Integrity Constraints

1 Foreign Keys and Constraints: Theory

1. What is a foreign key constraint? Solution:
   - A foreign key is an attribute of a relation that references a primary key of another relation. Thus, a foreign key implies a referential constraint between two relations.

2. Why are such constraints important? Solution:
   - Foreign key constraints are important to ensure consistency between relations, i.e., to prevent an inconsistent state of the database.

3. What is referential integrity? Solution:
   - Referential integrity means that for every value of one attribute, the same value must exist in another attribute of a different relation, i.e., in a relational database, any field in a table that is declared a foreign key, either contains a reference to an existing primary key of another table, or a NULL value (if the field was not explicitly defined as not null).

2 Foreign Keys and Constraints: Practice

Consider the following relational schema:

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader</td>
<td>(readerId, firstName, lastName, address, city, dateOfBirth)</td>
</tr>
<tr>
<td>Book</td>
<td>(isbn, title, author, numberOfPages, yearOfPublication, publisherName)</td>
</tr>
<tr>
<td>Publisher</td>
<td>(publisherName, placeOfPublication)</td>
</tr>
<tr>
<td>Categories</td>
<td>(categoryName, includedIn)</td>
</tr>
<tr>
<td>Copy</td>
<td>(ISBN, copyNumber, shelf, position)</td>
</tr>
<tr>
<td>Loan</td>
<td>(readerId, ISBN, copyNumber, returnDate)</td>
</tr>
<tr>
<td>BookCategory</td>
<td>(ISBN, categoryName)</td>
</tr>
</tbody>
</table>

1. What problems may appear if this schema is realized in SQL without considering integrity constraints? Solution:
   - If primary keys are not used, duplicates of the same data item are possible, e.g., several categories with the same name could exist.
   - Without foreign key constraints, dangling references after deleting or updating are not prevented, e.g., books that no longer exist could still be referenced by the ‘Copy’ table.
• Semantically incorrect values in certain columns could not be forbidden. Standard SQL allows no constraints in the data domain such as, e.g., prohibiting birthdays in the future.

2. What foreign keys are necessary and which strategies have to be applied when changing them? Please give an explanation why you chose a particular strategy. Solution:

• E.g. Book: publisherName references Publisher(publisherName): ON DELETE SET NULL or NO ACTION, ON UPDATE NO ACTION or CASCADE. If a publisher is deleted, the books which are published by this publisher should exist without the publisher entry (publisher entry could be set to NULL). One can also avoid the deletion. Changing the name of a publisher should be prevented (NO ACTION) or passed (CASCADE).

• E.g. Categories: includedIn references another Category: ON DELETE SET NULL, ON UPDATE NO ACTION or CASCADE. If a super-category is deleted, the sub-category should still exist without it. Changing the name of a super-category should be passed (CASCADE) or prevented (NO ACTION).

• E.g. BookCategory: ISBN references Book, categoryName references Category. For both: ON DELETE CASCADE, ON UPDATE CASCADE. If a referenced tuple in a N:M relationship is deleted, all references should be deleted.

• E.g. Copy: ISBN references Book: ON DELETE CASCADE, ON UPDATE CASCADE. If a tuple of book is deleted, the dependent tuples must be deleted as well.

• E.g. Loan: readerId references Reader, (ISBN, copyNumber) reference Copy. For both: ON DELETE CASCADE, ON UPDATE CASCADE. If a tuple of reader/copy is deleted, the dependent tuples must be deleted as well.

3. What is the implication of deleting a publisher? What is the consequence of updating a readerId? In both cases, take into account the keys and rules of your solution for question 2.2. Solution:

• If no books of the publisher are available, the publisher is deleted. There are no other effects. If books are available, the deletion is either stopped (NO ACTION concerning the books) or the publisher of these books is set to NULL.

• The modification of the reader number leads to a modification of the reader number at any lending of the reader. If the DBMS does not support ON UPDATE CASCADE, the update is prevented.

4. Enforce that only readers who live in Zurich can be inserted.

Solution: In the definition (SQL DDL) of the table “Reader”: (city in (‘Zürich’))

5. Enforce that a reader can only borrow up to 20 books! Give a solution using CHECK and a solution using Trigger.

Solution:

With CHECK:

In Loan:

CONSTRAINT CHECK (NOT EXISTS (SELECT COUNT(*) FROM Loan GROUP BY readerId HAVING COUNT(*) > 20 ))

Note, that this is not supported in certain DBMS, like DB2. DB2 does not allow subqueries in check.

With a trigger:

CREATE TRIGGER AMAX NO CASCADE
BEFORE INSERT ON Loan
REFERENCING NEW AS newl
FOR EACH ROW
WHEN ( (SELECT COUNT (*) FROM Loan
WHERE Loan.readerId = newl.readerId) >= 20)
BEGIN ATOMIC
SIGNAL SQLSTATE ‘-1’ SET Message_TEXT = 'Illegal Insert - too many books
per reader';
END

SQLSTATE is given as an example, a correct SQLSTATE should be defined as documented.
This Trigger treats only the case when the number of borrowed books is increased by INSERT (“BE-
FORE INSERT”). It is still possible that another reader borrows the books and then the reader
number is changed. In this case one needs another trigger with “ON UPDATE”.
This trigger works in DB2.

3 Foreign Keys and Constraints: Practice 2

Consider the following relational schema:

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emp</td>
<td>(eid: integer, ename: string, age: integer, salary: real)</td>
</tr>
<tr>
<td>Works</td>
<td>(eid: integer, did: integer, pcttime: integer)</td>
</tr>
<tr>
<td>Dept</td>
<td>(did: integer, dname: string, budget: real, managerid: integer)</td>
</tr>
</tbody>
</table>

1. Give an example of a foreign key constraint that involves the Dept relation. What are the options
for enforcing this constraint when a user attempts to delete a Dept tuple?

Solution:
These foreign key constraints are necessary:

- Works : FOREIGN KEY did REFERENCES Dept(did)
- Dept: FOREIGN KEY managerid REFERENCES Emp(eid)

When deleting a Dept tuple, we need to remove the respective Works tuple(s). This can be done
with the ON DELETE CASCADE rule.

2. Write the SQL statements to create the preceding relations, including appropriate versions of all
primary and foreign key integrity constraints.

Solution:
CREATE TABLE Emp ( eid INT, ename VARCHAR, age INT, salary REAL,
PRIMARY KEY (eid) );

CREATE TABLE Dept ( did INT, dname VARCHAR, budget REAL, managerId INT,
PRIMARY KEY (did),
FOREIGN KEY managerid REFERENCES Emp(eid) );

CREATE TABLE Works ( eid INT, did INT, pcttime INT,
PRIMARY KEY (eid, did),
FOREIGN KEY eid REFERENCES Emp(eid) ON DELETE CASCADE,
FOREIGN KEY did REFERENCES Dept(did) ON DELETE CASCADE);

Alternative: if we allow some departments to not have a manager, we could define the managerId
field of the Dept table as managerId INT ON DELETE SET NULL.

3. Define the Dept relation in SQL so that every department is guaranteed to have a manager.

Solution:
managerId INT NOT NULL.
4. Write an SQL statement to delete the Toy department. Given the referential integrity constraints you chose for this schema, explain what happens when this statement is executed.

Solution:
DELETE FROM Dept WHERE dname = 'Toy';
The tuple of the toy department is going to be deleted. Additionally, all tuples from the Works relation that reference the Toy department are going to be deleted as well.

4 Bonus: More SQL Practice

Consider the following relational table ‘Employees’ that stores employee data:

<table>
<thead>
<tr>
<th>EmployeeID</th>
<th>EmployeeName</th>
<th>ManagerID</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Peter Müller</td>
<td>15</td>
<td>85,000</td>
</tr>
<tr>
<td>2</td>
<td>Marco Berlusconi</td>
<td>NULL</td>
<td>350,000</td>
</tr>
<tr>
<td>3</td>
<td>Stephan Meier</td>
<td>2</td>
<td>210,000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

The ManagerID attribute is a foreign key that references the EmployeeID of the direct superior of a given employee. Marco Berlusconi is the CEO and has no direct superior, hence, the NULL value in the corresponding record.

**Task**: Write an SQL query that calculates the total salary of employees that work under direct supervision of a specific manager, i.e., that produces an ordered result of the following form (highest amount of total money at the top):

<table>
<thead>
<tr>
<th>EmployeeID</th>
<th>EmployeeName</th>
<th>TotalMoney</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Marco Berlusconi</td>
<td>2,498,750</td>
</tr>
<tr>
<td>3</td>
<td>Stephan Meier</td>
<td>798,900</td>
</tr>
<tr>
<td>8</td>
<td>Petra Hunziker</td>
<td>678,950</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Solution**:

```sql
SELECT a.EmployeeID, a.EmployeeName, SUM(b.Salary) AS TotalMoney
FROM Employees AS a INNER JOIN Employees AS b ON a.EmployeeID=b.ManagerID
GROUP BY a.EmployeeID, a.EmployeeName
ORDER BY SUM(b.Salary) DESC;
```