Foundations of Relational Query Languages
Relational Model

• Many ad hoc models before 1970
  − Hard to work with
  − Hard to reason about

• 1970: Relational Model by Edgar Frank Codd
  − Data are stored in relations (or tables)
  − Queried using a declarative language
  − DBMS converts declarative queries into procedural queries that are optimized and executed

• Key Advantages
  − Simple and clean mathematical model (based on logic)
  − Separation of declarative and procedural
Relational Databases

<table>
<thead>
<tr>
<th>Flight</th>
<th>origin</th>
<th>destination</th>
<th>airline</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>LHR</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>EDI</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>GLA</td>
<td>U2</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>VIE</td>
<td>OS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Airport</th>
<th>code</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>Vienna</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>Larnaca</td>
<td></td>
</tr>
<tr>
<td>GLA</td>
<td>Glasgow</td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td>Edinburgh</td>
<td></td>
</tr>
</tbody>
</table>
Relational Databases

**Flight**

<table>
<thead>
<tr>
<th>Flight</th>
<th>origin</th>
<th>destination</th>
<th>airline</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>LHR</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>EDI</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>GLA</td>
<td>U2</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>VIE</td>
<td>OS</td>
<td></td>
</tr>
</tbody>
</table>

**Airport**

<table>
<thead>
<tr>
<th>Airport</th>
<th>code</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td></td>
<td>Vienna</td>
</tr>
<tr>
<td>LHR</td>
<td></td>
<td>London</td>
</tr>
<tr>
<td>LGW</td>
<td></td>
<td>London</td>
</tr>
<tr>
<td>LCA</td>
<td></td>
<td>Larnaca</td>
</tr>
<tr>
<td>GLA</td>
<td></td>
<td>Glasgow</td>
</tr>
<tr>
<td>EDI</td>
<td></td>
<td>Edinburgh</td>
</tr>
</tbody>
</table>
Relational Databases

Relations

<table>
<thead>
<tr>
<th>Flight</th>
<th>origin</th>
<th>destination</th>
<th>airline</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>LHR</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>EDI</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>GLA</td>
<td>U2</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>VIE</td>
<td>OS</td>
<td></td>
</tr>
</tbody>
</table>

Tuples

<table>
<thead>
<tr>
<th>Airport</th>
<th>code</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>Vienna</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>Larnaca</td>
<td></td>
</tr>
<tr>
<td>GLA</td>
<td>Glasgow</td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td>Edinburgh</td>
<td></td>
</tr>
</tbody>
</table>

Constants

VIE, LHR, ...

BA, U2, ...

Vienna, London, ...

Relational atoms

Flight(LHR, EDI, BA)
Airport(LGW, London)
### Querying: Relational Algebra

#### List all the airlines

<table>
<thead>
<tr>
<th>Flight</th>
<th>origin</th>
<th>destination</th>
<th>airline</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>LHR</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>EDI</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>GLA</td>
<td>U2</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>VIE</td>
<td>OS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Airport</th>
<th>code</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>Vienna</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>Larnaca</td>
<td></td>
</tr>
<tr>
<td>GLA</td>
<td>Glasgow</td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td>Edinburgh</td>
<td></td>
</tr>
</tbody>
</table>
Querying: Relational Algebra

List all the airlines

\[ \pi_{\text{airline}} \text{ Flight} \]

<table>
<thead>
<tr>
<th>Flight</th>
<th>origin</th>
<th>destination</th>
<th>airline</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>LHR</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>EDI</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>GLA</td>
<td>U2</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>VIE</td>
<td>OS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Airport</th>
<th>code</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>Vienna</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>Larnaca</td>
<td></td>
</tr>
<tr>
<td>GLA</td>
<td>Glasgow</td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td>Edinburgh</td>
<td></td>
</tr>
</tbody>
</table>
List the codes of the airports in London

<table>
<thead>
<tr>
<th>Flight</th>
<th>origin</th>
<th>destination</th>
<th>airline</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>LHR</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>EDI</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>GLA</td>
<td>U2</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>VIE</td>
<td>OS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Airport</th>
<th>code</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>Vienna</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>Larnaca</td>
<td></td>
</tr>
<tr>
<td>GLA</td>
<td>Glasgow</td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td>Edinburgh</td>
<td></td>
</tr>
</tbody>
</table>
List the codes of the airports in London

\[ \pi_{\text{code}} (\sigma_{\text{city}='\text{London'}} \text{ Airport}) \]

\{LHR, LGW\}
Querying: Relational Algebra

List the airlines that fly directly from London to Glasgow

<table>
<thead>
<tr>
<th>Flight</th>
<th>origin</th>
<th>destination</th>
<th>airline</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>LHR</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>EDI</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>GLA</td>
<td>U2</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>VIE</td>
<td>OS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Airport</th>
<th>code</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>Vienna</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>Larnaca</td>
<td></td>
</tr>
<tr>
<td>GLA</td>
<td>Glasgow</td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td>Edinburgh</td>
<td></td>
</tr>
</tbody>
</table>
List the airlines that fly directly from London to Glasgow

\[ \pi_{\text{airline}} ((\text{Flight} \bowtie_{\text{origin}=\text{code}} (\sigma_{\text{city}='London'} \text{ Airport})) \bowtie_{\text{destination}=\text{code}} (\sigma_{\text{city}='Glasgow'} \text{ Airport})) \]
Querying: Relational Algebra

List the airlines that fly directly from London to Glasgow

$$\pi_{\text{airline}} \left( \left( \text{Flight} \bowtie_{\text{origin}=\text{code}} \left( \sigma_{\text{city}='\text{London'} \text{ Airport}} \right) \bowtie_{\text{destination}=\text{code}} \left( \sigma_{\text{city}='\text{Glasgow'} \text{ Airport}} \right) \right) \right)$$

defines the auxiliary relation Aux
Relational Algebra

- Selection: $\sigma$
- Projection: $\pi$
- Cross product: $\times$
- Natural join: $\Join$
- Rename: $\rho$
- Difference: $-$
- Union: $\cup$
- Intersection: $\cap$

Formal definitions can be found in any database textbook
List all the airlines

<table>
<thead>
<tr>
<th>Flight</th>
<th>origin</th>
<th>destination</th>
<th>airline</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>LHR</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>EDI</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>GLA</td>
<td>U2</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>VIE</td>
<td>OS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Airport</th>
<th>code</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td></td>
<td>Vienna</td>
</tr>
<tr>
<td>LHR</td>
<td></td>
<td>London</td>
</tr>
<tr>
<td>LGW</td>
<td></td>
<td>London</td>
</tr>
<tr>
<td>LCA</td>
<td></td>
<td>Larnaca</td>
</tr>
<tr>
<td>GLA</td>
<td></td>
<td>Glasgow</td>
</tr>
<tr>
<td>EDI</td>
<td></td>
<td>Edinburgh</td>
</tr>
</tbody>
</table>
List all the airlines

{BA, U2, OS}

\{z \mid \exists x \exists y \text{ Flight}(x,y,z)\}
List the codes of the airports in London

<table>
<thead>
<tr>
<th>Flight</th>
<th>origin</th>
<th>destination</th>
<th>airline</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>LHR</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>EDI</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>GLA</td>
<td>U2</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>VIE</td>
<td>OS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Airport</th>
<th>code</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>Vienna</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>Larnaca</td>
<td></td>
</tr>
<tr>
<td>GLA</td>
<td>Glasgow</td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td>Edinburgh</td>
<td></td>
</tr>
</tbody>
</table>
List the codes of the airports in London

\{x | \exists y \text{ Airport}(x,y) \land y = \text{London}\}
List the airlines that fly directly from London to Glasgow

<table>
<thead>
<tr>
<th>Flight</th>
<th>origin</th>
<th>destination</th>
<th>airline</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>LHR</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>EDI</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>GLA</td>
<td>U2</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>VIE</td>
<td>OS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Airport</th>
<th>code</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>VIE</td>
<td>Vienna</td>
</tr>
<tr>
<td>LHR</td>
<td>LHR</td>
<td>London</td>
</tr>
<tr>
<td>LGW</td>
<td>LGW</td>
<td>London</td>
</tr>
<tr>
<td>LCA</td>
<td>LCA</td>
<td>Larnaca</td>
</tr>
<tr>
<td>GLA</td>
<td>GLA</td>
<td>Glasgow</td>
</tr>
<tr>
<td>EDI</td>
<td>EDI</td>
<td>Edinburgh</td>
</tr>
</tbody>
</table>
List the airlines that fly directly from London to Glasgow

\{z \mid \exists x \exists y \exists u \exists v \text{ Airport}(x,u) \land u = \text{London} \land \text{Airport}(y,v) \land v = \text{Glasgow} \land \text{Flight}(x,y,z)\}
Domain Relational Calculus

\{x_1,\ldots,x_k \mid \varphi\}

first-order formula with free variables \{x_1,\ldots,x_k\}

But, we can express “problematic” queries, i.e., depend on the domain

\{x \mid \forall y \ R(x,y)\} \quad \{x \mid \neg R(x)\} \quad \{x,y \mid R(x) \lor R(y)\}

...thus, we adopt the active domain semantics – quantified variables range over the active domain, i.e., the constants occurring in the input database
Theorem: The following query languages are equally expressive

- Relational Algebra (RA)
- Domain Relational Calculus (DRC)
- Tuple Relational Calculus (TRC)

Note: Tuple relational calculus is the declarative language introduced by Codd. Domain relational calculus has been introduced later as a formalism closer to first-order logic.
Is Glasgow reachable from Vienna?

Recursive query – not expressible in RA/DRC/TRC (unless we bound the number of intermediate stops)
Complexity of Query Languages

• The goal is to understand the complexity of evaluating a query over a database

• Our main technical tool is complexity theory

• What to measure? Queries may have a large output, and it would be unfair to count the output as “complexity”

• We therefore consider the following decision problems:
  – Query Output Tuple (QOT)
  – Boolean Query Evaluation (BQE)
A Crash Course on Complexity Theory

we are going to recall some fundamental notions from complexity theory that will be heavily used in the context of this course – details can be found in the standard textbooks
Deterministic Turing Machine (DTM) is defined as:

\[ M = (S, \Lambda, \Gamma, \delta, s_0, s_{\text{accept}}, s_{\text{reject}}) \]

- \( S \) is the set of states
- \( \Lambda \) is the input alphabet, not containing the blank symbol \( \square \)
- \( \Gamma \) is the tape alphabet, where \( \square \in \Gamma \) and \( \Lambda \subseteq \Gamma \)
- \( \delta : S \times \Gamma \rightarrow S \times \Gamma \times \{L,R\} \)
- \( s_0 \) is the initial state
- \( s_{\text{accept}} \) is the accept state
- \( s_{\text{reject}} \) is the reject state, where \( s_{\text{accept}} \neq s_{\text{reject}} \)
Deterministic Turing Machine (DTM)

\[ M = (S, \Lambda, \Gamma, \delta, s_0, s_{\text{accept}}, s_{\text{reject}}) \]

\[ \delta(s_1, \alpha) = (s_2, \beta, R) \]

**IF** at some time instant \( \tau \) the machine is in state \( s_1 \), the cursor points to cell \( \kappa \), and this cell contains \( \alpha \)

**THEN** at instant \( \tau + 1 \) the machine is in state \( s_2 \), cell \( \kappa \) contains \( \beta \), and the cursor points to cell \( \kappa + 1 \)
Nondeterministic Turing Machine (NTM)

\[ M = (S, \Lambda, \Gamma, \delta, s_0, s_{\text{accept}}, s_{\text{reject}}) \]

- \( S \) is the set of states
- \( \Lambda \) is the input alphabet, not containing the blank symbol \( \square \)
- \( \Gamma \) is the tape alphabet, where \( \square \in \Gamma \) and \( \Lambda \subseteq \Gamma \)
- \( \delta : S \times \Gamma \rightarrow 2^S \times \Gamma \times \{L,R\} \)
- \( s_0 \) is the initial state
- \( s_{\text{accept}} \) is the accept state
- \( s_{\text{reject}} \) is the reject state, where \( s_{\text{accept}} \neq s_{\text{reject}} \)
Turing Machine Configuration

A perfect description of the machine at a certain point in the computation

\[
\begin{array}{cccccccc}
1 & 0 & 1 & 1 & 0 & 1 & 1 & \square \ \square \\
\end{array}
\]

\[
s
\]

is represented as a string: \textbf{1011s011}

- Initial configuration on input \(w_1,\ldots,w_n\) - \(s_0w_1,\ldots,w_n\)
- Accepting configuration - \(u_1,\ldots,u ks_{\text{accept}}u_{k+1},\ldots,u_{k+m}\)
- Rejecting configuration - \(u_1,\ldots,u ks_{\text{reject}}u_{k+1},\ldots,u_{k+m}\)
Turing Machine Computation

Deterministic

\[ S_0w_1, \ldots, w_n \]

The next configuration is unique

Nondeterministic

\[ S_0w_1, \ldots, w_n \]

Computation tree

Computation path
Deciding a Problem

(recall that an instance of a decision problem $\Pi$ is encoded as a word over a certain alphabet $\Lambda$ – thus, $\Pi$ is a set of words over $\Lambda$, i.e., $\Pi \subseteq \Lambda^*$)

A DTM $M = (S, \Lambda, \Gamma, \delta, s_0, s_{\text{accept}}, s_{\text{reject}})$ decides a problem $\Pi$ if, for every $w \in \Lambda^*$:

- $M$ on input $w$ halts in $s_{\text{accept}}$ if $w \in \Pi$
- $M$ on input $w$ halts in $s_{\text{reject}}$ if $w \notin \Pi$
Deciding a Problem

A NTM $M = (S, \Lambda, \Gamma, \delta, s_0, s_{\text{accept}}, s_{\text{reject}})$ \textit{decides} a problem $\Pi$ if, for every $w \in \Lambda^*$:

- The computation tree of $M$ on input $w$ is finite
- There exists \textbf{at least one} accepting computation path if $w \in \Pi$
- There is \textbf{no} accepting computation path if $w \notin \Pi$
Consider a function $f : \mathbb{N} \rightarrow \mathbb{N}$

\[
\begin{align*}
\text{TIME}(f(n)) &= \{\Pi \mid \Pi \text{ is decided by some DTM in time } O(f(n))\} \\
\text{NTIME}(f(n)) &= \{\Pi \mid \Pi \text{ is decided by some NTM in time } O(f(n))\} \\
\text{SPACE}(f(n)) &= \{\Pi \mid \Pi \text{ is decided by some DTM using space } O(f(n))\} \\
\text{NSPACE}(f(n)) &= \{\Pi \mid \Pi \text{ is decided by some NTM using space } O(f(n))\}
\end{align*}
\]
Complexity Classes

• We can now recall the standard time and space complexity classes:

\[
\begin{align*}
\text{PTIME} & = \bigcup_{k>0} \text{TIME}(n^k) \\
\text{NP} & = \bigcup_{k>0} \text{NTIME}(n^k) \\
\text{EXPTIME} & = \bigcup_{k>0} \text{TIME}(2^{n^k}) \\
\text{NEXPTIME} & = \bigcup_{k>0} \text{NTIME}(2^{n^k}) \\
\text{LOGSPACE} & = \text{SPACE}(\log n) \\
\text{NLOGSPACE} & = \text{NSPACE}(\log n) \\
\text{PSPACE} & = \bigcup_{k>0} \text{SPACE}(n^k) \\
\text{EXPSPACE} & = \bigcup_{k>0} \text{SPACE}(2^{n^k})
\end{align*}
\]

these definitions are relying on two-tape Turing machines with a read-only and a read/write tape

• For every complexity class \( C \) we can define its complementary class

\[
\text{coC} = \{ \Lambda^* \setminus \Pi | \Pi \in C \}
\]
An Alternative Definition for NP

**Theorem:** Consider a problem $\Pi \subseteq \Lambda^*$. The following are equivalent:

- $\Pi \in \text{NP}$
- There is a relation $R \subseteq \Lambda^* \times \Lambda^*$ that is polynomially decidable such that

  $$\Pi = \{ u \mid \text{there exists } w \text{ such that } |w| \leq |u|^k \text{ and } (u,w) \in R \}$$

  $\{xy \in \Lambda^* \mid (x,y) \in R \} \in \text{PTIME}$

**Example:**

$3\text{SAT} = \{ \varphi \mid \varphi \text{ is a 3CNF formula that is satisfiable} \}$

$$= \{ \varphi \mid \varphi \text{ is a 3CNF for which } \exists \text{ assignment } \alpha \text{ such that } |\alpha| \leq |\varphi| \text{ and } (\varphi,\alpha) \in R \}$$

where $R = \{(\varphi,\alpha) \mid \alpha \text{ is a satisfying assignment for } \varphi \} \in \text{PTIME}$
Relationship Among Complexity Classes

LOGSPACE $\subseteq$ NLOGSPACE $\subseteq$ PTIME $\subseteq$ NP, coNP $\subseteq$ PSPACE $\subseteq$ EXPTIME $\subseteq$ NEXPTIME, coNEXPTIME $\subseteq$ ...

Some useful notes:

- For a deterministic complexity class C, coC = C
- coNLOGSPACE = NLOGSPACE
- It is generally believed that PTIME $\neq$ NP, but we don’t know
- PTIME $\subset$ EXPTIME $\Rightarrow$ at least one containment between them is strict
- PSPACE = NPSPACE, EXPSPACE = NEXPSPACE, etc.
- But, we don’t know whether LOGSPACE = NLOGSPACE
Complete Problems

- These are the hardest problems in a complexity class
- A problem that is complete for a class $C$, it is unlikely to belong in a lower class

- A problem $\Pi$ is complete for a complexity class $C$, or simply $C$-complete, if:
  1. $\Pi \in C$
  2. $\Pi$ is $C$-hard, i.e., every problem $\Pi' \in C$ can be efficiently reduced to $\Pi$

  There exists a polynomial time algorithm (resp., logspace algorithm) that computes a function $f$ such that $w \in \Pi' \iff f(w) \in \Pi$ – in this case we write $\Pi' \leq_p \Pi$ (resp., $\Pi' \leq_L \Pi$)

- To show that $\Pi$ is $C$-hard it suffices to reduce some $C$-hard problem $\Pi'$ to it
Some Complete Problems

• **NP-complete**
  - SAT (satisfiability of propositional formulas)
  - Many graph-theoretic problems (e.g., 3-colorability)
  - Traveling salesman
  - etc.

• **PSPACE-complete**
  - Quantified SAT (or simply QSAT)
  - Equivalence of two regular expressions
  - Many games (e.g., Geography)
  - etc.
Back to Query Languages
Complexity of Query Languages

• The goal is to understand the complexity of evaluating a query over a database

• Our main technical tool is complexity theory

• What to measure? Queries may have a large output, and it would be unfair to count the output as “complexity”

• We therefore consider the following decision problems:
  - Query Output Tuple (QOT)
  - Boolean Query Evaluation (BQE)
Complexity of Query Languages

Some useful notation:

- Given a database $D$, and a query $Q$, $Q(D)$ is the answer to $Q$ over $D$
- $\text{adom}(D)$ is the active domain of $D$, i.e., the constants occurring in $D$
- We write $Q/k$ for the fact that the arity of $Q$ is $k \geq 0$

$L$ is some query language; for example, RA, DRC, etc. – we will see several query languages in the context of this course

<table>
<thead>
<tr>
<th>QOT(L)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> a database $D$, a query $Q/k \in L$, a tuple of constants $t \in \text{adom}(D)^k$</td>
</tr>
<tr>
<td><strong>Question:</strong> $t \in Q(D)$?</td>
</tr>
</tbody>
</table>
Complexity of Query Languages

Some useful notation:

- Given a database $D$, and a query $Q$, $Q(D)$ is the answer to $Q$ over $D$
- $\text{adom}(D)$ is the active domain of $D$, i.e., the constants occurring in $D$
- We write $Q/k$ for the fact that the arity of $Q$ is $k \geq 0$

$L$ is some query language; for example, RA, DRC, etc. – we will see several query languages in the context of this course

<table>
<thead>
<tr>
<th>BQE($L$)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> a database $D$, a Boolean query $Q/0 \in L$</td>
</tr>
<tr>
<td><strong>Question:</strong> $Q(D) \neq \emptyset$? (i.e., does $D$ satisfies $Q$?)</td>
</tr>
</tbody>
</table>
Complexity of Query Languages

QOT(L)
Input: a database $D$, a query $Q/k \in L$, a tuple of constants $t \in \text{adom}(D)^k$
Question: $t \in Q(D)$?

BQE(L)
Input: a database $D$, a Boolean query $Q/0 \in L$
Question: $Q(D) \neq \emptyset$? (i.e., does $D$ satisfies $Q$?)

Theorem: $QOT(L) \equiv_L BQE(L)$, where $L \in \{\text{RA}, \text{DRC, TRC}\}$

($\equiv_L$ means logspace-equivalent)
Complexity of Query Languages

(let us show this for domain relational calculus)

**Theorem:** $\text{QOT} (\text{DRC}) \equiv_L \text{BQE} (\text{DRC})$

**Proof:** $(\leq_L)$ Consider a database $D$, a $k$-ary query $Q = \{x_1, \ldots, x_k \mid \varphi\}$, and a tuple $(t_1, \ldots, t_k)$

Let $Q_{\text{bool}} = \{ | \varphi \land x_1 = t_1 \land x_2 = t_2 \land \ldots \land x_k = t_k\}$

Clearly, $(t_1, \ldots, t_k) \in Q(D)$ iff $Q_{\text{bool}} (D) \neq \emptyset$

$(\geq_L)$ Trivial – a Boolean domain RC query is a domain RC query

…henceforth, we focus on the Boolean Query Evaluation problem
Complexity Measures

- **Combined complexity** – both $D$ and $Q$ are part of the input

- **Query complexity** – fixed $D$, input $Q$

- **Data complexity** – input $D$, fixed $Q$

\[ \text{BQE}[D](L) \]

**Input:** a Boolean query $Q \in L$

**Question:** $Q(D) \neq \emptyset$?

\[ \text{BQE}[Q](L) \]

**Input:** a database $D$

**Question:** $Q(D) \neq \emptyset$?
Complexity of RA, DRC, TRC

**Theorem:** For $L \in \{\text{RA, DRC, TRC}\}$ the following hold:

- $\text{BQE}(L)$ is PSPACE-complete (combined complexity)
- $\text{BQE}[D](L)$ is PSPACE-complete, for a fixed database $D$ (query complexity)
- $\text{BQE}[Q](L)$ is in LOGSPACE, for a fixed query $Q \in L$ (data complexity)

**Proof hints:**

- Recursive algorithm that uses polynomial space in $Q$ and logarithmic space in $D$
- Reduction from QSAT (a standard PSPACE-hard problem)
Evaluating (Boolean) DRC Queries

Eval\((D, \varphi)\) – for brevity we write \(\varphi\) instead of \(\{ | \varphi\}\)

- If \(\varphi = R(t_1, \ldots, t_k)\), then \(YES\) iff \(R(t_1, \ldots, t_k) \in D\)
- If \(\varphi = \psi_1 \land \psi_2\), then \(YES\) iff \(Eval(D, \psi_1) = YES\) and \(Eval(D, \psi_2) = YES\)
- If \(\varphi = \neg \psi\), then \(NO\) iff \(Eval(D, \psi) = YES\)
- If \(\varphi = \exists x \psi(x)\), then \(YES\) iff for some \(t \in adom(D)\), \(Eval(D, \psi(t)) = YES\)

**Lemma**: It holds that

- \(Eval(D, \varphi)\) always terminates – in fact, this is trivial
- \(Eval(D, \varphi) = YES\) iff \(Q(D) \neq \emptyset\), where \(Q = \{ | \varphi\}\)
- \(Eval(D, \varphi)\) uses \(O(|\varphi| \cdot \log |\varphi| + |\varphi|^2 \cdot \log |D|)\) space
Theorem: For each $L \in \{\text{RA, DRC, TRC}\}$ the following holds:

- $\text{BQE}(L)$ is PSPACE-complete (combined complexity)
- $\text{BQE}[D](L)$ is PSPACE-complete, for a fixed database $D$ (query complexity)
- $\text{BQE}[Q](L)$ is in LOGSPACE, for a fixed query $Q \in L$ (data complexity)

Proof hints:

- Recursive algorithm that uses polynomial space in $Q$ and logarithmic space in $D$
- Reduction from QSAT (a standard PSPACE-hard problem)
- Actually, $\text{BQE}[Q](L)$ is in $\text{AC}_0 \subset \text{LOGSPACE}$ (a highly parallelizable complexity class defined using Boolean circuits)
Other Important Algorithmic Problems

SAT(L)
Input: a query $Q \in L$
Question: is there a (finite) database $D$ such that $Q(D) \neq \emptyset$?

EQUIV(L)
Input: two queries $Q_1 \in L$ and $Q_2 \in L$
Question: $Q_1 \equiv Q_2$? (i.e., $Q_1(D) = Q_2(D)$ for every (finite) database $D$?)

CONT(L)
Input: two queries $Q_1 \in L$ and $Q_2 \in L$
Question: $Q_1 \subseteq Q_2$? (i.e., $Q_1(D) \subseteq Q_2(D)$ for every (finite) database $D$?)
Other Important Algorithmic Problems

SAT(L)
Input: a query \( Q \in L \)
Question: is there a (finite) database \( D \) such that \( Q(D) \neq \emptyset \)?

EQUIV(L)
Input: two queries \( Q_1, Q_2 \in L \)
Question: \( Q_1 \equiv Q_2 \)? (i.e., \( Q_1(D) = Q_2(D) \) for every (finite) database \( D \))?

these problems are important for optimization purposes

CONT(L)
Input: two queries \( Q_1, Q_2 \in L \)
Question: \( Q_1 \subseteq Q_2 \)? (i.e., \( Q_1(D) \subseteq Q_2(D) \) for every (finite) database \( D \)?)
Other Important Algorithmic Problems

SAT(L)

Input: a query $Q \in L$

Question: is there a (finite) database $D$ such that $Q(D) \neq \emptyset$?

- If the answer is no, then the input query $Q$ makes no sense
- Query evaluation becomes trivial – the answer is always NO!
Other Important Algorithmic Problems

EQUIV(L)

Input: two queries $Q_1 \in L$ and $Q_2 \in L$

Question: $Q_1 \equiv Q_2$? (i.e., $Q_1(D) = Q_2(D)$ for every (finite) database $D$?)

- Replace a query $Q_1$ with a query $Q_2$ that is easier to evaluate
- But, we have to be sure that $Q_1(D) = Q_2(D)$ for every database $D$
Other Important Algorithmic Problems

CONT(L)

Input: two queries $Q_1 \in L$ and $Q_2 \in L$

Question: $Q_1 \subseteq Q_2$? (i.e., $Q_1(D) \subseteq Q_2(D)$ for every (finite) database $D$?)

- Approximate a query $Q_1$ with a query $Q_2$ that is easier to evaluate
- But, we have to be sure that $Q_2(D) \subseteq Q_1(D)$ for every database $D$
SAT is Undecidable

**Theorem:** For $L \in \{RA, DRC, TRC\}$, SAT($L$) is undecidable

**Proof hint:** By reduction from the halting problem.

Given a Turing machine $M$, we can construct a query $Q_M \in L$ such that:

$M$ halts on the empty string $\iff$ there exists a database $D$ such that $Q(D) \neq \emptyset$

**Note:** Actually, this result goes back to the 1950 when Boris A. Trakhtenbrot proved that the problem of deciding whether a first-order sentence has a finite model is undecidable.
EQUIV and CONT are Undecidable

An easy consequence of the fact that SAT is undecidable is that:

Theorem: For \( L \in \{RA, DRC, TRC\} \), \( \text{EQUIV}(L) \) and \( \text{CONT}(L) \) are undecidable

Proof: By reduction from the complement of \( \text{SAT}(L) \)

- Consider a query \( Q \in L \) – i.e., an instance of \( \text{SAT}(L) \)
- Let \( Q_{\perp} \) be a query that is trivially unsatisfiable, i.e., \( Q_{\perp}(D) = \emptyset \) for every \( D \)
- For example, when \( L = DRC \), \( Q_{\perp} \) can be the query \( \{ | \exists x \ R(x) \land \neg R(x) \} \)
- Clearly, \( Q \) is unsatisfiable \( \iff Q \equiv Q_{\perp} \) (or even \( Q \subseteq Q_{\perp} \))
Recap

- The main languages for querying relational databases are:
  - Relational Algebra (RA)
  - Domain Relational Calculus (DRC)
  - Tuple Relational Calculus (TRC)

  \[
  \text{RA} = \text{DRC} = \text{TRC}
  \]

  (under the active domain semantics)

- Evaluation is decidable, and highly tractable in data complexity
  - Foundations of the database industry
  - The core of SQL is equally expressive to RA/DRC/TRC

- Satisfiability, equivalence and containment are undecidable
  - Perfect query optimization is impossible
A Crucial Question

Are there interesting sublanguages of RA/DRC/TRC for which satisfiability, equivalence and containment are decidable?

### Conjunctive Queries

- \{\sigma,\pi,\bowtie\}-fragment of relational algebra
- relational calculus without \(\neg, \forall, \vee\)
- simple SELECT-FROM-WHERE SQL queries
  (only AND and equality in the WHERE clause)
Syntax of Conjunctive Queries (CQ)

\[ Q(x) := \exists y (R_1(v_1) \land \ldots \land R_m(v_m)) \]

- \( R_i \) (1 \( \leq \) i \( \leq \) m) are relations
- \( x, y, v_1, \ldots, v_m \) are tuples of variables
- each variable mentioned in \( v_i \) (1 \( \leq \) i \( \leq \) m) appears either in \( x \) or \( y \)
- the variables in \( x \) are free called distinguished variables

It is very convenient to see conjunctive queries as rule-based queries of the form

\[ Q(x) := R_1(v_1), \ldots, R_m(v_m) \]

this is called the body of \( Q \) that can be seen as a set of atoms
Conjunctive Queries: Example 1

List all the airlines

List of flights:

<table>
<thead>
<tr>
<th>Flight</th>
<th>origin</th>
<th>destination</th>
<th>airline</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>LHR</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>EDI</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>GLA</td>
<td>U2</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>VIE</td>
<td>OS</td>
<td></td>
</tr>
</tbody>
</table>

List of airports:

<table>
<thead>
<tr>
<th>Airport</th>
<th>code</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>Vienna</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>Larnaca</td>
<td></td>
</tr>
<tr>
<td>GLA</td>
<td>Glasgow</td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td>Edinburgh</td>
<td></td>
</tr>
</tbody>
</table>

\[ \pi_{\text{airline}} \text{ Flight} \]

\[ \{ z \mid \exists x \exists y \text{ Flight}(x, y, z) \} \]
Conjunctive Queries: Example 2

List the codes of the airports in London

<table>
<thead>
<tr>
<th>Flight</th>
<th>origin</th>
<th>destination</th>
<th>airline</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>LHR</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>EDI</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>GLA</td>
<td>U2</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>VIE</td>
<td>OS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Airport</th>
<th>code</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>Vienna</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>Larnaca</td>
<td></td>
</tr>
<tr>
<td>GLA</td>
<td>Glasgow</td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td>Edinburgh</td>
<td></td>
</tr>
</tbody>
</table>

\[ \pi_{\text{code}} \left( \sigma_{\text{city='London'}} \text{ Airport} \right) \]

\[ \{ x \mid \exists y \text{ Airport}(x,\text{London}) \land y = \text{London} \} \]

\[ Q(x) \ :- \ \text{Airport}(x,y), \ y = \text{London} \]
Conjunctive Queries: Example 2

List the codes of the airports in London

<table>
<thead>
<tr>
<th>Flight</th>
<th>origin</th>
<th>destination</th>
<th>airline</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>LHR</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>EDI</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>GLA</td>
<td>U2</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>VIE</td>
<td>OS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Airport</th>
<th>code</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>VIE</td>
<td>Vienna</td>
</tr>
<tr>
<td>LHR</td>
<td>LHR</td>
<td>London</td>
</tr>
<tr>
<td>LGW</td>
<td>LGW</td>
<td>London</td>
</tr>
<tr>
<td>LCA</td>
<td>LCA</td>
<td>Larnaca</td>
</tr>
<tr>
<td>GLA</td>
<td>GLA</td>
<td>Glasgow</td>
</tr>
<tr>
<td>EDI</td>
<td>EDI</td>
<td>Edinburgh</td>
</tr>
</tbody>
</table>

\[ \pi_{\text{code}} (\sigma_{\text{city}=\text{London}} \text{ Airport}) \]

\[ \{x \mid \exists y \text{ Airport}(x,\text{London}) \land y = \text{London}\} \]

\[ Q(x) \ :- \text{ Airport}(x,\text{London}) \]
Conjunctive Queries: Example 3

List the airlines that fly directly from London to Glasgow

\[ \pi_{\text{airline}} \left( \left( \text{Flight} \bowtie_{\text{origin}=\text{code}} (\sigma_{\text{city}=\text{'London'}} \text{ Airport})) \bowtie_{\text{destination}=\text{code}} (\sigma_{\text{city}=\text{'Glasgow'}} \text{ Airport}) \right) \right) \]

\{z \mid \exists x \exists y \exists u \exists v \text{ Airport}(x,u) \land u = \text{London} \land \text{ Airport}(y,v) \land v = \text{Glasgow} \land \text{Flight}(x,y,z)\}
List the airlines that fly directly from London to Glasgow

<table>
<thead>
<tr>
<th>Flight</th>
<th>origin</th>
<th>destination</th>
<th>airline</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIE</td>
<td>LHR</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LHR</td>
<td>EDI</td>
<td>BA</td>
<td></td>
</tr>
<tr>
<td>LGW</td>
<td>GLA</td>
<td>U2</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td>VIE</td>
<td>OS</td>
<td></td>
</tr>
</tbody>
</table>

\[ Q(z) \vdash Airport(x,\text{London}), Airport(y,\text{Glasgow}), \text{Flight}(x,y,z) \]
Homomorphism

• Semantics of conjunctive queries via the key notion of homomorphism

• A substitution from a set of symbols $S$ to a set of symbols $T$ is a function $h : S \rightarrow T$
i.e., $h$ is a set of mappings of the form $s \rightarrow t$, where $s \in S$ and $t \in T$

• A homomorphism from a set of atoms $A$ to a set of atoms $B$ is a substitution $h : \text{terms}(A) \rightarrow \text{terms}(B)$ such that:
  1. $t$ is a constant $\Rightarrow h(t) = t$
  2. $R(t_1,\ldots,t_k) \in A \Rightarrow h(R(t_1,\ldots,t_k)) = R(h(t_1),\ldots,h(t_k)) \in B$

$(\text{terms}(A) = \{t \mid t \text{ is a variable or constant that occurs in } A\})$
Exercise: Find the Homomorphisms

\[ S_1 = \{P(x,y), P(y,z), P(z,x)\} \]
\[ S_2 = \{P(x,x)\} \]
\[ S_3 = \{P(x,y), P(y,x), P(y,y)\} \]
\[ S_4 = \{P(x,y), P(y,x)\} \]
\[ S_5 = \{P(x,y), P(y,z), P(z,w)\} \]
Exercise: Find the Homomorphisms

\[ S_5 = \{ P(x,y), P(y,z), P(z,w) \} \]

\[ S_1 = \{ P(x,y), P(y,z), P(z,x) \} \]

\[ S_2 = \{ P(x,x) \} \]

\[ S_3 = \{ P(x,y), P(y,x), P(y,y) \} \]

\[ S_4 = \{ P(x,y), P(y,x) \} \]
Semantics of Conjunctive Queries

• A match of a conjunctive query \( Q(x_1, \ldots, x_k) :- \text{body} \) in a database \( D \) is a homomorphism \( h \) such that \( h(\text{body}) \subseteq D \).

• The answer to \( Q(x_1, \ldots, x_k) :- \text{body} \) over \( D \) is the set of k-tuples
  \[
  Q(D) := \{(h(x_1), \ldots, h(x_k)) \mid h \text{ is a match of } Q \text{ in } D\}
  \]

• The answer consists of the witnesses for the distinguished variables of \( Q \).
Conjunctive Queries: Example

List the airlines that fly directly from London to Glasgow

Q(z) :- Airport(x,London), Airport(y,Glasgow), Flight(x,y,z)

{ x → LGW, y → GLA, z → U2 }
Complexity of CQ

**Theorem:** It holds that:

- $\text{BQE}(\mathbf{CQ})$ is NP-complete (combined complexity)
- $\text{BQE}[D](\mathbf{CQ})$ is NP-complete, for a fixed database $D$ (query complexity)
- $\text{BQE}[Q](\mathbf{CQ})$ is in LOGSPACE, for a fixed query $Q \in \mathbf{CQ}$ (data complexity)

**Proof:**

**(NP-membership)** Consider a database $D$, and a Boolean CQ $Q : - \text{body}$

Guess a substitution $h : \text{terms(body)} \rightarrow \text{terms}(D)$

Verify that $h$ is a match of $Q$ in $D$, i.e., $h(\text{body}) \subseteq D$

**(NP-hardness)** Reduction from 3-colorability

**(LOGSPACE-membership)** Inherited from $\text{BQE}[Q](\mathbf{DRC})$ – in fact, in $\mathbf{AC}_0$
NP-hardness

(NP-hardness) Reduction from 3-colorability

3COL

**Input:** an undirected graph \( G = (V,E) \)

**Question:** is there a function \( c : \{\text{Red,Green,Blue}\} \rightarrow V \) such that

\[(v,u) \in E \Rightarrow c(v) \neq c(u)\]?

**Lemma:** \( G \) is 3-colorable \( \iff \) \( G \) can be mapped to \( K_3 \), i.e., \( G \rightarrow K_3 \)

therefore, \( G \) is 3-colorable \( \iff \) there is a match of \( Q_G \) in \( D = \{E(x,y),E(y,z),E(z,x)\} \)

\[\iff Q_G(D) \neq \emptyset \]

the Boolean CQ that represents \( G \)
Complexity of CQ

**Theorem:** It holds that:

- BQE(CQ) is NP-complete (combined complexity)
- BQE[D](CQ) is NP-complete, for a fixed database $D$ (query complexity)
- BQE[Q](CQ) is in LOGSPACE, for a fixed query $Q \in \text{CQ}$ (data complexity)

**Proof:**

**(NP-membership)** Consider a database $D$, and a Boolean CQ $Q : \text{body}$

Guess a substitution $h : \text{terms(body)} \rightarrow \text{terms}(D)$

Verify that $h$ is a match of $Q$ in $D$, i.e., $h(\text{body}) \subseteq D$

**(NP-hardness)** Reduction from 3-colorability

**(LOGSPACE-membership)** Inherited from BQE[Q](DRC) – in fact, in AC$_0$
What About Optimization of CQs?

**SAT(CQ)**

**Input:** a query $Q \in \text{CQ}$

**Question:** is there a (finite) database $D$ such that $Q(D) \neq \emptyset$?

**EQUIV(CQ)**

**Input:** two queries $Q_1 \in \text{CQ}$ and $Q_2 \in \text{CQ}$

**Question:** $Q_1 \equiv Q_2$? (i.e., $Q_1(D) = Q_2(D)$ for every (finite) database $D$?)

**CONT(CQ)**

**Input:** two queries $Q_1 \in \text{CQ}$ and $Q_2 \in \text{CQ}$

**Question:** $Q_1 \subseteq Q_2$? (i.e., $Q_1(D) \subseteq Q_2(D)$ for every (finite) database $D$?)
Canonical Database

• Convert a conjunctive query \( Q \) into a database \( D[Q] \) – the canonical database of \( Q \)

• Given a conjunctive query of the form \( Q(x) :- \) body, \( D[Q] \) is obtained from body by replacing each variable \( x \) with a new constant \( c(x) = x \)

• E.g., given \( Q(x,y) :- R(x,y), P(y,z,w), R(z,x) \), then \( D[Q] = \{R(x,y), P(y,z,w), R(z,x)\} \)

• Note: The mapping \( c : \{\text{variables in body}\} \rightarrow \{\text{new constants}\} \) is a bijection, where \( c(\text{body}) = D[Q] \) and \( c^{-1}(D[Q]) = \text{body} \)
Satisfiability of CQs

\textbf{SAT(CQ)}

\textbf{Input:} a query $Q \in \text{CQ}$

\textbf{Question:} is there a (finite) database $D$ such that $Q(D) \neq \emptyset$?

\textbf{Theorem:} A query $Q \in \text{CQ}$ is always satisfiable; thus, SAT(CQ) $\in O(1)$-time

\textbf{Proof:} Due to its canonical database – $Q(D[Q]) \neq \emptyset$
Equivalence and Containment of CQs

**EQUIV(CQ)**

**Input:** two queries $Q_1 \in \text{CQ}$ and $Q_2 \in \text{CQ}$

**Question:** $Q_1 \equiv Q_2$? (i.e., $Q_1(D) = Q_2(D)$ for every (finite) database $D$?)

**CONT(CQ)**

**Input:** two queries $Q_1 \in \text{CQ}$ and $Q_2 \in \text{CQ}$

**Question:** $Q_1 \subseteq Q_2$? (i.e., $Q_1(D) \subseteq Q_2(D)$ for every (finite) database $D$?)

$Q_1 \equiv Q_2 \iff Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$

$Q_1 \subseteq Q_2 \iff Q_1 \equiv (Q_1 \land Q_2)$

...thus, we can safely focus on CONT(CQ)
Homomorphism Theorem

A query homomorphism from $Q_1(x_1,\ldots,x_k) :- \text{body}_1$ to $Q_2(y_1,\ldots,y_k) :- \text{body}_2$ is a substitution $h : \text{terms}(\text{body}_1) \rightarrow \text{terms}(\text{body}_2)$ such that:

1. $h$ is a homomorphism from body$_1$ to body$_2$
2. $(h(x_1),\ldots,h(x_k)) = (y_1,\ldots,y_k)$

**Homomorphism Theorem:** Let $Q_1$ and $Q_2$ be conjunctive queries. It holds that:

$Q_1 \subseteq Q_2 \iff$ there exists a query homomorphism from $Q_2$ to $Q_1$
Homomorphism Theorem: Example

\[ Q_1(x,y) \ :- \ R(x,z), S(z,z), R(z,y) \]

\[ Q_2(a,b) \ :- \ R(a,c), S(c,d), R(d,b) \]

We expect that \( Q_1 \subseteq Q_2 \). Why?
Homomorphism Theorem: Example

\[ Q_1(x,y) \rightarrow R(x,z), S(z,z), R(z,y) \]

\[ Q_2(a,b) \rightarrow R(a,c), S(c,d), R(d,b) \]

\[ h = \{a \rightarrow x, b \rightarrow y, c \rightarrow z, d \rightarrow z\} \]

- h is a query homomorphism from \( Q_2 \) to \( Q_1 \) \( \Rightarrow \) \( Q_1 \subseteq Q_2 \)

- But, there is no homomorphism from \( Q_1 \) to \( Q_2 \) \( \Rightarrow \) \( Q_1 \subset Q_2 \)
Homomorphism Theorem: Proof

Assume that $Q_1(x_1,...,x_k) :- \text{body}_1$ and $Q_2(y_1,...,y_k) :- \text{body}_2$

$(\Rightarrow) \ Q_1 \subseteq Q_2 \Rightarrow$ there exists a query homomorphism from $Q_2$ to $Q_1$

- Clearly, $(c(x_1),...,c(x_k)) \in Q_1(D[Q_1])$ – recall that $D[Q_1] = c(\text{body}_1)$
- Since $Q_1 \subseteq Q_2$, we conclude that $(c(x_1),...,c(x_k)) \in Q_2(D[Q_1])$
- Therefore, there exists a homomorphism $h$ such that $h(\text{body}_2) \subseteq D[Q_1] = c(\text{body}_1)$ and $h((y_1,...,y_k)) = (c(x_1),...,c(x_k))$
- By construction, $c^{-1}(c(\text{body}_1)) = \text{body}_1$ and $c^{-1}((c(x_1),...,c(x_k))) = (x_1,...,x_k)$
- Therefore, $c^{-1} \circ h$ is a query homomorphism from $Q_2$ to $Q_1
Homomorphism Theorem: Proof

Assume that $Q_1(x_1,\ldots,x_k) :- \text{body}_1$ and $Q_2(y_1,\ldots,y_k) :- \text{body}_2$

$(\Leftarrow) \ Q_1 \subseteq Q_2 \iff$ there exists a query homomorphism from $Q_2$ to $Q_1$

- Consider a database $D$, and a tuple $t$ such that $t \in Q_1(D)$
- We need to show that $t \in Q_2(D)$
- Clearly, there exists a homomorphism $g$ such that $g(\text{body}_1) \subseteq D$ and $g((x_1,\ldots,x_k)) = t$
- By hypothesis, there exists a query homomorphism $h$ from $Q_2$ to $Q_1$
- Therefore, $g(h(\text{body}_2)) \subseteq D$ and $g(h((y_1,\ldots,y_k))) = t$, which implies that $t \in Q_2(D)$
Existence of a Query Homomorphism

**Theorem:** Let $Q_1$ and $Q_2$ be conjunctive queries. The problem of deciding whether there exists a query homomorphism from $Q_2$ to $Q_1$ is NP-complete.

**Proof:**

(NP-membership) Guess a substitution, and verify that is a query homomorphism.

(NP-hardness) Straightforward reduction from $\text{BQE}(\text{CQ})$.

By applying the homomorphism theorem we get that:

**Corollary:** $\text{EQUIV(CQ)}$ and $\text{CONT(CQ)}$ are NP-complete.
Recap

L ∈ \{RA, DRC, TRC\}

UNDECIDABLE

SAT(L)

BQE(L) (combined, query)

QOT(L)

EQUIV(L)

CONT(L)

BQE(CQ) (combined, query)

QOT(CQ)

EQUIV(CQ)

CONT(CQ)

BQE(L) (data)

QOT(L)

SAT(CQ)

QOT(CQ)

BQE(CQ) (combined, query)

EQUIV(CQ)

CONT(CQ)

BQE(L) (data)

QOT(L)

SAT(CQ)

QOT(CQ)

BQE(CQ) (combined, query)

EQUIV(CQ)

CONT(CQ)

BQE(L) (data)

QOT(L)

SAT(CQ)

QOT(CQ)

BQE(CQ) (combined, query)

EQUIV(CQ)

CONT(CQ)

BQE(L) (data)

QOT(L)

SAT(CQ)

QOT(CQ)

BQE(CQ) (combined, query)

EQUIV(CQ)

CONT(CQ)

BQE(L) (data)

QOT(L)

SAT(CQ)

QOT(CQ)

BQE(CQ) (combined, query)

EQUIV(CQ)

CONT(CQ)

BQE(L) (data)

QOT(L)

SAT(CQ)

QOT(CQ)

BQE(CQ) (combined, query)

EQUIV(CQ)

CONT(CQ)

BQE(L) (data)

QOT(L)

SAT(CQ)

QOT(CQ)

BQE(CQ) (combined, query)

EQUIV(CQ)

CONT(CQ)

BQE(L) (data)

QOT(L)

SAT(CQ)

QOT(CQ)

BQE(CQ) (combined, query)

EQUIV(CQ)

CONT(CQ)

BQE(L) (data)

QOT(L)

SAT(CQ)

QOT(CQ)

BQE(CQ) (combined, query)

EQUIV(CQ)

CONT(CQ)

BQE(L) (data)

QOT(L)

SAT(CQ)

QOT(CQ)

BQE(CQ) (combined, query)

EQUIV(CQ)

CONT(CQ)

BQE(L) (data)

QOT(L)

SAT(CQ)

QOT(CQ)

BQE(CQ) (combined, query)

EQUIV(CQ)

CONT(CQ)

BQE(L) (data)

QOT(L)

SAT(CQ)

QOT(CQ)

BQE(CQ) (combined, query)

EQUIV(CQ)

CONT(CQ)

BQE(L) (data)

QOT(L)

SAT(CQ)

QOT(CQ)

BQE(CQ) (combined, query)

EQUIV(CQ)

CONT(CQ)

BQE(L) (data)

QOT(L)

SAT(CQ)

QOT(CQ)

BQE(CQ) (combined, query)

EQUIV(CQ)

CONT(CQ)

BQE(L) (data)

QOT(L)

SAT(CQ)

QOT(CQ)

BQE(CQ) (combined, query)

EQUIV(CQ)

CONT(CQ)

BQE(L) (data)
Minimizing Conjunctive Queries

- **Goal**: minimize the number of joins in a query

- A conjunctive query $Q_1$ is **minimal** if there is no conjunctive query $Q_2$ such that:
  1. $Q_1 \equiv Q_2$
  2. $Q_2$ has fewer atoms than $Q_1$

- The task of **CQ minimization** is, given a conjunctive query $Q$, to compute a minimal one that is equivalent to $Q$
Minimization by Deletion

By exploiting the homomorphism theorem we can show the following:

**Theorem:** Consider a conjunctive query $Q_1(x_1,\ldots,x_k) :- \text{body}_1$.
If $Q_1$ is equivalent to a conjunctive query $Q_2(y_1,\ldots,y_k) :- \text{body}_2$, where $|\text{body}_2| < |\text{body}_1|$, then $Q_1$ is equivalent to a query $Q_1(x_1,\ldots,x_k) :- \text{body}_3$ such that $\text{body}_3 \subseteq \text{body}_1$.

The above theorem says that to minimize a conjunctive query $Q_1(x) :- \text{body}$ we simply need to remove some atoms from $\text{body}$. 
Minimization Procedure

Minimization($Q(x) :- \text{body}$)

Repeat until no change

    choose an atom $\alpha \in \text{body}$

    if there is a query homomorphism from $Q(x) :- \text{body}$ to $Q(x) :- \text{body} \setminus \{\alpha\}$

    then $\text{body} := \text{body} \setminus \{\alpha\}$

Return $Q(x) :- \text{body}$

Note: if there is a query homomorphism from $Q(x) :- \text{body}$ to $Q(x) :- \text{body} \setminus \{\alpha\}$, then the two queries are equivalent since there is trivially a query homomorphism from the latter to the former query.
Minimization Procedure: Example

Q(x) :- R(x,y), R(x,b), R(a,b), R(u,c), R(u,v), S(a,c,d)

{y → b}

Q(x) :- R(x,b), R(a,b), R(u,c), R(u,v), S(a,c,d)

{v → c}

Q(x) :- R(x,b), R(a,b), R(u,c), S(a,c,d)

minimal query

Note: the mapping x → a is not valid since x is a distinguished variable
Uniqueness of Minimal Queries

**Natural question:** does the order in which we remove atoms from the body of the input conjunctive query matter?

**Theorem:** Consider a conjunctive query $Q$. Let $Q_1$ and $Q_2$ be minimal conjunctive queries such that $Q_1 \equiv Q$ and $Q_2 \equiv Q$. Then, $Q_1$ and $Q_2$ are isomorphic (i.e., they are the same up to variable renaming).

Therefore, given a conjunctive query $Q$, the result of Minimization($Q$) is unique (up to variable renaming) and is called the **core of $Q$**.
Wrap-Up

• The main relational query languages – RA/DRC/TRC
  – Evaluation is decidable – foundations of the database industry
  – Perfect query optimization is impossible

• Conjunctive queries – an important query language
  – All the relevant algorithmic problems are decidable
  – Query minimization

RA = DRC = TRC*  

*under the active domain semantics
Associated Papers

• Ashok K. Chandra, Philip M. Merlin: Optimal Implementation of Conjunctive Queries in Relational Data Bases. STOC 1977: 77-90

  Criterion for CQ containment/equivalence


  A general account of connections between structural properties of databases and languages that capture efficient queries over them


  We can capture PTIME on databases that satisfy certain structural (graph-theoretic) restrictions
Associated Papers


Query languages that correspond to complexity classes


A connection between CQs and a central AI problem of constraint satisfaction

• Leonid Libkin: The finite model theory toolbox of a database theoretician. PODS 2009: 65-76

A toolbox for reasoning about expressivity and complexity of query languages
Associated Papers

  
  A specific application of the above toolbox for SQL

  
  Different types of complexity of database queries

  
  A finer way of measuring complexity, between data and combined