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:

$$\begin{aligned} & \text{conjunctive queries} \\ = \ & \text{SPJ queries} \\ = \ & \text{rule-based queries} \\ = \ & \text{simple SELECT-FROM-WHERE SQL queries} \\ & \text{(only AND and equality in the WHERE clause)} \end{aligned}$$

- 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

```
SELECT R1.B, R1.A
FROM R R1, R R2
WHERE R2.A=R1.B
```

Q2

```
SELECT R3.A, R1.A
FROM R R1, R R2, R R3
WHERE R1.B=R2.B AND R2.B=R3.A
```

- Are they equivalent?
- If they are, we saved one join operation.
- In relational algebra:

$$Q_1 = \pi_{2,1}(\sigma_{2=3}(R \times R))$$

$$Q_2 = \pi_{5,1}(\sigma_{2=4 \wedge 4=5}(R \times R \times R))$$

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 $\bowtie$ 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.

$$Q_1(x,y) \text{ :- } R(y,x), R(x,z)$$

$$Q_2(x,y) \text{ :- } R(y,x), R(w,x), R(x,u)$$

Conjunctive queries into tableaux

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

A	B	
y	x	
x	z	
x	y	← answer line

A	B	
y	x	
w	x	
x	u	
x	y	← answer line

- Variables in the answer line are called distinguished

Tableau homomorphisms

- A homomorphism of two tableaux $f : T_1 \to T_2$ is a mapping

$$f : \{\text{variables of } T_1\} \to \{\text{variables of } T_2\} \ \bigcup \ \{\text{constants}\}$$

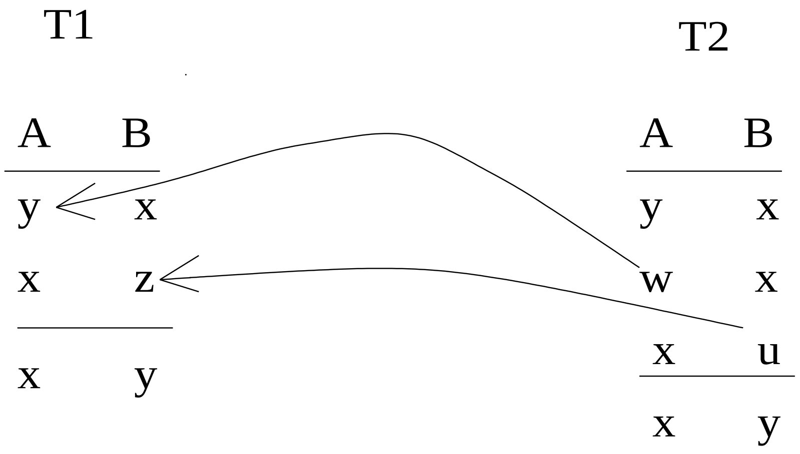
- For every distinguished x, $f(x) = x$
- For every row $x_1, \ldots, x_k$ in T_1, $f(x_1), \ldots, f(x_k)$ is a row of T_2
- Query containment: $Q \subseteq Q'$ if $Q(D) \subseteq Q'(D)$ for every database D
- **Homomorphism Theorem**: Let Q, Q' be two conjunctive queries, and T, T' their tableaux. Then

$$Q \subseteq Q'$$

if and only if

there exists a homomorphism $f : T' \to T$

Applying the Homomorphism Theorem: $Q_1 = Q_2$



T1

A	B
y	x
x	z
x	y

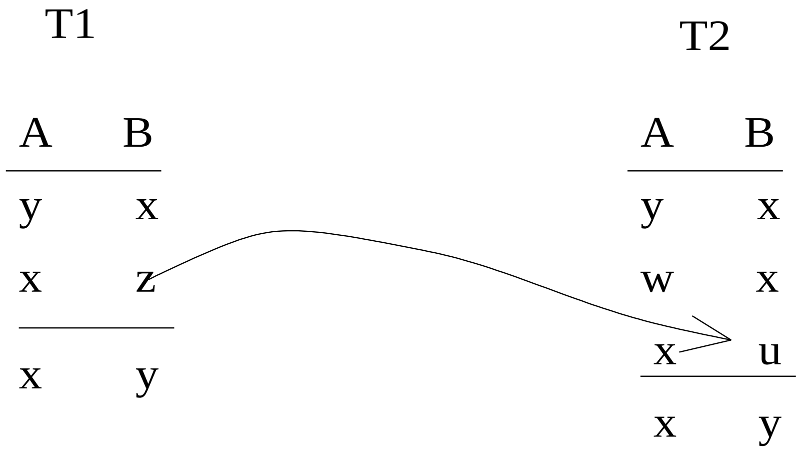
T2

A	B
y	x
w	x
x	u
x	y

f(x)=x, f(y)=y

f(u)=z, f(w)=y

Hence Q1 $\subseteq$ Q2



T1

A	B
y	x
x	z
x	y

T2

A	B
y	x
w	x
x	u
x	y

f(x)=x, f(y)=y

f(z)=u

Hence Q2 $\subseteq$ Q1

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

$$Q(\vec{x}) \;\; \text{:–} \;\; R_1(\vec{u}_1), \ldots, R_k(\vec{u}_k)$$

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

$$Q'(\vec{x}) \;\; \text{:–} \;\; S_1(\vec{v}_1), \ldots, S_l(\vec{v}_l)$$

  with $l < k$

- Then Q is equivalent to a query of the form

$$Q'(\vec{x}) \;\; \text{:–} \;\; R_{i_1}(\vec{u}_{i_1}), \ldots, R_{_l}(\vec{u}_{i_l})$$

- 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 $f : T' \rightarrow T' - \{t\}$
        then $T' := T' - \{t\}$
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

$$Q \quad = \quad \pi_{AB}(\sigma_{B=4}(R)) \bowtie \pi_{BC}(\pi_{AB}(R) \bowtie \pi_{AC}(\sigma_{B=4}(R)))$$

- Translate into relational calculus:

$$(\exists z_1\, R(x, y, z_1) \wedge y = 4) \wedge \exists x_1\, ((\exists z_2\, R(x_1, y, z_2)) \wedge (\exists y_1\, R(x_1, y_1, z) \wedge y_1 = 4))$$

- Simplify, by substituting the constant, and putting quantifiers forward:

$$\exists x_1, z_1, z_2\, (R(x, 4, z_1) \wedge R(x_1, 4, z_2) \wedge R(x_1, 4, z) \wedge y = 4)$$

- Conjunctive query:

$$Q(x, y, z) \;\; \text{:–} R(x, 4, z_1), R(x_1, 4, z_2), R(x_1, 4, z), y = 4$$

Minimizing SPJ/conjunctive queries cont'd

- Tableau T:

A	B	C
x	4	z_1
x_1	4	z_2
x_1	4	z
x	4	z

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

A	B	C
x_1	4	z_2
x_1	4	z
x	4	z

- 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

- Step 2: Is T equivalent to $$\begin{array}{ccc} \text{A} & \text{B} & \text{C} \\ \hline x & 4 & z_1 \\ x_1 & 4 & z \\ \hline x & 4 & z \end{array}$$
- Answer: Yes. Homomorphism f: $f(z_2) = z$, all other variables stay the same.
- The new tableau is not equivalent to
$$\begin{array}{ccc} \text{A} & \text{B} & \text{C} \\ \hline x & 4 & z_1 \\ \hline x & 4 & z \end{array} \qquad \text{or} \qquad \begin{array}{ccc} \text{A} & \text{B} & \text{C} \\ \hline x_1 & 4 & z \\ \hline x & 4 & z \end{array}$$
- Because $f(x) = x, f(z) = z$, and the image of one of the rows is not present.

Minimizing SPJ/conjunctive queries cont'd

- Minimal tableau:

A	B	C
x	4	z_1
x_1	4	z
x	4	z

- Back to conjunctive query:

$$Q'(x, y, z) \ \text{:-}\ R(x, y, z_1), R(x_1, y, z), y = 4$$

- An SPJ query:

$$\sigma_{B=4}(\pi_{AB}(R) \bowtie \pi_{BC}(R))$$

- Pushing selections:

$$\pi_{AB}(\sigma_{B=4}(R)) \ \bowtie \ \pi_{BC}(\sigma_{B=4}(R))$$

Review of the journey

- We started with

$$\pi_{AB}(\sigma_{B=4}(R)) \bowtie \pi_{BC}(\pi_{AB}(R) \bowtie \pi_{AC}(\sigma_{B=4}(R)))$$

- Translated into a conjunctive query
- Built a tableau and minimized it
- Translated back into conjunctive query and SPJ query
- Applied algebraic equivalences and obtained

$$\pi_{AB}(\sigma_{B=4}(R)) \;\bowtie\; \pi_{BC}(\sigma_{B=4}(R))$$

- 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:

$$Q(x, y) \text{:–} B(x, y), R(y, z), R(y, w), R(w, y)$$

- Tableau:

B:

A	B
x	y

R:

A	B
y	z
y	w
w	y

---

x y

- 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

B:

A	B
x	y

R:

A	B
y	w
w	y

---

x y

- This cannot be further reduced, as for any homomorphism f, $f(x) = x$, $f(y) = y$.
- Thus Q is equivalent to

$$Q'(x,y) \;\; \text{:-} \;\; B(x,y), R(y,w), R(w,y)$$

- 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 $A \rightarrow\!\!\!\rightarrow B$ and thus

$$R \quad = \quad \pi_{AB}(R) \bowtie \pi_{AC}(R)$$

- Tableaux can help with these optimizations!
- $\pi_{AB}(R) \bowtie \pi_{AC}(R)$ as a conjunctive query:

$$Q(x, y, z) \text{:–} R(x, y, z_1), R(x, y_1, z)$$

Query optimization and integrity constraints cont'd

- Tableau:

A	B	C
x	y	z_1
x	y_1	z
x	y	z

- Using the FD $A \rightarrow B$ infer $y = y_1$
- Next, minimize the resulting tableau

A	B	C
x	y	z_1
x	y	z
x	y	z

$\longrightarrow$

A	B	C
x	y	z
x	y	z

- And this says that the query is equivalent to $Q'(x, y, z)$:–$R(x, y, z)$, 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 $\Rightarrow$
                            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
```

Query optimization and integrity constraints: example

- R is over A, B, C; F contains $B \to A$
- $Q = \pi_{BC}(\sigma_{A=4}(R)) \bowtie \pi_{AB}(R)$
- Q as a conjunctive query:

$$Q(x, y, z) \ \text{:-}\ R(4, y, z), R(x, y, z_1)$$

- Tableau:

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

- 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:

$$Q \quad = \quad \pi_{BC}(\sigma_{A=4}(R)) \bowtie \pi_{AB}(\sigma_{A=5}(R))$$

- As a conjunctive query:

$$Q(x, y, z) \ \text{:-}\ R(4, y, z), R(x, y, z_1), x = 5$$

- Tableau:

$$\begin{array}{ccc} \text{A} & \text{B} & \text{C} \\ \hline 4 & y & z \\ 5 & y & z_1 \\ \hline 5 & y & z \end{array} \qquad \xrightarrow{\text{CHASE}} \qquad \emptyset$$

- 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 $A \to B$, the query is

$$Q \quad = \quad \pi_{AB}(R) \bowtie \pi_A(\sigma_{B=4}(R)) \bowtie \pi_{AB}(\pi_{AC}(R) \bowtie \pi_{BC}(R))$$

- Convert into conjunctive query:

$$Q(x, y) \ \text{:--}\ R(x, y, z_1), R(x, y_1, z), R(x_1, y, z), R(x, 4, z_2)$$

- Tableau:

$$\begin{array}{ccc} \mathsf{A} & \mathsf{B} & \mathsf{C} \\ \hline x & y & z_1 \\ x & y_1 & z \\ x_1 & y & z \\ x & 4 & z_2 \\ \hline x & y & \end{array} \quad \xrightarrow{\text{CHASE}} \quad \begin{array}{ccc} \mathsf{A} & \mathsf{B} & \mathsf{C} \\ \hline x & \mathsf{4} & z_1 \\ x & \mathsf{4} & z \\ x_1 & \mathsf{4} & z \\ x & 4 & z_2 \\ \hline x & \mathsf{4} & \end{array} \quad \xrightarrow{\text{minimize}} \quad \begin{array}{ccc} \mathsf{A} & \mathsf{B} & \mathsf{C} \\ \hline x & \mathsf{4} & z \\ \hline x & \mathsf{4} & \end{array}$$

Query optimization and integrity constraints: example cont'd

- $\begin{array}{ccc} \mathsf{A} & \mathsf{B} & \mathsf{C} \\ \hline x & 4 & z \\ \hline x & 4 & \end{array}$ is translated into

$$Q(x,y) \ \text{:-}\ R(x,y,z), y = 4$$

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

  $$\pi_{AB}(R) \bowtie \pi_A(\sigma_{B=4}(R)) \bowtie \pi_{AB}(\pi_{AC}(R) \bowtie \pi_{BC}(R)) \ = \ \pi_{AB}(\sigma_{B=4}(R))$$

  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: $\bowtie [X_1, \ldots, X_m]$
- It holds in a relation R iff

$$R = \pi_{X_1}(R) \bowtie \ldots \bowtie \pi_{X_m}(R)$$

- Important for decomposing relations and normalizing databases
- An FD $X \to Y$ over attributes U implies a JD $\bowtie [XY, X(U - Y)]$
  - ○ a simple exercise
- Let $\mathcal{F}$ be a set of FDs, $\mathcal{J}$ a set of JDs, and θ a dependency (FD or JD)
- $\mathcal{F}, \mathcal{J} \models \theta$ (in words, $\mathcal{F}$ and $\mathcal{J}$ imply θ) if for every relation R, if all of $\mathcal{F}$ and $\mathcal{J}$ dependencies are true in R, then θ 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:

A	B	C
$\mathbf{x}$	$\mathbf{y}$	x_1
x_2	$\mathbf{y}$	$\mathbf{z}$
x_2	$\mathbf{y}$	x_3

- 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 $chase_{\mathcal{F},\mathcal{J}}(T)$

CHASE for dependency implication

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

- Construct a tableau T_θ
- 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_θ has two rows: $(\mathbf{x}_1, \ldots, \mathbf{x}_m)$ and $(\mathbf{x}_1, \ldots, \mathbf{x}_k, y_{k+1}, \ldots, y_m)$
- Condition: $chase_{\mathcal{F},\mathcal{J}}(T_\theta)$ has only distinguished variables for A_{k+1}

Example: $\{\bowtie[AB, AC],\ AB \rightarrow C\}\ \models\ A \rightarrow C$

$T_{A\rightarrow C}$:

A	B	C
$\mathbf{x}$	$\mathbf{y}$	$\mathbf{z}$
$\mathbf{x}$	x_1	x_2

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

A	B	C
$\mathbf{x}$	$\mathbf{y}$	$\mathbf{z}$
$\mathbf{x}$	x_1	x_2
$\mathbf{x}$	$\mathbf{y}$	x_2

Then use $AB \rightarrow C$ and get

A	B	C
$\mathbf{x}$	$\mathbf{y}$	$\mathbf{z}$
$\mathbf{x}$	x_1	$\mathbf{z}$

Only distinguished variables in column C.

CHASE for JDs

- Let θ be $\bowtie [X_1, \ldots, X_n]$.
- T_θ has n rows.
- The ith 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: $chase_{\mathcal{F},\mathcal{J}}(T)$ 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

$$\mathsf{AC}^0 \subsetneq \mathsf{TC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{DLOG} \subseteq \mathsf{NLOG} \subseteq \mathsf{PTIME}$$

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

Beyond PTIME

$$\mathrm{PTIME} \subseteq \left\{ \begin{array}{l} \mathrm{NP} \\ \mathrm{coNP} \end{array} \right\} \subseteq \mathrm{PSPACE}$$

- 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:

Flights	Src	Dest
	'EDI'	'LHR'
	'EDI'	'EWR'
	'EWR'	'LAX'
	...	...

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

```
  SELECT F1.Src, F2.Dest
  FROM Flights F1, Flights F2
  WHERE F1.Dest=F2.Src
UNION
  SELECT * FROM Flights
```

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:

```
  SELECT F1.Src, F3.Dest
  FROM Flights F1, Flights F2, Flights F3
  WHERE F1.Dest=F2.Src AND F2.Dest=F3.Src
UNION
  SELECT F1.Src, F2.Dest
  FROM Flights F1, Flights F2
  WHERE F1.Dest=F2.Src
UNION
  SELECT * FROM Flights
```

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:

$$\begin{aligned} reach(x,y) &\text{ :– } \mathit{flights}(x,y) \\ reach(x,y) &\text{ :– } \mathit{flights}(x,z), \;\; reach(z,y) \end{aligned}$$

- One of these rules is $recursive$: $reach$ refers to itself.
- Evaluation:
  - Step 0: $reach_0$ is initialized as the empty set.
  - Step $i+1$: Compute

$$\begin{aligned} reach_{i+1}(x,y) &\text{ :– } \mathit{flights}(x,y) \\ reach_{i+1}(x,y) &\text{ :– } \mathit{flights}(x,z), \;\; reach_i(z,y) \end{aligned}$$

  - Stop condition: If $reach_{i+1} = reach_i$, then it is the answer to the query.

Evaluation of recursive queries

- Example: assume that $\mathit{flights}$ contains $(a, b), (b, c), (c, d)$.
- Step 0: $\mathit{reach} = \emptyset$
- Step 1: reach becomes $\{(a, b), (b, c), (c, d)\}$.
- Step 2: reach becomes $\{(a, b), (b, c), (c, d), (a, c), (b, d)\}$.
- Step 3: 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 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:

```
WITH RECURSIVE Reach(Src,Dest) AS
 (
    SELECT * FROM Flights
  UNION
    SELECT F.Src, R.Dest
    FROM Flights F, Reach R
    WHERE F.Dest=R.Src
 )
SELECT * FROM Reach
```

Recursion in SQL3: syntactic restrictions

- There is another way to do reachability as a recursive rule-based query:

$$\begin{array}{l} reach(x,y) \text{ :- } flights(x,y) \\ reach(x,y) \text{ :- } reach(x,z),\;\; reach(z,y) \end{array}$$

- This translates into an SQL3 query:

```
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:

```
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:

$$r(x) \;\text{:-}\; s(x), \neg r(x)$$

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: $r_3 = \{1, 2\}$;

  $\ldots$

  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

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

   Criterion for CQ containment/equivalence

2. Mihalis Yannakakis: Algorithms for Acyclic Database Schemes. VLDB 1981: 82-94

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

3. Georg Gottlob, Nicola Leone, Francesco Scarcello: The complexity of acyclic conjunctive queries. Journal of the ACM 48(3):431-498 (2001)

   An in-depth study of acyclicity

4. Georg Gottlob, Nicola Leone, Francesco Scarcello: Hypertree Decompositions and Tractable Queries. J. Comput. Syst. Sci. 64(3):579-627 (2002)

   A hierarchy of classes of efficient CQs, the bottom level of which is acyclic queries

5. Martin Grohe, Thomas Schwentick, Luc Segoufin: When is the evaluation of conjunctive queries tractable? STOC 2001: 657-666

   A different way of characterizing efficiency of CQs, this time via the notion of bounded treewidth

6. Moshe Y. Vardi: The Complexity of Relational Query Languages (Extended Abstract) .STOC 1982: 137-146

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

7. Christos H. Papadimitriou, Mihalis Yannakakis: On the Complexity of Database Queries. J. Comput. Syst. Sci. 58(3): 407-427 (1999)

   A finer way of measuring complexity, between data and combined

8. Neil Immerman: Languages that Capture Complexity Classes. SIAM J. Comput. 16(4): 760-778 (1987)

   Query languages that correspond to complexity classes

9. Martin Grohe: Fixed-point definability and polynomial time on graphs with excluded minors. Journal of the ACM 59(5): 27 (2012)

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

10. Phokion G. Kolaitis, Moshe Y. Vardi: Conjunctive-Query Containment and Constraint Satisfaction. J. Comput. Syst. Sci. 61(2): 302-332 (2000)

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

11. Martin Grohe: From polynomial time queries to graph structure theory. Commun. ACM 54(6): 104-112 (2011)

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

12. 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

13. Leonid Libkin: Expressive power of SQL. Theor. Comput. Sci. 296(3):

379-404 (2003)

... and a specific application for SQL

14. David Maier, Alberto O. Mendelzon, Yehoshua Sagiv: Testing Implications of Data Dependencies. ACM Trans. Database Syst. 4(4): 455-469 (1979)

    The paper that proposed CHASE

15. Alin Deutsch, Alan Nash, Jeffrey B. Remmel: The chase revisited. PODS 2008: 149-158

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