Normalisation

Purpose of Normalisation

- Database normalisation is a technique of organising the data in the database.
- Normalisation is a systematic approach of decomposing tables to eliminate data redundancy(unnecessary repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies.

Purpose of Normalisation

- It is a multi-step process that puts data into tabular form, removing redundant data from the relation tables.
- Normalisation is used for mainly two purposes,
  - Eliminating redundant(useless) data.
  - Ensuring data dependencies make sense i.e. data is logically stored.

Purpose of Normalisation

- Characteristics of a suitable set of relations include:
  - 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.

Purpose of Normalisation

- The benefits of using a database that has a suitable set of relations is that the database will be:
  - easier for the user to access and maintain the data;
  - take up minimal storage space on the computer.

Data Redundancy and Update Anomalies

- Major aim of relational database design is to group attributes into relations to minimise data redundancy.

- Potential benefits for implemented database include:
  - Updates to the data stored in the database are achieved with a minimal number of operations thus reducing the opportunities for data inconsistencies.
  - Reduction in the file storage space required by the base relations thus minimising costs.

Data Redundancy and Update Anomalies

- Problems associated with data redundancy are illustrated by the StaffBranch relation below.

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

- The **StaffBranch** relation has redundant data; the details of a branch are repeated for every member of staff.

Data Redundancy and Update Anomalies

- Relations that contain redundant information may potentially suffer from update anomalies.

- Types of update anomalies include
  - Insertion
  - Deletion
  - Modification

- We will now look at the potential of these problems using the **StaffBranch** relation.

Data Redundancy and Update Anomalies

- Insertion
  - To insert a new Staff record into StaffBranch, we must include either the attribute values for the branch that the employee works for, or NULLs. So if a staff member works for branch B005, we must enter all the attribute values for branch B005 correctly so that they are consistent with other records (i.e. other staff members who work in branch B005).
  - If we want to insert a new branch that has no staff members yet, this will also generate problems. We cannot enter NULL values for all of the staff attributes as staffNo is the primary key and primary key values cannot be NULL (Entity Integrity).

Data Redundancy and Update Anomalies

- Modification
  - If a staff member moves Branch, then we must ensure that the Branch address (bAddress) is also updated.
- Deletion
  - If we delete a record from the StaffBranch relation for any staff member that happens to represent the last employee working for a particular branch, the information concerning that branch is lost from the database.

- So, we need to design base relation schemas so that no insertion, modification, or deletion anomalies are present.

Data Redundancy and Update Anomalies

- We will now look at how the data should be stored.The data should be split over 2 tables, **Branch** and **Staff**.

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

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

Lossless-join and Dependency Preservation Properties

- There are two important properties associated with decomposition of a larger relation into smaller relations:

  - The *lossless-join* property ensures that any instance of the original relation can be identified from corresponding instances in the smaller relations.

  - The d*ependency preservation* property ensures that a constraint on the original relation can be maintained by simply enforcing some constraint on each of the smaller relations. In other words, we do not need to perform joins on the smaller relations to check whether a constraint on the original relation is violated.

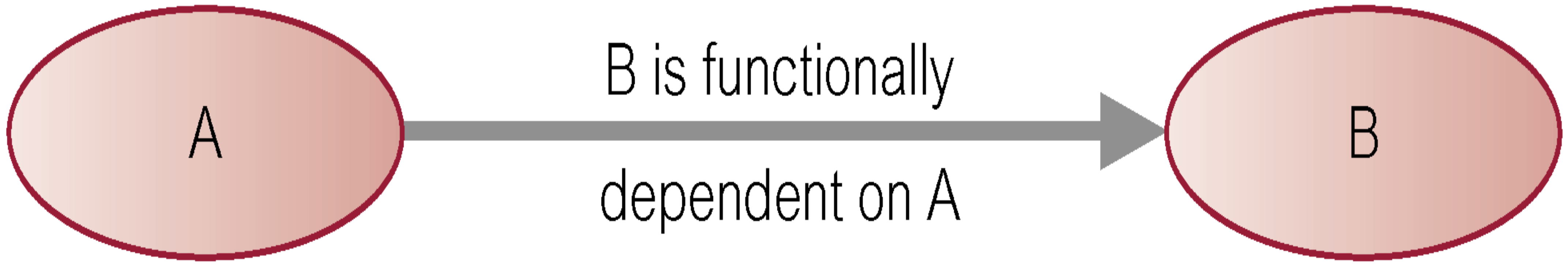
Functional Dependencies

- Important concept associated with normalisation.
- **Functional dependency** describes relationship between attributes.
- For example, if A and B are attributes of relation R, B is functionally dependent on A (denoted A $\rightarrow$ B), if each value of A in R is associated with exactly one value of B in R.
- It requires that the value for a certain set of attributes determines uniquely the value for another set of attributes.

Functional Dependencies

- **Definition:** A **functional dependency**, denoted by A→ B, between two sets of attributes A and B that are subsets of relation R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t1 and t2 in r that have t1[A] = t2[A], they must also have t1[B] = t2[B].

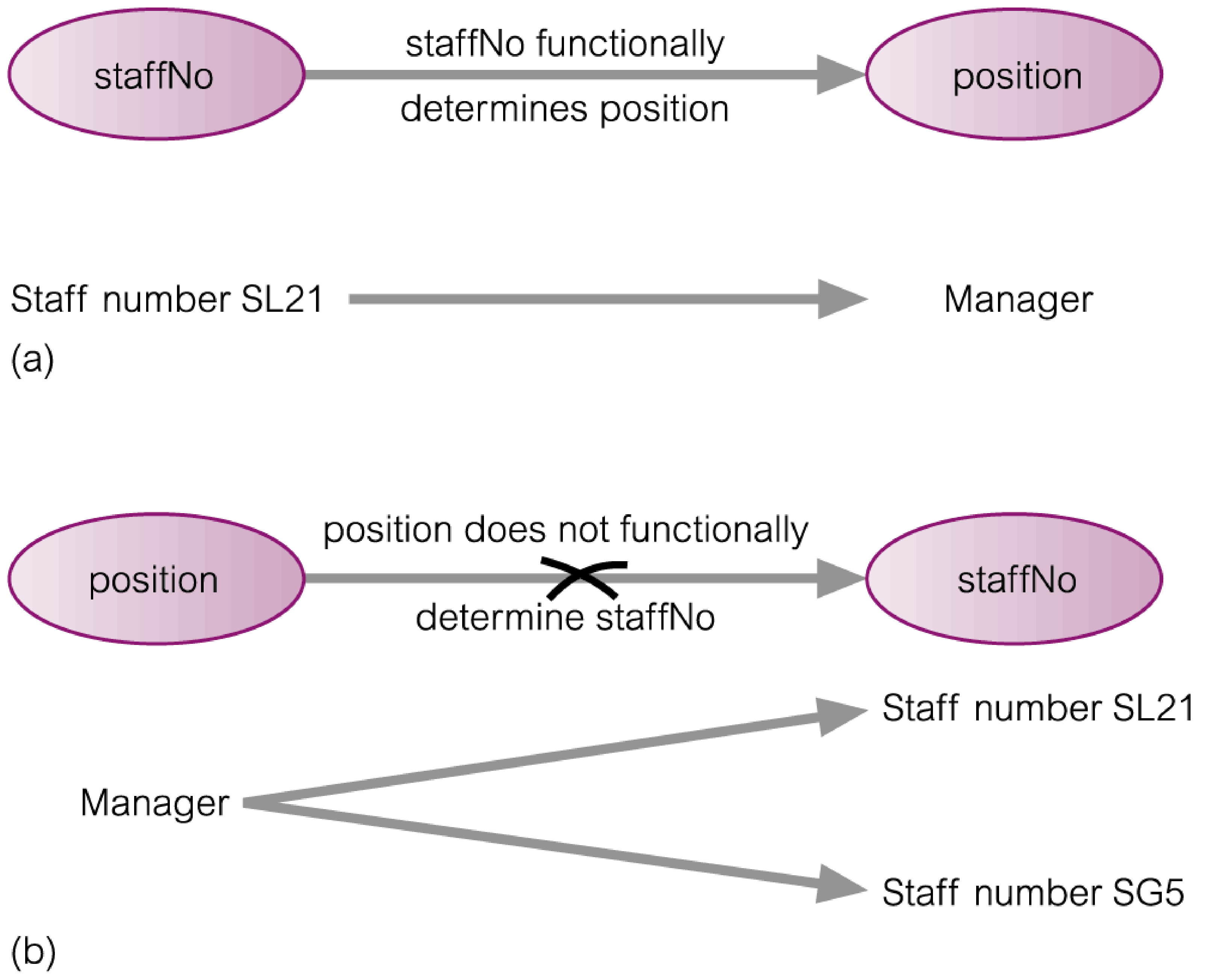
Characteristics of Functional Dependencies



- The *determinant* of a functional dependency refers to the attribute or group of attributes on the left-hand side of the arrow.

- The *dependent attribute* refers to the attribute or attributes on the right-hand side of the arrow.

An Example Functional Dependency



Example Functional Dependency that holds for all Time

- Consider the values shown for all the attributes in the Staff relation.

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

Example Functional Dependency that holds for all Time

- Based on sample data, the following functional dependencies appear to hold.

  staffNo → sName, position, salary, branchNo

  sName → staffNo **,** position, salary, branchNo

- However, the only functional dependency that remains true for all possible (future) values for the staffNo and sName attributes of the Staff relation is:

  staffNo → sName

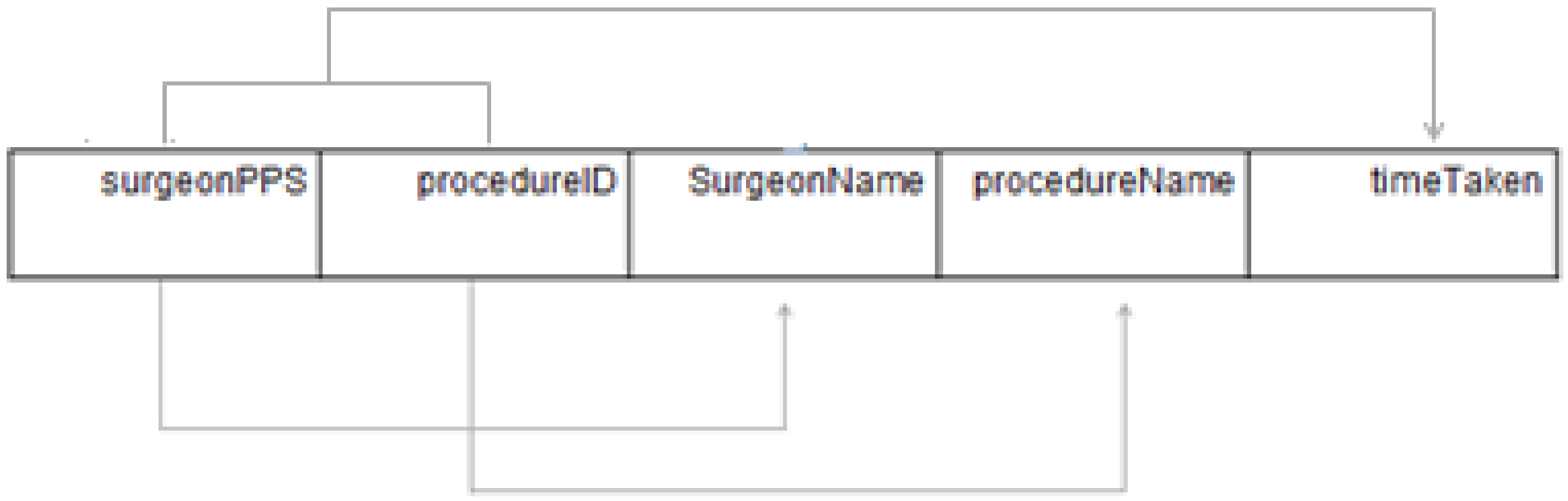
Characteristics of Full Functional Dependencies

- Determinants should have the minimal number of attributes necessary to maintain the functional dependency with the attribute(s) on the right hand-side.
- This requirement is called *full functional dependency.*

Characteristics of Full Functional Dependencies

- **Definition:** Full functional dependency indicates that if A, B and C are attributes of a relation, and C is fully functionally dependent on the composition of A and B, then C is functionally dependent on A,B but not on any proper subset of A,B.

Example Full Functional Dependency



Example Full Functional Dependency

- In the example table given we can see the following full functional dependency holds:
  surgeonPPS, procedureID→ timetaken as timetaken is not functionally dependent on a subset of surgeonPPS, procedureID.

- But surgeonPPS, procedureID→ surgeonName does not hold as surgeonName is functionally dependent on a subset of surgeonPPS, procedureID (i.e. surgeonPPS).

- Likewise surgeonPPS, procedureID→ procedureName does not hold as procedureName is functionally dependent on a subset of surgeonPPS, procedureID (i.e. procedureID).

Characteristics of Functional Dependencies

- Main characteristics of functional dependencies used in normalisation:
    - There is a *one-to-one* relationship between the attribute(s) on the left-hand side (determinant) and those on the right-hand side of a functional dependency.
    - Holds for *all* time.
    - The determinant has the *minimal* number of attributes necessary to maintain the dependency with the attribute(s) on the right hand-side.

Transitive Dependencies

- Important to recognise a transitive dependency because its existence in a relation can potentially cause update anomalies.

- When an indirect relationship causes functional dependency it is called **Transitive Dependency**.

- If P -> Q and Q -> R is true, then P-> R is a transitive dependency.

Transitive Dependencies

- **Definition:** Transitive dependency describes a condition where A, B, and C are attributes of a relation such that if $A \rightarrow B$ and $B \rightarrow C$, then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C).

Example Transitive Dependency

- Consider functional dependencies in the StaffBranch relation.

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

Example Transitive Dependency

- We can see the following functional dependencies:

  staffNo → sName, position, salary, branchNo, bAddress

  branchNo → bAddress

- Here, bAddress is transitively dependent on staffNo via branchNo.