche Notizen oder Hinweise sich er sich bezieht.

Nach der Beschreibung der Vorgehensweise folgt eine kurze Erklärung dazu, wie überprüft werden kann, dass die Passworteinstellungen tatsächlich geändert wurden, gefolgt von der erwarteten Konsolenausgabe, wenn das Passwort korrekt eingestellt ist. Problematisch ist allerdings, dass diese Textabschnitte alle unformatiert sind, wodurch sie sich visuell nicht voneinander abheben, und da sie auch nicht beschriftet sind, ist der Inhalt dieses Abschnittes nur schwer verständlich.

Unterkapitel 5.5. “Data Dictionary” ist sehr kurz, aber durchaus ausreichend. Das Data Dictionary kann wie jede andere Tabelle abgefragt werden, es gibt hier also keine neuen Befehle, und das Konzept ist knapp, aber deutlich erklärt.

Der letzte Abschnitt, Kapitel 6.1, “Transaktionen”, ist für diese Arbeit nur noch teilweise relevant, da er sich nicht mehr mit dem Tutorial und der Anwendung von SQL befasst, sondern die grundlegende Theorie der Datenbanktransaktionen und die Probleme, die im Mehrbenutzerbetrieb auftreten behandelt. An dieser Stelle soll trotzdem erwähnt werden, dass sich wenigstens eines der in Kapitel 5 identifizierten Probleme in Kapitel 6 fortsetzt: Die grafischen Darstellungen sind weiterhin durch ihre niedrige Auflösung und eng zusammengeschriebenen Text schwer zu lesen, Alternativtexte sind ebenfalls nicht vorhanden. Auch die dargestellten Bilder selbst haben Möglichkeiten zur Verbesserung.

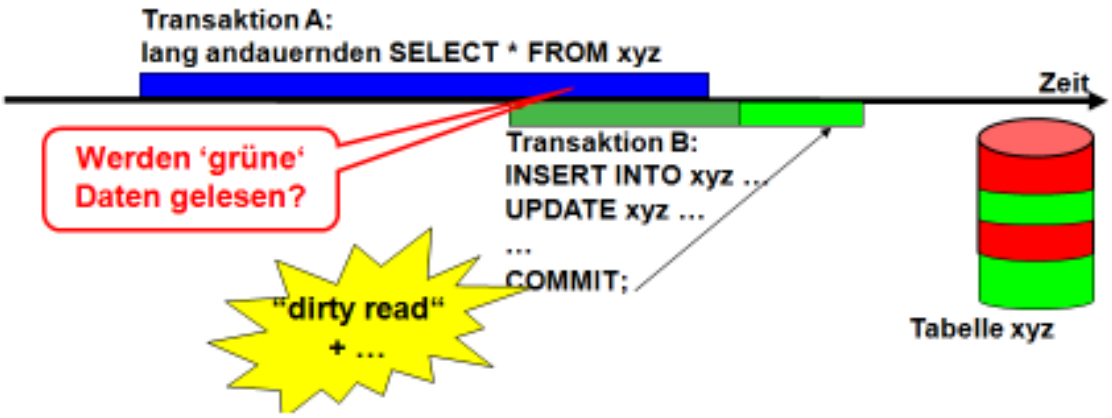


Abbildung 16: Grafische Darstellung eines "Dirty Reads", zu sehen auf Folie 6.1.5

Die oben stehende Grafik enthält mehrere Bildelemente, die den Blick stark auf sich lenken, aber nur wenige Informationen enthalten. Auf der rechten Seite ist ein buntes Datenbanksymbol als visuelle Repräsentation der bearbeiteten Daten zu sehen, aber die Inklusion dieses Symbols trägt nur wenig zu dem Verständnis bei: Das es sich hier um Transaktionen an einer Datenbank handelt, ist bereits durch das Thema der Vorlesung etabliert. Das große farbige Objekt, noch dazu in den stark kontrastierenden Farben hellgrün und rot, lenkt dennoch den Blick stark auf sich. Das zweite auffällige Bildelement ist die gelbe gezackte Form mit der Beschriftung “ “dirty read” + … ”. Die Form hier ist visuell sehr aufdringlich. Der Textinhalt ist wichtig, da es der Name des dargestellten Phänomens ist, aber die Darstellung als gelbe Form unterhalb des Zeitstrahls ist fragwürdig.

7 Identifizierung der Schwachstellen

In dieser Betrachtung haben sich einige Verbesserungsmöglichkeiten an dem SQL Tutorial herauskristallisiert. Inhaltlich gibt es nur wenig auszusetzen, alle nötigen Befehle und Konzepte sind vorhanden, und die Reihenfolge, in der sie behandelt werden, ist sinnvoll gewählt. Die größte inhaltliche Schwachstelle ist die Verwendung verschiedener Beispieldatenbanken im Verlauf des Tutorials und die dadurch verursachte zusätzliche mentale Last. Einheitlichere Beispiele wären wünschenswert.

Mehr Mängel finden sich dagegen in der äußerlichen Präsentation dieser Inhalte. Jeder dieser Mängel ist individuell belanglos, zusammengenommen erschweren sie jedoch das Verständnis der gut zusammengesetzten Inhalte.

Die identifizierten Verbesserungsmöglichkeiten sind wie folgt:

- Nicht einheitliche Beispiele: Die verwendete Beispieldatenbank wird in der Mitte des Tutorials ausgetauscht, wenn der durchgehende Einsatz einer Beispieldatenbank einfacher zu verstehen wäre.
- Inkonsistente oder mangelnde Formatierung: Einige der Folien sind übersichtlich gegliedert und formatiert, auf anderen fehlt eine visuelle Gliederung der Inhalte.
- Fehlende Erklärungen: Auf mehreren Folien sind Codeblöcke ohne Kommentare oder Erläuterung ihrer Funktion abgebildet.
- Unzureichendes Bildmaterial: Die grafischen Elemente sind oft schwer zu interpretieren und eine niedrige Auflösung erschwert das Lesen. Außerdem fehlen Alternativtexte.
- Schlecht gewähltes Bildmaterial: Einige der oben genannten grafischen Elemente sind grundlegend unnötig, es handelt sich um Tabellen, die als Bildelement eingebunden sind, anstatt die Tabelle mithilfe des im Ilias-Systems vorhandenen Tabellenelement zu erstellen.
- Rechtschreibung: Gelegentlich finden sich Rechtschreib- oder Grammatikfehler in den Texten des Tutorials. Diese sollten natürlich korrigiert werden, aber der weitere Text dieser Arbeit wird nicht auf einzelne Tippfehler eingehen.

8 Entwicklung der neuen Tutorialversion

Viele Grafiken wurden vereinfacht oder zumindest durch höher auflösende Bilder ersetzt, beispielsweise das Zeitstrahldiagramm zur Erklärung des „dirty reads“ in Kapitel 6:

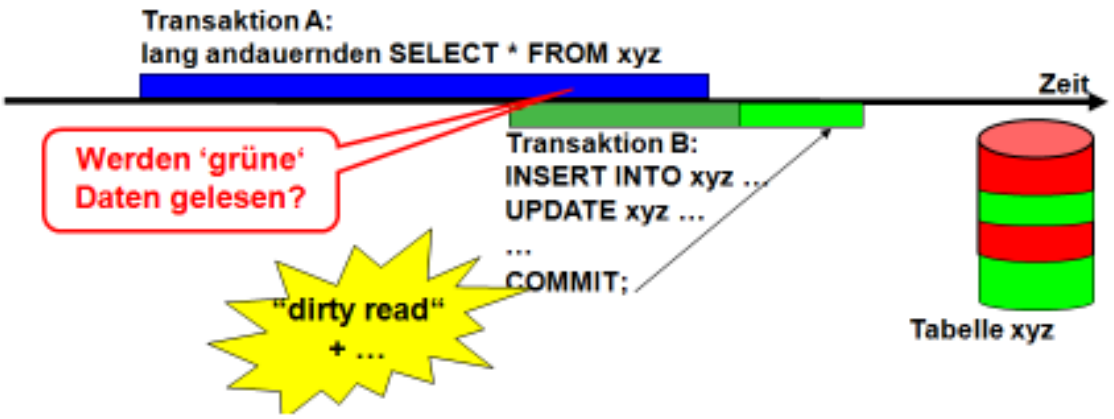


Abbildung 17: Ursprüngliche Version des "dirty read" Diagramms

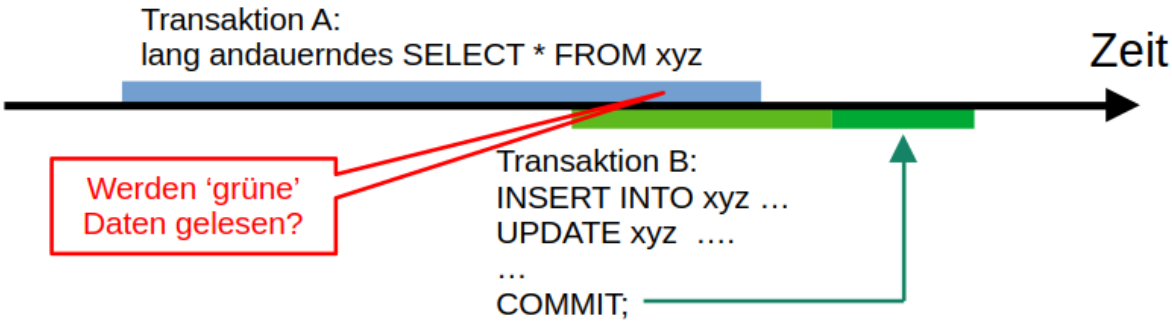


Abbildung 18: Vereinfachte und höher auflösende Version des "dirty read" Diagramms

Dabei wurde darauf geachtet, den Inhalt der Diagramme beizubehalten, und lediglich ihre äußere Präsentation zu verbessern, um das Auffassen dieser Inhalte einfacher zu gestalten. Wenn inhaltliche Veränderungen vorgenommen wurden (wie das Entfernen der Datenbank-Symbols im obigen Beispiel) dann nur, um die Informationsdichte des Bildmaterials zu erhöhen, indem redundante Elemente entfernt werden.

Ein anderes Beispiel hierfür ist das Diagramm für das 2-Phasen-Sperrprotokoll:

Abbildung 19: Das alte zwei-Phasen-Sperrprotokoll Diagramm

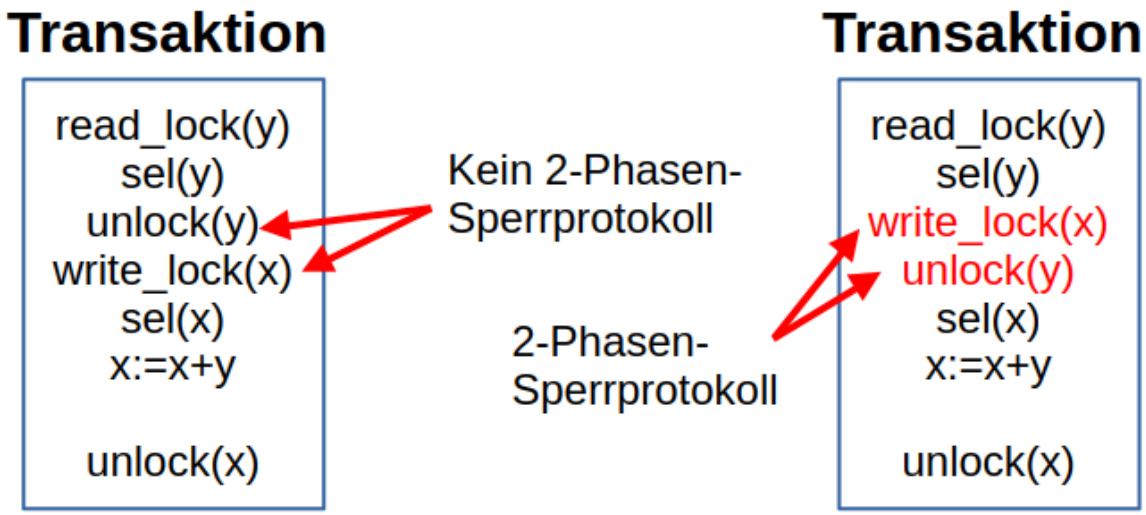
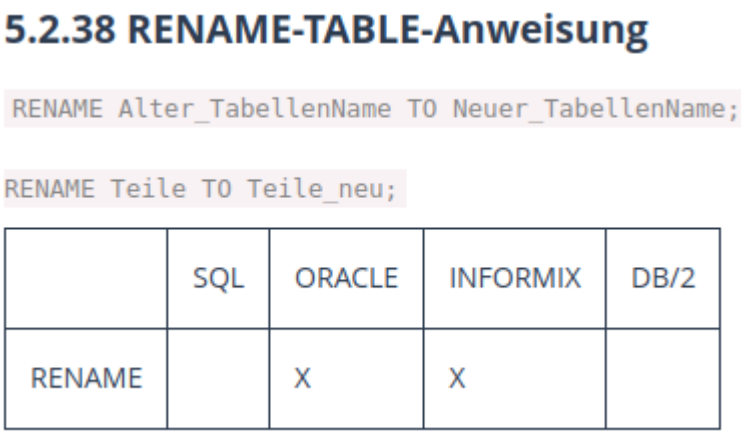


Abbildung 20: Die neue Version der selben Grafik

Auf der alten Version sind jeweils zwei Transaktionen auf jeder Seite zu sehen. Natürlich ist das Zwei-Phasen-Sperrprotokoll für den Mehrbenutzerbetrieb gedacht und kommt zum Tragen, wenn mehrere Transaktionen durchgeführt werden sollen, allerdings sind die beiden Transaktionen hier jeweils identisch, aber komplex genug, dass dies nicht auf einen Blick ersichtlich ist. Erschwerend kommt hinzu, dass an anderen Stellen des Tutorials Transaktionen nebeneinander gestellt werden, um darzustellen, dass sie zeitgleich durchgeführt werden, die Anordnung in Abbildung 14 suggeriert also, dass dies hier auch der Fall ist, bei den gegebenen Transaktionen ist dies jedoch nicht möglich. Sobald eine der beiden Transaktionen ein write_lock durchführt, wird die zweite Transaktion blockiert, bis das Objekt wieder für Schreibaktionen entsperrt wird.

Auch die Textelemente wurden neu formatiert, um ihre Lesbarkeit zu verbessern. Die ursprüngliche Folie zur RENAME TABLE Anweisung benötigte eine begleitende verbale Erklärung um vollständig verständlich zu sein. Zwar lässt sich vermuten, was die einzelnen Elemente der Folie bedeuten, aber idealerweise sollten solche Vermutungen durch den Lesenden nicht nötig sein.



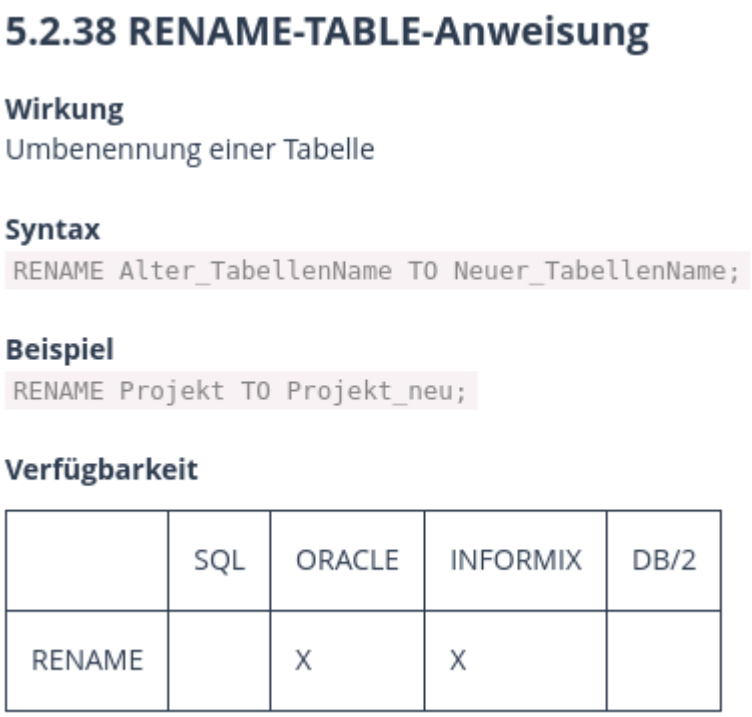
5.2.38 RENAME-TABLE-Anweisung

```
RENAME Alter_TabellenName TO Neuer_TabellenName;
```

```
RENAME Teile TO Teile_neu;
```

	SQL	ORACLE	INFORMIX	DB/2
RENAME		X	X	

Abbildung 21: Die alte Folie "RENAME TABLE Anweisung"



5.2.38 RENAME-TABLE-Anweisung

Wirkung
Umbenennung einer Tabelle

Syntax
```
RENAME Alter_TabellenName TO Neuer_TabellenName;
```

Beispiel
```
RENAME Projekt TO Projekt_neu;
```

Verfügbarkeit

	SQL	ORACLE	INFORMIX	DB/2
RENAME		X	X	

Abbildung 22: Die angepasste Version der "RENAME TABLE Anweisung" Folie

Durch das Einfügen der fett gedruckten Überschriften wird der Inhalt der Folie benannt und visuell stärker gegliedert, und der hinzugefügte Punkt „Wirkung“ macht den Nutzen des Befehls unmissverständlich klar. Ähnliche Änderungen wurden an mehreren Folien durchgeführt, vor allem an denjenigen, die sich mit der Verwendung bestimmter SQL Befehle befassen.

Die größte Veränderung lag jedoch in der Anpassung der verwendeten Beispieldatenbank. Die Mitarbeiter-Datenbank, die in der ersten Hälfte des Tutorials eingeführt wird, soll zukünftig für das gesamte Tutorial verwendet werden, und dafür ist es notwendig, dass an dieser Datenbank auch alle Übungen äquivalent durchgeführt werden können, für die bisher die Bibliotheks-Datenbank verwendet wurde.

Die Entity-Relationship Diagramme beider Datenbanken sind in den folgenden Abbildungen dargestellt.

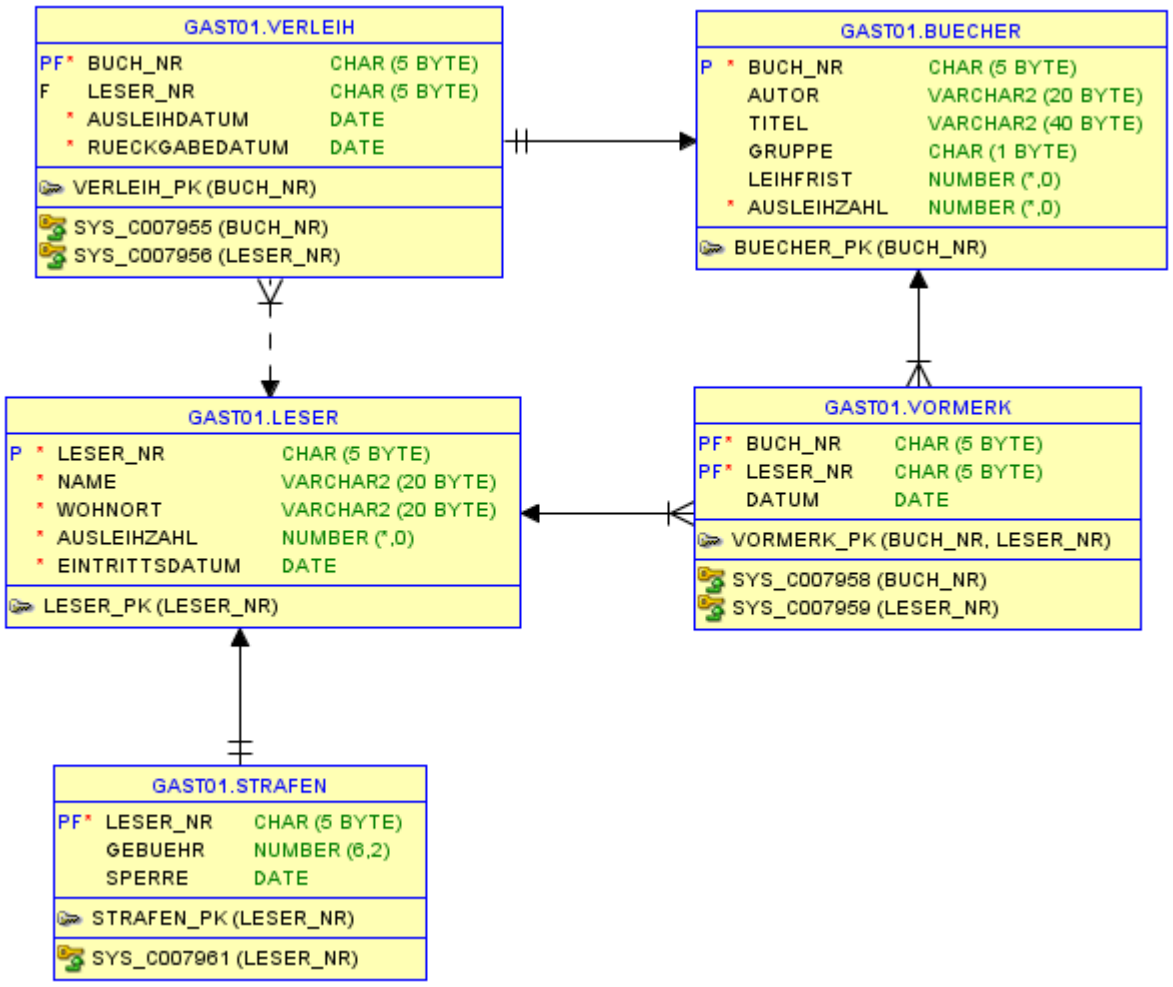


Abbildung 23: ERD der Datenbank "Bibliothek"

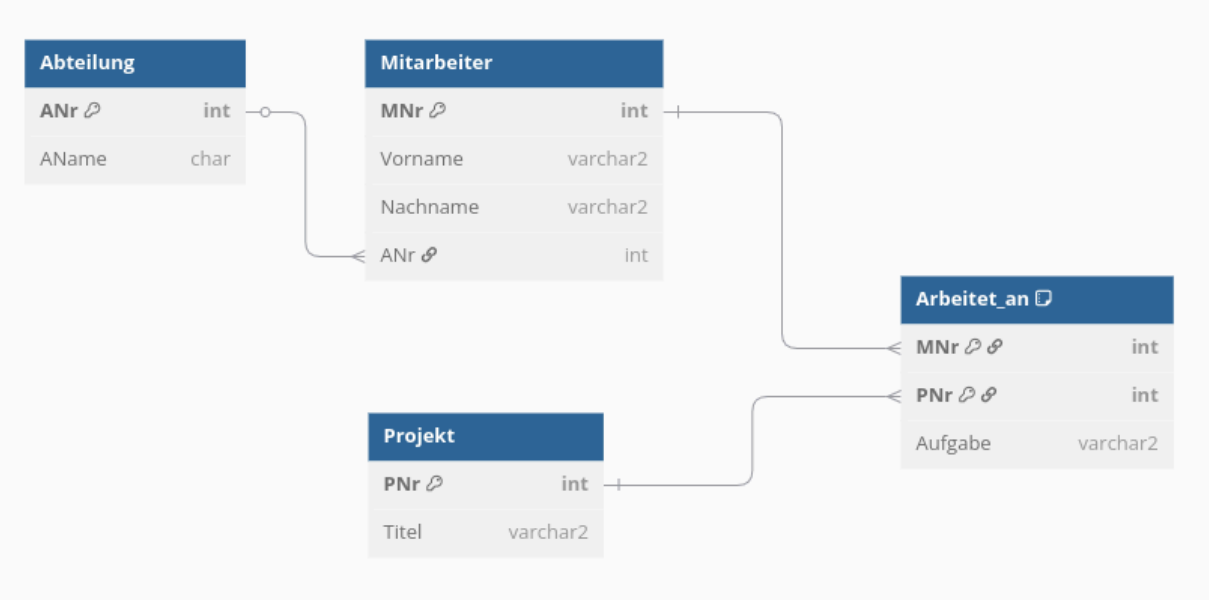


Abbildung 24: ERD der Datenbank "Mitarbeiter"

An diesen Beispielen ist auch der Unterschied zwischen den älteren Darstellungen der ERDs und dem moderneren grafischem Design, mit dem sie ersetzt wurden, zu sehen.

Auch wenn den Datenbanken verschiedenen Inhalte zugrunde liegen, gibt es trotzdem einige Gemeinsamkeiten zwischen ihnen. Beide Datenbanken enthalten eine Tabelle, die personenbezogene Daten enthält (Leser und Mitarbeiter), und beide Datenbanken verfügen über eine Tabelle, die die Personen mit einem anderen Sachverhalt in Verbindungen bringen (die Tabellen Verleih und Vormerk, die die Leser mit Büchern verknüpfen, und die Tabelle Arbeitet_an, die Mitarbeitern den verschiedenen Projekten zuordnet.

Mit diesen Grundlagen können einige der Übungsaufgaben der Bibliothekstabelle schon analog an der Mitarbeitertabelle durchgeführt werden. Aufgabe 5 der ersten Übung, „Ermitteln Sie eine Übersicht über die Verteilung der Bücher auf die einzelnen Gruppen. Wieviele Bücher gibt es pro Gruppe?“ fragt danach, die einzelnen Zeilen einer Tabelle nach dem Inhalt einer bestimmten Spalte zu gruppieren und zu zählen. In der Mitarbeiter-Tabelle ist dies mit den Mitarbeitern möglich, da sie verschiedenen Abteilungen zugeordnet sind. Die beiden nötigen Befehle

```
SELECT gruppe, COUNT(*)
FROM weinkauf.buecher
GROUP BY gruppe;
```

und

```
SELECT Anr, COUNT(*)
FROM Mitarbeiter
GROUP BY Anr;
```

sind abgesehen von den Namen der verwendeten Tabellen und Spalten identisch und können deshalb als gleichwertig betrachtet werden.

Eine weitere Kategorie von Übungsaufgaben besteht aus Abfragen, die an der gegebenen Mitarbeiter-Datenbank nicht umsetzbar sind, die jedoch durch das Hinzufügen einer einzigen, inhaltlich sinnvollen, Spalte umsetzbar werden. Ein gutes Beispiel hierfür findet sich schon in der erste Aufgabe der ersten Übung, „Ermitteln Sie Autoren und die Ausleihzahl ihrer Bücher!“. In der Mitarbeiter-DB, die in dem vorhergehenden ERD dargestellt wurde, gibt es keine Spalte, die in dieser Art aufsummiert werden können. Die einzigen Spalten mit numerischen Werten sind die jeweiligen Identifikationsnummern, und aus diesen eine Summe zu bilden, ist praktisch nicht sinnvoll. Eine sinnvollere Anwendung wäre es, die Summe aller Gehälter in jeder Abteilung zu ermitteln, was sich an dieser Übungsdatenbank leicht umsetzen lässt, indem die Mitarbeiter-Tabelle durch eine Gehalts-Spalte erweitert wird.

In der Bibliotheks-DB ist auch eine Spalte *Wohnort* in der Lesertabelle vorhanden, die für einige Aufgaben verwendet wird, und diese Spalte lässt sich direkt in die Mitarbeitertabelle übertragen. Außerdem wurden der Tabelle *Abteilung* eine Spalte namens „Stellen_Offen“ und der Tabelle *Projekt* eine Spalte „Geplante_Dauer“ hinzugefügt, um weitere numerische Spalten zu schaffen, an denen Vergleichsoperationen durchgeführt werden. Da der Datentyp DATE komplexer als Typen wie VARCHAR oder INT ist, sollte dieser auch in dem Tutorial behandelt werden, und hierfür wurde außerdem die Tabelle *Projekt* um eine Spalte „Datum_Beginn“ ergänzt.

Damit bleibt noch eine Eigenheit der Bibliotheks-DB übrig, die in der Mitarbeiter-DB kein Äquivalent hat: Die Tabelle *Strafen*. Die Strafen-Tabelle steht in einer 1:1 Beziehung zu der Tabelle *Leser*. Nicht jeder Leser ist in der Straf-Tabelle vorhanden, aber diejenigen, die einen Eintrag haben, haben genau einen zugewiesenen Strafbetrag. Die Mitarbeiter-Tabelle hat dafür kein Äquivalent, weshalb eine weitere Tabelle namens *Abzuege* hinzugefügt wurde, die sich wie die Strafen-Tabelle verhält. Um die Datenbank trotzdem so unkompliziert wie möglich zu halten, wird die Strafen-Tabelle erst nach der ersten Übung erstellt. Dadurch haben die Teilnehmer Zeit, sich zuerst mit der einfacheren Struktur mit nur 4 Tabellen vertraut zu machen, statt sofort mit einer fünften Tabelle konfrontiert zu werden, die in der ersten Übungsaufgabe nicht verwendet wird.

Mit diesen Änderungen ergibt sich folgendes erweitertes ERD für die Mitarbeiter-DB:

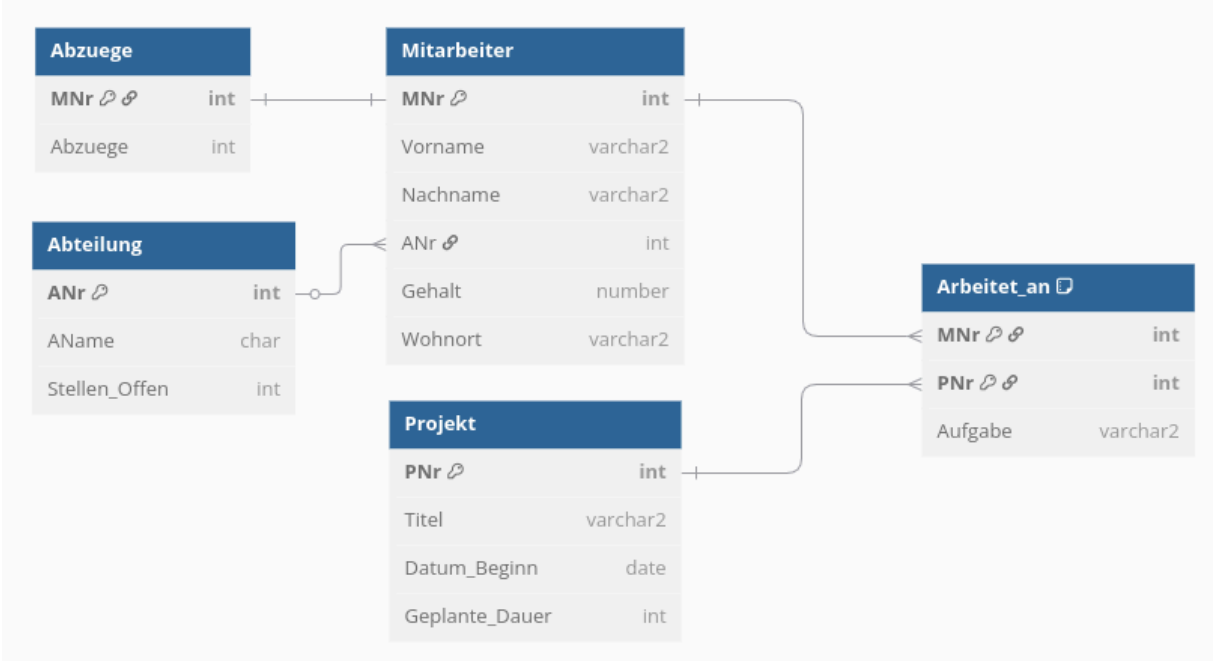


Abbildung 25: Das erweiterte ERD der Mitarbeiter-Datenbank

IV Evaluierung

9 Beobachtungen aus dem Praxis-Test

Für den Praxis-Einsatz sollte beachtet werden, dass das Tutorial vom Autor dieser Arbeit durchgeführt wurde, und ich keine Lehrausbildung- oder Erfahrung habe. Schwierigkeiten mit der Darstellung der Inhalte können in diesem Fall also nicht nur auf die Struktur des Tutorials, sondern auch auf die mangelnde Erfahrung der Lehrperson zurück zu führen sein.

Die erste Vorlesung in der Veranstaltung wurde von Prof. Dr. Weinkauf gehalten und diente als Einführung in das Thema Datenbanken, ihrer Geschichte und den Grundlagen ihrer industriellen Verwendung. Danach wurde das Tutorial in 2 Vorlesungen (insgesamt 3 Stunden Unterrichtszeit) von Bianca Mayer durchgeführt. Zwischen den zwei Vorlesungen des Tutorials lagen aufgrund des Osterwochenendes 14 Tage Abstand.

Die zwei Veranstaltungen waren für den behandelten Inhalt ausreichend, aber recht knapp bemessen. Da durch die unübliche Anordnung der Unterrichtsinhalte in diesem Semester einige Begriffe noch nicht von der theoretischen Seite behandelt wurden, waren an einigen Stellen zusätzliche Erklärungen nötig, die in dem erstellten Material nicht vorgesehen waren, beispielsweise zu Begriffen der Mengenlehre. Sollte das Tutorial weiterhin zu Beginn der Veranstaltung durchgeführt werden, sollte in Betracht gezogen werden, einige Inhalte aus späteren Unterrichtseinheiten hierher zu verschieben und den Zeitraum für die SQL Einheit entsprechend auszuweiten. Insbesondere Entity-Relationship-Diagramme sind für die Übungen ein sehr wichtiges Hilfsmittel, und da besonders die praktischen Anteile des Tutorials von den Studierenden als hilfreich eingeschätzt wurden, möchte ich hier den Vorschlag einbringen, das ERD für die Mitarbeiter-Datenbank zuerst in der Veranstaltung gemeinsam zu erarbeiten, um den Aufbau eines ERDs zu erklären, die Partizipation der Lernenden zu fördern, und zu gewährleisten, dass sie sich engagiert mit dem Aufbau der Datenbank befassen, bevor diese in den Übungsaufgaben verwendet wird. Damit kann auch eine Einführung zu Beziehungstypen verbunden werden.

Im Bezug auf das ERD wäre es auch hilfreich, wenn das ERD auf den Seiten der Übungsaufgaben zu sehen wäre, um eine visuelle Referenz für die Beziehungen zwischen den Tabellen und den enthaltenen Spalten zu bieten. Die grobe Struktur der Datenbank war für die Studierenden zwar einfach zu verstehen, aber den genauen Aufbau ohne Referenz zu reproduzieren, erschwerte merklich das Lösen der Aufgaben.

Die Übungen wurden kollektiv gelöst, und nur wenige Studierende nahmen aktiv an dem Gespräch teilt. Obwohl die Aufgaben in einer angemessenen Zeit korrekt gelöst wurden, ist es aus diesem Grund trotzdem schwer festzustellen, ob die Kursteilnehmer im die Aufgaben im Durchschnitt gut verstanden haben, oder ob diejenigen, die aktiv an der Diskussion beteiligt waren, eine Ausnahme sind.

Ein weiterer Mangel, der in dem Praxistest festgestellt wurde, ist, dass in dem Abschnitt über Join-Bedingungen eine visuelle Illustration der Schnittmengen die Erläuterung der verschiedenen Joins wesentlich erleichtert. In der Vorlesung wurde dies spontan mit Hilfe der Kreidetafel behoben, aber dauerhaft wäre eine Grafik im Vorlesungsmaterial wünschenswert.

10 Analyse des studentischen Feedbacks

Der Feedback-Fragebogen, der am Ende des zwei-wöchentlichen Tutorials ausgehändigt wurde, besteht aus fünf Fragen und einem Feld für freies Feedback.

Frage 1 dient zur Selbsteinschätzung der bereits vorhandenen SQL-Kenntnisse der Studierenden, mit den möglichen Antworten „Ja“, „Nein“ und „Ein wenig“, wobei „Ein wenig“ die mit Abstand häufigste Antwort ist. Da es sich hier um eine Selbsteinschätzung handelt, kann diese ungenaue Beschreibung natürlich ein weites Spektrum umschließen, zeigt aber trotzdem deutlich an, dass die meisten Studenten vorher zumindest flüchtigen Kontakt mit dem Thema der Veranstaltung hatten.

Die zweite Frage besteht darin, das Tutorial allgemein auf einer Skala von 1 (schlecht) bis 10 (sehr gut) einzuschätzen. Die Rückmeldung hier fiel überwiegend positiv aus. Die schlechteste vergebene Einschätzung ist eine Antwort 6, der Durchschnitt liegt bei 8,17. Der Schwierigkeitsgrad des Tutorials wird in Frage 3 von sämtlichen Befragten als angemessen eingeschätzt. Da in diesem Praxis-Versuch das Tutorial zu Beginn des Kurses durchgeführt wurde, statt wie üblich im mittleren Verlauf der Veranstaltung, soll mit Frage 4, „Hattest du das Gefühl, dass dein Verständnis durch den Mangel an theoretischen Grundlagen beeinträchtigt wurde?“, bestimmt werden, wie dieser Umstand von den Studierenden bewertet wird. Die Hälfte von ihnen gaben an, dass sie sich zumindest teilweise beeinträchtigt fühlten. Um zu bestimmen, ob die praktische Einleitung trotz dieser Schwierigkeiten sinnvoll ist, wäre es nötig, eine zweite Befragung am Ende der Veranstaltung durchzuführen, wodurch überprüft werden kann, ob die praktische SQL-Erfahrung umgekehrt beim Verständnis der theoretischen Anteile hilfreich ist.

Die fünfte und letzte Frage besteht aus mehreren Aussagen, die die Studierenden auf einer fünfstufigen Skala danach bewerten sollen, wie sehr sie der Aussage zustimmen. Die Aussagen „Die Erklärungen waren gut verständlich“, „Das Bildmaterial war anschaulich und hat das Tutorial gut unterstützt“ und „Ich fühle mich auf die folgenden Veranstaltungen gut vorbereitet“ wurden jeweils mit einer durchschnittlichen Zusprechung von 4 bewertet (ein Wert von 5 entspricht der höchsten möglichen Zustimmung). Die übrigen zwei Aussagen hatten ähnliche, aber etwas höhere Bewertungen. Die Aussage „Die Mitarbeiter-DB war als Beispiel gut nachvollziehbar und anschaulich“ wurde mit 4,3 bewertet und „Ich konnte die Übungsaufgaben gut mit verfolgen“ erhielt eine Bewertung von 4,25.

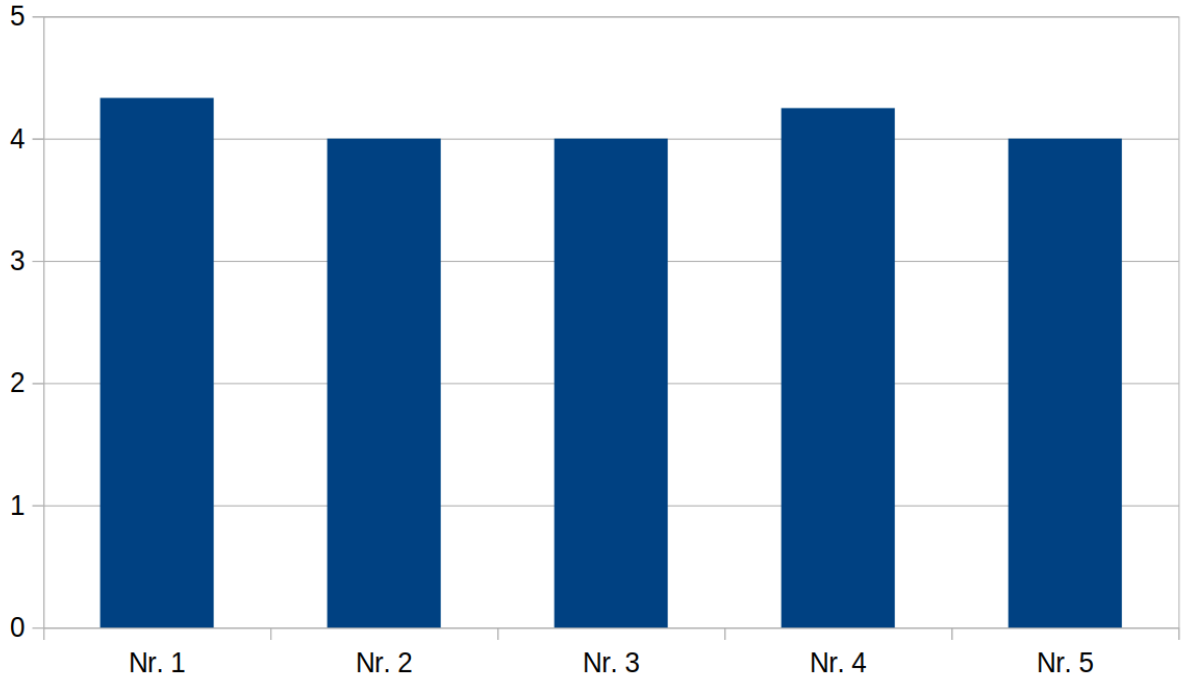


Abbildung 26: Bewertung der Aussagen in Frage 5

Die zu bewertenden Aussagen lauten:
1. Die Mitarbeiter-DB war als Beispiel gut nachvollziehbar und anschaulich.
2. Die Erklärungen waren gut verständlich.
3. Das Bildmaterial war anschaulich und hat das Tutorial gut unterstützt.
4. Ich konnte die Übungsaufgaben gut mit verfolgen.
5. Ich fühle mich auf die folgenden Veranstaltungen gut vorbereitet.

Aus diesen Aussagen lässt sich ableiten, dass das Tutorial von den Studierenden gut angenommen wurde, und dass die praktischen Übungen, die sich direkt auf die Datenbank beziehen, besonders beliebt waren. Da es ebenfalls keine Beschwerden über den Schwierigkeitsgrad der Veranstaltung gab, kann für die zukünftige Verwendung des Tutorials davon ausgegangen werden, dass es zumindest mit den praktischen Übungen keine Schwierigkeiten gibt und die Aufgaben weiter verwendet werden können.

V Fazit und zukünftige Möglichkeiten

11 Zukünftige Beobachtungsstellen

Leider fehlen Langzeitbeobachtungen für diese Auswertung. Da für diese Arbeit nicht überprüft werden konnte, ob sich der Notenspiegel in diesem Semester im Vergleich zu vorhergehenden Inkarnationen der Lehrveranstaltung verändert hat, kann das Tutorial nur durch die Einschätzung der Studierenden bewertet werden, und auch für diese besteht kein Vergleichswert, da nie ein ähnlicher Fragebogen über die vorige Version ausgefüllt wurde.

Besonders ein Vergleich des Notenspiegels mit vorherigen Jahrgängen wäre für eine ausführlichere Bewertung natürlich relevant, aber auch eine Beobachtung der Leistungen in den kommenden Übungen könnte aufschlussreich sein. Falls der Notenspiegel sich nicht wesentlich ändert, die Studierenden aber die Übungsaufgaben in kürzerer Zeit korrekt lösen konnten, könnte dies trotzdem als messbare Verbesserung des Tutorials betrachtet werden.

Sollte es eine Änderung in der Leistung des Jahrgang im Vergleich zu ihren Vorgängern geben, wäre es von Interesse, festzustellen, ob sie sich in bestimmten Bereichen oder Aufgabentypen häufen.

Außerdem müssen auch verschiedene Variablen berücksichtigt werden, die den Verlauf des Praxiseinsatzes beeinflusst haben können. Wichtig ist hier vor allem die unübliche Anordnung des Tutorials zu Beginn der Veranstaltung, eine Entscheidung, deren Vor- und Nachteile nicht getestet wurden, aber auch die Tatsache, dass das Tutorial von einer ungeübten Lehrperson angeleitet wurde und das es zwischen den beiden Lehreinheiten aufgrund der Osterfeiertage eine ungewöhnlich lange Unterbrechung gab, können die Ergebnisse beeinflussen.

12 Weitere Verbesserungsansätze

Wie bereits angedeutet, gab es eine starke Diskrepanz in der Teilnahme der einzelnen Studierenden während der Übungsaufgaben. Es wäre hier möglich, die Aufgaben interaktiver zu gestalten, um jeden Kursteilnehmer anzuregen, eine eigene Lösung für die Aufgaben zu formulieren, statt nur der Diskussion zu zuhören. Um sicherzustel-

len, dass tatsächlich jeder Teilnehmer zumindest einen Lösungsversuch durchführt, statt der Diskussion nur passiv zu folgen, müssten Arbeitsplätze mit Computern und der installierten Datenbank bereitgestellt und eventuell mehr Zeit eingeräumt werden. Es ist aber fraglich, ob dies nötig ist, da die Vorlesung mit einer separaten Übungsveranstaltung gepaart ist und laut der Umfrage die Studierenden der Übungen größtenteils folgen konnten.

Eine andere Möglichkeit, das Engagement der Studierenden anzuregen, wäre die Bereitstellung eines Selbsttests, damit Teilnehmer eigenständig feststellen können, in welchen Bereichen sie Schwierigkeiten haben. Dies könnte vor allen Dingen in der Prüfungsvorbereitung von großem Nutzen sein.

Außerdem sollte, auch wenn die einzelnen Teile des Tutorials positiv bewertet wurden, beachtet werden, dass das Bildmaterial und die verbalen Erklärungen schlechter abgeschnitten haben als die praktischen Beispiele. Sollten also Veränderungen vorgenommen werden, sind die Anteile, die in einem traditionellen Vorlesungs-Format gelehrt werden, von besonderem Interesse. Um festzustellen, welche Erklärungen besonderen Verbesserungsbedarf haben, wäre es möglich, zu beobachten, welche Aufgaben den Studierenden in den Übungsveranstaltungen besondere Schwierigkeiten bereiten, oder in einem offenen Dialog zu erfragen, welche Konzepte ihnen schwer fielen.

Hierbei sollte allerdings auch darauf Rücksicht genommen werden, dass in diesem Semester die theoretischen Grundlagen nicht im selben Umfang behandelt wurden, wie in anderen Jahrgängen, es ist also auch möglich, dass die Verständnisschwierigkeiten darauf beruhen.

13 Zusammenfassung

Im Zuge dieser Arbeit wurde das SQL Tutorial der Hochschule Merseburg analysiert und verbessert. Um den Hintergrund und Inhalt des Tutorials zu verdeutlichen, wurde zuerst die Geschichte und der Aufbau der Structured Query Language selbst in Kapitel 2 behandelt und mit einer Bestandsaufnahme der SQL Ausbildung allgemein ergänzt, bevor in Kapitel 3 dieses spezifische Tutorial behandelt wird. Dafür wird zuerst die bisher verwendete Version des Materials ausgiebig analysiert. Da die Struktur des Tutorials solide aufgebaut war und es kaum inhaltlichen Mängel gab, wurde kein von Grund auf neues Tutorial erstellt, sondern das vorhandene Material als Grundlage verwendet und ausgebessert.

Der umfangreichste Kritikpunkt an dem vorhanden Tutorial bestand in der Wahl zwei verschiedener Beispieldatenbanken, was dadurch behoben wurde, die erste Datenbank so anzupassen, dass sämtliche Aufgaben und Beispiele an ihr durchgeführt werden können. Abgesehen davon gab es vor allem formale Änderungen an der Formatierung und Darstellung von einzelnen Folien und das verwendete Bildmaterial wurde teilweise neu erstellt und angepasst.

Die erneuerte Version des Tutorials wurden von den Teilnehmern des Kurses Datenbanken im Sommersemester 2025 durchaus positiv angenommen, besonders die praktischen Inhalte und Übungsaufgaben wurden positiv bewertet. Allerdings gibt es einige noch offene Fragen, die nur durch zukünftige Beobachtungen beantwortet werden können. Besonders bleibt es abzuwarten, ob der Notenspiegel dieses Kurses sich im Vergleich zu vorherigen Jahrgängen verändert hat oder ob sich die Leistungen der Studierenden in den Übungsveranstaltungen maßgeblich verändert haben.

Literaturverzeichnis

[1] „Oracle SQL Developer Logo“. Oracle Corporation. [Online]. Available: https://www.oracle.com/technetwork/developer-tools/sql-developer/sql-dev-101614-2340766.jpg

[2] „IBM Data Processing Techniques: Flowcharting Techniques GC20-8152-1“. International Business Machines Corporation - Data Processing Division, 1970. [Online]. Available: https://web.archive.org/web/20120402204237/http://www.fh-jena.de/%7Ekleine/history/software/IBM-FlowchartingTechniques-GC20-8152-1.pdf

[3] A. Reinhold, „Dysan CE Disk Pack“. Wikimedia Commons, Veröffentlicht unter der Creative Commons Attribution-Share Alike 2.5 Generic License, https://commons.wikimedia.org/wiki/File:DysanRemovableDiskPack.agr.jpg. [Online]. Available: https://commons.wikimedia.org/wiki/File:DysanRemovableDiskPack.agr.jpg

[4] E. F. Codd, „A relational model of data for large shared data banks“, *Commun. ACM*, Bd. 13, Nr. 6, S. 377–387, Juni 1970, doi: 10.1145/362384.362685.

[5] D. D. Chamberlin, „Early history of SQL“, *IEEE Annals of the History of Computing*, Bd. 34, Nr. 4, S. 78–82, 2012, doi: 10.1109/MAHC.2012.61.

[6] M. Meier Andreasand Kaufmann, „NoSQL databases“, in *SQL & NoSQL databases: Models, languages, consistency options and architectures for big data management*, Wiesbaden: Springer Fachmedien Wiesbaden, 2019, S. 201–218. doi: 10.1007/978-3-658-24549-8_7.

[7] F. Chang *u. a.*, „Bigtable: A distributed storage system for structured data“, *ACM Trans. Comput. Syst.*, Bd. 26, Nr. 2, Juni 2008, doi: 10.1145/1365815.1365816.

[8] C. Crosbie und G. Elliot, „Bigtable transforms the developer experience with SQL support“. März 2024. [Online]. Available: https://cloud.google.com/blog/products/databases/announcing-sql-support-for-bigtable?hl=en

[9] „Technology 2024 Stack Overflow Developer Survey“. Stack Exchange Inc. [Online]. Available: https://survey.stackoverflow.co/2024/technology

[10] „DB-Engines Ranking - popularity ranking of database management systems“. Red Gate Software Ltd, 2025. [Online]. Available: https://db-engines.com/en/ranking

[11] E. F. Codd, „Further normalization of the data base relational model.“, IBM, RJ909, Aug. 1971.

[12] G. für Informatik (GI), „Objektrelationale Datenbanksysteme“. Juli 2005. [Online]. Available: https://gi.de/informatiklexikon/objektrelationale-datenbanksysteme

[13] T. Taipalus und V. Seppänen, „SQL Education: A Systematic Mapping Study and Future Research Agenda“, *ACM Transactions on Computing Education*, Bd. 20, Nr. 3, S. 1–33, Sep. 2020, doi: 10.1145/3398377.

[14] M. Laube, *Einstieg in SQL*, 1. Auflage, 1., Korrigierter Nachdruck. in Rheinwerk Computing. Bonn: Rheinwerk Verlag GmbH, 2018.

[15] E. Schicker, *Datenbanken und SQL: Eine praxisorientierte Einführung mit Anwendungen in Oracle, SQL Server und MySQL*, 4., Überarb. Aufl. 2014. in Informatik & Praxis. Wiesbaden: Springer Vieweg, 2014.

[16] F. Roeing, *ORACLE7-Datenbanken erfolgreich realisieren: Entwurf, Entwicklung, Tuning*. in Vieweg Datenbanksysteme. Braunschweig: Vieweg, 1996.

[17] „SQLBolt - Learn SQL - Introduction to SQL“. [Online]. Available: https://sqlbolt.com/

[18] R. Kearns, S. Shead, und A. Fekete, „A teaching system for SQL“, in *Proceedings of the second Australasian conference on Computer science education - ACSE '97*, The Univ. of Melbourne, Australia: ACM Press, 1996, S. 224–231. doi: 10.1145/299359.299391.

[19] W. C. Ogden und S. R. Brooks, „Query languages for the casual user: Exploring the middle ground between formal and natural languages“, in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems - CHI '83*, Boston, Massachusetts, United States: ACM Press, 1983, S. 161–165. doi: 10.1145/800045.801602.

[20] C. Welty und D. W. Stemple, „Human factors comparison of a procedural and a nonprocedural query language“, *ACM Transactions on Database Systems*, Bd. 6, Nr. 4, S. 626–649, Dez. 1981, doi: 10.1145/319628.319656.

[21] S. Sadiq, M. Orlowska, W. Sadiq, und J. Lin, „SQLator: An online SQL learning workbench“, in *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, Leeds United Kingdom: ACM, Juni 2004, S. 223–227. doi: 10.1145/1007996.1008055.

[22] T. Taipalus, H. Grahn, und H. Ghanbari, „Error messages in relational database management systems: A comparison of effectiveness, usefulness, and user confidence“, *Journal of Systems and Software*, Bd. 181, S. 111034, Nov. 2021, doi: 10.1016/j.jss.2021.111034.

[23] S. Brass und C. Goldberg, „Semantic errors in SQL queries: A quite complete list“, *Journal of Systems and Software*, Bd. 79, Nr. 5, S. 630–644, 2006, doi: https://doi.org/10.1016/j.jss.2005.06.028.

[24] M. B. Roeser *u. a.*, „NLS_DATE_FORMAT“, *Oracle Help Center*. [Online]. Available: https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/NLS_DATE_FORMAT.html

[25] „MySQL :: MySQL 8.4 Reference Manual :: 13.2.2 The DATE, DATETIME and TIMESTAMP Types“. Oracle. [Online]. Available: https://dev.mysql.com/doc/refman/8.4/en/datetime.html

[26] MikeRayMSFT, „Date (Transact-SQL) - SQL Server“. Zugegriffen: Juni 18, 2025. [Online]. Available: https://learn.microsoft.com/en-us/sql/t-sql/data-types/date-transact-sql?view=sql-server-ver17

A Anhang 1

Ergebnisse der studentischen Umfrage

Frage 1: Hattest du vorherige Erfahrungen mit SQL-basierten Datenbanken?

Antwort	Häufigkeit
Ja	2
Ein wenig	8
Nein	2

Frage 2: Auf einer Skala von 1 (schlecht) bis 10 (sehr gut) - Wie würdest du deine Erfahrung mit dem SQL-Tutorial insgesamt einschätzen?

Antwort	Häufigkeit
1	0
2	0
3	0
4	0
5	0
6	1
7	2
8	4
9	4
10	1

Frage 3: Wie würdest du den Schwierigkeitsgrad des Tutorials einschätzen?

Antwort	Häufigkeit
Zu leicht	0
Etwas zu leicht	0

Antwort	Häufigkeit
Angemessen	12
Etwas zu schwer	0
Zu schwer	0

Frage 4: Hattest du das Gefühl, dass dein Verständnis durch den Mangel an theoretischen Grundlagen beeinträchtigt wurde?

Antwort	Häufigkeit
Ja	1
Ein wenig	5
Nein	6

Frage 5: Bitte bewerte die folgenden Aussagen nach deiner persönlichen Meinung:

Aussage 1: Die Mitarbeiter-DB war als Beispiel gut nachvollziehbar und anschaulich.

Zuistimmung	Häufigkeit
Stimme voll und ganz zu	5
Stimme zu	6
Teils/Teils	3
Stimme nicht zu	0
Stimme überhaupt nicht zu	0

Aussage 2: Die Erklärungen waren gut verständlich.

Zuistimmung	Häufigkeit
Stimme voll und ganz zu	2
Stimme zu	8
Teils/Teils	2
Stimme nicht zu	0
Stimme überhaupt nicht zu	0

Aussage 3: Das Bildmaterial war anschaulich und hat das Tutorial gut unterstützt.

Zuistimmung	Häufigkeit
Stimme voll und ganz zu	2
Stimme zu	8
Teils/Teils	2
Stimme nicht zu	0
Stimme überhaupt nicht zu	0

Aussage 4: Ich konnte die Übungsaufgaben gut mit verfolgen.

Zuistimmung	Häufigkeit
Stimme voll und ganz zu	4
Stimme zu	7
Teils/Teils	1
Stimme nicht zu	0
Stimme überhaupt nicht zu	0

Aussage 5: Ich fühle mich auf die folgenden Veranstaltungen gut vorbereitet.

Zuistimmung	Häufigkeit
Stimme voll und ganz zu	1
Stimme zu	10
Teils/Teils	1
Stimme nicht zu	0
Stimme überhaupt nicht zu	0