Functional Dependencies and Normalization

COMP2411

Functional Dependencies

- Some attributes in a relation will naturally imply other attributes in the relations
- An example for a student: Their Degree will naturally imply their Department as well as the Credits Needed to Graduate
- These are Functional Dependencies (FDs) of a Relation in the DB

Functional Dependencies

- Formally, A Functional Dependency for a relation R exists if, $X \rightarrow Y$ holds for every (possible) legal instance of R.
  - For all tuples $t1, t2 \in R$, if $t1[X] = t2[X]$ then $t1[Y] = t2[Y]$

- Essentially, if two tuples in a relation share the same X value, they must share the same Y value

- Our previous example: Degree → Department, CreditsNeeded

Functional Dependencies

A	B	C	D
1	2	3	4
2	3	4	6
6	7	8	9
1	3	4	5

Which of the following could be FDs in this Relation?

1. $B \rightarrow C$
2. $B \rightarrow D$
3. $D \rightarrow B$

Functional Dependencies

- Trivial FDs are when an attribute infers itself:
  - $X \rightarrow X$
  - $X \rightarrow XY$
- We only really care about recording the non-trivial FDs:
  - $X \rightarrow Y$

Functional Dependencies

Why do we care about Functional Dependencies?

- They can be used to definitively find Superkeys, Candidate Keys and Primary Keys for a Relation
- They can be used to Normalize tables in order to avoid possible Anomalies as well as Reduce Memory requirements

Functional Dependencies

Why do we care about Functional Dependencies?

- They can be used to definitively find Superkeys, Candidate Keys and Primary Keys for a Relation
- They can be used to Normalize tables in order to avoid possible Anomalies as well as Reduce Memory requirements

Keys Refresher

- Keys are distinct for each Relation
  - i.e. no two entities can share the same key
- Superkey: A set of attributes which, taken together, uniquely identifies the Relation
- Candidate Key: A minimal set of attributes which, taken together, uniquely identifies the Relation
- Primary Key: A selected Candidate Key

Using FDs to Find Keys

- Suppose we have the Relation R(A, B, C, D) and the FDs
  - B → C
  - C → B
  - D → ABC
- Are any of the following Superkeys?

1. B
2. C
3. BD
4. ABCD

Using FDs to Find Keys

- Suppose we have the Relation R(A, B, C, D) and the FDs
  - B → C
  - C → B
  - D → ABC
- Are any of the following Candidate Keys?

1. B
2. C
3. BD
4. ABCD

Explicit and Implicit FDs

- Explicit FDs are FDs which are direct and given to you:
  - staffID → level
  - level → salary
- They are immediate connections between attributes
- Implicit FDs are when other FDs are used to infer FDs which hold true for the Relation
  - staffID → salary
- These require following some rules
  - Or Axioms

Armstrong's Axioms

Armstrong's Axioms for inferring Implicit FDs:

- Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$
- Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$
- Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

These rules are considered Sound and Complete:

- Sound: Given a set of FDs F holding on a relation schema R, any FD that we can infer from F by using these three rules holds in every legal relation instance
- Complete: Repeatedly applying these three rules generates all possible FDs that can be inferred

Secondary Inference Rules

These are additional Inference Rules which are useful that can be derived from Armstrong's Axioms:

- Decomposition: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- Union: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- Pseudo-Transitivity: If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$

These rules are useful and proving these rules via only Armstrong's Axioms is a neat exercise

Closure

- For a set of FDs F, the Closure of F (denoted by F^+) is the set of all FDs which can be implied by F
  - This includes both trivial and non-trivial (implicit) FDs
- For a set of attributes A, the Closure of A (denoted by A^+) is the set of all attributes which can be inferred by the attributes X and their associated FDs
- We can calculate X^+ given X and the FDs F by repeatedly applying the six Inference Rules

Calculating Closure

R(number, name, location, depNum, depName, manager)

FD1: number → {name, location, depNum}

FD2: depNum → {depName, manager}

Steps for calculating number$^+$:

1. number$^+$:= number, name, location, depNum (via FD1)
2. number$^+$:= number, name, location, depNum, depName, manager (via FD2)

Let's Try

- R(ABCDE)
- F = {A→B, BC → E, ED → A}

Determine $(ACD)^+$

Finding Keys using FDs

- If the closure of a set of attributes A is the entire relation, then A must be a Superkey
  - If no attribute in A can be removed and still have full closure, it is a Candidate Key

SupplierParts(sName, city, status, pNum, pName, qty)

FD1: sName → city

FD2: city → status

FD3: pNum → pName

FD4: {sName, pNum} → qty

Show that (sName, pNum) is a Superkey and Candidate Key

Finding Keys via FDs

Step 1: $(sName, pNum)^+ \rightarrow$ sName, pNum, city (via FD1)

Step 2: $(sName, pNum)^+ \rightarrow$ sName, pNum, city, status (via FD2)

Step 3: $(sName, pNum)^+ \rightarrow$ sName, pNum, city, status, pname (via FD3)

Step 3: $(sName, pNum)^+ \rightarrow$ sName, pNum, city, status, pname, qty (via FD4)

If we remove sName, we lose city, status and qty

If we remove pNum, we lose pName and qty

Let's Try

- R(ABCDE)
- F = {D → C, CE → A, D → A, AE → D}

Which are superkeys and which are candidate keys?

1. ABCE
2. BCE
3. CDE

Tips for finding Candidate Keys

- If you find a Candidate Key S, then all supersets of S cannot be a Candidate Key
  - If ABC is a Candidate Key, ABCD cannot be a Candidate Key
- Candidate Keys can be of different lengths
  - If ABC is a Candidate Key, ABDE can still be a Candidate Key
- If an attribute never appears in the Right Hand Side of any FD, then it must be in all Candidate Keys

Functional Dependencies

Why do we care about Functional Dependencies?

- They can be used to definitively find Superkeys, Candidate Keys and Primary Keys for a Relation
- They can be used to Normalize tables in order to avoid possible Anomalies as well as Reduce Memory requirements

Anomalies

- Anomalies are inconsistencies or errors which may arise in a Database when data is changed
- Basically, Anomalies are what happens in Integrity Constraints and Functional Dependencies are allowed to be broken
- Types of Anomalies:
  - Insertion Anomalies
  - Deletion Anomalies
  - Update Anomalies

Insertion Anomaly

- An Insertion Anomaly occurs when a new entry is either the wrong domain or is null when it can't be null

Employee ID	Name	Office	Office Address
1	Alice	Main Office	123 Apple Boulevard
2	Bob	Main Office	123 Apple Boulevard
3	Charlie	London Office	44 Elizabeth Street
4	NULL	NULL	NULL

We want to add employee 4 but can't since their name is null

Deletion Anomaly

- A Deletion Anomaly occurs when the removal of a tuple would cause unintended loss of information from the system

Employee ID	Name	Office	Office Address
1	Alice	Main Office	123 Apple Boulevard
2	Bob	Main Office	123 Apple Boulevard
3	Charlie	London Office	44 Elizabeth Street

If we delete Charlie, we have unintentionally deleted the address of the London Office from our Database

Update Anomaly

- An Update Anomaly causes inconsistencies to arise when updating a tuple

Employee ID	Name	Office	Office Address
1	Alice	Main Office	123 Apple Boulevard
2	Bob	Main Office	123 Apple Boulevard
3	Charlie	London Office	44 Elizabeth Street

We want to update the address of Alice's office

Update Anomaly

- An Update Anomaly causes inconsistencies to arise when updating a tuple

Employee ID	Name	Office	Office Address
1	Alice	Main Office	25 Banana Street
2	Bob	Main Office	123 Apple Boulevard
3	Charlie	London Office	44 Elizabeth Street

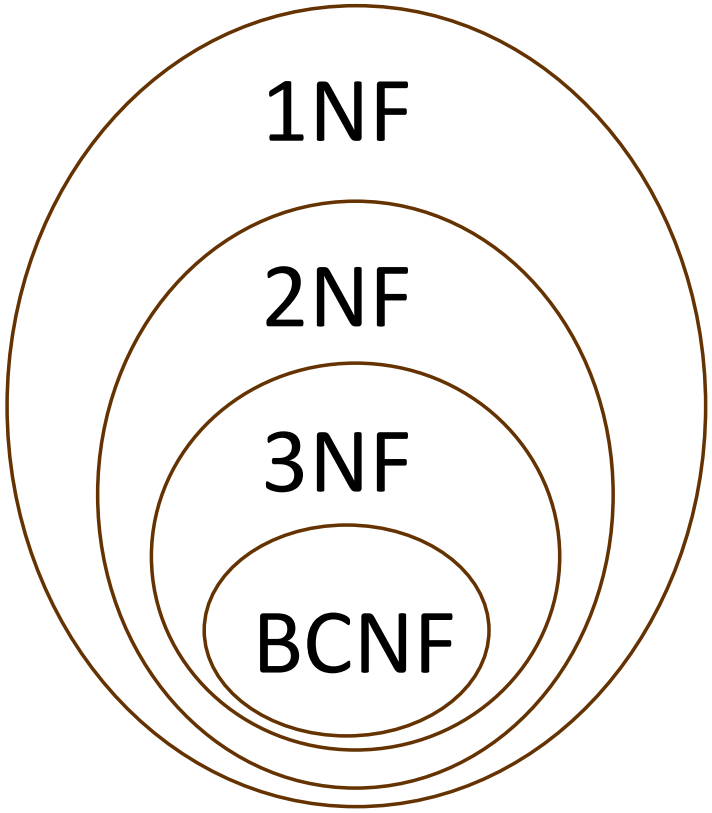
By updating only Alice's office, we have caused the Main Office to be in two places at the same time causing an inconsistency

Normalization

- Normalization is the process of improving our Database via its Functional Dependencies and Primary Keys
- The purpose of Normalization is:
  - To remove the possibility of anomalies
  - To make the DB more memory-efficient by reducing redundant information
- Your ER Diagram is a first draft which creates your Relational Schema
- Normalization is the proof-reading process

Normal Forms

- Normal Forms are the stages of improving a DB
- The Normal Forms we will cover:
  - First Normal Form (1NF)
  - Second Normal Form (2NF)
  - Third Normal Form (3NF)
  - Boyce-Codd Normal Form (BCNF)
- Normal Forms are nested



First Normal Form (1NF)

- For a Relational Database to be in 1NF, it simply needs to follow the most basic rule: All attributes are atomic and tuples are unique

Director	Movies
Steven Spielberg	Indiana Jones, Jaws, Jurassic Park

Director	Movie
Steven Spielberg	Indiana Jones
Steven Spielberg	Jaws
Steven Spielberg	Jurassic Park

Second Normal Form (2NF)

- A table is in 2NF if it contains no Partial Dependencies and is in 1NF
  - A Partial Dependency is where a non-prime attribute is not functionally dependent on an entire Candidate Key (for all Candidate Keys)

Assignments

Employee ID	Project ID	Office	Role
1	1	Main Office	Project Lead
1	2	London Office	Consultant
2	1	Main Office	Secretary
3	1	Main Office	Worker
4	2	London Office	Project Lead

{Employee ID, Project ID} -> {Role}
{Project ID} -> {Office}

Office is partially dependent on the key

Second Normal Form (2NF)

- To turn a table from 1NF into 2NF, for each Partial Dependency we build a new table and remove the offending attribute(s) from the original table

Assignments

Employee ID	Project ID	Role
1	1	Project Lead
1	2	Consultant
2	1	Secretary
3	1	Worker
4	2	Project Lead

Assignments: {Employee ID, Project ID} -> {Role}

Project_Offices

Project ID	Office
1	Main Office
2	London Office

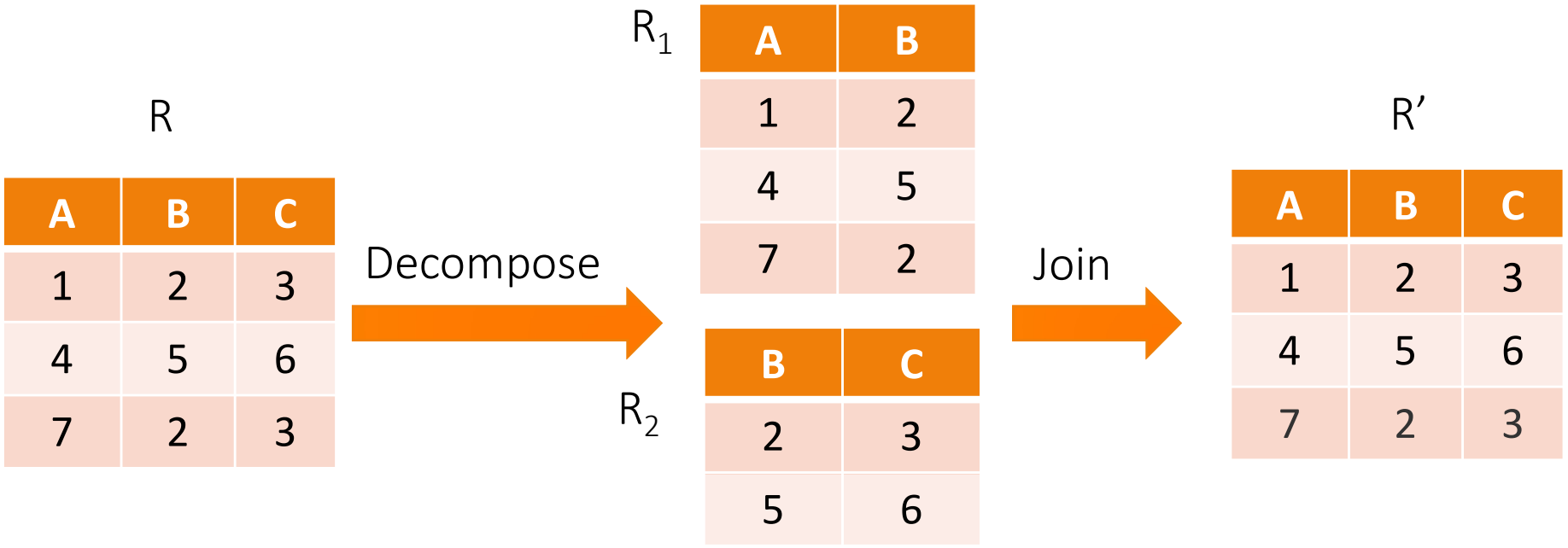
Project_Offices: {Project ID} -> {Office}

Decomposition

- The process of splitting tables up into smaller tables is called Decomposition
- This helps avoid anomalies
  - In the previous example, we now avoid Update and Deletion anomalies
- It also helps reduce redundancy
  - Previously, we had to write the office each time the Project ID appears, now we only do it once
- However, a Decomposition must result in a Lossless Join

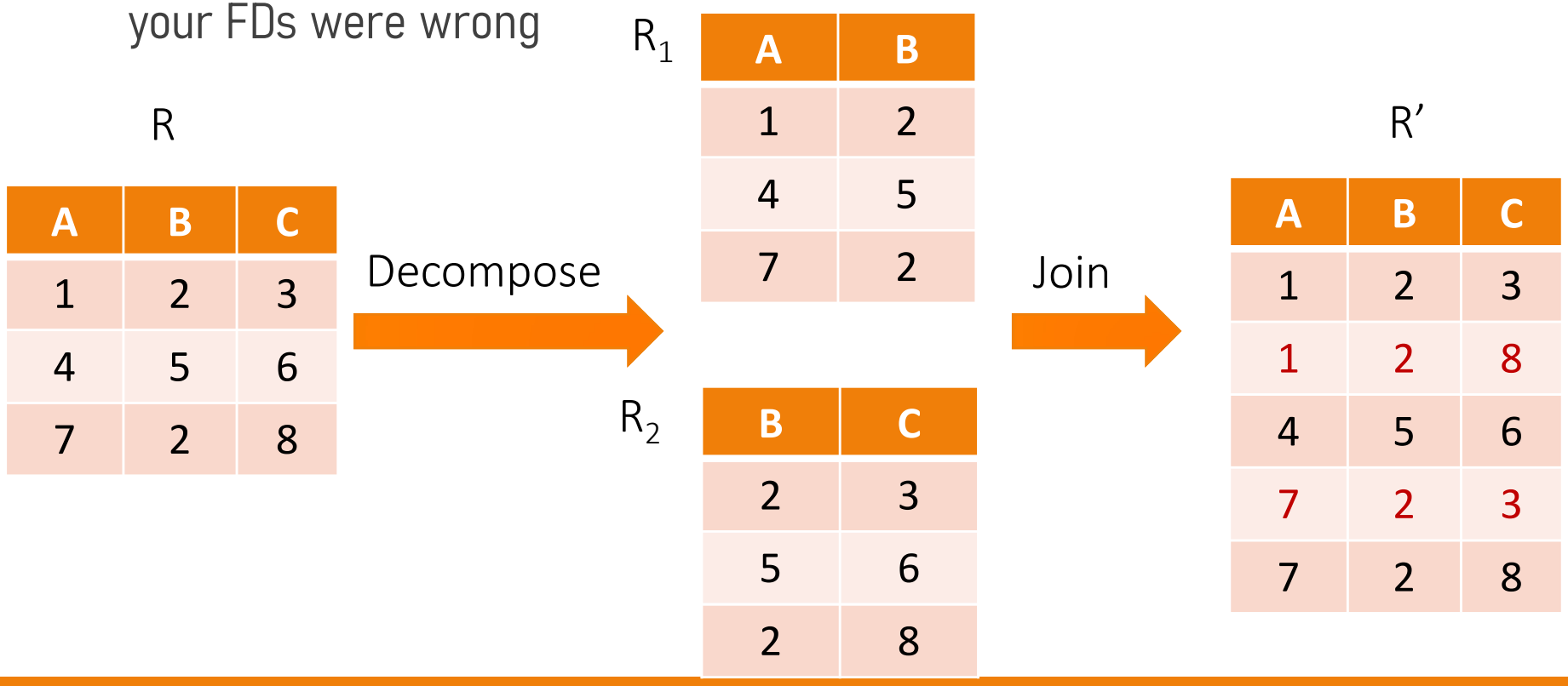
Lossless Join

- If we decompose R into R_1 and R_2 it is a Lossless Join if for every instance of r in R: $R = R_1 \bowtie R_2$
- In English: If we decompose R into R_1 and R_2, if we join R_1 and R_2 together, it should recreate R



Lossy Join

- Lossy Joins are when the recreated table is different from the original table
  - This means you decomposed something that you weren't supposed to or your FDs were wrong



Third Normal Form (3NF)

- A table is in 3NF if it contains no Transitive Dependencies and is in 2NF
  - A Transitive Dependency is where a non-prime attribute is not explicitly functionally dependent on any Candidate Key

Employees

Employee ID	Name	Office	Office Address
1	Alice	Main Office	123 Apple Boulevard
2	Bob	Main Office	123 Apple Boulevard
3	Charlie	London Office	44 Elizabeth Street

Employees: {Employee ID} -> {Name, Office}
{Office} -> {Office Address}

Office Address is transitively dependent on EmployeeID via Office

Third Normal Form (3NF)

- To turn a table from 2NF into 3NF, for each Transitive Dependency we build a new table and remove the offending attribute(s) from the original table

Employees

Employee ID	Name	Office
1	Alice	Main Office
2	Bob	Main Office
3	Charlie	London Office

Offices

Office	Office Address
Main Office	123 Apple Boulevard
London Office	44 Elizabeth Street

Employees: {Employee ID} -> {Name, Office}

Offices: {Office} -> {Office Address}

Boyce-Codd Normal Form (BCNF)

- BCNF (sometimes referred to as 3.5NF) is a slightly stricter version of 3NF aimed solely at reducing redundancy
- A table is in BCNF if for any non-trivial FD X → Y, X is a Superkey of R
  - This rule ensures the table is in 3NF already
- What we are essentially looking for is extra/excessive FDs which can cause redundancy

Boyce-Codd Normal Form (BCNF)

Study(studentID, courseID, lecturer)

- FD1: {studentID, courseID} → lecturer
- FD2: lecturer → courseID

The Candidate Keys: {studentID, courseID} and {studentID, lecturer}

- This table is in 3NF as courseID and lecturer are both prime-attributes

FD1 is fine, but the LHS of FD2 is not a Superkey which means Study is not in BCNF

Boyce-Codd Normal Form (BCNF)

- The best way to decompose and ensure a lossless join is to create a new table based on the offending FD and check how the remaining FDs are affected

StudentLecturer(studentID, lecturer)

No non-trivial FDs

Candidate Key: {studentID, lecturer}

LecturerCourse(lecturer, courseID)

FD: lecturer → courseID

Candidate Key: lecturer

Let's Try

R(A, B, C, D)

FD1: AD → BC

FD2: B → A

What are the Candidate Key(s) of R?

If R is not in BCNF, normalize it into BCNF

From Start to Finish

R(A, B, C, D, E, F, G, H, I, J)

FD1: {A, B} → C

FD2: A → {D, E}

FD3: B → F

FD4: F → {G, H}

FD5: D → {I, J}

Find the Candidate Key(s) of R

Determine the highest Normal Form of R

Decompose R into 2NF, 3NF and then BCNF

From Start to Finish

R has the Candidate Key {A, B} and is currently in 1NF

2NF:

R1 (A, B, C)	F = {{A, B} → C}	CK: {A, B}
R2 (A, D, E, I, J)	F = {A → {D, E}, D → {I, J}}	CK: A
R3 (B, F, G, H)	F = {B → F, F → {G, H}}	CK: B

We now must decompose into 3NF

From Start to Finish

3NF:

R1 (A, B, C)	F = {{A, B} → C}	CK: {A, B}
R21 (A, D, E)	F = {A → {D, E}}	CK: A
R22 (D, I, J)	F = {D → {I, J}}	CK: D
R31 (B, F)	F = {B → F}	CK: B
R32 (F, G, H)	F = {F → {G, H}}	CK: F

All relations are also in BCNF, no more decomposition required

Let's Try

R(A, B, C, D, E, F, G, H, I, J)

FD1: {A, B} → C

FD2: {B, D} → {E, F}

FD3: {A, D} → {G, H}

FD4: A → I

FD5: H → J

Find the Candidate Key(s) of R

Determine the highest Normal Form of R

Decompose R into 2NF, 3NF and then BCNF

Let's Try

R(A, B, C, D, E, F)

FD1: A → {B, C, D}

FD2: {B, C} → {A, D}

FD3: D → B

FD4: B → {E, F}

Find the Candidate Key(s) of R

Determine the highest Normal Form of R

Decompose R into 2NF, 3NF and then BCNF

ER Diagrams and Normalization

- Smart ER Diagram design should mean the resulting Relational Schema should already be in 3NF or BCNF (depending on which you choose for the DB)
- Normalization is simply a safeguard against “bad” ER Diagrams

Denormalization

- Normal Forms are basically just a guideline for theoretically better databases. The higher the Normal Form, the more tables are theoretically built.
  - The problem with having many tables is this makes queries potentially more expensive as you need to join many tables together
- Denormalization is the process of intentionally allowing redundancy in the DB to allow for faster query time
  - Trading extra memory for faster query performance
- Denormalization must be controlled as you are allowing anomalies into your DB

Objectives

- Use Functional Dependencies to find Superkeys and Candidate Keys of a Relation
- Understand the various types of anomalies which may occur given poor DB design
- Understand Normal Forms and their properties
- Be able to normalize any database into BCNF given its Functional Dependencies