



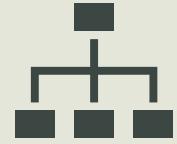
Vorlesung DB-Implementierung

Datenbankabfragen, Indizierung & Advanced Features

Lehrbeauftragter : Thomas Gutmann

Seminargruppe TINF21AI1

Die „SQL“-Familie



Übersicht



SQL, Die „SQL“-Familie

- SQL, Historie & ISO-Standard
- Definition
- SQL als ISO-Standard
- SQL, „Abfragefamilie“
- ORM-Tools,
Umsetzung der SQL-Integration in Frameworks

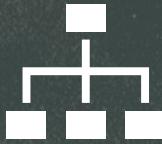


Datentypen & Besonderheiten

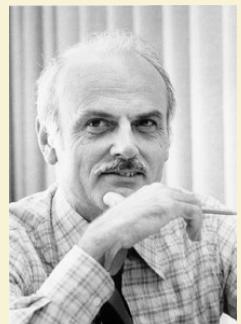
- Datentypen
- Datentypen & Besonderheiten
 - Inhaltstyp „NULL“
 - Status-Spalten und ihr Wesen
- Compliance
 - Datenschutz
 - Informationssicherheit
 - Governance / IT-Governance
 - Compliance

Rückblick

DIE ENTSTEHUNG VON SQL



■ SQL



- 1970er Edgar "Tedd" Codd (IBM), Entwicklung relationaler Datenmodelle
- 1974 Donald Chamberlin & Raymond Boyce veröffentlichen den Artikel
 - „SEQUEL“, A Structured English Query Language
 - Detaillierte technische Beschreibung der Sprachsyntax (BNF-Grammatik)
- 1980er IBM bezeichnete später die Veröffentlichung als „Fehler“
 - Verschiedene Anbieter setzen eine eigene SQL-Implementierung um
 - Die SQL-Dialekte waren teilweise stark abweichend
- 1985 Mitte der 1980er, erhebliche Inkompatibilitäten zwischen den SQL-Versionen
- 1986 **Standardisierung der Structured Query Language „SQL“ durch ANSI**

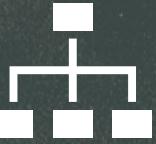




Rückblick

DIE ENTWICKLUNG VON SQL





Structured Quer Language „SQL“

- **SQL**

- Structured Query Language „SQL“ ist eine Sprache der 4. Generation (4GL)
- Ziel von SQL ist die **Trennung** des „**Was**“ vom „**Wie**“
- Schwerpunkt ist die **Abfrage von Inhalten** (einer Datenbank)
- **Unabhängig von der Art & Weise der Ausführung**
- Starke Reduzierung von Code-Elementen im Vergleich zu 3GL (Java, C#, C++)

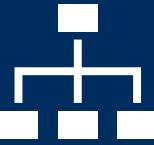
SQL ist eine Sprache zur Abfrage von relationalen Datenbanken.

!

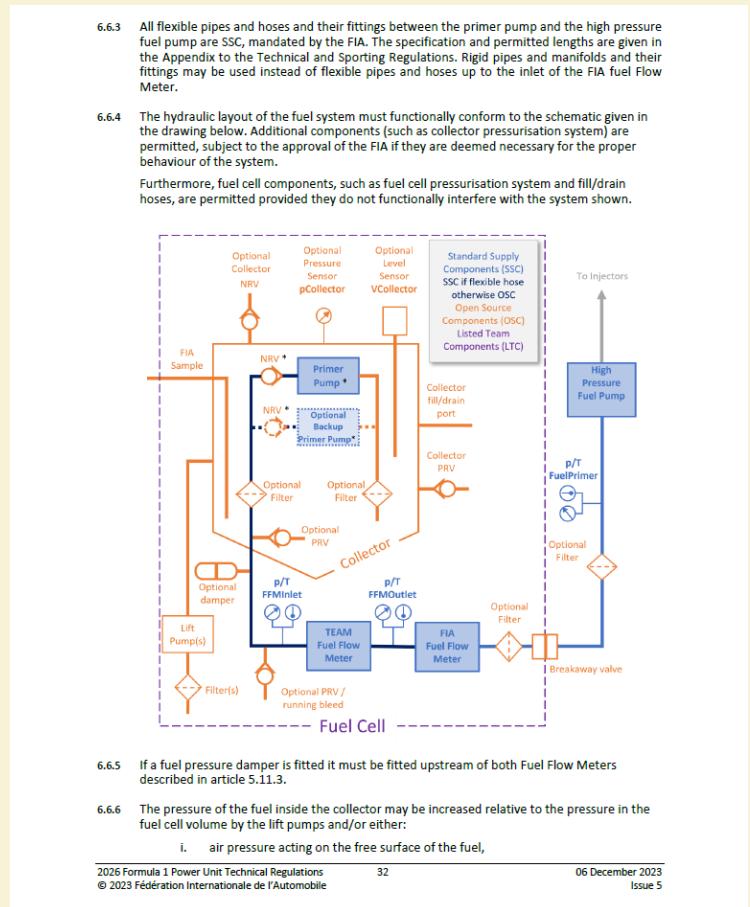
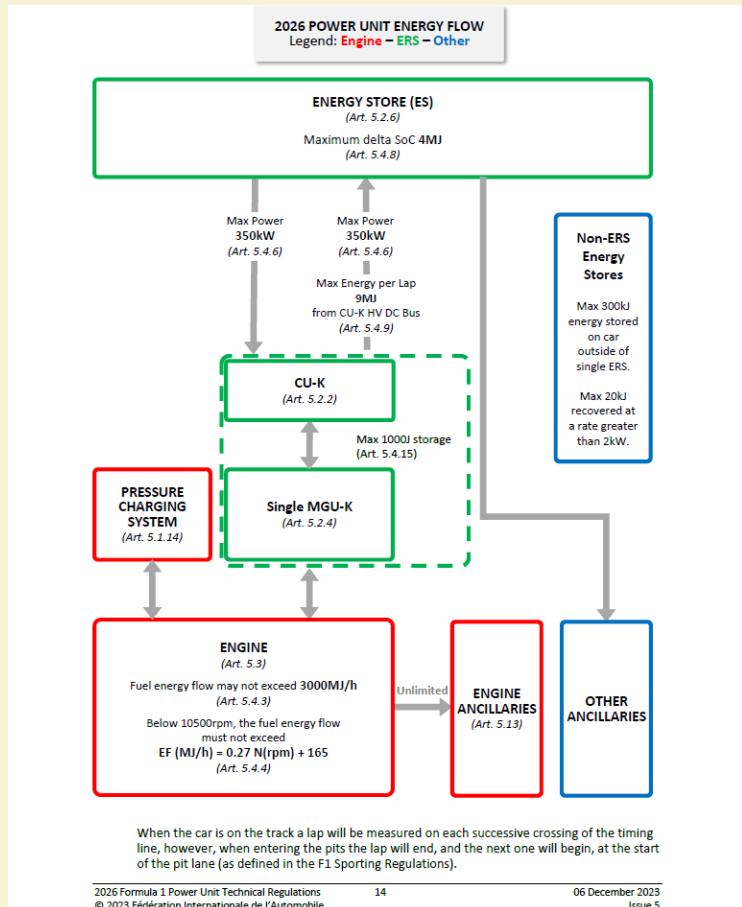
Gleichzeitig steht der Begriff als Synonym für die Sprachfamilie aller ausführbaren Anweisungen eines Datenbankmanagementsystems.



Fédération
Internationale
de l'Automobile



Herausforderung DER STANDARDISIERUNG



- 16.2.5 **Aromatics** Monocyclic and bicyclic aromatic rings with or without paraffinic side chains.
 - 16.2.6 **Oxygenates** Organic compounds containing oxygen.
 - 16.2.7 **Additive** An additive is a component added to the fuel at low concentration to improve a particular property of the fuel. These include (but are not limited to) antioxidants, antiknock agents, antistatic additives and deposit control additives.
 - 16.2.8 **Advanced Sustainable component** An Advanced Sustainable (AS) Component is one that is certified to have been derived from a renewable fuel of non-biological origin (RFNBO), municipal waste, or non-food biomass. Such biomass includes, but is not limited to, lignocellulosic biomass (including sustainable forest biomass), algae, agricultural residues or waste, and dedicated non-food energy crops grown on marginal land unsuitable for food production. RFNBOs are considered renewable when the hydrogen component is produced in an electrolyser that uses new renewable electricity generation capacity. Biocomponents from food crops can be regarded as an advanced sustainable component only if they have already fulfilled their food purpose (e.g. waste vegetable oil because it has already been used and is no longer fit for human consumption). Furthermore, the biomass, from which the advanced sustainable component was made, must not originate from land with high biodiversity such as undisturbed primary forest or woodland, land designated for nature protection or highly biodiverse grassland, and were in this state in or after January 2008. Additionally, the biomass must not originate from any land with high-carbon stock such as wetlands and peatlands.
 - 16.2.9 **Metals** are defined as alkali metals, alkaline earth metals, transition metals, actinides, lanthanides, post-transition metals and metalloids.
 - 16.2.10 **Alkali Metals** Group 1 elements, excluding hydrogen.

16.3 Fuel properties

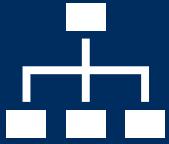
The only fuel permitted is petrol having the following characteristics

Property	Units	Min	Max	Test Method
RON		95.0 ^(d)	102.0 ^(d)	ISO 5164/ ASTM D2699
Sensitivity (RON-MON)			15.0 ^(d)	ISO 5164/ ASTM D2699 ISO 5163/ ASTM D2700
LHV	MJ/kg	38.0	41.0	GC
Density (at 15°C)	kg/m ³	720.0	785.0	ISO 12185/ ASTM D4052
Methanol ⁽²⁾	% v/v		3.0	EN 1601 or EN 13132 or EN ISO 22854
Oxygen	wt%	6.70	7.10	Elemental Analysis
Nitrogen	mg/kg		500	ASTM D 5762
Benzene	wt%		1	GCMs
DVPE	kPa	45	68	EN130161
Lead	mg/l		5	ASTM D 3237 or ICPOES
Manganese	mg/l		2	ASTM D 3831 or ICPOES
Metals (excluding alkali metals)	mg/l		5	ICPOES
Oxidation Stability	minutes	360		ASTM D 525
Sulphur	mg/kg		10	EN ISO 20846
Electrical conductivity	pS/m	200		ASTM D 2624
Distillation Characteristics:				

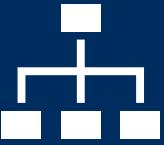


Fédération
Internationale
de l'Automobile

Herausforderung DER STANDARDISIERUNG



International Organization for Standardization



ISO-SQL



Griechisch
„ἴσος“
[isos]
Zu Deutsch
„gleich“



Datentyp

- BOOLEAN
- NUMERIC

Funktion

- LISTAGG()



BOOLEAN

- unsupported

NUMERIC

- supported

LISTAGG()

- unsupported
- alternativ:
STRING_AGG()



BOOLEAN

- supported

NUMERIC

- unsupported

- alternativ:
NUMBER

LISTAGG()

- supported



ISO & SQL, „Abfragefamilie“

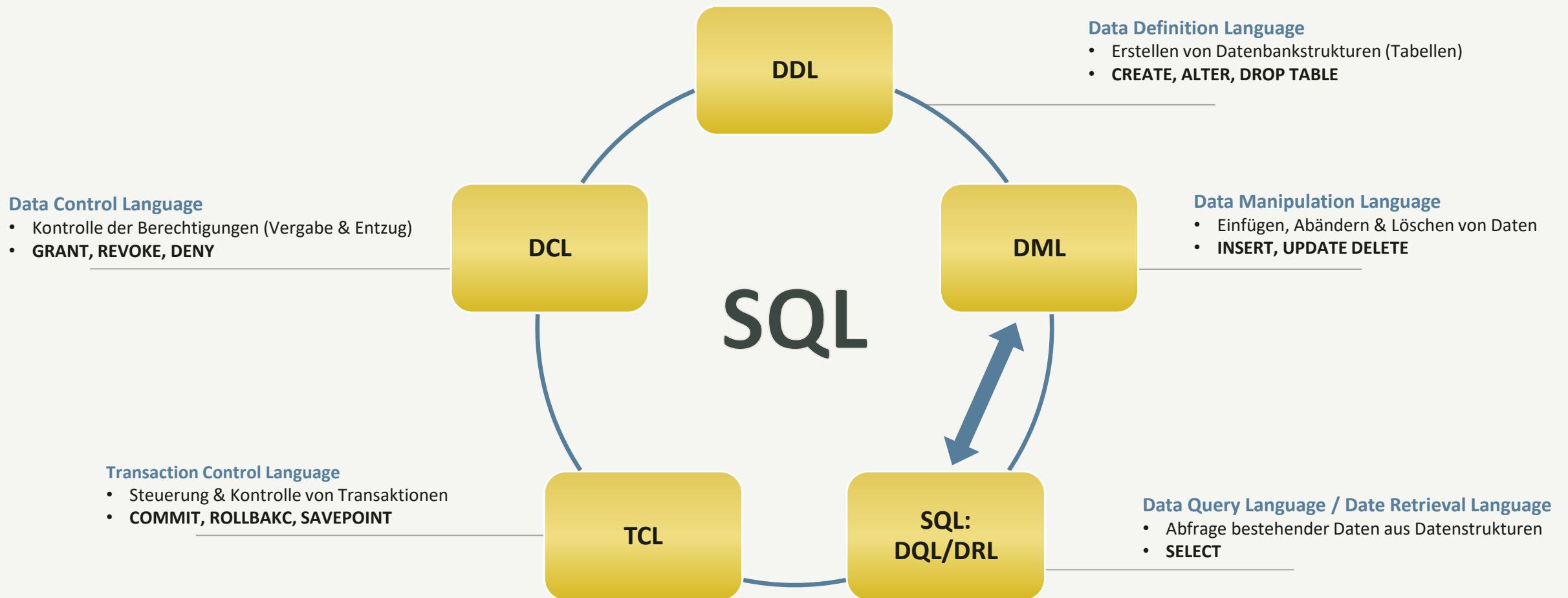
■ ISO-SQL

- Alle DBMS unterstützen in der Regel den Standard ISO-92
- Der SQL-Standard wird durch die DBMS-Hersteller funktional erweitert
- Microsoft SQL-Server mit Transact-SQL „T-SQL“
- Oracle mit dem Dialekt „PL/SQL“
- Dialekte sind **nicht kompatibel** zueinander

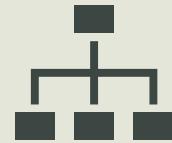
!

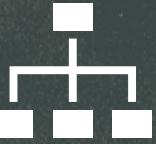
Bei der Verwendung des ISO-SQL-Standard ist Portabilität in der Regel zwischen den Datenbankmanagementsystemen gewährleistet. Dies hängt jedoch zusätzlich von der jeweiligen DBMS-Version sowie deren unterstützte ISO-SQL-Version ab.

SQL, „Abfragefamilie“



ORM-Tooling





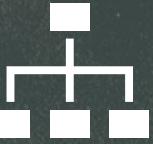
SQL-Generierung auf Basis ORM

■ ORM

- „Object Relational Mapping“-Tools
- Generierung von Datenmodellen auf Basis von Klassen
- Frameworks für die Persistenz-Schicht kapseln die Datenmodellierung
- Generische Erstellung von Query-Statements

■ ORM-Tools

- | | |
|--------------------|---|
| • Entity Framework | für .Net-Framework-/NET-Programmiersprachen |
| • Hibernate | für Java (C# und weitere) |
| • Doctrine | für PHP |
| • SQLAlchemy | für Python |
| • Active Record | für Ruby |
| • Diesel | für Rust |
| • DBIx::Class | für Perl |



ORM, Verfahren I – „Single Table“

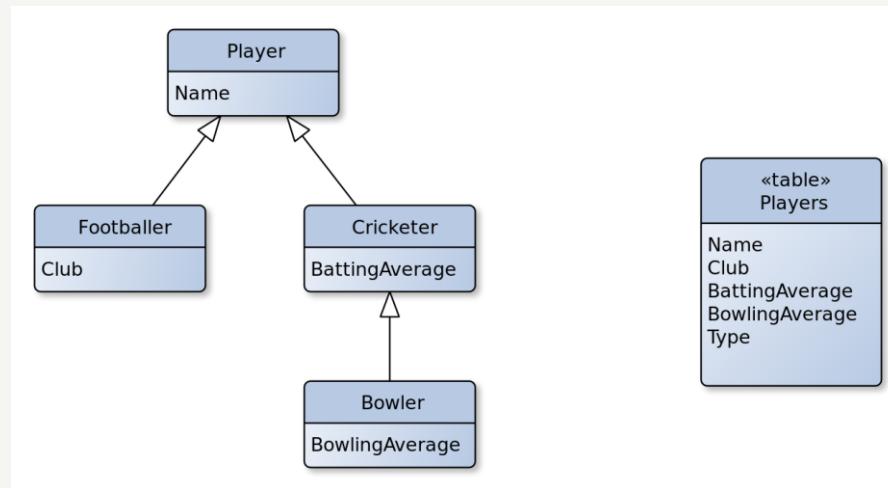


Tabelle pro Vererbungshierarchie „Single Table“

- alle Attribute der Basisklasse und alle davon abgeleiteten Klassen werden in einer Tabelle zusammengefasst
- Der „Diskriminator“ (Zusatz-Spalte) beinhaltet die Relation der Zeile zu zugehöriger Klasse & Objekt
- Attribute von abgeleiteten Klassen müssen NULL-Werte zulassen
- Beschränkungen des Verfahrens durch Maximalanzahl von Spalten pro Tabelle



ORM, Verfahren II – „Table per Class“

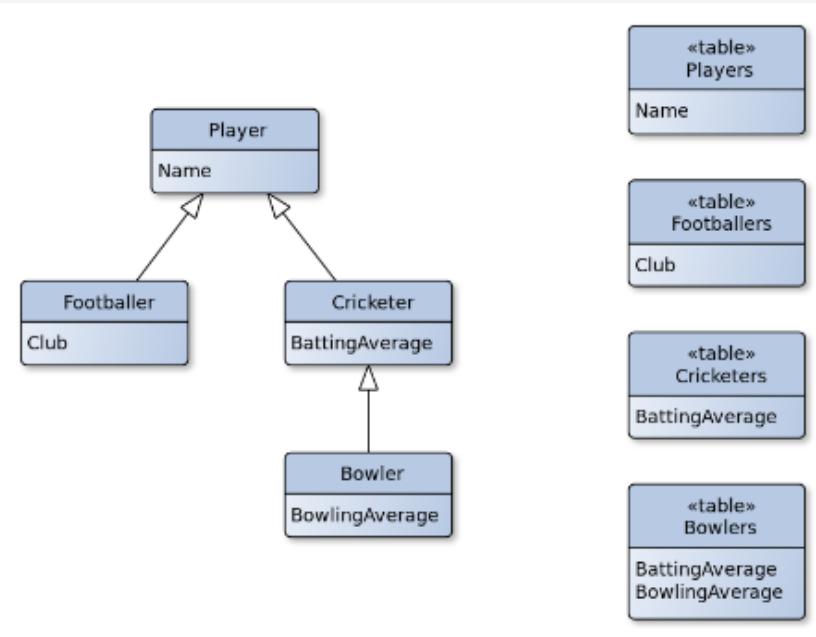
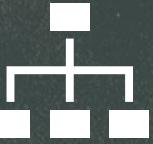


Tabelle pro Unterklasse (auch Class Table)

- eine Tabelle für jede Basisklasse sowie davon abgeleitete Unterklasse
- Ein „Diskriminator“ wird nicht benötigt
- Die Beziehung von Klasse zu Objekt ist durch die 1-zu-1-Beziehung zwischen dem Eintrag in der Tabelle der Basisklasse und einem Eintrag in einer der Tabellen der abgeleiteten Klassen festgelegt.



ORM, Verfahren III – „Table per SubClass“

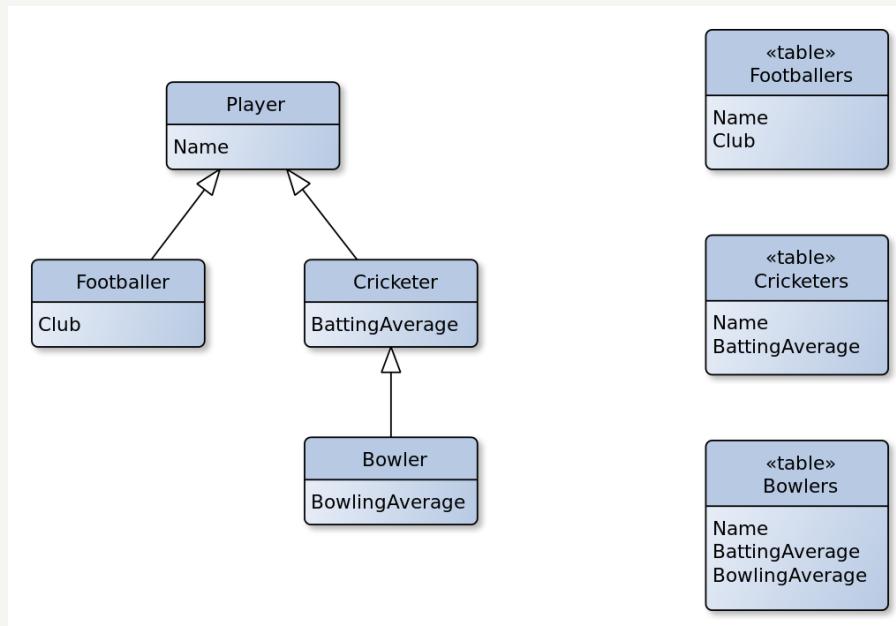


Tabelle pro konkrete Klasse

- Die Attribute der abstrakten Basisklasse in die Tabellen für die konkreten Unterklassen mit aufgenommen.
- Die Tabelle für die Basisklasse entfällt.
- Mehrere Abfragen zur Ermittlung der verschiedenen Klasseninstanzen erforderlich.

ORM, Verfahren IV



General Tales, Abbildung von Strukturen (Beziehungen, Vererbung) und Daten in generellen Tabellen

- Datenbank beinhaltet nur 5 Tabellen
- Tabelle 01: Klassen
- Tabelle 02: Beziehungen (einschließlich Vererbungsbeziehungen)
- Tabelle 03: Attribute
- Tabelle 04: Instanzen/Objekte (der Klassen)
- Tabelle 05: Werte (der Attribute)

ORM, Zusammenfassend

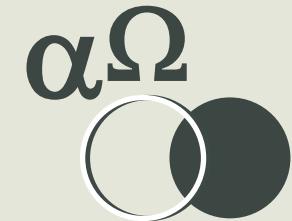


Einsatz von ORM-Tools

!

- ORM-Tools **unterstützen** die Integration der Datenbanken in die Anwendungsentwicklung
- ORM-Tools sind **kein Ersatz** für die Datenmodellierung
- ORM-Generierte Datenmodelle sind **nicht datenoptimiert, sondern objektgetrieben**
- Automatisch generierte Statements sind weniger effizient als Standardabfragen
- ORM-Tools können **keine DBMS-Optimierungen nutzen**

Datentypen & Besonderheiten



Übersicht

$\alpha\Omega$



SQL, Die „SQL“-Familie

- Definition
- SQL: Historie & ISO-Standard
- Begriffsdefinition & Abfragefamilie
- Abfragefamilie
- SQL als ISO-Standard
- Umsetzung der SQL-Integration in Frameworks

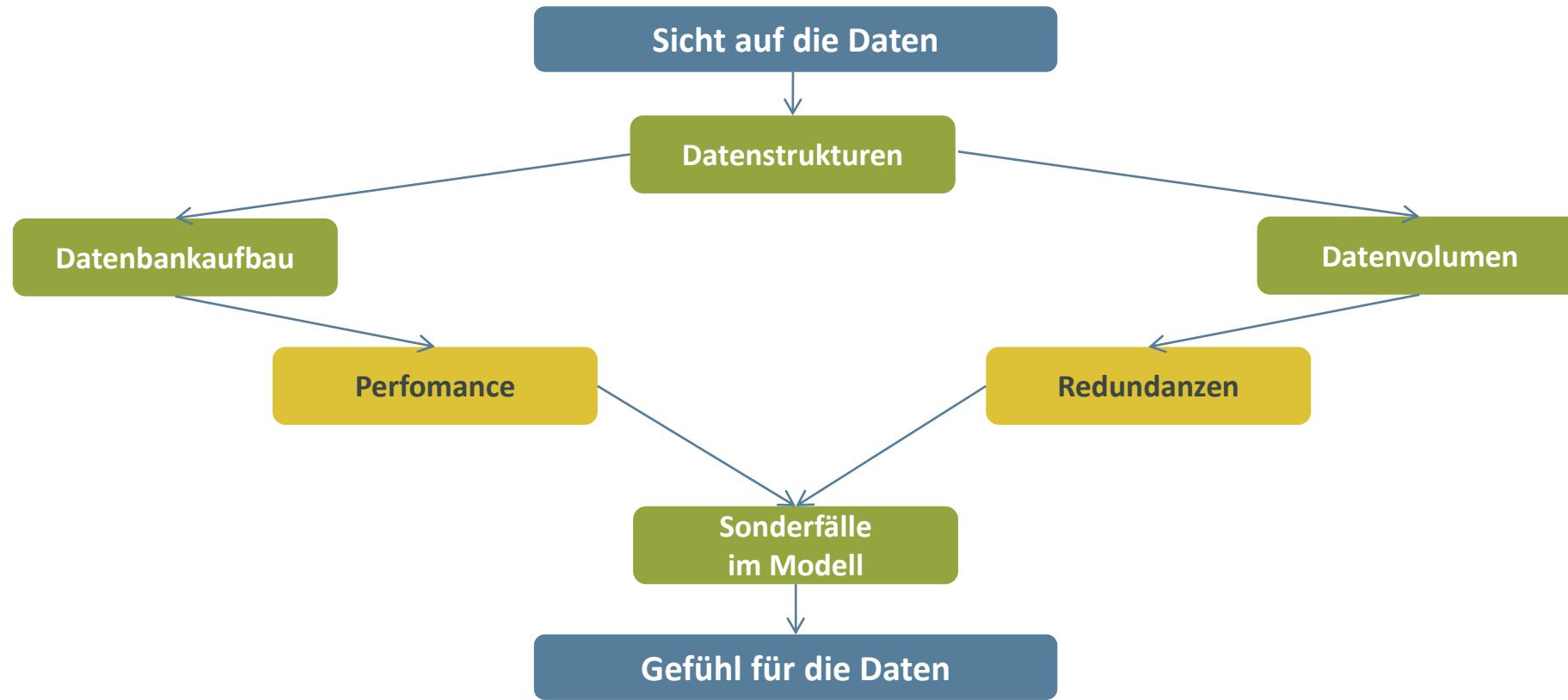


Datentypen & Besonderheiten

- Datentypen
- Datentypen & Besonderheiten
 - Inhaltstyp „NULL“
 - Status-Spalten und ihr Wesen
- Compliance
 - Datenschutz
 - Informationssicherheit
 - Governance / IT-Governance
 - Compliance

Datentypen, Hintergrund

$\alpha\Omega$



Datentypen, Klassifizierung

$\alpha\Omega$

Datentypen

Datentypen bestimmen den Inhalt von Attributen einer Tabelle.
Eine Attributspalte kann nur den Inhalt eines Datentyps beinhalten.

Zeichenfolgen

Numerische Werte

Datum & Uhrzeit

Spezialdatentypen

Zeichenfolgen

Exakte
numerische Werte

Datum

Hinweis

Unicode-
Zeichenfolgen

Ungewährte
numerische Werte

Uhrzeit

Die Basis-Datentypen
sind durch ISO definiert,
weichen jedoch zwischen
den DBMS-Systemen
voneinander ab.

Binär-
Zeichenfolgen

Datum & Uhrzeit

Datentypen, Zeichenketten

$\alpha\Omega$

Zeichenketten

Feste Länge

- **CHAR(N)**
- **NCHAR(N)**

Abkürzung

Variable Länge

- **VARCHAR(N)**
 - **NVARCHAR(N)**
- Oracle (PL/SQL)**
- **VARCHAR2(N)**

MAX
MAX

Hinweis

LOB, Large Object

- CLOB, Character Large Object
 - BLOB, Binary Large Object
- Datentypen mit fester Länge sollten nur bei Bedarf nicht grundsätzlich verwendet werden.
- „Deprecated“ Datentypen sind zu vermeiden.

Binär / CLOB / BLOB

- **BINARY(N)**
- **VARBINARY(N)**
- **IMAGE**
- **(XML)**
- **TEXT**
- **NTEXT**



MAX



deprecated

Zeichenketten

National Character

- **NCHAR(N), NVARCHAR(N), NTEXT**
 - Textspeicherung als Unicode-Text

Größenangabe

- **[N]CHAR(<Zeichenanzahl>), NVARCHAR(<Zeichenanzahl>)**
 - Angabe der Zeichenanzahl & dadurch Bestimmung der Speichergröße

Spezialgröße „MAX“

- **MAX**
 - Maximal mögliche Größe für Zeichenketten
 - Verwendung über der Grenze von 8.000 bzw. 4.000 Zeichen
 - Vorher Verwendung wie [N]VARCHAR(N)



Zeichenketten

National Character

- **VARCHAR2(N)**
 - Ersatzdatentyp für NVARCHAR(N)
 - Speicherung erfolgt in der Regel in UTF-8-Kodierung

„N“ Zeichenanzahl(N)

- **VARCHAR[2](N)**
 - N in Tabellen auf 4.000 Zeichen (Byte) begrenzt
 - N als Variable auf 32.767 Zeichen (Byte) begrenzt
 - Größere Werte als CLOB oder BLOB

Large Objects, LOB

- **CLOB & BLOB**
 - Datentypen zur Speicherung von großen Datenmengen
 - LOB-Objekte werden intern separaten Dateien vorgehalten
 - LOB-Spalten beinhalten intern eine Referenz-ID auf die LOB-Datei



Datentypen, Zeichenketten, String

$\alpha\Omega$

Zeichenketten

Datentyp	Kodierung	Beschreibung	max. Zeichenanzahl	Reservierter Speicher	Reservierung
CHAR(N)	ASCII	feste Länge	8.000	Anzahl N	grundsätzlich, auch ohne Inhalt
NCHAR(N)	Unicode	feste Länge	4.000	Anzahl 2xN	grundsätzlich, auch ohne Inhalt
VARCHAR(N)	ASCII	variable Länge	8.000	Anzahl N	nur bei Inhalt
NVARCHAR(N)	Unicode	variable Länge	4.000	Anzahl 2xN	nur bei Inhalt
VARCHAR(MAX)	ASCII	variable Länge	2.147.483.647	Anzahl N	nur bei Inhalt
NVARCHAR(MAX)	Unicode	variable Länge	1.073.741.823	Anzahl 2xN	nur bei Inhalt

Hinweis

- MAX: vor Erreichen der 8.000/4.000 Zeichen wird der Datentyp analog der Standard-Datentypen verwaltet.

Datentypen, Zeichenketten, String - Beispiel

$\alpha\Omega$

Zeichenketten

Aufgabe: Bestimmen Sie die Zeichenkettenlänge, die Byte-Länge der Zeichenkette sowie den reservierten Speicher für die Zeichenkette.

Datentyp	Inhalt	Länge der Zeichenkette "LEN()"	Länge der Zeichenkette in Byte "DATALENGTH()"	Reservierter Speicher in Byte
CHAR(25)	Informatik	10	10	25
NCHAR(30)	Daten	5	10	60
VARCHAR(20)	Variable	8	8	8
NVARCHAR(40)	Parameter	9	18	18
NVARCHAR(70)*	Επιστήμη των υπολογιστών	24	48	48

* Deutsch: Informatik

Datentypen, Zeichenketten, CLOB/BLOB

$\alpha\Omega$

Zeichenketten

Datentyp	Kodierung	Beschreibung	max. Zeichenanzahl	Reservierter Speicher	Reservierung	
BINARY(N)	Binär-Daten	feste Länge	8.000	Anzahl N	grundsätzlich, auch ohne Inhalt	
VARBINARY(N)	Binär-Daten	variable Länge	8.000	Anzahl N	nur bei Inhalt	
VARBINARY(MAX)	Binär-Daten	variable Länge	2GB	Anzahl N	nur bei Inhalt	BLOB
IMAGE	Binär-Daten	variable Länge	2GB	Anzahl N	nur bei Inhalt	BLOB
TEXT	ASCII	variable Länge	2GB (Text)	Anzahl N	nur bei Inhalt	 CLOB
NTEXT	Unicode	variable Länge	2GB (Text)	Anzahl 2xN	nur bei Inhalt	 CLOB

Datentypen, Zeichenketten, NVARCHAR(MAX)

$\alpha\Omega$

NVARCHAR(MAX)



TOLSTOJ KRIEG UND FRIEDEN

Roman
Erster Band
insel taschenbuch

Krieg und Frieden

Roman

Aus dem Russischen von Hermann Röhl und Wolfgang Kasack. Zwei Bände im Pappschuber

In seinem grandiosen Meisterwerk entfaltet Lew Tolstoj auf über 2000 Seiten ein großes Epos über das Schicksal dreier russischer Adelsfamilien in der Zeit der Napoleonischen Kriege Anfang des 19. Jahrhunderts.

In seinen epischen Dimensionen von Tolstoj selbst in die Nähe der *Ilias* gestellt, behauptet dieser Roman nicht nur seinen Rang als eines der herausragenden Werke der Weltliteratur, sondern er ist auch und vor allem eines: spannende Lektüre.

Bibliografische Angaben

Ersterscheinungstermin:
29.10.2007

Erscheinungstermin (aktuelle Auflage): 12.07.2021

Broschur, 2098 Seiten, Sprachen:
Deutsch

978-3-458-35007-1

Service

VLB-TIX

Teilen

Drucken

Downloads ▾

Zeichenvorrat: 1.073.741.823

Wörter: \approx 600.000

bei Ø 10 Buchstaben pro Wort

Zeichen: \approx 6 Mio.

ca. 178 Kopien

Datentypen, Numerische Werte

$\alpha\Omega$

Numerische Werte

DECIMAL

- Ganzzahlig
 - DECIMAL(p, 0)
- Dezimal
 - DECIMAL(p, s)

Alias: NUMERIC

Exakter num. Wert

- TINYINT ([T-SQL](#))
- SMALLINT
- INTEGER
- BIGINT

Ungefährer num. Wert

- FLOAT[(p)]
- REAL

Hinweis

- Der DECIMAL-Datentyp ist der numerische Datentyp und mit der größtmöglichen Genauigkeit, ein Zahlenkreisüberlauf.

Datentypen, DECIMAL (NUMERIC)

$\alpha\Omega$

DECIMAL(p, s)

- DECIMAL(p, s)
- ALIAS
- NUMERIC(p, s)

Oracle (PL/SQL)

- NUMBER(p, s)
- INTEGER \Rightarrow NUMBER(38, 0)

ganzahlig, Integer

- DECIMAL(p, 0)
- DECIMAL(3, 0) +/- 999
- DECIMAL(5, 0) +/- 99.999
- DECIMAL(10, 0) +/- 9.999.999.999

Vergleich zu INTEGER

- TINYINT (T-SQL) 0 bis 255
- SMALLINT -32.768
bis 32.767
- INTEGER -2.147.483.648
bis 2.147.483.647

dezimal, Nachkommastellen

- DECIMAL(p, s)
- DECIMAL(3, 2) +/- 9,99
- DECIMAL(5, 3) +/- 99,999
- DECIMAL(10, 9) +/- 9.999.999.999

Vergleich FLOAT/REAL

- alle 3 Beispiele sind durch FLOAT & REAL abbildbar
- DECIMAL stellt jedoch eine Echt-Wertspeicherung dar, FLOAT & REAL nicht

Datentypen, Numerische Werte

$\alpha\Omega$

DECIMAL(p, s)

- p, Precision = Genauigkeit
- s, Scale = Dezimalstellenwert
- p: ≤ 38
- **Exakter Wert wird gespeichert**

INTEGER

- exakt numerischer Wert
- **Exakter Wert wird gespeichert**

FLOAT / REAL

- ungefährer numerischer Wert
- **angenäherter Wert wird gespeichert, nicht der exakte Wert**
- Speicherung des Wertes als Exponentialwert

Precision	Storage
-----------	---------

1 – 9	5 Byte
10–19	9 Byte
20–28	13 Byte
29–38	17 Byte

Datentyp	Range	Storage
----------	-------	---------

Datentyp	Range	Storage
TINYINT	0 bis 255	1 Byte
SMALLINT	-32.768 bis 32.767	2 Byte
INTEGER	-2.147.483.648 bis 2.147.483.647	4 Byte
BIGINT	-9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807	8 Byte

Datentyp	Range	Storage
----------	-------	---------

FLOAT(n)	- 1,79E+308 bis -2,23E-308, 0 2,23E-308 bis 1,79E+308	4 Byte (n: 1- 24) 8 Byte (n:25-53)
REAL	- 3,40E + 38 bis -1,18E - 38, 0 1,18E-38 bis 3,40E + 38	4 Byte

Datentypen, DECIMAL & INTEGER

$\alpha\Omega$

Datentyp	Range	Storage
TINYINT	0 bis 255	1 Byte
SMALLINT	-32.768 bis 32.767	2 Byte
INTEGER	-2.147.483.648 bis 2.147.483.647	4 Byte
BIGINT	-9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807	8 Byte

Erstellen Sie basierend auf den Nummernbereichen der INTEGER-Datentypen die korrespondierenden DECIMAL-Datentypen, deren Nummernbereich sowie Speicherbedarf.

Datentyp	Range	Storage	DECIMAL	Range	Storage
TINYINT	0 bis 255	1 Byte	DECIMAL(3, 0)	-999 bis 999	5 Byte
SMALLINT	-32.768 bis 32.767	2 Byte	DECIMAL(5, 0)	-99,999 bis 99,999	5 Byte
INTEGER	-2,147,483,648 bis 2,147,483,647	4 Byte	DECIMAL(10, 0)	-9,999,999,999 bis 9,999,999,999	5 Byte
BIGINT	-9,223,372,036,854,775,808 bis 9,223,372,036,854,775,807	8 Byte	DECIMAL(19, 0)	-9,999,999,999,999,999,999 bis 9,999,999,999,999,999,999	9 Byte

Datentypen, BIT & BOOLEAN

$\alpha\Omega$

BIT

Exakter num. Wert

Ganzzahlig

- 0
- 1

Storage

- 1 Byte
(je 8 Bits)

Speicherorganisation

- Bits je Tabelle in Gruppen organisiert
- Gruppen zu je 8 Bits (= 1 Byte)
- Jede angefangene Gruppe reserviert ein neues Byte

Praxis

- Bits oftmals als Status-Flags verwendet (z.B. Lösch-Flag)
- Datentyp als **reines Status-Kennzeichen ungeeignet**,
da Status meistens **wachsende Ausprägungen** aufweisen

BOOLEAN

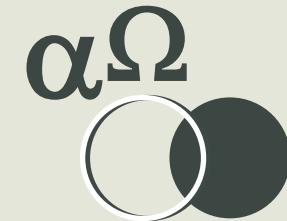
- Nicht in allen DBMS umgesetzt, da auf Bit verwiesen wird.
- T-SQL nicht vorhanden, PL/SQL vorhanden.

Datentypen, Spezialdatentypen

$\alpha\Omega$

Datentyp	Interner Datentyp	Beschreibung
• SQL_VARIANT	VARIANT	Spalten von Type SQL_VARIANT können unterschiedliche Datentypen als Inhalt haben (z.B. INT, DECIMAL, NVARCHAR)
• ROWVERSION	DECIMAL	Numerischer Wert mit der Versionsnummer für jede Tabellenzeile
• UNIQUEIDENTIFIER	Alphanumerisch	Universal Unique Identifier (UUID) ist eine 128-Bit-Zahl und dient als eindeutige ID über das gesamte DBMS (NEWID())
• HIERARCHYID	Alphanumerisch	Spezieller komprimierter Systemdatentyp zur Abbildung von Hierarchien
• TABLE	Struktur-Datentyp	Vereinfachte Datentyp zur Abbildung einer Tabelle, ist selbst keine vollständige Tabellenstruktur.
• CURSOR	Struktur-Datentyp	Cursor ist ein Abbild einer Tabelle ähnlich einer temporären Tabelle (z.B Rückgabetyp bei Funktionen)
• XML	Struktur-Datentyp	Inhalt muss ein gültiges wohlgeformtes XML-Dokument sein
• Räumliche Geometrietypen	Struktur-Datentyp	Spezialisierter Datentyp zur Behandlung von Geometrischen Formen wie z.B. Polygonen
• Räumliche Geografietypen	Struktur-Datentyp	Spezialisierter Datentyp zur Behandlung von geographischen Angaben wie z.B Geo-Koordinaten (Longitude (Länengrad), Latitude (Breitengrad))

Datentypen Sonderthemen



- Dateninhalt NULL
- Status-Zustands-Informationen

NULL, der Leerinhalt

Der „NULL“-Wert ist die Repräsentation eines **unbelegten** Spalteninhalts.

Eigenschaften

- NULL bedeutet keine Daten für Spalteninhalt
- Für NULL wird kein Speicher reserviert
- NULL kann nicht mit logischen Operatoren geprüft werden ($!=$, $<>$)
- NULL muss per IS NULL bzw. IS NOT NULL geprüft werden
- Operationen mit NULL selbst ergeben wieder NULL (ISO-Standard)

NULL, Besonderheiten



01: SELECT ('A' + NULL) AS string_null_concat;
02: SELECT (1 + NULL) AS number_null_add;

⇒ **Ausgabe 01:** NULL
⇒ **Ausgabe 02:** NULL

03: SELECT " AS string_empty;
04: SELECT ('A' || ") AS string_empty_concat;

⇒ **Ausgabe 03:**"
⇒ **Ausgabe 04:** A



01: SELECT ('A' || NULL) AS string_null_concat FROM dual;
02: SELECT (1 + NULL) AS number_null_add FROM dual;

⇒ **Ausgabe 01:** A
⇒ **Ausgabe 02:** NULL

03: SELECT " AS string_empty FROM dual;
04: SELECT ('A' || ") AS string_empty_concat FROM dual;

⇒ **Ausgabe 03:** NULL
⇒ **Ausgabe 04:** A

NULL, NULL-Prüfung & Ersetzung



01: **ISNULL(** <check value> , <replacement value>)
02: **CASE WHEN** <check value> **IS NULL**
 THEN <replacement value>
 ELSE <check value>
END;



01: **NVL(** <check value> , <replacement value>)
02: **NVL2(** <check value> , <NOT-NULL value> , <NULL value>)
03: **NULLIF(** <check value> , <replacement value>)
 ⇒ **NULL**, ist als statischer Wert zulässig
04: **CASE WHEN** <check value> **IS NULL**
 THEN <replacement value>
 ELSE <check value>
END

Datenmodellierung, „Status“-Informationen



Welcher ist der korrekte Datentyp für die Ausprägung eines Status bzw. Prozessstatus?

Fallbeispiel 01: Für jeder Person soll der Familienstand „verheiratet“ verfügbar sein

Datentyp	BIT	Zustände	INTEGER	VARCHAR(3)
	<ul style="list-style-type: none">• Spalte: Verheiratet• 0: nicht verheiratet• 1: verheiratet• Was besagt der Wert NULL ?	<ul style="list-style-type: none">• Ledig• Verheiratet• Geschieden• Eingetragene Lebensgemeinschaft• Unbekannt (Information ausstehend)	<ul style="list-style-type: none">• 1• 2• 3• 4• 0	<ul style="list-style-type: none">• LDG• VHT• GSD• ELG• UNK

NULL = Verarbeitungsfehler

4 Byte

3 Byte

Datenmodellierung, „Status“-Informationen



Für welchen möglichen Prozess stehen die folgenden Wertausprägungen?

Fallbeispiel 02: Prozessierung eines Import-Prozesses und dessen Verarbeitung zu Anwendungsdaten „ETL-Prozess“.

Status-Werte

Ausprägungen

- -1
- 0
- 1
- 2
- 1000
- 1001
- 9000
- 9001

Standardverarbeitung & Standardprozesssituation

Validierungsprozess mit Sonderergebnis

Validierungsprozess mit Fehlerergebnis

Zustände

- Daten importiert
- Daten in Verarbeitung
- Daten in Zieltabelle übertragen
- Ergänzungsprozess durchgeführt
- Identische Daten bereits vorhanden
- Aktuellere Daten vorhanden
- Fehlende Daten, keine Übernahme
- Inhaltliche Fehler, keine Übernahme



[1, 1000)



[1000, 9000)

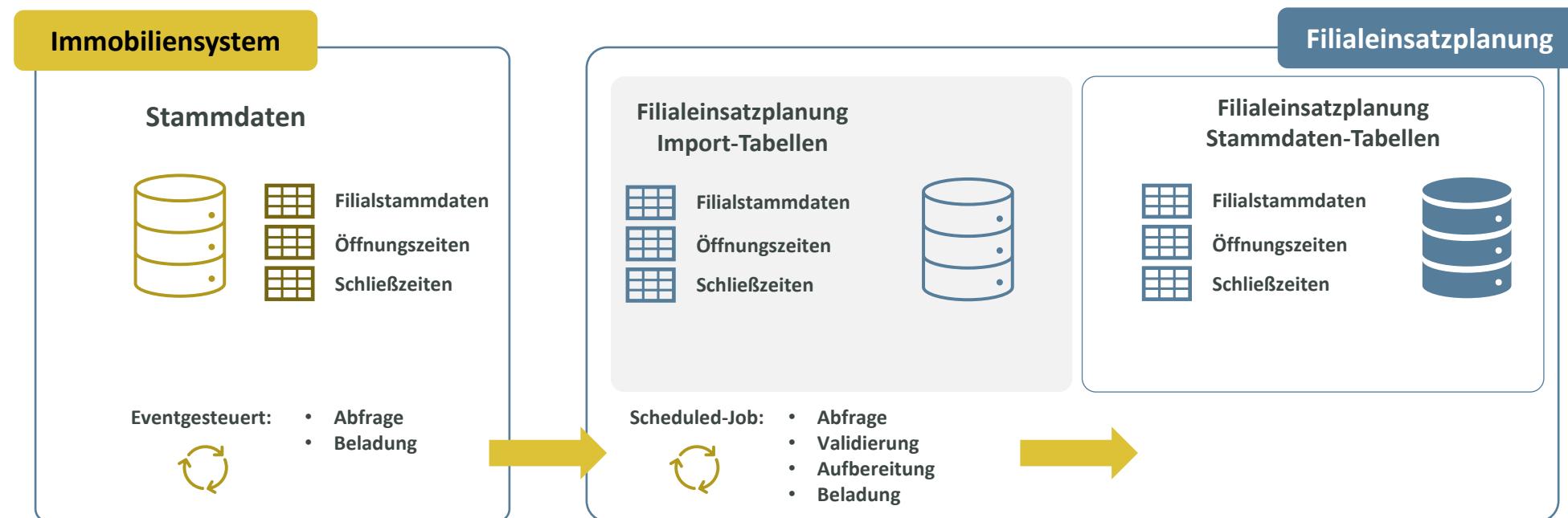


[9000, 10000)

Datenmodellierung, „Status“-Informationen



Der Extract-Transfer-Load-Prozess (ETL) von Stammdaten, am Beispiel von „Filialstammdaten“



Datenmodellierung, „Status“-Informationen



Welcher ist der korrekte Datentyp für die Ausprägung eines Status bzw. Prozessstatus?

Fallbeispiel 01: Für jeder Person soll der Familienstand „verheiratet“ verfügbar sein

Datentyp

BIT

- Spalte: **Verheiratet**
- 0: nicht verheiratet
- 1: verheiratet
- Was besagt der Wert NULL ?

1 Byte

Zustände

- Ledig
- Verheiratet
- Geschieden
- Eingetragene Lebensgemeinschaft
- Unbekannt
(Information ausstehend)



Compliance



- Datenschutz
- Informationssicherheit
- Governance / IT-Governance
- Compliance

Compliance, Datenschutz „EU-DSGVO“



DSGVO - Datenschutzgrundverordnung

EU-Gesetzgebung zur Regelung der Handhabung & Verarbeitung von personenbezogenen Daten



4.5.2016 DE Amtsblatt der Europäischen Union L 119/1

I
(Gesetzgebungsakte)

VERORDNUNGEN

VERORDNUNG (EU) 2016/679 DES EUROPÄISCHEN PARLAMENTS UND DES RATES
vom 27. April 2016
zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten, zum freien
Datenverkehr und zur Aufhebung der Richtlinie 95/46/EG (Datenschutz-Grundverordnung)
(Text von Bedeutung für den EWR)

DAS EUROPÄISCHE PARLAMENT UND DER RAT DER EUROPÄISCHEN UNION —

Compliance, Datenschutz „EU-DSGVO“



DSGVO - Datenschutzgrundverordnung

Schutz personenbezogener Daten als Grundrecht



gemäß dem ordentlichen Gesetzgebungsverfahren (¹),

in Erwägung nachstehender Gründe:

- (1) Der Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten ist ein Grundrecht. Gemäß Artikel 8 Absatz 1 der Charta der Grundrechte der Europäischen Union (im Folgenden „Charta“) sowie Artikel 16 Absatz 1 des Vertrags über die Arbeitsweise der Europäischen Union (AEUV) hat jede Person das Recht auf Schutz der sie betreffenden personenbezogenen Daten.
- (2) Die Grundsätze und Vorschriften zum Schutz natürlicher Personen bei der Verarbeitung ihrer personenbezogenen Daten sollten gewährleisten, dass ihre Grundrechte und Grundfreiheiten und insbesondere ihr Recht auf Schutz personenbezogener Daten ungeachtet ihrer Staatsangehörigkeit oder ihres Aufenthaltsorts gewahrt bleiben. Diese Verordnung soll zur Vollendung eines Raums der Freiheit, der Sicherheit und des Rechts und einer Wirtschaftsunion, zum wirtschaftlichen und sozialen Fortschritt, zur Stärkung und zum Zusammenwachsen der Volkswirtschaften innerhalb des Binnenmarkts sowie zum Wohlergehen natürlicher Personen beitragen.
- (3) Zweck der Richtlinie 95/46/EG des Europäischen Parlaments und des Rates (²) ist die Harmonisierung der Vorschriften zum Schutz der Grundrechte und Grundfreiheiten natürlicher Personen bei der Datenverarbeitung sowie die Gewährleistung des freien Verkehrs personenbezogener Daten zwischen den Mitgliedstaaten.

Compliance, Datenschutz „EU-DSGVO“



DSGVO - Datenschutzgrundverordnung

Grundrechte sind das höchst schützenswerteste Gut



- 25. Mai 2018 in der gesamten Europäischen Union (EU) Geltung
- Ersetzung der allgemeinen Datenschutz-Richtlinie 95/46/EG

Personenbezogenen Daten

- Alle Daten und Informationen, welche direkt einer natürlichen Person zugeordnet werden können.
- Dabei gibt kommt bestimmten Informationen ein besonderer Schutz zu wie dem Stand der Lebenssituation, der ethnischen/religiösen Zugehörigkeit, jegliche Informationen über den Gesundheitszustand einer Person sowie sexueller Orientierung.

Compliance, Datenschutz „EU-DSGVO“



DSGVO - Datenschutzgrundverordnung

Grundrechte sind das höchst schützenswerteste Gut



- 25. Mai 2018 in der gesamten Europäischen Union (EU) Geltung
- Ersetzung der allgemeinen Datenschutz-Richtlinie 95/46/EG

Personenbeziehbare Daten

- Daten und Informationen, welche nicht direkt auf eine natürliche Person schließen lassen, aber Referenzen besitzen mit der eine Person über Zuordnungseinheiten zugeordnet werden können.
- Referenzen können sein:
technische IDs, Benutzernamen, Sitzungsnummer, IP-Adressen

Compliance, Datenschutz „EU-DSGVO“



DSGVO - Datenschutzgrundverordnung

Verstöße können mit erheblichen Strafen geahndet werden



Mögliche Strafmaß

- Personen: Bußgeld bis zu einer Höhe von 20 Millionen EUR
- Unternehmen: bis zu 4 % des gesamten weltweit erzielten Jahresumsatzes des vorangegangenen Geschäftsjahres

Compliance, Datenschutz „EU-DSGVO“



DSGVO - Datenschutzgrundverordnung

Zur Speicherung von personenbezogenen oder beziehbaren Daten sind gesonderte Maßnahmen zu ergreifen.



Maßnahmen (Auszug)

- Alle Geschäftsprozesse, in denen für Geschäftsvorfälle Personendaten verarbeitet werden sind ausnahmslos in einem jeweiligen **Verarbeitungsverzeichnis** zu beschreiben und zu dokumentieren.
- Der **Zugriff** auf diese Daten ist **streng zu regulieren** und nur für prozessrelevante Teilnehmer zulässig.
- Ein Allgemeiner Zugriff auf diese Informationen ist unzulässig.
- Ein Allgemeiner Weitergabe der Informationen ist unzulässig.
- **Vorfälle** gegen die Datenschutzgrundverordnung sind umgehend der zuständigen **Aufsichtsbehörde zu melden** (Artikel 33 DSGVO).

Compliance, Datenschutzvorfälle, Beispiele



Datenschutz
Irland verhängt 225 Millionen Euro Bußgeld gegen WhatsApp
Die Facebook-Tochter hat nicht klar genug gemacht, welche Daten zu welchem Zweck verarbeitet werden. Das ursprüngliche Bußgeld dafür hat Irlands Datenschutzhörde nun deutlich erhöht – nicht ganz freiwillig.
02.09.2023, 13:54 Uhr

Amazon bestreitet Verstöße gegen EU-Datenschutzverordnung

Der weltgrößte Online-Händler Amazon wirft den Vorwurf zurück, gegen das EU-Datenschutzrecht verstoßen zu haben. Dabei geht es um eine Geldbuße in Höhe von 746 Millionen Euro.



Facebook-Mutter
265 Millionen Euro Strafe für Meta
Stand: 28.11.2022 17:22 Uhr

Weil Daten von Facebook- und Instagram-Nutzern in großem Stil online zugänglich waren, soll der Meta-Konzern 265 Millionen Euro Strafe in Irland zahlen. Schon mehrfach haben die Behörden Datenschutz-Verstöße bemängelt.

Verfahren wegen Datenschutzverstoss
WhatsApp soll 5,5 Millionen Euro Bußgeld bezahlen
Weil WhatsApp seine Nutzer zur Datenfreigabe verpflichten wollte, soll der Messenger-Anbieter nun eine vergleichsweise geringe Strafe zahlen. Der Beschwerdeführer reagiert empört.

19.01.2023, 16:26 Uhr
Artikel zum Hören • 3 Min
Anhören

Länder verheimlichen Rechtsgutachten zu Microsoft 365
DATENSCHUTZ
Kann die Landes-IT Microsoft 365 datenschutzkonform einsetzen? Die Antwort darauf steht in einem Rechtsgutachten. Golem.de wollte es einsehen, aber die Länder halten es lieber geheim.
Von Moritz Tremmel und Friedhelm Greis
8. Mai 2023, 12:30 Uhr

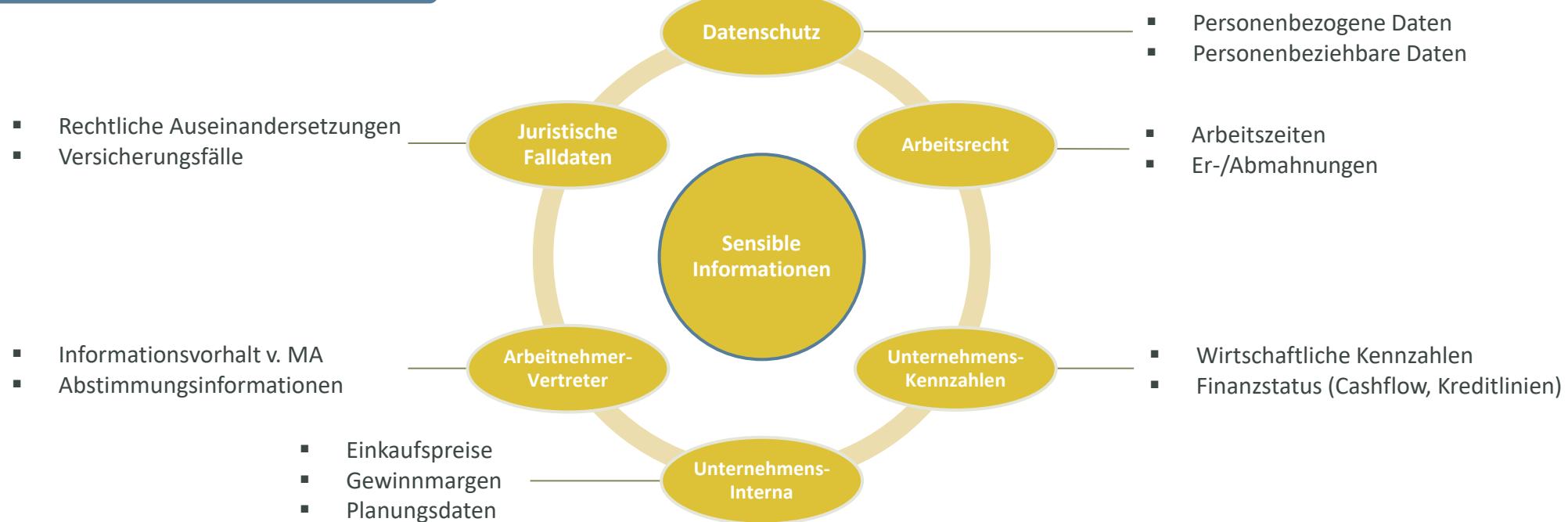
TOP SECRET

Compliance, Informationssicherheit



Vorgabe von technischen Prozessen und Verfahren zur Sicherung sensibler Informationen

Sensible Informationen



Compliance, Informationssicherheit



Vorgabe von technischen Prozessen und Verfahren zur Sicherung sensibler Informationen

Technische Prozesse

- Kommunikationsverfahren
- Verschlüsselungsverfahren
- Infrastruktur-Vorgaben
- Cyber-Security
- Access-Management
- Zutritts-Management

Technische Verfahren

- Datenbank-Encryption
(TDE-Verschlüsselung)
- Email-TLS-Verschlüsselung
- Vertrauenswürdige Zertifikatsstellen
- Zulässige Cipher
(mind. 128Bit AES)
- Logging von Systemzugriffen
sowie Systemaktivitäten

Compliance, IT-Governance & Governance



Die ganzheitliche Erstellung des Regelwerkes für die IT erfolgt im Rahmen der Definition der IT-Governance

IT-Governance

- Regelwerk für alle IT-Prozesse in einem Unternehmen
- Vollständige Abbildung der IT-Prozesse
- Zuordnung der regulatorischen Vorgaben je Prozess
- Vorgabe der Dokumentationspflichten (Logging)
- Definition der Monitorings & Reviews

Unternehmens-Governance



Compliance, die unternehmerische Klammer



Compliance dient der unternehmerischen Definition, Bewertung und Kontrolle aller Prozesse.



Compliance, die unternehmerische Klammer



Compliance dient der unternehmerischen Definition, Bewertung und Kontrolle aller Prozesse.

■ Compliance-Risiken

- Datenschutz
- IT-Sicherheit
- Bekämpfung von Geldwäsche und Terrorismusfinanzierung
- Durchsetzung von Embargos sowie Sanktionslisten
- Korruption/Betrug und Umgang mit Interessenkonflikten
- Meldepflichten von Mitarbeitenden bezüglich Eigenhandel (Insider-Handel)
- Kartellrecht und Marktmanipulation
- Arbeitsrecht
- Exportkontrolle

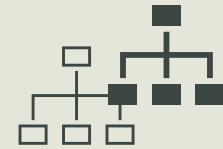
Compliance, die unternehmerische Klammer



Compliance Management, am Beispiel der „WACKER Chemie AG“



SQL-Familie & Indexierung



Übersicht



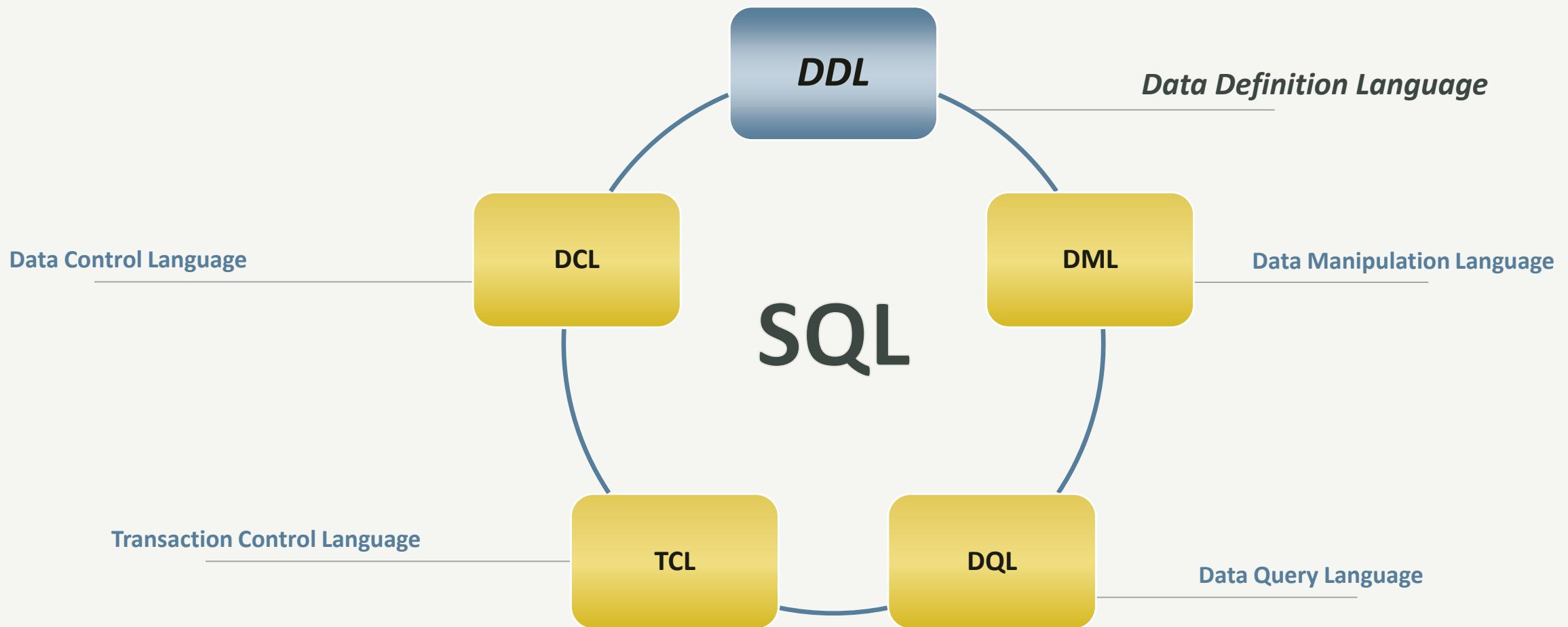
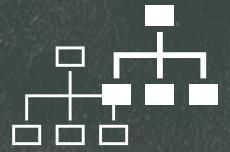
SQL, Befehle der SQL-Familie

- DQL, SQL-Abfrage
- DDL, Tabellenerstellung, Änderung, Löschen
- DML, Datenerstellung, Änderung, Löschen
- DTL, Transaktionssicherung,
COMMIT & ROLLBACK
- DCL, Berechtigungskontrolle,
GRANT & REVOKE

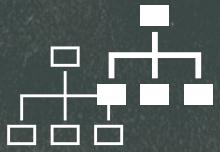
Indexierung

- Index-Verwendung
- Definition, Index als Datenstruktur
- Indexierung, Index-Baum
- Indexsuchen, wann und wie wirkt der Index
- Zusammenfassung
- Hinweise zur Indexverwendung

SQL – DDL, Data Definition Language



DDL, Data Definition Language



■ DDL

- Erstellen von Tabellen **CREATE TABLE**
- Anpassen von Tabellen **ALTER TABLE**
- Löschen von Tabellen **DROP TABLE**

CREATE TABLE

white	yellow	blue
white	yellow	blue
white	yellow	blue

AT: ADD column

white	yellow	blue	green
white	yellow	blue	green
white	yellow	blue	green

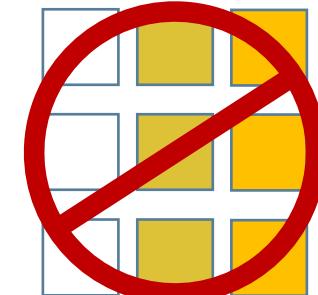
AT: ALTER column

white	yellow	blue	yellow
white	yellow	blue	yellow
white	yellow	blue	yellow

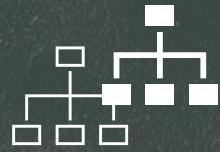
AT: DROP column

white	yellow	yellow
white	yellow	yellow
white	yellow	yellow

DROP TABLE



DDL, Data Definition Language, Befehle



CREATE

```
CREATE TABLE <table name> (
    <column name 1>      <data type>      [<null, NOT null>] [<constraint>] [<default>]
    , <column name 2>      <data type>      [<null, NOT null>] [<constraint>] [<default>]
    , <column name ...>   <data type>      [<null, NOT null>] [<constraint>] [<default>]
    , <column name N>     <data type>      [<null, NOT null>] [<constraint>] [<default>]
);
GO
```

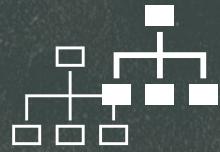
ALTER

```
ALTER TABLE <table name> ADD <column name> <data type>;
ALTER TABLE <table name> ALTER COLUMN <column name> <data type>;
ALTER TABLE <table name> DROP COLUMN <column name>;
```

DROP

```
DROP TABLE <table name>;
```

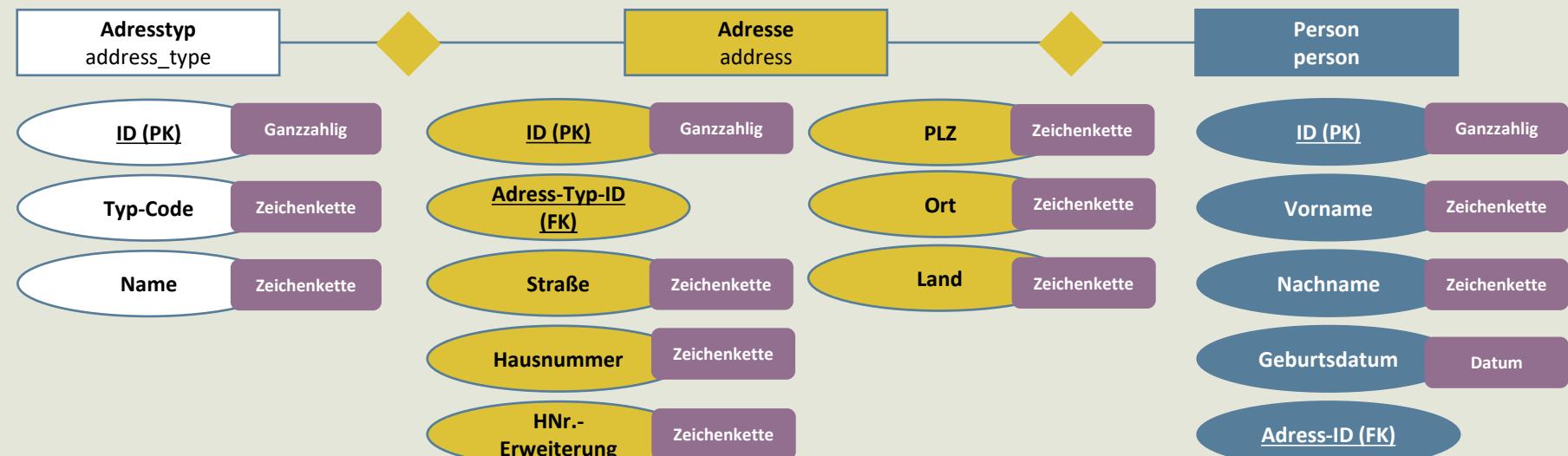
DDL, Data Definition Language, Aufgabe



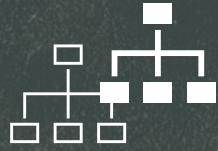
A

Aufgabe

Erstellen Sie je eine Tabelle für das folgende ERD. Jede Tabelle soll ein Erstellungs-datums und ein Datum der letzten Änderung aufweisen mit dem Default der Systemzeit.

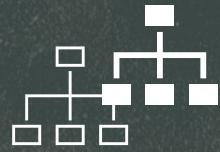


DDL, Data Definition Language, Tabellen 1



Typ	CREATE TABLE address_type (
	id	DECIMAL(2,0)	PRIMARY KEY
	, type_code	NVARCHAR(10)	
	, type_description	NVARCHAR(100)	NOT NULL
	, created_date	DATETIME2	DEFAULT SYSDATETIME()
	, changed_date	DATETIME2	DEFAULT SYSDATETIME()
);		
Adresse	CREATE TABLE address (
	id	DECIMAL(10, 0)	IDENTITY(1, 1) PRIMARY KEY
	, address_type_id	DECIMAL(2,0)	FOREIGN KEY REFERENCES address_type(id)
	, street	NVARCHAR(400)	NOT NULL
	, street_number	NVARCHAR(50)	
	, street_number_extension	NVARCHAR(50)	NULL
	, postal_code	NVARCHAR(50)	
	, city	NVARCHAR(100)	
	, country	NVARCHAR(100)	NOT NULL
	, created_date	DATETIME2 DEFAULT SYSDATETIME()	
	, changed_date	DATETIME2 DEFAULT SYSDATETIME()	
);		

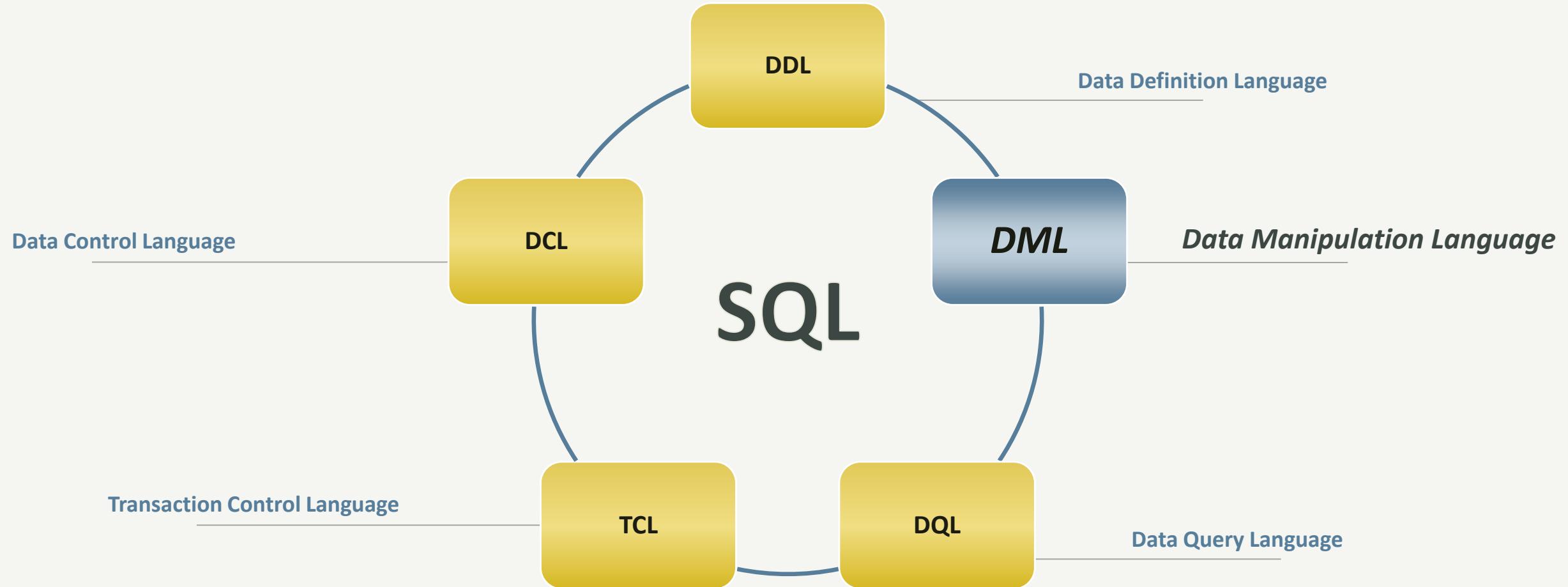
DDL, Data Definition Language, Tabellen 2



Person

```
CREATE TABLE person (
    id             DECIMAL(10,0)      PRIMARY KEY IDENTITY(1, 1)
    , first_name   NVARCHAR(100)     NOT NULL
    , last_name    NVARCHAR(100)     NOT NULL
    , date_of_birth DATE            NULL
    , address_id   DECIMAL(10,0)      FOREIGN KEY address(id)
    , created_date DATETIME2        DEFAULT SYSDATETIME()
    , changed_date DATETIME2        DEFAULT SYSDATETIME()
);
GO
```

SQL – DML, Data Manipulation Language

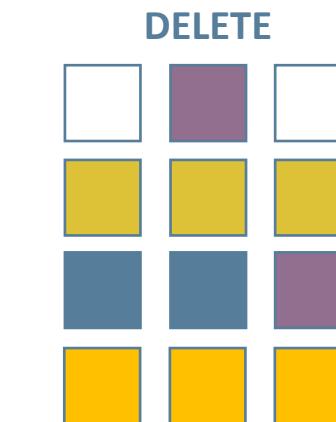
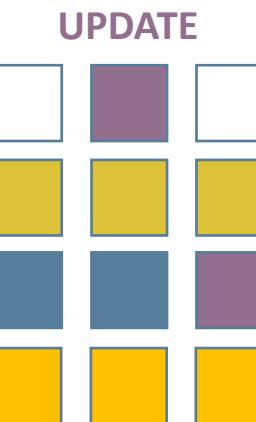
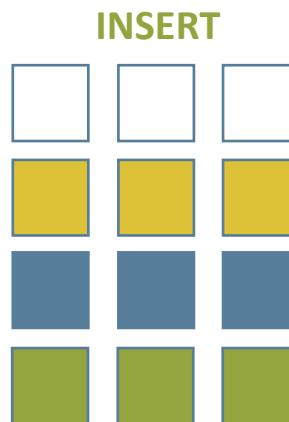


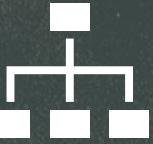
DML, Data Manipulation Language



- **DML**

- **Einfügen von Inhalten** **INSERT INTO**
- **Ändern von Inhalten** **UPDATE**
- **Löschen von Tabellen** **DELETE FROM**





DML, Data Manipulation Language, Befehle

INSERT

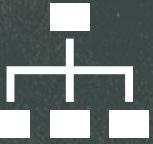
```
INSERT INTO <table name>
[(<column name 1>, <column name ...>, <column name N>
 )]
VALUES
[(<column name 1>, <column name ...>, <column name N>
 )]
;
```

UPDATE

```
UPDATE <table name>
SET <column name> = <value>
[ WHERE <condition>];
```

DELETE

```
DELETE FROM TABLE <table name>
[ WHERE <condition> ];
```



DML, Data Manipulation Language, INSERT

INSERT, Typ 1

```
INSERT INTO person VALUES ('Mustermann', 'Paula');
```

- Tabelle muss **identische** Spalten haben wie die Werteliste

```
INSERT INTO person (lastname, first name) VALUES ('Mustermann', 'Paula');
```

- **Single INSERTs:** Inhalte werden einzeln eingefügt

INSERT, Typ 2a

```
INSERT INTO person (lastname, first name)  
VALUES      ( 'Mustermann', 'Paula' )  
            , ( 'Tester', 'Paul' )  
            , ( 'Doe', 'John' );
```

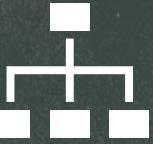
- **max. 1.000** VALUE-Listen

INSERT, Typ 2b

```
INSERT INTO person (lastname, first name)  
SELECT lastname, firstname FROM employee WHERE lastname like 'M%';
```

- **SELECT-FOR-INSERT**

- **Multiple INSERTs:** Inhalte werden in Form eines Batches eingefügt



DML, Data Manipulation Language, UPDATE

UPDATE, Typ 1

```
UPDATE person  
SET lastname = 'Müller'  
WHERE lastname = 'Mustermann';
```

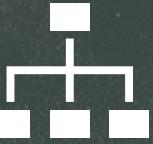
```
UPDATE person  
SET lastname = 'Müller', firstname = 'Claudia'  
WHERE lastname = 'Mustermann';
```

- Single UPDATE: Inhalte werden einzeln je Datensatz aktualisiert

UPDATE, Typ 2

```
UPDATE pers  
SET pers.ref_number = emp.ref_number  
FROM person pers  
JOIN employee emp ON pers.e,p_id = emp_id  
WHERE lastname = 'Mustermann';
```

- Multiple UPDATES: Inhalte werden in Form eines Batches aktualisiert



DML, Data Manipulation Language, DELETE

DELETE, Typ 1

```
DELETE    FROM    person  
WHERE    lastname = 'Mustermann';  
  
DELETE    FROM    person;
```

- **DELETE:** Bedingte Löschung & Löschung der gesamten Tabelle

DELETE, Typ 2

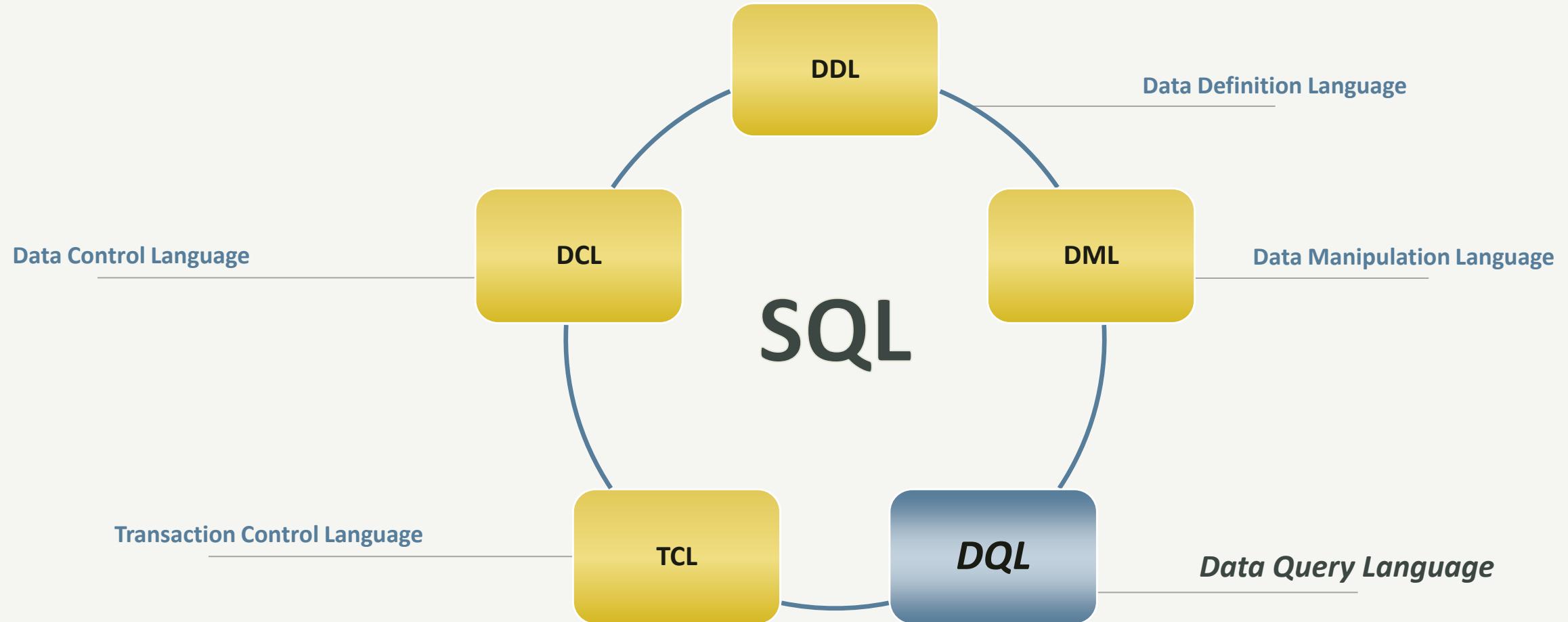
```
DELETE    FROM    person  
WHERE    person.emp_id IN (  
            SELECT id FROM employee  
        );
```

- **DELETE:** Bedingte Löschung mit einer Sub-Query

Hinweis

Löschen auf Basis von IDs, Vermeidung von BETWEEN-/Bereichs-Löschungen

SQL – DQL, Data Query Language



„SQL“, Data Query Language



- **SQL**

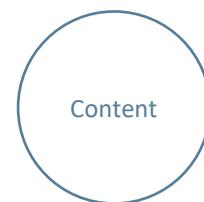
- Daten abfragen
- Daten verknüpfen
- Daten verschmelzen

SELECT

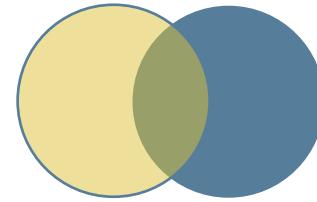
JOIN

UNION [INTERSECT, EXCEPT/MINUS]

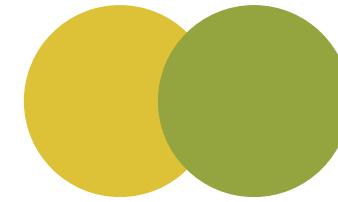
SELECT



JOIN



UNION



„SQL“, Data Query Language, SELECT



SELECT

```
SELECT
    <column name 1>          AS <alias cname 1>
    , <column name ...>      AS <alias cname ...>
    , <column name N>         AS <alias cname N>

FROM
    <table name> <alias tname>

WHERE
    <condition>
    [AND | OR] <condition>

GROUP BY
    <group column ...>

HAVING
    <group condition>
    [AND | OR] <group condition>

ORDER BY
    <order column ...> [ASC | DESC]
;
```

CONDITION

WHERE, HAVING, WHEN, IIF()

```
[<column name> | <value>]
<logical operator>
[<value> | <column name>]
```

„SQL“, Data Query Language, Operatoren & Funktionen



Operatoren

Logische Operatoren

- `=, !=, <>, >, >=, <, <=`
- `LIKE, IN, BETWEEN, EXISTS, IS NULL`
- **Negation:** `NOT`

Mathematische Operatoren

- `+, -, *, /, %, ()`

Funktionen

Zeilenorientierte Funktionen

Mathematische Funktionen

Zeichenketten-Funktionen

Gruppenfunktionen

Mathematische Funktionen

Zeichenketten-Funktionen

Bedingung: GROUP BY

„SQL“, Data Query Language, Zeilenorientierte Funktionen



Zeilenorient. Funktionen

Mathematische Funktionen

- FLOOR() abrunden
- CEIL() aufrunden
- ROUND() kaufmännisches Runden
- ABS() absoluter Wert
- **SQL Server 2022:**
- GREATEST() größter Wert einer Liste
- LEAST() kleinster Wert einer Liste

Zeichenketten-Funktionen

- LEN() Länge der Zeichenkette
- DATALENGTH() Byte-Länge der Zeichenkette
- SUBSTRING() Teilzeichenkette
- UPPER() Zeichenkette in Großbuchstaben
- LOWER() Zeichenkette in Kleinbuchstaben
- CHARINDEX() Suche nach einem Zeichen
- REPLACE() Textersetzung
- TRIM() Leerzeichen entfernen
- LTRIM() Leerzeichen, links entfernen
- RTRIM() Leerzeichen, rechts entfernen
- REPLICATE() Wiederholung eines Zeichens
- REVERSE() Umkehrung einer Zeichenkette

„SQL“, Data Query Language, Gruppen-Funktionen



Gruppen-Funktionen

Mathematische Funktionen

- COUNT() Zählung (Rückgabe INTEGER)
- COUNT_BIG() Zählung (Rückgabe BIGINT)
- AVG() Durchschnitt
- SUM() Summe
- MAX() Maximum
- MIN() Minimum
- STDEV() statistische Standardabweichung
- VAR() statistische Varianz

Zeichenketten-Funktionen

- STRING_AGG() Zeichenketten der Zellinhalte verknüpfen

Hinweis

- Die Aggregatsfunktionen stehen nur bei einem vorhandenem GROUP BY-Ausdruck zur Verfügung.

„SQL“, Data Query Language, DISTINCT



In **DISTINCT** dient der Entfernung von **doppelten Datensätzen** aus einem **Abfrageergebnis**.
Dabei müssen die Datensätze vollständig identisch sein

```
SELECT  
    DISTINCT  
    < column name 1 >  
    , < column name ... >  
    , < column name N >  
FROM  
    <table name>  
;
```

Distinct

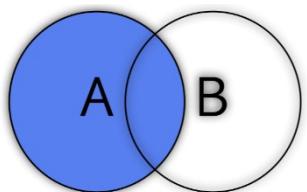
```
SELECT  
    DISTINCT  
    lastname  
    , firstname  
FROM  
    person  
;
```

Group By

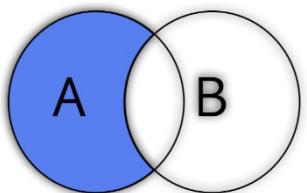
```
SELECT  
    lastname  
    , firstname  
FROM  
    person  
GROUP BY  
    lastname  
    , firstname  
;
```

- Einsatz von DISTINCT direkt im SELECT-Statement
- Einsatz von DISTINCT in Aggregatsfunktionen möglich
- **2 Fragen:** Handelt es sich um echt Duplikate? Worin sind diese begründet?

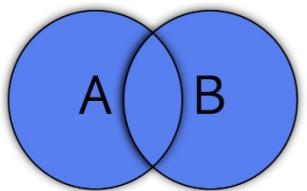
„SQL“, Data Query Language, JOIN



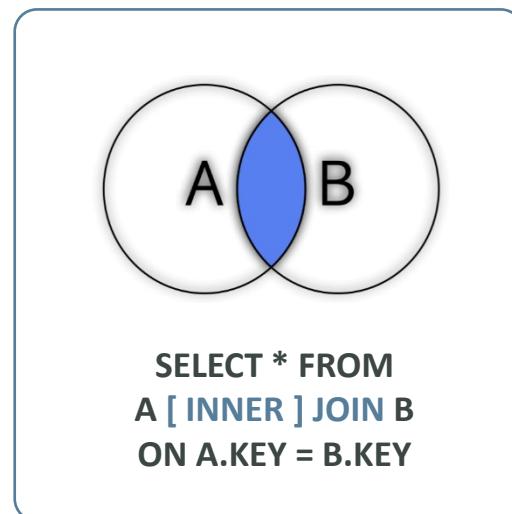
```
SELECT * FROM  
A LEFT JOIN B  
ON A.KEY = B.KEY
```



```
SELECT * FROM  
A LEFT JOIN B  
ON A.KEY = B.KEY  
WHERE B.KEY IS NULL
```

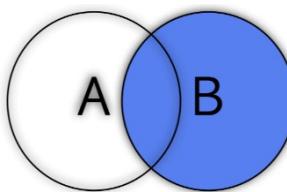


```
SELECT * FROM  
A FULL JOIN B  
ON A.KEY = B.KEY
```

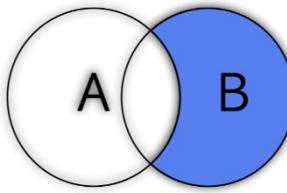


```
SELECT * FROM  
A [ INNER ] JOIN B  
ON A.KEY = B.KEY
```

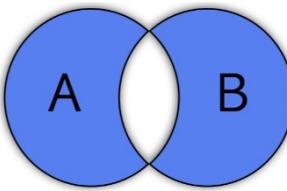
```
SELECT * FROM  
A RIGHT JOIN B ON A.KEY = B.KEY
```



```
SELECT * FROM  
A RIGHT JOIN B ON A.KEY = B.KEY  
WHERE A.KEY IS NULL
```



```
SELECT * FROM  
A FULL JOIN B ON A.KEY = B.KEY  
WHERE A.KEY IS NULL  
OR B.KEY IS NULL
```



Hinweis

NATURAL JOIN ist möglich, jedoch aufgrund seiner Fehleranfälligkeit hier nicht näher betrachtet

„SQL“, Data Query Language, JOIN-Aufbau



JOIN

SELECT

```
<column name 1>          AS <alias cname 1>
, <column name ...>      AS <alias cname ...>
, <column name N>         AS <alias cname N>
```

FROM

```
<table name 1> <alias tname 1>
[[INNER | LEFT | RIGHT | FULL OUTER] JOIN
<table name 2> <alias tname 2>
ON [<table name 1> | <alias tname 1>].<column name>
= [<table name 2> | <alias tname 2>].<column name>
```

WHERE

```
<condition>
```

```
;
```

„SQL“, Data Query Language, JOIN, Beispiel



JOIN

```
SELECT
    per.first_name
    , per.last_name
    , adr.street
    , adr.street_number      + adr.street_number_extension
    , adr.postal_code        + adr.city
    , adt.type_code
    , adt.type_description
AS per_first_name
AS per_last_name
AS adr_street
AS adr_street_number_full
AS adr_city_full
AS adt_type_code
AS adt_type_description

FROM
    person          per
    JOIN address      adr ON adr.id = per.address_id
    JOIN address_type  adt ON adt.id = adr.address_type_id

WHERE
    adt.type_code = 'invoice'
;
```

„SQL“, Data Query Language, UNION



UNION

„Vereinigungsmengen“ existieren in 3 Grundausprägungen

UNION ALL



- Darstellung **aller Werte** aus beiden Mengen

UNION ALL vs. UNION

- „**UNION ALL**“ fügt alle vorhandenen Datensätze zusammen
- „**UNION**“ Datensätze mit identischem Inhalt werden entfernt, damit erfolgt eine Änderung der Grundgesamtheit

INTERSECT



- Darstellung **ausschließlich der übereinstimmenden** Werte

EXCEPT (alias MINUS)



- Stellt die Werte der **linken Menge dar, die nicht** mit der rechten Menge übereinstimmen

„SQL“, Data Query Language, UNION



UNION

```
SELECT
    <column name 1> AS <alias cname 1>, <column name ...> AS <alias cname ...>, <column name N> AS <alias cname N>
FROM
    <table name 1>
UNION ALL | INTERSECT | MINUS/EXCEPT
SELECT
    <column name 1> AS <alias cname 1>, <column name ...> AS <alias cname ...>, <column name N> AS <alias cname N>
FROM
    <table name 2>
;
```

Beispiel

```
SELECT 'de' AS language, @text_char_de AS text_char, @text_nchar_de AS text_nchar
UNION ALL
SELECT 'gr' AS language, @text_char_de AS text_char, @text_nchar_gr AS text_nchar
;
```

„SQL“, Data Query Language, SUBQUERY

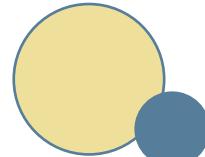


Subquery

Unterabfragen als ergänzende Informationsquelle

Subquery, Typ 1

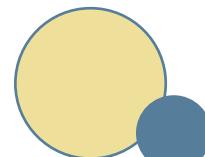
JOIN



- Unterabfrage als Bestandteil des JOIN-Ausdrucks

Subquery, Typ 2

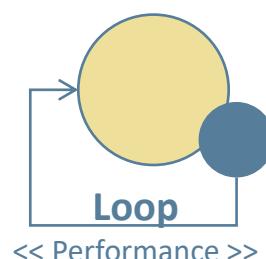
WHERE



- Unterabfrage als Bestandteil des WHERE-Ausdrucks

Subquery, Typ 3

COLUMN



- Unterabfrage als Spaltenausdruck verknüpft mit der Hauptabfrage als Filter-Loop

„SQL“, Data Query Language, SUBQUERY



Subquery, Typ 1

```
SELECT
*
FROM
address
JOIN (
    SELECT
        id AS adt_id
    FROM
        address_type
    WHERE type_code = 'main'
) adt
ON adt.adt_id = address.address_type_id
;
```

„SQL“, Data Query Language, SUBQUERY



Subquery, Typ 2

```
SELECT
*
FROM
address
WHERE
address_type_id IN ( SELECT id FROM address_type )
;
```

Subquery, Typ 3

```
SELECT
( SELECT type_description FROM address_type
  WHERE id = address.address_type_id )           AS adr_type_description_1

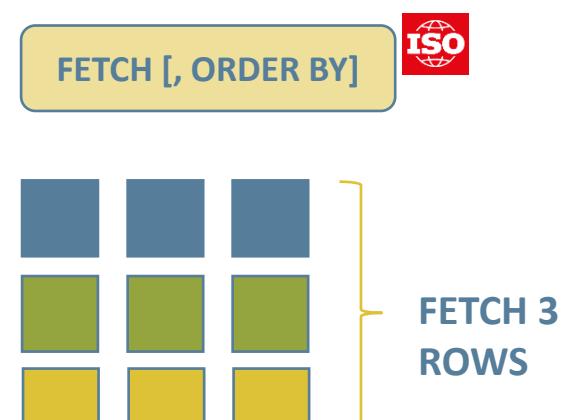
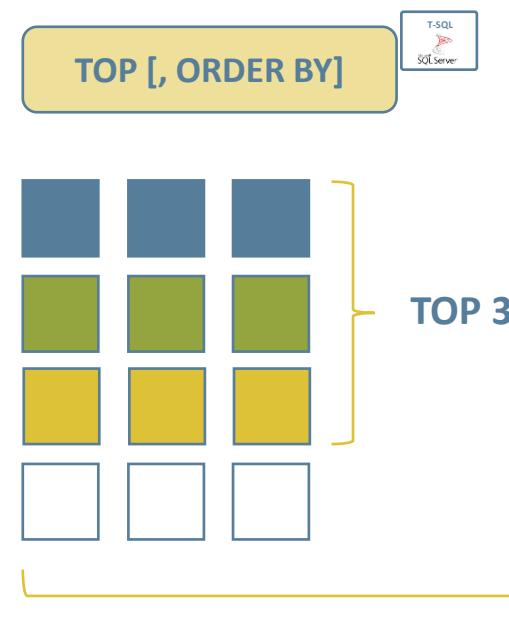
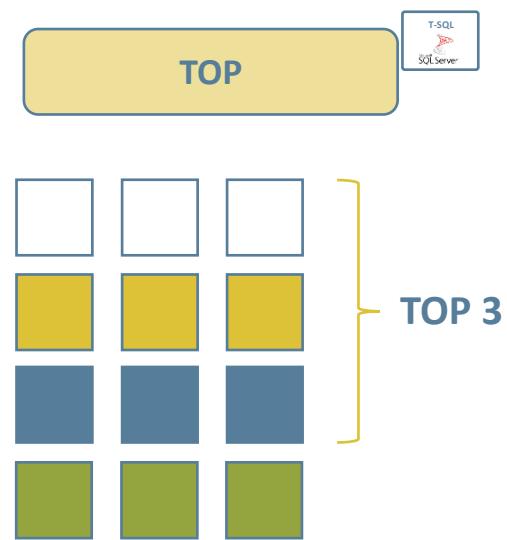
, ( SELECT MAX(type_description) FROM address_type ) AS adr_type_description_2
FROM
address
;
```

„SQL“, Data Query Language, TOP & FETCH



SUB-Mengen

Teilmengen durch die Bereichseinschränkung TOP & FETCH



- **TOP N** gibt beginnend ab Zeile 1 N-Zeile zurück

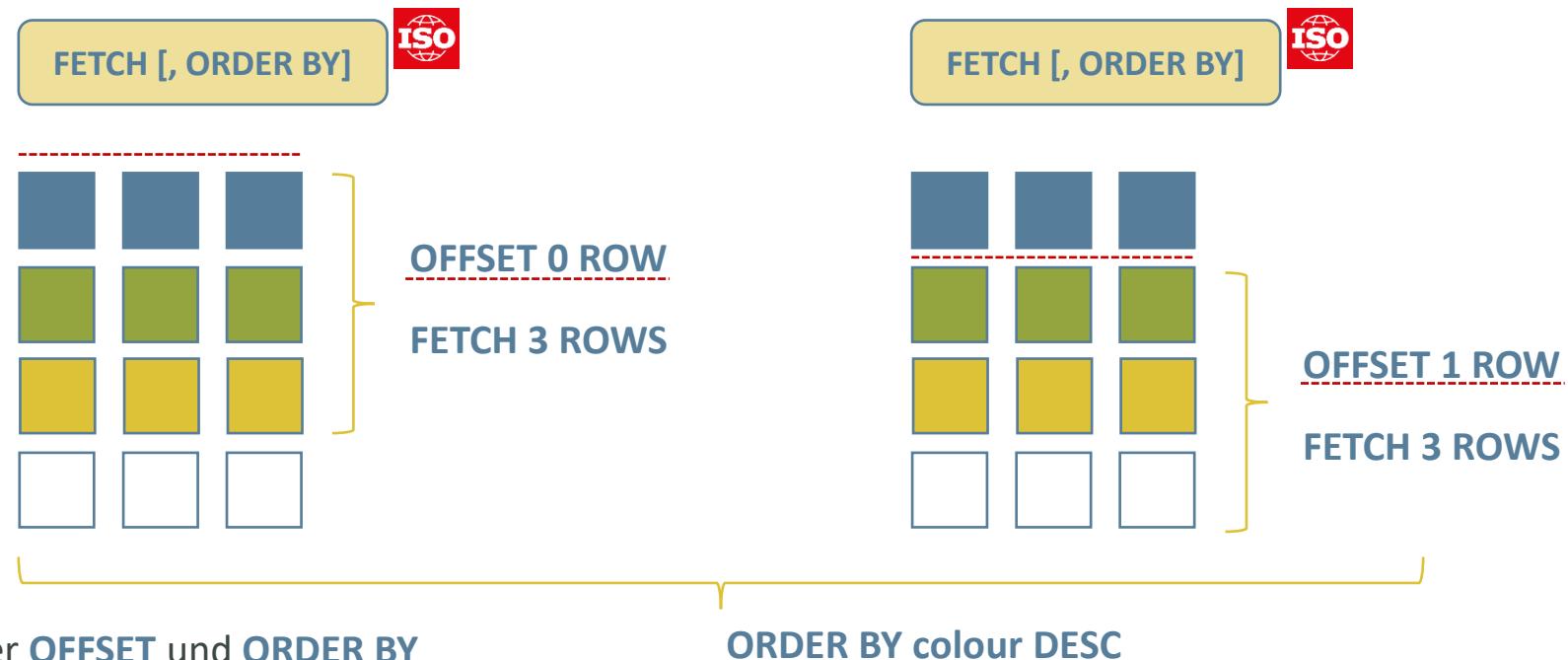
ORDER BY colour DESC

„SQL“, Data Query Language, TOP & FETCH



SUB-Mengen

Teilmengen durch FETCH mit OFFSET



„SQL“, Data Query Language, TOP & FETCH



TOP

```
SELECT  
    TOP <row count> *  
FROM  
    <table name>  
;
```

Beispiel

```
SELECT  
    TOP 5 *  
FROM  
    address  
;
```

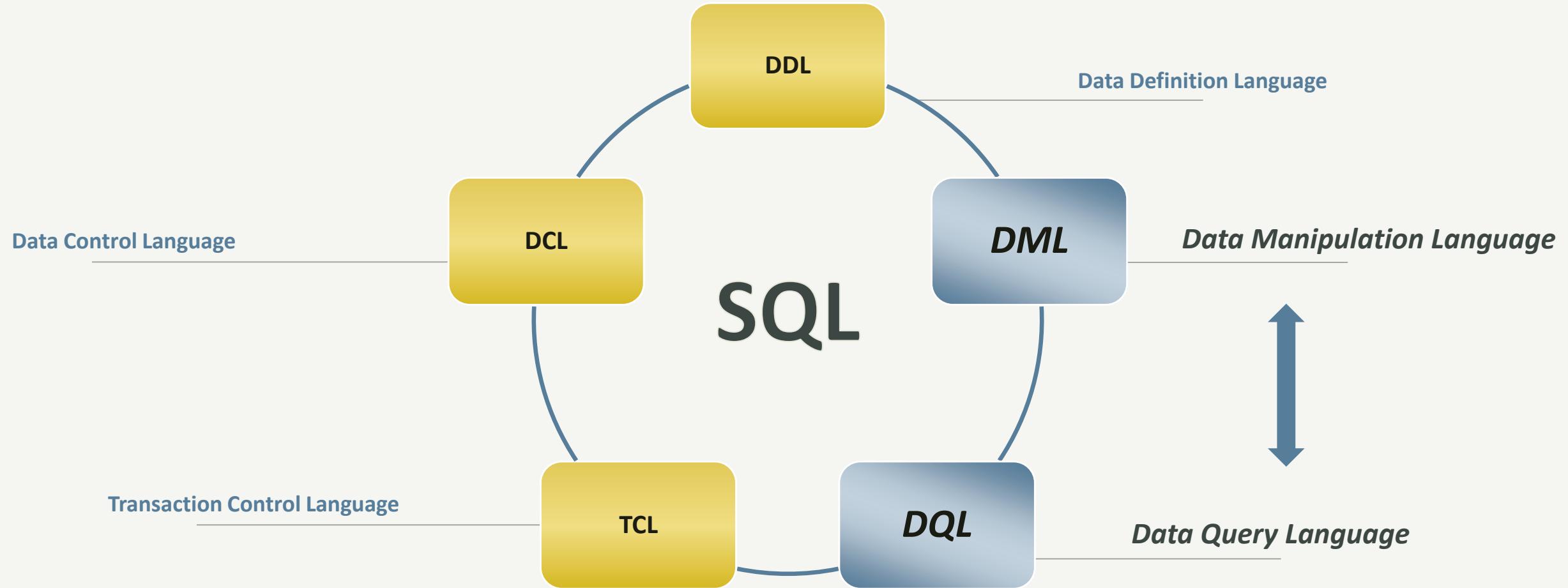
FETCH

```
SELECT  
    *  
FROM  
    <table name>  
ORDER BY  
    <column ...>  
OFFSET <row offset> ROW  
FETCH NEXT <row count> ROWS ONLY  
;
```

Beispiel

```
SELECT  
    *  
FROM  
    address  
ORDER BY  
    id  
OFFSET 3 ROW  
FETCH NEXT 3 ROWS ONLY  
;
```

SQL – DML, Data Manipulation Language

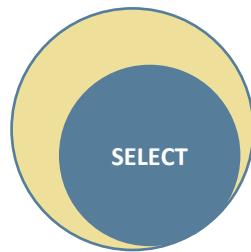


DDL, Data Definition Language, VIEW



View

Gespeicherte Abfragen & virtuelle Tabelle



CREATE OR ALTER VIEW <view name >
AS

```
SELECT  
    <column name ... >  
FROM  
    <table name>  
;
```

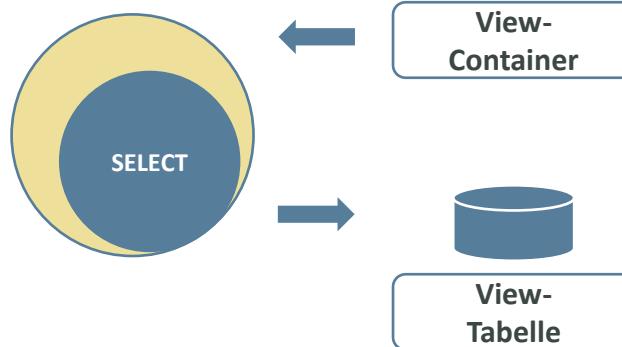
- Views sind gespeicherte Abfragen und stellen eine **Sicht auf abgefragte Tabellen dar**.
- Die Ergebnisdaten sind nicht in der View gespeichert.
- Das DBMS kompiliert bei der Anlage die Abfrage und löst statische Filter auf.
- Dynamische Filterwerte können nicht als Parameter übergeben werden.

DDL, Data Definition Language, VIEW



Materialized View

Gespeicherte Abfragen & persistenter Tabelle



```
CREATE OR ALTER MATERIALIZED VIEW <view name>
WITH ( <distribution_option> )
AS
SELECT
<column name ... >
FROM
<table name>
;
distribution_option
DISTRIBUTION = HASH ( [distribution_column_name [, ...n]] )
| DISTRIBUTION = ROUND_ROBIN
```

- Materialized Views sind **Sicht auf abgefragte Tabelle**, die das **Ergebnis** in einer internen Tabelle **speichern**.
- Damit kann eine Abfrage auf die View die **Ergebnisdaten direkt verwenden**.
- Tabellen, die in einer M-View verwendet werden, können nur bei deaktivierter View geändert werden.
- M-Views sind auf eine Größe von **8.096Byte je Zeile** beschränkt.
- Nicht alle Aggregatsfunktionen werden unterstützt, COUNT() ist z.B. nicht verfügbar.

DDL, Data Definition Language, TRIGGER



Trigger

Eventgesteuerte Zusatzausführungen

INSERT

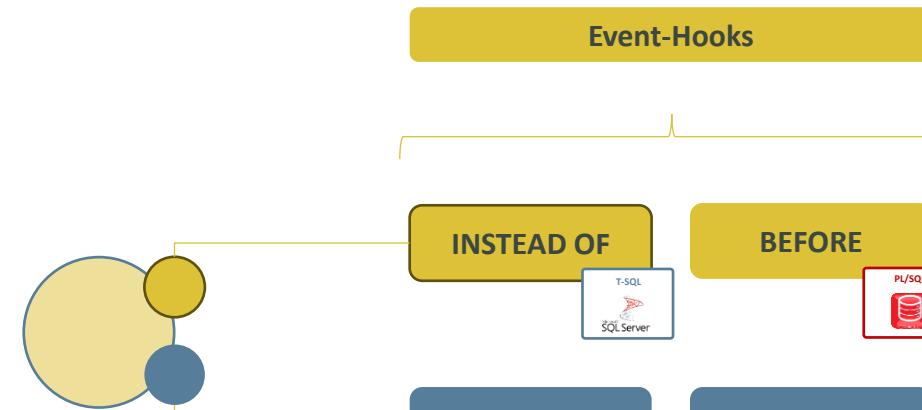
INSERTED
:NEW

UPDATE

INSERTED
:NEW

DELETE

DELETED
:OLD



Datenobjekt

- **Datenobjekt:** beinhaltet den zuletzt ausgeführten Datensatz

DDL, Data Definition Language, TRIGGER



Trigger, Typ 1

Tabellen-Trigger

```
CREATE OR ALTER TRIGGER <trigger name > ON <table name>
[ AFTER | INSTEAD OF ] [ INSERT, UPDATE, DELETE ]
AS
[BEGIN]
<command 1>
<command ...>
<command N>
[END];
```



Trigger, Typ 2

Objekt-Trigger

```
CREATE OR ALTER TRIGGER <trigger name > ON <datenbank objekt>
FOR <object event name>
AS
[BEGIN]
<command 1>
<command ...>
<command N>
[END];
```

DDL, Data Definition Language, TRIGGER, Beispiele



Beispiel, Typ 1

Tabellen-Trigger

```
CREATE OR ALTER TRIGGER tr_person_changed_date ON person
AFTER UPDATE
AS
BEGIN
    UPDATE per
    SET changed_date = SYSDATETIME()
    FROM person per JOIN inserted ins ON per.id = ins.id;
END;
```



Beispiel, Typ 2

Objekt-Trigger

```
CREATE OR ALTER TRIGGER tr_database_crttab_info ON DATABASE
FOR CREATE_TABLE
AS
DECLARE @var_command NVARCHAR(MAX) = NULL;
SELECT @var_command = EVENTDATA().value('/EVENT_INSTANCE/TSQLCommand/CommandText')[1],'NVARCHAR(MAX)');

PRINT N'CREATE TABLE executed.';
PRINT N'STATEMENT: ' + @var_command;
;
```

DDL, Data Definition Language, TRIGGER



!

Trigger sind ereignisgesteuerte Funktionen und können behilflich sein weiterführende Prozesserweiterungen sowohl auf Tabellen als auch Datenbankebene bereitzustellen.

- Trigger sind stark abhängig vom jeweils genutzten DBMS
- Nicht alle DBMS unterstützen Trigger (Snowflake stellt keine Trigger zur Verfügung)
- Trigger können Abhängigkeiten zu anderen Triggern aufbauen (Tabelle mit mehreren Triggern)
- Trigger sind nicht beliebig oft ineinander kaskadierbar (Kaskade aufgrund ihrer Event-Abhängigkeiten)
- Trigger können die jeweilige Ausführungsoperation verlangsamen
- Das Zielkonzept der Anwendung sollte den Trigger-Einsatz.

„SQL“, Data Query Language, *-Operator



!

Der ***-Operator** listet alle Spalten auf, die im aktuellen **Abfragekontext in der FROM-Klausel** geführt sind ohne dass die Spaltennamen explizit benannt werden müssen.

*-Operator

```
SELECT  
      *  
FROM  
    address;
```

Standard

```
SELECT  
  street  
, street_number  
, city  
, country  
FROM  
  address;
```

!

- Der **Operator** sollte nur während der **Entwicklungsphase** verwendet werden, jedoch **nicht grundsätzlich**.
- Durch den ***-Operator** **entfällt die Kontrolle** der Inhaltsrelevanten Spalten
- Durch die Verwendung **aller Spalten & Inhalte** wird die Performance beeinflusst,
da alle Daten bereitgestellt werden, auch wenn nur ein **Teil** der Inhalte verwendet wird.

„SQL“, Data Query Language, Konventionen I



!

Konventionen erleichtern die Erstellung von Programmen, die Zusammenarbeiten in den Teams und erhöhen die Wartbarkeit von Quelltexten.

Konventionen

- Namen & Bezeichnungen in Englisch
- Namen & Bezeichnungen in Lower-Case
- Zusammengesetzte Namen mit Unterstrich "_" trennen
- Befehle sind mit einem Semikolon abzuschließen
- Befehle in Upper-Case
- Befehle mit Zeilenumbrüchen strukturieren
- Komma von Listen am Anfang führen, z.B. bei Spaltenauflistungen
- *-Operator nur in Ausnahmefällen verwenden

SELECT

```
    id  
    , type_code  
    , type_description
```

FROM

```
address_type
```

```
;
```

„SQL“, Data Query Language, Konventionen II



!

Konventionen erleichtern die Erstellung von Programmen, die Zusammenarbeiten in den Teams und erhöhen die Wartbarkeit von Quelltexten.

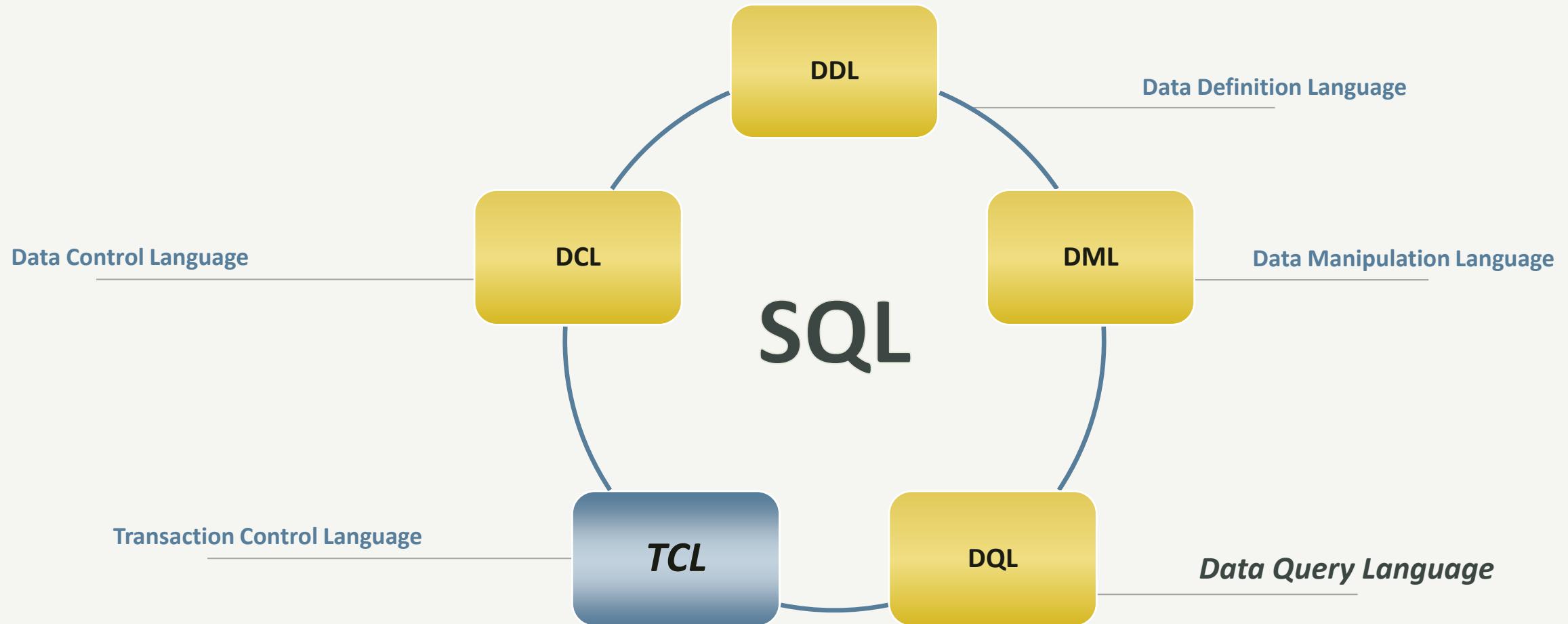
Konventionen

- Parameter mit dem Präfix „param_“ versehen
- Variablen mit dem Präfix „var_“ versehen
- Präfixe für weitere funktionale Objekte definieren, wie z.B. „v_“, „ix_“, „sp_“, „fn_“
- Jede Tabelle enthält eine ID-Spalte
- Jede Tabelle enthält ein Erstellungsdatum & ein Datum für die letzte Änderung

SELECT

```
    id  
    , type_code  
    , type_description  
FROM  
    address_type  
WHERE  
    type_code = @param_adr_code  
;
```

SQL – TCL, Transaction Control Language



Error-Handling, TRY-CATCH-Kontrollstruktur



Für die Behandlung von Ausnahmesituationen ist in **T-SQL** ein Sonderblock-Konstrukt verfügbar. Die **TRY-CATCH-Anweisung** besteht aus **2 Blöcken** zur Behandlung **TRY** und der Ausnahmebehandlung **CATCH**. Diese Blockanweisung kann verwendet werden, um jegliche Fehlerarten abzufangen.

Konstrukt

Ausführungsblock

```
BEGIN TRY  
    <command 01>  
    <command ...>  
  
END TRY  
BEGIN CATCH  
    <command 01>  
    <command ...>  
  
END CATCH  
;
```

Ausnahmebehandlung

Beispiel

```
T-SQL  
Microsoft SQL Server  
  
BEGIN TRY  
    SELECT 1/0 AS division_zero;  
END TRY  
BEGIN CATCH  
    SELECT 'error occurred' AS error_msg  
END CATCH  
;
```



TCL, Transaction Control Language



Die **Transaction Control Language** steuert die Ausführungen von Transaktionen und ermöglicht die Erstellung von Transaktionsblöcken, um Datenbankänderung blockweise zu schreiben **COMMIT** oder die Transaktionen im Fehlerfall vollständig rückgängig zu machen **ROLLBACK**.

Transaktion	BEGIN TRANS[ACTION] [<transaction name>;]	<ul style="list-style-type: none">• Beginn einer Transaktion
Schreiben	COMMIT [TRANSACTION <transaction name>;]	<ul style="list-style-type: none">• Speichern einer Transaktion• mit Namen bezogen auf die benannte Transaktion
Rückgängig	ROLLBACK [TRANSACTION <transaction name>;]	<ul style="list-style-type: none">• Transaktion wird vollständig zurückgesetzt• mit Namen bezogen auf die benannte Transaktion
Speicherpunkt	SAVE TRANS[ACTION] <transaction name>;	<ul style="list-style-type: none">• Speicherpunkt bis zudem ein Rollback ausgeführt wird• voranstehende Transaktionen werden COMMITTED

TCL, Transaktionsausführung mit COMMIT & ROLLBACK



Die Kombination aus Transaktions-Statements sowie dem Ausnahmeausführungsblock gewährleisten vollständige und sichere Transaktionsabbildungen im **Erfolgsfall COMMIT** wie auch **Fehlerfall ROLLBACK**.

Beispiel 01

```
BEGIN TRANSACTION employee_add
BEGIN TRY
    INSERT INTO employee (lastname, firstname) VALUES ('Muster 01', 'Max');
    INSERT INTO employee (lastname, firstname) VALUES ('Muster 02', 'Max');
    INSERT INTO employee (lastname, firstname) VALUES ('Muster 03', 'Max');
    COMMIT TRANSACTION employee_add;
END TRY
BEGIN CATCH
    PRINT 'error occurred';
    ROLLBACK TRANSACTION employee_add;
END CATCH;
```

Hinweis

- Nach der erfolgreichen Ausführung eines COMMIT ist die Transaktion abgeschlossen.
- Für weitere Operationen ist eine neue Transaktion zu eröffnen.
- Transaktionsblöcke können ineinander geschachtelt werden.

TCL, Transaktionsausführung mit COMMIT & ROLLBACK



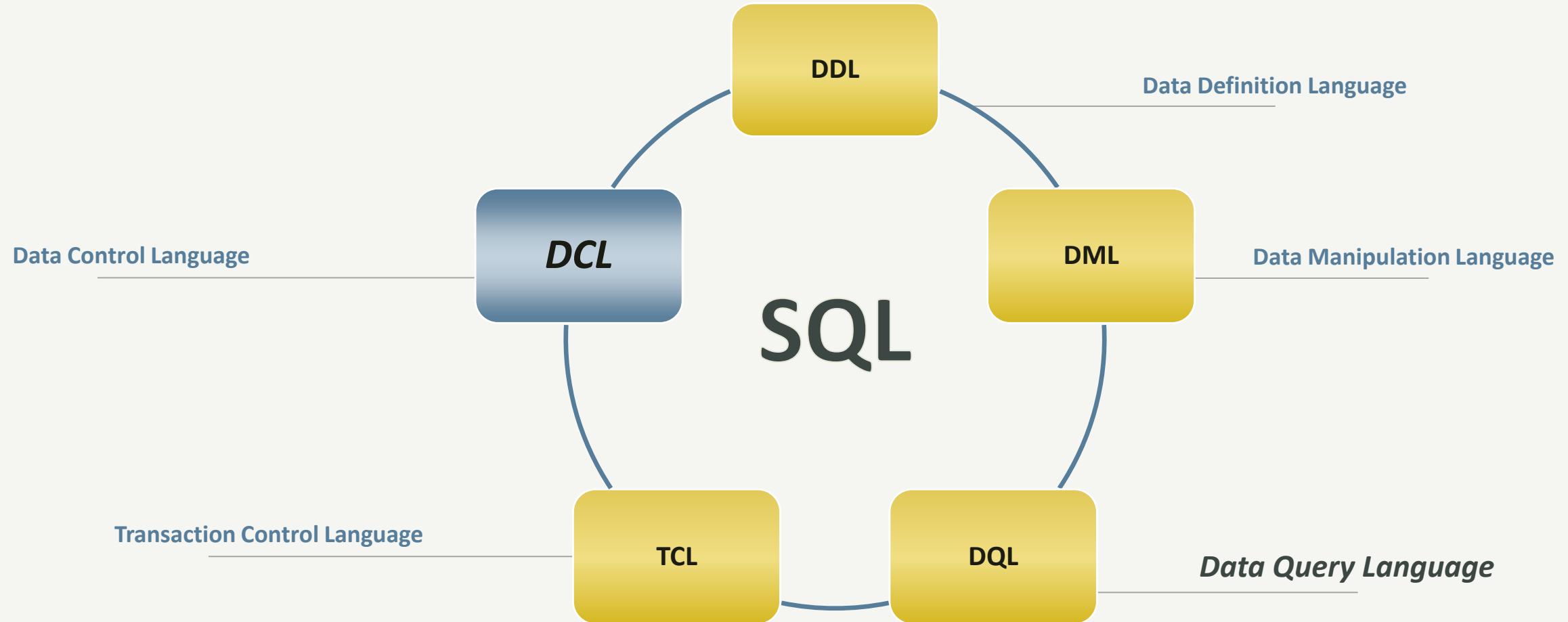
Beispiel 02

```
BEGIN TRANSACTION employee_add
BEGIN TRY
    INSERT INTO employee (lastname, firstname) VALUES ('Muster 01', 'Max');
    SAVE TRANSACTION record_added;
    INSERT INTO employee (lastname, firstname) VALUES ('Muster 02', 'Max');
    SAVE TRANSACTION record_added;
    INSERT INTO employee (lastname, firstname) VALUES ('Muster 03', 'Max');
    COMMIT TRANSACTION employee_add;
END TRY
BEGIN CATCH
    DECLARE @error_message      NVARCHAR(4000);
    DECLARE @error_severity     INT;
    DECLARE @error_state        INT;

    SELECT
        @error_message = ERROR_MESSAGE()          -- message text
    ,   @error_severity = ERROR_SEVERITY()        -- severity
    ,   @error_state = ERROR_STATE() -- state
    ;
    ROLLBACK TRANSACTION record_added;

    RAISERROR( @error_message, @error_severity, @error_state );
END CATCH;
```

SQL – DQL, Data Query Language

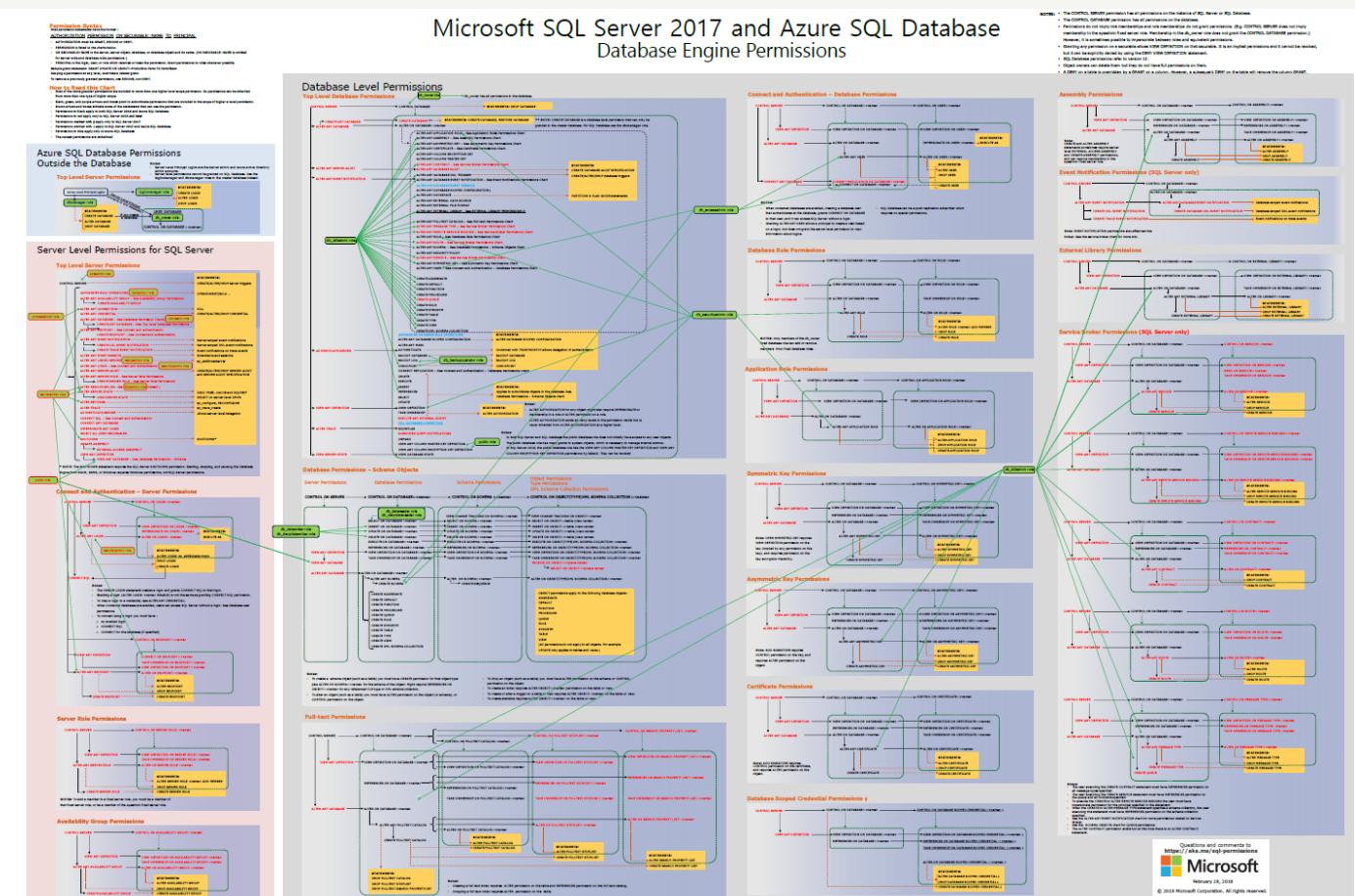


DCL, Data Control Language



Die **Data Control Language** steuert die Berechtigungszuweisungen **GRANT** sowie den Berechtigungsentzug **REVOKE** von Ausführungsprivilegien eines Datenbankbenutzers einschließlich der Erstellung und Verwaltung von Berechtigungsrollen.

(Quelle: https://raw.githubusercontent.com/Microsoft/sql-server-samples/master/samples/features/security/permissions-posters/Microsoft_SQL_Server_2017_and_Azure_SQL_Database_permissions_infographic.pdf, Stand: 21.12.2023)



DCL, Data Control Language - Zuweisung



!

Die **Data Control Language** dient neben der Berechtigungsverwaltung auch der Benutzer und Rollenverwaltung. Bestimmte Berechtigungen können jedoch nur vererbt werden. Dies wiederum erfolgt durch die Zuordnung der entsprechenden Berechtigungsgruppen wie z.B. der Rolle des **dbo**.

USER

Anmeldung

`CREATE LOGIN <login name> WITH [<password>, ...];`

Benutzer

`CREATE USER <user name> FOR LOGIN;`

GRANT

Berechtigung-
Zuweisen, Tabelle

`GRANT <execution> ON OBJECT::<object name> TO <user name>;`

Berechtigung-
Zuweisen, Execution

`GRANT <execution function> TO <user name>;`

DCL, Data Control Language - Entzug



!

Die **Data Control Language** ist in seiner Gesamtheit äußerst vielfältig sowie in seiner Umsetzung stark abhängig vom jeweiligen (R)DBMS.

USER

Anmeldung

DROP LOGIN <login name>;

Benutzer

DROP USER <user name>;

GRANT

Berechtigungs-
Entzug, Tabelle

REVOKE <execution> **ON OBJECT::<object name>** **FROM** <user name>;

Berechtigungs-
Entzug, Execution

REVOKE <execution function> **FROM** <user name>;



DCL, Data Control Language - Beispiel

GRANT-Beispiel

```
CREATE LOGIN tsql_user      WITH PASSWORD = '340$Uuxwp7Mcxo7Khy';
CREATE USER tsql_user       FOR LOGIN tsql_user;
GO
```

```
GRANT SELECT  ON OBJECT::study.dbo.address TO tsql_user;
GRANT INSERT   ON OBJECT::study.dbo.address TO tsql_user;
GRANT DELETE   ON OBJECT::study.dbo.address TO tsql_user;
GRANT UPDATE   ON OBJECT::study.dbo.address TO tsql_user;
GRANT ALTER    ON OBJECT::study.dbo.address TO tsql_user;
GO
```

```
GRANT CREATE TABLE TO tsql_user;
GRANT CREATE VIEW TO tsql_user;
GRANT CREATE FUNCTION TO tsql_user;
GRANT CREATE PROCEDURE TO tsql_user;
GRANT SHOWPLAN TO tsql_user;
GO
```

REVOKE-Beispiel

```
DROP LOGIN tsql_user;
DROP USER tsql_user;
GO
```

```
REVOKE SELECT  ON OBJECT::study.dbo.address FROM tsql_user;
REVOKE INSERT   ON OBJECT::study.dbo.address FROM tsql_user;
REVOKE DELETE   ON OBJECT::study.dbo.address FROM tsql_user;
REVOKE UPDATE   ON OBJECT::study.dbo.address FROM tsql_user;
REVOKE ALTER    ON OBJECT::study.dbo.address FROM tsql_user;
GO
```

```
REVOKE CREATE TABLE FROM tsql_user;
REVOKE CREATE VIEW FROM tsql_user;
REVOKE CREATE FUNCTION FROM tsql_user;
REVOKE CREATE PROCEDURE FROM tsql_user;
REVOKE SHOWPLAN FROM tsql_user;
```

Index-Verwendung



Übersicht



SQL, Befehle der SQL-Familie

- DQL, SQL-Abfrage
- DDL, Tabellenerstellung, Änderung, Löschen
- DML, Datenerstellung, Änderung, Löschen
- DTL, Transaktionssicherung,
COMMIT & ROLLBACK
- DCL, Berechtigungskontrolle,
GRANT & REVOKE

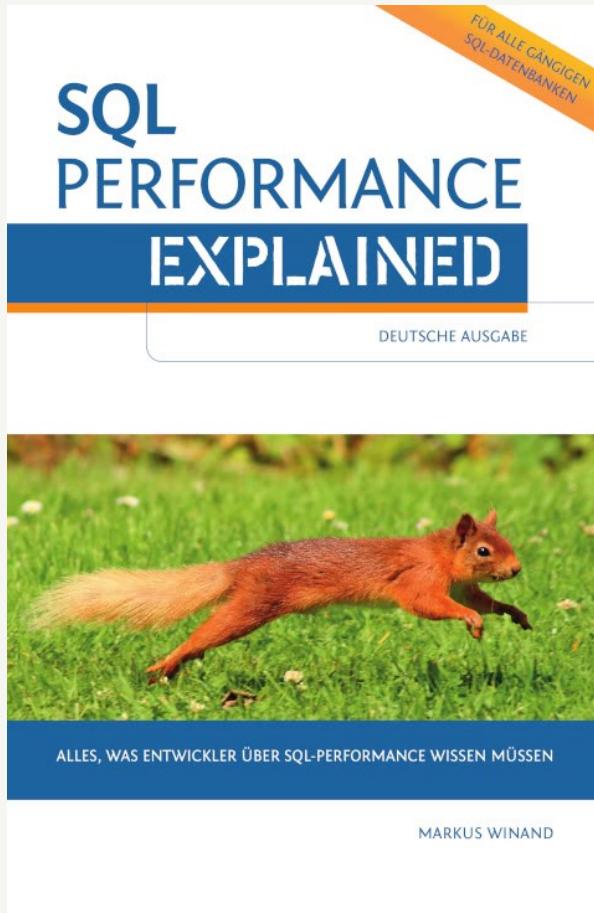


Indexierung

- Index-Verwendung
- Definition, Index als Datenstruktur
- Indexierung, Index-Baum
- Indexsuchen, wann und wie wirkt der Index
- Zusammenfassung
- Hinweise zur Indexverwendung



Referenz „Indexierung & Performance“



Markus Winand

„SQL ist tot!
hieß es noch vor wenigen
Jahren.

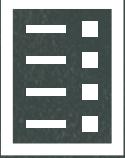
Heute setzen auch die großen
wieder verstärkt auf SQL.

Sind Sie bereit für das Revival?



Quelle: <https://sql-performance-explained.de> (Stand: 19.12.2023)

Index, Datenstruktur



!

Ein Index ist eine selbstständige Datenstruktur in einer Datenbank mit dem Primärziel die Suchanfragen der gebundenen Tabelle effizienter zu gestalten.

1.	Einleitung	4
1.1	Problemstellung	4
1.2	Ziel der Arbeit	5
1.3	Vorgehen	5
2.	Zielgruppencharakteristik	6
3.	Extralinguistische Merkmale des Kiezdeutschen	9
3.1	Historisch-politischer Hintergrund der Genesis des Kiezdeutschen	9
3.2	Kiezdeutsch als Subkultur	11
4.	Inter- und intralinguistische Merkmale des Kiezdeutschen	14
4.1	interlinguistische Merkmale als Begleiterscheinung der Subkultur	14
4.2	Grammatische Eigenschaften des Kiezdeutschen	18
4.2.1	Semantische Eigenschaften des Kiezdeutschen	19
4.2.2	Semantische Analysen des Kiezdeutschen	19
4.3	Syntaktische Eigenschaften des Kiezdeutschen	25
4.3.1	Wortklassen	25
4.3.2	Wortstellung	27
4.3.3	Flexion	29
4.3.3.1	Deklination	29
4.3.3.2	Konjugation	31
4.3.4	Präpositionen	33
4.3.5	Artikel	34
4.3.6	Partikel	35
4.3.7	Funktionsverbgefüge	38
4.3.8	Konjunktiv	40

- Eigenständige Datenstruktur
- Besteht aus **redundanten Daten** der gebundenen Tabelle der Indexspalte(n)
- Die Indexanlage lässt die gebundene Tabelle unverändert
- Bei Indexanlage erfolgt die Erstellung einer neuen Datenstruktur
- Ein Index verweist auf eine oder mehrere Spalten der gebundenen Tabelle
- **Index-Suche entspricht z.B. der Suche in einem Inhaltsverzeichnis oder Telefonbuch anhand ausgewählter Sortierkriterien.**

Index, Anlage & Löschung



Ein Index aus einer Index-Spalte ist ein Einzel-Index. Ein Index aus mehreren Spalten ist ein zusammengesetzter Index. Zusammengesetzte Indexe optimieren Suchanfragen über mehrere Spalten.

Index-Anlage

CREATE INDEX <name> ON <table> (<column [, column]>);

(Einzel-) Index

CREATE INDEX idx_person_lastname ON employee (lastname);

Zusammengesetzter Index

CREATE INDEX idx_person_fullname ON employee (lastname, firstname);

Index-Entfernung

DROP INDEX <name> ON <table>;

(Einzel-) Index

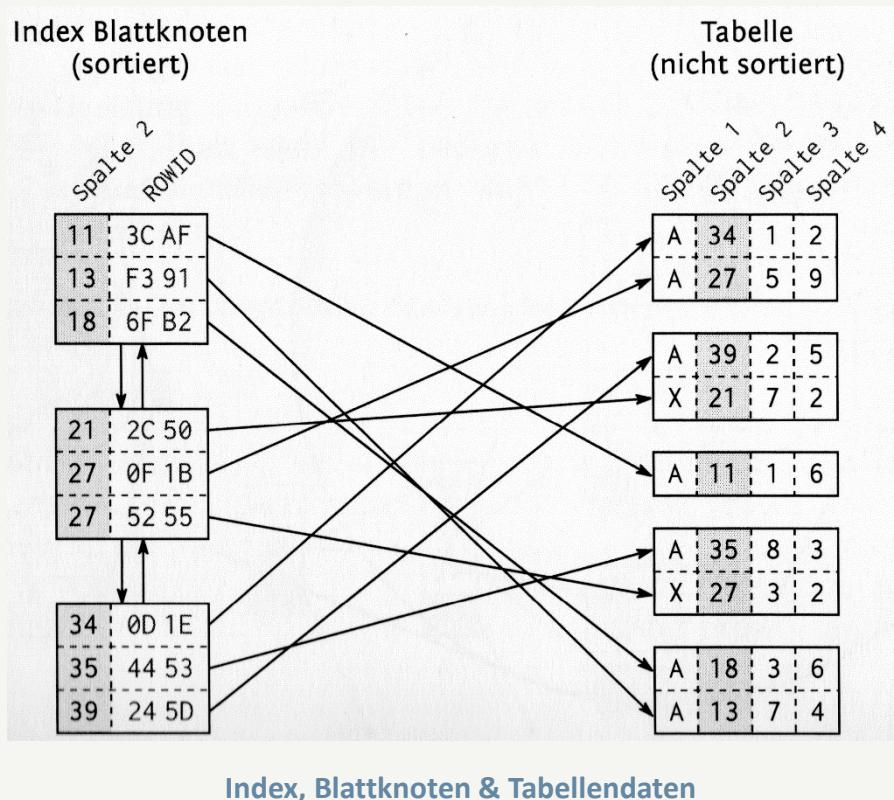
DROP INDEX idx_person_lastname ON employee;

Zusammengesetzter Index

DROP INDEX idx_person_fullname ON employee;



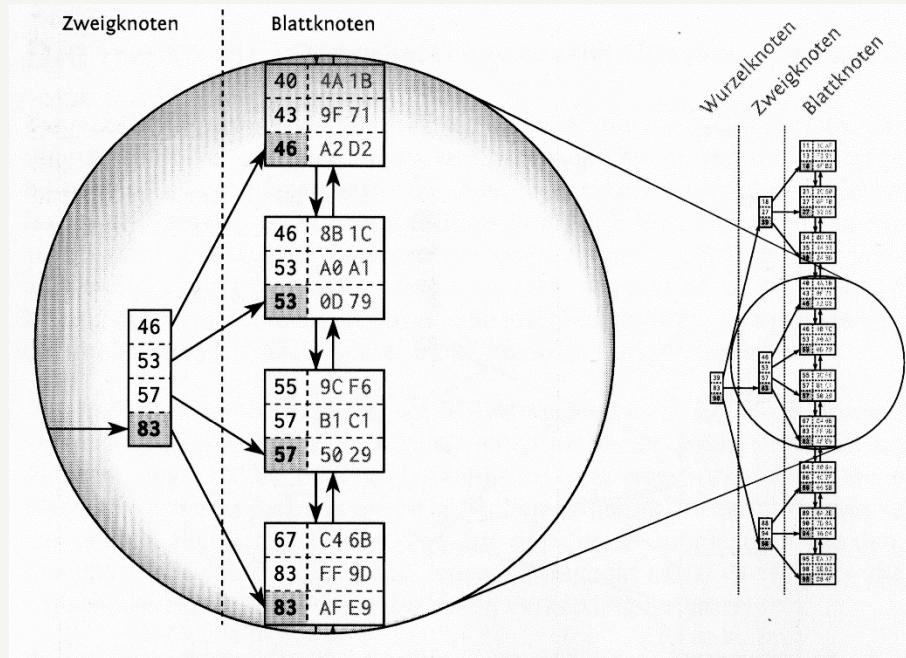
Index, Grundstruktur - Indexblätter



- Datensätze in der Tabelle sind unsortiert.
- Der Index verwendet ausschließlich die **Indexspalten** (Inhaltskopie) und die zugehörige **ROWID**.
- Die **Index-Spalte** ist immer **sortiert** (ASC, DESC).
- Jeder Indexeintrag ist ein Knoten mit Vorgänger & Nachfolger.
- Die logische Reihenfolge wird durch die Verkettung der Indexknoten erzeugt, die Verkettung erfolgt durch **doppelverkettete Listen zueinander** (Sicherstellung der Vorwärts- & Rückwertssuche).
- Mehrere Indexknoten werden durch das DBMS zu einem **Blattknoten (Leaf-Node)** zusammengefasst.
- Der Blattknoten wird immer in einem **Datenblock (Page)** gespeichert.
- Wenn ein Indexeintrag gefunden wurde, kann per ROWID direkt auf den Datensatz in der Tabelle zugegriffen werden.



Indexierung, B-Tree, Index-Baum



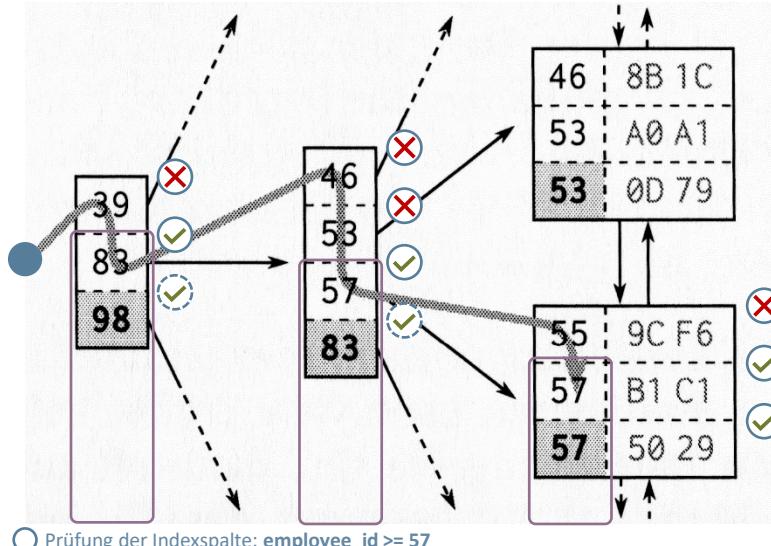
- B-Tree ist ein „**Balanced Tree**“ (kein Binary-Tree)
- **Verzweigungsschichten** entstehen durch die Beendigung eines Blattknotens.
- Durch einfügen eines Indexknotens (INSERT) und **überschreiten der Blattknotenkapazität** erfolgt die Anlage eines neuen Blattknotens.
- Zusätzlich wird ein **Zweigknoten erstellt**, der den **Maximalwert der Indexspalte** aus dem Blattknoten (letzter Eintrag) bekommt.
- Somit ergibt sich das Zweigknotenblatt 46, 53, 57 & 83.
- Die Verzweigungsschichten werden, solange fortgeführt bis ein **Root-Knoten** als oberster Baumknoten erreicht ist.



Indexierung, Suche

Suchabfrage mit Einzelindex „employee_id“

...WHERE employee_id >= 57;

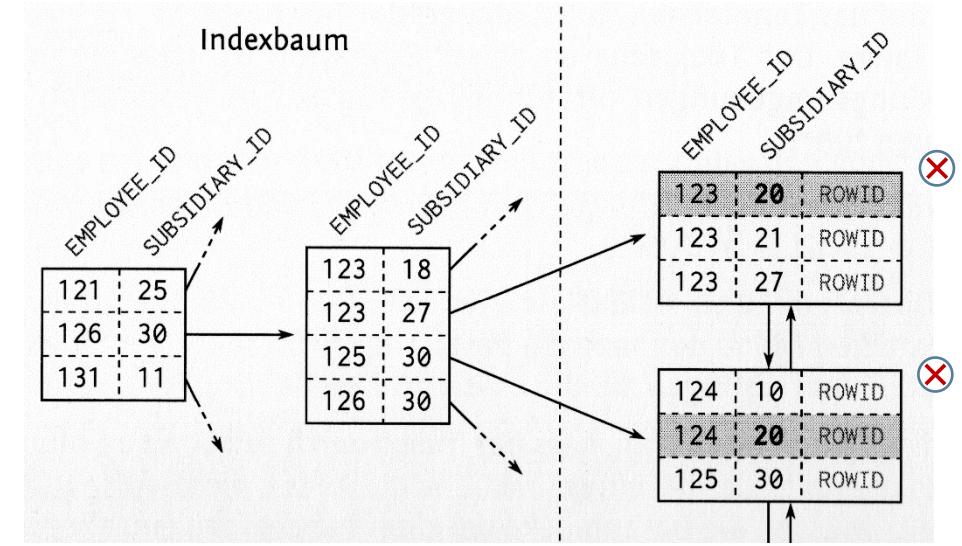


Einzelindex „employee_id“

Durchwandern des Suchbaums durch einen „Index Range Scan“

Suchabfrage

...WHERE subsidiary_id = 20;



Zusammengesetzter Index „employee_id, subsidiary_id“

Durchwandern des Suchbaums durch einen „Index Range Scan“

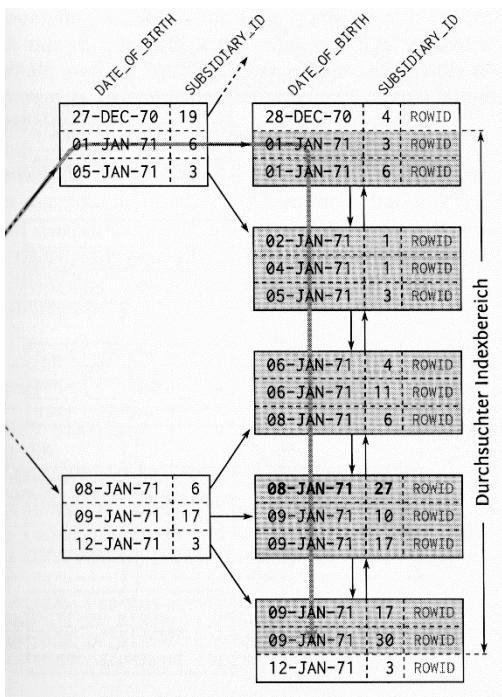


Indexierungsreihenfolge, Suche

Suchabfrage

...WHERE

date_of_birth BETWEEN CAST('1970-01-01' AS DATE) AND CAST('1970-01-08' AS DATE)
AND subsidiary_id = 27;

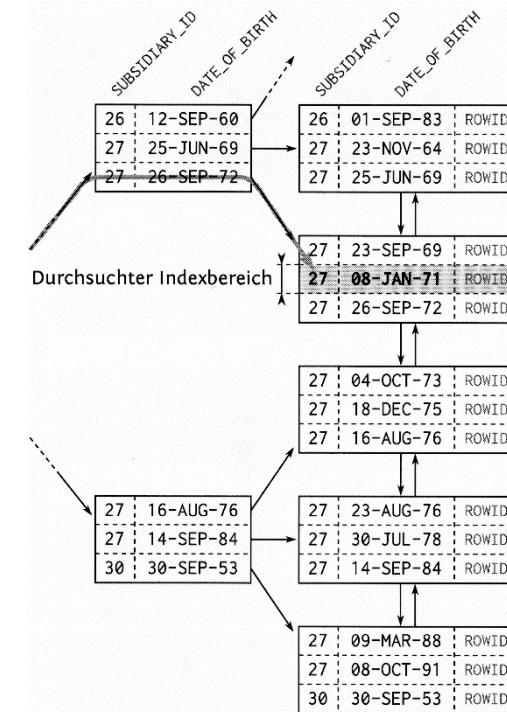


Index-Reihenfolge 01: date_of_birth, subsidiary_id
Suche durch „Index Range Scan“

- Zugriffsprädikat: date_of_birth
- Filterprädikat: subsidiary_id

Index-Reihenfolge 02: subsidiary_id, date_of_birth
Suche durch „Index Seek“

- Zugriffsprädikat: subsidiary_id
- Filterprädikat: date_of_birth



Wie kann der "Index 01" das Filterprädikat zum Zugriffsprädikat umgestellt werden?

Wie kann die Abfrage so optimiert werden, dass alle WHERE-Spalten zum Zugriffsprädikat werden?

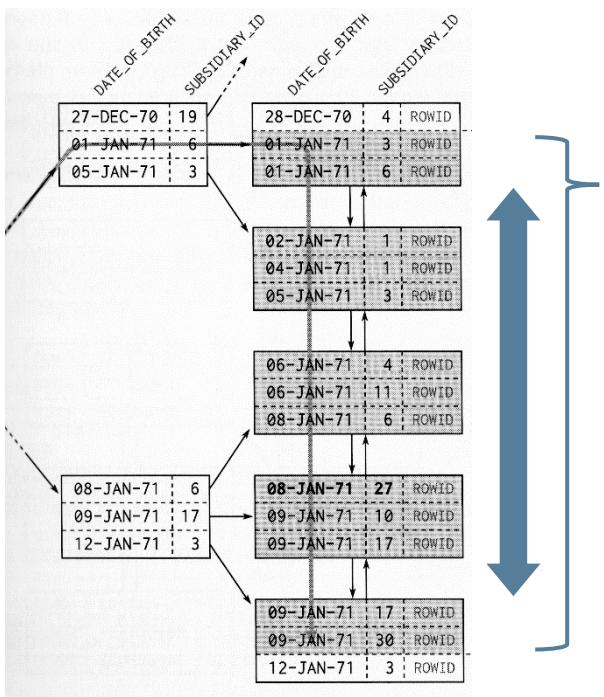


Indexierungsreihenfolge, Suche

Suchabfrage

...WHERE

date_of_birth BETWEEN CAST('1970-01-01' AS DATE) AND CAST('1970-01-08' AS DATE)
AND subsidiary_id = 27;



Zugriffsprädikat: `date_of_birth`

- Einschränkung des Index-Suchbereichs
- Definition des Anfangs- und Endpunktes für den Indexbaum

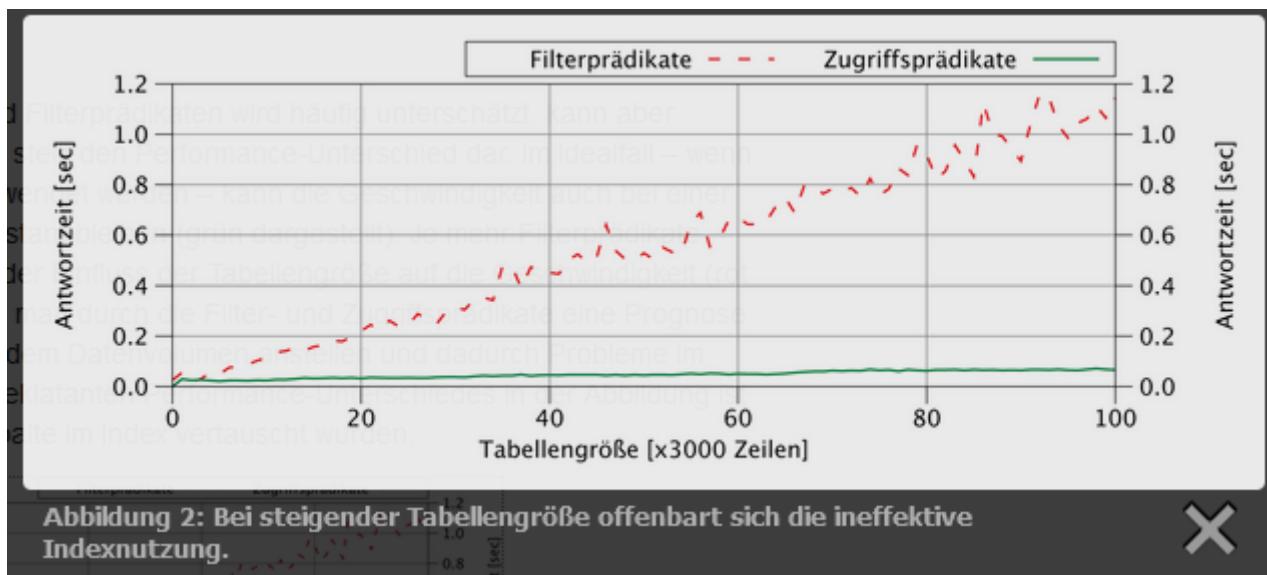
(Index-)Filterprädikat: `subsidiary_id`

- Reduzieren nicht den Indexsuchbereich
- Filterung innerhalb des Indexsuchbereiches



Indexierungsreihenfolge, Suche

Wirkung Filterprädikat & Zugriffsprädikate



Quelle: [https://www.admin-magazin.de/Das-Heft/2012/06/Datenbank-Ausfuehrungsplaene-lesen-und-verstehen/\(offset\)/2](https://www.admin-magazin.de/Das-Heft/2012/06/Datenbank-Ausfuehrungsplaene-lesen-und-verstehen/(offset)/2), Stand: 11.01.2024



Indexierung, Like-Suchen mit Index

Suchabfrage

Indexreihenfolge: lastname

`CREATE INDEX idx_employee_lastname ON employee (lastname);`

...WHERE `lastname = 'WINAND';`

Platz 1

INDEX UNIQUE SCAN, Gleichheitsoperator auf Indexspalte (= Zugriffsprädikat)

...WHERE `lastname like '%WINAND';`

Platz 5

Nur Filterprädikat, keine Index-Verwendung, durch Wildcard Filterlänge ungekannt

...WHERE `lastname like 'WI%ND';`

Platz 4

Zugriffs & Filterprädikat, Zugriffsprädikat aus 2 Buchstaben

...WHERE `lastname like 'WIN%D';`

Platz 3

Zugriffs & Filterprädikat, Zugriffsprädikat aus 3 Buchstaben

...WHERE `lastname like 'WINA%';`

Platz 2

Zugriffs & Filterprädikat, Zugriffsprädikat aus 4 Buchstaben

?

Bilde die Reihenfolge der Filterungen
von der schnellsten Ausführung zur langsamsten.



Indexierung, Like-Suchen mit Index

Suchabfrage

Indexreihenfolge: lastname

`CREATE INDEX idx_employee_lastname ON employee (lastname);`

✗ ...WHERE lastname like '%WINAND';

Platz 2

...WHERE lastname like 'WI%ND';

Platz 3

...WHERE lastname like 'WIN%D';

Platz 4

...WHERE lastname like 'WINA%';

✓ ...WHERE lastname = 'WINAND';



Bilde die Reihenfolge der Filterungen
von der schnellsten Ausführung zur langsamsten.



Platz 4	Platz 3	Platz 2
WIAW	WIAW	WIAW
WIBLQQNPUA	WIBLQQNPUA	WIBLQQNPUA
WIBYHSNZ	WIBYHSNZ	WIBYHSNZ
WIFMDWUQMB	WIFMDWUQMB	WIFMDWUQMB
WIGLZX	WIGLZX	WIGLZX
WIH	WIH	WIH
WIHTFVZNLC	WIHTFVZNLC	WIHTFVZNLC
WIJYAXAPP	WIJYAXAPP	WIJYAXAPP
WINAND	WINAND	WINAND
WINBKYDSKW	WINBKYDSKW	WINBKYDSKW
WIPOJ	WIPOJ	WIPOJ
WISRGPK	WISRGPK	WISRGPK
WITJIVQJ	WITJIVQJ	WITJIVQJ
WIW	WIW	WIW
WIWGPJMQGG	WIWGPJMQGG	WIWGPJMQGG
WIWKHLBJ	WIWKHLBJ	WIWKHLBJ
WIYETHN	WIYETHN	WIYETHN
WIYJ	WIYJ	WIYJ

Index-Wirkung bei der Like-Suche



Indexierung, Datenzugriff

Suchabfrage

Platz 1

WIAW
WIBLQQNPUA
WIBYHSNZ
WIFMDWUQMB
WIGLZX
WIH
WIHTFVZNLC
WIJYAXPP
WINAND
WINBKYDSKW
WIPOJ
WISRGPK
WITJIVQJ
WIW
WIWGPJMQGG
WIWKHLBJ
WIYETHN
WIYJ

```
SELECT lastname, firstname, date_of_birth  
FROM employee  
WHERE lastname = 'WINAND';
```

Table Access By Index ROWID

```
SELECT lastname  
FROM employee  
WHERE lastname = 'WINAND';
```

Direct Access Of Index Value

```
SELECT lastname, firstname, date_of_birth  
FROM employee  
WHERE lastname = 'WINAND';
```

employee			
ROWID	lastname	firstname	date_of_birth
8B 1C	WIMLE	Paula	1969-02-24
A0 A1	WINUM	Peter	1971-05-27
0D 79	WINAND	Tim	1981-05-01
50 29	XULL	Frank	1995-11-12

T-SQL, INCLUDE of columns

```
CREATE INDEX idx_employee_infos  
ON employee ( lastname )  
INCLUDE ( firstname, date_of_birth );
```



Indexierung, Like-Suchen mit Index

Suchabfrage

Indexreihenfolge: lastname

`CREATE INDEX idx_employee_lastname ON employee (lastname);
CREATE INDEX idx_employee_dob ON employee (date_of_birth);`

...WHERE `LOWER(lastname) = 'winand';`

Platz 2
Platz 4.1

Kein Zugriffsprädikat aufgrund Funktionskapselung,
Filterprädikat durch Ist-Gleichheitsoperator beschleunigt

...WHERE `TRIM(lastname) like 'WINA%';`

Platz 4
Platz 5.2

Kein Zugriffsprädikat aufgrund Funktionskapselung,
Filterprädikat mit Like-Suche und unbekannter Länge verlangsamt

...WHERE `lastname = UPPER('winand');`

Platz 1
Platz 1.1

INDEX UNIQUE SCAN (INDEX SEEK), Zugriffsprädikat,
Filterbegriff vor Nutzung zu transformieren

...WHERE `date_of_birt like '1971-01-08';`

Platz 3
Platz 4.2

Kein Zugriffsprädikat aufgrund impliziter Konvertierung,
Like-Operator konvertiert Datumsspalte zu Zeichenkette,
somit nur Filterprädikatensuche



Bilde die Reihenfolge der Filterungen
von der schnellsten Ausführung zur langsamsten.



Indexierung, Begriffszusammenfassung

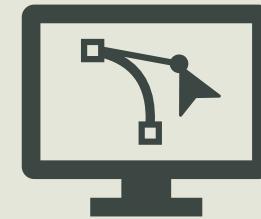
- **Zugriffsprädikat**, „access predicates“
Filterkriterien, welche die Start-Stopp-Bedingung für den Indexsuche bestimmen und somit den Indexbereich definieren. Die Index-Reihenfolge muss mit der Reihenfolge der WHERE-Bedingung übereinstimmen.
- **Filterprädikat**, „filter predicates“
Filterkriterium zur Ermittlung der Zielwerte. Der Indexbereich bleibt unverändert.
- **Full Table Scan**
Suche basierend auf einem Filterprädikat für die gesamte Tabellenbereich der Spalte durchsucht wird, da kein Index verfügbar oder bestehende Indexe nicht zur Anwendung kommen können.
- **Index Unique Scan**
Eine Filterbedingung bei der alle Spalten als Zugriffsprädikat verwendet werden können und mit dem Gleichheitsoperator arbeiten.
- **Index Range Scan**
Filterbedingungen bei der mehrere Zugriffsprädikate einen Index-Suchbereich aufspannen, um die zugehörigen Zeilen zu ermitteln.
- **Table Access By Index ROWID**
Zugriff nach erfolgreicher Indexsuche auf den Datensatz der Datentabelle per ROWID. Dieser Zugriff erfolgt für jedes Index-Suchergebnis.



Indexierung, Hinweise

- Indexstrukturen werden bei INSERT, UPDATE, DELETE aktualisiert
- Indexe zuerst an Gleichheitsprüfungen ausrichten
- Suchprädikat bei dem Like-Operator mit Prädikatinhalt vor der Wildcard „%“
- Vermeidung der Suche mit beginnender Wildcard „%“
- Vermeidung von impliziten Datumskonvertierungen bei suchen
- Indexe gezielt einsetzen
- Indexspalten bewusst wählen, besonders hinsichtlich Datentypgröße
- Teilindizes können verwendet werden

Advanced Features



Übersicht



SQL, Advanced Feature

- Standardbenutzer, Datenbanken, Schema
- DDL, Tabellenerstellung, Änderung, Löschen
- Kommentare
- Parameter
- Datentyp-Konvertierungen
- Kontrollstrukturen
 - CASE
 - IF-ELSE
 - WHILE
- Computed-Columns
- Sequences
- Temporäre Tabellen
- Common Table Expression
- Funktionen (UDF) & Prozeduren
- String-Split & -Merge
- JSON-Funktionen
- Systemtabellen

Advanced Features, Standardbenutzer



!

Jedes **Datenbankmanagementsystem** verwendet **Standardbenutzer** mit unterschiedlichen Standardberechtigungen. Diese sind direkt nach der Installation verfügbar und besitzen gesonderte Berechtigungsprivilegien.

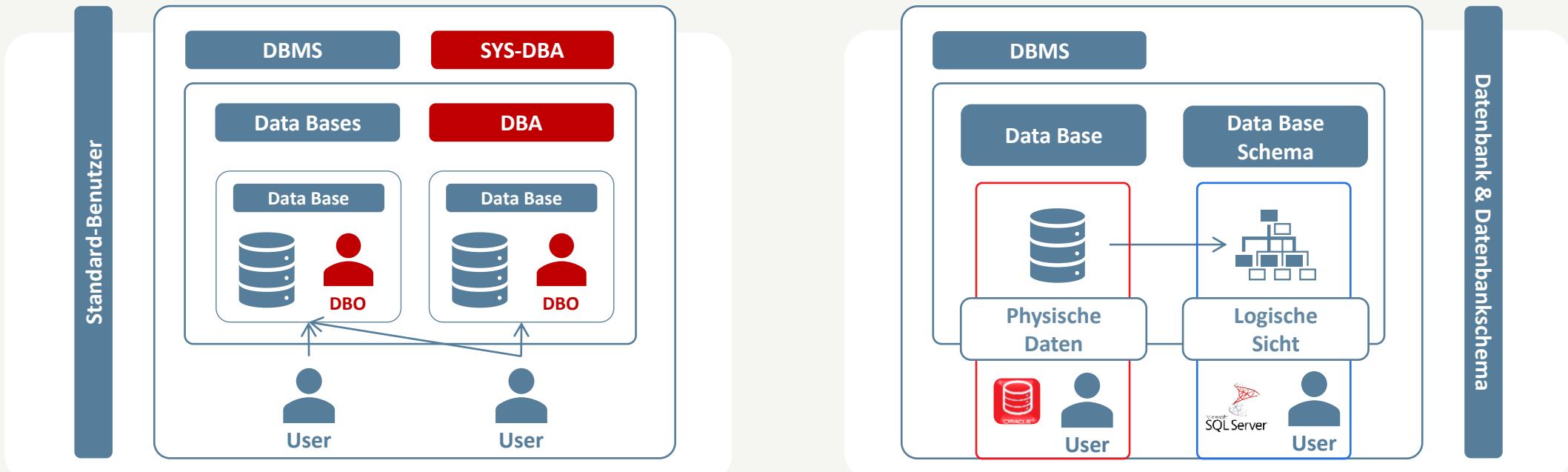
Administrator	SYS-DBA	System Data Base Administrator	Besitzt weitreichende Berechtigung, um das DBMS zu administrieren inkl. aller Datenbanken & Benutzer.
DBA	DBA	Data Base Administrator	Besitz weitreichende Berechtigung zur Administration aller Datenbanken & Benutzer, jedoch nicht zur DBMS-Administration.
DBO	DBO	Data Base Owner	Besitzt umfangreiche Berechtigung, die zugewiesene Datenbank zu administrieren und besitzt umfangreiche Anlage zur Datenbankgestaltung.
User	Benutzer		Besitzt ausschließlich die Ausführungsberechtigungen, die von einem der 3 Administrator-Gruppen zugewiesen wurden.

Advanced Features, Datenbank & Datenbankschema



!

Innerhalb des Datenbankmanagementsystems beschreibt die **Datenbank** die physikalische Abbildung der Datenhaltung und im engeren Sinn die Datenbank-Dateien. Das **Datenbankschema** beschreibt die logische **Sicht** auf die Daten der Datenbank.



Advanced Features, Vollqualifizierter DB-Zugriff



Wenn ein Benutzer gleichzeitig auf mehrere Datenbanken Berechtigungen aufweist, sind die zu verwendenden/erstellenden Tabellen per vollqualifiziertem Datenbankpfad anzusprechen. Im Standard wird grundsätzlich die Datenbank der aktuellen Session verwendet, sofern keine Vollqualifizierung erfolgt.

**Setzen der DB
für aktuelle Session**

`USE <database name>;`

**Vollqualifizierter
Datenbankpfad**

`...FROM <database name>.dbo.<table name>;`

`...FROM <database name>.<table name>;`



Advanced Features, Vollqualifizierter DB-Zugriff



!

In einem **JOIN-Verbund** kann auf Basis des **vollqualifizierten Datenbankpfads auf Tabellen verschiedener Datenbanken** gleichzeitig zugegriffen werden. Diese Zugriffe können in einer JOIN-Struktur direkt in Kombination gebracht werden analog zu Tabellen der gleichen Datenbank.

Vollqualifizierter Zugriff

```
SELECT
    pers1.lastname AS study_pers_lastname
    , pers2.lastname AS emp_pers_lastname
FROM
    study.dbo.person pers1
    JOIN employees.dbo.employee emp ON pers1.id = emp.study_person_id
    JOIN employees.dbo.person pers2 ON pers2.id = emp.person.id
WHERE
    pers1.lastname LIKE 'Mustermann'
;
```

Advanced Features, Kommentare



!

Kommentierungen sind wesentliche Bestandteile von **SQL-Skripten**, um diese **verständlich zu gestalten**. Je umfangreicher SQL-Skripte werden, um so wichtiger ist eine prägnante Kommentierung. Gleichzeitig **erhöht** sich hierdurch die **Wartbarkeit** der Statements.

Kommentare

2 Kommentarformen, **einzeilig** und **mehrzeilig**

Einzeilig

-- Einzeiliger Kommentar

- Kommentar kann innerhalb einer Zeile stehen
- Kommentar kann in mehrzeiligen Kommentar geschachtelt sein

Mehrzeilig

/* mehrzeiliger Kommentar

*/

- Kommentar kann innerhalb einer Zeile stehen
- Kommentare können über mehrere Zeilen reichen

Advanced Features, Kommentare



!

Besonders bei der **einzeligen Kommentierung** ist das Wissen über das Verfahren der Skript-Einlesung relevant. Anwendungs- & herstellerspezifisch werden **Skripte** teilweise sehr **unterschiedlich eingelesen**, insbesondere wenn der Zeilenumbruch entfernt wird, können sich kritische Effekte einstellen.

Beispiel

Textdatei mit UPDATE-Statement, Formatierung mit Zeilenumbrüchen

Erstelltes Skript

```
UPDATE
    employee
SET
    study_lead = 'Prof. Musterfach' -- change the leader
WHERE
    subject = 'Informatik';
```

Eingelesenes Skript

```
UPDATE employee SET study_lead = 'Prof. Musterfach' -- change the leader WHERE subject = 'Informatik';
```

Advanced Features, Parameter



Parameter dienen der **Flexibilisierung und Dynamisierung** von Datenbankskripten. Die Umsetzung von Parametern und deren Verwendung ist **je Hersteller und Datenbankmanagementsystem unterschiedlich**.

Datentypen

Steht allen Datentypen bereit



Deklaration

```
DECLARE @<parameter name> <data type> [ = <default value>];
```

Wertzuweisung 01

```
SET @<parameter name> = [value, (subquery), parameter];
```

Wertzuweisung 02

SELECT

```
@<parameter name> = [value, column name, (subquery), parameter]  
, @<parameter name> = [value, column name, (subquery), parameter]  
...;
```

Advanced Features, Parameter, Beispiele



Parameter, Einzelzuweisung

Deklaration

```
DECLARE @param_lastname      NVARCHAR(100);
DECLARE @param_firstname      NVARCHAR(100);
DECLARE @param_year   SMALLINT;
DECLARE @param_loops DECIMAL(10,2);
DECLARE @param_counter        INTEGER = 0;
```

Zuweisung

```
SET @param_loops = 5;
SET @param_counter = @param_counter + 1
SET @param_lastname = (SELECT
                           ISNULL( MAX( lastname ), '-undefined' )
                        FROM employee );
```

Advanced Features, Parameter, Beispiele



Parameter, Sammelzuweisung

Zuweisung

```
SELECT  
    @param_lastname = lastname  
    , @param_firstname = firstname  
    , @param_year = year( date_of_birth )  
FROM  
    employee;
```

Ausgabe

```
SELECT  
    @param_lastname AS lastname  
    , @param_firstname AS firstname  
    , lastname          AS lastname_original  
FROM  
    employee  
WHERE  
    lastname = @param_lastname  
;
```

```
SELECT  
    @param_lastname AS lastname  
    , @param_firstname AS firstname  
    , @param_year     AS year  
    , @param_loops    AS loops  
    , @param_counter  AS counter  
;
```

Advanced Features, Datentyp-Konvertierung



Datentypen können ineinander umgewandelt werden. Hier wird die explizite von der impliziten Konvertierung unterschieden. Die explizite Konvertierung ist grundsätzlich zu bevorzugen, um Seiteneffekte bei der impliziten Wandlung wie z.B. Wirkungsverlust von Indexen vorzubeugen.

Konstrukt

2 Möglichkeiten der expliziten Konvertierung sind möglich

Explizite Konvertierung 1 **CAST(<column name> AS <data type>);**

ISO-Standard

Explizite Konvertierung 2 **CONVERT(<data type>, <column name> [, <style>]);**



Implizite Konvertierung **DECLARE @<parameter name> <data type> = <value of different data type>;**

Hinweis

- Bei der impliziten Konvertierung versucht das DBMS aufgrund der Datentypanalyse von Spalten & Parametern den korrekten Datentyp für die Konvertierung zu ermitteln.

Advanced Features, Datentyp-Konvertierung



Beispiele

Deklaration

Expliziter Cast, CAST()

```
DECLARE @param_date DATE  
        = CAST( '2023-12-01' AS DATE);  
  
DECLARE @param_int INTEGER  
        = CAST( '42' AS INTEGER);  
  
DECLARE @param_string VARCHAR(10)  
        = CAST( SYSDATETIME() AS VARHCAR(10));
```

Expliziter Cast, CONVERT()

```
DECLARE @param_date DATE  
        = CONVERT( DATE, '2023-12-01');  
  
DECLARE @param_int INTEGER  
        = CONVERT( INTEGER, '42');  
  
DECLARE @param_string VARCHAR(10)  
        = CONVERT( VARHCAR(10), SYSDATETIME());
```

Abfrage

```
SELECT  
    ...  
FROM  
    employee  
WHERE  
    date_of_birth = CAST( '1971-01-09' AS DATE);
```

```
SELECT  
    ...  
FROM  
    employee  
WHERE  
    date_of_birth = CONVERT( DATE, '2023-12-01');
```

Hinweis

- Nicht konvertierbare Typumwandlungen führen zu einer SQL-Exception.
- Fehlerhandling mit **TRY-CATCH** möglich oder **TRY_CAST()**, **TRY_CONVERT()**



Advanced Features, Datentyp-Konvertierung



Beispiele

Implizite Cast

Deklaration

```
DECLARE @param_date DATE      = '2023-12-01' AS DATE);
DECLARE @param_int INTEGER    = '42';
DECLARE @param_string VARCHAR(10) = SYSDATETIME();
```

Abfrage

```
SELECT
...
FROM
    employee
WHERE
    date_of_birth = '1971-01-09';
```

Hinweis

- Die implizite Konvertierung kann zu Typ-Konvertierungsfehlern führen, da nicht explizit vorgegeben wird, welcher der korrekte Datentyp ist.
- Sind die Typkonvertierungen explizit vorgegeben, ist bekannt in welchen Datentyp zu wandeln ist.
- Durch implizite Wandlungen können Seiteneffekte auftreten wie nicht funktionierende Indexe oder Zeichenkettenlängenproblemen.

Advanced Features, Datentyp-Konvertierung



Beispiele

Deklaration

Expliziter Cast, TRY_CAST()	Expliziter Cast, TRY_CONVERT()
DECLARE @param_date DATE = TRY_CAST('2023-12-01' AS DATE);	DECLARE @param_date DATE = TRY_CONVERT(DATE, '2023-12-01');
DECLARE @param_int INTEGER = TRY_CAST('42' AS INTEGER);	DECLARE @param_int INTEGER = TRY_CONVERT(INTEGER, '42');
DECLARE @param_string VARCHAR(10) = TRY_CAST(SYSDATETIME() AS VARHCAR(10));	DECLARE @param_string VARCHAR(10) = TRY_CONVERT(VARHCAR(10), SYSDATETIME());

Abfrage

SELECT ... FROM employee WHERE date_of_birth = CAST('1971-01-09' AS DATE);	SELECT ... FROM employee WHERE date_of_birth = CONVERT(DATE, '2023-12-01');
---	--

Hinweis

- Nicht konvertierbare Typumwandlungen werden als NULL-Wert zurückgegeben.
- SQL-Exceptions nicht erzeugt



Advanced Features, Datentyp-Konvertierung



Ursprung

Formatierung, FORMAT()

- Spezialisierte Funktion zur Formatierung von Zeichenketten inkl. Culture-Values

Definition

FORMAT(<value>, <format> [, <culture>]);

Beispiel

```
DECLARE @param_date_string AS NVARCHAR(40)  
= FORMAT(SYSDATETIME(), 'd', 'de-DE');
```

Culture - Kurzform: de, Langform de-DE, de_DE

Aufgabe

Erstelle für die Länder Deutschland, USA und Schweden mit der Formatierungsfunktion eine Ausgabe mit den landesspezifischen Datumsformatierungen in Kurzform, Langform. Zusätzlich soll ein Dezimalwert in den landesspezifischen Nummern-Separatoren (Dezimalkomma, Tausender-Separator) dargestellt werden sowie ein ganzzahliger Wert mit einer individuellen Nummernmaskierung.

Formatierung, CONVERT()

- Bei Typwandlung direkte Formatierung insbesondere bei Zeichenketten

CONVERT(<data type>, <column name> [, <style>]);

```
DECLARE @param_date_string AS NVARCHAR(40)  
= CONVERT(NVARCHAR(40), SYSDATETIME(), 104);
```

T-SQL



Advanced Features, CASE-Kontrollstruktur



Die Kontrollstruktur **CASE-WHEN-ELSE** ermöglicht **Fallunterscheidungen als INLINE-Kontrollstruktur** in SQL-Statements. Ein CASE-Block kann **beliebig viele WHEN-Zweige** beinhalten sowie einen **ELSE-Zweig** zur Rückgabe eines Default-Wert. Die CASE-Anweisung steht nicht als Blockanweisung zur Verfügung.

Konstrukt

Mindestens eine WHEN-Anweisung mit optionalem ELSE-Zweig

```
SELECT  
  CASE  
    WHEN <condition 1> THEN [<value> | <column name>]  
    WHEN <condition ...> THEN [<value> | <column name>]  
    WHEN <condition N> THEN [<value> | <column name>]  
    ELSE [<value> | <column name>]  
  END AS <column name>  
  , <column name ...>  
  , <column name N>  
FROM  
  <table name>  
;
```

- WHEN-Zweige werden „**sequentiell**“ geprüft, bis eine Bedingung zutrifft
- Trifft keine Bedingung zu wird der ELSE-Zweig verwendet (Default)
- Ist kein ELSE-Zweig vorhanden ist der Spaltenwert NULL

Advanced Features, CASE-Kontrollstruktur, Beispiel



CASE-WHEN

```
SELECT
CASE
    WHEN YEAR(SYSDATETIME()) < 1900           THEN 1
    WHEN YEAR(SYSDATETIME()) BETWEEN 1900 AND 2000 THEN 2
    WHEN YEAR(SYSDATETIME()) BETWEEN 2000 AND 2023 THEN 3
END AS year_category
;
```

CASE-WHEN-ELSE

```
SELECT
CASE
    WHEN YEAR(SYSDATETIME()) < 1900           THEN 1
    WHEN YEAR(SYSDATETIME()) BETWEEN 1900 AND 2000 THEN 2
    WHEN YEAR(SYSDATETIME()) BETWEEN 2000 AND 2023 THEN 3
    ELSE 0
END AS year_category
;
```

Advanced Features, IF-ELSE-Kontrollstruktur



Die Kontrollstruktur **IF-ELSE** ermöglicht die **Verzweigung von SQL-Skripten** entsprechend zu anderen Programmiersprachen. Das IF-ELSE-Struktur kann nur **als BLOCK-Anweisung** verwendet werden **nicht als INLINE-Anweisung**, als INLINE-Anweisung steht **IIF()** zur Verfügung.

Konstrukt

2 IF-ELSE-Konstrukte möglich, Standard-Anweisung und Inline-Anweisung

Standard-Anweisung

```
IF <condition>
[BEGIN]
<sql command 01>
<sql command ...>
[END]
[ELSE]
[BEGIN]
<sql command 02>
<sql command ...>
[END];
```

Inline-Anweisung

IIF(<condition>, <true value>, <false value>)

Advanced Features, IF-ELSE-Kontrollstruktur, Beispiel



IF-ELSE ohne Blockanweisung

```
DECLARE @param_if_value  INTEGER = 2;  
  
IF @param_if_value = 1  
    PRINT 'IF true-CASE, @param_if_value: ' + CAST(@param_if_value AS NVARCHAR(10));  
ELSE  
    PRINT 'IF false-CASE, @param_if_value: ' + CAST(@param_if_value AS NVARCHAR(10));  
;
```

IF-ELSE mit Blockanweisung

```
DECLARE @param_if_value  VRACHAR(10) = 'case with true';  
IF @param_if_value LIKE '%true'  
    BEGIN  
        PRINT 'IF true-CASE, @param_if_value: ' + @param_if_value;  
    END  
ELSE  
    BEGIN  
        PRINT 'IF false-CASE, @param_if_value: ' + @param_if_value;  
    END  
;
```

Advanced Features, WHILE-Kontrollstruktur



!

Zur Abbildung von **Schleifenkonstrukten** steht in SQL die **kopfgesteuerte WHILE-Schleife** zur Verfügung. Damit können Blöcke wiederholt ausgeführt werden. Die Schleife wird so lang durchlaufen wie die Schleifenbedingung WAHR ist.

Konstrukt

WHILE-Schleife

WHILE, die kopfgesteuerte SQL-Schleife

```
WHILE <condition>
BEGIN
    <command 01>
    <command ...>
    [BREAK;]
END;
```

Beispiel

```
DECLARATION @param_counter INTEGER = 0;

WHILE ( @param_counter < 1000 )
BEGIN
    PRINT 'while-loop, counter: ' + CAST(@param_counter AS VARCHAR(4))
END;
```

Advanced Features, Computed Columns



Computed Columns sind berechnete Spalten, die in einer Tabellenstruktur bereitgestellt werden können. Die berechneten Spalten leiten ihren **Inhalt aus bestehenden Spalten** ab. Diese spezialisierten Spalten belegen **keinen Speicherplatz**.

Datentypen

Steht allen Datentypen bereit

Spaltenanlage

```
CREATE TABLE <name> (
    <column name> AS ( computed value )
    , ...
);
```

Alternativanlage

```
ALTER TABLE <name>
ADD COLUMN <column name> AS ( computed value );
```

- Anlage, Änderung, Löschung der Spalte analog Standard-Spalten (DDL)
- Befüllung der Spalte per INSERT oder UPDATE ist nicht zulässig

Advanced Features, Computed Columns



Computed Columns

Erstellung

```
CREATE TABLE employee (
    id          BIGINT
    ,date_of_birth DATE
    ,year_of_birth AS YEAR(date_of_birth)
);
```

Computed Columns, Persisted

```
CREATE TABLE employee (
    id          BIGINT
    ,date_of_birth DATE
    ,year_of_birth AS YEAR(date_of_birth) PERSISTED
);
```

T-SQL



Vorteile

- Abgeleitete Attribute können direkt dargestellt werden
- Zusätzliche Spalten benötigen keinen Speicherbedarf
- Bereitstellung der Werte optimiert, da Funktionen in Struktur kompiliert vorliegend
- Entfall von INSERT- / UPDATE-Trigger zur Belegung abgeleitete Attribute

PERSISTED (T-SQL)

- Speicherung ermittelter Werte, geeignet für umfangreichen Operationen
- Verbesserung der Performance durch Speicherung der Ergebniswerte
- Ergebniswerte werden durch INSERT, UPDATE, DELETE aktualisiert

Advanced Features, Sequences



Nummerierungen können als **eigenständige Datenbankstrukturen** in Form von **SEQUENCES** angelegt werden, um den Bedarf an individuellen Zählern zu unterstützen. Dabei können eigenständige Startwerte sowie Schrittweiten definiert werden.

Konstrukt

Sequence

```
CREATE SEQUENCE <sequence name>
    START WITH      <start no>
    INCREMENT BY   <steps>
    MINVALUE        <min value>
    MAXVALUE        <max value>
;
```

Typ-Wert-Funktion

```
CREATE SEQUENCE seq_employee_batch
    START WITH      1000
    INCREMENT BY   1000
;
-- Nächster Zählerwert
SELECT NEXT VALUE FOR seq_employee_batch;
-- Sequence-Informationsübersicht
SELECT * FROM sys.sequences;
```

Hinweis

- Die Neuanlage von Sequences oder Änderung der Start-Werte sollte immer unter Berücksichtigung des Verwendungs-Kontexts erfolgen, um mögliche Unique-Constraints nicht zu verletzen.

Advanced Features, Temporäre Datenhaltung



Erstellung

Computed Columns

```
CREATE TABLE employee (
    id          BIGINT
    , date_of_birth DATE
    , year_of_birth AS YEAR( date_of_birth )
);
```

Computed Columns, Persisted

```
CREATE TABLE employee (
    id          BIGINT
    , date_of_birth DATE
    , year_of_birth AS YEAR( date_of_birth ) PERSISTED
);
```



Vorteile

- Abgeleitete Attribute können direkt dargestellt werden
- Zusätzliche Spalten benötigen keinen Speicherbedarf
- Bereitstellung der Werte optimiert, da Funktionen in Struktur kompiliert vorliegend
- Entfall von INSERT- / UPDATE-Trigger zur Belegung abgeleitete Attribute

PERSISTED (T-SQL)

- Speicherung ermittelter Werte, geeignet für umfangreichen Operationen
- Verbesserung der Performance durch Speicherung der Ergebniswerte
- Ergebniswerte werden durch INSERT, UPDATE, DELETE aktualisiert

Advanced Features, Temporäre Datenhaltung



■ Temporäre Datenhaltung

- Nicht alle Prozessdaten / Batch-Daten müssen persistiert werden
- **Zwischenergebnisse** sind in der Regel **nur temporäre** zur Erreichung der Zielerreichung relevant
- **Zwischenergebnisse** können **nach** der Zielerreichung wieder **entfernt** werden.



Temporäre Tabellen



Common Table Expression

Advanced Features, Temporäre Tabellen



Temporäre Tabellen stellen eigenständige Tabellenstrukturen dar, welche identische Eigenschaften aufweisen wie dauerhaft persistierte Tabellen, sind jedoch an eine vorhandene Benutzer-Session gebunden.

Typen

Temporäre Tabellen, Globale temporäre Tabellen

Temporäre Tabellen

`CREATE TABLE #<name> (<column [, column]>);`

Globale Temporäre Tabellen

`CREATE TABLE ##<name> (<column [, column]>);`

Löschung

Temporäre Tabellen, Globale temporäre Tabellen

Temporäre Tabellen

`DROP TABLE #<name>;`

Globale Temporäre Tabellen

`DROP TABLE ##<name>;`



Kein ISO-Standard
Oracle abweichend

- Anlage, Änderung, Löschung analog Standard-Tabelle (DDL)
- Indexe auf temporäre Tabelle möglich
- Erstellung per Kurzreferenz möglich (T-SQL)

Advanced Features, Temporäre Tabellen



temporäre Tabelle

Erstellung

```
CREATE TABLE #temp_local (
    lastname NVARCHAR(100)
    ,firstname NVARCHAR(100)
);
INSERT INTO #temp_local
SELECT lastname, firstname
FROM employee;
```

Kurzerstellung

```
SELECT lastname, firstname
INTO #temp_local
FROM employee;
```

Hinweis

- Jede Spalte muss einen Spaltennamen besitzen, Fix-Werte sind mit einem Spalten-Alias zu benennen.
- Die Datentypen der temporären Tabelle werden von den Datentypen der angegebenen Spalten abgeleitet.
- Bei NULL-Spalten ist der Datentyp durch einen expliziten CAST() vorzugeben, z.B. CAST(NULL AS DECIMAL(10,2))

globale temporäre Tabelle

```
CREATE TABLE ##temp_global (
    lastname NVARCHAR(100)
    ,firstname NVARCHAR(100)
);
INSERT INTO #temp_local
SELECT lastname, firstname
FROM employee;
```

```
SELECT lastname, firstname
INTO ##temp_global
FROM employee;
```

Advanced Features, Temporäre Tabellen



temporäre Tabelle

Verfügbarkeit

- Stehen nur innerhalb der aktuellen Session zur Verfügung
- Fremde Sessions können nicht darauf zugreifen

Löschen (autom.)

- Die temporären Tabellen werden grundsätzlich gelöscht, wenn die **erzeugende Session beendet** wird, ungeachtet, ob die Tabelle von einer Fremd-Session im Zugriff ist.
- Verwaltungshoheit liegt ausschließlich bei der erzeugenden Session.

Speicherung

- Verwendung eines **eigenen Systemdatenbank-Schemas für temporäre Tabellen**.
- T-SQL: „tempdb“ verwaltet alle temporären Tabellen inkl. Zugehöriger Objekte wie Constraints, Trigger, Indexe

globale temporäre Tabelle

- Stehen global in der Datenbank zur Verfügung
- allen Sessions mit Zugriff auf die Datenbank, können diese verwenden

Advanced Features, Temporäre Tabellen



temporäre Tabelle

globale temporäre Tabelle

Vorteile

- Zwischenspeicherung von **reduzierten Ergebnismengen** aus umfangreichen Basistabellen mit hohem Datenvolumen
- Erstellung neuer **Submengen für weiterführende JOIN-Relationen**
- **Vermeidung imperfizienter Subqueries**
- **Optimierung von selbstreferenzierenden JOIN-Relationen** durch temporäre Sub-Mengen
- **Daten nur temporär** und wirken nicht auf die Speicherreservierung der Datenbank ein
- Automatische Tabellenentfernung bei Session-Schließung

Advanced Features, Temporäre Tabellen



temporäre Tabelle

globale temporäre Tabelle

Hinweis

- Im temporären Datenbankbereich können nicht nur Tabellen temporär abgelegt werden, dies gilt auch für andere Datenbank-Inhalte wie Prozeduren, Funktionen, etc.
- Grundsätzlich ist dem **Daten-Entwickler** die Pflicht auferlegt, die **Zugriffssicherheit** aufgrund der temporären Verfügbarkeit sicherzustellen, insbesondere bei „globalen temporären Objekten“.
- Die Anlage einer temporären Tabelle benötigt den gleichen zeitlichen Aufwand wie die Anlage einer Standard-Datenbanktabelle.

Advanced Features, Common Table Expression



!

Common Table Expression bildet eine eigenständige Datenstruktur ab, welche unterschiedliche Sub-Ergebnisse bilden kann, die wiederum in einer Ergebnisabfrage zusammengefasst werden können. Ziel ist die Strukturierung von komplexen Abfragen sowie die Abbildung von rekursiven Abfragen.

Konstrukt

```
WITH <result name 01> AS (
    <SELECT command 01>
)
[ ,<result name 02> AS (
    <SELECT command 01>
)]
SELECT
...
FROM
    <table name>
    ,<result name 01>
[,<result name 02>] ;
```

Advanced Features, Common Table Expression



■ CTE, strukturgeben

- CTEs besitzen 1 bis N Sub-Queries.
- Das Ergebnis der jeweilige Sub-Query steht in der nachfolgenden Query direkt zur Abfrage zur Verfügung.
- CTE müssen mit einem SELECT abgeschlossen werden.

■ CTE, rekursiv

- Bei einer Rekursion wird ein Sub-Query des CTE wiederholt aufgerufen.
- Das aufgerufenen rekursive Sub-Query muss eine Abbruchbedingung enthalten.

Hinweis

- CTEs sind unter PL/SQL die bessere Alternative im Vergleich zu temporären Tabellen
- Die Anwendung von temporären Tabellen ist ohne weiteres möglich, jedoch deutlich aufwendiger als unter T-SQL.



CTEs sind eine **Sonderform der temporären Tabellen**, da die Ergebnisdaten den verbleibenden Sub-Queries zur Verfügung stehen. Die **Ergebnisdaten** der Sub-Queries stehen ausschließlich diesem CTE zur Verfügung und nicht externen Abfragen „**private data**“. Die Daten werden nach der finalen Abfrage gelöscht.

Advanced Features, Common Table Expression



Beispiel 01 CTE, strukturgeben

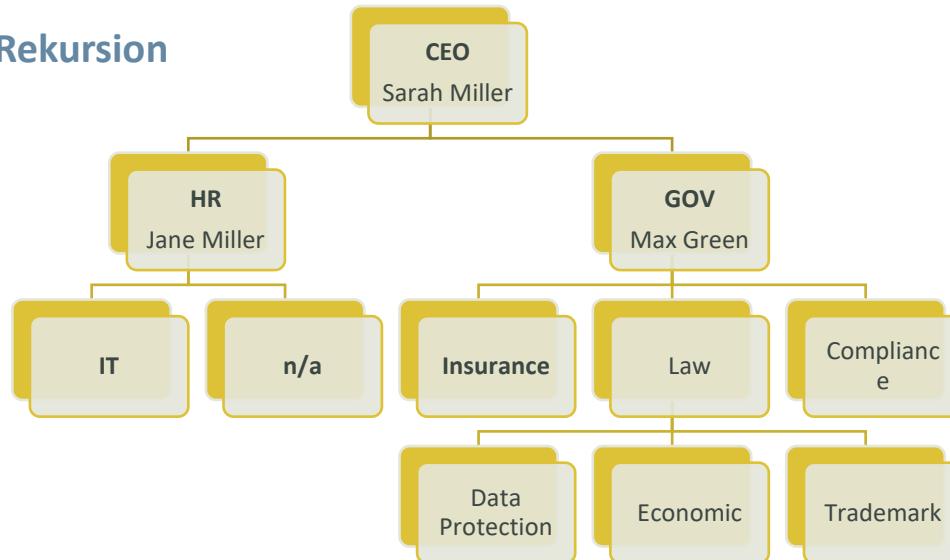
```
WITH cte_subq_curdate AS (
    SELECT
        SYSDATETIME() AS cur_date
)
, cte_subq_employee AS (
    SELECT
        COUNT(1) AS employee_count
    FROM
        study.dbo.employee
)
SELECT
    cte_em.employee_count
    , cte_cd.cur_date
FROM
    cte_subq_curdate cte_cd, cte_subq_employee cte_em
;
```

Advanced Features, Common Table Expression



Die Abbildung von **Hierarchien** erfolgt im Regelfall durch **baumartige Strukturen**. In diesen Strukturen herrscht eine **streng geregelte Mutter-Kind-/Vater-Kind-Beziehung**. Diese Baumstrukturen können am effektivsten durch **rekursive Algorithmen** behandelt werden.

Bäume & Rekursion



ID	Name	Department	Parent-ID
124	John Doe	IT	135
135	Jane Miller	HR	146
136	Max Green	Governance	146
146	Sarah Smith	CEO	NULL
400	Paul Newman	Insurance	136
401	Ina New	Law	136
402	Christi Next	Compliance	136
500	Paul Newman	Data Protection	401
501	Ina New	Economic	401
502	Christi Next	Trademark	401

Advanced Features, Common Table Expression



Beispiel 02 CTE, rekursiv

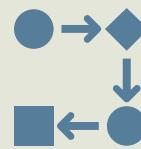
```
WITH employee_manager_cte AS (
    SELECT
        root.id
        , root.name
        , root.department
        , root.manager_id
        , 1 AS level
    FROM
        #tmp_employees root
    WHERE
        root.manager_id IS NULL
    UNION ALL
    SELECT
        e.id
        , e.name
        , e.department
        , e.manager_id
        , r.level + 1
    FROM
        #tmp_employees      e
    JOIN
        employee_manager_cte r ON e.manager_id = r.id
)
SELECT
    *
FROM
    employee_manager_cte;
```

Advanced Features, Funktionen & Prozeduren

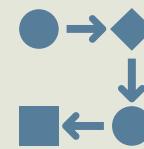


■ Modularität

- Für die Wiederverwendung von Abläufen und Algorithmen stehen im SQL-Umfeld Funktionen & Prozeduren zur Verfügung
- **Funktionen** sind dadurch definiert, dass die Funktion selbst ein Ergebnis zurückgibt.
- **Prozeduren** können unterschiedlich viele Ergebnisse zurückgeben.
- Beiden Methoden können beliebig viele Parameter entgegen nehmen.



USER DEFINED FUNCTION



STORED PROCEDURE

Advanced Features, User Defined Functions



Die **User Defined Functions** stellen eigenständige Funktionen dar, die im Rahmen der SQL-Statements **inline** verwendet werden können. Dabei gibt eine Funktion immer **einen Wert zurück**, der ein **Skalar-Wert** oder auch ein **Ergebnisset** einer Abfrage sein kann.

Konstrukt

Skalarwert-Funktion

```
CREATE OR ALTER FUNCTION <function name>
( [<parameter declaration>]
RETURNS <scalar value type>
AS
BEGIN
    <command 01>
    <command ...>
    RETURN <return value>;
END;
```

Hinweis

- Bei Funktionen sind die Parameterklammern immer erforderlich
- Die aufgeführten Strukturen sind die auf das Wesentliche reduzierte Funktionsdefinitionen.

Typ-Wert-Funktion

```
CREATE OR ALTER FUNCTION <function name>
( [<parameter declaration> ]
RETURNS TABLE
AS
RETURN (
    SELECT * FROM...
)
```

Advanced Features, User Defined Functions



Skalarwert-Funktion

Beispiele

```
USE study;

CREATE OR ALTER FUNCTION dbo.fn_department_nr_format(
    @param_id INTEGER)
RETURNS VARCHAR(100)
AS
BEGIN
    DECLARE @var_result VARCHAR(100);
    SET @var_result = CAST(@param_id AS VARCHAR(10));
    RETURN @var_result;
END;

SELECT study.dbo.fn_department_nr_format(1) AS number;
```

Typ-Wert-Funktion

```
USE study;

CREATE OR ALTER FUNCTION dbo.fn_department_tree_data()
RETURNS TABLE
AS
RETURN (
    SELECT *
    FROM
        study.dbo.department_tree
);
```



```
SELECT * FROM dbo.fn_department_tree_data();
```

Advanced Features, Stored Procedure



Die **Stored Procedures** stellen analog zu den *User Defined Functions* eigenständige Datenbankobjekte dar, die eine unterschiedliche Anzahl von Parametern empfangen und zurückgeben kann. Der Unterschied zu den Funktionen neben den Rückgabewerten, besteht im Aufruf, der **nicht Inline in SQL-Statements** erfolgen kann.

Konstrukt

Stored Procedure

```
CREATE OR ALTER PROCEDURE <procedure name>
( [<parameter declaration> [ IN [| INPUT] ]
  , [<parameter declaration> [ OUT [| OUTPUT] ]
)
AS
BEGIN
    <command 1>
    <command ..>
    SET @param_... = <result value>;
END;
```

```
DECLARE @param_in NVARCHAR(100) = 'in';
DECLARE @param_out NVARCHAR(100) = 'out';

EXEC <procedure name>
    [<param_in_1> ] [ IN ]
    , [<param_in_...> ] [ INPUT ]
    , [<param_out_1> ] OUT
    , [<param_out_...>] OUTPUT
;
```



A

Aufgabe

Erstellen Sie je eine **Stored Procedure** mit dem Ziel per Übergabeparameter einen der 3 Werte als Rückgabe bereitzustellen:

- DB-Version `@@VERSION`
- Servername `@@SERVERNAME`
- Microsoft Version `@@MICROSOFTVERSION`
- Der Rückgabewert soll vor Beendigung mit dem PRINT-Befehl ausgegeben werden.

Erstellen Sie zum Vergleich die gleiche Funktionalität in einer **UDF** und vergleichen Sie beide Umsetzungen.

Führen Sie an, welche Gemeinsamkeiten und Unterschiede bestehen, welche Besonderheiten bei dem PRINT-Befehl bestehen?

Advanced Features, Stored Procedure, Beispiel



Beispiele

```
CREATE OR ALTER PROCEDURE dbo.sp_version_details (
    @param_detail_mode INTEGER
    , @param_result NVARCHAR(100) OUT
)
AS
BEGIN
    SET NOCOUNT ON;
    PRINT 'sp, parameter: @param_detail_mode = ' + CAST(@param_detail_mode AS VARCHAR(10));

    IF @param_detail_mode      = 2
        SET @param_result = 'Server name: ' + @@SERVERNAME;

    ELSE IF @param_detail_mode     = 3
        SET @param_result = 'Microsoft version: ' + CAST(@@MICROSOFTVERSION AS VARCHAR(10));

    ELSE
        SET @param_result = 'Version: ' + @@version;

    PRINT 'sp, return value, ' + @param_result;
END;
GO

DECLARE @param_result_value VARCHAR(100) = '-empty-';
EXEC sp_version_details_2 1, @param_result_value OUTPUT;
SELECT @param_result_value AS param_result_value;
GO
```

Advanced Features, UDF, Beispiel



Beispiele

```
CREATE OR ALTER FUNCTION dbo.fn_version_details(
    @param_detail_mode INTEGER
)
RETURNS NVARCHAR(100)
AS
BEGIN
    DECLARE @param_result NVARCHAR(100);

    IF @param_detail_mode = 2
        SET @param_result = 'Server name: ' + @@SERVERNAME;

    ELSE IF @param_detail_mode = 3
        SET @param_result = 'Microsoft version: ' + CAST(@@MICROSOFTVERSION AS VARCHAR(10));

    ELSE
        SET @param_result = 'Version: ' + @@VERSION;

    RETURN (@param_result);
END;
GO

SELECT dbo.fn_version_details(3) AS version_details;
```

Advanced Features, String-Split & -Merge



■ STRING_SPLIT

- Aufteilung einer Zeichenkette anhand eines spezifischen Zeichens
- Das Ergebnis wird als Ergebnistabelle (= Result Set) zurückgegeben



```
SELECT VALUE FROM STRING_SPLIT (<column name>, ',');
```

■ STRING_AGG

- Zusammenfassung von mehreren Werten zu einem Gesamtwert
- Verkettung anhand eines spezifischen Zeichens
- Die Funktion erfordert **immer ein GROUP BY**

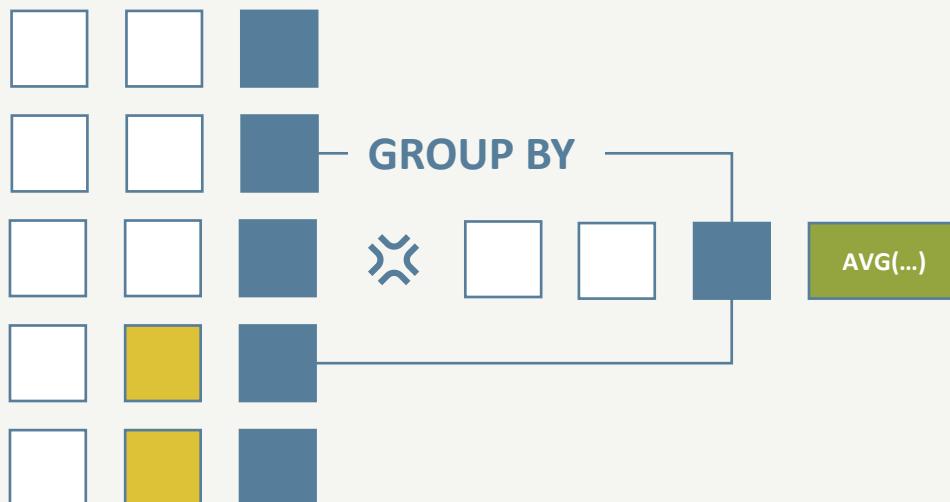
```
SELECT STRING_AGG(<column name>, ',') FROM <table> GROUP BY <column name>;
```

Advanced Features, Window-Funktionen



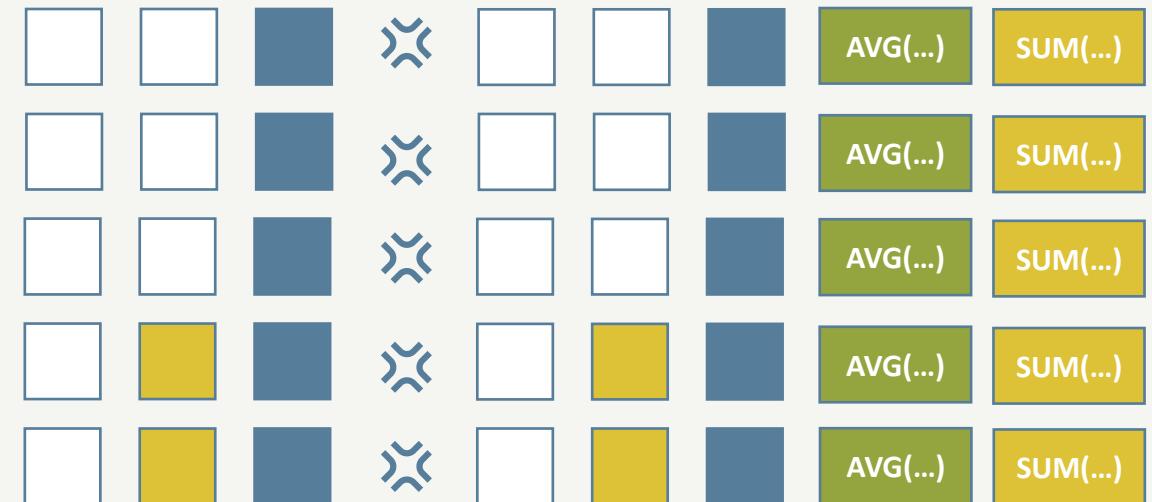
Bisher galt, zur Durchführung von **Gruppierungsfunktionen**, dass dies ausschließlich unter Verwendung der „**GROUP BY**“-Klausel möglich ist.

Group Functions



- “**Gruppierungs-Funktionen**” werden ausschließlich auf gruppierte Datengruppen angewendet

Window Functions

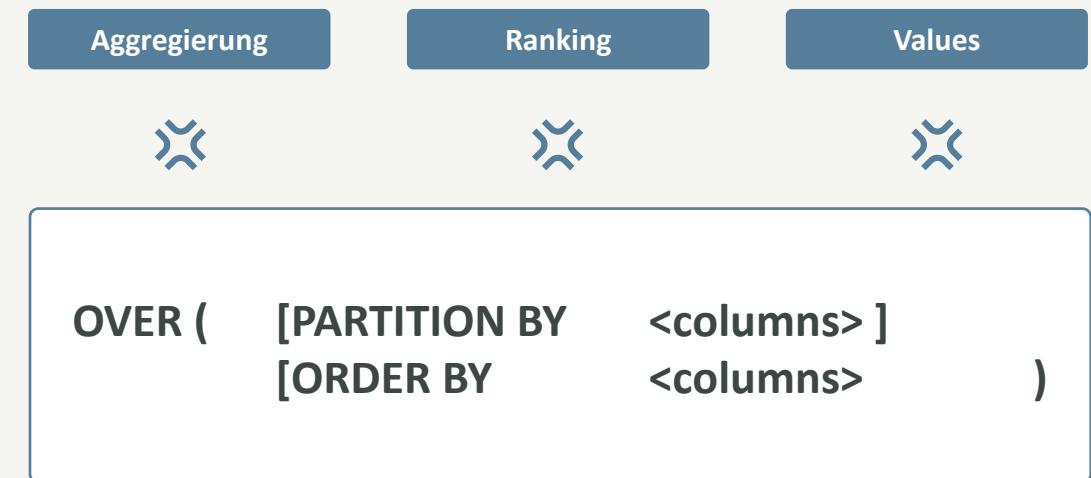
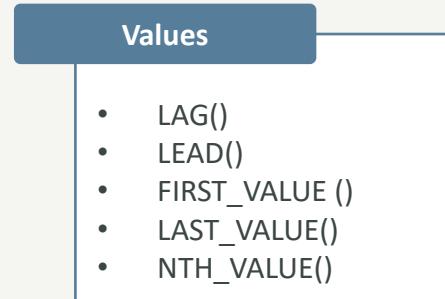


- “**Window Funktionen**” werden dynamisch auf jede Datenzeile angewendet ohne Haupt-Gruppierung

Advanced Features, Window-Funktionen



Bisher galt, zur Durchführung von **Gruppierungsfunktionen**, dass dies ausschließlich unter Verwendung der „**GROUP BY**“-Klausel möglich ist.



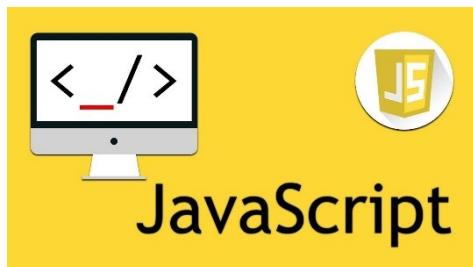
- **PARTITION BY**, Gruppierung innerhalb der Fensterfunktion
- **ORDER BY**, Sortierung innerhalb der Fensterfunktion

Advanced, JSON-Funktionen



In **Zeichenkettenfeldern** können genutzt werden, um strukturierte Daten abzuspeichern. Dies kann genutzt werden, um **JSON-Daten auf Datenbankebene abzulegen**. Unter **T-SQL stehen spezialisierte Funktionen** zur Verfügung, um diese Datenstrukturen auswertbar zu machen.

JSON – JavaScript Object Notation



- JavaScript ist die Programmiersprache für Browser-Content
- JavaScript dient der HTML-DOM-Manipulation sowie der dynamischen Ausgabe auf HTML-Ebene
- JSON ist eine Strukturabbildung eines JavaScript-Objektes
- JSON-Strukturen ermöglichen eine standardisierte Konvertierung der Struktur in JavaScript-Objekte sowie zurück, Interfaces **JSON.parse() & JSON.stringify()**



Advanced, JSON-Funktionen



JSON – JavaScript Object Notation

JSON-Struktur

```
`{"employees": [  
    { "firstName":"John", "lastName":"Doe" }  
    , { "firstName":"Anna", "lastName":"Smith" }  
    , { "firstName":"Peter", "lastName":"Jones" }  
]}
```

- Speicherung als NVARCHAR(N) möglich
- Keine gesonderter Datentyp wie z.B. bei XML

```
`{"external_numbers": [  
    { "type":"passport", "number":"987-654-321" }  
    , { "type":"phone_stationary", "number":"0690-9874-6523" }  
    , { "type":"phone_mobile", "number":"0177-989896" }  
]}
```



JSON-Strukturen finden teilweise Anwendung, um die Anlage von neuen Spalten und Entitäten zu vermeiden. Damit erfolgt eine direkte Verletzung der Normalformen, da die **Atomarität nicht mehr gegeben** ist. Zudem können die **Inhalte je Zeile unterschiedliche Strukturen** aufweisen.

Advanced, JSON-Funktionen



JSON – Inhaltszugriffe

JSON-Funktionen

ISJSON(<column name> **)**

- Prüfung, ob Inhalt die JSON-Kriterien erfüllt
- Rückgabe eines JSON-Werte, kann keine JS-Werte zurückgeben
- Rückgabe eines JS-Set (Array), kann keinen Einzelwert zurückgeben

JSON_VALUE(<column name>, <**search path**> **)**

JSON_QUERY(<column name>, <**search path**> **)**

Search path

\$.

- Root der JS-Struktur
- Ebenen-Zugriff
- Element-Array-Zugriff
- Array-Element-Zugriff

\$.external_numbers

\$. external_numbers[0]

\$. external_numbers[0].type



JSON – Array-Iteration

JSON-Funktionen

`OPENJSON(<column name>, <loop path>)`

- Iteration über ein JS-Array

Loop path

`lax $.<path>'`

`strict $.<path>'`

- Rückgabe der Pfad-Zeile, bei Fehler erfolgt Rückgabe NULL
- Rückgabe der Pfad-Zeile, bei Fehler erfolgt ERROR-Meldung

Advanced, JSON-Funktionen, Beispiele



JSON – Beispiel: Inhaltszugriffe

```
SELECT
    row_type
    , employees
    , external_numbers
    , JSON_VALUE(employees,      '$.employees'          ) AS json_value_employees
    , JSON_VALUE(external_numbers, '$.external_numbers' ) AS json_value_external_numbers
    , JSON_QUERY(employees,       '$.employees'          ) AS json_query_employees
    , JSON_QUERY(external_numbers, '$.external_numbers' ) AS json_query_external_numbers
    , JSON_VALUE(employees,      '$.employees[0]'        ) AS json_value_employees
    , JSON_VALUE(external_numbers, '$.external_numbers[0]' ) AS json_value_external_numbers
    , JSON_QUERY(employees,       '$.employees[0]'        ) AS json_query_employees
    , JSON_QUERY(external_numbers, '$.external_numbers[0]' ) AS json_query_external_numbers
    , JSON_VALUE(employees,      '$.employees[0].lastName' ) AS json_value_employees
    , JSON_VALUE(external_numbers, '$.external_numbers[0].number' ) AS json_value_external_numbers
    , JSON_QUERY(employees,       '$.employees[0].lastName' ) AS json_query_employees
    , JSON_QUERY(external_numbers, '$.external_numbers[0].number' ) AS json_query_external_numbers
FROM
    #tmp_json
;
```



Advanced, JSON-Funktionen, Beispiele



JSON – Beispiel: Array-Iteration

```
DECLARE @json NVARCHAR(MAX);

SELECT
    JSON_VALUE( value, '$.firstName' ) AS json_loop_emp_firstname
    , JSON_VALUE( value, '$.lastName' ) AS json_loop_emp_lastname
FROM
    OPENJSON( @json, N'!ax $.employees')
;
```



Advanced Features, Systeminformationen I



Eigens erstellte Datenbankobjekte werden in Systemtabellen dokumentiert und können abgefragt werden. Der zentrale Dokumentationsbereiche für **eigene Objekte** ist das **INFORMATION_SCHEMA** unter T-SQL.

INFORMATION_SCHEMA

INFORMATION_SCHEMA.TABLES

INFORMATION_SCHEMA.COLUMNS

INFORMATION_SCHEMA.ROUTINES

INFORMATION_SCHEMA.PARAMETERS

- Metadaten der Tabellen
- Metadaten aller Spalten tabellenunabhängig
- Metadaten erstellter Funktionen und Prozeduren
- Metadaten aller Parameter aus Funktionen und Prozeduren



Abfrage der Metadaten (per SELECT)

```
SELECT * FROM INFORMATION_SCHEMA.TABLES;
```

Advanced Features, Systeminformationen II



Alle Datenbankobjekte werden in Systemtabellen dokumentiert und können über System-Views abgefragt werden. Die **zentralen Systemsichten** sind unter T-SQL im Schema **SYS** zusammengefasst.

SYS

SYS.SYSOBJECTS

SYS.ALL_VIEWS

SYS.DM_DB_LOG_SPACE_USAGE

SYS.SYSLANGUAGES

- Metadaten aller Objekte (Tabellen, Prozeduren, etc.)
- Metadaten aller Sichten
- Metadaten des verwendeten Log-Spaces
- Metadaten zu integrierten Sprachen



Abfrage der Metadaten (per SELECT)

```
SELECT * FROM SYS.SYSLANGUAGES;
```