Course Intro & Relational Model
TODAY’S AGENDA

Course Logistics
Relational Model
Relational Algebra
I do **not** control the wait list.

Admins will move students moved off the wait list as new spots become available.

If you are not currently enrolled, the likelihood that you will get in is fairly low.
COURSE OVERVIEW

This course is about the design/implementation of database management systems (DBMSs).

This is **not** a course about how to use a DBMS to build applications or how to administer a DBMS.

→ See [CMU 95-703](#) (Heinz College)
PROJECTS

All projects will use the CMU DB Group BusTub academic DBMS.
→ Each project builds on the previous one.
→ We will not teach you how to write/debug C++17.

Total of four late days the entire semester for projects only.

You must complete Project #0 before January 29th.
COURSE LOGISTICS

Course Policies + Schedule: [Course Web Page]
Discussion + Announcements: [Piazza]
Homeworks + Projects: [Gradescope]
Final Grades: [Canvas]
PLAGIARISM WARNING

The homework and projects must be your own original work. They are not group assignments. You may not copy source code from other people or the web. Plagiarism is not tolerated. → Please ask me if you are unsure.

Differences between S23 and the Fall

1. I am not Andy Pavlo.
2. I am not Andy Pavlo.
3. I am not Andy Pavlo.
Databases
DATABASE

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

Databases are the core 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 for our 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"
- "GZA", 1990, "USA"

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

- "Enter the Wu-Tang", "Wu-Tang Clan", 1993
- "St. Ides Mix Tape", "Wu-Tang Clan", 1994
- "Liquid Swords", "GZA", 1990
FLAT FILE STRAWMAN

Example: Get the year that GZA went solo.

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

```
"Wu-Tang Clan",1992,"USA"
"Notorious BIG",1992,"USA"
"GZA",1990,"USA"
```

Artist(name, year, country)
FLAT FILES: DATA INTEGRITY

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?
FLAT FILES: IMPLEMENTATION

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?
A database management system (DBMS) is software that allows applications to store and analyze information in a database.

A general-purpose DBMS supports the definition, creation, querying, update, and administration of databases in accordance with some data model.
DATA MODELS

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 MODELS

- Relational  ← Most DBMSs
- Key/Value
- Graph
- Document / Object
- Wide-Column / Column-family
- Array / Matrix / Vectors  ← NoSQL
- Hierarchical
- Network
- Multi-Value  ← Machine Learning

← Obsolete / Legacy / Rare
Early database applications were difficult to build and maintain on available DBMSs in the 1960s.  
→ Examples: **IDS, IMS, CODASYL**  
→ Computers were expensive, humans were cheap.  

Tight coupling between logical and physical layers.  

Programmers had to (roughly) know what queries the application would execute before they could deploy the database.
TED C. GODDETTING
IBM RESEARCH LABORATORY
SUNNYVALE, CA

ABSTRACT: The large, integrated data bases of the future will
certainly store relations of various degrees in stored form. It
will not be unusual for this set of stored relations to be
inconsistent. Two types of redundancy are defined and discussed. The
type may be handled by stored consistency checks, whose
doctrine is that of asserting any "logical"
abnormality, and have some means of delaying or "logical"
checking and rechecking data. This problem cannot
be avoided by using stored consistency checks, since it
is not unusual for the data base to contain
inconsistencies in the total set of stored relations.
Consistency in stored data bases is defined and discussed in
this paper.

DEPARTMENT OF COMPUTER SERVICES
SUNNYVALE, CA

15-45/645 (Spring 2023)

EARLY DBMSs

Ted Codd was a mathematician
working at IBM Research in the late
1960s. He saw IBM's developers spending
their time rewriting database programs
every time the database's schema or
layout changed. Devised the relational model in 1969.

A Relational Model of Data for
Large Shared Data Banks

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

A new model of database systems is presented
where data is stored in relations rather than in
specific tables or files. The model is
called the "ER model," and it allows for
many-to-many relationships between different
tables. This model provides a formal
framework for specifying database applications.

1. Relational Model and Normal Forms

1.1. INTRODUCTION

This paper is concerned with the application of
elementary matrix theory to matrix algebra, which provides
a formalism for the analysis of data organization systems.
The goal is to provide a framework for the design
and implementation of large-scale data organization systems.

1.2. ORDERING DEPENDENCIES

Elements of data in a data base may be stored in a number of ways,
some involving no ordering other than that of the data elements.
Others may be stored in a number of ways, some involving
ordering of entries. The ordering of entries in some cases
may be determined by the database structure, while in other
cases it may be determined by the application requirements.

1.3. IMPLEMENTATION

The implementation of the ER model involves the
development of a system for specifying and implementing
database applications. This system must provide
a formal specification language for
database applications, as well as a
mechanism for the implementation
of database applications.

1.4. CONCLUSION

The ER model provides a formal
framework for specifying database
applications, and it can be implemented using
existing database management systems.

Communication of the ACM 337
The relational model defines a database abstraction based on relations to avoid maintenance overhead.

Key tenets:
→ Store database in simple data structures (relations).
→ Physical storage left up to the DBMS implementation.
→ Access data through high-level language, DBMS figures out best execution strategy.
RELATIONAL MODEL

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

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

**Manipulation:** Programming interface for accessing and modifying a database's contents.
RELATIONAL MODEL

A relation is an 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 (if allowed).

<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>GZA</td>
<td>1990</td>
<td>USA</td>
</tr>
</tbody>
</table>

n-ary Relation = Table with n columns
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)

<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>GZA</td>
<td>1990</td>
<td>USA</td>
</tr>
</tbody>
</table>
A foreign key specifies that an attribute from one relation has to map to a tuple in another relation.

**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>GZA</td>
<td>1990</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>123</td>
<td>1994</td>
</tr>
<tr>
<td>33</td>
<td>Liquid Swords</td>
<td>789</td>
<td>1995</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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>album_id</th>
<th>id</th>
<th>name</th>
<th>year</th>
<th>ear</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>11</td>
<td>Enter the Wu-Tang</td>
<td>1993</td>
<td>993</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>St.Ides Mix Tape</td>
<td>1994</td>
<td>994</td>
</tr>
<tr>
<td>33</td>
<td>33</td>
<td>Liquid Swords</td>
<td>1995</td>
<td>995</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 to find the desired result based on sets / bags.

**Non-Procedural (Declarative):**
→ 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
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: \( \sigma_{\text{predicate}}(R) \)

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

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

\[
\begin{array}{|c|c|}
\hline
\text{a_id} & \text{b_id} \\
\hline
a2 & 103 \\
\hline
\end{array}
\]

```
SELECT * FROM R
WHERE a_id='a2' AND b_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: $\pi_{A_1,A_2,\ldots,A_n}(R)$

### Example

<table>
<thead>
<tr>
<th>a_id</th>
<th>b_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>101</td>
</tr>
<tr>
<td>a2</td>
<td>102</td>
</tr>
<tr>
<td>a2</td>
<td>103</td>
</tr>
<tr>
<td>a3</td>
<td>104</td>
</tr>
</tbody>
</table>

$\pi_{b_id-100,a_id}(\sigma_{a_id='a2'}(R))$

<table>
<thead>
<tr>
<th>b_id-100</th>
<th>a_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>a2</td>
</tr>
<tr>
<td>3</td>
<td>a2</td>
</tr>
</tbody>
</table>

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

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

Syntax: \((R \cup S)\)

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

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

\[
\text{(SELECT \,*\, FROM R) UNION ALL (SELECT \,*\, FROM S);}
\]
RELATIONAL ALGEBRA: INTERSECTION

Generate a relation that contains only the tuples that appear in both of the input relations.

Syntax: \((R \cap S)\)

<table>
<thead>
<tr>
<th>a_id</th>
<th>b_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>101</td>
</tr>
<tr>
<td>a2</td>
<td>102</td>
</tr>
<tr>
<td>a3</td>
<td>103</td>
</tr>
</tbody>
</table>

\[ R(a_{id}, b_{id}) \]

<table>
<thead>
<tr>
<th>a_id</th>
<th>b_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>a3</td>
<td>103</td>
</tr>
</tbody>
</table>

\[ S(a_{id}, b_{id}) \]

<table>
<thead>
<tr>
<th>a_id</th>
<th>b_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>a3</td>
<td>103</td>
</tr>
</tbody>
</table>

\[ (R \cap S) \]

\[ (\text{SELECT * FROM } R) \text{ INTERSECT } (\text{SELECT * FROM } S); \]
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: \((R - S)\)

<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>a4</td>
<td>104</td>
</tr>
<tr>
<td>a5</td>
<td>105</td>
</tr>
</tbody>
</table>

\[(R - S)\]

\[
\begin{align*}
\text{(SELECT } \ast \text{ FROM R)} & \\
\text{EXCEPT} & \\
\text{(SELECT } \ast \text{ FROM S)};
\end{align*}
\]
RELATIONAL ALGEBRA: PRODUCT

Generate a relation that contains all possible combinations of tuples from the input relations.

Syntax: (R × S)

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

\[
\begin{array}{c|c}
\text{a_id} & \text{b_id} \\
\hline
a1 & 101 \\
a2 & 102 \\
a3 & 103 \\
a4 & 104 \\
a5 & 105 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
\text{R.a_id} & \text{R.b_id} & \text{S.a_id} & \text{S.b_id} \\
\hline
a1 & 101 & a3 & 103 \\
a1 & 101 & a4 & 104 \\
a1 & 101 & a5 & 105 \\
a2 & 102 & a3 & 103 \\
a2 & 102 & a4 & 104 \\
a2 & 102 & a5 & 105 \\
a3 & 103 & a3 & 103 \\
a3 & 103 & a4 & 104 \\
a3 & 103 & a5 & 105 \\
\end{array}
\]

\[
\text{SELECT * FROM R CROSS JOIN S;}
\]

\[
\text{SELECT * FROM R, S;}
\]
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: \((R \bowtie S)\)

$$R(a_{id}, b_{id})$$

$$S(a_{id}, b_{id})$$

\begin{align*}
R & : \\
\begin{array}{|c|c|}
\hline
a_{id} & b_{id} \\
\hline
a_1 & 101 \\
\hline
a_2 & 102 \\
\hline
a_3 & 103 \\
\hline
\end{array} \\
S & : \\
\begin{array}{|c|c|}
\hline
a_{id} & b_{id} \\
\hline
a_3 & 103 \\
\hline
a_4 & 104 \\
\hline
a_5 & 105 \\
\hline
\end{array}
\end{align*}

$$\text{SELECT } * \text{ FROM } R \text{ NATURAL JOIN } S;$$

$$\text{SELECT } * \text{ FROM } R \text{ JOIN } S \text{ USING } (a_{id}, b_{id});$$
RELATIONAL ALGEBRA: EXTRA OPERATORS

- Rename ($ρ$)
- Assignment ($R ← S$)
- Duplicate Elimination ($δ$)
- Aggregation ($γ$)
- Sorting ($τ$)
- Division ($R ÷ S$)
OBSERVATION

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 record[0] == "GZA":
        print(int(record[1]))

SELECT year FROM artists
WHERE name = 'GZA';
```
DATA MODELS

Relational
Key/Value
Graph
**Document / Object** ← Leading Alternative
Wide-Column / Column-family
Array / Matrix / Vectors
Hierarchical
Network
Multi-Value

mongoDB
Couchbase
elastic
fauna
splunk>
Embed data hierarchy into a single object.

**Application Code**

```java
class Artist {
    int id;
    String name;
    int year;
    Album albums[];
}

class Album {
    int id;
    String name;
    int year;
}
```

```json
{
    "name": "GZA",
    "year": 1990,
    "albums": [
        {
            "name": "Liquid Swords",
            "year": 1995
        },
        {
            "name": "Beneath the Surface",
            "year": 1999
        }
    ]
}
```
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

Modern SQL
→ Make sure you understand basic SQL before the lecture.