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
    - dvirdukhan@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)

(Semi-)Flipped Classroom

- Course material online, home reading!
  - Detailed presentations
- Questions & feedback online
  - Example: 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

Course Requirements

- Final exam (80%)
  - “Open book”
- Home assignments (20%)
  - 2x dry, 1x wet, 1x wet+dry
  - More details in tutorial this week

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

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)

More Detailed?

Type Inheritance?

Relational Design

Option 1: Single Table

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Address</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?

- 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

Option 2: Multiple Tables

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>address</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>
**Relational Design**

### Option 1: Single Table

<table>
<thead>
<tr>
<th>StudentCourseRegistry</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
<tr>
<td>1234</td>
</tr>
<tr>
<td>1234</td>
</tr>
<tr>
<td>2345</td>
</tr>
</tbody>
</table>

### Option 2: Multiple Tables

<table>
<thead>
<tr>
<th>Student</th>
<th>Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234 Avia Haifa</td>
<td>363</td>
</tr>
<tr>
<td>1234 Avia Haifa</td>
<td>319</td>
</tr>
<tr>
<td>2345 Boris Nesher</td>
<td>319</td>
</tr>
</tbody>
</table>

**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?**

**A student cannot get two grades for the same course** (check constraint)

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234 Avia Haifa</td>
<td>363</td>
<td>DB</td>
</tr>
<tr>
<td>2345 Boris Nesher</td>
<td>319</td>
<td>PL</td>
</tr>
</tbody>
</table>

**No two tuples have the same ID (key constraint)**

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234 Avia Haifa</td>
<td>363</td>
<td>DB</td>
</tr>
<tr>
<td>2345 Boris Nesher</td>
<td>319</td>
<td>PL</td>
</tr>
</tbody>
</table>

**Courses with the same number have the same name (functional dependency)**

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234 Avia Haifa</td>
<td>363</td>
<td>DB</td>
</tr>
<tr>
<td>2345 Boris Nesher</td>
<td>319</td>
<td>PL</td>
</tr>
</tbody>
</table>

**sID is a Student.ID, cNum is a Course.ID (referential constraint)**

**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

**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 database: specialized for graphs/triples with “nodes” and “edges,” queries tailored to traversal
- Relaxed guarantees on concurrent transactions for better performance

**Other Data Models: XML**

- Document store
- Key-value store
- Graph databases
- Value store

**RDF: Triples of the Semantic Web**

- RDF Example from DBPedia
- There’s More! Linked!

**Technion Technion**

http://dbpedia.org/page/Technion__Israel_Institute_of_Technology

- RDF Example from DBPedia
- There’s More! Linked!

- RDF Example from DBPedia
- There’s More! Linked!
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
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

![M. Stonebraker](image1.png)
![E. Wong](image2.png)

PostgreSQL

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

![E. Wong](image3.png)
![M. Stonebraker](image4.png)

![P. G. Selinger](image5.png)

- 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