Announcements

- HW and exams:
  - HW #3 in progress (shorter than HW #1, #2)
  - Midterm Exam 1 in one week (at this time)
- Today’s plan:
  - Relational DB design theory, Episode 2
  - This week’s evening discussion session plan is
    - Relational DB design quiz/discussion
    - Q&A about our given HW #2 solution
    - (All at your scheduled time and place ☀️)
Some Terms and Definitions

- If X is part of a (candidate) key, we will say that X is a prime attribute.
- If X (an attribute set) contains a candidate key, we will say that X is a superkey.
- X → Y can be pronounced as “X determines Y”, or “Y is functionally dependent on X”.
- Some types of dependencies (on a key):
  - Trivial: XY → X
  - Partial: XY is a key, X → Z
  - Transitive: X → Y, Y → Z, Y is non-prime, X → Z
First Normal Form (1NF)

- Rel’n R is in 1NF if all of its attributes are atomic.
  - No set-valued attributes! (1NF = “flat” 😊)
  - Usually goes w/o saying for relational model (but not for NoSQL systems, as we’ll see at the end of the quarter 😊).
  - Ex:

<table>
<thead>
<tr>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interlake</td>
<td>blue, red</td>
</tr>
<tr>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>Marine</td>
<td>red</td>
</tr>
</tbody>
</table>

Second Normal Form (2NF)

- Rel’n R is in 2NF if it is in 1NF and no non-prime attribute is partially dependent on a candidate key of R.
- Ex: Supplies(sno, sname, saddr, pno, pname, pcolor)
  
  where: sno → sname, sno → saddr, pno → pname, pno → pcolor

Q1: What are the candidate keys for Supplies?
Q2: What are the prime attributes for Supplies?
Q3: Why is Supplies not in 2NF?
Q4: What’s the fix?

Supplier(sno, sname, saddr)
Part(pno, pname, pcolor)
Supply(sno, pno)

Must not forget this! (Else “lossy join”!!)
Third Normal Form (3NF)

- Rel’n R is in 3NF if it is in 2NF and it has no transitive dependencies to non-prime attributes.

- Ex: Workers(eno, ename, esal, dno, dname, dfloor)
  
  where: eno → ename, eno → esal, eno → dno, dno → dname, dno → dfloor

Q1: What are the candidate keys for Workers?
Q2: What are the prime attributes for Workers?
Q3: Why is Workers not in 3NF?
Q4: What’s the fix?

Emp(eno, ename, esal, dno)
Dept(dno, dname, dfloor)

Note: A lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations is always possible.

Boyce-Codd Normal Form (BCNF)

- Rel’n R with FDs F is in BCNF if, for all X → A in F+  
  - A ∈ X (trivial FD), or else  
  - X is a superkey (i.e., 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! (i.e., key → attr)
  - Everything depends on “the key, the whole key, and nothing but the key” (so help me Codd 😊)

Don’t forget this! (Else “lossy join”!!)
Boyce-Codd Normal Form (Cont’d.)

- **Ex:** Supply2(sno, sname, pno)
  where: sno \(\rightarrow\) sname, sname \(\rightarrow\) sno
  
  Q1: What are the candidate keys for Supply2?
  Q2: What are the prime attributes for Supply2?
  Q3: Is Supply2 in 3NF?
  Q4: Why is Supply2 not in BCNF?
  Q5: What’s the fix?
  
  Supplier2(sno, sname)
  Supplies2(sno, pno)

  **Note:** A lossless-join, dependency-preserving decomposition of 
  R into a collection of BCNF relations is **NOT always possible.**

3NF Revisited (Alternative Def’n)

- Rel’n R with FDs F is in 3NF if, for all X \(\rightarrow\) A in F+
  - A \(\subseteq\) X (trivial FD), or else
  - X is a superkey (i.e., contains a key) for R, or else
  - A is part of some key for R (prime attribute).

- If R is in BCNF, clearly it is also in 3NF.
- If R is in 3NF, some redundancy is possible. 3NF is a compromise to use when BCNF isn’t achievable (e.g., no “good” decomp, or performance considerations).
  - **Important:** A lossless-join, dependency-preserving 
  decomposition of R into a collection of 3NF relations is always possible.
Reminder:

- Problems due to $R \rightarrow W$:
  - **Update anomaly**: Can we change $W$ in just the 1st tuple of 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!

How about two smaller tables?

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Name</th>
<th>Lot</th>
<th>Rating</th>
<th>Hourly Wage</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>40</td>
</tr>
</tbody>
</table>

Decomposition of a Relation Scheme

- Suppose a relation $R$ contains attributes $A_1 \ldots A_n$. A **decomposition** of $R$ consists of replacing $R$ by two or more relations such that:
  - Each new relation contains a subset of the attributes of $R$ (and no attributes that did not appear in $R$), and
  - Every attribute of $R$ appears as an attribute of at least one of the new relations.

- Intuitively, decomposing $R$ means we will store instances of the relations from the decomposition **instead** of instances of $R$.

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

- Decompositions should be used only when needed.
  - Suppose SNLRWH has 2 FDs: \( S \rightarrow SNLRWH \) and \( R \rightarrow W \)
  - Second FD violates 3NF (\( W \) values repeatedly associated with \( R \) values). Easiest fix creates a relation RW to store the associations, then remove \( W \) from the main schema:
    - I.e.: Decompose SNLRWH into SNLRH and RW.

- The information to be stored consists of SNLRWH tuples. So if we just store the projections of these tuples onto SNLRH and RW, are there potential problems that we should be aware of?

Decompositions: Possible Problems

- There are three potential problems to consider:
  1. Some queries become more expensive.
     - E.g., how much did sailor Joe earn? (\( W \times H \) now requires a join)
  2. Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation! (If “lossy”...)
     - Fortunately, not a problem in the SNLRWH example!
  3. Checking some dependencies may require joining the instances of the decomposed relations.
     - Fortunately, also not in the SNLRWH example.

- **Tradeoff:** Consider these issues vs. the redundancy.
Lossless Join Decompositions

- Decomposition of \( R \) into \( X \) and \( Y \) is \textit{lossless-join} w.r.t. 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, then the decomposition is called lossless-join.
  - Must ensure that \( X \) and \( Y \) overlap, and that the overlap contains a key for one of the two relations.
- Definition extends to decomposition into 3 or more relations as you would expect.
- \textit{Decompositions must be lossless!} (Avoids Problem (2).)

Dependency Preserving Decomposition

- Consider \( CSJDQV \), \( C \) is key, \( JP \Rightarrow C \) and \( SD \Rightarrow P \).
  - BCNF decomposition: \( CSJDQV \) and \( SDP \)
  - Problem: Checking \( JP \Rightarrow C \) requires a join!
- Dependency preserving decomposition (intuitive):
  - If \( R \) is decomposed into \( X, Y \), and \( Z \), and we enforce the FDs that hold on \( X, Y, \) and on \( Z \), then all FDs that were given to hold on \( R \) must also hold. (Avoids Problem (3).)
- \textit{Projection of set of FDs }\( F \): If \( R \) is decomposed into \( X, \ldots \), projection of \( F \) into \( X \) (denoted \( F_X \)) is the set of FDs \( U \Rightarrow V \) in \( F^+ \) (\textit{closure of }\( F \)) where \( U, V \) are both in \( X \).
**Dependency Preserving Decomp. (Cont’d.)**

- The decomposition of R into X and Y is **dependency preserving** if \((F_X \cup F_Y)^+ = F^+\)
  - 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, they **imply** all dependencies in \(F^+\).

- Important to consider \(F^+\), not \(F\), in this definition:
  - R(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:
  - R(ABC), A\(\rightarrow\)B, if decomposed into AB and BC. (Q: Key?)

- And vice-versa! (So we need to check for both.)
  - Must make sure some relation contains a **key** for R (!!)

---

**Decomposition into BCNF**

- Consider relation R with FDs F. **If** X \(\rightarrow\) Y violates **BCNF**, decompose R into R–Y and XY. **(R-Y has X still!)**
  - Repeated application of this idea will yield a collection of relations that are **BCNF**, a lossless join decomposition, and guaranteed to terminate. **(Didn’t say dependency preserving...)**
  - E.g., CSJDPQV, C\(\rightarrow\)CSJDPQV, 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 violations of **BCNF**. The order in which we “deal with them” can lead to different sets of relations!
**BCNF and Dependency Preservation**

- In general, there may not be a dependency preserving decomposition into BCNF.
  - E.g., R(CSZ) with CS \(\rightarrow\) Z, Z \(\rightarrow\) C.
  - Can’t decompose preserving 1st FD, not in BCNF.
- Decomposition of CSJDPQV into SDP, JS and CJDQV is *not* dependency preserving (w.r.t. the FDs JP \(\rightarrow\) C, SD \(\rightarrow\) P and J \(\rightarrow\) S).
  - However, it *is* a lossless join decomposition.
  - In this case, adding JPC to the collection of relations would give us a dependency preserving decomposition.
    - But: JPC tuples stored only for checking FD! (*Redundancy!*)

**Decomposition into 3NF**

- Obviously, the algorithm for lossless join decompp into BCNF can also be used to obtain a lossless join decompp into 3NF (and might possibly 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? (Which of course implies JP \(\rightarrow\) C.)
- The fix: Instead of using 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 $F = \text{closure of } G$, i.e., $F^+ = G^+$.
  - Right hand side (RHS) of each FD in $G$ is a single attribute.
  - If we modify $G$ by deleting an FD or by deleting attributes from a LHS of a FD in $G$, the closure changes. (So we can’t!)

- Intuitively, every FD in $G$ is needed, and $G$ is “as small as possible” to have the same closure as $F$.

- E.g., $A \rightarrow B, ABCD \rightarrow E, EF \rightarrow GH, ACDF \rightarrow EG$ has the following minimal cover:
  - $A \rightarrow B, ACD \rightarrow E, EF \rightarrow G$ and $EF \rightarrow H$

- M.C. $\rightarrow$ lossless-join, dep. pres. 3NF decomp! (See book!)

**Obtaining that 3NF Decomposition**

- Compute the minimal cover $G$, which is sometimes denoted $F^-$ (see book for how)

- Search for dependencies in $F^-$ with the same attribute set on the left hand side, $\alpha$:
  - $\alpha \rightarrow Y_1, \alpha \rightarrow Y_2, ... \alpha \rightarrow Y_k$
  - Construct one relation as $(\alpha, Y_1, Y_2, ...Y_k)$
  - Repeat this process for all the FDs’ $\alpha$’s
  - Construct a relation with any leftover attributes
  - If none of the relations contains a candidate key for the original relation $R$, add one more relation containing the attributes of a candidate key for $R$. (Q: Why?)
On Refining an ER Based Design

- 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)
  WorkersLots(D,L)
  Departments(D,M,B)
- Can further fine-tune this:
  Workers2(S,N,D,S)
  Departments(D,M,B,L)

Notice: Lot wasn’t really a “Worker attribute”!

DB Design Theory Summary

- 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 goal.
- If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
  - Are all FDs preserved? If a lossless-join, dependency-preserving decomposition into BCNF is not possible (or unsuitable for typical queries), consider 3NF instead.
  - Decompositions should be carried out while also keeping performance requirements in mind.