DB ABFRAGEN

Kompetenznachweis

Adriana Arteaga, Nuria Anaya
Andreas Nydeggger, Kevin Schleuniger

Inhalt

SQL

SQL (Structured Query Language) ist eine Datenbanksprache zur Definition von Datenstrukturen in relationalen Datenbanken sowie zum Bearbeiten und Abfragen von Darauf basierenden Datenbestände. Diese Beispiele von Abfragen werden alle mit SQL gezeigt, da das die häufigste verwendete Sprache ist.

Einfache Abfragen

Einfache Abfragen werden in der Form:

Befehl Spaltennamen FROM Tabelle;

Bsp. SELECT Name, Vorname FROM User;

dargestellt. Hier werden alle Werte in den Spalten Name und Vornamen von der Tabelle User ausgewählt. Das Resultat könnte so aussehen:

User	
Vorname	Name
Maria	Corneli
Hans	Peter
Nuria	Anaya
Adriana	Arteaga
Kevin	Schleuniger
Andreas	Nydegger

Befehle

Die einfachsten und am meisten verwendeten Befehle sind:

Anweisung	**Beispiel**	**Anwendung**
Select	SELECT * FROM User;	Select wählt die Tabelle und die Spalten die man ihr angibt an
Delete	DELETE * FROM User WHERE id=1;	Delete löscht in der entsprechender Tabelle ein Wert oder eine Zeile
Update	UPDATE name FROM User WHERE id = 2;	Update bearbeitet eine Zeile oder ein Wert einer angegebener Tabelle
Insert	INSERT INTO User (Name) VALUES („Nuria“) ;	Insert fügt in der Datenbank einen neuen Wert in die Tabelle ein.

Das Wort FORM wird immer gebraucht um die Tabelle auszuwählen.

Spalten

Will man alle Spalten in einer Tabelle Auswählen so benutz man das „*“. Das Sternchen symbolisiert alle Spaltennamen. Will man nur einige spezielle Spalte auswählen so schreibt man den exakten Spaltennamen hin. Und wenn man mehrere Spalten will trennt man Sie mit einem Komma (,) ausser beim letzten Spaltennamen kommt kein Komma mehr.

Definieren wir nun eine Tabelle User mit den Spalten Id, Name, Vorname und Geburtsdatum.

Abfrage	**Beispiel**
SELECT * FROM User	Hier würde jede Spalte ausgewählt werden: User Id | Vorname | Name | Geburtstag 1 | Nuria | Anaya | 14.06.200 2 | Adriana | Arteaga | 03.01.2002

	3 | Kevin | Schleuniger | 15.09.2001 4 | Andreas | Nydegger | 29.11.1996
SELECT Geburtsdatum FROM User	Hier würde man nur alle Geburtstage erhalten: User: Geburtstag 14.06.200 03.01.2002 15.09.2001 29.11.1996
SELECT Name, Vorname FROM User	Hier erhaltet man den Namen und Vorname aller Personen User: Vorname | Name Nuria | Anaya Adriana | Arteaga Kevin | Schleuniger Andreas | Nydegger

Bedingungen

Zu den einfachen Abfragen gehören auch die Bedingungen. Beim ersten Beispiel mit dem Befehl Update steht nach dem Tabellenname noch die Bedingung WHERE, was so viel wie WO bedeutet. Es gibt viele verschiedene Bedingungen, WHERE wird jedoch am häufigsten gebraucht. Hier einige Bedingungen:

Bedingung	Abfrage	Erklärung, Beispiel
WHERE	DELETE * FROM User **WHERE** id=1	Bei der WHERE Bedingung sucht man einen Datensatz der wie z.B. im die Id 1 besitzt. Man könnte auch nach dem Namen suchen mit WHERE Name = `Maria‘.
LIKE	SELECT * FROM User WHERE Name **LIKE** '%a%'	Beim der LIKE Bedingung wird etwas gesucht das ein bestimmtes Muster hat. Z.B der Name muss irgendwo ein a drin haben.
BETWEEN	SELECT * FROM Products WHERE Price **BETWEEN** 10 AND 20;	Bei der BETWEEN Bedingung wird nach einem Datensatz gesucht das z.B. der Preis zwischen 10 und 20 Franken liegt.

Wildcards

Bei der Bedingung LIKE werden „Wildcards“ verwendet. Wildcards sind Platzhalter. Hier die wichtigsten Platzhalter:

Symbol	Beschreibung	Beispiel
	MS Access	
*	Repräsentiert 0 oder mehrere Zeichen	bl* → black, blue, blob
?	Repräsentiert nur ein Zeichen	h?t → hot, hat, hit
!	Repräsentiert alle Zeichen ausser die zwischen [ ]	h[!oa]t → hit
#	Repräsentiert eine Zahl	2#5 → 205, 215, 225, 235, ...
	SQL Server	
%	Repräsentiert eine oder mehrere Zeichen	bl% → black, blue, blob

_	Repräsentiert ein Zeichen	h_t → hot, hat, hit
^	Repräsentiert alle Zeichen ausser die zwischen [ ]	h[^oa]t → hit
Für beide gleich		
-	Repräsentiert eine Reichweite der Zeichen	c[a-d]t → cat, cbt, cct, cdt
[]	Repräsentiert alle Zeichen zwischen der [ ]	h[oa]t → hot, hat

Die Wildcards kann man auch beliebig zusammen knüpfen wie z.B.

_r% → findet alle Wörter die ein R an der zweiten Stelle hat	a%o → findet alle Wörter die mit einem a anfangen und mit einem o aufhören

Order By

Order By wird gebraucht um die Resultaten Liste absteigen oder aufsteigen zu ordnen.

Bsp.
```
SELECT * FROM Kunden
ORDER BY Land DESC;
```

Hier werden die Kunden nach dem Land absteigen nach dem Land sortiert. DESC steht für absteigen und ASC für aufsteigen.

Das Resultat für DESC wäre also:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
33	GROSELLA-Restaurante	Manuel Pereira	5ª Ave. Los Palos Grandes	Caracas	1081	Venezuela
35	HILARIÓN-Abastos	Carlos Hernández	Carrera 22 con Ave. Carlos Soublette #8-35	San Cristóbal	5022	Venezuela
46	LILA-Supermercado	Carlos González	Carrera 52 con Ave. Bolívar #65-98 Llano Largo	Barquisimeto	3508	Venezuela
47	LINO-Delicateses	Felipe Izquierdo	Ave. 5 de Mayo Porlamar	I. de Margarita	4980	Venezuela
32	Great Lakes Food Market	Howard Snyder	2732 Baker Blvd.	Eugene	97403	USA

Und für ASC:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
12	Cactus Comidas para llevar	Patricio Simpson	Cerrito 333	Buenos Aires	1010	Argentina
54	Océano Atlántico Ltda.	Yvonne Moncada	Ing. Gustavo Moncada 8585 Piso 20-A	Buenos Aires	1010	Argentina
64	Rancho grande	Sergio Gutiérrez	Av. del Libertador 900	Buenos Aires	1010	Argentina
20	Ernst Handel	Roland Mendel	Kirchgasse 6	Graz	8010	Austria
59	Piccolo und mehr	Georg Pipps	Geislweg 14	Salzburg	5020	Austria

Group By

Das Group By wird verwendet um Datensätze zu gruppieren. Dies braucht man z.B. wen man die Anzahl von Kunden von jedem Land haben will. Dies würde dann so aussehen:

Bsp.
```
SELECT COUNT(KundenID, Land) FROM Kunden
GROUP BY Land;
```

Das Resultat wäre dann etwa das:

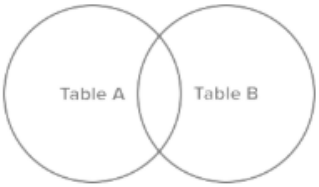
COUNT(CustomerID)	Country
3	Argentina
2	Austria
2	Belgium
9	Brazil
3	Canada
2	Denmark
2	Finland

Weitere Tricks für Abfragen

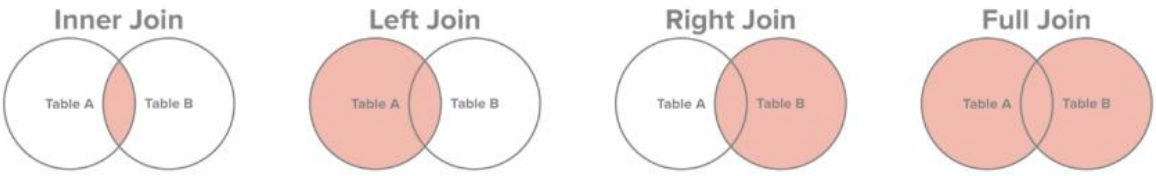
Für die SQL-Abfragen gibt es noch endliche mehr Tricks wie COUNT, AVG, MAX, MIN, AND, OR, NOT, etc. die werden wir aber nicht alle weiter erläutern. Hier gibt's ein link zu einer Webseite in der viele Informationen stehen und sie auch gut erklären.
Hier geht's zum weiter lernen: https://www.w3schools.com/sql/default.asp

Joins

Es gibt 4 Basic SQL Joins, Inner, Left, Right und Full Join. Join bedeutet eigentlich anschliessen, zusammenschliessen. Bei SQL Joins schliessen wir zwei Tabellen zusammen. Tabelle A und Tabelle B.

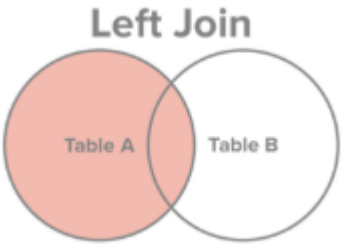


Das übereinanderliegende Feld sind die Datensätze die beide Tabellen gemeinsam haben. In der Datenbank also z.B. der Schlüssel mit dem Fremdschlüssel. Die Schlüssel verknüpfen zwei Datensätze miteinander von zwei verschiedenen Tabellen. Nun gibt es aber verschieden Typen um diese Verbindungen oder auch nicht Verbindungen aufzeigen zu lassen.
Das sind die vier Basic Joins.



Left Join

Der Left Join (Links Join) nimmt alle Datensätze der Tabelle A und nur die Datensätze aus der Tabelle B, die eine Verbindung zu einem Datensatz in der Tabelle A haben. Also können einige Datensätze der Tabelle A auch ohne einen dazugehörigen Datensatz der Tabelle B stehen, dort wird einfach der Wert NULL eingefügt.



Beispiel

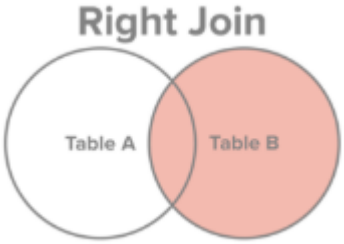
Nehmen wir an wir wollen Informationen zu Bestellungen an unserer Kundentabelle haben, unabhängig davon ob ein Kunde eine Bestellung aufgegeben hat oder nicht. Dann verwenden wir einen Left Join, die geben alle Datensätze aus Tabelle A (Kunden) und alle übereinstimmenden Datensätze aus Tabelle B (Bestellungen) zurück.

```
SELECT Vorname, Name, Datum FROM Kunde k
LEFT JOIN Bestellungen b
ON k.id = b.id
```

Merke, dass beim Resultat keine passende Datensätze für James Madison und Claire Muff bei der Tabellen Liste gab. Sie werden also einfach mit dem Wert NULL befüllt.

Vorname	Nachname	Datum	Bestellungspreis
George	Washinton	07.08.2018	CHF 243.56
John	Adams	07.11.2018	CHF 456.30
Thomas	Jefferson	03.10.2019	CHF 50.56
Thomas	Jefferson	07.10.2019	CHF 173.99
James	Madison	NULL	NULL
Claire	Muff	NULL	NULL

Right Join



Der Right Join (Rechts Join) ist eine Spiegelung des Left Joins. Es nimmt alle Datensätze der Tabelle B und nur die Datensätze aus der Tabelle A, die eine Verbindung zu einem Datensatz in der Tabelle B haben. Also können einige Datensätze der Tabelle B auch ohne einen dazugehörigen Datensatz der Tabelle A stehen, dort wird einfach der Wert NULL eingefügt.

Beispiel

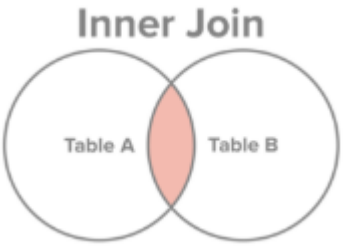
Der Right Join ist eine gespiegelte Version von Left Join. Es wird eine Liste aller Bestellungen abgerufen, an deren Kundeninformation angehängt sind. Sie geben alle Datensätze aus Tabelle B (Bestellung) und alle übereinstimmenden Datensätze aus Tabelle A (Kunden) zurück.

```
SELECT Vorname, Name, Datum FROM Kunde k
RIGTH JOIN Bestellungen b
ON k.id = b.id
```

Merke, dass beim Resultat keine passende Datensätze für die letzten zwei Bestellungen gab. Sie werden also einfach mit dem Wert NULL befüllt.

Vorname	Nachname	Datum	Bestellungspreis
George	Washinton	07.08.2018	CHF 243.56
John	Adams	07.11.2018	CHF 456.30
Thomas	Jefferson	03.10.2019	CHF 50.56
Thomas	Jefferson	07.10.2019	CHF 173.99
NULL	NULL	04.06.2019	CHF 40.30
NULL	NULL	28.09.2019	CHF 425.30

Inner Join



Der Inner Join nimmt nur alle Datensätze der Tabelle A und B, bei der die Join-Bedingung auch wirklich erfüllt ist.

Beispiel

Nehmen wir an wir wollen alle Kunden auflisten, die mal eine Bestellung aufgegeben haben. Dann würde den Query in etwa so aussehen:

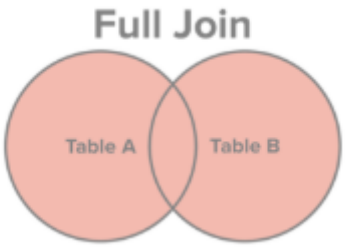
```
SELECT Vorname, Name, Datum FROM Kunde k
INNER JOIN Bestellungen b
ON k.id = b.id
```

Als Resultat würden alle aufgelistet werden die mal was bestellt haben. Wenn Sie mehrmals was bestellt haben, werden sie auch zweimal aufgelistet.

Vorname	Nachname	Datum	Bestellungspreis
George	Washinton	07.08.2018	CHF 243.56
John	Adams	07.11.2018	CHF 456.30
Thomas	Jefferson	03.10.2019	CHF 50.56
Thomas	Jefferson	07.10.2019	CHF 173.99

Full Join

Der Full Join(voller Join) nimmt alle Datensätze der Tabelle A und B, auch die Datensätze bei der die Join-Bedingung nicht erfüllt ist.



Beispiel

Der FULL JOIN gibt einfach alle Datensätze beider Tabellen aus, egal ob sie mit dem andern übereinstimmen.

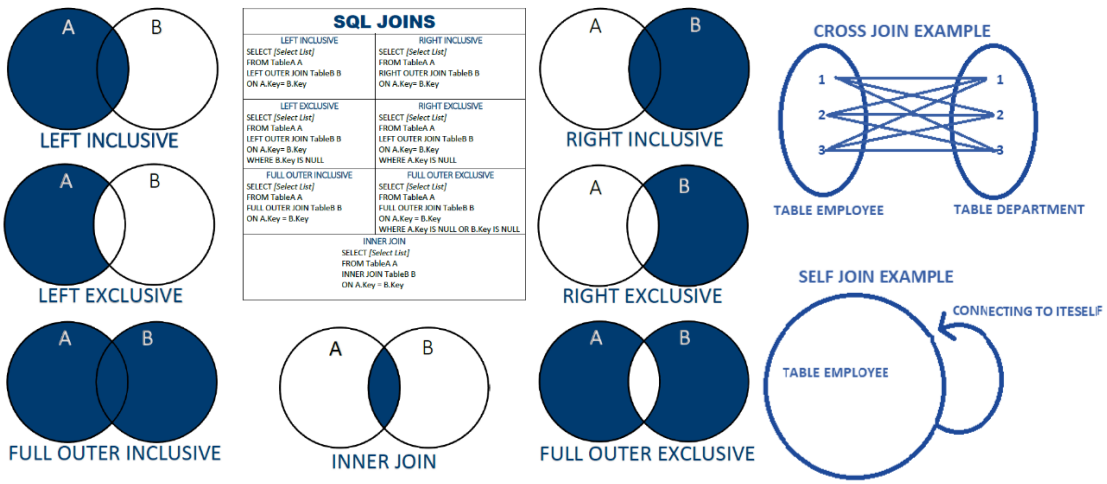
```
SELECT Vorname, Name, Datum FROM Kunde k
FULL JOIN Bestellungen b
ON k.id = b.id
```

Alle Resultate werden angezeigt, egal ob sie einen passenden Datensatz besitzen oder nicht.

Vorname	Nachname	Datum	Bestellungspreis
George	Washinton	07.08.2018	CHF 243.56
John	Adams	07.11.2018	CHF 456.30
Thomas	Jefferson	03.10.2019	CHF 50.56
Thomas	Jefferson	07.10.2019	
James	Madison	NULL	NULL
Claire	Muff	NULL	NULL
NULL	NULL	04.06.2019	CHF 40.30
NULL	NULL	28.09.2019	CHF 425.30

Weiter Joins

Es gibt noch weiter Joins, die wir nicht genauer anschauen werden aber hier lassen wir noch eine Grafik, in der die weiteren Joins noch gut beschrieben werden. Die Joins die wir noch nicht angeschaut haben sind Left Exclusive, Right Exculsive, Cross Join, Self Join und Full outer exclusive.



Transaktionssichere Abfrage

Als Transaktion wird eine Folge von Programmschritten bezeichnet, die als eine logische Einheit betrachtet werden, weil sie den Datenbestand nach fehlerfreier und vollständiger Ausführung in einem konsistenten Zustand hinterlassen.
Transaktionen kommen meist bei Datenbanksystemen zum Einsatz und sie werden von Transaktionssystemen verarbeitet; diese erzeugen dabei aus mehreren Transaktionen eine Historie.

Ziel eines Transaktionssystems ist es stets, möglichst viele Transaktionen in möglichst kurzer Zeit abzuwickeln. Die serielle Ausführung von Transaktionen ist zwar einfach zu realisieren, führt aber oft nicht zu einer optimalen Erfüllung dieses Leistungskriteriums. Transaktionssysteme spalten daher Transaktionen in ihre Operationen auf und setzen diese zu Historien zusammen.
In einem solchen System gibt es immer nur zwei Möglichkeiten eine Transaktion zu beenden, nämlich "commit", was bedeutet dass alles nach Plan verändert und gespeichert wurde,
oder "abort", was bedeutet dass alle Veränderungen mit einem Rollback zurückgesetzt werden.

ACID

Eine Transaktion stellt auch das sogenannte ACID sicher. Atomicity, Consistency, Isolation, Durability.

Atomicity (Atomarität)

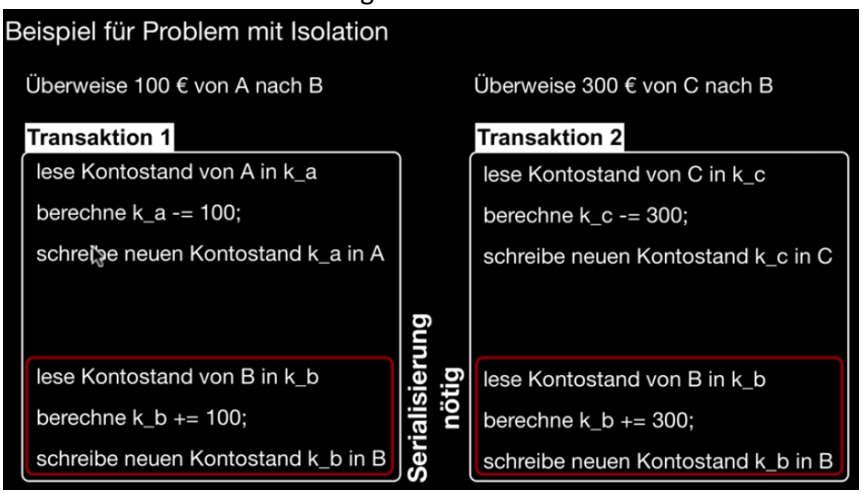
Atomarität stellt sicher das alle Aktionen der Transaktion durchgeführt werden oder keine. (alles oder nichts)

Consistency (Konsistenz)

Die Konsistenz ist dafür zuständig, dass die Transaktion einem konsistenten Zustand in der Datenbank hinterlässt. Z.B. darf kein Konto überzogen werden, wenn das er fall wäre müsste ein Rollback ausgelöst werden. Rollback, gibt alle Daten in den Zustand vor Beginn der Transaktion zurück.

Isolation

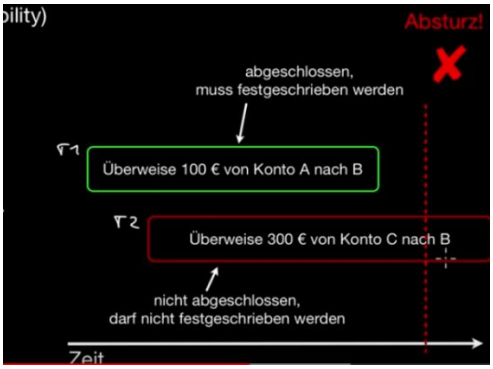
Die Isolation stellt sicher das Nebenläufige Transaktionen sich nicht gegenseitig beeinflussen. Transaktionen werden nacheinander ausgeführt.



Durability (Dauerhaftigkeit)

Die Dauerhaftigkeit sorgt, dass die abgeschlossenen Transaktionen auch wirklich festgeschrieben und gesichert sind. Nach einer erfolgreichen Transaktion werden die Daten

so gespeichert, dass diese selbst im Falle eines Fehlers oder Systemneustarts, im korrekten richtigen Zustand zugänglich sind.

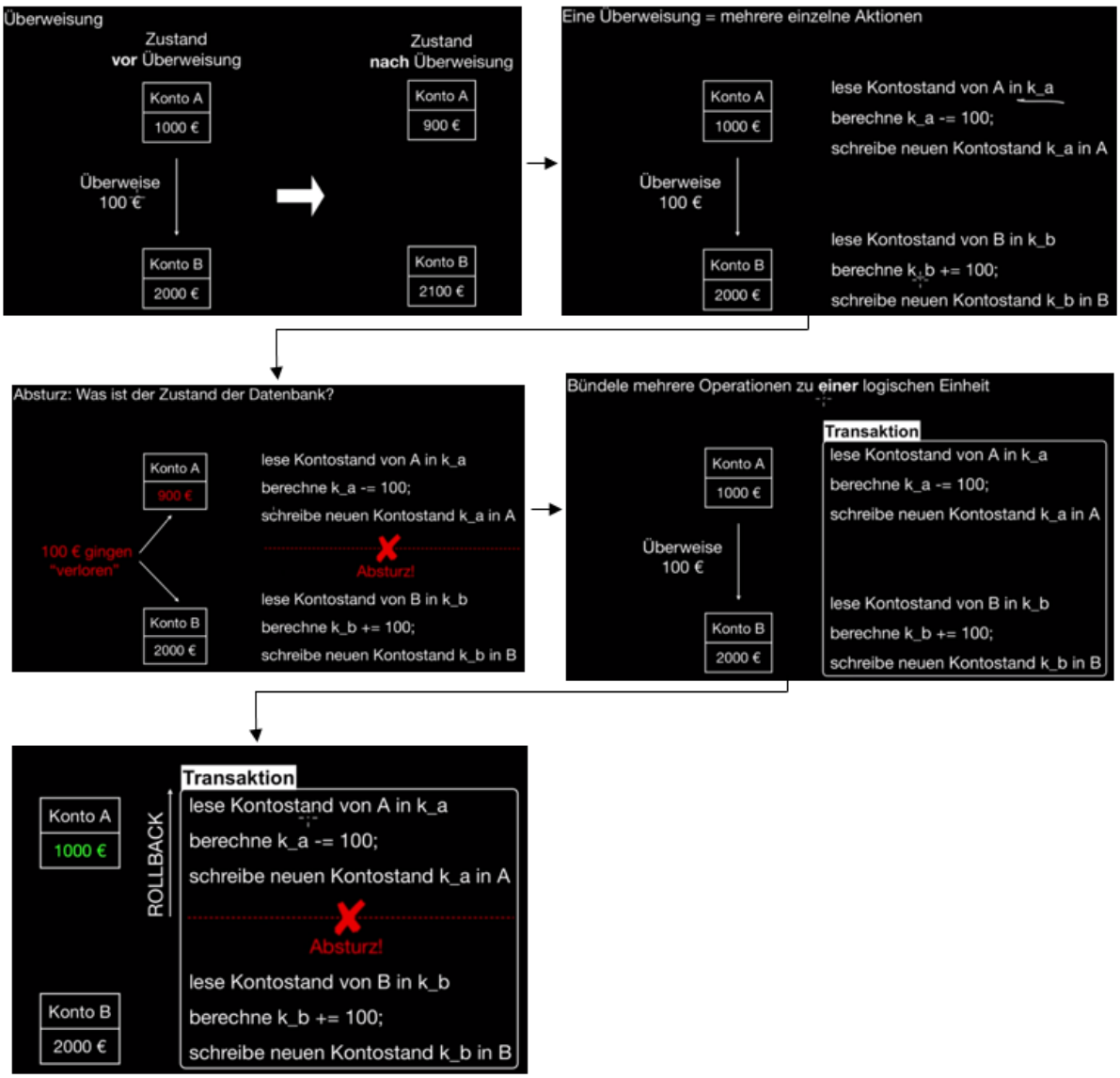


Rollback

Ein Rollback ist ein Befehl, die die Datenbank in einen Früheren Zustand zurückversetzt.
Rollback ist das Zurücksetzen einer Transaktion.
Ein Rollback wird im Fehlerfall gebraucht, falls z.B. ein Fehler in der Transaktion auftritt oder es zu einem Server Absturz kommt während eine Transaktion läuft und diese deshalb nicht vollständig ist.
In SQL ist ein ROLLBACK ein Befehl, der bewirkt, dass alle Datenänderungen seit dem letzten BEGIN WORK oder START TRANSACTION von den relationalen Datenbankverwaltungssystem verworfen werden, sodass der Status der Daten auf den ursprünglichen Status zurückgesetzt wird, bevor diese Änderungen.
Rollbacks sind wichtig für die Datenbankintegrität, da dies bedeutet, dass die Datenbank nach fehlerhaften Vorgängen, eine saubere Kopie wiederhergestellt werden kann. Ebenfalls ist es wichtig für die Wiederherstellung nach Abstürzen des Datenbankservers.

Problem ohne Transaktion

Die Probleme, die ohne Transaktionen auftauchen könnten wären, wenn man z.B. eine Geldüberweisung auf ein Konto machen will. Von Konto A wird 100.- auf ein Konto B überwiesen. Nehmen wir an während der Überweisung passiert ein Systemabsturz und von Konto A wurden aber schon 100.- abgezogen. Konto B hat das Geld aber nicht bekommen, da das System abgestürzt ist. Also gingen 100.- verloren.
Mit einer Transaktion wäre dies nicht möglich. Man würde jeden dieser einzelnen Schritte in eine Transaktion packen und dann ausführen. Würde also ein Systemabsturz während der Transaktion passieren. Würde die ganze Transaktion einen sogenannten Rollback machen und die Transaktion wird wieder auf den Anfangspunkt zurück gerollt, wo Konto A noch die 100.- hätte.



Beispiel

```
BEGIN TRY
	BEGIN TRANSACTION
		DELETE FROM Student WHERE Id=1
		-- some other codes
	COMMIT
END TRY
BEGIN CATCH
	ROLLBACK
END CATCH
```

Quellen

https://www.w3schools.com/sql/sql_wildcards.asp
http://www.sql-join.com/sql-join-types