Announcements

- HW stuff:
  - HW #2 moves into “late mode” today
  - HW #3 is available as of now on the wiki page
  - HW #4 will be AFTER Midterm1 (it’s coming up!)

- Today’s plan:
  - Relational DB design theory, Episode 1
  - Next week’s evening discussion sessions will
    - Start with a relational DB design quiz/discussion
    - Move on to Q&A time about HW #2’s solution
  - Please remember the new attendance restriction! (😊)
Relational Database Design

- There are two aspects to this problem:
  - Logical schema design: We just saw one approach, namely, doing E-R modeling followed by an E-R → relational schema translation step
  - Physical schema design: Later, once we learn about indexes, when should we utilize them?
- We will look at both problem aspects this term, starting first with relational schema design
  - Our power tools will be functional dependencies (FDs) and normalization theory
  - Note: FDs also play an important role in other contexts as well, e.g., in SQL query optimization

So, Given a Relational Schema...

- How do I know if my relational schema is a “good” logical database design or not?
  - What might make it “not good”?
  - How can I fix it, if indeed it’s “not good”?
  - How “good” is it, after I’ve fixed it?
- Note that your relational schema might have come from one of several places
  - You started from an E-R model (but maybe that model was “wrong” in some way?)
  - You went straight to relational in the first place
  - It’s not your schema – you inherited it! 😊
**Ex: Wisconsin Sailing Club**

Proposed schema design #1:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>date</th>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td>10/10/98</td>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td>10/10/98</td>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td>10/8/98</td>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td>10/7/98</td>
<td>104</td>
<td>Marine</td>
<td>red</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td>11/10/98</td>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td>11/6/98</td>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td>11/12/98</td>
<td>104</td>
<td>Marine</td>
<td>red</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Q:** Do you think this is a “good” design? (Why or why not?)

Proposed schema design #2:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td>104</td>
<td>Marine</td>
<td>red</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td>104</td>
<td>Marine</td>
<td>red</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Q:** What about this design?  
- Is #2 “better than #1...? Explain!  
- Is it a “best” design?  
- How can we go from design #1 to this one?
**Ex: Wisconsin Sailing Club**

Proposed schema design #3:

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/98</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>10/10/98</td>
</tr>
<tr>
<td>22</td>
<td>103</td>
<td>10/8/98</td>
</tr>
<tr>
<td>22</td>
<td>104</td>
<td>10/7/98</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>11/10/98</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11/6/98</td>
</tr>
<tr>
<td>31</td>
<td>104</td>
<td>11/12/98</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Q: What about *this* design?
- Is #3 “better” or “worse” than #2...?
- What sort of tradeoffs do you see between the two?

---

**The Evils of Redundancy**

(or: The Evils of Redundancy)

- **Redundancy** is at the root of several problems associated with relational schemas:
  - Redundant storage
  - Insert/delete/update anomalies
  
  Good rule to follow: **“One fact, one place!”**

- **Functional dependencies** can help in identifying problem schemas and suggesting refinements.

- Main refinement technique: *decomposition*, e.g., replace R(ABCD) with R1(AB) + R2(BCD).

- Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - Does the decomposition cause any problems?
**Functional Dependencies (FDs)**

- A functional dependency \( X \rightarrow Y \) holds over relation \( R \) if, for every allowable instance \( r \) of \( R \):
  - For \( t1 \) and \( t2 \) in \( r \), \( t1.X = t2.X \) implies \( t1.Y = t2.Y \)
  - I.e., given two tuples in \( r \), if the \( X \) values agree, then their \( Y \) values must also agree. (\( X \) and \( Y \) can be sets of attributes.)

- An FD is a statement about all allowable relations.
  - Identified based on application semantics (similar to E-R).
  - Given some instance \( r1 \) of \( R \), we can check to see if it violates some FD \( f \), but we cannot tell if \( f \) holds over \( R \)!

- Saying \( K \) is a candidate key for \( R \) means that \( K \rightarrow R \)
  - Note: \( K \rightarrow R \) alone does not require \( K \) to be minimal! If \( K \) is minimal, then \( K \) is a candidate key.

---

**Example: Constraints on an Entity Set**

- Suppose you’re given a relation called **HourlyEmps**:
  - **HourlyEmps** (\( ssn \), name, lot, rating, hrly_wages, hrs_worked)

- **Notation**: We will denote this relation schema by simply 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., HourlyEmps for SNLRWH).

- Suppose we also have some FDs on **HourlyEmps**:
  - **ssn** is the key: \( S \rightarrow SNLRWH \)
  - **rating** determines **hrly_wages**: \( R \rightarrow W \)
Example (Cont’d.)

- 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 proper hourly wage for his or her rating?
  - **Deletion anomaly**: If we delete all employees with rating 5, we lose the stored information about the wage for rating 5!

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>H</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>

How about two smaller tables?

Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - $ssn \rightarrow did$, $did \rightarrow lot$ implies $ssn \rightarrow lot$
- 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 that are implied by $F$.
- Armstrong’s Axioms ($X$, $Y$, $Z$ are sets of attributes):
  - **Reflexivity**: If $X \subseteq Y$, 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’d.)

(Recall: “two matching X’s always have the same Y”)

- A few additional rules (which follow from AA):
  - Union: If \( X \rightarrow Y \) and \( X \rightarrow Z \), then \( X \rightarrow YZ \)
  - Decomposition: If \( X \rightarrow YZ \), then \( X \rightarrow Y \) and \( X \rightarrow Z \)

- Example: Contracts(cid,sid,pjid,did,pid,qty,value), and:
  - The contract id is the key: \( C \rightarrow CSJDPQV \)
  - A project purchases each part using single contract: \( JP \rightarrow C \)
  - A 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 \( SDJ \rightarrow JP \) \(\text{(New candidate keys...!)}\)
  - \( SDJ \rightarrow JP \), \( JP \rightarrow CSJDPQV \) imply \( SDJ \rightarrow CSJDPQV \)

Reasoning About FDs (Examples)

Let’s consider \( R(ABCDE) \), \( F = \{A \rightarrow B, B \rightarrow C, CD \rightarrow E\} \)

- Let’s work our way towards inferring \( F^+ \) ...
  - (a) \( A \rightarrow B \) (b) \( B \rightarrow C \) (c) \( CD \rightarrow E \) \(\text{(given)}\)
  - (d) \( A \rightarrow C \) \(\text{(a, b, and transitivity)}\)
  - (e) \( BD \rightarrow CD \) \(\text{(b and augmentation)}\)
  - (f) \( BD \rightarrow E \) \(\text{(e and transitivity)}\)
  - (g) \( AD \rightarrow CD \) \(\text{(d and augmentation)}\)
  - (h) \( AD \rightarrow E \) \(\text{(g and transitivity)}\)
  - (i) \( AD \rightarrow C \) \(\text{(g and decomposition)}\)
  - (j) \( AD \rightarrow D \) \(\text{(a and augmentation)}\)
  - (k) \( AD \rightarrow BD \) \(\text{(k and decomposition)}\)
  - (l) \( AD \rightarrow B \) \(\text{(a and reflexivity)}\)
  - (m) \( AD \rightarrow A \) \(\text{(h, i, j, l, m, and union)} \ldots\)
  - (n) \( AD \rightarrow ABCDE \)  

Candidate key!

If an attribute \( X \) is not on the RHS of any initial FD, \( X \) must be part of the key!
Reasoning About FDs (Cont’d.)

- 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.
  - Then check if $Y$ is in $X^+$
- Does $F = \{ A \rightarrow B, \ B \rightarrow C, \ C \rightarrow D \rightarrow E \}$ imply $A \rightarrow E$?
  - I.e.: is $A \rightarrow E$ in the closure $F^+$? Equivalently: Is $E$ in $A^+$?

FDs & Redundancy

- Role of FDs in detecting redundancy:
  - Consider a relation $R$ with 3 attributes, $ABC$.
    - If no non-trivial FDs hold: There is no redundancy here then. (Think about this – in fact, think hard...!)
    - Given $A \rightarrow B$: Several tuples could have the same $A$ value – and if so, then they’ll all have the same $B$ value as well! (Thus if $A$ is repeated for some reason, it will always have the same $B$ “tagging along for the ride”.)
Normal Forms

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

We will define various normal form (BCNF, 3NF etc.) based on the nature of FDs that hold.

Depending upon the normal form a relation is in, it has different level of redundancy

- E.g., a BCNF relation has NO redundancy - will be clear soon!

Checking for which normal form a relation is in will help us decide whether to decompose the relation

- E.g., no point decomposing a BCNF relation!