Database Management Systems
Course 236363
Lecture 1:
Introduction

Announcement: Complement Lecture
- Sunday next week (April 2nd)
- 16:30 - 18:30
- Room: Taub 4
- Prof. Oded Shmueli
- (Tuesday lecture 4/4 as usual)

Modeling Reality
- The real world:
  - Objects
  - Relationships (between objects)
  - Operations (changing aspects of objects and relationships)
- Example: Bank
  - Objects: customers, branches, employees
  - Relationships: customer-account, account-branch
  - Operations: add customer, add account, deposit, withdrawal, balance, prediction
DBMS: faithful & efficient modeling of aspects of the real world for a specific operation

So, What is a Database?
- Simply, a persistent (cross session) repository of data
- Models differ in what “data” means, especially how to separate content from structure: Tables? Graphs? Objects? Maps?
- Database 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, query, consistency, security)
  - Engineering
  - Efficiency & scalability

What Services do Databases Provide?
- Centralized information management
  - Conceptual and 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)
  - Program in high levels of abstraction (concepts, entities, relationships, etc.)
  - No need for in-house implementation
  - Storage, disk, persistency, recovery, security, algo, 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)

Data Modeling

More Detailed?

Type Inheritance?
Relational Design

Option 1: Single Table

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>addr</th>
<th>cNum</th>
<th>cName</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Avia</td>
<td>Haifa</td>
<td>363</td>
<td>DB</td>
<td>Avia</td>
</tr>
<tr>
<td>1234</td>
<td>Avia</td>
<td>Haifa</td>
<td>319</td>
<td>PL</td>
<td>Barak</td>
</tr>
<tr>
<td>2345</td>
<td>Boris</td>
<td>Nesher</td>
<td>319</td>
<td>PL</td>
<td>Barak</td>
</tr>
</tbody>
</table>

Advantages?

- **Cost & Redundancy**: Why should the student's address be stored in each course she takes?
- **Incomplete**: What about students that do not take any courses? Courses with 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 the a semester column, every app will need to update its schema assumption

Drawbacks:

- **Incompleteness**: Can a course be taken by two students with the same ID have the same name, take the same courses, etc.)
- **Cost & Redundancy**: Beyond the relational structure (e.g., students with the same ID have the same name, take the same course)
- **Incompleteness**: With the same ID have the same name, take the same courses, etc.)
- **Cost & Redundancy**: Schema-level: talks about object and relationship types; not concrete instances
- **DBMS guarantees that constraints are always satisfied**
- **By disabling actions that cause violations**

Integrity Constraints

- **Schema-level 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.)
  - 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?

- **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?

- 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)

Relational Design

Option 1: Single Table

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>addr</th>
<th>cNum</th>
<th>cName</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Avia</td>
<td>Haifa</td>
<td>363</td>
<td>DB</td>
<td>Avia</td>
</tr>
<tr>
<td>1234</td>
<td>Avia</td>
<td>Haifa</td>
<td>319</td>
<td>PL</td>
<td>Barak</td>
</tr>
<tr>
<td>2345</td>
<td>Boris</td>
<td>Nesher</td>
<td>319</td>
<td>PL</td>
<td>Barak</td>
</tr>
</tbody>
</table>

Option 2: Multiple Tables

<table>
<thead>
<tr>
<th>student</th>
<th>course</th>
<th>took</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234 Avia Haifa</td>
<td>363 DB Avia</td>
<td>95</td>
</tr>
<tr>
<td>1234 Avia Haifa</td>
<td>319 PL Barak</td>
<td>82</td>
</tr>
<tr>
<td>2345 Boris Nesher</td>
<td>319 PL Barak</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 cName, etc.

Querying: Which Courses Avia Took?

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>addr</th>
<th>cNum</th>
<th>cName</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Avia</td>
<td>Haifa</td>
<td>363</td>
<td>DB</td>
<td>Avia</td>
</tr>
<tr>
<td>1234</td>
<td>Avia</td>
<td>Haifa</td>
<td>319</td>
<td>PL</td>
<td>Barak</td>
</tr>
<tr>
<td>2345</td>
<td>Boris</td>
<td>Nesher</td>
<td>319</td>
<td>PL</td>
<td>Barak</td>
</tr>
</tbody>
</table>

Assembly

```
... mov $1, yra
mov $1, first
movメッセージ, first
mov $1, id
mov 0, x,
sbr $1, x,
... 
```

Python

For x in T:
  if c in C:
    if f.x.cName == "Avia" and s.ID = t.cID and t.cNum = x.cNum:
      print x.cName

SQL

```
SELECT C.name, WHERE C.cName = "Avia" AND S.ID = T.cID
AND T.cNum = C.cNum
```

Logic Programming (Prolog)

```
Q(X) -> (C=y,x,"Avia"), C(cName, x), T(cName, X)
```

Logic (PL)

```
(\{C=y,x,"Avia"\}) ∧ (C=cNum, X)\}"
```
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 id="100007">Jack Doe</name>
    <age id="100008">21</age>
    <major id="100026">Physics</major>
    <result course="Math 101" grade="A"/>
    <result course="Statistics 101" grade="D"/>
    <result course="Biology 101" grade="C+"/>
  </student>
  <student id="100027">
    <name id="100007">Joe Doe</name>
    <age id="100008">18</age>
    <major id="100026">Technology</major>
    <result course="XML 101" grade="A"/>
    <result course="XML 102" grade="A"/>
  </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/triples with “nodes” and “edges,” queries tailored to traversal

**RDF: Triples of the Semantic Web**

```
RDF Example from DBPedia
```

**RDF Example from DBPedia**

```
http://dbpedia.org/resource/Technion_Israel_Institute_of_Technology
http://dbpedia.org/resource/Technion_Israel_Institute_of_Technology
http://dbpedia.org/resource/Petrae_Lavie
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```

```
RDF Example from DBPedia
```
Main Course Topics

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

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

http://lod-cloud.net/
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
- **1973**: Evolved to Postgres (now PostgreSQL) in 1989

Codd Catches On (1)

- **1973**: Michael Stonebraker and Eugene Wong implement Codd’s vision in **INGRES**
  - Commercialized in 1983
- **1977**: Influenced by Codd, Larry Ellison founds **Software Development Labs**
  - Evolved to **Oracle** in 1979
  - Becomes Oracle Systems Corp (1982), named after its Oracle database product

Publications 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 1989)
- Journals:
  - **TODS**: ACM Transactions on Database Systems (since 1976)
  - **VLDBJ**: The VLDB Journal (since 1992)
  - **SIGMOD Record**: ACM SIGMOD Record (since 1969)

Selected Database Research Topics*

<table>
<thead>
<tr>
<th>Year</th>
<th>1980</th>
<th>1990</th>
<th>2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Security</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query Handling</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel SQL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Integration</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Models</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OLAP, XML</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distributed, storage, memory, recovery</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coding, SQL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relational algebra, logic</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Based on SIGMOD session topics from DBLP