NORMALISATION

(Relational Database Schema Design Revisited)

Designing an ER Diagram is fairly intuitive, and faithfully following the steps to map an ER diagram to tables may not always result in the 'best' table design.

- This is a 'top-down' approach

In this lecture we discuss some design criteria, and introduce the concept of **normalisation**, which is the process of transforming a collection of attributes into relations which satisfy a collection of rules.

- This is a 'bottom-up' approach

In constructing a relational database there is a considerable freedom which ranges from:

- putting all of the data in one relation
- having lots of small relations

How do we decide which makes the best set of relations for our purpose?

Some Guidelines

Be guided by the semantics or meaning of the data

- put together in one relation data that belongs together (e.g. using the ER approach)

Avoid having too many null values

- if only 10% of employees have offices we would prefer to set up a separate relation pairing employees and offices than add a column to employee

Beware of keeping multiple versions of information

Avoid breaking up relations in such a way that spurious information is created

Preserve known **functional dependencies**

Ill Effects of Multiple Copies of Data

ni#	name	bDate	dNum	dName	mgrni#
666	Jim Grey	01/12/67	5	Admin	666
777	Jo White	04/06/53	5	Admin	666
888	Sue Low	12/09/88	6	Finance	888
999	Tim Lee	30/07/78	7	HR	999
898	Mary Law	22/02/60	7	HR	999

Insertion. We cannot add a new department which has no employees as yet. Entering a new employee in department 5 means repeating existing departmental data

Deletion. Deleting Sue Low loses details about department 6

Modification. If the manager of department 5 is changed, it must be changed in all tuples

It's Easy to Create False Data

Let's try to split up the data about employees and projects into two tables

ni#	name	pName	pLoc
666	Jim Grey	Jupiter	Perth
666	Jim Grey	Saturn	Dundee
888	Sue Low	Mars	Perth

ni#	name	pName
666	Jim Grey	Jupiter
666	Jim Grey	Saturn
888	Sue Low	Mars

ni#	pLoc
666	Perth
666	Dundee
888	Perth

This loses the info that Jim works on Project Jupiter AT Perth

ni#	name	pName	pLoc
666	Jim Grey	Jupiter	Perth
666	Jim Grey	Jupiter	Dundee
etc	etc	etc	etc

Joining them back together, we get NEW TUPLES!
Jim works on Jupiter and works at Perth

Preserving Functional Dependencies

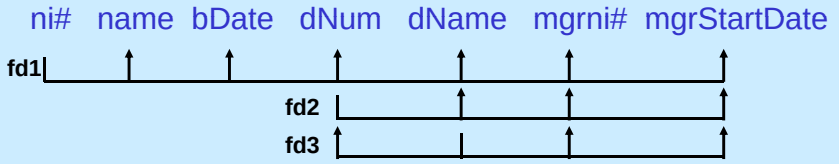
Functional Dependencies are **known** connections between items in the database which can be installed into the database by good schema design

- Given ni# , we can determine the employee's name, date of birth, salary and department
  - i.e. columns name, bDate, salary, department all **depend on** ni#
- Also, from knowing the ni#, we know the department number, and therefore we can find out all about the other department information as well

We will proceed to use functional dependencies to guide our database design

Some Examples of Functional Dependency

dName, mgrni#, mrgrStartDate all depend on dNum
dNum, mgrni#, mrgrStartDate all depend on dName



Another example is that hours depends on ni#, pNum

Defining Functional Dependence

Key Slide

An attribute, X, of a relation is functionally dependent on attributes A, B - N if the same values of R_A..... R_N always lead to the same value of R_X
We write R_A..... $R_N \rightarrow R_X$.

e.g. ni# → name, bDate, dNum
ni#, pNum → hours

This is a property of the **meaning** of the data, not a property that emerges from the current values of the data

i.e. if you happened to have no two employees with the same name, you may indeed be able to infer birth date from name for this data set, but this would not constitute genuine dependence because next week you might employ someone with the same name as a current staff member

Full Functional Dependency

Key Slide

X **fully depends** on A . . . N if it is not dependent on any subset of A . . . N.

Example for Employees :
bDate is dependent on ni# and name but only fully dependent on ni#

However
WorksOn : hours is fully dependent on ni# and pNum

Normalisation

Normalisation is the process of taking a set of relations and decomposing them into more relations which satisfy our criteria better

The decomposition is essentially a series of projections so that the original data can be reconstituted using joins

The form of the relations which satisfy our conditions is called a **normal form**

There are a number of these of increasing stringency

First Normal Form

Key Slide

A relation is in First Normal Form (1NF) if all values are **atomic**

- i.e. all the cells hold single values.

Specifically the following are not permitted

- Multi-valued attributes / sets
  e.g. An attribute locations which is intended to hold more than one location per department cannot be held in a single cell
- Composite attributes / nested relations
  e.g. An attribute name which is divided up into hold a person's title, surname and forenames cannot be held in a single cell

Universal Relation

Key Slide

We start by creating the **Universal Relation** which is one giant relation schema including every database attribute (using unique names)

It must be in first normal form (or it can't go in the database at all!)

A (usually composite) primary key can be chosen which uniquely identifies each row.

The following universal relation is for a subset of the Employee database consisting of Employee, Department and Project only.

Composite attributes have been broken down, multi-valued attributes are shown as one attribute. *What is the primary key?*

ni# , lastname, firstnames, street, city, postcode, salary, bDate, gender, dNum, dName, mgrni# , mgrStartDate, pNum, pName, pLocation, hours

Candidate Keys

A **candidate key** is a set of attributes which uniquely identifies each entry in the relation, and therefore can be the primary key of that relation

A relation can have more than one candidate key
(e.g. dNum and dName),
in which case one (usually the simplest) is chosen to be the primary key

Functional Dependencies for the Company Database

ni# -> lastname, firstnames, street, city, postcode, salary, bDate, gender, dNum, dName, mgrni# , mgrStartDate

pNum -> pName, pLocation

dNum -> dName, mgrni# , mgrStartDate

ni#, pNum -> hours

Note that everything is directly or indirectly dependent on ni# and pNum so these will make an effective key of the Universal Relation

Key Slide

Second Normal Form

By definition, every other attribute is functionally dependent on the primary key

The relation is in **Second Normal Form** (2NF), if it is in First Normal Form AND all the non-primary-key attributes are **fully** functionally dependent on the chosen primary key, and also on any other candidate key

- Clearly, any relation with a single primary key will be 2NF

If there are two primary key attributes, A & B, then each of the other attributes is

either (i) dependent on A alone
(ii) dependent on B alone
or (iii) dependent on both

- For a relation to be in 2NF : only the third option is allowed
- 2NF normalisation consists of creating a separate relation for each of the three cases

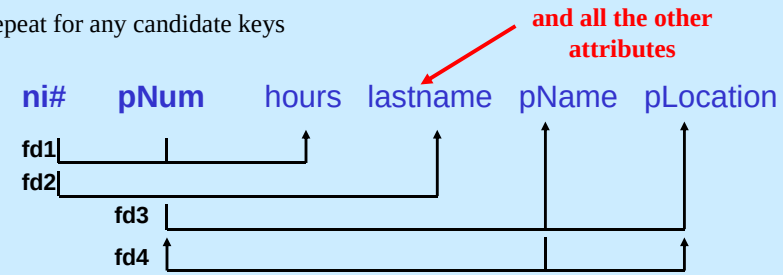
Making our Universal Relation 2NF

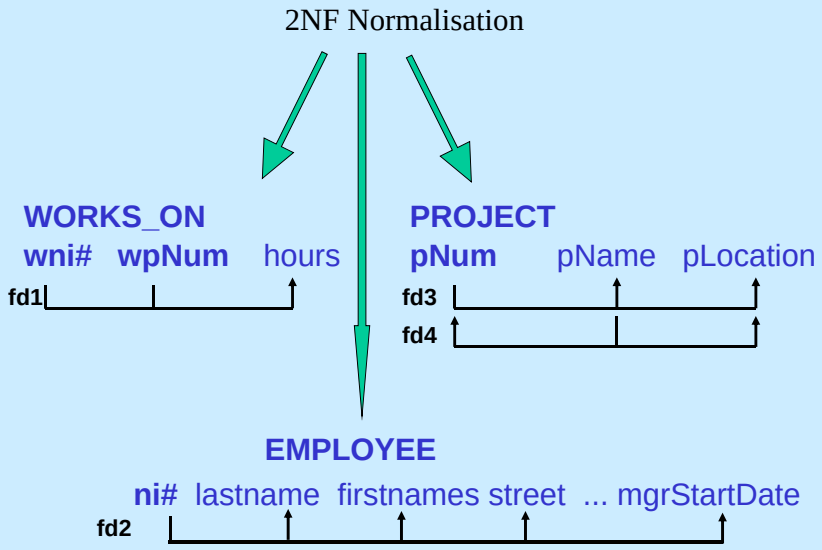
This uses a subset of the attributes. The national insurance number together with the project name is a candidate key

Consider each combination of the composite primary key attributes

Which other attributes depend on just a subset of them?

Repeat for any candidate keys





Third Normal Form

Key Slide

Third Normal Form eliminates **transitive dependencies** - i.e. those dependencies which hold only because of some intermediary attribute

An attribute is **transitively dependent** on the primary key if there is some other attribute which it is dependent on and which is, in turn, dependent on the key

ni# --> dNum --> mgrni#

Here mgrni# is dependent of ni# as required by 2NF, but only because it is dependent on dNum which is in turn dependent on ni#

- Non-3NF-relations are likely to hold redundant information

A relation is in **Third Normal Form** if it is in second normal form AND no non-prime-key attribute is transitively dependent on the primary key.

i.e. for any pair of attribute A & B such that $R_A \rightarrow R_B$, there is no attribute such that $R_A \rightarrow R_X$ and $R_X \rightarrow R_A$

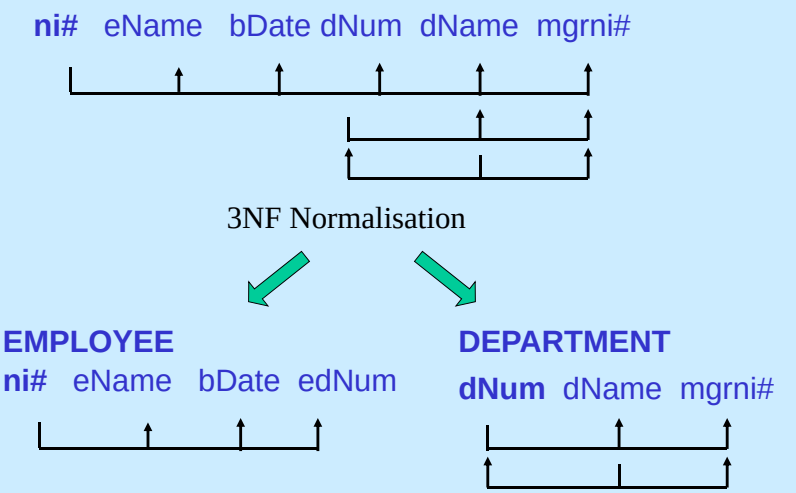
Employee/Dept Relation : turn to 3NF

Again using a subset of the Employee attributes

Look for transitive dependencies

ni# --> dNum --> dName, mgrni#
ni# --> dName --> dNum, mgrni#

We choose the primary key of the 3NF relation to be *dNum*



LOTS : Introduction

In this example, the database holds details of a number of real estate properties

The Universal Relations holds

propID county lot# area price taxRate

A **property** is identified by the **county** and the **lot number**

It also has a unique **property ID** number

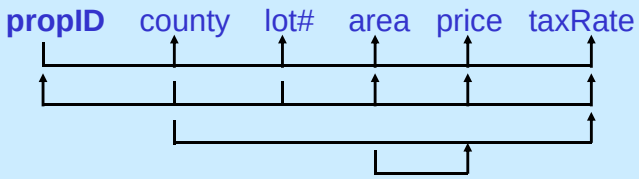
The **tax rate** is fixed by **county**
regardless of the area (size)

The **price** is dependent on the **area**
regardless of the county.

PropertyID is chosen to be the primary key

County and Lot Number are a composite candidate key, but they were not chosen to be the primary key

Here are the functional dependencies

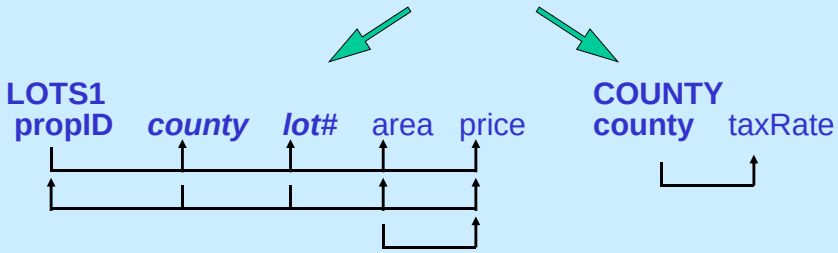


Consider each combination of the composite primary key attributes.

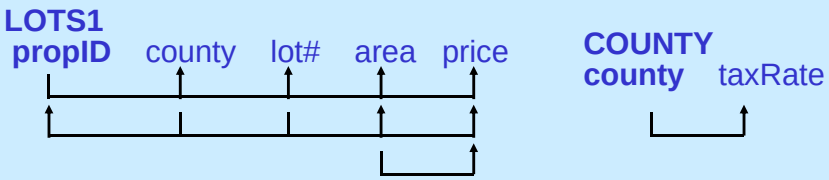
Which other attributes depend on just a subset of them?

Repeat for any candidate keys

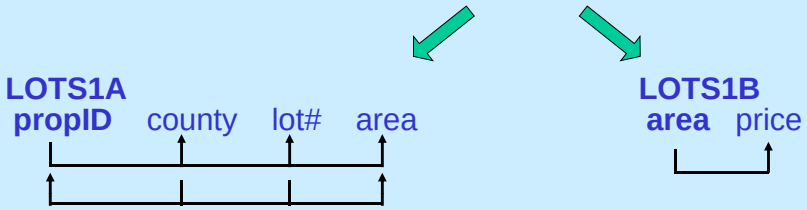
2NF Normalisation



Look for transitive dependencies, e.g. propID -> area -> price



3NF Normalisation



Boyce-Codd Normal Form

Key Slide

Even a 3NF database can have redundancy so there is a further strengthening called **Boyce Codd Normal Form**

Imagine that in the LOTS example different counties have different areas

For instance areas in Clackmannan are all small
while areas in Lanarkshire are all large

so we have the dependency: area -> county which means that counties are unnecessarily duplicated when areas are duplicated

BCNF removes this by insisting that :

if we have a dependency A -> B then A must be a candidate key (i.e. a set of attributes which could have been used as the primary key)

Area is not a candidate key, so we exclude this possibility by partitioning LOTS1A into

LOTS1AX
propID lot# area

LOTS1AY
area county

A Final Example : Medical Records

Patients have a unique NI#, each having a name and a GP;

Each GP has an id number, name & address;

A hospital appointment connects a patient, a date, a hospital and a consultant;

A consultant has a phone number and visits one hospital on any given day;

A hospital has an address.

The following example assumes a patient will only have one appointment on a particular day, and that consultant names are unique.

The Universal Relation (choose primary key)

U (**ni#, appDate**, appTime, patName, gp#, gpName, gpAddress, cName, cPhone, hosp,hospAdd)

Functional Dependencies

From a patient's national insurance number, we can determine the patient's name and GP

ni# → patName, gp# , gpAddress, gpName

From a GP's ID we determine the GP's name and address:

gp# → gpAddress, gpName

Each hospital has only one address:

hosp → hospAddress

From a particular patient on a particular day we can determine the consultant information, the time of the appointment and which was the hospital:

ni#, appDate → cName, cPhone, appTime, hosp, hospAdd

Given the consultant, appointment date and time, we can determine the patient:

cName, appDate, appTime → ni#

Each consultant is at one hospital on any day:

cName,appDate → hosp, hospAddress

Each consultant has one phone number

cName → cPhone

Normalising the Example

Starting with the Universal Relation we find all kinds of redundancy.

U (**ni#, appDate**, appTime, patName, gp#, gpName, gpAddress, cName, cPhone, hosp,hospAdd)

So moving to 2NF, split off those attributes only dependent on part of the primary key:

Patient (**ni#** , patName, gp# , gpAddress, gpName)

Appt (**ni# , appDate**, appTime, cName, cPhone, hosp, hospAddress)

Patient (**ni#** , patName, gp# , gpAddress, gpName)

Patient is 2NF but not 3NF since there is a transitive dependency

ni# → gp# → gpAdd, gpName

So split off the GP information

Pat (**ni#** , patName, gp#)
GP (**gp#** , gpAdd, gpName)

Appt is also 2NF but not 3NF since there is are 2 transitive dependencies

ni#, appDate → cName → cPhone
ni#, appDate → hosp → hospAdd

Appt (**ni# , appDate**, appTime, cName, cPhone, hosp, hospAddress)

Appt becomes

App (**ni# , appDate**, apptime, cName, hosp)
Con (**cName**, cPhone)
Hospital (**hosp**, hospAdd)

So far we have 3NF but not BCNF, because in the App table
App (**ni# , appDate**, apptime, cName, hosp)
there is the functional dependency

appDate, cName --> hosp

yet appDate, cName is not a candidate key

Therefore we could go to

Appointment (**ni# , appDate**, appTime, cName)
ConsHosp (**appDate, cName**, hosp)

Final Tables

Pat (**ni#** , patName, gp#)
GP (**gp#** , gpAdd, gpName)
Con (**cName**, cPhone)
Hospital (**hosp**, hospAdd)
Appointment (**ni# , appDate**, appTime, cName)
ConsHosp (**appDate, cName**, hosp)