Announcements

- HW#3 is underway...!
  - Today we will aim to finish the relevant material!
  - Because we are a little behind, you’ll see that we are heavily discounting the late penalties for this one. *(Note: Don’t expect similar light penalties again! 😊)*
- Today’s plan:
  - Relational DB design theory (II)
  - Focus will be on normal forms and normalization!
  - (Still not the most exciting part of CS122a... 😊)
Normal Forms

<table>
<thead>
<tr>
<th>All “relations”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1NF</td>
</tr>
<tr>
<td>2NF</td>
</tr>
<tr>
<td>3NF</td>
</tr>
<tr>
<td>BCNF</td>
</tr>
</tbody>
</table>

Some Terms and Definitions (Review)

- 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.
- $X \rightarrow Y$ can be pronounced as “$X$ determines $Y$”, or “$Y$ is functionally dependent on $X$”.

Some types of dependencies (on a key):
- **Trivial**: $XY \rightarrow X$
- **Partial**: $XY$ is a key, $X \rightarrow Z$
- **Transitive**: $X \rightarrow Y$, $Y \rightarrow Z$, $Y$ is non-prime, $X \rightarrow Z$
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)
```

- A1: (sno, pno)
- A2: sno, pno
- A3: Each of its four FDs violates 2NF!

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.

Don’t forget this! (Else “lossy join” !!)

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 😍)
Boyce-Codd Normal Form (Cont’d.)

Ex: Supply2(sno, sname, pno)
Given FDs: sno → sname, sname → 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 → A in F+
• A ∈ 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 (i.e., it’s a 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?

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 RW and SNLRH.
Example Decomposition

- Decompositions should be used only when needed.
  - Suppose \text{SNLRWH} has 2 FDs: \text{S} \rightarrow \text{SNLRWH} and \text{R} \rightarrow \text{W}
  - Second FD violates 3NF (W values repeatedly associated with R values). Easiest fix creates a relation RW to store the associations, then removes 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*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.

- \textit{Tradeoff}: Consider these issues vs. the redundancy.
**Lossless Join Decompositions**

- Decomposition of R into X and Y is **lossless-join** w.r.t. a set of FDs F if, for every instance r that satisfies F:
  - \( \pi_X (r) \bowtie \pi_Y (r) = r \) (Note: relational algebra)
- 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.
- **Decompositions must be lossless!** (Avoids Problem (2).)

**Dependency Preserving Decomposition**

- Consider CSJDPOQ, 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).)
- **Projection of set of FDs F**: If R is decomposed into X, ... projection of F into X (denoted \( F_X \)) is the set of FDs \( U \rightarrow V \) in \( F^+ \) (closure of F) where U, V are both in X.