

Zusammenfassung - Datenmanagement

17 February 2015 08:38

Version: 1.0.0

Study: 2. Semester, Bachelor in Business and Computer Science

School: Hochschule Luzern - Wirtschaft

Author: Janik von Rotz (<http://janikvonrotz.ch>)

License:

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License.

To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Anomalien

17 February 2015 08:38

Anomalien werden anhand einer fehlerbehafteten Tabellen beschrieben.

<i>id_pers</i>	<i>name</i>	<i>abt#</i>	<i>abt_budget</i>	<i>kurs#</i>	<i>kurs_name</i>	<i>datum</i>
505	Müller	15	72'000	X17	Verkauf	03.01.2005
425	Meier	15	72'000	X17	Verkauf	05.10.2005
425	Meier	15	72'000	L91	Rhetorik	15.09.2005
425	Meier	15	72'000	L98	Führung	11.11.2005
458	Schulze	18	33'500	L91	Rhetorik	15.09.2005
703	Schmid	26	61'900	X17	Verkauf	12.12.2005
703	Schmid	26	61'900	L31	Oracle	10.02.2005

Mutationsanomalie

Eine Mutation hat mehrere Zugriffe zur Folge.

Einfügeanomalie

Neue Datensätze können nicht korrekt eingefügt werden, bzw. wird die Eingabe der Werte nicht kontrolliert.

Löschanomalie

Ein Löschzugriff hat unbeabsichtigte Folgen. Möchte man z.B. Meier identifiziert durch die *id_pers* löschen, hat das gleich 3 Löschungen zur Folge.

Datenbankarchitektur

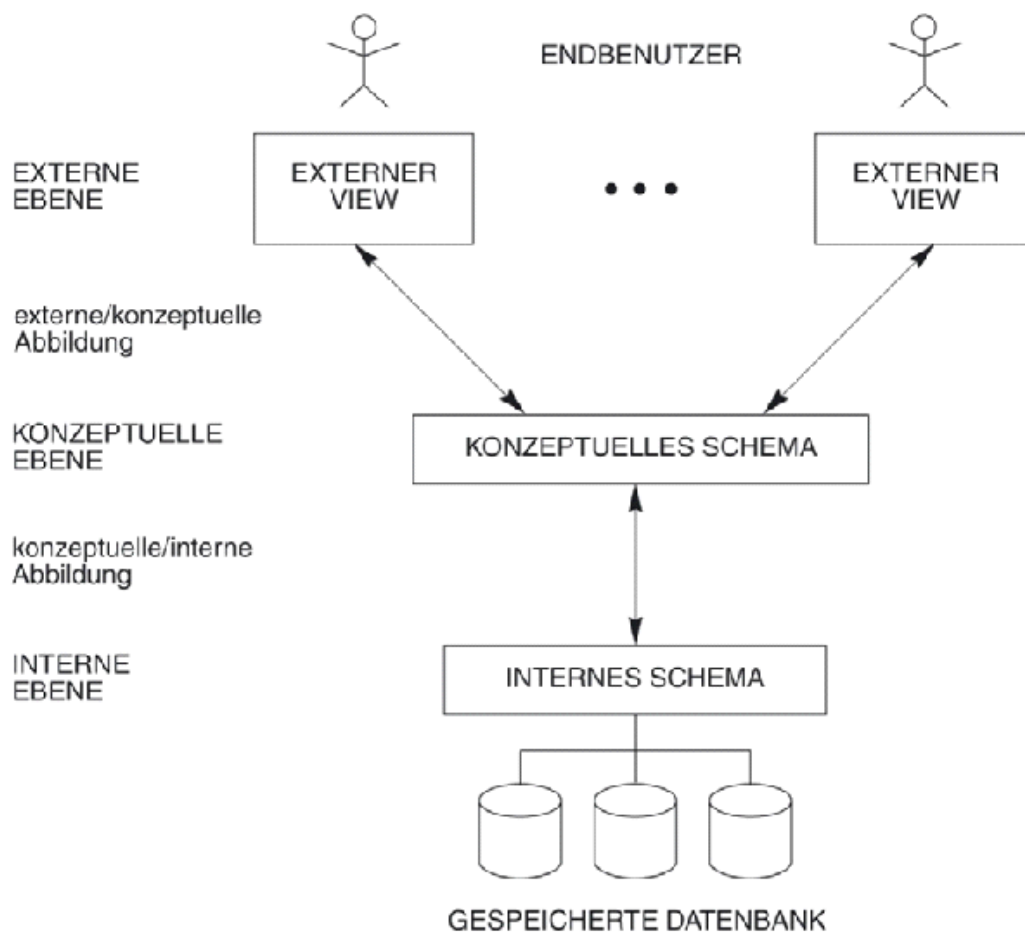
17 February 2015 08:47

Vorteile von Datenbanken

- Redundanzen vermeiden -> Keine Mehrfachspeicherung
- Verhinderung von Inkonsistenz -> Synchronisierung Zugriff und Mutation
- Durchsetzung Datenschutz -> Zugriffsrechte
- Gewährleistung Datensicherheit -> Datenverlust vermeiden
- Datenunabhängigkeit -> Zentrale Stelle

ANSI / SPARC Modell

Beschreibt eine 3-Ebenen Architektur.



Externe Ebene

Präsentiert die Benutzerschicht. Übernimmt die Darstellung der Daten in Form einer Applikation.

Konzeptuelle Ebene

Beschreibt die Struktur der Datenbank. Hier gibt es verschiedene Modellierungs-Modelle sogenannte ERM, Entity Relationship Model.

Interne Ebene

Beschreibt die physikalische Speicherstruktur, spricht wie werden die Daten effektiv und physisch abgespeichert.

Modellierung

17 February 2015 08:54

Ist die Vereinfachung und Abstraktion von Zusammenhängen in der Realität. Der Verwendungszweck spielt beim Prozess eine zentrale Rolle.

Modelle können anhand 3 semiotischen Aspekten beurteilt werden.

- Syntax -> Welche Elemente müssen in Beziehung gesetzt werden (Form).
- Semantik -> Welche Bedeutung haben die Elemente (Bedeutung).
- Pragmatik -> Welche Wirkung wird beabsichtigt (Verwendung).

Beispiel:



Syntax:

- Farbcode
- Kreise
- Linien
- Zonen

Semantik:

- Linien anhand Farbcodes
- Tarifzonen
- Verbindungen, Strecken
- Haltestellen

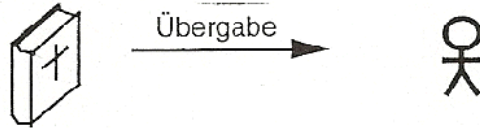
Pragmatik:

- Wegbeschreibung
- Orientierungskarte

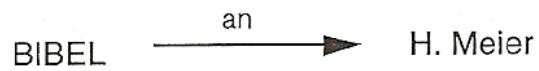
Vorgehen

Die Modellierung einer Daten erfolgt in verschiedenen Prozessen

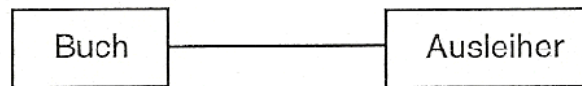
1. reale Welt,
Teile der realen Welt:



2. Informationen über
Teile der realen Welt,
Modelle:



3. Logisches
Datensystem:



4. Physisches
Datensystem:

physische Datensätze und Dateien

5. Computerspeicherung
(Hardware+Software):

Speichermedien und deren Inhalt

↑
Datenbankbereich
↓

ERM

17 February 2015 09:07

Das ERM arbeitet mit abstrahierten Objekten und deren Beziehungen zu einander.

Beispiel:

Wir haben eine Datensammlung von Autoren und Büchern.

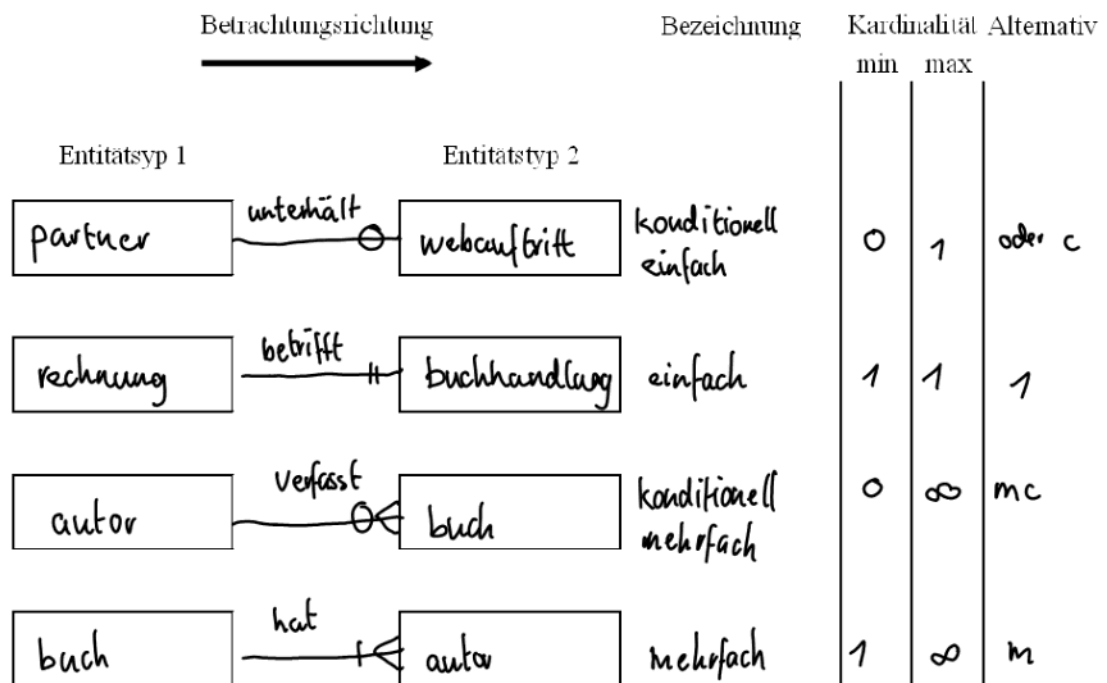
Diese Objekte haben jeweils eigene Attribute.

- Autor (Vorname, Nachname)
- Buch (Titel, Jahr, ISBN)

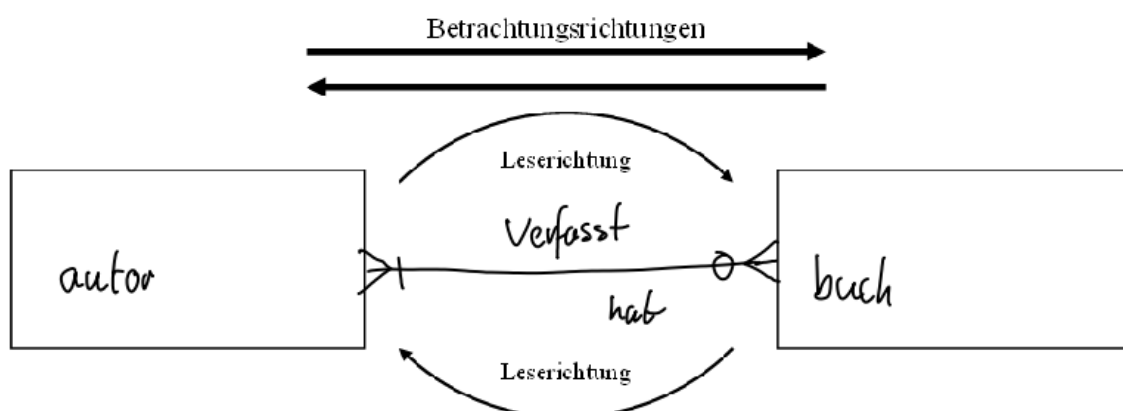
Diese 2 Objekte werden nun zueinander in Beziehung gesetzt. So beschreibt man welcher Autor welches Buch beschrieben hat. Dabei unterscheidet man zwischen verschiedenen Beziehungstypen.

Beziehungen

Assoziationstypen können visuell, mit Nummern, oder Codes dargestellt werden.



Diese Beziehungen müssen natürlich in beide Richtungen betrachtet werden.

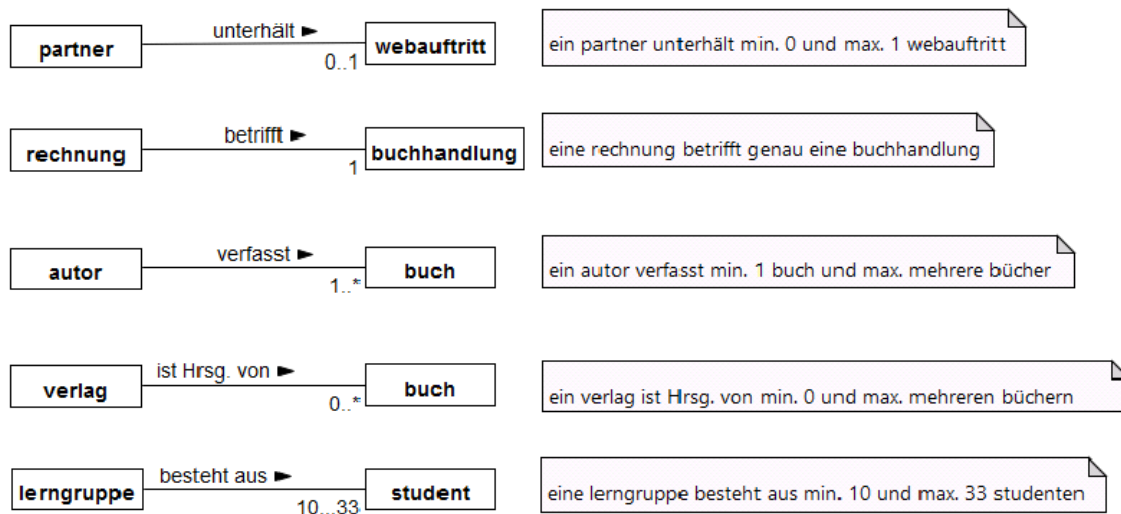


UML

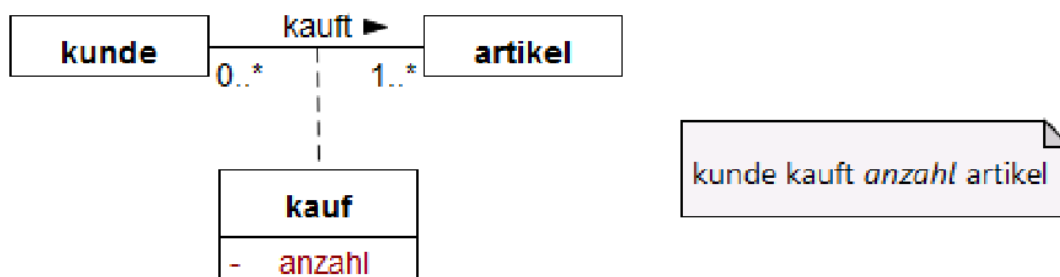
20 February 2015 11:49

In UML lautet der Entitätstyp Klasse und die Entität Objekt.

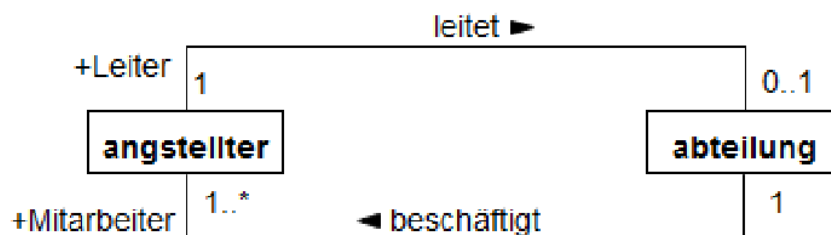
Neben den bereits bekannten Assoziationstypen kann UML konkrete natürliche Zahlen zur Präzisierung des Assoziationstyps.



Die Darstellung der Assoziationsklasse mit der Assoziation erfolgt über eine gestrichelte Linie.

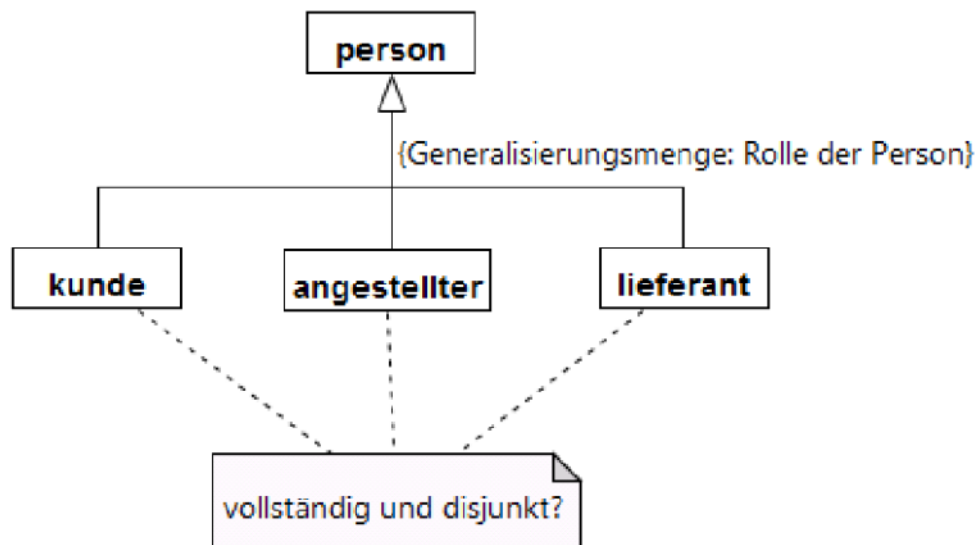


Beim Assoziationsende kann zusätzlich die Rolle der Klasse spezifiziert werden.



Generalisierung/ Spezialisierung

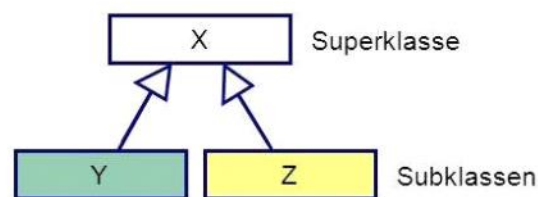
Die Generalisierung/ Spezialisierung ermöglicht die Darstellung von Beziehungen und Vererben der Klassen.



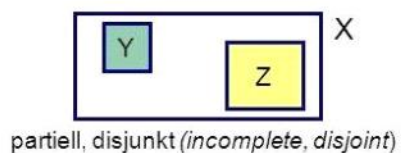
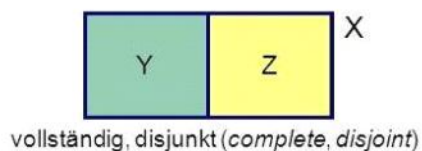
Der Subtyp und Supertyp wird mittels eine IS-A Beziehung verbunden.

Haben Subtypen keine gemeinsame Elemente, so ist das System disjunkt.

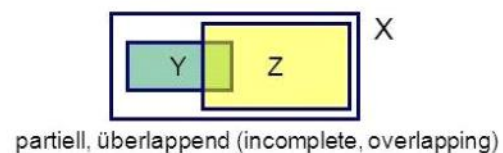
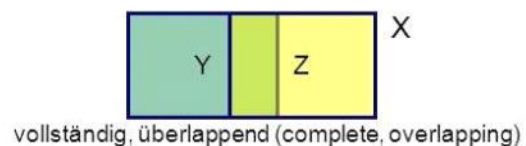
Hat der Supertyp keine eigene Elemente und diese folglich in den Subtypen enthalten sind, ist das System vollständig.



disjunkte Spezialisierungen (Partitionierung)

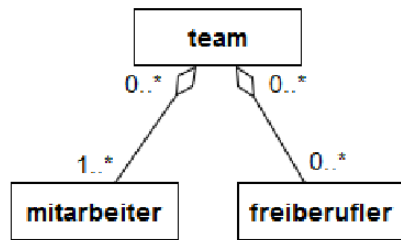


überlappende Spezialisierungen

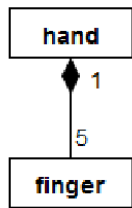


Aggregation/ Komposition

Über Aggregation lassen sich Teile-Ganzes-Beziehungen darstellen, sprich HAS und IS PART OF.



HAS und IS PART OF Beziehung



Es handelt sich um eine Komposition wenn Teile nicht ohne Elemente existieren.
Ein Finger existiert nicht ohne Hand.

ERD zu relationaler Datenbank

27 February 2015 11:16

Weitere Schritte für eine korrekte Modellierung des ERDs zur Verwendung in einer relationalen Datenbank

Attribute und Domänen

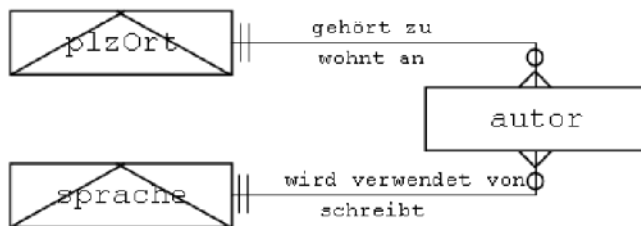
- Unter Domäne wird der Wertebereich eines Attributs verstanden.
- Bsp. Ampelfarben: Rot, Gelb, Grün
- Auch Primärschlüssel und Fremdschlüssel sind Attribute.
 - Primärschlüssel haben einen statischen Wertebereich
 - Fremdschlüssel einen dynamischen Wertebereich

Zusammengefasst

- Pro Tabelle hat Attribut eindeutigen Namen
- Attributwerte stammen aus einer Domäne
- Attributwerte sind atomar (Ein Wert pro Feld)
- Werte können NULL sein.

Arten von Entitätstypen

Attributiver Entitätstyp



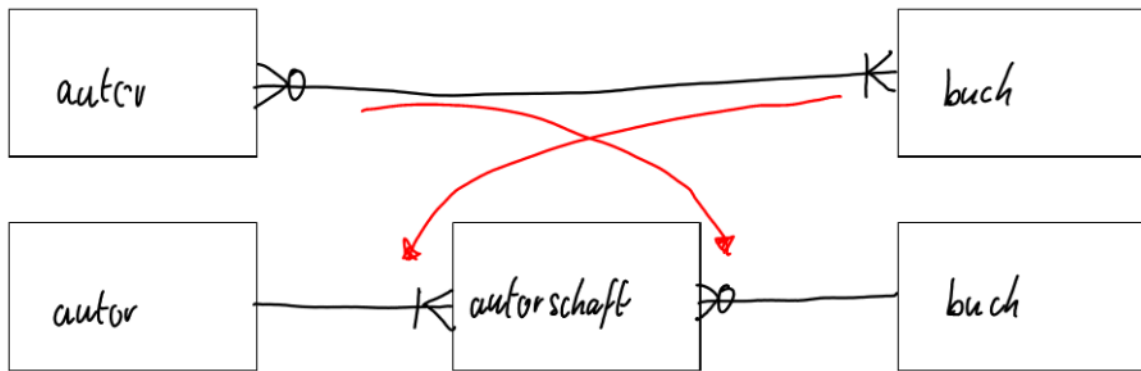
Das sind reine Attribut-"Spender".
Haben selber keine Fremdschlüssel eingetragen.

Fundamentaler Entitätstyp

Das sind reale Entitäten (Partner, Buch, Verlag, etc.)
Haben mehrere Fremdschlüsselattribute von attributiven Entitätstypen.

Assoziativer Entitätstyp

m:m Beziehungen sind zur Implementation zwingend aufzulösen.
Dabei wird ein künstlicher Entitätstyp benutzt.

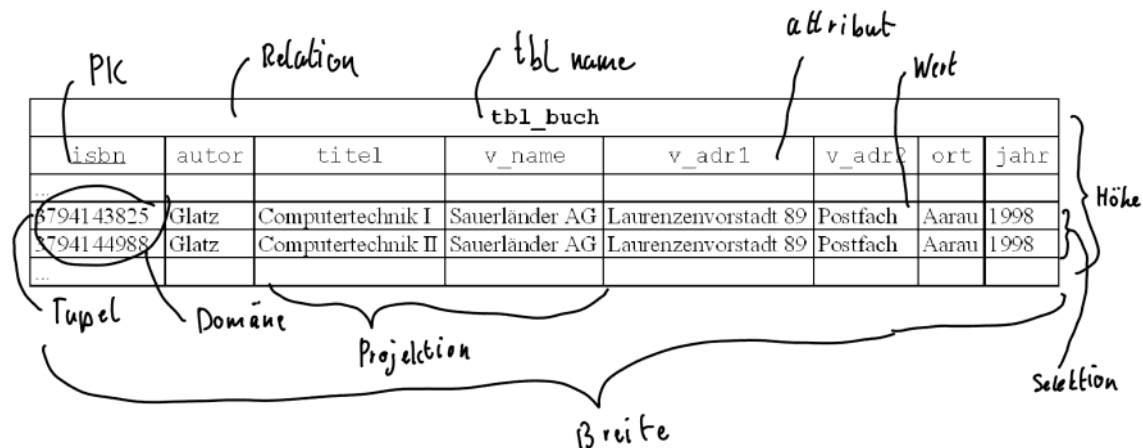


Relationsmodell

20 February 2015 12:36

Das Relationsmodell beinhaltet alle Entitäts- und Beziehungstypen in flachen Tabellen (Relationen). Diese Relationen werden nach strengen Regeln in Beziehung gesetzt.

Objekte, Begriffe, Spezifikationen



- Entitätstyp -> Tabelle
 - Name
 - kunde -> tbl_kunde
 - ist eindeutig
 - Entität -> Zeile (Tupel, Datensätze, Records)
 - Reihenfolge ist bedeutungslos
 - Attribut -> Spalte
 - Mindestens zwei
 - Name
 - ◆ Vorname -> vorname (natürlich)
 - ◆ Kunden-Nummer -> kunden_nummer (künstlich)
 - ◆ Eindeutig innerhalb der Tabelle
 - Primärschlüssel unterstreichen
 - Fremdschlüssel umkreisen
 - Koordinatenpunkt -> Feld
 - ◆ Enthalten einen Wert oder sind leer -> NULL
 - ◆ Reihenfolge bedeutungslos
 - ◆ Anordnung von links nach rechts mit abnehmender Relevant
 - Domäne -> Alle gültigen Werte eines Bereichs
- Relation -> Menge ungeordneter Tupel
 - waagrecht -> Grad, Breite (Degree)
 - Anzahl der attribute
 - senkrecht -> Höhe (Card)
 - Anzahl Tupel
 - Auswahl Attribute -> Projektion
 - Auswahl Tupel -> Selektion

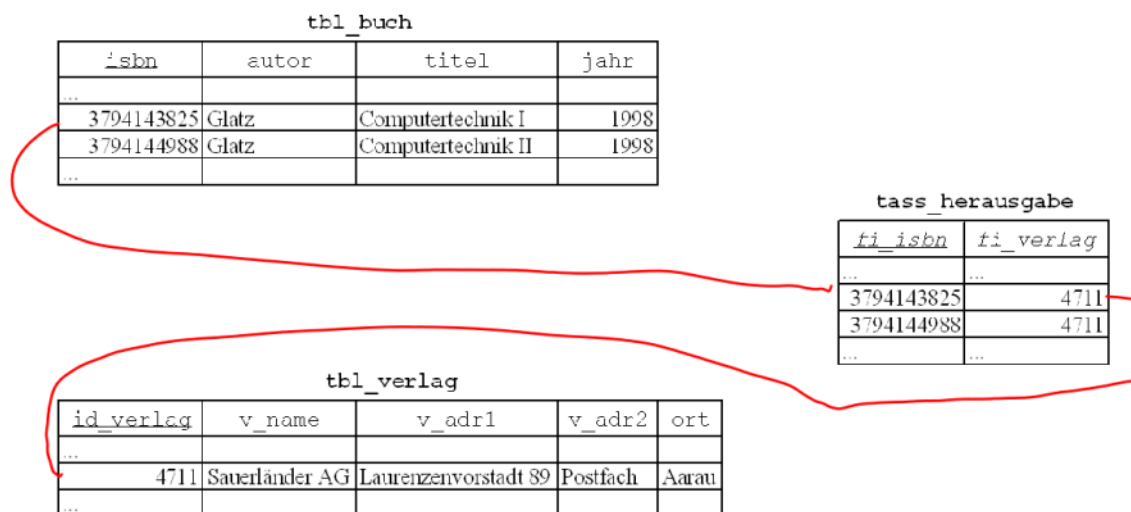
Begriffe im Überblick

<i>ERM</i>	<i>Relationenmodell</i>	<i>alternativ</i>
Entität	Tupel	Zeile, Datensatz, Record
Entitätstyp	Relation	Tabelle
Beziehung	Beziehung	
Beziehungstyp	Beziehungstyp	
Kardinalität	Kardinalität	wenig sinnvoll: Anzahl
Attribut	Attribut	Spalte; wenig sinnvoll: Feld

Primärschlüssel und Tupelintegrität

- Tupel darf nur einmal vorkommen -> mindestens 1 Attribut muss sich unterscheiden
- Tupel muss eindeutig identifizierbar sein -> Primärschlüssel (Primary Key)
 - Keine Mehrfachnennung möglich (UNIQUE)
 - Keine Leereinträge (NOT NULL)
 - Wird unterstrichen

Beziehungstypen



Wenn Gewissheit oder Möglichkeit zur Redundanz in einer Tabelle besteht, muss Tabelle aufgebrochen und in Beziehung gesetzt werden.

Folgendes Vorgehen ist empfohlen

1. Suche Redundanz verursachender Entitätstyp
2. Daraus neuen Entitätstyp erstellen mit PK
3. Beziehung erstellen mit dritten Entitätstyp
4. Beziehung als ERD erstellen

Fremdschlüssel

Für Fremdschlüsselwerte gilt:

- dürfen NULL sein
- können mehrfach vorkommen
- Gleiche Domäne wie Bezugsort
- Referenzielle Integrität
 - Wird bestimmt durch die Beziehung
 - Ist NULL erlaubt oder nicht
 - Wie mit falschen Werten umgehen
- Referenztyp
 - Handelt es sich um eine existenzielle Beziehung?
 - Welche Aktion wird beim Updaten oder Löschen ausgeführt?

Referenzielle Integrität

Referenz muss eindeutig zuweisbar sein. Bsp. FK verweist immer auf gültigen PK

Implementation SQL

Nun das vollständige Beispiel:

```
CREATE Database literaturliste
go

USE literaturliste
go

CREATE TABLE tkey_verlag
(id_verlag CHAR(6) NOT NULL PRIMARY
KEY,
v_name VARCHAR(25),
v_adr1 VARCHAR(40),
v_adr2 VARCHAR(40),
ort VARCHAR(25)
)

go

CREATE TABLE tbl_buch
(isbn CHAR(13) NOT NULL PRIMARY KEY,
autor VARCHAR(50),
titel VARCHAR(50),
jahr SMALLINT,
fi_verlag CHAR(6) NOT NULL
)

go

ALTER TABLE tbl_buch
ADD FOREIGN KEY (fi_verlag)
REFERENCES tkey_verlag(id_verlag)
ON UPDATE CASCADE
ON DELETE NO ACTION

go
```

Normalisierung

06 March 2015 11:13

- Ein Datenbankmodell ist dann konsistent, wenn es die realen Sachverhalte jederzeit wahrheitsgetreu und eindeutig widerspiegelt.
- Eine Nachricht ist dann redundant, wenn sie ohne Informationsverlust weggelassen werden kann.
- Die Normalformen gewährleisten die langfristige Redundanzfreiheit von Datenbanken.

Erste Normalform

Tabelle ist in erster Normalform wenn Wertebereiche der Attribute atomar sind.

<u>ISBN</u>	Autor	Titel	Typ
1234567890123	Willi	Wenn Nattern knattern	Sachbuch Lehrbuch
...

Zweite Normalform

Tabelle ist in zweiter Normalform wenn erste Normalform eingehalten wurde und jede Nichtschlüsselattribut vom zugehörigen Primärschlüssel voll funktional abhängig ist.

Funktionale Abhängigkeit

Wert aus Domäne A wird Wert aus Domäne B zugeordnet.
Dann ist B funktional abhängig von A.

Bsp.: ISBN -> Buchtitel, Kalenderdatum -> Wochentage

Primärschlüssel bestimmen für jedes Attribut genau einen Wert oder NULL.

$$B = f(A)$$

Voll Funktionale Abhängigkeit


Spezialfall bei zusammengesetzten Primärschlüssel. Attributswert wird durch zusammengesetzten Primärschlüssel bestimmt.

Bsp.: A1 Flugnummer, A2 Kalenderdatum, B Kapitän.

$$B = f(A1, A2, \dots)$$

Beispiel für Tabelle die nicht in zweiter Normalform ist.

<u>BstNr</u>	<u>ISBN</u>	Titel	Anzahl
KGB007	1234567890123	Wenn Nattern knattern	25
KGB007	2345678901234	Wenn Wanzen tanzen	10
...



⇒ $buch_bestell(bestnr, \textcircled{ISBN}, anzahl)$
 $buch(\textcircled{ISBN}, titel)$

Dritte Normalform

Tabelle ist in dritter Normalform wenn ersten zwei Normalform eingehalten wurden und kein Nichtschlüsselattribut von irgendeinem Schlüssel transitiv abhängig ist.

Transitive Abhängigkeit

Der Wert eines Nichtschlüsselattributs bestimmt einen anderen Attributswert.

$$\underline{B = f(A) \text{ und } C = f(B)}$$

Beispiel für Tabelle die nicht in dritter Normalform ist.

ISBN	VerlId	Titel	Verlag
1234567890123	De455	Wenn Nattern knattern	Kriecher
...

Dekomponiert ergibt sich:

buch (ISBN , VerlId , Titel)
 Verlag (VerlId , Verlag)

Praxisbeispiel

id_pers	name	abt#	abt_budget	kurs#	kurs_name	datum
505	Müller	15	72'000	X17	Verkauf	03.01.2005
425	Meier	15	72'000	X17	Verkauf	05.10.2005
425	Meier	15	72'000	L91	Rhetorik	15.09.2005
425	Meier	15	72'000	L98	Führung	11.11.2005
458	Schulze	18	33'500	L91	Rhetorik	15.09.2005
703	Schmid	26	61'900	X17	Verkauf	12.12.2005
703	Schmid	26	61'900	L31	Oracle	10.02.2005

1NF: Eingehalten, jedes Attribut ist atomar.

2NF: Es gibt kein Primärschlüssel, Schlüsselkandidat id_pers reicht nicht allein -> erst die Kombination id_pers und kurs# ist eindeutig.

kurs(id_pos, name, abt#, abt_budget, kurs#, kursname, datum)



3NF: Berücksichtigung transitive Abhängigkeiten

person(id_persn, name, abt#, abt_budget)
 kurs(kurs#, kursname)
 kurs_pers(id_pers, kurs#, datum)

Lösung:

person(id_persn, name, abt#)
 kurs(kurs#, kursname)
 kurs_pers(id_pers, kurs#, datum)
 abt(abt# , abt_budget)

Access Datentypen

12 April 2015 14:22

Numerische Typen: Zahlen

- Byte: $0 \leq \text{Ganzzahl} \leq 255$
(1 Byte pro Zahl)
- Integer: $-32'768 \leq \text{Ganzzahl} \leq 32'767$
(16 Bits = 2 Bytes pro Zahl)
- Long Integer: $-2'147'483'648 \leq \text{Ganzzahl} \leq 2'147'483'647$
(32 Bits = 4 Bytes pro Zahl)
- Single: pos./neg. Fließkommazahl mit einfacher Genauigkeit
(4 Bytes pro Zahl)
- Double: pos./neg. Fließkommazahl mit doppelter Genauigkeit
(8 Bytes pro Zahl).

Autowerte sind Long Integers!

Mit numerischen Werten kann das System rechnen. Eine Postleitzahl 6048 kann numerisch sein, muss aber nicht. Die Entscheidung liegt beim DB-Definierer.

Datum/Uhrzeit akzeptiert nur Werte, die in die Domäne passen und formatiert sie gemäss den Landeseinstellungen von Windows.

Währung akzeptiert nur Ganz- oder Fließkommazahlen und formatiert sie gemäss den Landeseinstellungen von Windows.

Ein Autowert ist ein Long Integer, der bei 0 beginnt. Mit jedem neuen Tupel vergibt das System eine um 1 grössere natürliche Zahl. Einmal vergebene Zahlen kommen nicht wieder, auch wenn das mit ihm identifizierte Tupel gelöscht wurde. Autowerte werden oft für Primärschlüsselattribute verwendet.

Ja/Nein ist ein so genannter Wahrheitswert, der nur die Eintragung "ja" und "nein" erlaubt. Beispiel: Attribut "Ehrenmitglied" in einer Vereinsverwaltung.

Access Queries

17 March 2015 09:32

GROUP BY

Gruppirt Tupel anhand bestimmter Attribut Werte.

Beispiel:

```
rechnung(id, name, betrag){  
  1, Meier, 300  
  2, Müller, 200  
  3, Meier, 250  
  4, Kübliz, 200  
}
```

Nach einem GROUP BY name

```
{  
  Meier, id:{1, 3}, betrag:{300, 250}  
  Müller, 2, 200  
  Kübliz, 4, 200  
}
```

Nach einem SUM(betrag)

```
{  
  Meier, id:{1, 3}, 550  
  Müller, 2, 200  
  Kübliz, 4, 200  
}
```

JOIN

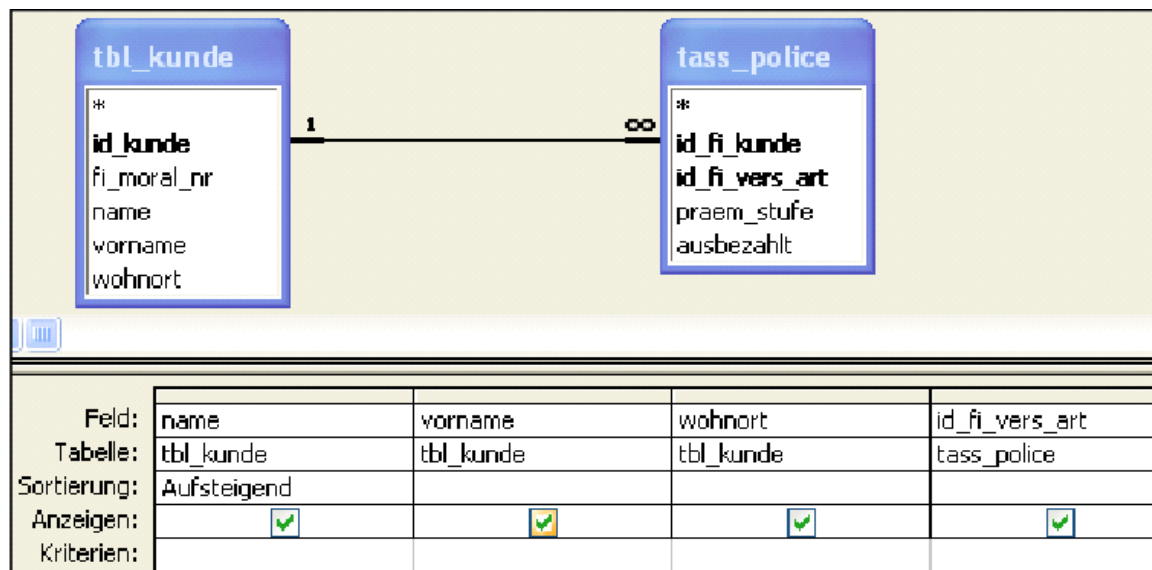
Über Join werden Tabellen verbunden.

Es gibt verschiedene Joins:

- Cross Join -> kartesisches Produkt Bild
- Inner Join -> Filtert PK=FK
- Outer Join -> äussere Zeilen hinzufügen

Inner Join

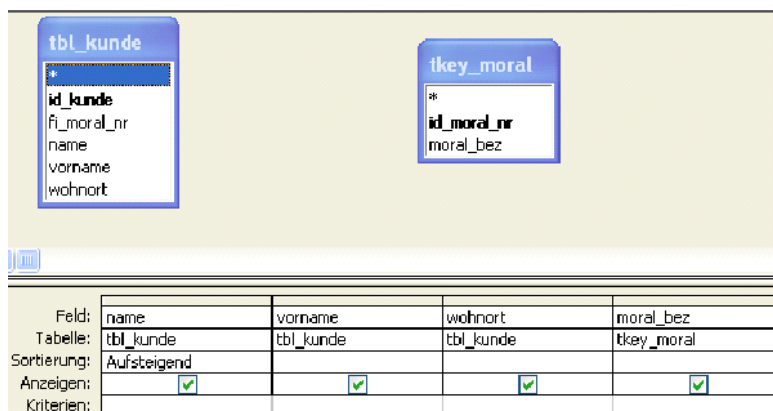
Beim Inner Join werden zwei Tabellen verbunden. Im Beispiel unten sieht man für jede Versicherung den dazugehörigen Kundenndatensatz.



	Kundenname	Kundenvorname	Wohnort	Versicherungscode
▶	Hauser	Martin	Kriens	1700
	Hauser	Martin	Kriens	1800
	Hauser	Martin	Kriens	1500
	Hauser	Martin	Kriens	1300
	Hauser	Martin	Kriens	1200
	Klausen	Heidi	Zürich	1500
	Klausen	Heidi	Zürich	1600
	Klausen	Heidi	Zürich	1200
	Meier	Max	Luzern	1000
	Meier	Max	Luzern	1300
	Menzer	Claudia	Horw	1300
	Menzer	Claudia	Horw	1500
	Menzer	Claudia	Horw	1700
	Menzer	Claudia	Horw	1900
	Menzer	Claudia	Horw	1800
	Müller	Eva	Bern	1000
	Müller	Eva	Bern	1400
	Müller	Eva	Bern	1300
	Müller	Eva	Bern	1200
*				

Cross Join

Beziehungen werden nicht berücksichtigt. Das Ergebnis eines Cross Joins ist ein kartesisches Produkt.

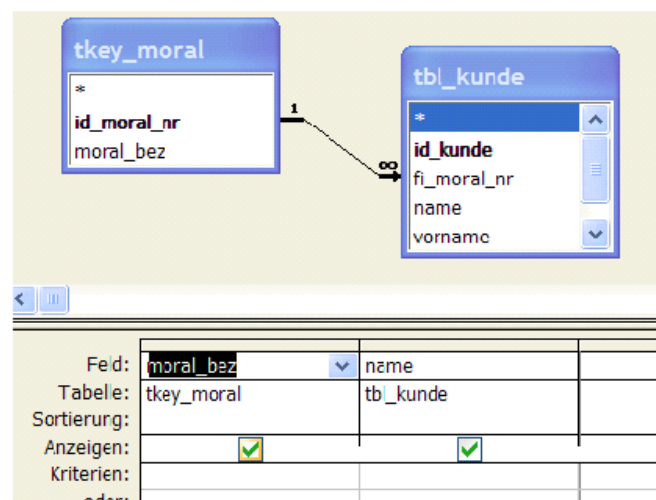


	Kundenname	Kundenvorname	Wohnort	Zahlungsmoral
►	Düsentrieb	Daniel	Entenhausen	gut
	Düsentrieb	Daniel	Entenhausen	schlecht
	Düsentrieb	Daniel	Entenhausen	sehr gut
	Hauser	Martin	Kriens	sehr gut
	Hauser	Martin	Kriens	gut
	Hauser	Martin	Kriens	schlecht
	Hilton		Malibu	gut
	Hilton		Malibu	sehr gut
	Hilton		Malibu	schlecht
	Klausen	Heidi	Zürich	sehr gut
	Klausen	Heidi	Zürich	gut
	Klausen	Heidi	Zürich	schlecht

Outer Join

Bestimmt in welche Richtung die Abfrage aufgelöst werden sollen. Dazu zwei Beispiele, einmal Left-Join und einmal Right-Join.

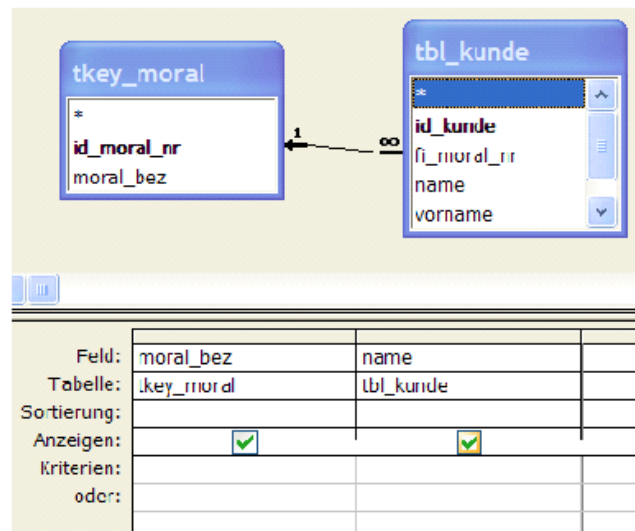
```
SELECT tkey_moral.moral_bez, tbl_kunde.name FROM tkey_moral
LEFT JOIN tbl_kunde ON tkey_moral.id_moral_nr =
tbl_kunde.fi_moral_nr;
```



Als Resultat erhalten wir folgendes:

Zahlungsmoral	Kundenname
schlecht	Hilton
schlecht	Hauser
schlecht	Düsentrieb
gut	Meier
gut	Klausen
sehr gut	Müller
sehr gut	Menzer

```
SELECT tkey_moral.moral_bez, tbl_kunde.name FROM tkey_moral
RIGHT JOIN tbl_kunde ON tkey_moral.id_moral_nr =
tbl_kunde.fi_moral_nr;
```



Als Resultat erhalten wir folgendes:

Zahlungsmoral	Kundenname
	Pellegrini
schlecht	Hilton
schlecht	Hauser
schlecht	Düsentrieb
gut	Meier
gut	Klausen
sehr gut	Müller
sehr gut	Menzer

Indizes

12 April 2015 14:23

- Tupel werden persistent an einer Stelle in der Datenbank gespeichert.
- Das DBMS nimmt keine selbständige Sortierung vor.
- Indizes beschleunigen Suchanfragen.
- Bestimmte Attribut werden indexiert, das DBMS erstellt eine Kopie und sortiert diese bei jeder Mutation, Erfassung oder Löschung.

Es gibt verschiedene Typen von Indizes:

DBMS	Analogie Buchsuche
(Full) Table Scan	Buch durchblättern
Nonclustered Index	Im Index Stichwort nachschlagen: Verweis auf Seitenzahlen
Clustered Index	Lexikon / Wörterbuch: Einträge liegen in sortierter Reihenfolge vor

Beispiel - Indexierter Nachname:

...	
Abegg	4807
Bieri	912
Eigensatz	3333
Grüniger	9005
Keller	5107
Luthiger	118
Meyer	2463
Ritschard	4716
Röösli	3456
Schibli	4721
Schick	967
Schmid-Toniolo	4718
Schmucki	2467
Senn	4719
Stalder	1367
Tuor	4726
Wuillemin	4715
...	

Der Name eines Tupel wurde indexiert und somit in eine Index-Tabelle kopiert und sortiert.
Der Zugriff erfolgt direkt über den Schlüssel.

4721	Herr	Schibli	Roland	Dr. rer. pu
4722	Herr	Meyer	Markus	Dr. iur./lic
4723	Herr	Keiser	Markus	
4724	Frau	Bühler	Elisabeth	Prof. Dr. h
4725	Herr	Oegerli	Richard	Betriebs
4726	Herr	Tuor	Franz	Marketin
4727	Herr	Mürner	Alois	lic. iur.
4728	...			

Der Index kann auf Anweisung Duplikate enthalten (Primärschlüssel natürlich nicht).

Vor- und Nachteile

- + Beschleunigung bei Suche
- + Beschleunigung bei Indexierung
- Zusatzaufwand bei Tupelmutationen
- Mehr Speicherung durch Indextabelle

Massname

- Index sorgfältig auswählen
- Such- oder Sortierkriterien bestimmte Attribute auswählen
- Attribut mit hoher Selektivität auswählen

Indexorganisation

Der Speicherort eines Datensatzes wird beim SQL-Server mit einer dreiteiligen Adresse, nämlich [File:page:Slot] vorgenommen:

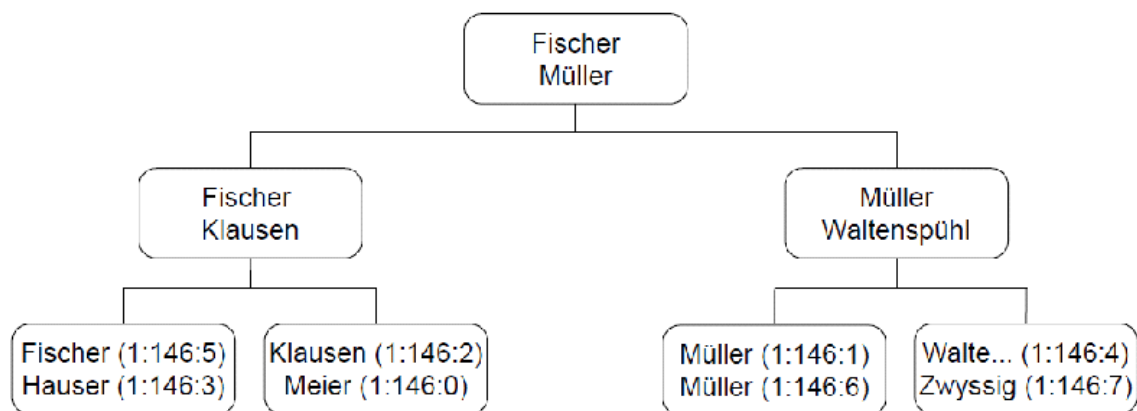
- *File*: SQL-Server-interne ID der Datendatei, in der sich der Datensatz befindet.
- *Page*: Seite der Datendatei, auf der sich der Datensatz befindet.
- *Slot*: Position innerhalb der Seite, die der Datensatz einnimmt.

Indextypen

Nonclustered Index

Anname: Es wird auf die Spalte `name` ein Index angelgt.

Es wird eine Tabelle geführt die den Index in einer Baustuktur führt.



Clustered Index

Es werden alle Datensätze gemäss Index sortiert auf den Datenträger gespeichert.

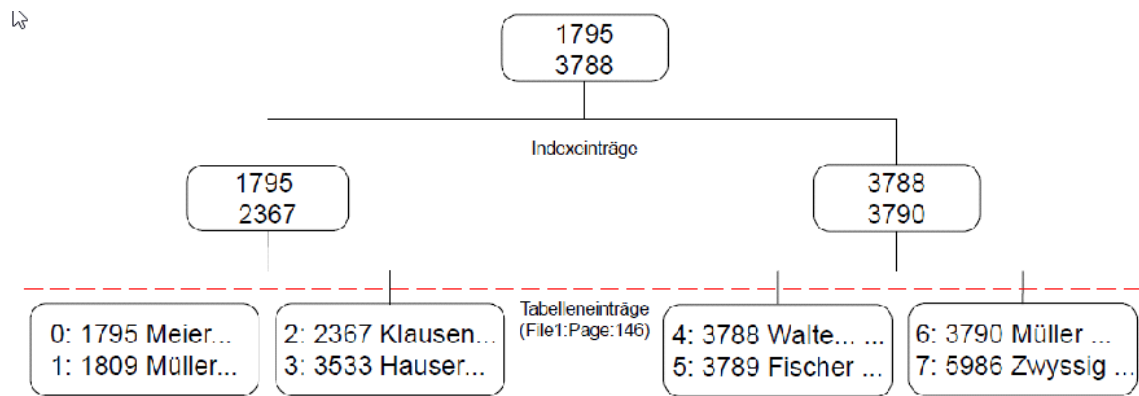


Abbildung 11: Struktur eines gruppierten Indexes (Blatt entspricht Tabellenseite mit Tupeleinträgen)

Der Clustered Index sortiert jeweils alle Daten!

Integritätsbedingungen

17 April 2015 11:53

Als solche *Integrity Constraints* kennt ANSI

- **NOT NULL**, welches die **Eingabe von Null-Marken** (Wertigkeit: *unknown*) **verhindert**
- **UNIQUE**, mit welchem die **Einmaligkeit eines Wertes in einer Tabellenspalte** erzwungen wird (und damit ein Schlüsselkandidat entsteht)
- **CHECK**, welches bei der Eingabe die Zugehörigkeit eines **Wertes zu einer Domäne** - einer zugelassenen Wertemenge - erzwingt
- **DEFAULT**, welches bei der Eingabe eines Tupels einen nicht **explizit erfassten Spaltenwert implizit** befüllt (Vorgabewert); Vorgabewerte werden auch etwa *Standards* genannt
- **PRIMARY KEY**, mit welchem die Einmaligkeit eines Tupels, also die **Tupelintegrität**, erzwungen wird
- **FOREIGN KEY**, mit welchem das tatsächliche **Vorhandensein eines Wertes am Bezugsort**, also die referenzielle Integrität, erzwungen wird
- **CREATE / ALTER / DROP DOMAIN**: Domänen können als **benutzerspezifische Datentypen** verstanden werden:
 - Eine Domäne muss einem Basistypen aufliegen.
 - Eine Domäne ist ein Objekt und trägt deshalb einen Namen: `kuNummer`.
 - Die Domäne wird danach verwendet wie ein Basistyp: `... spltA kuNummer NOT NULL ...`
- **CREATE / DROP ASSERTION**, welches eine **Zusicherung formuliert**, die dann zu wahr ausgewertet werden muss: *"Überzeuge dich davon, dass ..."*

Ein Vergleich mit NULL ist immer FALSE!

Datenmanipulation Access

24 March 2015 08:57

Für die Manipulation von Datensätzen stehen verschiedene SQL Befehle zur Verfügung.

Insert

Neuen Datensatz einfügen.

INSERT Into <tabelle> (<feldname1>, <feldname2>, ...) VALUES (<Wert1>, <Wert2>, ...)

Update

Vorhandener Datensatz aktualisieren.

Update <tabelle> SET <feldname1>=<neuer Wert> WHERE <Schlüssel>=<Schlüsselwert>

Delete

Vorhandener Datensatz löschen

DELETE FROM <tabelle> WHERE <Schlüssel>=<Schlüsselwert>

Einleitung RDBMS

12 April 2015 12:53

- pack alle Entitätstypen in Tabellen
- sichert die Tupelintegrität jeder Tabelle mithilfe eines Primärschlüssels ab
- Drückt Beziehung durch Primär- und Fremdschlüssel aus.
- Sicher referenzielle Integrität durch Löschen- und Mutationaktionen ab.
- Legt alle Definitionen als Metadaten in einer Systemtabelle ab.

Erweiterte RDBMS

- Authentisierung und Autorisierung von Benutzerenden
- Sicherung Anomalien durch gleichzeitige Zugriffe
- Verhinderung Phantomeffekte

Datenbank

- Systemweit eindeutigen Namen
- Besteht aus zwei oder mehreren Tabellen
- Besteht aus Metadaten

Tabellen

- Datenbankweit eindeutigen Namen
- Hat zwei oder mehr Attribute
- Hat einen Primärschlüssel, bestehend aus einem oder mehreren Attributen
- Hat eine relativ statische Breite (Anzahl Attribute)
- Hat eine relativ dynamische Höhe (Anzahl Tupel)

Attribut

- Tabellenweit eindeutigen Namen
- Werte bestimmt durch Domäne (Datentyp)
- Hat bestimmte Feldgröße (bestimmt durch Datentyp)

SQL Server

07 April 2015 09:05

Systemdatenbanken

master

Enthält alle Metainformationen des DBMS

model

Ist die Vorlage für neu entstehende Datenbanken

msdb

Zur Speicherung von automatisierten Aufgaben.

- Datensicherung
- Replikation
- Warnmails

Arbeitet eng mit SQL Server Agent zusammen.

tempdb

Enthält temporäre Daten.

Wird bei einem Neustart gelöscht.

Grundregeln Relationale Datenbanken

Regel 0: Eine RDBMS repräsentiert alle seine Konfigurationen in seiner Sprache.

Regel 1: Informationsregel

Jede Information einer relationalen Datenbank wird in genau einer Weise durch Werte in Relationen dargestellt und damit redundanzfrei und einheitlich verwaltet. Dazu gehören die Anwendungsdaten und die Metadaten, also Daten über den Aufbau der Datenbank.

Regel 2: Garantierter Zugriff

Jedes einzelne Feld einer Datenbank ist durch eine Kombination von Relationsname, Primärschlüssel und Spaltenname erreichbar.

Regel 3: Systematische Behandlung fehlender Information

In einer relationalen Datenbank müssen Spalten mit fehlender Information (Nullwerte) einheitlich darstellbar sein. Diese Nullwerte werden systematisch als fehlende Informationen von den Standardwerten (z. B. Strings mit Leerzeichen) unterschieden. Spalten können auch so eingerichtet werden, dass Nullwerte nicht erlaubt sind.

Regel 4: Dynamischer Online-Katalog (Data Dictionary)

Ein Datenbankschema wird in derselben Weise wie die gespeicherten Daten selbst beschrieben – nämlich in Tabellen, dem sogenannten Data Dictionary. Autorisierte Benutzer können das Data Dictionary in gleicher Weise abfragen wie die eigentliche Datenbasis.

Regel 5: Allumfassende Sprache

Ein (relationales) Datenbanksystem muss eine Sprache unterstützen, die allumfassend im folgenden Sinne ist. Diese Sprache erfüllt die folgenden Aufgaben:

- Definition der Benutzerdaten
- Definition von Sichten als virtuelle Tabellen
- Manipulation von Benutzerdaten
- Überprüfung von Integritätsregeln
- Vergabe von Benutzerrechten und Autorisierung
- Transaktionskontrolle und Transaktionshandling (eine Transaktion ist eine Folge von Datenänderungen, die immer ganz oder gar nicht durchgeführt werden muss)

Regel 6: Benutzersichten und Datenänderungen

Für unterschiedliche Benutzergruppen und Anwendungen sind unterschiedliche Sichten (Views) auf die Datenbank notwendig. In einfachen Views, z. B. Teilansichten einer Tabelle, sollen auch Datenänderungen möglich sein.

Regel 7: HIGH-LEVEL INSERT, UPDATE, und DELETE

In einer Datenbank muss das Einfügen, Ändern und Löschen von Daten möglich sein. Dabei soll das System sich den optimalen Zugriffspfad zur schnellen Durchführung der Transaktion selbst suchen.

Regel 7: HIGH-LEVEL INSERT, UPDATE, und DELETE

In einer Datenbank muss das **Einfügen, Ändern und Löschen von Daten** möglich sein. Dabei soll das System sich den optimalen Zugriffspfad zur schnellen Durchführung der Transaktion selbst suchen.

Regel 8: Physische Datenunabhängigkeit

Anwendungsprogramme und Anwenderoberflächen bleiben unverändert, wenn Veränderungen an der Speicherstruktur oder der Zugriffsmethode in der Datenbank vorgenommen werden.

Regel 9: Logische Datenunabhängigkeit

Anwendungsprogramme und Anwenderoberflächen bleiben unverändert, wenn sich Basisrelationen verändern, die nicht direkt die Anwendungsprogramme betreffen. Es können z. B. Spalten in Relationen ergänzt oder neue Relationen hinzugefügt werden.

Regel 10: Integritätsunabhängigkeit

Integritätsbedingungen, die von der Datenbank erfüllt werden müssen, werden mithilfe der **relationalen Datenbanksprache definiert**, im Data Dictionary abgelegt und vom DBMS ausgeführt. Diese Integritätsbedingungen gehören nicht ins Anwenderprogramm.

Regel 11: Verteilungsunabhängigkeit

Eine relationale Datenbank besitzt die **Verteilungsunabhängigkeit**. Das heisst, die **Anwendungsprogramme bleiben unverändert**, wenn die **verteilte Datenhaltung auf mehreren Rechnern eingeführt** oder wieder wieder zurückgenommen wird und mehrere Datenbanken zu einer Datenbank zusammengelegt werden.

Regel 12: Unterwanderungsverbot

Falls das DBMS eine andere **3GL-Sprache wie C oder Java zulässt**, darf diese Sprache nicht die **aufgestellten Regeln 1 bis 11 verletzen** oder ausser Kraft setzen.

Begriffe

Eine Structured Query Language besteht aus 4 Komponenten:

DCL (Data Control Language): Definition von Zugriffsberechtigungen und Speicherstrukturen (z. B. CREATE LOGIN...)

DDL (Data Definition Language): Definition von Datenbankobjekten (z. B. ALTER TABLE...)

DML (Data Manipulation Language): Anlegen, Ändern oder Löschen von Daten (z. B. DELETE FROM...)

DQL (Data Query Language): Abfragen von Daten (z. B. SELECT * FROM...)

ACID

SQL muss ferner eine konsistente Datenhaltung gewährleisten:

A – Atomicity: Transaktionen sind atomar, also als unteilbare Einheiten zu betrachten, Transaktionen werden ganz oder gar nicht ausgeführt.

C – Consistency: Eine Transaktion überführt eine Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand. Ein Datenzustand heisst konsistent, wenn alle Daten semantisch richtig, also im Anwendungskontext korrekt sind.

I – Isolation: Transaktionen laufen isoliert ab. Obwohl im Mehrbenutzerbetrieb gleichzeitig mehrere Transaktionen abgearbeitet werden, läuft jede einzelne Transaktion wie in einem simulierten Einbenutzerbetrieb ab.

D – Durability: Die Ergebnisse einer Transaktion werden dauerhaft (persistent) in der Datenbank gespeichert.

Transact SQL

07 April 2015 09:49

Beschreibt die Konfiguration einer Datenbank und deren Schema.

Beispiel - Quick'n Dirty:

```
drop table poops;

create table poops(
    id int primary key identity,
    bzch varchar not null
);

insert poops (bzch) values ('keep rollin'),('keep rockin');

select * from poops;
```

Zugriff, User, Rollen, Berechtigungen

21 April 2015 08:21

Der Zugriffsschutz ist ein essentieller und wichtiger Bestandteil von jedem DBMS.

Informationssicherheit

Zum Schutz der Daten wurden verschiedene Ziele definiert. Die Verletzung eines Schutzziels sagt aus, auf welche Art und Weise die Daten abgefangen, manipuliert oder blockiert worden sind.

Authentizität

Unter der Authentizität eines Objekts bzw. Subjekts (engl. authenticity) wird die Echtheit und Glaubwürdigkeit des Objekts bzw. Subjekts verstanden, die anhand einer eindeutigen Identität und charakteristischen Eigenschaften überprüfbar sind.

Integrität

Ein System gewährleistet die Datenintegrität (engl. integrity), wenn es Subjekten nicht möglich ist, die zu schützenden Daten unautorisiert und unbemerkt zu manipulieren.

Vertraulichkeit

Ein System gewährleistet die Informationsvertraulichkeit (engl. confidentiality), wenn es keine unautorisierte Informationsgewinnung ermöglicht. Die Gewährleistung der Eigenschaft der Informationsvertraulichkeit erfordert in datensicheren Systemen die Festlegung von Berechtigungen und Kontrollen der Art, dass sichergestellt ist, dass Subjekte nicht unautorisiert Kenntnis von Informationen erlangen.

Verfügbarkeit

Ein System gewährleistet die Verfügbarkeit (engl. availability), wenn authentifizierte und autorisierte Subjekte in der Wahrnehmung ihrer Berechtigungen nicht unautorisiert beeinträchtigt werden können. Verzögerungen, die aus „normalen“ Verwaltungsmaßnahmen resultieren (u.a. Prozess-Scheduling), werden als autorisierte Beeinträchtigungen betrachtet, was noch keine Verletzung der Verfügbarkeit darstellt.

Verbindlichkeit

Ein System gewährleistet die Verbindlichkeit bzw. Zuordenbarkeit (engl. non repudiation) einer Menge von Aktionen, wenn es nicht möglich ist, dass ein Subjekt im Nachhinein die Durchführung einer solchen Aktion abstreiten kann. Verbindlichkeitseigenschaften sind besonders im Bereich des elektronischen Handels (engl. Electronic Commerce) und der elektronischen Geschäfte (engl. Elec-tronic Business) von großer Bedeutung, um die

Rechtsverbindlichkeit durchgeführter geschäftlicher Transaktionen (Käufe, Verträge etc.) zu garantieren.

Access Control

Modelle der Zugriffssteuerung:⁹

- **DAC Discretionary Access Control**; Autorisierung und Zugriffssteuerung auf der Basis der Zuordnung von Rechten an Subjekte durch den Eigentümer von Objekten; siehe z. B. GRANT; flexibel aber wenig sicher; Alternativen: MAC und RBAC
- **MAC Mandatory Access Control**; Autorisierung und Zugriffssteuerung auf der Basis des Vergleichs von Sicherheitsklassen bei Objekten (Classifications) mit den Freiheitsgraden bei den Subjekten (Clearances); hochsicher; Alternativen: DAC und RBAC
- **RBAC Role Based Access Control**; Autorisierung und Zugriffssteuerung ausschliesslich aufgrund der Zugehörigkeit zu betrieblichen Gruppen mit funktionalen Rollen; bei der Zugehörigkeit zu mehreren Gruppen (Rollen) gilt die Vereinigungsmenge der Rechte; ...

Spezielle Benutzer und Gruppen

Benutzerin sa

sa ist ein vordefinierter Datenbankbenutzer, der **nicht gelöscht**, wohl aber **aktiviert** und **deaktiviert** werden kann. Er sollte nicht verwendet werden, da mindestens sein Login schon mal bekannt ist.

Benutzer guest

guest ist ein *vordefinierter* Datenbankbenutzer, der **nicht gelöscht**, wohl aber **aktiviert** und **deaktiviert** werden kann. **Guest kann sich nicht am Server anmelden!** Wollen wir jeder Person, welche am SQL Server autorisiert ist, eine Verbindung zur Datenbank gestatten, dann tun wir dies, indem **guest in unsere Benutzerliste** aufnehmen:

- Danach kann sich jede am SQL Sever autorisierte Person mit unserer Datenbank verbinden!
- Diese Möglichkeit ist vorgabemässig nicht vorgesehen.
- Sie ist Mitglied der Gruppe **public** und darf alles, was die Gruppe darf.
- Auf **individueller Ebene** können **guest noch weitere Berechtigungen** geöffnet werden.
- In den Datenbanken master und tempdb ist guest immer eine aktive Benutzerin.

Gruppe public

public ist eine vordefinierte Gruppe, die nicht gelöscht werden kann. Jeder Datenbankbenutzer - je nachdem also auch guest - ist zwingend mindestens Mitglied von public. Mitglieder von public können vorgabemässig

- **Systemtabellen und Metadaten lesen**
- **gewisse Speicherprozeduren** verwenden
- gewisse nicht Datenbasis-bezogene Anweisungen verwenden (z.B. PRINT)

public ist also eine bequeme Art, **minimale, pauschal Rechte für "normale" Benutzende** zu definieren.

Optimierung

05 May 2015 08:38

Hauptaufgabe einer Datenbankmaschine ist die Selektierung, d.h. Abfragen und Manipulationen wie SELECT, Update oder Delete.

Die benötigten Operationen für solche Aktionen sind:

Scanning: Syntaxprüfung

Parsing: Zerlegen in semantische Tokens, erstellen eines Strukturbaums

Existenzprüfung der verlangten Ressourcen (Tabelle, Attribute)

logische Optimierung: Bewertung verschiedener semantisch äquivalenter Ausführungspläne für die Selektionen, Projektionen und Kreuzprodukte der Abfrage

physische Optimierung: Zuzug von Serverressourcen wie Indizes, Hashes und Statistiken auf der Ebene Datenbankmaschine; Ermittlung der Kosten; Entscheidung für einen der Pläne

Compilierung: Übersetzung in Maschinenbefehle

systemtechnische Optimierung: Cachings, Bursts, Prefetchings, spekulative Bearbeitungen, Prozess-Priorisierungen usw. auf Ebene Betriebssystem oder Hardware.

Ausführungspläne

Dabei handelt sich um repetitive Aufgaben, die die Struktur des DBMS optimieren.

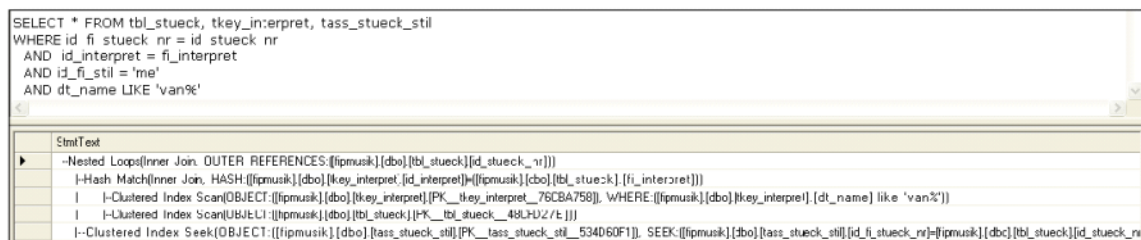


Abbildung 22: Ausführungsplan I textlich

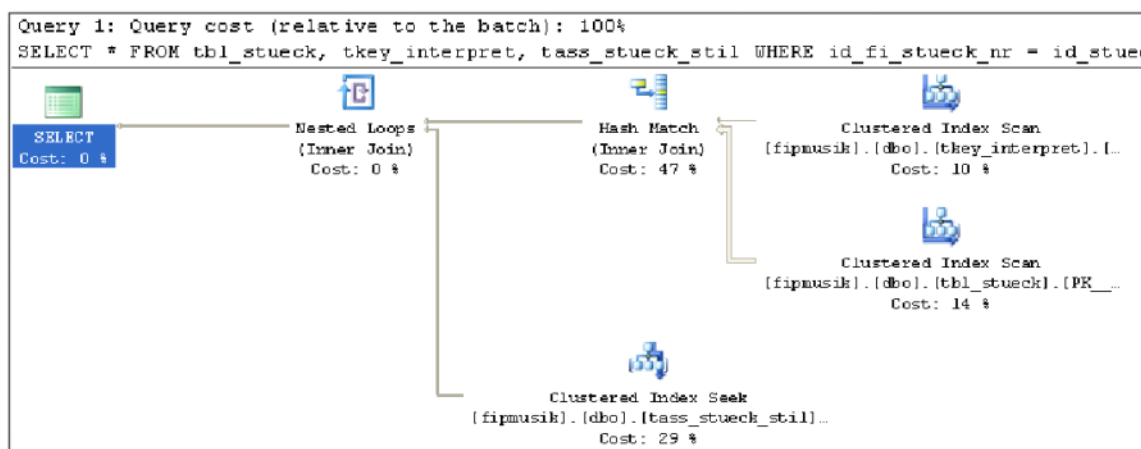


Abbildung 23: Ausführungsplan I grafisch

Anwender

Aus Anwender-Sicht lassen sich Selektionen ebenfalls optimieren, indem er ...

Abfragen schon optimiert eingegeben nach dem Primat der frühen Selektion

Indizes aktivieren, deaktivieren, manuell erneuern

Statistiken aktivieren, deaktivieren, manuell erneuern

als Abfragebestandteil dem Optimierer Anweisungen mitgeben,

- welche Indizes (nicht) zu verwenden sind
- in welcher algorithmischen Weise intern die Joins zu bewerkstelligen sind - die Theorie kennt drei algorithmische Join-Verfahren in Abhängigkeit von Tabellenhöhe, Indizes, Sortierungen usw.

Transaktionen

05 May 2015 08:42

Zuerst ein Beispiel von Anforderungen, die an Transaktionen gestellt werden.

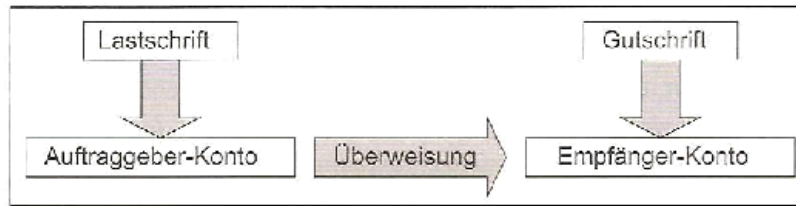


Abbildung 26: Aufteilung einer Finanztransaktion in zwei Komponenten¹⁰

Atomicity: «**Alles oder Nichts**» bedeutet in diesem Fall

1. sowohl Lastschrift als auch Gutschrift durchgeführt oder (XOR)
2. weder Lastschrift noch Gutschrift durchgeführt

Consistency: die Kontostände des Auftraggebers und des Empfängers **entsprechen** sowohl vor der Transaktion als auch nach der Transaktion den definierten **Integritäts- und Geschäftsregeln**.

Isolation: die Finanztransaktion läuft in «**geschütztem Rahmen**» ab; **keine andere Transaktion** kann während der Finanztransaktion auf die Kontostände des Auftraggebers und des Empfängers **zugreifen**.

Durability: Falls während des Betriebs ein **Systemfehler auftritt und das System «abstürzt»** wird die Finanztransaktion auf den im **Transaktionsprotokoll zuletzt bestätigten Prüfpunkt** gesetzt.

Die Transaktionsverwaltung löst Probleme wie:

- Mehrbenutzerzugriff
- Mehrbenutzeranomalien
- Phantomeffekte
- Sperrungen
- Verklemmungen
- usw.

«Unter einer **Transaktion** versteht man die '**Bündelung**' **mehrerer Datenbankoperationen**, die in einem **Mehrbenutzersystem ohne unerwünschte Einflüsse** durch andere Transaktionen als Einheit fehlerfrei ausgeführt werden sollen.» (Kemper; Eickler, S. 295)

Transaktion Modus

Der Transaktions Modus bestimmt wie und wann Transaktionen durchgeführt werden.

Mod	Einstellung SQL Server	T-Start	T-Ende
1	Standardeinstellung	automatisch	automatisch
2	–	BEGIN TRAN	COMMIT / ROLLBACK
3	SET IMPLICIT_TRANSACTIONS ON ¹²	implizit	COMMIT / ROLLBACK

Tabelle 6: Unterschiedliche Arten der Transaktionsverarbeitung

Die Beispiele basieren auf folgender Tabelle:

```
CREATE TABLE transi_test
( id INT NOT NULL PRIMARY KEY,
  bzch CHAR(50),
  bem VARCHAR(50)
);
```

Man unterscheidet zwischen drei Arten:

Autocommit-Modus

Automatische Transaktionen werden beim SQL Server standardmäßig verwendet, falls keine Transaktion explizit gestartet wird.

Beispiel:

```
INSERT INTO transi_test VALUES (1, 'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (2, 'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (3, 'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (4, 'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (5, 'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (6, 'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (7, 'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (8, 'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (6, 'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (10, 'Bezeichnung', 'Bemerkung');
GO
```

Für jedes Insert wird eine Transaktion durchgeführt.

Expliziter Modus

Explizite Transaktionen werden vom Benutzer gestartet und beendet.

Beispiel:

```
BEGIN TRANSACTION
BEGIN TRY
    INSERT INTO transi_test VALUES (1, 'Bezeichnung', 'Bemerkung');
    INSERT INTO transi_test VALUES (2, 'Bezeichnung', 'Bemerkung');
    INSERT INTO transi_test VALUES (3, 'Bezeichnung', 'Bemerkung');
    INSERT INTO transi_test VALUES (4, 'Bezeichnung', 'Bemerkung');
    INSERT INTO transi_test VALUES (5, 'Bezeichnung', 'Bemerkung');
    INSERT INTO transi_test VALUES (6, 'Bezeichnung', 'Bemerkung');
    INSERT INTO transi_test VALUES (7, 'Bezeichnung', 'Bemerkung');
    INSERT INTO transi_test VALUES (8, 'Bezeichnung', 'Bemerkung');
    INSERT INTO transi_test VALUES (6, 'Bezeichnung', 'Bemerkung');
    INSERT INTO transi_test VALUES (10, 'Bezeichnung', 'Bemerkung');
    PRINT 'Transaktion erfolgreich';
    COMMIT TRANSACTION;
END TRY
BEGIN CATCH
    PRINT 'Transaktion NICHT erfolgreich';
    ROLLBACK TRANSACTION;
END CATCH
```

Nach dem Alles oder Nichts Prinzip wird hier die Transaktion durchgeführt.

Impliziter Modus

Wenn Sie nicht jede Transaktion manuell starten möchten, aber dennoch die Möglichkeit bieten wollen Änderungen rückgängig zu machen.

```
SET IMPLICIT_TRANSACTIONS ON;
-- SET IMPLICIT_TRANSACTIONS OFF;

INSERT INTO transi_test VALUES (1,'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (2,'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (3,'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (4,'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (5,'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (6,'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (7,'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (8,'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (6,'Bezeichnung', 'Bemerkung');
INSERT INTO transi_test VALUES (10,'Bezeichnung', 'Bemerkung');
GO
-- COMMIT;
-- ROLLBACK;
```

Meldet Primärschlüsselverletzung, füllt alle Tupel ein, ausgenommen das 9. Tupel.

Die eingefügten Tupel sind jedoch nur für den **aktuellen Benutzer** sichtbar. Erst bei **erfolgreichem COMMIT bzw. ROLLBACK** werden die **Änderungen für andere Benutzer sichtbar**. Schliessen Sie

Änderungen erfolgen nur für den aktuellen Benutzer.

Isolationsstufe

Die Isolationsstufe bestimmt inwiefern die Daten während einer Transaktions für andere Prozesse zur Verfügung stehen.

Isolation level	Dirty read	Nonrepeatable read	Phantom
Read uncommitted	Yes	Yes	Yes
Read committed	No	Yes	Yes
Repeatable read	No	No	Yes
Snapshot	No	No	No
Serializable	No	No	No

Lost Update: Zwei Transaktionen schreiben hintereinander in die gleiche Variable - die erste Schreiboperation ist überschrieben und verloren.

Dirty Read: Aus einer Transaktion die mit ROLLBACK zurückgesetzt wird, wurde zu ihrer Laufzeit gelesen; die gelesenen Werte existierten nie in einem konsistenten Zustand.

Nonrepeatable Read: Die gleiche Variable wird durch eine Transaktion mehrmals gelesen und von "aussen" gleichzeitig mutiert. Die Transaktion liest immer andere Werte.

Phantom Row: Eine laufende Transaktion liest wiederholt in der gleichen Tabelle. Eine zweite Transaktion hat in der Zwischenzeit aber Tupel eingefügt oder gelöscht. Die erste Transaktion sieht Phantome.

Folgendermassen kann die Isolationsstufe ermittelt werden:

```

/* aktuelle Isolationsstufe */
SELECT
CASE transaction_isolation_level
WHEN 0 THEN 'Unspecified'
WHEN 1 THEN 'ReadUncommitted'
WHEN 2 THEN 'ReadCommitted'
WHEN 3 THEN 'Repeatable'
WHEN 4 THEN 'Serializable'
WHEN 5 THEN 'Snapshot'
END AS Isolationsstufe
FROM sys.dm_exec_sessions
WHERE session_id = @@SPID;

```

Read uncommitted

```

BEGIN TRANSACTION;
SET TRANSACTION ISOLATION LEVEL READ
UNCOMMITTED;
/*dirty read */
SELECT * FROM konto;

```

Unabhängig ob eine Transaktion stattfindet wird der Zugriff durchgeführt. Diese Art von Zugriff beschreibt man als Dirty Read, da man dabei inkonsistente Daten lesen kann.

Read committed

```

BEGIN TRANSACTION;
SET TRANSACTION ISOLATION LEVEL READ
COMMITTED;
SELECT * FROM konto
WHERE id konto = 'A';

```

Aktionen werden sequenzweise durchgeführt, d.h. erst wenn die vorgängige Aktion abgeschlossen ist wird die folgende durchgeführt.

Repeatable Read

V01	V02
<pre> BEGIN TRANSACTION; SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; SELECT * FROM konto WHERE betrag > 4000; </pre>	
	<pre> BEGIN TRANSACTION; SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED; INSERT konto VALUES ('F', 6000); COMMIT; </pre>
<pre> UPDATE konto SET betrag += 99; COMMIT; </pre>	

Die zweite V01 Aktion aktualisiert auch das unbekannte F element. Vor jeder Manipulation werden die Daten neu eingelesen.

Funktionen und Prozeduren

08 May 2015 11:17

Funktionen

- Implementieren Algorithmen und/oder Verarbeitungsschritte.
- Es gibt vordefinierte Funktionen.

Zeichenketten-Funktionen		
ASCII	NCHAR	SOUNDEX
CHAR	PATINDEX	SPACE
CHARINDEX	QUOTENAME	STR
DIFFERENCE	REPLACE	STUFF
LEFT	REPLICATE	SUBSTRING
LEN	REVERSE	UNICODE
LOWER	RIGHT	UPPER
LTRIM	RTRIM	
Systemstatistische Funktionen		
@@CONNECTIONS	@@PACK_RECEIVED	@@TOTAL_ERRORS
@@CPU_BUSY	@@PACK_SENT	@@TOTAL_READ
@@IDLE	@@PACKET_ERRORS	@@TOTAL_WRITE
@@IO_BUSY	@@TIMETICKS	fn_virtualfilestats

- Funktionen können parametrisiert werden und geben immer einen Wert zurück.
- Funktionen gelten immer als UDF (User Defined Functions).
- Es können keine bleibenden Änderungen in einer Funktion vorgenommen werden. -> Kein Insert

Beispiel - Rundungsfunktion:

```
CREATE FUNCTION f_runden(@betrag FLOAT,@genauigkeit FLOAT)
  RETURNS DECIMAL(10,4) -- Genauigkeit ggf. anpassen!!!
  AS
  BEGIN
    IF @genauigkeit=0 RETURN (@betrag)
    RETURN (ROUND(@betrag/@genauigkeit,0) * @genauigkeit)
  END;
```

Prozeduren

- Prozeduren sind im Grunde Funktionen ohne Rückgabewerte.
- Sie erlauben Veränderungen vorzunehmen.

Trigger

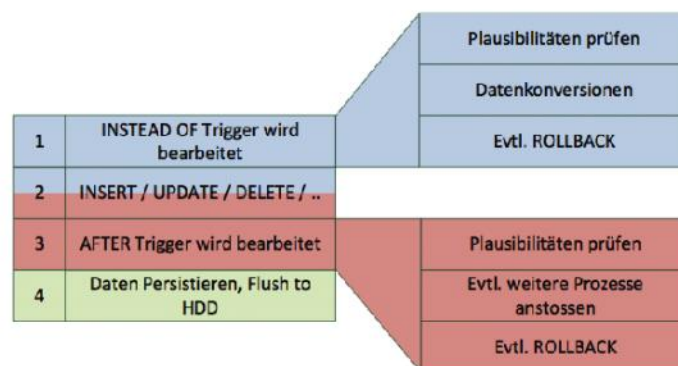
15 May 2015 11:06

- Trigger werden bei bestimmten Events ausgeführt.
- Sind für Benutzer nicht sichtbar.
- Können nicht implizit aufgerufen werden.
- Haben keine Parameter und Rückgabewerte.

Beispiel - Trigger:

```
CREATE TRIGGER constraint_triggi ON t_demo
AFTER INSERT, UPDATE
AS
BEGIN
    SET nocount on;
    IF (SELECT COUNT(*) FROM t_demo JOIN inserted ON t_demo.id=inserted.id) >
        (SELECT COUNT(*) FROM inserted)
    BEGIN
        RAISERROR ('Attribut id lässt keine Doppelnennungen zu.', 15, 1);
        ROLLBACK TRANSACTION;
    END
    ELSE
    IF ((SELECT COUNT(*) FROM inserted WHERE id IS NULL)> 0)
    BEGIN
        RAISERROR ('Attribut id darf nicht NULL sein.', 15, 1);
        ROLLBACK TRANSACTION;
    END
END
GO
```

- Jeder Insert und Update auf die Tabelle t_demo löst den Trigger aus.
- Trigger verhindert doppelte und leere IDs.



Der Unterschied zwischen Instead of und After

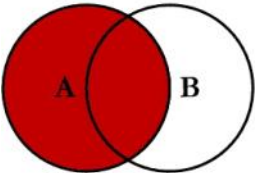
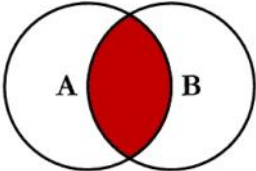
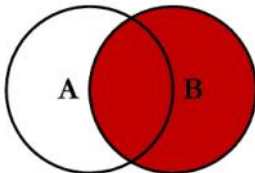
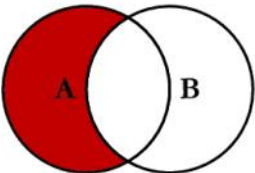
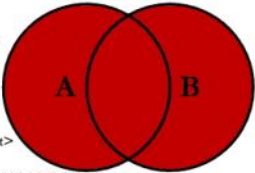
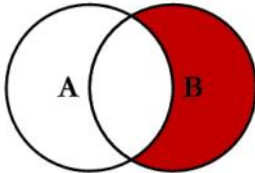
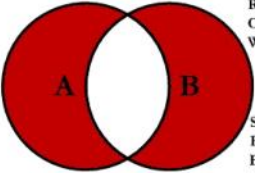
INSTEAD OF: Der Triggercode wird anstatt (instead of) der eigentlichen Anweisung ausgeführt. Der Besitzer des Triggers muss den jeweiligen Mutationsbefehl selber im Trigger definieren. Bsp: INSTEAD OF INSERT unten

AFTER: Der Triggercode wird erst ausgeführt nachdem das eigentliche SQL Kommando bereits abgeschlossen ist. Die Daten befinden sich im Arbeitsspeicher, sind aber noch nicht persistiert. Sprich ist ein Rollback immer noch möglich!

SQL Syntax

07 April 2015 10:09

Bezeichnung	Beschreibung
Kommentare	<p>Für <i>einzeilige Kommentare</i> gibt es das "Doppelminus":</p> <pre>-- Die folgende Zeile selektiert alle Angestellten von northwind SELECT * FROM dbo.employees</pre> <p>Für <i>Kommentare am Zeilenende</i> gilt dasselbe:</p> <pre>SELECT * FROM dbo.employees -- alle Angestellten anzeigen</pre> <p>Für <i>mehrzeilige Kommentare</i> klammern Sie mit</p> <pre>/* Mehrzeilenkommentar ohne go Mehrzeilenkommentar ohne go Mehrzeilenkommentar ohne go */</pre>
GO	<p>Mit dem GO-Befehl kann man den Code in mehrere Sequenzen unterteilen. Schlägt eine Fehl läuft der Code weiter.</p>
DATABASE	<p>Braucht die Rechte</p> <ul style="list-style-type: none">• sysadmin• dbcreator• db_ddladmin <p>-- neue Datenbank anlegen</p> <pre>USE master go CREATE DATABASE database_name [ON [PRIMARY] <filespec> [LOG ON <filespec>] [COLLATE collation_name]] [;] go USE database_name go</pre> <p>-- Datenbankdefinition aendern</p> <pre>USE master go ALTER DATABASE database_name { <datei_optionen> MODIFY NAME = new_database_name COLLATE collation_name } [;] go USE database_name go</pre> <p>-- Datenbank loeschen</p> <pre>USE master go DROP DATABASE database_name [,...n] [;] go</pre>
Metadaten	<p>Metadaten können entweder über die System Datenbank oder über Funktionen abgefragt werden.</p> <p>Datenbankobjekte (DBO)</p> <pre>SELECT * FROM sys.servers; GO EXECUTE sys.sp_helpserver; GO SELECT * FROM sys.databases; GO SELECT DB_ID('northwind'); GO SELECT DB_NAME(1); GO EXECUTE sys.sp_helpdb 'northwind';</pre> <p>Tabellenobjekte</p> <pre>USE northwind SELECT * FROM INFORMATION_SCHEMA.TABLES; -- ISO konform GO</pre>

	<h1 style="text-align: center;">SQL JOINS</h1> <div style="display: flex; flex-wrap: wrap; justify-content: space-around;"> <div style="text-align: center;">  <pre>SELECT <select_list> FROM TableA A LEFT JOIN TableB B ON A.Key = B.Key</pre> </div> <div style="text-align: center;">  <pre>SELECT <select_list> FROM TableA A INNER JOIN TableB B ON A.Key = B.Key</pre> </div> <div style="text-align: center;">  <pre>SELECT <select_list> FROM TableA A RIGHT JOIN TableB B ON A.Key = B.Key</pre> </div> <div style="text-align: center;">  <pre>SELECT <select_list> FROM TableA A LEFT JOIN TableB B ON A.Key = B.Key WHERE B.Key IS NULL</pre> </div> <div style="text-align: center;">  <pre>SELECT <select_list> FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key</pre> </div> <div style="text-align: center;">  <pre>SELECT <select_list> FROM TableA A RIGHT JOIN TableB B ON A.Key = B.Key WHERE A.Key IS NULL</pre> </div> <div style="text-align: center;">  <pre>SELECT <select_list> FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key WHERE A.Key IS NULL OR B.Key IS NULL</pre> </div> </div> <p style="text-align: center; font-size: small;">©CL. Moffatt, 2008</p>
UPDATE	<p>Diese DML Anweisung mutiert Tupel in eine Tabelle.</p> <pre>UPDATE tabellen_name SET spalten_name = { ausdruck DEFAULT NULL } [,...n] [FROM tabellen_verbund [,...n]] [WHERE bedingung] [;]</pre>
DELETE	<p>Diese DML Anweisung löscht selektierte Tupel aus einer Tabelle.</p> <pre>DELETE [FROM] tabellen_name [FROM tabellen_verbund [,...n]] [WHERE bedingung] [;]</pre> <ul style="list-style-type: none"> • Respektiert Referenzielle Integrität • Sperrt da sTupel • Protokollierung im Transaktions-Log • Event-Trigger
TRUNCATE	<p>Löscht alle Tupel einer Tabelle.</p> <pre>TRUNCATE TABLE tabellen_name [;]</pre> <ul style="list-style-type: none"> • Respektiert Referenzielle Integrität • Sperrt da sTupel • Protokollierung im Transaktions-Log erfolgt atomar • Keine Event-Trigger
ALTER	<p>DDL Anweisung zur Änderung von Tabellendefinitionen.</p> <pre>ALTER TABLE tabellen_name ADD spalten_name datentyp [spaltenbezogene_einschraenkung] [...n] DROP COLUMN spalten_name ALTER COLUMN alt_spalten_name neu_datentyp [NULL NOT NULL] [COLLATE kollation] ADD [CONSTRAINT] tabellenbezogene_einschraenkung] Einschraenkung [,...n] DROP [CONSTRAINT] einschraenkung_name [, ...n] [;]</pre>
Index	<p>DDL Anweisungen zur Mutation von Indexten.</p> <pre>CREATE [UNIQUE] [CLUSTERED NONCLUSTERED] INDEX index_name ON <object> (column [ASC DESC] [,...n]) [INCLUDE (column_name [,...n])] [WITH (<Optionen> [,...n])] [;]</pre>

	<pre> ALTER INDEX [index_name ALL] ON <object> { REBUILD DISABLE REORGANIZE SET (<Option> [,...n]) } [;] DROP INDEX index_name ON table_or_view_name [,...n] </pre>
TYPE	<pre> CREATE TYPE tp_domizil FROM VARCHAR(30) CREATE TYPE tp_kohle FROM numeric(7,2) CREATE TYPE tp_statuscode FROM DECIMAL(2) </pre> <p>Kann mit Check constraints ergänzt werden</p>
CHECK	<pre> id_dings INT NOT NULL PRIMARY KEY CHECK (id_dings > 999), ding_name VARCHAR(25) UNIQUE NOT NULL, ding_stao CHAR(10) CHECK(ding_stao LIKE '____-____'), ding_anlager CHAR(1) DEFAULT 'n' CHECK (ding_anlager IN ('j', 'J', 'n', 'N')), ding_sitz tp_domizil NOT NULL DEFAULT 'Luzern' CHECK(ASCII(ding_sitz) BETWEEN 65 AND 90), ding_status tp_statuscode CHECK(ding_status BETWEEN 0 AND 15), umsatz tp_kohle) go </pre>
LOGIN	<p>Zugangsdaten auf den Server werden über das Login-Objekt verwaltet.</p> <pre> CREATE LOGIN login_name WITH PASSWORD = 'password' [HASHED] [MUST_CHANGE] [, DEFAULT_DATABASE = database DEFAULT_LANGUAGE = language CHECK_EXPIRATION = { ON OFF } CHECK_POLICY = { ON OFF } [,...]] </pre>
USER	<p>Zugangsdaten auf Datenbanken werden über das User-Objekt verwaltet.</p> <pre> CREATE USER user_name [FOR LOGIN login_name WITHOUT LOGIN] [WITH DEFAULT_SCHEMA = schema_name] </pre>
ROLE	<ul style="list-style-type: none"> • ein Rollenname bedarf keiner Abbildung auf einen Serverbenutzer • eine Rolle gilt auf Datenbankebene • nur die Datenbankeigentümerin (und andere privilegierte Datenbankrollen) kann eine Rolle anlegen • eine Rolle gehört dem Benutzer, der sie angelegt hat oder kann mit AUTHORIZATION übertragen werden <pre> CREATE ROLE role_name [AUTHORIZATION owner_name]; </pre>
GRANT	<p>Die mit GRANT erteilbaren und mit REVOKE rückrufbaren Objektberechtigungen (<i>Permission</i>) auf Tabellen und Sichten sind: SELECT, INSERT, DELETE, REFERENCES und UPDATE.</p> <p>Die mit GRANT erteilbaren und mit REVOKE rückrufbaren erweiterten Berechtigungen (<i>Permission</i>) in der Datenbank sind: CREATE DEFAULT, CREATE FUNCTION, CREATE PROCEDURE, CREATE ROLE, CREATE TABLE, CREATE TYPE, CREATE VIEW, DELETE, EXECUTE, INSERT, SELECT, UPDATE.</p> <pre> GRANT { ALL [PRIVILEGES] permission [(column [,...n])] [,...n] [ON objekt] TO principal [,...n] [WITH GRANT OPTION] } </pre>

REVOKE	<pre> REVOKE [GRANT OPTION FOR] { [ALL [PRIVILEGES]] permission [(column [,...n])] [,...n] } [ON objekt] FROM principal [,...n] [CASCADE] </pre>
VIEW	<pre> CREATE VIEW view_name [(column [,...n])] AS select_statement [;] go DROP VIEW sicht_name go </pre>
SELECT	<p>Es gilt folgende Ausführungshierarchie:</p> <ol style="list-style-type: none"> 1. FROM 2. WHERE 3. GROUP BY 4. HAVING 5. SELECT 6. ORDER BY <pre> SELECT { * [ALL DISTINCT] [tabellen_name.] spalten_name [AS alias] [, ...n] } FROM tabellen_name [[AS] alias] [, ...n] [WHERE Bedingung_Einzel_Tupel] [GROUP BY spalten_name] [HAVING Bedingung_fuer_Tupel_Aggregat] [ORDER BY spalten_name [ASC DESC]] </pre> <p>Aggregierungen sind Gruppenbildungen. Sie erfolgen explizit mit GROUP BY attribut oder implizit durch die Verwendung einer Aggregatsfunktion.</p> <p>SQL kennt die hauptsächlichen Aggregatsfunktionen (SQL-Nutshell p. 438ff): MIN, MAX, COUNT(*), COUNT([DISTINCT] spalte), SUM, AVG.</p> <p>Ein Aggregat ist immer eine Zusammenfassung. Daher gilt: In einer SELECT-Anweisung mit Aggregatsfunktionen können nicht zusätzlich Spaltenprojektionen angezeigt werden: Eine Anweisung <i>nur die in der Groupby Klausel vorkommen</i></p> <pre> SELECT dt_name, MAX(dt_zeit) FROM ... </pre> <p>ist deshalb <u>nicht möglich</u>. <i>wenn dt_name in group by klausel</i></p> <p>HAVING funktioniert wie WHERE - aber es selektiert ein Aggregat.</p> <p>Beispiel 1: Wirkung auf das Aggregat COUNT()</p> <pre> SELECT dt_name AS 'Interpret', COUNT(dt_stueck_titel) AS 'Anzahl' FROM tkey_interpret JOIN tbl_stueck ON id_interpret = fi_interpret GROUP BY dt_name HAVING COUNT(dt_stueck_titel) > 6 ORDER BY 'Anzahl' DESC, dt_name ASC go </pre> <p>Abfragen können auch verschachtelt werden:</p> <pre> SELECT dt_stueck_titel AS 'Titel', dt_zeit AS 'Zeit' FROM tbl_stueck WHERE dt_zeit > 2 * (SELECT AVG(dt_zeit) FROM tbl_stueck) ORDER BY dt_zeit; </pre> <p>Dabei gehört die Unterklausel ins WHERE statement.</p> <p>Select liefert in der Regel eine Menge. Verwendet man eine Aggregatfunktion liefert Select einen Wert.</p>
Mengenoperation	<p>SQL ist mengenorientiert (siehe Join). Die Ergebnisse geordneter Mengen können mit Mengenoperationen verglichen werden.</p> <ul style="list-style-type: none"> • Die folgenden Operationen setzen voraus, dass die Abfragen strukturell vergleichbar aufgebaut sind, d.h. gleich viele Spalten projizieren. • Gleiche Spaltenpositionen müssen kompatible Datentypen haben. • Dupletten werden eliminiert

Schnittmenge

Schnittmenge: Die Resultat-Tupel zweier einzelner Abfragen haben in allen gemeinsamen Attributen dieselben Werte und werden als Ergebnis ausgegeben. Lösung:

- entweder Syntax: `abfrage1 INTERSECT abfrage2`
- oder entsprechend aufwändig formulierte `abfrage3` mit einer WHERE-Klausel unter Verwendung von `EXISTS` oder `IN`.

Differenzmenge

Differenzmenge: Die Resultat-Tupel kommen in Abfrage 1, nicht aber ebenfalls in Abfrage 2 vor. Lösung:

- entweder Syntax: `abfrage1 EXCEPT abfrage2`
(in gewissen Dialekten `MINUS` oder `DIFFERENCE` statt `EXCEPT`)
- oder entsprechend aufwändig formulierte `abfrage3` mit einer WHERE-Klausel unter Verwendung von `NOT EXISTS` oder `NOT IN`.

Vereinigungsmenge

Vereinigungsmenge: Die Resultat-Tupel kommen in mindestens einer der beteiligten Abfragen vor. Lösung:

- entweder Syntax: `abfrage1 UNION abfrage2`
- oder entsprechend aufwändig formulierte `abfrage3` mit `WHERE ... OR`.

FUNCTION

Benötigt create function berechtigung zum anlegen.

```
CREATE FUNCTION funktions_name  
  ([[parameter_name [AS] Datentyp [= Initialisierung] [,...n ]])  
  RETURNS Datentyp
```

```
  [ AS ]  
  BEGIN  
    function_body  
    RETURN [Ausdruck]  
  END
```

go

```
ALTER FUNCTION funktions_name ...
```

go

- ganzer Code nochmals

Löschen

```
DROP FUNCTION funktions-name [, ...]
```

go

Rechte eines (eröffneten) Benutzers

```
GRANT EXECUTE ON funktions_name TO db_prinzipal [, ...]
```

go

Beispiel - Zugriff auf Tabelle

```

CREATE FUNCTION f_bezahlt_kunde (@ku_name VARCHAR(30))
RETURNS DECIMAL(8,2)
AS
BEGIN
RETURN (SELECT SUM(bezahlt) FROM tbl_kunde JOIN tass_police
ON id_kunde = id_fi_kunde WHERE name = @ku_name)
END;
GO

SELECT name, dbo.f_bezahlt_kunde(name) FROM tbl_kunde;

```

Die Funktion wird wie folgt verwendet:

```

SELECT name, dbo.f_bezahlt_kunde(name) AS Total FROM tbl_kunde;

```

PROCEDURE

Benötigt create procedure berechtigung zum anlegen.

```

CREATE PROCEDURE prozedur_name
[@parameter_name [AS] Datentyp [= Initialisierung] [OUTPUT]]
[, ...n ]
[ AS ]
BEGIN
    procedure_body
    [RETURN Ausdruck]
END
go

```

Ändern

```

ALTER PROCEDURE prozedur_name ...
- ganzer Code nochmals

```

Löschen

```

DROP PROCEDURE prozedur_name [, ...]
go

```

Rechte eines (eröffneten) Benutzers

```

GRANT EXECUTE ON prozedur_name TO userid [, ...];
go

EXEC[UTE] [@return_status =] prozedur_name [[@parameter_name =] expression [, ...]]

```

Beispiel - Variable setzen

Die folgende Prozedur wird mit einer Variablen als Parameter aufgerufen. Sie verändert deren Wert, der im gleichen Stapel verfügbar bleibt, wenn die Prozedur beendet ist (es könnten auch mehrere Variablen so verändert werden):

```

CREATE PROC p_ums_tot @umsatz int OUTPUT -- Variante 1
AS
BEGIN
    SET @umsatz = (SELECT sum(bezahlt) FROM tass_police);
END;
GO

```

Folgender Aufruf muss als *ein* go-Stapel laufen. Er zeigt ferner, dass Kontrollstrukturen (z.B. IF) auch in normalen SQL-Stapeln verwendet werden können:

```

DECLARE @umsatz_total INT;
EXECUTE p_ums_tot @umsatz_total OUTPUT;

IF (@umsatz_total > 20000)
    PRINT 'Wir haben wieder mal vorwärts gemacht.';
ELSE PRINT 'Wir sollten uns sputen.';
GO

```

Weil obige Prozedur nur einen Wert zurück gibt, könnte auch wie folgt codiert werden. Die beiden Formen (OUTPUT-Variablen und RETURN) können vermischt werden. Das Beispiel zeigt ferner, dass in der Prozedur und ausserhalb die selben Variablennamen verwendet werden können - aber nicht müssen. Das Gleiche gilt auch für die obige Variante:

```
ALTER PROCEDURE p_ums_tot -- Variante 2
AS
BEGIN
    DECLARE @u INT
    SET @u = (SELECT sum(bezahlt) FROM tass_police);
    RETURN @u;
END
GO

DECLARE @umsatz INT;
EXEC @umsatz = p_ums_tot;

IF (@umsatz > 20000)
    PRINT 'Wir haben wieder mal vorwärts gemacht.';
ELSE PRINT 'Wir sollten uns sputen.';
GO
```



TRIGGER

Ist eine DDL Anweisung.

```
CREATE TRIGGER trigger_name
ON { table | view }
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS
BEGIN
    Anweisungsblock
END
```

Ändern

```
ALTER TRIGGER trigger-name ...

go

- ganzer Code nochmals
- ALTER TRIGGER ist nicht Standard
```

Löschen

```
DROP TRIGGER trigger-name [, ...n]

go
```

Beispiel - Löschung Kunde verhindern.

```
CREATE TRIGGER t_ku_del ON tbl_kunde
AFTER DELETE
AS
BEGIN
    DECLARE @m AS TINYINT, @e AS VARCHAR(80);
    SET nocount on;
    SET @m = (SELECT MAX(fi_moral_nr) FROM deleted);
    IF @m >= 3
    BEGIN
        SET @e = 'Kunden mit Moral ' + CAST(@m AS CHAR(1))
            + ' werden nicht gelöscht!';
        RAISERROR(@e,15,1);
        ROLLBACK TRANSACTION;
    END
END
```

- Wenn irgendein Kunde aus den gelöschten Datensätze eine Kundenmoral höher 3 hat, wird der Löschvorgang rückgängig gemacht.
- Wichtig!: Das ACID Prinzip verifiziert, dass keine teilweise Ausführungen gemacht werden. Deshalb wird die Löschung immer für alle Elemente verhindert.