COMP 421: Files & Databases

Lecture 1: The Relational Algebra

UNC
DEPARTMENT OF
COMPUTER SCIENCE

Wait List Update

- First few admissions off the waitlist are starting to trickle in
- If you are in line, stay in line

Announcements

- Thanks for RSVPing to Bootcamp 1:
  - Monday, 8/25 @ 6:30 FB009
  - Tuesday, 8/26 @ 6:30 FB009

  Note the room change!
- Please fill out the when2meet poll for office hours on canvas
  - Stopgap office hours: Friday 8/22, 10:00-11:30 & 2:00-4:00, FB 336
- You should be starting on P0
  - Please have your dev environment set up before Bootcamp 1
  - Follow instructions on P0 / Github README
  - Come resolve issues on Friday

Today…

- 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.

- Information we need to keep track of in our store:

→ Information about Artists

→ The Albums those Artists released

Flat File Strawman

- Store our database as comma-separated value (CSV) files that we manage ourselves in 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",1993,"USA"
"GZA",1991,"USA"
```

Album(name, artist, year)

```
"Enter the Wu-Tang","Wu-Tang Clan",1993
"St.Ides Mix Tape","Wu-Tang Clan",1994
"Liquid Swords","GZA",1995
```

Flat File Strawman

- Example: Get the year of the first GZA album.

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

```
"Wu-Tang Clan",1992,"USA"
"Notorious BIG",1993,"USA"
"GZA",1991,"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 spelled 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?

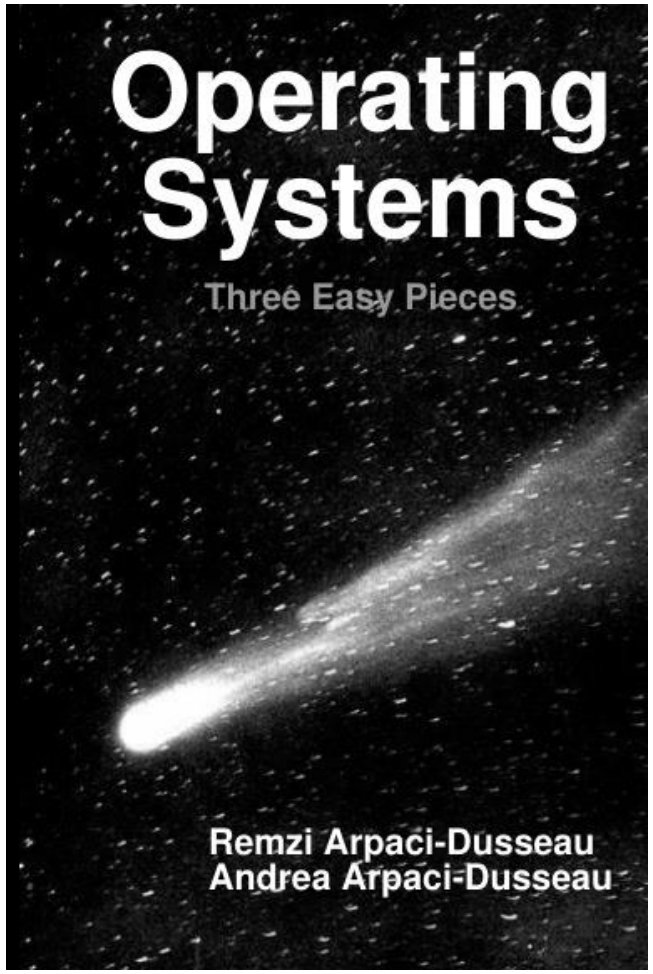
Flat Files: Implementation

- 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?

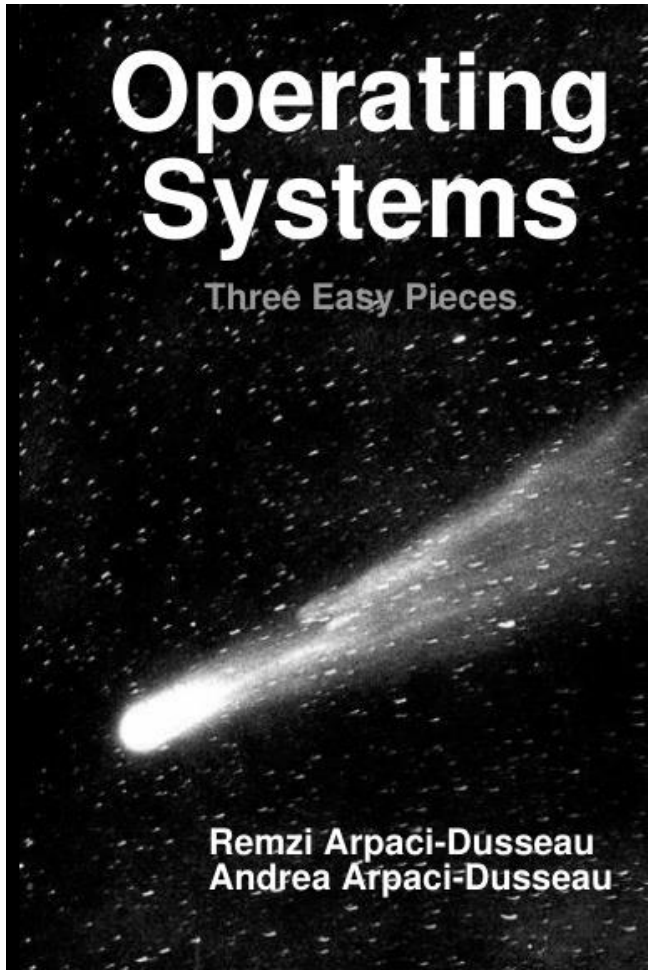
Flat Files: Durability



Some applications (such as databases) don't enjoy this trade-off. Thus, to avoid unexpected data loss due to write buffering, they simply force writes to disk, by calling `fsync()`, by using **direct I/O** interfaces that work around the cache, or by using the **raw disk** interface and avoiding the file system altogether[2]. While most applications live with the trade-offs made by the file system, there are enough controls in place to get the system to do what you want it to, should the default not be satisfying.

[2]Take a database class to learn more about old-school databases and their former insistence on avoiding the OS and controlling everything themselves. But watch out! Those database types are always trying to bad mouth the OS. Shame on you, database people. Shame.

Flat Files:



Some applica
to avoid unexp
writes to disk,
work around t
the file system
offs made by t
system to do v

[2]Take a databas
tence on avoiding
database types are a

s,
ce
at
ng
de-
the
.

r former insis-
ch out! Those
people. Shame.

Are You Sure You Want to Use MMAP in Your Database Management System?

Andrew Crotty
Carnegie Mellon University
andrewcr@cs.cmu.edu

Viktor Leis
University of Erlangen-Nuremberg
viktor.leis@fau.de

Andrew Pavlo
Carnegie Mellon University
pavlo@cs.cmu.edu

ABSTRACT

Memory-mapped (mmap) file I/O is an OS-provided feature that maps the contents of a file on secondary storage into a program's address space. The program then accesses pages via pointers as if the file resided entirely in memory. The OS transparently loads pages only when the program references them and automatically evicts pages if memory fills up.

mmap's perceived ease of use has seduced database management system (DBMS) developers for decades as a viable alternative to implementing a buffer pool. There are, however, severe correctness and performance issues with mmap that are not immediately apparent. Such problems make it difficult, if not impossible, to use mmap correctly and efficiently in a modern DBMS. In fact, several popular DBMSs initially used mmap to support larger-than-memory databases but soon encountered these hidden perils, forcing them to switch to managing file I/O themselves after significant engineering costs. In this way, mmap and DBMSs are like coffee and spicy food: an unfortunate combination that becomes obvious after the fact.

Since developers keep trying to use mmap in new DBMSs, we wrote this paper to provide a warning to others that mmap is not a suitable replacement for a traditional buffer pool. We discuss the main shortcomings of mmap in detail, and our experimental analysis demonstrates clear performance limitations. Based on these findings, we conclude with a prescription for when DBMS developers might consider using mmap for file I/O.

1 INTRODUCTION

An important feature of disk-based DBMSs is their ability to support databases that are larger than the available physical memory. This functionality allows a user to query a database as if it resides entirely in memory, even if it does not fit all at once. DBMSs achieve this illusion by reading pages of data from secondary storage (e.g., HDD, SSD) into memory on demand. If there is not enough memory for a new page, the DBMS will evict an existing page that is no longer needed in order to make room.

Traditionally, DBMSs implement the movement of pages between secondary storage and memory in a buffer pool, which interacts with secondary storage using system calls like `read` and `write`. These file I/O mechanisms copy data to and from a buffer in user space, with the DBMS maintaining complete control over how and when it transfers pages.

Alternatively, the DBMS can relinquish the responsibility of data movement to the OS, which maintains its own file mapping and page cache. The POSIX mmap system call maps a file on secondary storage into the virtual address space of the caller (i.e., the DBMS), and the OS will then load pages lazily when the DBMS accesses them. To the DBMS, the database appears to reside fully in memory, but the OS handles all necessary paging behind the scenes rather than the DBMS's buffer pool.

On the surface, mmap seems like an attractive implementation option for managing file I/O in a DBMS. The most notable benefits are ease of use and low engineering cost. The DBMS no longer needs to track which pages are in memory, nor does it need to track how often pages are accessed or which pages are dirty. Instead, the DBMS can simply access disk-resident data via pointers as if it were accessing data in memory while leaving all low-level page management to the OS. If the available memory fills up, then the OS will free space for new pages by transparently evicting (ideally unneeded) pages from the page cache.

From a performance perspective, mmap should also have much lower overhead than a traditional buffer pool. Specifically, mmap does not incur the cost of explicit system calls (i.e., `read`/`write`) and avoids redundant copying to a buffer in user space because the DBMS can access pages directly from the OS page cache.

Since the early 1980s, these supposed benefits have enticed DBMS developers to forgo implementing a buffer pool and instead rely on the OS to manage file I/O [36]. In fact, the developers of several well-known DBMSs (see Section 2.3) have gone down this path, with some even touting mmap as a key factor in achieving good performance [20].

Unfortunately, mmap has a hidden dark side with many sordid problems that make it undesirable for file I/O in a DBMS. As we describe in this paper, these problems involve both data safety and system performance concerns. We contend that the engineering steps required to overcome them negate the purported simplicity of working with mmap. For these reasons, we believe that mmap adds too much complexity with no commensurate performance benefit and strongly urge DBMS developers to avoid using mmap as a replacement for a traditional buffer pool.

The remainder of this paper is organized as follows. We begin with a short background on mmap (Section 2), followed by a discussion of its main problems (Section 3) and our experimental analysis (Section 4). We then discuss related work (Section 5) and conclude with a summary of our guidance for when you *might* consider using mmap in your DBMS (Section 6).

2 BACKGROUND

This section provides the relevant background on mmap. We begin with a high-level overview of memory-mapped file I/O and the POSIX mmap API. Then, we discuss real-world implementations of mmap-based systems.

 12th Annual Conference on Innovative Data Systems Research (CIDR '22). January 9-12, 2022, Chaminade, USA.

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.
  - This defines the structure of data for a data model.
  - Otherwise, you have random bits with no meaning.

Data Models

- Relational ← Most DBMSs
- Key/Value ← Simple Apps / Caching
- Graph
- Document / JSON / XML / Object
- Wide-Column / Column-family

← NoSQL

- Array (Vector, Matrix, Tensor) ← ML / Science
- Hierarchical
- Network
- Semantic
- Entity-Relationship

← Obsolete / Legacy / Rare

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.

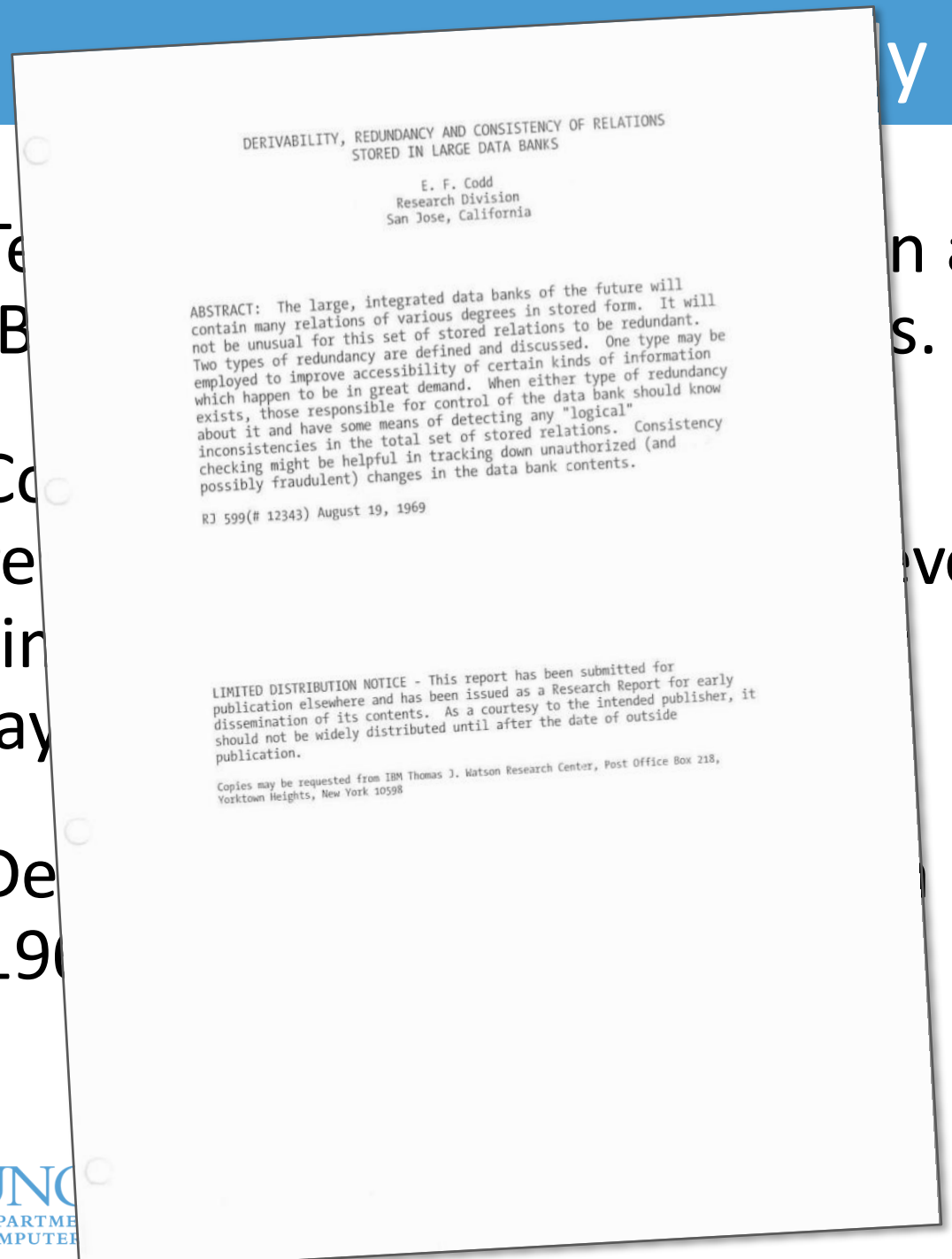
y DBMS

- Te
  IB
- Co
  re
  tir
  lay
- De
  19

n at

s.

very



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

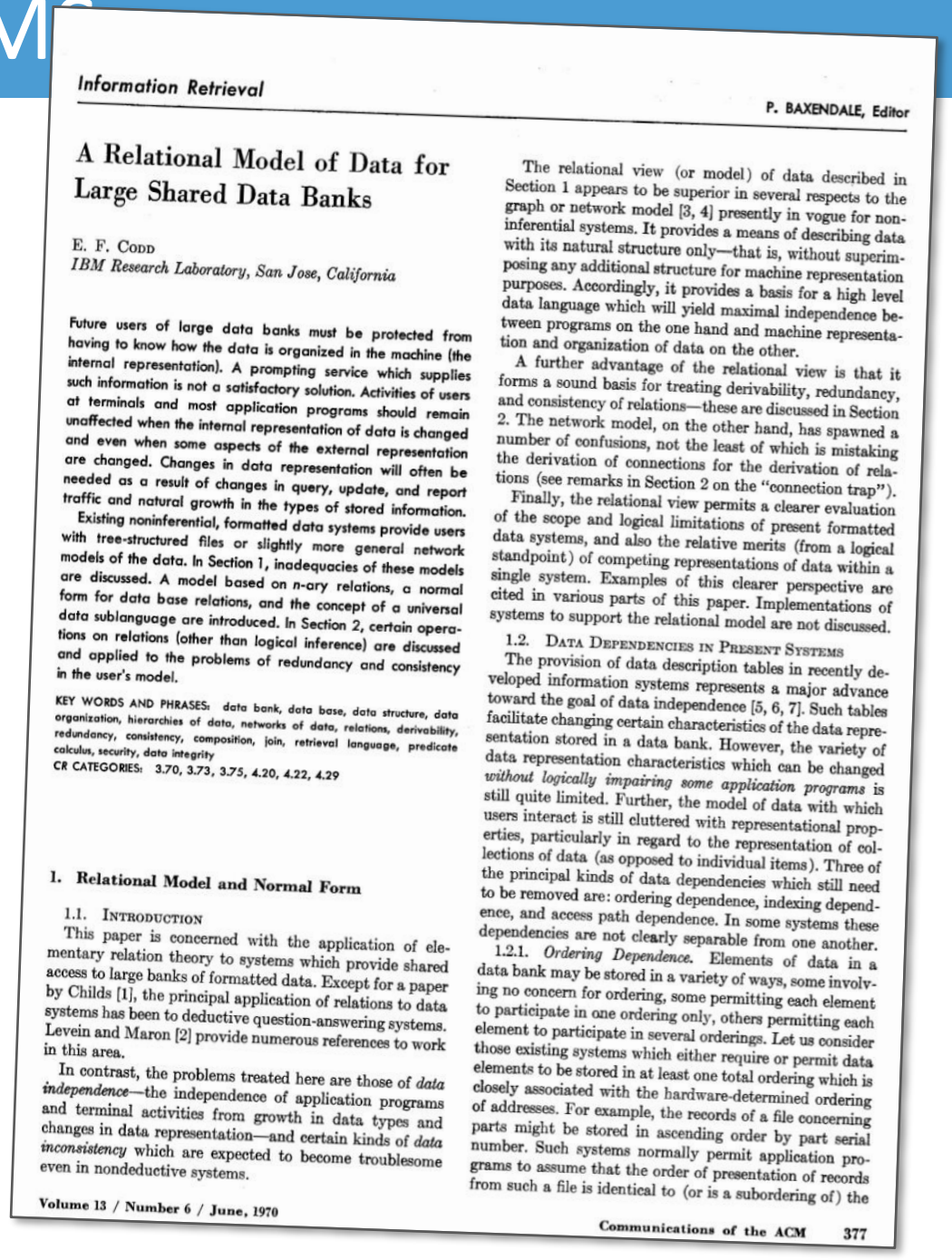
E. F. Codd
Research Division
San Jose, California

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

LIMITED DISTRIBUTION NOTICE - This report has been submitted for publication elsewhere and has been issued as a Research Report for early dissemination of its contents. As a courtesy to the intended publisher, it should not be widely distributed until after the date of outside publication.

Copies may be requested from IBM Thomas J. Watson Research Center, Post Office Box 218, Yorktown Heights, New York 10598



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 noninferential 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.

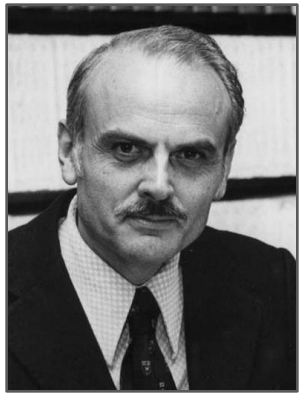
1.2. Data Dependencies in Present Systems

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.

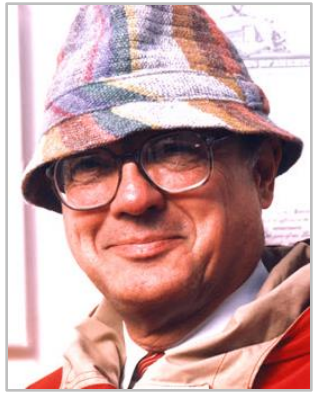
1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing 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

- **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



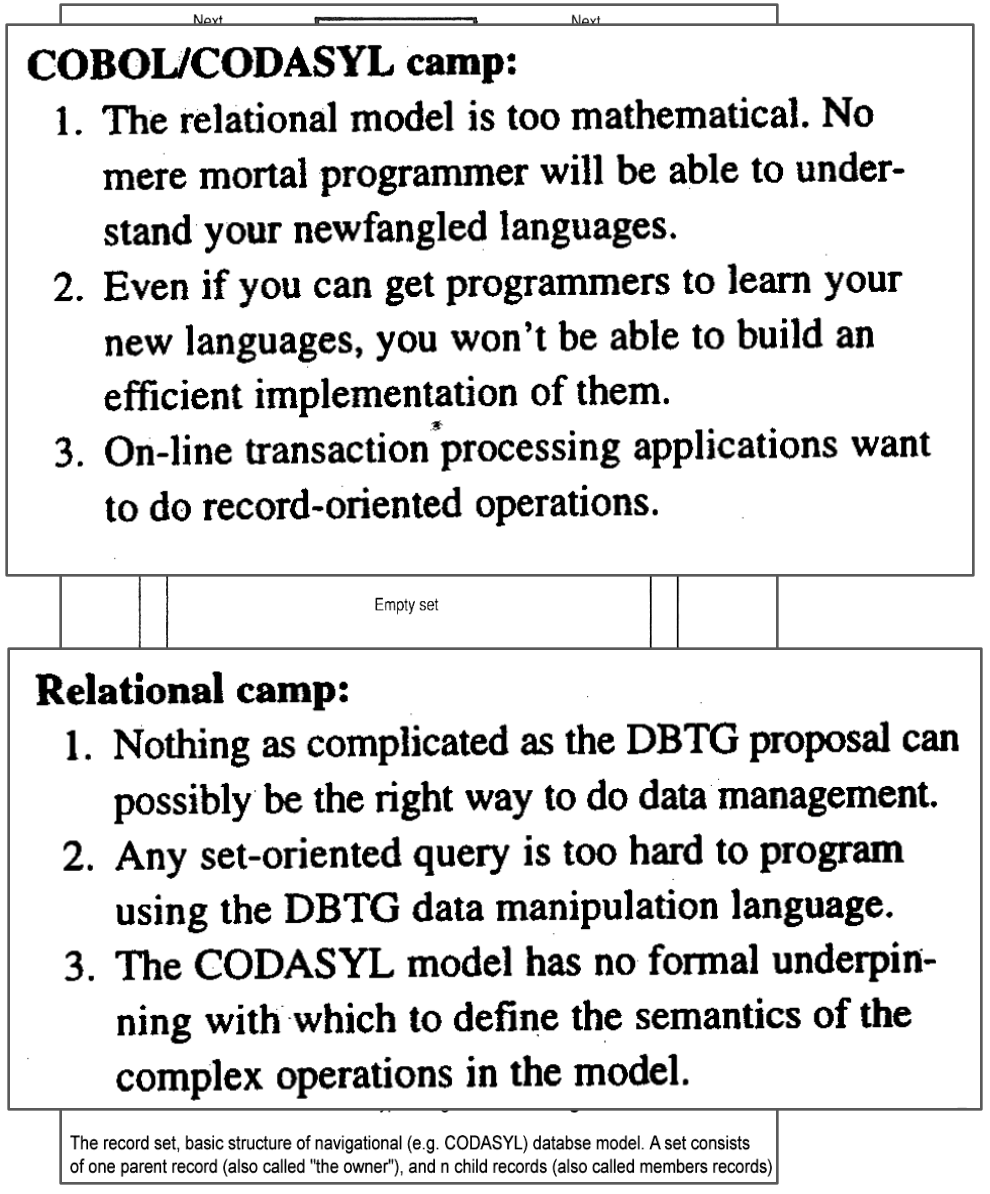
Codd



Bachman

Gray

Stonebraker



Relational Model

- The **relational model** defines a database abstraction based on relations to avoid maintenance overhead.

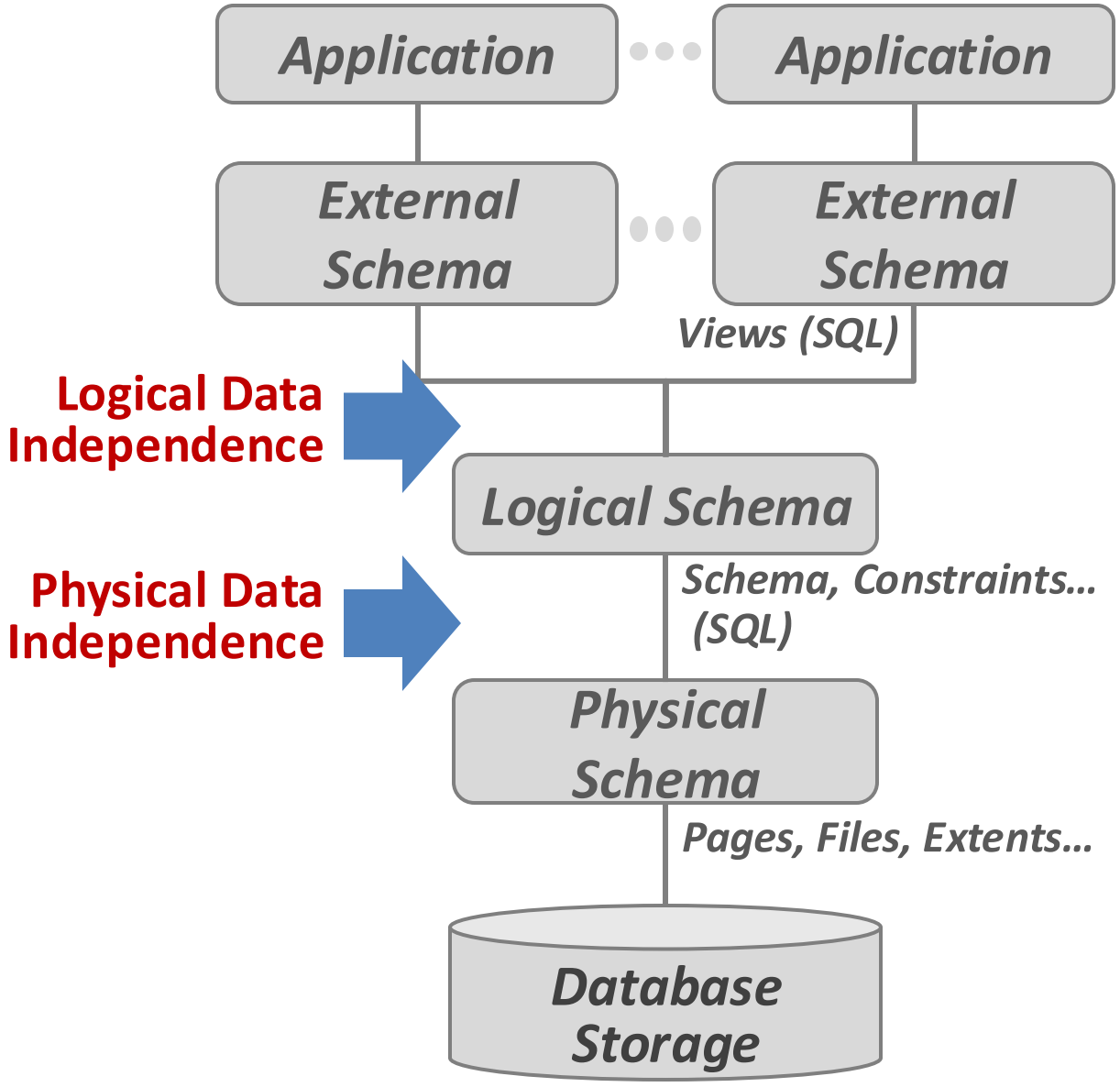
- Key tenets:
  - Represent database using simple data structures (relations).
  - Physical storage left up to the DBMS implementation.
  - Access data through high-level language, DBMS figures out best execution strategy.

Relational Model

- **Structure:** The definition of the database's relations and their contents independent of their physical representation.
- **Integrity:** Ensure the database's contents satisfy constraints.
- **Manipulation:** Programming interface for accessing and modifying a database's contents.

Data Independence

- Isolate the user/application from low-level data representation.
  - The user only worries about high-level application logic.
  - DBMS optimizes the layout according to operating environment, database contents, and workload.
  - DBMS can then re-optimize the database if/when these factors changes.



Relational Model

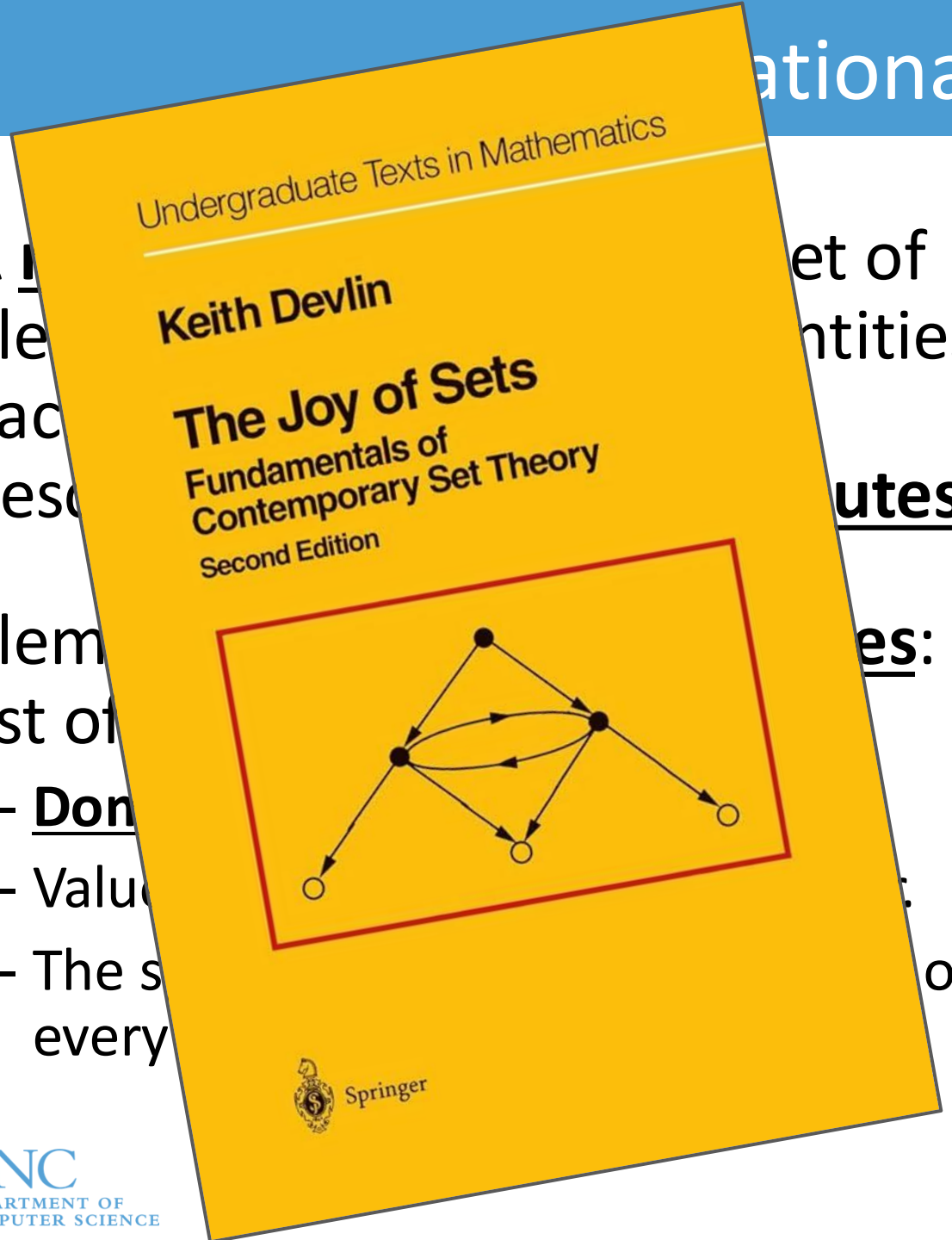
- A **relation** is an unordered set of elements describing some entities. Each element in a relation is described via the same **attributes**.
- Elements are modeled as **tuples**: a list of attribute values.
  - **Domain**: possible attribute values
  - Values are (normally) atomic/scalar.
  - The 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

ational Model



- A **r** et of ele ntities. Eac **utes**. desc
- Elem **es**: a list of
  - **Don**
  - Valu :
  - The s of every

`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

- **Structure:** The definition of the database's relations and their contents independent of their physical representation.
- **Integrity:** Ensure the database's contents satisfy constraints.
- **Manipulation:** Programming interface for accessing and modifying a database's contents.

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 must map to a some attribute of a tuple in another relation.

ArtistAlbum(artist_id, album_id)

artist_id	album_id
101	11
101	22
103	22
102	22

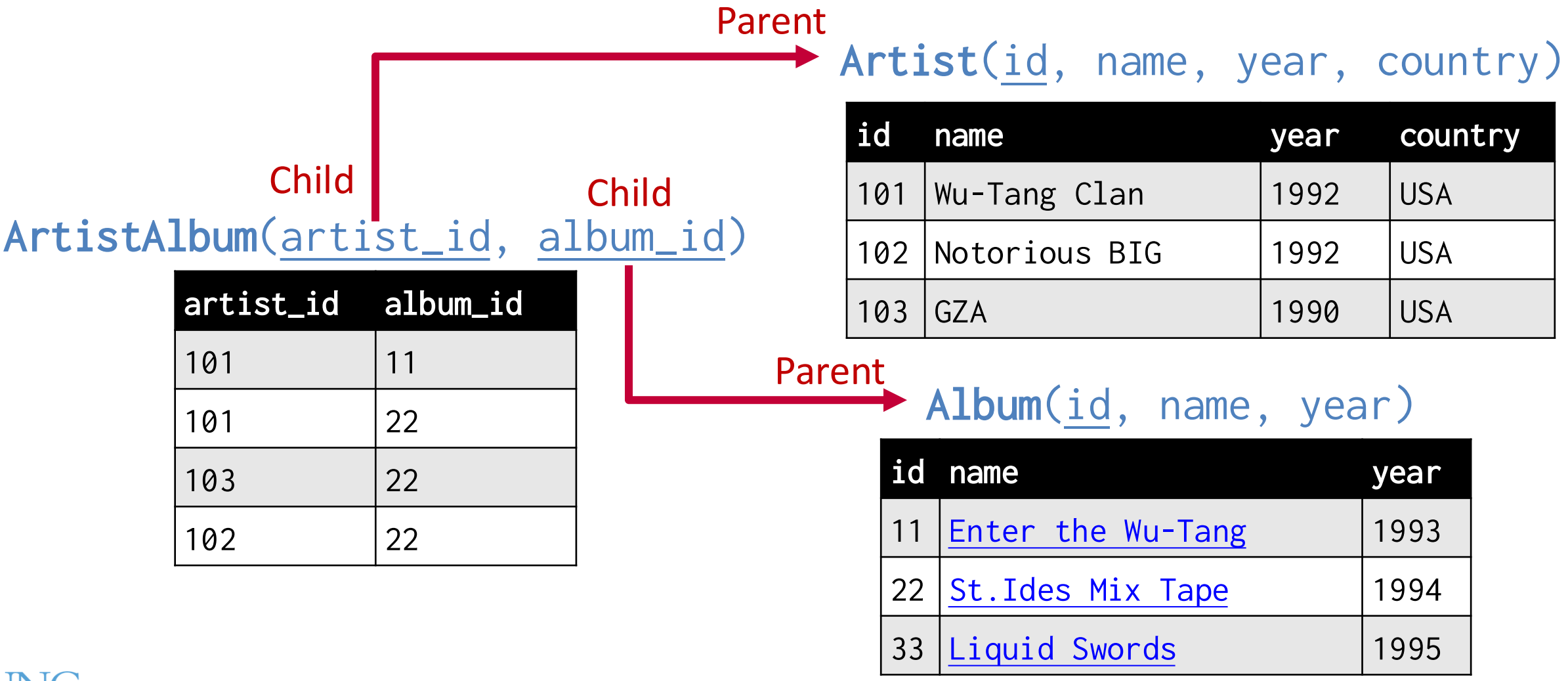
Artist(id, name, year, country)

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

Album(id, name, artist, year)

id	r	id	name	year	ear
11	E	11	Enter the Wu-Tang	1993	993
22	S	22	St.Ides Mix Tape	1994	994
33	L	33	Liquid Swords	1995	995

Relational Model: Foreign Keys



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

artist_id	album_id
101	11
101	22
103	22
102	22

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 INT,
 country CHAR(60),
 CHECK (year > 1900)
);
```

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

Relational Model

- **Structure:** The definition of the database's relations and their contents independent of their physical representation.
- **Integrity:** Ensure the database's contents satisfy constraints.
- **Manipulation:** Programming interface for accessing and modifying a database's contents.

Data Manipulation Languages (DML)

- The API that a DBMS exposes to applications 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
∪	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:** $\sigma_{predicate}(R)$

R(a_id,b_id)

a_id	b_id
a1	101
a2	102
a2	103
a3	104

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

a_id	b_id
a2	102
a2	103

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

a_id	b_id
a2	103

```
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 at least one input relation.
- **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: Cartesian 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)

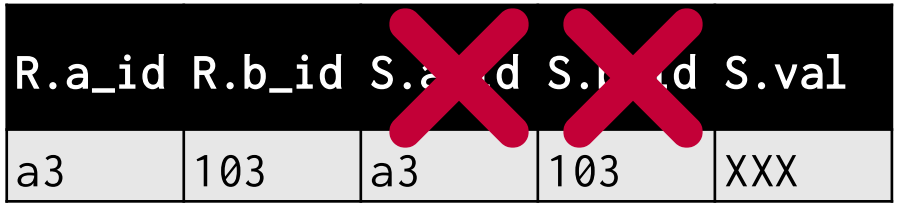
a_id	b_id
a1	101
a2	102
a3	103

S(a_id,b_id,val)

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

(R ⋈ 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.

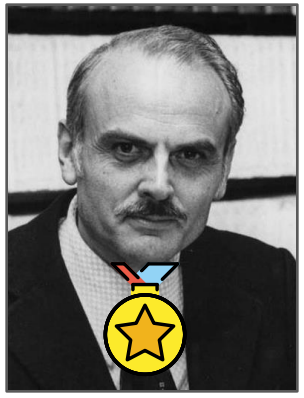
Data Manipulation Languages (DML)

- **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



Codd's Theorem

Relational Algebra ⇔ Relational Calculus

"Logically Equivalent"

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';
```

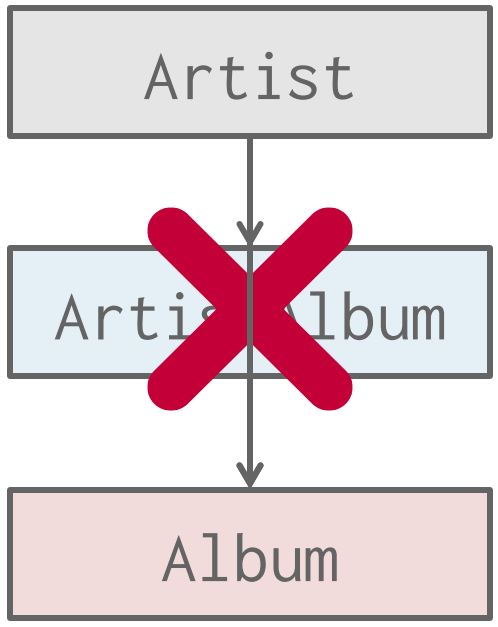
Data Models

- Relational
- Key/Value ← This Course
- Graph
- Document / JSON / XML / Object ← Leading Alternative
- Wide-Column / Column-family
- Array (Vector, Matrix, Tensor) ← New Hotness
- Hierarchical
- Network
- Semantic
- Entity-Relationship

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



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.

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, ..]

Vector Index

Ranked List of Ids

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.

Next Class

- Modern SQL
  - Make sure you understand basic SQL (e.g. textbook Ch. 3) before the lecture.