Normalization

Contents

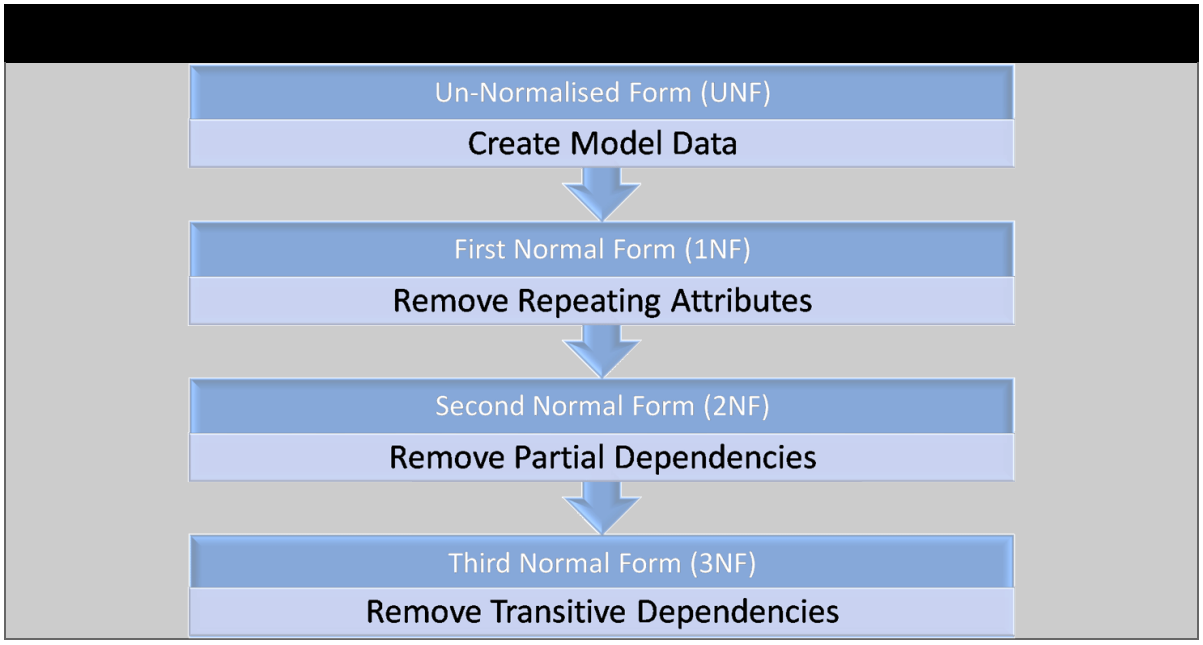
- Introduction
- Database Design
- Basic ER Modelling
- Relationship
- Mapping Constraints
- Keys

1. Normalization Process

Normalization was developed by Dr. E.F. Codd in 1972 as part of the Relational Database Theory as a means of breaking data into its related groups and defining the relationships between those groups. It is said the name normalization was initially a political gag taken from President Nixon and his initiative for 'Normalizing' relations with China. Codd figured if you can normalize relations with a country, you should be able to normalize the relations with data as well.

Normalization is a specific relational database analysis and design technique used to model groups of related data within an organization. Its purpose is to ensure data stored within the database adheres to best practices by following a set of rules with the purpose of eliminating redundancies and optimizing the process of information retrieval. Normalization leaves us with a structure that groups like data into relational models referenced by keys and linked to other relational models to form a relational database schema.

Normalization is represented by a logical set of steps that follow simple rules that are applied to each stage of the modelling process. At the highest level, the stages are separated into something called Normal Forms, identified by a particular named process.



Initially there were only three normal forms, First Normal Form (1NF), Second Normal Form (2NF) and Third Normal Form (3NF), but over time three more were added. In general, terms the first three are more commonly used in database modelling. The additional three are identification of potential redundancies that could be considered but however when applied practically can lead to inefficiencies in performance and tend to be used under special circumstances or for consideration with complex data structures.

In addition we have something called Un-Normalized Form (UNF), though not generally considered as part of the normalization rules, is representative of the very first stages of the normalization process.

We can identify each of the normal forms as follows and will define each in detail thereafter:

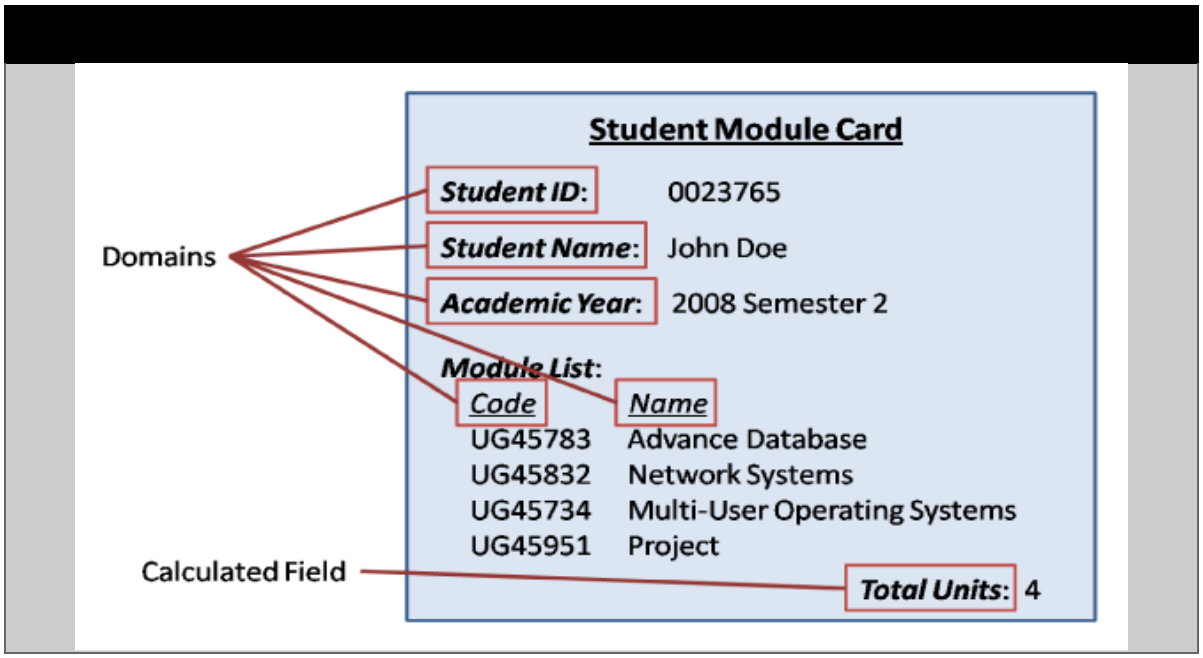
- ✓ ***Un-Normalized Form (UNF) – Data Modelling***
- ✓ ***First Normal Form (1NF) – Repeating Groups***
- ✓ ***Second Normal Form (2NF) – Partial Dependencies***
- ✓ ***Third Normal Form (3NF) – Transitive Dependencies***

☒ ***Un-Normalized Form (UNF)***

n-normalized form is a preparatory stage of the normalization process allowing us to create a structured frame, representative of a piece of organizational data such as a form or document (e.g invoice, report, purchase order etc.). This is our initial normalization 'relation' that contains both real data, taken from the form or document, and modelled data, based upon and extended from the original from or document.

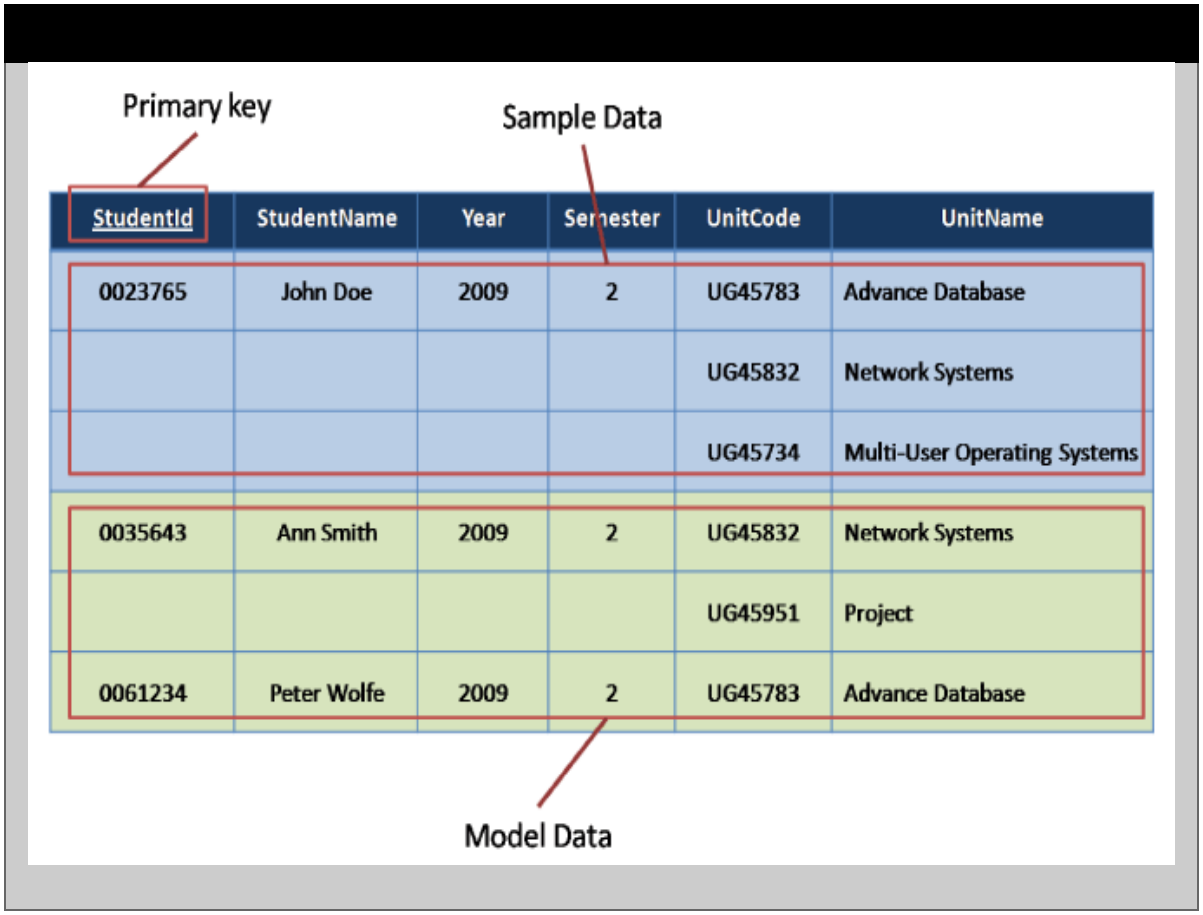
At this point the un-normalized relation is just a big jumble of data but this preparatory stage is the most important. As each stage of the normalization process is dependent upon the previous it is vital for this, as the starting stage, is set up with the right domains and data to ensure a smooth transition between the stages.

As with all the normalization stages, to create an un-normalized relation you simply follow a set of logical steps.
Based on the form or document you are working from, draw up a table structure creating column heading for each of the data items. These column headings represent a normalization domain and should be named following good naming convention standards. When selecting the domains make sure you don't include calculated fields as in fields that can be derived from other fields.



Using the form or document from step one, select a sample of data to create rows under the column headings. Try and create at least 3 rows of data taken directly from the form then create at least 3 more model data rows to provide a good range of data. These rows of data represent a normalization tuple and are a very important part of the process as without good model data it is harder to achieve good model design. We now need to select a suitable key from our domains that will allow us to have a unique reference. Identify the candidate keys and from this select a suitable Primary Key. Underline the selected domain(s), this will be our starting key. Our table should be looking complete but the last thing we must do is remove any repeating data as this will help us with our first normal form. Repeating data is data that because of its direct relationship with the Primary key, repeats itself in each of the tuples where the key is the same. You must be careful not to misread domains where

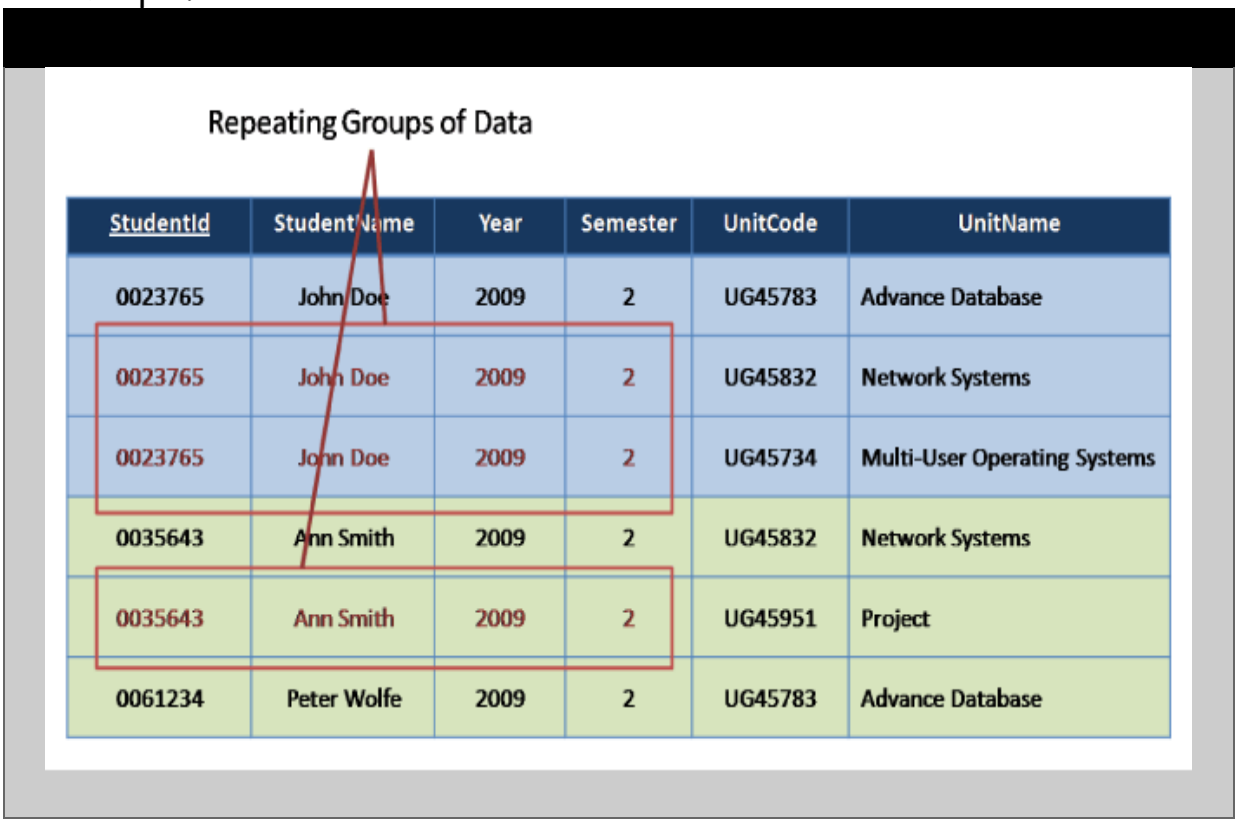
the data appears to repeat but this is due to the restrictions of the model data selected and not because of its relation with the key.



StudentId	StudentName	Year	Semester	UnitCode	UnitName
0023765	John Doe	2009	2	UG45783	Advance Database
				UG45832	Network Systems
				UG45734	Multi-User Operating Systems
0035643	Ann Smith	2009	2	UG45832	Network Systems
				UG45951	Project
0061234	Peter Wolfe	2009	2	UG45783	Advance Database

☒ *First Normal Form (1NF)*

With our un-normalized relation now complete, we are ready to start the normalization process. First Normal form is probably the most important step in the normalization process as it facilities the breaking up of our data into its related data groups, with the following normalized forms fine tuning the relationships between and within the grouped data. With First Normal Form, we are looking to remove repeating groups. A repeating group is a domain or set of domains, directly relating to the key, that repeat data across tuples in order to cater for other domains where the data is different for each tuple.



StudentId	StudentName	Year	Semester	UnitCode	UnitName
0023765	John Doe	2009	2	UG45783	Advance Database
0023765	John Doe	2009	2	UG45832	Network Systems
0023765	John Doe	2009	2	UG45734	Multi-User Operating Systems
0035643	Ann Smith	2009	2	UG45832	Network Systems
0035643	Ann Smith	2009	2	UG45951	Project
0061234	Peter Wolfe	2009	2	UG45783	Advance Database

In this example with Student ID as the primary key we see the three domains, ***StudentName***, Year and Semester repeat themselves across the tuples for each of the different ***UnitCode*** and ***UnitName*** entries. Though workable it means our relation could potentially be huge with loads of repeating data taking up valuable space and costing valuable time to search through.

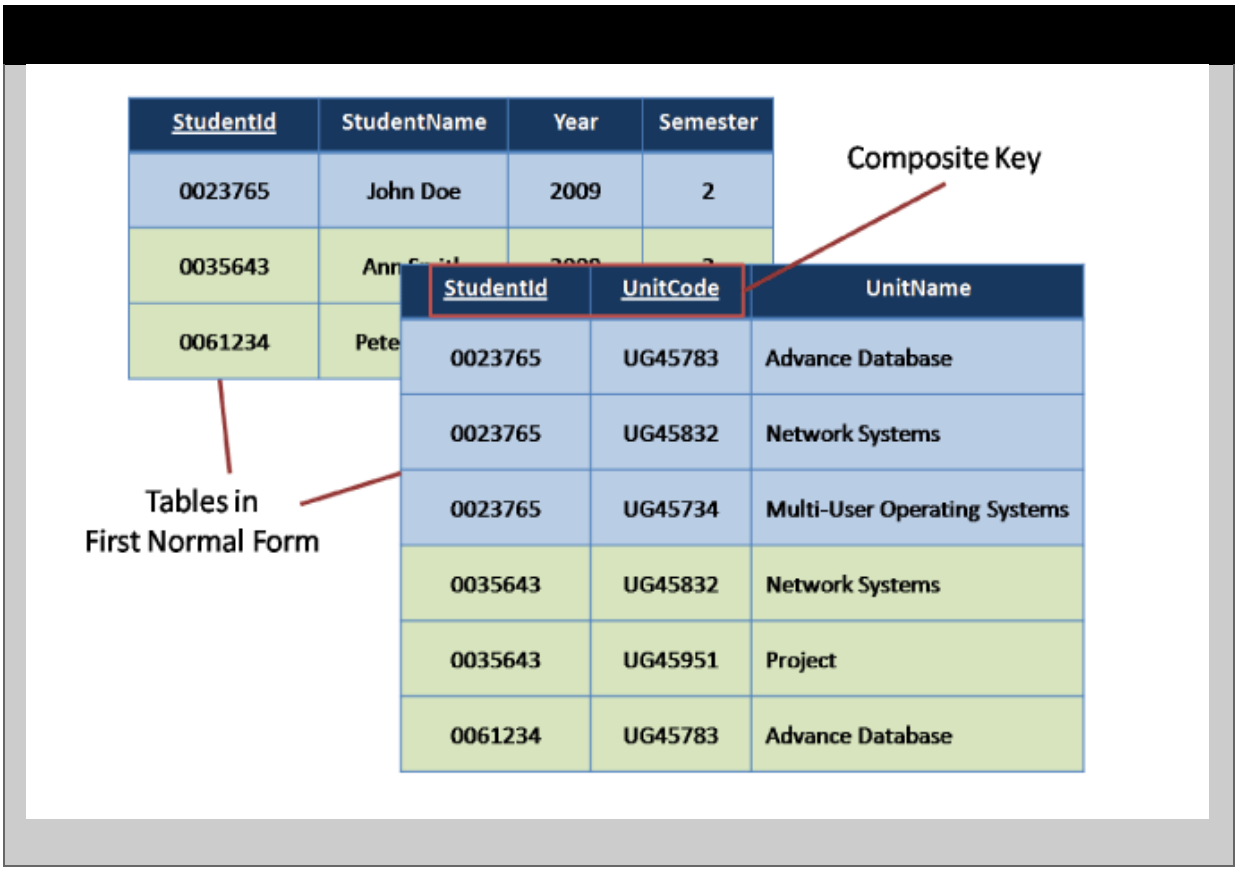
The rules of First Normal Form break this relation into two and relate them to each other so the information needed can be found

without storing unneeded data. So from our example we would have one table with the student information and another with the Unit Information with the two relations linked by a domain common to both, in this case, the ***StudentId***.

So the steps from UNF to 1NF are:

- ✓ Identify repeating groups of data. Make sure your model data is of good quality to help identify the repeating groups and don't be afraid to move the domains around to help with the process.
- ✓ Remove the domains of the repeating groups to a new relation leaving a copy of the primary key with the relation that is left.
- ✓ The original primary key will not now be unique so assign a new primary key to the relation using the original primary key as part of a compound or composite key.
- ✓ Underline the domains that make up the key to distinguish them from the other domains.

Taking our original example once we have followed these simple steps we have relations that looks like this:



☒ ***Second Normal Form (2NF)***

Now our data is grouped into sets of related data we still need to check we are not keeping more data than we need to in our relation. We know we don't have any repeating groups as we removed these with First Normal Form. But if we look at our example we can see for every ***UnitCode*** we are also storing the ***UnitName***.

StudentId	UnitCode	UnitName
0023765	UG45783	Advance Database
0023765	UG45832	Network Systems
0023765	UG45734	Multi-User Operating Systems
0035643	UG45832	Network Systems
0035643	UG45951	Project
0061234	UG45783	Advance Database

Would it not seem more sensible to have a different relation we could use to look up UG45783 and find the unit name 'Advanced Database'? This way we wouldn't have to store lots of additional duplicate information in our Student/Unit relation.
This is exactly what we aim to achieve with Second Normal Form and its purpose is to remove partial dependencies.
We can consider a relation to be in Second Normal Form when: The relation is in First Normal Form and all partial key dependencies are removed so that all non key domains are functionally dependent on all of the domains that make up the primary key.
Before we start with the steps, if we have a table with only a single simple key this can't have any partial dependencies as there is only one domain that is a key therfroe these relations can be moved directly to 2nd normal form.

For the rest the steps from 2NF to 3NF are:

Take each non-key domain in turn and check if it is only dependent on part of the key?

If yes

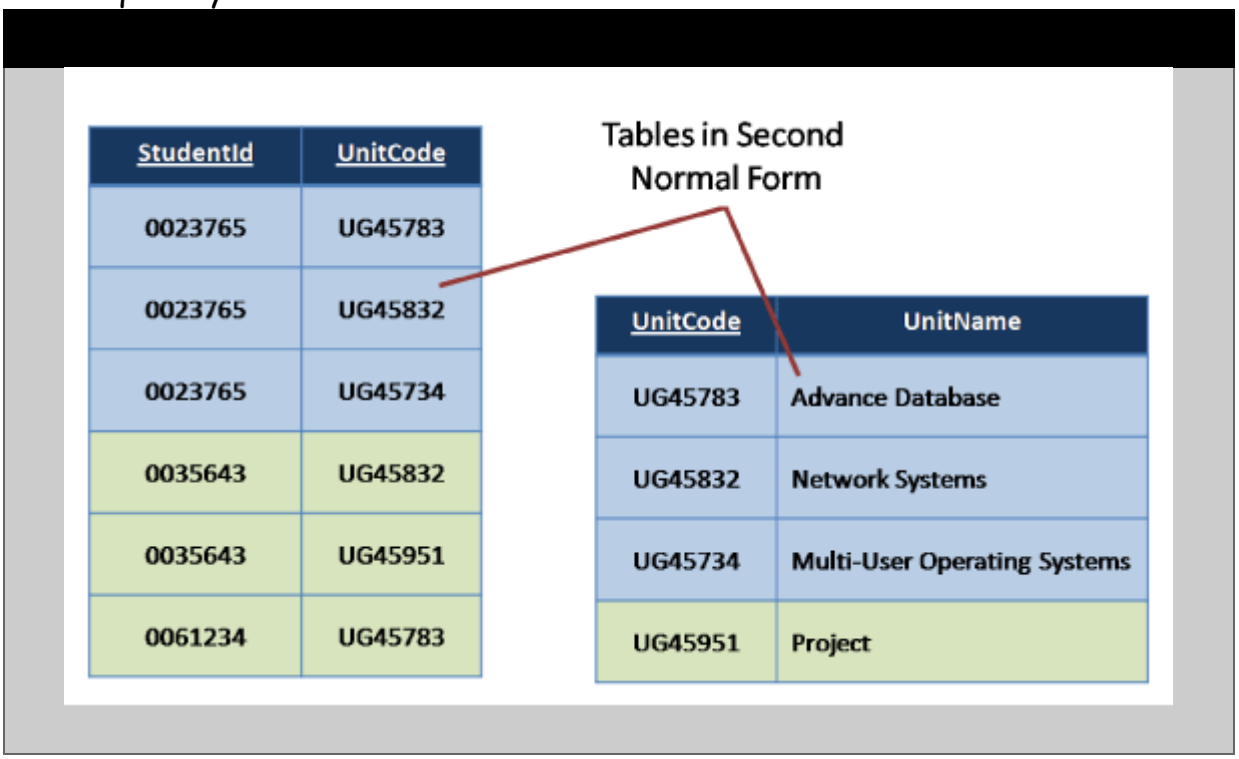
Remove the non-key domain along with a copy of the part of the key it is dependent upon to a new relation.

Underline the copied key as the primary key of the new relation.

Move down the relation to each of the domains repeating steps 1 and 2 till you have covered the whole relation.

Once completed with all partial dependencies removed, the table is in 2nd normal form.

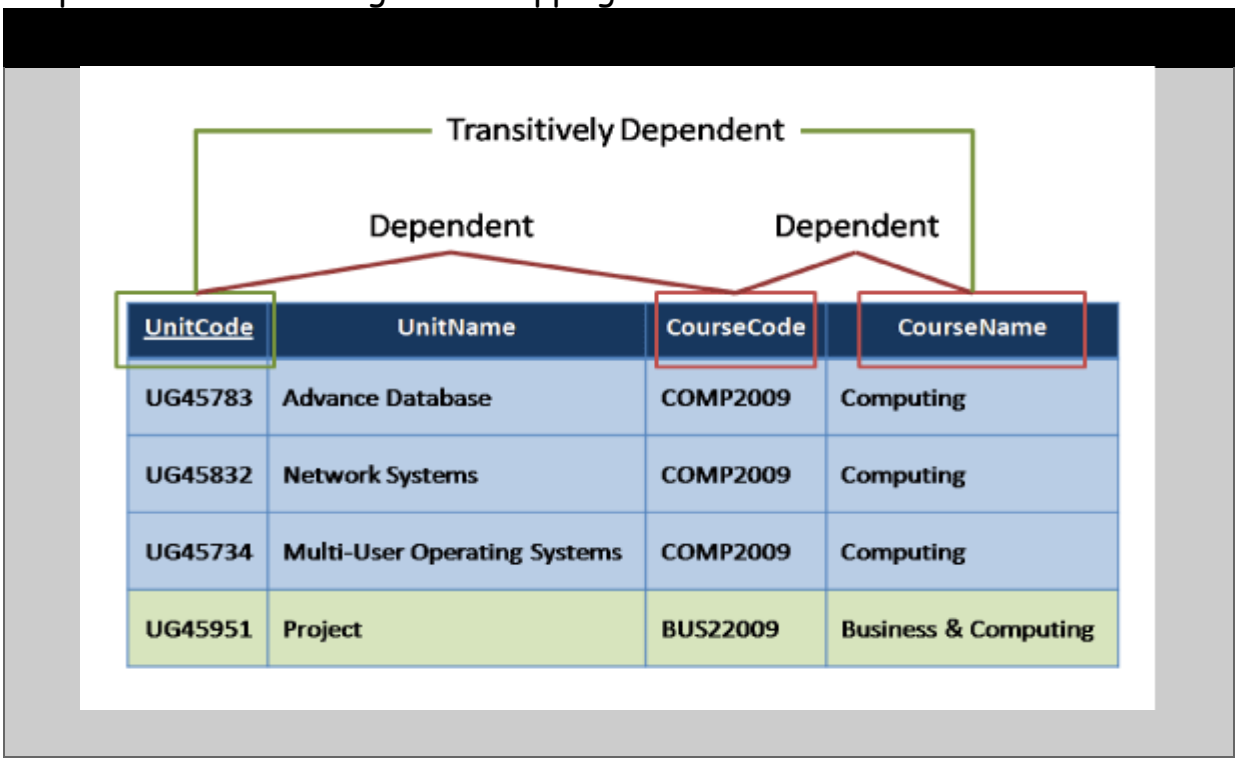
In our example above, ***UnitName*** is only ***dependant*** on ***unitCode*** and has no dependency on ***studentId***. Applying the steps above we move the ***unitName*** to a new relation with a copy of the part of the key it is dependent upon. Our table in second normal form would subsequently look like this:



☒ ***Third-Normal Form (3NF)***

Third Normal Form deals with something called 'transitive' dependencies. This means if we have a primary key A and a non-key domain B and C where C is more dependent on B than A and B is directly dependent on A, then C can be considered transitively dependent on A.

Another way to look at it is a bit like a stepping stone across a river. If we consider the primary key A to be the far bank of the river and our non-key domain C to be our current location, in order to get to A, our primary key, we need to step on a stepping stone B, another non-key domain, to help us get there. Of course we could jump directly from C to A, but it is easier, and we are less likely to fall in, if we use our stepping stone B. Therefore current location C is transitively dependent on A through our stepping stone B.



UnitCode	UnitName	CourseCode	CourseName
UG45783	Advance Database	COMP2009	Computing
UG45832	Network Systems	COMP2009	Computing
UG45734	Multi-User Operating Systems	COMP2009	Computing
UG45951	Project	BUS22009	Business & Computing

Before we start with the steps, if we have any relations with zero or only one non-key domain we can't have a transitive dependency so these move straight to 3rd Normal Form

For the rest the steps from 2NF to 3NF are:

Take each non-key domain in turn and check it is more dependent on another non-key domain than the primary key.

If yes

Move the dependent domain, together with a copy of the non-key attribute upon which it is dependent, to a new relation.

Make the non-key domain, upon which it is dependent, the key in the new relation.

Underline the key in this new relation as the primary key.

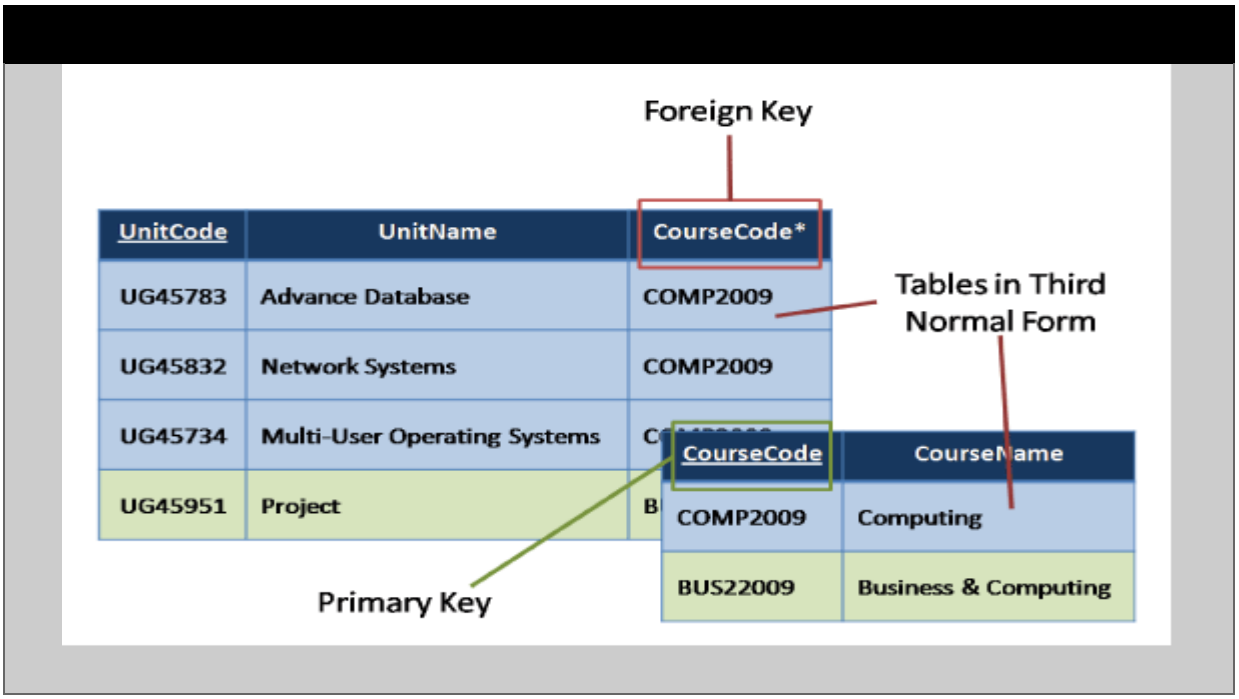
Leave the non-key domain, upon which it was dependent, in the original relation and mark it a foreign key (*).

Move down the relation to each of the domains repeating steps 1 and 2 till you have covered the whole relation.

Once completed with all transitive dependencies removed, the table is in 3rd normal form.

In our example above, we have ***unitCode*** as our primary key, we also have a ***courseName*** that is ***dependent*** on ***courseCode*** and ***courseCode***, dependent on ***unitCode***. Though ***couseName*** could be dependent on ***unitCode*** it more dependent on ***courseCode***, therefore it is transitively dependent on ***unitCode***.

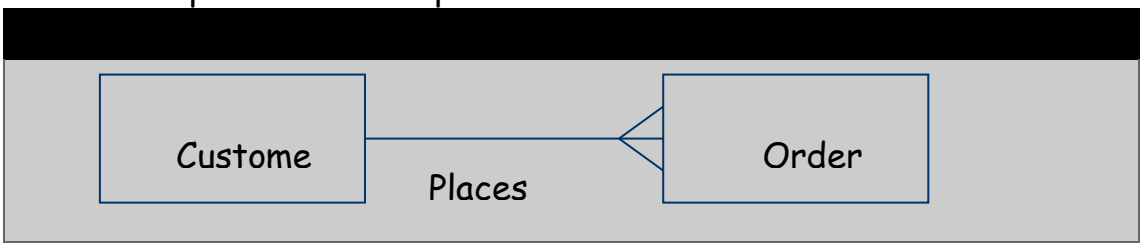
So following the steps, remove ***courseName*** with a copy of course code to another relation and make ***courseCode*** the primary key of the new relation. In the original table mark ***courseCode*** as our foreign key



- ***The identification of the data objects or entities in the system, their structure and relationships between entities.***
  The construction of a model of the stored data requirements of the system which is independent of specific processing requirements.

- ***The construction of a robust data model i.e. Minimal model of the data required to be stored in the system.***
  The construction of a logical model of data i.e. a model that is not concerned with how the data storage will be, or is currently physically implemented.
- ***Data modelling aims to identify the system entities, i.e. items which the system needs to store data (e.g. customers, orders products etc.)***
- ***The data model also shows the relationships between entities.***
  Customer places an order
- ***The data model concentrates on the properties of the data itself, independent of the processing requirements of the system.***
  The data model aims to collect all the data that needs to be stored for the system to operate and to organize that data into a sound structure

- ***Its is important that little redundant data is stored.***
- ***As far as possible one item of data should be stored in one place and only one place only.***
  If the same data is stored more than once this can lead to discrepancies between data that has been updated in place but not in another
- ***Data modelling is not concerned with how the data is physically stored in the current system or in the computer-based system being developed.***
  The data model aims to develop an efficient model of the data, independently of how it is implemented
- ***A relationship is a link between two entities.***
  For example a customer places an order

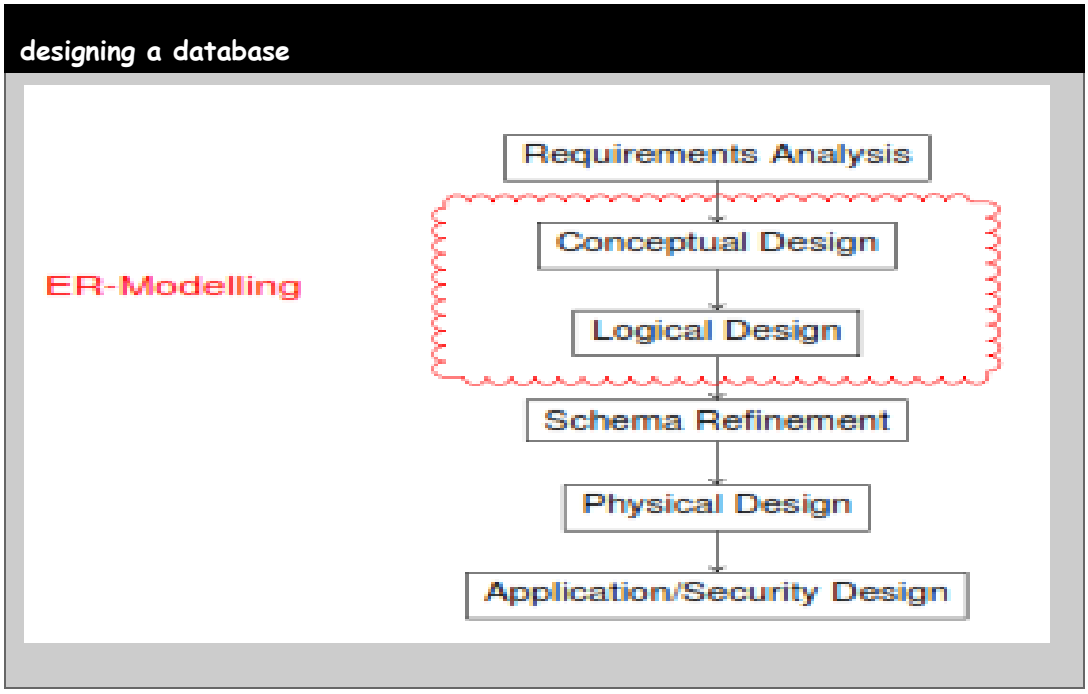


1- Database Design Process

The six main steps in designing a database

- Application & Security Design
- Physical Design
- Schema Refinement

- Logical Design
- Conceptual Design
- Requirements Analysis



☒ ***Requirements Analysis***

Requirements Analysis is the process of determining what the database is to be used for. It involves interviews with user groups and other stakeholders to identify what functionality they require from the database, what kinds of data they wish to process and the most frequently performed operations. This discussion is at a non-technical level and enables the database designers to understand the business logic behind the desired database

☒ ***Conceptual & Logical Design***

Using the ER data model, the conceptual design stage involves identifying the relevant entities and the relationships between them and producing an entity-relationship diagram. The logical design stage involves translating the ER diagram into actual relational database schema. Once a suitable ER model has been constructed, then this can be translated into a relational database fairly easily. However ER

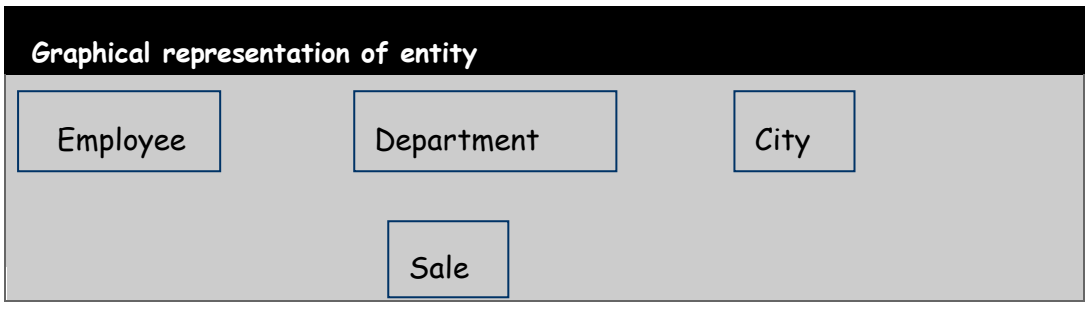
modelling is as much an art as a science, as there are usually many choices to be made and the consequences of each choice sometimes does not become apparent until problems arise later.

2-Basic ER Modelling

The E-R (entity-relationship) data model views the real world as a set of basic objects (entities) and relationships among these objects. It is intended primarily for the DB design process by allowing the specification of an enterprise scheme. This represents the overall logical structure of the DB.

- ✓ ***Entity***
  Real-world object distinguishable from other objects. An entity is described (in DB) using a set of attributes. An entity set represent classes of objects (facts, things, people,...) that have properties in common and an autonomous existence, e.g., City, Department, Employee, Purchase and Sale.

- ✓ ***Entity set***
  An entity is an instance/member of an entity set, e.g., Stockholm, Helsinki, are examples of instances of the entity City; Peterson and Johanson are examples of instances of the Employee entity set



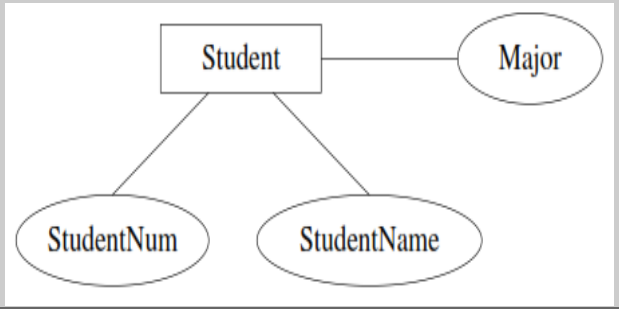
- ✓ ***Attribute***
  properties that describe an Entity's characteristics
  - ***Simple*** - an attribute which cannot be broken down into smaller parts
    Example - Social Security Number
  - ***Composite*** - an attribute which can be broken down into smaller parts
    Example - student's full name

- ***Multi-Valued*** - an attribute that may have more than one value for a given instance. Example - course IS480 has Section A and B...which quarter…which year
- Derived - an attribute whose value can be calculated from other attributes .Example - gross wages = hours * rate…net wages = gross wages Minus Taxes

✓ ***Domain***:

The domain of the attribute is the set of permitted values (Example the telephone number must be seven positive integers).

Graphical representation of attributes



✓ ***Identifier***–

at attribute or combination of attributes that uniquely identifies an entity instance

Example. Student - social security number

Composite Identifier - an identifier consisting of 2 or more attributes

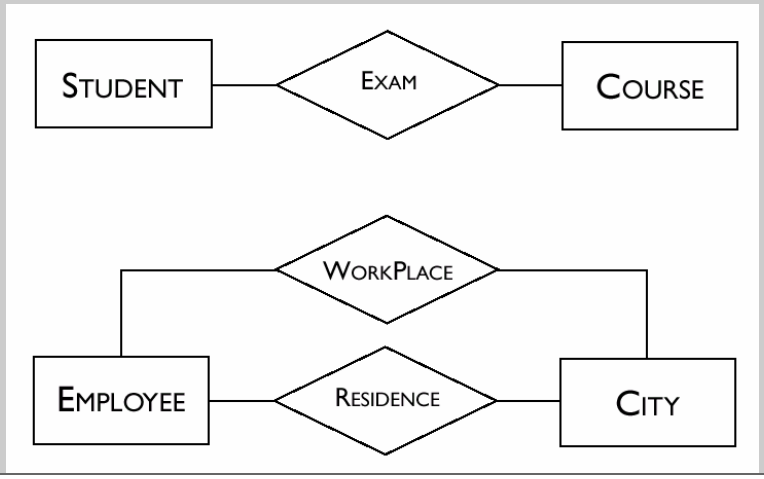
Example. Course Identifier - department (IS); course number (480); section letter (A)

3-Basic Relationship

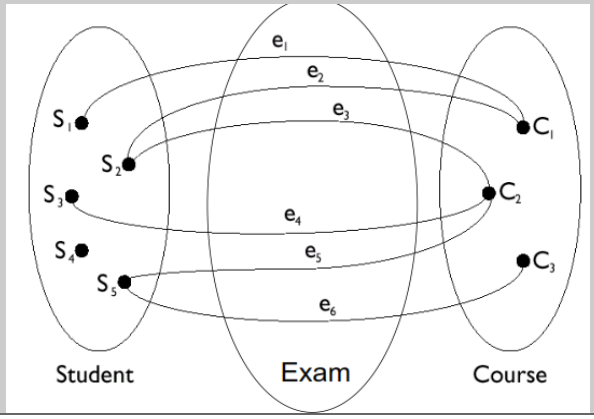
Association among two or more entities or logical links between two or more entities. Relationships are between *entities,* not attributes.

Residence is a relationship set that can exist between entity sets City and Employee; Exam is an example of a relationship that can exist between the entities Student and Course.

Graphical representation of attributes



Example Instances for Exam



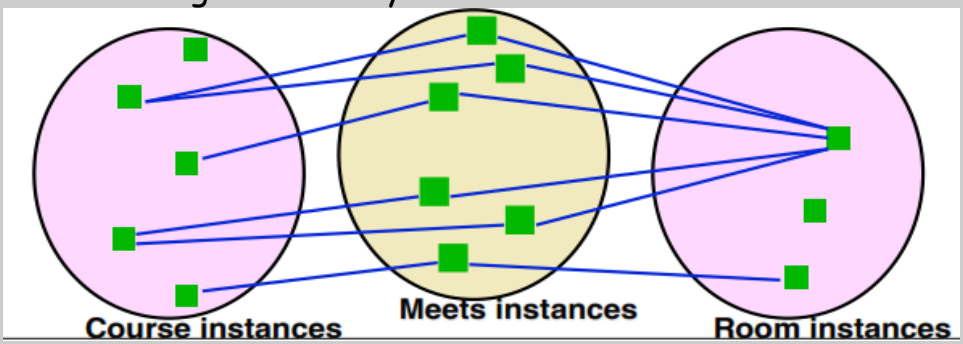
- ✓ ***Relationship Set***

  Collection of similar relationships. An n-ary relationship set R relates n entity sets E1 ... En; each relationship in R involves

entities e1∈ E1, …, en ∈ En Same entity set could participate in different relationship sets, or in different "roles" in same set.

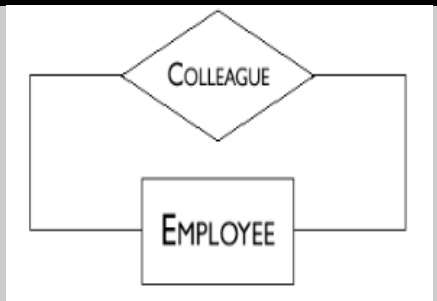
What Does An ER Diagram Really Mean?

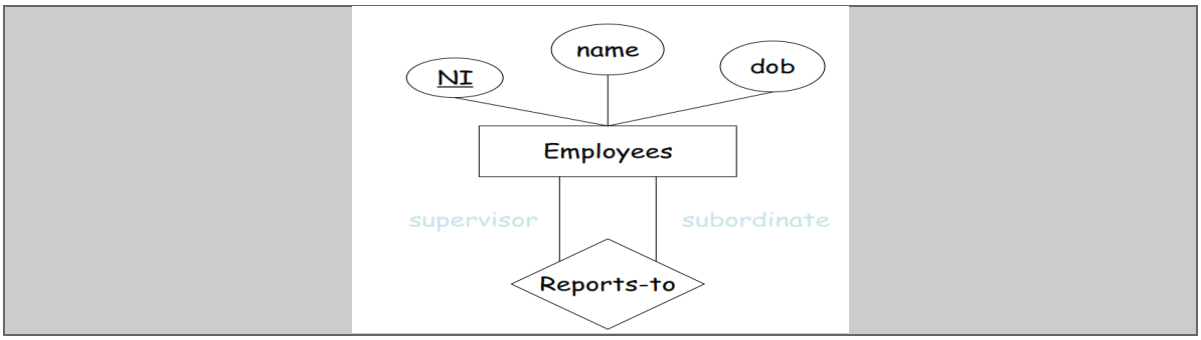
- ✓ ***Course and Room are entities.***
  Their instances are particular courses (eg CSC340F) and rooms (eg MS2172)
- ✓ ***Meets is a relationship.***
  -Its instances describe particular meetings.
  -Each meeting has exactly one associated course and room



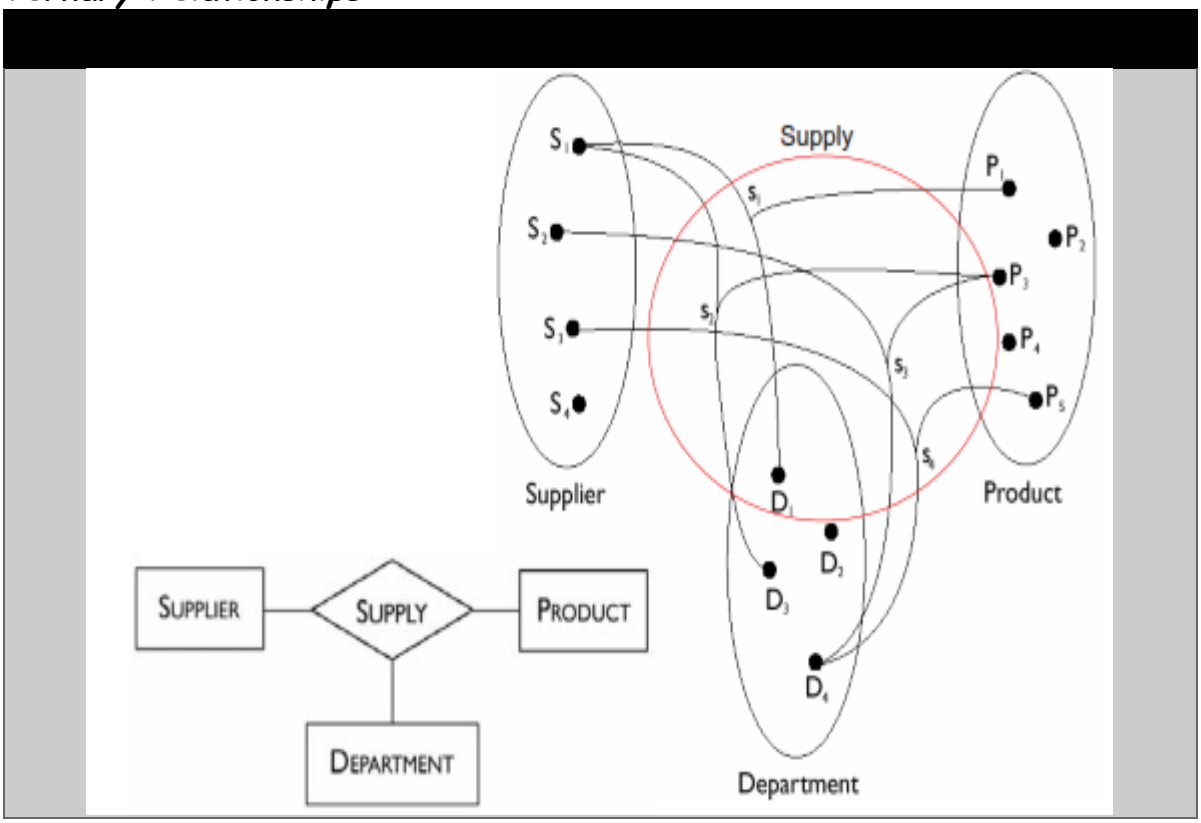
- ✓ ***Recursive Relationships***

An entity can have relationships with itself.



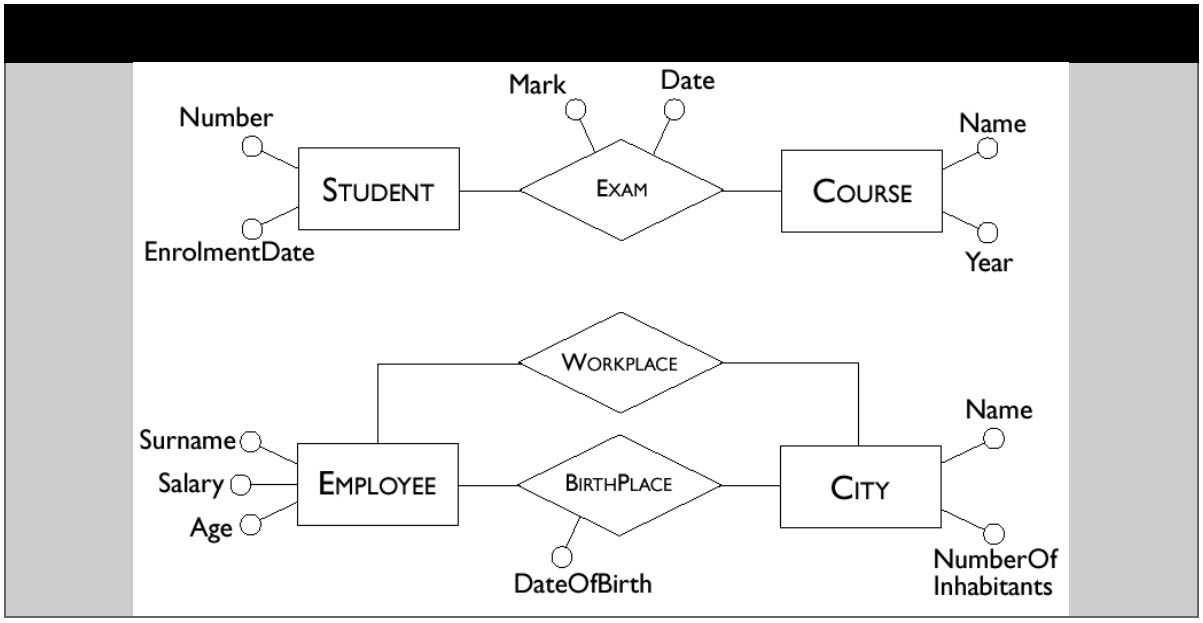


✓ *Ternary Relationships*

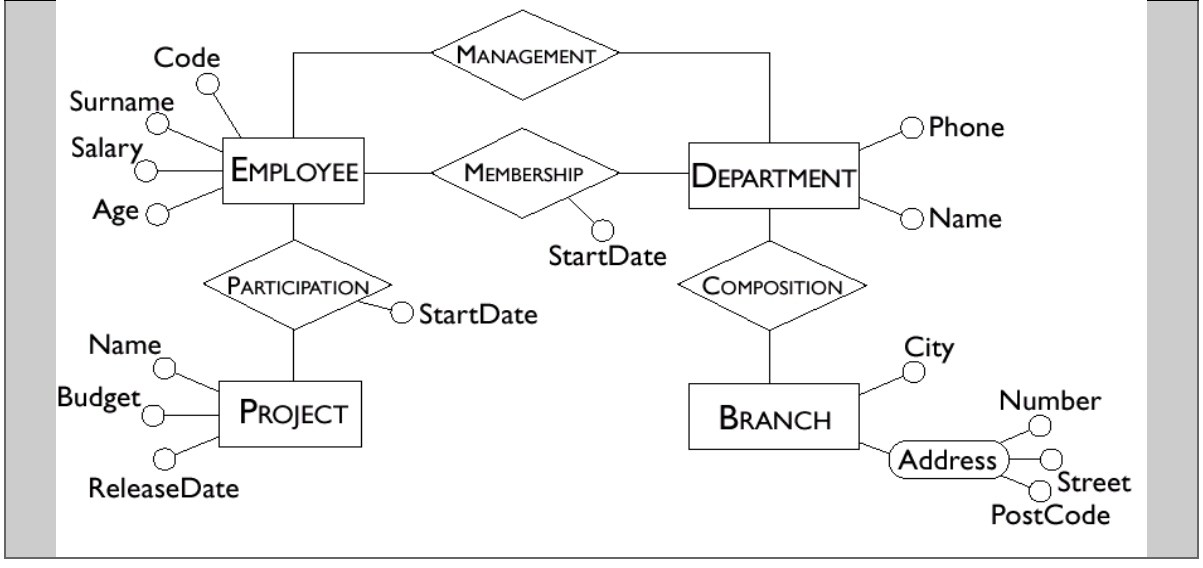


✓ *Attributes*

Associates with each instance of an entity (or relationship) a value belonging to a set (the domain of the attribute).



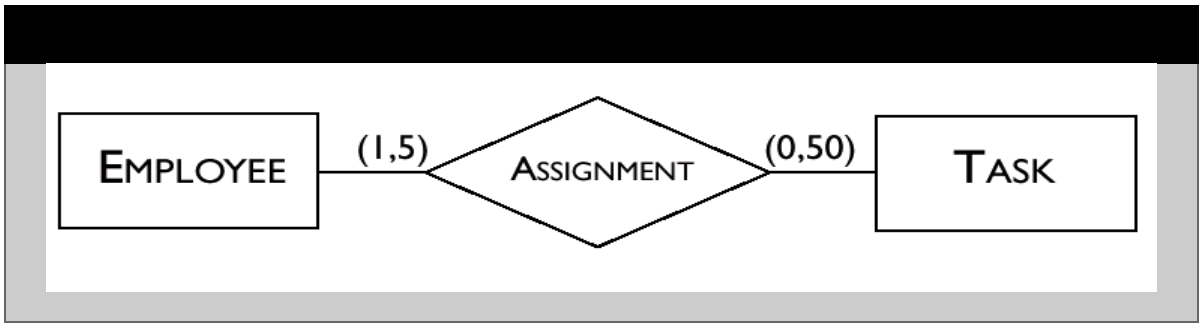
Schema with Attributes



- ✓ ***Cardinalities***

Cardinalities constrain participation in relationships

- ***maximum*** and ***minimum*** number of relationship instances in which an entity
- Instance can participate.

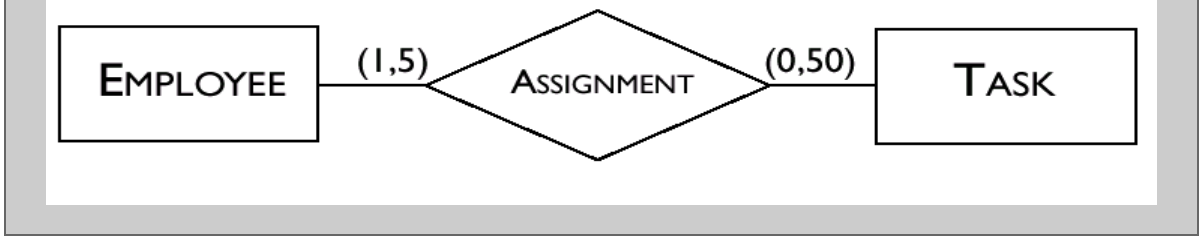


Cardinality is any pair of non-negative integers (a,b)

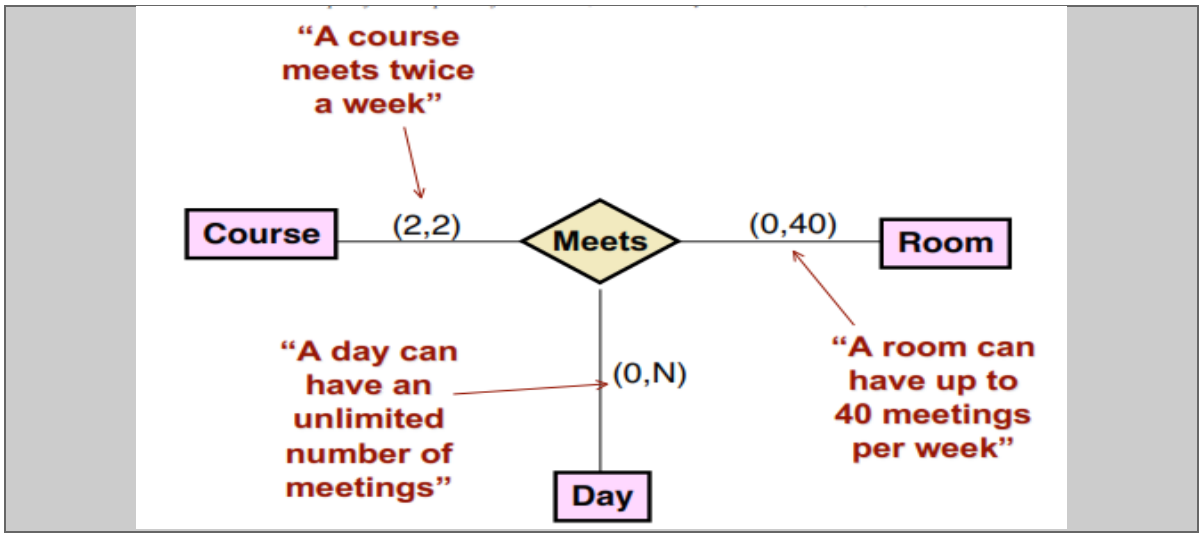
such that a≤b.

- If a=0 then entity participation in a relationship is optional
- If a=1 then entity participation in a relationship is mandatory.
- If b=1 each instance of the entity is associated at most with a single instance of the relationship
- If b="N" then each instance of the entity is associated with an arbitrary number of instances of the relationship.

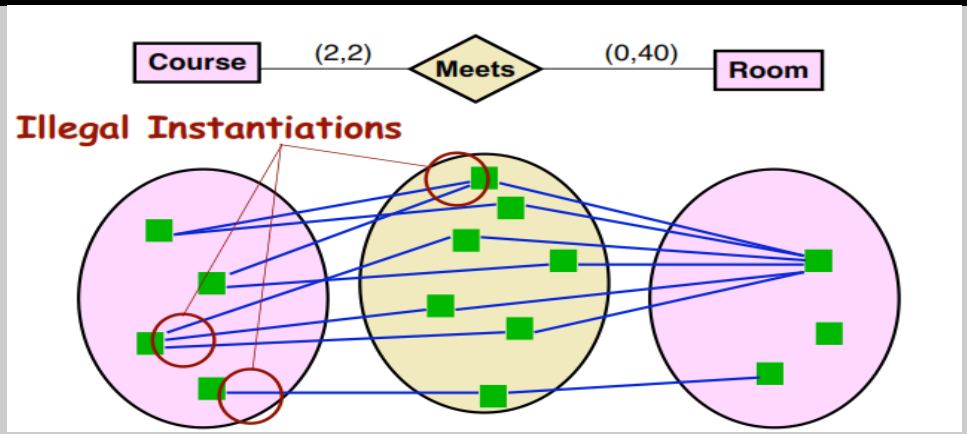
Cardinality Example



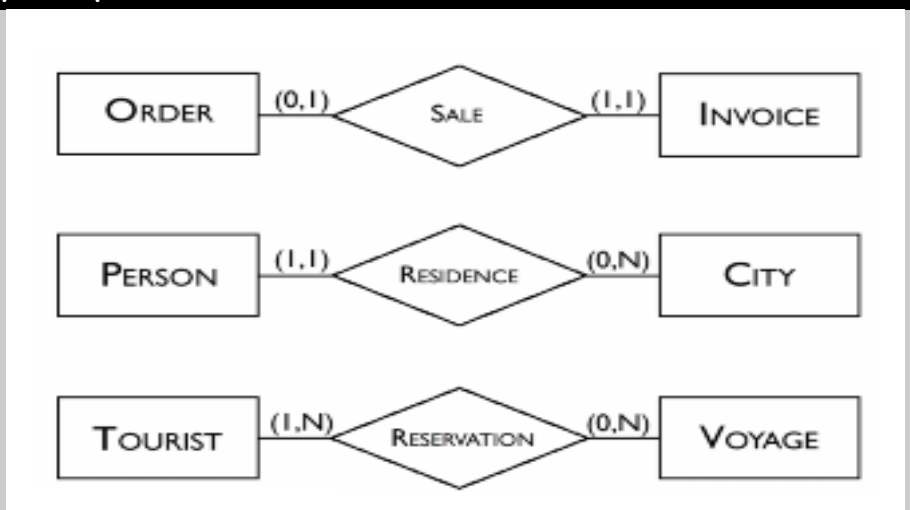
Cardinality Example



Cardinality Example



Cardinality Example



4-Mapping Constraints

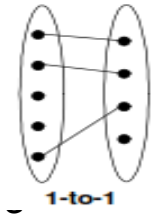
An E-R scheme may define certain constraints to which the contents of a database must conform.

✓ ***Mapping Cardinalities***:

Express the number of entities to which another entity can be associated via a relationship. For binary relationship sets between entity sets A and B, the mapping cardinality must be one of:

1. One-to-one:

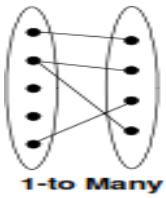
   An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

2. 

Example

-to-many:

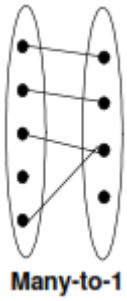
An entity in A is associated with any number in B. An entity in B is associated with at most one entity in A.



Example

3. Many-to-one:

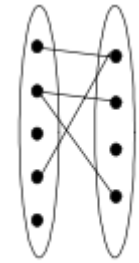
   An entity in A is associated with at most one entity in B. An entity in B is associated with any number in A.



Example

4. Many-to-many:

Entities in A and B are associated with any number from each other.



Example

The appropriate mapping cardinality for a particular relationship set depends on the real world being modeled.

- ✓ ***Existence Dependencies***:
  If the existence of entity X depends on the existence of entity Y, then X is said to be existence dependent on Y. (Or we say that Y is the dominant entity and X is the subordinate entity.)

For example,

- Consider account and transaction entity sets, and a relationship log between them.
- This is one-to-many from account to transaction.
- If an account entity is deleted, its associated transaction entities must also be deleted.
- Thus, account is dominant and transaction is subordinate.

5- Keys

key is an attribute (also known as column or field) or a combination of attribute that is used to identify records. Sometimes we might have to retrieve data

from more than one table, in those cases we require to join tables with the help of keys. The purpose of the key is to bind data together across tables without repeating all of the data in every table.

Differences between entities must be expressed in terms of attributes.

- A superkey is a set of one or more attributes which, taken collectively, allow us to identify uniquely an entity in the entity set. For example, in the entity set customer, customer-name and S.I.N. is a superkey.
- Note that customer-name alone is not, as two customers could have the same name.
- A superkey may contain extraneous attributes, and we are often interested in the smallest superkey. A superkey for which no subset is a superkey is called a ***candidate key***.
- In the example above, S.I.N. is a candidate key, as it is minimal, and uniquely identifies a customer entity.
- A primary key is a candidate key (there may be more than one) chosen by the DB designer to identify entities in an entity set.

✓ ***Superkeys.***
A superkey is any collection of attributes, which uniquely identifies each entity in an entity set.

✓ ***Candidate Keys***.
A candidate key is the smallest combination of attributes that uniquely identifies an entity in an entity set. Multiple candidate keys can exist in an entity set.

✓ ***Primary Keys***:
A primary key is one of the candidate keys of an entity set, chosen to be the chief means of identifying entities in the database.

Note: Only primary keys of entity sets have graphical representation in E-R diagrams (underlined attributes). However, if other important candidate keys exist in an entity set, they must be identified in the conceptual model.

Definition - What does *Primary Key* mean?

A primary key is a special relational database table column (or combination of columns) designated to uniquely identify all table records.

A primary key's main features are:

1. It must contain a unique value for each row of data.
2. It cannot contain null values.

A primary key is either an existing table column or a column that is specifically generated by the database according to a defined sequence.

Definition - What does *Foreign Key* mean?

A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables. It acts as a cross-reference between tables because it references the primary key of another table, thereby establishing a link between them.

The majority of tables in a relational database system adhere to the foreign key concept. In complex databases and data warehouses, data in a domain must be added across multiple tables, thus maintaining a relationship between them. The concept of referential integrity is derived from foreign key theory.

Foreign keys and their implementation are more complex than primary keys.

Example: Consider a Relation or Table R1. Let A,B,C,D,E are the attributes of this relation.

R(A,B,C,D,E)
A→BCDE This means the attribute 'A' uniquely determines the other attributes B,C,D,E.
BC→ADE This means the attributes 'BC' jointly determines all the other attributes A,D,E in the relation.

Primary Key :A

Candidate Keys :A, BC
Super Keys : A,BC,ABC,AD

ABC,AD are not Candidate Keys since both are not minimal super keys.

A Practical Example
Consider the relation 'Store' with the attributes 'storeNo', 'street', 'city' and 'postcode'.

Store (storeNo, street, city, postcode)

given a value of 'city' we may be able to determine several stores (there could be more than one store in a particular city), as such this attribute cannot be used as a candidate key.
if we say store number is a unique number that is given to each store then given a value of 'storeNo' we can only return at most one tuple(because it is a unique identifier) so that is a candidate key.
similarly in this example 'postcode' is also a candidate key because it is unique to each store.

now consider a relation "Sales" which contains information relating to sales at the stores

Sales (customerNo, productNo, saleDate)

given a 'customerNo' there may be several corresponding sales for different products. Similarly given a 'productNo' there may be several customers who have purchased this item. Therefore customer number by its self or product number by its self cannot be selected as a candidate key. However a combination of both 'customerNo' and 'productNo' identifies at one tuple uniquely so this is a candidate key (combining may be refereed to as a composite key but that is just a type of candidate key)

Primary keys ARE candidate keys but as i have shown there is more than one possible choice of a primary key. A candidate key is a CANDIDATE(possible choice) for the primary key of a relation. This may seem unnecessarily complicated, but when designing large relational database systems it is necessary to define candidate keys to ensure you choose an appropriate primary key.

With this in mind, if we look back to the candidate key we created for the 'Sales' relation, It was a combination of attributes customerNo and productNo. That seems like a good candidate key and we could use that as the primary key. However, if one customer could buy the same product more than once on the same day this would make the candidate key return more than one tuple. Therefore the correct candidate key for this relation would include the customerNo, productNo and saleDate attributes. This is an example of a primary key that is both the candidate key and the superkey. Idealy we want to use the least number of attributes possible for a relational key.

. For example, given an employee schema, consisting of the attributes employeeID, name, job, and departmentID, we could use the employeeID in combination with any or all other attributes of this table to uniquely identify a tuple in the table. Examples of superkeys in this schema would be {employeeID, Name}, {employeeID, Name, job}, and {employeeID, Name, job, departmentID}. The last example is known as trivial superkey, because it uses all attributes of this table to identify the tuple.

key is an attribute (also known as column or field) or a combination of attribute that is used to identify records. Sometimes we might have to retrieve data from more than one table, in those cases we require to join tables with the help of keys. The purpose of the key is to bind data together across tables without repeating all of the data in every table.