Course Intro & Relational Model
TODAY’S AGENDA

Wait List
Overview
Course Logistics
Relational Model
Relational Algebra
WAIT LIST

There are 227 people on the waiting list.
There are 226 people enrolled in the course.
The max capacity is 200.

If you are not currently enrolled in this class, the likelihood that you will get in is very low.
LECTURE RULES

Please interrupt me any time during the lecture:
→ I am speaking too fast.
→ You don't understand what I am talking about.
→ You have a database-related question.

If you are unable to speak, post your question to Zoom chat and the attending TA will ask it.
This course is on the design and implementation of disk-oriented database management systems.

This is **not** a course on how to use a database to build applications or how to administer a database. → See [CMU 95-703](#) (Heinz College)

Database Applications (15-415/615) is **not** offered this semester.
COURSE OUTLINE

Relational Databases
Storage
Execution
Concurrency Control
Recovery
Distributed Databases
Potpourri
COURSE LOGISTICS

Course Policies + Schedule:
→ Refer to course web page.

Academic Honesty:
→ Refer to CMU policy page.
→ If you're not sure, ask the professors.
→ Don't be stupid.

All discussion + announcements will be on Piazza.
Database System Concepts
7th Edition
Silberschatz, Korth, & Sudarshan

We will also provide lecture notes that covers topics not found in textbook.
COURSE RUBRIC

Homeworks (15%)
Projects (45%)
Midterm Exam (20%)
Final Exam (20%)
HOMEWORKS

Five homework assignments throughout the semester.

First homework is a SQL assignment. The rest will be pencil-and-paper assignments.

All homework should be done individually.
PROJECTS

You will build your own database engine from scratch of the course of the semester. Each project builds on the previous one.

We will not teach you how to write/debug C++17

You must complete Project #0 before Sept 13th.
All projects will use the CMU's **BusTub** academic DBMS.

Architecture:

→ Disk-Oriented Storage
→ Volcano-style Query Processing
→ Pluggable APIs
→ Currently does not support SQL.
LATE POLICY

You will lose 10% of a project or homework's points for every 24 hours it is late.

We will grant no-penalty extensions due to extreme circumstances (e.g., medical emergencies).
→ If something comes up, please contact the instructor as soon as you can.
PLAGIARISM WARNING

The homework and projects must be your own work. They are not group assignments.

You may not copy source code from other people or the web.

Plagiarism will not be tolerated. See CMU's Policy on Academic Integrity for additional information.
OFFICE HOURS

All instructor + TA office hours are over Zoom.

Andy's Office Hours:
→ Monday @ 11am ET
→ Monday/Wednesday @ 10pm ET (By Appointment Only)
Quarantine Database Tech Talks
→ Mondays @ 4:30pm
→ [https://db.cs.cmu.edu/seminar2020](https://db.cs.cmu.edu/seminar2020)

Ask Andy Anything
→ Wednesdays (Immediately After Class)
→ Q&A sessions will not be recorded or made public.
→ [https://cmudb.io/f20-askandy](https://cmudb.io/f20-askandy) (CMU students only)
Carnegie Mellon University
Quarantine 2020
Database Talks

https://db.cs.cmu.edu/seminar2020
DATABASE

Organized collection of inter-related data that models some aspect of the real-world.

Databases are core the component of most computer applications.
DATABASE EXAMPLE

Create a database that models a digital music store to keep track of artists and albums.

Things we need store:
→ Information about Artists
→ What Albums those Artists released
FLAT FILE STRAWMAN

Store our database as comma-separated value (CSV) files that we manage ourselves in our application code.
→ Use a separate file per entity.
→ The application must parse the files each time they want to read/update records.
FLAT FILE STRAWMAN

Create a database that models a digital music store.

**Artist** (name, year, country)

- "Wu-Tang Clan", 1992, "USA"
- "Notorious BIG", 1992, "USA"
- "Ice Cube", 1989, "USA"

**Album** (name, artist, year)

- "Enter the Wu-Tang", "Wu-Tang Clan", 1993
- "St.Ides Mix Tape", "Wu-Tang Clan", 1994
- "AmeriKKKa's Most Wanted", "Ice Cube", 1990
Example: Get the year that Ice Cube went solo.

```
Artist((name, year, country))

"Wu-Tang Clan",1992,"USA"
"Notorious BIG",1992,"USA"
"Ice Cube",1989,"USA"
```

```python
for line in file.readlines():
    record = parse(line)
    if "Ice Cube" == record[0]:
        print(int(record[1]))
```
How do we ensure that the artist is the same for each album entry?

What if somebody overwrites the album year with an invalid string?

What if there are multiple artists on an album?

What happens if we delete an artist that has albums?
How do you find a particular record?

What if we now want to create a new application that uses the same database?

What if two threads try to write to the same file at the same time?
FLAT FILES: DURABILITY

What if the machine crashes while our program is updating a record?

What if we want to replicate the database on multiple machines for high availability?
DATABASE MANAGEMENT SYSTEM

A **DBMS** is software that allows applications to store and analyze information in a database.

A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases.
Database applications were difficult to build and maintain.

Tight coupling between logical and physical layers.

You have to (roughly) know what queries your app would execute before you deployed the database.
EARLY DBMSs

Database applications were difficult to build and maintain. Tight coupling between logical and physical layers. You had to (roughly) know what queries your app would execute before you deployed the database.

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

The relational view (of models) is described in Section 1. It appears to be superior in several respects to the graph or network model [2,4] presently in vogue for non-relational systems. It provides a means of describing data with its natural structure only—that is, without superposing any additional structures for machine representation purposes. Additionally, it provides a basis for a high-level data, language which will yield maximum independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a natural basis for treating redundancy, and consistency of relations—topics discussed in Section 2. The network model, on the other hand, has gained a number of definitions, not the least of which are the derivation of the rule for the extinction of relations (one commodity in the "conception theory"). Finally, the relational view presents a clearer realization of the scope of logical and physical limitations of present systems (network systems, and the more remote future from a logical standpoint) of implementing representations of data within a single cycle. Examples of these clearer perspectives are cited in various parts of this paper. Implementation of systems to support the relational model is discussed.

The relational view (of models) is described in Section 1. It appears to be superior in several respects to the graph or network model [2,4] presently in vogue for non-relational systems. It provides a means of describing data with its natural structure only—that is, without superposing any additional structures for machine representation purposes. Additionally, it provides a basis for a high-level data, language which will yield maximum independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a natural basis for treating redundancy, and consistency of relations—topics discussed in Section 2. The network model, on the other hand, has gained a number of definitions, not the least of which are the derivation of the rule for the extinction of relations (one commodity in the "conception theory"). Finally, the relational view presents a clearer realization of the scope of logical and physical limitations of present systems (network systems, and the more remote future from a logical standpoint) of implementing representations of data within a single cycle. Examples of these clearer perspectives are cited in various parts of this paper. Implementation of systems to support the relational model is discussed.

1. Relational Model and Normal Form

1.1. INTRODUCTION

This paper is concerned with the application of an extra-relational theory to systems which provide shared storage in large banks of quaeratedium data. Through a paper by Codd [5], the principal application of relations to data banks has been to define the question-answering systems. Section 1 and 2 of this paper provide necessary references to work in this area.

In contrast, the problems treated here are those of data independence—the independence of application programs and terminal activities from growth in data types and changes in data representation—and certain kinds of data autonomy which are expected to become troublesome even in traditional systems.

Volume 17 / Number 6 / June, 1970
A data model is a collection of concepts for describing the data in a database.

A schema is a description of a particular collection of data, using a given data model.
DATA MODEL

Relational
Key/Value
Graph
Document
Column-family
Array / Matrix
Hierarchical
Network
Multi-Value
RELATIONAL MODEL

**Structure:** The definition of the database's relations and their contents.

**Integrity:** Ensure the database's contents satisfy constraints.

**Manipulation:** Programming interface on how to access and modify a database's contents.
A relation is unordered set that contain the relationship of attributes that represent entities.

A tuple is a set of attribute values (also known as its domain) in the relation.
→ Values are (normally) atomic/scalar.
→ The special value NULL is a member of every domain.

<table>
<thead>
<tr>
<th>name</th>
<th>year</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wu-Tang Clan</td>
<td>1992</td>
<td>USA</td>
</tr>
<tr>
<td>Notorious BIG</td>
<td>1992</td>
<td>USA</td>
</tr>
<tr>
<td>Ice Cube</td>
<td>1989</td>
<td>USA</td>
</tr>
</tbody>
</table>

Artist(name, year, country)

n-ary Relation = Table with n columns
RELATIONAL MODEL: PRIMARY KEYS

A relation's primary key uniquely identifies a single tuple.
Some DBMSs automatically create an internal primary key if a table does not define one.

Auto-generation of unique integer primary keys:
→ **SEQUENCE** (SQL:2003)
→ **AUTO_INCREMENT** (MySQL)

**Artist(name, year, country)**

<table>
<thead>
<tr>
<th>name</th>
<th>year</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wu-Tang Clan</td>
<td>1992</td>
<td>USA</td>
</tr>
<tr>
<td>Notorious BIG</td>
<td>1992</td>
<td>USA</td>
</tr>
<tr>
<td>Ice Cube</td>
<td>1989</td>
<td>USA</td>
</tr>
</tbody>
</table>
A relation's primary key uniquely identifies a single tuple.
Some DBMSs automatically create an internal primary key if a table does not define one.

Auto-generation of unique integer primary keys:
→ **SEQUENCE** (SQL:2003)
→ **AUTO_INCREMENT** (MySQL)
RELATIONAL MODEL: FOREIGN KEYS

A foreign key specifies that an attribute from one relation has to map to a tuple in another relation.
### RELATIONAL MODEL: FOREIGN KEYS

**Artist**\((id, \text{name, year, country})\)

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>year</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Wu-Tang Clan</td>
<td>1992</td>
<td>USA</td>
</tr>
<tr>
<td>456</td>
<td>Notorious BIG</td>
<td>1992</td>
<td>USA</td>
</tr>
<tr>
<td>789</td>
<td>Ice Cube</td>
<td>1989</td>
<td>USA</td>
</tr>
</tbody>
</table>

**Album**\((id, \text{name, artists, year})\)

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>artists</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Enter the Wu-Tang</td>
<td>123</td>
<td>1993</td>
</tr>
<tr>
<td>22</td>
<td>St.Ides Mix Tape</td>
<td>???</td>
<td>1994</td>
</tr>
<tr>
<td>33</td>
<td>AmeriKKKa's Most Wanted</td>
<td>789</td>
<td>1990</td>
</tr>
</tbody>
</table>
**RELATIONAL MODEL: FOREIGN KEYS**

### Artist (id, name, year, country)

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>year</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Wu-Tang Clan</td>
<td>1992</td>
<td>USA</td>
</tr>
<tr>
<td>456</td>
<td>Notorious BIG</td>
<td>1992</td>
<td>USA</td>
</tr>
<tr>
<td>789</td>
<td>Ice Cube</td>
<td>1989</td>
<td>USA</td>
</tr>
</tbody>
</table>

### Album (id, name, artists, year)

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>artists</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Enter the Wu-Tang</td>
<td>123</td>
<td>1993</td>
</tr>
<tr>
<td>22</td>
<td>St.Ides Mix Tape</td>
<td>???</td>
<td>1994</td>
</tr>
<tr>
<td>33</td>
<td>AmeriKKKa's Most Wanted</td>
<td>789</td>
<td>1990</td>
</tr>
</tbody>
</table>

### ArtistAlbum (artist_id, album_id)

<table>
<thead>
<tr>
<th>artist_id</th>
<th>album_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>11</td>
</tr>
<tr>
<td>123</td>
<td>22</td>
</tr>
<tr>
<td>789</td>
<td>22</td>
</tr>
<tr>
<td>456</td>
<td>22</td>
</tr>
</tbody>
</table>
RELATIONAL MODEL: FOREIGN KEYS

Artist(id, name, year, country)

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>year</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Wu-Tang Clan</td>
<td>1992</td>
<td>USA</td>
</tr>
<tr>
<td>456</td>
<td>Notorious BIG</td>
<td>1992</td>
<td>USA</td>
</tr>
<tr>
<td>789</td>
<td>Ice Cube</td>
<td>1989</td>
<td>USA</td>
</tr>
</tbody>
</table>

ArtistAlbum(artist_id, album_id)

<table>
<thead>
<tr>
<th>artist_id</th>
<th>album_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>11</td>
</tr>
<tr>
<td>456</td>
<td>22</td>
</tr>
<tr>
<td>789</td>
<td>22</td>
</tr>
</tbody>
</table>

Album(id, name, year)

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Enter the Wu-Tang</td>
<td>1993</td>
</tr>
<tr>
<td>22</td>
<td>St.Ides Mix Tape</td>
<td>1994</td>
</tr>
<tr>
<td>33</td>
<td>AmeriKKKa's Most Wanted</td>
<td>1990</td>
</tr>
</tbody>
</table>
DATA MANIPULATION LANGUAGES (DML)

Methods to store and retrieve information from a database.

**Procedural:**
→ The query specifies the (high-level) strategy the DBMS should use to find the desired result.

**Non-Procedural:**
→ The query specifies only what data is wanted and not how to find it.

← Relational Algebra
← Relational Calculus
RELATIONAL ALGEBRA

Fundamental operations to retrieve and manipulate tuples in a relation.
→ Based on set algebra.

Each operator takes one or more relations as its inputs and outputs a new relation.
→ We can "chain" operators together to create more complex operations.

σ  Select
π  Projection
∪  Union
∩  Intersection
−  Difference
×  Product
⋈  Join
RELATIONAL ALGEBRA: SELECT

Choose a subset of the tuples from a relation that satisfies a selection predicate.
→ Predicate acts as a filter to retain only tuples that fulfill its qualifying requirement.
→ Can combine multiple predicates using conjunctions / disjunctions.

Syntax: \( \sigma_{\text{predicate}}(R) \)

\[
\begin{array}{|c|c|}
\hline
\text{a}_{\text{id}} & \text{b}_{\text{id}} \\
\hline
\text{a1} & 101 \\
\text{a2} & 102 \\
\text{a2} & 103 \\
\text{a3} & 104 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{a}_{\text{id}} & \text{b}_{\text{id}} \\
\hline
\text{a2} & 102 \\
\text{a2} & 103 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{a}_{\text{id}} & \text{b}_{\text{id}} \\
\hline
\text{a2} & 103 \\
\hline
\end{array}
\]

\[
\text{SELECT \star FROM } R \text{ WHERE } a_{\text{id}}=\text{a2} \text{ AND } b_{\text{id}}>102;
\]
RELATIONAL ALGEBRA: PROJECTION

Generate a relation with tuples that contains only the specified attributes.
→ Can rearrange attributes’ ordering.
→ Can manipulate the values.

Syntax: \( \Pi_{A_1,A_2,\ldots,A_n}(R) \)

\[
\begin{array}{|c|c|}
\hline
\text{a_id} & \text{b_id} \\
\hline
\text{a1} & 101 \\
\text{a2} & 102 \\
\text{a2} & 103 \\
a3 & 104 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{b_id-100} & \text{a_id} \\
\hline
2 & \text{a2} \\
3 & \text{a2} \\
\hline
\end{array}
\]

SELECT \text{b_id-100, a_id}
FROM \text{R}
WHERE \text{a_id} = 'a2';
RELATIONAL ALGEBRA: UNION

Generate a relation that contains all tuples that appear in either only one or both input relations.

Syntax: \((R \cup S)\)
Generate a relation that contains only the tuples that appear in both of the input relations.

Syntax: \((R \cap S)\)
RELATIONAL ALGEBRA: DIFFERENCE

Generate a relation that contains only the tuples that appear in the first and not the second of the input relations.

Syntax: \((R - S)\)

```
\[
\begin{array}{|c|c|}
\hline
\text{a_id} & \text{b_id} \\
\hline
\text{a1}  & 101  \\
\text{a2}  & 102  \\
\text{a3}  & 103  \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{a_id} & \text{b_id} \\
\hline
\text{a3}  & 103  \\
\text{a4}  & 104  \\
\text{a5}  & 105  \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{a_id} & \text{b_id} \\
\hline
\text{a1}  & 101  \\
\text{a2}  & 102  \\
\hline
\end{array}
\]
```

\[(SELECT * FROM R) \ EXCEPT \ (SELECT * FROM S);\]
Generate a relation that contains all possible combinations of tuples from the input relations.

**Syntax:** $(R \times S)$

```
SELECT * FROM R CROSS JOIN S;
SELECT * FROM R, S;
```
RELATIONAL ALGEBRA: JOIN

Generate a relation that contains all tuples that are a combination of two tuples (one from each input relation) with a common value(s) for one or more attributes.

Syntax: \((R \bowtie S)\)

<table>
<thead>
<tr>
<th>R(a_id, b_id)</th>
<th>S(a_id, b_id)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_id</td>
<td>b_id</td>
</tr>
<tr>
<td>a1</td>
<td>101</td>
</tr>
<tr>
<td>a2</td>
<td>102</td>
</tr>
<tr>
<td>a3</td>
<td>103</td>
</tr>
<tr>
<td>a_id</td>
<td>b_id</td>
</tr>
<tr>
<td>a3</td>
<td>103</td>
</tr>
<tr>
<td>a4</td>
<td>104</td>
</tr>
<tr>
<td>a5</td>
<td>105</td>
</tr>
</tbody>
</table>

(R \bowtie S) =

\[
\begin{array}{|c|c|}
\hline
a_id & b_id \\
\hline
a3 & 103 \\
\hline
\end{array}
\]

SELECT * FROM R NATURAL JOIN S;
RELATIONAL ALGEBRA: EXTRA OPERATORS

Rename ($\rho$)
Assignment ($R \leftarrow S$)
Duplicate Elimination ($\delta$)
Aggregation ($\gamma$)
Sorting ($\tau$)
Division ($R \div S$)
Relational algebra still defines the high-level steps of how to compute a query.

→ $\sigma_{b \_id=102}(R \bowtie S)$ vs. $(R \bowtie (\sigma_{b \_id=102}(S)))$

A better approach is to state the high-level answer that you want the DBMS to compute.

→ Retrieve the joined tuples from $R$ and $S$ where $b \_id$ equals 102.
The relational model is independent of any query language implementation.

**SQL** is the *de facto* standard (many dialects).

```python
for line in file.readlines():
    record = parse(line)
    if "Ice Cube" == record[0]:
        print(int(record[1]))
```

```sql
SELECT year FROM artists
WHERE name = 'Ice Cube';
```
CONCLUSION

Databases are ubiquitous.

Relational algebra defines the primitives for processing queries on a relational database.

We will see relational algebra again when we talk about query optimization + execution.
NEXT CLASS

Crash Course on SQL