Xi'an Jiaotong-Liverpool University

西交利物浦大學

# Database: Normalisation

Jianjun Chen

# Characteristics of Good DB Designs

- The minimal number of attributes necessary to support the data requirements of the enterprise;
- Attributes with a close logical relationship are found in the same relation;
- Minimal redundancy with each attribute represented only once with the important exception of attributes that form all or part of foreign keys.

# What is Normalisation?

- Normalisation is a technique of re-organising data into multiple related tables, so that data redundancy is minimised.

- ER modeling VS. Normalisation:
  - ER diagram is useful when you have detailed database specifications but no existing table design.
  - When you already have a database, but the tables are not well designed, you can use normalisation techniques to improve them.

# Problems with Data Redundancy

Staff

| staffNo | sName | position | salary | branchNo |
|---|---|---|---|---|
| SL21 | John White | Manager | 30000 | B005 |
| SG37 | Ann Beech | Assistant | 12000 | B003 |
| SG14 | David Ford | Supervisor | 18000 | B003 |
| SA9 | Mary Howe | Assistant | 9000 | B007 |
| SG5 | Susan Brand | Manager | 24000 | B003 |
| SL41 | Julie Lee | Assistant | 9000 | B005 |

Branch

| branchNo | bAddress |
|---|---|
| B005 | 22 Deer Rd, London |
| B007 | 16 Argyll St, Aberdeen |
| B003 | 163 Main St, Glasgow |

VS.

Staff Branch

| staffNo | sName | position | salary | branchNo | bAddress |
|---|---|---|---|---|---|
| SL21 | John White | Manager | 30000 | B005 | 22 Deer Rd, London |
| SG37 | Ann Beech | Assistant | 12000 | B003 | 163 Main St, Glasgow |
| SG14 | David Ford | Supervisor | 18000 | B003 | 163 Main St, Glasgow |
| SA9 | Mary Howe | Assistant | 9000 | B007 | 16 Argyll St, Aberdeen |
| SG5 | Susan Brand | Manager | 24000 | B003 | 163 Main St, Glasgow |
| SL41 | Julie Lee | Assistant | 9000 | B005 | 22 Deer Rd, London |

# Problems with Data Redundancy

- StaffBranch:
  - The details of a branch (bAddress, in this case) are repeated for every member of staff.

- Staff and Branch:
  - The branch information appears only once for each branch in the Branch relation
  - Only the branch number (branchNo) is repeated in the Staff relation, to represent where each member of staff is located.

StaffBranch

| staffNo | sName | position | salary | branchNo | bAddress |
|---|---|---|---|---|---|
| SL21 | John White | Manager | 30000 | B005 | 22 Deer Rd, London |
| SG37 | Ann Beech | Assistant | 12000 | B003 | 163 Main St, Glasgow |
| SG14 | David Ford | Supervisor | 18000 | B003 | 163 Main St, Glasgow |
| SA9 | Mary Howe | Assistant | 9000 | B007 | 16 Argyll St, Aberdeen |
| SG5 | Susan Brand | Manager | 24000 | B003 | 163 Main St, Glasgow |
| SL41 | Julie Lee | Assistant | 9000 | B005 | 22 Deer Rd, London |

Branch

| branchNo | bAddress |
|---|---|
| B005 | 22 Deer Rd, London |
| B007 | 16 Argyll St, Aberdeen |
| B003 | 163 Main St, Glasgow |

Staff

| staffNo | sName | position | salary | branchNo |
|---|---|---|---|---|
| SL21 | John White | Manager | 30000 | B005 |
| SG37 | Ann Beech | Assistant | 12000 | B003 |
| SG14 | David Ford | Supervisor | 18000 | B003 |
| SA9 | Mary Howe | Assistant | 9000 | B007 |
| SG5 | Susan Brand | Manager | 24000 | B003 |
| SL41 | Julie Lee | Assistant | 9000 | B005 |

# Update Anomalies

- Data redundancy not only increases memory usage, but also leads to update anomalies.
- Let's take StaffBranch for example:
  - Insert anomalies: One cannot add a new branch without the information of a staff in this branch (staffNo is PK).
    - Also, must input the correct branch information every time a new staff is added.
  - Delete anomalies: Deleting the last staff of a branch also deletes the information of that branch.
  - Modification anomalies: To change one branch information, all staffs (rows) in that branch must also be updated.

StaffBranch

| staffNo | sName | position | salary | branchNo | bAddress |
|---|---|---|---|---|---|
| SL21 | John White | Manager | 30000 | B005 | 22 Deer Rd, London |
| SG37 | Ann Beech | Assistant | 12000 | B003 | 163 Main St, Glasgow |
| SG14 | David Ford | Supervisor | 18000 | B003 | 163 Main St, Glasgow |
| SA9 | Mary Howe | Assistant | 9000 | B007 | 16 Argyll St, Aberdeen |
| SG5 | Susan Brand | Manager | 24000 | B003 | 163 Main St, Glasgow |
| SL41 | Julie Lee | Assistant | 9000 | B005 | 22 Deer Rd, London |

# How to Re-organise Tables?

- The StaffBranch table is better split into the Staff table and the Branch table.
  - We know the answer beforehand.

- What information can help us to redesign tables when we don't know the answer?
  - We can observe the data (see next few slides) in the table and find out the relationship between attributes.
  - Common sense may also help, but be careful when applying common sense!


Queue No. A31
Table type: up to 4

31 People are waiting ahead of you.

A1, A2, A3… A99, A999 -> A1

# Observing Table Data

- The staffNo refers to a unique staff member in the real life.
  - There's only one sName for that staffNo, because a person can only have one single name.
  - Similarly, there's only one staff position, one staff salary value, one branchNo and one bAddress for that person. (Assume a staff can only have one position and works in only one branch office)

**Staff Branch**

| staffNo | sName | position | salary | branchNo | bAddress |
|---|---|---|---|---|---|
| SL21 | John White | Manager | 30000 | B005 | 22 Deer Rd, London |
| SG37 | Ann Beech | Assistant | 12000 | B003 | 163 Main St, Glasgow |
| SG14 | David Ford | Supervisor | 18000 | B003 | 163 Main St, Glasgow |
| SA9 | Mary Howe | Assistant | 9000 | B007 | 16 Argyll St, Aberdeen |
| SG5 | Susan Brand | Manager | 24000 | B003 | 163 Main St, Glasgow |
| SL41 | Julie Lee | Assistant | 9000 | B005 | 22 Deer Rd, London |

# Observing Table Data

- Can we claim the opposite:
  - There's only one staffNo associated with a sName?
  - There's only one staffNo associated with a position?
  - No
- As a result, staffNo has M:1 relationship with
  - sName, position, salary, branchNo and bAddress

Staff Branch

| staffNo | sName | position | salary | branchNo | bAddress |
|---|---|---|---|---|---|
| SL21 | John White | Manager | 30000 | B005 | 22 Deer Rd, London |
| SG37 | Ann Beech | Assistant | 12000 | B003 | 163 Main St, Glasgow |
| SG14 | David Ford | Supervisor | 18000 | B003 | 163 Main St, Glasgow |
| SA9 | Mary Howe | Assistant | 9000 | B007 | 16 Argyll St, Aberdeen |
| SG5 | Susan Brand | Manager | 24000 | B003 | 163 Main St, Glasgow |
| SL41 | Julie Lee | Assistant | 9000 | B005 | 22 Deer Rd, London |

# Observing Table Data

- The branchNo refers to a unique branch office in the real life.
  - There's only one unique bAddress associated with one branchNo.
- Can we claim the opposite:
  - There's only one unique branchNo associated with one bAddress?
  - Yes
- branchNo has a 1:1 relationship with bAddress.

Staff Branch

| staffNo | sName | position | salary | branchNo | bAddress |
|---|---|---|---|---|---|
| SL21 | John White | Manager | 30000 | B005 | 22 Deer Rd, London |
| SG37 | Ann Beech | Assistant | 12000 | B003 | 163 Main St, Glasgow |
| SG14 | David Ford | Supervisor | 18000 | B003 | 163 Main St, Glasgow |
| SA9 | Mary Howe | Assistant | 9000 | B007 | 16 Argyll St, Aberdeen |
| SG5 | Susan Brand | Manager | 24000 | B003 | 163 Main St, Glasgow |
| SL41 | Julie Lee | Assistant | 9000 | B005 | 22 Deer Rd, London |

# Observing Table Data

- Can we claim these:
  - A sName is associated with a unique bAddress?
  - A position is associated with a unique salary?
  - A branchNo is associated with a unique position?
  - …
- These attributes are not related in any aspects.

Staff Branch

| staffNo | sName | position | salary | branchNo | bAddress |
|---|---|---|---|---|---|
| SL21 | John White | Manager | 30000 | B005 | 22 Deer Rd, London |
| SG37 | Ann Beech | Assistant | 12000 | B003 | 163 Main St, Glasgow |
| SG14 | David Ford | Supervisor | 18000 | B003 | 163 Main St, Glasgow |
| SA9 | Mary Howe | Assistant | 9000 | B007 | 16 Argyll St, Aberdeen |
| SG5 | Susan Brand | Manager | 24000 | B003 | 163 Main St, Glasgow |
| SL41 | Julie Lee | Assistant | 9000 | B005 | 22 Deer Rd, London |

# Functional Dependency

A method for finding relationships between attributes within a table

# Functional Dependencies

- We want to find relationship between attributes within a table so that we can regroup attributes based on their context and split the big table.
- Introducing **functional dependency (FD)**:

  "If A and B are attribute sets of relation R, B is functionally dependent on A (**denoted A → B**), if each value of A in R is associated with exactly one value of B in R."

- A is called **determinant**.

- The concept of FD is closely related to M:1 and 1:1 relationships.

# Functional Dependencies

For the attributes below:

PersonalTutors{LecID, LName, LEmail, StudentID, SName}

- We assume each Lecturer has an unique email address (1:1) and several students have one personal tutor (M:1).
- Obvious FDs:
  - **LecID → LName, Lemail**
    Each LecID of a lecturer is associated with exactly one lecturer name and his email address.
  - **StudentID → Sname**
    Given a studentID, you can find that student's name.

# Functional Dependencies

- **Other FD considerations:**
  - **LName → LecID**
    Two staffs may have the same name. Thus one name might be associated with two IDs.
  - **StudentID → LecID**
    Given a student id, there will be a personal tutor assigned to him.
  - **LecID → Sname**
    Given a lecturer ID, it is associated with one unique student name?
    No, a lecturer has many students as personal tutees.
  - **LEmail → LecID, LName**?
    This is another valid FD if email addresses are unique to lecturers.

PersonalTutors{LecID, LName, LEmail, StudentID, SName}

# Functional Dependencies

Observe these functional dependencies carefully, you will realise that:

- If these attributes are put together, they can form a relation, with the determinant being the unique key or primary key of the relation.

- For example:
  - LecID → LName, LEmail
  - StudentID → SName, LecID
- Or:
  - LEmail → LecID, LName
  - StudentID → SName, LEmail

PersonalTutors{LecID, LName, LEmail, StudentID, SName}

# Functional Dependencies

- However, the FD below is also true:
  - LecID, LName, LEmail → LName, LEmail
- Is (LecID, LName, LEmail) a good primary key?
- Recall the definition of super key, candidate key and primary key.

PersonalTutors{LecID, LName, LEmail, StudentID, SName}

# Full and Partial FD

- **Full functional dependency:** If A and B are two sets of attributes of a relation, B is fully functionally dependent on A, if B is functionally dependent on A, but not on any proper subset of A.
  - In other words, determinants should have the minimal number of attributes necessary to maintain the functional dependency with the attribute(s) on the right hand-side.

- **Partial FDs**: A → B, is a partial FD, if some attribute of A can be removed and the FD still holds
  - Formally, there is some proper subset of A, C ⊂ A, such that C → B

# Examples

- Full functional dependency in the Staff relation:
  - staffNo → branchNo
- How about:
  - staffNo, sName → branchNo?
  - This is a partial dependency since branchNo is also functionally dependent on a subset of (staffNo, sName), namely staffNo.

Staff

| staffNo | sName | position | salary | branchNo |
|---|---|---|---|---|
| SL21 | John White | Manager | 30000 | B005 |
| SG37 | Ann Beech | Assistant | 12000 | B003 |
| SG14 | David Ford | Supervisor | 18000 | B003 |
| SA9 | Mary Howe | Assistant | 9000 | B007 |
| SG5 | Susan Brand | Manager | 24000 | B003 |
| SL41 | Julie Lee | Assistant | 9000 | B005 |

# Determinant in Full/Partial FDs

Determinants in full functional dependencies:

- Will become candidate keys if we split the table

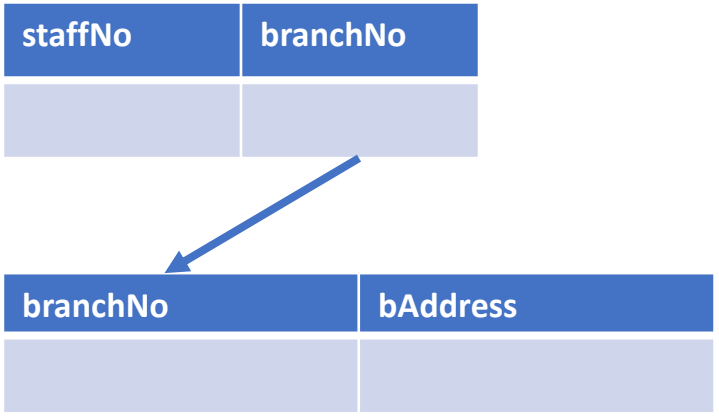
Determinants in partial functional dependencies:

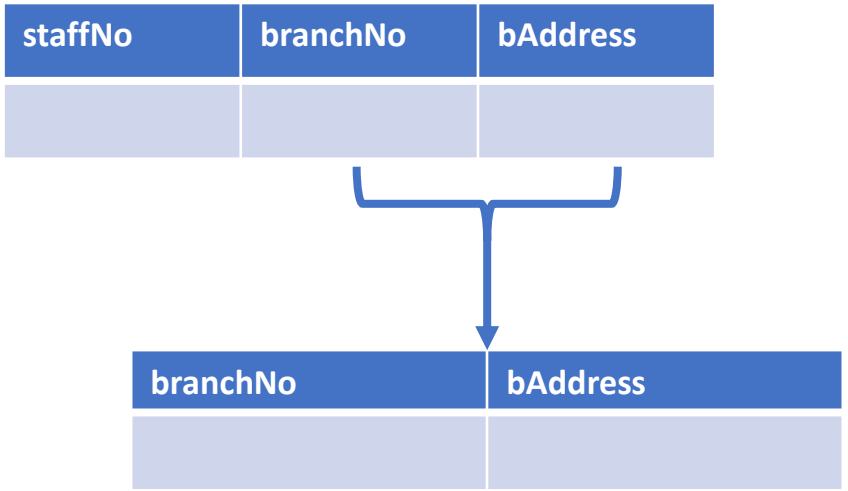
- Will become super keys.

# FDs in Normalisation

We only care about full FDs in normalisation. Why?

- Partial FDs results in super keys, if you use a foreign key to refer to a super key in another table, you will need extra columns in the referencing table.
  - branchNo VS (branchNo, bAddress) inside Staff table.
  - Unnecessary!



# More about Determinants

- If you observe the tuples in the Staff and Branch table, you can find out that the determinant has a M:1 or 1:1 relationship with other attributes in FD.

- Two staff members may have the same name.
  - Thus, one staff name can be associated with more than one staff number. (M:1)
- Two staff members may have the same position.
  - Thus, one position may be associated with more than one staff number (M:1)

Staff

| staffNo | sName | position | salary | branchNo |
|---|---|---|---|---|
| SL21 | John White | Manager | 30000 | B005 |
| SG37 | Ann Beech | Assistant | 12000 | B003 |
| SG14 | David Ford | Supervisor | 18000 | B003 |

# More about Determinants

"The determinant has a M:1 or 1:1 relationship with other attributes in FD"

- Two staffs may work in the same branch.
  - Thus, one branchNo may be associated with more than one staffNo. (M:1)
- A branch address is associated with exactly one branchNo
  - 1:1 relationship

Branch

| branchNo | bAddress |
|---|---|
| B005 | 22 Deer Rd, London |
| B007 | 16 Argyll St, Aberdeen |
| B003 | 163 Main St, Glasgow |

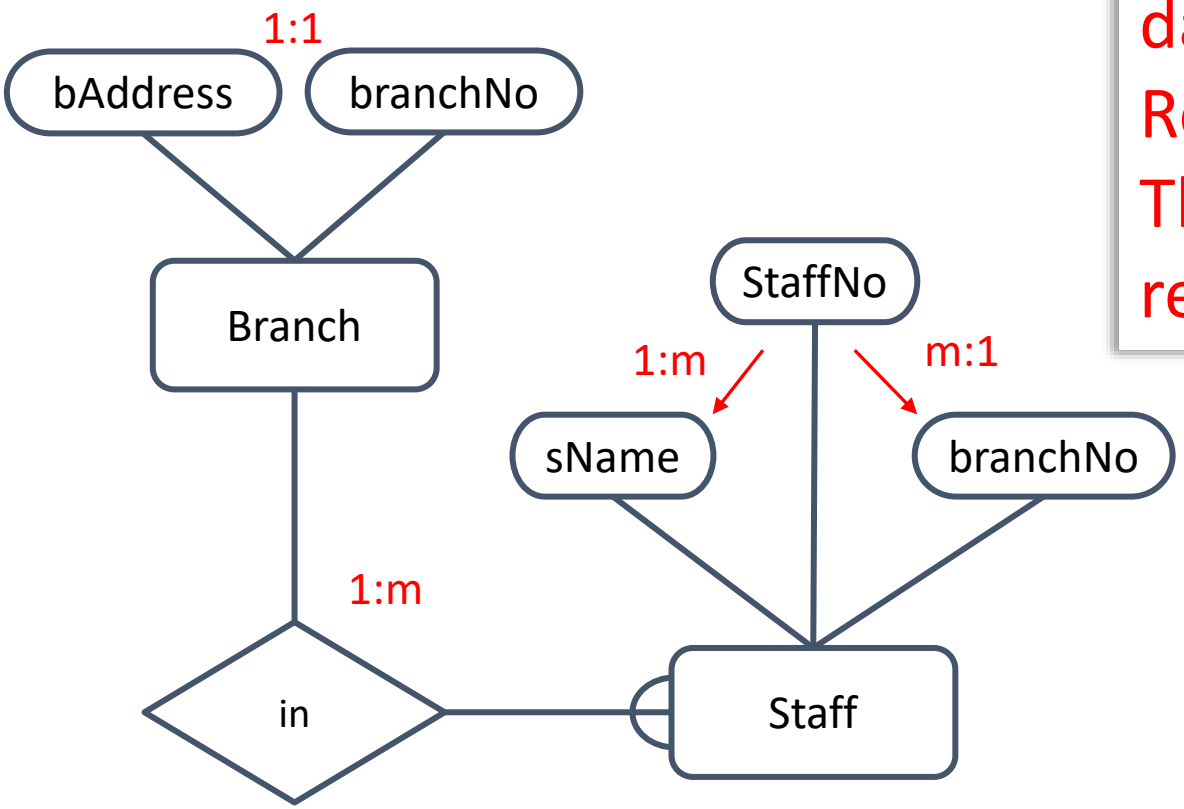
# More about Determinants

"The determinant has a M:1 or 1:1 relationship with other attributes in FD"

- Remember in ER Modelling, M:1 relationships between tables will become foreign keys.

- We can infer that, For attributes that have a M:1 relationships, if they belong to the same context, they will be grouped into one relation, otherwise, they can be split into two tables and linked with a foreign key.

"If they belong to the same context, they will be grouped into one relation, otherwise, they will be split into two tables and linked with a foreign key"




That's why these databases are called Relational databases: They are really about relationships! ☺

# Transitive Dependency

1. staffNo → sName, position, salary, branchNo, bAddress
2. branchNo → bAddress

Observe this dependency chain carefully:

- staffNo → branchNo → bAddress
  - This looks like a relationship between staff and branch.

Staff Branch

| staffNo | sName | position | salary | branchNo | bAddress |
|---|---|---|---|---|---|
| SL21 | John White | Manager | 30000 | B005 | 22 Deer Rd, London |
| SG37 | Ann Beech | Assistant | 12000 | B003 | 163 Main St, Glasgow |
| SG14 | David Ford | Supervisor | 18000 | B003 | 163 Main St, Glasgow |
| SA9 | Mary Howe | Assistant | 9000 | B007 | 16 Argyll St, Aberdeen |
| SG5 | Susan Brand | Manager | 24000 | B003 | 163 Main St, Glasgow |
| SL41 | Julie Lee | Assistant | 9000 | B005 | 22 Deer Rd, London |

# Transitive Dependency

The previous example is a **transitive dependency**.

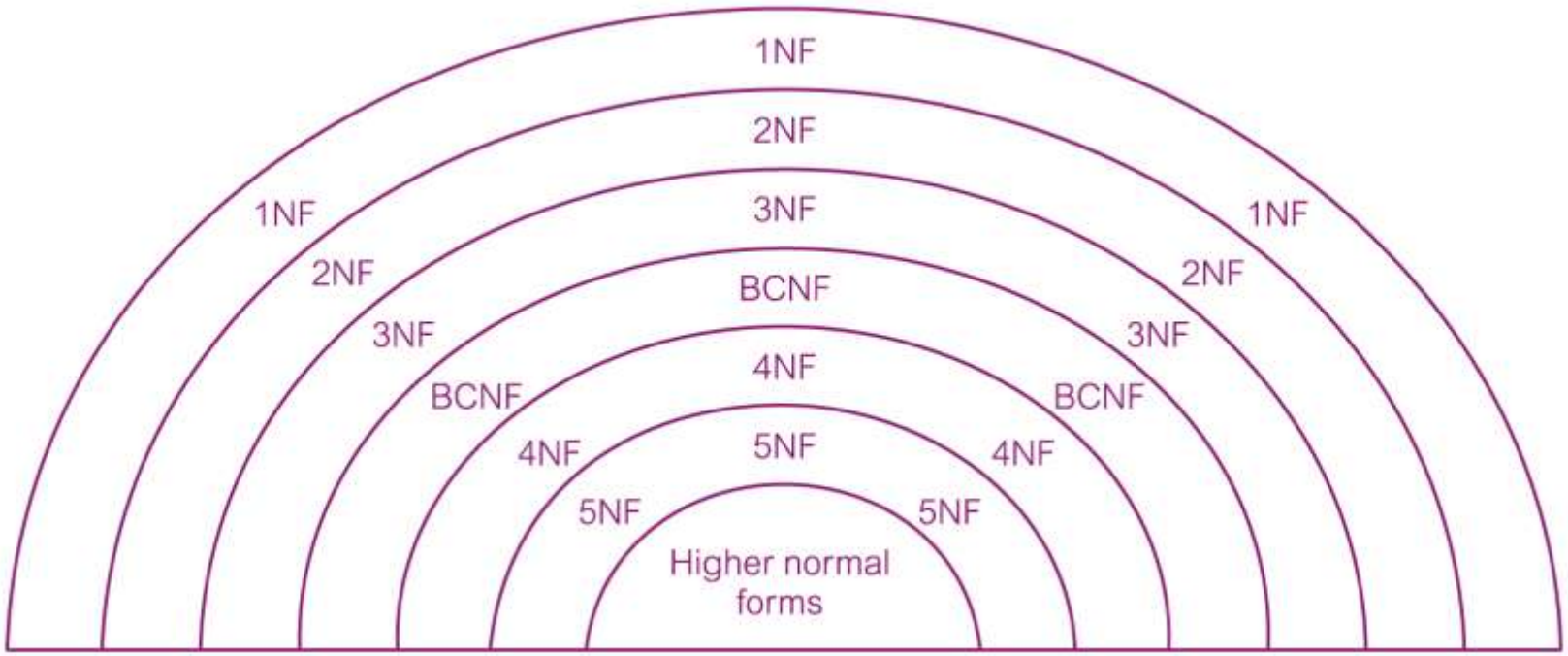
"Transitive dependency describes a condition where A, B, and C are attributes of a relation such that if A → B and B → C, then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C)."

- Why is the underlined constraint needed? (Explained later)

# Normal Forms

How can we use FDs and TDs to redesign tables

# The Process of Normalisation




Steps of redesigning tables to make them “suitable”

# Normalisation: Some Notes

- In relational database, we always try to assign tables with primary keys (or at least candidate keys).

- Why? Because relational databases are about relations.
  - Attributes within one context forms an entity (table).
  - Primary keys make referencing possible.
  - Attributes connecting entities become foreign keys.
  - Foreign keys form these connections.

- Can I insist that no unique keys are used?
  - Then its beyond the topic of this module :-)
  - Remember, we are learning **one of the possible design decisions** of managing data, don't limit your imagination.

# First Normal Form

- In most definitions of the relational model
  - All data values should be atomic
  - This means that table entries should be single values, not sets or composite objects
- A relation is said to be in **first normal form (1NF)**:
  - if all data values are atomic
  - Otherwise, the relation is in unnormalised Form (UNF)

Unnormalised relation

| Module | Dept | Lecturer | Texts |
|---|---|---|---|
| M1 | D1 | L1 | T1, T2 |
| M2 | D1 | L1 | T1, T3 |
| M3 | D1 | L2 | T4 |
| M4 | D2 | L3 | T1, T5 |
| M5 | D2 | L4 | T6 |

# Problems With UNF

Look at “Texts” attribute:

- To update the textbook name “T1” to something else, you need to manually update all “T1”s.
- To delete T1 from the pool of textbooks, you need to manually do so, too.
- You cannot add a textbook without giving the information of Module (because it’s a Primary Key).

Unnormalised relation

| Module | Dept | Lecturer | Texts |
|---|---|---|---|
| M1 | D1 | L1 | T1, T2 |
| M2 | D1 | L1 | T1, T3 |
| M3 | D1 | L2 | T4 |
| M4 | D2 | L3 | T1, T5 |
| M5 | D2 | L4 | T6 |

# Normalise to 1NF

Method 1:

1. Remove the repeating group by entering appropriate data into the empty columns of rows containing the repeating data (also called 'flattening' the table).
2. Assign a new primary/unique key to the new table.

Unnormalised relation

| Module | Dept | Lecturer | Texts |
|---|---|---|---|
| M1 | D1 | L1 | T1, T2 |
| M2 | D1 | L1 | T1, T3 |
| M3 | D1 | L2 | T4 |
| M4 | D2 | L3 | T1, T5 |
| M5 | D2 | L4 | T6 |

Primary key

1NF

| Module | Dept | Lecturer | Text |
|---|---|---|---|
| M1 | D1 | L1 | T1 |
| M1 | D1 | L1 | T2 |
| M2 | D1 | L1 | T1 |
| M2 | D1 | L1 | T3 |
| M3 | D1 | L2 | T4 |
| M4 | D2 | L3 | T1 |
| M4 | D2 | L3 | T5 |
| M5 | D2 | L4 | T6 |

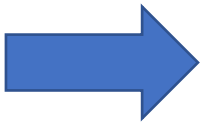
# Normalise to 1NF

Method 2:

1. Identify primary/unique key.
2. Place the repeating data along with a copy of the original (unique) key attribute(s) into a separate relation.

Unnormalised relation

| Module | Dept | Lecturer | Texts |
|---|---|---|---|
| M1 | D1 | L1 | T1, T2 |
| M2 | D1 | L1 | T1, T3 |
| M3 | D1 | L2 | T4 |
| M4 | D2 | L3 | T1, T5 |
| M5 | D2 | L4 | T6 |



Primary key

1NFa

| Module | Texts |
|---|---|
| M1 | T1 |
| M1 | T2 |

1NFb

| Module | Dept | Lecturer |
|---|---|---|
| M1 | D1 | L1 |
| M2 | D1 | L1 |
| M3 | D1 | L2 |
| … | … | … |

Primary key

# Problems in 1NF

- Changing module code from “M1” to something else requires you to check the whole table.
- Adding a new lecturer with no modules and text is impossible.
- If a (department, lecturer) pair is modified, the change must be made to all (Module, Text) pairs.
  - For example, to change (D1, L1) to some other value, you must manually change the first four rows.
- If (M3, T4) is deleted, L2 will be permanently lost!

1NF

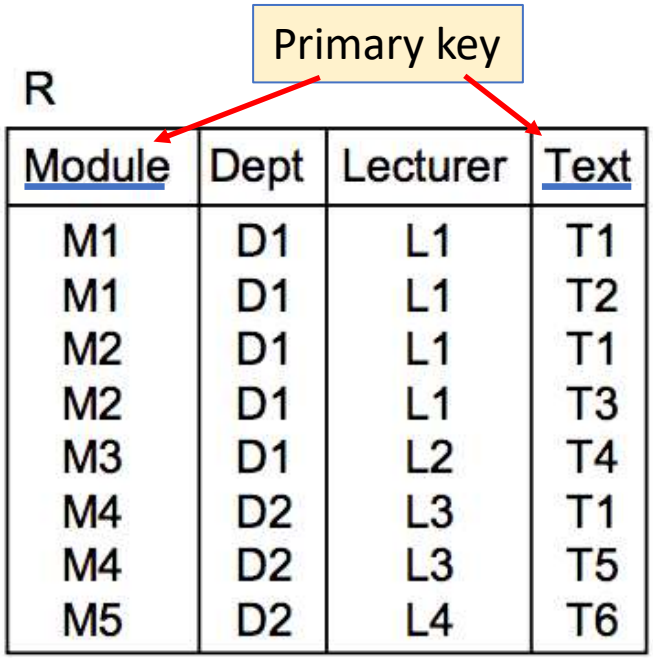
| Module | Dept | Lecturer | Text |
|---|---|---|---|
| M1 | D1 | L1 | T1 |
| M1 | D1 | L1 | T2 |
| M2 | D1 | L1 | T1 |
| M2 | D1 | L1 | T3 |
| M3 | D1 | L2 | T4 |
| M4 | D2 | L3 | T1 |
| M4 | D2 | L3 | T5 |
| M5 | D2 | L4 | T6 |

# Second Normal Form

- A relation is in second normal form (2NF) if it is in 1NF and no non-key attribute is partially dependent **on the primary key**

- In other words, no C → B where C is a strict subset of a primary key and B is a non-key attribute.

# Second Normal Form: Example




- We have the FD:

{Module, Text} →{Lecturer, Dept}

- But also

{Module} → {Lecturer, Dept}

- And so Lecturer and Dept are partially dependent on the primary key
- R is in 1NF but not in 2NF

# 1NF to 2NF

- Identify the primary key(s) for the 1NF relation.
- Identify the functional dependencies in the relation.
- If partial dependencies exist on the primary key, remove them by placing attributes of the corresponding full FD in a new relation and leave the its determinant in the original table.

1NF

| Module | Dept | Lecturer | Text |
|---|---|---|---|
| M1 | D1 | L1 | T1 |
| M1 | D1 | L1 | T2 |
| M2 | D1 | L1 | T1 |
| M2 | D1 | L1 | T3 |
| M3 | D1 | L2 | T4 |
| M4 | D2 | L3 | T1 |

2NFa

| Module | Dept | Lecturer |
|---|---|---|
| M1 | D1 | L1 |
| M2 | D1 | L1 |
| M3 | D1 | L2 |
| M4 | D2 | L3 |
| M5 | D2 | L4 |

2NFb

| Module | Text |
|---|---|
| M1 | T1 |
| M1 | T2 |
| M2 | T1 |
| M2 | T3 |
| M3 | T4 |
| M4 | T1 |
| M4 | T5 |
| M1 | T6 |

{Module, Text} → {Lecturer, Dept}
{Module} → {Lecturer, Dept}

# Problems in 2NF

Look at the table 2NFa

- We cannot add a lecturer who is not assigned with any module. Because module is the primary key.
  - This is a theoretical discussion. Similar issues can be a real problem in some other tables.
- By deleting M3, we lose L2 forever.
- To change the department for L1, we need to change multiple rows manually

2NFa

| Module | Dept | Lecturer |
|---|---|---|
| M1 | D1 | L1 |
| M2 | D1 | L1 |
| M3 | D1 | L2 |
| M4 | D2 | L3 |
| M5 | D2 | L4 |

# Third Normal Form (3NF)

- Based on the concept of transitive dependency.
- A relation that is in 1NF and 2NF and in which no non-key attribute is transitively dependent on the primary key.

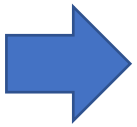
# 2NF to 3NF

- Identify the primary key in the 2NF relation.
- Identify functional dependencies in the relation.
- If transitive dependencies exist on the primary key, remove them by placing them in a new relation along with a copy of their determinant.

2NFa

| Module | Dept | Lecturer |
|---|---|---|
| M1 | D1 | L1 |
| M2 | D1 | L1 |
| M3 | D1 | L2 |
| M4 | D2 | L3 |

{Module} → {Lecturer} → {Dept}



3NFa

| Lecturer | Dept |
|---|---|
| L1 | D1 |
| L2 | D1 |
| L3 | D2 |
| L4 | D2 |

3NFb

| Module | Lecturer |
|---|---|
| M1 | L1 |
| M2 | L1 |
| M3 | L2 |
| M4 | L3 |
| M5 | L4 |

# Problems Resolved in 3NF

- Problems in 2NF
  - INSERT: Can't add lecturers who teach no modules
  - UPDATE: To change the department for L1 we must alter two rows
  - DELETE: If we delete M3 we delete L2 as well
- In 3NF all of these are resolved (for this relation – but 3NF can still have anomalies!)

3NFa

| Lecturer | Dept |
|---|---|
| L1 | D1 |
| L2 | D1 |
| L3 | D2 |
| L4 | D2 |

3NFb

| Module | Lecturer |
|---|---|
| M1 | L1 |
| M2 | L1 |
| M3 | L2 |
| M4 | L3 |
| M5 | L4 |

# Transitive Dependency and FK

Let's go back and check the definition of the transitive dependency:

"Transitive dependency describes a condition where A, B, and C are attributes of a relation such that if A → B and B → C, then C is transitively dependent on A via B. Provided that A is not functionally dependent on B or C"

- Translation:

"Provided that B → A or C → A is not true"

# Transitive Dependency

Example:

- staffNo (A) → branchNo (B) → bAddress (C)
  - This is transitive dependency.
- LecID (A) → LEmail (B) → LName (C)
  - Not a transitive dependency. Because LEmail is functionally dependent on LecID.

- Why the second example should not be considered as a transitive dependency (should not be split into two tables)?

# Transitive Dependency and FK

- LEmail (B) → LecID (A)
  - For each Email address, you can find one LecID.
  - Below are valid values of email and id:
    - Oyo@abc.com → 001
    - Yoy@abc.com → 001

- But LecID is a candidate key, it is not allowed to have duplicates.
  - Thus, for each Email address, you can find a **unique** LecID.

- Thus LEmail and LecID has a 1:1 relationship.

# Transitive Dependency and FK

- After splitting them:

Table1: LecID, LEmail

Table2: LEmail, LName

- Table1 and Table2 has 1:1 relationship, with LEmail being the foreign key.
- You have learned E/R modelling, this is not worth splitting.

# Transitive Dependency and FK

LecID (A) → LEmail (B) → LName (C)

- The confusion comes from the name “transitive”.
- The above dependency chain is indeed “transitive”.
- But it is not the type of “transitive” we are looking for.

- The type of transitive dependencies we are looking for should support **relationships, or foreign keys**.

# Normalisation: Practice

Example process of normalising a database

# Normalisation Example

- We have a table representing orders in an online store, Each entry in the table represents an item on a particular order.

  {order, product, customer, address, quantity, unitPrice}

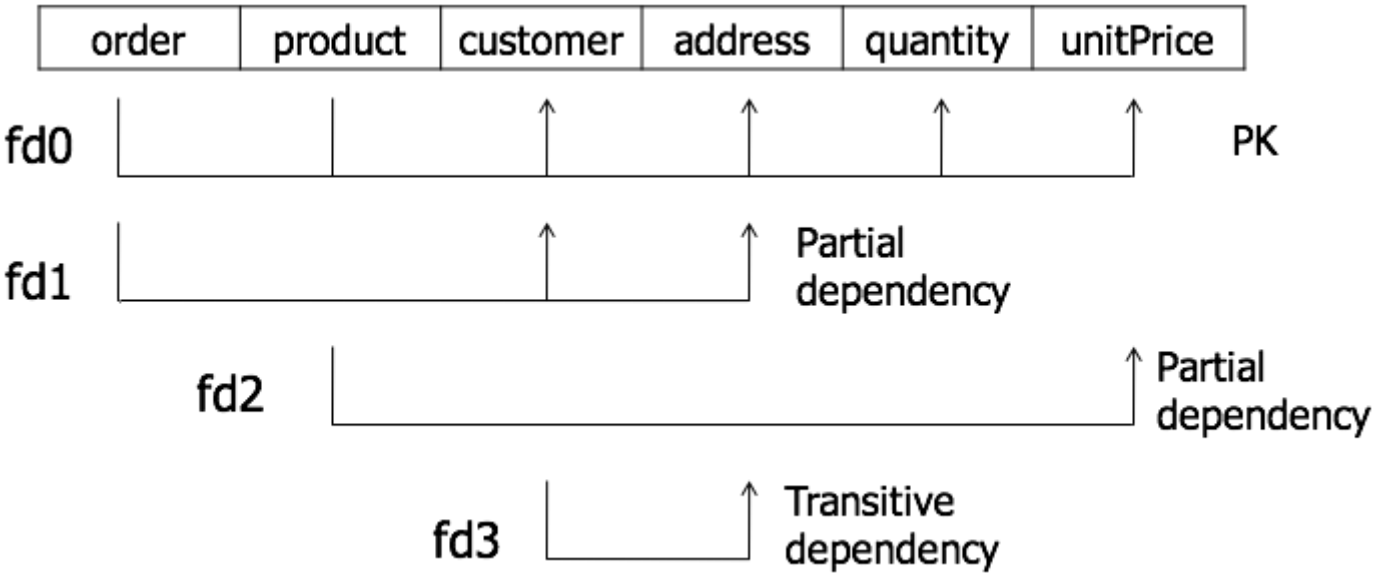
  - For example: (001, Laptop, Jianjun, SEB435 UNNC, 1, $500)

- Primary key is {order, product}
- No other candidate key
- Task: Normalise it to 3NF.

# Functional Dependencies

- Each order is for a single customer and each customer has a single address.
  - FD1: {order} →{customer, address}
  - FD2: see FD 4 below
- Each product has a single price
  - FD3: {product} →{unitPrice}
- Each order transitively determines address via customer.
  - FD4: {order} → {customer} → {address}

# Functional Dependencies



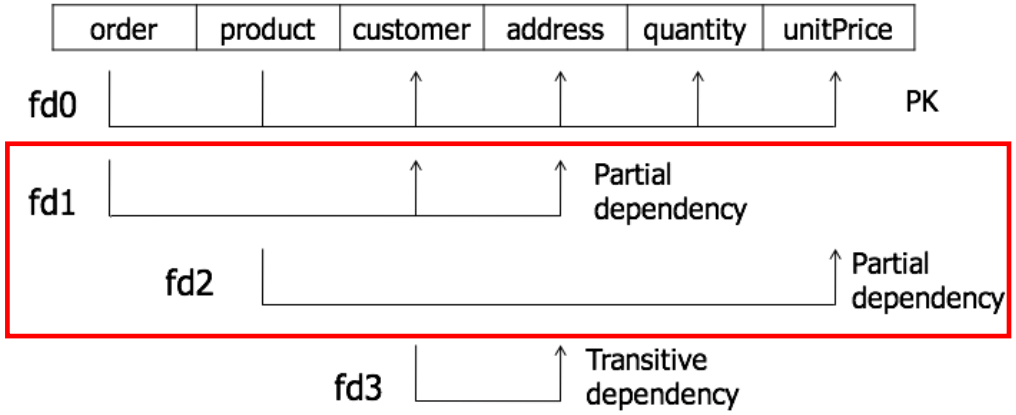
# Normalisation to 2NF

2NF: no partial dependencies on candidate keys

{order} → {customer, address}

{product} → {unitPrice}

- To remove the first FD we move {customer, address} to another relation, along with a copy of its determinant, *order*.
  - R1: {<u>order</u>, customer, address}
- With remaining relation
  - R2: {<u>order</u>, <u>product</u>, quantity, unitPrice}



# Normalisation to 2NF

Current Tables

R1: {order, customer, address}  2NF

R2: {order, product, quantity, unitPrice}

Next:

- There is a partial FD in R2: {Product} → {UnitPrice}
- To remove this we move it to another relation R3
  - R3: {Product, unitPrice}
- and with remaining relation
  - R4: {order, product,  quantity}

# Normalisation to 3NF

Current Tables:

R1: {order, customer, address}

R3: {Product, unitPrice} 3NF

R4: {order, product, quantity} 3NF

Next:

- R1 has a transitive FD on its key

- To remove {order} → {customer} → {Address}
- we decompose R1 over
  - R5: {order, customer}
  - R6: {customer, address}

# Normalisation

- 1NF:
  - {order, product, customer, address, quantity, unitPrice}
- 2NF:
  - R1: {order, customer, address} 2NF
  - R3: {product, unitPrice} 3NF
  - R4: {order, product, quantity} 3NF
- 3NF:
  - R3: {product, unitPrice} 3NF
  - R4: {order, product, quantity}, 3NF
  - R5: {order, customer} 3NF
  - R6: {customer, address} 3NF

# SQL Support

CREATE TABLE AS SELECT …

INSERT INTO SELECT …

# SQL Support for Normalisation

- SQL provides sufficient support for normalisations.
- If tables have not been created, you can use CREATE TABLE to build a database that matches your normalisation result.
- If you need to apply normalisation to an existing database with data in its tables, you can do the following:
  - Create new tables that match the results of normalisation.
  - Copy data to these new tables from old tables, using:

```
INSERT INTO SELECT …
```

# INSERT INTO SELECT …

- For example, to split the StaffBranch table into staff and branch:
  - CREATE TABLE branch ( … );
  - CREATE TABLE staff ( … );
  - INSERT INTO branch SELECT DISTINCT branchno, baddress FROM staffbranch;
  - INSERT INTO staff SELECT staffno, …, branchno FROM staffbranch;

StaffBranch

| staffNo | sName | position | salary | branchNo | bAddress |
|---|---|---|---|---|---|
| SL21 | John White | Manager | 30000 | B005 | 22 Deer Rd, London |
| SG37 | Ann Beech | Assistant | 12000 | B003 | 163 Main St, Glasgow |
| SG14 | David Ford | Supervisor | 18000 | B003 | 163 Main St, Glasgow |
| SA9 | Mary Howe | Assistant | 9000 | B007 | 16 Argyll St, Aberdeen |
| SG5 | Susan Brand | Manager | 24000 | B003 | 163 Main St, Glasgow |
| SL41 | Julie Lee | Assistant | 9000 | B005 | 22 Deer Rd, London |

# CREATE TABLE AS SELECT …

- You may also create staff and branch using the results of SELECT directly.
  - CREATE TABLE branch SELECT DISTINCT branchno, baddress FROM staffbranch
  - CREATE TABLE staff SELECT staffno, …, branchno FROM staffbranch.

- Remember to create the primary keys and foreign keys manually as they are not automatically created.