KUTAISI INTERNATIONAL UNIVERSITY

Homework 3

Fundamentals of Databases 1
Dimitri Tabatadze
19:25 Thursday 26th October, 2023

1. (a) done.
   (b) SQL Error (1062): Duplicate entry '<timeslot>-<member>' for key 'PRIMARY'
   (c) SQL Error (1062): Duplicate entry '<timeslot>-<deviceID>' for key 'index 4' which I suppose is the UNIQUE key.
2. (a) `to_days(`entryDate`) - to_date('2015-01-01') >= 0`
   (b) `` `isTrainer` in (0, 1) ``
   (c) `` `parentName` <> `memName` `` (also set *Allow Null* to false)
3. - We don't need to store the same information twice — DoB already encodes age.
   - ```
     ALTER TABLE `member`
         ADD COLUMN `age` INT UNSIGNED AS
         (year(from_days(to_days(curdate()) - to_days(`DoB`))))
         VIRTUAL AFTER `parentName`;
     ```
   - It indeed does work.
4. Did all of these but I can't really demonstrate.
5. The given relation is in the first normal form. The following, are the assumed functional dependencies:
   - {article_ID, storage_location, supplier} → stock
   - {supplier} → telephone_supplier
   - article_ID → {article_description, article_price}

   Given the data, we can see that article_price → article_description, but in the real world, we may have two different ID's with different prices but with the same drescription.

supplier	telephone_supplier
A	123456
B	234567
C	345678
D	589944

article_ID	article_price	article_description
1	200	skis
2	150	tent
3	100	snowshoes
4	50	boots

article_ID (FK)	storage_location	supplier (FK)	stock
1	Vienna	A	50
1	Munich	B	25
2	Berlin	C	10
3	Berlin	A	50
4	Vienna	A	150
4	Cologne	A	5
4	Munich	B	15
4	Munich	D	5
     ```

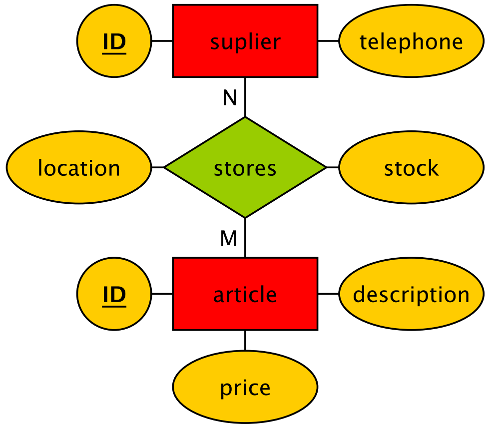

Figure 1: Reverse engineered ER model out of my normalized relations I got in the task 5.

6. 1) The candidate keys:
    - {SSN, ProjectID}
    - {SSN, Function}
    - startDate

   I don't consider startDate as a valid PK since more than one freelancer can satrt working on projects at the same time. Same for {SSN, Function}, the same freelancer can work on multiple projects with the same Function.

   2) Without looking at the context, we get these FDs:
    - {SSN, ProjectID} $\rightarrow$ {Function, startDate}
    - {SSN, Function} $\rightarrow$ {ProjectID, startDate}
    - {Company_Name, Function} $\rightarrow$ Company_ID
    - ProjectID $\rightarrow$ Company_ID $\rightarrow$ Company_Name
    - Company_Name $\rightarrow$ Company_ID
    - startDate $\rightarrow$ {SSN, Company_ID, Company_Name, Function, ProjectID}

   But considering the context, we get:
    - {SSN, ProjectID} $\rightarrow$ {Function, startDate}
    - ProjectID $\rightarrow$ Company_ID $\rightarrow$ Company_Name

   3) 1st normal form.

   4) We get the following relations:
    - gig: {[SSN:int, ProjectID:int (FK), startDate:datetime, Function:string]}
    - company: {[Company_ID:int, Company_Name:string]}
    - project: {[ProjectID:int, Company_ID:int (FK)]}

   giving the tables:

PorjectID	Company_ID (FK)
1	1
2	2
3	3
4	4
5	4

Company_ID	Company_Name
1	TBG Bank
2	Bank Imereti
3	Telavi Wines
4	Poti Port

SSN	PorjectID (FK)	startDate	Function
01136	1	2020-01-01	Project Manager
74589	2	2020-10-01	Developer
55587	3	2019-10-01	Project Manager
01136	4	2021-02-01	Tester
12345	4	2019-11-01	Developer
12345	4	2021-01-01	Tester

7. 1)
   - user: {[userName:string, isModerator:bool, userMail:string, selfDescription:string, country:string, moderates:int (FK) (UNIQUE)]}
   - forum: {[forumID:int, forumTitle:string, forumdesc:string, moderator:string (FK) (UNIQUE)]}
   - post: {[postNumber:int, belongsTo:int (FK), postTitle:string, date:datetime, content:string, author:string (FK), respondsTo:int (FK)]}
   - like: {[postNumber:int (FK), userName:string (FK), date:datetime]}
   - registration: {[forumID:int (FK), userName:string (FK), regDate:datetime]}

   2) Got 5 relations.

   3) done.

   4) done.

   5) Look-up table is a table indexed by one attribute that gives us access to the second attribute in $O(\log n)$ time. It'd be suitable to either find the country of a user, or all users of a certain country (both cases would need diferent orderings in the LUT).

   6) Okay.

8. 1) The first variant's schema, using vertical integration.
   Let's say that **address** = {country, city}
   - waterBody: {[waterID:int, waterName:string, type:enum]}
   - RunningWater: {[waterID:int (FK), length:int, flowRate:int, milagePoint:int, flowsInto:int (FK)]}
   - StandingWater: {[waterID:int (FK), surface:int]}
   - project: {[projectID:int, projectTitle:string, totalCost:int]}
   - researcher: {[email:string, rName:string]}
   - organization: {[name:string, country:string, city:string]}
   - researchOn: {[waterID:int (FK), projectID:int (FK)]}
   - finances: {[orgName:string (FK), projectID:int (FK), percentage:float]}
   - provByWorksOn: {[projectID:int (FK), rEmail:string (FK), function:string, orgName:string (FK)]}

   2) The first variant's schema, using horizontal integration.
   Let's say that **address** = {country, city}
   - RunningWater: {[waterID:int, waterName:string, length:int, flowRate:int, milagePoint:int, flowsInto:int (FK)]}
   - StandingWater: {[waterID:int, waterName:string, surface:int]}
   - project: {[projectID:int, projectTitle:string, totalCost:int]}
   - researcher: {[email:string, rName:string]}
   - organization: {[name:string, country:string, city:string]}
   - researchOn: {[waterID:int (FK), projectID:int (FK)]}
   - finances: {[orgName:string (FK), projectID:int (FK), percentage:float]}
   - provByWorksOn: {[projectID:int (FK), rEmail:string (FK), function:string, orgName:string (FK)]}

   3) For the first variant, with horizontal I get 8 relations and with vertical I get 9 relations.

   4) Done.

   5) There are none.

   6) The differences between the first and the second variant look like this:

- RunningWater: {[waterID:int, waterName:string, length:int, flowRate:int, milagePoint:int, flowsInto:int (FK)]}
- StandingWater: {[waterID:int, waterName:string, surface:int]}
- project: {[projectID:int, projectTitle:string, totalCost:int]}
- researcher: {[email:string, rName:string, providedBy:int (FK)]}
- organization: {[name:string, country:string, city:string]}
- researchOn: {[waterID:int (FK), projectID:int (FK)]}
- finances: {[orgName:string (FK), projectID:int (FK), percentage:float]}
- worksOn: {[projectID:int (FK), rEmail:string (FK), function:string]}
- ~~provByWorksOn: {[projectID:int (FK), rEmail:string (FK), function:string, orgName:string (FK)]}~~

7) Okay.

9. 1) Without context, startDate can be considered as a candidate key. Other than that, the obvious candidate keys are {SSN, ProjectID} and {Emp_ID, ProjectID}.

   2)
   - {SSN, ProjectID} → {Function, startDate}
   - SSN → Emp_ID

   3) 1st normal form. Emp_ID is not fully functionally dependent on the entire key.

   4)

Emp_ID (FK)	PorjectID	startDate	Function
1	ERP	2020-01-01	Project Manager
2	ERP	2020-10-01	Developer
3	Portal	2019-10-01	Project Manager
4	CRM	2021-02-01	Tester
4	Portal	2019-11-01	Developer
4	Linux	2021-01-01	Tester

Emp_ID	SSN
1	01136
2	74589
3	55587
4	12345

10. 1) Not very accurately, since the judgement will be based only on our understanding. This relation can be considered to be in the 2nd normal form.

    2)
    - ISBN → {title, authorID, publishinHouse, publishingDate}
    - authorID → {autorFName, authorLName, authorEmail, authorTelNumber}

    3)
    - Book: {[ISBN:int, title:string, publishinHouse:string, publishingDate:date, authorID:int (FK)]}
    - Author: {[authorID:int, autorFName:string, authorLName:string, authorEmail:string, authorTelNumber:int]}