Background knowledge

- Conjunctive queries: the basis for data integration/exchange, metadata management, ontology-based data access, a very important class of database queries
- Chase: reasoning about constraints and a way to build new database instances
- Datalog: a recursive database language
Optimization of conjunctive queries

• Reminder:

  conjunctive queries
  = SPJ queries
  = rule-based queries
  = simple SELECT-FROM-WHERE SQL queries
  (only AND and equality in the WHERE clause)

• Extremely common, and thus special optimization techniques have been developed

• Reminder: for two relational algebra expressions $e_1, e_2$, $e_1 = e_2$ is undecidable.

• But for conjunctive queries, even $e_1 \subseteq e_2$ is decidable.

• Main goal of optimizing conjunctive queries: reduce the number of joins.
Optimization of conjunctive queries: an example

- Given a relation $R$ with two attributes $A, B$
- Two SQL queries:
  \[
  Q_1 \quad \text{SELECT R1.B, R1.A} \\
  \text{FROM R R1, R R2} \\
  \text{WHERE R2.A=R1.B}
  \]
  \[
  Q_2 \quad \text{SELECT R3.A, R1.A} \\
  \text{FROM R R1, R R2, R R3} \\
  \]
- Are they equivalent?
- If they are, we saved one join operation.
- In relational algebra:
  \[
  Q_1 = \pi_{2,1}(\sigma_{2=3}(R \times R))
  \]
  \[
  Q_2 = \pi_{5,1}(\sigma_{2=4 \land 4=5}(R \times R \times R))
  \]
Optimization of conjunctive queries cont’d

• Are $Q_1$ and $Q_2$ equivalent?

• If they are, we cannot show it by using equivalences for relational algebra expression.

• Because: they don’t decrease the number of $\land$ or $\times$ operators, but $Q_1$ has 1 join, and $Q_2$ has 2.

• But $Q_1$ and $Q_2$ are equivalent. How can we show this?

• But representing queries as databases. This representation is very close to rule-based queries.

$$Q_1(x, y) \ := \ R(y, x), R(x, z)$$

$$Q_2(x, y) \ := \ R(y, x), R(w, x), R(x, u)$$
Conjunctive queries into tableaux

- Tableau: representing of a conjunctive query as a database
- We first consider queries over a single relation
  - $Q_1(x, y) : \neg R(y, x), R(x, z)$
  - $Q_2(x, y) : \neg R(y, x), R(w, x), R(x, u)$
- Tableaux:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>z</td>
</tr>
</tbody>
</table>
  | x | y | ← answer line

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>w</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>u</td>
</tr>
</tbody>
</table>
| x | y | ← answer line

- Variables in the answer line are called distinguished
Tableau homomorphisms

- A homomorphism of two tableaux $f : T_1 \rightarrow T_2$ is a mapping
  
  $$f : \{\text{variables of } T_1\} \rightarrow \{\text{variables of } T_2\} \cup \{\text{constants}\}$$

- For every distinguished $x$, $f(x) = x$

- For every row $x_1, \ldots, x_k$ in $T_1$, $f(x_1), \ldots, f(x_k)$ is a row of $T_2$

- Query containment: $Q \subseteq Q'$ if $Q(D) \subseteq Q'(D)$ for every database $D$

- **Homomorphism Theorem**: Let $Q, Q'$ be two conjunctive queries, and $T, T'$ their tableaux. Then

  $$Q \subseteq Q'$$

  if and only if

  there exists a homomorphism $f : T' \rightarrow T$
Applying the Homomorphism Theorem: $Q_1 = Q_2$

Hence $Q_1 \subseteq Q_2$

Hence $Q_2 \subseteq Q_1$
Applying the Homomorphism Theorem: Complexity

• Given two conjunctive queries, how hard is it to test if $Q_1 = Q_2$?
• It is easy to transform them into tableaux, from either SPJ relational algebra queries, or SQL queries, or rule-based queries.
• But testing the existence of a homomorphism between two tableaux is hard: NP-complete. Thus, a polynomial algorithm is unlikely to exists.
• However, queries are small, and conjunctive query optimization is possible in practice.
Minimizing conjunctive queries

• Goal: given a conjunctive query $Q$, find an equivalent conjunctive query $Q'$ with the minimum number of joins.

• Assume $Q$ is

$$Q(\vec{x}) \leftarrow R_1(\vec{u}_1), \ldots, R_k(\vec{u}_k)$$

• Assume that there is an equivalent conjunctive query $Q'$ of the form

$$Q'(\vec{x}) \leftarrow S_1(\vec{v}_1), \ldots, S_l(\vec{v}_l)$$

with $l < k$

• Then $Q$ is equivalent to a query of the form

$$Q'(\vec{x}) \leftarrow R_{i_1}(\vec{u}_{i_1}), \ldots, R_{i_l}(\vec{u}_{i_l})$$

• In other words, to minimize a conjunctive query, one has to delete some subqueries on the right of :-
Minimizing conjunctive queries cont’d

• Given a conjunctive query $Q$, transform it into a tableau $T$.

• Let $Q'$ be a minimal conjunctive query equivalent to $Q$. Then its tableau $T'$ is a subset of $T$.

• Minimization algorithm:
  
  $T' := T$
  
  repeat until no change
    
    choose a row $t$ in $T'$
    
    if there is a homomorphism $f : T' \rightarrow T' \setminus \{t\}$
      
      then $T' := T' \setminus \{t\}$
    
  end

• Note: if there exists a homomorphism $T' \rightarrow T' \setminus \{t\}$, then the queries defined by $T'$ and $T' \setminus \{t\}$ are equivalent. Because: there is always a homomorphism from $T' \setminus \{t\}$ to $T'$. (Why?)
Minimizing SPJ/conjunctive queries: example

- \( R \) with three attributes \( A, B, C \)
- SPJ query
  \[
  Q = \pi_{AB}(\sigma_{B=4}(R)) \Join \pi_{BC}(\pi_{AB}(R) \Join \pi_{AC}(\sigma_{B=4}(R)))
  \]
- Translate into relational calculus:
  \[
  (\exists z_1 R(x, y, z_1) \land y = 4) \land \exists x_1 ((\exists z_2 R(x_1, y, z_2)) \land (\exists y_1 R(x_1, y_1, z) \land y_1 = 4))
  \]
- Simplify, by substituting the constant, and putting quantifiers forward:
  \[
  \exists x_1, z_1, z_2 (R(x, 4, z_1) \land R(x_1, 4, z_2) \land R(x_1, 4, z) \land y = 4)
  \]
- Conjunctive query:
  \[
  Q(x, y, z) :\neg R(x, 4, z_1), R(x_1, 4, z_2), R(x_1, 4, z), y = 4
  \]
Minimizing SPJ/conjunctive queries cont’d

• Tableau $T$:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>4</td>
<td>z</td>
<td>$z_1$</td>
</tr>
<tr>
<td>$x_1$</td>
<td>4</td>
<td>$z_2$</td>
<td></td>
</tr>
<tr>
<td>$x_1$</td>
<td>4</td>
<td>$z$</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>4</td>
<td>$z$</td>
<td></td>
</tr>
</tbody>
</table>

• Minimization, step 1: is there a homomorphism from $T$ to

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>4</td>
<td>$z_2$</td>
<td></td>
</tr>
<tr>
<td>$x_1$</td>
<td>4</td>
<td>$z$</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>4</td>
<td>$z$</td>
<td></td>
</tr>
</tbody>
</table>

• Answer: No. For any homomorphism $f$, $f(x) = x$ (why?), thus the image of the first row is not in the small tableau.
Minimizing SPJ/conjunctive queries cont’d

\[
\begin{array}{ccc}
A & B & C \\
\hline
\text{x} & 4 & \text{z}_1 \\
\text{x}_1 & 4 & \text{z} \\
\text{x} & 4 & \text{z} \\
\end{array}
\]

• Step 2: Is \( T \) equivalent to

• Answer: Yes. Homomorphism \( f: f(\text{z}_2) = \text{z}, \) all other variables stay the same.

• The new tableau is not equivalent to

\[
\begin{array}{ccc}
A & B & C \\
\hline
\text{x} & 4 & \text{z}_1 \\
\text{x} & 4 & \text{z} \\
\text{x} & 4 & \text{z} \\
\end{array}
\]  
\[
\begin{array}{ccc}
A & B & C \\
\hline
\text{x}_1 & 4 & \text{z} \\
\text{x} & 4 & \text{z} \\
\end{array}
\]

• Because \( f(\text{x}) = \text{x}, f(\text{z}) = \text{z}, \) and the image of one of the rows is not present.
Minimizing SPJ/conjunctive queries cont’d

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>4</td>
<td>z₁</td>
<td></td>
</tr>
<tr>
<td>x₁</td>
<td>4</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>4</td>
<td>z</td>
<td></td>
</tr>
</tbody>
</table>

• Minimal tableau:

• Back to conjunctive query:

\[ Q'(x, y, z) :\neg R(x, y, z₁), R(x₁, y, z), y = 4 \]

• An SPJ query:

\[ \sigma_{B=4} (\pi_{AB}(R) \Join \pi_{BC}(R)) \]

• Pushing selections:

\[ \pi_{AB}(\sigma_{B=4}(R)) \Join \pi_{BC}(\sigma_{B=4}(R)) \]
Review of the journey

• We started with

\[ \pi_{AB}(\sigma_{B=4}(R)) \Join \pi_{BC}(\pi_{AB}(R) \Join \pi_{AC}(\sigma_{B=4}(R))) \]

• Translated into a conjunctive query

• Built a tableau and minimized it

• Translated back into conjunctive query and SPJ query

• Applied algebraic equivalences and obtained

\[ \pi_{AB}(\sigma_{B=4}(R)) \Join \pi_{BC}(\sigma_{B=4}(R)) \]

• Savings: one join.
All minimizations are equivalent

• Let $Q$ be a conjunctive query, and $Q_1$, $Q_2$ two conjunctive queries equivalent to $Q$.

• Assume that $Q_1$ and $Q_2$ are both minimal, and let $T_1$ and $T_2$ be their tableaux.

• Then $T_1$ and $T_2$ are isomorphic; that is, $T_2$ can be obtained from $T_1$ by renaming of variables.

• That is, all minimizations are equivalent.

• In particular, in the minimization algorithm, the order in which rows are considered, is irrelevant.
Equivalence of conjunctive queries: the general case

• So far we assumed that there is only one relation $R$, but what if there are many?

• Construct tableaux as before:

$$Q(x, y) : \neg B(x, y), R(y, z), R(y, w), R(w, y)$$

• Tableau:

\[
\begin{array}{c|cc}
B: & A & B \\
\hline
x & y \\
\end{array}
\quad
\begin{array}{c|cc}
R: & A & B \\
\hline
y & z \\
y & w \\
w & y \\
\hline
x & y \\
\end{array}
\]

• Formally, a tableau is just a database where variables can appear in tuples, plus a set of distinguished variables.
Tableaux and multiple relations

• Given two tableaux $T_1$ and $T_2$ over the same set of relations, and the same distinguished variables, a homomorphism $h : T_1 \rightarrow T_2$ is a mapping

\[ f : \{ \text{variables of } T_1 \} \rightarrow \{ \text{variables of } T_2 \} \]

such that

- $f(x) = x$ for every distinguished variable, and
- for each row $\vec{t}$ in $R$ in $T_1$, $f(\vec{t})$ is in $R$ in $T_2$.

• Homomorphism theorem: let $Q_1$ and $Q_2$ be conjunctive queries, and $T_1, T_2$ their tableaux. Then

\[ Q_2 \subseteq Q_1 \]

if and only if

there exists a homomorphism $f : T_1 \rightarrow T_2$
Minimization with multiple relations

- The algorithm is the same as before, but one has to try rows in different relations. Consider homomorphism $f(z) = w$, and $f$ is the identity for other variables. Applying this to the tableau for $Q$ yields

$$
\begin{array}{c|c|c}
\text{B:} & A & B \\
\hline
x & y & \\
\end{array}
\begin{array}{c|c|c}
\text{R:} & A & B \\
\hline
y & w & w \\
\end{array}
\begin{array}{c|c|c}
 & x & y \\
\end{array}
\begin{array}{c|c|c}
 & w & y \\
\end{array}
$$

- This cannot be further reduced, as for any homomorphism $f$, $f(x) = x$, $f(y) = y$.

- Thus $Q$ is equivalent to

$$Q'(x, y) := B(x, y), R(y, w), R(w, y)$$

- One join is eliminated.
Static analysis of conjunctive queries: complexity

- Problem: given queries $Q_1, Q_2$, is $Q_1$ contained in $Q_2$?
- For full relational calculus, undecidable.
- For conjunctive queries, there is an algorithm:
  - guess a mapping $h$ between the tableaux of $Q_2$ and $Q_1$
  - check if it is a homomorphism.
  - Thus it is in NP.
- The problem is in fact NP-complete (sketch: blackboard).
- Hence efficient algorithms unlikely to exist unless P=NP.
- But the input is a query, not a database, hence algorithms are quite practical (heavily used in data integration)
  - still in the worst case they need exponential time
Query optimization and integrity constraints

- Additional equivalences can be inferred if integrity constraints are known
- Example: Let \( R \) have attributes \( A, B, C \). Assume that \( R \) satisfies \( A \rightarrow B \).
- Then \( R \) satisfies \( A \rightarrow B \) and thus
  \[
  R = \pi_{AB}(R) \bowtie \pi_{AC}(R)
  \]
- Tableaux can help with these optimizations!
- \( \pi_{AB}(R) \bowtie \pi_{AC}(R) \) as a conjunctive query:
  \[
  Q(x, y, z) : \neg R(x, y, z_1), R(x, y_1, z)
  \]
Query optimization and integrity constraints cont'd

- **Tableau:**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>z_1</td>
</tr>
<tr>
<td>x</td>
<td>y_1</td>
<td>z</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
</tr>
</tbody>
</table>

- Using the FD $A \rightarrow B$ infer $y = y_1$

- Next, minimize the resulting tableau

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>z_1</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
</tr>
</tbody>
</table>

$\rightarrow$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
</tr>
</tbody>
</table>

- And this says that the query is equivalent to $Q'(x, y, z): \neg R(x, y, z)$, that is, $R$. 
Query optimization and integrity constraints cont'd

- General idea: simplify the tableau using functional dependencies and then minimize.
- Given: a conjunctive query $Q$, and a set of FDs $F$
- Algorithm:
  
  Step 1. Compute the tableau $T$ for $Q$.
  
  Step 2. Apply algorithm $\text{CHASE}(T, F)$.

  Step 3. Minimize output of $\text{CHASE}(T, F)$.

  Step 4. Construct a query from the tableau produced in Step 3.
The CHASE

Assume that all FDs are of the form $X \rightarrow A$, where $A$ is an attribute.

for each $X \rightarrow A$ in $F$ do
    for each $t_1, t_2$ in $T$ such that $t_1.X = t_2.X$ and $t_1.A \neq t_2.A$ do
        case $t_1.A, t_2.A$ of
            both nondistinguished $\Rightarrow$
                replace one by the other
            one distinguished, one nondistinguished $\Rightarrow$
                replace nondistinguished by distinguished
            one constant, one variable $\Rightarrow$
                replace variable by constant
            both constants $\Rightarrow$
                output $\emptyset$ and STOP
        end
    end
end
Query optimization and integrity constraints: example

- \( R \) is over \( A, B, C \); \( F \) contains \( B \rightarrow A \)
- \( Q = \pi_{BC}(\sigma_{A=4}(R)) \land \pi_{AB}(R) \)
- \( Q \) as a conjunctive query:
  \[ Q(x, y, z) :– R(4, y, z), R(x, y, z_1) \]

- Tableau:
  \[
  \begin{array}{ccc}
  A & B & C \\
  4 & y & z \\
  x & y & z_1 \\
  x & y & z \\
  \end{array}
  \quad \xrightarrow{\text{CHASE}} \quad
  \begin{array}{ccc}
  A & B & C \\
  4 & y & z \\
  4 & y & z_1 \\
  4 & y & z \\
  \end{array}
  \]

- Final result: \( Q(x, y, z) :– R(x, y, z), x = 4 \), that is, \( \sigma_{A=4}(R) \).
Query optimization and integrity constraints: example

- Same $R$ and $F$; the query is:
  \[ Q = \pi_{BC}(\sigma_{A=4}(R)) \land \pi_{AB}(\sigma_{A=5}(R)) \]

- As a conjunctive query:
  \[ Q(x, y, z) :\leftarrow R(4, y, z), R(x, y, z_1), x = 5 \]

- Tableau:

  \[
  \begin{array}{ccc}
    A & B & C \\
    4 & y & z \\
    5 & y & z_1 \\
    5 & y & z
  \end{array}
  \]

  CHASE $\rightarrow \emptyset$

- Final result: $\emptyset$

- This equivalence is not true without the FD $B \rightarrow A$
Query optimization and integrity constraints: example

• Sometimes simplifications are quite dramatic

• Same \( R \), FD is \( A \rightarrow B \), the query is

\[
Q = \pi_{AB}(R) \land \pi_A(\sigma_{B=4}(R)) \land \pi_{AB}(\pi_{AC}(R) \land \pi_{BC}(R))
\]

• Convert into conjunctive query:

\[
Q(x, y) :– R(x, y, z_1), R(x, y_1, z), R(x_1, y, z), R(x, 4, z_2)
\]

• Tableau:

\[
\begin{array}{ccc}
A & B & C \\
 x & y & z_1 \\
 x & y_1 & z \\
 x_1 & y & z \\
 x & 4 & z_2 \\
x & y
\end{array}
\rightarrow
\begin{array}{ccc}
A & B & C \\
x & 4 & z_1 \\
x_1 & 4 & z \\
x & 4 & z_2 \\
x & 4
\end{array}
\]

minimize

\[
\begin{array}{ccc}
A & B & C \\
x & 4 & z \\
x & 4 \\
x & 4
\end{array}
\]
Query optimization and integrity constraints: example cont’d

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>4</td>
<td>z</td>
</tr>
<tr>
<td>x</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

is translated into

\[ Q(x, y) := R(x, y, z), y = 4 \]

• or, equivalently \( \pi_{AB}(\sigma_{B=4}(R)) \).

• Thus,

\[ \pi_{AB}(R) \Join \pi_A(\sigma_{B=4}(R)) \Join \pi_{AB}(\pi_{AC}(R) \Join \pi_{BC}(R)) = \pi_{AB}(\sigma_{B=4}(R)) \]

in the presence of FD \( A \rightarrow B \).

• Savings: 3 joins!

• This cannot be derived by algebraic manipulations, nor conjunctive query minimization without using CHASE.
Chase procedures

• In general, CHASE may refer to a family of procedures of a similar flavor: keep changing entries in a database instance as dictated by constraints

• Main uses:
  ○ checking constraints satisfiability and implication (and thus important for reasoning about metadata)
  ○ building instances that satisfy constraints (e.g., in data exchange)

• Many papers refer to CHASE procedures; we now review the classical one for implication of functional and join dependencies
FD and JD implication by CHASE

- Reminder: JDs are join dependencies
- A JD: \( \Join [X_1, \ldots, X_m] \)
- It holds in a relation \( R \) iff
  \[
  R = \pi_{X_1}(R) \Join \ldots \Join \pi_{X_m}(R)
  \]
- Important for decomposing relations and normalizing databases
- An FD \( X \rightarrow Y \) over attributes \( U \) implies a JD \( \Join [XY, X(U - Y)] \)
  - a simple exercise
- Let \( F \) be a set of FDs, \( J \) a set of JDs, and \( \theta \) a dependency (FD or JD)
- \( F, J \models \theta \) (in words, \( F \) and \( J \) imply \( \theta \)) if for every relation \( R \), if all of \( F \) and \( J \) dependencies are true in \( R \), then \( \theta \) is true in \( R \).
CHASE: tableaux and rules

- CHASE procedure consists of CHASE steps that apply to instances or tableaux. In tableaux, we shall mark distinguished variables in bold:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>x₁</td>
<td></td>
</tr>
<tr>
<td>x₂</td>
<td>y</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>x₂</td>
<td>y</td>
<td>x₃</td>
<td></td>
</tr>
</tbody>
</table>

- Rules for FDs we have already seen
CHASE: JD rule

Let $\mathcal{J}$ contain a join dependency $\bowtie [X_1, \ldots, X_m]$ and let $T$ be a tableau.

If $u$ is a tuple not in $T$ such that there are tuples $u_1, \ldots, u_n \in T$ such that $u_i[X_i] = u[X_i]$ for every $i \in [1, m]$, then the result of applying this JD over $T$ is the new tableau $T' = T \cup \{u\}$.
CHASE sequences

- A CHASE sequence of $T$ by a set of FDs and JDs is a sequence of tableaux $T = T_0, T_1, T_2, \ldots$, such that for each $i \geq 0$, $T_{i+1}$ is the result of applying some dependency to $T_i$.

- For JDs and FDs, all such sequences are finite (in other cases they won't be, and chase termination is a very important issue, particularly in data exchange).

- A sequence terminates when no more rules apply.

- No matter how we apply the rules, sequences terminate with the same tableau (up to renaming of non-distinguished variables).

- This tableau is denoted by $\text{chase}_{\mathcal{F}, \mathcal{J}}(T)$
CHASE for dependency implication

To check if $\mathcal{F}, \mathcal{J} \models \theta$:

- Construct a tableau $T_\theta$
- Compute $\text{chase}_{\mathcal{F}, \mathcal{J}}(T_\theta)$
- Check if a certain condition is satisfied.

If $\theta = A_1, \ldots, A_k \rightarrow A_{k+1}$ (attributes are $A_1, \ldots, A_m$):

- $T_\theta$ has two rows: $(x_1, \ldots, x_m)$ and $(x_1, \ldots, x_k, y_{k+1}, \ldots, y_m)$
- Condition: $\text{chase}_{\mathcal{F}, \mathcal{J}}(T_\theta)$ has only distinguished variables for $A_{k+1}$
Example: \{\bowtie [AB, AC], AB \to C\} \models A \to C

\[\begin{array}{ccc}
A & B & C \\
x & y & z \\
x & x_1 & x_2 \\
\end{array}\]

\[\begin{array}{ccc}
A & B & C \\
x & y & z \\
x & x_1 & x_2 \\
x & y & x_2 \\
\end{array}\]

Chase sequence: use \(\bowtie [AB, AC]\) and get:

Then use \(AB \to C\) and get

Only distinguished variables in column \(C\).
CHASE for JDs

- Let $\theta$ be $\Join [X_1, \ldots, X_n]$.
- $T_\theta$ has $n$ rows.
- The $i$th row has distinguished variables in the $X_i$-columns and non-distinguished variables in the remaining columns.
- Each non-distinguished variable appears exactly once.
- Condition: $\text{chase}_{\mathcal{F}, \mathcal{J}}(T)$ has a row with all distinguished variables.
Length of chase sequences

- In general, could be exponential
- An important question is when it is polynomial
- Then implication is solved in polynomial time
- Conditions known: essentially acyclicity of JDs
- We shall come back to the idea of acyclicity and polynomial chase termination in data exchange: this is how instances of exchanged data are constructed
Complexity classes: a very brief intro

- In databases, we reason about complexity in two ways:
  - The big-O notation ($O(n \log n)$ vs $O(n^2)$ vs $O(2^n)$)
  - Complexity-theoretic notions: PTIME, NP, DLOGSPACE, etc
- You see a lot of the latter in the literature
- Advantage of complexity-theoretic notions: if you have a $O(2^n)$ algorithm, is it because the problem is inherently hard, or because we are not smart enough to come up with a better algorithm (or both)?
The big divide

PTIME (computable in polynomial time, i.e. \(O(n^k)\) for some fixed \(k\))

Inside PTIME: tractable queries (although high-degree polynomial are real-life intractable)

Outside PTIME: intractable queries (efficient algorithms are unlikely)

Way outside PTIME: provably intractable queries (efficient algorithms do not exist)

- EXPTIME: \(c^n\)-algorithms for a constant \(c\). Could still be ok for not very large inputs

- Even further – 2-EXPTIME: \(c^{c^n}\). Cannot be ok even for small inputs (compare \(2^{10}\) and \(2^{2^{10}}\)).
Inside PTIME

\[ \text{AC}^0 \subsetneq \text{TC}^0 \subseteq \text{NC}^1 \subseteq \text{DLOG} \subseteq \text{NLOG} \subseteq \text{PTIME} \]

- \text{AC}^0: very efficient parallel algorithms (constant time, simple circuits)
  - relational calculus
- \text{TC}^0: very efficient parallel algorithms in a more powerful computational model with counting gates
  - basic SQL (relational calculus/grouping/aggregation)
- \text{NC}^1: efficient parallel algorithms
  - regular languages
- \text{DLOG}: very little \( O(\log n) \) – space is required
  - SQL + (restricted) transitive closure
- \text{NLOG}: \( O(\log n) \) space is required if nondeterminism is allowed
  - SQL + transitive closure (as in the SQL3 standard)
Beyond PTIME

\[ \text{PTIME} \subseteq \left\{ \begin{array}{c} \text{NP} \\ \text{coNP} \end{array} \right\} \subseteq \text{PSPACE} \]

- **PTIME**: can solve a problem in polynomial time
- **NP**: can check a given candidate solution in polynomial time
  - another way of looking at it: guess a solution, and then verify if you guessed it right in polynomial time
- **coNP**: complement of NP – verify that all “reasonable” candidates are solutions to a given problem.
  - Appears to be harder than NP but the precise relationship isn’t known
- **PSPACE**: can be solved using memory of polynomial size (but perhaps an exponential-time algorithm)
Complete problems

- These are the hardest problems in a class.
- If our problem is as hard as a complete problem, it is very unlikely it can be done with lower complexity.

- For NP:
  - SAT (satisfiability of Boolean formulae)
  - many graph problems (e.g. 3-colourability)
  - Integer linear programming etc

- For PSPACE:
  - Quantified SAT
  - Are two regular languages equivalent?
  - Many games, e.g., Geography.
Measuring complexity in databases

Problem: Given a database $D$, and a query $Q$, find $Q(D)$.

Complexity measurements are defined for decision problems, so: Given $D$, $Q$, and a tuple $u$, is $u \in Q(D)$?

- Combined complexity: all $D$, $Q$, $u$ are inputs to the problem.
- Data complexity: $Q$ is fixed.
  - Rationale: $Q$ is much smaller than $D$, can disregard it
Limitations of SQL

• Reachability queries:

<table>
<thead>
<tr>
<th>Flights</th>
<th>Src</th>
<th>Dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>'EDI'</td>
<td>'LHR'</td>
<td></td>
</tr>
<tr>
<td>'EDI'</td>
<td>'EWR'</td>
<td></td>
</tr>
<tr>
<td>'EWR'</td>
<td>'LAX'</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

• Query: Find pairs of cities \((A, B)\) such that one can fly from \(A\) to \(B\) with at most one stop:

\[
\begin{align*}
&\text{SELECT } F1.\text{Src}, F2.\text{Dest} \\
&\text{FROM Flights } F1, \text{Flights } F2 \\
&\text{WHERE } F1.\text{Dest}=F2.\text{Src} \\
&\text{UNION} \\
&\text{SELECT } \ast \text{FROM Flights}
\end{align*}
\]
Reachability queries cont’d

• Query: Find pairs of cities \((A, B)\) such that one can fly from \(A\) to \(B\) with at most two stops:

\[
\begin{align*}
&\text{SELECT F1.Src, F3.Dest} \\
&\text{FROM Flights F1, Flights F2, Flights F3} \\
&\text{UNION} \\
&\text{SELECT F1.Src, F2.Dest} \\
&\text{FROM Flights F1, Flights F2} \\
&\text{WHERE F1.Dest=F2.Src} \\
&\text{UNION} \\
&\text{SELECT * FROM Flights}
\end{align*}
\]
Reachability queries cont’d

• For any fixed number \( k \), we can write the query

  Find pairs of cities \((A, B)\) such that one can fly from \( A \) to \( B \) with at most \( k \) stops

  in SQL.

• What about the general reachability query:

  Find pairs of cities \((A, B)\) such that one can fly from \( A \) to \( B \).

• SQL cannot express this query.

• Solution: SQL3 adds a new construct that helps express reachability queries. (May not yet exist in some products.)
Reachability queries cont’d

- To understand the reachability query, we formulate it as a rule-based query:

\[
\text{reach}(x, y) := \text{flights}(x, y) \\
\text{reach}(x, y) := \text{flights}(x, z), \text{reach}(z, y)
\]

- One of these rules is recursive: \text{reach} refers to itself.

- Evaluation:

  - Step 0: \text{reach}^0 is initialized as the empty set.
  - Step \(i + 1\): Compute
    \[
    \text{reach}^i(x, y) := \text{flights}(x, y) \\
    \text{reach}^i(x, y) := \text{flights}(x, z), \text{reach}^i(z, y)
    \]
  - Stop condition: If \(\text{reach}^{i+1} = \text{reach}^i\), then it is the answer to the query.
Evaluation of recursive queries

• Example: assume that \textit{flights} contains \((a, b), (b, c), (c, d)\).

• Step 0: \textit{reach} = \emptyset

• Step 1: \textit{reach} becomes \\{(a, b), (b, c), (c, d)\}.

• Step 2: \textit{reach} becomes \\{(a, b), (b, c), (c, d), (a, c), (b, d)\}.

• Step 3: \textit{reach} becomes \\{(a, b), (b, c), (c, d), (a, c), (b, d), (a, d)\}.

• Step 4: one attempts to use the rules, but infers no new values for \textit{reach}. The final answer is thus:

\[
\{(a, b), (b, c), (c, d), (a, c), (b, d), (a, d)\}
\]
Recursion in SQL3

- SQL3 syntax mimics that of recursive rules:

\[
\text{WITH RECURSIVE Reach(Src,Dest) AS} \\
\quad ( \\
\quad \quad \text{SELECT * FROM Flights} \\
\quad \quad \text{UNION} \\
\quad \quad \text{SELECT F.Src, R.Dest} \\
\quad \quad \text{FROM Flights F, Reach R} \\
\quad \quad \text{WHERE F.Dest= R.Src} \\
\quad ) \\
\quad \text{SELECT * FROM Reach}
\]
Recursion in SQL3: syntactic restrictions

- There is another way to do reachability as a recursive rule-based query:
  \[
  \text{reach}(x, y) := \text{flights}(x, y) \\
  \text{reach}(x, y) := \text{reach}(x, z), \text{reach}(z, y)
  \]

- This translates into an SQL3 query:
  
  ```sql
  WITH RECURSIVE Reach(Src,Dest) AS
  ( SELECT * FROM Flights
    UNION
    SELECT R1.Src, R2.Dest
    FROM Reach R1, Reach R2
    WHERE R1.Dest=R2.Src )
  SELECT * FROM Reach
  ```

- However, most implementations will disallow this, since they support only linear recursion: recursively defined relation is only mentioned once in the FROM line.
Recursion in SQL3 cont’d

• A slight modification: suppose Flights has another attribute aircraft.
• Query: find cities reachable from Edinburgh.

WITH Cities AS SELECT Src, Dest FROM Flights
RECURSIVE Reach(Src, Dest) AS
(
  SELECT * FROM Cities
  UNION
  SELECT C.Src, R.Dest FROM Cities C, Reach R WHERE C.Dest = R.Src
)
SELECT R.Dest FROM Reach R WHERE R.Src = 'EDI'
A note on negation

- Problematic recursion:

```sql
WITH RECURSIVE R(A) AS
  (SELECT S.A
   FROM S
   WHERE S.A NOT IN
     SELECT R.A FROM R)

SELECT * FROM R
```

- Formulated as a rule:

\[
r(x) \leftarrow s(x), \neg r(x)
\]
A note on negation cont’d

• Let $s$ contain $\{1, 2\}$.

• Evaluation:
  
  After step 0: $r_0 = \emptyset$;
  
  After step 1: $r_1 = \{1, 2\}$;
  
  After step 2: $r_2 = \emptyset$;
  
  After step 3: $r_3 = \{1, 2\}$;
  
  ... 
  
  After step $2n$: $r_{2n} = \emptyset$;
  
  After step $2n + 1$: $r_{2n+1} = \{1, 2\}$.

• Problem: it does not terminate!

• What causes this problem? Answer: Negation (that is, NOT IN).
A note on negation cont’d

• Other instances of negation:
  
  EXCEPT

  NOT EXISTS

• SQL3 has a set of very complicated rules that specify when the above operations can be used in WITH RECURSIVE definitions.

• A general rule: it is best to avoid negation in recursive queries.
Notes on proposed papers

   Criterion for CQ containment/equivalence

   Notion of acyclicity of CQs and fast evaluation scheme based on it

   An in-depth study of acyclicity

   A hierarchy of classes of efficient CQs, the bottom level of which is acyclic queries
   A different way of characterizing efficiency of CQs, this time via the notion of bounded treewidth

   Different types of complexity of database queries, and a language for PTIME

   A finer way of measuring complexity, between data and combined

   Query languages that correspond to complexity classes

We can capture PTIME on some databases if they satisfy certain structural (graph-theoretic) restrictions


An intriguing connection between conjunctive queries and a central AI problem of constraint satisfaction


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


A toolbox for reasoning about expressivity and complexity of query languages

and a specific application for SQL


The paper that proposed CHASE


and the paper that looked at how to make it efficient more often