Data Integration and Data Exchange
Traditional approach to databases

• A single large repository of data.
• Database administrator in charge of access to data.
• Users interact with the database through application programs.
• Programmers write those (embedded SQL, other ways of combining general purpose programming languages and DBMSs)
• Queries dominate; updates less common.
• DMBS takes care of lots of things for you such as
  query processing and optimisation
  concurrency control
  enforcing database integrity
Traditional approach to databases cont’d

• This model works very within a single organisation that either
  ◦ does not interact much with the outside world, or
  ◦ the interaction is heavily controlled by the DB administrators

• What do we expect from such a system?
  1. Data is relatively clean; little incompleteness
  2. Data is consistent (enforced by the DMBS)
  3. Data is there (resides on the disk)
  4. Well-defined semantics of query answering (if you ask a query, you
     know what you want to get)
  5. Access to data is controlled
The world is changing

- The traditional model still dominates, but the world is changing.
- Many huge repositories are publicly available
  - In fact many are well-organised databases, e.g., imdb.com, the CIA World Factbook, many genome databases, the DBLP server of CS publications, etc etc etc)
- Many queries cannot be answered using a single source.
- Often data from various sources needs to be combined, e.g.
  - company mergers
  - restructuring databases within a single organisation
  - combining data from several private and public sources
What industry offers now: ETL tools

• ETL stands for Extract–Transform–Load
  ○ Extract data from multiple sources
  ○ Transform it so it is compatible with the schema
  ○ Load it into a database

• Many self-built tools in the 80s and the 90s; through acquisition fewer products exist now

• The big players – IBM, Microsoft, Oracle – all have their ETL products; Microsoft and Oracle offer them with their database products.

• A few independent vendors, e.g. Informatica PowerCenter.

• Several open source products exist, e.g. Clover ETL.
ETL tools

• Focus:
  ○ Data profiling
  ○ Data cleaning
  ○ Simple transformations
  ○ Bulk loading
  ○ Latency requirements

• What they don’t do yet:
  ○ nontrivial transformations
  ○ query answering

• But techniques now exist for interesting data integration and for query answering – and we shall learn them.

• They soon will be reflected in products (IBM and Microsoft are particularly active in this area)
Data profiling/cleaning

• Data profiling: gives the user a view of data:
  ○ Samples over large tables
  ○ statistics (how many different values etc)
  ○ Graphical tools for exploring the database

• Cleaning:
  ○ Same properties may have different names
    e.g. Last_Name, L_Name, LastName
  ○ Same data may have different representations
    • e.g. (0131)555-1111 vs 01315551111,
    • George Str. vs George Street
  ○ Some data may be just wrong
Data transformation

• Most transformation rules tend to be simple:
  ◦ Copy attribute LName to Last_Name
  ◦ Set age to be current_year – DOB

• Heavy emphasis on industry specific formats

• For example, Informatica B2B Data Exchange product offers versions for Healthcare and Financial services as well as specialised tools for formats including:
  ◦ MS Word, Excel, PDF, UN/EDIFACT (Data Interchange For Administration, Commerce, and Transport), RosettaNet for B2B, and many specialised healthcare and financial form.

• These are format/industry specific and have little to do with the general tasks of data integration.
More on ETL Tools

- ETL = Extract – Transform – Load
- Typically: data integration software for building data warehouse
- Pull large volumes of data from different sources, in different formats, restructure them and load into a warehouse
- A variety of tools:
  - major database vendors (IBM, Microsoft, Oracle)
  - independent companies (Informatica)
  - Open source (e.g. Clover ETL)
- Significant demand: $1.5B/year, with >15% annual growth rate
IBM

- Product name: InfoSphere DataStage
- Main claims:
  - variety of data sources (almost any database, text, XML, web services)
  - capable of handling data arriving in real-time
  - scalability
- Unix (Linux) and Windows Platforms
InfoSphere DataStage cont’d

• InfoSphere – product line that includes software from WebSphere and Information Server lines.

• Includes lots of other things
  ○ application integration and transformation
  ○ online marketing tools
  ○ mobile, speech middleware
  ○ business process management
  ○ change data capture
  ○ information analyzer
  ○ data quality tools
InfoSphere Federation Server

- Federated (virtual) integration: “Access and integrate diverse data and content sources as if they were a single resource - regardless of where the information resides.”
- Integration across different relational products (db2, Oracle, SQL server)
- Integrity and accuracy guarantees
- Distributed query optimizer
- XML support
- Security strategies
- These are expensive products (>US$60K license)
IBM’s view of data integration

- Key tasks, with associated products
- Tasks:
  - Connect to information (products: information server; data publisher)
  - Understand information (data architect, models for ... (banking, insurance, retail, telecom))
  - Cleanse information (QualityStage: matching engine, cleaning rules etc)
  - Transform information (DataStage)
  - Deliver information (Federation Server, DataStage)
IBM: data exchange

- Clio Project (IBM Almaden Research Center).
- Includes:
  - a semi-automatic schema mapping generation tool
  - universal solutions are the semantics of data exchange
  - they are generated by extended SQL queries
  - Extension: Skolem functions
- Part of IBM Product “Rational® Data Architect”
- Other features:
  - discovery of attribute correspondence; interactive construction of mappings
  - Extended schemas (not full XML but more than relations)
Microsoft

- Integration Services – part of SQL Server (SSIS)
- Supports multiple formats; converts everything into tabular format
- Transformations:
  - join, union
  - sort
  - aggregate
  - lookup
  - convert
- Has a data quality tool
- Goes beyond traditional ETL: e.g., data and text mining tools
Oracle

- Oracle Warehouse Builder (OWB)
- Data integration and metadata management tasks:
  - Extraction, transformation, and loading (ETL) for data warehouses
  - Migrating data from legacy systems
  - Designing and managing corporate metadata
  - Data profiling
  - Data cleaning
- Included in the Oracle database product.
Oracle: transformations

• Scalar value transformations (plenty of predefined ones):
  ◦ Characters
  ◦ Conversions
  ◦ Dates
  ◦ Numbers
  ◦ Spatial objects
  ◦ XML transformations (from very simple – select nodes by XPath expressions – to very complex, such as applying XSLT style sheet)

• Also user-defined (functions, procedures, packages)
Informatica

- Market leader – Informatica PowerCenter
- Provides support for
  - migration
  - synchronization
  - warehousing
  - cross-enterprise integration
- Works with multiple data formats
- Provides support for metadata management
- Real-time capabilities
Informatica: Transformation language

- Main orientation: scalar value transformations
- Functions: change data in a mapping
- Operators: create transformation expressions
- Syntax is SQL-based
- Part of it is essentially a programming language in a Java-like syntax for manipulating values.
- Roughly: looks at a portion of the source data, modifies it, and changes the target data accordingly.
Informatica: Transformation language cont’d

- **DD_DELETE** and **DD_INSERT** specify what to do with data items.

- E.g., \( \text{IIF(job='CEO', DD_DELETE, DD_INSERT)} \) says: items with job being CEO are marked for deleting, others for insertion.

- **Operators:**
  - Arithmetic
  - String
  - Comparisons
  - Logical
  - (almost) everything you can imagine

- Many functions for dealing with dates in different formats.
Informatica: Transformation language cont’d

- Large number of functions
- Aggregates: \texttt{AVG}, \texttt{COUNT}, \texttt{MIN}, \texttt{MAX}, \texttt{MEDIAN}, \texttt{PERCENTILE}, \texttt{STDDEV}, \texttt{SUM}, etc.
- Character functions: \texttt{CONCAT}, \texttt{LENGTH}, \texttt{TRIM}, etc
- Conversion functions (e.g., \texttt{TO_CHAR} for Date, \texttt{TO_DECIMAL}, \texttt{TO_FLOAT}, \texttt{TO_DATE})
- Date functions: \texttt{ADD_TO_DATE}, \texttt{DATE_DIFF}, \texttt{DATE_COMPARE}, etc
- Numerical: the usual suspects.
- Scientific: \texttt{SIN}, \texttt{COS}, \texttt{TAN}, etc
- Search for a value in the source: \texttt{LOOKUP}
- This was quick; full manual – almost 250 pages.
Summary

- Complex tools; very good at transforming data values, and at working with specific formats (MS Word, Excel, PDF, UN/EDIFACT, RosettaNet, etc) and for specific industries (finance, insurance, health)

- Much better these days at getting real-time data; very good at bulk loading, supporting multiple formats

- Not so good:
  - virtual integration
  - complex structural transformation
  - query answering
  - metadata management

- A lot of effort will be put there over the coming years
Data integration, scenario 1

GLOBAL SCHEMA

QUERY: Q?
Data integration

GLOBAL SCHEMA

QUERY: Q?
Data integration

Answer to $Q$ is obtained by querying the views $V_1, \ldots, V_n$
Data integration, query answering

• We have our view of the world (the Global Schema)
• We can access (parts of) databases $DB_1, \ldots, DB_n$ to get relevant data.
• It comes in the form of views, $V_1, \ldots, V_n$
• Our query against the global schema must be reformulated as a query against the views $V_1, \ldots, V_n$
• The approach is completely virtual: we never create a database the conforms to the global schema.
Data integration, query answering, a toy example

- List courses taught by permanent teaching staff during Winter 2007
- We have two databases:
  - $D_1$(name, age, salary) of permanent staff
  - $D_2$(teacher, course, semester, enrollment) of courses
- $D_1$ only publishes the value of the name attribute
- $D_2$ does not reveal enrollments
- The views:
  \[ V_1 = \pi_{\text{name}}(D_1) \]
  \[ V_2 = \pi_{\text{teacher, course, semester}}(D_2) \]
- Next step: establish correspondence between attributes name of $V_1$ and teacher of $V_2$
Data integration, query answering, a toy example cont’d

- To answer query, we need to import the following data:

  \[
  V_1
  \]

  \[
  W_2 = \sigma_{semester='Winter 2007'}(V_2)
  \]

- Answering query:

  \[
  \{ course \mid \exists name, sem \ V_1(name) \land W_2(name, course, sem) \}
  \]

- Or, in relational algebra

  \[
  \pi_{course}(V_1 \bowtie_{name=teacher} W_2)
  \]
Toy example, lessons learned

- We don’t have access to all the data
- Some human intervention is essential (someone needs to tell us that teacher and name refer to the same entity)
- We don’t run a query against a single database. Instead, we
  - run queries against different databases based on restrictions they impose
  - get results to use them locally
  - run another query against those results
Toy example, things getting more complicated

- Find informatics permanent staff who taught during the Winter 2007 semester, and their phone numbers.
- We have additional personnel databases:
  - an informatics database $D_3(\text{employee, phone, office})$, and
  - a university-wide database $D_4(\text{employee, school, phone})$
- for simplicity, assume all this information is public.
- Now we have a choice:
  - use $D_3$ to get information about phones
  - use $D_4$ to get information about phones
  - use both $D_3$ and $D_4$ to get information about phones
Toy example cont’d

• First, we need some human involvement to see that employee, name, and teacher refer to the same category of objects

• If one uses $D_3$, then the query is

\[
\{ \text{name, phone} \mid \exists \text{sem, course, office} \ V_1(\text{name}) \land \\
W_2(\text{name, course, sem}) \land D_3(\text{name, phone, office}) \} 
\]

• If one uses $D_4$, then the query is

\[
\{ \text{name, phone} \mid \exists \text{sem, course, school} \ V_1(\text{name}) \land \\
W_2(\text{name, course, sem}) \land D_4(\text{name, school, phone}) \} 
\]

• But what if one uses both $D_3$ and $D_4$?
Toy example cont’d

• We could insist on the phone number being:
  ◦ in either $D_3$ or $D_4$
  ◦ in both $D_3$ and $D_4$, but not necessarily the same
  ◦ in both $D_3$ and $D_4$, and the same in both databases

• One can write queries for all the cases, but which one should we use?

• New lessons:
  ◦ databases that are being integrated are often inconsistent
  ◦ query answering is by no means unique – there could be several ways to answer a query
  ◦ different possibilities for answering queries are a result of inconsistencies and incomplete information
Toy example cont’d

• Suppose phone numbers in $D_3$ and $D_4$ are different.
• What is a sensible query answer then?
• A common approach is to use certain answers – these are guaranteed to be true.
• Another question: what if there is no record at all for the phone number in $D_3$ and $D_4$?
• Then we have an instance of incomplete information.
A different scenario

- So far we looked at virtual integration: no database of the global schema was created.
- Sometimes we need such a database to be created, for example, if many queries are expected to be asked against it.
- In general, this is a common problem with data integration: materialize vs federate.
- Materialize = create a new database based on integrating data from different sources.
- Federate = the virtual approach: obtain data from various sources and use them to answer queries.
Virtual vs Materialization

- A common situation for the materialization approach: merger of different organizations.
- A common situation for the federated approach: we don’t have full access to the data, and the data changes often.
Common tasks in data integration

• How do we represent information?
  ○ Global schema, attributes, constraints
  ○ data formats of attributes
  ○ reconciling data from different sources
  ○ abbreviations, terminology, ontologies

• How do we deal with imperfect information?
  ○ resolve overlaps
  ○ handling missing data
  ○ handling inconsistencies
Common tasks in data integration cont’d

• How do we answer queries?
  ○ what information is available?
  ○ Can we get the answer?
  ○ if not, what is the semantics of query answering?
  ○ Is query answering feasible?
  ○ Is it possible to compute query answers at all?
  ○ If now, how do we approximate?

• Materialize or federate?
Common tasks in data integration cont’d

• Do it from scratch or use commercial tools?
  ○ many are available (just google for “data integration”)
  ○ but do we fully understand them?
  ○ lots of them are very ad hoc, with poorly defined semantics
  ○ this is why it is so important to understand what really happens in data integration
Data Exchange

\[
\begin{align*}
\text{SOURCE DATABASE} \\
\text{Source Schema } S & \quad \text{Target Schema } T
\end{align*}
\]
Data Exchange

Source Schema $S$ 

Target Schema $T$
Data Exchange

Query over the target schema: \( Q \)

How to answer \( Q \) so that the answer is consistent with the data in the source database?
Data exchange vs Data integration

Data exchange appears to be an easier problem:

- there is only one source database;
- and one has complete access to the source data.

But there could be many different target instances.

Problem: which one to use for query answering?
When do we have the need for data exchange

- 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.
Query answering using views

- General setting: database relations $R_1, \ldots, R_n$.
- Several views $V_1, \ldots, V_k$ are defined as results of queries over the $R_i$’s.
- We have a query $Q$ over $R_1, \ldots, R_n$.
- Question: Can $Q$ be answered in terms of the views?
  - In other words, can $Q$ be reformulated so it only refers to the data in $V_1, \ldots, V_k$?
Query answering using views in data integration

- **LAV:**
  - $R_1, \ldots, R_n$ are global schema relations; $Q$ is the global schema query
  - $V_i$’s are the sources defined over the global schema
  - We must answer $Q$ based on the sources (virtual integration)

- **GAV:**
  - $R_1, \ldots, R_n$ are the sources that are not fully available.
  - $Q$ is a query in terms of the source (or a query that was reformulated in terms of the sources)
  - Must see if it is answerable from the available views $V_1, \ldots, V_k$.

- We know the problem is impossible to solve for full relational algebra, hence we concentrate on conjunctive queries.
Query answering using views: example

• Two relations in the database: \textit{Cites}(A,B) (if A cites B), and \textit{SameTopic}(A,B) (if A, B work on the same topic)

• Query \(Q(x,y) \leftarrow \text{SameTopic}(x,y), \text{Cites}(x,y), \text{Cites}(y,x)\)

• Two views are given:
  - \(V_1(x,y) \leftarrow \text{Cites}(x,y), \text{Cites}(y,x)\)
  - \(V_2(x,y) \leftarrow \text{SameTopic}(x,y), \text{Cites}(x,x'), \text{Cites}(y,y')\)

• Suggested rewriting: \(Q'(x,y) \leftarrow V_1(x,y), V_2(x,y)\)

• Why? Unfold using the definitions:
  \(Q'(x,y) \leftarrow \text{Cites}(x,y), \text{Cites}(y,x), \text{SameTopic}(x,y), \text{Cites}(x,x'), \text{Cites}(y,y')\)

• Equivalent to \(Q\)
Query answering using views

- Need a formal technique (algorithm): cannot rely on the semantics.
- Query $Q$:

$$\begin{align*}
\text{SELECT} & \ R1.A \\
\text{FROM} & \ R \ R1, \ R \ R2, \ S \ S1, \ S \ S2 \\
\text{WHERE} & \ R1.A=R2.A \ \text{AND} \ S1.A=S2.A \ \text{AND} \ R1.A=S1.A \\
& \ \text{AND} \ R1.B=1 \ \text{and} \ S2.B=1
\end{align*}$$

- $Q(x) := R(x, y), R(x, 1), S(x, z), S(x, 1)$
- Equivalent to $Q(x) := R(x, 1), S(x, 1)$
- So if we have a view
  - $V(x, y) := R(x, y), S(x, y)$ (i.e. $V = R \cap S$), then
  - $Q = \pi_A(\sigma_{B=1}(V))$
  - $Q$ can be rewritten (as a conjunctive query) in terms of $V$
Query rewriting

• Setting:
  ○ Queries $V_1, \ldots, V_k$ over the same schema $\sigma$ (assume to be conjunctive; they define the views)
  ○ Each $Q_i$ is of arity $n_i$
  ○ A schema $\omega$ with relations of arities $n_1, \ldots, n_k$

• Given:
  ○ a query $Q$ over $\sigma$
  ○ a query $Q'$ over $\omega$

• $Q'$ is a rewriting of $Q$ if for every $\sigma$-database $D$,
  \[ Q(D) = Q'(V_1(D), \ldots, V_k(D)) \]
Maximal rewriting

- Sometimes exact rewritings cannot be obtained
- \(Q'\) is a maximally-contained rewriting if:
  - it is contained in \(Q\):
    \[
    Q'( V_1(D), \ldots, V_k(D) ) \subseteq Q(D)
    \]
    for all \(D\)
  - it is maximal such: if
    \[
    Q''( V_1(D), \ldots, V_k(D) ) \subseteq Q(D)
    \]
    for all \(D\), then
    \[
    Q'' \subseteq Q'
    \]
Query rewriting: a naive algorithm

- Given:
  - conjunctive queries $V_1, \ldots, V_k$ over schema $\sigma$
  - a query $Q$ over $\sigma$

- Algorithm:
  - guess a query $Q'$ over the views
  - Unfold $Q'$ in terms of the views
  - Check if the unfolding is contained in $Q$

- If one unfolding is equivalent to $Q$, then $Q'$ is a rewriting

- Otherwise take the union of all unfoldings contained in $Q$
  - it is a maximally contained rewriting
Why is it not an algorithm yet?

- **Problem**: the guess stage.
  - There are infinitely many conjunctive queries.
  - *We cannot check them all.*
  - **Solution**: we only need to check a few.
Guessing rewritings

- A basic fact:
  - If there is a rewriting of $Q$ using $V_1, \ldots, V_k$, then there is a rewriting with no more conjuncts than in $Q$.
  - E.g., if $Q(x) := R(x, y), R(x, 1), S(x, z), S(x, 1)$, we only need to check conjunctive queries over $V$ with at most 4 conjuncts.
- Moreover, maximally contained rewriting is obtained as the union of all conjunctive rewritings of length of $Q$ or less.
- Complexity: enumerate all candidates (exponentially many); for each an NP (or exponential) algorithm. Hence exponential time is required.
- Cannot lower this due to NP-completeness.
Query rewriting

- Recall the algorithm, for a given $Q$ and view definitions $V_1, \ldots, V_k$:
  - Look at all rewritings that have as at most as many joins as $Q$
  - check if they are contained in $Q$
  - take the union of those that are
- This is the maximally contained rewriting
- There are algorithms that prune the search space and make looking for rewritings contained in $Q$ more efficient
  - the bucket algorithm
  - MiniCon
How hard is it to answer queries using views?

• Setting: we now have an actual content of the views.

• As before, a query is $Q$ posed against $D$, but must be answered using information in the views.

• Suppose $I_1, \ldots, I_k$ are view instances. Two possibilities:
  - Exact mappings: $I_j = V_j(D)$
  - Sound mappings: $I_j \subseteq V_j(D)$

• We need certain answers for given $\mathcal{I} = (I_1, \ldots, I_k)$:
  $$\text{certain}_{\text{exact}}(Q, \mathcal{I}) = \bigcap_{D: I_j = V_j(D) \text{ for all } j} Q(D)$$
  $$\text{certain}_{\text{sound}}(Q, \mathcal{I}) = \bigcap_{D: I_j \subseteq V_j(D) \text{ for all } j} Q(D)$$
How hard is it to answer queries using views?

• If $\text{certain}_{\text{exact}}(Q, I)$ or $\text{certain}_{\text{sound}}(Q, I)$ are impossible to obtain, we want maximally contained rewritings:
  
  $\circ$ $Q'(I) \subseteq \text{certain}_{\text{exact}}(Q, I)$, and
  
  $\circ$ if $Q''(I) \subseteq \text{certain}_{\text{exact}}(Q, I)$ then $Q''(I) \subseteq Q'(I)$
  
  $\circ$ (and likewise for sound)

• How hard is it to compute this from $I$?
Complexity of query answering

• We want the complexity of finding
  \[ \text{certain}_{\text{exact}}(Q, I) \text{ or } \text{certain}_{\text{sound}}(Q, I) \]
  in terms of the size of \( I \)

• If all view definitions are conjunctive queries and \( Q \) is a relational algebra or a SQL query, then the complexity is \( \text{coNP} \).

• This is too high!

• If all view definitions are conjunctive queries and \( Q \) is a conjunctive query, then the complexity is \( \text{PTIME} \).
  
  ○ Because: the maximally contained rewriting computes certain answers!
Complexity of query answering

<table>
<thead>
<tr>
<th>view language</th>
<th>CQ</th>
<th>CQ$\neq$</th>
<th>relational calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQ</td>
<td>ptime</td>
<td>coNP</td>
<td>undecidable</td>
</tr>
<tr>
<td>CQ$\neq$</td>
<td>ptime</td>
<td>coNP</td>
<td>undecidable</td>
</tr>
<tr>
<td>relational calculus</td>
<td>undecidable</td>
<td>undecidable</td>
<td>undecidable</td>
</tr>
</tbody>
</table>

CQ – conjunctive queries

CQ$\neq$ – conjunctive queries with inequalities
(for example, $Q(x) :\neg R(x, y), S(y, z), x \neq z$)
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 $S$

Source DATABASE

Target Schema $T$

TARGET DATABASE

?????
**Data exchange: an example**

- We want to create a **target** database with the schema

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

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

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

- 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?
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:

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

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

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

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

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

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

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

- This constraint is a relational calculus query that evaluates to \textit{true} or \textit{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>

BG:

<table>
<thead>
<tr>
<th>Country</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK</td>
<td>London</td>
</tr>
<tr>
<td>UK</td>
<td>Edinburgh</td>
</tr>
<tr>
<td>NL</td>
<td>Amsterdam</td>
</tr>
<tr>
<td>GER</td>
<td>Frankfurt</td>
</tr>
</tbody>
</table>

Look at the rule

\[ \text{Flight}(c1, c2, _, \text{dept}, _) \Leftarrow \text{Route}(c1, c2, \text{dept}) \]

The right hand side is satisfied by

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

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

Rule:  \( \text{Flight}(c_1, c_2, \_\_, \text{dept}, \_\_) \leftarrow \text{Route}(c_1, c_2, \text{dept}) \)

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

Possible targets:

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

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, ⊥₁, 0600, ⊥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

\[
\begin{array}{ccc}
A & B & C \\
1 & 2 & \bot_1 \\
\bot_2 & \bot_1 & 3 \\
\bot_3 & 5 & 1 \\
2 & \bot_3 & 3 \\
\end{array}
\]

Semantics via valuations

\[
\begin{array}{ccc}
A & B & C \\
1 & 2 & 4 \\
3 & 4 & 3 \\
5 & 5 & 1 \\
2 & 5 & 3 \\
3 & 7 & 8 \\
4 & 2 & 1 \\
\end{array}
\]

\[ [T]_{owa} = \{ R \mid v(T) \subseteq R \text{ for some valuation } v \} \]
Good solutions

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

\[ [T]_{\text{owa}} = \{ R \mid R \text{ is a solution without nulls} \} \]

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

\[ [T]_{\text{owa}} \supseteq [T']_{\text{owa}} \text{ for every other solution } T' \]

• 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, \( h(\perp_1) = B737 \), \( h(\perp_2) = 0845 \).
• 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

\[
t = \text{Flight}(\text{Edinburgh, Amsterdam, } \perp_1, 0600, \perp_2)
\]

then

\[
h(t) = \text{Flight}(\text{Edinburgh, Amsterdam, B737, 0600, 0845})
\]

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

• 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

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


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 ⊥₁, ..., ⊥ₖ witness a conjunctive query, so do h(⊥₁), ..., h(⊥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
  o $T$ and $T'$ are two universal solutions
  o $Q$ is a conjunctive query, and
  o $t$ is a tuple without nulls,
  then
  \[ t \in Q(T) \iff t \in Q(T') \]
  because we have homomorphisms $T \rightarrow T'$ and $T' \rightarrow T$.

• Furthermore, if
  o $T$ 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 $M$. 
Universal solutions cont’d

- To answer conjunctive queries, one needs an arbitrary universal solution.
- We saw some; intuitively, it is better to have:
  
  Flight(Edinburgh, Amsterdam, ⊥₁, 0600, ⊥₂)

  than

  Flight(Edinburgh, Amsterdam, ⊥₁, 0600, ⊥₂)
  Flight(Edinburgh, Amsterdam, ⊥₃, 0600, ⊥₄)
  ...
  Flight(Edinburgh, Amsterdam, ⊥₂ⁿ₋₁, 0600, ⊥₂ⁿ)

- We now define a canonical universal solution.
Canonical universal solution

• Convert each rule into a rule of the form:

\[ \psi(x_1, \ldots, x_n, z_1, \ldots, z_k) \implies \varphi(x_1, \ldots, x_n, y_1, \ldots, y_m) \]

(for example,

\[ \text{Flight}(c1, c2, \_ \_ \_, \text{dept}, \_ \_ \_) \implies \text{Route}(c1, c2, \text{dept}) \]

becomes

\[ \text{Flight}(x_1, x_2, z_1, x_3, z_2) \implies \text{Route}(x_1, x_2, x_3) \]

• 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:

\[
\text{Newroute}(x_1, z) \land \text{Newroute}(z, x_2) \quad :- \quad \text{Oldroute}(x_1, x_2)
\]

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

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

into the target.

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

\[
O\left( \sum_{\text{rules } \psi \quad :- \quad \varphi} \text{Evaluation of } \varphi \text{ on } S \right)
\]
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.

L. Libkin 82
Beyond conjunctive queries cont’d

• Thus if $S$ is empty and we have a rule $T(x, y) \leftarrow S(x, y)$, 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 \( \ldots T(\ldots, z, \ldots) \ldots :\rightharpoonup \varphi(\ldots) \) with \( \varphi \) satisfied in the source.

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

- No unjustified facts about targets should be invented:
  - assume we have \( T(x, z) :\rightharpoonup \varphi(x), \ T(z', x) :\rightharpoonup \psi(x) \) and \( \varphi(a), \psi(b) \) are true in the source.
  - Then we put \( T(a, \perp) \) and \( T(\perp', b) \) in the target but not \( T(a, \perp), T(\perp, b) \) which would invent a new “fact”: \( a \) and \( b \) are connected by a path of length \( 2 \).
Solutions under the CWA: summary

- There are homomorphisms

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

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

- \( T \) contains the core of \( \text{CanSol}(S) \)

- Core: the smallest \( C' \subseteq \text{CanSol}(S) \) such that there is a homomorphism from \( \text{CanSol}(S) \) to \( C' \).

- Often saves space, but takes time to compute.

- Data exchange systems try to move from \( \text{CanSol}(S) \) to the core but usually stop half-way due to the complexity of computation.
Query answering under the CWA

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

  a tuple $t$ is in $\text{certain}_{M}^{\text{CWA}}(Q, S)$ if it is in $Q(R)$ for every
  - solution $T$ under the CWA, and
  - $R \in [T]_{\text{owa}}$

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

• That is, to compute the answer to query one needs to:
  ○ Compute the canonical solution $\text{CanSol}(S)$ – which has nulls in it
  ○ 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!
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.
Target constraints cause problems

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

• Data exchange setting:
  o \( T(x, y) := S(x, y) \) and
  o 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.
Schema mappings

• Rules used in data exchange specify mappings between schemas.
• To understand the evolution of data one needs to study operations on schema mappings.
• Most commonly we need to deal with two operations:
  ○ composition
  ○ inverse
Composition and inverse

\[ S_1 \xrightarrow{\Sigma} S_2 \xrightarrow{\Delta} S_3 \]
Composition and inverse

\[ \Sigma \circ \Delta \]

Diagram:

- **S1** → **S2** by \(\Sigma\)
- **S2** → **S3** by \(\Delta\)

\[ \Sigma \circ \Delta \]
Composition and inverse

\[ \Sigma \circ \Delta \]

\[
\begin{array}{ccc}
S1 & \xrightarrow{\Sigma} & S2 \\
\Gamma & \downarrow & \\
S1' & \xrightarrow{\Delta} & S3
\end{array}
\]
Composition and inverse

\[ \Sigma \circ \Delta \]

\[ S_1 \xrightarrow{\Sigma} S_2 \xrightarrow{\Delta} S_3 \]

\[ \Gamma \]

\[ \Gamma^{-1} \circ (\Sigma \circ \Delta) \]
Mappings

- Schema mappings are typically given by rules
  \[ \psi(\bar{x}, \bar{z}) := \exists \bar{u} \ \varphi(\bar{x}, \bar{y}, \bar{u}) \]
  
  where
  - \( \psi \) is a conjunction of atoms over the target:
    \[ T_1(\bar{x}_1, \bar{z}_1) \land \ldots \land T_m(\bar{x}_m, \bar{z}_m) \]
  - \( \varphi \) is a conjunction of atoms over the source:
    \[ S_1(\bar{x}_1', \bar{y}_1, \bar{u}_1) \land \ldots \land S_k(\bar{x}_k', \bar{y}_k, \bar{u}_k) \]

- Example: \( Served(x_1, x_2, z_1, z_2) := \exists u_1, u_2 \ Route(x_1, u_1, u_2) \land BG(x_1, x_2) \)

L. Libkin

101
The closure problem

• Are mappings closed under
  ○ composition?
  ○ inverse?
• If not, what needs to be added?
• It turns out that mappings are not closed under inverses and composition.
• We next see what might need to be added to them.
Skolem functions

- Source: `EP(empl_name, dept, project)`
  Target: `EDPH(empl_id, dept, phone), DP(dept, project)`

- A natural mapping is:
  
  \[
  EDPH(z_1, x_2, z_3) \land DP(x_2, x_3) :\leftarrow EP(x_1, x_2, x_3)
  \]

- This is problematic: if we have tuples
  
  \[(John, CS, P_1) \quad (John, CS, P_2)\]

  in `EP`, the canonical solution would have

<table>
<thead>
<tr>
<th></th>
<th>CS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>_1</td>
<td></td>
<td>_1'</td>
</tr>
<tr>
<td>_2</td>
<td></td>
<td>_2'</td>
</tr>
</tbody>
</table>

  corresponding to two projects `P_1` and `P_2`.

- So `empl_id` is hardly an id!
Skolem functions cont’d

• Solution: make empl_id a function of empl_name.

• Such “invented” functions are called Skolem functions (see Logic 001 for a proper definition)

• Source: EP(empl_name,dept,project);
  Target: EDPH(empl_id,dept,phone), DP(dept,project)

• A new mapping is:
  
  \[ \text{EDPH}(f(x_1), x_2, z_3) \land \text{DP}(x_2, x_3) \land \text{EP}(x_1, x_2, x_3) \]

• \( f \) assigns a unique id to every name.
Other possible additions

• One can look at more general queries used in mappings.
• Most generally, relational algebra queries, but to be more modest, one can start with just adding inequalities.
• One may also disjunctions: for example, if we want to invert

\[
\begin{align*}
T(x) & :\neg S_1(x) \\
T(x) & :\neg S_2(x)
\end{align*}
\]

it seems natural to introduce a rule

\[
S_1(x) \lor S_2(x) :\neg T(x)
\]
Composition: definition

- Recall the definition of composition of binary relations $R$ and $R'$:
  \[(x, z) \in R \circ R' \iff \exists y : (x, y) \in R \text{ and } (y, z) \in R'\]

- A schema mapping $\Sigma$ for two schemas $\sigma$ and $\tau$ is viewed as a binary relation
  \[\Sigma = \{(S, T) \mid \begin{array}{l}
  S \text{ is a } \sigma\text{-instance} \\
  T \text{ is a } \tau\text{-instance} \\
  T \text{ is a solution for } S
  \end{array}\}\]

- The composition of mappings $\Sigma$ from $\sigma$ to $\tau$ and $\Delta$ from $\tau$ to $\omega$ is now
  \[\Sigma \circ \Delta\]

- Question (closure): is there a mapping $\Gamma$ between $\sigma$ and $\omega$ so that
  \[\Gamma = \Sigma \circ \Delta\]
Composition: when it works

- If $\sum$
  - does not generate any nulls, and
  - no variables $\bar{u}$ for source formulas

- Example:
  
  $\sum : \quad T(x_1, x_2) \wedge T(x_2, x_3) :\leftarrow S(x_1, x_2, x_3)$
  
  $\Delta : \quad W(x_1, x_2, z) :\leftarrow T(x_1, x_2)$

- First modify into:
  
  $\sum : \quad T(x_1, x_2) :\leftarrow S(x_1, x_2, x_3)$
  
  $\sum : \quad T(x_2, x_3) :\leftarrow S(x_1, x_2, x_3)$
  
  $\Delta : \quad W(x_1, x_2, z) :\leftarrow T(x_1, x_2)$

- Then substitute in the definition of $W$: 
Composition: when it cont’d

\[
W(x_1, x_2, z) := S(x_1, x_2, y) \\
W(x_1, x_2, z) := S(y, x_1, x_2)
\]

to get \( \Sigma \circ \Delta \).

Explaining the second rule:

\[
W(x_1, x_2, z) \\
\rightarrow T(x_1, x_2) \quad \text{using} \quad T(var_1, var_2) := S(var_3, var_1, var_2) \\
\rightarrow S(y, x_1, x_2)
\]
Composition: when it doesn’t work

- Schema $\sigma$: Takes(st_name, course)
- Schema $\tau$: Takes’(st_name, course), Nameld(st_name, st_id)
- Schema $\omega$: Enroll(st_id, course)
- Mapping $\Sigma$ from $\sigma$ to $\tau$:
  \[
  \begin{align*}
  \text{Takes}'(s, c) & : \leftarrow \text{Takes}(s, c) \\
  \text{Nameld}(s, i) & : \leftarrow \exists c \text{Takes}(s, c)
  \end{align*}
  \]

- Mapping $\Delta$ from $\tau$ to $\omega$:
  \[
  \begin{align*}
  \text{Enroll}(i, c) & : \leftarrow \text{Nameld}(s, i) \land \text{Takes}'(s, c)
  \end{align*}
  \]

- A first attempt at the composition: $\text{Enroll}(i, c) : \leftarrow \text{Takes}(s, c)$
Composition: when it doesn’t work cont’d

• What’s wrong with $\Gamma$: $\text{Enroll}(i, c) :\leftarrow \text{Takes}(s, c)$?
• Student id $i$ depends on both name and course!

\[
\begin{array}{c|c|c}
\text{Takes:} & \text{John} & \text{CS1} \\
& \text{John} & \text{CS2} \\
\hline
\sum & \Rightarrow & \text{Takes':} \\
\text{Nameld:} & \text{John} & \bot \\
\hline
\end{array}
\Rightarrow
\begin{array}{c|c|c}
\text{Takes':} & \text{John} & \text{CS1} \\
& \text{John} & \text{CS2} \\
\hline
\Delta & \Rightarrow & \text{Enroll:} \\
\bot & \text{CS1} \\
\bot & \text{CS2} \\
\hline
\end{array}
\]

But:

\[
\begin{array}{c|c|c}
\text{Takes:} & \text{John} & \text{CS1} \\
& \text{John} & \text{CS2} \\
\hline
\overset{\Gamma}{\Rightarrow} & \Rightarrow & \text{Enroll:} \\
\bot_1 & \text{CS1} \\
\bot_2 & \text{CS2} \\
\hline
\end{array}
\]

L. Libkin
Composition: when it doesn’t work cont’d

• Solution: Skolem functions.

• $\Gamma'$: \( \text{Enroll}(f(s), c) \leftarrow \text{Takes}(s, c) \)

• Then:

\[
\begin{array}{c|c|c|c|}
\text{Takes:} & \text{John} & \text{CS1} & \text{John} & \text{CS2} \\
& \Rightarrow & & & \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|}
\text{Enroll:} & \bot & \text{CS1} & \bot & \text{CS2} \\
& & & & \\
\end{array}
\]

• where \( \bot = f(\text{John}) \)
Composition: another example

• Schema $\sigma$: Empl(eid)
• Schema $\tau$: Mngr(eid,mngid)
• Schema $\omega$: Mngr'(eid,mngid), SelfMng(id)
• Mapping $\Sigma$ from $\sigma$ to $\tau$:

$\text{Mngr}(e,m) :\leftarrow \text{Empl}(e)$

• Mapping $\Delta$ from $\tau$ to $\omega$:

$\text{Mngr}'(e,m) :\leftarrow \text{Mngr}(e,m)$
$\text{SelfMng}(e) :\leftarrow \text{Mngr}(e,e)$

• Composition:

$\text{Mngr}'(e, f(e)) :\leftarrow \text{Empl}(e)$
$\text{SelfMng}(e) :\leftarrow \text{Empl}(e) \land e = f(e)$
Composition and Skolem functions

- Schema mappings with Skolem functions \textit{compose!}

- Algorithm:
  
  - replace all nulls by Skolem functions
    - \( \text{Mngr}(e, f(e)) \rightarrow \text{Empl}(e) \)
    - \( \Delta \) stays as before

  - Use substitution:
    - \( \text{Mngr}'(e, m) \rightarrow \text{Mngr}(e, m) \) becomes
      \( \text{Mngr}'(e, f(e)) \rightarrow \text{Empl}(e) \)
    - \( \text{SelfMng}(e) \rightarrow \text{Mngr}(e, e) \) becomes
      \( \text{SelfMng}(e) \rightarrow \text{Empl}(e) \land e = f(e) \)
Inverting mappings

• Harder than composition.
• Intuition: $\Sigma \circ \Sigma^{-1} = \text{ID}$.
• But even what $\text{ID}$ should be is not entirely clear.
• Some intuitive examples will follow.
Examples of inversion

- The inverse of projection is null invention:
  - $T(x) \Leftarrow S(x, y)$
  - $S(x, y) \Leftarrow T(x)$

- Inverse of union requires disjunction:
  - $T(x) \Leftarrow S(x)$  \hspace{1em}  $T(x) \Leftarrow S'(x)$
  - $S(x) \lor S'(x) \Leftarrow T(x)$

- So reversing the rules doesn’t always work.
Examples of inversion cont’d

- Inverse of decomposition is join:
  - $T(x_1, x_2) \land T'(x_2, x_3) \leftarrow S(x_1, x_2, x_3)$
  - $S(x_1, x_2, x_3) \leftarrow T(x_1, x_2) \land T'(x_2, x_3)$

- But this is also an inverse of $T(x_1, x_2) \land T'(x_2, x_3) \leftarrow S(x_1, x_2, x_3)$:
  - $S(x_1, x_2, z) \leftarrow T(x_1, x_2)$
  - $S(z, x_2, x_3) \leftarrow T'(x_2, x_3)$
Examples of inversion cont’d

• One may need to distinguish nulls from values in inverses.
• $\Sigma$ given by

$$
\begin{align*}
T_1(x) & \leftarrow S(x, x) \\
T_2(x, z) & \leftarrow S(x, y) \land S(y, x) \\
T_3(x_1, x_2, z) & \leftarrow S(x_1, x_2)
\end{align*}
$$

• Its inverse $\Sigma^{-1}$ requires:
  
  ○ a predicate $\text{NotNull}$ and
  
  ○ inequalities:

$$
\begin{align*}
S(x, x) & \leftarrow T_1(x) \land T_2(x, y_1) \land T_3(x, x, y_2) \land \text{NotNull}(x) \\
S(x_1, x_2) & \leftarrow T_3(x_1, x_2, y) \land (x_1 \neq x_2) \land \text{NotNull}(x_1) \land \text{NotNull}(x_2)
\end{align*}
$$
**Integrating preferences/rankings**

**Problem statement**

- Each object has \( m \) grades, one for each of \( m \) criteria.
- The grade of an object for field \( i \) is \( x_i \).
- Normally assume \( 0 \leq x_i \leq 1 \).
  - Typically evaluations based on different criteria
  - The higher the value of \( x_i \), the better the object is according to the \( i \)th criterion
- The objects are given in \( m \) sorted lists
  - the \( i \)th list is sorted by \( x_i \) value
  - These lists correspond to different sources or to different criteria.
- Goal: find the top \( k \) objects.
## Example

<table>
<thead>
<tr>
<th>Grade 1</th>
<th>Grade 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(17, 0.9936)</td>
<td>(235, 0.9996)</td>
</tr>
<tr>
<td>(1352, 0.9916)</td>
<td>(12, 0.9966)</td>
</tr>
<tr>
<td>(702, 0.9826)</td>
<td>(8201, 0.9926)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(12, 0.3256)</td>
<td>(17, 0.406)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Aggregation Functions

• Have an aggregation function $F$.
• Let $x_1, \ldots, x_m$ be the grades of object $R$ under the $m$ criteria.
• Then $F(x_1, \ldots, x_m)$ is the overall grade of object $R$.
• Common choices for $F$:
  ◦ min
  ◦ average or sum
• An aggregation function $F$ is monotone if
  \[ F(x_1, \ldots, x_m) \leq F(x'_1, \ldots, x'_m) \]
  whenever $x_i \leq x'_i$ for all $i$. 
Other Applications

- Information retrieval
- Objects $R$ are documents.
- The $m$ criteria are search terms $s_1, \ldots, s_m$.
- The grade $x_i$: how relevant document $R$ is for search term $s_i$.
- Common to take the aggregation function $F$ to be the sum

$$F(x_1, \ldots, x_m) = x_1 + \cdots + x_m.$$
Modes of Access

• **Sorted** access
  - Can obtain the next object with its grade in list $L_i$
  - cost $c_S$.

• **Random** access
  - Can obtain the grade of object $R$ in list $L_i$
  - cost $c_R$.

• **Middleware cost:**
  $$c_S \cdot (\# \text{ of sorted accesses}) + c_R \cdot (\# \text{ of random accesses}).$$
Algorithms

• Want an algorithm for finding the top $k$ objects.
• Naive algorithm:
  o compute the overall grade of every object;
  o return the top $k$ answers.
• Too expensive.
Fagin’s Algorithm (FA)

1. Do sorted access in parallel to each of the $m$ sorted lists $L_i$.
   - Stop when there are at least $k$ objects, each of which have been seen in all the lists.

2. For each object $R$ that has been seen:
   - Retrieve all of its fields $x_1, \ldots, x_m$ by random access.
   - Compute $F(R) = F(x_1, \ldots, x_m)$.

3. Return the top $k$ answers.
Fagin’s algorithm is correct

- Assume object $R$ was not seen
  - its grades are $x_1, \ldots, x_m$.
- Assume object $S$ is one of the answers returned by FA
  - its grades are $y_1, \ldots, y_m$.
- Then $x_i \leq y_i$ for each $i$
- Hence

$$F(R) = F(x_1, \ldots, x_m) \leq F(y_1, \ldots, y_m) = F(S).$$
Fagin’s algorithm: performance guarantees

- Typically probabilistic guarantees
- Orderings are independent
- Then with high probability the middleware cost is

\[ O\left( N \cdot \frac{m}{\sqrt{N}} \sqrt{k/N} \right) \]

- i.e., sublinear
- But may perform poorly
  - e.g., if \( F \) is constant:
    - still takes \( O\left( N \cdot \frac{m}{\sqrt{N}} \sqrt{k/N} \right) \) instead of a constant time algorithm
Optimal algorithm: The Threshold Algorithm

1. Do sorted access in parallel to each of the $m$ sorted lists $L_i$. As each object $R$ is seen under sorted access:
   - Retrieve all of its fields $x_1, \ldots, x_m$ by random access.
   - Compute $F(R) = F(x_1, \ldots, x_m)$.
   - If this is one of the top $k$ answers so far, remember it.
   - Note: buffer of bounded size.

2. For each list $L_i$, let $\hat{x}_i$ be the grade of the last object seen under sorted access.

3. Define the threshold value $t$ to be $F(\hat{x}_1, \ldots, \hat{x}_m)$.

4. When $k$ objects have been seen whose grade is at least $t$, then stop.

5. Return the top $k$ answers.
Threshold Algorithm: correctness and optimality

• The Threshold Algorithm is correct for every monotone aggregate function $F$.

• Optimal in a very strong sense:
  ○ it is as good as any other algorithm on every instance
  ○ any other algorithm means: except pathological algorithms
  ○ as good means: within a constant factor
  ○ pathological means: making wild guesses.
Wild guesses can help

- An algorithm “makes a wild guess” if it performs random access on an object not previously encountered by sorted access.
- Neither FA nor TA make wild guesses, nor does any “natural” algorithm.
- Example: The aggregation function is $\min$; $k = 1$.

<table>
<thead>
<tr>
<th>LIST $L_1$</th>
<th>LIST $L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 1)</td>
<td>(2n+1, 1)</td>
</tr>
<tr>
<td>(2, 1)</td>
<td>(2n, 1)</td>
</tr>
<tr>
<td>(3, 1)</td>
<td>(2n-1, 1)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(n+1, 1)</td>
<td>(n+1, 1)</td>
</tr>
<tr>
<td>(n+2, 0)</td>
<td>(n, 0)</td>
</tr>
<tr>
<td>(n+3, 0)</td>
<td>(n-1, 0)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(2n+1, 0)</td>
<td>(1, 0)</td>
</tr>
</tbody>
</table>
Threshold Algorithm as an approximation algorithm

- Approximately finding top \( k \) answers.
- For \( \varepsilon > 0 \), an \( \varepsilon \)-approximation of top \( k \) answers is a collection of \( k \) objects \( R_1, \ldots, R_k \) so that
  - for each \( R \) not among them,
    \[
    (1 + \varepsilon) \cdot F(R_i) \geq F(R)
    \]

- Turning TA into an approximation algorithm:
- Simply change the stopping rule into:
  - When \( k \) objects have been seen whose grade is at least
    \[
    t \geq \frac{1}{1 + \varepsilon'},
    \]
    then stop.