Introduction

Faculty of Computer Science
Technion – Israel Institute of Technology
Winter 2018-2019

Course 236378

Assumed Background

- Basic database theory
  - Relational model, schema, integrity constraints, database querying, relational algebra
- Basic algorithms and complexity
  - Asymptotic running time, polynomial time, NP, reduction, completeness
- Basic probability theory
  - Probability spaces, random variables, conditional probability

Requirements

1. Home assignments
   - 4 x dry (18% each)
   - 1 x wet&dry (28%)

2. Mandatory attendance
   - Contact me in advance if you are having a problem attending a specific lecture

Pre-Relational Databases

- Cross-app solutions for data store/access proposed already in the 1960s
- Examples:
  - The CODASYL committee standardized a network data model (CODASYL Data Model)
    - A network of entities linked to each other, very similar to object-oriented models
  - Integrated Data Stores (Charles Bachman)
    - High-performance graph database from 1964 (!)
  - IBM's Information Management System (IMS)
    - Driven (in 1966) by the Apollo program
    - Hierarchical data model, index and transaction support

1970's Dispute

General-purpose DB to be abstracted as a network

As logic!

Edgar F. Codd (1922-2003)

C. W. Bachman (1924-2017)
Codd’s Vision (1)

- 1970: Codd invents the relational database model
  - Idea: interface via First-Order Logic!
  - Data = collection of relations, interconnected via keys
  - Relations conform to a schema
  - Questions via a query language over the schema
  - System translates queries into actual execution plans
- Principle: separate logical from physical layers
- Work done at IBM San Jose, now IBM Almaden

Codd’s Vision (2)

- 1970-1972: Codd introduced the relational algebra and the relational calculus
  - Algebraic and logical QLs, respectively
  - Proved their equal expressive power
  - [E. F. Codd: Relational Completeness of Data Base Sublanguages. IBM Research Report RJ987 (1972)]

Codd Catches On (1)

- 1973: Michael Stonebraker and Eugene Wong implement Codd’s vision in INGRES
  - Commercialized in 1983
  - Evolved to Postgres (now PostgreSQL) in 1989

Codd Catches On (2)

- 1974: A group from the IBM San Jose lab implements Codd’s vision in System R, which evolved to DB2 in 1983
  - SQL initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce
- 1977: Influenced by Codd, Larry Ellison founds Software Development Labs
  - Becomes Relational Software in 1979
  - Becomes Oracle Systems Corp (1982), named after its Oracle database product

Relational Databases Still Popular?

- MySQL 58.7%
- SQL Server 41.2%
- PostgreSQL 32.9%
- MongoDB 25.9%
- SQLite 19.7%
- Redis 16.0%
- Elasticsearch 14.1%
- MariaDB 13.4%
- Oracle 11.1%

Yes!

Most popular databases in 2018 according to StackOverflow survey

Publication Venues for DB Research

- **Conferences:**
  - **General:**
    - SIGMOD: ACM Special Interest Group on Management of Data (since 1975)
    - VLDB: Intl. Conf. on Very Large Databases (since 1975)
    - ICDE: IEEE Intl. Conf. on Data Engineering (since 1984)
  - **Theory oriented:**
    - ICDT: Intl. Conference on Database Theory (since 1986)
- **Journals:**
  - ACM Transactions on DB Systems (since 1976)
  - The VLDB Journal (since 1992)
  - ACM SIGMOD Record (since 1969)

Selected Database Research Topics*

<table>
<thead>
<tr>
<th>1980</th>
<th>1990</th>
<th>2000</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System Design</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Distributed storage, in-memory, recovery</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Query Languages</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Codasyl, SQL, recursion, nesting</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Schema Design</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- ER models, normal forms, dependency</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Transaction &amp; Concurrency</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Query process &amp; util</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DB Performance</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Data integration</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Models</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Multimodal, DMM, Text, XML</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Logic</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Deductive (Datalog)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Incompleteness (null)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Entity Resolution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Further XML</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Query and optimization</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Database Security</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- View-based access</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Database Privacy</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Incremental maintain</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Crowdsourcing</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Using crowd input in databases</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Models</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Streaming data</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Models</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Social Networks &amp; Social Media</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Models</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Data models</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DB &amp; AI</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- DB for search</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Entity Resolution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Search for DB</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Entity Resolution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Semantic Web (RDF, ontologies)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Entity Resolution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- NoSQL (doc, graph, key-value)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Entity Resolution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Data Merging &amp; Discovery</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Entity Resolution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Data exchange</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Entity Resolution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Cloud Databases</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Entity Resolution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Categorical Stores</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* Based on SIGMOD session topics from DBLP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Turing Awards for Database Research

1973
CHARLES WILLIAM BACHMAN
Goldwater – 1963
For his outstanding contributions to database technology.

1981
EDGAR F. (“TED”) CODD
Goldwater – 1960
For fundamental contributions to the theory and practice of database management systems.

1998
JAMES ("Jim") NICHOLAS GRAY
Goldwater – 1969
For fundamental contributions to database and transaction processing research and technical leadership in system implementation.

2014
MICHAEL STONEBRAKER
Goldwater – 1966
For fundamental contributions to the concepts and practice underpinning modern database systems.

Some Modern Database Content

2016 Dagstuhl Perspective Workshop

Gathering of top database theorists, evaluating the field and planning ahead
In the Course

• Foundations of database complexity
  – Data/combined complexity, join acyclicity, hypertree width

• Principled, application-independent paradigms to managing uncertainty in data
  – Incomplete / inconsistent / probabilistic databases
  – Two key aspects for every paradigm:
    • Representation & Semantics
      – How do we represent what we know? What is missing? What is conflicting?
        What is our confidence?
    • Query evaluation
      – What is the meaning of query answering in the presence of uncertainty?
        What is the computational complexity?

Principles of Managing Uncertain Data: Introduction
BASIC DATABASE CONCEPTS

Schema and Databases

• A database schema is a finite set of relation names, each mapped into a relation schema
  – Ex: `Student(sid, name, year)` , `Course(cid, topic)` , `Studies(sid, cid)`

• A database (or instance) over a schema consists of a relation for each relation schema

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Studies</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>name</td>
<td>year</td>
</tr>
<tr>
<td>861</td>
<td>Alma</td>
<td>2</td>
</tr>
<tr>
<td>753</td>
<td>Amir</td>
<td>1</td>
</tr>
<tr>
<td>955</td>
<td>Ahuna</td>
<td>2</td>
</tr>
<tr>
<td>cid</td>
<td>topic</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>PL</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>DB</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>CS</td>
<td></td>
</tr>
<tr>
<td>sid</td>
<td>cid</td>
<td></td>
</tr>
<tr>
<td>861</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>861</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>753</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>955</td>
<td>76</td>
<td></td>
</tr>
</tbody>
</table>

Integrity Constraints in Databases

What are Integrity Constraints?

• Schema-level (data-independent) specs on how records should behave
  – Beyond the mere relational structure
  – (e.g., students with the same ID have the same name, take the same courses, etc.)

• DBMS guarantees that constraints are always satisfied, blocking actions that cause violations

• What if we get data that violates the constraints to begin with??
  – Wait for “inconsistent databases”

Why Integrity Constraints?

• Development and maintenance
  – Consistency assured without custom code

• Optimization: operations may be optimized if we know that some constraints hold
  – Once a sought student ID is found, you can stop; you won’t find it again

  – Queries typically join on keys, so key-based indexes can provide a faster access
Querying: Which Courses Avia Took?

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Avia</td>
<td>Haifa</td>
</tr>
<tr>
<td>2345</td>
<td>Boris</td>
<td>Vachar</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>C</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>name</td>
<td>number</td>
</tr>
<tr>
<td>365</td>
<td>EB</td>
<td>Avia</td>
</tr>
<tr>
<td>310</td>
<td>FL</td>
<td>Barky</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>number</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>363</td>
<td>95</td>
</tr>
<tr>
<td>2345</td>
<td>310</td>
<td>85</td>
</tr>
</tbody>
</table>

Assembly
...
mov $1, srax
mov $1, srax
mov message, srax
mov $message, srax
test
mov $100, srax
xor srax, srax
...

RC Query

- Person(id, gender, country)
- Parent(parent, child)
- Spouse(person1, person2)

\[
\{ (x,y) \mid \text{Person}(x, 'female', 'Canada') \land \\
\exists y, z [\text{Parent}(y,x) \land \text{Parent}(z,y) \land \\
\exists w [\text{Parent}(z,w) \land w \neq w \land (w = w \lor \text{Spouse}(u, w))] \}
\]

Equivalence Between RA and D.I. RC

THEOREM: RA and domain-independent RC have the same expressive power.

More precisely:

- For every RA expression \( E \) there is a domain-independent RC query \( Q \) such that \( Q \equiv E \)
- For every domain-independent RC query \( Q \) there is an RA expression \( E \) such that \( Q \equiv E \)

\( Q \equiv E \) means that \( Q(D) = E(D) \) for all \( D \)

Relational Algebra

- Primitive operators:
  1. Projection (\( \pi \))
  2. Selection (\( \sigma \))
  3. Renaming (\( \rho \))
  4. Union (\( \cup \))
  5. Difference (\( \setminus \))
  6. Cartesian Product (\( \times \))

- Natural join (\( \bowtie \)) can be defined using \( p \times x \pi y \)

- Conjunctive Queries (CQ): \( \bowtie \pi \rho \)

- Unions of CQs (UCQs, "positive RA"): \( \bowtie \pi \rho \cup \)

Domain Independence

What is the Meaning of the Following?

\[
\{ (x) \mid \text{Person}(x, 'female', 'Canada') \}
\]

Well defined in FO where the structure contains a domain, bad for DBs

\[
\{ (x,y) \mid \exists z [\text{Likes}(x,z) \land y = z] \}
\]

Informally, an RC formula is domain independent if its result depends only on the database, not on the domain

Conjunctive Queries (Join Queries)

Friends(x,y), School(x,y), School(y,z)
Friend from the same high school

Colleagues(x,y), Univ(x,y), Univ(y,z)
Colleague from the same university

Married(x,y), Parent(x,y), Parent(y,x)
Spouse with a common child

Same(x):

- Artist(x)
- Friends(x,y), School(x,y), School(y,z)
- Colleagues(x,y), Univ(x,y), Univ(y,z)
- Married(x,y), Parent(x,y), Parent(y,x)
- Same(x,y), Same(x,z), Same(x,a)
Join Algorithms

- Classic algorithms select a join ordering with “easier” intermediate joins [Selinger+1979]
- Yannakakis [1981] for acyclic queries
  - Generalize to cyclic queries with low hypertree width
- New breed of joiners: worst-case optimal
  - Ngo, Porat, Ré, Rudra [2012]
  - Meet the Atserias-Grohe-Marx [2008] bound
  - Example: $R(x_1,x_2) \bowtie S(x_3,x_4) \bowtie T(x_5,x_6) - n^2 \leq n^{1.5}$
- In-memory, scan all relations simultaneously
  - NPRR [2012], Leapfrog Trie Join [Veldhuizen2014], Minesweeper [Ngo+2014], Cached Trie Join [Kalinsky+2017]

Complexity of Joins

Which join is more complicated?
In what sense?

$\Lambda_{1 \leq i < j \leq n} R(x_i,x_j)$

$\Lambda_{1 \leq i < j \leq n} R(x_i,x_{i+1}) \land R(x_n,x_1)$

In the Course

- How big can the result of a join be?
  - AGM bound
- How does acyclicity help in join computation?
  - The Yannakakis algorithm
- How to quantify the tree-likelihood of a graph?
  - Bounded treewidth
- What is acyclicity in general (non-binary) joins?
  - Hypergraph $\alpha$-acyclicity
- What is tree-likelihood in hypergraphs?
  - Bounded hypertree width

Missing Information

- Problem: pieces of data missing, but we need to keep whatever partial knowledge we have

  Registrations
<table>
<thead>
<tr>
<th>student</th>
<th>course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>PL</td>
</tr>
</tbody>
</table>

  Courses
<table>
<thead>
<tr>
<th>course</th>
<th>lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL</td>
<td>Eran</td>
</tr>
</tbody>
</table>

  A source tells us that Alon is a student of Keren
  - How can we represent it in our DB?

  Registrations
<table>
<thead>
<tr>
<th>student</th>
<th>course</th>
<th>lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alon</td>
<td>PL</td>
<td>Keren</td>
</tr>
</tbody>
</table>
SQL’s NULL

- NULL is SQL’s special “missing value"
- Same queries as complete tables, but SQL assigns a special behavior to logic over NULL
  - “Three-valued logic”: true, false, unknown
- Alas, there are some issues...

Try It Yourself (psql)

CREATE TABLE Registrations(
    student varchar(40),
    course varchar(40)
);

INSERT INTO Registrations VALUES ('Ahuva', 'PL'), ('Alon', NULL);

CREATE TABLE Courses(
    course varchar(40),
    lecturer varchar(40)
);

INSERT INTO Courses VALUES ('PL', 'Eran'), (NULL, 'Keren');

SELECT student, lecturer
FROM Registrations R, Courses C
WHERE R.course = C.course;

Of course, we’ve lost our initial association (join)...

Labeled Nulls in “Naive” Tables

- Just like nulls, but each null has a name
  - We do not know what the value is, but we do know that two nulls with the same name are the same

Possible Worlds

Closed-World Assumption:

Open-World Assumption:

Semantics of Query Answering

Incomplete DB

Possible Worlds
Semantics of Query Answering

In a Global Schema, a student source instance can relate to a set of students, 
\[ S \]
where 
\[ \Sigma = \text{StudLecturer}(x,y) \rightarrow 3x \text{ Registrations}(x,y) \land \text{Course}(x,y) \]

The Clio Project

IBM + U. Toronto – tool for data exchange
Commercialized in IBM DB2

Application: Data Exchange

Formalism [Fagin et al. 05]

A schema mapping is defined by a source schema \( S \), a target schema \( T \), and a set \( \Sigma \) of logical assertions stating how \( S \) relates to \( T \).

Formalism [Fagin et al. 05]

A schema mapping is defined by a source schema \( S \), a target schema \( T \), and a set \( \Sigma \) of logical assertions stating how \( S \) relates to \( T \).

\[ S \]

\[ T \]

\[ \Sigma \]

?? We don’t have z! So 2 options:
1) *Abort*
2) Do our best to max usability
Problems Studied in Data Exchange

- Materialization (covered)
  - Many solutions exist; what makes one solution “better” than another? If there a “best” solution? How to find it?
- Target query answering (covered)
  - Given a source instance and a query over the target, evaluate the query (semantics / complexity)
- Manipulating schema mappings (not covered)
  - Composition and inversion of mappings

Inconsistency

- An inconsistent database contains inconsistent (or impossible) information
  - Two students have the same ID
  - A student gets credit for the same course twice
  - A student takes a course that is not listed in the course database
  - A student has a grade for this course but a grade is missing for an assignment
- Modeling: \((D, \Sigma)\) where \(D\) is a database, \(\Sigma\) a set of integrity constraints over DBs; alas, \(D\) violates \(\Sigma\)

Query Answering

<table>
<thead>
<tr>
<th>Grades</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>student</td>
<td>course</td>
</tr>
<tr>
<td>Ahuva</td>
<td>PL</td>
</tr>
<tr>
<td>Alon</td>
<td>PL</td>
</tr>
<tr>
<td>Alon</td>
<td>PL</td>
</tr>
</tbody>
</table>

Functional Dependency: student, course \(\rightarrow\) grade

Integrity Constraints I

\[\text{SELECT student FROM Grades G, Courses C WHERE G.grade >= 85 AND G.course = C.course AND C.lecturer='Eran'}\]

Database D

<table>
<thead>
<tr>
<th>Grades</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>student</td>
<td>course</td>
</tr>
<tr>
<td>Ahuva</td>
<td>PL</td>
</tr>
<tr>
<td>Alon</td>
<td>PL</td>
</tr>
<tr>
<td>Alon</td>
<td>PL</td>
</tr>
</tbody>
</table>

Functional Dependency: student, course \(\rightarrow\) grade

Integrity Constraints I

\[\text{SELECT student FROM Grades G, Courses C WHERE G.grade >= 80 AND G.course = C.course AND C.lecturer='Eran'}\]
**Minimal Repairs**

[Arenas, Bertossi, Chomicki 99]:

**DEFINITION:** Let $(D, \Sigma)$ be an inconsistent DB. A repair is a DB $D'$, such that:

1. $D'$ is consistent (with respect to $\Sigma$)
2. $D'$ differs from $D$ in a “minimal way”

<table>
<thead>
<tr>
<th>Grades</th>
<th>student</th>
<th>course</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>PL</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>Alon</td>
<td>PL</td>
<td>86</td>
<td></td>
</tr>
</tbody>
</table>

Inconsistent database $D$

<table>
<thead>
<tr>
<th>Grades</th>
<th>student</th>
<th>course</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>PL</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>Alon</td>
<td>PL</td>
<td>81</td>
<td></td>
</tr>
</tbody>
</table>

Repair $D''$

**Semantics of Query Answering**

Inconsistent DB

$Q \rightarrow ?$

Repairs (consistent DBs)

$Q \rightarrow a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow \cdots \rightarrow a_n$

Consistent Answers

$Q \rightarrow ?$

Repairs (consistent DBs)

**Algorithms / Complexity**

Koutris & Wijsen [2015]: For consistent query answering with key constraints, we know how Select-Project-Join queries w/o self joins are classified into 3 categories:

1. Inconsistent DB
   - Ignore inconsistency
   - $Q \rightarrow Q' \rightarrow \neg a_i$
   - Rewriting

2. Inconsistent DB
   - coNP-complete
   - (exptime under standard complexity assumptions)
   - Efficient algorithm

3. Consistent
   - $Q \rightarrow Q'$
   - ALG
   - $\neg a_i$

**Probabilistic Databases**
How to accommodate the probabilistic nature of data at the DB & query level?

- Find the students that are employed as engineers
- How many students work at Intel?
- Is any PM a Technion student?

<table>
<thead>
<tr>
<th>Student</th>
<th>University</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>Technion</td>
</tr>
<tr>
<td>Alon</td>
<td>HaifaU</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee</th>
<th>Employer</th>
<th>Role</th>
<th>Pr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>Intel</td>
<td>Eng</td>
<td>0.7</td>
</tr>
<tr>
<td>Alon</td>
<td>Intel</td>
<td>PM</td>
<td>0.8</td>
</tr>
<tr>
<td>Alon</td>
<td>Google</td>
<td>Eng</td>
<td>0.4</td>
</tr>
<tr>
<td>Alon</td>
<td>Intel</td>
<td>PM</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Database Semantics

Semantics of Query Answering

Rep of the probability space

Mapping tuple \( \rightarrow \) marginal probability
Algorithms for Query Answering

- Dalvi & Suciu dichotomy: positive RA can be fully classified into:
  - Queries that can be solved in polynomial time
    - By repeated decomposition into simpler queries
  - Queries for which answering is \#P-hard
    - Hence, cannot be computed in polynomial time under standard complexity assumptions

- Practical solvers via Knowledge Compilation

- Guaranteed approximation via sampling
  - Additive approx. $p \pm \epsilon$ is simple
  - Multiplicative approx. $(1 \pm \epsilon)p$ requires more work

---

PLANNED SCHEDULE

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>21/10</td>
<td>Introduction</td>
</tr>
<tr>
<td>26/10</td>
<td>Basics Review</td>
</tr>
<tr>
<td>04/11</td>
<td>No lecture (at WEPA2018)</td>
</tr>
<tr>
<td>11/11</td>
<td>Querying Complexity 2</td>
</tr>
<tr>
<td>18/11</td>
<td>Acyclic Joins</td>
</tr>
<tr>
<td>25/11</td>
<td>Incomplete Information</td>
</tr>
<tr>
<td>02/12</td>
<td>No lecture (Hanukkah)</td>
</tr>
<tr>
<td>09/12</td>
<td>Data Exchange</td>
</tr>
<tr>
<td>20/12</td>
<td>Inconsistent Databases</td>
</tr>
<tr>
<td>16/12</td>
<td>Probabilistic Diffs</td>
</tr>
<tr>
<td>Assignment 5 due 23/12</td>
<td></td>
</tr>
<tr>
<td>22/12</td>
<td>Extras</td>
</tr>
<tr>
<td>Assignment 5 due 23/01</td>
<td></td>
</tr>
<tr>
<td>24/01</td>
<td>Semester ends</td>
</tr>
</tbody>
</table>

Connect to Google Calendar