Sean Higgins

2-11-2015

COMP 4115

Prof. Strange

Problem Set 2

* 1. City: CREATE TABLE City(

Name = VARCHAR(20),

State = VARCHAR(20),

Population = INT,

Elevation = INT,

PRIMARY KEY (Name, State),

FOREIGN KEY (State) REFERENCES State(Name)

);

State: CREATE TABLE State(

Name = VARCHAR(20) ,

Region = VARCHAR(20),

PRIMARY KEY (Name)

);

Mayor: CREATE TABLE Mayor(

Name = VARCHAR(50),

City = VARCHAR(20),

State = VARCHAR(20),

Parth = ENUM(‘Democrat’, ‘Republican’),

PRIMARY KEY (Name, City, State),

FOREIGN KEY (State) REFERENCES State(Name),

FOREIGN KEY (City) REFERENCES City(Name)

);

Governor: CREATE TABLE Governor(

Name = VARCHAR(50),

State = VARCHAR(20),

Party = ENUM(‘Democrat’, ‘Republican’),

PRIMARY KEY (Name, State),

FOREIGN KEY (State) REFERENCES State(Name)

);

* 1. The valid drop sequences would be ‘Mayor, Gov., City, State’ and ‘Gov., Mayor, City, State’.
  2. The only way to prevent data integrity issues from table drops would be by having all tables have no dependencies on one another. This is impossible, as, at the bare minimum, the relation between governor and state depends on the existence of a given state. Without enforcing that link between a ‘Governor’ table and a ‘State’ table, a governor can govern a state that doesn’t exist, which is a data integrity issue in and of itself. If you do enforce that link, however, you must then delete the ‘Governor’ table before the ‘State’ table, making it impossible to have any order of table deletions be successful.

1. Facebook\_Posts(post\_id, timestamp, message\_body, location, priority, user\_id, real\_name, tagged\_user, image) Only 1 image and 1 tagged user per post.

post\_id 🡪(timestamp, message\_body, location, priority, user\_id, real\_name, tagged\_user, image)

user\_id 🡪(real\_name)

image 🡪(tagged\_user, location)

message\_body 🡪 (image)

(user\_id, timestamp) 🡪 (post\_id)

(image, tagged\_user) 🡪 (real\_name)

1. Assumption: PK is (custNo, license, rentalDate) there is only 1 shop per city, a customer can only rent 1 car per day.
   1. Given this primary key, it is impossible to insert a new car into the database, as each entry requires not only a license plate, but a rental date and a customer number. A car that is just being added will have never been rented, having neither a rental date or customer number. Similarly, if you delete the last customer who rented a given vehicle, you lose all data you had on that vehicle, deleting its make, model, and license plate info rather than just the customer that rented it. Finally, update anomalies can easily occur in the ‘license’ column specifically. The shop number of a specific car has to be re-entered over and over and over again, despite the shop number being consistent for a given car. If there is an error in inputting the data—the user types in 02 instead of 01, for instance—then you’d have the same car at two locations, which is impossible.
   2. license 🡪 (make, model, shopNo)

shopNo 🡪 shopLocation //And vice versa

custNo 🡪 custName

(license, rentalDate) 🡪 custNo //And (custNo, rentalDate) 🡪 license

* 1. From these functional dependencies, one simply needs to split up the original table into these four smaller tables, keeping consistency between the different instances of ‘license’, ‘shopNo’, and ‘custNo’, consistent via foreign keys.

CREATE TABLE licenseInfo(

license = CHAR(8),

make = VARCHAR(20),

model = VARCHAR(20),

shopNo = INT,

PRIMARY KEY(license)

FOREIGN KEY(shopNo) REFERENCES shopLocation(shopNo)

);

CREATE TABLE shopLocation(

shopNo = INT,

shopLocation = VARCHAR(20),

PRIMARY KEY(shopNo)

);

CREATE TABLE custName(

custNo = INT,

custName = VARCHAR(30)

PRIMARY KEY(custNo)

);

CREATE TABLE rentalDate(

license = CHAR(8),

rentalDate = DATE,

custNo = INT,

PRIMARY KEY(license, rentalDate),

FOREIGN KEY(custNo) REFERENCES custName(custNo),

FORIENG KEY(license) REFERENCES licenseInfo(license)

);

1. RoomsInABuilding(roomNum, professor, wing, department)

|  |  |  |  |
| --- | --- | --- | --- |
| roomNum | professor | wing | department |
| 1123 | Smith | North | Comp. Sci. |
| 2234 | John | South | Physics |
| 2225 | Alice | South | Physics |
| 1111 | Bob | North | Comp. Sci. |

With the assumption that the department can be determined by which wing a room is in, and that the department can be determined by a room number naming convention—11XX for CS, 22XX for Physics, etc.—then this table is in 2NF, but not 3NF. The professor assigned to an office, which wing the room is in, and the department can all be determined by the room number, making the table 2NF as all attributes are determined by the primary key, but the wing can also determine the department, excluding the table from being 3NF, as ‘wing’ is a non-key attribute.

* 1. An atomic attribute is an attribute that cannot be broken up into meaningful smaller parts, i.e., the attribute ‘Full Name’ can be broken up into ‘First Name’, ‘Middle Name’, and ‘Last Name’.
  2. HouseOwnerInfo(firstName, lastName, SSN, address)

Where address(streetName, streetNum, city, state, ZIP)

* 1. The schema on p9 is better as it more accurately reflects the relationship and dependency of subject number and room number on the teacher, time, and day. It does so by getting rid of unnecessary ‘leaf’ relationships and objects that clutter up the ‘root’ relationship, the TIME\_TABLE.
  2. What is missing from the normal forms we’ve covered in class is a way to handle multiple keys and relations in the same relationship. In the Supply-Part example, with the relation:

SUPPLY(supplyNum, supplyName, part)

Where PART(partNum, partName, color, quantity)

In this relation, some data, namely ‘color’ is derived not only from the ‘PART’ relation, but from the ‘SUPPLY’ relation as well; you need both the supplyNum and the partNum in order to determine the color of a specific part. This kind of derivation with multiple keys is not accounted for using the current normal forms.