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
- Automata: the basis for formalisms for XML and graph databases
Review of Relational Databases

- Relational model
- Schemas
- Relational algebra
- Relational calculus
- SQL
- Constraints (keys, foreign keys)
The relational model

- Data is organized in relations (tables)
- Relational database schema:
  - set of table names
  - list of attributes for each table
- Tables are specified as: `<table name>:<list of attributes>`
- Examples:
  - Account: number, branch, customerId
  - Movie: title, director, actor
  - Schedule: theater, title
- Attributes within a table have different names
- Tables have different names
Declarative vs Procedural

- In our queries, we ask **what** we want to see in the output.
- But we do not say **how** we want to get this output.
- Thus, query languages are **declarative**: they specify what is needed in the output, but do not say how to get it.
- Database system figures out **how** to get the result, and gives it to the user.
- Database system operates internally with different, **procedural** languages, which specify how to get the result.
Declarative vs Procedural: example

Declarative:

\{ \text{title} | (\text{title}, \text{director}, \text{actor}) \in \text{movies} \} 

Procedural:

for each tuple \( T=(t,d,a) \) in relation movies do
  output \( t \)
end

In relational algebra: \( \pi_{\text{title}}(\text{Movies}) \).

in SQL:

SELECT title FROM Movies
Relational Calculus

- Codd 1970: Relational databases are queried using first-order predicate logic.
- Relational calculus: another name for it. Queries written in the logical notation using:
  - relation names (e.g., Movies)
  - constants (e.g., 'Shining', 'Nicholson')
  - conjunction $\land$, disjunction $\lor$, negation $\neg$
  - existential quantifiers $\exists$, universal quantifiers $\forall$
- $\land$, $\exists$, $\neg$ suffice:
  - $\forall x F(x) = \neg \exists x \neg F(x)$
  - $F \lor G = \neg (\neg F \land \neg G)$
Relational Calculus cont’d

- Bound variable: a variable $x$ that occurs in $\exists x$ or $\forall x$
- Free variable: a variable that is not bound.
- Free variables are those that go into the output of a query.
- Two ways to write a query:
  \[ Q(\vec{x}) = F, \]  
  where $\vec{x}$ is the tuple of free variables
  \[ \{ \vec{x} \mid F \} \]
- Examples:
  \[ \{ x, y \mid \exists z (R(x, z) \land S(z, y)) \} \]
  \[ \{ x \mid \forall y R(x, y) \} \]
  \[ \{ \text{dir} \mid \forall (\text{th}, \text{tl}) \in \text{schedule} \}
    \exists (\text{tl'}, \text{act}): (\text{tl'}, \text{dir}, \text{act}) \in \text{movies} \land (\text{th}, \text{tl'}) \in \text{schedule} \} \]
Relational Algebra

• Procedural language

• Six ( = 5 + 1 ) operations:
  ○ Projection $\pi$
  ○ Selection $\sigma$
  ○ Cartesian product $\times$
  ○ Union $\cup$
  ○ Difference $-$
  ○ Renaming $\rho$

• Renaming changes names of attributes

• $\rho_{A\leftarrow C,B\leftarrow D}(R)$ turns a relation with attributes $C, D$ into a relation with attributes $A, B$. 
Relational Algebra cont’d

• Projection: chooses some attributes in a relation
• \( \pi_{A_1, \ldots, A_n}(R) \): only leaves attributes \( A_1, \ldots, A_n \) in relation \( R \).
• Selection: Chooses tuples that satisfy some condition
• \( \sigma_c(R) \): only leaves tuples \( t \) for which \( c(t) \) is true
• Conditions: conjunctions of
  \( R.A = R.A' \) – two attributes are equal
  \( R.A = constant \) – the value of an attribute is a given constant
  Same as above but with \( \neq \) instead of \( = \)
Relational Algebra cont’d

- Cartesian Product: puts together two relations
- $R_1 \times R_2$ puts together each tuple $t_1$ of $R_1$ and each tuple $t_2$ of $R_2$
- Example:

\[
\begin{array}{|c|c|} \hline
R_1 & \quad R_2 \\
\hline A & B \\
\hline a_1 & b_1 \\
a_2 & b_2 \\
\hline
\end{array} \times \begin{array}{|c|c|} \hline
R_1 & \quad R_2 \\
\hline A & C \\
\hline a_1 & c_1 \\
a_2 & c_2 \\
a_3 & c_3 \\
\hline
\end{array} = \begin{array}{|c|c|c|c|c|} \hline R_1.A & R_1.B & R_2.A & R_2.C \\
\hline a_1 & b_1 & a_1 & c_1 \\
a_1 & b_1 & a_2 & c_2 \\
a_1 & b_1 & a_3 & c_3 \\
a_2 & b_2 & a_1 & c_1 \\
a_2 & b_2 & a_2 & c_2 \\
a_2 & b_2 & a_3 & c_3 \\
\hline \end{array}
\]
Relational Algebra cont’d

- Union $R \cup S$ is the union of relations $R$ and $S$
- $R$ and $S$ must have the same set of attributes.
- Difference $R - S$: tuples in $R$ but not in $S$.

- Every declarative query has a procedural implementation:

  \[ \text{Relational Calculus} = \text{Relational Algebra} \]
SQL

• Structured Query Language
• Developed originally at IBM in the late 70s
• First standard: SQL-86
• De-facto standard of the relational database world – replaced all other languages.
Examples of SQL queries

• Find titles of current movies

SELECT Title
FROM Movies

• SELECT lists attributes that go into the output of a query

• FROM lists input relations
Examples of SQL queries cont’d

• Find theaters showing movies in which Nicholson played:

```sql
SELECT Schedule.Theater
FROM Schedule, Movies
WHERE Movies.Title = Schedule.Title
    AND Movies.Actor='Nicholson'
```

Differences:

• SELECT now specifies which relation the attributes came from – because we use more than one.

• FROM lists two relations

• WHERE specifies the condition for selecting a tuple.
Joining relations

• WHERE allows us to join together several relations

• Consider a query: list directors, and theaters in which their movies are playing

  SELECT Movies.Director, Schedule.Theater
  FROM Movies, Schedule
  WHERE Movies.Title = Schedule.Title

• This operation is called join.

• Notation: Schedule \( \Join \) Movies
Join cont’d

- Join is not a new operation of relational algebra
- It is definable with π, σ, ×
- Suppose \( R \) is a relation with attributes \( A_1, \ldots, A_n, B_1, \ldots, B_k \)
- \( S \) is a relation with attributes \( A_1, \ldots, A_n, C_1, \ldots, C_m \)
- \( R \bowtie S \) has attributes \( A_1, \ldots, A_n, B_1, \ldots, B_k, C_1, \ldots, C_m \)

\[
R \bowtie S = \pi_{A_1, \ldots, A_n, B_1, \ldots, B_k, C_1, \ldots, C_m} (\sigma_{R.A_1 = S.A_1 \land \ldots \land R.A_n = S.A_n} (R \times S))
\]
Conjunctive queries

- Also known as select-project-join queries
- Fragment of relational algebra that consists of $\sigma, \pi, \star$ (or $\sigma, \pi, \times$)
- In logic, $\exists$ and $\land$
- Theaters showing movies where Nicholson played:
  \[
  \pi_{\text{theater}}(\sigma_{\text{actor}=\text{Nicholson}}(\text{Movies} \star \text{Schedule}))
  \]
  (hence called SPJ – select, project, join – queries)

\[
\exists t \exists d \text{Movies}(t, d, \text{Nicholson}) \land \text{Schedule}(t, th)
\]

often write as rules:

\[
Q(th) : \leftarrow \text{Movies}(t, d, \text{Nicholson}), \text{Schedule}(t, th)
\]
Beyond simple queries

- So far we mostly used $\pi, \sigma, \bowtie$ in relational algebra.
- It is harder to do queries with “for all conditions”.
- Query: Find directors whose movies are playing in all theaters:

$$\pi_{\text{director}}(M) - \pi_{\text{director}}(\pi_{\text{theater}}(S) \times \pi_{\text{director}}(M) - \pi_{\text{theater,director}}(M \bowtie S))$$

- They don’t look easy in relational algebra
For all and negation in SQL

- Find directors whose movies are playing in all theaters.
- SQL's way of saying this: Find directors such that there does not exist a theater where their movies do not play.
- Because: \( \forall x \, f(x) \iff \neg \exists x \, \neg f(x) \).

```sql
SELECT M1.Director
FROM Movies M1
WHERE NOT EXISTS (SELECT S.Theater
FROM Schedule S
WHERE NOT EXISTS (SELECT M2.Director
FROM Movies M2
WHERE M2.Title=S.Title
AND
M1.Director=M2.Director))
```
Other features of SQL

- Datatypes, type-specific operations
- Table declaration, constraint enforcement
- Aggregation
Simple aggregate queries

Count the number of tuples in Movies

```sql
SELECT COUNT(*)
FROM Movies
```

Add up all movie lengths

```sql
SELECT SUM(Length)
FROM Movies
```

Find the number of directors.

```sql
SELECT COUNT(DISTINCT Director)
FROM Movies
```
Aggregation and grouping

For each theaters playing at least one long (over 2 hours) movie, find the average length of all movies played there:

```
SELECT S.Theater, AVG(M.Length)
FROM Schedule S, Movies M
WHERE S.Title=M.Title
GROUP BY S.Theater
HAVING MAX(M.Length) > 120
```
Database Constraints

- In our examples we assumed that the *title* attribute identifies a movie.

- But this may not be the case:

<table>
<thead>
<tr>
<th>title</th>
<th>director</th>
<th>actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dracula</td>
<td>Browning</td>
<td>Lugosi</td>
</tr>
<tr>
<td>Dracula</td>
<td>Fischer</td>
<td>Lee</td>
</tr>
<tr>
<td>Dracula</td>
<td>Badham</td>
<td>Langella</td>
</tr>
<tr>
<td>Dracula</td>
<td>Coppola</td>
<td>Oldman</td>
</tr>
</tbody>
</table>

- Database constraints: provide additional semantic information about the data.

- Most common ones: functional and inclusion dependencies, and their special cases: keys and foreign keys.
Constraints cont’d

• If we want the title to identify a movie uniquely (i.e., no Dracula situation), we express it as a **functional dependency**

\[
title \rightarrow \text{director}
\]

• In general, a relation \( R \) satisfies a functional dependency \( A \rightarrow B \), where \( A \) and \( B \) are attributes, if for every two tuples \( t_1, t_2 \) in \( R \):

\[
\pi_A(t_1) = \pi_A(t_2) \quad \text{implies} \quad \pi_B(t_1) = \pi_B(t_2)
\]
Functional dependencies and keys

- More generally, a functional dependency is $X \rightarrow Y$ where $X, Y$ are sequences of attributes. It holds in a relation $R$ if for every two tuples $t_1, t_2$ in $R$:
  \[\pi_X(t_1) = \pi_X(t_2) \implies \pi_Y(t_1) = \pi_Y(t_2)\]

- A very important special case: keys

- Let $K$ be a set of attributes of $R$, and $U$ the set of all attributes of $R$. Then $K$ is a key if $R$ satisfies functional dependency $K \rightarrow U$.

- In other words, a set of attributes $K$ is a key in $R$ if for any two tuples $t_1, t_2$ in $R$,\[\pi_K(t_1) = \pi_K(t_2) \implies t_1 = t_2\]

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

- We expect every Title listed in Schedule to be present in Movies.
- These are referential integrity constraints: they talk about attributes of one relation (Schedule) but refer to values in another one (Movies).
- These particular constraints are called inclusion dependencies (ID).
- Formally, we have an inclusion dependency \( R[A] \subseteq S[B] \) when every value of attribute \( A \) in \( R \) also occurs as a value of attribute \( B \) in \( S \):
  \[
  \pi_A(R) \subseteq \pi_B(S)
  \]
- As with keys, this extends to sets of attributes, but they must have the same number of attributes.
- There is an inclusion dependency \( R[A_1, \ldots, A_n] \subseteq S[B_1, \ldots, B_n] \) when
  \[
  \pi_{A_1,\ldots,A_n}(R) \subseteq \pi_{B_1,\ldots,B_n}(S)
  \]
Foreign keys

• Most often inclusion constraints occur as a part of a foreign key
• Foreign key is a conjunction of a key and an ID:

\[ R[A_1, \ldots, A_n] \subseteq S[B_1, \ldots, B_n] \quad \text{and} \]
\[ \{B_1, \ldots, B_n\} \rightarrow \text{all attributes of } S \]

• Meaning: we find a key for relation \( S \) in relation \( R \).

• Example: Suppose we have relations:

\[
\begin{align*}
\text{Employee(EmplId, Name, Dept, Salary)} \\
\text{ReportsTo(Empl1, Empl2)}.
\end{align*}
\]

• We expect both Empl1 and Empl2 to be found in Employee; hence:

\[
\begin{align*}
\text{ReportsTo[Empl1]} \subseteq \text{Employee[EmplId]} \\
\text{ReportsTo[Empl2]} \subseteq \text{Employee[EmplId]}.
\end{align*}
\]

• If EmplId is a key for Employee, then these are foreign keys.
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:

  Q1
  \[
  \text{SELECT } R1.B, R1.A \\
  \text{FROM } R R1, R R2 \\
  \text{WHERE } R2.A = R1.B
  \]

  Q2
  \[
  \text{SELECT } R3.A, R1.A \\
  \text{FROM } R R1, R R2, R R3 \\
  \text{WHERE } R1.B = R2.B \land R2.B = R3.A
  \]

• 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.

$$Q_1(x, y) \leftarrow R(y, x), R(x, z)$$

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

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

  $$
  \begin{array}{cc}
  \hline
  A & B \\
  \hline
  y & x \\
  x & z \\
  \hline
  x & y \leftarrow \text{answer line}
  \end{array}
  $$

  $$
  \begin{array}{cc}
  \hline
  A & B \\
  \hline
  y & x \\
  w & x \\
  x & u \\
  \hline
  x & y \leftarrow \text{answer line}
  \end{array}
  $$

- 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: \( Q_1 = Q_2 \)

**T1**

<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>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
</tr>
</tbody>
</table>

**T2**

<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>---</td>
<td>---</td>
</tr>
<tr>
<td>x</td>
<td>u</td>
</tr>
</tbody>
</table>

\[ f(x) = x, \ f(y) = y \]
\[ f(u) = z, \ f(w) = y \]

Hence \( Q_1 \subseteq Q_2 \)

---

**T1**

<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>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
</tr>
</tbody>
</table>

**T2**

<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>---</td>
<td>---</td>
</tr>
<tr>
<td>x</td>
<td>u</td>
</tr>
</tbody>
</table>

\[ f(x) = x, \ f(y) = y \]
\[ f(z) = u \]

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:

\[
T' := T \\
\text{repeat until no change} \\
\quad \text{choose a row } t \text{ in } T' \\
\quad \text{if there is a homomorphism } f : T' \rightarrow T' - \{t\} \\
\quad \quad \text{then } T' := T' - \{t\} \\
\text{end}
\]

• Note: if there exists a homomorphism $T' \rightarrow T' - \{t\}$, then the queries defined by $T'$ and $T' - \{t\}$ are equivalent. Because: there is always a homomorphism from $T' - \{t\}$ 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:
  \[
  Q(x, y, z) :– R(x, 4, z_1), R(x_1, 4, z_2), R(x_1, 4, z), y = 4
  \]
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></td>
</tr>
<tr>
<td>x₁</td>
<td>4</td>
<td>z₂</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>

- 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>x₁</td>
<td>4</td>
<td>z₂</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>

- 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

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

- Step 2: Is $T$ equivalent to

- Answer: Yes. Homomorphism $f$: $f(z_2) = z$, all other variables stay the same.

- The new tableau is not equivalent to

- Because $f(x) = x$, $f(z) = z$, and the image of one of the rows is not present.
Minimizing SPJ/conjunctive queries 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>z₁</td>
</tr>
<tr>
<td>x₁</td>
<td>4</td>
<td>z</td>
</tr>
<tr>
<td>x</td>
<td>4</td>
<td>z</td>
</tr>
</tbody>
</table>

• Minimal tableau:

• Back to conjunctive query:

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

• 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:

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

<table>
<thead>
<tr>
<th>R:</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>y</td>
<td></td>
</tr>
</tbody>
</table>

• 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 $f : T_1 \rightarrow T_2$
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

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

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

$$R = \pi_{AB}(R) \Join \pi_{AC}(R)$$

- Tableaux can help with these optimizations!
- $\pi_{AB}(R) \Join \pi_{AC}(R)$ 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>z₁</td>
</tr>
<tr>
<td>x</td>
<td>y₁</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 $y = y₁$

- 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>z₁</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>

<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 $CHASE(T, F)$.
  Step 3. Minimize output of $CHASE(T, F)$.
  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 ⇒
       replace one by the other
    one distinguished, one nondistinguished ⇒
       replace nondistinguished by distinguished
    one constant, one variable ⇒
       replace variable by constant
    both constants ⇒
       output $\emptyset$ and STOP
  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:

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

- 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:
  \[ Q = \pi_{BC}(\sigma_{A=4}(R)) \Join \pi_{AB}(\sigma_{A=5}(R)) \]

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

- 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 $B \rightarrow A$

Beijing 52 Topics in Foundations of DB
Query optimization and integrity constraints: example

- Sometimes simplifications are quite dramatic
- Same $R$, FD is $A \rightarrow B$, the query is
  \[ Q = \pi_{AB}(R) \Join \pi_A(\sigma_{B=4}(R)) \Join \pi_{AB}(\pi_{AC}(R) \Join \pi_{BC}(R)) \]
- 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:

<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>$z_1$</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>$y_1$</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>$x_1$</td>
<td>y</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>4</td>
<td>$z_2$</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CHASE

\[ \begin{array}{ccc}
  x & 4 & z_1 \\
  x_1 & 4 & z \\
  x & 4 & z_2 \\
  x & 4
\end{array} \]

minimize
Query optimization and integrity constraints: example cont’d

A | B | C
---|---|---
x | 4 | z
x | 4

is translated into

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

• 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 \( \mathcal{F} \) be a set of FDs, \( \mathcal{J} \) a set of JDs, and \( \theta \) a dependency (FD or JD)

• \( \mathcal{F}, \mathcal{J} \models \theta \) (in words, \( \mathcal{F} \) and \( \mathcal{J} \) imply \( \theta \)) if for every relation \( R \), if all of \( \mathcal{F} \) and \( \mathcal{J} \) 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:

\[
\begin{array}{ccc}
A & B & C \\
x & y & x_1 \\
x_2 & y & z \\
x_2 & y & x_3 \\
\end{array}
\]

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

Let $J$ contain a join dependency $\Join [X_1, \ldots, X_m]$ 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 $chase_{\mathcal{F}, \mathcal{J}}(T_\theta)$
- 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: $chase_{\mathcal{F}, \mathcal{J}}(T_\theta)$ has only distinguished variables for $A_{k+1}$
Example: \( \{\bowtie [AB, AC], AB \rightarrow C\} \models A \rightarrow C \)

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

\( T_{A \rightarrow C} \):

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

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

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

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

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
Automata and regular languages

- The key toolkit for XML and graph data
- For XML, we need automata working on both words (strings) and trees
- We’ll define them later and for now review automata on words
Automata and regular languages

A nondeterministic finite automaton (NFA) over a finite alphabet $\Sigma$ is $A = (Q, q_0, \delta, F)$ where

- $Q$ is a set of states
- $q_0$ is an initial state (sometimes people assume $Q_0 \subseteq Q$ of initial states: no difference)
- $\delta : Q \times \Sigma \rightarrow 2^Q$: transition function
- $F \subseteq Q$: set of final states

A run of $A$ on a word $a_0a_1 \ldots a_n$ is a map $\rho$ from positions to states such that:

- $\rho(0) \in \delta(q_0, a_0)$
- $\rho(i + 1) \in \delta(\rho(i), a_{i+1})$
Automata and regular languages cont’d

• Intuition: $\rho$ indicates where in which state the automaton could be after reading a portion of the word
• A run is accepting is $\rho(n) \in F$: it accepts after reading everything
• A word is accepted by $A$ if there is an accepting run
• $L(A)$: the language of automaton – set of all accepted words
• These are regular languages
  ○ also given by regular expressions
  ○ also given by monoid homomorphisms
  ○ also given by monadic second order logic
  ○ and many other formalisms (don’t worry if you don’t know the last two)
Automata and computational problems

- **Membership**: Given a word $w \in \Sigma^*$ and $A$, is $w \in L(A)$?
  - Complexity: NLOG. Think of guessing where the automaton will go.

- **Nonemptiness**: given $A$, is $L(A) \neq \emptyset$?
  - Linear time: reachability of $F$ from $q_0$; also NLOG-complete.

- **Universality**: given $A$, is $L(A) = \Sigma^*$
  - PSPACE-complete. Think of converting to a DFA and then checking emptiness of the complement.

- **Variations**: Given $A_1, A_2$, is $L(A_1) \cap L(A_2) \neq \emptyset$?
  - Of course we can construct $A = A_1 \times A_2$ and check $L(A) \neq \emptyset$, but one can do better (on-the-fly); we’ll see how when we talk about graph database 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

9. Martin Grohe: Fixed-point definability and polynomial time on

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