Carnegie Mellon University

Intro to Database Systems (15-445/645)

Lecture #01

Relational Model & Algebra

SPRING 2024 ❯❯ Prof. Jignesh Patel

COURSE LOGISTICS

Course Policies + Schedule: Course Web Page

Discussion + Announcements: Piazza

Homeworks + Projects: Gradescope

Final Grades: Canvas

Do **not** post your solutions on Github.

Do **not** email instructors / TAs for help.

WAITLIST

We do **not** control the waitlist.

Admins will move students off the waitlist as spots become available.

This class will be offered in Fall'24 too!

LECTURE RULES

Do interrupt for the following reasons:

→ I'm speaking too fast.

→ You don't understand what I'm talking about.

I'll will **not** answer questions about the lecture immediately after class.

Email sent to you on Jan 2.

C++ REQUIREMENT

All the projects are in C++ …. If you are new to C++, you must pick it up quickly… If you can take and get all the questions on the following quizzes right, you are all set:

Scoping: https://www.learncpp.com/cpp-tutorial/chapter-7-summary-and-quiz/

Type Conversion: https://www.learncpp.com/cpp-tutorial/chapter-10-summary-and-quiz/

lvalues/rvalues: https://www.learncpp.com/cpp-tutorial/chapter-12-summary-and-quiz/

Stack and heap: https://www.learncpp.com/cpp-tutorial/chapter-20-summary-and-quiz/

Move Semantics: https://www.learncpp.com/cpp-tutorial/chapter-22-summary-and-quiz/

Templates: https://www.learncpp.com/cpp-tutorial/chapter-26-summary-and-quiz/

… **take it upon yourself to catch up** …

… also https://db.in.tum.de/teaching/ss23/c++praktikum/slides/lecture-10.2.pdf?lang=en

C++ Bootcamp: This Friday 1/19 from 3:30pm-4:30pm in GHC 6115

PROJECT 0: GOALS

→ Get you started on C++, so you are not surprised later.

→ Get you thinking about algorithms and concurrency.

→ P0 is about Conflict-Free Replicated Data Type (CRDT) – a distributed data structure that coordinates changes and produces an "eventually" consistent state.

→ Will connect to P0 with the last few topics in the class that is all about transactions)

→ I wanted an excuse to introduce CRDTs. As members of the data tribe, we should all know about this concept.

That is you by the end of this semester.

If you can't score 100% on P0, you can't stay in this class.

TODAY'S AGENDA

Database Systems Background

Relational Model

Relational Algebra

Alternative Data Models

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.

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.

Artist(name, year, country)

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

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

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 that application is running on a different machine?

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 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.

Preview of the relational model

```
CREATE TABLE Artist(name VARCHAR, year DATE, country CHAR(60));
CREATE TABLE Album(Albumid INTEGER, name VARCHAR, year DATE);
• • •
```

DATA MODELS

Relational
Key/Value
Graph
Document / XML / Object
Wide-Column / Column-family
Array / Matrix / Vectors
Hierarchical
Network
Multi-Value

DATA MODELS

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

DATA MODELS

- Relational
- Key/Value
- Graph
- Document / XML / Object
- Wide-Column / Column-family
- Array / Matrix / Vectors
- Hierarchical
- Network
- Multi-Value

← NoSQL (Key/Value, Graph, Document / XML / Object, Wide-Column / Column-family)

DATA MODELS

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

DATA MODELS

Relational
Key/Value
Graph
Document / XML / Object
Wide-Column / Column-family
Array / Matrix / Vectors
Hierarchical
Network
Multi-Value

← Obsolete / Legacy / Rare

DATA MODELS

Relational
Key/Value
Graph
Document / XML / Object
Wide-Column / Column-family
Array / Matrix / Vectors
Hierarchical
Network
Multi-Value

DATA MODELS

Relational ← This Course
Key/Value
Graph
Document / XML / Object
Wide-Column / Column-family
Array / Matrix / Vectors
Hierarchical
Network
Multi-Value

EARLY DBMSS

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.

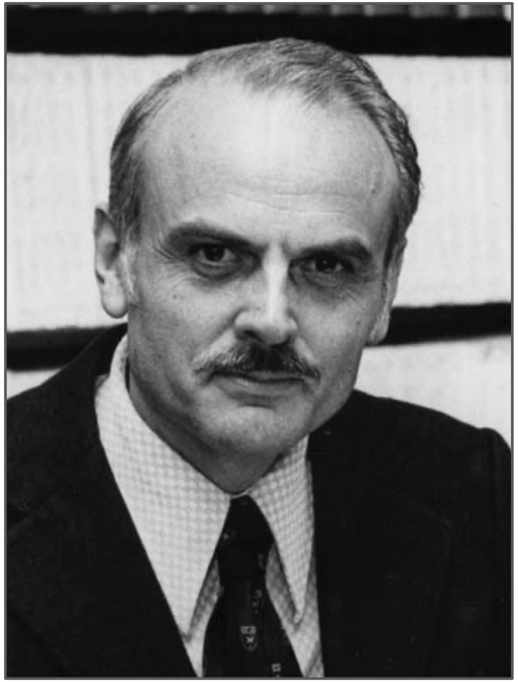
Programmers had to (roughly) know what queries the application would execute before they could deploy the database.

EARLY DBMSS

Ted Codd was a mathematician at IBM Research in the late 1960s.

Codd saw IBM's developers rewriting database programs every time the database's schema or layout changed.

Devised the relational model in 1969.



Edgar F. Codd

DERIVABILITY, REDUNDANCY AND CONSISTENCY OF RELATIONS
STORED IN LARGE DATA BANKS

E. F. Codd
Research Division
San Jose, California

ABSTRACT: The large, integrated data banks of the future will contain many relations of various degrees in stored form. It will not be unusual for this set of stored relations to be redundant. Two types of redundancy are defined and discussed. One type may be employed to improve accessibility of certain kinds of information which happen to be in great demand. When either type of redundancy exists, those responsible for control of the data bank should know about it and have some means of detecting any "logical" inconsistencies in the total set of stored relations. Consistency checking might be helpful in tracking down unauthorized (and possibly fraudulent) changes in the data bank contents.

RJ 599(# 12343) August 19, 1969

Information Retrieval **P. BAXENDALE, Editor**

A Relational Model of Data for Large Shared Data Banks

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

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity
CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

1. Relational Model and Normal Form

1.1. Introduction

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous 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 inconsistency* which are expected to become troublesome even in nondeductive systems.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for non-inferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal 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 sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

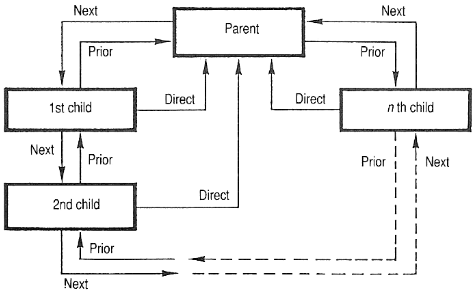
Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. Data Dependencies in Present Systems

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed *without logically impairing some application programs* is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is closely associated with the hardware-determined ordering of addresses. For example, the records of a file concerning parts might be stored in ascending order by part serial number. Such systems normally permit application programs to assume that the order of presentation of records from such a file is identical to (or is a subordering of) the

CODASYL



A closed chain of records in a navigational database model (e.g. CODASYL), with ***next pointers***, ***prior pointers*** and ***direct pointers*** provided by keys in the various records.

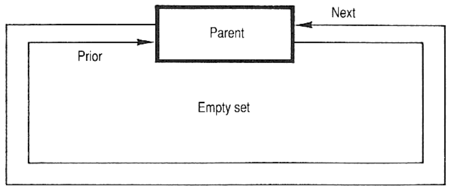


Illustration of an ***empty set***

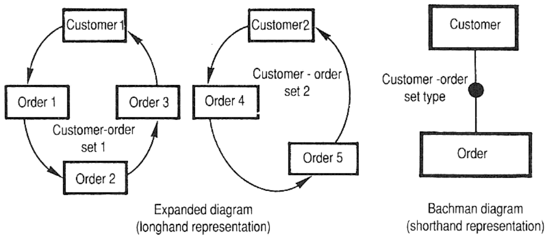


Illustration of a set type using a ***Bachman diagram***

The record set, basic structure of navigational (e.g. CODASYL) databse model. A set consists of one parent record (also called "the owner"), and n child records (also called members records)

COBOL/CODASYL camp:

1. The relational model is too mathematical. No mere mortal programmer will be able to understand your newfangled languages.
2. Even if you can get programmers to learn your new languages, you won't be able to build an efficient implementation of them.
3. On-line transaction processing applications want to do record-oriented operations.

Relational camp:

1. Nothing as complicated as the DBTG proposal can possibly be the right way to do data management.
2. Any set-oriented query is too hard to program using the DBTG data manipulation language.
3. The CODASYL model has no formal underpinning with which to define the semantics of the complex operations in the model.

The great debate:
"The Differences and Similarities Between the Data Base Set and Relational Views of Data."

ACM SIGFIDET Workshop on Data Description, Access, and Control in Ann Arbor, Michigan, held 1–3 May 1974

RELATIONAL MODEL

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.

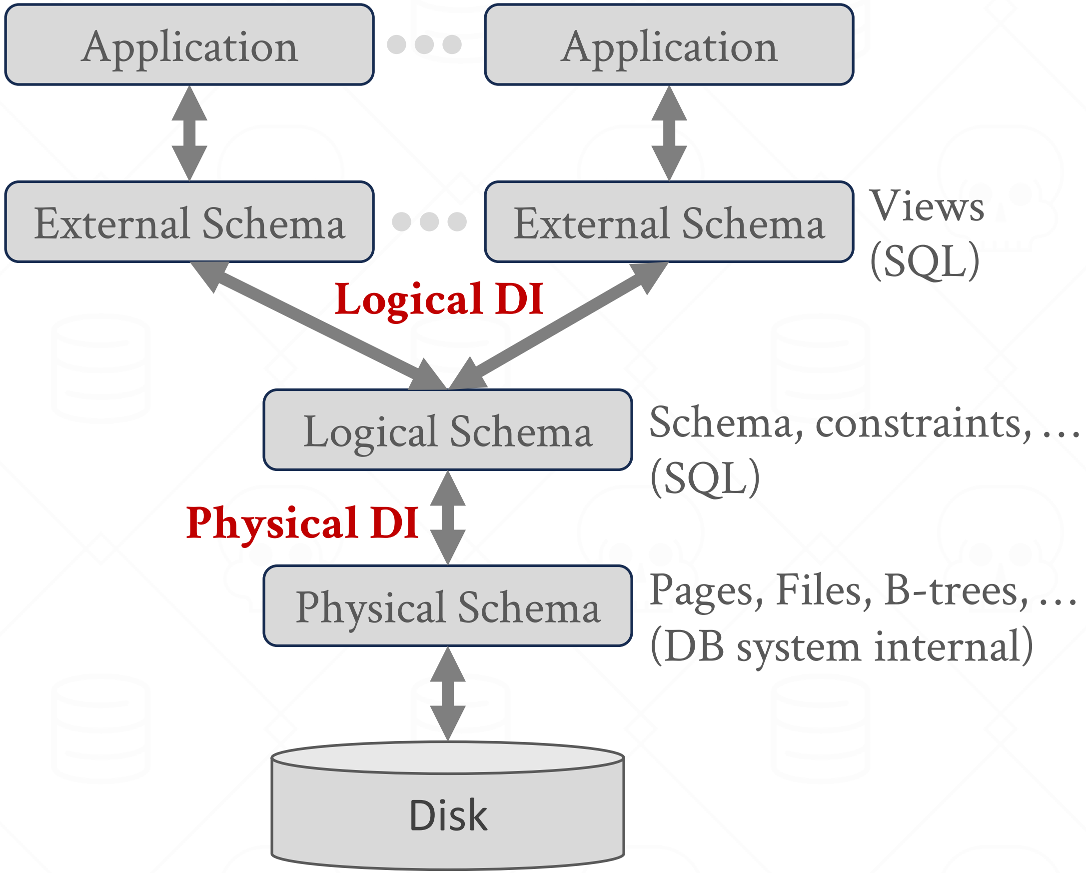
Manipulation: Programming interface for accessing and modifying a database's contents.

KEY CONCEPT: DATA INDEPENDENCE (DI)

Isolate the user/application from low level data representation.

→ The user only worries about application logic.

→ Database can optimize the layout (and re-optimize as the workload changes).



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).

Artist(name, year, country)

name	year	country
Wu-Tang Clan	1992	USA
Notorious BIG	1992	USA
GZA	1990	USA

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.

DBMS can auto-generation unique primary keys via an identity column:

IDENTITY (SQL Standard)

SEQUENCE (PostgreSQL / Oracle)

AUTO_INCREMENT (MySQL)

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

id	name	year	country
101	Wu-Tang Clan	1992	USA
102	Notorious BIG	1992	USA
103	GZA	1990	USA

RELATIONAL MODEL: FOREIGN KEYS

A foreign key specifies that an attribute from one relation maps to a tuple in another relation.

RELATIONAL MODEL: FOREIGN KEYS

Artist(id, name, year, country)

id	name	year	country
101	Wu-Tang Clan	1992	USA
102	Notorious BIG	1992	USA
103	GZA	1990	USA

Album(id, name, artists, year)

id	name	artists	year
11	Enter the Wu-Tang	101	1993
22	St.Ides Mix Tape	???	1994
33	Liquid Swords	103	1995

RELATIONAL MODEL: FOREIGN KEYS

Artist(id, name, year, country)

id	name	year	country
101	Wu-Tang Clan	1992	USA
102	Notorious BIG	1992	USA
103	GZA	1990	USA

Album(id, name, artists, year)

id	name	artists	year
11	Enter the Wu-Tang	101	1993
22	St.Ides Mix Tape	???	1994
33	Liquid Swords	103	1995

RELATIONAL MODEL: FOREIGN KEYS

Artist(<u>id</u>, name, year, country)

id	name	year	country
101	Wu-Tang Clan	1992	USA
102	Notorious BIG	1992	USA
103	GZA	1990	USA

ArtistAlbum(<u>artist_id</u>, <u>album_id</u>)

artist_id	album_id
101	11
101	22
103	22
102	22

Album(<u>id</u>, name, artists, year)

id	name	artists	year
11	Enter the Wu-Tang	101	1993
22	St.Ides Mix Tape	???	1994
33	Liquid Swords	103	1995

RELATIONAL MODEL: FOREIGN KEYS

ArtistAlbum(artist_id, album_id)

artist_id	album_id
101	11
101	22
103	22
102	22

Artist(id, name, year, country)

id	name	year	country
101	Wu-Tang Clan	1992	USA
102	Notorious BIG	1992	USA
103	GZA	1990	USA

Album(id, name, year)

id	name	year
11	Enter the Wu-Tang	1993
22	St.Ides Mix Tape	1994
33	Liquid Swords	1995

RELATIONAL MODEL: FOREIGN KEYS

Artist(id, name, year, country)

id	name	year	country
101	Wu-Tang Clan	1992	USA
102	Notorious BIG	1992	USA
103	GZA	1990	USA

ArtistAlbum(artist_id, album_id)

artist_id	album_id
101	11
101	22
103	22
102	22

Album(id, name, year)

id	name	year
11	Enter the Wu-Tang	1993
22	St.Ides Mix Tape	1994
33	Liquid Swords	1995

RELATIONAL MODEL: CONSTRAINTS

User-defined conditions that must hold for any instance of the database.

→ Can validate data within a single tuple or across entire relation(s).

→ DBMS prevents modifications that violate any constraint.

Unique key and referential (fkey) constraints are the most common.

SQL:92 supports global asserts but these are rarely used (too slow).

Artist(id, name, year, country)

id	name	year	country
101	Wu-Tang Clan	1992	USA
102	Notorious BIG	1992	USA
103	GZA	1990	USA

```
CREATE TABLE Artist(
name VARCHAR NOT NULL,
year DATE,
country CHAR(60),
CHECK (date_trunc('year', year) = year));
```

```
CREATE ASSERTION myAssert
 CHECK ( <SQL> );
```

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.

← Relational Algebra

Non-Procedural (Declarative):

The query specifies only what data is wanted and not how to find it.

← Relational Calculus

RELATIONAL ALGEBRA

Fundamental operations to retrieve and manipulate tuples in a relation.

→ Based on set algebra (unordered lists with no duplicates).

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
$\cup$	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: $\sigma_{\text{predicate}}(R)$

R(a_id,b_id)

a_id	b_id
a1	101
a2	102
a2	103
a3	104

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

a_id	b_id
a2	102
a2	103

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

a_id	b_id
a2	103

```sql
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.

→ Rearrange attributes' ordering.

→ Remove unwanted attributes.

→ Manipulate values to create derived attributes.

Syntax: $\Pi_{A1,A2,\ldots,An}(R)$

R(a_id,b_id)

a_id	b_id
a1	101
a2	102
a2	103
a3	104

$\Pi_{b\_id-100,a\_id}(\sigma_{a\_id='a2'}(R))$

b_id-100	a_id
2	a2
3	a2

```
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 ∪ S)

R(a_id,b_id)

a_id	b_id
a1	101
a2	102
a3	103

S(a_id,b_id)

a_id	b_id
a3	103
a4	104
a5	105

```
(SELECT * FROM R)
UNION
(SELECT * FROM S);
```

(R ∪ S)

a_id	b_id
a1	101
a2	102
a3	103
a4	104
a5	105

RELATIONAL ALGEBRA: INTERSECTION

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

Syntax: (R ∩ S)

R(a_id,b_id)

a_id	b_id
a1	101
a2	102
a3	103

S(a_id,b_id)

a_id	b_id
a3	103
a4	104
a5	105

(R ∩ S)

a_id	b_id
a3	103

```
(SELECT * FROM R)
INTERSECT
(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)

R(a_id,b_id)

a_id	b_id
a1	101
a2	102
a3	103

S(a_id,b_id)

a_id	b_id
a3	103
a4	104
a5	105

(R - S)

a_id	b_id
a1	101
a2	102

```
(SELECT * FROM R)
EXCEPT
(SELECT * FROM S);
```

RELATIONAL ALGEBRA: PRODUCT

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

Syntax: (R × S)

```
SELECT * FROM R CROSS JOIN S;
```

```
SELECT * FROM R, S;
```

R(a_id,b_id)

a_id	b_id
a1	101
a2	102
a3	103

S(a_id,b_id)

a_id	b_id
a3	103
a4	104
a5	105

(R × S)

R.a_id	R.b_id	S.a_id	S.b_id
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

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 ⋈ S)

R(a_id,b_id)

a_id	b_id
a1	101
a2	102
a3	103

S(a_id,b_id,val)

a_id	b_id	val
a3	103	XXX
a4	104	YYY
a5	105	ZZZ

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.

R(a_id,b_id)

a_id	b_id
a1	101
a2	102
a3	103

S(a_id,b_id,val)

a_id	b_id	val
a3	103	XXX
a4	104	YYY
a5	105	ZZZ

(R ⋈ S)

a_id	b_id	val
a3	103	XXX

R.a_id	R.b_id	S.a_id	S.b_id	S.val
a3	103	a3	103	XXX

Syntax: (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 ⋈ S)

R(a_id,b_id)

a_id	b_id
a1	101
a2	102
a3	103

S(a_id,b_id,val)

a_id	b_id	val
a3	103	XXX
a4	104	YYY
a5	105	ZZZ

R.a_id	R.b_id	~~S.a_id~~	~~S.b_id~~	S.val
a3	103	~~a3~~	~~103~~	XXX

(R ⋈ S)

a_id	b_id	val
a3	103	XXX

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)

a_id	b_id
a1	101
a2	102
a3	103

S(a_id,b_id,val)

a_id	b_id	val
a3	103	XXX
a4	104	YYY
a5	105	ZZZ

$(R \bowtie S)$

a_id	b_id	val
a3	103	XXX

```
SELECT * FROM R NATURAL JOIN S;
```

```
SELECT * FROM R JOIN S USING (a_id, b_id);
```

```
SELECT * FROM R JOIN S
    ON R.a_id = S.a_id AND R.b_id = S.b_id;
```

RELATIONAL ALGEBRA: EXTRA OPERATORS

Rename (ρ)

Assignment (R←S)

Duplicate Elimination (δ)

Aggregation (γ)

Sorting (τ)

Division (R÷S)

OBSERVATION

Relational algebra defines an ordering of the high-level steps of how to compute a query.

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

→ Example: Retrieve the joined tuples from **R** and **S** where **b_id** equals 102.

RELATIONAL MODEL: QUERIES

The relational model is independent of any query language implementation.

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

```
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 / XML / Object ← Leading Alternative
Wide-Column / Column-family
Array / Matrix / Vectors ← Hot these days
Hierarchical
Network
Multi-Value

DOCUMENT DATA MODEL

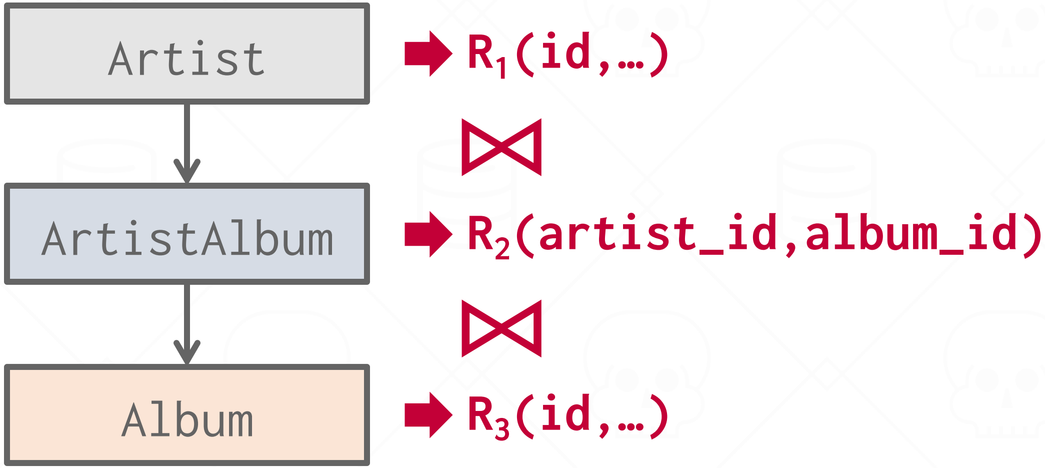
A collection of record documents containing a hierarchy of named field/value pairs.

→ A field's value can be either a scalar type, an array of values, or another document.

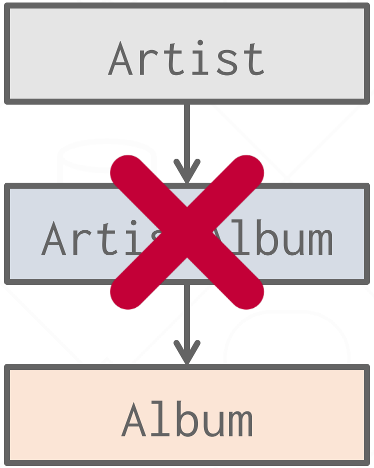
→ Modern implementations use JSON. Older systems use XML or custom object representations.

Avoid "relational-object impedance mismatch" by tightly coupling objects and database.

DOCUMENT DATA MODEL



DOCUMENT DATA MODEL



$R_1(\texttt{id},\ldots)$

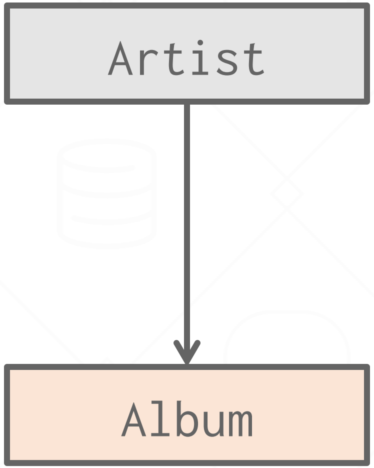
$\bowtie$

$R_2(\texttt{artist\_id},\texttt{album\_id})$

$\bowtie$

$R_3(\texttt{id},\ldots)$

DOCUMENT DATA MODEL



Application Code

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

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

VECTOR DATA MODEL

One-dimensional arrays used for nearest-neighbor search (exact or approximate).

→ Used for semantic search on embeddings generated by ML-trained transformer models (think ChatGPT).

→ Native integration with modern ML tools and APIs (e.g., LangChain, OpenAI).

At their core, these systems use specialized indexes to perform NN searches quickly.

Pinecone Weaviate milvus qdrant

marqo LanceDB featureform

VECTOR DATA MODEL

Album(id, name, year)

id	name	year
11	Enter the Wu-Tang	1993
22	St.Ides Mix Tape	1994
33	Liquid Swords	1995

OpenAI Hugging Face

Transformer

Embeddings

```
Id1 → [0.32, 0.78, 0.30, ...]
Id2 → [0.99, 0.19, 0.81, ...]
Id3 → [0.01, 0.18, 0.85, ...]
⋮
```

Query

Find albums similar to "Liquid Swords"

```
[0.02, 0.10, 0.24, ...]
```

Ranked List of Ids

Vector Index

HNSW, IVFFlat
Meta Faiss, Spotify Annoy

CONCLUSION

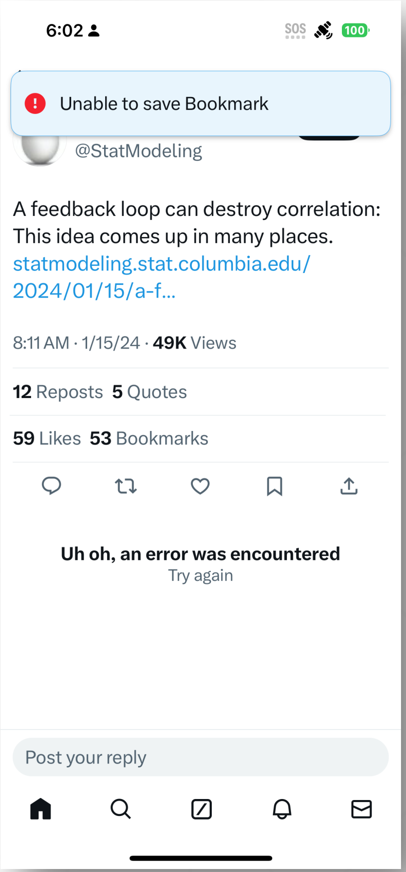
Databases are ubiquitous.

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.

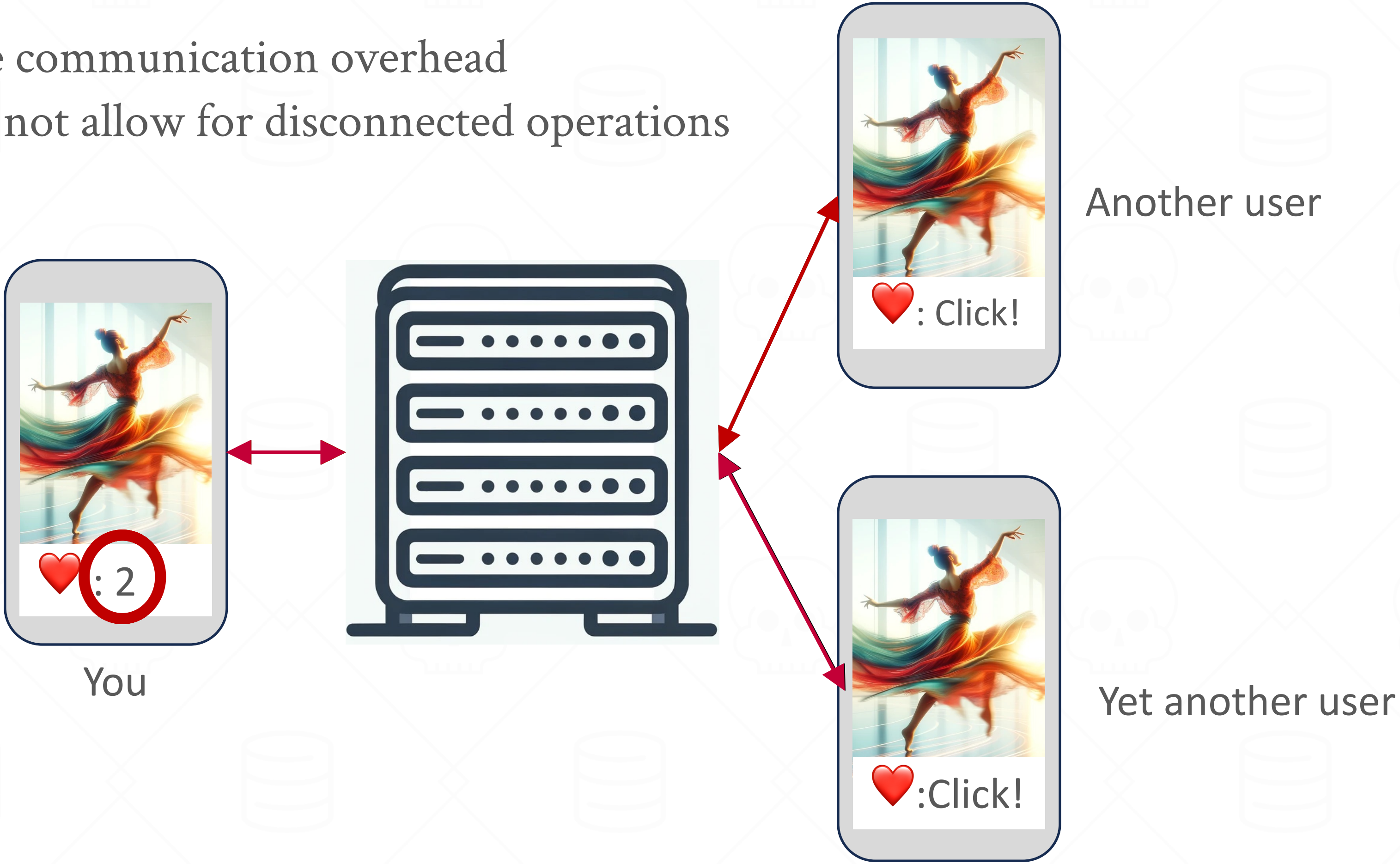
P0: CONFLICT-FREE REPLICATED DATA TYPE (CRDT)

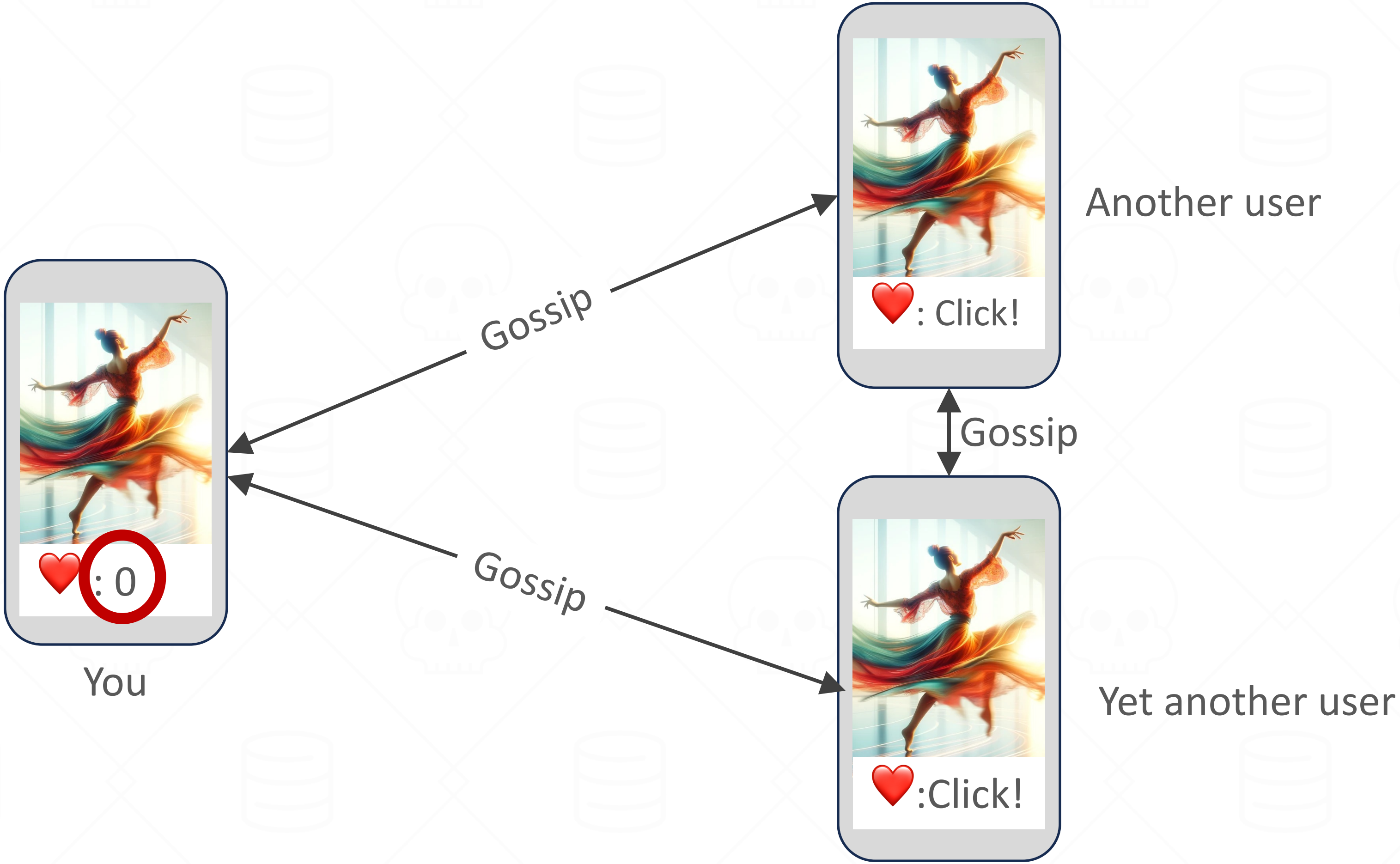
Problem: We want a distributed data structure (many copies) that allows local updates to be made independently and the states *eventually converge.*



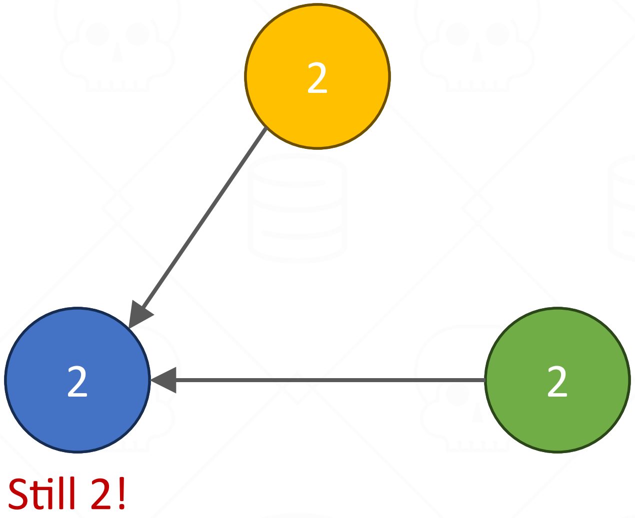
Large communication overhead

Does not allow for disconnected operations

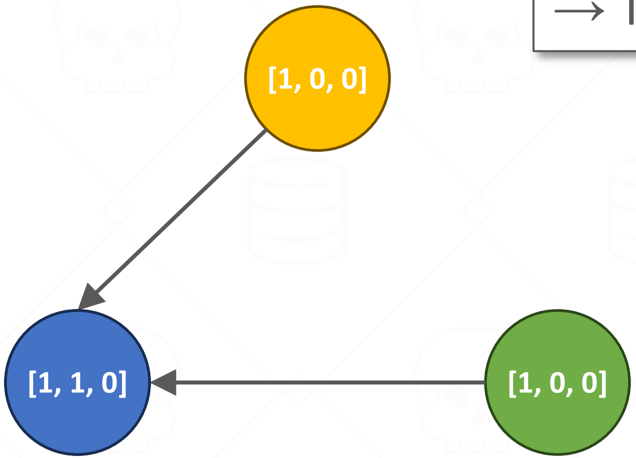



Gossip
Gossip
Gossip
: Click!
:Click!
: 0
Another user
Yet another user
You

Simple example: A "global" counter that each node can independently increment.
Nodes can gossip, anytime, and exchange their state.
Want each node to "eventually" have the same real global value.



Magic: Merge() function
→ Commutative
→ Associative
→ Idempotent



Merge() : myCounter[i] = MAX (myCounter[i], incomingCounter[i]) i = 0 … sizeof(myCounter)

Value() : SUM (myCounter[i] i = 0 … sizeof(myCounter))

P0: CONFLICT-FREE REPLICATED DATA TYPE (CRDT)

Problem: We want a distributed data structure (many copies) that allows local updates to be made independently and the states *eventually converge*.

For P0 the data structure is a set.

Homework #1 is due Jan 28th @ 11:59pm.

NEXT CLASS

Modern SQL

Make sure you understand basic SQL before the lecture.