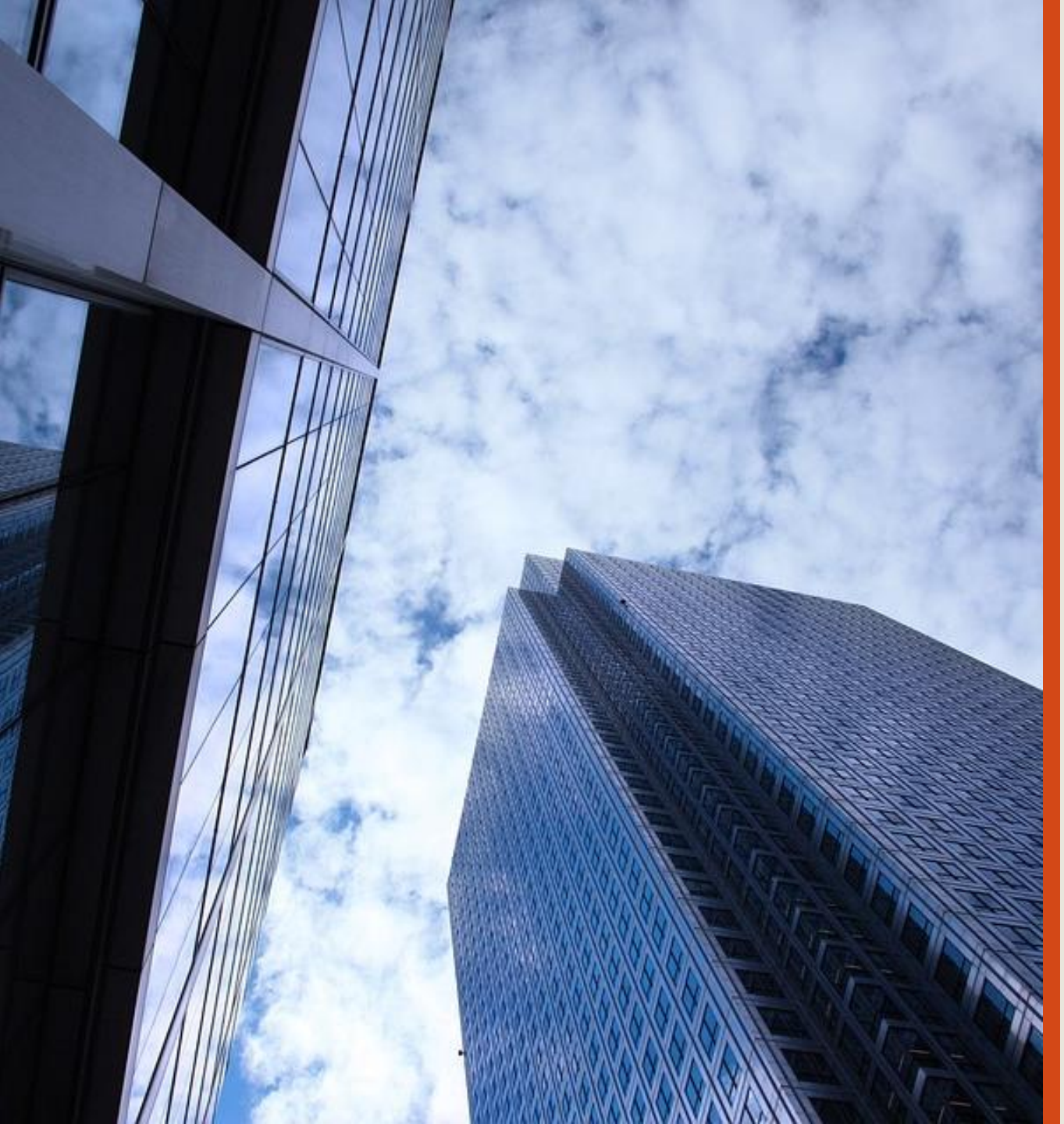


NORMALIZATION

PART 01

Jayathma Chathurangani

ejc@ucsc.cmb.ac.lk

LESSON SHOULD COVER..

- Introduction to data normalization and normal forms (Background)
  - What is normalization
  - Benefits of normalization
  - Normalization Rules
- First Normal Form
- Second Normal Form
- Third Normal Form
- Normalization Considerations

OUTLINE

- **Background**

1) Data Redundancy

2) Update Anomalies

3) Important Areas

4) Functional Dependency

BACKGROUND

- When we design a database for an enterprise, the main objective is to create an **accurate representation of the data, relationships between the data, and constraints on the data** that is pertinent to the enterprise.
- To help achieve this objective, we can use one or more database design techniques.
- Earlier you learnt a technique called ER modeling.
- In this topic we learn another database design technique called **normalization**.

1

DATA REDUNDANCY

1.1 WHAT IS IT?

- A major aim of relational database design is to group attributes into relations to minimize data redundancy.
- If this aim is achieved, the potential benefits for the implemented database include the following:
  - ✓ **updates to the data stored in the database are achieved with a minimal number of operations**, thus reducing the opportunities for data inconsistencies occurring in the database;
  - ✓ **reduction in the file storage space** required by the base relations thus minimizing costs.
- Of course, relational databases also rely on the existence of a certain amount of data redundancy. This redundancy is in the form of copies of primary keys (or candidate keys) acting as foreign keys in related relations to enable the modeling of relationships between data.

1.2 PROBLEMS ASSOCIATED WITH UNWANTED 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

- Consider the below Relations

Set 01

Staff (staffNo, sName, position, salary, branchNo)
Branch (branchNo, bAddress)

Set 02

StaffBranch (staffNo, sName, position, salary, branchNo, bAddress)

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

1.2 PROBLEMS ASSOCIATED WITH UNWANTED DATA REDUNDANCY (CONTINUED)

- The problems associated with unwanted data redundancy is illustrated by comparing the Staff and Branch relations shown with the StaffBranch relation shown in Figure.
- The StaffBranch relation is an alternative format of the Staff and Branch relations.
- In the StaffBranch relation there is redundant data; the details of a branch are repeated for every member of staff located at that branch.
- In contrast, the branch details appear 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.
- Relations that have redundant data may have problems called **update anomalies**, which are classified as insertion, deletion, or modification anomalies.

02

UPDATE ANOMALIES

2.1 INSERTION ANOMALY

- An insertion anomaly occurs when data cannot be inserted into a database due to other missing data
- This is most common for fields where a foreign key must not be NULL, but lacks the appropriate data
- An example of this anomaly can be explained with a simple user database
  - A user must have a *group ID* as a foreign key
  - No groups have yet been created
  - Thus, a user can not be inserted in to the database as the *group ID* must not be NULL
- This can result in data redundancy due to the omission of data

2.2 DELETION ANOMALY

- A deletion anomaly occurs when data is unintentionally lost due to the deletion of other data
- For example,
  - if a database row contained "Username" and "User Group"
  - "John" and "Fred" are in the user group "Contributors"
  - If John and Fred are removed from the database, our Contributors group will also disappear
- This is because we haven't normalised our data, meaning the only reference to the Contributors user group lies within the same database row (or record)
- Hence, removing the only two references of our user group results in the loss of data accuracy and integrity
- This also goes to show why it's important for us to normalise our data and how combining unlike information can be problematic

2.3 MODIFICATION ANOMALY

- An update anomaly occurs when data is only partially updated in a database
- A database that hasn't undergone normalization may reference the same data element in more than one location
- As these locations haven't been consolidated and referenced, we have to make sure each location is manually updated
- This can cause problems as we then need to spend time searching for and updating each reference to the data element
- An example of this is a database containing two records; Users and Mailing List
  - John has an email address of john@mail.com in the Users record
  - John has the same email address in the Mailing List record
  - John decides to change his email preferences, which in turn updates the User record for John
  - However, the system did not automatically update the Mailing List record, leaving John with two different associated emails and thus creating inconsistencies within our database

03

MORE IMPORTANT AREAS

3.1 IMPORTANT PROPERTIES ASSOCIATED WITH DECOMPOSITION OF A LARGER RELATION INTO SMALLER RELATIONS:

- Decomposition should have below properties:
- The **lossless-join** property ensures that any instance of the original relation can be identified from corresponding instances in the smaller relations.
- The **dependency 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. Or Simple it preserve the associated set of functional dependencies.
- Although the **StaffBranch** relation is subject to update anomalies, we can avoid these anomalies by decomposing the original relation into the Staff and Branch relations.

Let us populate R with sample data and try the experiment;

A	B	C	D
a_1	a_2	a_3	a_4
a_1	a_4	a_3	a_2

According to the decomposition, we shall get R_1 and R_2 as follows;

R_1		
A	**B**	**C**
a_1	a_2	a_3
a_1	a_4	a_3

R_2		
A	**C**	**D**
a_1	a_3	a_4
a_1	a_3	a_2

Join back R_1 and R_2 must result in R if the decomposition is lossless.

$R_1 \bowtie R_2 = R'$

A	B	C
a_1	a_2	a_3
a_1	a_4	a_3

$\bowtie$

A	C	D
a_1	a_3	a_4
a_1	a_3	a_2

$=$

A	B	C	D
a_1	a_2	a_3	a_4
a_1	a_2	a_3	a_2
a_1	a_4	a_3	a_2
a_1	a_4	a_3	a_4

R′ is the result of natural join of R_1 and R_2, and R′ is not equal to R the base relation. Hence, **the decomposition is not lossless join decomposition**.

3.2 KEY TYPES

- **Primary Key**: This key uniquely identify each entry in a table. This value cannot be repeated inside a table and cannot hold null values. Generally first columns is defined as primary key. Example (Student ID).
- **Foreign Key:** This key can have repetitive values, but to uniquely identify each entry the table can still have primary key column separate of foreign key column. However, the foreign key will create a relation with another table where those values are defined as primary keys.
- **Compound Key**: This is the methods of defining multiple columns as primary key. Situations where no column have unique values in a table, we can define a combination of two or more than two columns as unique and set it as primary key. For example: (Student Name, Address, Marks, …, etc.) Here it is likely that student can have same names, therefore we define combination of student name and address as primary key. Now it is more unlikely that there can be student with same name and same address.
- **Candidate Key**: In simple words a candidate key is a key that can also serve as a primary key. For example: (Student ID, Student Roll No., Address, Marks) Here student id is primary key because it does not have repetitive value, does not have null values. However, student roll number also holds all the properties of primary key and thus considered as candidate key.
- **Surrogate Key**: This means an artificially created value that uniquely identify each entry in a table when no other column was able to hold properties of a primary key. It is an additional column and generally holds integer values.

04

FUNCTIONAL DEPENDENCY

4.1 WHAT IS IT?

- **Functional Dependency (FD)** is a constraint that determines the relation of one attribute to another attribute in a Database Management System (DBMS).
- Functional Dependency helps to maintain the quality of data in the database. Helps in Normalization.
- It plays a vital role to find the difference between good and bad database design.
- A functional dependency is denoted by an arrow "→". The functional dependency of Y on determinant X is represented by X → Y.
- Functional dependency is a property of the meaning or semantics of the attributes in a relation. The semantics indicate how attributes relate to one another, and specify the functional dependencies between attributes. When a functional dependency is present, the dependency is specified as a **constraint** between the attributes.
- There are mainly four types of Functional Dependency in DBMS. Following are the types of Functional Dependencies in DBMS: Multivalued Dependency, Trivial Functional Dependency, Non-Trivial Functional Dependency, Transitive Dependency. Apart from that there are fully functional and Partial dependencies.

When identifying functional dependencies between attributes in a relation, it is important to distinguish clearly between the values held by an attribute at a given point in time and the *set of all possible values* that an attribute may hold at different times. In other words, a functional dependency is a property of a relational schema (intension) and not a property of a particular instance of the schema (extension) (see Section 4.2.1). This point is illustrated in the following example.

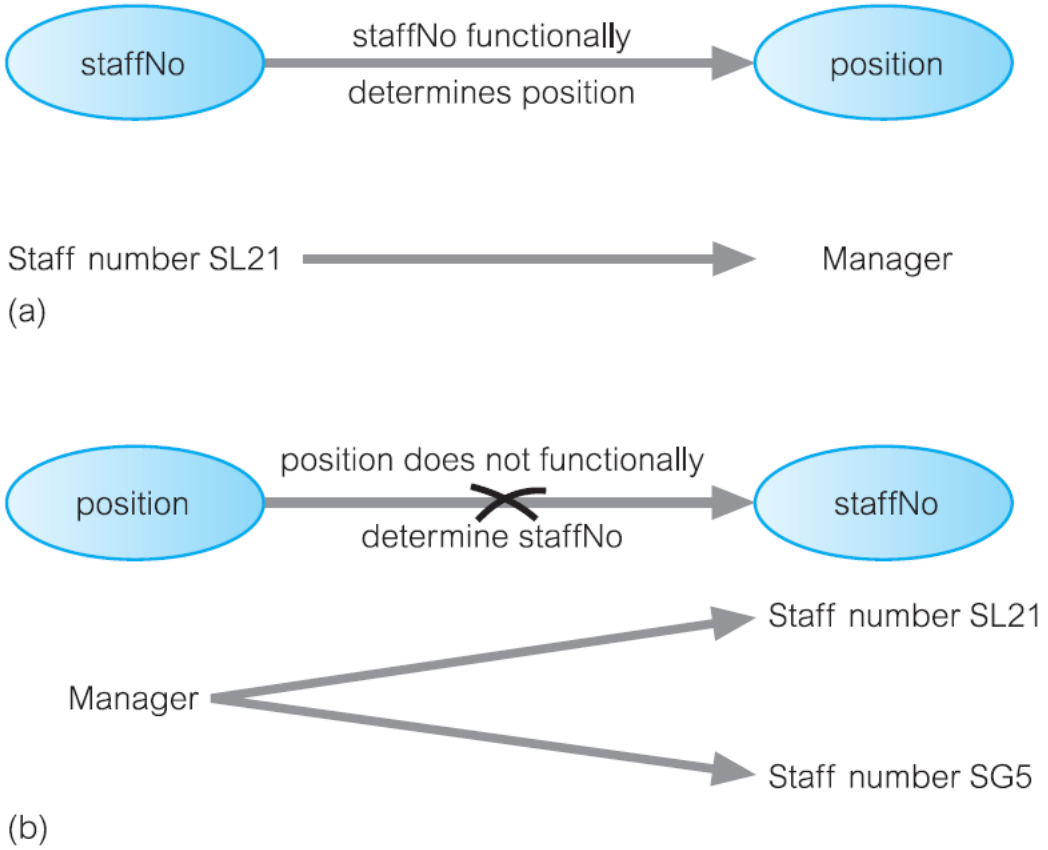


Figure 14.5 (a) staffNo functionally determines position (staffNo ® position); (b) position does *not* functionally determine staffNo (position ⊗ staffNo).

Example

- In below example, if we know the value of Employee number, we can obtain Employee Name, city, salary, etc.
- By this, we can say that the city, Employee Name, and salary are functionally depended on Employee number.

Employee number	Employee Name	Salary	City
1	Dana	50000	San Francisco
2	Francis	38000	London
3	Andrew	25000	Tokyo

4.2 RULES

Key Terms	Description
Axiom	Axioms is a set of inference rules used to infer all the functional dependencies on a relational database.
Decomposition	It is a rule that suggests if you have a table that appears to contain two entities which are determined by the same primary key then you should consider breaking them up into two different tables.
Dependent	It is displayed on the right side of the functional dependency diagram.
Determinant	It is displayed on the left side of the functional dependency Diagram.
Union	It suggests that if two tables are separate, and the PK is the same, you should consider putting them. together

4.2 RULES (CONTINUED)

Armstrong's axioms

- ***Reflexive rule:*** If **X** is a set of attributes and **Y** is_subset_of **X**, then **X** holds a value of **Y**.
- ***Augmentation rule:*** When **X** → **Y** holds, and **c** is attribute set, then **XC** → **YC** also holds. That is adding attributes which do not change the basic dependencies.
- ***Transitivity rule:*** This rule is very much similar to the transitive rule in algebra if **X** → **Y** holds and **Y** → **Z** holds, then **X** → **Z** also holds. **X** → **Y** is called as functionally that determines **Y**.

SECONDARY AXIOMS

- ***Union:*** If A→B and A→C, then A→BC
- ***Decomposition:*** If A → BC then A → B and A → C (Split)
- ***Pseudo Transitivity rule:*** If A → B and BC → D, then AC → D
- **Composition:** If X → Y and A → B, then XA → YB

4.3 FUNCTIONAL DEPENDENCY FURTHER

Describes the relationship between attributes in a relation.

The database is described by a single universal relation.

$$\mathbf{R = (A, B, C, \ldots, Z).}$$

Consider the functional dependency;

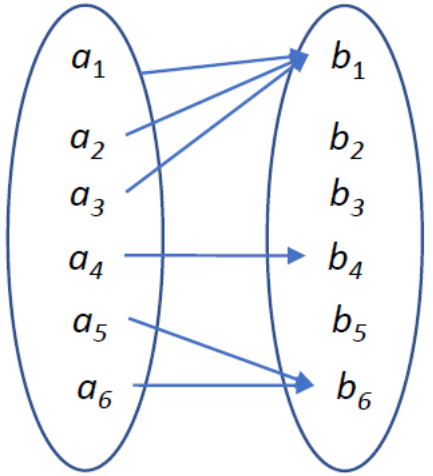
$A \rightarrow B$ where $A, B \in R$

B is functionally dependent on A

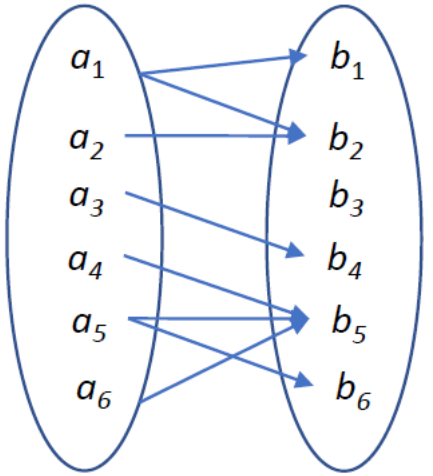
A determines B

A is the determinant of B

Each value of A is associated with exactly one value of B



√



X

4.4 FULLY FUNCTIONAL DEPENDENCY

- All non-key attributes depend on the key attribute, it says Full Functional Dependency.
- A full functional dependency occurs when you already meet the requirements for a functional dependency and the set of attributes on the left side of the functional dependency statement cannot be reduced any further.
- **X** → **Y** is a full-functional dependency because of removal of any attribute a form **X** would result in the cancellation of dependency.
- **If there is a single determinant it automatically becomes a full functional dependency**
- For example, **{SSN, age}** → **name** is a functional dependency, but it is not a full functional dependency because you can remove age from the left side of the statement without impacting the dependency relationship.

4.5 PARTIAL FUNCTIONAL DEPENDENCY

- A functional dependency **X** → **Y** is a Partial Dependency if some attributes **A** ϵ **X** can remove from **X** and the dependency still holds.
- A partial dependency is when you have a composite primary key (a primary key that is made up of multiple attributes), and on of the non-key attributes functionally dependent on one, but not all of the attributes that makeup the composite primary key.
- It is possible to derive full functional dependency from partial dependency.

StaffNo,sName → branchNo

StaffNo → branchNo

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

4.6 TRANSITIVE DEPENDENCY

- A functional dependency **X** → **Y** about scheme **R** is a Transitive Dependency if there is a set of attributes **Z** that is neither a candidate key nor a subset key of **R** and both **X** → **Z** and **Z** → **Y** hold.
- Transitive Dependency occurs because one non-key attribute is dependent on other non-key attributes. (A Transitive Dependency is when you have a column that is functionally dependent on a column that is not the primary key)
- A general case of transitive dependencies is given as follows: A, B, C are three columns in a table.
  - **If C is related to B**
  - **If B is related to A**
  - **Then C is indirectly related to A**
- Transitive dependencies occur when there is an indirect relationship that causes a functional dependency.
- *Note:* You need to remember that transitive dependency can only occur in a relation of three or more attributes.

4.6 TRANSITIVE DEPENDENCY (CONTINUED)

EXAMPLE

- **{Company}** → **{CEO}** (if we know the compay, we know its CEO's name)
- **{CEO }** → **{Age}** (If we know the CEO, we know the Age)
- Therefore according to the rule of rule of transitive dependency:

{Company} → **{Age}** should hold, that makes sense because if we know the company name, we can know his age.

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Alibaba	Jack Ma	54

4.7 MULTIVALUED DEPENDENCY

- Multivalued dependency occurs in the situation where there are **multiple independent multivalued attributes** in a single table.
- A multivalued dependency is a complete constraint between two sets of attributes in a relation. It requires that certain tuples be present in a relation.
- If we have **two or more multivalued independent attributes** in the same relation schema. We would get into the problem of repeating every value of one of the attributes with every value of the other attribute to keep the relation instances consistent.
- (Multivalued Dependencies are a consequence of the first normal form. The first normal form doesn't allow an attribute in the tuple to have more than one value.)

4.7 MULTIVALUED DEPENDENCY (CONTINUED)

EXAMPLE

- In this example, **maf_year** and **color** are independent of each other but dependent on **car_model**.
- Therefore, these two columns are said to be multivalued dependent on **car_model**.
- This dependence can be represented like this:
  - **car_model → maf_year**
  - **car_model → colour**

Car_model	Maf_year	Color
H001	2017	Metallic
H001	2017	Green
H005	2018	Metallic
H005	2018	Blue
H010	2015	Metallic
H033	2012	Gray

4.8 TRIVIAL FUNCTIONAL DEPENDENCY

- A trivial functional dependency occurs when you describe a functional dependency of an attribute on a collection of attributes that includes the original attribute.
- The Trivial dependency is a set of attributes which are called a trivial if the set of attributes are included in that attribute. So,

  X → **Y** is a trivial functional dependency if **Y** is a subset of **X**.
- **Example:** Consider this table of with two columns **Emp_id** and **Emp_name**.

{Emp_id, Emp_name} → **Emp_id** is a trivial functional dependency as **Emp_id** is a subset of **{Emp_id,Emp_name}**.

Emp_id	Emp_name
AS555	Harry
AS811	George
AS999	Kevin

4.9 NON-TRIVIAL FUNCTIONAL DEPENDENCY

- Functional dependency which also known as a nontrivial dependency occurs when **A** → **B** holds true where **B** is not a subset of **A**. In a relationship, if attribute **B** is not a subset of attribute **A**, then it is considered as a non-trivial dependency.
- **Example:**

(Company} → **{CEO}** (if we know the Company, we knows the **CEO** name)

But **CEO** is not a subset of Company, and hence it's non-trivial functional dependency.

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	57

4.10 ADVANTAGES

- It helps you to maintain the quality of data in the database
- It helps you to defined meanings and constraints of databases
- It helps you to identify bad designs
- It helps you to find the facts regarding the database design

WRAP UP