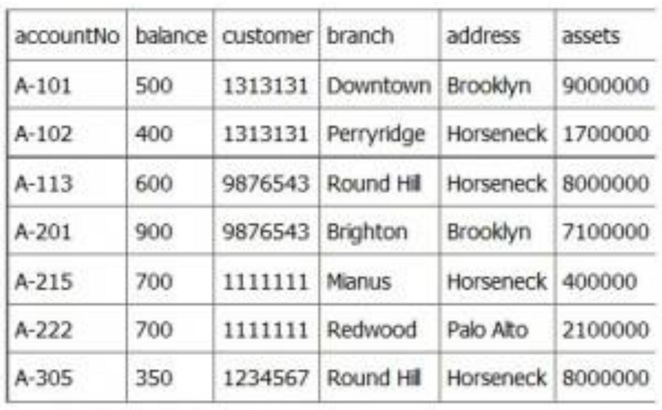
ER Modelling and Normalization

- One important theory developed for the entity relational (ER) involves the notion of functional dependency (FD).
- The aim of studying this is to improve our understanding of relationships among data and to gain enough formalism to assist with practical database design.
- Like constraints, FDs are drawn from the semantics of the application domain.
- Essentially, functional *dependencies* describe how individual attributes are related. FDs are a kind of constraint among attributes within a relation and contribute to a good relational schema design. In this topic, we will look at:
  - The basic theory and definition of functional dependency
  - The methodology for improving schema designs, also called normalization

Relational Design and Redundancy

- Generally, a good relational database design must capture all of the necessary attributes and associations. The design should do this with a minimal amount of stored information and no redundant data.
- In database design, redundancy is generally undesirable because it causes problems maintaining consistency after updates. However, redundancy can sometimes lead to performance improvements; for example, when redundancy can be used in place of a *join* to connect data. A *join* is used when you need to obtain information based on two related tables.
- Consider the Figure below: customer 1313131 is displayed twice, once for account no. A-101 and again for account A-102. In this case, the customer number is not redundant, although there are deletion anomalies with the table. Having a separate customer table would solve this problem. However, if a branch address were to change, it would have to be updated in multiple places. If the customer number was left in the table as is, then you wouldn't need a branch table and no join would be required, and performance is improved.



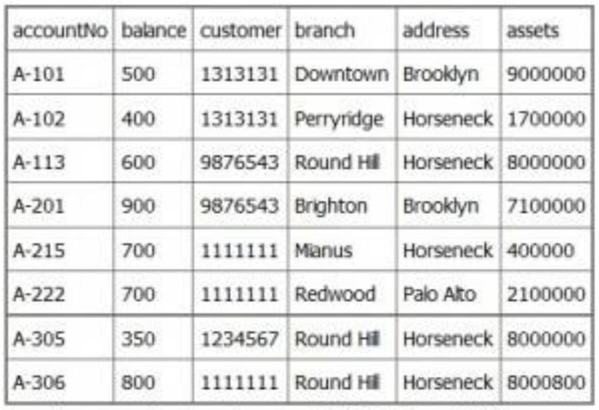
accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000

Bank Accounts

An example of redundancy used with bank accounts and branches.

Insertion Anomaly

- An *insertion anomaly* occurs when you are inserting inconsistent information into a table. When we insert a new record, such as account no. A-306 in Figure below, we need to check that the branch data is consistent with existing rows.



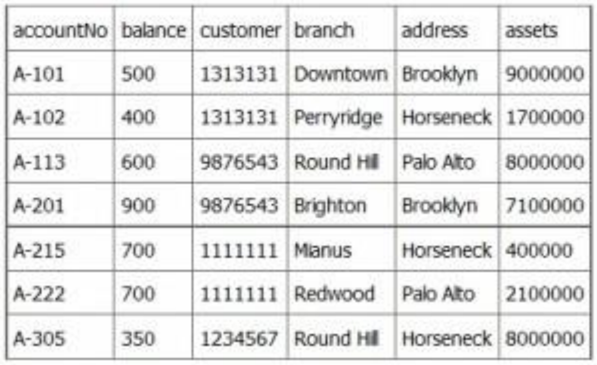
accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Manus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000
A-306	800	1111111	Round Hill	Horseneck	8000800

Insertion anomaly - Insert account A-306 at Round Hill

Example of an insertion anomaly.

Update Anomaly

- If a branch changes address, such as the Round Hill branch in the figure below, we need to update all rows referring to that branch. Changing existing information incorrectly is called an *update anomaly*.



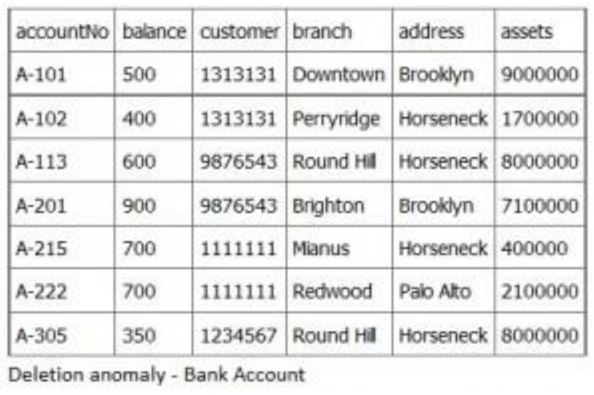
accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Palo Alto	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Manus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000

Update Anomaly - Round Hill branch address

Example of an update anomaly.

Deletion Anomaly

- A *deletion anomaly* occurs when you delete a record that may contain attributes that shouldn't be deleted. For instance, if we remove information about the last account at a branch, such as account A-101 at the Downtown branch in the Figure below all of the branch information disappears.

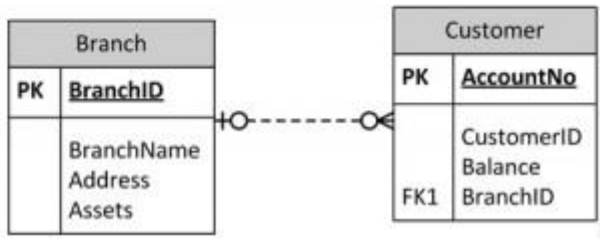


accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000

Deletion anomaly - Bank Account

Example of a deletion anomaly.

- The problem with deleting the A-101 row is we don't know where the Downtown branch is located and we lose all information regarding customer 1313131. To avoid these kinds of update or deletion problems, we need to decompose the original table into several smaller tables where each table has minimal overlap with other tables.
- Each bank account table must contain information about one entity only, such as the Branch or Customer.



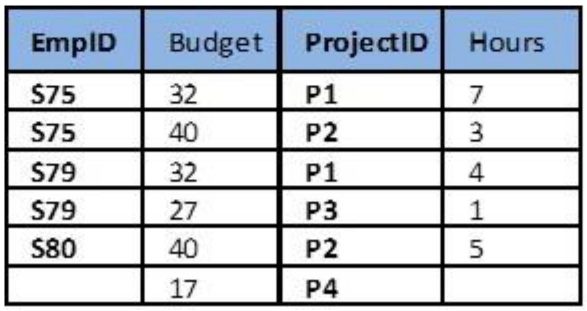
Examples of bank account tables that contain one entity each

- Following this practice will ensure that when branch information is added or updated it will only affect one record. So, when customer information is added or deleted, the branch information will not be accidentally modified or incorrectly recorded.

Example: employee project table and anomalies

The Figure below shows an example of an employee project table. From this table, we can assume that:

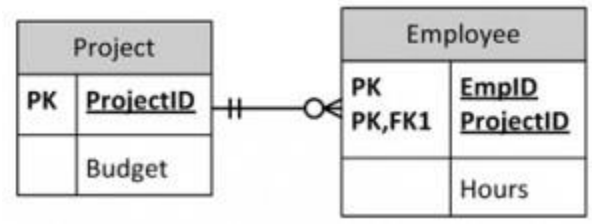
1. EmpID and ProjectID are a composite PK.
2. Project ID determines Budget (i.e., Project P1 has a budget of 32 hours).



EmpID	Budget	ProjectID	Hours
S75	32	P1	7
S75	40	P2	3
S79	32	P1	4
S79	27	P3	1
S80	40	P2	5
	17	P4	

Example of an employee project table

- Next, let's look at some possible anomalies that might occur with this table during the following steps.
  1. **Action**: Add row {S85,35,P1,9}
  2. **Problem**: There are two tuples with conflicting budgets
  3. **Action**: Delete tuple {S79, 27, P3, 1}
  4. **Problem**: Step #3 deletes the budget for project P3
  5. **Action**: Update tuple {S75, 32, P1, 7} to {S75, 35, P1, 7}
  6. **Problem**: Step #5 creates two tuples with different values for project P1's budget
  7. Solution: Create a separate table, each, for Projects and Employees, as shown in the Figure below



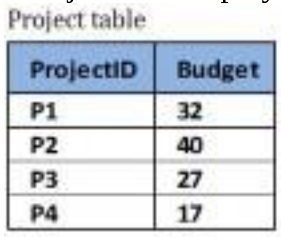
Solution: separate tables for Project and Employee

How to Avoid Anomalies

The best approach to creating tables without anomalies is to ensure that the tables are normalized, and that's accomplished by understanding functional dependencies.

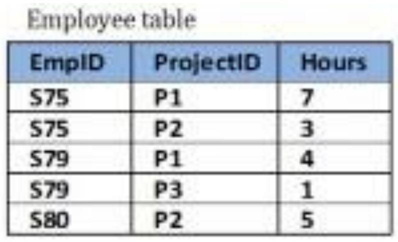
FD ensures that all attributes in a table belong to that table. In other words, it will eliminate redundancies and anomalies.

Example: separate Project and Employee tables



Project table

ProjectID	Budget
P1	32
P2	40
P3	27
P4	17



Employee table

EmpID	ProjectID	Hours
S75	P1	7
S75	P2	3
S79	P1	4
S79	P3	1
S80	P2	5

Separate Project and Employee tables with data

By keeping data separate using individual Project and Employee tables:

1. No anomalies will be created if a budget is changed.
2. No dummy values are needed for projects that have no employees assigned.
3. If an employee's contribution is deleted, no important data is lost.
4. No anomalies are created if an employee's contribution is added.

Functional Dependencies

A *functional dependency* (FD) is a relationship between two attributes, typically between the PK and other non-key attributes within a table.

For any relation R, attribute Y is functionally dependent on attribute X (usually the PK), if for every valid instance of X, that value of X uniquely determines the value of Y. This relationship is indicated by the representation below:

X ————> Y

The left side of the above FD diagram is called the *determinant*, and the right side is the *dependent*.

Examples

1. In the example below, ID determines Name, Address and Birthdate. Given ID, we can determine any of the other attributes within the table.

   ID ————> Name, Address, Birthdate

2. For the second example, ID and Course determine the date completed (DateCompleted). This must also work for a composite PK.

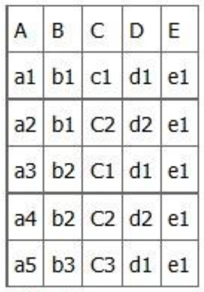
   ID, Course ————> DateCompleted

3. The third example indicates that ISBN determines Title.

   ISBN ————> Title

Rules of Functional Dependencies

Consider the following table of data r(R) of the relation schema R(ABCDE) shown in the Table below.



A	B	C	D	E
a1	b1	c1	d1	e1
a2	b1	C2	d2	e1
a3	b2	C1	d1	e1
a4	b2	C2	d2	e1
a5	b3	C3	d1	e1

Table R

Functional dependency example

What kind of dependencies can we observe among the attributes in Table R?

Since the values of A are unique (a1, a2, a3, etc.), it follows from the FD definition that:

$A \rightarrow B$, $A \rightarrow C$, $A \rightarrow D$, $A \rightarrow E$

- It also follows that $A \rightarrow BC$ (or any other subset of ABCDE).
- This can be summarized as $A \rightarrow BCDE$.
- From our understanding of primary keys, A is a primary key.

Since the values of E are always the same (all e1), it follows that:

$A \rightarrow E$, $B \rightarrow E$, $C \rightarrow E$, $D \rightarrow E$

However, we cannot generally summarize the above with $ABCD \rightarrow E$ because, in general

$A \rightarrow E$, $B \rightarrow E$, $AB \rightarrow E$.

Other observations:
1. Combinations of BC are unique, therefore $BC \rightarrow ADE$.
2. Combinations of BD are unique, therefore $BD \rightarrow ACE$.
3. If C values match, so do D values.
    1. Therefore, $C \rightarrow D$
    2. However, D values don't determine C values
    3. So C does not determine D, and D does not determine C.

Looking at actual data can help clarify which attributes are dependent and which are determinants.

Inference Rules

- *Armstrong's axioms* are a set of inference rules used to infer all the functional dependencies on a relational database. They were developed by William W. Armstrong.
- The following describes what will be used, in terms of notation, to explain these axioms.

Let R(U) be a relation scheme over the set of attributes U. We will use the letters X, Y, Z to represent any subset of and, for short, the union of two sets of attributes, instead of the usual X, U, Y.

Axiom of reflexivity

This axiom says, if Y is a subset of X, then X determines Y

$$\text{If } Y \subseteq X \text{, then } X \rightarrow Y$$

Equation for axiom of reflexivity.

For example, **PartNo —> NT123** where X (PartNo) is composed of more than one piece of information; i.e., Y (NT) and partID (123).

Axiom of augmentation

The axiom of augmentation, also known as a partial dependency, says if X determines Y, then XZ determines YZ for any Z.

$$\text{If } X \rightarrow Y\text{, then } XZ \rightarrow YZ \text{ for any } Z$$

Equation for axiom of augmentation.

- The axiom of augmentation says that every non-key attribute must be fully dependent on the PK.

  In the example shown below, StudentName, Address, City, County, and PC (postal code) are only dependent on the StudentNo, not on the StudentNo and Course.

  StudentNo, Course —> StudentName, Address, City, Prov, PC, Grade, DateCompleted

  This situation is not desirable because every non-key attribute has to be fully dependent on the PK. In this situation, student information is only partially dependent on the PK (StudentNo).

To fix this problem, we need to break the original table down into two as follows:

- Table 1: StudentNo, Course, Grade, DateCompleted
- Table 2: StudentNo, StudentName, Address, City, County, PC
-

Axiom of transitivity

The axiom of transitivity says if X determines Y, and Y determines Z, then X must also determine Z.

$$\text{If } X \rightarrow Y \text{ and } Y \rightarrow Z\text{, then } X \rightarrow Z$$

Equation for axiom of transitivity.

The table below has information not directly related to the student; for instance, ProgramID and ProgramName should have a table of its own. ProgramName is not dependent on StudentNo; it’s dependent on ProgramID.

StudentNo —> StudentName, Address, City, Prov, PC, ProgramID, ProgramName

This situation is not desirable because a non-key attribute (ProgramName) depends on another non-key attribute (ProgramID).

To fix this problem, we need to break this table into two: one to hold information about the **student** and the other to hold information about the **program**.

- Table 1: StudentNo —> StudentName, Address, City, Prov, PC, ProgramID
- Table 2: ProgramID —> ProgramName

However, we still need to leave an FK in the student table so that we can identify which program the student is enrolled in.

Union

This rule suggests that if two tables are separate, and the PK is the same, you may want to consider putting them together.

It states that if X determines Y and X determines Z then X must also determine Y and Z.

$$\text{If } X \rightarrow Y \text{ and } X \rightarrow Z \text{ then } X \rightarrow YZ$$

Equation for the Union rule.

For example, if:

- ID —> EmpName
- ID —> SpouseName

You may want to join these two tables into one as follows:

ID –> EmpName, SpouseName

Some database administrators (*DBA*) might choose to keep these tables separated for a couple of reasons.

1. One, each table describes a different entity so the entities should be kept apart.
2. Two, if SpouseName is to be left NULL most of the time, there is no need to include it in the same table as EmpName.

Decomposition

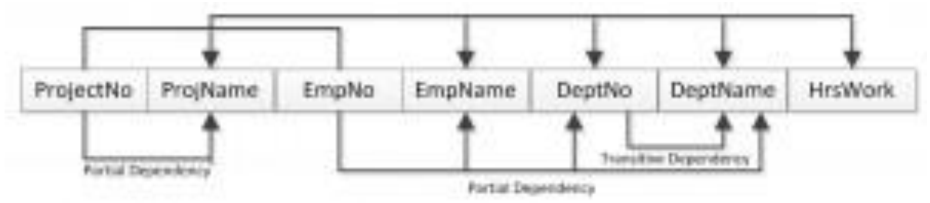
- Decomposition is the reverse of the Union rule.
- If you have a table that appears to contain two entities that are determined by the same PK, consider breaking them up into two tables. This rule states that if X determines Y and Z, then X determines Y and X determines Z separately

$$\text{If } X \rightarrow YZ \text{ then } X \rightarrow Y \text{ and } X \rightarrow Z$$

Equation for decompensation rule.

Dependency Diagram

- A dependency diagram illustrates the various dependencies that might exist in a *non-normalized table*.
- A non-normalized table is one that has data redundancy in it.



Dependency diagram

The following dependencies are identified in this table:

- ProjectNo and EmpNo, combined, are the PK.
- Partial Dependencies:
  - ProjectNo —> ProjName
  - EmpNo —> EmpName, DeptNo,
  - ProjectNo, EmpNo —> HrsWork
- Transitive Dependency:
  - DeptNo —> DeptName

Normalization

- Normalization should be part of the database design process. However, it is difficult to separate the normalization process from the ER modelling process so the two techniques should be used concurrently.
- Use an entity relation diagram (ERD) to provide the big picture, or macro view, of an organization's data requirements and operations. This is created through an iterative process that involves identifying relevant entities, their attributes and their relationships.
- If a database design is not perfect, it may contain anomalies. Managing a database with anomalies is next to impossible.
  - **Update anomalies** − If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, tring to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.
  - **Deletion anomalies** − Trying to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.
  - **Insert anomalies** − Trying to insert data in a record that does not exist at all.
- Normalization procedure focuses on characteristics of specific entities and represents the micro view of entities within the ERD.

What Is Normalization?

- *Normalization* is the branch of relational theory that provides design insights.
- It is the process of determining how much redundancy exists in a table.
- Normalization is a method used to remove all these anomalies and bring the database to a consistent state.
- The goals of normalization are to:
  - Be able to characterize the level of redundancy in a relational schema
  - Provide mechanisms for transforming schemas in order to remove redundancy

- Normalization theory draws heavily on the theory of functional dependencies.
- Normalization theory defines four normal forms (NF). Each normal form involves a set of dependency properties that a schema must satisfy and each normal form gives guarantees about the presence and/or absence of update anomalies. This means that higher normal forms have less redundancy, and as a result, fewer update problems.

Normal Forms

- All the tables in any database can be in one of the four normal forms. Ideally we only want minimal redundancy for PK to FK. Everything else should be derived from other tables.
- There are six normal forms, but we will only look at the first four, which are:
  - First normal form (1NF)
  - Second normal form (2NF)
  - Third normal form (3NF)
  - Boyce-Codd normal form (BCNF)
- BCNF is rarely used.

First Normal Form (1NF)

- In the *first normal form*, only single values are permitted at the intersection of each row and column; hence, there are no repeating groups.
- To normalize a relation that contains a repeating group, remove the repeating group and form two new relations.
- The PK of the new relation is a combination of the PK of the original relation plus an attribute from the newly created relation for unique identification.

Process for 1NF

Let us use the **Student_Grade_Report** table below, from a University database, as our example to explain the process for 1NF.

Student_Grade_Report (StudentNo, StudentName, Programme, CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation, Grade)

- In the Student Grade Report table, the repeating group is the course information. A student can take many courses.
- Remove the repeating group. In this case, it's the course information for each student.
- Identify the PK for your new table.
- The PK must uniquely identify the attribute value (StudentNo and CourseNo).
- After removing all the attributes related to the course and student, you are left with the student course table (**StudentCourse**).
- The Student table (**Student**) is now in first normal form with the repeating group removed.
- The two new tables are shown below.

  Student (StudentNo, StudentName, Programme)
  StudentCourse (StudentNo, CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation, Grade)

How to update 1NF anomalies

StudentCourse (<u>StudentNo, CourseNo</u>, CourseName, InstructorNo, InstructorName, InstructorLocation, Grade)

- To add a new course, we need a student.
- When course information needs to be updated, we may have inconsistencies.
- To delete a *student*, we might also delete critical information about a course.

Second Normal Form (2NF)

- For the *second normal form*, the relation must first be in 1NF.
- The relation is automatically in 2NF if, and only if, the PK comprises a single attribute.
- If the relation has a composite PK, then each non-key attribute must be fully dependent on the entire PK and not on a subset of the PK (i.e., there must be no partial dependency or augmentation).

Process for 2NF

To move to 2NF, a table must first be in 1NF.

- The Student table is already in 2NF because it has a single-column PK.
- When examining the Student Course table, we see that not all the attributes are fully dependent on the PK; specifically, all course information. The only attribute that is fully dependent is grade.
- Identify the new table that contains the course information.
- Identify the PK for the new table.
- The three new tables are shown below.

Student (StudentNo, StudentName, Programme)
CourseGrade (<u>StudentNo, CourseNo</u>, Grade)
CourseInstructor (CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation)

How to update 2NF anomalies

- When adding a new instructor, we need a course.
- Updating course information could lead to inconsistencies for instructor information.
- Deleting a course may also delete instructor information.

Third Normal Form (3NF)

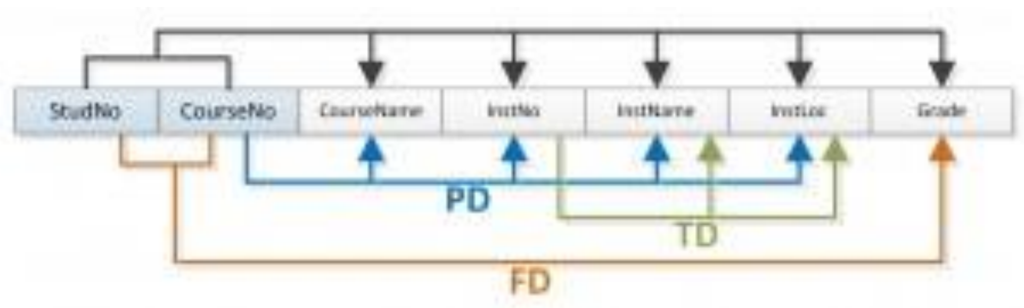
To be in *third normal form*, the relation must be in second normal form. Also all transitive dependencies must be removed; a non-key attribute may not be functionally dependent on another non-key attribute.

Process for 3NF

- Eliminate all dependent attributes in transitive relationship(s) from each of the tables that have a transitive relationship.
- Create new table(s) with removed dependency.

- Check new table(s) as well as table(s) modified to make sure that each table has a determinant and that no table contains inappropriate dependencies.
- See the four new tables below.
  Student (StudentNo, StudentName, Programme)
  CourseGrade (<u>StudentNo, CourseNo</u>, Grade)
  Course (<u>CourseNo</u>, CourseName, InstructorNo)
  Instructor (InstructorNo, InstructorName, InstructorLocation)

At this stage, there should be no anomalies in third normal form. Let's look at the dependency diagram below for this example. The first step is to remove repeating groups, as discussed above.



Dependency diagram

Student (StudentNo, StudentName, Programme)
StudentCourse (StudentNo, CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation, Grade)

To recap the normalization process for the University database, review the dependencies shown in diagram above

The abbreviations used are as follows:

- **PD**: partial dependency
- **TD**: transitive dependency
- **FD**: full dependency (Note: FD typically stands for **functional** dependency).

Boyce-Codd Normal Form (BCNF)

When a table has more than one candidate key, anomalies may result even though the relation is in 3NF.

Boyce-Codd normal form is a special case of 3NF. A relation is in BCNF if, and only if, every determinant is a candidate key.

BCNF Example 1

Consider the following table (**St_Maj_Adv**).

Student_id	**Major**	**Advisor**
111	Physics	Smith
111	Music	Chan
320	Math	Dobbs
671	Physics	White
803	Physics	Smith

The *semantic rules* (business rules applied to the database) for this table are:
1. Each Student may major in several subjects.
2. For each Major, a given Student has only one Advisor.
3. Each Major has several Advisors.
4. Each Advisor advises only one Major.
5. Each Advisor advises several Students in one Major.

The functional dependencies for this table are listed below. The first one is a candidate key; the second is not.

1. Student_id, Major ——> Advisor
2. Advisor ——> Major

Anomalies for this table include:
- Delete – student deletes advisor information
- Insert – a new advisor needs a student
- Update – inconsistencies

Note: No single attribute is a candidate key.

PK can be Student_id, Major or Student_id, Advisor**.**

To reduce the **St_Maj_Adv** relation to BCNF, you create two new tables:
1. **St_Adv** (Student_id, Advisor)
2. **Adv_Maj** (Advisor, Major)

St_Adv table

Student_id	**Advisor**
111	Smith
111	Chan
320	Dobbs
671	White
803	Smith

Adv_Maj table

Advisor	**Major**
Smith	Physics
Chan	Music
Dobbs	Math
White	Physics

BCNF Example 2

Consider the following table (**Client_Interview)**.

ClientNo	**InterviewDate**	**InterviewTime**	**StaffNo**	**RoomNo**
CR76	13-May-02	10.30	SG5	G101
CR56	13-May-02	12.00	SG5	G101
CR74	13-May-02	12.00	SG37	G102
CR56	1-July-02	10.30	SG5	G102

FD1 – ClientNo, InterviewDate –> InterviewTime, StaffNo, RoomNo (PK)

FD2 – staffNo, interviewDate, interviewTime –> clientNO (candidate key: CK)

FD3 – roomNo, interviewDate, interviewTime –> staffNo, clientNo (CK)

FD4 – staffNo, interviewDate –> roomNo

A relation is in BCNF if, and only if, every determinant is a candidate key. We need to create a table that incorporates the first three FDs (**Client_Interview2** table) and another table (**StaffRoom** table) for the fourth FD.

Client_Interview2 table

ClientNo	**InterviewDate**	**InterViewTime**	**StaffNo**
CR76	13-May-02	10.30	SG5
CR56	13-May-02	12.00	SG5
CR74	13-May-02	12.00	SG37
CR56	1-July-02	10.30	SG5

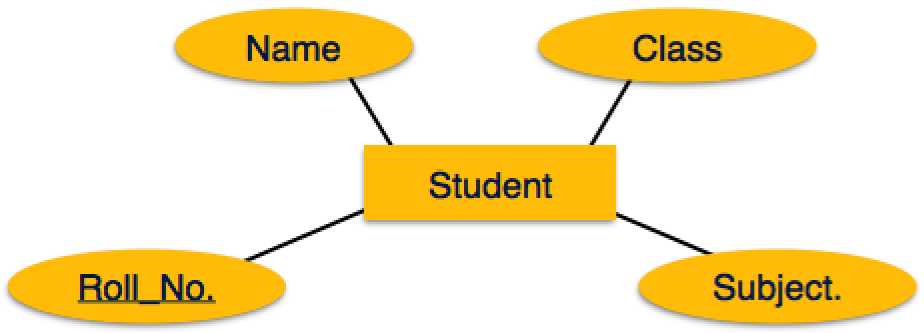
StaffRoom table

StaffNo	**InterviewDate**	**RoomNo**
SG5	13-May-02	G101
SG37	13-May-02	G102
SG5	1-July-02	G102

Normalization and Database Design
During the normalization process of database design, make sure that proposed entities meet required normal form before table structures are created. Many real-world databases have been improperly designed or burdened with anomalies if improperly modified during the course of time. You may be asked to redesign and modify existing databases. This can be a large undertaking if the tables are not properly normalized.

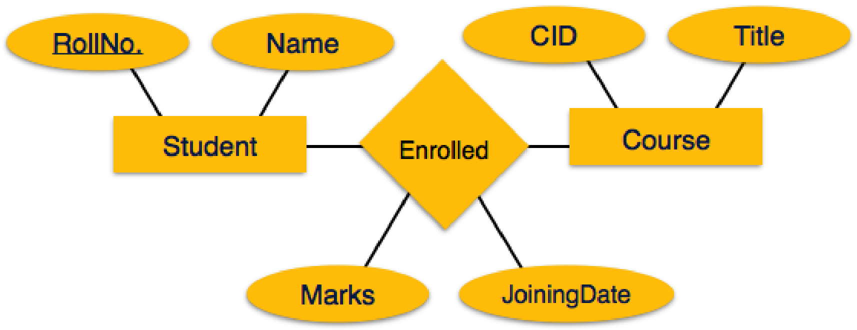
ER Model to Relational Model
- ER Model, when conceptualized into diagrams, gives a good overview of entity-relationship, which is easier to understand. ER diagrams can be mapped to relational schema, that is, it is possible to create relational schema using ER diagram. We cannot import all the ER constraints into relational model, but an approximate schema can be generated.
- There are several processes and algorithms available to convert ER Diagrams into Relational Schema. Some of them are automated and some of them are manual.
- ER diagrams mainly comprise of:
  - Entity and its attributes
  - Relationship, which is association among entities.

Mapping Entity
An entity is a real-world object with some attributes.



Mapping Process (Algorithm)
- Create table for each entity.
- Entity's attributes should become fields of tables with their respective data types.
- Declare primary key.

Mapping Relationship
A relationship is an association among entities.

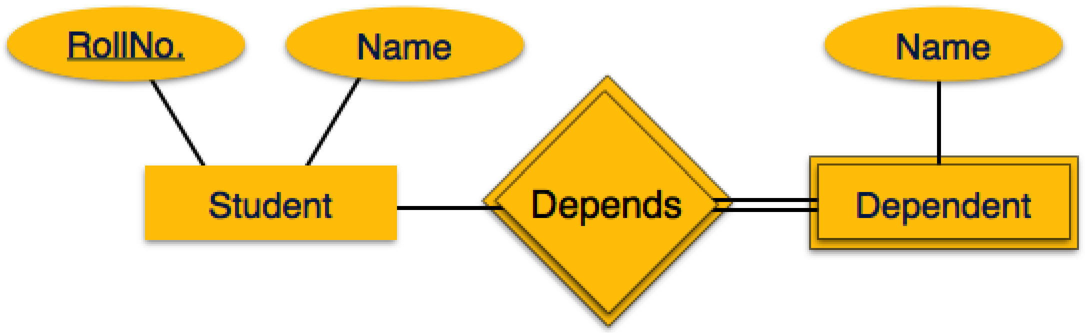


Mapping Process

- Create table for a relationship.
- Add the primary keys of all participating Entities as fields of table with their respective data types.
- If relationship has any attribute, add each attribute as field of table.
- Declare a primary key composing all the primary keys of participating entities.
- Declare all foreign key constraints.

Mapping Weak Entity Sets

A weak entity set is one which does not have any primary key associated with it.

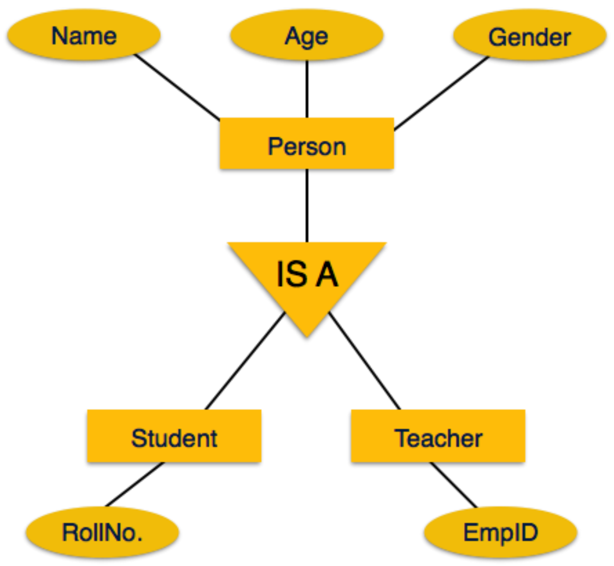


Mapping Process

- Create table for weak entity set.
- Add all its attributes to table as field.
- Add the primary key of identifying entity set.
- Declare all foreign key constraints.

Mapping Hierarchical Entities

ER specialization or generalization comes in the form of hierarchical entity sets.



Mapping Process

- Create tables for all higher-level entities.
- Create tables for lower-level entities.
- Add primary keys of higher-level entities in the table of lower-level entities.
- In lower-level tables, add all other attributes of lower-level entities.
- Declare primary key of higher-level table and the primary key for lower-level table.
- Declare foreign key constraints.

Review Questions

1. Normalize the database below

Attribute Name	Sample Value	Sample Value	Sample Value
StudentID	1	2	3
StudentName	John Smith	Sandy Law	Sue Rogers
CourseID	2	2	3
CourseName	Programming Level 1	Programming Level 1	Business
Grade	75%	61%	81%
CourseDate	Jan 5th, 2014	Jan 5th, 2014	Jan 7th, 2014

2. Create a logical ERD for an online movie rental service (no many to many relationships). Use the following description of operations on which your business rules must be based: The online movie rental service classifies movie titles according to their type: comedy, western, classical, science fiction, cartoon, action, musical, and new release. Each type contains many possible titles, and most titles within a type are available in multiple copies.

   For example, note the following summary:

   TYPE TITLE: Musical
   Papa Ogungo (Copy 1)
   Papa Ogungo (Copy 2)

Sun in the city (Copy 1)
Sun in the city (Copy 2)
Sun in the city (Copy 3)
etc.

3. What three data anomalies are likely to be the result of data redundancy? How can such anomalies be eliminated?