THE PLACE OF USEFUL LEARNING

University of Strathclyde Glasgow

Database Fundamentals – CS990

Database and Web Systems Development - CS952

Image source: https://www.strath.ac.uk/branding/

Course Content

1. Introduction to Relational Databases *(Introduction + Relational Model)*
2. Data Modelling - *(Entity Relationship Modelling + The Enhanced Entity Relationship Model)*
3. Database Design and SQL - *(Logical modelling + Introduction to SQL)*
4. Further SQL - *(Advanced SQL queries + Creating tables with SQL)*
5. Normalisation - *(Normalisation to second normal form + Third normal form)*

Today

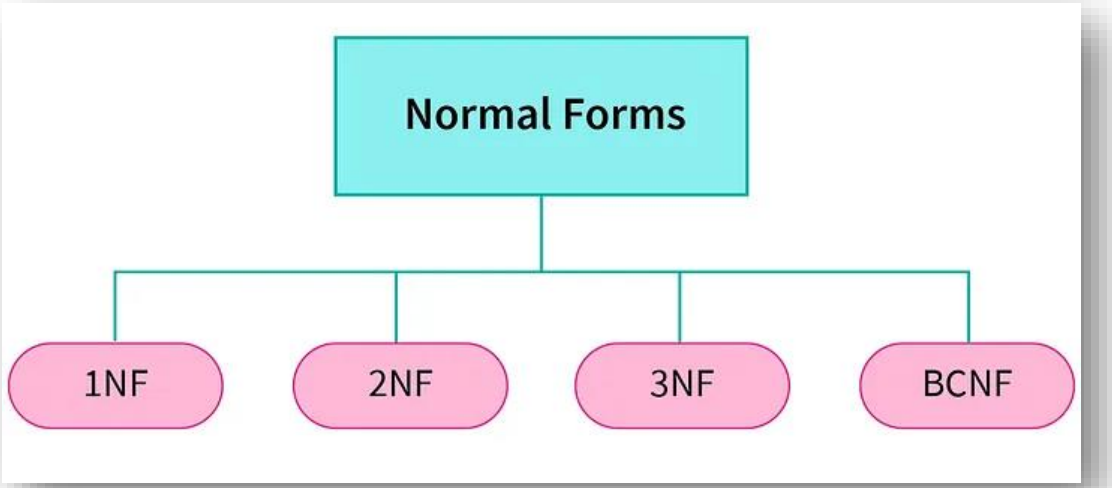
1. *Normalization*
2. *Dependencies*
3. *All three forms of Normalization*
4. *Examples*

Reference material

- Database, A Practical Approach to Design, Implementation, and Management (Thomas Connolly•Carolyn Begg), Sixth Edition
- 451 – 473

- Modern Database Management (12 Edition) (Jeff Hoffer et al.)
- 214 -223

Overview



- Normalisation of your database is a good thing
  - It makes the storage of data more efficient
  - It simplifies maintenance of the data
- There are lots of normal forms
  - 1NF, 2NF, 3NF, BCNF, 4NF, 5NF
- We will go through some of these, identifying the problems at each stage to motivate the next stage.
  - We will stop at 3NF

https://medium.com/@yg17381/normalization-in-dbms-297269fe7e9f

Motivation example

Student_ID	Student_Name	Courses	Instructor(s)
101	Alice	Database, AI	Smith, Lee
102	Bob	Web Dev, AI, Networks	Brown, Lee, Green
103	Charlie	Database	Smith
-	-	-	-
-	-	-	-

- **Multi-valued attributes**:
  - The Courses column contains multiple values (**Database, AI, etc**.).
  - The Instructor column also has **multiple values**.

- **Redundancy & Inconsistency**:
  - If an instructor's name changes, we need to update **multiple** records.
  - Data integrity is hard to maintain.

- **Difficult Queries**: Finding all students taking **AI** is harder because **AI is stored inside a string**.

Normalisation

- What makes a well-designed set of relations? Are some relations *better* than others? Do we know what a relation is?
- Also, what should we do if we are given an informally set-up table of data and asked to convert this to a relational database?
- Usually, if we have designed a database from scratch using E-R modelling, we will end up with a well-designed set of relations.
- But in other cases, we need to apply ***NORMALISATION***.

- **Example**: We are given a table (next slide) of newspaper **readers** and the **newspapers** they read.
  - For example, reader ***Smith*** likes to read the ***Record*** and the ***Mail***. We are asked to transform this table into a database.

Example…

Reader	Newspapers
Smith	Record, Mail
Johnson	Mail, Times
Lee	Record, Herald
Kim	Mail, Herald

Reader_ID	Reader_Name
1	Smith
2	Johnson
3	Lee
4	Kim

Newspaper_ID	Newspaper_Name
1	Record
2	Mail
3	Times
4	Herald

Reader_ID	Newspaper_ID
1	1
1	2
2	2
2	3
3	1
3	4
4	2
4	4

1.Readers table: Contains information about readers.
2.Newspapers table: Contains information about newspapers.
3.Readers_Newspapers table (or a junction table): Represents the many-to-many relationship between readers and newspapers.

Data Normalisation to remove all 3 Anomalies

Student_ID	Student_Name	Module_ID	Title	Staff_Name	Staff_email	Job_title
40319000	Lee Chong Wei	CS990	Databases	Jacques Kallis	j.k@strath.ac.uk	Lecturer
40319001	Victor Axelson	CS990	Databases	Jacques Kallis	j.k@strath.ac.uk	Lecturer
40319002	Low Kean Yew	CS952	Object Oriented Programming	Peter Smith	d.n@strath.ac.uk	Lecturer
~~40319003~~	~~John Dory~~	~~CSC1030~~	~~Web Technologies~~	~~Andrew Tenenbaum~~	a.t@strath.ac.uk	~~Senior Lecturer~~
40319004	Harry Ramsdens	CS980	Software Design Principles	Daren Shaw	D.S@strath.ac.uk	Reader
??	??	??	??	Neil Johnson	n.j@strath.ac.uk	Professor

Deletion Anomaly

Student withdrawal, but module and staff information are deleted too!

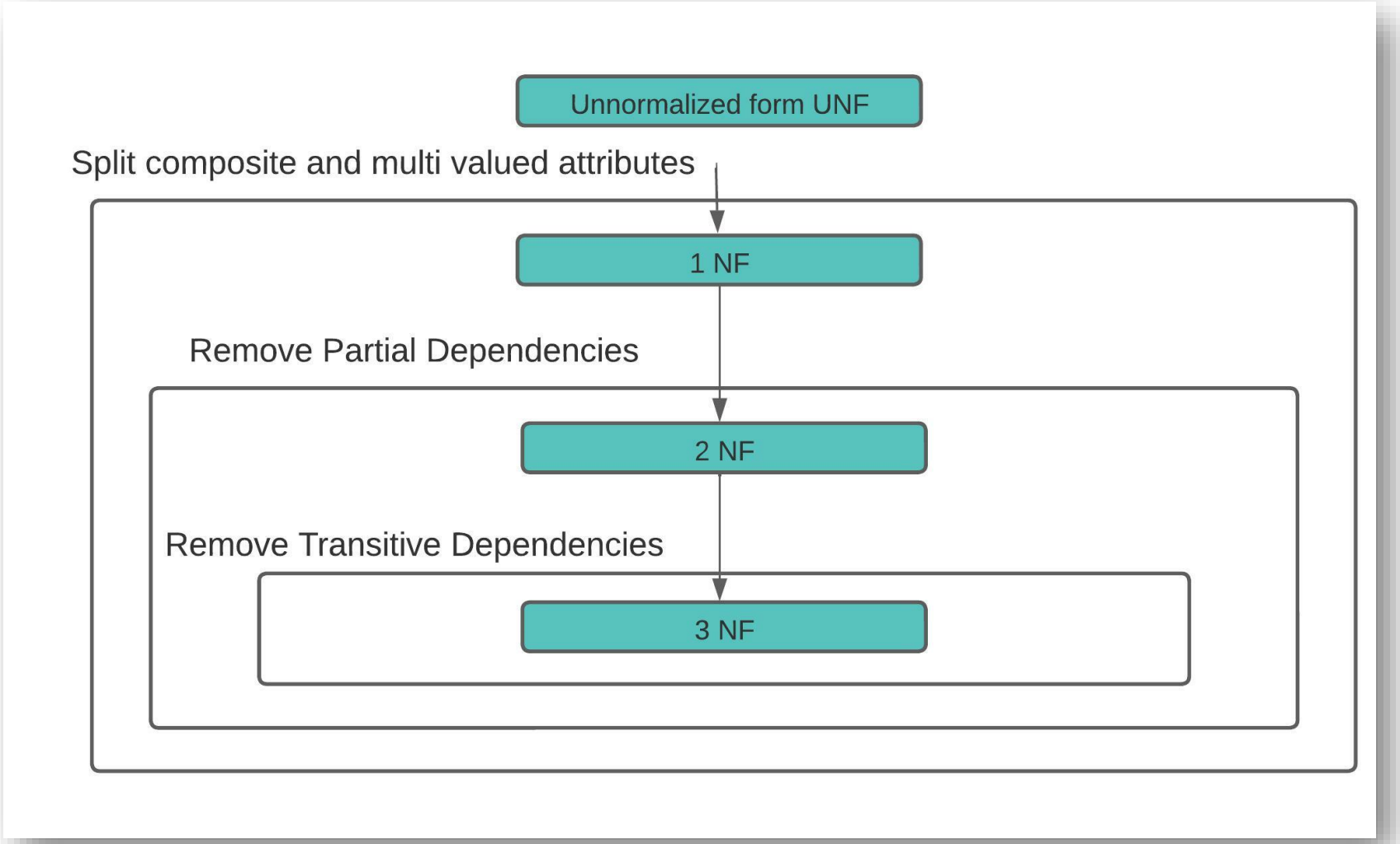
Insertion Anomaly

Added a new staff, but Student and module information are missing!

Update Anomaly

Staff changed to Peter, but also need to change email, job title!

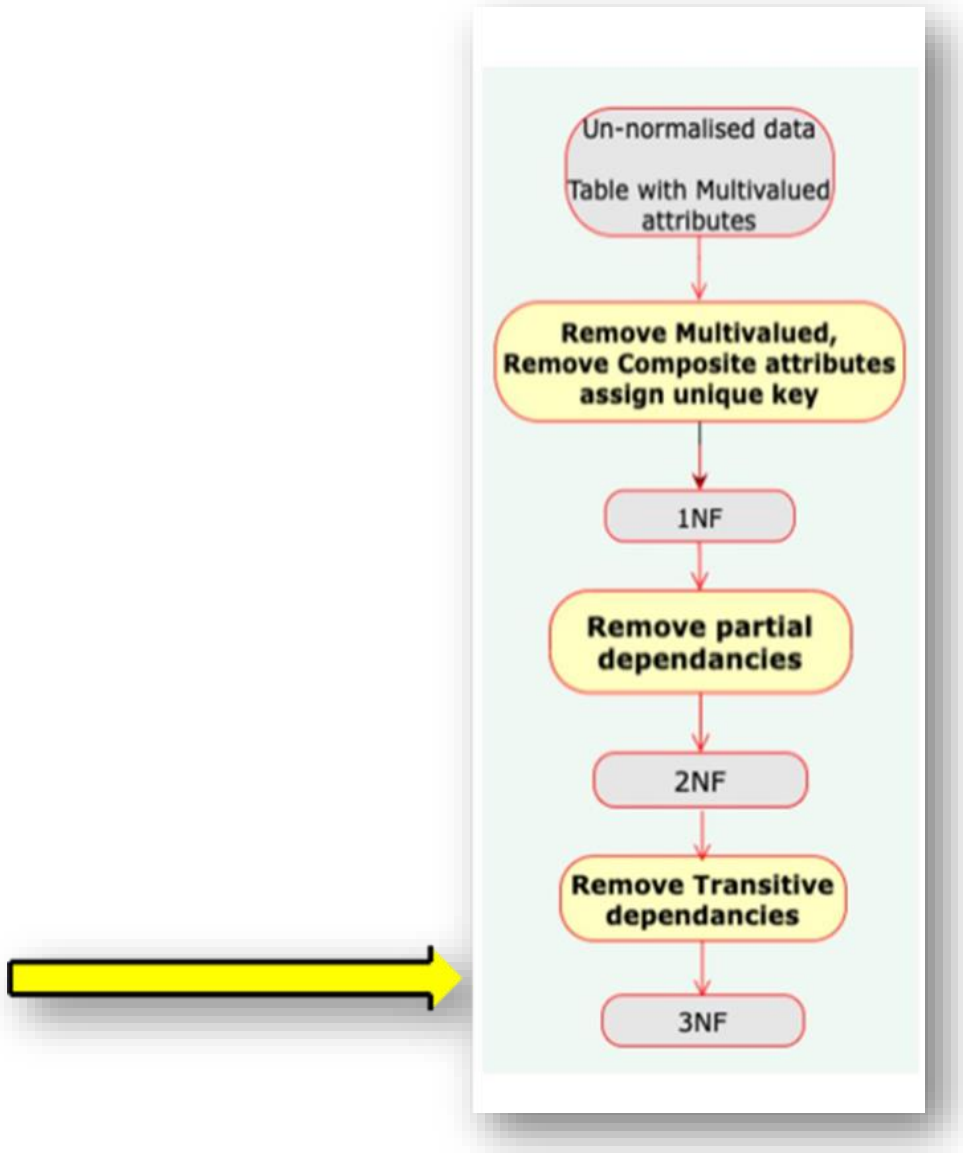
Steps…



https://www.analyticsvidhya.com/blog/2022/08/database-normalization-a-step-by-step-guide-with-examples/

What you need to know…

- ✓ Further learning of Database design
  - Learn more about transforming **ER** data models into **relational** designs.
  - To turn customer's requirements into well-formed relations by undergoing **data normalisation.**
- ✓ Understand the Normalisation process
  - Know how to implement 1NF, 2NF and 3NF.



Unnormalized Relations

- Here's a first try:

READS	
Name	***PaperList***
Smith	Record, Mail
Lee	Herald

- This is not ideal. Each person is associated with an unspecified number of papers. The items in the ***PaperList*** column do not have a consistent form.
- Generally, RDBMS can't cope with relations like this. Each entry in a table needs to have ***a single data item*** in it.
- This is an ***unnormalized*** relation.
- All RDBMS require relations *not* to be like this
  - - not to have multiple values in any column (i.e. ***no repeating groups***)

First Normal Form (1NF)

- Here is another try:

READS2	
Name	***Paper***
Smith	Record
Smith	Mail
Lee	Herald

- This clearly contains the same information.
- And it has the property that we sought. It is 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)
- So, this will be the first requirement in designing our databases:
  - our relations *must* be in 1NF.

Achieving 1NF

- In general, to achieve 1NF we need to get rid of *repeating groups* in our tables. There are two alternative ways of doing this.

- The *one-table approach*: we extend the table rows by replicating the non-repeated columns for each repeated item value. This is what we did in the previous slide.

- The two-table approach: split the repeating and non-repeating data into separate tables (Non-loss Decomposition)
  - We must then choose a primary key for the repeating data table
  - …and insert this as a foreign key in the non-repeating data table

- The two-table approach is often better as it takes up *less* space and leads us to 2nd Normal Form
- Any other problems with 1NF?

Recap: A Relation in 1NF

- Relations that are in 1NF can still have considerable problems.
- Example : Staff borrowers in a Library.
  - their staff number functions as a **Library** number
  - and someone has had the ingenious idea of adding details of books borrowed (in the same relation/table)
  - Here's a first go:

StaffBorrower						
Sno	***Sname***	***Sdept***	***Grade***	***Salary***	***Bno***	***Date_out***
1	Smith	Computing	2.7	26813	1	30/06/2002
2	Black	Marketing	1.5	17278	8	08/07/2002

- Primary key: *(Sno,Bno)*

Problems with a 1NF Relation: Duplication

- Now suppose that **Smith** borrows another book.
- To ensure 1NF, we shall have to have a complete new row:

StaffBorrower						
Sno	***Sname***	***Sdept***	***Grade***	***Salary***	***Bno***	***Date_out***
1	Smith	Computing	2.7	26813	1	30/06/2002
2	Black	Marketing	1.5	17278	8	08/07/2002
1	Smith	Computing	2.7	26813	53	12/07/2002

- We have stored all the other details about this member of staff *again*, in the new row.
- Not only is information about **Smith** duplicated, but
  - the fact that staff on grade 2.7 earn £26,813 is duplicated

Problems with a 1NF Relation: Update Anomalies

- Such repetition means that updates can be difficult.
- Suppose that **Smith** goes on to a new grade.
  - Changes would be required to all records for **Smith**.
  - (And there is a danger that we may miss some.)
- 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.
- A fact should be stored only *once*. Updates are then problem-free.
- This example relation is poorly structured, being subject to *update anomalies*.

StaffBorrower						
Sno	***Sname***	***Sdept***	***Grade***	***Salary***	***Bno***	***Date_out***
1	Smith	Computing	2.7	26813	1	30/06/2002
2	Black	Marketing	1.5	17278	8	08/07/2002
1	Smith	Computing	2.7	26813	53	12/07/2002

More Problems in a 1NF Relation: *Insertion and Deletion 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 do we record this? The following violates entity integrity:

StaffBorrower						
Sno	***Sname***	***Sdept***	***Grade***	***Salary***	***Bno***	***Date_out***
Null	Null	Null	2.8	27491	Null	Null

- We call this an *insertion anomaly.*
- Also: suppose that *Smith* returns all their 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?
    - **-leaving duplicated information about Smith**
- These are *deletion anomalies.*

Solution: Non-loss decomposition (NLD)

- How do we deal with these problems?
- We carry out a ***non-loss decomposition (NLD):*** we replace the relation by multiple relations representing part of the data (*projections*) from which the original relation can be re-created (*by joining*)
- The re-creation must give no less and no more than we started with
- See some examples next

An example of an NLD - I

SUC

Student	Unit	Coordinator
Gary	3131	Hamilton
Tracy	3131	Hamilton
Sinead	3133	Clark
Sean	3133	Clark

– decomposes into

SU

Student	Unit
Gary	3131
Tracy	3131
Sinead	3133
Sean	3133

UC

Unit	Coordinator
3131	Hamilton
3133	Clark

- If we Join `SU` and `UC` (over `Unit`, obviously) we get back exactly the four rows of `SUC` we started with
  – so this is an NLD

An example of an NLD - II

- Similarly, we can carry out an NLD of the `Borrower` relation

STAFF_BORROWER2

Sno	Sname	Sdept	Grade	Salary
1	Smith	CSM	2.7	21123

LOAN

Sno	Bno	Date_out
1	1	30/6/2004
1	7	1/7/2004

- At first it looks as though the new scheme will take more space (but actually it takes *less*, because we remove repetitions)

A formal apparatus

- We need a method of analysing relations to detect and prevent these problems
- We need a set of definitions and procedures to
  - diagnose whether relations have a ‘silly’ design
  - turn them into other, better designed, relations in a systematic way
- We begin by reminding ourselves what a relation “means”
  - a relation’s interpretation is not always obvious from the names of its columns ..e.g.

SUML

Stu	UCode	Lect	Mark
Gary	3131	Hamilton	64

The predicate of a relation

- Any relation has a ***predicate*** - a definition of what any row means
- This will usually just be a statement in natural language, e.g.
  - **SUML1** : "the student `Stu` took unit `UCode` and obtained a mark of `Mark`. Unit `UCode` is coordinated by lecturer `Lect`."
  - This means that each unit has one coordinator (a single lecturer who manages the unit). If the same unit appears multiple times in the table, it does not mean that multiple lecturers teach it; just that this lecturer is the coordinator.
- Or perhaps
  - **SUML2** : "the student `Stu` took unit `UCode` and obtained a mark of `Mark`. In that unit, their tutorial was taken by lecturer `Lect`."
  - Different students in the same unit may have different tutors.
- Or even
  - **SUML3** : "the student `Stu` took unit `UCode` and obtained a mark of `Mark`. In that unit, lecturer `Lect` was one of the lecturers".
  - This means that multiple lecturers could be teaching the unit.

Relations: Tuples represent true statements

SUML

Stu	UCode	Lect	Mark
Gary	3131	Hamilton	64

- Each row ("tuple") is a true statement: so, in `SUML1`, "Gary took 3131 and got 64. 3131 is coordinated by Dr Hamilton"
- In `SUML2` that would become "Gary took 3131 and got 64. His tutorial was taken by Dr Hamilton"
- The *Closed World Assumption:* if a row isn't in the relation, the corresponding statement is false
  - So, if we don't see

Stu	UCode	Lect	Mark
Fiona	3131	Null	Null

  - then Fiona didn't do 3131

Normalisation: the big picture

The basic idea is as follows:

- Some earlier (ER) analysis has given a database design consisting of one or more relations: for each relation, ...
- We inspect the relation's predicate to find
  - which attribute(s) is/are the candidate *key*
  - what *functional dependencies* exist (see slide after next)
- From these, we decide what *normal form* the relation is in
- And hence whether the relation should be decomposed
- And, if so, how to do that decomposition

Recap: Candidate Keys

- As you will know, sometimes there can be several possible candidate keys for a relation
  - suppose departments have unique names and also unique ids

DEPT

ID	Name	HoD
Mkt	Marketing	Prof Burt
CSM	Computing Science and Maths	Dr Clark

  - For the above relation, `ID` and `Name` are ***candidate** keys*
- We can choose which we designate as *the **primary** key* of the relation
- A ***key** field* is a field (column) that is [part of] a candidate key
  - `ID` and `Name` are key fields
- A ***non-key** field* is a field that isn't part of any candidate key
  - `HoD` is a non-key field
- Every relation has a **candidate/primary** key (rows are unique by definition)

Functional Dependency

- In any relation, a column (or set of columns) Y is *functionally dependent* on a column (or set of columns) X if at any one time exactly one Y value is associated with any X value.
  - We write $X \rightarrow Y$.
- For example, at any one time a single surname (Y) is associated with a particular registration-number (X)
  - note that the surname may change
  - and that the dependency is only one-way (i.e. *directional*): registration-number determines surname, but the reverse is not true
- By definition, the attribute(s) on the left of a functional dependency (FD) is called the *determinant* of that FD
  - E.g. we call X a *determinant* and we write `X -> Y`
- To know the FDs, you must know the predicate

Functional Dependency - More

- Here are some examples of FDs from *StaffBorrower*:

  Sno → *Sname*
  Sno → *Sdept*
  Sno → *Grade*
  Grade → *Salary*
  (Sno,Bno) → *Date_out*
  (Sno,Bno) → *Sname*
  …(there are many others)

- We note that `(Sno,Bno)` is the ***primary*** key of `STAFF_BORROWER`
- ***Non-key*** columns, by definition, are FD on primary key columns
- Definition: $X \rightarrow Y$ is a *full* FD (FFD) if Y is dependent on the *whole of* X.
  - Note that here *Sname* is not *fully* FD on the primary key *(Sno,Bno)*

- When normalising a database, we are only interested in *full* FDs.

Second Normal Form (2NF)

- A relation is in *Second Normal Form* (2NF) if
  - it is in 1NF and
  - every non-key column is *fully* FD (FFD) on the primary key.

- The relation *StaffBorrower* is not in 2NF. Why not?
  - Because a non-key column (such as `Sname`) is not fully FD on the primary key `(Sno,Bno)`

StaffBorrower						
Sno	***Sname***	***Sdept***	***Grade***	***Salary***	***Bno***	***Date_out***
1	Smith	Computing	2.7	26813	1	30/06/2002
2	Black	Marketing	1.5	17278	8	08/07/2002

- We can achieve 2NF by splitting our 1NF into two or more relations in 2NF.

2NF Decomposition

- We can split any relation in 1NF into two or more relations in 2NF.
- For example, previously we saw that we could decompose `STAFF_BORROWER` into

StaffBorrower2

Sno	***Sname***	***Sdept***	***Grade***	***Salary***
1	Smith	Computing	2.7	26813

Loan

Sno	***Bno***	***Date_out***
1	1	30/06/2002
1	53	12/07/2002

- Both of the above are in 2NF.

- *Recap:* At first it looks as though the new scheme will require more space (but actually it takes *less*, because we avoid repetitions)
- And the 2NF decomposition solves many of the problems
  - but not all…..

But ...

- The scheme above does not solve all problems: the Grade/Salary problem remains.
- We may observe that the dependency of *Salary* on *Sno* has a special property. It is indirect (or "*transitive*"), as it is achieved via a third, non-key attribute:
  - *Sno* → *Grade* → *Salary*

- We can fix the problem by insisting that our relations have a stronger property. This will lead us from 2NF to 3NF.

Third Normal Form

- A relation is in *Third Normal Form* (3NF) if
  - it is in 1NF + 2NF and
  - every non-key column is *non-transitively* fully FD on the primary key.
- We can always find a decomposition into 3NF.
- In our decomposition above, the Loan relation is already in 3NF:
  - the only non-key column is *Date_out*
  - the primary key is *(Sno,Bno)*, and *(Sno,Bno)* $\rightarrow$ *Date_out* (i.e. FFD)
- But (as we have seen) StaffBorrower2 is not in 3NF.
- It is easy to see what to do:

StaffBorrower3			
Sno	***Sname***	***Sdept***	***Grade***
1	Smith	Computing	2.7

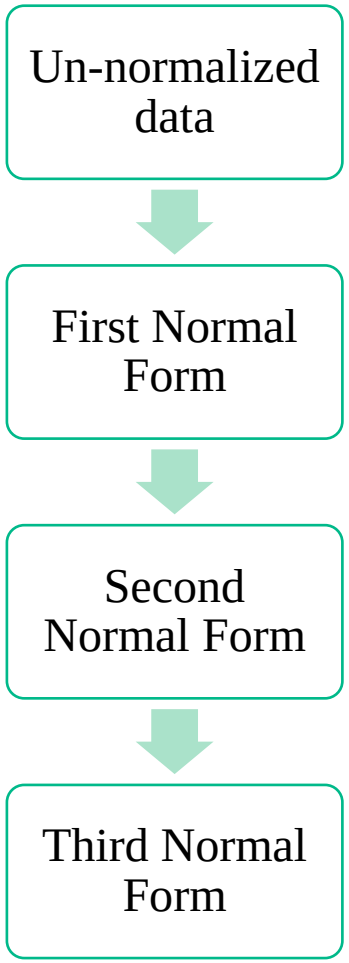
PayScale	
Grade	***Salary***
2.7	26813

Other/Higher Normal Forms

- So, we have a hierarchy of normal forms (1NF, 2NF, 3NF).
  - To be useful, a database structure really must be in 3NF (and we can always find an NLD into 3NF)
  - It is the highest normal form with no disadvantages
  - The result may look as though it would take more space but actually takes less
- 3NF solves most problems, but there are some rare anomalies that can still arise (especially where there are *overlapping* candidate keys).
- Hence, there are further normal forms:
  - Boyce-Codd normal form (BCNF)
  - Fourth normal form
  - Fifth normal form
- And algorithms to carry out decompositions…
- … but we shall NOT deal with these in this course… see Ritchie for details.

Summary

Design the database using E/R diagrams and specify the relations from the diagrams. Check the form of the relations.



Remove all multi-values attributes

If the primary key has more than one field, ensure that all fields are dependent on the whole key

Remove all transitive dependencies, i.e. fields dependent on non-key fields

Summary 2

Step	Action	Problem Solved
UNF → 1NF	Remove **multi-valued** attributes	**Atomic** data (one value per cell)
1NF → 2NF	Remove **partial** dependencies	Non-key columns **depend** on full PK
2NF → 3NF	Remove **transitive** dependencies	**No** non-key column depends on another non-key column

Normalisation isn’t always best

- Sometimes, there are practical reasons not to normalise.
- Performance issues:
  - If a specific query is slow this may be because of multiple operations to join normalized tables together.
  - Consider maintaining the query result as a relation (with functional dependencies and duplication). Especially if that query is used a lot.
  - Similarly, for regular reports and commonly used computed values (grouped subtotals, or expressions).

- Downside of not normalising:
  - Duplicated data / more storage space
  - Potential update / deletion / insertion anomalies
    - **(therefore need code to avoid these)**
  - Slower insert / modification / deletion

Things to do!

- Class Test 3 – closing tomorrow at 17:00
- Download updated PPTs from MyPlace
- No more labs

Thank you.

Course Content

1. Introduction to Relational Databases *(Introduction + Relational Model)*
2. Data Modelling - *(Entity Relationship Modelling + The Enhanced Entity Relationship Model)*
3. Database Design and SQL - *(Logical modelling + Introduction to SQL)*
4. Further SQL - *(Advanced SQL queries + Creating tables with SQL)*
5. Normalisation - *(Normalisation to second normal form + Third normal form)*

THE PLACE OF USEFUL LEARNING

University of Strathclyde Glasgow

Database Fundamentals – CS990

Database and Web Systems Development - CS952

Image source: https://www.strath.ac.uk/branding/