CENG352 Written Assignment 1

Bartu Kılıçkaya

2380640

1. XML and JSON

1.1 XML

a)

```
<X>
  <A>
      <B>
          <A>one</A>
          <A>two</A>
      </B>
      <C>three</C>
  </A>
  <A>
      <A>four</A>
      <B>
          <B>five</B>
          <B>six</B>
      </B>
      <C>seven</C>
  </A>
</X>
```

b)

i. three, four, seven

ii. three

iii. five, six

iv. one, two, five, six

v. five, six

vi. one, two, five, six

vii. four, five, six, seven

1.2 JSON


```
{
	"Customers": [{
			"cid": 1,
			"name": "John Smith",
			"email": "john@gmail.com",
			"Products": [{
					"pid": 100,
					"pname": "iPhone14",
					"price": 999.00,
					"date": "2/15/23"
				},
				{
					"pid": 101,
					"pname": "Samsung S21",
					"price": 799.00,
					"date": "2/18/23"
				}
			]
		},
		{
			"cid": 2,
			"name": "Jane Doe",
			"email": "jane@gmail.com",
			"Products": [{
					"pid": 101,
					"pname": "Samsung S21",
					"price": 799.00,
					"date": "2/20/23"
				},
				{
					"pid": 102,
					"pname": "MacBook Air",
					"price": 1299.00,
					"date": "2/22/23"
				}
			]
		},
		{
			"cid": 3,
			"name": "Bob Johnson",
			"email": "bob@yahoo.com",
			"Products": []
		}
	],
	"Unordered Products": [{
			"pid": 103,
			"pname": "Bose Headphones",
			"price": 299.00
		},
		{
			"pid": 104,
			"pname": "iPad Air",
			"price": 599.00
		}
	]
}
```


The model above does not avoid redundancy since the details of products whose pids 101, are listed twice under two different customers. To avoid redundancy, we could use a separate array for products and refer to them using pid.

For instance:

```
"cid": 1,
"name": "John Smith",
"email": "john@gmail.com",
"Products": [100, 101]
```


III. Database Design

3.1. BCNF Decomposition

a) First trying to find one length keys A,B,C,D,E,F,G but none of these gives us R so we try two length keys.

{A,F},{E,F},{C,F}. Since using these keys separately with given fds give us {A,B,C,D,E,F,G}. Additionally these are minimal since we cannot further reduce the set {A,F},{E,F},{C,F} and get R.

b) Because the keys are {A,F},{E,F},{C,F} and left hand side of the dependencies must be a candidate key. If we find at least one bad fd, then this R is not in BCNF. Since

in AB->CD , AB is not a candidate key, R doesn't satisfy the condition for BCNF.

c)

Decompose R with the fd that violates the BCNF, which is F->G:

R1={FG}: F->G (BCNF)

R2={ABCDEF}:AB->CD,C->E,EF->A (Not in BCNF)

Decompose R2 with the fd that violates the BCNF, which is C->E:

R3={CE}: C->E (BCNF)

R4={ABCDF}:AB->CD (Not in BCNF)

Decompose R4 with the fd that violates the BCNF, which is AB->CD:

R5={ABCD}:AB->CD (BCNF)

R6={ABF}:ABF->ABF (BCNF)

R1={F,G}; F->G , R3={C,E}; C->E, R5={A,B,C,D}; AB->CD,R6={A,B,F}

d)

i) No, its not dependency preserving, because we lost the dependencies EF → A, AG → B

ii) Since the decompositions are made according to the keys, the BCNF decomposition above is lossless.

3.2 3NF Decomposition

a)

Make RHS of each FD a single attribute:

ABC->C, ABC->G ,A->C,D->E,DE->G,DH->A,DH->C,BH->G,CH->D, CH->E

Eliminate redundant attributes from LHS:

Eliminate BC from ABC->C since A->C

Eliminate C from ABC->G since AB->C

Eliminate E from DE->G since D->E

AB->G ,A->C,D->E,D->G,DH->A,DH->C,BH->G,CH->D, CH->E

Delete redundant FDs from F

DH->C,CH->E,

U = {AB->G ,A->C,D->E,D->G,DH->A,BH->G,CH->D} is minimal cover

b)

U1 = {AB->G}, U2={A->C},U3={D->E,D->G},U4={DH->A},U5={BH->G}, U6={CH->D}

R1={ABG:AB->G}, R2={AC:A->C},R3={DEG:D->E,D->G},

R4={ADH:DH->A},R5={BGH:BH->G},R6={CDH:CH->D}

R0 = {BDFH:} (added this since none of the R's contains the key BDFH)

c)

Yes, because all of the relations' LHS contains candidate key, so its in BCNF form.

3.3 Finding Functional Dependencies

a) Functional Dependencies

- A->D

- C->B

```
select count(*)
from t t1
join t t2
on t1.a = t2.a
where t1.b != t2.b
-- returns 3286, therefore A->B doesn't hold

select count(*)
from t t1
join t t2
on t1.a = t2.a
where t1.c != t2.c
-- returns 4620, therefore A->C doesn't hold

select count(*)
from t t1
join t t2
on t1.a = t2.a
where t1.d != t2.d
-- returns 0, therefore A->D holds

select count(*)
from t t1
join t t2
on t1.b = t2.b
where t1.a != t2.a
-- returns 61398, therefore B->A doesn't hold

select count(*)
from t t1
join t t2
on t1.b = t2.b
where t1.c != t2.c
-- returns 48032, therefore B->C doesn't hold

select count(*)
from t t1
join t t2
on t1.b = t2.b
where t1.d != t2.d
-- returns 55170, therefore B->D doesn't hold
```

```
select count(*)
from t t1
join t t2
on t1.c = t2.c
where t1.a != t2.a
-- returns 14700, therefore C->A doesn't hold

select count(*)
from t t1
join t t2
on t1.c = t2.c
where t1.b != t2.b
-- returns 0, therefore C->B holds

select count(*)
from t t1
join t t2
on t1.c = t2.c
where t1.d != t2.d
-- returns 13208, therefore C->D doesn't hold

select count(*)
from t t1
join t t2
on t1.d = t2.d
where t1.a != t2.a
-- returns 17906, therefore D->A doesn't hold

select count(*)
from t t1
join t t2
on t1.d = t2.d
where t1.b != t2.b
-- returns 14964, therefore D->B doesn't hold

select count(*)
from t t1
join t t2
on t1.d = t2.d
where t1.c != t2.c
-- returns 21034, therefore D->C doesn't hold
```

b)

```
create table r1(
    A varchar(50),
    D varchar (2),
    primary key (A)
);

create table r2(
    C varchar(5),
    B varchar (10),
    primary key (C)
);

create table r3(
    A varchar(50),
    C varchar (5),
    primary key (A,C),
    foreign key (A) references r1(A),
    foreign key (C) references r2(C)
);
```

c)

```
CREATE TABLE if not exists t (
    A VARCHAR ( 50 ) NOT NULL,
    B VARCHAR ( 10 )  NOT NULL,
    C varchar (5) NOT NULL,
    D varchar (2) NOT NULL
);

--\copy t from 'C:\Users\lebro\Desktop\Written Assignment 1\hw1-data.csv' csv header;


insert into r1(A,D)
select distinct A,D
from t;

insert into r2(C,B)
select distinct C,B
from t;

insert into r3(A,C)
select distinct A,C
from t;

drop table t;
```