

Lecture 20: Functional Dependencies and Normalisation

An Example

An example to motivate and introduce this new topic.

- A database schema that is clearly WRONG
- What's wrong with it
- How to fix it

The next lecture will cover the new ideas more formally.

This section of the course is covered in [E&N Chapter 10]
(and Chapter 11 to some extent)

Announcements

- Lab 4 will *not* be next week, but the following one 22-26 September.

Announcements

- Lab 4 will *not* be next week, but the following one 22-26 September.
- There *will* be a lecture next Wednesday.

Announcements

- Lab 4 will *not* be next week, but the following one 22-26 September.
- There *will* be a lecture next Wednesday.
- We are aiming to get assignments and mid-semester exams marked before the break.

What are Normal Forms All About?

- Normal forms are a measure of how well a database schema fits its subject

What are Normal Forms All About?

- Normal forms are a measure of how well a database schema fits its subject
- The main idea used to define them is *functional dependencies*

What are Normal Forms All About?

- Normal forms are a measure of how well a database schema fits its subject
- The main idea used to define them is *functional dependencies*
- There are algorithms to take a schema in one normal form, and create a new one in the next higher normal form

What are Normal Forms All About?

- Normal forms are a measure of how well a database schema fits its subject
- The main idea used to define them is *functional dependencies*
- There are algorithms to take a schema in one normal form, and create a new one in the next higher normal form
- *bottom up* design method: start with attributes and functional dependencies, produce relation schemas

Example of a BAD Database

This example is from C. J. Date *An Introduction to Database Systems* Fifth Edition, Addison-Wesley 1990, §21.3

- The relation records information about shipments of parts from their suppliers.

```
FIRST (  
    SUPPLIER : string,  
    STATUS : integer,  
    CITY : string,  
    PART : string,  
    QUANTITY : integer  
)
```

Example of a BAD Database

This example is from C. J. Date *An Introduction to Database Systems* Fifth Edition, Addison-Wesley 1990, §21.3

- The relation records information about shipments of parts from their suppliers.
- **primary key is SUPPLIER, PART**

```
FIRST (  
    SUPPLIER : string,  
    STATUS : integer,  
    CITY : string,  
    PART : string,  
    QUANTITY : integer  
)
```

Example of a BAD Database

This example is from C. J. Date *An Introduction to Database Systems* Fifth Edition, Addison-Wesley 1990, §21.3

- The relation records information about shipments of parts from their suppliers.
- primary key is `SUPPLIER, PART`
- (we are only interested in the most recent shipment of each part from each supplier)

```
FIRST (  
    SUPPLIER : string,  
    STATUS : integer,  
    CITY : string,  
    PART : string,  
    QUANTITY : integer  
)
```

Example - Functional Dependencies

- some sets of attributes determine other sets of attributes

SUPPLIER \longrightarrow *CITY*

SUPPLIER \longrightarrow *STATUS*

CITY \longrightarrow *STATUS*

SUPPLIER, PART \longrightarrow *QUANTITY*

Example - Functional Dependencies

- some sets of attributes determine other sets of attributes
- eg, a SUPPLIER can only be based in one CITY

SUPPLIER \longrightarrow *CITY*

SUPPLIER \longrightarrow *STATUS*

CITY \longrightarrow *STATUS*

SUPPLIER, PART \longrightarrow *QUANTITY*

Example - Functional Dependencies

- some sets of attributes determine other sets of attributes
- eg, a SUPPLIER can only be based in one CITY
- if 2 tuples had the same value for SUPPLIER, but different values for CITY, we have inconsistent information

SUPPLIER \longrightarrow *CITY*

SUPPLIER \longrightarrow *STATUS*

CITY \longrightarrow *STATUS*

SUPPLIER, PART \longrightarrow *QUANTITY*

Example - Functional Dependencies

- some sets of attributes determine other sets of attributes
- eg, a SUPPLIER can only be based in one CITY
- if 2 tuples had the same value for SUPPLIER, but different values for CITY, we have inconsistent information
- this is possible because the database allows the same fact to be stored in different places

SUPPLIER \longrightarrow *CITY*

SUPPLIER \longrightarrow *STATUS*

CITY \longrightarrow *STATUS*

SUPPLIER, PART \longrightarrow *QUANTITY*

Example - Functional Dependencies

- some sets of attributes determine other sets of attributes
- eg, a SUPPLIER can only be based in one CITY
- if 2 tuples had the same value for SUPPLIER, but different values for CITY, we have inconsistent information
- this is possible because the database allows the same fact to be stored in different places
- the goal is: *one fact, one place*

SUPPLIER \longrightarrow *CITY*

SUPPLIER \longrightarrow *STATUS*

CITY \longrightarrow *STATUS*

SUPPLIER, PART \longrightarrow *QUANTITY*

Example - Data

SUPPLIER	STATUS	CITY	PART	QUANTITY
s1	20	London	p1	300
s1	20	London	p2	200
s1	20	London	p3	400
s1	20	London	p4	200
s1	20	London	p5	100
s1	20	London	p6	100
s2	10	Paris	p1	300
s2	10	Paris	p2	400
s3	10	Paris	p2	200
s4	20	London	p2	200
s4	20	London	p4	300
s4	20	London	p5	400

Observations: Redundancy and Inconsistency

- there is obviously lots of redundancy here

Observations: Redundancy and Inconsistency

- there is obviously lots of redundancy here
- and therefore potential inconsistency

Observations: Redundancy and Inconsistency

- there is obviously lots of redundancy here
- and therefore potential inconsistency
- (schema is *too expressive?*)

Observations: Redundancy and Inconsistency

- there is obviously lots of redundancy here
- and therefore potential inconsistency
- (schema is *too* expressive?)
- we can insert a tuple that says *s1* is in Paris (so long as it has a different PART)

Observations: Redundancy and Inconsistency

- there is obviously lots of redundancy here
- and therefore potential inconsistency
- (schema is *too* expressive?)
- we can insert a tuple that says s1 is in Paris (so long as it has a different PART)
- **which violates the functional dependency**
SUPPLIER \longrightarrow *CITY*

Observations: Redundancy and Inconsistency

- there is obviously lots of redundancy here
- and therefore potential inconsistency
- (schema is *too* expressive?)
- we can insert a tuple that says s1 is in Paris (so long as it has a different PART)
- which violates the functional dependency $SUPPLIER \longrightarrow CITY$

Observations: Redundancy and Inconsistency

- there is obviously lots of redundancy here
- and therefore potential inconsistency
- (schema is *too expressive*?)
- we can insert a tuple that says s1 is in Paris (so long as it has a different PART)
- which violates the functional dependency $SUPPLIER \longrightarrow CITY$

SUPPLIER	STATUS	CITY	PART	QUANTITY
s1	20	London	p1	300
s1	20	London	p2	200
s1	11	Paris	p42	12

Observations: Inexpressive

There is also a *lack of expressiveness* from this schema

- we can not record where a supplier is located unless we have an order with them

Observations: Inexpressive

There is also a *lack of expressiveness* from this schema

- we can not record where a supplier is located unless we have an order with them
- (entity integrity forbids PART=NULL)

Observations: Inexpressive

There is also a *lack of expressiveness* from this schema

- we can not record where a supplier is located unless we have an order with them
- (entity integrity forbids PART=NULL)
- we can not delete the last shipment for a supplier without forgetting where they are

Observations: Inexpressive

There is also a *lack of expressiveness* from this schema

- we can not record where a supplier is located unless we have an order with them
- (entity integrity forbids PART=NULL)
- we can not delete the last shipment for a supplier without forgetting where they are
- **must be careful with updates**

Observations: Inexpressive

There is also a *lack of expressiveness* from this schema

- we can not record where a supplier is located unless we have an order with them
- (entity integrity forbids PART=NULL)
- we can not delete the last shipment for a supplier without forgetting where they are
- must be careful with updates
- eg, if a supplier moves, we must update every tuple that records their location

What is the Problem?

Each tuple seems to be doing two different jobs

- recording facts about shipments

What is the Problem?

Each tuple seems to be doing two different jobs

- recording facts about shipments
- recording facts about suppliers

What is the Problem?

Each tuple seems to be doing two different jobs

- recording facts about shipments
- recording facts about suppliers

What is the Problem?

Each tuple seems to be doing two different jobs

- recording facts about shipments
- recording facts about suppliers

What we want is relations where

- each tuple represents exactly one fact

What is the Problem?

Each tuple seems to be doing two different jobs

- recording facts about shipments
- recording facts about suppliers

What we want is relations where

- each tuple represents exactly one fact
- the fact is *about* the thing identified by the key

What is the Problem?

Each tuple seems to be doing two different jobs

- recording facts about shipments
- recording facts about suppliers

What we want is relations where

- each tuple represents exactly one fact
- the fact is *about* the thing identified by the key
- the remainder of the attributes are the details of that fact

What is the Problem?

Each tuple seems to be doing two different jobs

- recording facts about shipments
- recording facts about suppliers

What we want is relations where

- each tuple represents exactly one fact
- the fact is *about* the thing identified by the key
- the remainder of the attributes are the details of that fact

What is the Problem?

Each tuple seems to be doing two different jobs

- recording facts about shipments
- recording facts about suppliers

What we want is relations where

- each tuple represents exactly one fact
- the fact is *about* the thing identified by the key
- the remainder of the attributes are the details of that fact

SUPPLIER	STATUS	CITY	PART	QUANTITY
s1	20	London	p1	300
s1	20	London	p2	200

Solution? Split it!

	<u>SUPPLIER</u>	STATUS	CITY
facts about suppliers	s1	20	London
	s2	10	Paris
	⋮	⋮	⋮
	s5	30	Athens

Solution? Split it!

	<u>SUPPLIER</u>	STATUS	CITY
facts about suppliers	s1	20	London
	s2	10	Paris
	⋮	⋮	⋮
	s5	30	Athens

	<u>SUPPLIER</u>	<u>PART</u>	QUANTITY
facts about shipments	s1	p1	300
	s1	p2	200
	⋮	⋮	⋮
	s4	p5	400

Solution? Split it!

	<u>SUPPLIER</u>	STATUS	CITY
facts about suppliers	s1	20	London
	s2	10	Paris
	⋮	⋮	⋮
	s5	30	Athens

	<u>SUPPLIER</u>	<u>PART</u>	QUANTITY
facts about shipments	s1	p1	300
	s1	p2	200
	⋮	⋮	⋮
	s4	p5	400

Note: *The original table can be recovered by joining these two.*