Announcements

- HW1 deadline:
  - Due on 09/21 (Thurs), 11:55 pm, no late days
- Project proposal deadline:
  - Preliminary idea and team members due by 09/18 (Mon) by email to the instructor
  - Proposal due on sakai by 09/25 (Mon), 11:55 pm

Today

- Finish RC from Lecture 4
  - DRC
  - More example
- Normalization

DRC: example

\[
\begin{align*}
\text{Sailors} & : (\text{sid}, \text{sname}, \text{rating}, \text{age}) \\
\text{Boats} & : (\text{bid}, \text{bname}, \text{color}) \\
\text{Reserves} & : (\text{sid}, \text{bid}, \text{day})
\end{align*}
\]

• Find the name and age of all sailors with a rating above 7

TRC:

\[
\{ P \mid \exists S \in \text{Sailors} (S.\text{rating} > 7 \land P.\text{name} = S.\text{name} \land P.\text{age} = S.\text{age}) \}
\]

DRC:

\[
\{<N, A> \mid \exists I, N, T, A \in \text{Sailors} \land T > 7\}
\]

• Variables are now domain variables
• We will use use TRC
  - both are equivalent

More Examples: RC

- The famous “Drinker-Beer-Bar” example!

Drinker Category 1

\[
\begin{align*}
\text{Likes} & : (\text{drinker, beer}) \\
\text{Frequents} & : (\text{drinker, bar}) \\
\text{Serves} & : (\text{bar, beer})
\end{align*}
\]

Find drinkers that frequent some bar that serves some beer they like.

UNDERSTAND THE DIFFERENCE IN ANSWERS FOR ALL FOUR DRINKERS

Drinker Category 1

Find drinkers that frequent some bar that serves some beer they like.

\[ Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z) \]

A shortcut for:

\[ \{ x | \exists y \in \text{Frequents} \land \exists z \in \text{Serves} \land \exists w \in \text{Likes} (T.drinker = x.drinker \land T.bar = Z.bar \land W.beer = \ldots .) \} \]

The difference is that in the first one, one variable = one attribute 
in the second one, one variable = one tuple (Tuple RC)
Both are equivalent and feel free to use the one that is convenient to you

Drinker Category 2/3/4

Find drinkers that frequent some bar that serves some beer they like.

\[ Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z) \]

Find drinkers that frequent only bars that serves some beer they like.

\[ Q(x) = \]

Find drinkers that frequent some bar that serves only beers they like.

\[ Q(x) = \]

Find drinkers that frequent only bars that serves only beer they like.

\[ Q(x) = \]

Why should we care about RC

- RC is declarative, like SQL, and unlike RA (which is operational)
- Gives foundation of database queries in first-order logic
  - you cannot express all aggregates in RC, e.g. cardinality of a relation or sum (possible in extended RA and SQL)
  - still can express conditions like "at least two tuples" (or any constant)
- RC expression may be much simpler than SQL queries
  - and easier to check for correctness than SQL
  - power to use \( \land \) and \( \lor \) ->
  - then you can systematically go to a "correct" SQL query

From RC to SQL

Query: Find drinkers that like some beer (so much) that they frequent all bars that serve it

\[ Q(x) = \exists y. \exists z. (\text{Likes}(x, y) \land \text{Serves}(z, y) \Rightarrow \text{Frequents}(x, z)) \]

Step 1: Replace \( \land \) with \( \lor \) using de Morgan’s Laws

\[ \neg Q(x) = \exists y. \exists z. (\text{Likes}(x, y) \lor \text{Serves}(z, y) \land \neg \text{Frequents}(x, z)) \]

Step 2: Translate into SQL

```
SELECT DISTINCT L.drinker
FROM Likes L
WHERE not exists (SELECT S.bar
FROM Serves S
WHERE L.beer=S.beer
AND not exists (SELECT * FROM Frequents F
WHERE F.drinker=L.drinker
AND F.bar=S.bar))
```
Summary

- You learnt three query languages for the Relational DB model
  - SQL
  - RA
  - RC

- All have their own purposes

- You should be able to write a query in all three languages and convert from one to another
  - However, you have to be careful, not all “valid” expressions in one may be expressed in another
  - \( \{ s \mid s \notin S \} \) – infinitely many tuples – an "unsafe" query
  - More when we do “Datalog”, also see Ch. 4.4 in [RG]

Where are we now?

We learnt

- Relational Model and Query Languages
  - SQL, RA, RC
  - Postgres (DBMS)
  - XML (overview)

- HW1

Next

- Database Normalization
  - (for good schema design)

Reading Material

- Database normalization
  - [RG] Chapter 19.1 to 19.5, 19.6.1, 19.8 (overview)
  - [GUW] Chapter 3

Acknowledgement:

- The following slides have been created adapting the instructor material of the [RG] book provided by the authors
  - Dr. Ramakrishnan and Dr. Gehrke.
- Some slides have been adapted from slides by Prof. Jun Yang.

What will we learn?

- What goes wrong if we have redundant info in a database?
- Why and how should you refine a schema?
- Functional Dependencies – a new kind of integrity constraints (IC)
- Normal Forms
- How to obtain those normal forms

Example

The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>lot</th>
<th>rating</th>
<th>hourly-wage</th>
<th>hours-worked</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

- key = SSN
Decompositions should be used judiciously

1. Do we need to decompose a relation?
   - Several normal forms
   - If a relation is not in one of them, may need to decompose further

2. What are the problems with decomposition?
Functional Dependencies (FDs)

- An FD is a statement about all allowable relations
  - Must be identified based on semantics of application
  - Given some allowable instance \( r_1 \) 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 \( K \rightarrow R \)
  - denoting \( R = \) all attributes of \( R \) too
  - However, \( S \rightarrow R \) does not require \( S \) to be minimal
    - e.g. \( S \) can be a superkey

Example

- Consider relation obtained from Hourly_Emps:
  - Hourly_Emps (ssn, name, lot, rating, hourly_wage, hours_worked)
- Notation: We will denote a relation schema by listing the attributes: SNLRWH
  - Basically the set of attributes (S,N,L,R,W,H)
  - here first letter of each attribute

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d2</td>
</tr>
<tr>
<td>a1</td>
<td>b2</td>
<td>c2</td>
<td>d1</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>c3</td>
<td>d1</td>
</tr>
</tbody>
</table>

  Possible FDs:
  - ssn is the key: \( S \rightarrow SNLRWH \)
  - rating determines hourly_wages: \( R \rightarrow W \)

Armstrong’s Axioms

- \( X, Y, Z \) are sets of attributes
  - Reflexivity: If \( X \supseteq Y \), then \( X \rightarrow Y \)
  - Augmentation: If \( X \rightarrow Y \), then \( XZ \rightarrow YZ \) for any \( Z \)
  - Transitivity: If \( X \rightarrow Y \) and \( Y \rightarrow Z \), then \( X \rightarrow Z \)

  Apply these rules on \( AB \rightarrow C \) and check

Additional Rules

- Follow from Armstrong’s Axioms
  - Union: If \( X \rightarrow Y \) and \( X \rightarrow Z \), then \( X \rightarrow YZ \)
  - Decomposition: If \( X \rightarrowYZ \), then \( X \rightarrow Y \) and \( X \rightarrow Z \)

Closure of a set of FDs

- Given some FDs, we can usually infer additional FDs:
  - SSN → DEPT, and DEPT → LOT implies SSN → LOT
  - An FD \( f \) is implied by a set of FDs \( F \) if \( f \) holds whenever all FDs in \( F \) hold.

  \( F^* \) = closure of \( F \) is the set of all FDs that are implied by \( F \)
To check if an FD belongs to a closure

- Computing the closure of a set of FDs can be expensive
  - Size of closure can be exponential in #attributes

- Typically, we just want to check if a given FD $X \rightarrow Y$ is in the closure of a set of FDs $F$

- No need to compute $F^+$
  1. Compute attribute closure of $X$ (denoted $X^+$) wrt $F$:
     - Set of all attributes $A$ such that $X \rightarrow A$ is in $F^+$
  2. Check if $Y$ is in $X^+$

Computing Attribute Closure

Algorithm:

- $\text{closure} = X$
- Repeat until no change
  - if there is an FD $U \rightarrow V$ in $F$ such that $U \subseteq \text{closure}$, then $\text{closure} = \text{closure} \cup V$

- 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 $A^+$?

Normal Forms

- Question: given a schema, how to decide whether any schema refinement is needed at all?

- If a relation is in a certain normal forms, it is known that certain kinds of problems are avoided/minimized

- Helps us decide whether decomposing the relation is something we want to do

FDs play a role in detecting redundancy

Example

- Consider a relation $R$ with 3 attributes, ABC
  - No FDs hold: There is no redundancy here – no decomposition needed
  - Given $A \rightarrow B$: Several tuples could have the same A value, and if so, they’ll all have the same B value – redundancy – decomposition may be needed if A is not a key

- Intuitive idea:
  - if there is any non-key dependency, e.g. $A \rightarrow B$, decompose!

Boyce-Codd Normal Form (BCNF)

- Relation $R$ with FDs $F$ is in BCNF if, for all $X \rightarrow A$ in $F$
  - $A \in X$ (called a trivial FD), or
  - $X$ contains a key for $R$
    - i.e. $X$ is a superkey

Definitions next
### Third Normal Form (3NF)

- Relation R with FDs $F$ is in 3NF if, for all $X \rightarrow A$ in $F$:
  - $A \in X$ (called 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 in 3NF
  - every attribute is part of some superkey ($= \{ \text{set of all attributes} \}$)

- If $R$ is in BCNF, obviously in 3NF
- If $R$ is in 3NF, some redundancy is possible
  - when $X \rightarrow A$ and $A$ is part of a key (not allowed in BCNF)

### Decomposition of a Relation Schema

- Consider relation $R$ contains attributes $A_1 \ldots A_n$
- A decomposition of $R$ consists of replacing $R$ by two or more relations such that “no attribute is lost” and “no new attribute appears”, i.e.
  - Each new relation schema contains a subset of the attributes of $R$
  - Every attribute of $R$ appears as an attribute of one of the new relations
  - E.g., Can decompose $\text{SNLRWH}$ into $\text{SNLRH}$ and $\text{RW}$

- What are the potential problems with an arbitrary decomposition?

### 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 $F$: $\pi_X(r) \bowtie \pi_Y(r) = r$

### Algorithm: Decomposition into BCNF

- Input: relation $R$ with FDs $F$
- If $X \rightarrow Y$ violates BCNF, decompose $R$ into $R - Y$ and $X - Y$
  - Repeat until all new relations are in BCNF w.r.t. the given $F$

- NOTE: Need to consider all possible FDs that can be inferred from the current set of FDs (closure), not only the given ones!

- Gives a collection of relations that are
  - in BCNF
  - lossless join decomposition
- and guaranteed to terminate

### Decomposition into BCNF (example)

- $\text{CSIDPQV}$, key $C$, $F = \{ JP \rightarrow C, SD \rightarrow P, J \rightarrow S \}$
  - To deal with $SD \rightarrow P$, decompose into $SDP, CSIDQV$.
  - To deal with $J \rightarrow S$, decompose $CSIDQV$ into $JS$ and $CJDQV$

- Note:
  - several dependencies may cause violation of BCNF
  - The order in which we pick them may lead to very different sets of relations
  - there may be multiple correct decompositions

### BCNF decomposition example

- $\text{UserJoinsGroup} (uid, uname, twitterid, gid, fromDate)$
- $\text{uid} \rightarrow \text{uname}, \text{twitterid}$
- $\text{uid, gid} \rightarrow \text{fromDate}$
Another example

\text{UserJoinsGroup} (\text{uid, uname, twitterid, gid, fromDate})

Recap

- Functional dependencies: a generalization of the key concept
- Non-key functional dependencies: a source of redundancy
- BCNF decomposition: a method for removing redundancies
  - BCNF decomposition is a lossless join decomposition
- BCNF: schema in this normal form has no redundancy due to FD’s

BCNF = no redundancies?

- User (uid, gid, place)
  - A user can belong to multiple groups
  - A user can register places she’s visited
  - Groups and places have nothing to do with other
  - FD’s?
  - BCNF?
  - Redundancies?

Multivalued dependencies

- A multivalued dependency (MVD) has the form
  \( X \rightarrow Y \), where \( X \) and \( Y \) are sets of attributes in a relation \( R \)
- \( X \rightarrow Y \) means that whenever two rows in \( R \) agree on all the attributes of \( X \), then we can swap their \( Y \) components and get two rows that are also in \( R \)

MVD examples

User (uid, gid, place)

- \( \text{uid} \rightarrow \text{gid} \)
- \( \text{uid} \rightarrow \text{place} \)
  - Intuition: given \( \text{uid} \), attributes \( \text{gid} \) and \( \text{place} \) are “independent”
- \( \text{uid}, \text{gid} \rightarrow \text{place} \)
  - Trivial: \( \text{LHS} \cup \text{RHS} = \text{all attributes of R} \)
- \( \text{uid}, \text{gid} \rightarrow \text{uid} \)
  - Trivial: \( \text{LHS} \supseteq \text{RHS} \)

Complete MVD + FD rules

- FD reflexivity, augmentation, and transitivity
- MVD complementation:
  - If \( X \rightarrow Y \), then \( X \rightarrow \text{attrs}(R) – X – Y \)
- MVD augmentation:
  - If \( X \rightarrow Y \) and \( V \subseteq W \), then \( XY \rightarrow YV \)
- MVD transitivity:
  - If \( X \rightarrow Y \) and \( Y \rightarrow Z \), then \( X \rightarrow Z – Y \)
- Replication (FD is MVD):
  - If \( X \rightarrow Y \), then \( X \rightarrow Y \)
- Coalescence:
  - If \( X \rightarrow Y \) and \( W \) is disjoint from \( Y \) such that \( W \rightarrow Z \), then \( X \rightarrow Z \)

Verify these yourself!
An elegant solution: “chase”

* Given a set of FD’s and MVD’s \( \mathcal{D} \), does another dependency \( d \) (FD or MVD) follow from \( \mathcal{D} \)?

* Procedure
  - Start with the premise of \( d \), and treat them as “seed” tuples in a relation
  - Apply the given dependencies in \( \mathcal{D} \) repeatedly
    - If we apply an FD, we infer equality of two symbols
    - If we apply an MVD, we infer more tuples
    - If we infer the conclusion of \( d \), we have a proof
    - Otherwise, if nothing more can be inferred, we have a counterexample

Proof by chase

* In \( R(A, B, C, D) \), does \( A \rightarrow B \) and \( B \rightarrow C \) imply that \( A \rightarrow C \)?

<table>
<thead>
<tr>
<th>Have:</th>
<th>Need:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A \rightarrow B )</td>
<td>( a_1 b_1 c_1 d_1 )</td>
</tr>
<tr>
<td>( A \rightarrow B )</td>
<td>( a_2 b_2 c_2 d_2 )</td>
</tr>
<tr>
<td>( B \rightarrow C )</td>
<td>( b_1 c_2 d_1 )</td>
</tr>
<tr>
<td>( B \rightarrow C )</td>
<td>( b_2 c_1 d_2 )</td>
</tr>
</tbody>
</table>

Another proof by chase

* In \( R(A, B, C, D) \), does \( A \rightarrow B \) and \( B \rightarrow C \) imply that \( A \rightarrow C \)?

<table>
<thead>
<tr>
<th>Have:</th>
<th>Need:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A \rightarrow B )</td>
<td>( a_1 b_1 c_1 d_1 )</td>
</tr>
<tr>
<td>( A \rightarrow B )</td>
<td>( a_2 b_2 c_2 d_2 )</td>
</tr>
<tr>
<td>( B \rightarrow C )</td>
<td>( b_1 c_2 d_1 )</td>
</tr>
<tr>
<td>( B \rightarrow C )</td>
<td>( b_2 c_1 d_2 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Have:</th>
<th>Need:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A \rightarrow B )</td>
<td>( b_1 = b_2 )</td>
</tr>
<tr>
<td>( B \rightarrow C )</td>
<td>( c_1 = c_2 )</td>
</tr>
</tbody>
</table>

Counterexample by chase

* In \( R(A, B, C, D) \), does \( A \rightarrow BC \) and \( CD \rightarrow B \) imply that \( A \rightarrow B \)?

<table>
<thead>
<tr>
<th>Have:</th>
<th>Need:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A \rightarrow BC )</td>
<td>( a_1 b_2 c_2 d_1 )</td>
</tr>
<tr>
<td>( A \rightarrow BC )</td>
<td>( a_1 b_2 c_1 d_2 )</td>
</tr>
<tr>
<td>( CD \rightarrow B )</td>
<td>( b_1 = b_2 )</td>
</tr>
</tbody>
</table>

4NF decomposition algorithm

* A relation \( R \) is in Fourth Normal Form (4NF) if
  - For every non-trivial MVD \( X \rightarrow Y \) in \( R \), \( X \) is a superkey
  - That is, all FD’s and MVD’s follow from “key \( \rightarrow \) other attributes” (i.e., no MVD’s and no FD’s besides key functional dependencies)

* 4NF is stronger than BCNF
  - Because every FD is also a MVD

Counterexample by chase

<table>
<thead>
<tr>
<th>Have:</th>
<th>Need:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A \rightarrow BC )</td>
<td>( a_1 b_2 c_2 d_1 )</td>
</tr>
<tr>
<td>( A \rightarrow BC )</td>
<td>( a_1 b_2 c_1 d_2 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Have:</th>
<th>Need:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A \rightarrow BC )</td>
<td>( b_1 = b_2 )</td>
</tr>
</tbody>
</table>

In general, with both MVD’s and FD’s, chase can generate both new tuples and new equalities

4NF

* A relation \( R \) is in Fourth Normal Form (4NF) if
  - For every non-trivial MVD \( X \rightarrow Y \) in \( R \) where \( X \) is not a superkey
  - Decompose \( R \) into \( R_1 \) and \( R_2 \), where
    - \( R_1 \) has attributes \( X \cup Y \)
    - \( R_2 \) has attributes \( X \cup Z \) (where \( Z \) contains \( R \) attributes not in \( X \) or \( Y \))

* Repeat until all relations are in 4NF

* 4NF is stronger than BCNF
  - Because every FD is also a MVD

4NF decomposition algorithm

* Find a 4NF violation
  - A non-trivial MVD \( X \rightarrow Y \) in \( R \) where \( X \) is not a superkey
  - Decompose \( R \) into \( R_1 \) and \( R_2 \), where
    - \( R_1 \) has attributes \( X \cup Y \)
    - \( R_2 \) has attributes \( X \cup Z \) (where \( Z \) contains \( R \) attributes not in \( X \) or \( Y \))

* Repeat until all relations are in 4NF

* 4NF is stronger than BCNF
  - Because every FD is also a MVD

* Any decomposition on a 4NF violation is lossless
### 4NF decomposition example

<table>
<thead>
<tr>
<th>User (uid, gid, place)</th>
<th>4NF violation: uid ↠ gid</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid: 142, gid: dps, place: Springfield</td>
<td></td>
</tr>
<tr>
<td>uid: 142, gid: dps, place: Australia</td>
<td></td>
</tr>
<tr>
<td>uid: 456, gid: abc, place: Springfield</td>
<td></td>
</tr>
<tr>
<td>uid: 456, gid: abc, place: Morocco</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Member (uid, gid)</th>
<th>4NF</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid: 142, gid: dps</td>
<td></td>
</tr>
<tr>
<td>uid: 456, gid: abc</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Visited (uid, place)</th>
<th>4NF</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid: 142, place: Springfield</td>
<td></td>
</tr>
<tr>
<td>uid: 142, place: Australia</td>
<td></td>
</tr>
<tr>
<td>uid: 456, place: Springfield</td>
<td></td>
</tr>
<tr>
<td>uid: 456, place: Morocco</td>
<td></td>
</tr>
</tbody>
</table>

### Other kinds of dependencies and normal forms

- Dependency preserving decompositions
- Join dependencies
- Inclusion dependencies
- 5NF
- See book if interested (not covered in class)

### Summary

- **Philosophy behind BCNF, 4NF:**
  
  Data should depend on the key, the whole key, and nothing but the key!
  
  — You could have multiple keys though

- **Redundancy is not desired typically**
  
  — not always, mainly due to performance reasons

- **Functional/multivalued dependencies — capture redundancy**

- **Decompositions — eliminate dependencies**

- **Normal forms**
  
  — Guarantees certain non-redundancy
  
  — 3NF, BCNF, and 4NF

- **Lossless join**

- **How to decompose into BCNF, 4NF**

- **Chase**