Overview

• Topics covered in this part
  – Heterogeneity and Autonomy
  – Data Integration Tasks
  – Data Integration Architectures (Methods)
  – Some Formal Background (sorry!)

1.1 Heterogeneity + Autonomy

• Taxonomy of Heterogeneity
1.1 System Heterogeneity

- Interface Heterogeneity
  - Different interfaces for accessing data from a source
    - HTML forms
    - Services (SOA)
    - Declarative language
    - Files
    - Proprietary network protocol
    - …

- Interface Heterogeneity – Expressiveness
  - Keyword-search vs. query language
  - Predicates: equality (=), inequality (<, !=)
  - Logical connectives: conjunctive (AND), disjunctive (OR), negation
  - Complex operations: aggregation, quantification
  - Limitations: restriction to particular tables, predicates, fixed queries with parameters, …

- Interface Heterogeneity – Examples
  - Google search (+/-, site:, intitle:, filetype:
  - SQL
  - Web-form (with DB backend?)

Keyword

Fixed choices

"Bound parameter"
1.1 System Heterogeneity

- Interface Heterogeneity – Examples
  - Email-client
  - Name Query
  - Disjointive or conjunctive
  - Comparison operator

- Problems with interface heterogeneity
  - Global query language is more powerful
    - User queries may not be executable
  - Integration system has to evaluate part of the query
  - Bound parameters are incompatible with query
    - User query may not be executable

- Example: more expressive global language
  - SQL with one table
    - books (title, author, year, isbn, genre)
  - Web form for books about history shown below
  - What problems may arise translating user queries?

- Integration system has to process part of the query
  - SELECT title
  - FROM books
  - WHERE author = 'Steven King'
  - AND year = 2012;

- Query requires multiple requests
  - SELECT title
  - FROM books
  - WHERE author LIKE '%King%';

- Query cannot be answered
  - SELECT title
  - FROM books
  - WHERE genre = 'SciFi';

How do we know what authors exist?

Web form is for history book only!
1.1 Heterogeneity + Autonomy

• Taxonomy of Heterogeneity

1.1 Structural Heterogeneity

• Data model
  – Different semantic/expressiveness
  – Different structure

• Schema
  – Integrity constraints, keys
  – Schema elements:
    • use attribute or separate relations
  – Structure:
    • e.g., normalized vs. denormalized relational schema

1.1 Structural Heterogeneity

• Example: data model
  – Relational model
  – XML model
  – JSON
  – OO

• Person and their addresses

1.1 Structural Heterogeneity

• Schema
  – Modeling choices
    • Relation vs. attribute
    • Attribute vs. value
    • Relation vs. value
  – Naming
  – Normalized vs. denormalized (relational concept)
  – Nesting vs. reference
1.1 Structural Heterogeneity

- **Relation-relation conflicts**
  - Naming conflicts
    - Relations with different name representing the same data (synonym)
    - Relations with same name representing different information (homonym)
  - Structural conflicts
    - Missing attributes
    - Many-to-one
    - Missing, but derivable attributes

- **Attribute-attribute conflicts**
  - Naming conflicts
    - Attributes with different name representing the same data (synonym)
    - Attributes with same name representing different information (homonym)
  - Default value conflict
  - Integrity constraint conflicts
    - Datatype
    - Constraints restricting values

**Example:** Conflicts between attributes and attributes

<table>
<thead>
<tr>
<th>SSN</th>
<th>FirstName</th>
<th>LastName</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>Peter</td>
<td>Schmeter</td>
<td>30</td>
</tr>
<tr>
<td>333-333-9999</td>
<td>Hans</td>
<td>Glanz</td>
<td>NULL</td>
</tr>
</tbody>
</table>

**Example:** Conflicts between relations

Person: Id, firstname, lastname, male, female
Person: Id, name, gender, birthday
Manager: Id, name, gender, age

**Example:** Conflicts between attributes

Name: firstname, lastname, male, female
Name: name, gender, birthday
Name: firstname, lastname, male, female

**Example:** Conflicts between attributes and attributes

Name: firstname, lastname, male, female
Name: name, gender, birthday
Manager: Id, name, gender, age

**Example:** Conflicts between relations

Person: Id, firstname, lastname, male, female
Person: Id, name, gender, birthday
Manager: Id, name, gender, age
1.1 Structural Heterogeneity

- Normalized vs. denormalized
  - E.g., relational model: Association between entities can be represented using multiple relations and foreign keys or one relation

Example:

```
Person  Name  Address
  |     |     |
   M   S   A
   E   C   Y
```

1.1 Structural Heterogeneity

- Nested vs. flat
  - Association between entities can be represented using nesting or references (previous slides)

Example:

```
Person  Name  {Address
  |     |     |
   M   S   A
   E   C   Y

  |     |     |
   I   T   Y
   Z   P
```

1.1 Structural Heterogeneity

- Problems caused by schema heterogeneity
  - Unified access to multiple schemas or integrate schemas into new schema
    - **Schema level**: schema mapping, model management operators, schema languages
    - **Data Level**: virtual data integration, data exchange, warehousing (ETL)

1.1 Heterogeneity +Autonomy

- Taxonomy of Heterogeneity

1.1 Semantic Heterogeneity

- Semantic Heterogeneity
  - Naming Conflicts
  - Identity Conflicts (Entity resolution)
  - Value Conflicts (Data Fusion)

1.1 Semantic Heterogeneity

- Naming Conflicts
  - Ontological (concepts)
    - Birds vs. Animals
  - Synonyms
    - Surname vs. last name
  - Homonyms
  - Units
    - Gallon vs. liter
  - Values
    - Manager vs. Boss
1.1 Semantic Heterogeneity

• Ontological concepts
  – Relationships between concepts
    • $A = B$ - Equivalence
    • $A \subseteq B$ - Inclusion
    • $A \cap B$ - Overlap
    • $A \neq B$ - Disjunction

Example:
- Equivalence: Human vs Homo sapiens
- Inclusion: Bird vs Animal
- Overlap: Mammal vs aquatic lifeform
- Disjunction: Fish vs Mammal

• Naming concepts (synonyms)
  • Different words with same meaning

Example:
- Person (Title, Name, Age)
- Human (LastName, Age)

• Naming concepts (homonyms)
  • Same words with different meaning

Example:
- Person (Title, Name)
- Movie (Title, Year)

• Naming concepts (units)

Example:
- Person (Title, Name, Salary)
- Person (Title, Name, Salary)

• Identity Conflicts
  – What is an object?
    • E.g., multiple tuples in relational model
  – Central question:
    • Does object A represent the same entity as B
  – This problem has been called
    • Entity resolution
    • Record linkage
    • Deduplication
    • …
1.1 Semantic Heterogeneity

- Identity Conflicts

Example:

IBM, 3010000000, USA
International Business Machines Corporation, 50000

1.1 Semantic Heterogeneity

- Value Conflicts
  - Objects representing the same entities have conflicting values for semantically equivalent attributes
  - We have to identify that these objects are represent the same entity first!
  - Resolving such conflicts require Data Fusion
    - Pick value from conflicting values
    - Numerical methods: e.g., average
    - Preferred value
    - ...

1.1 Autonomy

- How autonomous are data sources
  - One company
    - Can enforce, e.g., schema and software
  - ...
  - The web
    - Website decides
      - Interface
      - Determines access restrictions and limits
      - Availability
      - Format
      - Query restrictions
  - ...

1.2 Data integration tasks

- Cleaning and preparation
- Entity resolution
- Data Fusion
- Schema matching
- Schema mapping
- Query rewrite
- Data translation

1.3 Data integration architectures

- Virtual data integration
- Data Exchange
- Peer-to-peer data integration
- Datawarehousing
- Big Data analytics

1.4 Formal Background

- Query Equivalence
  - Complexity for different query classes
- Query Containment
  - Complexity for different query classes
- Datalog
  - Recursion + Negation
- Integrity Constraints
  - Logical encoding of integrity constraints
- Similarity Measures/Metrics
1.4 Integrity constraints

- You know some types of integrity constraints already
  - Functional dependencies
    • Keys are a special case
  - Foreign keys
    • We have not really formalized that

- Other types are
  - Conditional functional dependencies
    • E.g., used in cleaning
  - Equality-generating dependencies
  - Multi-valued dependencies
  - Tuple-generating dependencies
  - Join dependencies
  - Denial constraints
  - ...

- How to manage all these different types of constraints?
  - Has been shown that these constraints can be expressed in a logical formalism.
  - Formulas which consist of relational and comparison atoms. Variables represent values
    • $R(x, y, z)$
    • $x = y$

- Types of constraints we will use a lot
  - Tuple-generating dependencies ($tgds$)
    • Implication with conjunction of relational atoms
    • Foreign keys and schema mappings (later)
      $\forall \bar{x} : \phi(\bar{x}) \rightarrow \exists \bar{y} : \psi(\bar{x}, \bar{y})$
  - Equality-generating dependencies ($egds$)
    • Generalizes keys, FDs
      $\forall \bar{x} : \phi(\bar{x}) \rightarrow \wedge_{k=1}^{n} x_{i_k} = x_{j_k}$

- What is datalog?
  - Prolog for databases (syntax very similar)
  - A logic-based query language
  - Queries (Program) expressed as set of rules
    $Q(\bar{x}) : - R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n)$.
  - One Q is specified as the answer relation (the relation returned by the query)
1.4 Datalog - Intuition

- A Datalog rule
  \[ Q(\bar{x}) : -R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n). \]
- For all bindings of variables in the right-hand side (RHS) that makes the RHS true (conjunction) return bindings of \( \bar{x} \)

Example

\[ Q(\text{Name}) : \text{Person(\text{Name}, \text{Age})}. \]
Return names of persons

1.4 Datalog - Syntax

- A Datalog program is a set of datalog rules
  - Optionally a distinguished answer predicate
- A Datalog rule is
  \[ Q(\bar{x}) : -R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n). \]
- \( X \)'s are lists of variables and constants
- \( R_i \)'s are relation names
- \( Q \) is a relation name

1.4 Datalog - Terminology

- Left-hand side of a rule is called it's head
- Right-hand side of a rule is called it's body
- Relation are called predicates
- \( R(\bar{x}) \) is called an atom
- An instance \( I \) of a database is the data
- The active domain \( \text{adom}(I) \) of an instance \( I \) is the set of all constants that occur in \( I \)
  \[ Q(\bar{x}) : -R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n). \]

Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>34</td>
</tr>
<tr>
<td>Bob</td>
<td>45</td>
</tr>
</tbody>
</table>

Activate domain
\( \text{adom}(I) = \{ \text{peter, bob, 34, 45} \} \)

1.4 Datalog - Terminology

- Intensional vs. extensional
  - Extensional database (edb)
    - What we usually call database
  - Intensional database (idb)
    - Relations that occur in the head of rules (are populated by the query)
    - Usually we assume that these do not overlap

1.4 Datalog - Safety

- A datalog program is safe if all its rules are safe
- A rule is safe if all variables in \( \bar{x} \) occur in at least one \( x_i \)
  \[ Q(\bar{x}) : -R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n). \]

Example

\[ Q(\text{Name}) : \text{Person(\text{Name}, \text{Age})}. \text{ (safe)} \]
\[ Q(\text{Name}, \text{Sal}) : \text{Person(\text{Name}, \text{Age})}. \text{ (unsafe)} \]
1.4 Datalog - Semantics

• The instance of an idb predicate Q in a datalog program for an edb instance I contains all facts that can be derived by applying rules with Q in the head

• A rule derives a fact Q(c) if we can find a binding of variables of the rule to constants from \( \text{adom}(I) \) such that x is bound to c and the body is true

\[
Q(x) : \neg R_1(x_1), \ldots, \neg R_n(x_n).
\]

1.4 Datalog

• Different flavors of datalog
  – Conjunctive query
  • Only one rule
  • Expressible as Select-project-join (SPJ) query in relational algebra
  – Union of conjunctive queries
  • Also allow union
  • SPJ + set union in relational algebra
  • Rules with the same head in Datalog
  – Conjunctive queries with inequalities
  • Also allow inequivalences, e.g., <

1.4 Datalog - Semantics

Example

\[ Q(N) : \text{Person}(N,A). \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>peter</td>
<td>34</td>
</tr>
<tr>
<td>bob</td>
<td>34</td>
</tr>
</tbody>
</table>

Activate domain

\( \text{adom}(I) = \{ \text{peter}, \text{bob}, 34 \} \)

Example

Relation \( \text{hops}(A,B) \) storing edges of a graph.

\[ Q_{2\text{hop}}(x,z) : \text{hop}(x,y), \text{hop}(y,z). \]
\[ Q_{\text{reach}}(x,y) : \text{hop}(x,y). \]
\[ Q_{\text{reach}}(x,z) : Q_{\text{reach}}(x,y), Q_{\text{reach}}(y,z). \]
\[ Q_{\text{node}}(x) : \text{hop}(x,y). \]
\[ Q_{\text{node}}(x) : \text{hop}(y,x). \]
1.4 Datalog

**Example**

Relation \( \text{hops}(A, B) \) storing edges of a graph.

\[ Q_{\text{node}}(x) : \text{hop}(x, y). \]
\[ Q_{\text{node}}(x) : \text{hop}(y, x). \]
\[ Q_{\text{notReach}}(x, y) : Q_{\text{node}}(x) \land Q_{\text{node}}(y) \land \neg Q_{\text{reach}}(x, y). \]

1.4 Containment and Equivalence

**Definition: Query Equivalence**

Query Q is equivalent to Q’ if for every database instance I both queries return the same result.

\[ Q \equiv Q' \iff \forall I : Q(I) = Q'(I) \]

**Definition: Query Containment**

Query Q is contained in query Q’ if for every database instance I the result of Q is contained in the result of Q’.

\[ Q \subseteq Q' \iff \forall I : Q(I) \subseteq Q'(I) \]

1.4 Equivalence

- The problem of checking query equivalence is of different complexity depending on the query language and whether we consider set or bag semantics.

1.4 Equivalence

**Example**

Relation \( \text{hops}(A, B) \) storing edges of a graph.

\[ Q_{\text{2hop}}(x, z) : \text{hop}(x, y) \land \text{hop}(x, z). \]
\[ Q_{\text{up2Hop}}(x, z) : \text{hop}(x, y) \land \text{hop}(x, z). \]
\[ Q_{\text{sym}}(x, y) : \text{hop}(x, y). \]
\[ Q_{\text{sym}}(x, y) : \text{hop}(y, x). \]
\[ Q_{\text{sym2Hop}}(x, y) : Q_{\text{sym}}(x, y) \land Q_{\text{sym}}(y, z). \]

1.4 Containment and Equivalence

**Example**

Relation \( \text{hops}(A, B) \) storing edges of a graph.

\[ Q_{\text{2hop}}(x, z) : \text{hop}(x, y) \land \text{hop}(x, z). \]
\[ Q_{\text{up2Hop}}(x, z) : \text{hop}(x, y) \land \text{hop}(x, z). \]
\[ Q_{\text{sym}}(x, y) : \text{hop}(x, y). \]
\[ Q_{\text{sym}}(x, y) : \text{hop}(y, x). \]
\[ Q_{\text{sym2Hop}}(x, y) : Q_{\text{sym}}(x, y) \land Q_{\text{sym}}(y, z). \]

1.4 Complexity of Eq. and Cont.

**Set semantics**

<table>
<thead>
<tr>
<th>Relational Algebra</th>
<th>Conjunctive Queries (CQ)</th>
<th>Union of Conjunctive Queries (UQO)</th>
<th>Monotone Queries/ CQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Evaluation</td>
<td>PSPACE-complete</td>
<td>NP-complete</td>
<td>NP-complete</td>
</tr>
<tr>
<td>(Combined Complexity)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query Evaluation</td>
<td>LOGSPACE</td>
<td>LOGSPACE</td>
<td>LOGSPACE</td>
</tr>
<tr>
<td>(Data Complexity)</td>
<td>(that means in P)</td>
<td>(that means in P)</td>
<td>(that means in P)</td>
</tr>
<tr>
<td>Query Equivalence</td>
<td>Undecidable</td>
<td>NP-complete</td>
<td>( \Pi_2 )-complete</td>
</tr>
<tr>
<td>Query Containment</td>
<td>Undecidable</td>
<td>NP-complete</td>
<td>( \Pi_2 )-complete</td>
</tr>
</tbody>
</table>
### 1.4 Complexity of Eq. and Cont.

<table>
<thead>
<tr>
<th>Bag semantics</th>
<th>Relational Algebra</th>
<th>Conjunctive Queries (CQ)</th>
<th>Union of Conjunctive Queries (UCQ)</th>
<th>Monotone Queries/CDQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>Undecidable</td>
<td>Equivalent to graph isomorphism</td>
<td>It is in PSPACE, lower-bound unknown</td>
<td></td>
</tr>
<tr>
<td>Containment</td>
<td>Undecidable</td>
<td>Open Problem</td>
<td>Undecidable</td>
<td>( \Pi_2 )-complete</td>
</tr>
</tbody>
</table>

### 1.4 Containment Mappings

- NP-completeness for set semantics CQ and UCQ for the containment, evaluation, and equivalence problems is based on reducing these problems to the same problem
  - [Chandra & Merlin, 1977]
- Notational Conventions:
  - \( \text{head}(Q) \) = variables in head of query \( Q \)
  - \( \text{body}(Q) \) = atoms in body of \( Q \)
  - \( \text{vars}(Q) \) = all variable in \( Q \)

### 1.4 Boolean Conjunctive Queries

- A conjunctive query is boolean if the head does not have any variables
  - \( Q() :\ hop(x,y), hop(y,z) \)
  - We will use \( Q :\ … \) as a convention for \( Q() :\ … \)
  - What is the result of a boolean query
    - Empty result \{\}, e.g., no \( hop(x,y), hop(y,z) \)
    - If there are tuples matching the body, then a tuple with zero attributes is returned \{\} |
  - \( \rightarrow \) We interpret \{\} as false and \{\} as true
- Boolean query is essentially an existential check

### 1.4 Boolean Conjunctive Queries

- **BCQ in SQL**

  **Example**

  ```sql
  SELECT CASE WHEN EXISTS (SELECT * FROM hop)
    THEN 1 ELSE 0
  END AS x
  FROM dual;
  ```

  **Notes:**
  - Oracle and DB2 FROM not optional
  - Oracle has no boolean datatype

- **BCQ in SQL**

  **Example**

  ```sql
  Q : hop(x,y), hop(y,z)
  SELECT EXISTS (SELECT * FROM hop)
  ```

  **Note:** in Oracle and DB2 we need a FROM clause
1.4 Containment Mappings

- How to check for containment of CQs (set)

**Definition: Variable Mapping**
A variable mapping \( \psi \) from query \( Q \) to query \( Q' \) maps the variables of \( Q \) to constants or variables from \( Q' \).

**Definition: Containment Mapping**
A containment mapping from query \( Q \) to \( Q' \) is a variable mapping \( \psi \) such that:

\[
\psi(\text{head}(Q)) = \text{head}(Q')
\]

\[
\forall R(x_i) \in \text{body}(Q) : \psi(x_i) \in \text{body}(Q')
\]

Theorem: Containment Mapping and Query Containment

Query \( Q \) is contained in query \( Q' \) if there exists a containment mapping \( \psi \) from \( Q' \) to \( Q \).

**Example**

\[Q_1(u, z): R(u, z) . \]
\[Q_2(x, y): R(x, y) . \]

Do containment mappings exist?

\[Q_1 \rightarrow Q_2: \text{none exists} \]
\[Q_2 \rightarrow Q_1: \psi(x) = a, \psi(y) = b \]
1.4 Containment Background

- It was shown that query evaluation, containment, equivalence as all reducible to homomorphism checking for CQ
  - Canonical conjunctive query $Q^I$ for instance $I$
    - Interpret attribute values as variables
    - The query is a conjunction of all atoms for the tuples
    - $I = \{ hop(a,b), hop(b,c) \} \Rightarrow Q^I : hop(a,b), hop(b,c)$
  - Canonical instance $I^0$ for query $Q$
    - Interpret each conjunct as a tuple
    - Interpret variables as constants
    - $Q : hop(a,a) \Rightarrow I^0 = \{ hop(a,a) \}$

1.4 Containment Mappings

Example

$Q_1() : R(a,b), R(c,b)$.
$Q_2() : R(x,y)$.
$Q_2 \Rightarrow Q_1 : \Psi(x)=a, \Psi(y)=b$

$D=\{R(1,1), R(1,2)\}$
$Q_1(D)=\{(1,1), (1,2)\}$
$\phi(a)=1, \phi(b)=2, \phi(c)=1$
$\Psi\phi(x)=1, \Psi\phi(y)=2$

$\Phi$ is our containment mapping $\Psi$

1.4 Containment Background

- Containment Mapping <-> Containment
- Proof idea (boolean queries)
  - (if direction)
    - Assume we have a containment mapping $Q_1$ to $Q_2$
    - Consider database $D$
    - $Q_2(D)$ is true then we can find a mapping from $\text{vars}(Q_2)$ to $D$
    - Compose this with the containment mapping and prove that this is a result for $Q_1$

Example

$Q_1() : R(a,b), R(c,b)$.
$Q_2() : R(x,y)$.
$Q_2 \Rightarrow Q_1 : \Psi(x)=a, \Psi(y)=b$

$I^{Q_1} = \{(a,b), (c,b)\}$
$Q_2(I^{Q_1})=\{()\}$
$\phi(x)=a, \phi(y)=b$

$\phi$ is our containment mapping $\Psi$

1.4 Containment Background

- If you are not scared and want to know more:
  - Look up Chandra and Merlin's paper(s)
  - The text book provides a more detailed overview of the proof approach
  - Look at the slides from Phokion Kolaitis excellent lecture on database theory
    - [https://classes.soe.ucsc.edu/cmps277/Winter10/](https://classes.soe.ucsc.edu/cmps277/Winter10/)
1.4 Containment Background

• A more intuitive explanation why containment mappings work
  – Variable naming is irrelevant for query results
  – If there is a containment mapping \( Q \) to \( Q' \)
    • Then every condition enforced in \( Q \) is also enforced by \( Q' \)
    • \( Q' \) may enforce additional conditions

1.4 Containment Mappings

Example

\[ Q_1 \text{(} a,b \text{)}: R(a,b), R(c,b). \]
\[ Q_2 \text{(} x,y \text{)}: R(x,y). \]
\[ Q_2 \rightarrow Q_1: \Psi(x) = a, \Psi(y) = b \]

If there exists tuples \( R(a,b) \) and \( R(c,b) \) in \( R \) that make \( Q_1 \) true, then we take \( R(a,b) \) to fulfill \( Q_2 \).

1.4 Similarity Measures

• Problem faced by multiple integration tasks
  – Given two objects, how similar are they
  – E.g., given two attribute names in schema matching, given two values in data fusion/entity resolution, …
1.4 Similarity Measures

**Definition: Similarity Measure**

Function \(d(p,q)\) where \(p\) and \(q\) are objects, that returns a real score with
- Non-negative \(d(p,q) \geq 0\)
- Symmetry \(d(p,q) = d(q,p)\)
- Identity of indiscernibles \(d(p,q) = 0 \iff p = q\)
- Triangle inequality \(d(p,q) + d(q,r) \geq d(p,r)\)

- Interpretation: the lower the score the "more similar" the objects are
- We require \(d(p,p) = 0\), because nothing can be more similar to an object than itself
- Note: often scores are normalized to the range \([0,1]\)

---

**Definition: Metric**

Function \(d(p,q)\) where \(p\) and \(q\) are objects, that returns a real score with
- Non-negative \(d(p,q) \geq 0\)
- Symmetry \(d(p,q) = d(q,p)\)
- Identity of indiscernibles \(d(p,q) = 0 \iff p = q\)
- Triangle inequality \(d(p,q) + d(q,r) \geq d(p,r)\)

- Metric is a stricter definition
- Which of the previous similarity measure is a metric?
  - All of them!

---

**Example**

- **String equality**: \(d(p,q) = 0 \iff p = q\)
- **Euclidian distance**: \(d(p,q) = \sqrt{\sum_{i=1}^{N}(p[i] - q[i])^2}\)
- **Edit distance**: \(d(p,q) = \text{minimum number of single character insertions, deletions, replacements to transform } p \text{ into } q\)

---

**Summary**

- Heterogeneity
  - Types of heterogeneity
  - Why do they arise?
  - Hint at how to address them
- Autonomy
- Data Integration Tasks
- Data Integration Architectures
- Background
  - Datalog + Query equivalence/containment + Similarity + Integrity constraints
<table>
<thead>
<tr>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>0) Course Info</td>
</tr>
<tr>
<td>1) Introduction</td>
</tr>
<tr>
<td>2) Data Preparation and Cleaning</td>
</tr>
<tr>
<td>3) Schema matching and mapping</td>
</tr>
<tr>
<td>4) Virtual Data Integration</td>
</tr>
<tr>
<td>5) Data Exchange</td>
</tr>
<tr>
<td>6) Data Warehousing</td>
</tr>
<tr>
<td>7) Big Data Analytics</td>
</tr>
<tr>
<td>8) Data Provenance</td>
</tr>
</tbody>
</table>