SQL

Structured Query Language

Teil1:

Data **D**efinition **L**anguage

Die Datenbankabfragesprache SQL

1. Geschichte

Die **Datenbankabfragesprache SQL** wurde aus der Abfragesprache **SEQUEL** (**S**tructured **E**nglish **QUE**ry **L**anguage) der Firma IBM entwickelt. Mit der Entwicklung der Sprache wurde das Ziel verfolgt, auch für Nicht-Programmierer eine „relativ“ einfache Sprache zur Abfrage von Datenbanken zur Verfügung zu stellen, die keine mathematischen Notationen verwendet.

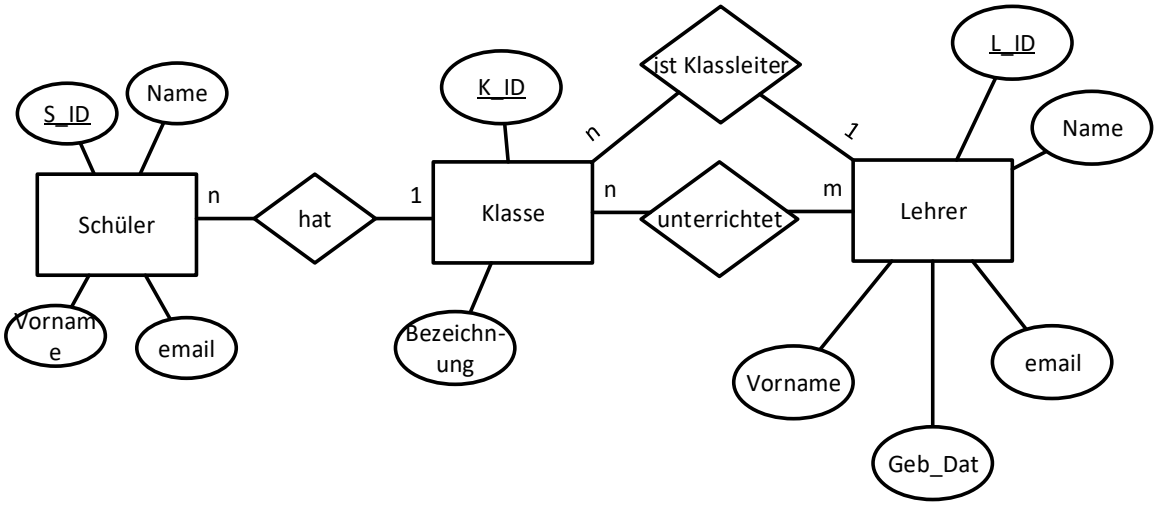
SQL (Structured Query Language) ist eine interaktive Sprache und erlaubt Ad-hoc-Abfragen von Datenbanken (Abfragen, die sofort ausgeführt werden). Über sogenannte SQL-Skripts können Sie mehrere SQL-Anweisungen automatisiert ausführen lassen. Außer zur interaktiven Eingabe wird SQL auch in Anwendungen zur Kommunikation mit einer Datenbank eingesetzt. Dabei können lesende wie auch schreibende Zugriffe durchgeführt werden.

Zum Sprachumfang von SQL gehören vier Befehlsgruppen:

DDL (Data Definition Language)	Erstellen von Datenbanken, Tabellen (Relationen) und Indizes
DQL (Data Query Language)	Abfragen von Daten
DML (Data Manipulation Language)	Anlegen, Ändern und Löschen von Datensätzen
DCL (Data Control Language)	Anlegen von Benutzern und Vergabe von Zugriffsrechten

2. Einführung

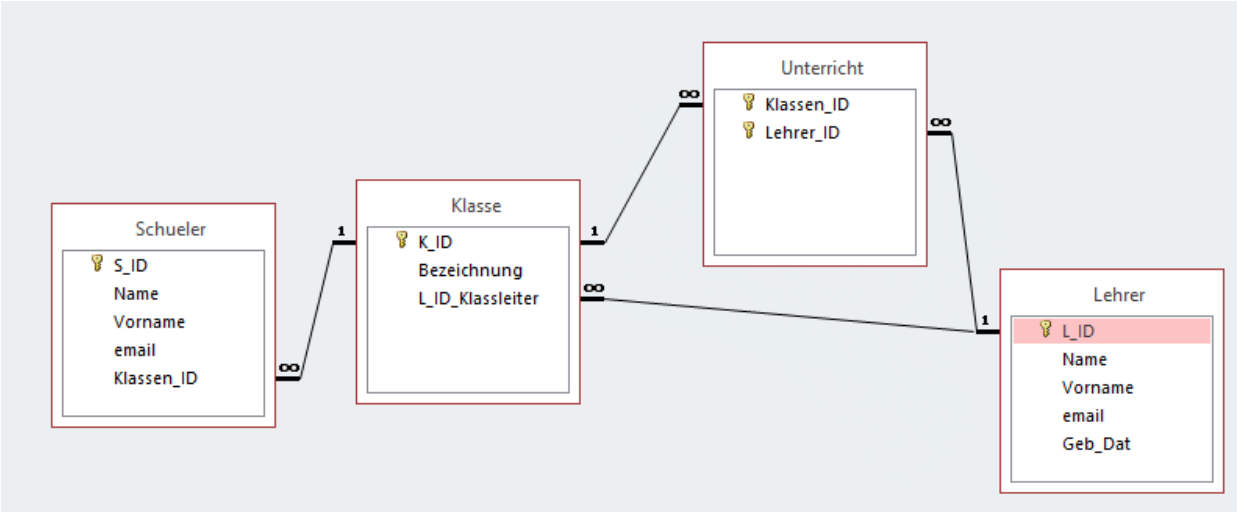
Es wurde folgendes ER-Modell einer einfachen Schulverwaltung entworfen:



Daraus wird das relationale Datenbankmodell (Datenbank-Schema) erstellt. Zwischen den Beziehungen der einzelnen Tabellen herrscht **referentielle Integrität**.

Die **Beispiele** in den **darauffolgenden Kapiteln** beziehen sich immer auf nachfolgendes relationale Datenmodell.

Relationales Datenmodell



3. Data Definition Language (DDL)

Die **Datendefinitionssprache (DDL)** von SQL stellt Sprachmittel zur Definition, zur Änderung und zum Löschen von

- Relationen-Schemata,
- Zugriffsrechten,
- Views,
- Indexen (Zugriffspfaden) und
- Integritätsbedingungen

zur Verfügung. Die DDL kommt fast überwiegend bei einer neuen Datenbank am Anfang zum Einsatz.

Beim Aufbau einer Datenbank wird prinzipiell wie folgt vorgegangen:

A. Zuerst wird die Datenbank angelegt.

   Sie dient als Container für die Tabellen und alle weiteren Datenbankobjekte.

B. Danach werden die dazugehörigen Tabellen erstellt.

   Dabei muss zunächst nur die Struktur der Tabellen festgelegt werden. D. h. welche Spalten die Tabelle hat bzw. welche Datentypen darin gespeichert werden und welcher Primärschlüssel verwendet wird.

C. Anschließend werden die Tabellen mit Daten gefüllt.

   Tabellen sind die einzigen Datenbankobjekte, in denen die Daten physisch gespeichert werden. Hier können Auswertungen der Daten vorgenommen oder weitere Aktionen ausgeführt (z. B. Daten ändern oder löschen) werden.

2.1 Datenbank erstellen

Das Erstellen einer Datenbank bedeutet das Erzeugen eines neuen Verzeichnisses im Dateisystem mit dem entsprechenden Datenbanknamen.

Syntax

```
CREATE DATABASE Datenbank_Name;
```

```
CREATE DATABASE [IF NOT EXISTS] Datenbank_Name;
```

Beispiel: Schulverwaltung

```
create database Schulverwaltung;
```

2.2 Datenbank löschen

Über die Anweisung DROP DATABASE wird eine Datenbank mit allen enthaltenen Daten gelöscht. Dabei wird keine Warnmeldung ausgegeben, der Vorgang kann nicht rückgängig gemacht werden. Mit diesem Befehl werden die Katalogeinträge der Datenbank sowie das Verzeichnis inklusive der Daten gelöscht.

Syntax

```
DROP DATABASE Datenbank_Name;
```

```
DROP DATABASE [IF EXISTS] Datenbank_Name;
```

Beispiel: Schulverwaltung

```
drop database Schulverwaltung;
```

2.3 Datenbank auswählen

Bevor mit einer bestimmten Datenbank und den darin befindlichen Datenbankobjekten gearbeitet werden kann, muss diese ausgewählt werden.

Syntax

```
USE Datenbank_Name;
```

Beispiel: Schulverwaltung

```
use Schulverwaltung;
```

2.4 Datenbanken anzeigen

Wenn der Name einer bestimmten Datenbank nicht bekannt ist, kann eine Liste der vorhandenen Datenbanken angezeigt werden.

Syntax

```
SHOW DATABASES;
```

2.5 Datentypen

Es gibt eine Vielzahl von Datentypen, die bei der Erstellung eines Relationen-Schemata verwendet werden können. Im Folgenden ist eine kleine Auswahl an wichtigen Datentypen dargestellt, die für die Bearbeitung der Übungsaufgaben ausreichen.

CHARACTER(n), **char(n)**	Zeichenreihe einer festen Länge von n Zeichen
CHARACTER VARYING(n), **VARCHAR(n)**	Zeichenreihe variabler Länge mit bis zu n Zeichen
INTEGER, **int**, SMALLINT	Ganze Zahl (mit oder ohne Vorzeichen)
NUMERIC(p,q), **DECIMAL(p,q**), DEC(p,q)	Dezimalzahl mit p Ziffern insgesamt und q Ziffern nach dem Dezimalpunkt
float(n), float, REAL	Gleitkommazahl mit Stelligkeit n oder systemabhängiger Genauigkeit
DATE, TIME, **DATETIME**, TIMESTAMP	Datum und/oder Uhrzeit

Eine Übersicht über alle Datentypen sind unter folgenden Links zu finden:

https://docs.microsoft.com/de-de/biztalk/adapters-and-accelerators/adapter-sql/basic-sql-server-data-types

https://docs.microsoft.com/de-de/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver15

2.6 Tabelle erstellen

Relationenschemata gehören zur logischen Schicht eines Datenbanksystems.
Die Definition eines Relationenschemas erfolgt in SQL durch das CREATE TABLE-Statement.

Hinweis:

Anweisungen, die in den eckigen Klammern [ ] stehen, sind **optionale** Angaben!

Syntax

```
CREATE TABLE tabellenname
     (
          <Attribut1> [ NOT NULL ],
          ...
          <AttributN> DatentypN [ NOT NULL ],

          [PRIMARY KEY (Attr1, Attr2,…) ],
          [FOREIGN KEY (Attr_1) REFERENCES Tabelle(Attr1) ],
          ...
          [FOREIGN KEY (Attr_M) REFERENCES Tabelle(Attr1) ],
     ) ;
```

2.7 Tabellenstruktur ändern

Die Struktur einer Tabelle kann jederzeit über die Anweisung ALTER TABLE geändert werden. Man kann …

- Datenfelder hinzufügen oder löschen,
- Datenfelddefinitionen verändern,
- Gültigkeitsprüfungen hinzufügen oder löschen,
- Schlüssel und Indizes hinzufügen oder löschen.

Die ALTER TABLE-Anweisung wird verwendet, um Spalten in einer vorhandenen Tabelle hinzuzufügen, zu löschen oder zu ändern. Sie wird aber auch verwendet, um einer vorhandenen Tabelle verschiedene Einschränkungen hinzuzufügen und zu löschen.

Detaillierte Erklärungen zum ALTER-Befehl finden Sie dabei unter folgendem Link:

https://www.w3schools.com/sql/sql_alter.asp

Syntax

```
ALTER TABLE Tabellen_Name
<ALTER-Spezifikationen>;
```

2.8 Umsetzungsbeispiel der Schulverwaltung

In diesem Kapitel werden die **Tabellen** der **Datenbank „Schulverwaltung“** mithilfe der oben genannten SQL-Anweisungen erstellt. Die **Erstellung** der Datenbank ist in Kapitel 2.1 zu finden. Es ist zu **beachten**, dass zuerst die Tabellen **ohne Fremdschlüssel** erstellt werden müssen, da referentielle Integrität verwendet wird.

Erstellen der Tabelle *Lehrer*

```
create table Lehrer
(
	L_ID char(4) primary key,
	Nachname varchar(50) not null,
	Vorname varchar(50) not null,
	email varchar(50),
	Geb_Dat date
);
```

Alternativ:

```
create table Lehrer
(
	L_ID char(4),
	Nachname varchar(50) not null,
	Vorname varchar(50) not null,
	email varchar(50),
	Geb_Dat date,
	primary key(L_ID)
);
```

Mit **NOT NULL** wird die Eingabe eines Datenwertes für das entsprechende Datenfeld erzwungen. Mit **PRIMARY KEY** wird der Primärschlüssel der Tabelle festgelegt.

Diese weiteren **optionalen Parameter** sind bei der Definition von Datenfeldern möglich:

- **NULL**

  Mit diesem Parameter wird festgelegt, dass das Datenfeld standardmäßig keinen Wert (auch nicht 0 oder eine leere Zeichenkette) enthält.
- **NOT NULL** (siehe oben)
- **DEFAULT standardwert**

  Der Parameter DEFAULT definiert einen Standardwert für das Datenfeld. Erhält dieses Datenfeld bei der Eingabe der Daten keinen Wert, wird der Standardwert verwendet.
- **UNIQUE**

  Der Parameter UNIQUE legt die Einmaligkeit des Wertes fest.
- **AUTO_INCREMENT bzw. IDENTITY(startwert, inkrement)**

  Der Wert dieses Datenfeldes wird automatisch beim Anlegen eines neuen Datensatzes aus dem Wert des Datenfeldes des vorherigen Datensatzes plus eins errechnet. Dieser Wert kann vom Benutzer nicht geändert werden. Auto_Increment wird beim SQL-Server nicht akzeptiert, daher muss IDENTITY(a,b) verwendet werden.

Erstellen der Tabelle *Klasse*

```
create table Klasse
(
      K_ID char(6) primary key,
      Bezeichnung varchar(20),
      L_ID_Klassleiter char(4),
      foreign key (L_ID_Klassleiter) references Lehrer
);
```

Alternativ:

```
create table Klasse
(
      K_ID char(6) primary key,
      Bezeichnung varchar(20),
      L_ID_Klassleiter char(4),
      foreign key (L_ID_Klassleiter) references Lehrer(L_Id)
);
```

Das Attribut **L_ID_Klassleiter** ist der **Fremdschlüssel**, der sich auf den **Primärschlüssel** der Tabelle Lehrer bezieht. Da jede Tabelle nur einen Primärschlüssel besitzt ist die Angabe des Primärschlüssels „**references Lehrer(L_Id)**“ optional. Daher reicht „**references Lehrer**“.

Erstellen der Tabelle *Unterricht*

Beachte:

Die Tabelle Unterricht besitzt **zwei Fremdschlüssel** und einen **zusammengesetzten Primärschlüssel**.

```
create table Unterricht
(
      Klassen_ID char(6),
      Lehrer_ID char(4),
      foreign key (Klassen_ID) references Klasse,
      foreign key (Lehrer_ID) references Lehrer,
      primary key (Klassen_ID,Lehrer_ID)
      -- zusammengesetzter Primärschlüssel
);
```

Alternativ:

```
create table Unterricht
(
	Klassen_ID char(6),
	Lehrer_ID char(4),
	foreign key (Klassen_ID) references Klasse (K_ID),
	foreign key (Lehrer_ID) references Lehrer (L_ID),
	primary key (Klassen_ID, Lehrer_ID)
	-- zusammengesetzter Primärschlüssel
);
```

Erstellen der Tabelle *Schueler*

```
create table Schueler
(
	S_ID char(8) primary key,
	Nachname varchar(50),
	Vorname varchar(50),
	email varchar(50),
	Klassen_ID char(6),
	foreign key(Klassen_ID) references Klasse
);
```

Hinzufügen des Attributes *Fach* in der Tabelle *Lehrer*

In der Tabelle Lehrer wird nun ein Attribut Namens „Fach“ hinzugefügt.

```
alter table Lehrer
add Fach varchar(30);
```

Da dieses Attribut jedoch keinen Sinn bei der Tabelle Lehrer macht, wird es wieder gelöscht.

```
alter table Lehrer
drop column  Fach;
```

Aufgabenstellung 1:

- **Erstellen** Sie die **Relationale Datenbank** für den Sportverein in 3. Normalform!
  - **Überführen** Sie zunächst das ER-Modell in ein Relationales Datenmodell!
- **Verwenden** Sie dazu den Microsoft SQL-Server!
- **Bennen** Sie Ihre Datenbank immer nach folgendem Muster:
  - DATENBANKNAME_IT-KENNUNG
  - **Beispiel:** Sportverein_19ITZ001

Anhang: Email des Auftraggebers

Hallo IT'ler,

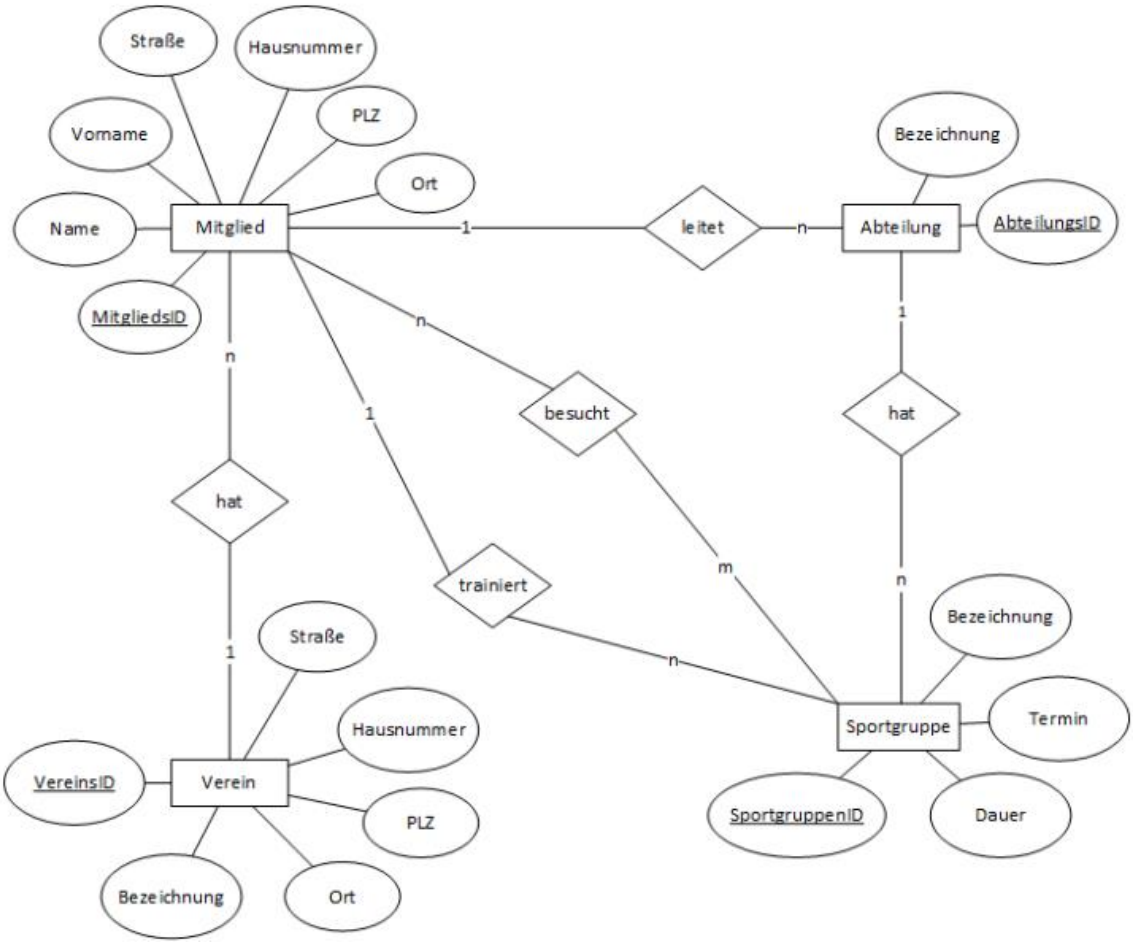
der Vorstand vom SSV Jahn Passau möchte zukünftig die Mitgliederverwaltung in einer Relationalen Datenbank durchführen.

Bitte **entwickeln** Sie mir eine **relationale Datenbank** zur **Mitgliederverwaltung**, welches die geforderten Kundenwünsche umsetzt!

Das ER-Modell ist in der angefügten Word-Datei zu finden.

Beste Grüße
A. Huber

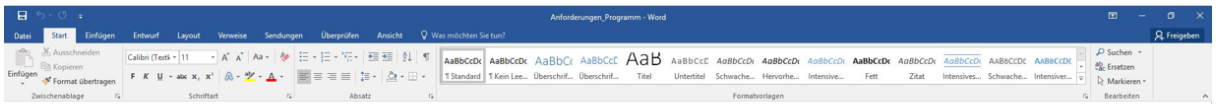
Anhang: ER-Modell des Sportvereins



Eine Abteilung besitzt im Modell eine Bezeichnung sowie eine AbteilungsID als Primärschlüssel. Dabei leitet ein Mitglied des Sportvereins 0 bis viele Abteilungen. Eine Abteilung hat 0 bis viele Sportgruppen, jede Sportgruppe ist genau einer Abteilung zugeordnet.

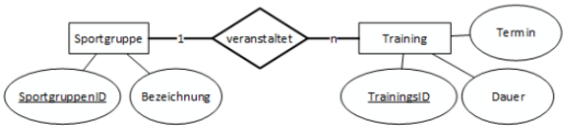
Aufgabenstellung 2:

- **Setzen** Sie die folgenden Änderungswünsche des Kunden um!
- **Erweitern** Sie dazu Ihr SQL-Skript bzw. Ihren SQL-Dump!



Änderungswünsche des Kunden:

- Geburtsdatum (gebdat) bei den Mitgliedern hinzufügen (Datentyp: *date*)
- 3. Normalform muss erreicht werden
  - Tabelle *Ort* erstellen
- Zur Erfassung einzelner Trainingseinheiten einer Sportgruppe ist die Einführung einer Tabelle *Training* notwendig



Beispiel für Termin: Montag

Beispiel für Dauer: 90min

- Mitgliedsbeiträge müssen noch abgebildet werden
  - Tabelle *Beitrag* erstellen
  - Felder: b ID, bezeichnung, beitrag (Datentyp: *decimal(n,p)*)