Chapter 5
Functional dependency and Normalization

- Functional dependency is a relationship of one attribute or field in a record to another.
- In DB, we often have the case where one field *defines* the other.

Eg1: Social Security Number (SSN) defines a name.

- if I know someone's SSN, then I can find their name.
- we will say that we have *defined name as being functionally dependent on SSN.*

Functional dependency

Eg2: suppose that a company assigned each employee a unique EmpNo. Each employee has a number and a name. Names might be the same for two different employees, but their employee numbers would always be different and unique because the company defined them that way.

- It would be inconsistent in the database if there were two occurrences of the same employee number with different names.

➢ We write a functional dependency (FD) connection with an arrow: SSN → Name

EmpNo → Name.

❖ The expression SSN → Name is read as "SSN defines Name" or "SSN implies Name."

Functional dependency

Eg:

EmpNo	**Job**	**Name**
101	President	Herbert
104	Programmer	Fred
103	Designer	Beryl
103	Programmer	Beryl

- Is there a problem here? No.
- We have the FD that EmpNo → Name. This means that every time we find 104, we find the name, Fred.

Just because something is on the left-hand side of a FD, it does not imply that you have a key or that it will be unique in the database. i.e the FD $X \rightarrow Y$ only means that for every occurrence of X you will get the same value of Y.

Functional dependency

- Eg:going back to the *SSN → Name* example and add a couple more attributes.

SSN	**Name**	**School**	**Location**
101	David	Alabama	Tuscaloosa
102	Chrissy	MSU	Starkville
103	Kaitlyn	LSU	Baton Rouge
104	Stephanie	MSU	Starkville
105	Lindsay	Alabama	Tuscaloosa
106	Chloe	Alabama	Tuscaloosa

Here, we will define two FDs:

1. SSN → Name and School → Location.
2. SSN → School.

➢First, have we violated any FDs with our data? Because all SSNs are unique, there cannot be a FD violation of SSN → Name. Why? Because a FD X → Y says that given some value for X, you always get the same Y. Because the X's are unique, you will always get the same value. The same comment is true for SSN → School.

Functional dependency

- How about our second FD, School→ Location? There are only three schools in the example and you may note that for every school, there is only one location, so no FD violation.
- Now, we want to point out something interesting. If we define a functional dependency *X* → *Y* and we define a functional dependency Y → Z, then we know by inference that X → Z.
- Here, we defined *SSN → School*. We also defined *School → Location*, so we can *infer that SSN → Location* although that FD was not originally mentioned.
- The inference we have illustrated is called *the* **transitivity rule of FD inference**. *Here is the transitivity* rule restated:

  Given X → Y

  Given Y → Z

  Then X → Z

Functional dependency

- To see that the FD SSN→ Location is true in our data, you can note that given any value of SSN, you always find a unique location for that person.
- Another way to demonstrate that the transitivity rule is true is to try to invent a row where it is not true and then see if you violate any of the defined FDs.
- We defined these FD's:

  Given: SSN → Name

  SSN → School

  School → Location
- We are claiming by inference using the **transitivity rule** that SSN→ Location.

Functional dependency

There are other inference rules for functional dependencies.

A. ***The Reflexive Rule*** If X is a composite, composed of A and B, then X→ A and X→ B.

Eg: X= Name, City. Then we are saying that $X \rightarrow Name$ and $X \rightarrow City$.

Example:

Name	**City**
David	Mobile
Kaitlyn	New Orleans
Chrissy	Baton Rouge

The rule, which seems quite obvious, says if I give you the combination <Kaitlyn, New Orleans>, what is this person's Name? What is this person's City? While this rule seems obvious enough, it is necessary to derive other functional dependencies.

Functional dependency

B. The Augmentation Rule If X→ Y, then XZ→ Y. You might call this rule, "more information is not really needed, but it doesn't hurt." Suppose we use the same data as before with Names and Cities, and define the FD Name → City.

Now, suppose we add a column, Shoe Size:

Name	City	Shoe Size
David	Mobile	10
Kaitlyn	New Orleans	6
Chrissy	Baton Rouge	3

Now, I claim that because Name→ City, that Name + Shoe Size → City (i.e., we augmented Name with Shoe Size).

Will there be a contradiction here, ever? No, because we defined Name → City, Name plus more information will always identify the unique City for that individual. We can always add information to the LHS of an FD and still have the FD be true.

Functional dependency

C. The Decomposition Rule The decomposition rule says that if it is given that X → YZ (that is, X defines both Y and Z), then X → Y and X → Z.

example:

Name	**City**	**Shoe Size**
David	Mobile	10
Kaitlyn	New Orleans	6
Chrissy	Baton Rouge	3

Suppose I define Name → City, Shoe Size. This means for every occurrence of Name, I have a unique value of City and a unique value of Shoe Size.

The rule says that given Name → City and Shoe Size together, then Name → City and Name → Shoe Size. A partial proof using the reflexive rule would be:

Name → City, Shoe Size (given)
City, Shoe Size → City (by the reflexive rule)
Name → City (using steps 1 and 2 and the transitivity rule)

Functional dependency

D. The Union Rule The union rule is the reverse of the decomposition rule in that if X → Y and X → Z, then X → YZ.

- The same example of Name, City, and Shoe Size illustrates the rule. If we found independently or were given that Name →City and Name → Show Size, we can immediately write Name→ City, Shoe Size.

- You might be a little troubled with this example in that you may say that Name is not a reliable way of identifying City; Names might not be unique. You are correct in that Names may not ordinarily be unique, but note the language we are using. In this database, we *define that Name → City and,*hence, in this database are restricting Name to be unique by definition.

Functional dependency

Functional dependencies (FDs)

- Are used to specify *formal measures* of the "goodness" of relational designs
- And keys are used to define **normal forms** for relations
- Are **constraints** that are derived from the *meaning* and *interrelationships* of the data attributes

❖***Functional Dependency*** The value of one attribute in a table is determined entirely by the value of another.

Functional dependency

- **Full Dependency** In a relation, the attribute(s) B is fully functional dependent on A if B is functionally dependent on A, but not on any proper subset of A.
- **Partial Dependency** A type of functional dependency where an attribute is functionally dependent on only part of the primary key (primary key must be a composite key).

  Eg: <u>SalesOrderNo</u>, <u>ItemNo</u>, Qty, UnitPrice
- **Transitive Dependency** In a relation, if attribute(s) A→B and B→C, then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C)

  Eg: Staff_No→Branch_No and Branch_No→BAddress

Keys and FDs

- The main reason we identify the FDs and inference rules is to be able to find keys and develop normal forms for relational databases.
- In any relational table, we want to find out which, if any attribute(s), will identify the rest of the attributes. An attribute that will identify all the other attributes in row is called a "candidate key." A key means a ‘unique identifier’ for a row of inform
- Hence, if an attribute or some combination of attributes will always identify all the other attributes in a row, it is a "candidate" to be "named" a key.

Keys and FDs

Example

SSN	Name	School	Location
101	David	Alabama	Tuscaloosa
102	Chrissy	MSU	Starkville
103	Kaitlyn	LSU	Baton Rouge
104	Stephanie	MSU	Starkville
105	Lindsay	Alabama	Tuscaloosa
106	Chloe	Alabama	Tuscaloosa

suppose I define the following fFDs:

SSN → Name

SSN → School

School → Location

SSN → Name **(given)**

SSN → School **(given)**

SSN → Location **(derived by the transitive rule)**

SSN → SSN **(reflexive rule (obvious))**

SSN → SSN, Name, School, Location **(union rule)**

So SSN can be a candidate key and primary key as well.

Keys and FDs

- Keys should be a minimal set of attributes whose closure is all the attributes in the relation — "minimal" in the sense that you want the fewest attributes on the LHS of the FD that you choose as a key.
- In our example, SSN will be minimal (one attribute), whose closure includes all the other attributes.

- Once we have found a set of candidate keys (or perhaps only one as in this case), we designate one of the candidate keys as the primary key and move on to normal forms.

Normalization

- Dr. Codd discovered that unnormalized relations presented certain problems when attempts were made to update the data in them.
- Information is stored redundantly and Wastes storage, Errors and/or inconsistencies will appear because of the redundant data

He used the term anomalies for these problems.

The reason we normalize the relations is to remove these anomalies from the data.

There are three types of anomalies:

- Insertion anomalies
- Deletion anomalies
- Update anomalies

Normalization

- The update anomaly: refers to a situation where an update of a single data value requires multiple tuples (rows) of data to be updated.
- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Update Anomaly:
  - Changing the name of project number P1 from “Billing” to “Customer-Accounting” may cause this update to be made for all 100 employees working on project P1.

Normalization

- The insert anomaly: refers to a situation where in one cannot insert a new tuple into a relation because of an artificial dependency on another relation.
- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Insert Anomaly:
  - Cannot insert a project unless an employee is assigned to it.
- Conversely
  - Cannot insert an employee unless an he/she is assigned to a project.

Normalization

- The deletion anomaly: refers to a situation where in a deletion of data about one particular entity causes unintended loss of data that characterizes another entity.
- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Delete Anomaly:
  - When a project is deleted, it will result in deleting all the employees who work on that project.
  - Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.

Normalization

- Guideline I: Informally, each tuple in a relation should represent one entity or relationship instance. (Applies to individual relations and their attributes).
  - Attributes of different entities (EMPLOYEEs, DEPARTMENTs, PROJECTs) should not be mixed in the same relation
  - Only foreign keys should be used to refer to other entities
  - Entity and relationship attributes should be kept apart as much as possible.
- Guideline II:
  - Design a schema that does not suffer from the insertion, deletion and update anomalies.
  - If there are any anomalies present, then note them so that applications can be made to take them into account.

Normalization

Normalization is the process of organizing data in a database. This includes creating tables and establishing relationships between those tables according to rules designed both to *protect the data* and to make the database more *flexible* by eliminating two factors: *redundancy* and *inconsistent dependency*.

- Normalization is the *analysis of FDs* between attributes.
- Is process of decomposing relations with anomalies to produce well-structured relations.
- Well-structured relation contains minimal redundancy and allows insertion, modification, and deletion without errors or inconsistencies.

Normalization

- Normalization theory is based on the concepts of normal forms.
- A relational table is said to be a particular normal form if it satisfied a certain set of constraints.

- Edgar F. Codd originally established three normal forms: 1NF, 2NF and 3NF. There are now others that are generally accepted, but 3NF is widely considered to be sufficient for most applications. Most tables when reaching 3NF are also in BCNF (Boyce-Codd Normal Form).

Third normal form is sufficient for most typical database applications.

Normalization

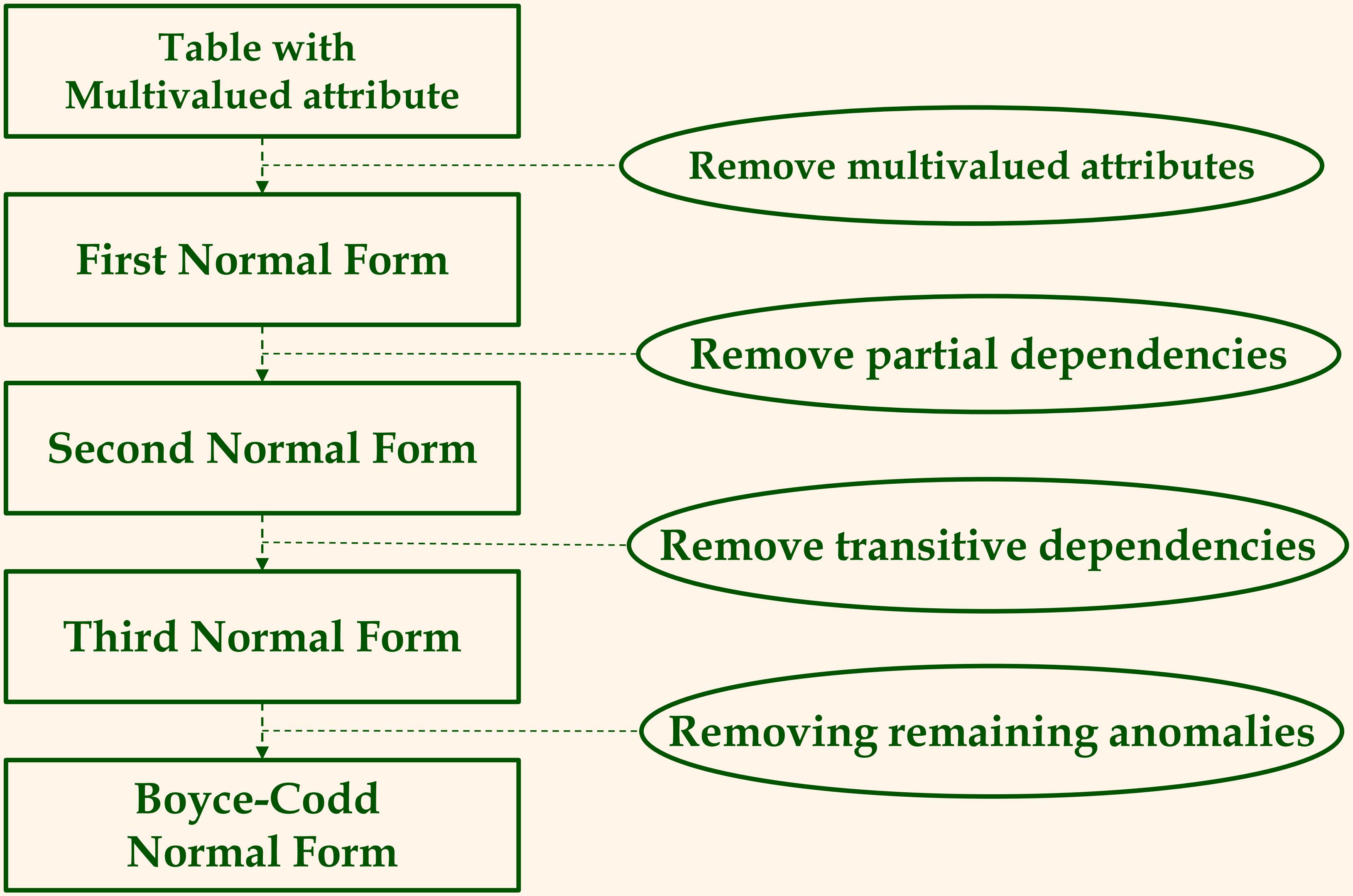
Table with Multivalued attribute
Remove multivalued attributes
First Normal Form
Remove partial dependencies
Second Normal Form
Remove transitive dependencies
Third Normal Form
Removing remaining anomalies
Boyce-Codd Normal Form

First Normal Form (1NF)

- A relation is said to be in 1NF, if it contains *no repeating group*.
- The value at the intersection of a row and column *must be atomic*(*having one value*)
- If you developed a logical design by transforming ER diagram into relations, there should not be any multivalued attributes remaining
- Consider the following relation:

Student (RegNo,Name,Program, C-Code, C-Title, C-Grade)

- This relation has a repeating group and therefore it has the insert, delete and update anomalies.

Second Normal Form (2NF)

- A relation is in 2NF if:
  - It is in 1NF
  - Every nonkey attribute is fully functionally dependent on the primary key
- A situation of Partial Functional Dependency arises when PK of a relation is composite and a non key attribute is functionally dependent on part (but not all) of the PK.
- Referring to the Course relation:

  Course (RegNo, C-Code,C-Title, C_Grade)
- The functional dependencies are:

  C-Code -> C_Title (Partial FD)

  RegNo,C_Code -> C_Grade (Full FD)

Second Normal Form (2NF)

- Since all the non key attributes are not fully functionally dependent on the PK or there is partial functional dependency in the relation, therefore it is not in 2NF.
- The Anomalies associated with the course relation are:

a. Insert Anomaly: A course instance cant be inserted without a student (RegNo)

b. Delete Anomaly:Deleting a student will unnecessarily delete course data.

c. Update Anomaly:A course cant be updated independently.

Second Normal Form (2NF)

- The relation Course can be converted into 2NF by decomposing it into the following relations:

  Course (C-Code,C-Title)

  Result (RegNo, C-Code, C_Grade)

- A relation in 1NF will be in 2NF if:
  - The PK consists of only one attribute **OR**
  - No nonkey attributes exist in the relation **OR**
  - Every nonkey attribute is functionally dependent on the full set of primary key attributes

Third Normal Form (3NF)

- A relation is said to be in 3NF, if it is in 2NF and there is no Transitive Dependency.
- A Transitive Dependency is a functional dependency between two or more non key attributes of a relation.
- Consider the following relation:

  Emp (EmpNo, EName, Job, Sal, Proj-No,Proj-Details)
- In the above relation, there is a following transitive dependency:

  Proj-No -> Proj-Details
- Due to this, project information cant be maintained independent of a employee record and hence there are anomalies in the relation.

Third Normal Form (3NF)

- You can remove transitive dependency from a relation in the following way:
- Create a new relation against transitively dependent attributes and leave the PK of new relation in the old relation to serve as a FK.

  Emp (EmpNo, EName, Job, Sal, Proj-No)

  Project (Proj-No, Proj-Details)

Complete Example:

PNo, PName, PBudget, EmpNo, EName, Job, ChgHour, Hours

Normalization - Exercise

- Order Relation:

  OrderNo, OrderDetails, OrderDate, CustNo, CustName, ProductNo, ProdName, Price, QtyOrdered

- Student Relation:

  RegNo, Name, Address, Program, C-Code, C-Title, C-Grade, T-Code, T-Name

- Patient Relation:

  VisitNo, VisitDate, PatNo, PatName, PatAge, DNo, DName, DSpeciality, Diagnosis

Boyce-Codd Normal Form (BCNF)

- A relation is said to be in BCNF, if it is in 3NF and every determinant is PK or there is no overlapping of candidate keys.
- If a table contains atomic candidate keys, the 3NF and BCNF are equivalent.
- Consider a relation R(A, B, C, D) such that

  A,B -> C, D and C->B.
- The relation R has no partial dependency nor it contains transitive dependency. Thus the relation R is in 3NF.

Boyce-Codd Normal Form (BCNF)

- Consider the following relation:
  PROJECT (RegNo, PTool, Supervisor)

Constraints:

1. For each Project Tool (PTool), a student has only one supervisor.
2. A project may be in more than one tools.
3. Each supervisor can supervise only one too.

- In the above relation, no single attribute is a PK.
- Possible candidate keys are RegNo, PTool and RegNo, Supervisor.
- The candidate keys overlap as they share RegNO.
- The Anomalies associated with the course relation are:

Boyce-Codd Normal Form (BCNF)

a) Insert Anomaly:

❖ Supervisor and PTool cant be defined unless a student takes a project.

b) Delete Anomaly.

❖ Deleting a student will unnecessarily delete project data.

c) Update Anomaly.

❖ Updating a PTool may result in unwanted changes.

❖ The relation Project can be converted into BCNF by decomposing it into the following relations:

PROJECT1 (<u>RegNo, PTool</u>)

PROJECT1 (Supervisor, PTool)

- **Normalization:**
  - The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations

Thank You for your Attention!