

Zusammenfassung SQL 17.06.24

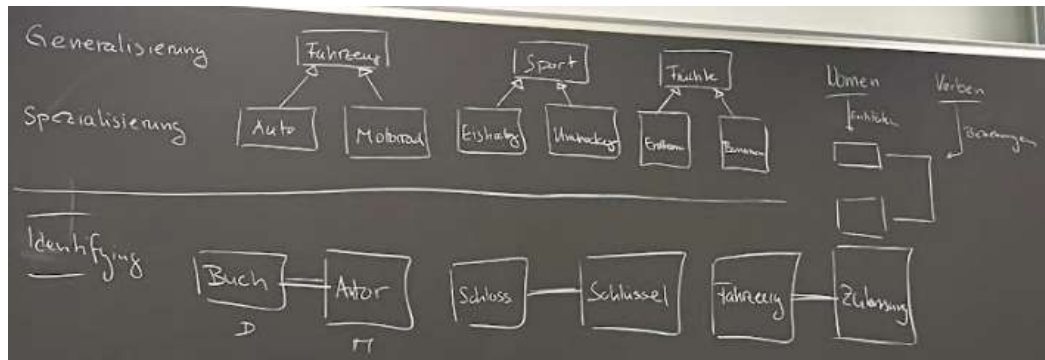
Generalisierung/Spezialisierung

Erklärung -> Beruht auf Attribut Konzept, man definiert Attribute mit Attribut Ausprägungen und ordnet diesen Entitätstyp zu. Wenn mehrere Entitätstypen Attribute gemeinsam haben, kann Redundanz entstehen. Beispiel ist, dass Mitarbeiter auch als Kunden auftreten oder Fahrer als Disponenten. *Lösung ist, dass alle gemeinsamen Attribute in einem allgemeinen Entitätstyp zusammengefasst werden (Generalisierung). Die nicht gemeinsamen Attribute verbleiben den Entitätstypen (Spezialisierung).*

Beispiele:

Früchte-Bananen-Erdbeeren, Fahrzeuge-Auto-Motorrad

Arzt-Augenarzt-Ohrenarzt



Beziehungsarten Identifying und Non-Identifying

Die Parent Tabelle ist die erste und die zweite ist immer Child Tabelle. Child Tabelle hat den Primary Key der Parent Tabelle als FK. *Je nach Zusammensetzung von Primary Key der Child Tabelle ist eine identifying oder identifying Beziehung.*

Identifying-Relationship - Foreign Key ist Teil des Primary Key der Child Tabelle. ID ist also aus mehreren (mind. 2) Attributen zusammengesetzt. Beispiel Person und Ausweis

```
CREATE TABLE Kunde ( KundenID INT PRIMARY KEY, Kundenname VARCHAR(100) );  
CREATE TABLE Bestellung ( BestellID INT, KundenID INT, Bestelldatum DATE, PRIMARY KEY  
(BestellID, KundenID), FOREIGN KEY (KundenID) REFERENCES Kunde(KundenID) );
```

Non-Identifying Relationship - Foreign Key ist KEIN Teil vom Primary Key der Child Table.

Fremdschlüssel auf Arbeitgeber zb gehört nicht zu Identität von Person, weil Arbeitgeber wechseln kann.

```
CREATE TABLE Kunde ( KundenID INT PRIMARY KEY, Kundenname VARCHAR(100) );  
CREATE TABLE Bestellung ( BestellID INT PRIMARY KEY, KundenID INT, Bestelldatum DATE,  
FOREIGN KEY (KundenID) REFERENCES Kunde(KundenID) );
```

DBMS (Datenbank Management System)

System zur *elektronischen Datenverwaltung*, Aufgabe ist grosse Datenmengen effizient, widerspruchsfrei und dauerhaft zu speichern und bedarfsgerechte Darstellungen für Benutzer bereitzustellen.

2 Teile: *Verwaltungsteil (Datenbankmanagementsystem) und Menge der zu verwaltenden Daten (Datenbank).* Verwaltungssoftware organisiert strukturierte Speicherung und kontrolliert Zugriffe.

Verschiedene Formen: Art und Weise, wie das System Daten speichert und verwaltet wird, wird durch Datenbankmodelle dargestellt. Bekannteste Form relationales Datenbanksystem.

Merkmale DBMS

Integrierte Datenhaltung - ermöglicht eine einheitliche Verwaltung von Daten. Jedes logische Datenelement wie Name des Kunden z.B. wird nur an 1 Stelle in der Datenbank gespeichert. DBMS muss Vielzahl komplexer Beziehungen zwischen Daten definieren und Daten schnell und effizient verknüpfen.

Sprache - *Stellt Datenbanksprache (query Language) zur Verfügung*

- Datenanfrage (Data Query/Retrieval Language, DQL/DRL -> **SELECT**)
- Verwaltung der Datenstruktur (Data Definition Language, DDL → **CREATE TABLE, DROP TABLE, ALTER TABLE (Change, rename, add column, modify) CREATE SCHEMA**,
- Datenmanipulation (Data Manipulation Language, DML → **INSERT**),
- Berechtigungssteuerung (Data Control Language, DCL → **GRANT**),
- Transaktionen (Transaction Control Language, TCL → **COMMIT**).

Viele Datenbanken, verschiedene Benutzer mit unterschiedlichen Kenntnissen, sollen eine Vielzahl von GUIs vorhanden sein.

Benutzersichten: unterschiedliche Klassen von Benutzer verschiedene Sichten mit Ausschnitt

Konsistenzkontrolle: *Integritätssicherung übernimmt die Gewährleistung der Korrektheit.* Korrekter Datenbankzustand wird durch Constraints (Integritätsbedingungen) definiert

Daten Zugriffskontrolle: Festlegen von Regeln, unautorisierte Zugriffe verweigern (*access control*), z.B. bei datenschutzrechtlichen, personenbezogenen, firmenspezifischen Daten

Transaktionen: Mehrere *Datenbankänderungen in Transaktionen zusammenfassen* und ausführen

Mehrbenutzerfähigkeit: Transaktionen mehrerer Benutzer synchronisiert, um *Concurrency Control zu vermeiden*. Benutzer erscheinen nur isoliert

Datensicherung: Hard oder Softwarefehler wieder korrekt herstellen (*recovery*)

Vorteile Datenbankeinsatz

- Nutzung von Standards: Erleichtert Einführung und Umsetzung von zentralen Standards
- Effizienter Datenzugriff: Vielzahl komplizierter Techniken um *effizient speichern und wieder auslesen von grossen Datenmengen*
- Kürzere Softwareentwicklungszeiten: *schnellere Anwendungsmöglichkeit*
- Hohe Flexibilität: Struktur der Datenbank kann bei Änderungen Anforderungen ohne grosse Konsequenzen für bestehende Daten modifiziert werden
- Hohe Verfügbarkeit: Stellt Datenbank allen Benutzern dank Synchronisierung Eigenschaft gleichzeitig dar
- Grosse Wirtschaftlichkeit: *erlaubt Investition in leistungsstärkere Hardware*, reduziert Betriebs- und Verwaltungskosten

Nachteile DBMS

- Hohe *Anfangsinvestitionen*
- DBMS kann nur für Teil der Anwendungsprogramme optimiert werden
- *Mehrkosten für Bereitstellung* von Datensicherheit, Synchronisation etc.
- *Hochqualifiziertes Personal* erforderlich

Character Set

ASCII, ANSI, Unicode, UTF-8, UTF-16, UTF-32

DDL- Datenbank erstellen

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_option] ...

create_option: [DEFAULT] {
    CHARACTER SET [=] charset_name
  | COLLATE [=] collation_name
  | ENCRYPTION [=] {'Y' | 'N'}
}
```

Datentypen Tabelle

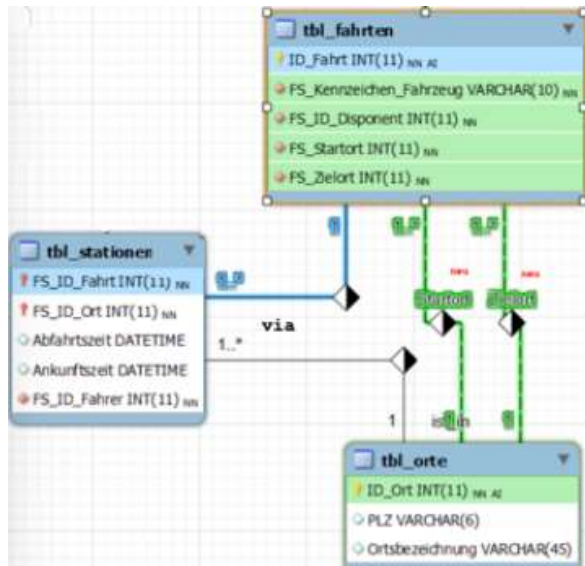
Datentyp	MariaDB (MySQL)	Beispiel	Bemerkung / Einstellungen
Ganze Zahlen	INT	1234	auch TINYINT etc.
Natürliche Zahlen	INT UNSIGNED	1234	unsigned nicht negativ
Festkommazahlen (Dezimalzahlen)	Decimal(M,D)	Decimal(6,2) 1234.56	M=Gesamte Anzahl Stellen D=Nachkommastellen
Aufzählungstypen	ENUM	('small', 'medium', 'large')	M=Gesamte Anzahl D=Nachkomma Liste von Werten
Boolean (logische Werte)	BOOLEAN	TRUE FALSE	
Zeichen (einzelnes Zeichen)	CHAR	'A'	Zeichenkette
Gleitkommazahlen	FLOAT, DOUBLE	7.4	Genauigkeit, Anzahl Stellen
Zeichenkette fester Länge	CHAR(N)	10	Zeichenkette
Zeichenkette variabler Länge	VARCHAR	255	Zeichenkette (Variabel)
Datum und/oder Zeit	DATE, TIME, YEAR	2024-06-03	Datum etc in Formaten
Zeitstempel	TIMESTAMP	2024-06-03 14:30:00	Zeitstempel, automatisch
Binäre Datenobjekte variabler Länge (z.B. Bild)	BLOB, LONGBLOB	BLOB	binäre Daten
Verbund	no support	—	keine Verbund-datentypen
JSON	JSON	{"key": "value"}	JSON-Daten

← DATENSATZ IST VERBUND

Mehrfachbeziehungen

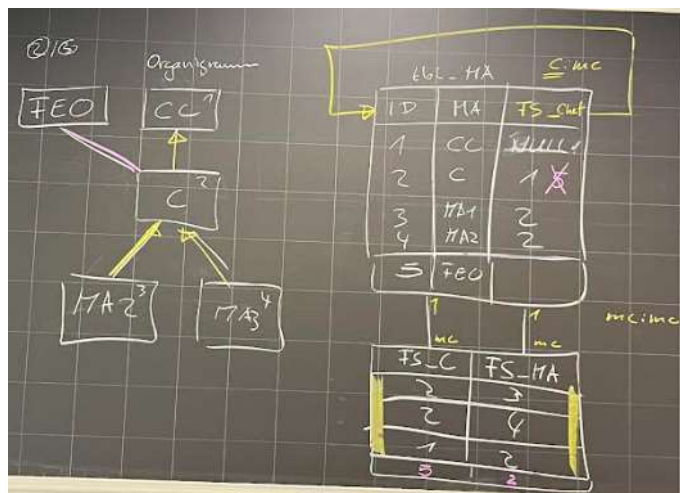
Zwei Tabellen haben mehrere Beziehungen, die unabhängig voneinander sind und anderen Sachverhalt repräsentieren. Für die Kennzeichnung aussagekräftige Beschriftung.

Beispiel drei unabhängige Beziehungen



Rekursion

Assoziation ist nur ein einziger Entitätstyp beteiligt, der Datensatz einer Tabelle steht mit einem anderen Datensatz aus der gleichen Tabelle in Beziehung. Beispielsweise eine Firmenorganisation, bei der jede Person genau eine andere Person als Vorgesetzte hat. Die höchste Person hat aber keinen Vorgesetzten mehr, also c:mc Beziehung. Würde man Fremdschlüsselattribute auf Not Null belassen, könnte die höchste Person nicht erfasst werden.



Einfache Hierarchie

mc:mc Beziehung und Transformationstabelle benötigt, beide Fremdschlüssel der Transformationstabelle zeigen auf Identifikationsschlüssel der gleichen Tabelle aber in unterschiedlichen Rollen



Stücklistenproblem

Tabelle mit Produkten und Eigenschaften, Produkt kann sich aus mehreren anderen Produkte zusammensetzen

Lösung mit Rekursion, neben der produkttabelle weitere Tabelle die Zusammensetzungen enthält

Beziehung mit Einschränkung (constraint) erstellen

Im logischen und physischen Datenmodell Beziehungen der relationalen Datenbank aufgrund constraints der Fremdschlüssel eingeschränkt

Beziehung <i>MasterTab.links : DetailTab.rechts</i>	möglich	nicht möglich ¹ → wird zu	Constraints FK
eins zu eins	1:c c:c	1:1 → 1:c -	NN UQ -- UQ
eins zu viele	1:mc c:mc	1:m → 1:mc c:m → c:mc	NN -- -- --
viele zu viele nur via Transformationstabelle		m:m, m:mc, mc:m, mc:mc → 1:mc-[TT]-mc:1	- NN -- & NN --

Beim Fremdschlüssel Constraint mit Not Null erstellen

```
CREATE TABLE child_table (  
  id INT PRIMARY KEY,  
  parent_id INT NOT NULL,  
  FOREIGN KEY (parent_id) REFERENCES parent_table(id)  
);
```

Weshalb wird für jeden Fremdschlüssel ein Index erstellt? Lesen Sie hier

Damit die Performance der Datenbank besser ist. Indexe ermöglichen schnellere Suchvorgänge und Join Operationen. Ohne Index müsste Datenbank jedes Mal einen Scan durch ganze Tabelle durchführen.

Wie wird der Constraint UNIQUE für einen Fremdschlüssel im Workbench mit Forward Engineering erstellt?

Indem man die Spalte auswählt und als einzigartig markiert

```
CREATE TABLE example_table (  
  id INT PRIMARY KEY,  
  unique_key_column INT,  
  FOREIGN KEY (unique_key_column) REFERENCES another_table(id),  
  UNIQUE (unique_key_column)
```

);

Jede Beziehung wird auch mit einer Beziehungs-Überprüfung (Constraint ...) versehen. Erstellen Sie eine allgemeine Syntax für die CONSTRAINT-Anweisung.

```
CREATE TABLE table_name (  
    column1 datatype [constraint],  
    column2 datatype [constraint],  
    ...  
    CONSTRAINT constraint_name  
    FOREIGN KEY (column_name) REFERENCES parent_table(parent_column)  
    [ON DELETE action]  
    [ON UPDATE action]  
);
```

Anstelle UNIQUE-Index kann nur ein Fremdschlüssel auf UNIQUE gesetzt werden: FK_Fahrer INT UNIQUE;.

Ergänzung ALTER TABLE tbl ADD CONSTRAINT

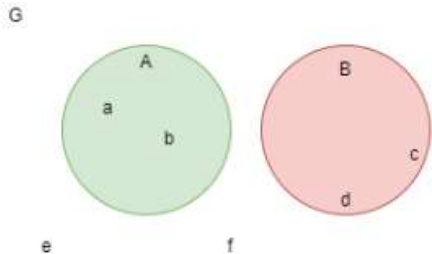
Fremdschlüssel hinzufügen:

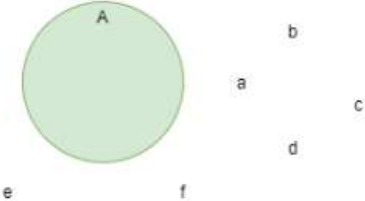
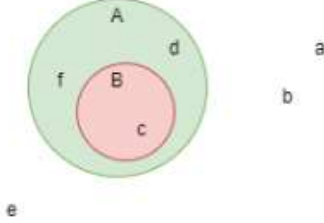
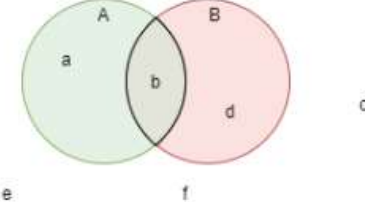
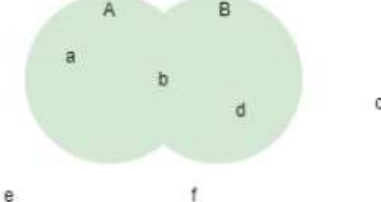
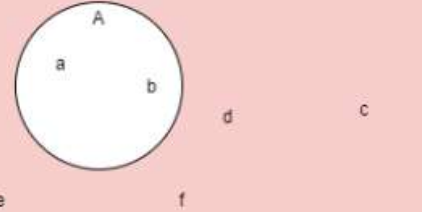
```
ALTER TABLE <DetailTab>  
    ADD CONSTRAINT <Constraint> FOREIGN KEY (<Fremdschlüssel>)  
    REFERENCES <MasterTab> (Primärschlüssel);
```

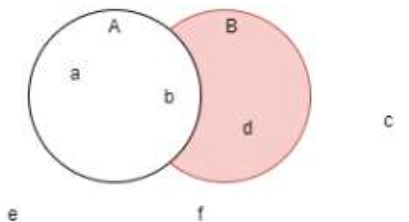
Eindeutiger, einmaliger Schlüssel:

```
ALTER TABLE <Tabelle>  
    ADD UNIQUE (<FS_Name>);
```

Mengenlehre

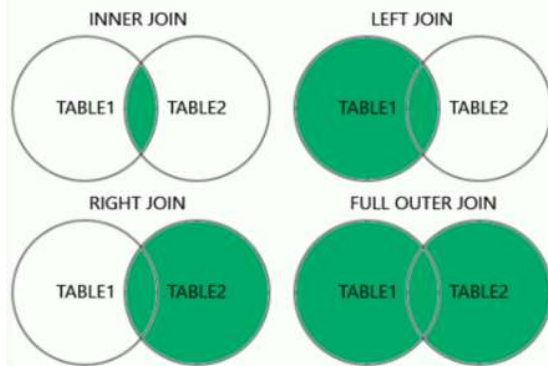
Gross- und Klein-Buchstaben und \in, \notin	Grossbuchstaben = Menge Kleinbuchstaben = Elemente welcher Menge zugewiesen sind	$G=\{a,b,c,d,e,f\}$, $A=\{a,b\}$, $B=\{c,d\}$	
--	---	---	---

$\{\}$ oder \emptyset	leere Menge	$A=\{\}$	
\subset, \subseteq	Teilmenge von	$B \subset A, x \in B$ auch $x \in A$	
\cap	Schnittmenge	$A \cap B, x \in B$ und $x \in A$	
\cup	Vereinigungsmenge	$A \cup B, x \in A$ oder $x \in B$	
X^c	Komplementärmenge	<i>alle Elemente, die nicht in der Menge X enthalten sind, $x \in G$ und $x \notin A$</i>	

\	Differenzmenge	$B \setminus A = x \in B \text{ und } x \notin A$	
---	----------------	---	---

SELECT JOIN

- **(INNER) JOIN** : Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN** : Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN** : Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN** : Returns all records when there is a match in either left or right table



Checkpoints

- *Welche Schwierigkeiten beim Einfügen von Daten ergeben sich, wenn z.B. der FK Vorgesetzter als Constraint definiert ist? (Tipp: ref. Integrität)*
 Beim Einfügen von Daten in eine Tabelle mit dem FK Vorgesetzter als Constraint kann man Schwierigkeiten bekommen, wenn der referenzierte Vorgesetzte noch nicht existiert. Das verletzt die referenzielle Integrität und verhindert das Einfügen. Man muss sicher gehen, dass alle referenzierten Vorgesetzten vorher schon in der Tabelle existieren.
- *Warum ist der Wert NULL in der tbl_Hierarchie nicht zulässig? (Tipp: Kardinalität)*
 Ein NULL-Wert in der tbl_Hierarchie ist nicht erlaubt, da die Kardinalität eine Hierarchie verlangt. Jede Position muss eindeutig einer anderen Position oder dem obersten zugeordnet sein, um die Struktur der Hierarchie zu bewahren. Ein NULL-Wert würde diese Zuordnung verstossen.
- *Wann muss eine Hierarchie-Tabelle anstelle einer rekursiven Beziehung eingesetzt werden? (Tipp: Chefs)*

Eine Hierarchie-Tabelle wird anstelle einer rekursiven Beziehung eingesetzt, wenn die Anzahl der Hierarchieebenen fest ist. Bei grossen Organisationen kann eine feste Struktur übersichtlicher und performanter sein als rekursive Abfragen.

- *Ref. Integrität: Was ist das?* Machen Sie ein Beispiel dazu! Die Definition von referentieller Integrität ist, dass Beziehungen zwischen Tabellen in einer relationalen Datenbank konsistent bleiben. Ein Fremdschlüssel in einer Tabelle verweist immer auf einen gültigen Eintrag in einer anderen Tabelle.
 - *Welche Constraints kann eine Beziehung haben? (Tipp: Mehr als eine!)* Sie kann die Beziehung NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY und CHECK haben
 - *Was ist der Unterschied zwischen LEFT JOIN und RIGHT JOIN?* Left Join gibt alle Datensätze aus der linken Tabelle und die Passenden aus der rechten zurück, wobei right join das gleiche macht, aber aus der rechten Tabelle.
 - *Wie wird eine 1:1-Beziehung und eine c:m-Beziehung umgesetzt? Warum?* Durch einen Fremdschlüssel, der auch als UNIQUE und NOT NULL gilt. Das durch eine Zwischentabelle, welche beide PKs als FKs enthält.
 - *Was ist der Nachteil, wenn eine Beziehung nur mit Primär- und Fremdschlüssel definiert werden, d.h. ohne die Constraint-Anweisung?* Es gibt dann keine weiteren Sicherungen bezüglich der Integrität. Es könnten also doppelte Einträge möglich sein oder Null Werte in Fremdschlüssel spalten.
 - *Welche Folge hat ein Eintrag eines Fremdschlüssels Wertes, der als ID-Wert in der verbundenen Tabelle nicht vorhanden ist?* a) mit Constraint-Anweisung auf dem FS Es gibt einen Fehler aus b) ohne Constraint-Anweisung auf dem FS Er kann eingefügt werden auch wenn der FK nicht existiert
-

Spick/Lernblatt M164 - LB1

DBMS → Database Management System

Software die für die Verwaltung von Datenbanken dient. Es ermöglicht das Löschen, Hinzufügen, Aktualisieren und Lesen von Daten (CRUD). Es ist eine Schnittstelle zwischen user und Datenbank.

Funktionen eines DBMS:

- Datenintegration: Ermöglicht die Kombination von Daten und verschiedenen Quellen.
- Datenkonsistenz: Gewährleistet die Genauigkeit und Konsistenz der Daten.
- Datensicherheit: Schützt die Daten vor unbefugtem Zugriff
- Datenabfrage/manipulation: CRUD-Methoden
- Datensicherung/wiederherstellung: Sichert Daten ermöglicht wiederherstellung

Beispiele für DBMS:

Relationale DBMS: MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server

NoSQL DBMS: MongoDB, Cassandra, Redis

RDBMS → verwendet SQL Daten werden in Tabellen gespeichert

NoSQL-DBMS → Daten werden in schemalosen Formaten gespeichert

DML → Data Manipulation Language

wird verwendet um Daten innerhalb der Datenbank zu manipulieren. Umfasst das Einfügen, Aktualisieren, Löschen und Abrufen von Daten.

Wichtigste DML-Befehle:

Neue Datensätze in Table einfügen:

```
INSERT INTO students (id, name, age) VALUES (1, 'Papi Chulo', 20);
```

Aktualisieren von Daten

```
UPDATE students SET age = 21 WHERE id = 1;
```

Daten löschen:

```
DELETE FROM students WHERE id = 1;
```

DDL → Data Definition Language

Wird verwendet um die Struktur der Datenbank zu definieren.

Mit DDL können Datenbank Objekte wie Tabellen erstellt, bearbeitet und gelöscht werden.

Wichtigste DDL Befehle:

Table erstellen:

```
CREATE TABLE students (  
  id INT PRIMARY KEY,  
  name VARCHAR (100),  
  age INT  
);
```

Table ändern:

```
ALTER TABLE students ADD (COLUMN grade VARCHAR (2));
```

Table löschen:

```
DROP TABLE students;
```

SELECT * FROM students;

Ruft Daten von Table auf;

Wichtigster DAL-Befehl:

DAL → Data Query Language
wird verwendet um Daten abzurufen

Modul 164

Ihr versteht was ein DBMS ausmacht.

Ein DBMS ist eine Software zur Verwaltung von Datenbanken. Es bietet Funktionen zum Erstellen, Abfragen, Aktualisieren und Löschen von Datenbanken und deren Objekten.

Ihr könnt die Schritte erklären wie ihr Forward-Engeneering vom modell alle "create" statements erzeugen könnt.

Beim Forward Engineering geht es darum, aus einem Modell (z.B. einem ER-Diagramm oder einem objektorientierten Modell) die entsprechenden SQL "create" Statements zu generieren, um die Datenbank und ihre Tabellen zu erstellen.

Ihr wisst wie ihr mit Synchronize Model "alter" statements erzeugen könnt.

Mit "Synchronize Model" werden "alter" Statements generiert, um Änderungen an einer bestehenden Datenbankstruktur vorzunehmen, basierend auf einem aktualisierten Modell.

Ihr wisst was *alter, drop,create* Statements tun.

- Alter: Ändert die Struktur einer bestehenden Tabelle
- Drop: Löscht eine Tabelle oder ein anderes Datenbankobjekt
- Create: Erstellt neue Tabellen oder andere Datenbankobjekte

Ihr könnt eine "einfache" Tabelle mit einen paar spalten mit *create* Statement erstellen.

```
CREATE TABLE Beispiel (  
  id INT PRIMARY KEY,  
  name VARCHAR(50),  
  geburtsdatum DATE  
);
```

Ihr könnt eine Tabelle mit einem *drop* Statement löschen.

DROP Table Beispiel;

Ihr könnt mit *alter* eine Spalte zu einer bestehenden Tabelle hinzufügen.

```
ALTER TABLE Beispiel  
ADD email VARCHAR(100);
```

Ihr könnt mit *insert* in lang und Kurzform neue Zeilen in einer Tabelle hinzufügen.

Langform:

```
INSERT INTO Beispiel (id, name, geburtsdatum)  
VALUES (1, 'Max Mustermann', '1990-01-01');
```

Kurzform (wenn alle Spalten gefüllt werden):

```
INSERT INTO Beispiel  
VALUES (2, 'Erika Musterfrau', '1995-05-15');
```

Ihr könnt mit *update* einen neuen wert in genau einem einzelnen Feld einer Tabelle ersetzen.

```
UPDATE Beispiel  
SET name = 'Max Mustermann Jr.'  
WHERE id = 1;
```

Ihr könnt eine oder mehrere Zeilen in einer Tabelle mit einem *delete* Statement löschen

```
DELETE FROM Beispiel  
WHERE id = 2;
```

Like Statement

Like wird verwendet, um nach einem Muster in einer Spalte zu suchen.

```
SELECT *  
FROM Beispiel  
WHERE Vorname LIKE 'Ma%';
```

Dieses Beispiel wählt alle Zeilen aus der Tabelle Beispiel aus, in denen der Vorname mit 'Ma' beginnt, gefolgt von beliebigen Zeichen (% ist ein Platzhalter für eine beliebige Anzahl von Zeichen).

OR Statement

OR wird verwendet, wenn man eine Abfrage macht welche wahr ist, sie jedoch nur dann erfüllt wird wenn eine der Bedingungen stimmt.

```
SELECT *  
FROM Mitarbeiter  
WHERE Vorname = 'Max' OR Nachname = 'Mustermann';
```

AND Statement

AND wird verwendet, wenn man Abfragen macht welche auf mehrere Bedingungen zutreffen müssen.

```
SELECT *  
FROM Mitarbeiter  
WHERE Vorname = 'Max' AND Abteilung = 'Vertrieb';
```

Alle Statements zusammen.

Erstellen der Tabelle Bestellungen

```
CREATE TABLE Bestellungen (  
    BestellID INT PRIMARY KEY,  
    KundenID INT,  
    BestellDatum DATE,  
    Betrag DECIMAL(10, 2)  
);
```

Hinzufügen einer neuen Spalte BestellID zur Tabelle Bestellungen

```
ALTER TABLE Bestellungen  
ADD BestellID INT;
```

Einfügen von Daten in die Tabelle Bestellungen

```
INSERT INTO Bestellungen (BestellID, KundenID, BestellDatum, Betrag)  
VALUES (1, 101, '2024-06-01', 49.99),  
      (2, 102, '2024-06-02', 29.99),  
      (3, 103, '2024-06-03', 99.99);
```

Löschen der Bestellung mit BestellID = 3
`DELETE FROM Bestellungen
WHERE BestellID = 3;`

Aktualisieren der Kundenalter

```
UPDATE Kunden  
SET Alter = 35  
WHERE Vorname = 'Max' AND Nachname = 'Mustermann';
```

Abrufen von Kundeninformationen, die den Namen 'Max' enthalten
`SELECT *
FROM Kunden
WHERE Vorname LIKE '%Max%' OR Nachname LIKE '%Max%';`

Abrufen von Kunden, die älter als 30 Jahre sind und männlich
`SELECT *
FROM Kunden
WHERE Alter > 30 AND Geschlecht = 'm';`