Databases Autumn 2025

Hand-In Exercise 1

October 25, 2025

Aiysha Frutiger
Jannick Seper
Luis Tritschler

Total Points	

Task	Points

Databases
Autumn 2025
Exercise 1

Aiysha Frutiger
Jannick Seper
Luis Tritschler
Page 1 of 8

Task 1

Given excerpt of the relational schema plus added relations and integrety constrain befor the first
bullet point.

```
1  ===== Entity =====
2  CREATE TABLE Lecturer (
3          LecturerID   INT PRIMARY KEY,
4          FirstName    VARCHAR(255),
5          LastName     VARCHAR(255),
6          Title        VARCHAR(255)
7  );
8
9  CREATE TABLE Lecture (
10         Title              VARCHAR(255) PRIMARY KEY,
11         CreditPoints       INT,
12         SemesterWeekHours  INT,
13         LecturerID         INT,
14         FOREIGN KEY (LecturerID) REFERENCES Lecturer (LecturerID)
15 );
16
17 CREATE TABLE Exercise (
18         ExID          INT PRIMARY KEY,
19         No            INT,
20         Semester      VARCHAR(255),
21         LectureTitle  VARCHAR(255),
22         FOREIGN KEY (LectureTitle) REFERENCES Lecture (Title)
23 );
24
25 CREATE TABLE Author (
26         AuthorID     INT PRIMARY KEY,
27         LastName     VARCHAR(255),
28         FirstName    VARCHAR(255),
29         Title        VARCHAR(255)
30 );
31
32 CREATE TABLE Task (
33         TaskID       INT PRIMARY KEY,
34         Points       INT,
35         Difficulty   INT,
36         Text         VARCHAR(65535),
37         AuthorID     INT,
38         FOREIGN KEY (AuthorID) REFERENCES Author (AuthorID)
39 );
40
41 ===== Relationships =====
```

Databases
Autumn 2025
Exercise 1

Aiysha Frutiger
Jannick Seper
Luis Tritschler
Page 2 of 8

```sql
42  CREATE TABLE Contains (
43          ExID        INT,
44          TaskID      INT,
45          PRIMARY KEY (ExID, TaskID),
46          FOREIGN KEY (ExID)  REFERENCES Exercise (ExID),
47          FOREIGN KEY (TaskID) REFERENCES Task (TaskID),
48          Sequence    INT
49  );
50
51  CREATE TABLE Consists_of (
52          SuperTaskID INT,
53          SubTaskID   INT,
54          PRIMARY KEY (SuperTaskID, SubTaskID),
55          FOREIGN KEY (SuperTaskID) REFERENCES Task (TaskID),
56          FOREIGN KEY (SubTaskID)   REFERENCES Task (TaskID),
57          Sequence    INT
58  );
59
60  ===== Integrity =====
61  CREATE ASSERTION contains_only_super_tasks
62    CHECK ( NOT EXISTS (
63      SELECT *
64      FROM Contains c
65      WHERE EXISTS (
66        SELECT *
67        FROM Consists_of co
68        WHERE co.SubTaskID = c.TaskID
69      )
70    ) );
71
72  CREATE ASSERTION consists_of_non_recursive
73    CHECK ( NOT EXISTS (
74      SELECT *
75      FROM Consists_of x
76      WHERE EXISTS (
77        SELECT *
78        FROM Consists_of y
79        WHERE y.SuperTaskID = x.SubTaskID
80      )
81    ) );
```

Databases
Autumn 2025
Exercise 1

Aiysha Frutiger
Jannick Seper
Luis Tritschler
Page 3 of 8

- *The title of the lecture has to be unique and may not be altered if any exercise is available for the lecture.*
  The first part of the point is already enforced bc the title of the lecture is a `PRIMARY KEY` and therefore must be unique. The second part can be enforced with this addition.

```
1  CREATE TABLE Exercise (
2          ExID            INT PRIMARY KEY,
3          No              INT,
4          Semester        VARCHAR(255),
5          LectureTitle    VARCHAR(255),
6          FOREIGN KEY (LectureTitle) REFERENCES Lecture (Title)
7  -->     ON UPDATE RESTRICT
8  );
```

- *For a lecture, no more than 10 credit points may be awarded.*
  This part can be enforced with this assertion.

```
1  CREATE ASSERTION no_more_than_10_credits
2          CHECK (NOT EXISTS (
3                  SELECT * FROM Lecture
4                  WHERE CreditPoints > 10))
5          NOT DEFERRABLE;
```

- *Lecturers may give multiple lectures.*
  This part is beeing enforced with this.

```
1  CREATE TABLE Lecture (
2          Title               VARCHAR(255) PRIMARY KEY,
3          CreditPoints        INTEGER,
4          SemesterWeekHours   INTEGER,
5          LecturerID          INTEGER,
6  -->     FOREIGN KEY (LecturerID) REFERENCES Lecturer (LecturerID)
7  );
```

- *A lecture may include several exercises. An exercise always belongs to exactly one lecture.*
  The first part is already enforced via the foreign key while for the second part we have to add the `NOT NULL` so we guaratee that e exersise must be in one lecture.

```
1  CREATE TABLE Exercise (
2          ExID            INT PRIMARY KEY,
3          No              INT,
4          Semester        VARCHAR(255),
5  -->     LectureTitle    VARCHAR(255) NOT NULL,
6  -->     FOREIGN KEY (LectureTitle) REFERENCES Lecture (Title)
7          ON UPDATE RESTRICT
8  );
```

Databases
Autumn 2025
Exercise 1

Aiysha Frutiger
Jannick Seper
Luis Tritschler
Page 4 of 8

- *Before a new author is entered into the system, it should be checked that no other author with the same first name, last name and title is present.*
  This part is beeing enforced with this.

```
CREATE TABLE Author (
        AuthorID    INT PRIMARY KEY,
        LastName    VARCHAR(255),
        FirstName   VARCHAR(255),
        Title       VARCHAR(255),
-->     UNIQUE (FirstName, LastName, Title)
);
```

Task 2   Foreign Keys

2a) **Foreign Keys**

For the relation `Lecture`, the following foreign key is defined in SQL:

```
CREATE TABLE Lecture (
   ...,
   FOREIGN KEY (fk_lecturer) REFERENCES Lecturer(pk_lecturer)
);
```

In the following, this foreign key is simulated using *(i)* an assertion and *(ii)* triggers.

(i) **Assertion (static integrity constraint)**

An `ASSERTION` ensures that no tuple in `Lecture` references a non-existent `Lecturer`. We formulate this as a double `NOT EXISTS` statement that expresses the same semantics as a foreign key.

```
CREATE ASSERTION FK_Lecture_Lecturer_OK
CHECK (
  NOT EXISTS (
    SELECT 1
    FROM Lecture L
    WHERE L.fk_lecturer IS NOT NULL
      AND NOT EXISTS (
        SELECT 1
        FROM Lecturer R
        WHERE R.pk_lecturer = L.fk_lecturer
      )
  )
)
DEFERRABLE INITIALLY IMMEDIATE;
```

This assertion guarantees that for every non-NULL value in `Lecture.fk_lecturer`, a matching `Lecturer.pk_lecturer` exists. The condition is checked globally and can be deferred to commit time using the `DEFERRABLE` clause.

Databases
Autumn 2025
Exercise 1

Aiysha Frutiger
Jannick Seper
Luis Tritschler
Page 5 of 8

(ii) **Triggers (dynamic integrity constraint)**
The same referential integrity can be dynamically enforced by triggers (E–C–A pattern):
- **Child-side:** Prevent inserting or updating a `Lecture` with a non-existent `fk_lecturer`.
- **Parent-side:** Prevent deleting or updating a `Lecturer` that is still referenced by a `Lecture`.

Child-side triggers (on `Lecture`)

```
1   CREATE TRIGGER Lecture_FK_Check_Ins
2   BEFORE INSERT ON Lecture
3   WHEN (NEW.fk_lecturer IS NOT NULL AND
4         NOT EXISTS (SELECT 1 FROM Lecturer R
5                     WHERE R.pk_lecturer = NEW.fk_lecturer))
6   ( ROLLBACK WORK );
7
8   CREATE TRIGGER Lecture_FK_Check_Upd
9   BEFORE UPDATE OF fk_lecturer ON Lecture
10  REFERENCING NEW AS Lnew
11  WHEN (Lnew.fk_lecturer IS NOT NULL AND
12        NOT EXISTS (SELECT 1 FROM Lecturer R
13                    WHERE R.pk_lecturer = Lnew.fk_lecturer))
14  ( ROLLBACK WORK );
```

Parent-side triggers (on `Lecturer`)

```
1   CREATE TRIGGER Lecturer_Ref_Block_Del
2   BEFORE DELETE ON Lecturer
3   REFERENCING OLD AS Pold
4   WHEN (EXISTS (SELECT 1 FROM Lecture L
5                 WHERE L.fk_lecturer = Pold.pk_lecturer))
6   ( ROLLBACK WORK );
7
8   CREATE TRIGGER Lecturer_Ref_Block_Upd
9   BEFORE UPDATE OF pk_lecturer ON Lecturer
10  REFERENCING OLD AS Pold NEW AS Pnew
11  WHEN (EXISTS (SELECT 1 FROM Lecture L
12                WHERE L.fk_lecturer = Pold.pk_lecturer))
13  ( ROLLBACK WORK );
```

- The **child-side** triggers prevent inserting or updating a lecture referencing a non-existent lecturer.
- The **parent-side** triggers prevent deleting or changing a lecturer ID if at least one lecture still refers to it.
- The `ROLLBACK WORK` aborts the transaction on violation (`NO ACTION`-like).

2b) **Simulating `FOREIGN KEY (FkExam) REFERENCES Exam(PkExam) ON DELETE SET NULL` with triggers**

We enforce referential integrity procedurally:

Databases
Autumn 2025
Exercise 1

Aiysha Frutiger
Jannick Seper
Luis Tritschler
Page 6 of 8

- **Child-side (Student).** Block INSERT/UPDATE that sets FkExam to a non-existent Exam.PkExam (allow NULL).

- **Parent-side (Exam) on DELETE.** Before deleting an Exam row, set Student.FkExam := NULL for all referencing students (ON DELETE SET NULL).

- **Parent-side (Exam) on UPDATE PkExam.** Default is *NO ACTION*; block if referenced.

Child-side checks (INSERT/UPDATE on Student)

```
1  CREATE TRIGGER Student_FK_Check_Ins
2  BEFORE INSERT ON Student
3  WHEN ( NEW.FkExam IS NOT NULL AND
4         NOT EXISTS (SELECT 1 FROM Exam E
5                       WHERE E.PkExam = NEW.FkExam) )
6  ( ROLLBACK WORK );
7
8  CREATE TRIGGER Student_FK_Check_Upd
9  BEFORE UPDATE OF FkExam ON Student
10 REFERENCING NEW AS Snew
11 WHEN ( Snew.FkExam IS NOT NULL AND
12        NOT EXISTS (SELECT 1 FROM Exam E
13                      WHERE E.PkExam = Snew.FkExam) )
14 ( ROLLBACK WORK );
```

Parent-side action (DELETE on Exam) — simulate ON DELETE SET NULL

```
1  CREATE TRIGGER Exam_Delete_SetNull
2  BEFORE DELETE ON Exam
3  REFERENCING OLD AS Eold
4  (
5    UPDATE Student
6    SET FkExam = NULL
7    WHERE FkExam = Eold.PkExam
8  );
```

Parent-side block (UPDATE of PkExam) — simulate ON UPDATE NO ACTION

```
1  CREATE TRIGGER Exam_Block_Update_PK
2  BEFORE UPDATE OF PkExam ON Exam
3  REFERENCING OLD AS Eold NEW AS Enew
4  WHEN ( EXISTS (SELECT 1 FROM Student S
5                   WHERE S.FkExam = Eold.PkExam) )
6  ( ROLLBACK WORK );
```

Notes.

- Ensure Student(FkExam) allows NULL; otherwise SET NULL would fail.

- The DELETE trigger clears references before parent removal.

- Using ROLLBACK WORK follows the lecture's "block on violation" pattern.

Databases
Autumn 2025
Exercise 1

Aiysha Frutiger
Jannick Seper
Luis Tritschler
Page 7 of 8

Task 3

Using Assertions or Triggers because integrity constraints span multiple relations and not only one (Create Table Statement).

a) Using Assertions because constraints too general and not specific event.
e.g. no new orders when ...

U2109: each User must have SUM(c.CreditScore) >= -1000 creditpoints over all credits.

```
CREATE ASSERTION CreditLimitU2109
CHECK(NOT EXISTS(
    SELECT u.UID
    FROM User u
    JOIN Regulation r ON u.Country = r.Country
    JOIN Credit c ON u.UID = c.UID
    WHERE r.RegulationCode = 'U2109'
    GROUP u.UID
    HAVING SUM(c.CreditScore) < -1000))
DEFFERABLE INITIALLY DEFERRED
```

U2304: each User must have SUM(c.CreditScore) >= (-0.1 * MAX(b.BetAmount))

```
CREATE ASSERTION CreditLimitU2109
CHECK(NOT EXISTS(
    SELECT u.UID
    FROM User u
    JOIN Regulation r ON u.Country = r.Country
    JOIN Credit c ON u.UID = c.UID
    JOIN Bet b ON u.UID = b.UID
    WHERE r.RegulationCode = 'U2304'
    GROUP u.UID
    HAVING SUM(c.CreditScore) < (-0.1 * MAX(b.BetAmount))))
DEFFERABLE INITIALLY DEFERRED
```

b)

Task 4

When we say something is **DEFERRABLE INITIALLY DEFERRED**, it means that the database will wait until the end of a transaction before checking certain rules, such as foreign key constraints. This is useful when two tables depend on each other. For example, a `Lecture` table and a `Lecturer` table. A lecture record might point to the lecturer who gives it, while the lecturer

Databases

Autumn 2025

Exercise 1

Aiysha Frutiger

Jannick Seper

Luis Tritschler

Page 8 of 8

record might also depend on the lecture. If we try to insert both of these in the same transaction, the database might raise an error because when we insert the first one, the other does not yet exist. If the rule is **NOT DEFERRABLE**, the database checks immediately, and the operation fails. If it is **DEFERRABLE INITIALLY IMMEDIATE**, we would need to manually tell the database to delay the check. However, if it is **DEFERRABLE INITIALLY DEFERRED**, the database automatically waits until the transaction ends to perform the checks. This allows both inserts to complete first, and the database only verifies the rules afterward, which helps prevent errors.