ÜK 105 - Zusammenfassung

Autor:

Mike Dätwyler

Modul 105:

Datenbanken mit SQL bearbeiten

Stand vom:

23.02.2021 bis 03.03.2021

Inhaltsverzeichnis

Allgemeine Informationen

Rationale Datenbanken

Definition:

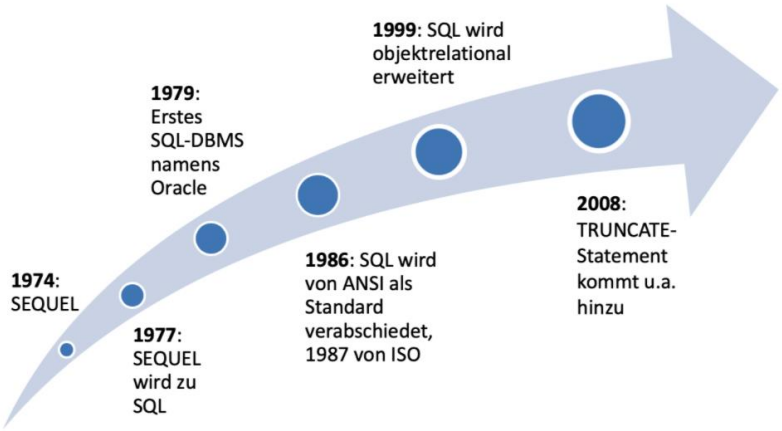
Sammlung von Tabellen (Relationen) und Beziehungen (Verknüpfungen). Beziehungen werden über Schlüsselpaare hergestellt.

Beispiele:

- Oracle Database
- Microsoft SQL Server
- IBM DB2
- MySQL Oracle
- Postgre SQL

Merkmale – SQL

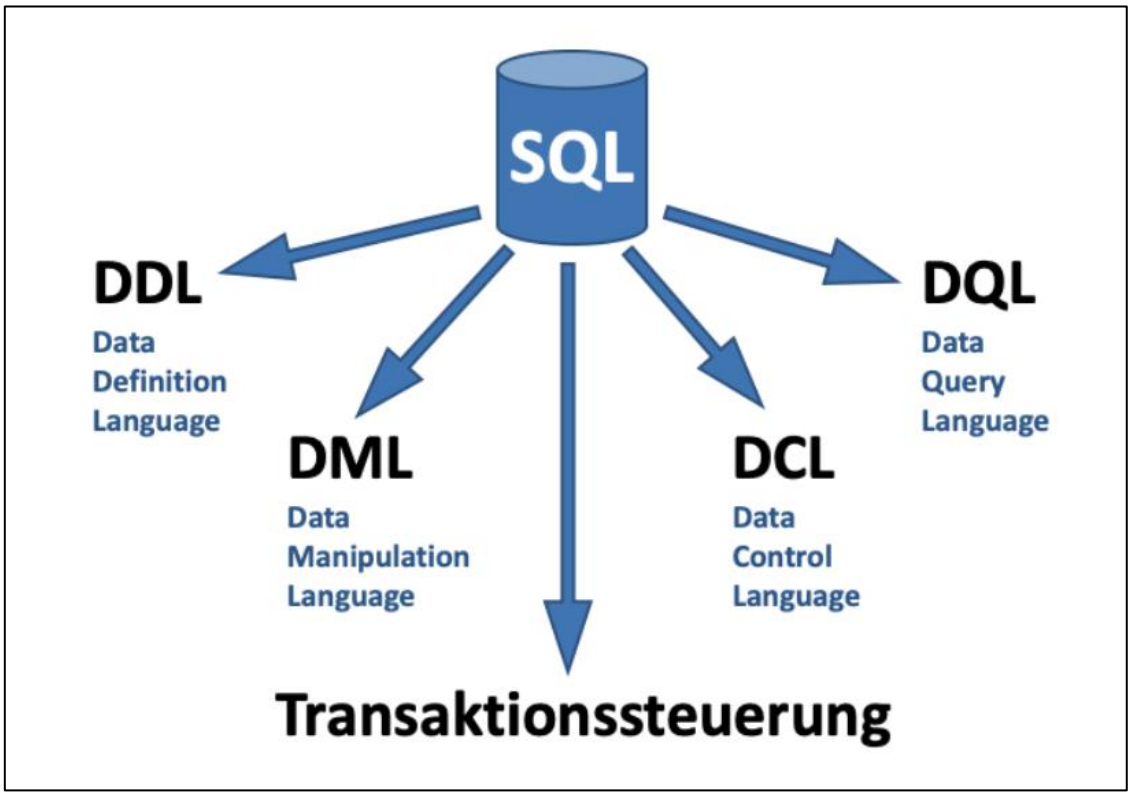
- SQL ist eine Anweisungssprache
- SQL arbeitet tabellen- und mengenorientiert sowie deklarativ
- Merkmale einer Programmiersprache fehlen
- Kein UNDO möglich!
- Nicht case-sensitiv
- Beliebiger Einsatz von Leerzeichen, Zeilenumbrüchen und Tabulatoren



SQL – Datentypen

Datentyp	Wertebereich von	Wertebereich bis	Kategorie
[varchar]		Maximal 8000 Zeichen (variabel)	Character String
[int]	-2,147,483,648	2,147,483,647	Exakt Nummerisch
[datetime]	01.01.1753	31.12.9999	Datum und Zeit
[nchar]		Maximal 4000 Zeichen (fix)	Unicode Character String
[char]		Maximal 8000 Zeichen (fix)	Character String
[float]	-1.79E + 308	1.79E + 308	Approximativ Nummerisch
[decimal]	-10^38 +1	10^38 -1	Exakt Nummerisch
[smalldatetime]	01.01.1900	06.06.2079	Datum und Zeit
[real]	-3.40E + 38	3.40E + 38	Approximativ Nummerisch
[bit]	0	1	Exakt Nummerisch
[binary]		Maximal 8000 Bytes (fix)	Binärer String
[nvarchar]		Maximal 8000 Zeichen (variabel)	Unicode Character String
[varbinary]		Maximal 8000 Bytes (variabel)	Binärer String
[money]	-922,337,203,685,477.5808	+922,337,203,685,477.5807	Exakt Nummerisch (Währung)
[smallint]	-32,768	32,767	Exakt Nummerisch
[date]	01.01.000	31.12.9999	Datum und Zeit
[time]	00:00:00.0000000	23:59:59.9999999	Datum und Zeit

SQL – Sprachbereiche



DDL – Data Definition Language

Datenbank anlegen

Wechselt zu anderer Datenbank	USE master;
Erstellt eine Datenbank	CREATE DATABASE database_name;
GO = Batch-Trennzeichen, ist keine SQL-Anweisung, sondern Editor-Befehl, eine Steueranweisung: Stapel von SQL-Anweisungen werden einzeln zum Server geschickt.	CREATE DATABASE database_name GO USE database_name;
Ändern einer Datenbankdefinition	ALTER DATABASE database_name;

Tabelle anlegen

Tabelle **anlegen**	CREATE TABLE table_name (spalte1 datentyp [NULL | NOT NULL] , spalte2 datentyp [NULL | NOT NULL] , spaltex datentyp [NULL | NOT NULL]);
Beispiel:	CREATE TABLE kurse (kursid int NOT NULL, kursname varchar(50) NOT NULL, beginn smalldatetime, ende smalldatetime);
Spalte **hinzufügen**	ALTER TABLE table_name ADD spalte datentyp [NOT NULL];
Spalte **ändern**	ALTER TABLE table_name ALTER COLUMN spalte datentyp [NOT NULL]
Spalte **löschen**	ALTER TABLE table_name DROP COLUMN spalte;

Datenbank / Tabelle löschen

Datenbank **löschen**	DROP DATABASE database_name;
Tabelle **löschen**	DROP TABLE table_name;
Alle Datensätze einer Tabelle **löschen** (ohne Protokollierung → schneller)	TRUNCATE TABLE table_name;
Löscht alle Datensätze oder selektive (dazu mehr unter DML). Diese Operation wird protokolliert, da es sich um eine DML-Anweisung handelt.	DELETE FROM table_name [WHERE …];

CONSTRAINTS

CONSTRAINT – Typen

PRIMARY KEY	• Erzwingt Einmaligkeit eines Datensatzes • eindeutig • NOT NULL • automatische Indexerstellung • sollte jede Tabelle haben • kann aus einer oder mehreren Spalten bestehen (NOT NULL gilt für jede Spalte) • nur einen PK pro Tabelle möglich
FOREIGN KEY	• Erzwingt Referentielle Integrität • verweist auf den PK einer anderen Tabelle • Spalten dürfen nur in der anderen Tabelle enthaltene Werte aufnehmen oder NULL sein • NOT NULL muss wenn benötigt extra definiert sein
UNIQUE KEY	• Erzwingt Einmaligkeit eines Werts in einer Spalte • eindeutig • NULL-Werte sind erlaubt • automatische Indexerstellung • kann aus einer oder mehreren Spalten bestehen • mehrere pro Tabelle möglich

CHECK	• Erzwingt Zugehörigkeit eines Wertes zu Bereich • Einfache Gültigkeitsregeln / Geschäftsregeln • Verweis auf Inhalte derselben Datenzeile • kein Verweis auf andere Datensätze in derselben oder anderen Tabelle möglich • bei komplexen Geschäftsregeln werden Trigger benötigt
DEFAULT	• Befüllt bei der Eingabe einen Vorgabewert • Standardwerte • werden übernommen, wenn kein Eintrag erfolgt • werden nur bei Neuerfassungen befüllt • Ausnahme: Neue NOT NULL-Spalte wird mit Default-Wert an die Tabelle angefügt
NOT NULL	• Verhindert Eingabe von NULL-Marken

CONSTRAINT erstellen

<u>Beispiel 1:</u> Checkt ob **A**,**B** oder **C** in «artkat» vorhanden ist / Setzt Standardwert von «aktiv» auf 1	CREATE TABLE artikel (artnr int PRIMARY KEY, artbez varchar(100) NOT NULL, artkat char(1) CHECK (artkat in ('A','B','C')), aktiv bit DEFAULT 1,);
<u>Beispiel 2:</u> Checkt ob «artkat» **zwischen A** & **K** ist	CREATE TABLE kategorie (artkat char(1) PRIMARY KEY CHECK (artkat between 'A' and 'K'), katbez varchar(100) NOT NULL);
<u>Beispiel 3:</u> Checkt ob Alter **grösser gleich 18** ist / Benennt den *Check* **CHK_PersAge**	CREATE TABLE person (ID int NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255), Age int CONSTRAINT CHK_PersAge CHECK (Age >= 18));

Primary Key in bestehende Tabelle hinzufügen	ALTER TABLE Person ADD PRIMARY KEY (ID);
Foreign Key in bestehende Tabelle hinzufügen	ALTER TABLE artikel ADD FOREIGN KEY (artkat) REFERENCES kategorie(artkat);
Constraint-Namensvergebung in bestehende Tabelle	ALTER TABLE Person ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);
Constraint-Name löschen	ALTER TABLE Person DROP CONSTRAINT PK_Person;
Check hinzufügen	ALTER TABLE Persons ADD CHECK (Age>=18);
Check mit Namen hinzufügen	ALTER TABLE Persons ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');

DML – Data Manipulation Language

- Die DML Befehle werden verwendet, um den Inhalt einer Tabelle, also die Daten, zu manipulieren.
- Man wird vor keiner Mutation oder Löschung gewarnt!
- Umwandlung **Integer → String möglich**
- Umwandlung **String → Integer nicht möglich**
- INSERT:
  - Muss nach spalten Reihenfolge angegeben werden
  - Auf 'NOT NULL' achten!
  - Wenn keine Spalte angegeben wird muss überall etwas eingetragen werden. (Sei es bloss *NULL*)
    - Ausser bei Primary-/Foreign Key, da diese automatisch eingetragen werden
- DELETE:
  - Ohne WHERE-Angabe wird der ganze Tabelleninhalt gelöscht
  - Man kann nur eine Tabelle pro DELETE *löschen*

INSERT	**Daten erfassen** Einfügen neuer Werte in einer Tabelle unter Wahrung der Schlüsselintegrität. Übernehmen von Daten aus anderen Tabellen.
UPDATE	**Daten mutieren** Inhalt einer oder mehrerer Felder von bestehenden Datensätzen verändern.
DELETE	**Daten löschen** Löschen einer oder mehrerer ganzen Zeilen. Achtung: Löschen eines Feldinhaltes ist ein «UPDATE», kein «DELETE»!

INSERT – Syntax

Werte in Tabelle **eintragen**	INSERT [INTO] tabelle VALUES (wert1, wert2, wertx);
Werte in einzelne Spalten **eintragen**	INSERT [INTO] tabelle (spalte2, spalte3) VALUES (wert2, wert3);
Beispiel 1:	INSERT [INTO] tabelle VALUES (1, 'Meier', 'Ueli', NULL, '01.1.1970');
Beispiel 2:	INSERT [INTO] tabelle (spalte2, spalte3, spalte 5) VALUES ('Meier', 'Ueli', '01.1.1970');

BULK-INSERT aus anderer Tabelle

Daten von anderer Tabelle **übertragen**	INSERT INTO tabelle1 SELECT spalte1, spalte2, spalte3, spalte4 FROM tabelle2;

UPDATE – Syntax

Datensatz in einer Tabelle **ändern**	UPDATE tabelle SET spalte1 = wert1, spalte2 = wert2 WHERE spalte1 = wertx
Beispiel 1:	UPDATE Customers SET Name = 'Ueli', spalte2 = spalte4 WHERE CustomerID = 9;

DELETE – Syntax

Datensatz in einer Tabelle **löschen**	DELETE FROM tabelle WHERE spalte1 = wert1;
Beispiel 1:	DELETE FROM tabelle WHERE spalte5 < '01.01.2000';

Transaktionssteuerung

Merkmale – Transaktion

- Die Anweisungen laufen nach dem Prinzip ab: "ALLES ODER NICHTS" (vgl. ACID).
- Vor und nach einer Transaktion herrscht absolute Konsistenz.
- Änderungen werden erst gespeichert und für andere Benutzer sichtbar, wenn die Transaktion beendet ist.
- Solange die Transaktion läuft, sind alle betroffenen Daten gesperrt.
- Transaktionen sollten wegen der Sperren so kurz wie möglich dauern und möglichst wenig SQL-Code kapseln.
- Es besteht die Möglichkeit des Zurückrollens (Rollback), wenn in der Transaktion ein Fehler auftritt oder diese abbricht. Es handelt sich dabei um ein implizites Rollback in den Zustand vor der Transaktion.
- Für die Rückgängigmachung wird ein Logbuch geführt, auch Transaktions-Protokoll genannt. Dieses ist logisch aufgebaut, bestehend aus Anweisungen und nicht aus Zuständen.

TRANSACTION – Syntax

Transaktion **beginnen**	`BEGIN TRANSACTION test1;`

Datenbank ändern:
Tabelle hinzufügen
	`CREATE TABLE tabelle1(`
	`testID INT NOT NULL,`
	`testName VARCHAR(50)`
	`);`

Aktuellen Stand **zwischenspeichern**	`SAVE TRANSACTION test1;`
Datenbank ändern: Datensatz in Tabelle eintragen	`INSERT INTO tabelle1 (testName)` `VALUES ('Max');`
Transaktion zum Beginn oder zum letzten Zwischenspeicherpunkt **zurück setzen**	`ROLLBACK TRANSACTION test1;`
Datenbank ändern: Neuer Datensatz in Tabelle eintragen	`INSERT INTO tabelle1 (testName)` `VALUES ('Mike');`
Transaktion **beenden**	`COMIT TRANSACTION test1;`

DQL – Data Query Language

DQL Anweisungen werden zur Auswahl und Anzeige von Daten aus der Datenbank verwendet. Abfragen erstellen eine temporäre, speicherresidente (nicht persistente) Tabelle mit 0 oder mehreren Datenzeilen. Sie beginnen immer mit SELECT, gefolgt von der FROM-Klausel.

Mit SELECT wird ausgewählt, welche Spalten der Tabelle in welcher Reihenfolge zurückgegeben werden sollen. Mit * werden alle Spalten ausgewählt, entspricht der Spaltenreihenfolge der in der zugrundeliegenden Tabelle.

Nach FROM gibt man den Namen der Tabelle an, welche abgefragt werden soll.

SELECT – Syntax

Alle Spalten von «tabelle» anzeigen lassen	SELECT * FROM tabelle;
Nur **spezifische Spalten** anzeigen lassen	SELECT spalte1, spalte2, spalte3 FROM tabelle;
Aliasnamen für Spalten	SELECT spalte1 AS alias1, ausdr1 AS alias2 FROM tabelle;
Doppelte Einträge weglassen	SELECT DISTINCT spalte1 FORM tabelle;
Berechnungsausdrücke werden wie Tabellenspalten angegeben	SELECT spalte2, (spalte1 – spalte3) * 5 AS Spaltenname FROM tabelle;
Verkettung von Texten	SELECT spalte1 + ' zwischentext ' + spalte2 FROM tabelle;
Beispiel:	SELECT plz + ' ' + city AS 'PLZ/Ort' FROM tabelle;

Bedingung ausdrücken	SELECT spalte1, spalte2, CASE ausdr WHEN bedingungsausdr1 THEN fall1 WHEN bedingungsausdr2 THEN fall2 … ELSE fallx END AS Spaltenname FROM tabelle
Beispiel:	SELECT OrderID, Quantity, CASE WHEN Quantity > 30 THEN 'Die Anzahl ist grösser als 30' WHEN Quantity = 30 THEN 'Die Anzahl ist 30' ELSE 'Die Anzahl ist kleiner als 30' END AS QuantityText FROM OrderDetails;

WHERE – Klausel

WHERE – Merkmale

- Auswertung nach mathematischen Gesichtspunkten
- Alle Vergleichsoperatoren sind möglich
- Für gültige Vergleiche muss auf das Datumsformat geachtet werden
- Unterschiedlich bei verschiedenen Gerstellern
- Numerische Datumsformate:
  - 01.01.2021
  - 01-01-2021
- Texte werden wie Datumswerte von einfachen Hochkommas umschlossen
- Vergleiche erfolgen von links nach rechts nach dem Alphabet
- Gross-/Kleinschreibung bei SQL-Server in der Regel nicht relevant (bei Oracle schon)
- Die Vergleichsspalte in der WHERE-Klausel muss **nicht** in der SELECT-Klausel enthalten sein

WHERE – Syntax

Nur Zeilen anzeigen welche ein Land **gleich** Mexico eingetragen haben	SELECT * FROM Customers WHERE Country = 'Mexico';
Länder welche einen Anfangsbuchstaben von **grösser als** L haben	SELECT * FROM Customers WHERE Country > 'L';
Bestelldaten von **kleiner als** 15.07.2020	SELECT * FROM Orders WHERE OrderDate < '15.07.2020';
Produkte mit Preis von **grösser gleich** 50	SELECT * FROM Products WHERE Price >= 50;
«spalte» muss **a**, **b** oder **c** beinhalten	… WHERE spalte LIKE '[abc]';
«spalte» darf **a**, **b** oder **c** nicht beinhalten	… WHERE spalte LIKE '[^abc]';
Städte die mit **ber** beginnen	… WHERE city LIKE 'ber%';
Städte die mit **La** oder **Li** beginnen	… WHERE city LIKE 'L[ai]%';
Städte welche als zweiten Buchstaben **e** haben	… WHERE city LIKE '_e%';
Städte die *kein* **a** beinhalten	… WHERE city LIKE '%[a]%';
Geburtsdatum zwischen **01.01.1950** und **31.12.1959**	… WHERE BirthDate BETWEEN '01.01.1950' AND '31.12.1959';
Kunden die aus den Städten **Bern** oder **Paris** kommen	SELECT * FROM Customers WHERE City IN ('Bern', 'Paris');
Artikel die **keinen** Preis eingetragen haben	SELECT artbez, artkat FROM artikel WHERE artpreis IS NULL;
Artikel die **einen** Preis eingetragen haben	SELECT artbez, artkat FROM artikel WHERE artpreis IS NOT NULL;

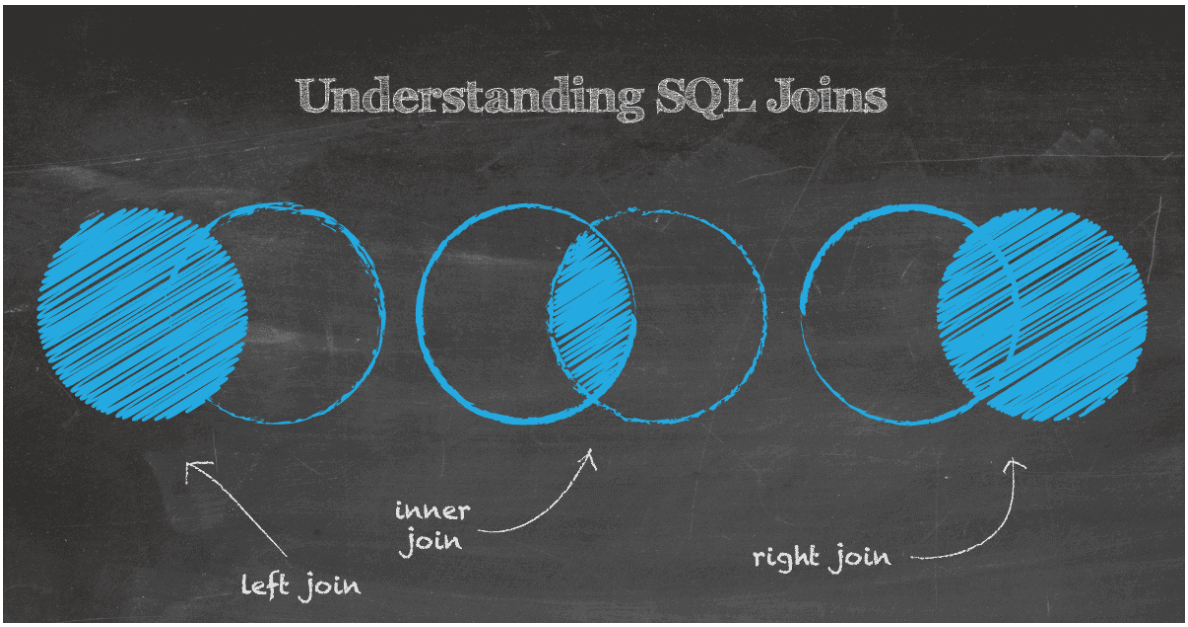
ORDER BY – Syntax

Sortierung hinzufügen	SELECT * FROM tabelle WHERE spalte = wert ORDER BY spalte3 [ASC|DESC];
Mit **ASC** aufsteigend sortieren (keine Angabe = **ASC**)	… ORDER BY City ASC; -- oder… … ORDER BY City;
Mit **DESC** absteigend sortieren	… ORDER BY City DESC;

TOP – Syntax

Die «obersten/untersten» **10** anzeigen (je nachdem ob *ASC* oder *DESC*)	SELECT TOP 10 * FROM…
Die «obersten/untersten» **10 Prozent** anzeigen (je nachdem ob *ASC* oder *DESC*)	SELECT TOP 25 PERCENT * FROM…

JOIN – Tabellen verknüpfen



Verknüpfung von zwei Tabellen kann, muss aber nicht über die *Primär-/Fremdschlüssel-Beziehung* erfolgen. Die meisten Joins sind EQUI JOINS. Darin werden zwei Tabellen über übereinstimmende Spalten verknüpft. Als Join-Operator wird ein = verwendet. Beispiel: Rechnungs- und Kundentabelle werden über die Kundennummer, die in beiden Tabellen enthalten ist, verknüpft.

Die seltene Sonderform eines Joins ist der NONEQUI JOIN. Bei Nonequi Joins gibt es keine direkt aufeinander referenzierenden Spalten. Als Join-Operator wird kein = verwendet, sondern meist kommt ein BETWEEN … AND … zum Einsatz.

Beispiel – JOIN

Tabelle: tab1

	id	firstname	lastname
1	1	arun	prasanth
2	2	ann	antony
3	3	sruthy	abc
4	6	new	abcd

Tabelle: tab2

	id2	age	place
1	1	24	kerala
2	2	24	usa
3	3	25	ekm
4	5	24	chennai

https://stackoverflow.com/questions/5706437/whats-the-difference-between-inner-join-left-join-right-join-and-full-join/28719292#28719292?newreg=9fa05e2b422541628ed691c3d1c3fdef

INNER JOIN

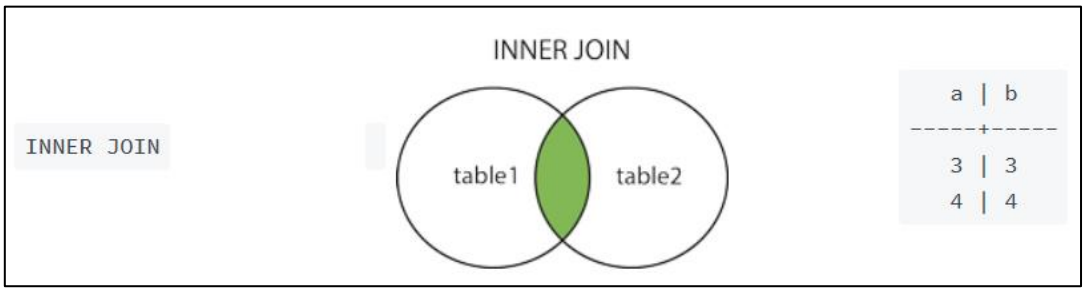
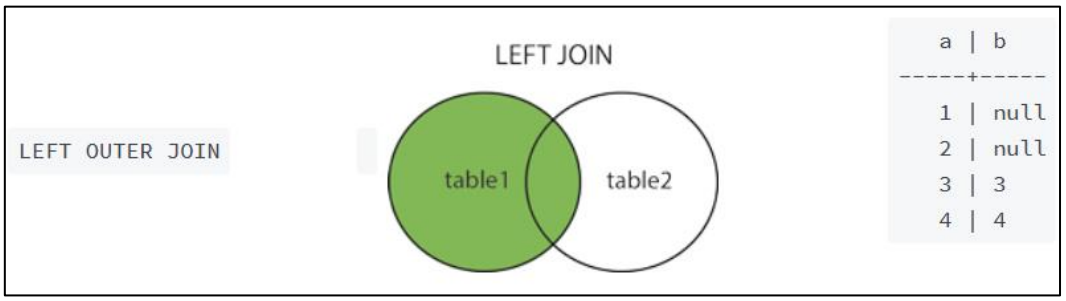


tabelle1 mit tabelle2 **verknüpfen**	SELECT tabelle1.spalte1, tabelle2.spalte1 FROM tabelle1 [INNER] JOIN tabelle2 ON tabelle1.spaltex = tabelle2.spaltey
Tabellen-Alias hinzufügen	SELECT t1.spalte1, t2.spalte1 FROM tabelle1 AS t1 [INNER] JOIN tabelle2 AS t2 ON tabelle1.spaltex = tabelle2.spaltey
Beispiel 1:	SELECT c.ContactName, o.OrderID FROM Customers c JOIN Orders o ON c.CustomerID = o.CustomerID
Beispiel 2:	SELECT c.ContactName, o.OrderID FROM Customers c JOIN Orders o ON c.CustomerID = o.CustomerID JOIN [Order Details] od ON o.OrderID = od.OrderID JOIN Products p ON od.ProductID = p.ProductID
Beispiel - JOIN:	SELECT tab1.FirstName, tab1.LastName, tab2.Age, tab2.Place FROM tab1 JOIN tab2 ON tab1.id = tab2.id2

	firstName	lastName	age	Place
1	arun	prasanth	24	kerala
2	ann	antony	24	usa
3	sruthy	abc	25	ekm

LEFT OUTER JOIN

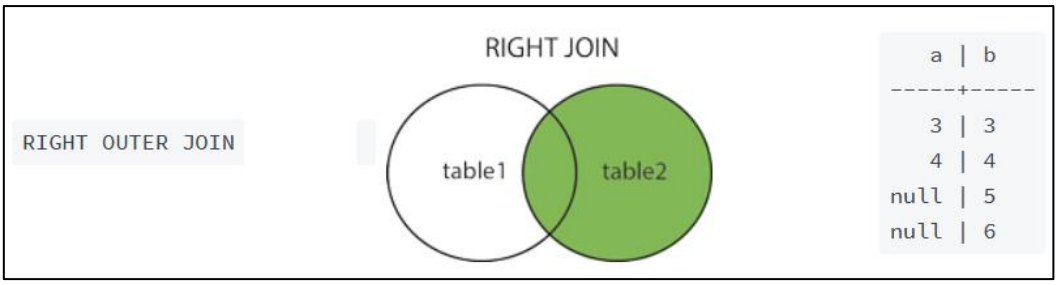


Beispiel - JOIN:

SELECT tab1.FirstName, tab1.LastName, tab2.Age, tab2.Place FROM tab1 LEFT [OUTER] JOIN tab2 ON tab1.id = tab2.id2

	firstName	lastName	age	Place
1	arun	prasanth	24	kerala
2	ann	antony	24	usa
3	sruthy	abc	25	ekm
4	new	abcd	*NULL*	*NULL*

RIGHT OUTER JOIN

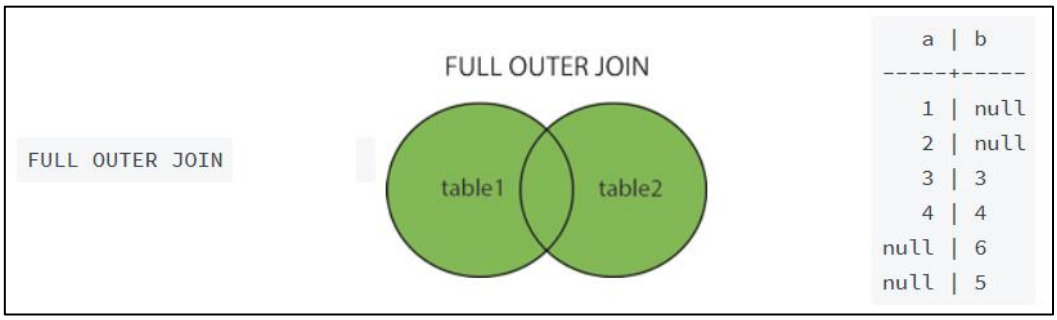


Beispiel - JOIN:

SELECT tab1.FirstName, tab1.LastName, tab2.Age, tab2.Place FROM tab1 RIGHT [OUTER] JOIN tab2 ON tab1.id = tab2.id2

	firstName	lastName	age	Place
1	arun	prasanth	24	kerala
2	ann	antony	24	usa
3	sruthy	abc	25	ekm
4	*NULL*	*NULL*	24	chennai

FULL OUTER JOIN



Beispiel - JOIN:

```
SELECT tab1.FirstName, tab1.LastName,
tab2.Age, tab2.sPlace FROM tab1
FULL [ OUTER ] JOIN tab2 ON tab1.id = tab2.id2
```

	firstName	lastName	age	Place
1	arun	prasanth	24	kerala
2	ann	antony	24	usa
3	sruthy	abc	25	ekm
4	new	abcd	*NULL*	*NULL*
5	*NULL*	*NULL*	24	chennai

SELF JOIN

staff_id	first_name	last_name	email	phone	active	store_id	manager_id
1	Fabiola	Jackson	fabiola.jackson@bikes.shop	(831) 555-5554	1	1	NULL
2	Mireya	Copeland	mireya.copeland@bikes.shop	(831) 555-5555	1	1	1
3	Genna	Serrano	genna.serrano@bikes.shop	(831) 555-5556	1	1	2
4	Virgie	Wiggins	virgie.wiggins@bikes.shop	(831) 555-5557	1	1	2
5	Jannette	David	jannette.david@bikes.shop	(516) 379-4444	1	2	1
6	Marcelene	Boyer	marcelene.boyer@bikes.shop	(516) 379-4445	1	2	5
7	Venita	Daniel	venita.daniel@bikes.shop	(516) 379-4446	1	2	5
8	Kali	Vargas	kali.vargas@bikes.shop	(972) 530-5555	1	3	1
9	Layla	Terrell	layla.terrell@bikes.shop	(972) 530-5556	1	3	7
10	Bernardine	Houston	bernardine.houston@bikes.shop	(972) 530-5557	1	3	7

Beispiel

```
SELECT e.first_name + ' ' + e.last_name AS employee,
m.first_name + ' ' + m.last_name AS manager
FROM sales.staffs m
JOIN sales.staffs m ON m.staff_id = e.manager_id
ORDER BY manager;
```

employee	manager
Mireya Copeland	Fabiola Jackson
Jannette David	Fabiola Jackson
Kali Vargas	Fabiola Jackson
Marcelene Boyer	Jannette David
Venita Daniel	Jannette David
Genna Serrano	Mireya Copeland
Virgie Wiggins	Mireya Copeland
Layla Terrell	Venita Daniel
Bernardine Houston	Venita Daniel

Aggregatfunktionen – Syntax

Anzahl Mitarbeiter **zählen**	SELECT COUNT(*) [Anzahl Mitarbeiter] FROM Employees
Anzahl Regionen **zählen**	SELECT COUNT(Region) [Anzahl Regionen] FROM Employees
Anzahl Regionen **zählen** (inkl. leere Einträge)	SELECT COUNT(ISNULL(Region,1)) [Anzahl Regionen] FROM Employees
Summe der Bestellmenge	SELECT SUM(Quantity) [Bestellmenge] FROM Orders
Summe des Bestellumsatzes	SELECT SUM(Quantity*UnitPrice) [Bestellumsatz] FROM Orders
kleinste & **grösste** Geburtsdatum (jüngste / älteste)	SELECT MIN(BirthDate), MAX(BirthDate) FROM Employees
Durchschnittlicher Produktpreis	SELECT AVG(UnitPrice) [Durchschnittspreis] FROM Products

Weitere Funktionen

SQL References → SQL Server Functions w3schools SQL-Seite als .zip

GROUP BY – Syntax

Nach Stadt **gruppieren** (GROUP BY)	SELECT city, COUNT(*) [Anzahl MA pro Stadt] FROM Employees GROUP BY city;
Bedingung für Gruppierung hinzufügen (HAVING)	SELECT OrderID, SUM(Quantity) [Bestellmenge pro Bestelleung] FROM [Order Details] WHERE OrderID > 11000 GROUP BY OrderID HAVING SUM(Quantity) < 100 ORDER BY 2 DESC;

Unterabfragen – Syntax

Alle Länder welche in «Suppliers» **vorhanden** sind	SELECT * FROM Customers WHERE Country IN (SELECT Country FROM Suppliers)
Land welches mit dem **grössten** Buchstaben beginnt	SELECT * FROM Customers WHERE Country IN (SELECT MAX(City) FROM Customers)
Bestellung mit einer «OrderID» von **11'000**	SELECT * FROM Customers c JOIN (SELECT OrderID, CustomerID FROM Orders WHERE OrderID = 11000) AS o ON c.CustomerID = o.CustomerID

Mengen-Operationen

Merkmale – Mengen-Operationen

- UNION [ALL] verbindet mehrere SELECT-Anweisungen zu einem Gesamtergebnis «ALL» (Duplikate aus verschiedenen Anweisungen werden nicht unterdrückt)
  - Spalten der Tabellen müssen den gleichen Namen tragen
- INTERSECT bringt alle Zeilen, die jede der SELECT-Anweisungen zurückliefert. Dies ist die Schnittmenge.
- EXCEPT bringt alles aus der ersten SELECT-Anweisung, das in der folgenden Anweisung nicht vorkommt. (Unter Oracle «MINUS»)

UNION – Syntax

Zeigt **alle Datensätze**, der beiden Spalten, von diesen Tabellen an	SELECT City, Country FROM Employees UNION SELECT City, Country FROM Customers
ALL zeigt auch **doppelte** Einträge an	SELECT City, Country FROM Employees UNION ALL SELECT City, Country FROM Customers

INTERSECT – Syntax

Zeigt **nur** Datensätze an, welche in **beiden** Tabellen vorkommen	SELECT City, Country FROM Employees INTERSECT SELECT City, Country FROM Customers

EXCEPT – Syntax

Zeigt Datensätze, welche **nur** in der **ersten** SELECT-Anweisung vorkommen	SELECT City, Country FROM Employees EXCEPT SELECT City, Country FROM Customers

VIEW – Syntax

VIEW erstellen	CREATE VIEW view_name AS SELECT … [WITH CHECK OPTION]
VIEW abfragen	SELECT * FROM view_name
VIEW ändern	ALTER VIEW view_name AS SELECT … [WITH CHECK OPTION]
VIEW löschen	DROP VIEW view_name

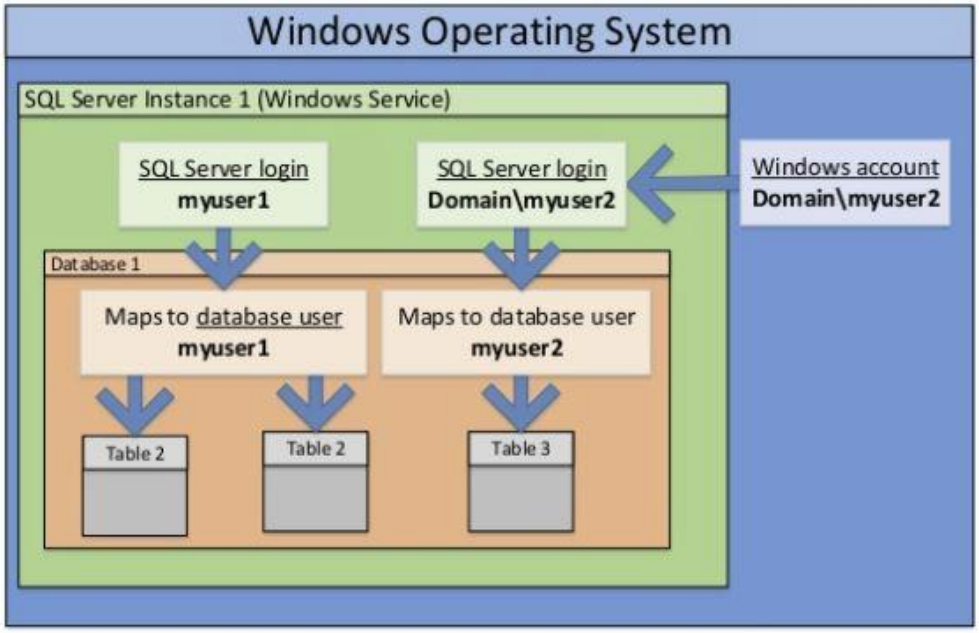
Einschränkungen / Darf nicht enthalten sein…

- ORDER BY – Klausel
- INTO – Schlüsselwort
- Verweis auf temporäre Tabellen
- DML enthält:
  - Aggregatfunktionen
  - GROUP BY
  - TOP
  - UNION
  - DISTINCT
  - Berechnete Spalten in der SELECT – Klausel

DCL – Data Control Language

SQL Server – Berechtigungskonzept

Bei SQL-Server erfolgt der Zugriff auf Datenbanken in zwei Stufen: Mit dem **Server Login** meldet man sich auf dem SQL-Server an, mit dem **Database User** greift man auf eine Datenbank zu. In jeder einzelnen Datenbank, auf die ein Anwender zugreifen möchte, wird ein separater User benötigt. Dieser User wird beim Anlegen einem Login zugeordnet. So kommt ein Endanwender lediglich mit dem Login in Kontakt. Ein Passwort ist nur für das Login erforderlich. Damit in der Praxis die Zugriffsverwaltung übersichtlich bleibt und erleichtert wird, ist es ratsam, dem einem Login zugeordneten User jeweils denselben Namen zu vergeben.



Die Authentifizierung am SQL-Server kann entweder über ein Windows-Konto oder über ein SQL-Server-Konto erfolgen. Bei der Windows-Authentifizierung werden Windows-Domänenkonten als Logins auf dem SQL-Server registriert. Diese Variante hat für den Endanwender den Vorteil, dass dieser sich keinen weiteren Kontonamen samt Passwort für die Anmeldung merken muss. Für alle anderen Anwender ausserhalb der Windows-Domäne muss ein eigenes SQL-Server-Login samt Passwort erstellt werden.

Principal erstellen & Rolle zuweisen

- Vor dem Zuordnen des Users zum Login muss man sich in der entsprechenden DB befinden.
- Jeder neu angelegte User befindet sich automatisch in der Datenbankrolle **public.**
- Benutzername von Database **muss nicht gleich** Server Benutzername sein!

Server Login	Zuweisung von Berechtigungen auf Serverebene. In der Regel keine direkte Zuweisung von Berechtigungen, sondern Mitgliedschaft in bestimmten Serverrollen. Login-Berechtigungen werden in der System-Datenbank *master* gespeichert.	- sysadmin - serveradmin - securityadmin
Database User	Zuweisungen von Berechtigungen innerhalb einer Datenbank. Mit der Zuweisung von bereits vordefinierten Datenbankrollen kann eine einfache Berechtigungsverwaltung umgesetzt werden. User, Datenbankrollen und die an sie erteilten Berechtigungen werden in der jeweiligen Datenbank gespeichert.	- db_datareader - db_datawriter

Benutzer anlegen – Syntax

Server-Login inkl. **Passwort** anlegen	CREATE LOGIN user1 WITH PASSWORD = 'pwd123'
Ablauf des Kennworts & Anwendung der **Kennwortrichtlinien**	CREATE LOGIN user1 WITH PASSWORD = 'pwd123', CHECK_EXPIRATION = OFF, CHECK_POLICY = ON
Benutzer eine **Server-Rolle** hinzufügen	ALTER SERVER ROLE role_name ADD MEMBER user1
Database-User für Server-Login anlegen	CREATE USER user1 FOR LOGIN user1
Benutzer eine **Datenbank-Rolle** hinzufügen	ALTER ROLE role_name ADD MEMBER user1

Berechtigungen verwalten

- **Objektberechtigungen** erlauben den Zugriff auf Objekte innerhalb der Datenbank.
- **Anweisungsberechtigungen** werden "gewöhnlichen" Datenbankbenutzern in der Regel nicht gewährt. Sie beziehen sich nicht auf bestehende Objekte, sondern legen fest, wer Datenbankobjekte erstellen, verwalten und sichern darf.
- Datenbanken sind Hochsicherheitstrakte: Alles was nicht **explizit** erlaubt wird, ist verboten.
- Mit **DENY** kann man Berechtigungen explizit verweigern, damit diese nicht indirekt über Rollenmitgliedschaften erlangt werden können.

Anweisungsberechtigungen	Objektberechtigungen
CREATE DATABASE	SELECT
CREATE DEFAULT	INSERT
CREATE FUNCTION	DELETE
CREATE PROCEDURE	REFERENCES
CREATE TABLE	UPDATE
CREATE VIEW	EXECUTE
BACKUP DATABASE	
BACKUP LOG	

Berechtigungen vergeben – Syntax

Anweisungsberechtigung	GRANT statement TO user
Beispiel:	GRANT CREATE TABLE TO user1
Objektberechtigung	GRANT berechtigung ON objekte TO user
User darf diese Berechtigung **weitergeben**	GRANT berechtigung ON objekte TO user WITH GRANT OPTION
Rechte an **alle** vergeben	GRANT berechtigung ON objekte TO PUBLIC
Beispiel:	GRANT SELECT, UPDATE ON tabelle1 TO user1, user2

Berechtigungen entziehen – Syntax

Anweisungsberechtigung	REVOKE statement FROM user
Beispiel:	REVOKE CREATE TABLE FROM user1
Objektberechtigung	REVOKE berechtigung ON objekte FROM user
Rechte zur **Weitergabe** entziehen	REVOKE GRANT OPTION FOR berechtigung ON objekte FROM user CASCADE
Beispiel:	REVOKE SELECT, UPDATE ON tabelle1 FROM user1, user2

Berechtigungen verweigern – Syntax

Anweisungsberechtigung	DENY statement TO user
Beispiel:	DENY CREATE TABLE TO user1
Objektberechtigung	DENY berechtigung ON objekte TO user
Beispiel:	DENY SELECT, UPDATE ON tabell1 TO user1, user2

Datenbanksicherung

Vollständige/Differenzielle Sicherung

Vollständige Sicherung

Enthält alle Daten einer bestimmten Datenbank zum Zeitpunkt der Sicherung. Bei einem Restore wird aus dieser Sicherung die komplette Datenbank wiederhergestellt.

```
BACKUP DATABASE [datenbank_name]
TO DISK = N'pfad'
WITH NOFORMAT, NOINIT, NAME = N'name', SKIP, NOREWIND, NOUNLOAD, STATS = 10
GO
```

Differenzielle Sicherung

Basiert auf der letzten vollständigen Sicherung und enthält nur die Daten, die sich geändert haben. Es werden nur jene Datenblöcke gesichert, die sich seit der letzten vollständigen Sicherung geändert haben. Bei einem Restore wird auch die entsprechende Vollsicherung benötigt. Voraussetzung für die differenzielle Sicherung ist also eine vollständige Datenbanksicherung.

```
BACKUP DATABASE [NORTHWIND]
TO DISK = 'pfad'
WITH DIFFERENTIAL;
GO
```

Beispiel:

```
BACKUP DATABASE [NORTHWIND]
TO DISK = N' C:\Program Files\Microsoft SQL
Server\MSSQL14.SQLEXPRESS\MSSQL\Backup\NORTHWIND.bak'
WITH NOFORMAT, NOINIT, NAME = N' NORTHWIND-Vollständig Datenbank Sichern ',
SKIP, NOREWIND, NOUNLOAD, STATS = 10
GO
```

Transaktionsprotokoll-Sicherung

Transaktionsprotokoll-Sicherung

Das Transaktionsprotokoll beinhaltet alle Veränderungen auf der Datenbank, die seit der letzten vollständigen, differenziellen oder Transaktionsprotokoll-Sicherung abgeschlossen wurden. Mithilfe einer Transaktionsprotokollsicherung kann der Zustand der Datenbank bei einem Ausfall bis zu diesem Zeitpunkt wiederhergestellt werden.

```
BACKUP LOG [datenbank_name]
TO DISK = N'pfad'
WITH INIT, NAME = N'name'
GO
```

Beispiel:

```
BACKUP LOG [NORTHWIND]
TO DISK = N' C:\Program Files\Microsoft SQL
Server\MSSQL14.SQLEXPRESS\MSSQL\Backup\NORTHWIND_LOG.trn'
WITH INIT, NAME = N'Northwind_Log'
GO
```

Wiederherstellen der Datenbank

Drei Modelle für die Wiederherstellung

Einfach

SQL Server verwaltet das zugehörige Transaktionsprotokoll automatisch, inaktive Teile des Protokolls werden automatisch gelöscht. Das Transaktionsprotokoll wird automatisch abgeschnitten, sobald die Transaktionen sicher im dauerhaften Speicher angekommen sind. Das Transaktionsprotokoll wird somit immer klein gehalten und kann nicht unbegrenzt wachsen. Nachteil: Das Transaktionsprotokoll kann nicht extra gesichert werden. Dieses Modell wird nur bei Test- oder Entwicklungsdatenbanken empfohlen.

Vollständig

Es werden alle Transaktionen im Protokoll gehalten und der inaktive Teil nicht gelöscht. Erst wenn die Transaktionsprotokollsicherung ausgeführt wird, wird der inaktive Teil gelöscht. Im Fall einer Wiederherstellung der Datenbank wird zuerst die DB Sicherung und danach die Transaktionsprotokollsicherung eingespielt. Denn nur mit der Transaktionsprotokollsicherung erreicht man, dass alles bis zur letzten Datenbankänderung restored wurde. Nachteil: Das Transaktionsprotokoll kann schnell ziemlich gross werden, falls nicht laufend eine Transaktionsprotokoll- Sicherung durchgeführt wird und dieses gelöscht wird. In diesem Fall läuft man Gefahr, dass die Festplatte voll und die Datenbank lahmgelegt wird. Dieses Modell wird üblicherweise bei Produktionsdatenbanken verwendet.

Massenprotokolliert

Ähnlich wie beim vollständigen Modell, das Transaktionsprotokoll wächst allerdings nicht so schnell. Diese Option wird vor allem bei Datenbanken mit vielen Massenoperationen (bulk inserts) gewählt.

Wiederherstellungsmodelle bei SQL Server

Die Wahl der Sicherungsvariante hängt eng mit dem Wiederherstellungsmodell der Datenbank zusammen. Beim Wiederherstellungsmodell handelt es sich um eine Datenbankeigenschaft, welche festlegt, wie SQL Server die Daten zum Transaktionsprotokoll speichert und verwaltet, wenn die Transaktionen abgeschlossen sind.

```
RESTORE DATABASE [datenbank_name]
FROM DISK = N'pfad'
```

Beispiel:

```
RESTORE DATABASE [NORTHWIND]
FROM DISK = N' C:\Program Files\Microsoft SQL
Server\MSSQL14.SQLEXPRESS\MSSQL\Backup\NORTHWIND.bak'
```

Anhang

Definition von Begriffen

Referentielle Integrität

Wenn aus einer anderen Tabelle ein Fremdschlüssel auf einen hier zu löschenden Datensatz zeigt, kann er nicht gelöscht werden. Um den Löschvorgang durchführen zu können, muss zuerst der Datensatz aus der anderen Tabelle gelöscht werden.

Konsistenz

Als Konsistenz wird in Datenbanken die Korrektheit der dort gespeicherten Daten bezeichnet. Inkonsistente Datenbanken können zu schweren Fehlern führen, falls die darüberliegende Anwendungsschicht nicht damit rechnet.

Persistenz

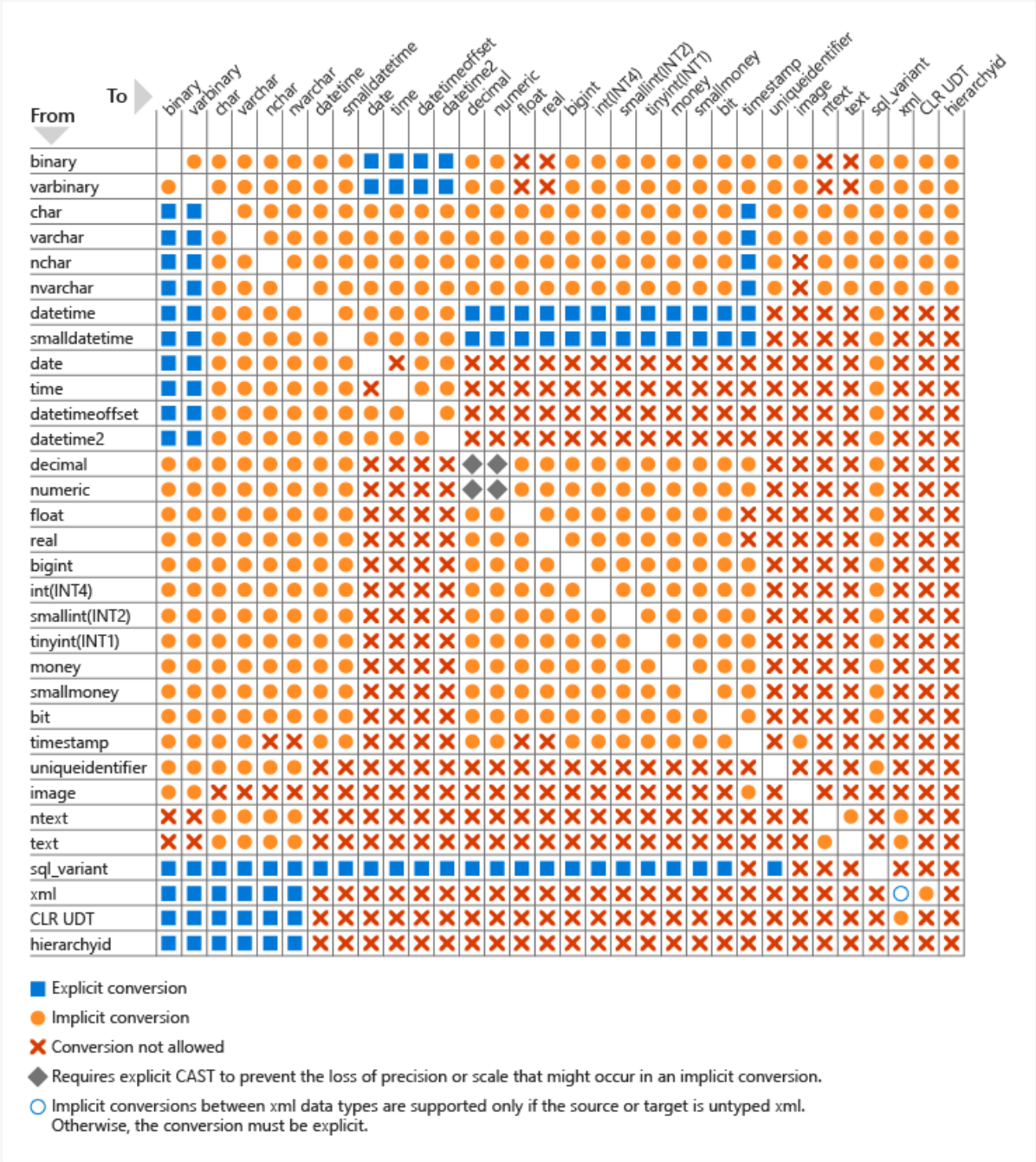
Datenpersistenz meint, dass in einem DBMS einzelne Daten solange aufbewahrt werden müssen, bis sie explizit gelöscht werden.

Deterministisch

Bei gleichen Ausgangsbedingungen durchlaufen diese immer dieselben Schritte und liefern das gleiche Ergebnis.

Zulässige Datentypkonvertierungen bei SQL Server

Implizite Konvertierungen sind Konvertierungen, die ohne Angabe der CAST- oder CONVERT-Funktion durchgeführt werden. Explizite Konvertierungen sind Konvertierungen, die die Angabe der CAST- oder CONVERT-Funktion erfordern. In der folgenden Abbildung werden alle expliziten und impliziten Datentypkonvertierungen aufgeführt, die für die vom SQL Server-System bereitgestellten Datentypen zulässig sind.



Restore Fehler – Lösung

```
USE [master]
RESTORE DATABASE [dbname]
FROM DISK = N'C:\Program Files\Microsoft SQL
Server\MSSQL14.MSSQLSERVER\MSSQL\Backup\dbname.bak'
WITH FILE = 1,
MOVE N'dbname' TO N'C:\Program Files\Microsoft SQL
Server\MSSQL14.MSSQLSERVER\MSSQL\DATA\dbname.mdf',
MOVE N'dbname_log' TO N'C:\Program Files\Microsoft SQL
Server\MSSQL14.MSSQLSERVER\MSSQL\DATA\dbname_log.ldf', NOUNLOAD, STATS = 5
```

Datenbank kann nicht gelöscht werden – Lösung

```
USE master;
GO
ALTER DATABASE database_name SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
GO
DROP DATABASE database_name;
GO
```

w3schools SQL-Seite als .zip

w3schools-SQL.zip

Darf benutzt werden, da man die Site lokal öffnet.
Somit wird nicht vom Internetzugang gebraucht gemacht.