INFO20003 Tutorial 9

Starting ~2.20 pm

Today's tutorial

- Review of Normalisation Concepts
- Exercises
  - group work

Review of Normalisation Concepts (15 mins)

Key Concepts:

NOTE for students: *This is a brief summary of some of the concepts taught in lectures. The lectures contain detailed content related to these and many more concepts. These notes should be considered quick revision instead of a sole resource for the course material.*

- Anomalies

Consider the following instance of the relation Allocation:

CourseNumber	Tutor	Room	Seats
INFO20003	Farah	Alice Hoy 109	30
~~COMP10001~~	Farah	EDS 6	25
INFO30005	~~Patrick~~	~~Sidney Myer G09~~	~~20~~ 30
COMP20005	Alan	Sidney Myer G09	~~20~~ 30

- Functional dependency
- Determinants
- Key and non-key attributes
- Partial functional dependency
- Transitive functional dependency
- Armstrong's Axioms
- Normalisation and normal forms

Anomalies

- Update
- Deletion
- Insertion
- ***Look at the table - what anomalies can we identify?***

An **update** anomaly is a data inconsistency that results from data redundancy and partial update when one or more instances of duplicated data are updated but not all.

- For example, suppose the room Sidney Myer G09 has been improved, and there are now 30 seats. In this single entity, we will have to update all other rows where room = Sidney Myer G09.
- Lots of other examples, e.g. changing a tutor name (Farah -> Farah Khan)

A **deletion** anomaly is an unintentional loss of certain attribute values due to the deletion of other data for other attributes.
- If we remove COMP10001 from the above table, the details of room EDS 6 are also deleted.
- Or if we remove the tutor Patrick, we also lose all the details of the subject INFO30005

An **insertion** anomaly is the inability to add certain attributes to a database due to absence of other attributes.
- For example, a new room “NewAlice109” has been built but has not yet been timetabled for any course or staff members.
- Can't add a course without any tutors or a room

If the relation is in **denormalised** form, it can lead to the above-mentioned **anomalies**.

What is normalisation?

Normalisation helps us remove such anomalies and get rid of redundant data by iteratively improving relations.

- This requires understanding of the functional dependencies of the attributes of a given relation and resolving transitive and partial dependencies to achieve normal forms

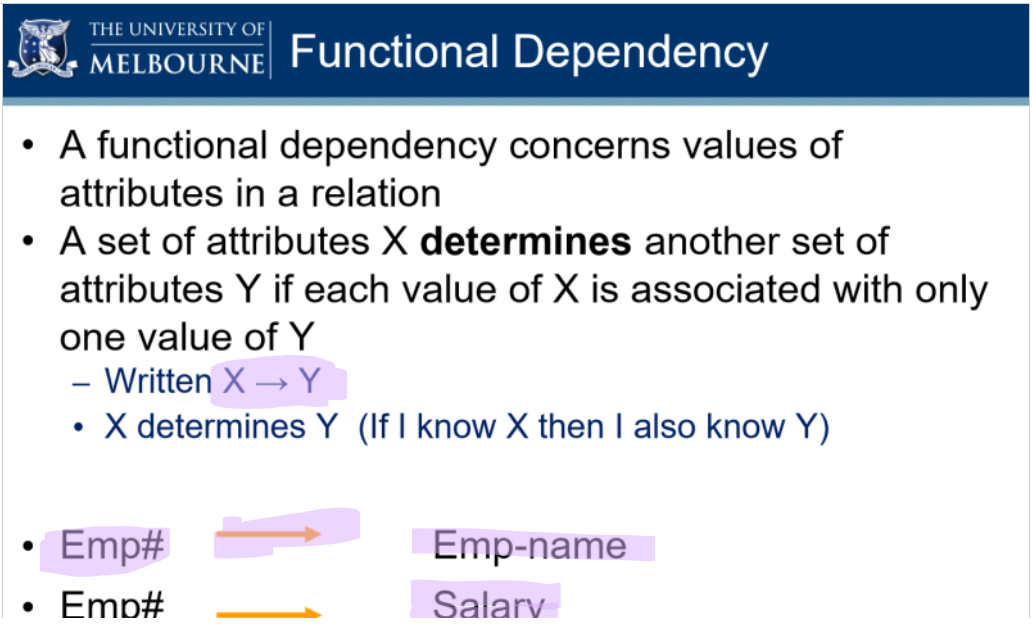
Functional dependency

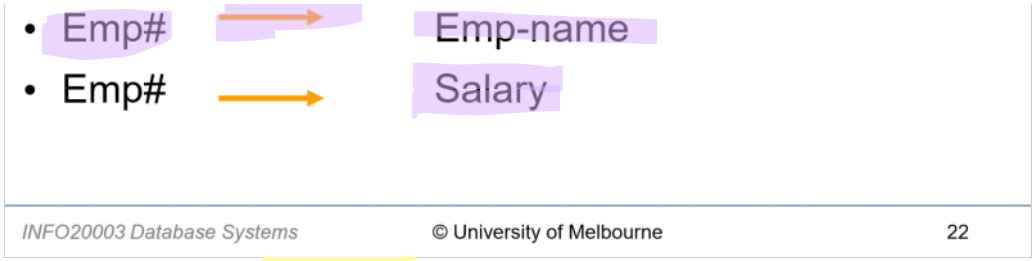
- Functional dependency

For a particular relation R, a functional dependency occurs when a subset of R's attributes $\{A_1, A_2, ..., A_n\}$ determine attributes $\{B_1, B_2, \ldots, B_n\}$. If several tuples agree on the values of $A_1, A_2, ..., A_n$, they must agree on the values of $B_1, B_2, \ldots, B_n$ – that is, if two records have the same $A_1, A_2, ..., A_n$ then they have the same $B_1, B_2, \ldots, B_n$. Therefore $A_1, A_2, ..., A_n$ "functionally determine" $B_1, B_2, \ldots, B_n$. This is written as:

$$\mathbf{A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_n}$$

A relation R satisfies a functional dependency (FD) if and only if the FD is true for every instance of R.





- Read arrow as "determines"

Determinants

- The attributes that determine the value of other attributes are called determinants.
  - These are the attributes on the **LHS** of the arrow

- For example,
  - Consider the relation below:

    Person (ssn, name, birthdate, address, age)

    birthdate → age
    ssn → name, birthdate, age, address

In this example, birthdate and ssn are determinants, as birthdate determines age and ssn determines the rest of the attributes.

Key and non-key attributes

Key attributes = Attributes that are part of the **PK**

Non-key attributes = Attributes which aren't part of PK

A **key** is a set of attributes {A1, A2, …, An} for a relation R, such that {A1, A2, ..., An} functionally **determines all other attributes** of R and no subset of {A1, A2, …, An} functionally determines all other attributes of R. The key must be **minimal**.

In the relation Person (ssn, name, birthdate, address, age), ***ssn* is the minimal key** of the Person relation, but *{ssn, name}* is **not** (it is a “super key”).

Partial functional dependency

A partial functional dependency arises when one or more **non-key** attributes are (functionally) determined by a **subset** of the primary key.

For example:

- A(X, Y, Z, D)

- Y -> Z is a partial functional dependency

- Z only depends on a part (not all) of the PK
  - (X, Y -> Z would be okay)

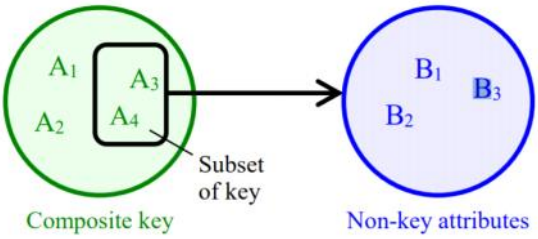


Figure 1: A partial functional dependency (subset of composite key determining some non-key attributes)

For example, in the relation **Order (<u>Order#, Item#,</u> Desc, Qty)** from the lecture,
Order# and Item# are the keys.

Nonetheless, as **Item# -> Desc**
- The item description, Desc, can be determined by Item# alone
- This is a partial functional dependency

Can you have a partial functional dependency if there is only 1 PK attribute?

Transitive functional dependency
When a **non-key** attribute is determined by another **non-key** attribute (or by a subset of PK and non-key attributes), such a dependency is called a transitive functional dependency.

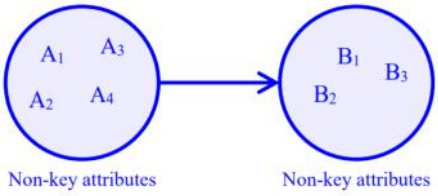


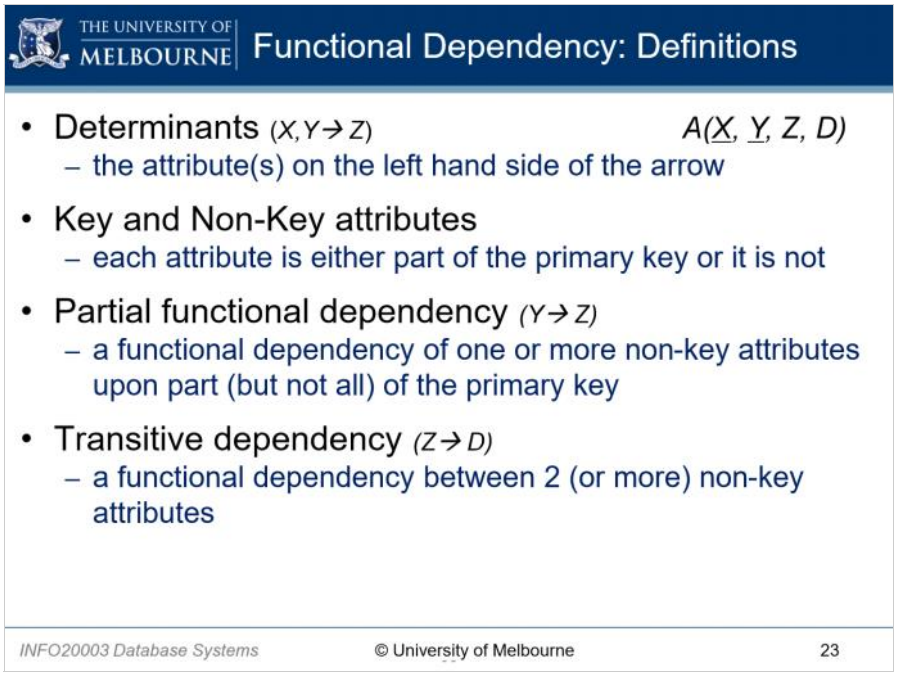
Figure 2: A transitive functional dependency (non-key attributes determining other non-key attributes)

For example:
- A(<u>X, Y</u>, Z, D)

- Z -> D is a transitive functional dependency

- Z and D are both non-key attributes

Summary from lectures:



Armstrong's Axioms

- When you are given a list of FDs, you can **discover** more FDs by applying these axioms

- Some of the FDs you discover might be **useful** in the normalisation process

- **Sometimes** these axioms **aren't useful** at all but **sometimes** they **can be** - you will mostly use the **transitivity axiom**

Given a relation and a set of functional dependencies (FDs), we can **discover** new functional dependencies using some rules generally known as Armstrong's Axioms.

Three important axioms required to determine FDs are Reflexivity (also known as "trivial FDs"), Augmentation and Transitivity.

Reflexivity:

Suppose A = {A1, A2, ..., An} is a subset of attributes of R, and B = {B1, B2, …, Bn} is also a subset of attributes of R such that B is a subset of A.

Reflexivity implies that
$B \subseteq A \Longrightarrow A \rightarrow B$

If B is a subset of A, then can say A determines B
Any subset is determined by a bigger set

For example, in the case of Person (ssn, name, birthdate, address, age),
ssn, name → name

Obviously, if you know both ssn and name, you know the name!

Augmentation:

Consider another subset of attributes C = {C1, C2, …, Cn}.
Augmentation implies that
$A \rightarrow B \Longrightarrow AC \rightarrow BC$

If A determines B, then we can say that A and C together (bigger set) determines B and C (bigger set)

(Basically add C to both sides)

In the case of Person (ssn, name, birthdate, address, age),
we can take the above FD
ssn, name → name

and write
ssn, name, age → name, age

Transitivity:
A → B and B → C ⟹ A → C

→ = "determines"
⟹ = "implies"

For our relation Person,
we can say ssn → birthdate, birthdate → age ⟹ ssn → age.

** This will be the axiom you'll use the most to solve normalisation questions

Normalisation and normal forms

Normalisation is a technique used to iteratively improve relations to **remove undesired redundancy** by decomposing relations and eliminating anomalies.

- Want to **reduce data duplication / redundancy** (storing the same piece of data in many places)
- Allows users to **insert, update and delete** rows **without anomalies**
- **Break one large table into smaller tables**

The process is iterative and can be performed in stages generally referred to as **Normal Forms.**

In First Normal Form (**1NF**),
the relation is analysed and all repeating groups are identified to be decomposed into new relations.

(**Remove** repeating groups and cells which store multiple values)

In Second Normal Form (**2NF**),
all the **partial** dependencies are resolved/removed.

The next stage is Third Normal Form (**3NF**)
where all the **transitive** dependencies are removed.

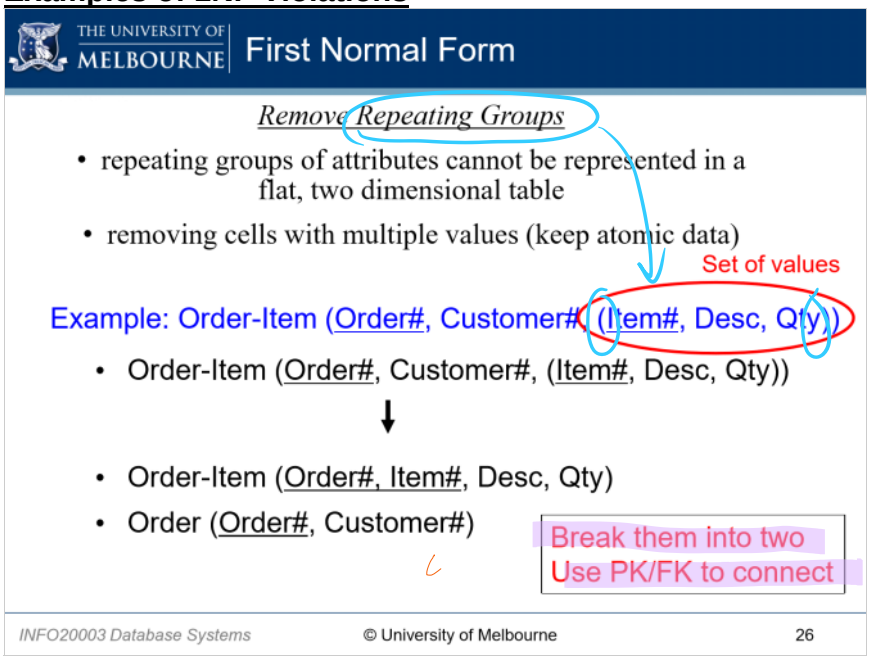
If I ask you to put something into 3NF, what do you need to do?

Remember!

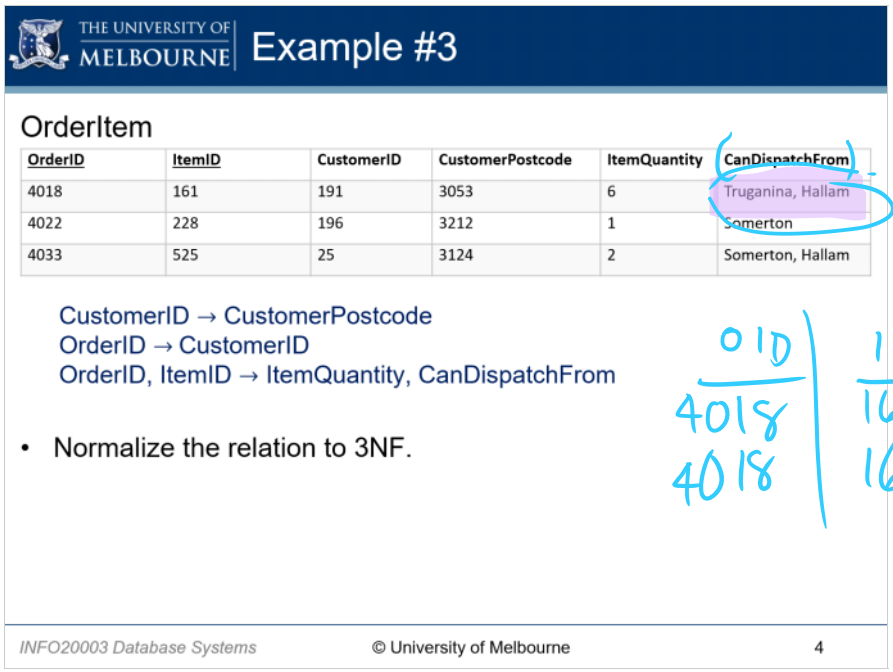
Before you try putting a relation into 3NF, first check that 1NF and 2NF is satisfied.

Same with trying to put into 2NF (first check 1NF is satisfied)

Examples of 1NF Violations



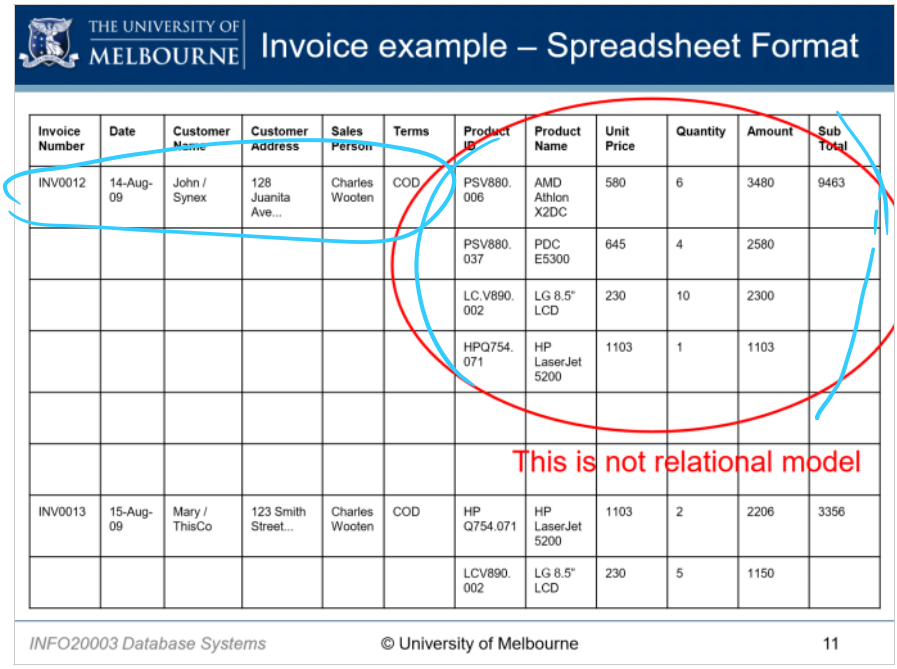
- Extra pair of brackets indicates repeating groups



- *What is the issue here? How do we fix this?*

Need to add CanDispatchFrom to the PK!

Location(**OrderID, Item ID, CanDispatchFrom**)
OrderItem(**OrderID, ItemID,** custID, CustCode, ItemQuantity)



Invoice Number	Date	Customer Name	Customer Address	Sales Person	Terms	Product ID	Product Name	Unit Price	Quantity	Amount	Sub Total
INV0012	14-Aug-09	John / Synex	128 Juanita Ave...	Charles Wooten	COD	PSV880.006	AMD Athlon X2DC	580	6	3480	9463
						PSV880.037	PDC E5300	645	4	2580	
						LC.V890.002	LG 8.5" LCD	230	10	2300	
						HPQ754.071	HP LaserJet 5200	1103	1	1103	
INV0013	15-Aug-09	Mary / ThisCo	123 Smith Street...	Charles Wooten	COD	HP Q754.071	HP LaserJet 5200	1103	2	2206	3356
						LCV890.002	LG 8.5" LCD	230	5	1150	

Exercise 1 (Do together)

Exercises:

1. Consider the relation Diagnosis with the schema Diagnosis (DoctorID, DocName, PatientID, DiagnosisClass) and the following functional dependencies:

   DoctorID → DocName

   DoctorID, PatientID → DiagnosisClass

   Consider the following instance of Diagnosis:

DoctorID	DocName	PatientID	DiagnosisClass
D001	Alicia	P888	Flu
D002	John Miller	P999	Lactose intolerance
D003	Jennifer	P000	Flu
D002	John	P111	Fever

   Identify different anomalies that can arise from this schema using the above instance.

From the two given FDs, we can infer that the key for Diagnosis is (DoctorID, PatientID) since together they are sufficient to uniquely identify each record.

The following anomalies can arise from the given schema:

Insertion anomaly: If we require inserting data for a **new doctor like DoctorID** and DocName, we must insert data of **at least one patient** associated with the doctor. This inability to insert records of particular fields is insertion anomaly.

Deletion anomaly: Deleting patient's data can result in the loss of doctor's data as well

resulting in deletion anomaly. For example, **if we delete P888 data** from the table we **lose** record for the **doctor** named **Alicia** as well.

Update anomaly: One doctor may be associated with more than one patient. In such case, an update anomaly may result if a doctor's name is changed for only one patient. For example, if we want to change the doctor's name from **"John" to "John Miller",** we have to do it for two records. Failing to do so will result in an update anomaly.

1NF 3NF: R1 (A, C, FK B)

Exercise 2 & 3: Group work

2NF R2 (B, D)

2. Consider a relation R (A, B, C, D) with the following FDs:

$$AB \rightarrow C,\ AC \rightarrow B,\ BC \rightarrow A,\ B \rightarrow D$$

The possible candidate keys of R are AB, AC, and BC, since each of those combinations is sufficient to uniquely identify each record. Let's consider AB for instance. From AB → C we see that AB uniquely identifies C, and since B alone uniquely identifies D, AB together have covered CD, i.e. the entire set of attributes.

AB

FK

R1(A, B, C)

R2 (B, D)

List all the functional dependencies that violate 3NF. If any, decompose R accordingly. After decomposition, check if the resulting relations are in 3NF, if not decompose further.

3. Consider the following relation StaffPropertyInspection:

StaffPropertyInspection (propertyNo, pAddress, iDate, iTime, comments, staffNo, sName)

The FDs stated below hold for this relation:

propertyNo, iDate → iTime, comments, staffNo, sName
propertyNo → pAddress
staffNo → sName

From these FDs, it is safe to assume that propertyNo and iDate can serve as a primary key. Your task is to normalise this relation to 3NF. Remember in order to achieve 3NF, you first need to achieve 1NF and 2NF.

StaffPropInspection(propNo, pAddress, iDate, iTime, comments, staffNo, sName)

2. Consider a relation R (A, B, C, D) with the following FDs:

$$AB \rightarrow C,\ AC \rightarrow B,\ BC \rightarrow A,\ B \rightarrow D$$

The possible candidate keys of R are AB, AC, and BC, since each of those combinations is sufficient to uniquely identify each record. Let's consider AB for instance. From AB → C we see that AB uniquely identifies C, and since B alone uniquely identifies D, AB together have covered CD, i.e. the entire set of attributes.

List all the functional dependencies that violate 3NF. If any, decompose R accordingly. After decomposition, check if the resulting relations are in 3NF, if not decompose further.

To be in 3NF, a relation should be in 2NF and all the transitive functional dependencies should be removed.

The relation is not in 2NF as there is a partial functional dependency B → D.

B is part of a composite candidate key (AB and BC) and does not require any other key attribute to determine D.
Hence, we need to decompose R into following relations:

R1 (<u>A, B FK</u>, C) and R2 (<u>B</u>, D).

R1 and R2 don't have any partial or transitive functional dependencies hence both are in 3NF.

Notice that whether something is a partial functional dependency depends on which attributes you choose as PK!

$AB \rightarrow C,\ AC \rightarrow B,\ BC \rightarrow A,$

A lot of these other functional dependencies you can ignore
- These are not violations of 2NF or 3NF

- 2NF violated if a non-key attribute is determined by part of PK
  - The non-key attributes if you choose AB as PK are C and D
  - These are the attributes on we want on RHS of arrow
  - AB -> C is perfectly fine
  - and D doesn't appear on RHS at all in this list

- 3NF violated if non-key attribute is determined by a non-key attribute
  - This also never happens
  - Requires non-key attributes (C, D) on RHS
  - and non-key on LHS as well

- So we don't need to care about the consequences of these 3 functional dependencies - can ignore them!

Other solutions are also possible:
R1 (<u>A, C,</u> B FK) and R2(<u>B</u>,D)
- If AC = PK, then this is 3NF violation (transtiive dependency)

If BC = PK, R1(B FK, C, A) and R2(B, D)

3. Consider the following relation StaffPropertyInspection:

StaffPropertyInspection (propertyNo, pAddress, iDate, iTime, comments, staffNo, sName)

The FDs stated below hold for this relation:

propertyNo, iDate → iTime, comments, staffNo, sName
propertyNo → pAddress
staffNo → sName

From these FDs, it is safe to assume that propertyNo and iDate can serve as a primary key. Your task is to normalise this relation to 3NF. Remember in order to achieve 3NF, you first need to achieve 1NF and 2NF.

As the relation is already in 1NF, because there are no repeating groups, we can directly check if it is in 2NF or not.

Second Normal Form: The functional dependency propertyNo → pAddress shows that pAddress is determined by part of the key. Hence, the relation is not in 2NF. It can be decomposed as:

2 NF

Property (propertyNo, pAddress)

FK
PropertyInspection (propertyNo, iDate, iTime, comments, staffNo, sName)

Third Normal Form: Now out of the two relations, Property is already in 3NF since there is no transitive dependency in this relation. Regarding PropertyInspection, we can observe that the FD staffNo → sName violates 3NF, as it is a transitive dependency. We further decompose this relation to:

3 NF

Property (propertyNo, pAddress)

Staff (staffNo, sName)

FK FK
PropertyInspection (propertyNo, iDate, iTime, comments, staffNo)

- Remember to write in "FK" if your attribute refers to the PK of another table
- Tip!
  - First write in the attributes of your table, then have a look at all the tables and attributes and decide on a good name for the table

StaffPropInspection(propNo, pAddress, iDate, iTime, comments, staffNo, sName)

1NF is achieved as no repeating groups.

Not in 2NF as propNo -> propAddress
StaffPropInspection(propNo FK, iDate, iTime, comments, staffNo, sName)

Property(propNo, pAddress,)

2NF Achieved

Not in 3NF as staffNo -> sName
StaffPropInspection(propNo FK, iDate, iTime, comments, staffNo FK)

Property(propNo, pAddress)

Staff(staffNo, sName)

All tables are in 3NF