Faculty of Computer Science
Technion – Israel Institute of Technology

Database Management Systems

מרכזת מסדי נתונים

Lecture 1:
Introduction
Course Administration
Course Staff

• Lecturer: Benny Kimelfeld
  – bennyk@cs.technion.ac.il, Taub 631

• Teaching assistants:
  – Liat Peterfreund (in charge)
    • liatpf@cs.technion.ac.il
    • Taub 642
  – Dvir Dukhan
    • dvir.dukhan@campus.technion.ac.il
    • TD&K lab (645)
  – Shoval Lagziel
    • shovallagziel@gmail.com
    • Taub 227
  – Yoav Nahshon
    • yoavn@cs.technion.ac.il
    • TD&K lab (645)
Course Requirements

• Final exam (80%)
  – “Open book”

• Home assignments (20%)
  – 2x dry, 1x wet, 1x wet+dry
  – More details in tutorial this week
(Semi-)Flipped Classroom

• Course material online, **home reading!**
  – Detailed presentations
• Questions & feedback online
  – Example: [https://goo.gl/forms/ofWE6agx8JFfEOYC3](https://goo.gl/forms/ofWE6agx8JFfEOYC3)
• In classroom:
  – Intro & background per topic
  – Nontrivial concepts / algorithms / proofs
  – Q&A on online material
  – Exercise (past exams, Kahoot, …)
  – Tasks for next home reading
Technical Background
Modeling Reality

• Real world:
  – Entities
  – Relationships (between entities)
  – Operations (changing aspects of entities & relationships)

• Example: Bank
  – Entities: customers, accounts, branches, employees
  – Relationships: customer-account, account-branch
  – Ops: add customer/account, deposit, withdrawal, balance, prediction

Database Management System (DBMS): faithful & efficient modeling of real-world information for specific applications
So, What is a Database?

Software system for creating, maintaining, updating, and querying the database

- General purpose—not for any specific application
- Interacts with a user (e.g., DBA) or an application

- Challenges:
  1. Modeling: data, query, consistency, security
  2. Engineering: solid / reliable implementation
  3. Performance: efficiency, scalability, hw utilization
What Services do Databases Provide?

• Centralized information management
  – Conceptual & physical data manager

• Core operations
  – Access control
  – A “smart” query processor
  – Transaction processing, ACID
  – Recovery

• Interoperability
  – Uniform data access across various platforms
  – Logical-physical independence
Why are Databases Needed?

• Facilitate (save time, skills, money)
  – Program in high levels of abstraction (concepts, entities, relationships, etc.)
  – No need for in-house implementation
    • Storage, disk, persistency, recovery, security, algs, etc.
  – Easier to accommodate architecture changes
  – Democratize data management (not only experts)

• Boost performance (often)
  – Adopt optimization & hardware utilization programmed already by the database vendor

• Safer software
  – The chance of bugs & security leaks reduces dramatically (past users suffered for us)
The IMDb Application

Frozen (I) (2013)
PG | 102 min | Animation, Adventure, Comedy | 27 November 2013 (USA)

Your rating: 7.6/10 from 369,436 users  Metascore: 74/100
Reviews: 876 user | 401 critic | 43 from Metacritic.com

When the newly crowned Queen Elsa accidentally uses her power to turn things into ice to curse her home in infinite winter, her sister, Anna, teams up with a mountain man, his playful reindeer, and a snowman to change the weather condition.

Directors: Chris Buck, Jennifer Lee
Writers: Jennifer Lee (screenplay), Hans Christian Andersen (inspired by the story "The Snow Queen" by), 4 more credits »
Stars: Kristen Bell, Idina Menzel, Jonathan Groff | See full cast and crew »

Won 2 Oscars. Another 76 wins & 53 nominations. See more awards »
Steps in Database Setup

• Requirement analysis
  – What information needs to be stored? How will it be used?

• Conceptual database design
  – Define/describe/discuss the semantic modeling of data in the application

• Logical database design
  – Translate conceptual design into a database schema

• Physical database design
  – Translate the database schema into a physical storage plan on available hardware (done by DBMS)
Faculty Example

• Design a database for the faculty’s administrative assistants

• Several types of *entities*:
  – Student: student name, id, address
  – Course: name, catalogue number, lecturer
  – Lecturer? Faculty? Building? Academic track?
    • Depending on the application needs

• Various *relationships*:
  – Student took course (and got a grade), lecturer teaches course, course-exam, exam-room, etc.
Data Modeling

An Entity-Relationship Diagram (ERD)
# Relational Design

Option 1: Single Table

**StudentCourseRegistry**

<table>
<thead>
<tr>
<th>sID</th>
<th>sName</th>
<th>sAddr</th>
<th>cNum</th>
<th>cName</th>
<th>cLecturer</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Avia</td>
<td>Haifa</td>
<td>363</td>
<td>DB</td>
<td>Anna</td>
<td>95</td>
</tr>
<tr>
<td>1234</td>
<td>Avia</td>
<td>Haifa</td>
<td>319</td>
<td>PL</td>
<td>Barak</td>
<td>82</td>
</tr>
<tr>
<td>2345</td>
<td>Boris</td>
<td>Nesher</td>
<td>319</td>
<td>PL</td>
<td>Barak</td>
<td>73</td>
</tr>
</tbody>
</table>

Option 2: Multiple Tables

**Student**

<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>Nesher</td>
</tr>
</tbody>
</table>

**Course**

<table>
<thead>
<tr>
<th>number</th>
<th>name</th>
<th>lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>363</td>
<td>DB</td>
<td>Anna</td>
</tr>
<tr>
<td>319</td>
<td>PL</td>
<td>Barak</td>
</tr>
</tbody>
</table>

**Took**

<table>
<thead>
<tr>
<th>sID</th>
<th>cNum</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>363</td>
<td>95</td>
</tr>
<tr>
<td>1234</td>
<td>319</td>
<td>82</td>
</tr>
<tr>
<td>2345</td>
<td>319</td>
<td>73</td>
</tr>
</tbody>
</table>
Option 1: Single Table

<table>
<thead>
<tr>
<th>sID</th>
<th>sName</th>
<th>sAddr</th>
<th>cNum</th>
<th>cName</th>
<th>cLecturer</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Avia</td>
<td>Haifa</td>
<td>363</td>
<td>DB</td>
<td>Anna</td>
<td>95</td>
</tr>
<tr>
<td>1234</td>
<td>Avia</td>
<td>Haifa</td>
<td>319</td>
<td>PL</td>
<td>Barak</td>
<td>82</td>
</tr>
<tr>
<td>2345</td>
<td>Boris</td>
<td>Nesher</td>
<td>319</td>
<td>PL</td>
<td>Barak</td>
<td>73</td>
</tr>
</tbody>
</table>

**StudentCourseRegistry**

**Advantages?**

**Drawbacks:**

- **Cost & redundancy:** Why should the student’s address be stored in each course she takes?
- **Insufficient expressiveness:** What about students that do not take any courses? Course w/o students?
- **Harder to maintain:** If a student changes address, need to update all records of relevant tuples; risk inconsistency or require more expensive controls
- **Harder to evolve:** If we wish to add the a semester column, every app will need to update its schema assumption
Relational Design

Option 1: Single Table

### StudentCourseRegistry

<table>
<thead>
<tr>
<th>sID</th>
<th>sName</th>
<th>sAddr</th>
<th>cNum</th>
<th>cName</th>
<th>cLecturer</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Avia</td>
<td>Haifa</td>
<td>363</td>
<td>DB</td>
<td>Anna</td>
<td>95</td>
</tr>
<tr>
<td>1234</td>
<td>Avia</td>
<td>Haifa</td>
<td>319</td>
<td>PL</td>
<td>Barak</td>
<td>82</td>
</tr>
<tr>
<td>2345</td>
<td>Boris</td>
<td>Nesher</td>
<td>319</td>
<td>PL</td>
<td>Barak</td>
<td>73</td>
</tr>
</tbody>
</table>

Option 2: Multiple Tables

### Student

<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>Nesher</td>
</tr>
</tbody>
</table>

### Course

<table>
<thead>
<tr>
<th>number</th>
<th>name</th>
<th>lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>363</td>
<td>DB</td>
<td>Anna</td>
</tr>
<tr>
<td>319</td>
<td>PL</td>
<td>Barak</td>
</tr>
</tbody>
</table>

### Took

<table>
<thead>
<tr>
<th>sID</th>
<th>cNum</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>363</td>
<td>95</td>
</tr>
<tr>
<td>1234</td>
<td>319</td>
<td>82</td>
</tr>
<tr>
<td>2345</td>
<td>319</td>
<td>73</td>
</tr>
</tbody>
</table>

Drawback: join required more often...

How can we formalize what “goodness” means?
Need to understand the connection between sID and sName, etc.
Integrity Constraints

• Schema-level spec (logic) on how records should behave
  – Beyond the relational structure (e.g., students with the same ID have the same name, take the same courses, etc.)
  – Schema-level: talks about object and relationship types; not concrete instances

• DBMS guarantees that constraints are always satisfied
  – By disabling actions that cause violations
Why Schema-Level Constraints?

• Safety
  – Consistency assured w/o custom code

• Development complexity
  – No reliance on consistency tests
  – But exceptions need to be handled

• Computational complexity
  – Operations may have smarter algorithms if we can rely on satisfaction of the constraints
    • e.g., once a sought student ID is found, you can stop; you won’t find it again
    • More interesting examples later in the course
### Which Constraints Should Hold Here?

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Took</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ID</strong></td>
<td><strong>number</strong></td>
<td><strong>sID</strong></td>
</tr>
<tr>
<td>1234</td>
<td>363</td>
<td>1234</td>
</tr>
<tr>
<td>2345</td>
<td>319</td>
<td>2345</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td><strong>name</strong></td>
<td><strong>cNum</strong></td>
</tr>
<tr>
<td>Avia</td>
<td>DB</td>
<td>363</td>
</tr>
<tr>
<td>Boris</td>
<td>PL</td>
<td>319</td>
</tr>
<tr>
<td><strong>addr</strong></td>
<td><strong>grade</strong></td>
<td><strong>grade</strong></td>
</tr>
<tr>
<td>Haifa</td>
<td>95</td>
<td>73</td>
</tr>
</tbody>
</table>

- No two tuples have the same ID (key constraint)
- Courses with the same number have the same name (functional dependency)
- A student cannot get two grades for the same course
- Grade must be > 53 (check constraint)
- sID is a Student.ID; cNum is a Course.number (referential constraint)
Querying: Which Courses Avia Took?

<table>
<thead>
<tr>
<th>S</th>
<th>C</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>name</td>
<td>addr</td>
</tr>
<tr>
<td>1234</td>
<td>Avia</td>
<td>Haifa</td>
</tr>
<tr>
<td>2345</td>
<td>Boris</td>
<td>Nesher</td>
</tr>
<tr>
<td></td>
<td>number</td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>363</td>
<td>DB</td>
</tr>
<tr>
<td></td>
<td>319</td>
<td>PL</td>
</tr>
<tr>
<td></td>
<td>sID</td>
<td>cNum</td>
</tr>
<tr>
<td>1234</td>
<td>363</td>
<td>95</td>
</tr>
<tr>
<td>1234</td>
<td>319</td>
<td>82</td>
</tr>
<tr>
<td>2345</td>
<td>319</td>
<td>73</td>
</tr>
</tbody>
</table>

Assembly

... mov $1, %rax
mov $1, %rdi
mov $message, %rsi
mov $13, %rdx
syscall
mov $60, %rax
xor %rdi, %rdi
...

Python

for s in S:
    for c in C:
        for t in T:
            if s.sName == 'Avia' and s.ID == t.sID and t.cNum == c.number:
                print c.name

QL

\[
\pi_{C.name}(\sigma_{s.name='Avia', number=cNum, ID=sID}(S \times C \times T))
\]

SQL

```
SELECT C.name
FROM S, C, T
WHERE S.name = 'Avia' AND S.ID = T.sID
    AND T.cNum = C.number
```

Algebra (RA)

\[
\pi_{C.name}(\sigma_{s.name='Avia', number=cNum, ID=sID}(S \times C \times T))
\]

Logic Programming (Datalog)

\[
Q(x) \leftarrow S(y, n, 'Avia'), C(z, x, l), T(y, z, g)
\]

Logic (RC)

\[
\{ (x) \mid \exists y, n, z, l, g \\
    [S(y, n, 'Avia') \land C(z, x, l) \land T(y, z, g)] \}
\]
**What is a Query Language?**

- A language for specifying how desired information is retrieved/derived from the database
- Usually, does not change the database
  - At least not the user-defined tables
- Specialized to the database model
  - As opposed to a general programming language
- In contrast, a *Data Definition Language (DDL)* is a language for manipulating (creating / updating / deleting) schemas and data
“Goodness” of a Query Language

• Simple
  – Users: easier to use
  – DBMS: easier to implement, easier to optimize

• High-level
  – Declare what, not program how
  – Users: easier, less control
  – DBMS: more flexibility, more responsibility

• Expressive
  – NOT: predefined queries; YES: ops w/ composition
  – Users: better
  – DBMS: harder to implement/optimize
<students>
  <student id="100026">
    <name>Joe Average</name>
    <age>21</age>
    <major>Biology</major>
    <results>
      <result course="Math 101" grade="C-"/>
      <result course="Biology 101" grade="C+"/>
      <result course="Statistics 101" grade="D"/>
    </results>
  </student>
  <student id="100078">
    <name>Jack Doe</name>
    <age>18</age>
    <major>Physics</major>
    <major>XML Science</major>
    <results>
      <result course="Math 101" grade="A"/>
      <result course="XML 101" grade="A-"/>
      <result course="Physics 101" grade="B+"/>
      <result course="XML 102" grade="A"/>
    </results>
  </student>
</students>
NoSQL Databases

• A collection of restricted/specialized database models to allow for scalability / distribution
  – Key-value store: specialized for hash tables
  – Document store: similar to key-value, but values have an internal structure (e.g., XML, JSON)
  – Graph databases: specialized for graphs/triples with “nodes” and “edges;” queries tailored to traversal

• Relaxed guarantees on concurrent transactions for better performance
RDF: Triples of the Semantic Web

http://dbpedia.org/page/Technion_-_Israel_Institute_of_Technology

```xml
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:dbp="http://dbpedia.org/property/"
    xmlns:dbo="http://dbpedia.org/ontology/>
<rdf:Description rdf:about="http://dbpedia.org/resource/Technion_Israel_Institute_of_Technology">
    <dbp:students rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">13253</dbp:students>
    <dbp:president rdf:resource="http://dbpedia.org/resource/Peretz_Lavie" />
</rdf:Description>
</rdf:RDF>
```
RDF Example from DBPedia

http://dbpedia.org/resource/Technion_Israel_Institute_of_Technology

http://dbpedia.org/resource/Peretz_Lavie

http://dbpedia.org/ontology/birthYear

http://dbpedia.org/ontology/birthPlace

http://dbpedia.org/property/president

13253

1949

http://dbpedia.org/resource/Petah_Tikva

http://dbpedia.org/resource/Israel

http://dbpedia.org/ontology/country

http://dbpedia.org/property/students
There’s More! Linked!

http://lod-cloud.net/
Main Course Topics

1. Relational database design
   - ERD
   - Integrity & design theory
2. Relational query languages
   - SQL, Algebra, Datalog, Relational Calculus
3. Additional models
   - XML
   - NoSQL
   - RDF (Semantic Web)
FYI: Complementary Courses

- **236510 Database Systems Implementation**
  - Concurrency control, recovery, query processing, distributed DBs & replication, in-depth acquaintance with a commercial system

- **236378 Principles of Managing Uncertain Data**
  - Database foundations (theory), nulls and missing information, inconsistent databases, probabilistic databases

- **234322 Information Storage Systems**
  - Used to be “File Systems”
  - Files & disks, secondary-memory computation, DB index, query-plan optimization (single node, MR), concurrency control, recovery

- **236323 Data Processing Laboratory**

- **236803 Grad Seminar on Data & Knowledge**
Historical Overview
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 database models
  – Integrated Data Stores (Charles Bachman)
  – IBM’s IMS, driven by the Apollo program
    • Hierarchical data model; focused mainly on storage interface; low-level access to retrieve record segments
Codd’s Vision (1)

• 1970: Codd invents the *relational database* model
  – Idea: Use logic!
    • Data stored as a collection of *relations*, connected by 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
  – Proves their equal expressive power

\[(r_1[2], r_1[3]): P_1 r_1 \land \forall r_2 \exists r_3 \left((r_1[1] = r_3[1]) \land (r_3[3] = r_2[1])\right).\]

Applying the reduction procedure of Section 4.1, we obtain the following defining equations:

\[S_i = R_i \quad (i=1,2,3)\]

\[S = S_1 \Join S_2 \Join S_3\]

\[T_3 = S[1=6] \cap S[8=4]\]
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
• 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
Publication Venues for DB Research

• Conferences:
  – **SIGMOD**: ACM Special Interest Group on Management of Data (since 1975)
  – **PODS**: ACM Symp. on Principles of Database Systems (since 1982)
  – **VLDB**: Intl. Conf. on Very Large Databases (since 1975)
  – **ICDE**: IEEE Intl. Conf. on Data Engineering (since 1984)
  – **ICDT**: Intl. Conference on Database Theory (since 1986)
  – **EDBT**: Intl. Conference on Extending Database Technology (since 1988)

• Journals:
  – **TODS**: ACM Transactions on Database Systems (since 1976)
  – **VLDBJ**: The VLDB Journal (since 1992)
  – **SIGMOD REC**: 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>&lt;br&gt;• Distributed, storage, in-memory, recovery</td>
<td><strong>Database Security</strong>&lt;br&gt;<strong>Views</strong>&lt;br&gt;• View-based access&lt;br&gt;• Incremental maintain</td>
<td><strong>Further XML</strong>&lt;br&gt;• Query eval / optimize&lt;br&gt;• Compression</td>
</tr>
<tr>
<td><strong>Query Languages</strong>&lt;br&gt;• Codasyl, SQL, recursion, nesting</td>
<td><strong>System Optimization</strong>&lt;br&gt;• Caching &amp; replication&lt;br&gt;• Indexing&lt;br&gt;• Clustering</td>
<td><strong>Database Privacy</strong>&lt;br&gt;<strong>Data Models</strong>&lt;br&gt;• Streaming data&lt;br&gt;• Graph data</td>
</tr>
<tr>
<td><strong>Schema Design</strong>&lt;br&gt;• ER models, normal forms, dependency</td>
<td><strong>Benchmarking</strong>&lt;br&gt;<strong>Heterogeneity</strong>&lt;br&gt;• Data Integration&lt;br&gt;• Interoperability</td>
<td><strong>DB Uncertainty</strong>&lt;br&gt;• Inconsistency &amp; cleaning&lt;br&gt;• Probabilistic DB</td>
</tr>
<tr>
<td><strong>Transaction &amp; concur.</strong></td>
<td><strong>Analytics (OLAP)</strong>&lt;br&gt;<strong>Data Models</strong>&lt;br&gt;• Multimedia, DNA&lt;br&gt;• Text, XML</td>
<td><strong>DB &amp; IR</strong>&lt;br&gt;• DB for search&lt;br&gt;• Search for DB</td>
</tr>
<tr>
<td><strong>DB Performance</strong>&lt;br&gt;• Query process &amp; opt.&lt;br&gt;• Evaluation methods</td>
<td><strong>Mining &amp; Discovery</strong>&lt;br&gt;• Discovering association rules</td>
<td><strong>Schema Matching &amp; Discovery</strong>&lt;br&gt;<strong>Data Exchange</strong>&lt;br&gt;<strong>Ranking &amp; personalization</strong></td>
</tr>
<tr>
<td><strong>Data Models</strong>&lt;br&gt;• OO, geo, temporal</td>
<td><strong>Logic</strong>&lt;br&gt;• Deductive (Datalog)&lt;br&gt;• Integrity/constraints</td>
<td><strong>Provenance/lineage</strong>&lt;br&gt;<strong>Data Models</strong>&lt;br&gt;• Semantic Web (RDF, ontologies)&lt;br&gt;• NoSQL (doc, graph, key-value)</td>
</tr>
<tr>
<td><strong>Incompleteness (null)</strong></td>
<td><strong>Entity Resolution</strong>&lt;br&gt;<strong>Information Extraction from Web/text</strong>&lt;br&gt;<strong>Crowdsourcing</strong>&lt;br&gt;• Utilizing crowd input in databases</td>
<td><strong>Social Networks &amp; Social Media</strong>&lt;br&gt;<strong>Data Models</strong>&lt;br&gt;<strong>DB &amp; ML &amp; AI</strong>&lt;br&gt;<strong>Provenance/lineage</strong>&lt;br&gt;<strong>Cloud Databases</strong>&lt;br&gt;<strong>Column Stores</strong></td>
</tr>
</tbody>
</table>

* Based on SIGMOD session topics from DBLP
Turing Awards for DB Technology

1973

**CHARLES WILLIAM BACHMAN**
United States – 1973

**CITATION**
For his outstanding contributions to database technology.

1981

**EDGAR F. ("TED") CODD**
United States – 1981

**CITATION**
For his fundamental and continuing contributions to the theory and practice of database management systems.

1998

**JAMES ("JIM") NICHOLAS GRAY**
United States – 1998

**CITATION**
For seminal contributions to database and transaction processing research and technical leadership in system implementation.

2014

**MICHAEL STONEBRAKER**
United States – 2014

**CITATION**
For fundamental contributions to the concepts and practices underlying modern database systems.