Inconsistent Databases
Today’s Reading

Summary/overview papers (available online):

   – Acknowledgement: Slides available online of this invited talk
2. Bertossi: Consistent Query Answering in Databases. ACM SIGMOD Record, 2006
   – See this paper for references

Also see

Arenas-Bertossi-Chomicki
Consistent Query Answers in Inconsistent Databases
PODS’99
Why should we care about consistent query answering?
Application 1: Data Warehousing

• Data from several sources, may violate Integrity Constraints (IC)

• Standard: clean data before storing

  1. Need to determine which data is already clean so that unclean data can be removed

  2. Or, mark query answers as consistent or inconsistent

      • information loss can be avoided
Application 2: Data Integration

• Many databases are integrated together to provide a unified view to the user

• Databases may have different ICs
  – Can be satisfied locally but violated globally
  – e.g. different addresses for the same individual in the tax-payer database and voter-registration database

• May not be possible to clean the databases
  – each database is autonomous and independent

• Need to find out which query answers are consistent and which are not
Application 3: Unenforced IC

• Database systems allow IC
• But sometimes they cannot be enforced
  – May be a “legacy data source” that does not support IC
  – IC checking may be too costly
  – DBMS may support only a few IC and not the complex ones
Application 4: Active and Reactive Databases

- Active databases may violate ICs temporarily
  - e.g. inventory falls below a minimum level in a warehouse
  - it is ok if the replenishments have been ordered
  - but query answers should give an indication of the inconsistency with the ICs
Database D

<table>
<thead>
<tr>
<th>Ename</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Sales</td>
</tr>
<tr>
<td>Bob</td>
<td>Sales</td>
</tr>
<tr>
<td>Tom</td>
<td>HR</td>
</tr>
</tbody>
</table>

Manager

<table>
<thead>
<tr>
<th>Ename</th>
<th>Mname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>Harry</td>
</tr>
<tr>
<td>HR</td>
<td>Tom</td>
</tr>
</tbody>
</table>

Integrity Constraints IC

Manager[mname] ⊆ Emp[Ename]

Here Inclusion Dependency

First order structure: a set of tuples/facts/ground atoms

• A database instance D is consistent if D satisfies IC
  – D ⊨ IC
  – Otherwise, D is inconsistent

Here D ⊭ IC

Inconsistent Database
Example - 2

Database D

<table>
<thead>
<tr>
<th>Student</th>
<th>Degree</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Masters</td>
<td>CS</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>Stat</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>CS</td>
</tr>
</tbody>
</table>

Integrity Constraints IC

Student → Degree, Dept

First order structure: a set of tuples/facts/ground atoms

• A database instance D is consistent if D satisfies IC
  – D ⊨ IC
  – Otherwise, D is inconsistent

Here, D ⊭ IC

Inconsistent Database
### Problems with inconsistent database

**Database D**

<table>
<thead>
<tr>
<th>Student</th>
<th>Degree</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Masters</td>
<td>CS</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>Stat</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>CS</td>
</tr>
</tbody>
</table>

**IC**

\[
\text{Student} \rightarrow \text{Degree, Dept}
\]

**SELECT**

\[
\text{Student} \\
\text{From D} \\
\text{Where Dept = 'CS'}
\]

**Results not reliable**
Solution: Database Repair

- “clean” an inconsistent database that satisfies ICs
- simulate manual cleaning automatically
Repairs: Example

Database D

<table>
<thead>
<tr>
<th>Student</th>
<th>Degree</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Masters</td>
<td>CS</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>Stat</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>CS</td>
</tr>
</tbody>
</table>

IC
Student → Degree, Dept
D1, D2 ⊨ IC
Repairs for D

Database D1

<table>
<thead>
<tr>
<th>Student</th>
<th>Degree</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Masters</td>
<td>CS</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>Stat</td>
</tr>
</tbody>
</table>

Database D2

<table>
<thead>
<tr>
<th>Student</th>
<th>Degree</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Masters</td>
<td>CS</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>CS</td>
</tr>
</tbody>
</table>

Are these two only possible repairs? Several intuitive ways to repair a database
Repairs: Definition

• Distance between two database instances
  \[ \Delta(D, D') = (D - D') \cup (D' - D) \]

• A repair $D'$ of $D$ is another instance over the same schema
  – that satisfies IC
  – and makes $\Delta(D, D')$ minimal under set inclusion
Repairs: Example

Database D

<table>
<thead>
<tr>
<th>Student</th>
<th>Degree</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Masters</td>
<td>CS</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>Stat</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>CS</td>
</tr>
</tbody>
</table>

IC

Student $\rightarrow$ Degree, Dept

D1, D2 $\models$ IC

Repairs for D

Database D1

<table>
<thead>
<tr>
<th>Student</th>
<th>Degree</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Masters</td>
<td>CS</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>Stat</td>
</tr>
</tbody>
</table>

Database D2

<table>
<thead>
<tr>
<th>Student</th>
<th>Degree</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Masters</td>
<td>CS</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>CS</td>
</tr>
</tbody>
</table>

Are these two only possible repairs?
Yes (only these two are minimal)
The set of repairs is denoted by $Rep(D, IC)$
Consistent Query Answering

• How do we compare different minimal repairs
  – we do not need to
  – we can to figure out what answers will always appear irrespective of the repair

= Consistent query answering on inconsistent databases
Consistent Query Answering (CQA)

Database D

<table>
<thead>
<tr>
<th>Student</th>
<th>Degree</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Masters</td>
<td>CS</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>Stat</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>CS</td>
</tr>
</tbody>
</table>

IC
Student → Degree, Dept

CQA:
The answer should belong to Q(D’) for ALL repairs D’

Database D1

<table>
<thead>
<tr>
<th>Student</th>
<th>Degree</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Masters</td>
<td>CS</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>Stat</td>
</tr>
</tbody>
</table>

Database D2

<table>
<thead>
<tr>
<th>Student</th>
<th>Degree</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Masters</td>
<td>CS</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>CS</td>
</tr>
</tbody>
</table>

SELECT Student From D
Where Dept = ‘CS’

Recall sure tuples in incomplete db
CQA: Definition

- Given a query $Q(x_1, ..., x_n)$
- A tuple $t = (t_1, ..., t_n)$ is a consistent answer
  - to $Q$ in $D$ w.r.t. $IC$
- If for every $D' \in \text{Rep}(D, IC)$
- $D' \models Q(t)$
  - i.e. $Q$ becomes true in $D$ when $x_1, ..., x_n$ takes values $t_1, ..., t_n$
- If $n = 0$, i.e. $Q$ is Boolean, then “yes” is a consistent answer if $D' \models Q$ for all $D' \in \text{Rep}(D, IC)$
  - Otherwise, “no” is the consistent answer
The Impact of Inconsistency

• Traditional view toward IC
  – Ignore ICs for query answering
  – Use ICs for query optimization
    • Semantic Query Optimization

• Newer approach
  – Inconsistency = Uncertainty
  – Throwing away inconsistent tuples may not be a good idea (difficult or undesirable)
  – Query results may or may not depend on ICs
  – Eliminate or tolerate inconsistency
How to find CQA?

• Naïve solution
  – Find all possible repairs Q
  – evaluate query Q[D]
  – take intersection

• There can be exponentially many repairs (why)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
</tr>
<tr>
<td>a1</td>
<td>c1</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
</tr>
<tr>
<td>a2</td>
<td>c2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>an</td>
<td>bn</td>
</tr>
<tr>
<td>an</td>
<td>cn</td>
</tr>
</tbody>
</table>

IC: Functional dependency A -> B
How to find CQA?

• Better solution

• Rewrite Q into a new query Q’
  – the usual answers to Q’ in D = the consistent answers to Q from D

• Compute query Q’ iteratively by appending to Q additional conditions called “residues”
  – obtained from Q and ICs
How to find CQA?

• Example of query rewriting

SELECT A from R

• R(A, B)
• Q(x) :- R(x, y)
• Q'(x) :- R(x, y) ∧ 
  ∨ v [¬ R(x, v) ∨ (y=v)]
• A → B
  ⇔ R(x, y) ∧ R(x, v) ⇒ y = v

• Recall that
  m ⇒ n ⇔ ¬m ∨ n
Query Rewriting

Database D

<table>
<thead>
<tr>
<th>Student</th>
<th>Degree</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Masters</td>
<td>CS</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>Stat</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>CS</td>
</tr>
</tbody>
</table>

IC

Student $\rightarrow$ Degree, Dept

Query $Q(x, y, z) :- D(x, y, z)$

IC: $\forall x, y, z, y', z' \ Q(x, y, z) \land Q(x, y', z') \rightarrow z = z'$

Rewritten query

$\neg Q(x, y, z) \land \forall y', z' \ \neg Q(x, y', z') \lor z = z'$
What else?

• Different classes/notions of
  – queries
  – integrity constraints
  – repairs
  – minimality
  – actions
## Types of ICs

<table>
<thead>
<tr>
<th>IC Type</th>
<th>General Form</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universal Constraints</td>
<td>( \forall \ldots \lnot A_1 \lor \ldots \lor \lnot A_n \lor B_1 \lor \ldots \lor B_n )</td>
<td>( \forall x [\text{Par}(x) \Rightarrow M(x) \lor F(x)] = \forall x [\lnot \text{Par}(x) \lor M(x) \lor F(x)] )</td>
</tr>
<tr>
<td>Denial Constraints</td>
<td>( \forall \ldots \lnot A_1 \lor \ldots \lor \lnot A_n )</td>
<td>( \forall x [\lnot \text{CS}(x) \lor \lnot \text{MATH}(x)] = \forall x \lnot [\text{CS}(x) \land \text{MATH}(x)] )</td>
</tr>
</tbody>
</table>
| Functional Dependencies| \( X \rightarrow Y \)  
A key dependency in \( F \)  
if \( X \) is a key                      | \( \text{SSN} \rightarrow \text{Name} \)                                 |
| Inclusion Dependencies | \( R[X] \subseteq S[Y] \)  
A foreign key constraint if \( Y \) is a key of \( S \)                               | \( \text{Manager}[\text{mname}] \subseteq \text{Employee}[\text{ename}] \) |
Complexity of CQA

• If the query rewriting technique works (First order queries), the complexity of CQA is poly-time

• But it may be hard
  – e.g. scalar aggregate queries
Aggregate Queries

IC: Employee -> Salary

<table>
<thead>
<tr>
<th>Employee</th>
<th>Salary</th>
<th>Year of Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>80k</td>
<td>10</td>
</tr>
<tr>
<td>Bob</td>
<td>100k</td>
<td>13</td>
</tr>
<tr>
<td>Bob</td>
<td>90k</td>
<td>13</td>
</tr>
</tbody>
</table>

Query:
- SELECT Max(year_of_service)
  FROM E
- always 13
- SELECT SUM(Salary)
  FROM E
  180k or 170k

Repairs

<table>
<thead>
<tr>
<th>Employee</th>
<th>Salary</th>
<th>Year of Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>80k</td>
<td>10</td>
</tr>
<tr>
<td>Bob</td>
<td>100k</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee</th>
<th>Salary</th>
<th>Year of Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>80k</td>
<td>10</td>
</tr>
<tr>
<td>Bob</td>
<td>90k</td>
<td>13</td>
</tr>
</tbody>
</table>
Aggregate Queries

IC: Employee $\rightarrow$ Salary

<table>
<thead>
<tr>
<th>Employee</th>
<th>Salary</th>
<th>year_of_service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>80k</td>
<td>10</td>
</tr>
<tr>
<td>Bob</td>
<td>100k</td>
<td>13</td>
</tr>
<tr>
<td>Bob</td>
<td>90k</td>
<td>13</td>
</tr>
</tbody>
</table>

Repairs

<table>
<thead>
<tr>
<th>Employee</th>
<th>Salary</th>
<th>year_of_service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>80k</td>
<td>10</td>
</tr>
<tr>
<td>Bob</td>
<td>100k</td>
<td>13</td>
</tr>
</tbody>
</table>

Query:

- SELECT Max(year_of_service) FROM E
- SELECT SUM(Salary) FROM E

always 13

- Range Semantics
  - Find an optimal interval $(L, U)$ such that the answer in ALL repairs is always
  - $\geq L$
  - $\leq U$
Graph-Theoretic Characterization of Repairs Conflict Hypergraph

Database D

<table>
<thead>
<tr>
<th>Student</th>
<th>Degree</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Masters</td>
<td>CS</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>Stat</td>
</tr>
<tr>
<td>Bob</td>
<td>Ph.D.</td>
<td>CS</td>
</tr>
</tbody>
</table>

IC: Student -> Degree, Dept

- What is a hypergraph?
- Vertices
  - Tuples in the database
- Edges
  - Minimal sets of tuples violating a constraint
- Repairs
  - Maximal Independent sets in the conflict graph
- What is an independent set?
  - Maximum vs. maximal
- Complexity varies with queries and ICs
  - from PTIME to Co-NP complete
- Another example on whiteboard
Tractable Queries & Join Graphs

- Query Q
- Join graph G(Q)
- Vertices are database atoms
- edge (L, L')
  - if there is a variable that occurs in a non-key attribute in L and also occurs in L'
    - appears twice if L = L'

- Class C-Tree
  - Q does not have repeated symbol = self-join free
  - G(Q) is a forest (collection of rooted trees, no cycle)
  - every non-key to key join of Q is full (involves whole key)

- CQA is tractable for C-Tree
  - conjunctive queries and FDs
Example

• Q1: $P(x, y) \land R(y, w) \land T(u, v, y)$
• Q2: $T(x, y, y)$
• Q3: $R(x, y) \land S(w, z) \land P(y, u)$
Other Repair Semantics

• Attribute-based repairs
  – A-repairs

• Cardinality-based repairs
  – C-repairs
Attribute-based repairs

• Change some attribute values in the existing tuples
• Minimize an aggregate function over changes
  – (tuple, attr, newvalue)
• Examples:
  – for each change: count 1
  – minimize sum of square difference (numeric attribute)
• Decision Problems
  – If there is a repair with cost <= a given budget
  – Repair under range semantics (aggregate queries)
Cardinality-based repairs

• Recall: $\Delta(D, D') = (D - D') \cup (D' - D)$

• Minimize $|\Delta(D, D')|$
Conclusions

• Completes our current discussion on uncertain data
• We covered
  – Probabilistic databases
    • possible world
    • dichotomy
    • Safe vs. unsafe queries
    • #P-hardness
  – Incomplete databases
    • Codd-table, V-table, c-table
  – Inconsistent databases
    • repairs, CQA
• Possible other topics in TBD lectures
  – Probabilistic Relational Model
  – Data exchange and schema mappings
• Next lecture
  – on Thursday (fall break on Tuesday)
  – New topic: Causality (in AI, Stat, DB)