Database Theory & Applications DTA(M)

BOKYUNG LEE 2431088L
12-10-2019

Contents

Task 1. Relational Schema Modelling

Task 1.1:

Table1. STUDENT(<u>SID</u>, SNAME, HCID, HCNAME, <u>TID</u>, TNAME, JYEAR)
Candidate Key: {SID, TID}

In case of STUDENT relation, it is in 1NF.
Each attributes in STUDENT relation refers only to atomic values. Moreover, this relation has redundant and repeated values in every attributes.

e.g.
FD1: Full FD. The combination of student ID (SID) and team ID (TID) determines join year (JYEAR).
{SID, TID} -> JYEAR

FD2: Partial FD. The student ID (SID) determines the student name (SNAME).
SID -> SNAME

FD3: Partial FD. The student ID (SID) determines student's home city ID (HCID).
SID -> HCID

FD4: Partial FD. The team ID (TID) determines the team name (TNAME).
TID -> TNAME

FD5: Partial FD. The student's home city ID (HCID) determines the home city name (HCNAME).
HCID -> HCNAME

Table2. TOPIC(SID, ADVNAME, <u>ADVID</u>, TOPIC)
Candidate Key: {ADVID}
Foreign key: {SID}

The TOPIC relation in table 2 is in the 1NF.
FD1 and FD2 are both full FD, since there is only one primary key in this relation.

e.g.
FD1: Full FD. The advisor's ID (ADVID) determines the advisor's name (ADVNAME).
ADVID -> ADVNAME

FD2: Partial FD. The advisor's ID (ADVID) determines the topic (TOPIC).
ADVID -> TOPIC

Table 3. TEXTBOOK(<u>COURSE, ADVID, TEXTBOOK</u>)

Textbook relation has no functional dependencies as it has no non-key attributes.
{ADVID} is a foreign key from {ADVID} in Topic relation.
{TEXTBOOK} attribute does not depend on {ADVID} or {COURSE}.

{COURSE} attribute has no functional dependencies with other two attributes.

Therefore, Textbook relation is full-keyed relation.

Task 1.2:

The provided tables did not assume any primary keys and foreign keys therefore insertion anomalies would not occur, however, based on the FDs identified in Task 1.1, each relation now have assumed candidate keys. In this Task 1.2, the solutions would be given with assuming every tables have candidate keys.

Table 1. STUDENT relation could cause update anomaly and delete anomaly due to the redundancy of values. Moreover, it could cause insertion anomaly if programmer wants to insert new team into the STUDENT relation until a student joins as SID attribute and TID attribute are candidate keys. The possible example of update anomaly for STUDENT relation is drawn below:

> Context: If the name of home city (HCNAME) of student ID=1 changes from ‘Bradford’ to ‘London’ then it is necessary to enforce consistency.

Value ‘Bradford’ has two tuples in STUDENT relation, hence should update all tuples referring to student ID=1.

Table 2. TOPIC relation could also cause update anomaly and delete anomaly due to the redundancy furthermore, if new topic (TOPIC) values added without advisor’s ID (ADVID), it could also cause insertion anomaly as ADVID attribute is a candidate key in TOPIC relation. The possible example of delete anomaly for TOPIC relation is drawn below:

> Context: If the value ‘Database’ in TOPIC attribute is deleted, all the advisor’s ID who advises in ‘Database’ field will be deleted as well.

As the value ‘Database’ in TOPIC attribute has redundancy, ADVID=1 and ADVID=3 will deleted also and SID=1,2,3 will deleted also with leaving only SID=1 with values for TOPIC=’Java’.

Table 3. TEXTBOOK relation could cause all three anomalies as it has redundancy among values and foreign key (ADVID) from TOPIC relation. The possible example of insertion anomaly is given below:

> Context: If a tuple inserted in TEXTBOOK with ADVID=4, because the ADVID is a foreign key of ADVID attribute in TOPIC relation, this insertion would not be success.

In TOPIC relation, advisor’s ID is only given from 1 to 3, therefore inserting advisor’s ID in TEXTBOOK relation could be available from 1 to 3 as well. However, as ADVID is a foreign key, this attribute still can have null value.

Task 1.3:

To normalise the tables: STUDENT and TOPIC to the highest possible Normal Form (NF), STUDENT relation had normalised to four associated relations first than TOPIC relation had normalised to two associated relations.

Table1. STUDENT relation is assumed to have five functional dependencies:

- FD1: SID->SNAME
- FD2: HCID->HCNAME
- FD3: TID->TNAME
- FD4: {SID, TID}->JYEAR
- FD5: SID->HCID

As FDs can be merged with other FDs that has same depended attribute, FD1 and FD5 could be merged together as follows:

- FD6 (FD1+FD5): The student ID (SID) determines the student name (SNAME) and home city ID (HCID):
  SID->{SNAME, HCID}

Therefore, Table1 can be split into Table4 and Table5:
Table4. STUDENT2(<u>SID</u>, SNAME, HCID, HCNAME): Student2 relation is in 2NF.
STUDENT2 relation is assumed to have two functional dependencies:

- FD6 (FD1+FD5): SID->{SNAME,HCID}
- FD2: HCID->HCNAME

Table5. STUDENT3(<u>SID</u>,<u>TID</u>,TNAME,JYEAR): Student3 relation is in 1NF.
STUDENT3 relation is assumed to have two functional dependencies:

- FD3: TID->TNAME
- FD4: {SID, TID}->JYEAR

As both Table4 and Table5 is in 2NF and 1NF respectively, it is possible to normalise higher.
Firstly, Table4 can be split into Table6 and Table7 based on the FDs:

Table6. STUDENT_DETAIL(<u>SID</u>, SNAME, HCID). STUDENT_DETAIL relation is in BCNF.
STUDENT_DETAIL relation has one primary key (SID) and assumed to have one functional dependency.

- FD6 (FD1+FD5): SID->{SNAME,HCID}

The left-hand side of FD in STUDENT_DETAIL relation is superkey and there is no non-prime attribute which is transitively dependent on the primary key.

Table7. HOME(<u>HCID</u>,HCNAME) is in BCNF. As the left-hand side of FD is superkey.
HOME relation has one primary key (HCID) and assumed to have one functional dependency.

- FD2: HCID->HCNAME

Secondly, the normalisation of Table5 is Table8 and Table9 based on the FDs:

Table8. TEAM(<u>TID</u>, TNAME) is in BCNF. As the left-hand side of FD is superkey.
TEAM relation has one primary key (TID) and assumed to have one functional dependency.

- FD3: TID->TNAME

Table9. JOIN(<u>SID</u>,<u>TID</u>,JYEAR).
JOIN relation has two primary keys (SID, TID) and assumed to have one functional dependency.

- FD4: {SID, TID}->JYEAR

As the left-hand side of FD are superkeys and neither SID->JYEAR nor TID->JYEAR hold true, JOIN relation is in BCNF.

After finishing normalisation for STUDENT relation, TOPIC relation had normalised as below:

Table2. TOPIC (SID, ADVNAME, <u>ADVID</u>, TOPIC). TOPIC relation is assumed to have two functional dependencies:

- FD1: ADVID->ADVNAME
- FD2: ADVID->TOPIC

As the two FDs depended by same attribute (ADVID), it can be merged:

- FD3: ADVID->{ADVNAME, TOPIC}

Therefore, TOPIC relation can be split into Table10 based on its FD3:

Table10. ADVISOR(<u>ADVID</u>, ADVNAME, TOPIC) is in BCNF. As the left-hand side of FD is superkey.
ADVISOR relation holds one primary key (ADVID) and one functional dependency.

- FD3: ADVID->{ADVNAME, TOPIC}

Yet, in TOPIC relation, there is one attribute (SID) that does not have any functional dependency with primary key or other non-primary key. Therefore, could build a table with ADVID (which is a primary key that has a full FDs) only as below:

Table11. ADVISEE(<u>SID</u>, <u>ADVID</u>).
ADVISEE relation holds two primary keys (SID, ADVID) with no functional dependency.

Task 1.4:

Table3. TEXTBOOK(COURSE, ADVID, TEXTBOOK)
Insertion, deletion, update anomalies are possible in original TEXTBOOK relation. Although almost every elements in TEXTBOOK relation has redundancy, only two value which are ‘Java & DB’ and ‘Oracle & DB’ in TEXTBOOK attribute shows no reputation therefore could be updated and deleted. However, other values in TEXTBOOK attribute and every values in both COURSE attribute and ADVID attribute has redundancy therefore cause updating and deleting anomalies, while as the relation has no functional dependency with each attributes, insertion anomalies would only occur when inserting values to ADVID attribute since it is a foreign key from TOPIC relation.

Possible anomalies from original TEXTBOOK relation are illustrated below:

- Insertion anomalies.

Since there is no functional dependency in the TEXTBOOK relation, and as there are no assumed primary key, inserting values in TEXTBOOK relation is available.

- Update anomalies:
    - Update the value of COURSE attribute which has ADVID=1 to ‘Programming’.

The value 1 in ADVID attribute has redundancy of 5: points ‘Databases’ and ‘Advanced DB’. Therefore ‘Databases’ and ‘Advanced DB’ will updated after the query runs which occur update anomalies.

- Delete anomalies:
    - Delete the course name value where advisor’s ID is 2, could occur delete anomaly.

According to the deleting query, the pointed value of COURSE attribute is ‘Databases’ but will also delete two textbooks which are ‘Intro of DB’ and ‘DB Management’. As the value has redundancy, this sql will occur delete anomaly.

To eliminate or reduce these anomalies, it is necessary to split the relation into smaller relation.

If the TEXTBOOK relation can be split into two relation when COURSE attribute becomes a splitting attribute. The reason of choosing COURSE attribute in a splitting attribute is that the number of values it has; although advisor’s ID in TEXTBOOK relation also has only two values, COURSE attribute seems more suitable because it is not a foreign key.

With the splitting attribute (COURSE) the new relations are given as below:

Table12. COURSEBOOK(COURSE, TEXTBOOK).

COURSE attribute and TEXTBOOK attribute are primary keys and there is no foreign key in this relation. As both attributes have no dependency, it is necessary to make two attributes in primary keys.

COURSE	TEXTBOOK
Databases	Intro to DB
Databases	DB Management
Advanced DB	Intro to DB
Advanced DB	Java & DB
Advanced DB	Oracle & DB

Table13. WORKS_IN(COURSE, ADVID).

Both COURSE attribute and ADVID attribute are primary keys in this relation and foreign keys from COURSEBOOK relation and TOPIC relation respectively. Due to the data redundancy, both attributes should be primary keys, therefore could cause more anomalies while inserting.

COURSE	ADVID
Databases	1
Databases	2
Advanced DB	1

In the light of two relations above, the possible anomalies from the split tables will be compared with the original table.

- Insertion anomalies:
  - If value 4 had inserted to ADVID attribute in WORKS_IN relation, this can cause insertion anomaly because COURSE attribute cannot be null as it is a primary key also.
  - If value 'Programming' had inserted to COURSE attribute in COURSEBOOK relation, this can cause insertion anomaly because TEXTBOOK attribute cannot have null value as it is a primary key also.

Inserting values into COURSEBOOK relation and WORKS_IN relation cause insertion anomaly since all the attributes are primary keys.

- Update anomalies from original relation:
  - Update the value of COURSE attribute which has ADVID=1 to 'Programming'.

Still, updating ADVID=1 could still cause update anomalies due to the redundancy of COURSE values.

- Delete anomalies from original relation will not occur:
  - Delete the course name value where advisor's ID is 2, could occur delete anomaly.

As COURSE attribute and ADVID attribute is in the WORKS_IN relation, the deleted value would be only one which is 'Databases'.

Unfortunately, it is considered to have no significant impact by splitting the relation with TEXTBOOK relation. However, if the TEXTBOOK relation has one primary key, for instance Row_Num attribute that indicates row number of every rows, as it is unique, insertion anomalies, update anomalies and delete anomalies would reduce significantly.

Task 2. SQL Statements

Task 2.1:

Setting conditions for deleting depends on the attribute that is primary key or not, if the key is both foreign key and prime key, it should set in default as prime key does not accept null value. If there are no redundancy, the setting could be cascade but this is setting will not be used frequently to preserve other associated values.

In case of setting conditions for updating, unless there are redundancy in the relation, it should set in cascade.

Table1.STUDENT had split in:
Table6. STUDENT2(<u>SID</u>, SNAME, HCID)
Table7. HOME(<u>HCID</u>,HCNAME)

Table8. TEAM(<u>TID</u>, TNAME)
Table9. JOIN(<u>SID</u>,<u>TID</u>,JYEAR)

```
CREATE SCHEMA STUDENT;

CREATE TABLE STUDENT.HOME (
HCID                  VARCHAR(5)   NOT NULL,
HCNAME                VARCHAR(15)  NOT NULL,
CONSTRAINT HOMEPK
 PRIMARY KEY(HCID));

-- Set null when delete foreign key HCID.
-- Updating HCID is available however, as HCID attribute is a primary key in HOME relation, --
values in HOME relation should also updated.
CREATE TABLE STUDENT.STUDENT2 (
SID                   INT                  NOT NULL,
SNAME                 VARCHAR(15)  NOT NULL,
HCID                  VARCHAR(5),
CONSTRAINT STUPK
 PRIMARY KEY(SID),
CONSTRAINT STUHOMEFK
 FOREIGN KEY (HCID) REFERENCES STUDENT.HOME(HCID)
        ON DELETE SET NULL             ON UPDATE CASCADE);

CREATE TABLE STUDENT.TEAM(
TID           INT                  NOT NULL,
TNAME         VARCHAR(15)  NOT NULL,
CONSTRAINT TEAMPK
 PRIMARY KEY(TID));

 -- Set default when deleting SID foreign key in table JOIN
--        according to the primary keys in current table and STUDENT2 table.
--        However, updating is available.
-- Set default when deleting TID foreign key in table JOIN
--        according to the primary keys in current table and TEAM table.
--        However, updating is available.
CREATE TABLE STUDENT.JOIN(
SID           INT            NOT NULL,
TID           INT            NOT NULL,
JYEAR         INT            NOT NULL,
CONSTRAINT JOINPK
 PRIMARY KEY(SID, TID),
CONSTRAINT JOINSTUFK
 FOREIGN KEY(SID) REFERENCES STUDENT.STUDENT2(SID)
        ON DELETE SET DEFAULT          ON UPDATE CASCADE,
CONSTRAINT JOINTEAMFK
 FOREIGN KEY(TID) REFERENCES STUDENT.TEAM(TID)
        ON DELETE SET DEFAULT          ON UPDATE CASCADE);
```

Table2. TOPIC had split in:
Table10. ADVISOR(ADVID, ADVNAME, TOPIC)
Table11. ADVISEE(ADVID, SID)

```
CREATE TABLE STUDENT.ADVISOR(
ADVID           INT       NOT NULL,
ADVNAME         VARCHAR(15) NOT NULL,
TOPIC           VARCHAR(15),
CONSTRAINT ADVPK
 PRIMARY KEY(ADVID));
```

```
-- Set default when deleting ADVID according to the primary keys in current table and ADVISOR table.
-- Updating had set default because this relation has redundancies in values.
CREATE TABLE STUDENT.ADVISEE(
ADVID INT       NOT NULL,
SID   INT       NOT NULL,
CONSTRAINT TOPPK
 PRIMARY KEY(ADVID, SID),
CONSTRAINT TOPADVFK
 FOREIGN KEY(ADVID) REFERENCES STUDENT.ADVISOR(ADVID)
        ON DELETE SET DEFAULT           ON UPDATE SET DEFAULT);
```

Task 2.2:

1. **SQL 1:**

```
SELECT H.HCNAME AS HOME_CITY,COUNT(SID) AS STUDENTS
FROM STUDENT.HOME H, STUDENT.STUDENT2 S
WHERE H.HCID=S.HCID
GROUP BY H.HCNAME;
```

Result table:

HOME_CITY	STUDENTS
Rome	1
Bradford	1
Glasgow	1
Munich	1

2. **SQL 2:**

```
SELECT T.TNAME AS TEAM, COUNT(DISTINCT(J.SID)) AS STUDENTS_IN_2001
FROM STUDENT.TEAM T, STUDENT.JOIN J
WHERE T.TID=J.TID
AND J.JYEAR<=2001
```

GROUP BY T.TNAME;

Result table:

TEAM	STUDENTS_IN_2001

To show the number of each team, it is available by using left join-on as below:

```
SELECT T.TNAME AS TEAM, COUNT(DISTINCT(J.SID)) AS STUDENTS_IN_2001
FROM STUDENT.TEAM T
LEFT JOIN STUDENT.JOIN J
ON T.TID=J.TID
AND J.JYEAR<=2001
GROUP BY T.TNAME;
```

Result table:

TEAM	STUDENTS_IN_2001
Team1	0
Team2	0

However, counts for the year 2020 per team will show each numbers without changing to above sql (that includes left join-on).

TEAM	STUDENTS_IN_2020
Team1	3
Team2	4

3. **SQL 3:**

```
SELECT R.ADVNAME AS ADVISOR, COUNT(DISTINCT(E.SID)) AS ADVISEE
FROM STUDENT.ADVISOR R, STUDENT.ADVISEE E
WHERE R.ADVID=E.ADVID
GROUP BY R.ADVNAME
HAVING COUNT(E.SID)>10;
```

Result table:

ADVISOR	ADVISEE

As there are no advisees according to advisor's more than 10, the count() function did not print results on the screen, however if showing the names of the advisors with more than 0 advisees, the result will be shown as below:

SQL:

```
SELECT R.ADVNAME AS ADVISOR, COUNT(DISTINCT(E.SID)) AS ADVISEE
FROM STUDENT.ADVISOR R, STUDENT.ADVISEE E
WHERE R.ADVID=E.ADVID
```

```
GROUP BY R.ADVNAME
HAVING COUNT(E.SID)>0;
```

ADVISOR	ADVISEE
Burn	1
Edward	1
McReader	2

4. **SQL 4:**

```
SELECT TOPIC, COUNT(DISTINCT(ADVID)) AS ADVISOR
FROM STUDENT.ADVISOR
GROUP BY TOPIC
ORDER BY COUNT(ADVID) DESC
LIMIT 1;
```

Result Table:

TOPIC	ADVISOR
Database	2

5. **SQL 5:**

```
SELECT S.SNAME AS STUDENT, COUNT(J.JYEAR) AS COUNT_TEAM
FROM STUDENT.STUDENT2 S, STUDENT.JOIN J
WHERE S.SID=J.SID
GROUP BY S.SNAME
HAVING COUNT(J.JYEAR)>1
ORDER BY COUNT(J.JYEAR) DESC;
```

Result table:

STUDENT	COUNT_TEAM
Chris	2
John	2
Stella	2

To show every students who joined teams since 2001, for instance:

SQL:

```
SELECT S.SNAME AS STUDENT, COUNT(J.JYEAR)
FROM STUDENT.STUDENT2 S, STUDENT.JOIN J
WHERE S.SID=J.SID
AND J.JYEAR>=2001
GROUP BY S.SNAME
```

```
HAVING COUNT(J.JYEAR)>=0
ORDER BY COUNT(J.JYEAR) DESC;
```

The following result table would be:

STUDENT	COUNT_TEAM
Chris	2
John	2
Stella	2
Philip	1

Task 3. Data Cleaning & Analytics Query

Task 3.1:

```
SELECT SSN, FName, Salary
FROM TASK3.EMPLOYEE
GROUP BY SSN, FName, Salary
HAVING COUNT(*)>1
ORDER BY SSN;
```

Task 3.2:

```
SELECT SSN, FName, Salary, COUNT(*) AS DUPLICATE_EMPLOYEE
FROM TASK3.EMPLOYEE
GROUP BY SSN, FName, Salary
ORDER BY SSN;
```

Result table:

SSN	FName	Salary	DUPLICATE_EMPLOYEE
1	Philip	30000	3
2	Stella	40000	2
3	Lona	50000	1

Task 3.3:

```
SELECT A.SSN, A.FName, A.Salary
FROM (SELECT *, ROW_NUMBER() OVER(ORDER BY Salary DESC) Salary_Rank
            FROM TASK3.EMPLOYEE
            GROUP BY SSN) AS A
WHERE Salary_Rank<=3
ORDER BY Salary_Rank;
```

Since the SSN attribute had declared as the primary key of the relation EMPLOYEE, pretend EMPLOYEE relation could be as below:

SSN	FName	Salary
1	Philip	30000
2	Philip	30000
3	Stella	40000
4	Stella	40000
5	Iona	50000
6	Philip	30000

-- Row_number() function allocates distinct number according to the following phrase (order by phrase in this case) therefore could show top 3 highest salary values by giving less or same as 3 in WHERE clause.

The SQL query than should finally show only the top 3 values ordered by salary as below:

SSN	FName	Salary
5	Iona	50000
3	Stella	40000
4	Stella	40000