Lecture 1:
Introduction
THE COURSE
Main Topics

1. Database modeling & design

2. Relational Databases
   - Querying: SQL, Algebra, Logic
   - Integrity & design theory

3. Additional models
   - XML
   - NoSQL
   - RDF
Course Staff

• Lecturer
  – Oded Shmueli, Taub 716
    • oshmu@technion.ac.il
    • Reception on Wednesday, 11:00-12:00

• TAs
  – Liat Peterfreund  Taub 314 (in charge)
    • liatpf@cs.technion.ac.il
  – Dvir Dukhan
    • dvir.dukhan@campus.technion.ac.il
  – Idan Hasson
    • idanha@cs.technion.ac.il
  – Inbar Kaslasi
    • inbark@campus.technion.ac.il
• **Prerequisites**: Logic and Set Theory 234293

• **Grading**:
  – 80% - Final exam
  – 20% - Assignments: 2 dry, 2 wet+dry (TAKEF)

• **Assignments are for pairs**

• **Assignment grades will be taken into consideration only for students who get at least 55 in the final exam**

• **Exam Dates**:
  – Moed A - 4/7/18
  – Moed B - 12/10/18
FYI: Complementary Courses

• 234322 Information Storage Systems
  – Used to be “File Systems”
  – Relevant content: files and disks, secondary-memory computation, database indexes, query-plan optimization (single node, MR), concurrency control, database recovery

• 236510 Database Systems Implementation
  – Concurrency Control
  – Recovery
  – Query Processing
  – Distributed Databases and Replication
  – In-depth acquaintance with a commercial system

• 236378 Principles of managing uncertain data
  - Nulls and missing information, inconsistent databases, probabilistic databases
DATABASES
The real world:

- Objects - unique
- Relationships between objects – at different complexity levels
- Operations: Changing aspects of objects and relationships

Example: Bank

- Objects- customers, branches, employees
- Relationships- “customer-owns-account”, “account-managed_at- branch
- Operations: add a customer, add an account, update an account, deposit, withdrawal, balance, prediction

A DBMS is charged with faithful and efficient modeling of aspects of the real world for a specific set of operations
So, What is a Database?

• Simply, a persistent (cross session) repository of data

• Data Base Management System (DBMS): A 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:
  – Modeling (data, languages, consistency, security)
  – Engineering
  – Efficiency & scalability
What Services do Databases Provide?

- Centralized management of information at the conceptual and physical levels
- A “smart” query processor
- Transaction Processing, ACID
- Centralized access control
- Centralized level of operational recovery
- A language for data access
- Accessing data from various platforms
- A high degree of logical-physical independence
Frozen (I) (2013)

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

Your rating: ★★★★★★★★ 7.6/10
Ratings: 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)
An Entity-Relationship Diagram (ERD)
Type Inheritance?
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>

Advantages?

Drawbacks:

- **Cost & Redundancy**: *Why should the student’s address be stored in each course she takes?*
- **Incompleteness**: *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 maintain**: If we wish to add 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>sName</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 (data-independent) specifications 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.)

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

• Maintenance: consistency assured w/o custom code
• Development complexity: no reliance on consistency tests
  – But exceptions need to be handled
• Optimization: operations may be optimized if we know that some constraints hold
  – (e.g., once a sought student ID is found, you can stop; you won’t find it again)
### Which Constraints Should Hold Here?

<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>

<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>

<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>2345</td>
<td>319</td>
<td>73</td>
</tr>
</tbody>
</table>

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

### S
<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>

### C
<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>

### T
<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>

---

**Assembly**

... 

```assembly
mov 1, %ax
mov 1, %di
mov $message, %si
mov $13, %dx
syscall
mov $60, %ax
xor %di, %di
```

---

**Python**

```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**

```sql
QL
{⟨x⟩| ∃y, n, z, l, g
    [ S(y, n, 'Avia') ∧ C(z, x, l) ∧ T(y, z, g) ]}
```

---

**SQL**

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

---

**Logic Programming (Datalog)**

```prolog
Q(x) ← S(y, n, 'Avia'), C(z, x, l), T(y, z, g)
```

---

**Logic (RC)**

```prolog
{(x)| ∃y, n, z, l, g
    [ S(y, n, 'Avia') ∧ C(z, x, l) ∧ T(y, z, g) ]}
```

---

**Algebra (RA)**

```prolog
πC.name(σ_{S.name='Avia', number=cNum} ID=sID(S×C×T))
```
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
Other Data Models: XML

```xml
<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

• Really, “no general relations”
• 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 with “nodes” and “edges;” queries tailored to traversal
End of Introduction

• Additional material on the history of databases follows
• Also, read https://en.wikipedia.org/wiki/Database
• Next: Entity Relationship Diagrams

בצלחת בקורס!
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 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:
    • 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 in 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

\[(r_1[2], r_1[3]): P_1 \land P_2 \land P_3 \land (r_1[1] = r_3[1]) \land (r_3[3] = r_2[1])\]

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

\[S_i = R_i \quad (i=1,2,3)\]
\[S = S_1 \land S_2 \land S_3\]
\[T_3 = S[1=6] \land S[8=4]\]
• 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
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)
<table>
<thead>
<tr>
<th>Selected Database Research Topics*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System Design</strong></td>
</tr>
<tr>
<td>- Distributed, storage, in-memory, recovery</td>
</tr>
<tr>
<td><strong>Query Languages</strong></td>
</tr>
<tr>
<td>- Codasyl, SQL, recursion, nesting</td>
</tr>
<tr>
<td><strong>Schema Design</strong></td>
</tr>
<tr>
<td>- ER models, normal forms, dependency</td>
</tr>
<tr>
<td><strong>Transaction &amp; concur.</strong></td>
</tr>
<tr>
<td><strong>DB Performance</strong></td>
</tr>
<tr>
<td>- Query process &amp; opt.</td>
</tr>
<tr>
<td>- Evaluation methods</td>
</tr>
<tr>
<td><strong>Data Models</strong></td>
</tr>
<tr>
<td>- OO, geo, temporal</td>
</tr>
<tr>
<td><strong>Logic</strong></td>
</tr>
<tr>
<td>- Deductive (Datalog)</td>
</tr>
<tr>
<td>- Integrity/constraints</td>
</tr>
<tr>
<td><strong>Incompleteness (null)</strong></td>
</tr>
<tr>
<td><strong>Database Security</strong></td>
</tr>
<tr>
<td>- View-based access</td>
</tr>
<tr>
<td>- Incremental maintain</td>
</tr>
<tr>
<td><strong>System Optimization</strong></td>
</tr>
<tr>
<td>- Caching &amp; replication</td>
</tr>
<tr>
<td>- Indexing</td>
</tr>
<tr>
<td>- Clustering</td>
</tr>
<tr>
<td><strong>Database Privacy</strong></td>
</tr>
<tr>
<td><strong>Heterogeneity</strong></td>
</tr>
<tr>
<td>- Data Integration</td>
</tr>
<tr>
<td>- Interoperability</td>
</tr>
<tr>
<td><strong>Analytics (OLAP)</strong></td>
</tr>
<tr>
<td><strong>Data Models</strong></td>
</tr>
<tr>
<td>- Multimedia, DNA</td>
</tr>
<tr>
<td>- Text, XML</td>
</tr>
<tr>
<td><strong>Mining &amp; Discovery</strong></td>
</tr>
<tr>
<td>- Discovering association rules</td>
</tr>
<tr>
<td><strong>Incompleteness (null)</strong></td>
</tr>
<tr>
<td><strong>Further XML</strong></td>
</tr>
<tr>
<td>- Query eval / optimize</td>
</tr>
<tr>
<td>- Compression</td>
</tr>
<tr>
<td><strong>Database Privacy</strong></td>
</tr>
<tr>
<td><strong>DB Uncertainty</strong></td>
</tr>
<tr>
<td>- Inconsistency &amp; cleaning</td>
</tr>
<tr>
<td>- Probabilistic DB</td>
</tr>
<tr>
<td><strong>DB &amp; IR</strong></td>
</tr>
<tr>
<td>- DB for search</td>
</tr>
<tr>
<td>- Search for DB</td>
</tr>
<tr>
<td><strong>Data Models</strong></td>
</tr>
<tr>
<td><strong>Entity Resolution</strong></td>
</tr>
<tr>
<td><strong>Information Extraction from Web/text</strong></td>
</tr>
<tr>
<td><strong>Crowdsourcing</strong></td>
</tr>
<tr>
<td>- Utilizing crowd input in databases</td>
</tr>
<tr>
<td><strong>Social Networks &amp; Social Media</strong></td>
</tr>
<tr>
<td><strong>Data Models</strong></td>
</tr>
<tr>
<td>- Semantic Web (RDF, ontologies)</td>
</tr>
<tr>
<td>- NoSQL (doc, graph, key-value)</td>
</tr>
<tr>
<td><strong>DB &amp; ML &amp; AI</strong></td>
</tr>
<tr>
<td><strong>Provenance/ lineage</strong></td>
</tr>
<tr>
<td>- Model / compute</td>
</tr>
<tr>
<td><strong>Cloud Databases</strong></td>
</tr>
<tr>
<td><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.