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.
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 → ename
- eno, pno → hours

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 $t1, t2 \in r$,
  - 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?)

Dependency Set closure & Attribute Set Closure

- Armstrong’s axioms allow computing closure $F^+$ of set of dependencies F
  - All dependencies that can be inferred from F

- Equivalence of functional dependency sets
  - $F$ is equivalent to $G$ if and only if $F^+ = G^+$

- Algorithm to compute Attribute set closure $X^+$ for set of attributes X
  - Set of attributes functionally dependent on X
    - Set X is key, if $X^+ = \{ \text{all attributes in the relation} \}$

- Concept of minimal cover $G$ of functional dependencies $F$
  - Smallest set $G$ such that $G^+ = F^+$

Algorithm to determine the key of a relation

- Finding a Key $K$ for $R$, given a set $F$ of Functional Dependencies
  - Input: A universal relation $R$ and a set of functional dependencies $F$ on the attributes of $R$.
    1. Set $K := R$;
    2. For each attribute $A$ in $K$
       - Compute $(K - A)^+$ with respect to $F$;
       - If $(K - A)^+$ contains all the attributes in $R$,
         then set $K := K - [A]$;


Definitions of Keys & Superkeys

- A **superkey** of a relation schema \( R = \{A_1, A_2, ..., A_n\} \) is a set of attributes \( S \subset R \) with the property that no two tuples \( t_1 \) and \( t_2 \) in any legal relation state \( r \) of \( R \) will have \( t_1[S] = t_2[S] \).

- A **key** \( K \) is a superkey with the additional property that removal of any attribute from \( K \) will cause \( K \) not to be a superkey any more.

Definitions of Attributes Participating in Keys

- If a relation schema has more than one key, each is called a **candidate key**.
- One of the candidate keys is arbitrarily designated to be the **primary key**, and the others are called **secondary keys**.
- A **Prime attribute** must be a member of some candidate key.
- A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.

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

Finding Minimal Cover of F.D.s

- Just as we applied inference rules to expand on a set \( F \) of FDs to arrive at \( F^+ \), its closure, it is possible to think in the opposite direction to see if we could shrink or reduce the set \( F \) to its minimal form so that the minimal set is still equivalent to the original set \( F \).
- **Definition:** An attribute in a functional dependency is considered **extraneous attribute** if we can remove it without changing the closure of the set of dependencies. Formally, given \( F \), the set of functional dependencies and a functional dependency \( X \rightarrow A \in F \), attribute \( Y \) is extraneous in \( X \) if \( Y \) is a subset of \( X \), and \( F \) logically implies \((F - (X \rightarrow A)) \cup \{(X - Y) \rightarrow A \})\)
Minimal Sets of FDs

- A set of FDs is **minimal** if it satisfies the following conditions:
  1. Every dependency in F has a single attribute for its RHS.
  2. We cannot remove any dependency from F and have a set of dependencies that is equivalent to F.
  3. We cannot replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$, where Y is a proper-subset of X and still have a set of dependencies that is equivalent to F.

Algorithm for Minimal Sets of FDs

- **Algorithm:** Finding a Minimal Cover F for a Set of Functional Dependencies E
  2. Replace each functional dependency $X \rightarrow \{A_1, A_2, ..., A_n\}$ in F by the n functional dependencies $X \rightarrow A_1$, $X \rightarrow A_2$, ..., $X \rightarrow A_n$.
  3. For each functional dependency $X \rightarrow A$ in F for each attribute B that is an element of X if $(F - \{X \rightarrow A\}) \cup \{(X - \{B\}) \rightarrow A\}$ is equivalent to F then replace $X \rightarrow A$ with $(X - \{B\}) \rightarrow A$ in F.
    (*The above constitutes a removal of the extraneous attribute B from X*)
  4. For each remaining functional dependency $X \rightarrow A$ in F if $(F - \{X \rightarrow A\})$ is equivalent to F, then remove $X \rightarrow A$ from F.
    (*The above constitutes a removal of the redundant dependency X \rightarrow A from F*)

Algorithm for Computing the Minimal Sets of FDs

Example:
Let the given set of FDs be $E : \{B \rightarrow A, D \rightarrow A, AB \rightarrow D\}$. We have to find the minimum cover of E.
- All above dependencies are in canonical form; so we have completed step 1 of Algorithm 10.2 and can proceed to step 2. In step 2 we need to determine if $AB \rightarrow D$ has any redundant attribute on the left-hand side; that is, can it be replaced by $B \rightarrow D$ or $A \rightarrow D$?
- Since $B \rightarrow A$, by augmenting with $B$ on both sides (IR2), we have $BB \rightarrow AB$, or $B \rightarrow AB$ (i). However, $AB \rightarrow D$ as given (ii).
- Hence by the transitive rule (IR3), we get from (i) and (ii), $B \rightarrow D$. Hence $AB \rightarrow D$ may be replaced by $B \rightarrow D$.
- We now have a set equivalent to original E, say $E' : \{B \rightarrow A, D \rightarrow A, B \rightarrow D\}$.
  No further reduction is possible in step 2 since all FDs have a single attribute on the left-hand side.
- In step 3 we look for a redundant FD in E'. By using the transitive rule on $B \rightarrow D$ and $D \rightarrow A$, we derive $B \rightarrow A$. Hence $B \rightarrow A$ is redundant in $E'$ and can be eliminated.
- Hence the minimum cover of E is $(B \rightarrow D, D \rightarrow A)$.

Minimal Sets of FDs -- Complexity

- Every set of FDs has an equivalent minimal set
- There can be several equivalent minimal sets
- There is no simple algorithm for computing a minimal set of FDs that is equivalent to a set F of FDs. The process of previous algorithm is used until no further reduction is possible.
- To synthesize a set of relations, we assume that we start with a set of dependencies that is a minimal set.
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

Functional Dependencies and Schema Design: Normal Forms

- 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

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 $X \to A$ 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 $X \to A$ 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 $R$.

Now ready to define Normal Forms...

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.
  - NoSQL DBs do not have this assumption...in fact, it *is a feature*!

- A relation that is not in 1NF is an *unnormalized* relation.

A non-1NF Relation

<table>
<thead>
<tr>
<th>emp</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,P2}</td>
<td>Analyst,Analyst</td>
<td>{24,6}</td>
</tr>
<tr>
<td>E3</td>
<td>A. Lee</td>
<td>{P3,P4}</td>
<td>Consultant,Engineer</td>
<td>{10,48}</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>
Converting non-1NF to 1NF

Two ways to convert a non-1NF relation to 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

<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>E2</td>
<td>M. Smith</td>
<td>P2</td>
<td>Analyst</td>
<td>6</td>
</tr>
<tr>
<td>E3</td>
<td>A. Lee</td>
<td>P3</td>
<td>Consultant</td>
<td>10</td>
</tr>
<tr>
<td>E3</td>
<td>A. Lee</td>
<td>P4</td>
<td>Engineer</td>
<td>48</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>

Repeating group: (pno, resp, hours)

Also need original primary key: eno

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>E2</td>
<td>M. Smith</td>
<td>P2</td>
<td>Analyst</td>
<td>6</td>
</tr>
<tr>
<td>E3</td>
<td>A. Lee</td>
<td>P3</td>
<td>Consultant</td>
<td>10</td>
</tr>
<tr>
<td>E3</td>
<td>A. Lee</td>
<td>P4</td>
<td>Engineer</td>
<td>48</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: R1 = X* and R2 = (R – X*) U X

- Note:
  
  - By definition, any relation with a single key attribute is in 2NF.
Partial Dependency

- A FD $X \rightarrow Y$ is a partial dependency if there exists an attribute $A \in X$ such that $X - A \rightarrow Y$
- $Y$ 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 dependencies

- Insert tuple
  - $<987654321, 3, 12, Jones, Sprite, Atlanta>$
- 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

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

```sql
{SSN, PNUMBER} → HOURS
{SSN, PNUMBER} → ENAME
SSN → ENAME, ENAME
```

Second Normal Form (2NF) Example

- **EmpProj relation:**
  - `eno` `pno` `resp` `hours`
  - `ename` `title` `bdate` `salary` `supereno` `dno` `pno` `pname` `budget`

  ```sql
  fd1
  fd2
  fd3
  fd4
  ```

  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

- **Emp relation:**
  - `eno` `pno` `resp` `hours`
  - `ename` `title` `bdate` `salary` `supereno` `dno`

  ```sql
  fd1
  fd2
  ```

- **WorksOn relation:**
  - `eno` `pno` `resp` `hours`

- **Proj relation:**
  - `pno` `pname` `budget`
Transitive Dependencies

• FD $X \rightarrow Y$ is a transitive dependency in relation $R$ if there exists set of attributes $Z \subseteq R$ such that
  • $X \rightarrow Z$ and $Z \rightarrow Y$
  • $Z$ is not a subset of any key of $R$

Problem with Transitive Dependencies

• EMP_DEPT(ENAME, SSN, BDATE, ADDRESS, DNO, DNAME, MGRSSN)
  FDs in relation:
  • $\{SSN\} \rightarrow \{DNO\}$
  • $\{DNO\} \rightarrow \{MGRSSN\}$
  • $\{DNO\} \rightarrow \{DNAME\}$

  ▪ Insertion, Deletion, Modification anomalies in above schema
  ▪ Recall earlier discussion on the same example.....

Transitive dependencies in the schema below?

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

FDs in relation:
  • $\{SSN\} \rightarrow \{DNO\}$
  • $\{DNO\} \rightarrow \{MGRSSN\}$
  • $\{DNO\} \rightarrow \{DNAME\}$

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 $R$
**Problem with 3NF?**

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

Possible keys: \{CITY, ADDRESS\} or \{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 \(X \rightarrow Y\) 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 \(X \rightarrow Y\), either
  - \(X\) is a superkey or
  - \(Y\) is a prime attribute

- \(R\) is in BCNF if for every FD \(X \rightarrow Y\), \(X\) is a superkey
  - \(R\) in BCNF \(\Rightarrow\) \(R\) is in 3NF

---

**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

Problems with decomposition?

- Spurious tuples
- Lost dependencies

Therefore, we want

- Lossless join decomposition
- Dependency preserving decomposition

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

- A loss join will not create spurious tuples
  - We can reconstruct the ‘original’ table without introducing ‘false’ data
  - $R_1, \ldots, R_k$ is a lossless join decomposition of $R$ w.r.t. an FD set $F$ if for every instance $r$ of $R$ that satisfies $F$, $\pi_{R_1}(r) \bowtie \ldots \bowtie \pi_{R_k}(r) = r$
Is (sid, name) and (cid, subj, crnum, exp-grade) a lossless join decomposition?

<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>5701</td>
<td>SW</td>
<td>cs143</td>
<td>B</td>
</tr>
<tr>
<td>23</td>
<td>Dan</td>
<td>5501</td>
<td>DB</td>
<td>cs178</td>
<td>A</td>
</tr>
</tbody>
</table>

Testing for Lossless Join

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

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

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

So for the FD set:

\[ \text{sid} \rightarrow \text{name} \]
\[ \text{cid} \rightarrow \text{crnum}, \text{exp-grade} \]
\[ \text{crnum} \rightarrow \text{subj} \]

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

Functional dependencies after decompositions

- Definition: Given a set of dependencies \( F \) on \( R \), the projection of \( F \) on \( R_i \), denoted by \( p_{R_i}(F) \) where \( R_i \) is a subset of \( R \), is the set of dependencies \( X \rightarrow Y \) in \( F^+ \) such that the attributes in \( X \cup Y \) are all contained in \( R_i \).

- Informally: each decomposed relation will have a set of dependencies, such that LHS and RHS from the original set \( F \) are both in the relation.
  - Let this set be \( G_i \) for each decomposed relation \( R_i \)
  - The set of relations after decomposition will have \( G = \text{union of all the sets } G_i \)

Properties of Relational Decompositions (5)

- Dependency Preservation Property of a Decomposition (cont.):
  - Dependency Preservation Property:
    - A decomposition \( D = \{ R_1, R_2, ..., R_m \} \) of \( R \) is dependency-preserving with respect to \( F \) if the union of the projections of \( F \) on each \( R_i \) in \( D \) is equivalent to \( F \); that is \( ((p_{R_1}(F)) \cup ... \cup (p_{R_m}(F)))^+ = F^+ \).
    - (See examples in Fig 14.13a and Fig 14.12)
  - Claim 1:
    - It is always possible to find a dependency-preserving decomposition \( D \) with respect to \( F \) such that each relation \( R_i \) in \( D \) is in 3nf.
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
    $$\{X \rightarrow Y \mid X \rightarrow Y \in F^+, X \subseteq Y \cap Z\}$$
    i.e., it is those FDs local to $Z$’s attributes
  - The dependencies that apply after decomposition into
    $(R_1, R_2, \ldots, R_k)$ are $F_{R_1}, F_{R_2}, \ldots, F_{R_k}$.
  - 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
$$R(\text{name, street, city, st, zip, item, price})$$
And FD set $\text{name} \rightarrow \text{street, city}$
street, city $\rightarrow$ st
street, city $\rightarrow$ zip
name, item $\rightarrow$ price
Consider the decomposition
$(R_1(\text{name, street, city, st, zip})$ and $R_2(\text{name, item, price})$

- 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 ?
  - Is there anything in SQL that can help us enforce this dependency condition ?
Decomposition Algorithms
- Algorithms for lossless join dependency preserving 3NF Decomposition
- Algorithm for lossless join BCNF decomposition

Decomposition into BCNF: Problems
- Consider relation R with FDs F. If X → 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., CSJDPQV, key C (C → everything), JP → C, SD → P, J → S
  - To deal with SD → P, decompose into SDP, CSJDQV.
  - To deal with J → S, decompose CSJQV 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 Decomposition Algorithm
- Input: Relation R (consisting of all attributes), set of functional dependencies F
- Output: BCNF schema result
- Algorithm: Initialize result := \{R\}
  - while there is a schema R_i in result that is not in BCNF
    { let A → B be a FD that violates BCNF in relation R_i
      result := (result – R_i) ∪ \{(R_i - B), (A,B)\}
    }
- Note that decomposition \{(R_i - B), (A,B)\} is lossless join
  - Since A is key for (A,B) and A is intersection of these two relations

BCNF and Dependency Preservation
- In general, there may not be a dependency preserving decomposition into BCNF.
- Example: decomposition of CSJQV into SDP, JS and CJDQV is not dependency preserving (we lose the FD JP → C, non BCNF FDs SD → P and J → S).
  - However, it is a lossless join decomposition.
  - In this case, adding JP → C to the collection of relations gives us a dependency preserving decomposition.
    JPC tuples stored only for checking FD! (Redundancy?)
Exercise 1
- $R = (C,T,H,R,S)$
  - Course (C), Time (T), Hour (H), Room (R), Section (S), Grade (G)
  - $C \rightarrow T$
  - $CS \rightarrow G$
  - $HS \rightarrow R$
  - $HR \rightarrow C$
  - $HT \rightarrow R$
  - What are the keys?
  - Is it in 3NF?
  - Is it in BCNF? If not, then decompose into BCNF

Exercise 2: Test if in BCNF or 3NF
- $R = (A,B,C,D,E)$
  - $A \rightarrow BC$
  - $CD \rightarrow E$
  - $B \rightarrow D$
  - $E \rightarrow A$
  - Keys:
  - Is it in 3NF?
  - Is it in BCNF? If not, then decompose into BCNF

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!!