Relational Model: Definitions Review

- Relations/tables, Attributes/Columns, Tuples/rows
- Attribute domains
- Superkey
- Key
  - No two tuples can have the same value in the key attribute
  - Primary key, candidate keys
  - No primary key value can be null
- Referential integrity constraints
  - Foreign key
  - Next…how to design the schema?

Relational Schema Design

- Logical Level
  - Whether schema has intuitive appeal for users
- Manipulation level
  - Whether it makes sense from an efficiency or correctness point of view

Summary of Problems

- Insertion, Deletion, modification anomalies
- Too many NULLs
- Spurious tuples – called non-additive join
- We need a theory of schema design
  - Functional dependencies and normalization
- Using functional dependencies define “normal forms” of schema
  - A schema in a “Third Normal Form” will avoid certain anomalies
Normalization

- **Normalization** is a technique for producing relations with desirable properties.
  - Using concept of functional dependencies
  - Normalization decomposes relations into smaller relations that contain less redundancy. This decomposition requires that no information is lost and reconstruction of the original relations from the smaller relations must be possible.
  - Normalization is a bottom-up design technique for producing relations. It pre-dates ER modeling and was developed by Codd in 1972 and extended by others over the years.
    - Normalization can be used after ER modeling or independently.
    - Normalization may be especially useful for databases that have already been designed without using formal techniques.

Normalization Motivation

- The goal of normalization is to produce a set of relational schemas \( R_1, R_2, ..., R_m \) from a set of attributes \( A_1, A_2, ..., A_n \).
  - Imagine that the attributes are originally all in one big relation \( R = \{A_1, A_2, ..., A_n\} \) which we will call the **Universal Relation**.
  - Normalization divides this relation into \( R_1, R_2, ..., R_m \).

Desirable Relational Schema Properties

- Relational schemas that are well-designed have several important properties:
  - 1) The most basic property is that relations consist of attributes that are logically related.
      - The attributes in a relation should belong to only one entity or relationship.
  - 2) **Lossless-join property** ensures that the information decomposed across many relations can be reconstructed using natural joins.
  - 3) **Dependency preservation property** ensures that constraints on the original relation can be maintained by enforcing constraints on the normalized relations.
  - 4) Avoid update anomalies

Functional Dependencies

- Functional dependencies represent constraints on the values of attributes in a relation and are used in normalization.

- A **functional dependency** (abbreviated **FD**) is a statement about the relationship between attributes in a relation. We say a set of attributes \( X \) functionally determines an attribute \( Y \) if given the values of \( X \) we always know the only possible value of \( Y \).
  - Notation: \( X \rightarrow Y \)
  - \( X \) functionally determines \( Y \)
  - \( Y \) is functionally dependent on \( X \)
  - Example:
    - \( eno \rightarrow ename \)
    - \( eno, pno \rightarrow hours \)
**Functional Dependencies**

Describe “Key-Like” Relationships

A key is a set of attributes where:
- If keys match, then the tuples match

A functional dependency (FD) is a generalization:
- If an attribute set determines another, written \( A \rightarrow B \)
  - then if two tuples agree on attribute set \( A \), they must agree on \( B \):

\[
\text{sid} \rightarrow \text{name}
\]

- FDs are independent of our schema design choice

---

**Notation for Functional Dependencies**

- A functional dependency has a left-side called the **determinant** which is a set of attributes, and one attribute on the right-side.

\[
\text{eno, pno} \rightarrow \text{hours}
\]

- Strictly speaking, there is always only one attribute on the RHS, but we can combine several functional dependencies into one:

\[
\text{eno, pno} \rightarrow \text{hours} \\
\text{eno, pno} \rightarrow \text{resp}
\]

\[
\text{eno, pno} \rightarrow \text{hours, resp}
\]

- Remember that this is really short-hand for two functional dependencies.

---

**Why the Name “Functional” Dependencies?**

- Functional dependencies get their name because you could imagine the existence of some function that takes in the parameters of the left-hand side and computes the value on the right-hand side of the dependency.

- Example: \( \text{eno, pno} \rightarrow \text{hours} \)

\[
\text{f}(\text{eno, pno}) \rightarrow \text{hours}
\]

\[
\text{int } f (\text{String eno, String pno})
\]

\[
\begin{cases}
// \text{ Do some lookup...} \\
\text{return hours;}
\end{cases}
\]

- Remember that no such function exists, but it may be useful to think of FDs this way.

---

**Formal Definition of FD’s**

Given a relation schema \( R \) and subsets \( X, Y \) of \( R \):

- An instance \( r \) of \( R \) satisfies FD \( X \rightarrow Y \) if,
  - for any two tuples \( t_1, t_2 \in r \),
  - \( t_1[X] = t_2[X] \) implies \( t_1[Y] = t_2[Y] \)
    - if they have the same values in \( X \) attributes/columns, they have the same values in \( Y \) attributes/columns

**If we know value of \( X \) then values of \( Y \) can be determined**

- For an FD to hold for schema \( R \), it must hold for every possible instance of \( r \)
  - (Can a DBMS verify this? Can we determine this by looking at an instance?)
The Semantics of Functional Dependencies

- Functional dependencies are a property of the domain being modeled NOT of the data instances currently in the database.
  - This means that similar to keys you cannot tell if one attribute is functionally dependent on another by looking at the data.

Example: Emp Relation

<table>
<thead>
<tr>
<th>eno</th>
<th>ename</th>
<th>bdate</th>
<th>title</th>
<th>salary</th>
<th>supereno</th>
<th>dno</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>J. Doe</td>
<td>01-05-75</td>
<td>EE</td>
<td>30000</td>
<td>E2</td>
<td>null</td>
</tr>
<tr>
<td>E2</td>
<td>M. Smith</td>
<td>06-04-66</td>
<td>SA</td>
<td>50000</td>
<td>E5</td>
<td>D3</td>
</tr>
<tr>
<td>E3</td>
<td>A. Lee</td>
<td>07-05-66</td>
<td>ME</td>
<td>40000</td>
<td>E7</td>
<td>D3</td>
</tr>
<tr>
<td>E4</td>
<td>J. Miller</td>
<td>09-01-50</td>
<td>PR</td>
<td>20000</td>
<td>E6</td>
<td>D3</td>
</tr>
<tr>
<td>E5</td>
<td>B. Casey</td>
<td>12-25-71</td>
<td>SA</td>
<td>50000</td>
<td>E8</td>
<td>D3</td>
</tr>
<tr>
<td>E6</td>
<td>L. Chu</td>
<td>11-30-65</td>
<td>EE</td>
<td>30000</td>
<td>E8</td>
<td>D3</td>
</tr>
<tr>
<td>E7</td>
<td>R. Davis</td>
<td>09-08-77</td>
<td>ME</td>
<td>40000</td>
<td>E8</td>
<td>D1</td>
</tr>
<tr>
<td>E8</td>
<td>J. Jones</td>
<td>10-11-72</td>
<td>SA</td>
<td>50000</td>
<td>null</td>
<td>D1</td>
</tr>
</tbody>
</table>

- List the functional dependencies of the attributes in this relation.

Trivial Functional Dependencies

- A functional dependency is trivial if the attributes on its left-hand side are a superset of the attributes on its right-hand side.

Examples: eno → eno
eno, ename → eno
eno, pno, hours → eno, hours

- Trivial functional dependencies are not interesting because they do not tell us anything.
  - Trivial FDs basically say "If you know the values of these attributes, then you uniquely know the values of any subset of those attributes.
- We are only interested in nontrivial FDs.

The Semantics of Functional Dependencies (2)

- Functional dependencies are directional.
  - eno → ename does not mean that ename → eno

Example:

Emp Relation

<table>
<thead>
<tr>
<th>eno</th>
<th>ename</th>
<th>bdate</th>
<th>title</th>
<th>salary</th>
<th>supereno</th>
<th>dno</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>J. Doe</td>
<td>01-05-75</td>
<td>EE</td>
<td>30000</td>
<td>E2</td>
<td>null</td>
</tr>
<tr>
<td>E2</td>
<td>M. Smith</td>
<td>06-04-66</td>
<td>SA</td>
<td>50000</td>
<td>E5</td>
<td>D3</td>
</tr>
<tr>
<td>E3</td>
<td>A. Lee</td>
<td>07-05-66</td>
<td>ME</td>
<td>40000</td>
<td>E7</td>
<td>D3</td>
</tr>
<tr>
<td>E4</td>
<td>J. Miller</td>
<td>09-01-50</td>
<td>PR</td>
<td>20000</td>
<td>E6</td>
<td>D3</td>
</tr>
<tr>
<td>E5</td>
<td>B. Casey</td>
<td>12-25-71</td>
<td>SA</td>
<td>50000</td>
<td>E8</td>
<td>D3</td>
</tr>
<tr>
<td>E6</td>
<td>L. Chu</td>
<td>11-30-65</td>
<td>EE</td>
<td>30000</td>
<td>E8</td>
<td>D3</td>
</tr>
<tr>
<td>E7</td>
<td>R. Davis</td>
<td>09-08-77</td>
<td>ME</td>
<td>40000</td>
<td>E8</td>
<td>D1</td>
</tr>
<tr>
<td>E8</td>
<td>J. Jones</td>
<td>10-11-72</td>
<td>SA</td>
<td>50000</td>
<td>null</td>
<td>D1</td>
</tr>
</tbody>
</table>

- Given an employee name there may be multiple values for eno if we have employees in the database with the same name.
- Thus knowing ename does not uniquely tell us the value of eno.

Question

- We have attributes (A1, A2, A3, A4) in a table
- We have functional dependencies

A1 → A2
A1 → A3
A1 → A4

Question: What property does A1 have?

A1 is a key since it determines all other attributes in the table
Functional Dependencies & Normal Forms

- Normalization requires decomposing a relation into smaller tables
- Normal forms are properties of relations
- We say a relation is in xNF if its attributes satisfy certain properties
  - Properties formally defined using functional dependencies
  - For example, test the relation to see if it is in 3NF
  - If not in 3NF, then change design…how?
  - Decomposition

How to go about designing a good schema?

- How to create a 3NF database schema? (i.e., a good design)?
- Ad-hoc approach
  - Create relations intuitively and hope for the best!
- Formal method – procedure Start with single relation with all attributes
  - Systematically decompose relations that are not in the desired normal form
  - Repeat until all tables are in desired normal form
  - Can decomposition create problems if we are not careful?
    Yes: (i) Spurious tuples and (ii) lost dependencies
  - Can we automate the decomposition process…
    Input: Set of attributes and their functional dependencies
    Output: A 'good' schema design

General Thoughts on Good Schemas

We want all attributes in every tuple to be determined only by the tuple’s key attributes, i.e. part of a superkey (for key $X \rightarrow Y$, a superkey is a “non-minimal” $X$)

What does this say about redundancy?

But:
- What about tuples that don’t have keys (other than the entire value)?

Sets of Functional Dependencies

- Relation EMP-DEPT(SSN, NAME, ADDRESS, DNUMBER, DNAME, MGRSSN)
  - Employee info; the dept they are assigned to; their manager’s ssn
  - Key is SSN
- Some obvious functional dependencies
  - [SSN] -> [NAME, ADDRESS, DNUMBER]
  - [DNUMBER] -> [DNAME, MGRSSN]
  - What else can we infer from above dependencies?
Sets of Functional Dependencies

- Some obvious functional dependencies
  - \{SSN\} \rightarrow \{NAME, ADDRESS, DNUMBER\}
  - \{DNUMBER\} \rightarrow \{DNAME, MGRSSN\}
- From above dependencies, we can infer
  - \{SSN\} \rightarrow \{DNAME, MGRSSN\}
- Concept of a set of dependencies that can be inferred from the given set
  - Inference rules?
  - Closure: F\+ is all dependencies that can be inferred from F

Some Questions:

- Given a set of functional dependencies (properties on the data), what other properties can we infer?
- What is the formal definition of a key?
- How can we use the formal framework of Functional dependencies to define a ‘good schema design’?
- Can we automate the process (develop algorithms)?

Next question:

- Given a set F of functional dependencies, what are all the properties, i.e. dependencies, we can infer?
- Do two sets of functional dependencies, F and G, imply the same set of properties?
- How to formally define this property?

Armstrong’s Axioms: Inferring FDs

Some FDs exist due to others; can compute using Armstrong’s axioms:

- Reflexivity: If \( Y \subseteq X \) then \( X \rightarrow Y \) (trivial dependencies)
  - name, sid \rightarrow name
- Augmentation: If \( X \rightarrow Y \) then \( XW \rightarrow YW \)
  - cid \rightarrow subj so cid, exp-grade \rightarrow subj, exp-grade
- Transitivity: If \( X \rightarrow Y \) and \( Y \rightarrow Z \) then \( X \rightarrow Z \)
  - cid\rightarrow crnum and crnum \rightarrow subj
  - so cid \rightarrow subj
Armstrong’s Axioms Lead to...

- **Union:** If \( X \rightarrow Y \) and \( X \rightarrow Z \) then \( X \rightarrow YZ \)
- **Pseudotransitivity:** If \( X \rightarrow Y \) and \( WY \rightarrow Z \) then \( XW \rightarrow Z \)
- **Decomposition:** If \( X \rightarrow Y \) and \( Z \subseteq Y \) then \( X \rightarrow Z \)

Can prove these from Armstrong’s Axioms...

Why Armstrong’s Axioms?

Why are Armstrong’s axioms (or an equivalent rule set) appropriate for FD’s? They are:

- **Consistent:** any relation satisfying FD’s in \( F \) will satisfy those in \( F^+ \)
- **Complete:** if an FD \( X \rightarrow Y \) cannot be derived by Armstrong’s axioms from \( F \) then there exists some relational instance satisfying \( F \) but not \( X \rightarrow Y \)

➢ In other words, Armstrong’s axioms derive all the FD’s that should hold

Computing the Closure of FDs

- The transitivity rule of FDs can be used for three purposes:
  - 1) To determine if a given FD \( X \rightarrow Y \) follows from a set of FDs \( F \).
  - 2) To determine if a set of attributes \( X \) is a superkey of \( R \).
  - 3) To determine the set of all FDs (called the closure \( F^+ \)) that can be inferred from a set of initial functional dependencies \( F \).

- The basic idea is that given any set of attributes \( X \), we can compute the set of all attributes \( X^* \) that can be functionally determined using \( F \). This is called the **closure of \( X \) under \( F \)**.
  - For purpose #1, we know that \( X \rightarrow Y \) holds if \( Y \) is in \( X^* \).
  - For purpose #2, \( X \) is a superkey of \( R \) if \( X^* \) is all attributes of \( R \).
  - For purpose #3, we can compute \( X^* \) for all possible subsets \( X \) of \( R \) to derive all FDs (the closure \( F^+ \)).

Closure of a Set of FD’s

**Defn.** Let \( F \) be a set of FD’s.

Its **closure**, \( F^+ \), is the set of all FD’s:

\[ \{ X \rightarrow Y \mid X \rightarrow Y \text{ is derivable from } F \text{ by Armstrong’s Axioms} \} \]

**Which of the following are in the closure of our Student-Course FD’s?**

- \( \text{name} \rightarrow \text{name} \)
- \( \text{crnum} \rightarrow \text{subj} \)
- \( \text{cid} \rightarrow \text{subj} \) 
  - \( (\text{sid},\text{cid}) \rightarrow \text{expgrade} \)
- \( \text{crnum}, \text{sid} \rightarrow \text{subj} \)
- \( \text{crnum} \rightarrow \text{sid} \)
Computing the Attribute Closure

- The algorithm is as follows:
  - Given a set of attributes $X$.
  - Let $X^+ = X$
  - Repeat
    - Find a FD in $F$ whose left side is a subset of $X^+$.
      - Add the right side of $F$ to $X^+$.
    - Until ($X^+$ does not change)

- After the algorithm completes, you have a set of attributes $X^+$ that can be functionally determined from $X$. This allows you to produce FDs of the form:
  - $X \rightarrow A$ where $A$ is in $X^+$

Computing Attribute Set Closure

- For attribute set $X$, compute closure $X^+$ by:

$$
Closure \ X^+ := X; \\
repeat \ until \ no \ change \ in \ X^+ \ { \\
  if \ there \ is \ an \ FD \ U \rightarrow V \ in \ F \ such \ that \ U \ is \ in \ X^+ \ then \ add \ V \ to \ X^+ }
$$

Attribute Closure: Example

- Let $F$ be:
  - SSN $\rightarrow$ EName
  - PNUMBER $\rightarrow$ PNAME, PLOCATION
  - SSN, PNUMBER $\rightarrow$ HOURS

- What is the closure of [SSN, PNUMBER]

Attribute Set Closure and Keys

- If $X$ is a key over relation scheme $R$, then what is $X^+$
  - Formal definition of a Key
- How to determine the keys for relation $R$?
  - $R$ is a set of attributes $\{A_1, A_2, \ldots, A_n\}$
  - For each subset $S$ of $R$, compute $S^+$
    - If $S^+ = R$ then $S$ is Key
  - What is the "catch" here?
  - Can you improve this?
Example

- R = (C,T,H,R,S)
  - Course (C), Time (T), Hour (H), Room (R), Section (S), Grade (G)

C → T
CS → G
HS → R
HR → C
HT → R

Find all keys for this relation
Hint: What is the smallest attribute set that must be part of the key?

Attribute Set Closures

- If attribute A does not appear on RHS of any FD, then any key must contain A
- If X is a key, then anything containing X is a superkey
- If X is a key, and Y → X is a FD then Y is a key

Example 2: Find All keys of R

- R = (A,B,C,D,E)
- A → BC
- CD → E
- B → D
- E → A

Equivalence of FD sets

Defn. Two sets of FD’s, F and G, are equivalent if their closures are equivalent, $F^+ = G^+$

E.g., these two sets are equivalent:

\{XY → Z, X → Y\} and
\{X → Z, X → Y\}

- $F^+$ could contain a huge number of FD’s (exponential in the size of the schema?)
- Would like to have smallest “representative” FD set – the “cover” set for F
  - Why?
**Minimal Cover**

**Defn.** A FD set $F$ is **minimal** if:

1. Every FD in $F$ is of the form $X \rightarrow A$, where $A$ is a single attribute.
2. For no $X \rightarrow A$ in $F$ is:
   
   $F - \{ X \rightarrow A \}$ equivalent to $F$.
3. For no $X \rightarrow A$ in $F$ and $Z \subseteq X$ is:
   
   $F - \{ X \rightarrow A \} \cup \{ Z \rightarrow A \}$ equivalent to $F$.

**Defn.** $F$ is a **minimum cover** for $G$ if $F$ is minimal and is equivalent to $G$.

E.g.,

$\{X \rightarrow Z, X \rightarrow Y\}$ is a minimal cover for $\{XY \rightarrow Z, X \rightarrow Y\}$.

---

**Minimal Cover Algorithm**

- Put the FDs of $F$ in a standard form.
  - Obtain collection $G$ of equivalent FDs with single attribute on RHS (using decomposition axiom).
- Minimize the left hand side of each FD.
  - For each FD in $G$, check each attribute on LHS to see if it can be deleted while preserving equivalence to $F^+$.
- Delete redundant FDs.
- There is a polynomial time algorithm to find min cover.

---

**Functional Dependencies and Schema Design: Normal Forms**

- Normal forms are properties of relations.
- We say a relation is in $x$NF if its attributes satisfy certain properties.
  - Properties formally defined using functional dependencies.
  - For example, test the relation to see if it is in 3NF.
  - If not in 3NF, then change design…how?
    - Decomposition.
Normal Forms

- A relation is in a particular normal form if it satisfies certain normalization properties.
- There are several normal forms defined:
  - 1NF - First Normal Form
  - 2NF - Second Normal Form
  - 3NF - Third Normal Form
  - BCNF - Boyce-Codd Normal Form
  - 4NF - Fourth Normal Form
  - 5NF - Fifth Normal Form
- Each of these normal forms are stricter than the next. For example, 3NF is better than 2NF because it removes more redundancy/anomalies from the schema than 2NF.
- 3NF and BCNF are relevant to 'real design'…
  - Others are of academic interest

The two Important Normal Forms

- Boyce-Codd Normal Form (BCNF). For every relation scheme R and for every \( X \rightarrow A \) that holds over R,
  - either \( A \in X \) (it is trivial), or
  - \( X \) is a superkey for R
- Third Normal Form (3NF). For every relation scheme R and for every \( X \rightarrow A \) that holds over R,
  - either \( A \in X \) (it is trivial), or
  - \( X \) is a superkey for R, or
  - A is a member of some key for R

How to go about designing a good schema?

- How to create a 3NF database schema? (i.e., a good design)?
- Ad-hoc approach
  - Create relations intuitively and hope for the best!
- Formal method – procedure (automated tool)
  - Start with single relation with all attributes
  - Systematically decompose relations that are not in the desired normal form
  - Repeat until all tables are in desired normal form
- Can decomposition create problems if we are not careful?
  - (i) Spurious tuples and (ii) lost dependencies
Recall some Definitions
- Relation schema R
  - Superkey
  - Key
  - Candidate key – same as key
  - Primary key – a key designated for common use
  - Prime attribute – an attribute that belongs to some candidate key
  - Non-prime attribute – does not belong to any key

Formal Definitions
- We discussed lossless joins...how to define it formally?
  - Recall: bad decompositions create spurious tuples, and/or we cannot reconstruct the original data

Lossless Join Decomposition

\[ R_1, \ldots, R_k \text{ is a lossless join decomposition of } R \text{ w.r.t. an FD set } F \text{ if for every instance } r \text{ of } R \text{ that satisfies } F, \]
\[ \Pi_{R_1}(r) \bowtie \cdots \bowtie \Pi_{R_k}(r) = r \]

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>cid</th>
<th>subj</th>
<th>crnum</th>
<th>exp-grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sam</td>
<td>570103</td>
<td>SW</td>
<td>cs143</td>
<td>B</td>
</tr>
<tr>
<td>23</td>
<td>Dan</td>
<td>550103</td>
<td>DB</td>
<td>cs178</td>
<td>A</td>
</tr>
</tbody>
</table>

Testing for Lossless Join

\( R_1, R_2 \text{ is a lossless join decomposition of } R \text{ with respect to } F \text{ iff at least one of the following dependencies is in } F^+ \)

\[
(R_1 \cap R_2) \rightarrow R_1 - R_2 \\
(R_1 \cap R_2) \rightarrow R_2 - R_1
\]

- Set of attributes common to the two tables are key to one of the two table.

So for the FD set:
- sid → name
- cid → crnum, exp-grade
- crnum → subj

Is (sid, name) and (crnum, subj, cid, exp-grade) a lossless decomposition?
Dependency Preservation

- Ensures we can “easily” check whether a FD $X \rightarrow Y$ is violated during an update to a database:
  - The projection of an FD set $F$ onto a set of attributes $Z$, $F_Z$ is
    $\{X \rightarrow Y | X \rightarrow Y \in F, X \cup Y \subseteq Z\}$
    i.e., it is those FDs local to $Z$’s attributes
  - A decomposition $R_1, \ldots, R_k$ is dependency preserving if $F^+ = (F_{R_1} \cup \ldots \cup F_{R_k})^+$

- Why is this important/desirable?
- The decomposition hasn’t “lost” any essential FD’s, so we can check without doing a join

Example of Lossless and Dependency-Preserving Decompositions

Given relation scheme

| R(name, street, city, st, zip, item, price) |

And FD set

- name $\rightarrow$ street, city
- street, city $\rightarrow$ st
- street, city $\rightarrow$ zip
- name, item $\rightarrow$ price

Consider the decomposition

- $R_1$(name, street, city, st, zip)
- $R_2$(name, item, price)

- Is it lossless?
- Is it dependency preserving?

What if we added FD street, city $\rightarrow$ item?

FD’s and Keys

- Ideally, we want a design s.t. for each nontrivial dependency $X \rightarrow Y$, $X$ is a superkey for some relation schema in $R$ and all dependencies are preserved
  - We just saw that this isn’t always possible
  - What if a dependency is lost during decomposition, but we want to enforce the condition? ??
    - Is there anything in SQL that can help us enforce this dependency condition? Assert our value with Assertions (and triggers)
**1NF and 2NF**
- First Normal Form - assumed as condition
  - Value of any attribute is a single value
  - Domains of attribute contain only atomic values
    Cannot be sets of values

**First Normal Form (1NF)**
- A relation is in **first normal form (1NF)** if all its attribute values are atomic.
  - That is, a 1NF relation cannot have an attribute value that is:
    - a set of values (multi-valued attribute)
    - a set of tuples (nested relation)
  - 1NF is a standard assumption in relational DBMSs.
    - However, object-oriented DBMSs and nested relational DBMSs relax this constraint.
  - A relation that is not in 1NF is an **unnormalized** relation.

**A non-1NF Relation**
- Two ways to convert a non-1NF relation into a 1NF relation.
  - 1) **Splitting Method** - Divide the existing relation into two relations: non-repeating attributes and repeating attributes.
    Make a relation consisting of the primary key of the original relation and the repeating attributes. Determine a primary key for this new relation.
    Remove the repeating attributes from the original relation.
  - 2) **Flattening Method** - Create new tuples for the repeating data combined with the data that does not repeat.
    Introduces redundancy that will be later removed by normalization.
    Determine primary key for this flattened relation.

**Converting a non-1NF Relation to 1NF Using Splitting**
- Also need original primary key: eno

- Repeating group: (pno, resp, hours)
Converting a non-1NF Relation to 1NF Using Flattening

<table>
<thead>
<tr>
<th>eno</th>
<th>ename</th>
<th>pno</th>
<th>resp</th>
<th>hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>J. Doe</td>
<td>P1</td>
<td>Manager</td>
<td>12</td>
</tr>
<tr>
<td>E2</td>
<td>M. Smith</td>
<td>P1</td>
<td>Analyst</td>
<td>24</td>
</tr>
<tr>
<td>E3</td>
<td>A. Lee</td>
<td>P3</td>
<td>Consultant</td>
<td>10</td>
</tr>
<tr>
<td>E4</td>
<td>J. Miller</td>
<td>P2</td>
<td>Programmer</td>
<td>18</td>
</tr>
<tr>
<td>E5</td>
<td>B. Casey</td>
<td>P2</td>
<td>Manager</td>
<td>24</td>
</tr>
<tr>
<td>E6</td>
<td>L. Chu</td>
<td>P4</td>
<td>Manager</td>
<td>48</td>
</tr>
<tr>
<td>E7</td>
<td>J. Jones</td>
<td>P3</td>
<td>Engineer</td>
<td>36</td>
</tr>
</tbody>
</table>

Second Normal Form (2NF)
- A relation is in second normal form (2NF) if it is in 1NF and every non-prime attribute is fully functionally dependent on a candidate key.
- A prime attribute is an attribute in any candidate key.
- Alternate definition: there is no partial dependency
- If there is a FD X → Y that violates 2NF:
  - Compute X⁺.
  - Replace R by relations: \( R_1 = X⁺ \) and \( R_2 = (R - X⁺) U X \)
- Note:
  - By definition, any relation with a single key attribute is in 2NF.

Partial Dependency
- A FD X → Y is a partial dependency if there exists an attribute A ∈ X such that X – A → Y
- Y is partially dependent on X
- Second Normal Form: Relation is in 2NF if no non-prime attribute is partially dependent on the primary key.

Problems with Partial Dependency

EMP_PROJ( SSN, PNUMBER, HOURS, ENAME, PNAME, PLOCATION)

Some FDs:
- \{SSN, PNUMBER\} → HOURS
- \{SSN, PNUMBER\} → ENAME

Since SSN → ENAME, ENAME is partially dependent on the primary key \{SSN, PNUMBER\}
- So why is this a problem?
Problems with partial dependencies

- Insert tuple
  - \( <987654321, 3, 12, \text{Jones}, \text{Sprite}, \text{Atlanta}> \)
- We have insertion anomaly
  - Check if 987654321 is Jones, project 3 is Sprite...
- We have deletion problem
  - If last tuple with Project #1 is deleted
- Similarly, we have modification anomaly
  - Smith changes name to Brown

Second Normal Form (2NF) Example

- EmpProj relation:

<table>
<thead>
<tr>
<th>eno</th>
<th>ename</th>
<th>title</th>
<th>bdate</th>
<th>salary</th>
<th>supereno</th>
<th>dno</th>
<th>pno</th>
<th>pname</th>
<th>budget</th>
<th>resp</th>
<th>hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fd2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fd3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fd4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

fd1 and fd4 are partial functional dependencies. Normalize to:
- Emp (eno, ename, title, bdate, salary, supereno, dno)
- WorksOn (eno, pno, resp, hours)
- Proj (pno, pname, budget)

Second Normal Form (2NF) Example (2)

- Emp relation:

<table>
<thead>
<tr>
<th>eno</th>
<th>ename</th>
<th>title</th>
<th>bdate</th>
<th>salary</th>
<th>supereno</th>
<th>dno</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fd2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- WorksOn relation:

<table>
<thead>
<tr>
<th>eno</th>
<th>pno</th>
<th>resp</th>
<th>hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fd4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Proj relation:

<table>
<thead>
<tr>
<th>pno</th>
<th>pname</th>
<th>budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem with Transitive Dependencies

- EMP_DEPT(ENAME, SSN, BDATE, ADDRESS, DNO, DNAME, MGRSSN)

FDs in relation:

- \( \{\text{SSN}\} \rightarrow \{\text{DNO}\} \)
- \( \{\text{DNO}\} \rightarrow \{\text{MGRSSN}\} \)
- \( \{\text{DNO}\} \rightarrow \{\text{DNAME}\} \)

- Insertion, Deletion, Modification anomalies in above schema

Third Normal Form (3NF)

- A relation is in third normal form (3NF) if it is in 2NF and there is no non-prime attribute that is transitively dependent on the primary key.

- That is, for all functional dependencies \( X \rightarrow Y \) of \( R \), one of the following holds:
  - \( Y \) is a prime attribute of \( R \)
  - \( X \) is a superkey of \( R \)

Normalization Question

- Consider the universal relation \( R(A,B,C,D,E,F,G,H,I,J) \) and the set of functional dependencies:
  - \( F = \{ A,B \rightarrow C ; A \rightarrow D,E ; B \rightarrow F ; F \rightarrow G,H ; D \rightarrow I,J \} \)

- List the keys for \( R \).

- Decompose \( R \) into 2NF and then 3NF relations.

Problem with 3NF?

- ADDR_INFO(CITY, ADDRESS, ZIP)
  - \( \{\text{CITY, ADDRESS}\} \rightarrow \text{ZIP} \)
  - \( \{\text{ZIP}\} \rightarrow \{\text{CITY}\} \)

Possible keys: \( \{\text{CITY, ADDRESS}\} \) or \( \{\text{ADDRESS, ZIP}\} \)

Is it in 3NF?
Problems with the 3NF schema

- Delete <Washington, 800 22nd St, 20052>
- What if this is the last 20052 tuple?
  - We lose the info that 20052 is in Washington
  - We also have insert, modify anomalies
- Why the problem?
  - Dependencies from an attribute to part of a key
- Solution?
  - Make all LHS of dependencies be key or superkey!
- **BCNF – Boyce Codd Normal Form**: if all FDs are of the form X → Y where X is superkey.

General Definition of 3NF, BCNF

- Can simplify the 3NF definition to remove the reference to partial dependencies/2NF
- R is in 3NF if for every FD X → Y, either
  - X is a superkey or
  - Y is a prime attribute
- R is in BCNF if for every FD X → Y, X is a superkey
  - R in BCNF ⇒ R is in 3NF

BCNF Decomposition Algorithm

Input: Relation R (consisting of all attributes), set of functional dependencies F
Output: BCNF schema result

result := {R}
while there is a schema Ri in result that is not in BCNF
{| let A → B be a FD that violates BCNF in relation Ri |
  | result := (result - Ri) ∪ {(Ri - B), (A,B)} |
|}

Example 1

- R= (C,T,H,R,S)
  - Course (C), Time (T), Hour (H), Room (R), Section (S), Grade (G)
  - C → T, CS → G
  - HS → R, HR → C
  - HT → R
- Key= {HS}
- Prime attributes = {H,S}
**Testing for 3NF, BCNF**
- Is the schema in BCNF?
  - Check if there are non-BCNF dependencies
- Is the schema in 3NF?
  - Check if there are non-3NF dependencies
  - Is there a dependency to non-prime attribute from something that is not a key?

**Normalization Procedure: Summary**
- Input= (Set of dependencies F, Set of attributes – single table schema)
  1. Use attribute set closure algo to find (a) keys and (b) prime attributes
    - Prune the search using the various “tricks”
  2. Test each FD in F to see if it satisfies 3NF/BCNF properties
  3. Decompose into smaller relations using decomposition algorithm
  4. If BCNF is not dependency preserving, then go with a 3NF decomposition

**Decomposition into BCNF: Problems**
- Consider relation R with FDs F. If \( X \rightarrow Y \) violates BCNF, decompose R into \( R - Y \) and \( XY \).
  - Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
  - e.g., CSJDQV, key C (C \( \rightarrow \) everything), JP \( \rightarrow \) C, SD \( \rightarrow \) P, J \( \rightarrow \) S
  - To deal with SD \( \rightarrow P \), decompose into SDP, CSJDQV.
  - To deal with J \( \rightarrow S \), decompose CSJDQV into JS and CJDQV
- In general, several dependencies may cause violation of BCNF. The order in which we “deal with” them could lead to very different sets of relations!

**BCNF and Dependency Preservation**
- In general, there may not be a dependency preserving decomposition into BCNF.
- Example: decomposition of CSJDQV into SDP, JS and CJDQV is not dependency preserving (we lose the FD JP \( \rightarrow C \), non BCNF FDs SD \( \rightarrow P \) and J \( \rightarrow S \)).
  - However, it is a lossless join decomposition.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
  - JPC tuples stored only for checking FD! *(Redundancy!)*
Example 2: Test if in BCNF or 3NF

- $R = (A,B,C,D,E)$
- $A \rightarrow BC$
- $CD \rightarrow E$
- $B \rightarrow D$
- $E \rightarrow A$

Normal Forms Compared

- BCNF is preferable, but sometimes in conflict with the goal of dependency preservation
  - It’s strictly stronger than 3NF
- Let’s see algorithms to obtain:
  - A BCNF lossless join decomposition
  - A 3NF lossless join, dependency preserving decomposition
  - Read this on your own from the resources

Conclusion

- **Normalization** is produces relations with desirable properties and reduces redundancy and update anomalies.

- Normal forms indicate when relations satisfy certain properties.
  - 1NF - All attributes are atomic.
  - 2NF - All attributes are fully functionally dependent on a key.
  - 3NF - There are no transitive dependencies in the relation.
  - BCNF – 3NF and all LHS are superkeys.
- In practice, normalization is used to improve schemas produced after ER design and existing relational schemas.
  - Full normalization is not always beneficial as it may increase query time.
  - Problem with relational DBs and large data sets….NoSQL !!