Data exchange

- Source schema, target schema; need to transfer data between them.
- A typical scenario:
  - Two organizations have their legacy databases, schemas cannot be changed.
  - Data from one organization 1 needs to be transferred to data from organization 2.
  - Queries need to be answered against the transferred data.
Data Exchange

Source Schema $S$          Target Schema $T$
Data Exchange

Source Schema $S$ → Target Schema $T$
Data exchange: an example

• We want to create a target database with the schema

  \[ \text{Flight}(\text{city1}, \text{city2}, \text{aircraft}, \text{departure}, \text{arrival}) \]
  \[ \text{Served}(\text{city}, \text{country}, \text{population}, \text{agency}) \]

• We don't start from scratch: there is a source database containing relations

  \[ \text{Route}(\text{source}, \text{destination}, \text{departure}) \]
  \[ \text{BG}(\text{country}, \text{city}) \]

• We want to transfer data from the source to the target.
Data exchange – relationships between the source and the target

How to specify the relationship?

ROUTE | Source | Dest | Departure
---|---|---|---

FLIGHT | city1 | city2 | aircraft | departure | arrival

BG | Country | City

SERVED | city | country | population | agency
Relationships between the source and the target

- Formal specification: we have a *relational calculus query* over both the source and the target schema.
- The query is of a restricted form, and can be thought of as a sequence of rules:

  \[
  \text{Flight}(c_1, c_2, _, \text{dept}, _) \leftarrow \text{Route}(c_1, c_2, \text{dept})
  \]

  \[
  \text{Served}(\text{city}, \text{country}, _, _) \leftarrow \text{Route}(\text{city}, _, _), \text{BG}(\text{country}, \text{city})
  \]

  \[
  \text{Served}(\text{city}, \text{country}, _, _) \leftarrow \text{Route}(_, \text{city}, _), \text{BG}(\text{country}, \text{city})
  \]
Data exchange – targets

• Target instances should satisfy the rules.
• What does it mean to satisfy a rule?
• Formally, if we take:

\[
\text{Flight}(c_1, c_2, _, \text{dept}, _) :– \text{Route}(c_1, c_2, \text{dept})
\]

then it is satisfied by a source \( S \) and a target \( T \) if the constraint

\[
\forall c_1, c_2, d (\text{Route}(c_1, c_2, d) \rightarrow \exists a_1, a_2 (\text{Flight}(c_1, c_2, a_1, d, a_2)))
\]

• This constraint is a relational calculus query that evaluates to true or false
Data exchange – targets

• What happens if there no values for some attributes, e.g. *aircraft*, *arrival*?

• We put in null values or some real values.

• But then we may have multiple solutions!
Data exchange – targets

Source Database:

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Departure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edinburgh</td>
<td>Amsterdam</td>
<td>0600</td>
</tr>
<tr>
<td>Edinburgh</td>
<td>London</td>
<td>0615</td>
</tr>
<tr>
<td>Edinburgh</td>
<td>Frankfurt</td>
<td>0700</td>
</tr>
</tbody>
</table>

Look at the rule

\[ \text{Flight}(c_1, c_2, _, \text{dept}, _) \vdash \text{Route}(c_1, c_2, \text{dept}) \]

The right hand side is satisfied by

\[ \text{Route}(\text{Edinburgh}, \text{Amsterdam}, 0600) \]

But what can we put in the target?
Data exchange – targets

Rule:  \( \text{Flight}(c1, c2, \_ , \_ , \text{dept}, \_ ) \leftarrow \text{Route}(c1, c2, \text{dept}) \)

Satisfied by: \( \text{Route}(\text{Edinburgh, Amsterdam, 0600}) \)

Possible targets:

- \( \text{Flight}(\text{Edinburgh, Amsterdam, } \bot_1, 0600, \bot_2) \)
- \( \text{Flight}(\text{Edinburgh, Amsterdam, B737, 0600, } \bot) \)
- \( \text{Flight}(\text{Edinburgh, Amsterdam, } \bot, 0600, 0845) \)
- \( \text{Flight}(\text{Edinburgh, Amsterdam, } \bot, 0600, \bot) \)
- \( \text{Flight}(\text{Edinburgh, Amsterdam, B737, 0600, 0845}) \)

They all satisfy the constraints!
Which target to choose

- One of them happens to be right:
  - Flight(Edinburgh, Amsterdam, B737, 0600, 0845)
- But in general we do not know this; it looks just as good as
  - Flight(Edinburgh, Amsterdam, ’The Spirit of St Louis’, 0600, 1300), or
  - Flight(Edinburgh, Amsterdam, F16, 0600, 0620).
- Goal: look for the “most general” solution.
- How to define “most general”: can be mapped into any other solution.
- It is not unique either, but the space of solution is greatly reduced.
- In our case Flight(Edinburgh, Amsterdam, \(\bot_1\), 0600, \(\bot_2\)) is most general as it makes no additional assumptions about the nulls.
Towards good solutions

A solution is a database with nulls.
Reminder: every such database $T$ represents many possible complete databases, without null values:

Example

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>⊥</td>
<td>1</td>
</tr>
<tr>
<td>⊥</td>
<td>⊥</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>⊥</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>⊥</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Semantics via valuations

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>v(⊥₁) = 4</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>v(⊥₂) = 3</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>v(⊥₃) = 5</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

$⇒$

$POSS(T) = \{ R \mid v(T) \subseteq R \text{ for some valuation } v \}$
Good solutions

• An optimistic view – A good solution represents ALL other solutions:

\[
\text{POSS}(T) = \{ R \mid R \text{ is a solution without nulls} \}
\]

• Shouldn’t settle for less than – A good solution is at least as general as others:

\[
\text{POSS}(T) \supseteq \text{POSS}(T') \text{ for every other solution } T'
\]

• Good news: these two views are equivalent. Hence can take them as a definition of a good solutions.

• In data exchange, such solutions are called universal solutions.
Universal solutions: another description

- A **homomorphism** is a mapping $h : \text{Nulls} \rightarrow \text{Nulls} \cup \text{Constants}$.
- For example, $h(\bot_1) = B737$, $h(\bot_2) = 0845$.
- If we have two solutions $T_1$ and $T_2$, then $h$ is a homomorphism from $T_1$ into $T_2$ if for each tuple $t$ in $T_1$, the tuple $h(t)$ is in $T_2$.
- For example, if we have a tuple
  \[
  t = \text{Flight}(\text{Edinburgh}, \text{Amsterdam}, \bot_1, 0600, \bot_2)
  \]
  then
  \[
  h(t) = \text{Flight}(\text{Edinburgh}, \text{Amsterdam}, B737, 0600, 0845).
  \]
- A solution is **universal** if and only if there is a homomorphism from it into every other solution.
Universal solutions: still too many of them

• Take any $n > 0$ and consider the solution with $n$ tuples:

  Flight(Edinburgh, Amsterdam, $\bot_1$, 0600, $\bot_2$)
  Flight(Edinburgh, Amsterdam, $\bot_3$, 0600, $\bot_4$)
  ...
  Flight(Edinburgh, Amsterdam, $\bot_{2n-1}$, 0600, $\bot_{2n}$)

• It is universal too: take a homomorphism

$$h'(\bot_i) = \begin{cases} 
\bot_1 & \text{if } i \text{ is odd} \\
\bot_2 & \text{if } i \text{ is even}
\end{cases}$$

• It sends this solution into

  Flight(Edinburgh, Amsterdam, $\bot_1$, 0600, $\bot_2$)
Universal solutions: cannot be distinguished by conjunctive queries

- There are queries that distinguish large and small universal solutions (e.g., does a relation have at least 2 tuples?)
- But these cannot be distinguished by conjunctive queries
- Because: if $\bot_{i_1}, \ldots, \bot_{i_k}$ witness a conjunctive query, so do $h(\bot_{i_1}), \ldots, h(\bot_{i_k})$ — hence, one tuple suffices
- In general, if we have
  - a homomorphism $h : T \rightarrow T'$,
  - a conjunctive query $Q$
  - a tuple $t$ without nulls such that $t \in Q(T)$
- then $t \in Q(T')$
Universal solutions and conjunctive queries

- If
  - $T$ and $T'$ are two universal solutions
  - $Q$ is a conjunctive query, and
  - $t$ is a tuple without nulls,

  then
  \[ t \in Q(T) \iff t \in Q(T') \]
  because we have homomorphisms $T \to T'$ and $T' \to T$.

- Furthermore, if
  - $T$ is a universal solution, and $T''$ is an arbitrary solution,

  then
  \[ t \in Q(T) \implies t \in Q(T'') \]
Universal solutions and conjunctive queries cont’d

• Now recall what we learned about answering conjunctive queries over databases with nulls:
  ○ $T$ is a naive table
  ○ the set of tuples without nulls in $Q(T)$ is precisely certain$(Q, T)$ – certain answers over $T$

• Hence if $T$ is an arbitrary universal solution
  
  \[
  \text{certain}(Q, T) = \bigcap \{Q(T') \mid T' \text{ is a solution}\}
  \]

• $\bigcap \{Q(T') \mid T' \text{ is a solution}\}$ is the set of certain answers in data exchange under mapping $M$: certain$_M(Q, S)$. Thus
  
  \[
  \text{certain}_M(Q, S) = \text{certain}(Q, T)
  \]

  for every universal solution $T$ for $S$ under $M$. 
To answer conjunctive queries, one needs an arbitrary universal solution.

We saw some; intuitively, it is better to have:

\[
\text{Flight(Edinburgh, Amsterdam, } \bot_1, 0600, \bot_2)\]

than

\[
\text{Flight(Edinburgh, Amsterdam, } \bot_1, 0600, \bot_2) \\
\text{Flight(Edinburgh, Amsterdam, } \bot_3, 0600, \bot_4) \\
\ldots \\
\text{Flight(Edinburgh, Amsterdam, } \bot_{2n-1}, 0600, \bot_{2n})
\]

We now define a \textit{canonical} universal solution.
Canonical universal solution

- Convert each rule into a rule of the form:
  \[ \psi(x_1, \ldots, x_n, z_1, \ldots, z_k) \quad :\!\!:: \quad \varphi(x_1, \ldots, x_n, y_1, \ldots, y_m) \]
  (for example,
  \[ Flight(c1, c2, _, dept, _) \quad :\!\!:: \quad Route(c1, c2, dept) \]
  becomes
  \[ Flight(x_1, x_2, z_1, x_3, z_2) \quad :\!\!:: \quad Route(x_1, x_2, x_3) \]
- Evaluate \( \varphi(x_1, \ldots, x_n, y_1, \ldots, y_m) \) in \( S \).
- For each tuple \( (a_1, \ldots, a_n, b_1, \ldots, b_m) \) that belongs to the result (i.e. \( \varphi(a_1, \ldots, a_n, b_1, \ldots, b_m) \) holds in \( S \),
  do the following:
Canonical universal solution cont’d

• ... do the following:
  ◦ Create new (not previously used) null values $\bot_1, \ldots, \bot_k$
  ◦ Put tuples in target relations so that
    \[ \psi(a_1, \ldots, a_n, \bot_1, \ldots, \bot_k) \]
    holds.

• What is $\psi$?

• It is normally assumed that $\psi$ is a conjunction of atomic formulae, i.e.
  \[ R_1(\bar{x}_1, \bar{z}_1) \land \ldots \land R_l(\bar{x}_l, \bar{z}_l) \]

• Tuples are put in the target to satisfy these formulae
Canonical universal solution cont’d

• Example: no-direct-route airline:

\[
\text{Newroute}(x_1, z) \land \text{Newroute}(z, x_2) \leftarrow \text{Oldroute}(x_1, x_2)
\]

• If \((a_1, a_2) \in \text{Oldroute}(a_1, a_2)\), then create a new null \(\bot\) and put:

\[
\begin{align*}
\text{Newroute}(a_1, \bot) \\
\text{Newroute}(\bot, a_2)
\end{align*}
\]

into the target.

• Complexity of finding this solution: polynomial in the size of the source \(S'\):

\[
O(\sum_{\text{rules } \psi \leftarrow \varphi} \text{Evaluation of } \varphi \text{ on } S')
\]
Canonical universal solution and conjunctive queries

- Canonical solution: $\text{CANSOL}_M(S)$.
- We know that if $Q$ is a conjunctive query, then $\text{certain}_M(Q, S) = \text{certain}(Q, T)$ for every universal solution $T$ for $S$ under $M$.
- Hence
  \[ \text{certain}_M(Q, S) = \text{certain}(Q, \text{CANSOL}_M(S)) \]

- Algorithm for answering $Q$:
  - Construct $\text{CANSOL}_M(S)$
  - Apply naive evaluation to $Q$ over $\text{CANSOL}_M(S)$
Beyond conjunctive queries

• Everything still works the same way for \( \sigma, \pi, \bowtie, \cup \) queries of relational algebra. Adding union is harmless.
• Adding difference (i.e. going to the full relational algebra) is not.
• Reason: same as before, can encode validity problem in logic.
• Single rule, saying “copy the source into the target”

\[
T(x, y) \leftarrow S(x, y)
\]

• If the source is empty, what can a target be? Anything!
• The meaning of \( T(x, y) \leftarrow S(x, y) \) is

\[
\forall x \forall y \left( S(x, y) \rightarrow T(x, y) \right)
\]
Beyond conjunctive queries cont’d

• Look at $\varphi = \forall x \forall y (S(x, y) \rightarrow T(x, y))$

• $S(x, y)$ is always false ($S$ is empty), hence $S(x, y) \rightarrow T(x, y)$ is true ($p \rightarrow q$ is $\neg p \lor q$)

• Hence $\varphi$ is true.

• Even if $T$ is empty, $\varphi$ is true: universal quantification over the empty set evaluates to true:
  - Remember SQL’s ALL:
    ```sql
    SELECT * FROM R
    WHERE R.A > ALL (SELECT S.B FROM S)
    ```
  - The condition is true if SELECT $S$.B FROM $S$ is empty.
Beyond conjunctive queries cont’d

• Thus if \( S \) is empty and we have a rule \( T(x, y) :- S(x, y) \), then all \( T \)'s are solutions.

• Let \( Q \) be a Boolean (yes/no) query. Then

\[
\text{certain}_M(Q, S) = \text{true} \iff Q \text{ is valid}
\]

• Valid = always true.

• Validity problem in logic: given a logical statement, is it:
  - valid, or
  - valid over finite databases

• Both are undecidable.
Beyond conjunctive queries cont’d

• If we want to answer queries by rewritings, i.e. find a query $Q'$ so that

$$\text{certain}_M(Q, S) = Q'(\text{CanSol}_M(S))$$

then there is no algorithm that can construct $Q'$ from $Q$!

• Hence a different approach is needed.
Key problem

• Our main problem:
  
  Solutions are open to adding new facts

• How to close them?

• By applying the CWA (Closed World Assumption) instead of the OWA (Open World Assumption)
More flexible query answering: dealing with incomplete information

- Key issue in dealing with incomplete information:
  - Closed vs Open World Assumption (CWA vs OWA)
- CWA: database is closed to adding new facts except those consistent with one of the incomplete tuples in it.
- OWA opens databases to such facts.
- In data exchange:
  - we move data from source to target;
  - query answering should be based on that data and not on tuples that might be added later.
- Hence in data exchange CWA seems more reasonable.
Solutions under CWA – informally

- Each null introduced in the target must be justified:
  - there must be a constraint \( T(\ldots, z, \ldots) \ldots : \varphi(\ldots) \) with \( \varphi \) satisfied in the source.

- The same justification shouldn't generate multiple nulls:
  - for \( T(\ldots, z, \ldots) : \varphi(\bar{a}) \) only one new null \( \bot \) is generated in the target.

- No unjustified facts about targets should be invented:
  - assume we have \( T(x, z) : \neg \varphi(x), \quad T(z', x) : \neg \psi(x) \) and \( \varphi(a), \psi(b) \) are true in the source.
  - Then we put \( T(a, \bot) \) and \( T(\bot', b) \) in the target but not \( T(a, \bot), T(\bot, b) \) which would invent a new “fact”: \( a \) and \( b \) are connected by a path of length 2.
How to formalize this – idea

Source-to-target dependencies of the form:

\[ \psi_i(\bar{a}, z_1, \ldots, z_j, \ldots, z_k) :– \varphi_i(\bar{a}, \bar{b}) \]

Justification for a null consists of:

- a dependency (\(i\))
- a witness (\(\bar{a}, \bar{b}\)) for \(\varphi_i(\bar{a}, \bar{b})\)
- a position (\(j\)) of a null in the head of the rule.
Example

- Rule: \( \text{Flight}(c_1, c_2, z_1, \text{dept}, z_2) \leftarrow \text{Route}(c_1, c_2, \text{dept}) \)
- Witness: \( \text{Route}(\text{Edinburgh}, \text{Amsterdam}, 0600) \)
- This justifies up to two nulls:
  \[
  \text{Flight}(\text{Edinburgh}, \text{Amsterdam}, \bot_1, 0600, \bot_2)
  \]
  or
  \[
  \text{Flight}(\text{Edinburgh}, \text{Amsterdam}, \bot, 0600, \bot)
  \]
- but not
  \[
  \text{Flight}(\text{Edinburgh}, \text{Amsterdam}, \bot_1, 0600, \bot_2)
  \]
  \[
  \text{Flight}(\text{Edinburgh}, \text{Amsterdam}, \bot_3, 0600, \bot_4)
  \]
  \[
  \ldots
  \]
  \[
  \text{Flight}(\text{Edinburgh}, \text{Amsterdam}, \bot_{2n-1}, 0600, \bot_{2n})
  \]
Solutions under the CWA

- Each justification generates a null in $\text{CanSol}(S)$
- Hence for each solution $T$ under CWA there is a homomorphism
  \[ h : \text{CanSol}(S) \rightarrow T \]
  so that $T = h(\text{CanSol}(S))$
- The third requirement rules out tuples like
  \[ \text{Flight}(\text{Edinburgh}, \text{Amsterdam}, \bot, 0600, \bot) \]
- It invents a new fact: the same null is used twice in a tuple.
  - Not justified by the source and the rules
Solutions under the CWA

- The third requirement implies two facts:
  - There is a homomorphism $h' : T \rightarrow \text{CanSol}(S)$
  - $T$ contains the core of $T$
- What is the core?
- Suppose the Route relation has an extra attribute, in addition to source, destination, and departure time: it is flight#
- The same actual flight can have many flight numbers due to “code-sharing” so we might have
  Route(Edinburgh, Amsterdam, 0600, KLM 123)
  Route(Edinburgh, Amsterdam, 0600, AF 456)
  Route(Edinburgh, Amsterdam, 0600, CSA 789)
Solutions under the CWA and cores cont’d

- The canonical solution then is:
  - Flight(Edinburgh, Amsterdam, \( \bot_1 \), 0600, \( \bot_2 \))
  - Flight(Edinburgh, Amsterdam, \( \bot_3 \), 0600, \( \bot_4 \))
  - Flight(Edinburgh, Amsterdam, \( \bot_5 \), 0600, \( \bot_6 \))

- The core collapses it by means of a homomorphism
  \[
  h(\bot_1) = h(\bot_3) = h(\bot_5) = \bot_1 \quad h(\bot_2) = h(\bot_4) = h(\bot_6) = \bot_2
  \]
  to
  - Flight(Edinburgh, Amsterdam, \( \bot_1 \), 0600, \( \bot_2 \))

- Core: A minimal subinstance \( T \) of \( \text{CANSol}(S) \) so that there is a homomorphism \( h : \text{CANSol}(S) \rightarrow T \)
Cores and CWA

• Cores are universal solutions too.
  ◦ Advantage: space savings
  ◦ Disadvantage: harder to compute
    - but still in polynomial time
• Basic fact: solutions under the CWA contain the core.
• Hence tuples such as

  Flight(Edinburgh, Amsterdam, ⊥, 0600, ⊥)

are disallowed.
Solutions under the CWA: summary

• There are homomorphisms

\[ h : \text{CanSol}(S) \to T \quad h' : T \to \text{CanSol}(S) \]

○ so that \( T = h(\text{CanSol}(S)) \)

• \( T \) contains the core of \( \text{CanSol}(S) \)
Query answering under the CWA

- Given
  - a source $S$,
  - a set of rules $M$,
  - a target query $Q$,

  a tuple $t$ is in
  $$\text{certain}^\text{CWA}_M(Q, S)$$

  if it is in $Q(R)$ for every
  - solution $T$ under the CWA, and
  - $R \in \text{POSS}(T)$

- (i.e. no matter which solution we choose and how we interpret the nulls)
Query answering under the CWA – characterization

• Given a source $S$, a set of rules $M$, and a target query $Q$:

$$\text{certain}^\text{CWA}_M(Q, S) = \text{certain}(Q, \text{CanSol}(S))$$

• That is, to compute the answer to query one needs to:
  o Compute the canonical solution $\text{CanSol}(S)$ – which has nulls in it
  o Find certain answers to $Q$ over $\text{CanSol}(S)$

• If $Q$ is a conjunctive query, this is exactly what we had before
• Under the CWA, the same evaluation strategy applies to all queries!
Query answering under the CWA cont’d

• Finding certain answers is possible for many classes of queries, e.g. for all relational algebra queries.

• Complexity of finding certain answers to a query over a table with nulls

\[ \text{complexity of finding certain answers to a query over a table with nulls} \]

• polynomial time for conjunctive queries

• coNP-complete for relational algebra queries
CWA vs OWA: a comparison

• Recall the problematic case we had before:

\[ T(x, y) := S(x, y) \]

• Possible targets are extensions of the source

• Hence finding certain answers to an arbitrary relational algebra query \( Q \) was undecidable.

• Under the CWA:
  
  o The only solution is a copy of \( S \) itself (and hence it is the canonical solution)
  
  o So certain answers to \( Q \) are just \( Q(S) \) – i.e. we copy \( S \), and evaluate queries over it, as suggested by the rule.
Data exchange and integrity constraints

• Integrity constraints are often specified over target schemas
• In SQL’s data definition language one uses keys and foreign keys most often, but other constraints can be specified too.
• Adding integrity constraints in data exchange is often problematic, as some natural solutions – e.g., the canonical solution – may fail them.
• Plan:
  ○ review most commonly used database constraints
  ○ see how they may create problems in data exchange
Functional dependencies and keys

- **Functional dependency**: \( X \rightarrow Y \)

  where \( X, Y \) are sequences of attributes. It holds in a relation \( R \) if for every two tuples \( t_1, t_2 \) in \( R \):

  \[
  \pi_X(t_1) = \pi_X(t_2) \quad \text{implies} \quad \pi_Y(t_1) = \pi_Y(t_2)
  \]

- The most important special case: **keys**

- **\( K \rightarrow U \)**, where \( U \) is the set of all attributes:

  \[
  \pi_K(t_1) = \pi_K(t_2) \quad \text{implies} \quad t_1 = t_2
  \]

- That is, a key is a set of attributes that uniquely identify a tuple in a relation.
### Inclusion constraints

- **Referential** integrity constraints: they talk about attributes of one relation but refer to values in another.

- An inclusion dependency

\[
R[A_1,\ldots,A_n] \subseteq S[B_1,\ldots,B_n]
\]

It holds when

\[
\pi_{A_1,\ldots,A_n}(R) \subseteq \pi_{B_1,\ldots,B_n}(S)
\]
Foreign keys

• Most often inclusion constraints occur as a part of a foreign key
• Foreign key is a conjunction of a key and an ID:
  \[ R[A_1, \ldots , A_n] \subseteq S[B_1, \ldots , B_n] \quad \text{and} \quad \{B_1, \ldots , B_n\} \rightarrow \text{all attributes of } S \]

• Meaning: we find a key for relation \( S \) in relation \( R \).
• Example: Suppose we have relations:
  \( \text{Employee(EmplId, Name, Dept, Salary)} \)
  \( \text{ReportsTo(Empl1,Empl2)} \).

• We expect both Empl1 and Empl2 to be found in Employee; hence:
  \( \text{ReportsTo[Empl1] \subseteq Employee[EmplId]} \)
  \( \text{ReportsTo[Empl2] \subseteq Employee[EmplId]} \).
• If EmplId is a key for Employee, then these are foreign keys.
Target constraints cause problems

The simplest example:
- Copy source to target
- Impose a constraint on target not satisfied in the source

Data exchange setting:
- \( T(x, y) \leftarrow S(x, y) \) and
- Constraint: the first attribute is a key

Instance \( S \):
\[
\begin{array}{cc}
1 & 2 \\
1 & 3 \\
\end{array}
\]

Every target \( T \) must include these tuples and hence violates the key.
Target constraints: more problems

- A common problem: an attempt to repair violations of constraints leads to an sequence of adding tuples.

- Example:
  - Source $\textit{DeptEmpl}(dept\_id, manager\_name, empl\_id)$
  - Target
    - $\textit{Dept}(dept\_id, manager\_id, manager\_name)$,
    - $\textit{Empl}(empl\_id, dept\_id)$
  - Rule $\textit{Dept}(d, z, n), \textit{Empl}(e, d) \leftarrow \textit{DeptEmpl}(d, n, e)$
  - Target constraints:
    - $\textit{Dept}[manager\_id] \subseteq \textit{Empl}[empl\_id]$
    - $\textit{Empl}[dept\_id] \subseteq \textit{Dept}[dept\_id]$
Target constraints: more problems cont’d

- Start with \((CS, \text{John, 001})\) in DeptEmpl.
- Put \(\text{Dept}(CS, \bot_1, \text{John})\) and \(\text{Empl}(001, CS)\) in the target
- Use the first constraint and add a tuple \(\text{Empl}(\bot_1, \bot_2)\) in the target
- Use the second constraint and put \(\text{Dept}(\bot_2, \bot_3, \bot_3')\) into the target
- Use the first constraint and add a tuple \(\text{Empl}(\bot_3, \bot_4)\) in the target
- Use the second constraint and put \(\text{Dept}(\bot_4, \bot_5, \bot_5')\) into the target
- this never stops....
Target constraints: avoiding this problem

• Change the target constraints slightly:
  ◦ Target constraints:
    - Dept[dept_id, manager_id] ⊆ Empl[empl_id, dept_id]
    - Empl[dept_id] ⊆ Dept[dept_id]

• Again start with (CS, John, 001) in DeptEmpl.

• Put Dept(CS, ⊥₁, John) and Empl(001, CS) in the target

• Use the first constraint and add a tuple Empl(⊥₁, CS)

• Now constraints are satisfied – we have a target instance!

• What’s the difference? In our first example constraints are very cyclic causing an infinite loop. There is less cyclicity in the second example.

• Bottom line: avoid cyclic constraints.