Lectures 11, 13, 14

Functional Dependencies,
Normalization
The Dangers of Redundant Storage

- **Redundancy** is at the root of several problems associated with relational schemas:
  - redundant storage, insert/delete/update anomalies
- Redundancy arises when a relational schema forces an association between attributes that is not natural
- Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements
- Can *null* values be used to address this?
Decomposition

• Main refinement technique: decomposition (replacing ABCD with, say, AB and BCD, or ACD and ABD; note that when discussing this topic we shall refer to a relation schema by a string of letters, one per attribute)

• Decomposition: replace the relation schema by two (or more) schemas that each contain a subset of the attributes of R and together contain them all

• Decomposition should be used judiciously:
  – Is there reason to decompose a relation?
  – What problems (if any) does the decomposition cause?
Functional Dependencies (FDs)

• A functional dependency $X \rightarrow Y$ holds over relation $R$ if, for every allowable instance $r$ of $R$:
  – $t1 \in r$, $t2 \in r$, $\pi_X(t1) = \pi_X(t2)$ implies $\pi_Y(t1) = \pi_Y(t2)$
  – i.e., given two tuples in $r$, if the $X$ values agree, then the $Y$ values must also agree. ($X$ and $Y$ are sets of attributes.)

• An FD is a statement about all allowable relations.
  – Must be identified based on semantics of application.
  – Given some allowable instance $r1$ of $R$, we can check if it violates some FD $f$, but we cannot tell if $f$ holds over $R$!

• $K$ is a candidate key for $R$ means that $K \rightarrow R$
  – However, $K \rightarrow R$ does not require $K$ to be minimal!
FDs (cont.)

• An FD is a property of a particular relation schema and not of a specific instance

<table>
<thead>
<tr>
<th>eno</th>
<th>ename</th>
<th>byr</th>
<th>sal</th>
<th>dno</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Mike</td>
<td>60</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>Sally</td>
<td>55</td>
<td>85</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>Sara</td>
<td>65</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>Julie</td>
<td>70</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>John</td>
<td>48</td>
<td>80</td>
<td>1</td>
</tr>
</tbody>
</table>

• It may appear that $ename \rightarrow byr$ and $ename \rightarrow dno$
• However, I could add tuple

| 22 | Mike | 49 | 22 | 3  |
Example 1: Constraints on Entity Set

• Consider relation obtained from Hourly_Emps:
  – Hourly_Emps (ssn, name, lot, rating, hrly_wages, hrs_worked)

• We will denote this relation schema by listing the attributes: SNLRWH
  – This is really the set of attributes {S,N,L,R,W,H}.
  – Sometimes, we will refer to all attributes of a relation by using the relation name (e.g., Hourly_Emps for SNLRWH).

• Some FDs on Hourly_Emps:
  – ssn is the key: S → SNLRWH
  – rating determines hrly_wages: R → W
Example 1 (cont.)

- Problems due to $R \rightarrow W$:
  - *Update anomaly:* Can we change $W$ in just the 1st tuple of \text{SNLRWH}? 
  - *Insertion anomaly:* What if we want to insert an employee and don’t know the hourly wage for his rating? 
  - *Deletion anomaly:* If we delete all employees with rating 5, we lose the information about the wage for rating 5!

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>W</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

$\text{Hourly\_Emps2}$
Example 2

• Schema 1:
  – Emp(eno, ename, byr, sal, dno)
  – Dept(dno, dname, floor, mgr)

• Find all employees who make more than their manager

\[
\text{SELECT } E1.\text{ename} \\
\text{FROM Emp E1, Dept D, Emp E2} \\
\text{WHERE (D.mgr = E2.\text{eno}) AND (E.dno = D.dno)} \\
\text{AND (E1.sal > E2.sal)}
\]

• Find all departments with a max salary more than twice the average salary

\[
\text{SELECT D.dname} \\
\text{FROM Emp E, Dept D} \\
\text{WHERE (D.dno = E.dno)} \\
\text{GROUP BY D.dno, D.dname} \\
\text{HAVING MAX (E.sal) > 2*AVG (E.sal)}
\]
Example 2 (cont.)

- Schema 2:
  - ED(eno, ename, byr, sal, dno, dname, floor, mgr)
- Find all employees who make more than their manager:
  
  ```sql
  SELECT E1.ename
  FROM ED E1, ED E2
  WHERE (E1.mgr = E2.eno) AND (E1.sal > E2.sal)
  ```

- Find all departments with a max salary more than twice the average salary
  
  ```sql
  SELECT E.dname
  FROM ED E
  GROUP BY E.dno, E.dname
  HAVING MAX (E.sal) > 2*AVG (E.sal)
  ```
Example 2 (cont.)

• We got a schema that makes queries simpler but …
  – Redundancy
    • Each dept is repeated for each employee
    • There’s potential inconsistency (update anomalies)
  – No independent existence
    • A dept cannot exist without employees
    • Delete all employees of a dept and automatically lose the dept
Objectives of DB Design

- No redundancy for space efficiency
- Update/Insert/Delete integrity
- Semantic clarity
- Linguistic efficiency – the simpler the queries the better for both the user and the query optimizer
- Performance – binary relationships imply most queries will have a large number of joins
Kinds of FDs

• Distinguishing between the types of FDs helps to identify bad designs:
  – Trivial Dependency: e.g., AB → A (Identity)
  – Partial Dependency: AB is a key and A → C, C is not part of a candidate key
    • E.g., Supply(sno, sloc, projno, projloc, pno, qty)
      – Key is (sno, projno, pno)
      – sno → sloc, projno → projloc are partial dependencies
  – Transitive Dependency: A is a key, B is not, and A → B → C (B does not equal any candidate key, but B may or may not contain some attributes of a key)
    • E.g., ED(eno, ..., dno, ..., mgr)
      – Key is eno, eno → dno → mgr, so eno → mgr is a transitive dependency
Reasoning about FDs

- Given some FDs, can usually infer additional FDs:
  - $\text{ssn} \rightarrow \text{did}, \text{did} \rightarrow \text{lot}$ implies $\text{ssn} \rightarrow \text{lot}$
- An FD $f$ is implied by a set of FDs $F$ if $f$ holds whenever all FDs in $F$ hold.
  - $F^+ = \text{closure of } F$ is the set of all FDs implied by $F$.
- Armstrong’s Axioms ($X$, $Y$, $Z$ are sets of attributes):
  - Reflexivity: If $X$ is a subset of $Y$, then $Y \rightarrow X$
  - Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any $Z$
  - Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
- These are sound and complete inference rules for FDs!
Reasoning about FDs (cont.)

• Couple of additional rules (that follow from AA):
  – **Union**: If \( X \rightarrow Y \) and \( X \rightarrow Z \), then \( X \rightarrow YZ \)
    * Proof: \( X \rightarrow XZ \), \( XZ \rightarrow YZ \), hence \( X \rightarrow YZ \)
  – **Decomposition**: If \( X \rightarrow YZ \), then \( X \rightarrow Y \) and \( X \rightarrow Z \)

• Note: if \( B \rightarrow C \), then \( AB \rightarrow AC \) holds; however, the reverse does not necessarily hold, i.e. one cannot cancel \( A \) in \( AB \rightarrow AC \)!

• Example: Contracts(\( cid, sid, jid, did, pid, qty, value \)) and:
  – \( C \) is the key: \( C \rightarrow CSJDPQV \)
  – Project purchases each part using single contract: \( JP \rightarrow C \)
  – Dept purchases at most one part from a supplier: \( SD \rightarrow P \)

• \( JP \rightarrow C, C \rightarrow CSJDPQV \) imply \( JP \rightarrow CSJDPQV \)
• \( SD \rightarrow P \) implies \( SDJ \rightarrow JP \)
• \( SDJ \rightarrow JP, JP \rightarrow CSJDPQV \) imply \( SDJ \rightarrow CSJDPQV \)
Computing the Closure

• Using Armstrong’s Axioms, find the closure of the set of functional dependencies
  \( F = \{ A \rightarrow B, A \rightarrow D, B \rightarrow C, C \rightarrow A \} \)

• It can be helpful to use a graph to help infer all possible transitive dependencies.

• Are we done? No, augmentation yields more dependencies like \( AC \rightarrow BC, BD \rightarrow CD \), etc.
Computing the Closure (cont.)

• Computing the closure of a set of FDs can be expensive. (Size of closure is exponential in # attrs!)

• Typically, we just want to check if a given FD $X \rightarrow Y$ is in the closure of a set of FDs $F$. An efficient check:
  – Compute attribute closure of $X$ (denoted $X^+$) wrt $F$:
    • Set of all attributes $A$ such that $X \rightarrow A$ is in $F^+$
    • There is a linear time algorithm to compute this.
  – Check if $Y$ is in $X^+$

• Does $F = \{A \rightarrow B, B \rightarrow C, CD \rightarrow E\}$ imply $A \rightarrow E$?
  – i.e., is $A \rightarrow E$ in the closure $F^+$? Equivalently, is $E$ in $A^+$?
Normal Forms

• Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!

• If a relation is in a certain normal form (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized. This can be used to help us decide whether decomposing the relation will help.

• Role of FDs in detecting redundancy:
  – Consider a relation R with 3 attributes, ABC.
    • No FDs hold: There is no redundancy here.
    • Given A → B: Several tuples could have the same A value, and if so, they’ll all have the same B value!
  – Depending on the existence of various FDs, schemas are classified into different normal forms.
Overview of Normal Forms

• 1NF – Every attribute has an atomic value (as opposed to being set-valued). Note: all relational model schemas are by definition 1NF!
• 2NF – 1NF and no partial dependencies (a source of redundancy)
  – E.g., 1NF but not 2NF: Supply(sno, sloc, projno, projloc, pno, qty) and projno → projloc
• 3NF – 2NF and no transitive dependencies to attributes that are not part of a key.
  – E.g., 2NF but not 3NF: ED(eno, …, dno, …, floor, mgr) and eno → dno, dno → floor
• BCNF – 3NF and no transitive dependencies.
  – E.g., 3NF but not BCNF: (cust_id, vin, purchase_id) and (cust_id, vin) → purchase_id, purchase_id → vin
• 4NF, 5NF, etc. – 3NF and restrictions on multivalued dependencies (rarely used in practice).
• Note that each normal form is contained in the next one higher up.
Boyce-Codd Normal Form (BCNF)

- Relation R with FDs $F$ is in BCNF if, for all $X \rightarrow A$ in $F^+$
  - $A \in X$ (a trivial FD), or
  - $X$ contains a key for R.
- In other words, R is in BCNF if the only non-trivial FDs that hold over R are key constraints.
  - No dependency in R that can be predicted using FDs alone.
  - If we are shown two tuples that agree upon the $X$ value, we cannot infer the $A$ value in one tuple from the $A$ value in the other.
  - If example relation is in BCNF and $X$ in fact does determine the value of $A$, then 2 tuples must be identical (since $X$ must then be a key).
Third Normal Form (3NF)

• Relation R with FDs F is in 3NF if, for all $X \rightarrow A$ in $F^+$
  – $A \in X$ (a trivial FD), or
  – X contains a key for R, or
  – A is part of some key for R.

• Minimality of a key is crucial in third condition above!
• If R is in BCNF, obviously in 3NF.
• If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no `good’ decomposition, or performance considerations).
  – Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible. On the other hand, testing whether a given schema is 3NF is an NP-complete problem!
What Does 3NF Achieve?

• If 3NF violated by $X \rightarrow A$, one of the following holds ($X$ is not equal to a key):
  - $X$ is a subset of some key $K$. Therefore, we store $(X, A)$ pairs redundantly.
  - $X$ is not a proper subset of any key, but may overlap part of a key, and so there is a chain of FDs $K \rightarrow X \rightarrow A$, which means that we cannot associate an $X$ value with a $K$ value unless we also associate an $A$ value with an $X$ value.

• But: even if relation is in 3NF, these problems could arise.
  - e.g., Reserves SBDC, $C \rightarrow S$, $S \rightarrow C$ is in 3NF, but for each reservation of sailor $S$, same $(S, C)$ pair is stored.

• Thus, 3NF is indeed a compromise relative to BCNF.
Decomposition of a Relation Schema

• Suppose that $R$ contains attributes $A_1, ..., A_n$. A decomposition of $R$ consists of replacing $R$ by two or more relations such that:
  – Each new relation schema contains a subset of the attributes of $R$ (and no attributes that do not appear in $R$), and
  – Every attribute of $R$ appears as an attribute of one of the new relations.

• Intuitively, decomposing $R$ means we will store instances of the relation schemas produced by the decomposition, instead of instances of $R$.

• E.g., can decompose SNLRWH into SNLRH and RW.
Example Decomposition

• Decompositions should be used only when needed.
  – SNLRWH has FDs $S \rightarrow SNLRWH$ and $R \rightarrow W$
  – Second FD causes violation of 3NF; W values repeatedly associated with R values. Easiest way to fix this is to create a relation RW to store these associations, and to remove W from the main schema:
    • i.e., we decompose SNLRWH into SNLRH and RW

• The information to be stored consists of SNLRWH tuples. If we just store the projections of these tuples onto SNLRH and RW, are there any potential problems that we should be aware of?
Problems with Decompositions

• There are three potential problems to consider:
  – (1) Some queries become more expensive.
    • e.g., How much did employee Joe earn? (salary = W*H)
  – (2) Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation, i.e. joining smaller relations resulting from decomposition may result in spurious tuples which did not exist in the original instance.
    • Fortunately, not in the SNLRWH example.
  – (3) Checking some dependencies may require joining the instances of the decomposed relations.
    • Fortunately, not in the SNLRWH example.

• Tradeoff: Must consider these issues vs. redundancy.
Lossless Join Decompositions

- Decomposition of R into X and Y is *lossless-join* wrt a set of FDs F if for every instance r that satisfies F:
  - $\pi_X(r) \bowtie \pi_Y(r) = r$

- It is always true that $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$
  - In general, the other direction does not hold! If it does, the decomposition is lossless-join.

- Definition extended to decomposition into 3 or more relations in a straightforward way.

- *It is essential that all decompositions used to deal with redundancy be lossless!* (Avoids Problem 2)
More on Lossless Join

- The decomposition of \( R \) into \( X \) and \( Y \) is lossless-join wrt \( F \) if and only if the closure of \( F \) contains:
  - \( X \cap Y \rightarrow X \), or
  - \( X \cap Y \rightarrow Y \)

- In particular, the decomposition of \( R \) into \( UV \) and \( R - V \) is lossless-join if \( U \rightarrow V \) holds over \( R \).
Proof of Lossless Join Theorem

- R is decomposed into X and Y.
- Assume $X \cap Y \rightarrow X$.
- Then, $X \cap Y$ is a (super)key for X.
- Then, each tuple in Y is joined with exactly one tuple in X.
- By a previous observation, R is contained in the join of X and Y, so $|Y| \geq |R|$.
- The number of tuples in Y cannot exceed the number of tuples in R since Y is a projection of R.
- Therefore, the number of tuples in Y has to equal the number of tuples in R and so the join of X and Y must equal R.
- The “only if” direction can be proven by assuming lossless join and the negation of the key fd and deriving a contradiction.
Dependency Preserving Decomposition

- Consider CSJDPQV, C is key, JP → C and SD → P.
  - BCNF decomposition: CSJDPQV and SDP
  - Problem: Checking JP → C requires a join!

- Dependency preserving decomposition (Intuitive):
  - If R is decomposed into X, Y and Z, and we enforce the FDs that hold on X, on Y and on Z, then all FDs that were given to hold on R must also hold. (Avoids Problem 3)

- Projection of set of FDs F: If R is decomposed into X, ..., projection of F onto X (denoted $F_X$) is the set of FDs $U → V$ in $F^+$ (closure of F) such that U, V are in X.
Dependency Preserving Decomposition (cont.)

- Decomposition of R into X and Y is dependency preserving if $(F_X U F_Y)^+ = F^+$
  - i.e., if we consider only dependencies in the closure $F^+$ that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in $F^+$.

- Important to consider $F^+$, not F, in this definition:
  - ABC, $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow A$, decomposed into AB and BC.
  - Is this dependency preserving? Is $C \rightarrow A$ preserved?

- Dependency preserving does not imply lossless join:
  - ABC, $A \rightarrow B$, decomposed into AB and BC.

- And vice-versa! (Example?)
Decomposition into BCNF

• 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, JP → C, SD → P, J → S
  - To deal with SD → P, decompose into SDP, CSJDQV.
  - To deal with J → 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.
  - e.g., SBD, SB → D, D → B
  - Can’t decompose while preserving 1st FD; not in BCNF.

• Similarly, decomposition of CSJDQV into SDP, JS and CJDQV is not dependency preserving (wrt the FDs JP → C, SD → P and J → 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!)
Decomposition into 3NF

- Obviously, the algorithm for lossless join decomposition into BCNF can be used to obtain a lossless join decomposition into 3NF (typically, can stop earlier).

- To ensure dependency preservation, one idea:
  - If \( X \rightarrow Y \) is not preserved, add relation \( XY \).
  - Problem is that \( XY \) may violate 3NF! e.g., consider the addition of CJP to `preserve’ JP \( \rightarrow C \). What if we also have \( J \rightarrow C \)?

- Refinement: Instead of the given set of FDs F, use a *minimal cover for F*. 
Minimal Cover for a Set of FDs

• *Minimal cover* $G$ for a set of FDs $F$:
  - Closure of $F = $ closure of $G$.
  - Right hand side of each FD in $G$ is a single attribute, i.e. every dependency is of the form $X \rightarrow A$, where $A$ is a single attribute but $X$ can be a collection of attributes.
  - If we modify $G$ by deleting an FD or by deleting attributes from an FD in $G$, the closure of $G$ is no longer equal to the closure of $F$.

• Intuitively, every FD in $G$ is needed, and "as small as possible" in order to get the same closure as $F$. 
Example of a Minimal Cover

• A → B, ABCD → E, EF → GH, ACDF → EG

• First, rewrite
  – EF → GH as EF → G and EF → H
  – ACDF → EG as ACDF → E and ACDF → G

• Then, consider ACDF → G
  – Implied by A → B, ABCD → E, EF → G (Why?)
  – Delete it!

• Then, consider ABCD → E
  – Replace by ACD → E since A → B

• We have arrived at the following minimal cover:
  – A → B, ACD → E, EF → G and EF → H
Minimal Cover Algorithm

- **Put the FDs in a Standard Form:** Obtain a collection $G$ of equivalent FDs with a single attribute on the right side (using the decomposition axiom).
- **Minimize the Left Side of Each FD:** For each FD in $G$, check each attribute in the left side to see if it can be deleted while preserving equivalence to $F^+$ (i.e., delete extraneous attributes).
- **Delete Redundant FDs:** Check each remaining FD in $G$ to see if it can be deleted while preserving equivalence to $F^+$.
- **Note 1:** the order in which FDs are considered is important – different orders may produce different minimal covers.
- **Note 2 (Important!):** it is essential to perform the second and third steps of the algorithm in correct order!
Note on Minimal Cover Algorithm

- Consider ABCD → E, E → D, A → B, and AC → D.
- None of these FDs redundant.
- ABCD → E can be replaced by AC → E.
- If we stop here, the resulting set is not a minimal cover!
  - AC → E and E → D imply AC → D!
  - Hence, left side minimization must be performed before checking for redundant FDs!
Practice Questions

• Let $F = \{ \text{ABH} \rightarrow \text{CDEFG}, \text{A} \rightarrow \text{D}, \text{C} \rightarrow \text{E}, \text{BGH} \rightarrow \text{F}, \text{F} \rightarrow \text{AD}, \text{E} \rightarrow \text{F}, \text{BH} \rightarrow \text{E} \}$
  – Find the attribute closure of
    • C
    • B
    • AG
  – Find the minimal cover of F
  – Find a BCNF decomposition of ABCDEFGH
    • Is it dependency-preserving?
Dependency-Preserving Decomposition into 3NF

- Let \( R \) be a relation with a set \( F \) of FDs that is a minimal cover and \( R_1, ..., R_n \) be a lossless-join decomposition of \( R \) with each \( R_i, 1 \leq i \leq n \) in 3NF and let \( F_i \) denote the projection of \( F \) onto the attributes of \( R_i \).
  - Identify the set \( N \) of dependencies in \( F \) that is not included in the closure of the union of \( F_i \)s.
  - For each FD \( X \rightarrow A \) in \( N \), create a relation schema \( XA \) and add it to the decomposition of \( R \).

- The resulting schema is 3NF.
  - Proof: Each \( R_i \) is already in 3NF. Since \( X \rightarrow A \) is in the minimal cover, \( Y \rightarrow A \) for any strict subset \( Y \) of \( X \) does not hold and hence \( X \) is a key for \( XA \). Also, any other dependency in \( XA \) must have the right side that involves only attributes in \( X \) since \( A \) is a single attribute (required of all dependencies in the minimal cover!). 3NF allows such dependencies since \( X \) is a key for \( XA \)!
Dependency-Preserving Decomposition into 3NF (cont.)

• Optimization: if the set $N$ contains several FDs with the same left side, $X \rightarrow A_1, \ldots, X \rightarrow A_n$, replace these with $X \rightarrow A_1 \ldots A_n$ and create a single relation $XA_1 \ldots A_n$.

• Example: Contracts relation with attributes $CSJDJPQV$ and FDs $JP \rightarrow C$, $SD \rightarrow P$, and $J \rightarrow S$.
  - As before, decompose into 3NF schema (actually, BCNF) $SDP$, $JS$, and $CJDQV$.
  - Since $JP \rightarrow C$ is not preserved, add $CJP$ to the decomposition (the schema is still BCNF, but, in general, this may not always be so). We trade redundancy for dependency preservation.
3NF Synthesis

- Alternative approach: take a minimal cover $F$ for the FDs that hold over the original relation $R$ and add a relation schema $XA$ to the decomposition of $R$ for each FD $X \rightarrow A$ in $F$.
  - If the resulting collection of schemas is not lossless-join, add a schema with just the attributes that appear in some key.
- A polynomial time algorithm! (Surprising since checking whether a particular schema is in 3NF is NP-complete!)
- Example: $ABC$ with FDs $F = \{ A \rightarrow B, C \rightarrow B \}$. The first step gives two relations $AB$ and $BC$. Need to add $AC$ since $AB \cap BC = B$ and $B$ is not a key for either $AB$ or $BC$ (Why?).
3NF Synthesis (cont.)

• Contracts relation CSJDPQV
  – FDs: C → CSJDPQV, JP → C, SD → P, and J → S

• Obtain minimal cover
  – F = { C → J, C → D, C → Q, C → V, JP → C, SD → P, J → S }

• 3NF schema: CJ, CD, CQ, CV, CJP, SDP, JS.

• Optimize: CDJPQV, SDP, JS. Different schema than the one obtained with the previous algorithm!
Refining an ER Diagram

- Example 1: Contracts relation (contract_id, supplier_id, dept_id, part_id, qty) CSDPQ with an FD DS → P decomposes into CQSD and SDP.
  - ER Design cannot capture this FD and it is unlikely that CQSD and SDP entities are defined in the ER modeling step – they have no intuitive meaning.

- Example 2: 1st diagram translated –
  Workers(S,N,L,D,S),
  Departments(D,M,B)
  - Lots associated with workers.

- Suppose all workers in a dept are assigned the same lot: D → L

- Redundancy; fixed by:
  Workers2(S,N,D,S), Dept_Lots(D,L)

- Can fine-tune this:
  Workers2(S,N,D,S),
  Departments(D,M,B,L)
Summary of Schema Refinement

• If a relation is in BCNF, it is free of redundancies that can be detected using FDs. Thus, trying to ensure that all relations are in BCNF is a good heuristic.

• If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
  – Must consider whether all FDs are preserved. If a lossless-join, dependency preserving decomposition into BCNF is not possible (or unsuitable, given typical queries), should consider decomposition into 3NF.
  – Decompositions should be carried out and/or re-examined while keeping *performance requirements* in mind.
Review Questions

• Does the relation ABCDEF with the functional dependencies \{ A \rightarrow ABCDEF, EF \rightarrow A \} need to be decomposed into BCNF? 3NF?
  – What if we add B \rightarrow D?
  – Provide decompositions if needed.

• Is the decomposition of the above relation into ABCD and AEF lossless join? What about ABCDE and EF?
  – Add B \rightarrow D, C \rightarrow EF. Is ABCD and AEF dependency preserving? What about ABCEF and BD?
Review Questions (cont.)

• Consider the following relation schema
  Lots(property_id, county_name, lot#, area, price, tax_rate)
  abbreviated to PCLART
• Let $F = \{ P \rightarrow CLART, CL \rightarrow PART, C \rightarrow T, A \rightarrow R \}$. Note that $F^+ = F$.
• Is Lots BCNF? Decompose if needed.
• Is it (or the decomposed relations) dependency preserving?
• Add $A \rightarrow C$ to $F$ and reconsider the above questions (if needed, decompose into 3NF).