Course Intro & Relational Model
TODAY’S AGENDA

Wait List
Overview
Course Logistics
Relational Model
Relational Algebra
WAIT LIST

There are currently 150 people on the waiting list. Max capacity is 100.

We will enroll people based on your S3 position.
COURSE OVERVIEW

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%)
Extra Credit (+10%)
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.
PROJ ECTS

You will build your own storage manager 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
All projects will use the new **BusTub** academic DBMS.
→ Source code will be released on Github.

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

You are allowed **four** slip days for either homework or projects.

You lose 25% of an assignment’s points for every 24hrs it is late.

Mark on your submission (1) how many days you are late and (2) how many late days you have left.
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.
**Database Group Meetings**

→ Mondays @ 4:30pm (GHC 8102)
→ [https://db.cs.cmu.edu](https://db.cs.cmu.edu)

**Advanced DBMS Developer Meetings**

→ Tuesdays @ 12:00pm (GHC 8115)
→ [https://github.com/cmu-db/terrier](https://github.com/cmu-db/terrier)
Databases
DATABASE

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

Databases are core the component of most computer applications.
Create a database that models a digital music store to keep track of artists and albums.

Things we need to store:

→ Information about Artists
→ What Albums those Artists released
FLAT FILE STRAWMAN

Store our database as comma-separated value (CSV) files that we manage in our own code.
→ Use a separate file per entity.
→ The application has to parse the files each time they want to read/update records.
Create a database that models a digital music store.

<table>
<thead>
<tr>
<th>Artist</th>
<th>Album</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Wu Tang Clan&quot;</td>
<td>&quot;Enter the Wu Tang&quot;, &quot;Wu Tang Clan&quot;, 1993</td>
</tr>
<tr>
<td>&quot;Notorious BIG&quot;</td>
<td>&quot;St. Ides Mix Tape&quot;, &quot;Wu Tang Clan&quot;, 1994</td>
</tr>
<tr>
<td>&quot;Ice Cube&quot;</td>
<td>&quot;AmeriKKKa's Most Wanted&quot;, &quot;Ice Cube&quot;, 1990</td>
</tr>
</tbody>
</table>
Example: Get the year that Ice Cube went solo.

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

```
Artist(name, year, country)
"Wu Tang Clan",1992,"USA"
"Notorious BIG",1992,"USA"
"Ice Cube",1989,"USA"
```
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?

How do we store that there are multiple artists on an album?
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 **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.

Edgar F. Codd
EARLY DBMSs

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.
RELATIONAL MODEL

Proposed in 1970 by Ted Codd.

Database abstraction to avoid this maintenance:
→ Store database in simple data structures.
→ Access data through high-level language.
→ Physical storage left up to implementation.
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 MODEL

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

← Most DBMSs
DATA MODEL

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

← NoSQL
DATA MODEL

Relational
Key/Value
Graph
Document
Column-family
Array / Matrix
Hierarchical
Network
← Machine Learning
DATA MODEL

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

← Obsolete / Rare
DATA MODEL

Relational ← This Course
Key/Value
Graph
Document
Column-family
Array / Matrix
Hierarchical
Network
RELATIONAL MODEL

**Structure:** The definition of relations and their contents.

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

**Manipulation:** How to access and modify a database’s contents.
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.

### 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>

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 you don't define one.

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

<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 you don't define one.

Auto-generation of unique integer primary keys:

→ **SEQUENCE** (SQL:2003)
→ **AUTO_INCREMENT** (MySQL)
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, 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>
# RELATIONAL MODEL: FOREIGN KEYS

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

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

<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** $(\text{artist_id}, \text{album_id})$

<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 Table

<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>

## Artist-Album Table

<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>

## Album Table

<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>
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>123</td>
<td>22</td>
</tr>
<tr>
<td>789</td>
<td>22</td>
</tr>
<tr>
<td>456</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>
How 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
How 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.

\[ \sigma \] Select
\[ \pi \] Projection
\[ \cup \] Union
\[ \cap \] Intersection
\[ - \] Difference
\[ \times \] Product
\[ \bowtie \] 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: $\sigma_{\text{predicate}}(R)$

![Table](image)

$\sigma_{a\text{\_id}='a2'}(R)$

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

$\sigma_{a\text{\_id}='a2' \land b\_id>102}(R)$

<table>
<thead>
<tr>
<th>a_id</th>
<th>b_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>a2</td>
<td>103</td>
</tr>
</tbody>
</table>

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>

\( R(a_id, b_id) \)

\( \sigma_{a_id='a2'}(R) \)

\( \Pi_{b_id-100, a_id}(R) \)

**SQL:**

```
SELECT b_id-100, a_id
FROM R
WHERE a_id = 'a2';
```
Generate a relation that contains all tuples that appear in either only one or both input relations.

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

### R(a_id, b_id)

<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>

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

\[
\text{(SELECT \ Star \ FROM } R) \ \text{UNION ALL} \ \text{(SELECT \ Star \ FROM } S)\]

### (R \cup 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>
<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>
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: \((R - S)\)

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

\[
\begin{array}{|c|c|}
\hline
\text{a}_\text{id} & \text{b}_\text{id} \\
\hline
\text{a3} & 103 \\
\text{a4} & 104 \\
\text{a5} & 105 \\
\hline
\end{array}
\]

\[
(\text{SELECT} \star \text{FROM } R) \text{ EXCEPT } (\text{SELECT} \star \text{FROM } S);
\]
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;
```
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: \((R \bowtie S)\)

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

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

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

\[
\text{SELECT} \ast \text{FROM R NATURAL JOIN S;}
\]
RELATIONAL ALGEBRA: EXTRA OPERATORS

Rename ($\rho$)
Assignment ($R \leftarrow S$)
Duplicate Elimination ($\delta$)
Aggregation ($\gamma$)
Sorting ($\tau$)
Division ($R \div S$)
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.

```python
for line in file:
    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.