Modul 105: Datenbanken mit SQL bearbeiten

SQL Crashkurs III: INSERT, UPDATE und DELETE

Ausgangslage

Wir haben im letzten Arbeitsauftrag mit der Schülerdatenbank gearbeitet. Die Daten waren bereits vorhanden, wir bearbeiteten noch keine Daten, sondern führten einfache bis komplexe Abfragen über eine oder mehrere Tabellen durch. In diesem Arbeitsblatt geht es darum, die Daten zu bearbeiten: Neue Datensätze einfügen, vorhandene Datensätze ändern und Datensätze löschen.

Ziele

- INSERT-Befehl in Theorie und Praxis
- UPDATE-Befehl in Theorie und Praxis
- DELETE-Befehl in Theorie und Praxis
- Aufgaben

Vorgehen

Die bereits bekannte Übungsdatenbank *schuelerV2.sqlite* wird auch für dieses Arbeitsblatt benötigt. Starten Sie *Sqliteman* und öffnen Sie die Datenbank, um mit der Arbeit loslegen zu können. Führen Sie die vorgestellten Beispiele gleich aus, um den jeweiligen Befehl zu testen und das Ergebnis zu prüfen.

Gehen Sie dieses Arbeitsblatt Schritt für Schritt durch. Konsultieren Sie die Unterlagen, suchen Sie zusätzliche Informationen im Internet oder fragen Sie Ihre Lehrperson.

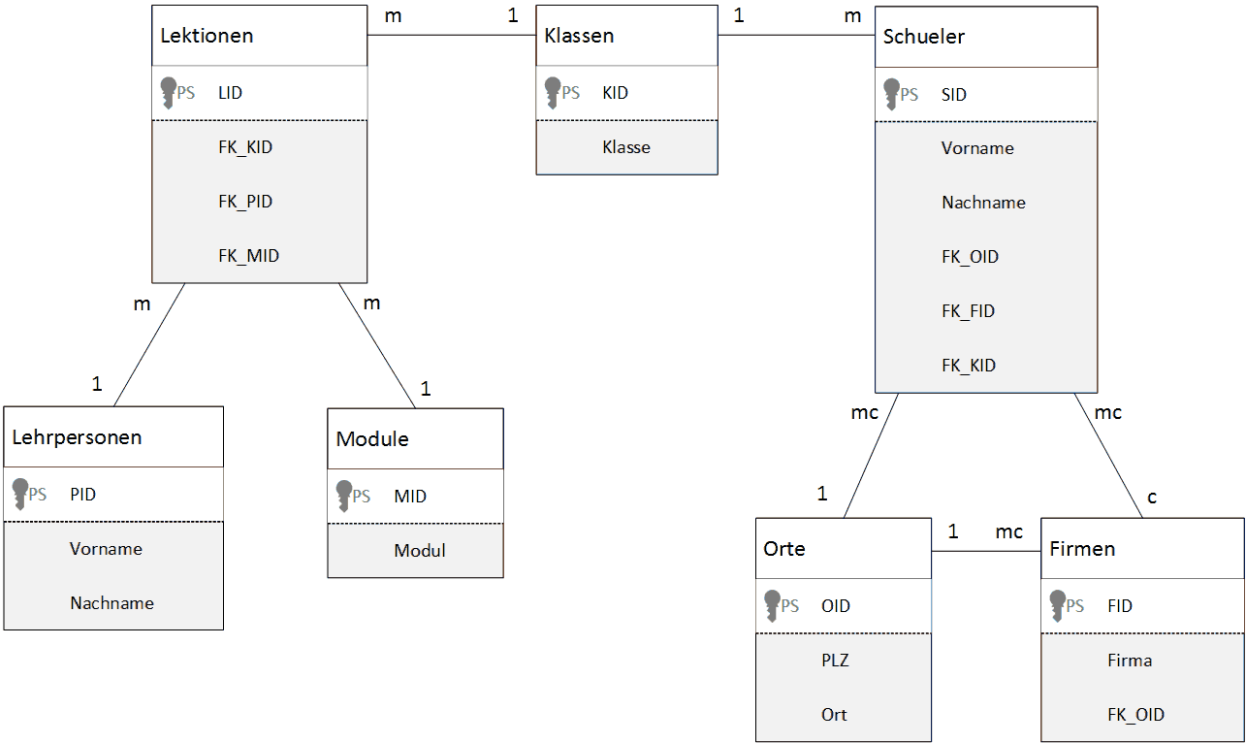
Unterlagen / Quellen

- SQL Referenz auf Moodle: SQL-Referenz.pdf
- Allgemeine SQL-Tutorials:
  - https://de.wikibooks.org/wiki/Einf%C3%BChrung_in_SQL
  - https://www.w3schools.com/sql
- SQLite-spezifisches Tutorial:
  - https://www.tutorialspoint.com/sqlite/

Zeit: 4 Lektionen

Schüler-Datenbank

Erinnern wir uns an das logische Datenmodell: Nehmen Sie dieses jeweils zur Hand, um Befehle über mehrere Tabellen abzusetzen.



Warum brauchen wir die Befehle INSERT, UPDATE und DELETE?

In der Schülerdatenbank waren bzw. sind die Daten bereits vorhanden, und wir konnten in den vergangenen Arbeitsblättern SELECT-Befehle über eine und mehrere Tabellen (JOIN) absetzen. Wie aber kamen die Daten in die Datenbank? Die Daten wurden in diesem Fall einmalig importiert. Es kann also sein, dass wir bereits mit Daten starten, wenn wir mit einer Datenbank arbeiten. Es kann aber auch sein, dass noch keine Daten vorhanden sind.

So oder so werden die Daten in den allermeisten Fällen im Betrieb gepflegt. Dazu verwenden wir die Befehle INSERT (Datensätze einfügen), UPDATE (Datensätze ändern) und DELETE (Datensätze löschen).

INSERT

Einfügen eines Datensatzes

```
INSERT INTO <Tabelle>
   [ (<Spaltenliste>) ]
   VALUES (<Werteliste>);
```

Spaltenliste

Die Attribute (Spalten), in die Werte eingefügt werden sollen.

Werteliste

Die Werte, die in die entsprechenden Spalten eingefügt werden sollen.

Die Anzahl der Werte muss identisch sein mit den Spalten, die Reihenfolge muss identisch sein und der Datentyp der Werte muss demjenigen der Spalten entsprechen.

INSERT mit Angabe der Spalten

```
INSERT INTO Orte
   (PLZ, Ort)
    VALUES (3140, "Gasel");
```

Es wird der Ort *Gasel* in der Tabelle *Orte* eingefügt. In der Spalte *PLZ* wir der Wert *3140* (Integer) und in der Spalte *Ort* der Wert *"Gasel"* (String) eingetragen.

Aufgabe 1

Fügen Sie in der Tabelle *Lehrpersonen* die Lehrkraft mit Namen *Reto Glarner* ein und notieren Sie den entsprechenden Befehl.

```
insert into lehrpersonen Values (31, "Reto", "Glarner");
```

INSERT ohne Angabe der Spalten

```
INSERT INTO Orte
   VALUES (190, 3097, "Liebefeld");
```

Es wird der Ort *Liebefeld* in der Tabelle *Orte* eingefügt. Zusätzlich zu Postleitzahl und Ort wird an erster Stelle *190* eingetragen, das ist der Wert für den Primärschlüssel *OID*.

Wenn die Spaltenliste weglassen wird, müssen für sämtliche Spalten Werte (VALUES) vorhanden sein, in der richtigen Reihenfolge und mit dem richtigen Datentyp.

Wann ist diese Variante zu empfehlen?

Diese Variante ist nur zu empfehlen, wenn der Primärschlüssel nicht automatisch eingefügt wird (AUTOINCREMENT), was hier nicht der Fall ist. Das vorherige Beispiel ist also korrekt, wo der Primärschlüssel nicht explizit eingefügt, sondern vom System eingefügt wird. Mit dem Wert *190* haben wir nämlich bereits ein "Loch" in der Tabelle generiert, Gasel hat den Wert *188* und der Wert *189* fehlt.

Die Variante ohne Spaltenliste erspart zwar Schreibarbeit, ist aber fehleranfälliger und deshalb nur bedingt zu empfehlen!

Der Vollständigkeit halber sei noch der korrekte INSERT-Befehl aufgeführt, wenn der Primärschlüsselwert nicht automatisch eingefügt wird:

```
INSERT INTO Orte
   VALUES ((SELECT MAX(OID)+1 FROM Orte), 3097, "Liebefeld");
```

Es wird der maximale Wert für den Primärschlüssel ermittelt, der Wert um 1 erhöht und der neue Wert ist der korrekte Primärschlüsselwert für den neuen Datensatz.

Hinweis: Um herauszufinden, ob der Primärschlüssel auf AUTOINCREMENT gesetzt ist, gehen Sie im *Sqliteman* im *Datenbank Explorer* mit der rechten Maustaste auf die gewünschte Tabelle und wählen *Tabelle beschreiben* aus. In der Ausgabe sehen Sie den CREATE-Befehl für die Tabelle und beim Attribut PRIMARY KEY steht AUTOINCREMENT (oder eben nicht…).

Einfügen von mehreren Datensätzen

```
INSERT INTO <Tabelle>
   [ (<Spaltenliste>) ]
   VALUES (<Werteliste1>),
          (<Werteliste2>),
           ...;
```

Es ist möglich, mit einem INSERT-Befehl mehrere Datensätze einzufügen.

Beispiel

```
INSERT INTO Orte
   (PLZ, Ort)
    VALUES (3096, "Oberbalm"),
           (3122, "Kehrsatz"),
           (3150, "Schwarzenburg");
```

Es werden mit einem Befehl die drei Orte *Oberbalm*, *Kehrsatz* und *Schwarzenburg* eingefügt.

Prüfen Sie das Ergebnis der letzten drei INSERT-Befehlen mit:

```
SELECT * FROM Orte
   WHERE PLZ > 3090
   AND PLZ < 3199
   ORDER BY PLZ;
```

Aufgabe 2

Fügen Sie in der Tabelle *Lehrpersonen* die beiden Lehrkräfte mit Namen *Markus Studer* und *Kurt Järmann* ein und notieren Sie den entsprechenden Befehl.

```
insert into lehrpersonen Values (32, "Marcus", "Studer");
insert into lehrpersonen Values (33, "Kurt", "Jährmann");
```

Daten aus einer andern Tabelle einfügen

```
INSERT INTO <Tabelle1>
   [ Spalte1, Spalte2, Spalte3 ... ]
   SELECT Spalte1, Spalte2, Spalte3,...
   FROM <Tabelle2>
   [ WHERE-Klausel ];
```

Es gibt Situationen, wo Sie Daten aus einer Tabelle in eine andere überführen wollen. Natürlich sind das nicht alle Daten (das würde keinen Sinn machen), sondern nur bestimmte Datensätze und/oder nur einzelne Attribute. Es kann auch sein, dass Daten aus zwei Tabellen in einer vereint werden sollen. Für solche Aufgaben kann innerhalb des INSERT-Befehls ein SELECT durchgeführt werden, das von andern Tabellen Daten zurückliefert.

Beispiel

Zur Vorbereitung erstellen wir die neue Tabelle *Zimmer*. Führen Sie folgenden Befehl aus:

```
CREATE TABLE Zimmer (
    ZID INTEGER PRIMARY KEY AUTOINCREMENT,
    Zimmer TEXT
);
```

Wir wollen nun alle Zimmer, in denen das Modul 105 unterrichtet wird, in diese Tabelle einfüllen. Führen Sie den folgenden INSERT durch:

```
INSERT INTO Zimmer (Zimmer)
   SELECT DISTINCT Zimmer FROM Lektionen AS l, Module AS m
```

```
    ON l.FK_MID = m.MID
    WHERE m.Modul LIKE "Modul 105%";
```

UPDATE

Nachdem wir gesehen haben, wie man Datensätze einfügt, wollen wir nun schauen, wie wir bestehende Daten ändern können.

Die UPDATE-Syntax sieht wie folgt aus:

```
UPDATE <Tabelle>
   SET <Spalte1> = <Wert1>
   [ , <Spalte2> = <Wert2> ]
   [ , <Spalte3> = <Wert3> ]
   ...
   [ WHERE-Klausel ];
```

Bei jedem Update muss mindestens ein Attribut angegeben werden. Bei allen Datensätzen, die der WHERE-Bedingung entsprechen, wird der Attributwert der *Spalte1* auf den *Wert1* gesetzt.

Tipp: Seien Sie vorsichtig mit diesem Befehl. Wenn die WHERE-Klausel nicht genau den Anforderungen entspricht, kann grosser Schaden angerichtet werden.
Deshalb: Führen Sie zuerst einen SELECT mit exakt denselben WHERE-Bedingungen aus und prüfen Sie das Resultat, bevor Sie den UPDATE-Befehl absetzen!

Beispiel

Die Postleitzahl von Gasel ist nicht 3140, sondern 3144. Wir möchten das ändern:

```
UPDATE Orte
   SET PLZ = 3144
   WHERE Ort = "Gasel";
```

Aufgabe 3

Wir haben bemerkt, dass wir den Namen eines Lehrers falsch geschrieben haben. Der Lehrer heisst nicht *Marcus Stadler*, sondern *Markus Studer*. Ändern Sie Vor- und Nachname mit einem UPDATE-Befehl.

```
update lehrpersonen SET Vorname = "Markus" WHERE PID=32;
```

UPDATE mit Sub-SELECT
oder: Wie man beim SELECT-Befehl die JOIN-Klausel umgehen kann

Mit einem UPDATE-Befehl können immer nur Datensätze von einer Tabelle geändert werden. Manchmal ist es jedoch notwendig, dass in der WHERE-Bedingung weitere Tabellen verwendet werden, um die gewünschten Datensätze zu ändern. Da wir beim UPDATE-Befehl nicht mit JOIN arbeiten können, müssen wir eine andere Syntax verwenden.

Zur Vorbereitung fügen wir die neue *Klasse INF.1N* ein:

```
INSERT INTO Klassen (Klasse) VALUES ("INF.1N");
```

Wir wollen nun alle Schüler der Klasse *INF.1M* in die Klasse *INF.1N* zügeln.

Dazu geben wir zuerst alle Schüler der Klasse *INF.1M* aus. Der uns vertraute SELECT-Befehl lautet wie folgt:

```
SELECT s.Vorname, s.Nachname
   FROM Schueler AS s
   JOIN Klassen AS k
   ON s.FK_KID = k.KID
   WHERE k.Klasse = "INF.1M";
```

Es gibt eine alternative Möglichkeit für diesen Befehl:

```
SELECT Vorname, Nachname
   FROM Schueler
   WHERE FK_KID =
      (SELECT KID FROM Klassen
         WHERE Klasse = "INF.1M");
```

Wann ist die Variante des SELECT-Befehls mit dem Sub-SELECT zu empfehlen?

Beide Befehle oben geben exakt dasselbe Resultat aus. Sie können bei SQL-Befehlen jeweils diejenige Variante verwenden, die Ihnen besser zusagt. Es gibt Datenbankspezialisten, die die erste und solche, die die zweite Variante empfehlen.

Die Variante mit dem Sub-SELECT hat aber einen grossen Nachteil: Die Spaltenliste für die Ausgabe kann nur Attribute der Tabelle *Schueler* enthalten. Wenn Sie also sowohl die Namen der Schüler als auch die Klasse ausgeben wollen, ist nur die Variante mit der JOIN-Klausel möglich.

UPDATE mit Sub-SELECT

Nun können wir den UPDATE-Befehl formulieren, als Vorlage dient der SELECT-Befehl mit dem Sub-SELECT:

```
UPDATE Schueler
   SET FK_KID =
     (SELECT KID FROM Klassen
        WHERE Klasse = "INF.1N")
   WHERE FK_KID =
      (SELECT KID FROM Klassen
         WHERE Klasse = "INF.1M");
```

Hinweis: Hier haben wir es mit zwei Sub-SELECT zu tun. Einerseits haben wir den Sub-SELECT in der WHERE-Klausel, den wir vom SELECT-Befehl oben übernommen haben. Andererseits verwenden wir einen Sub-SELECT, um die ID der Klasse *INF.1N* zu ermitteln. Dieser Sub-SELECT muss **genau einen Wert** zurückliefern, damit der UPDATE-Befehl erfolgreich ausgeführt werden kann.

Sub-SELECT gibt mehr als einen Wert zurück

"`WHERE FK_ID =`" erwartet genau einen Wert vom Sub-SELECT zurück. Wenn mehr als ein Wert zurückgeliefert wird, erhalten wir eine Fehlermeldung. Auch für diese Situation gibt es eine Lösung: Anstelle von "=" verwenden wir das Schlüsselwort *IN*:

```
SELECT Vorname, Nachname
   FROM Schueler
   WHERE FK_KID IN
      (SELECT KID FROM Klassen
         WHERE Klasse = "INF.1L"
         OR Klasse = "INF.1N");
```

Diese Abfrage gibt alle Schüler der Klassen *1L* und *1N* aus. Bedeutung des SELECT-Befehls: Es werden alle Schüler ausgegeben, die sich in den Klassen befinden, die vom Sub-SELECT zurückgegeben werden.

Aufgabe 4

Geben Sie alle Schüler aus, die in *Niederscherli* oder *Mittelhäusern* wohnen. Verwenden Sie nicht die JOIN-Klausel, sondern Sub-SELECT.

```
SELECT *
FROM schueler
 WHERE  FK_OID IN (SELECT OID
            FROM orte
           WHERE Ort="Niederscherli" OR Ort="Mittelhäusern");
```

Aufgabe 5

Alle Schüler aus der letzten Aufgabe zügeln nach *Gasel*. Notieren Sie den dafür notwendigen UPDATE-Befehl und prüfen Sie am Ende das Ergebnis mit einem SELECT.

```
UPDATE schueler
SET FK_OID =(SELECT OID from orte WHERE Ort="Gasel")
 WHERE FK_OID IN (SELECT OID FROM orte WHERE Ort="Niederscherli" OR
Ort="Mittelhäusern")
```

DELETE

Wenn wir Datensätze einfügen können, muss es natürlich auch möglich sein, diese wieder zu löschen.

Die DELETE-Syntax sieht wie folgt aus:

```
DELETE FROM <Tabelle>
   [ WHERE-Klausel ];
```

Tipp: Seien Sie vorsichtig mit diesem Befehl. Wenn die WHERE-Klausel nicht genau den Anforderungen entspricht, kann grosser Schaden angerichtet werden.
Deshalb: Führen Sie zuerst einen SELECT mit exakt denselben WHERE-Bedingungen aus und prüfen Sie das Resultat, bevor Sie den DELETE-Befehl absetzen!

Beispiel

Beim Ort *Schwarzenburg* haben wir uns getäuscht, wir wollen den Datensatz wieder löschen:

```
DELETE FROM Orte
   WHERE Ort = "Schwarzenburg";
```

Aufgabe 6

Löschen Sie alle Zimmer aus der vorhin erstellten Tabelle *Zimmer*, die sich im 3. Stock befinden. Führen Sie zuerst den SELECT-Befehl aus, der die gewünschten Zimmer ausgibt, und erst danach den DELETE-Befehl.

Habe zwar diese Tabelle nicht würde aber etwa so ausschauen:

```
SELECT * FROM Zimmer WHERE stock=3
DELETE FROM zimmer where stock=3;
```

Aufgaben

Speichern Sie sämtliche SQL-Befehle in die Datei *AB105-04.sql* ab.

Aufgabe 7

Erfassen Sie das neue Modul "Modul 123 –Serverdienste in Betrieb nehmen".

```
insert into module values (9, "Modul 123 - Serverdienste in Betrieb nehmen");
```

Aufgabe 8

Erfassen Sie folgende Firma neu:
Firma = SBB, Adresse = Hilfikerstr. 1, PLZ = 3000, Ort = Bern 65 SBB (der Ort existiert bereits).

Hinweis: Um den Wert des Fremdschlüssels *FK_OID* zu ermitteln, wird ein Sub-SELECT benötigt.

```
insert into firmen values (123, "SBB", "Hilfikerstr. 1", (SELECT oid from orte where PLZ=3000 AND Ort="Bern") )
```

Aufgabe 9

Fügen Sie in *Sqliteman* der Tabelle *Klassen* das Attribut *Fachrichtung* mit dem Datentyp *Text* an.
Füllen Sie anschliessend die Attributwerte ein:

- Klassen INF.1A – 1E mit der Fachrichtung Systemtechnik
- Klassen INF.1F – 1J mit der Fachrichtung Applikationsentwicklung
- Klassen INF.1K – 1N mit der Fachrichtung Betriebsinformatik

(Diese Einteilung entspricht einer Annahme, die Daten dazu sind leider nicht vorhanden…).

```
insert into klassen
values    (14, "INF.1A"),(15, "INF.1B"),(16, "INF.1C"),(17, "INF.1D"),(18, "INF.1E"),
                    (19, "INF.1F"),(20, "INF.1G"),(21, "INF.1H"),(21, "INF.1I"),(22, "INF.1J"),
      (23, "INF.1K"),(24, "INF.1L"),(25, "INF.1M"),(26, "INF.1N");
```

Was ist bezüglich Redundanzen und Inkonsistenzen zu dieser Aufgabe zu sagen?

```
Die Fachrichtung wird nicht angegeben und es gibt doppelte Klassennamen welche nur durch die KlassenID (KID)
identifiziert werden können.
```

Aufgabe 10

Bei allen Lektionen wird das Modul 117 durch das Modul 123 ersetzt.

Hinweis: Verwenden Sie einen Sub-SELECT, um die ID des Moduls 117 zu ermitteln und den Wert in *FK_MID* einzufügen.

```
UPDATE lektionen
SET FK_MID=(SELECT MID from module WHERE Modul LIKE "Modul 123%")
WHERE FK_MID=(SELECT MID from module where Modul LIKE "Modul 117%");
```

Prüfen Sie das Resultat, indem Sie alle Lektionen mit dem Modul 123 ausgeben (Ausgabeliste: Modul, Klasse und Lehrperson).

Aufgabe 11

Da wir von Modul 117 zu Modul 123 gewechselt haben, wollen wir nun das Modul 117 löschen.

```
DELETE FROM module WHERe modul like "Modul 117%";
```

Aufgabe 12

Alle Schüler aus Biel sollen gelöscht werden. Geben Sie zuerst alle Schüler aus Biel aus:

```
SELECT * FROM schueler WHERe FK_OID=(SELECT OID from ORte where Ort="Biel");
```

Wie viele Schüler wohnen in Biel?

```
keine
```

Führen Sie nun das Löschen durch:

```
es gibt keine
```

Zusatzaufgaben

Diese Aufgaben sind schwieriger als die vorherigen und sollen als Herausforderung betrachtet werden!

Aufgabe 13

Erfassen Sie den folgenden Schüler:
Vorname = Marc, Nachname = Muster, Jahrgang = 2000, Ort = Gasel, Firma = SBB, Klasse = INF.1N.

```
insert into schueler values (274, "Marc", "Muster", "", 2000, (select OID FROM orte where Ort="Gasel"),(SELECT FID from firmen
WHERE Firma="SBB"), (SELECT KID from klassen where Klasse="INF.1N"));
```

Prüfen Sie, ob der Schüler erfolgreich eingefügt worden ist:

```
Der ort gasel gab es nicht => Basel
```

Bei allen drei Fremdschlüsseln FK_OID, FK_FID und FK_KID sollte ein Wert vorhanden sein!

Aufgabe 14

Die Klasse *INF.1I* wird nicht mehr von *Ralph Maurer*, sondern neu von *Kurt Järmann* unterrichtet.

Erstellen Sie zuerst den SELECT-Befehl, der folgende Lektionen ausgibt (alle Attribute der Tabelle *Lektionen*): Lehrperson = *Ralph Maurer*, Modul = *105*, Klasse = *INF.1I*.
Verwenden Sie ~~nicht~~ die JOIN-Klausel, ~~sondern Sub-SELECT~~.

```
SELECT * FROM lektionen left join lehrpersonen on lektionen.FK_PID = lehrpersonen.PID WHERE lehrpersonen.Vorname="Ralph"
AND lehrpersonen.Nachname="Maurer" AND lektionen.FK_MID = (SELECT module.MID FROM module WHERE module.Modul
Like "Modul 100%") AND lektionen.FK_KID = (SELECT klassen.KID from klassen where klassen.Klasse = "INF.1I")
```

Erstellen Sie nun den UPDATE-Befehl, um die Lehrperson von *Ralph Maurer* zu *Kurt Järmann* zu wechseln.

```
UPDATE lektionen, lehrpersonen
SET FK_PID=(SELECT PID from lehrpersonen WHERE vorname="Kurt" AND nachname="Jährmann")
WHERE lehrpersonen.Vorname="Ralph"
AND lehrpersonen.Nachname="Maurer"
AND lektionen.FK_MID = (SELECT module.MID FROM module WHERE module.Modul Like "Modul 100%")
AND lektionen.FK_KID = (SELECT klassen.KID from klassen where klassen.Klasse = "INF.1I")
```

Prüfen Sie das Resultat, indem Sie den geänderten Datensatz mit folgenden Attributen ausgeben: *Klasse* (= *INF.1I*), *Modul* (= *Modul 105…*), *Vorname* und *Nachname* Lehrperson (= *Kurt Järmann*).

```
SELECT * FROM "lektionen" WHERE FK_PID=(SELECT PID from lehrpersonen WHERE Vorname="Kurt"
Nachname="Jährmann");
```

Aufgabe 15

Geben Sie *Vorname*, *Nachname* und *Ort* von jedem Schüler aus, Sortierreihenfolge *Ort* aufsteigend. Verwenden Sie einen INNER JOIN.

```
SELECT * FROM schueler INNER join orte on orte.OID = schueler.FK_OID order by orte.Ort asc
```

Wie viele Schüler sind das?

270

Löschen Sie den Ort Burgdorf aus der Tabelle Orte.

```
Delete from orte where ort="Burgdorf";
```

Führen Sie den SELECT-Befehl nochmals aus. Wie viele Schüler werden nun ausgegeben?

267

Wie erklären Sie sich den Unterschied?

Der join funktioniert nicht da es den ort nicht mehr gibt

Wie können wir trotzdem alle Schüler inklusive Ort ausgeben?

Ja ausser die schüler die in Burgdorf wohnen

Wir haben einen Zustand in der Datenbank herbeigeführt, den wir unter allen Umständen vermeiden wollen. Was ist damit gemeint? Und wie können wir einen solchen Zustand verhindern?

Dateninkonsestenz

Man kann die Felder auf not null setzen sodass das Feld FK_OID in schueler nie leer ist