Lecture 8

Database Normalization

Dr. Vandana Kushwaha

Department of Computer Science
Institute of Science, BHU, Varanasi

Database Anomalies

- In a **Database management system** (DBMS), **insertion, deletion,** and **update anomalies** refer to **issues** that can arise when modifying the data in a **relational database.**
- These **anomalies** can **lead to inconsistencies** and **errors** in the **database.**
- **Insertion Anomaly**
- An **insertion anomaly** occurs when we try to **insert a new record** into a database, but due to the way the tables are structured, we are required to provide values for attributes that may not be applicable or available at the time of insertion.
- As a result, we may be **forced to insert incomplete or incorrect data** into the **database,** which can lead to **inconsistencies** and **inaccuracies.**
- **Insertion anomalies** make it **difficult to add new data** without violating the database's **integrity constraints.**

Database Anomalies

- **Example Insertion Anomaly**
- Consider a **Database** that stores information about **students** and their **courses:**
  - ***Students_Courses(Student_ID, Name,Course_ID, Course_Name)***
- Suppose a student named Alice has not yet enrolled in any course.
- When we try to insert Alice's record into the "Students" table, we are required to provide a Course_ID since it is a non-null attribute.
- However, since Alice hasn't enrolled in any course yet, we are forced to enter a placeholder or default value for Course_ID.
- This results in an **insertion anomaly** because we are adding incomplete or irrelevant data to the database.

Database Anomalies

- **Deletion Anomaly**
- A **deletion anomaly** occurs when we **delete** a record from a database, but unintentionally lose other relevant information that is associated with that record.
- It happens when deleting a record also removes data that is needed by other records or queries in the database.
- **Deletion anomalies** can lead to **data loss** and affect the **overall integrity** of the database.
- **Example** ***Students_Courses(Student_ID, Name,Course_ID, Course_Name)***
- Let us assume that **Alice** is the only student enrolled in a particular course.
- If we delete Alice's record from the "Students" table, we will unintentionally lose information about the course itself.
- Even though the course may still be relevant and other students may want to enroll in it, deleting the single record causes the loss of course information.

Database Anomalies

- **Update Anomaly**
- An **update anomaly** arises when updating data in a database results in **inconsistencies** or **data duplication.**
- It occurs when modifying an attribute value in one place of the database and failing to update the corresponding values in other places where the same data is stored.
- **Update anomalies** can lead to data inconsistencies and make it challenging to maintain the accuracy and coherence of the database.

Database Anomalies

- **Example Update Anomaly**
- Imagine a database that tracks inventory for a retail store with table:
- ***Products(Product_ID, Product_Name, Quantity, Price)***
- Suppose a product's price changes, and we updated the Price attribute for that product in the table.
- However, if the same product appears in multiple records/tables, we must update the price value in each occurrence.
- If we miss updating any record, it leads to inconsistencies, and the database would contain different prices for the same product, resulting in an **update anomaly.**

Database Anomalies

- To **mitigate** these **anomalies,** proper **Database Normalization** can be applied.
- By applying **normalization techniques** and structuring the database appropriately, these anomalies can be minimized, ensuring **data integrity** and **consistency** within the DBMS.

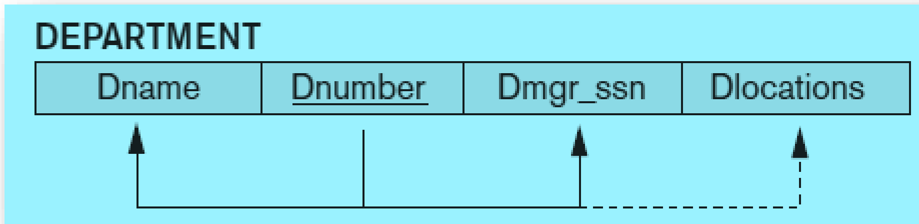
Normalization

- The **Normalization process**, as first proposed by **Codd (1972**), takes a **relation schema** through a **series of tests** to certify whether it satisfies a certain **normal form.**
- **Normalization** of data can be considered a process of **analyzing** the given **relation schemas** based on their **FDs** and **primary keys** to achieve the **desirable properties** of:
  - **Minimizing redundancy** and
  - **Minimizing the insertion, deletion**, and **update anomalies**.
- **Unsatisfactory relation schemas** that do not meet certain constraints are **decomposed** into smaller relation schemas that meet the tests and hence possess the desirable properties.
- Initially, **Codd proposed three normal forms**, which he called **first, second**, and **third normal form.**
- A **stronger definition of 3NF**—called **Boyce-Codd normal form (BCNF)**—was proposed later by **Boyce and Codd.**

First Normal Form

- **First normal form (1NF)** was defined to disallow **multivalued attributes** and **composite attributes**.
- It states that the **domain** of an **attribute** must include **only** ***atomic*** **(simple, indivisible)** ***values.***
- **Example:**
- Consider the **DEPARTMENT** relation schema.
- We assume that each **department** can have *a* ***number of*** **locations.**
- The **DEPARTMENT relation** state is not in **1NF** because **Dlocations** is not an **atomic attribute.**

First Normal Form



DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

First Normal Form

- There are **three** main techniques to achieve **first normal** form for such a relation:
- **Method 1:**
- **Remove** the attribute **Dlocations** that violates **1NF** and place it in a separate relation **DEPT_LOCATIONS** along with the **primary key** Dnumber of DEPARTMENT.
- The **primary key** of this relation is the combination **{Dnumber, Dlocation}.**
- A distinct tuple in DEPT_LOCATIONS exists for *each location* of a department.
- This **decomposes** the non-1NF relation into **two 1NF relations**.

DEPARTMENT

Dname	Dnumber	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

First Normal Form

- **Method 2:**
- **Expand the key** so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT.
- In this case, the **primary key** becomes the combination **{Dnumber, Dlocation}.**
- This solution has the **disadvantage** of introducing ***redundancy*** in the relation.

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

First Normal Form

- **Method 3:**
- If a ***maximum number of values*** is known for the **attribute**—for example, if it is known that ***at most three locations*** can exist for a department—replace the **Dlocations** attribute by **three atomic attributes**: **Dlocation1, Dlocation2**, and **Dlocation3.**
- This solution has the **disadvantage** of introducing ***NULL values*** if most departments have fewer than three locations.
- It further introduces spurious semantics about the ordering among the location values that is not originally intended.
- Querying on this attribute becomes more difficult; for example, consider how you would write the **query:** *List the departments that have 'Bellaire' as one of their locations* in this design.

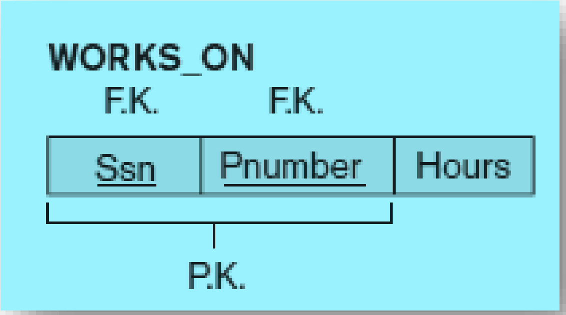
Second Normal Form

- **Second normal form (2NF)** is based on the concept of ***full functional dependency***.
- A **functional dependency** $X \rightarrow Y$ is a **full functional dependency** if **removal of any attribute** A from X means that the **dependency does not hold** any more;
- That is, for any attribute $A \in X$, $(X - \{A\})$ does *not* **functionally determine** Y.
- A **functional dependency** $X \rightarrow Y$ is a **partial dependency** if some attribute $A \in X$ can be removed from X and the **dependency still holds**; that is, for some :

  $A \in X, (X - \{A\}) \rightarrow Y$.
- **Example:**
- **{Ssn, Pnumber} → Hours** is a **full dependency** (neither Ssn → Hours nor Pnumber→Hours holds).
- However, the dependency **{Ssn, Pnumber}→Ename** is **partial** because **Ssn→Ename** holds.

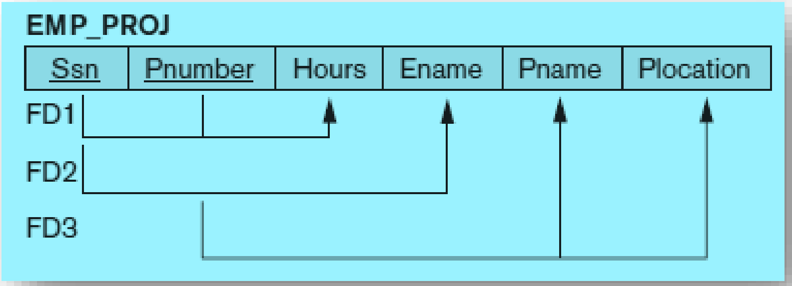
Second Normal Form

- **Prime attribute**
- An **attribute** of **relation schema *R*** is called a **prime attribute** of ***R*** if it is a **member** of *some* ***candidate key*** of *R*.
- An **attribute** is called **nonprime** if it is not a **prime attribute**—that is, if it is not a member of any **candidate key.**
- **Example:** Both **Ssn** and **Pnumber** are **prime attributes** of **WORKS_ON**, whereas other attribute of **WORKS_ON** are **nonprime.**



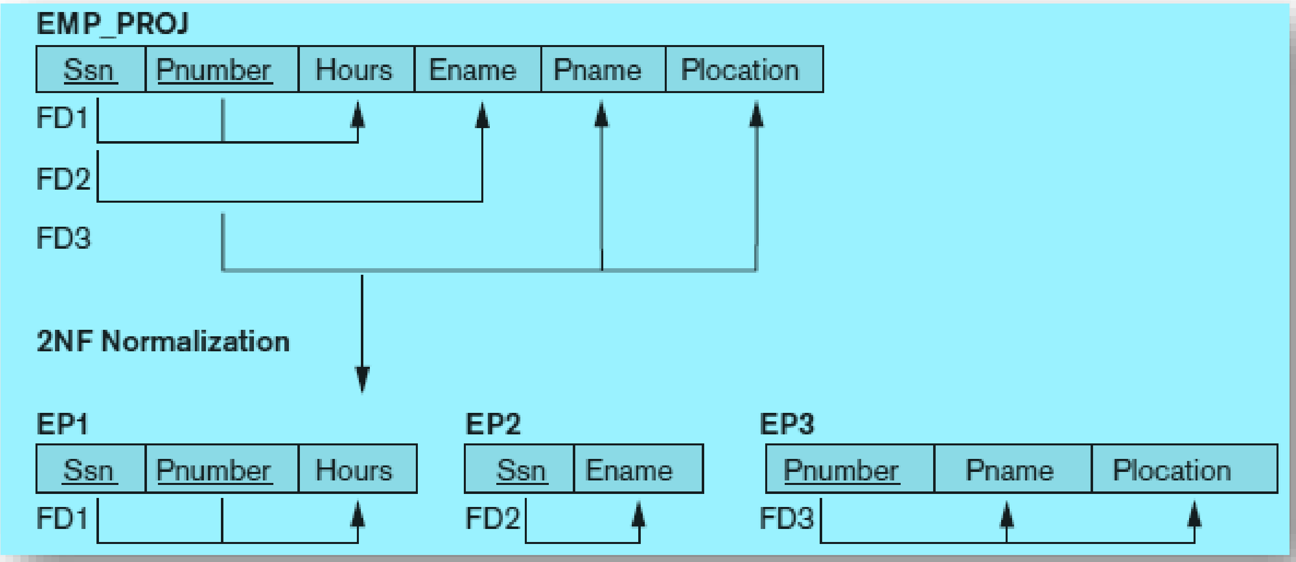
Second Normal Form

- **2NF Definition**
- A relation schema ***R*** is in **2NF** if every **nonprime attribute *A*** in ***R*** is ***fully functionally dependent*** on the **primary key of *R***.
- **Example:**
- The **EMP_PROJ** relation in is in **1NF** but is not in **2NF.**
- The **nonprime attribute** Ename **violates 2NF** because of **FD2**, as do the **nonprime attributes Pname and Plocation** because of **FD3.**
- The **functional dependencies FD2** and **FD3** make Ename, Pname, and Plocation **partially dependent** on the **primary key** {Ssn, Pnumber} of **EMP_PROJ**, thus violating the **2NF test.**



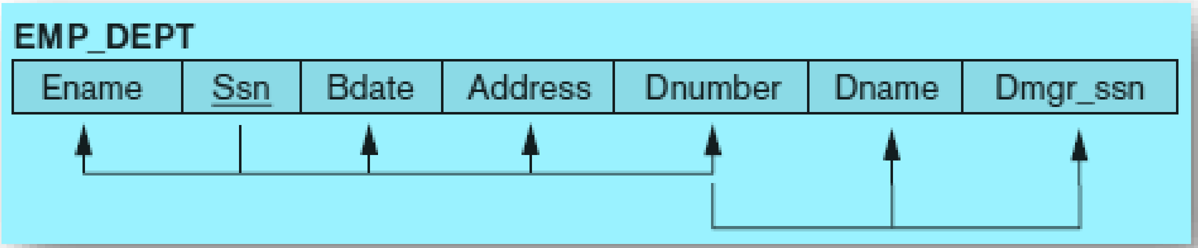
Second Normal Form

- If a **relation schema** is not in **2NF,** it can be ***second normalized*** or ***2NF normalized*** into a number of **2NF relations** in which **nonprime attributes** are associated only with the part of the **primary key** on which they are **fully functionally dependent**.
- Therefore, the **functional dependencies FD1, FD2**, and **FD3** lead to the **decomposition** of **EMP_PROJ** into the **three relation** schemas **EP1, EP2,** and **EP3** shown below, each of which is in **2NF.**



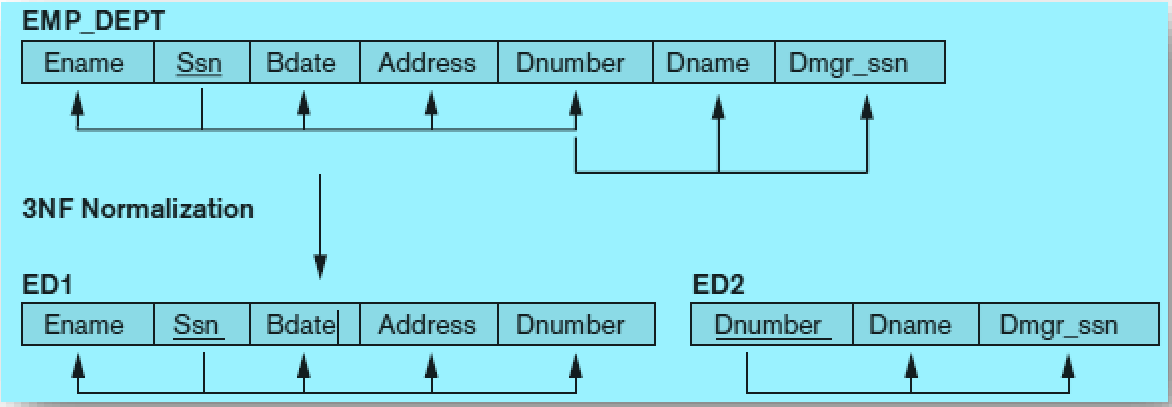
Third Normal Form

- **Third normal form (3NF)** is based on the concept of ***transitive dependency***.
- A **functional dependency** ***X→Y*** in a **relation schema** ***R*** is a **transitive dependency** if there exists a set of attributes ***Z* in *R*** that is **neither a candidate key** nor a **subset of any key of *R***, and both ***X→Z*** and ***Z→Y*** hold.
- The dependency **Ssn→Dmgr_ssn** is **transitive** through **Dnumber** in **EMP_DEPT,** because both the dependencies **Ssn → Dnumber** and **Dnumber → Dmgr_ssn** hold and **Dnumber** is **neither a key** itself **nor a subset of the key** of **EMP_DEPT.**



Third Normal Form

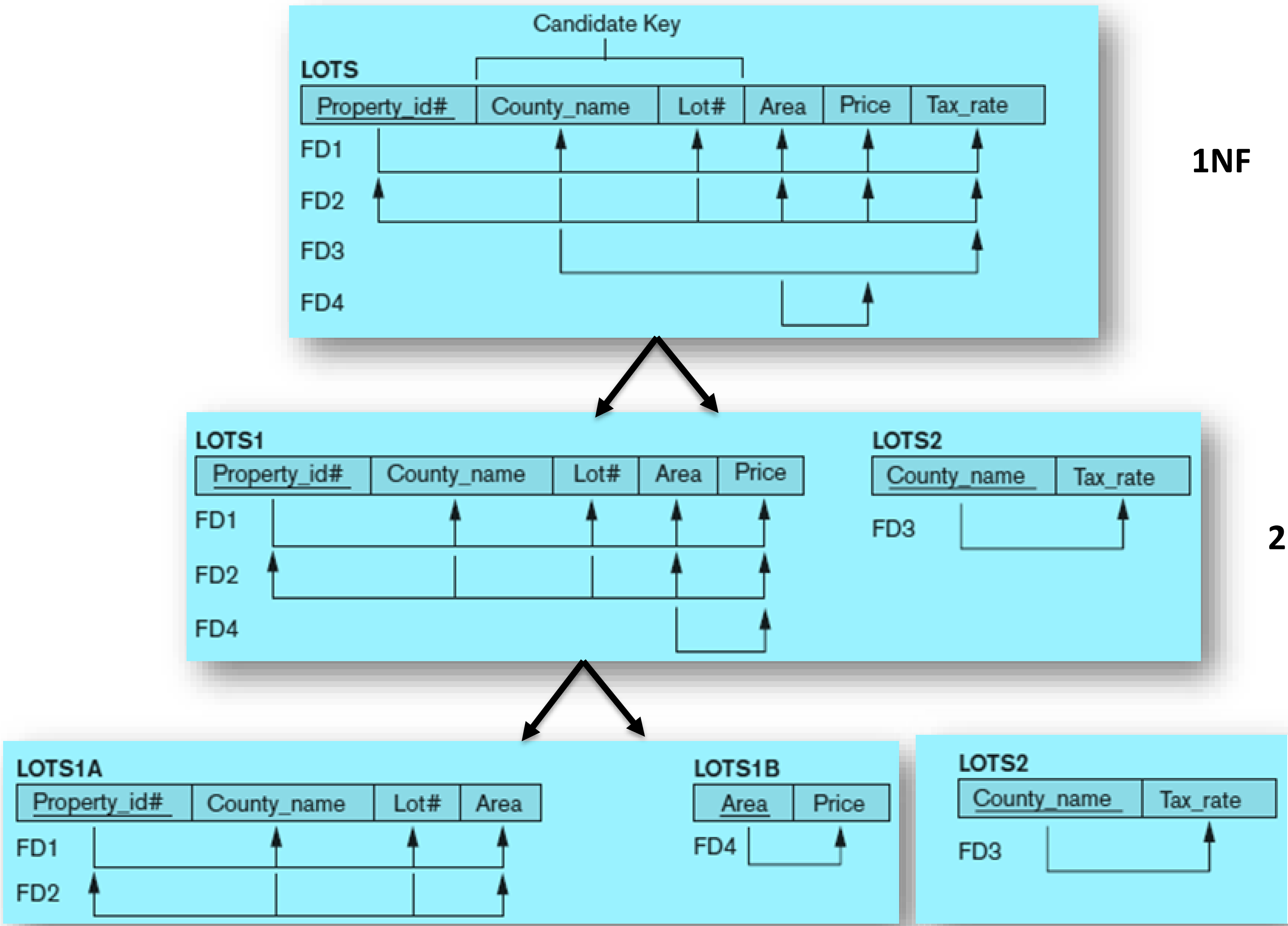
- **Definition 3NF:** **A** relation schema ***R*** is in **3NF** if it **satisfies 2NF** *and* **no nonprime attribute** of ***R*** is **transitively dependent** on the **primary key.**
- The relation schema **EMP_DEPT** is in **2NF**, since **no partial dependencies** on a **key** exist.
- However, **EMP_DEPT** is **not in 3NF** because of the **transitive dependency** of **Dmgr_ssn** (and also **Dname**) on **Ssn** via **Dnumber.**
- We can **normalize EMP_DEPT** by **decomposing** it into the **two 3NF relation** schemas **ED1** and **ED2.**



General Definition of Third Normal Form

- **Definition.** A relation schema *R* is in **third normal form (3NF)** if, whenever a ***nontrivial*** **functional dependency** $\boldsymbol{X \rightarrow A}$ holds in ***R***, either
  - **(a)** ***X*** **is a superkey of** ***R*****, or**
  - **(b)** ***A*** **is a prime attribute of** ***R*****.**

Example



1NF

2NF

3NF

Example of 3NF

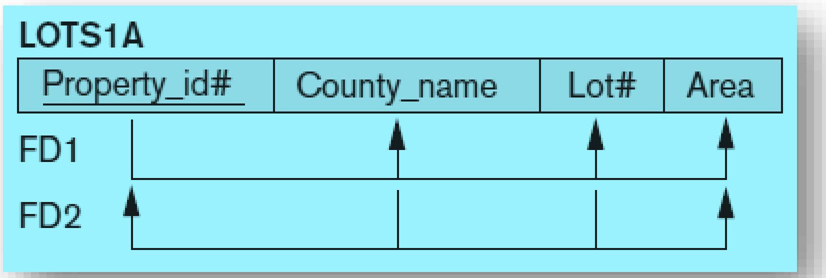
- According to general definition**, LOTS2** is in **3NF.**
- However, **FD4** in **LOTS1 violates 3NF** because **Area** is not a **superkey** and **Price** is **not a prime attribute** in **LOTS1.**
- To normalize **LOTS1** into **3NF**, we decompose it into the relation schemas **LOTS1A** and **LOTS1B.**
- We construct LOTS1A by removing the attribute Price that violates 3NF from LOTS1 and placing it with Area (the lefthand side of FD4 that causes the transitive dependency) into another relation LOTS1B.
- Both **LOTS1A** and **LOTS1B** are in **3NF.**

General Definition of 3NF

- This **general definition** can be **applied *directly*** to test whether a **relation schema** is in **3NF.**
- It **does *not*** have to go through **2NF first.**
- If we apply the above **3NF definition** to **LOTS** with the dependencies **FD1** through **FD4,** we find that ***both*** **FD3** and **FD4 violate 3NF.**
- Therefore, we could decompose **LOTS** into **LOTS1A, LOTS1B**, and **LOTS2** directly.

Boyce-Codd Normal Form

- **Boyce-Codd normal form (BCNF)** was proposed as a simpler form of **3NF**, but it was found to be **stricter than 3NF.**
- That is, **every relation in BCNF is also in 3NF**; however, a **relation in 3NF is *not necessarily* in BCNF.**
- **Definition BCNF:** A relation schema ***R*** is in **BCNF** if whenever **a *nontrivial* functional dependency *X*→*A* holds in *R*, then *X* is a super key of *R*.**
- Consider the **LOTS1A relation** schema with its two functional dependencies **FD1** and **FD2.**



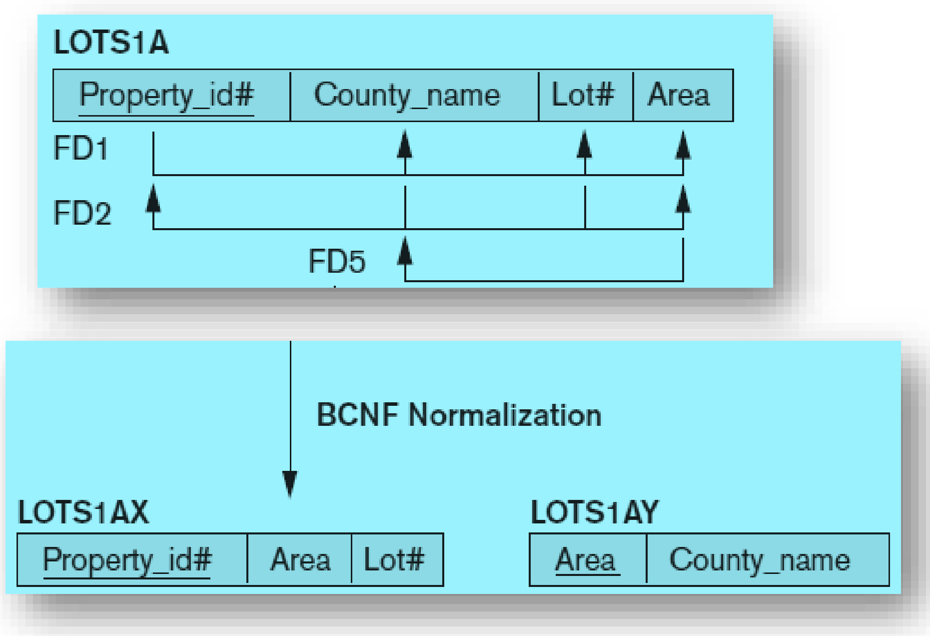
- Suppose that we have **thousands of lots** in the relation but the lots are from **only two counties**: **ABC** and **XYZ.**

Boyce-Codd Normal Form

- Suppose also that **lot sizes** in **ABC County** are only **0.5, 0.6, 0.7, 0.8, 0.9**, and **1.0 acres**, whereas **lot sizes** in **XYZ County** are restricted to **1.1, 1.2, ..., 1.9, and 2.0 acres.**
- In such a situation we would have the **additional functional dependency**

  FD5: *Area→County_name*

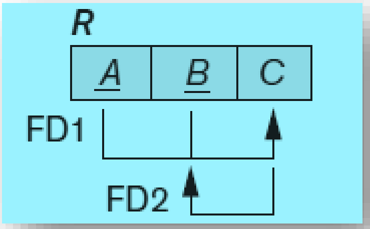
- If we add this to the other dependencies, the relation schema **LOTS1A** still is in **3NF** because **County_name** is a **prime attribute** but it is **not in BCNF** as **area is not a superkey.**



Boyce-Codd Normal Form

- We can decompose **LOTS1A** into **two BCNF relations LOTS1AX** and **LOTS1AY.**
- But **this decomposition loses** the **functional dependency FD2** because its attributes no longer coexist in the same relation after **decomposition.**

- **Example:** Consider a relation schema **R**



- The relation schema **R** is in **3NF, but not in BCNF.**

Desirable properties of Decomposition

- There are **two desirable properties** of a **decomposition** of a **relation schema:**

1. **Loss-less Join(Non-additive join)**

2. **Dependency preservation**

Lossy Decomposition

- Suppose we have a relation **R(ABC)** with **F=(A->B, C->B)**
- **R** is decomposed into **two relations R1(A,B)** and **R2(B,C)**

R(ABC)

A	B	C
A1	B1	C1
A3	B1	C2
A2	B2	C3
A4	B2	C4

R1

A	B
A1	B1
A3	B1
A2	B2
A4	B2

R2

B	C
B1	C1
B1	C2
B2	C3
B2	C4

R1 ⋈ R2

A	B	C
A1	B1	C1
A1	B1	C2
A3	B1	C1
A3	B1	C2
A2	B2	C3
A2	B2	C4
A4	B2	C3
A4	B2	C4

- There are **spurious tuples** in **joined relation.**
- **Spurious tuples** results in **loss of information (not loss of tuples).**
- Thus **R -> R1 R2** is a **lossy decomposition.**

Lossless Decomposition

- Suppose we have a relation **R(XYZ)** with **F=(XY->X, X->Y, X->Z)**
- **R** is **decomposed** into **two relations R1(X,Y)** and **R2(X,Z)**

R(ABC)		
X	**Y**	**Z**
X1	Y1	Z1
X2	Y2	Z2
X3	Y2	Z1
X4	Y1	Z2

R1	
X	**Y**
X1	Y1
X2	Y2
X3	Y2
X4	Y1

R2	
X	**Z**
X1	Z1
X2	Z2
X3	Z1
X4	Z2

R1 ⋈ R2		
X	**Y**	**Z**
X1	Y1	Z1
X2	Y2	Z2
X3	Y2	Z1
X4	Y1	Z2

- There is **no spurious tuples** in **joined relation.**
- Thus **R -> R1 R2** is a **loss-less decomposition.**

Lossless Decomposition

Definition: A decomposition **D={R1,R2……Rm}** of **R** has the **lossless join property** w.r.t. the set of dependencies **F** on **R** if, for every **relation instance r of R** that satisfies **F**, the following holds: $\bowtie (\pi_{R1}(r) \ldots\ldots\ldots\ldots\ldots \pi_{Rm}(r))=r$

Check lossless Decomposition:

- Let **R** be a relation schema, and let **F** be a set of functional dependencies on **R.**
- Let **R1** and **R2** form a **decomposition** of **R.**
- This **decomposition** is a **lossless-join decomposition** of **R** if **at least one** of the following functional **dependencies** is in F^+ **:**
  - $R1 \cap R2 \rightarrow R1$
  - $R1 \cap R2 \rightarrow R2$
- In other words, **if R1 ∩ R2 forms a super key of either R1 or R2**, the **decomposition** of **R** is a **lossless-join decomposition.**

Dependency Preservation

- Let **F** be a set of **functional dependencies** on a schema **R**, and let **R1, R2,...,Rn** be a **decomposition** of **R.**
- The **restriction** of **F** to **Ri** is the set **Fi** of all functional dependencies in $\mathbf{F^+}$ that include only attributes of **Ri.**
- Note that the definition of **restriction** uses **all dependencies** in $\mathbf{F^+}$, not just those in **F.**
- For instance, suppose a relation schema **R(ABC)**, **F = {A → B, B → C},** and we have a **decomposition** into **AC** and **AB.**
- The **restriction of F** to **AC** is then **F1=A → C,** since **A → C** is in $\mathbf{F^+}$, even though it is **not in F**, **restriction of F** to **AB** is **F2=A → B.**
- **Let F' = F1 ∪ F2 ∪ ··· ∪ Fn**
- Then this **decomposition is dependency preserving if** $\mathbf{F'^+ = F^+}$

Dependency Preservation

- **Example 1:**
- **Relation schema R(ABC)**, **F = {A → B, B → C}**
- The two **decompositions** are **R1(AC)**, **F1**={A-> C} and **R2(AB)**, **F2**={A → B}
- Is **not a dependency preserving decomposition** as **F'$^+$ is not equal to F$^+$.**

  F' = F1UF2 = **{A->C, A->B}**

- **Example 2:**
- **Relation schema R(ABCD), F={A->B, A->C, A->D}**
- The two **decompositions** are **R1(ABD)**, **F1**={A->B, A->D} and **R2(BC)**, **F2**={ }
- Thus **F'** **= F1UF2**= **{A->B, A->D}**
- It is **not a dependency preserving decomposition** as **F'$^+$ will not be equal to F$^+$.**

Summary: BCNF

- **Lossless-join** and **Dependency preserving** decomposition into **BCNF impossible**?
- It is **not possible** to have **all three** of the following:
  - (1**) Guaranteed lossless design**
  - (2**) Guaranteed dependency preservation**
  - (3) **All relations in BCNF.**
- The **first condition** is **a must** and **cannot be compromised.**
- The **second condition is desirable**, but not a must, and may have to be relaxed if we insist on achieving **BCNF.**

Exercise 1

- Suppose that we decompose the schema ***R* = (*A, B, C, D, E*)** into **R1**(*A*, *B*, *C*) and **R2**(*A*, *D*, *E*). Show that this decomposition is a **lossless-join decomposition** if the following set ***F*** of functional dependencies holds: **F=**{ *A → BC, CD → E, B → D, E → A}*
- **Solution:**
- As R1∩ R2 = **A**
- Ristriction of R1 over F is **F1'**= *A → BC thus **A** is a **candidate key** of **R1.***
- Ristriction of R2 over F is **F2'**= E → *A thus **E** is **not a candidate key** of **R2.***
- *As* **R1∩ R2 = A** is a **superkey of R1** thus this **decomposition** is a **lossless-join decomposition .**

Exercise 2

- Show that the following decomposition of the schema ***R* = (*A, B, C, D, E*)** is not a lossless-join decomposition: **R1(A, B, C)**, **R2(C, D, E)** if the following set ***F*** of functional dependencies holds: **F**={ *A → BC, CD → E, B → D, E → A}*

- **Solution:**
- As R1∩ R2 = **C**
- Ristriction of **R1** over F is **F1'**= *A → BC thus **A** is a **candidate key** of **R1**.*
- Ristriction of **R2** over F is **F2'**= CD → *E thus **CD** is a **candidate key** of **R2.***
- *As* **R1∩ R2 = C** is **not a superkey** of either **R1** or **R2** thus this **decomposition** is not a **lossless-join decomposition .**

Exercise 3

- Consider the universal relational schema **R**(A, B, C, D, E, F, G, H, I, J) and a set of following functional dependencies **F**= {AB → C, A → DE, B → F, F → GH, D → IJ}. Determine the **keys** for R? Decompose **R** into **2NF** and then in **3NF.**
- **Solution:** Here, **AB** are not present in the **RHS**, so Let us find **AB⁺** w.r.t. the set of functional dependencies **F**:
- **AB⁺** = AB

  = ABC using AB → C

  = ABCDE using A → DE

  = ABCDEF using B → F

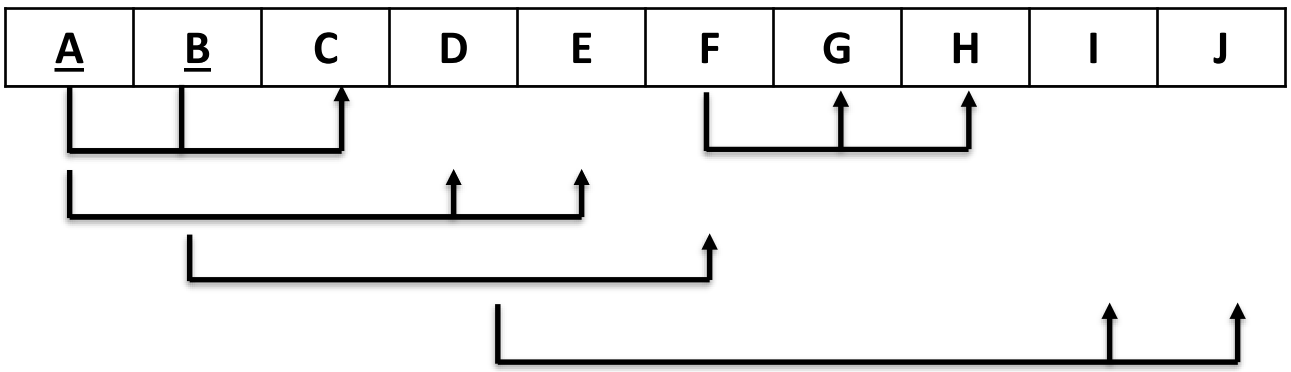
  = ABCDEFGH using F → GH

  = **ABCDEFGHIJ** using D → IJ

As **AB⁺** contains **all the attributes** of R thus **AB** is a **key attribute** of **R.**

Exercise 3

- Given a relational schema **R(A, B, C, D, E, F, G, H, I, J)** and a set of following functional dependencies **F=** {AB → C, A → DE, B → F, F → GH, D → IJ}.



2NF

R1(A,B,C)

R2(A,D,E,I,J)

R3(B,F,G,H)

3NF

R1(A,B,C)

R21(A,D,E) R22(D,I,J)

R31(B,F) R32(F,G,H)

Exercise 4

- Consider the universal relation **R**={A, B, C, D, E, F, G, H, I, J} and the set of functional dependencies **F**={ AB → C, BD → EF, AD → GH, A → I, H → J}. What is the key for R? Normalize the relation R upto **3NF**, justify your answer.

- **Solution:** To find the key check the RHS of the given functional dependencies, and find the closure of the attributes not present at the RHS.
- Here, **ABD** are not present in the **RHS**, so

  (ABD)+ = ABD = ABCD (since AB → C)

  = ABCDEF(since BD → EF)

  =ABCDEFGH(since AD → GH)

  =ABECDFGHI(since A → I)

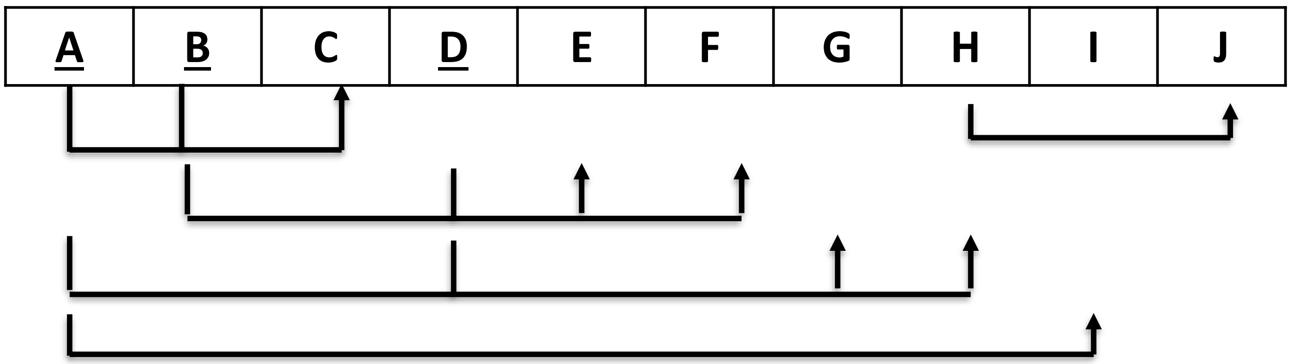
  =ABECDFGHIJ(since H → J)

  =**ABCDEFGHI**

  As **ABD**$^{+}$ contains **all the attributes** of **R** thus **ABD** is a **key attribute** of **R.**

Exercise 4

- Given a relational schema **R(A, B, C, D, E, F, G, H, I, J)** and a set of following functional dependencies **F=** {AB → C, BD → EF, AD → GH, A → I, H → J}.



2NF

R1(<u>A,B</u>,C)

R2(<u>B,D</u>,E,F)

R3(<u>A,D</u>,G,H,J)

R4(<u>A</u>,I)

3NF

R1(<u>A,B</u>,C)

R2(<u>B,D</u>,E,F)

R31(<u>A,D</u>,G,H) , R31(<u>H</u>,J)

R4(<u>A</u>,I)