Solution to Homework 3

1. The following are some of the relations transformed from the ER diagram for the Student Registration System (note that some changes have been made due to the creation of a single-attribute key for Classes):

TAs(B#, level, pay_rate, office, office_hour_start_time, office_hour_end_time)
Courses(dept_code, course#, title, credits, deptname)
Classes(classid, dept_code, course#, sect#, year, semester, start_time, end_time, limit, size, room, TA_B#) /* note: classid is added to serve as a single-attribute key */
Faculty(B#, first_name, last_name, rank, office, email, phone#, deptname)
Enrollments(Student_B#, classid, lgrade, ngrade)

Do the following for each relation schema:

(a) (22 points) Identify all non-trivial functional dependencies based on the Requirements Document (but take into consideration that classid is added as the primary key for Classes). For this question, we also make the following assumptions: (1) each dept_code corresponds to a unique department and vice versa; (2) only faculty members in the same department could share an office and a phone number. Don't make other assumptions about the data. Use the union rule to combine the functional dependencies as much as possible to avoid having multiple functional dependencies with the same left-hand side but different right-hand side. Furthermore, if a functional dependency is redundant (i.e., it can be derived from the ones you keep), it should not be included.

Answer:

TAs(B#, level, pay_rate, office, office_hour_start_time, office_hour_end_time)
FDs: B# → level office office_hour_start_time office_hour_end_time
level → pay_rate

Courses(dept_code, course#, title, credits, deptname)
FDs: dept_code course# → title
dept_code → deptname
deptname → dept_code
Course# → credits

Classes(classid, dept_code, course#, sect#, year, semester, start_time, end_time, limit, size, room, TA_B#)
FDs: classid → dept_code course# sect# year semester start_time end_time limit size room TA_B#
dept_code course# sect# year semester → classid

Faculty(B#, first_name, last_name, rank, office, email, phone#, deptname)
FDs: B# → first_name last_name rank office email phone#
email → B#
office → deptname
phone# → deptname

Enrollments(<u>Student_B#, classid,</u> lgrade, ngrade)
FDs: Student_B# classid → lgrade
lgrade → ngrade
ngrade → lgrade

(b) (22 points) Use functional dependencies identified in Question 1(a) to determine whether or not the schema is in 3NF or in BCNF. Justify your conclusion. You need to compute all candidate keys for each relation first.

Answer: Note that if a schema is in BCNF, it is also in 3NF, and if a schema is not in 3NF, it is also not in BCNF.

TAs(<u>B#</u>, level, pay_rate, office, office_hour_start_time, office_hour_end_time)
Candidate key: B#
The schema is NOT in 3NF because in level → pay_rate, we have a non-prime non-trivially depending on a non-superkey.

Courses(<u>dept_code, course#</u>, title, credits, deptname)
Candidate keys: dept_code course#, deptname course#
The schema is not in 3NF because in Course# → credits, we have a non-prime non-trivially depending on a non-superkey.

Classes(<u>classid</u>, dept_code, course#, sect#, year, semester, start_time, end_time, limit, size, room, TA_B#)
Candidate keys: classid, dept_code course# sect# year semester
The schema is in BCNF because the left-hand side of each non-trivial FD, namely either classid or (dept_code course# sect# year semester), is a superkey.

Faculty(<u>B#</u>, first_name, last_name, rank, office, email, phone#, deptname)
Candidate keys: B#, email
The schema is in not in 3NF because in office → deptname, we have a non-prime non-trivially depending on a non-superkey. Phone# → deptname also makes the schema not in 3NF.

Enrollments(<u>Student_B#, classid,</u> lgrade, ngrade)
Candidate key: Student_B# classid
The schema is not in 3NF because in lgrade → ngrade, we have a non-prime non-trivially depending on a non-superkey. ngrade → lgrade also makes the schema not in 3NF.

(c) (20) For each schema that is not in 3NF, decompose it into 3NF schemas using Algorithm LLJD-DPD-3NF. Show the result after each step of the algorithm, i.e., show the candidate keys (from Question 1(b)), show the minimal cover (Step 2), show the decomposition based on functional dependencies in the minimal cover (Step 3), and mention whether an additional schema needs to be added to the decomposition (Step 4). Don't forget to underscore the primary key of each new relation.

Answer: Four of the five schemas from Question 1(b) are not in 3NF and they are TAs, Courses, Faculty and Enrollments.

Decompose TAs:

(1) All candidate keys: B#
(2) Minimal cover: B# → level, B# → office, B# → office_hour_start_time, B# → office_hour_end_time, level → pay_rate
(3) Decomposition: R1 = B# level office office_hour_start_time office_hour_end_time
R2 = level pay_rate
(4) No change because R1 contains the candidate key

Decompose Courses:

(1) All candidate keys: dept_code course#, deptname course#
(2) Minimal cover: dept_code course# → title, dept_code → deptname, deptname → dept_code Course# → credits
(3) Decomposition: R1 = dept_code course# title
R2 = dept_code deptname /* alternatively deptname could be the primary key
R3 = course# credits
(4) No change because R1 contains a candidate key

Decompose Faculty:

(1) All candidate keys: B#, email
(2) Minimal cover: B# → first_name, B# → last_name, B# → rank, B# → office, B# → email, B → phone#, email → B#, office → deptname, phone# → deptname
(3) Decomposition: R1 = B# first_name last_name rank office email phone#
R2 = office deptname
R3 = phone# deptname
(4) No change because R1 contains a candidate key

Decompose Enrollments:

(1) All candidate keys: Student_B# classid
(2) Minimal cover: B# classid → lgrade, lgrade → ngrade, ngrade → lgrade
(3) Decomposition: R1 = B# classid lgrade
R2 = lgrade ngrade
(4) No change because R1 contains the candidate key

2. Consider the following table schema for accounts in a banking system:

Accounts(acct#, owner_id, owner_name, acct_type, balance, interest_rate, time_opened)

It has the following functional dependencies F = {acct# → own_id acct_type balance time_opened, owner_id → owner_name, acct_type balance → interest_rate}.

(1) (5 points) Explain why the schema is not in BCNF.

Answer: There is just one candidate key: acct#.

The schema is not in BCNF because owner_id → owner_name is non-trivial and owner_id is not a superkey. acct_type balance → interest_rate also makes the schema not in BCNF.

(2) (12 points) Use Algorithm LLJD-BCNF to decompose it. Show the iterations and steps.

Answer: Let the initial decomposition be D = {acct# owner_id owner_name acct_type balance interest_rate time_opened}.

1st iteration: owner_id → owner_name makes the schema not in BCNF, decompose the original schema into (owner_id owner_name) and (acct# owner_id acct_type balance interest_rate time_opened). Now D = {owner_id owner_name, acct# owner_id acct_type balance interest_rate time_opened}. The first schema is already in BCNF (only 2 attributes).

2nd iteration: For schema acct# owner_id acct_type balance interest_rate time_opened, the only candidate key is still acct#. acct_type balance → interest_rate makes the schema not in BCNF. Decompose this schema into (acct_type balance interest_rate) and (acct# owner_id acct_type balance time_opened). Now D = {owner_id owner_name, acct_type balance interest_rate, acct# owner_id acct_type balance time_opened}. In acct_type balance interest_rate, acct_type balance form the candidate key so the schema is already in BCNF (no other non-trivial functional dependencies). In acct# owner_id acct_type balance time_opened, acct# is the only candidate key and there is no other non-trivial functional dependencies, so it is also in BCNF.

Final result: D = {owner_id owner_name, acct_type balance interest_rate, acct# owner_id acct_type balance time_opened}

(3) (5 points) Is your decomposition dependency-preserving? Justify your answer.

Answer: Yes. owner_id → owner_name is preserved by owner_id owner_name and acct_type balance → interest_rate is preserved by acct_type balance interest_rate.

.

3. (14 points) Prove or disprove the following rules:

(a) (8 points) {B → CD, AB → E, E → C} |= {AE → CD}
(b) (6 points) {B → CD, AD → E} |= {AB → E}

When proving a rule, you can use all the six inference rules (i.e., reflexivity rule, augmentation rule, transitivity rule, decomposition rule, union rule and pseudotransitivity rule). To disprove a rule, construct a relation with appropriate attributes and tuples such that the tuples of the relation satisfy the functional dependencies on the left of the rule but do not satisfy the functional dependency on the right of the rule.

Answer:

(a) Disapprove: From the table below, we can see that B → CD, AB → E, E → C are all true. However, AE → D is not true (look at the second and third tuples, they have the same values under AE, i.e., (a2, e2), but they have different values under D, i.e., d and d1), which makes AE → CD not true. This means that AE → CD cannot be derived from B → CD, AD → E, E → C.

A	B	C	D	E
a1	b	c	d	e1
a2	b	c	d	e2
a2	b1	c	d1	e2

(b) Prove: By B → CD and decomposition rule: B → D
By B → D, AD → E and pseudotranstivity rule: AB → E