COM106: Introduction to Databases

Normalisation

Normalisation

See Chapter 14 of Connolly & Beggs (6th ed)
Or similar chapters in other textbooks.

Database Design Process – A Recap

Process	Tasks Involved	End Result
Data Investigation	Identification of nature and use of data	Outline Data Dictionary
Data Modelling	***Shaping real world facts into real world concepts***	***Generalised Conceptual Model***
Database Conceptual Design	***Mapping conceptual model to specific data model***	***DBMS-specific and Conceptual External Schema***
Database Implementation	Structuring the actual data in the physical database	Internal Schema and Database
Database Security, Monitoring and Tuning	Determining and implementing security measures. Monitoring database usage and Restructuring/Reorganising Database	Optimised and Secure Database

To produce a single **conceptual model** of the system we use:

Entity Relationship Modelling (based on the results of a data investigation)
a ***top down*** approach (generalisation of entities/relationships)

Normalisation
a ***bottom up*** approach (linked to relational data model)

To yield a **Relational Schema**, ready for implementation in a Relational DBMS (**RDBMS**).

Normalisation

Normalisation can be used either:

- as a bottom-up stand-alone database design technique, or
- as a **validation** of the relational schema produced as the result of an ER Modelling process

The second approach (validation) is normally used in the design of large database systems:

1. Create an **ER Diagram** based on the results of a data investigation (**ER Modelling**)
2. Using the mapping (translation rules) to create a **relational schema** (a set of linked tables)
3. **Normalisation** - checking that each table is correctly designed (i.e. in **3rd Normal Form**) which may involve decomposition (splitting into more tables).

What is Normalisation?

Normalisation is a technique for producing a set of relations with **desirable properties**, given the data requirements of an enterprise. *(from Connelly & Begg)*

Desirable properties include:

- The **minimal number of attributes** necessary
- Attributes with a close logical relationship are found in the same table (**functional dependency**)
- **minimal redundancy** – each attribute appears only once (except for foreign keys)

Normalisation

To understand why these properties are desirable (or **essential?**) we will look at normalisation as a bottom-up **data analysis technique**.

Starting from a position where data is held in one table (UnNormalised Form – **UNF**), normalisation transforms the data into First Normal Form (**1NF**), Second Normal Form (**2NF**) and Third Normal Form (**3NF**).

For implementation, a database should be at least in **3NF**. There are higher order normal forms, but these are outside the scope of the module.

An Example

Consider a simple database to hold the newspapers read by various people:

Pat reads the Times and the Sun, Jenny reads the Express, etc

A first attempt might be to hold the data in a single table:

name	paper_list
Pat	Times, Sun
Jenny	Express

This is not ideal – each entry in the ***paper_list*** column holds an unspecified number of papers – **Repeating Groups**.

A table containing one or more repeating groups is said to be in Unnormalised Form (**UNF**)

RDBMS can't cope with this. Each entry in a table must contain *a* **single data item**.

To transform a table from **UNF** to First Normal Form (**1NF**), **repeating groups must be removed**. There cannot be multiple values in any column.

Normalisation

One approach to getting rid of repeating groups would be to extend the table rows by replicating the non-repeated columns for each repeated item value:

name	paper_list
Pat	Times
Pat	Sun
Jenny	Express

The table clearly contains the same information and is now in First Normal Form (**1NF**).

A relation is in 1NF if no entry consists of more than one value

(i.e. does not have repeating groups)

An alternative two-table approach to removing repeating groups would be to split the repeating and non-repeating data into separate tables (PAPER and NAME)

choose a primary key for the repeating data table (PAPER)

and insert this as a foreign key in the non-repeating data table (NAME)

The two-table approach is often better as it takes up less space and leads to Second Normal Form (**2NF**)- more later!

Problems with a Relation in 1NF

Relations that are in 1NF can have considerable problems.

Example: Staff borrowers in a Library
where the staff number functions as a Library number
and details of books borrowed are held in the same table

Normalisation - Problems with a Relation in 1NF

STAFFBORROWER (sno, name, dept, grade, salary, bno, date_out)

sno	name	dept	grade	salary	bno	date_out
1	Smith	Computing	2.7	26813	1	30/06/2016
2	Black	Marketing	1.5	19890	8	08/01/2016

There are no repeating groups – so the relation is in 1NF

Suppose Smith borrows another book:

A new row is needed to ensure the table remains in 1NF

sno	name	dept	grade	salary	bno	date_out
1	Smith	Computing	2.7	26813	1	30/06/2016
2	Black	Marketing	1.5	19890	8	08/01/2016
1	**Smith**	**Computing**	**2.7**	**26813**	53	18/07/2016

Smith's details are **duplicated**.

This duplication contravenes one of the main aims of relational database design – to group attributes into relations to **minimise data redundancy**.

Relations that have redundant data may suffer from **update anomalies**, classified as **insertion**, **deletion** or **modification anomalies**

Normalisation - Problems with a Relation in 1NF

sno	name	dept	grade	salary	bno	date_out
1	Smith	Computing	2.7	26813	1	30/06/2016
2	Black	Marketing	1.5	19890	8	08/01/2016
1	**Smith**	**Computing**	**2.7**	**26813**	53	18/07/2016

Modification Anomalies

Suppose that Smith goes on to a new grade.

changes are required to all records for Smith - with the danger that one is missed

Suppose that the salary for grade 2.7 is changed.

all records for all staff members on grade 2.7 would have to be changed.

In general, a fact (a piece of data) should be stored only **once**:

Storage space (and thus cost) is reduced

Updates to data are achieved with a minimal number of operations – reducing time, costs and the opportunity for data inconsistencies.

Insertion Anomalies

Suppose that a new scale point is created (2.8 earns £27,491) and as yet there is no-one on that scale point.

How should this be recorded?

Normalisation - Problems with a Relation in 1NF

The following violates entity integrity – the primary key cannot be NULL

sno	name	dept	grade	salary	bno	date_out
Null	Null	Null	2.8	27491	Null	Null

Deletion Anomalies

Suppose that Smith returns all her books

Do we delete all the corresponding rows - **removing all trace of Smith**?

Or do we remove the *bno* and *date_out* entries from all the rows for Smith?

But this **leaves duplicated information** about Smith and in this case also **removes part of the composite primary key**.

Solution to Update Anomalies

The solution to these anomalies is to carry out a **non-loss decomposition** (**NLD**)

replace the relation by **projections** from which the original relation can be re-created (by joining)

The re-creation must give no less and no more than the original table

From the original **STAFFBORROWER** table create two new table **STAFF** and **LOAN**

STAFF (sno, name, dept, grade, salary) **LOAN** (bno, date-out, sno*)

Normalisation - Problems with a Relation in 1NF

- Data redundancy is removed (except for the PK/FK - essential for joining the tables)
- No data is lost – the original table can be recreated by joining
- Because duplicated data has been removed, the new tables will take less storage space
- Some (but not all) Insertion, deletion and modification **anomalies has been resolved**
- Attributes in each new table have a close logical relationship. This was identified earlier as one of the desirable properties of a relational schema (**functional dependency**)

A Worked Example – Pet Health Record

A vets has a pet health history report that they store on index cards. On each record card they record the pets id, name, type, age and owners name. They also record the date of each visit, the procedure received and the procedure code.

PET ID	PET NAME	PET TYPE	PET AGE	OWNER	VISIT DATE	PROCEDURE
246	ROVER	DOG	12	SAM COOK	JAN 13/2002	01 - RABIES VACCINATION
					MAR 27/2002	10 - EXAMINE and TREAT WOUND
					APR 02/2002	05 - HEART WORM TEST
298	SPOT	DOG	2	TERRY KIM	JAN 21/2002	08 - TETANUS VACCINATION
					MAR 10/2002	05 - HEART WORM TEST
341	MORRIS	CAT	4	SAM COOK	JAN 23/2001	01 - RABIES VACCINATION
					JAN 13/2002	01 - RABIES VACCINATION
519	TWEEDY	BIRD	2	TERRY KIM	APR 30/2002	20 - ANNUAL CHECK UP
					APR 30/2002	12 - EYE WASH

Normalisation - A Worked Example – Pet Health Record

Devise a single table schema to hold this information.

PET (pet_id, pet_name, pet_type, pet_age, owner,
{visitdate, procedure_no, procedure_name})

*where attributes in parenthesis **{visitdate, …..}** represent **repeating groups** - i.e. an individual pet may have multiple visits.*

with primary key - *pet_id, visitdate, procedure_no* (*composite primary key*)

Since there are repeating groups, **PET** is in **UNF**

Now, use **non-loss decomposition** to transform (or **decompose**) the table to **1NF**

Replace **PET** with two new tables:

one containing the attributes from the repeating groups – the details of each visit (**VISIT_DETAILS**)

the other containing the remaining attributes – the details of each pet (**PET_DETAILS**)

with the **PK** of the **PET_DETAILS** table (*pet_id*) – the non-repeating groups table included as a **FK** in the **VISIT_DETAILS** table – the repeating groups table

PET_DETAILS (pet_id, pet_name, pet_type, pet_age, owner)

VISIT_DETAILS (pet_id*, visitdate, procedure_no, procedure_name)

Normalisation - A Worked Example – Pet Health Record

Original table can be recovered, **without loss of data**, by a join on the PK/FK match - *pet_id*

Both tables are in **1NF** as there are **no repeating groups** but ….

VISIT_DETAILS still contains **redundant data**

…. and so, will suffer from **update anomalies**

PET_DETAILS

pet_id	pet_name	pet_type	pet_age	owner
246	ROVER	DOG	12	SAM COOK
298	SPOT	DOG	2	TERRY KIM
341	MORRIS	CAT	4	SAM COOK
519	TWEEDY	BIRD	2	TERRY KIM

VISIT_DETAILS

pet_id	visitdate	procedure_no	procedure_name
246	JAN 13/2002	01	RABIES VACCINATION
246	MAR 27/2002	10	EXAMINE & TREAT WOUND
246	APR 02/2002	05	HEART WORM TEST
298	JAN 21/2002	08	TETANUS VACCINATION
298	MAR 10/2002	05	HEART WORM TEST
341	JAN 23/2001	01	RABIES VACCINATION
341	JAN 13/2002	01	RABIES VACCINATION
519	APR 30/2002	20	ANNUAL CHECK UP
519	APR 30/2002	12	EYE WASH

Normalisation – Functional Dependencies

Functional Dependencies

Before considering higher Normal Forms (2NF, 3NF and higher) we need to understand **Functional Dependencies**

Redundancy in 1NF tables is the root cause of update anomalies.

Functional dependencies can identify redundancy and suggest refinements.

Decomposition is the main refinement technique – using non-loss decomposition to create new tables.

A **functional dependency** (FD) is an **integrity constraint** between two sets of attributes in a relation. It exists when one set of attributes **uniquely determines** another set of attributes.

For example, if R is a relation with attributes **X** and **Y**,

and **Y** can be **uniquely determined** if **X** is known,

then **Y** is said to be **functionally dependant** on **X.** Written **X → Y**

An FD is a statement about **all allowable relations** – identified based on **semantics**, NOT instances

FDs are a generalisation of keys

if K → all attributes of R

then K is a **superkey** for R (does not require K to be minimal)

Read "→" as "determines"
***CAUTION**: The opposite is not true. Knowing Y does **NOT** determine X*

Normalisation – Functional Dependencies

An alternative definition: In a relation R, an attribute **Y** is functionally dependent on attribute **X** (i.e. **X** functionally determines **Y**) if and only if each **X**-value has associated with it exactly one **Y**-value

X is known as a **determinant**. It is a potential **primary key** which eventually may be in its own table with those attributes directly depending on it (**Y**) as non-key attributes.

To identify a functional dependency look for

a determinant (potential primary key)

attributes whose values only change whenever the value of the determinant changes

For example, consider the **TAKES** relation recording university students' exam results:

TAKES (Studentnum, StudentName, ModuleCode, ModuleName, ExamResult)

With composite primary key *Studentnum, ModuleCode*

TAKES

Studentnum	StudentName	ModuleCode	ModuleName	ExamResult
123	Morris	M23	Ethics	62
276	Whyte	M23	Ethics	38
123	Morris	M92	Politics	56
169	Kenny	M92	Politics	47
276	Whyte	M14	Economics	56
169	Kenny	M14	Economics	71
234	Feeney	M14	Economics	48

Identify the functional dependencies in **TAKES**

Normalisation – Functional Dependencies

TAKES (Studentnum, StudentName, ModuleCode, ModuleName, ExamResult)

Ideally, all attributes in the relation should be functionally dependant on the primary key.

In **TAKES**, since each student will have a result for each module they take, it is clear that *ExamResult* is functionally dependant on the composite primary key

i.e., both Studentnum and ModuleCode

So Studentnum, ModuleCode → ExamResult

However, *StudentName* does not change when a student takes another module

i.e., StudentName depends on Studentnum alone (not the composite primary key)

So Studentnum → StudentName ***partial dependency***

And *ModuleName* does not change for different students

i.e., ModuleName depends on *ModuleCode* alone

So ModuleCode → ModuleName ***partial dependency***

In general, if a **non-key attribute** (*i.e., any attribute other than an attribute making up the primary key*) is functionally dependent on only part of a composite primary key, then it is said to be a **partial dependency**

A relation (table) is in Second Normal Form (2NF) if it does not contain partial dependencies

Normalisation – Functional Dependencies

The dependencies in a relation can be shown in a **Dependency Diagram**

- List the attributes in the composite primary key
- Put a box round **individual** attributes of the PK and draw arrows to the associated **partial dependencies** *(on the left)*
- Put a box round **all** attributes of the PK and draw arrows to the associated **full dependencies** *(on the left)*

The Dependency Diagram is used to **justify** how a table should be split

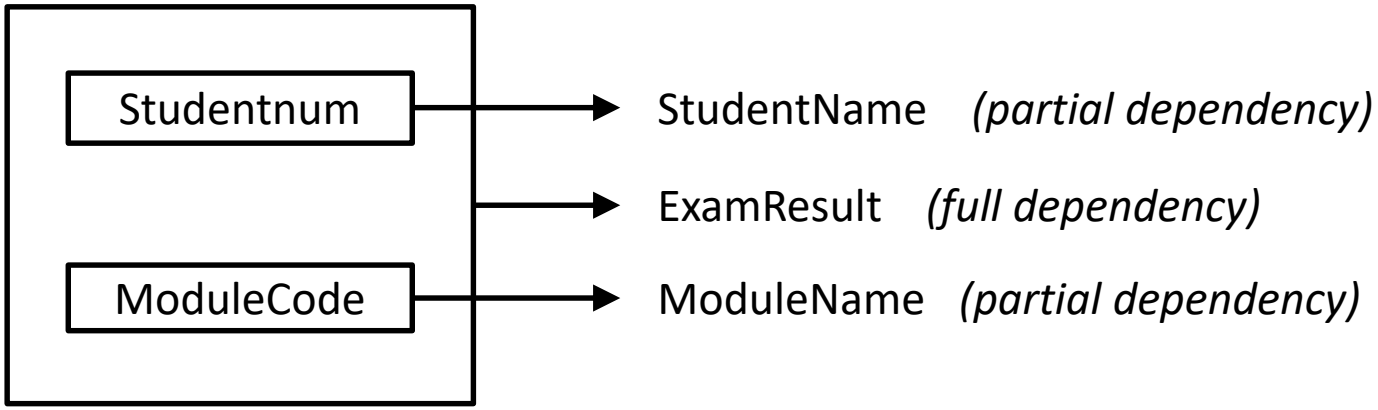
They are **only** created for a table which does not obey a normal form

The decomposition of the tables **MUST** match the dependencies shown

Studentnum, ModuleCode → ExamResult

Studentnum → StudentName

ModuleCode → ModuleName



How should the **TAKES** table be decomposed?

Normalisation – Functional Dependencies

Non-loss decomposition is used to **make each determinant a PK in its own table**

MODULE (ModuleCode , ModuleName)
RESULT (Studentnum* , ModuleCode*, ExamResult)
STUDENT (Studentnum , StudentName)

Note that the only duplication of attributes allowed are the PK/FK matches

ModuleCode	ModuleName
M23	Ethics
M92	Politics
M14	Economics

Studentnum	StudentName
123	JMorris
276	TWhyte
169	MKenny
234	SWhyte

Studentnum	ModuleCode	ExamResult
123	M23	62
276	M23	38
123	M92	56
169	M92	47
276	M14	56
169	M14	71
234	M14	48

The resulting tables can be rejoined (using PK/FK matches) to give **exactly the same data**

```
SELECT  *
FROM STUDENT, RESULT, MODULE
WHERE STUDENT.Studentnum = RESULTS.Studentnum
AND RESULTS.ModuleCode = MODULE.ModuleCode;
```

Normalisation – Functional Dependencies

Definition: A relation is in **second normal form (2NF)** if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key

MODULE (ModuleCode , ModuleName)
RESULT (Studentnum* , ModuleCode*, ExamResult)
STUDENT (Studentnum , StudentName)

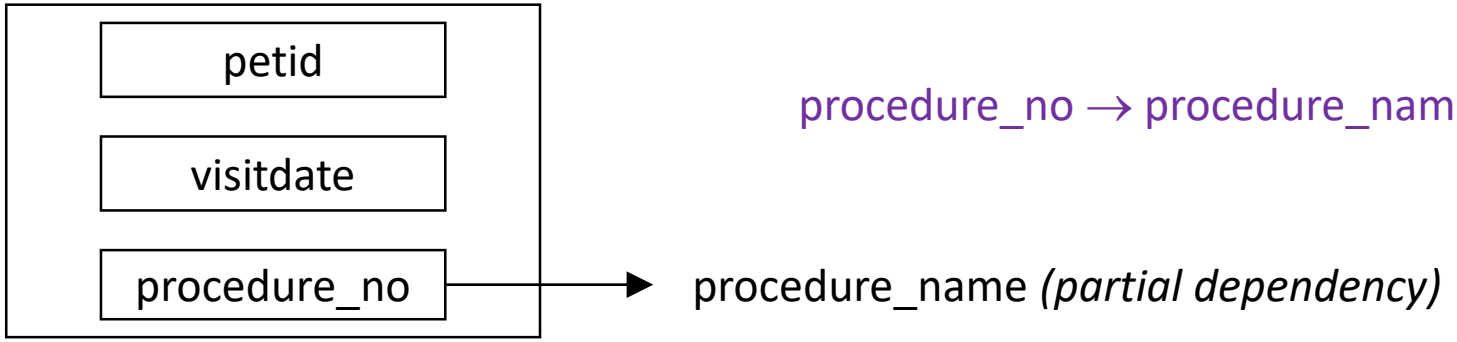
All three relations are in **2NF**

2NF is only relevant if a composite primary key is found (otherwise move straight to 3NF as with ORDER table)

Recall the Pet Health Record Example:

PET_DETAILS (pet_id, pet_name, pet_type, pet_age, owner)
VISIT_DETAILS (pet_id*, visitdate, procedure_no, procedure_name)

- **PET_DETAILS** and **VISIT_DETAILS** are in **1NF** – no repeating groups
- Since **PET_DETAILS** does not have a composite primary key, it is also in **2NF**
- **VISIT_DETAILS** has a composite primary key – need to check for **partial dependencies**



Normalisation – Functional Dependencies

By definition, if a relation is in **2NF** it does not contain any **partial dependencies**.

However, a table in **2NF may** contain another type of functional dependency – **transitive dependency** – which can cause update anomalies of the type seen earlier.

First, some mathematical foundation:

Consider a relation **R** with attributes (or sets of attributes) **X, Y** and **Z**

The **axiom of transitivity** (one of **Armstrong's Axioms** – *more later*) states that

If **X → Y** and **Y → Z** then **X → Z**

If, in addition, **Y not → X** and **Z not → X**

Then the functional dependency **X → Z** is a **transitive dependency**. **Z** is said to be **transitively dependant** on **X**

For example, consider the table **EMP** (empno, ename, salary, deptnum, dname)

With dependencies: **empno → ename, salary, deptnum, dname**

deptnum → dname

Since **empno → deptnum** and **deptnum → dname**, then, by the **axiom of transitivity**,

empno → dname

And, since **deptnum not → empno** and **dname not → empno** then

empno → dname is a **transitive dependency**.

Normalisation – Functional Dependencies

Transitive dependencies can be shown on a Dependency Diagram:

To construct the diagram

PK in a box on the left (***NOT the name of a table!***) with arrows to the dependent attributes on the right

Here, the PK is a single attribute so ideally all **non-key attributes** should be functionally dependant on the primary key

But there is also a functional dependency between two of the non-key attributes

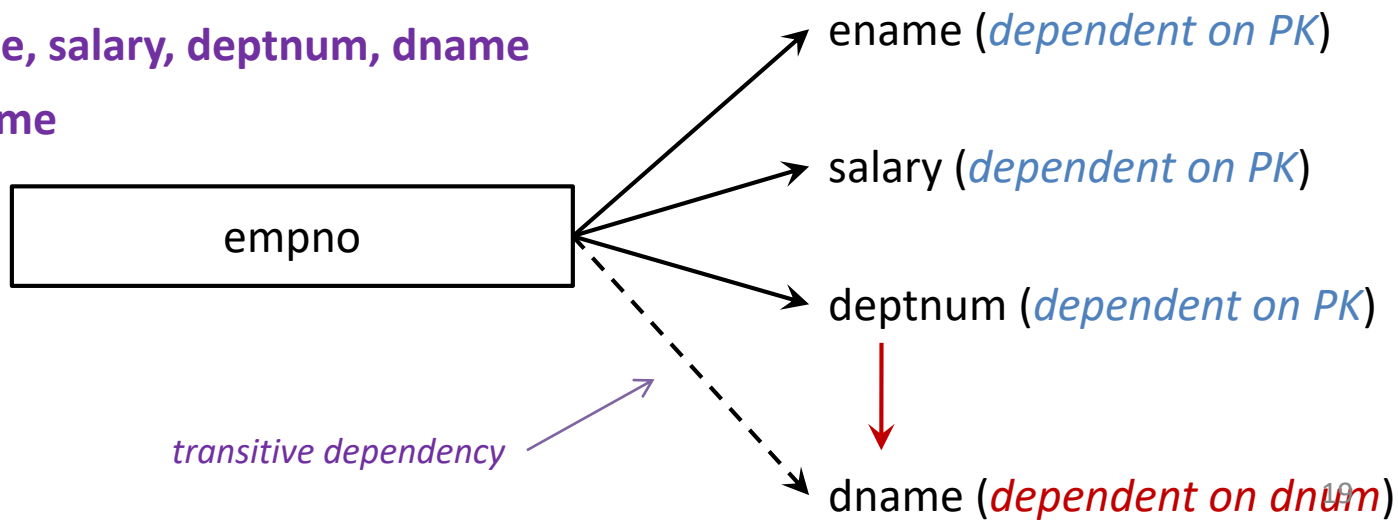
deptnum → dname

leading to the **transitive dependency empno → dname** shown as a dashed arrow.

EMP (<u>empno</u>, ename, salary, deptnum, dname)

empno → ename, salary, deptnum, dname

deptnum → dname



Normalisation – Functional Dependencies

Transitive dependencies are removed through **non-loss decomposition** –

make each determinant a PK in its own table

A relation in which there are no transitive dependencies is in **Third Normal Form** (**3NF**)

A slightly more formal definition- A relation is in **3NF** if:

- it is in **1NF** and in **2NF** and
- every non-key attribute is **non-transitively fully functionally dependant** on the primary key.

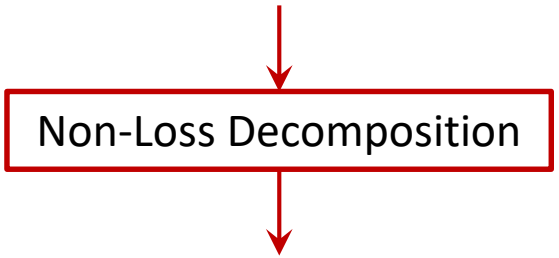
It is always possible to find a decomposition into 3NF.

Note: A table in 2NF with zero or just one non-key attribute is already in 3NF

For example, consider the university students' exam results table (seen earlier):

TAKES (Studentnum, StudentName, ModuleCode, ModuleName, ExamResult)

TAKES is in **1NF**



Studentnum, ModuleCode → ExamResult
Studentnum → StudentName
ModuleCode → ModuleName

MODULE (ModuleCode, ModuleName)
RESULT (Studentnum*, ModuleCode*, ExamResult)
STUDENT (Studentnum, StudentName)

All three relations are in **2NF**

AND in **3NF**

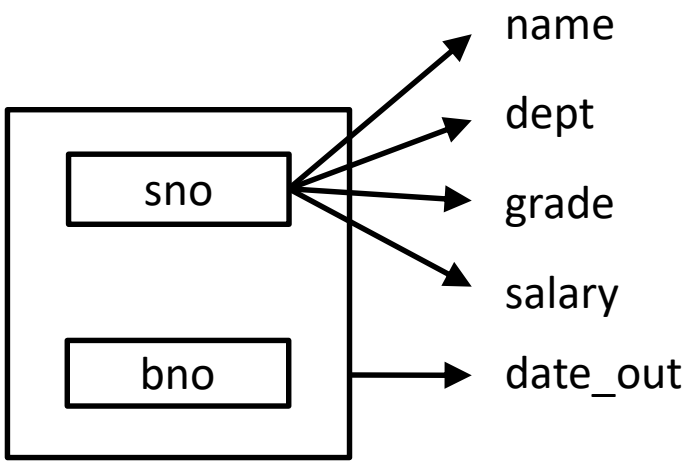
Normalisation – Examples

Consider the STAFFBORROWER example (seen earlier):

STAFFBORROWER (sno, name, dept, grade, salary, bno, date-out)

STAFFBORROWER is in **1NF** (since **no repeating groups**) but, since there is a composite PK *(sno, bno)*, need to check if it is in 2NF

Functional dependencies: **sno → name, dept, grade, salary** **&** **sno, bno → date_out**



STAFFBORROWER contains **partial dependencies**

Only *date_out* is fully dependant on the entire PK

Non-Loss Decomposition gives:

LOAN (sno*, bno, date-out)
STAFF (sno, name, dept, grade, salary)

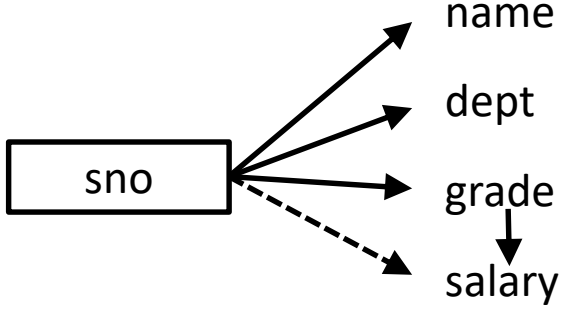
Both relations are in **2NF**

However, if it is the case that everyone on the same grade earns the same salary there is an **FD between two non-key attributes** of **STAFF**

grade → salary

So **STAFF** contains a **transitive dependency**

sno → salary



Normalisation – Examples

Non-Loss Decomposition of the **STAFF** relation gives:

LOAN (sno*, bno, date-out)
STAFF_1 (sno, name, dept, grade*)
GRADE (grade, salary)

All three relations are in **3NF**

Example – Pet Health Record Revisited

Recall that the information on the Pet Health index cards was captured in a single table schema:

PET (pet_id, pet_name, pet_type, pet_age, owner,
{visitdate, procedure_no, procedure_name})

Since there are repeating groups, **PET** is in **UNF**

Non-Loss Decomposition results in:

PET_DETAILS (pet_id, pet_name, pet_type, pet_age, owner)
VISIT_DETAILS (pet_id*, visitdate, procedure_no, procedure_name)

Both relations are in **1NF**

PET_DETAILS is also in **2NF** – since the PK is not composite.

PET_DETAILS is also in **3NF** – since there are no **transitive dependencies** and each **non-key attribute** is **fully functionally dependent** on the PK.

However, **VISIT_DETAILS** has a composite PK, so need to check if it is in 2NF –

is each non-key attribute dependent on the complete PK?

There is a **partial dependency:** procedure_no → procedure_name

Normalisation – Examples

Using Non-Loss Decomposition to remove the partial dependence in **VISIT_DETAILS** gives:

PET_DETAILS (pet_id, pet_name, pet_type, pet_age, owner)
VISIT_DETAILS_1 (pet_id*, visitdate, procedure_no*)
PROCEDURE (procedure_no, procedure_name)

VISIT_DETAILS_1

- is in **2NF** – all non-key attributes (there are none!) are dependent on the complete PK
- Is in **3NF** – there are no non-key attributes so there can be no transitive dependencies.

PROCEDURE

- is in **2NF** – the PK is a single attribute; since the PK is not composite there can be no partial dependencies
- Is in **3NF** – since there is only one non-key attribute, there can be no transitive dependencies.

Hence, all tables in the relational schema are in **3NF** – **update anomalies have been resolved** and there is **normally** no need to decompose further before implementing the database.

Normalisation - Examples

A Worked Example:

A Company holds order information on (paper-based) order forms as shown:

Order No: 1001 **Order Date:** 01/05/2016 **Delivery Date:** 06/05/2016

Customer No: 21 **Customer Name:** J Smith **Phone Number:** 90345122

Part Number	Part Name		Unit Cost	Quantity
101	Cog		2.50	5
134	Block		5.00	3

Where an individual customer places an order for one or more parts.

Devise a relational schema in 3NF to hold the information.

Step 1. Devise a single table to capture the information on the form - **ORDERFORM**

orderno	partno	pname	unitcost	quantity	orderdate	deldate	custno	custname	custphone
1001	101	Cog	2.50	5	01/05/2016	06/05/2016	21	J Smith	90345122
1001	134	Block	5.00	3	01/05/2016	06/05/2016	21	J Smith	90345122
1002	175	Bolt	0.45	200	04/05/2016	11/05/2016	36	P Jones	71452620
1002	115	Widget	0.70	100	04/05/2016	11/05/2016	36	P Jones	71452620
1003	134	Block	5.00	2	12/05/2016	18/05/2016	21	J Smith	90345122
1003	175	Bolt	0.45	100	12/05/2016	18/05/2016	21	J Smith	90345122

Normalisation - Examples

The **ORDERFORM** table contains repeating groups and has the following schema:

ORDERFORM (orderno, orderdate, deldate, custno, custname, custphone, {partno, pname, unitcost, quantity})

UNF

With composite PK *(orderno, partno)*

Step 2. Decompose **ORDERFORM** using Non-Loss Decomposition (**NLD**) to bring the schema to **1NF**

ORDER (orderno, orderdate, deldate, custno, custname, custphone)
ORDEREDPART (orderno*, partno, pname, unitcost, quantity)

Both relations are in **1NF**

The **ORDER** relation is also in **2NF** – since the PK is not composite there can be no partial dependencies.

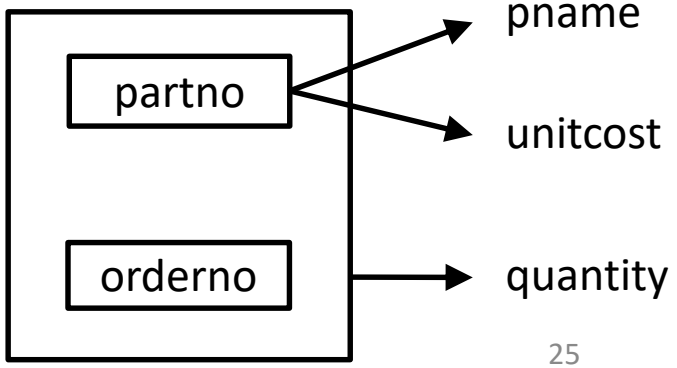
The **ORDEREDPART** relation has a composite PK – so need to check for partial dependencies

Step 3. Check **ORDEREDPART** for partial dependencies

quantity is the only non-key attribute dependent on the complete PK **orderno, partno → quantity**

The remaining non-key attributes (*unitcost* & *pname*) are dependent on *partno* only

partno → pname, unitcost **partial dependency**



Normalisation - Examples

Step 4. Decompose **ORDEREDPART** using **NLD** to bring the schema to **2NF**

PART (partno, pname, unitcost)
ORDEREDPART_1 (orderno*, partno*, quantity)
ORDER (orderno, orderdate, deldate, custno, custname, custphone)

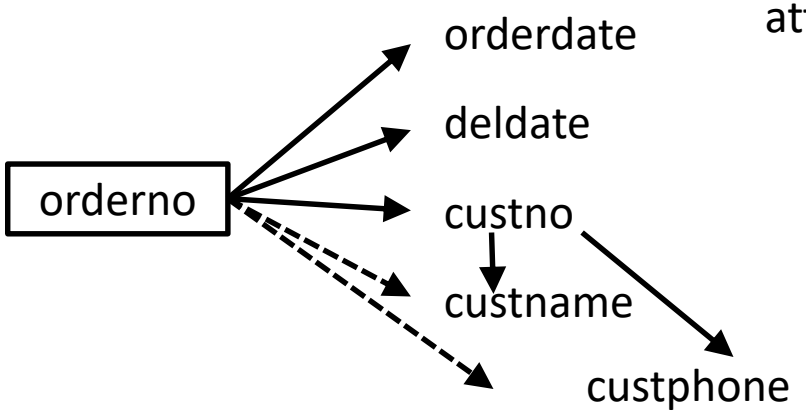
All 3 relations are in **2NF**

Step 5. Check the relations for transitive dependencies

The **PART** relation is also in **3NF** – since there are **no transitive dependencies**; all non-key attributes are **mutually independent** and fully functionally dependent on the PK.

The **ORDEREDPART_1** relation is also in **3NF** – since, with only one non-key attribute, a transitive dependency is not possible.

However, the **ORDER** relation does contain a **transitive dependency** – since there are functional dependencies between the non-key attributes. They are **NOT mutually independent**.



custno → custname, custphone

leads to the **transitive dependency**

orderno → custname, custphone

Normalisation - Examples

Step 6. Decompose **ORDER** using **NLD** to bring the schema to **3NF**

PART (partno, pname, unitcost)
ORDEREDPART_1 (orderno*, partno*, quantity)
ORDER (orderno, orderdate, deldate, custno*)
CUSTOMER (custno, custname, custphone)

All 4 relations are in **3NF** and ready for implementation

Normalisation improves the **quality** of the data stored in a RDBMS. In **3NF**:

- all non-key attribute values appear once in separate tables – **resolving update anomalies** and making update easier
- tables can be joined to reform original 1NF table as required (with penalty of **join overhead**)

However, Normalisation does not consider **efficiency** of implementation.

For example, if a **3NF** table has little/no **update activity** then it might be better left in lower form (e.g. **2NF**) for efficiency reasons (**Denormalisation**)

e.g. in table **EMPLOYEE** (empno, address, postcode) the transitive dependency

empno → address → postcode exists

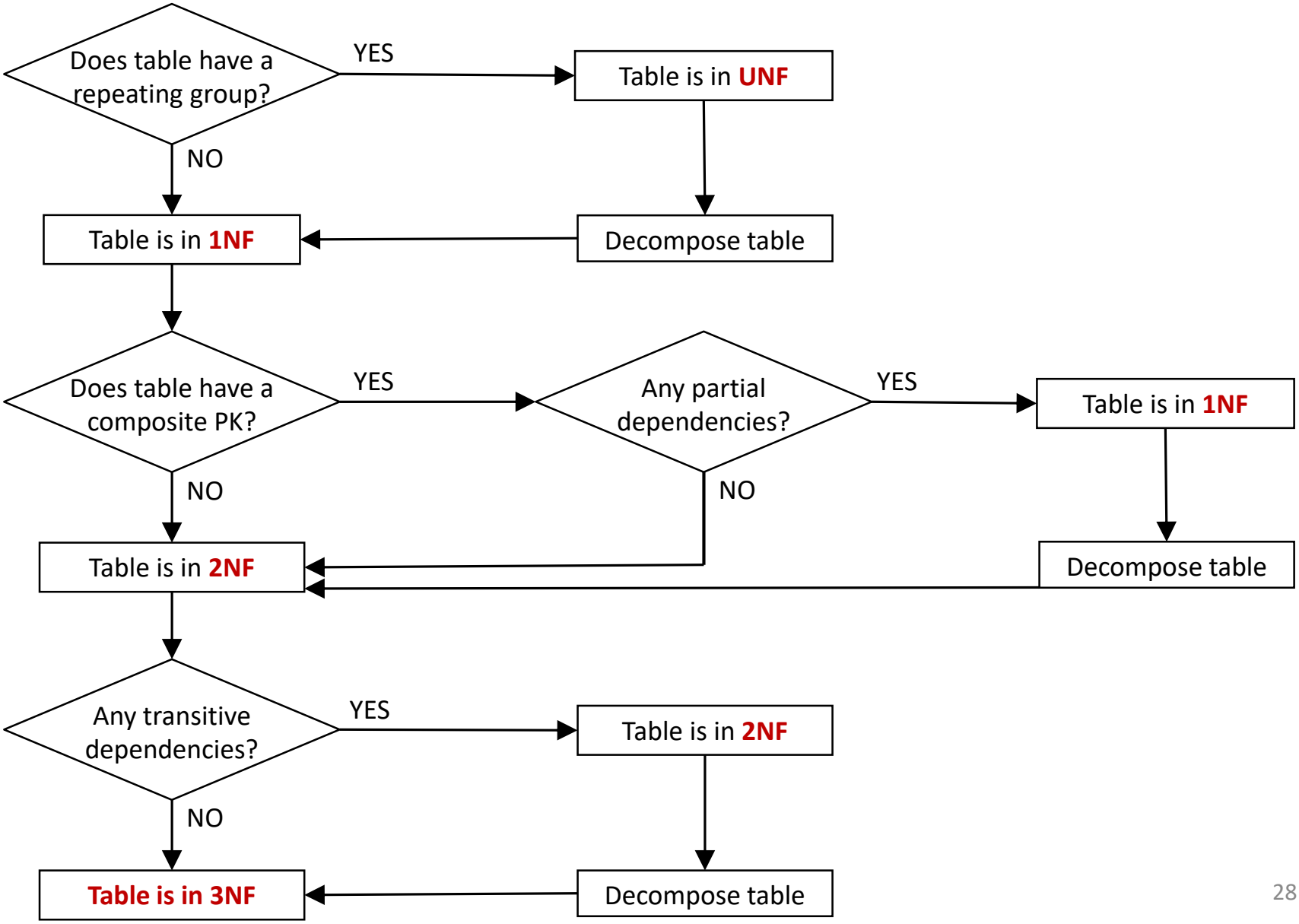
Normalisation dictates that this is decomposed into

EMPLOYEE (empno , address*)
ADDRESS (address, postcode)

This however is inefficient since **ADDRESS** is updated rarely, but data is retrieved frequently.

The **join overhead** is avoided if **EMPLOYEE** is left in **2NF**.

Normalisation - Flowchart



Normalisation

Recall the desirable properties of a relational schema:

- The **minimal number of attributes** necessary
- Attributes with a close logical relationship are found in the same table (**functional dependency**)
- **minimal redundancy** – each attribute appears only once (except for foreign keys)

A database system of any size is **NOT** normally designed bottom-up using normalisation.

In designing a database:

1. create an ER Diagram
2. use the relational mapping rules to create a linked relational schema
3. check that each relation is in **3NF** and if it is not
   - either decompose to **3NF**
   - or leave in **2NF** for performance reasons if justified by update rate

3NF is normally sufficient for most aspects of database design.

However, there are rarer issues that can arise which may require further decomposition

Boyce Codd Normal Form (**BCNF**), 4NF, 5NF, 6NF