Carnegie Mellon University

Database Systems

15-445/645 SPRING 2026

ANDY PAVLO
JIGNESH PATEL

Lecture #01

Relational Model & Algebra

WAITLIST

We do not control the waitlist. Admins will move students off the waitlist as spots become available.
→ See FAQ #2

To improve your chances of enrolling (though not a guarantee), stay in the class and complete Project #0.

We do not know whether this course will be offered in Fall 2026 yet.


ClickHouse


DATASTAX


dbt


FIREBOLT


MotherDuck


RelationalAI


SingleStore


spiral


TiDB


Yellowbrick


yugabyteDB

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)

COURSE LOGISTICS

Course Syllabus + Schedule: Course Web Page

Discussion + Announcements: Piazza

Homework + Projects: Gradescope

Final Grades: Canvas

Non-CMU students can complete assignments using Gradescope (Code: **5R4XPZ**).

→ See FAQ #7

→ Do not post your solutions on Github.

→ Do not email instructors / TAs for help.

→ Discord Channel: https://discord.gg/YF7dMCg

LECTURE RULES

Do interrupt us for the following reasons:

→ I am speaking too fast.
→ You don't understand what I am talking about.
→ You have a database-related question.

Do not interrupt for the following reasons:

→ Whether you can use the bathroom.
→ Questions about blockchains.

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

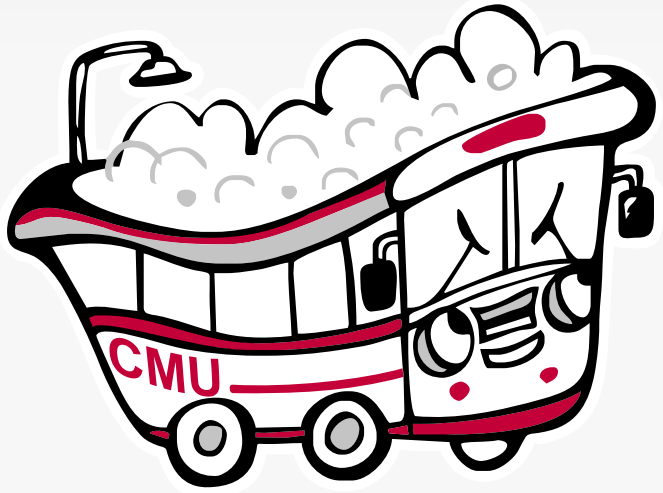
PROJECTS

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++20.
→ See the 15-445/645 Bootcamp.

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

We will hold an online recitation for each project after it is released.



BusTub

PROJECT 0

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

Get you thinking about algorithms and concurrency.

Project #0 is released today: Count-min Sketch

→ Due on Sunday January 25th @ 11:59pm

→ No late days are allowed!

Each student must score 100% on this project before the deadline or you will be asked to drop the course.

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.

→ You are allowed to use generative AI tools.

Plagiarism is not tolerated. You will get lit up.

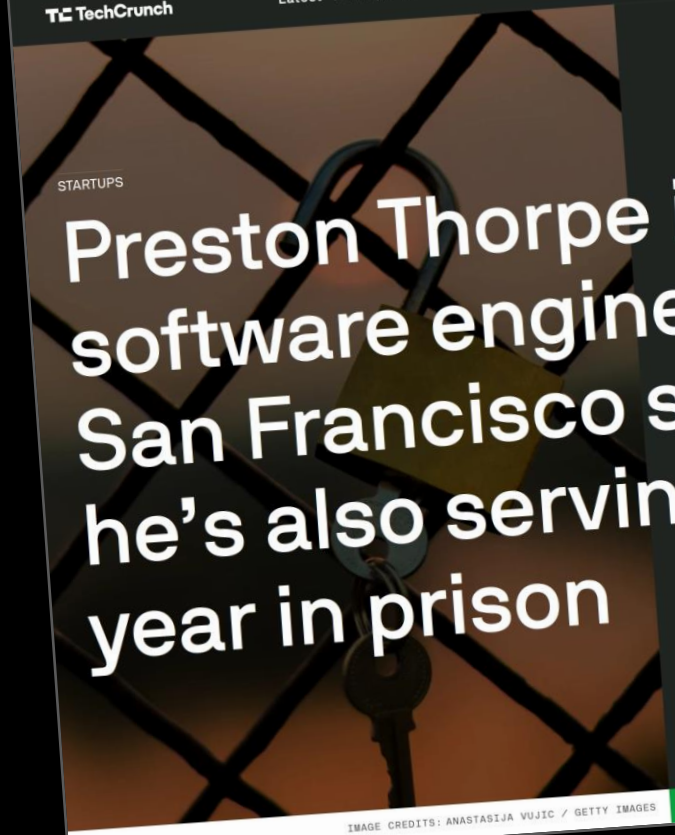
→ Please ask instructors (not TAs!) if you are unsure.

See CMU's Policy on Academic Integrity for additional information.

DB FLASH TALKS

Quick 10-minute talks from CMU-DB IAP partners about their DBMSs at end of every Wednesday lecture.

It is late in the hiring season, but we will post internship + full-time openings on Piazza.



STARTUPS

Preston Thorpe is a software engineer at a San Francisco startup — he's also serving his 11th year in prison

Amanda Silberling

8:48 AM PDT · July 24, 2025

IMAGE CREDITS: ANASTASIJA VUJIC / GETTY IMAGES

TechCrunch

Latest Startups Venture Apple Security AI Apps | Events

STARTUPS

Preston Thorpe is software engine San Francisco st he's also servin year in prison

IMAGE CREDITS: ANASTASIJA VUJIC / GETTY IMAGES

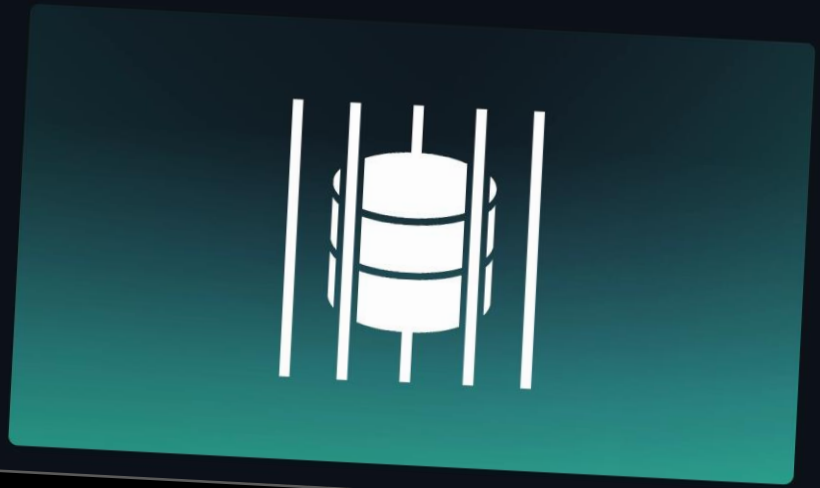
Docs Cloud Pricing Blog Changelog

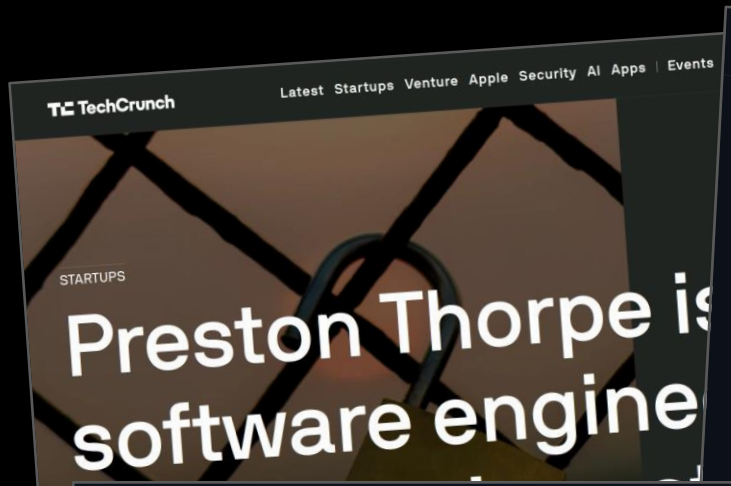
Login Sign Up

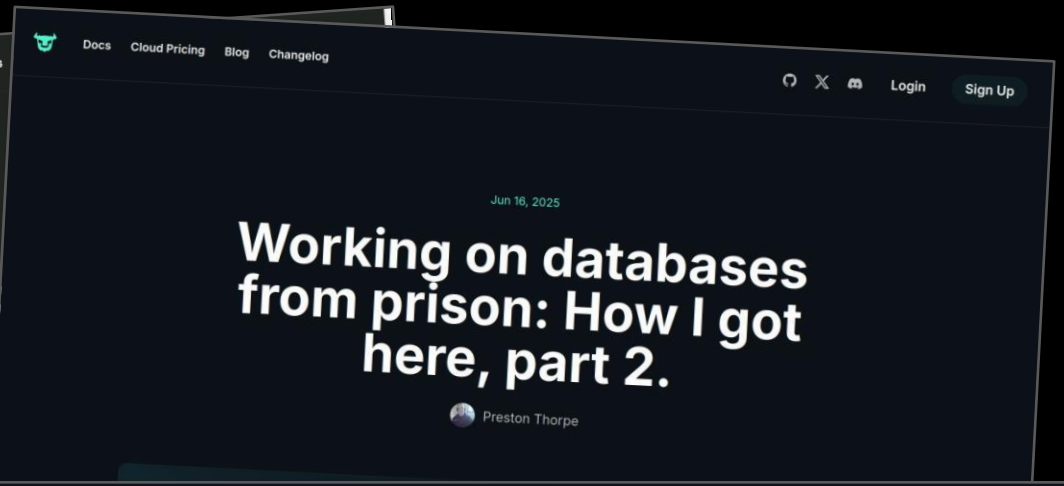
Jun 16, 2025

Working on databases from prison: How I got here, part 2.

Preston Thorpe







Helping build Limbo quickly became my new obsession. I split my time between my job and diving deep into SQLite source code, academic papers on database internals, and Andy Pavlo's CMU lectures. I was active on the Turso Discord but I don't think I considered whether anyone was aware that one of the top contributors was doing so from a prison cell. My story and information are linked on my GitHub, but it's subtle enough where you could miss it if you didn't read the

DATABASE PRISON PROGRAM

CMU's Intro to Database Systems course available to people locked on the inside at no cost.

If you are in prison or know somebody in prison that wants to learn about databases, please contact:
db-prison@cs.cmu.edu

Sponsored By:

Databases

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.

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",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 to ensure that the artist's name is consistent for all their album entries?
→ Example: "Wu-Tang Clan" vs. "WuTang Clan"

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 computer 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.

→ Rules that define the types of things that could exist and how they relate.

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

→ This defines the structure of database for a data model.

→ Otherwise, you have random bits with no meaning.

DATA MODELS

A **data model** is a collection of concepts for describing the data in a database.

→ Rules that define the types of things that could exist and how they relate.

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

→ This defines the structure of database for a data model.

→ Otherwise, you have random bits with no meaning.

DATA MODELS

Relational ← Most DBMSs

Key/Value

Graph

Document / JSON / XML / Object

Wide-Column / Column-family

Array (Vector, Matrix, Tensor)

Hierarchical

Network

Semantic

Entity-Relationship

DATA MODELS

Relational

Key/Value ← Simple Apps / Caching

Graph

Document / JSON / XML / Object

Wide-Column / Column-family

Array (Vector, Matrix, Tensor)

Hierarchical

Network

Semantic

Entity-Relationship

DATA MODELS

Relational

Key/Value

Graph

Document / JSON / XML / Object

Wide-Column / Column-family

← NoSQL

Array (Vector, Matrix, Tensor)

Hierarchical

Network

Semantic

Entity-Relationship

DATA MODELS

Relational

Key/Value

Graph

Document / JSON / XML / Object

Wide-Column / Column-family

Array (Vector, Matrix, Tensor) ← ML / Science

Hierarchical

Network

Semantic

Entity-Relationship

DATA MODELS

Relational
Key/Value
Graph
Document / JSON / XML / Object
Wide-Column / Column-family
Array (Vector, Matrix, Tensor)
Hierarchical
Network
Semantic
Entity-Relationship

← Obsolete / Legacy / Rare

DATA MODELS

Relational ← This Course

Key/Value

Graph

Document / JSON / XML / Object

Wide-Column / Column-family

Array (Vector, Matrix, Tensor)

Hierarchical

Network

Semantic

Entity-Relationship

EARLY DATABASE SYSTEMS

In the late 1960s, early DBMSs required developers to write queries using procedural code.

→ Examples: IDS, IMS, CODASYL

The developer had to choose access paths and execution ordering based on the current database contents.

→ If the database changes, then the developer must rewrite the query code.

1973 ACM Turing Award Lecture

The Turing Award citation read by Richard G. Canning, chairman of the 1973 Turing Award Committee, at the presentation of this lecture on August 28 at the ACM Annual Conference in Atlanta:

A significant change in the computer field in the last five to eight years has been made in the way we treat and handle data. In the early days of our field, data was intimately tied to the application programs that used it. Now we see that we want to break that tie. We want data that is independent of the application programs that use it—that is, data that is organized and structured to serve many applications and many users. What we seek is the *data base*.

This movement toward the data base is in its infancy. Even so, it appears that there are now between 1,000 and 2,000 true data base management systems installed worldwide. In ten years very likely, there will be tens of thousands of such systems. Just from the quantities of installed systems, the impact of data bases promises to be huge.

This year's recipient of the A.M. Turing Award is one of the real pioneers of data base technology. No other individual has had the influence that he has had upon this aspect of our field. I single out three prime examples of what he has done. He was the creator and principal architect of the first commercially available data base management system—the Integrated Data Store—originally developed from 1961 to 1964.[1,2,3,4] I-D-S is today one of the three most widely used data base management systems. Also, he was one of the founding members of the CODASYL Data Base Task Group, and served on that task group from 1966 to 1968. The specifications of that task group are being implemented by many suppliers in various parts of the world.[5,6] Indeed, currently these specifications represent the only proposal of stature for a common architecture for data base management systems. It is to his credit that these specifications, after extended debate and discussion, embody much of the original thinking of the Integrated Data Store. Thirdly, he was the creator of a powerful method for displaying data relationships—a tool for data base designers as well as application system designers.[7,8]

His contributions have thus represented the union of imagination and practicality. The richness of his work has already had, and will continue to have, a substantial influence upon our field.

I am very pleased to present the 1973 A.M. Turing Award to Charles W. Bachman.

The Programmer as Navigator

by Charles W. Bachman

This year the whole world celebrates the five-hundredth birthday of Nicolaus Copernicus, the famous Polish astronomer and mathematician. In 1543, Copernicus published his book, *Concerning the Revolutions of Celestial Spheres*, which described a new theory about the relative physical movements of the earth, the planets, and the sun. It was in direct contradiction with the earth-centered theories which had been established by Ptolemy 1400 years earlier.

Copernicus proposed the heliocentric theory, that planets revolve in a circular orbit around the sun. This theory was subjected to tremendous and persistent criticism. Nearly 100 years later, Galileo was ordered to appear before the Inquisition in Rome and forced to state that he had given up his belief in the Copernican theory. Even this did not placate his inquisitors, and he was sentenced to an indefinite prison term, while Copernicus's book was placed upon the Index of Prohibited Books, where it remained for another 200 years.

I raise the example of Copernicus today to illustrate a parallel that I believe exists in the computing or, more properly, the information systems world. We have spent the last 50 years with almost Ptolemaic information systems. These systems, and most of the thinking about systems, were based on a "computer centered" concept. (I choose to speak of 50 years of history rather than 25, for I see today's information systems as dating from the beginning of effective punched card equipment rather than from the beginning of the stored program computer.)

Just as the ancients viewed the earth with the sun revolving around it, so have the ancients of our information systems viewed a tab machine or computer with a sequential file flowing through it. Each was an

Author's address: Honeywell Information Systems, Inc., 200 Smith Street, Waltham, MA 02154.

The abstract, key words, etc., are on page 654.
[1-8] Footnotes are on page 658.

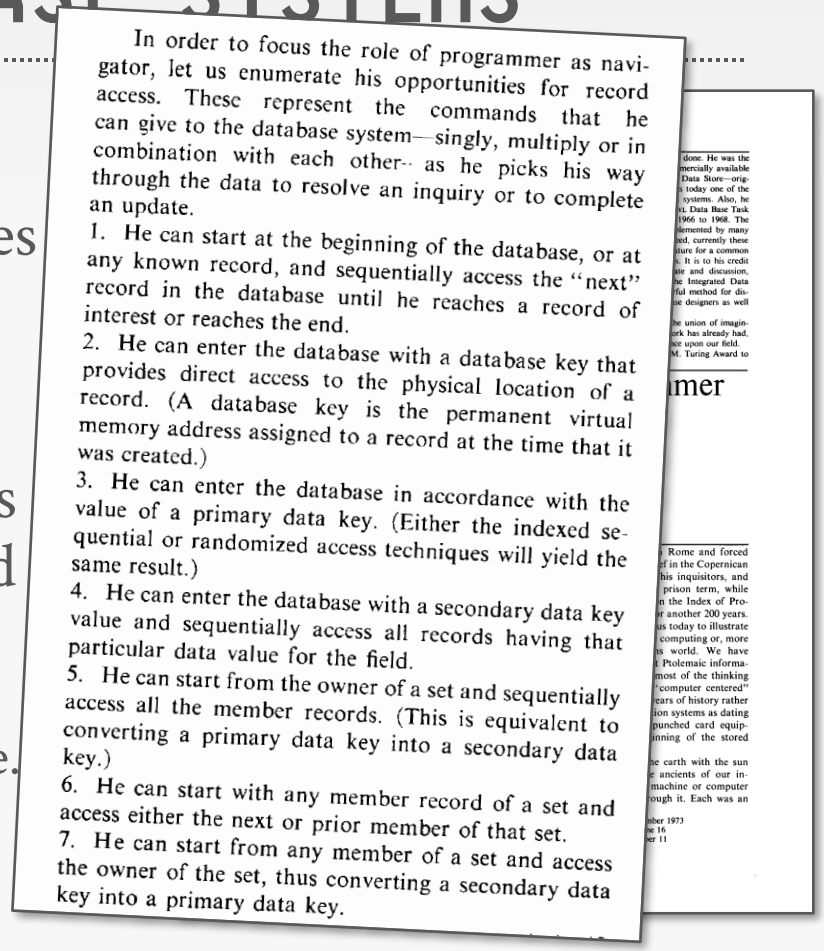
EARLY DATABASE SYSTEMS

In the late 1960s, early DBMSs required developers to write queries using procedural code.

→ Examples: IDS, IMS, CODASYL

The developer had to choose access paths and execution ordering based on the current database contents.

→ If the database changes, then the developer must rewrite the query code.



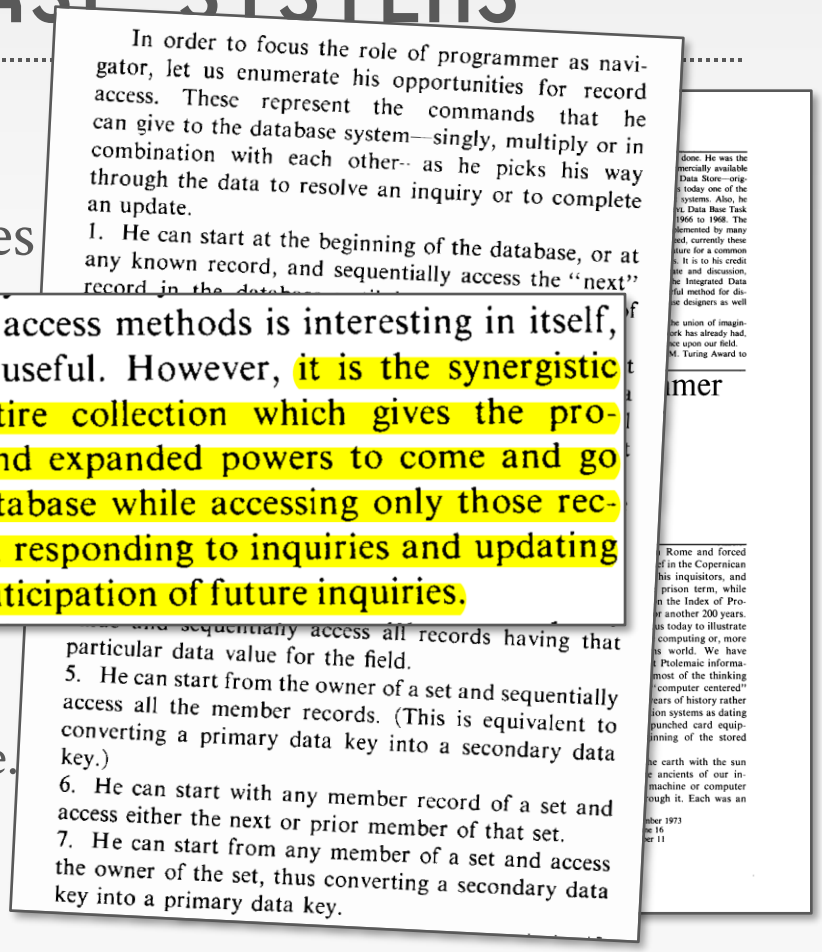
In order to focus the role of programmer as navigator, let us enumerate his opportunities for record access. These represent the commands that he can give to the database system—singly, multiply or in combination with each other-- as he picks his way through the data to resolve an inquiry or to complete an update.

1. He can start at the beginning of the database, or at any known record, and sequentially access the "next" record in the database until he reaches a record of interest or reaches the end.
2. He can enter the database with a database key that provides direct access to the physical location of a record. (A database key is the permanent virtual memory address assigned to a record at the time that it was created.)
3. He can enter the database in accordance with the value of a primary data key. (Either the indexed sequential or randomized access techniques will yield the same result.)
4. He can enter the database with a secondary data key value and sequentially access all records having that particular data value for the field.
5. He can start from the owner of a set and sequentially access all the member records. (This is equivalent to converting a primary data key into a secondary data key.)
6. He can start with any member record of a set and access either the next or prior member of that set.
7. He can start from any member of a set and access the owner of the set, thus converting a secondary data key into a primary data key.

EARLY DATABASE SYSTEMS

In the late 1960s, early DBMSs required developers to write queries using procedural cod
→ Examples: IDS, IMS, C

The developer had to
paths and execution
on the current database contents.
→ If the database changes, then the developer must rewrite the query code.



In order to focus the role of programmer as navigator, let us enumerate his opportunities for record access. These represent the commands that he can give to the database system—singly, multiply or in combination with each other-- as he picks his way through the data to resolve an inquiry or to complete an update.

1. He can start at the beginning of the database, or at any known record, and sequentially access the "next" record in the

Each of these access methods is interesting in itself, and all are very useful. However, it is the synergistic usage of the entire collection which gives the programmer great and expanded powers to come and go within a large database while accessing only those records of interest in responding to inquiries and updating the database in anticipation of future inquiries.

particular data value for the field.

5. He can start from the owner of a set and sequentially access all the member records. (This is equivalent to converting a primary data key into a secondary data key.)
6. He can start with any member record of a set and access either the next or prior member of that set.
7. He can start from any member of a set and access the owner of the set, thus converting a secondary data key into a primary data key.

EARLY DATABASE SYSTEMS

In the late 1960s, early DBMSs required developers to write queries using procedural code.

→ Examples: IDS, IMS, CODASYL

The developer had to choose access paths and execution ordering based on the current database contents.

→ If the database changes, then the developer must rewrite the query code.

Retrieve the names of artists that appear on the DJ Mooshoo Tribute mixtape.

```
PROCEDURE GET_ARTISTS_FOR_ALBUM;
BEGIN
  /* Declare variables */
  DECLARE ARTIST_RECORD ARTIST;
  DECLARE APPEARS_RECORD APPEARS;
  DECLARE ALBUM_RECORD ALBUM;

  /* Start navigation */
  FIND ALBUM USING ALBUM.NAME = "Mooshoo Tribute"
      ON ERROR DISPLAY "Album not found" AND EXIT;

  /* For each appearance on the album */
  FIND FIRST APPEARS WITHIN APPEARS_ALBUM OF ALBUM_RECORD
      ON ERROR DISPLAY "No artists found for this album" AND EXIT;

  /* Loop through the set of APPEARS */
  REPEAT
      /* Navigate to the corresponding artist */
      FIND OWNER WITHIN ARTIST_APPEARS OF APPEARS_RECORD
          ON ERROR DISPLAY "Error finding artist";
      /* Display artist name */
      DISPLAY ARTIST_RECORD.NAME;
      /* Move to the next APPEARS record in the set */
      FIND NEXT APPEARS WITHIN APPEARS_ALBUM OF ALBUM_RECORD
          ON ERROR EXIT;
  END REPEAT;
END PROCEDURE;
```

EARLY DATABASE SYSTEMS

In the late 1960s, early DBMSs required developers to write queries using procedural code.

→ Examples: IDS, IMS, CODASYL

The developer had to choose access paths and execution ordering based on the current database contents.

→ If the database changes, then the developer must rewrite the query code.

Retrieve the names of artists that appear on the DJ Mooshoo Tribute mixtape.

```
PROCEDURE GET_ARTISTS_FOR_ALBUM;
BEGIN
  /* Declare variables */
  DECLARE ST_RECORD ARTIST;
  DECLA S_RECORD APPEARS;
  DECLA ORD ALBUM;

  /* Start n gation
  FIND ALBUM USING "
      ON ERROR DI

  /* For each a
  FIND FIRST A WITHIN A BUM OF AL ECORD
      ON ERROR LAY "No arti ound for thi um" AND EXIT;

  /* Loop thro he set of A */
  REPEAT
      /* Navigat ond
      FIND OWNER _APPEAR _RECORD
          ON ERROR "
      /* Dis artis
      DISP ST_RECO
      / e next in the
      F APPEARS WI N A ARS BUM OF AL D
          RROR EXIT;
  END REPEAT;
END PROCEDURE;
```

EARLY DATABASE SYSTEMS

In the late 1960s, early DBMSs required developers to write queries using procedural code.

→ Examples: IDS, IMS, CODASYL

The developer had to choose access paths and execution ordering based on the current database contents.

→ If the database changes, then the developer must rewrite the query code.

Retrieve the names of artists that appear on the DJ Mooshoo Tribute mixtape.

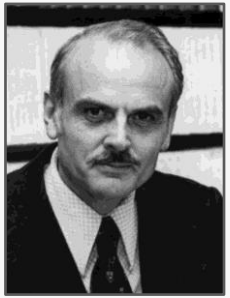
```
SELECT ARTIST.NAME
  FROM ARTIST, APPEARS, ALBUM
 WHERE ARTIST.ID=APPEARS.ARTIST_ID
   AND APPEARS.ALBUM_ID=ALBUM.ID
   AND ALBUM.NAME="Mooshoo Tribute"
```

EARLY DATABASE SYSTEMS

The Differences and Similarities Between the Data Base Set and Relational Views of Data.

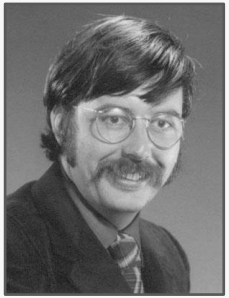
→ ACM SIGFIDET Workshop on Data Description, Access, and Control in Ann Arbor, Michigan (May 1974)

Bachman

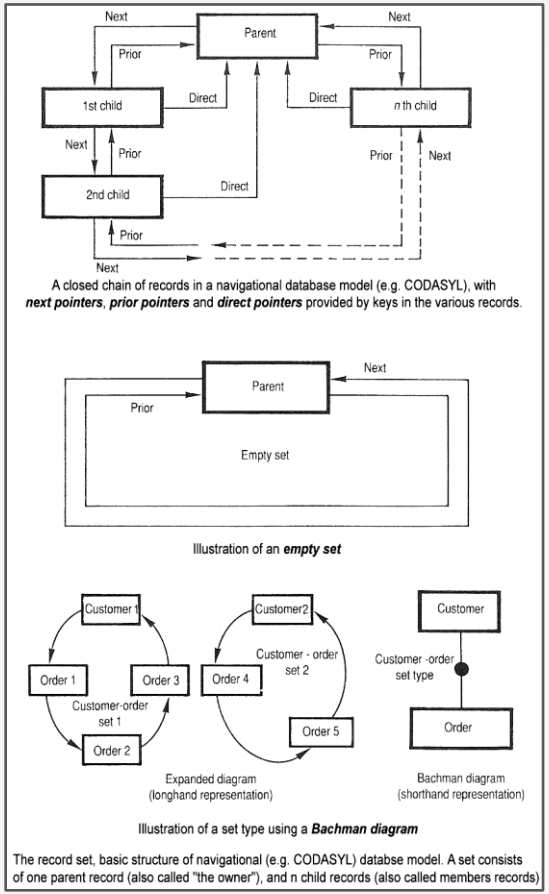


Codd

Gray



Stonebraker



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***

Expanded diagram (longhand representation)

Bachman diagram (shorthand representation)

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

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

EARLY DATABASE SYSTEMS

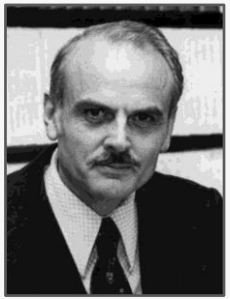
The Differences and Similarities Between the Data Base Set and Relational Views of Data.

→ ACM SIGFIDET Workshop on Data Description, Access, and Control in Ann Arbor, Michigan (May 1974)

COBOL/CODASYL camp:

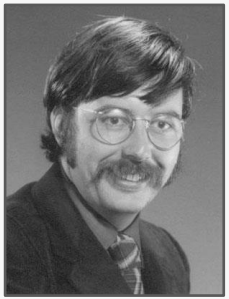
1. The relational model is too mathematical. No mere mortal programmer will be able to understand your newfangled languages.
2. Even if you can get programmers to learn your new languages, you won't be able to build an efficient implementation of them.
3. On-line transaction processing applications want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

Bachman



Codd

Gray



Stonebraker

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.

RELATIONAL MODEL

Structure: The definition of the database's relations and their contents are independent of their physical representation.

Integrity: Ensure the database's contents satisfy constraints.

Manipulation: Declarative API for accessing and modifying a database's contents via relations (sets).

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

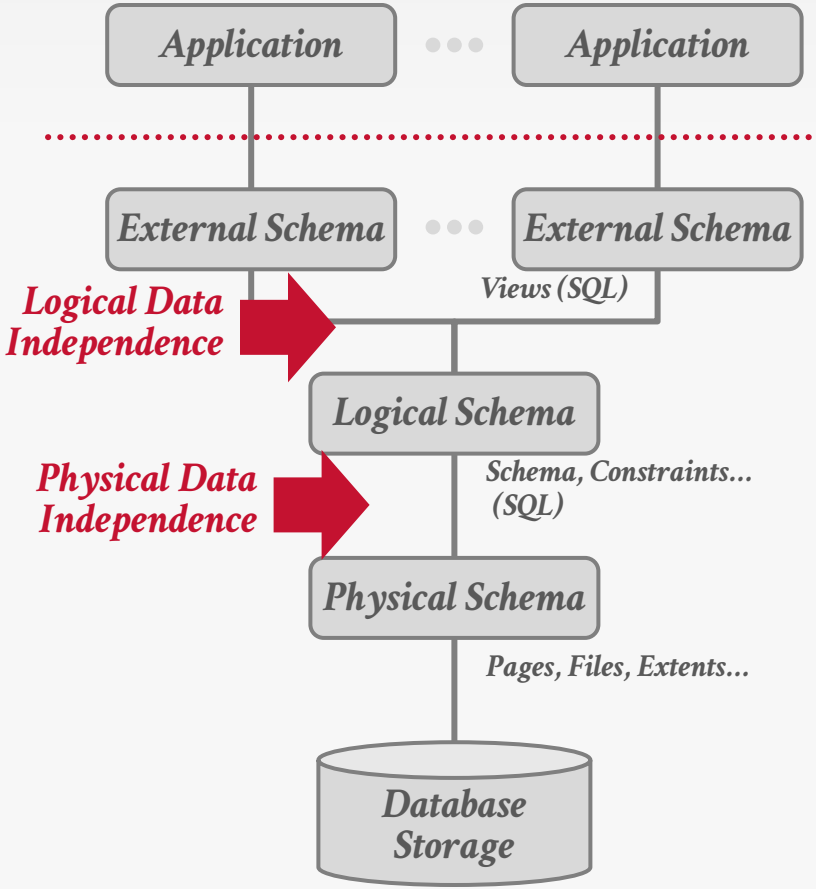
DATA INDEPENDENCE

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.

→ Re-optimize if/when these factors 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 (aka 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(name, year, country)

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

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: 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, artist, year)

id	name	artist	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

ArtistAlbum(artist_id, album_id)

artist_id	album_id
101	11
101	22
103	22
102	22

Album(id, name, artist, year)

id	name	artist	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: 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 supported (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)
);
```

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 supported (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> );
```

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

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 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 in a relation satisfying 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

RELATIONAL ALGEBRA: SELECT

Choose a subset of the tuples in a relation satisfying 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: SELECT

Choose a subset of the tuples in a relation satisfying 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 either only one or both input relations, eliminating duplicates.

→ Both relations must have the same attributes (based on names).

Syntax: (R U S)

```
(SELECT * FROM R)
UNION
(SELECT * FROM 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 U 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 input relations.

→ Both relations must have the same attributes (based on names).

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.

→ Both relations must have the same attributes (based on names).

Syntax: (R - S)

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

RELATIONAL ALGEBRA: PRODUCT

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

→ Input relations do not have to have the same attributes.

→ Output includes all the attributes 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

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

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

Syntax: (R ⋈ S)

(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.

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

Syntax: (R ⋈ S)

```
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 **S.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 ← This Course

Key/Value

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

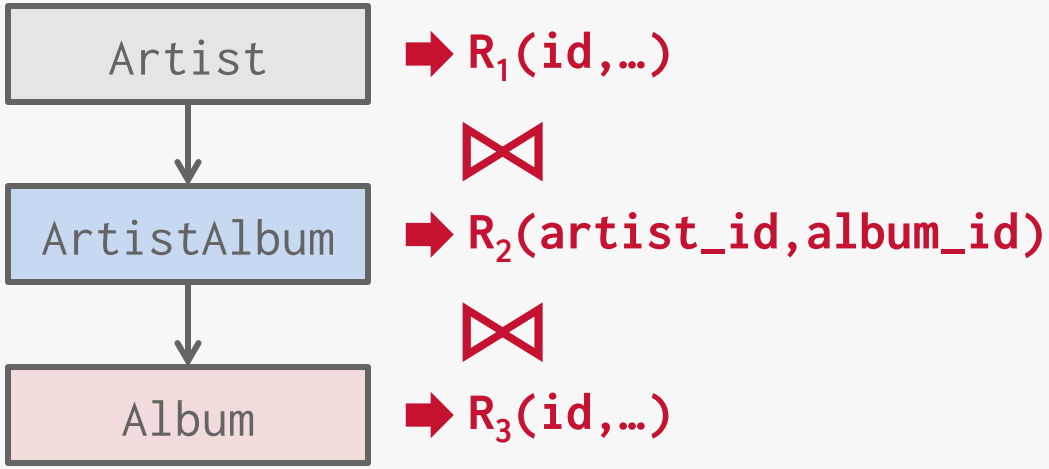
A collection of record documents containing a hierarchy of named field/value pairs.

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

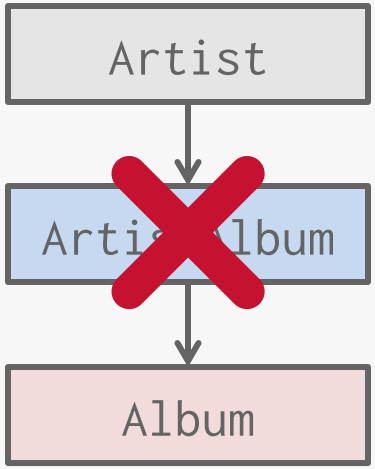
→ Modern implementations use JSON. Older systems use XML or custom object representations.

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

DOCUMENT DATA MODEL



DOCUMENT DATA MODEL



➡ R_1(id,…)

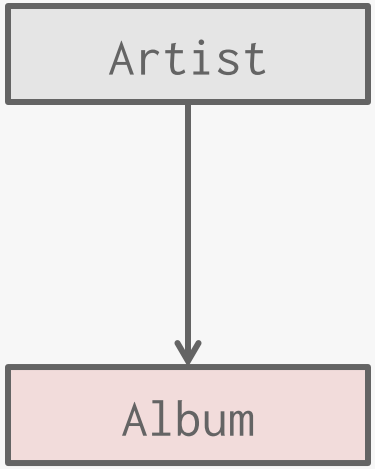
⋈

➡ R_2(artist_id,album_id)

⋈

➡ R_3(id,…)

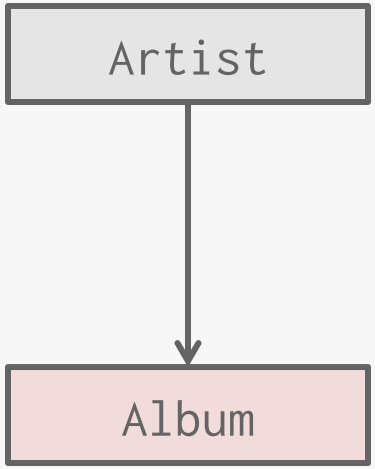
DOCUMENT DATA MODEL



Application Code

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

DOCUMENT DATA MODEL

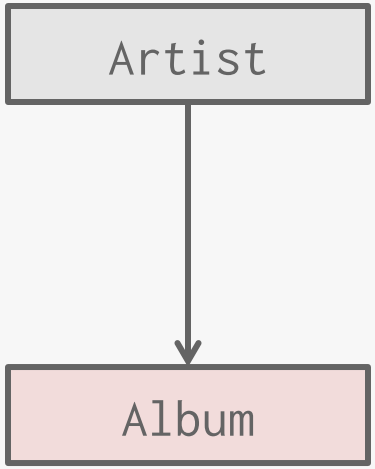


Application Code

```
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
  }
 ]
}
```

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 drant nucliadb marqo CloseVector VectorDB turbopuffer <(°O°)> BagelDB

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 drant nucliadb marqo CloseVector VectorDB turbopuffer <(°O°)> BagelDB

VECTOR DATA MODEL

Album(id, name, year, lyrics)

id	name	year	lyrics
Id1	Enter the Wu-Tang	1993	**<text>**
Id2	Run the Jewels 2	2015	**<text>**
Id3	Liquid Swords	1995	**<text>**
Id4	We Got It from Here	2016	**<text>**

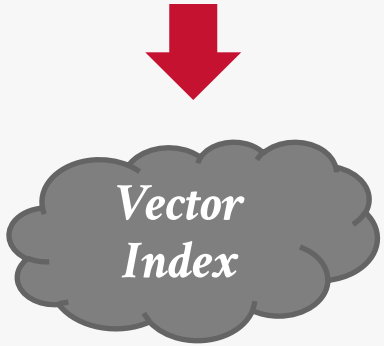
VECTOR DATA MODEL

Album(id, name, year, lyrics)

id	name	year	lyrics
Id1	Enter the Wu-Tang	1993	<text>
Id2	Run the Jewels 2	2015	<text>
Id3	Liquid Swords	1995	<text>
Id4	We Got It from Here	2016	<text>

Embeddings

```
Id1 → [0.32, 0.78, 0.30, ...]
Id2 → [0.99, 0.19, 0.81, ...]
Id3 → [0.01, 0.18, 0.85, ...]
Id4 → [0.19, 0.82, 0.24, ...]
        ⋮
```



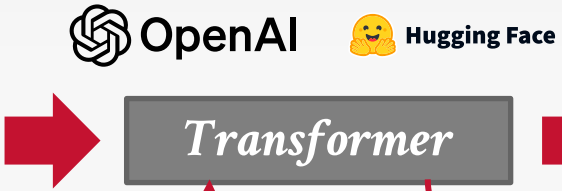
Query

Find albums with lyrics about **running from the police**

VECTOR DATA MODEL

Album(id, name, year, lyrics)

id	name	year	lyrics
Id1	Enter the Wu-Tang	1993	<text>
Id2	Run the Jewels 2	2015	<text>
Id3	Liquid Swords	1995	<text>
Id4	We Got It from Here	2016	<text>



Embeddings

```
Id1 → [0.32, 0.78, 0.30, ...]
Id2 → [0.99, 0.19, 0.81, ...]
Id3 → [0.01, 0.18, 0.85, ...]
Id4 → [0.19, 0.82, 0.24, ...]
⋮
```

Query

Find albums with lyrics about **running from the police**

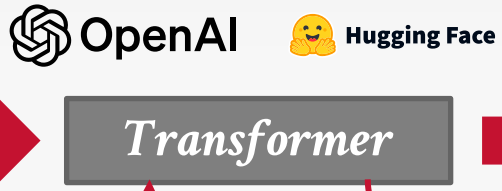
[0.02, 0.10, 0.24, ...]

Vector Index

VECTOR DATA MODEL

Album(id, name, year, lyrics)

id	name	year	lyrics
Id1	Enter the Wu-Tang	1993	<text>
Id2	Run the Jewels 2	2015	<text>
Id3	Liquid Swords	1995	<text>
Id4	We Got It from Here	2016	<text>



Embeddings

```
Id1 → [0.32, 0.78, 0.30, ...]
Id2 → [0.99, 0.19, 0.81, ...]
Id3 → [0.01, 0.18, 0.85, ...]
Id4 → [0.19, 0.82, 0.24, ...]
⋮
```

Query

Find albums with lyrics about **running from the police**

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

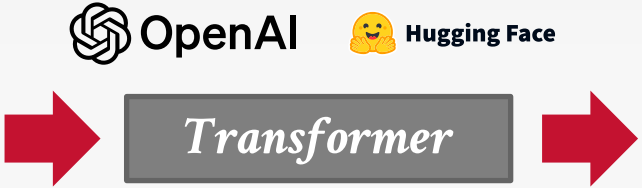
Vector Index

Ranked List of Ids

VECTOR DATA MODEL

Album(id, name, year, lyrics)

id	name	year	lyrics
Id1	Enter the Wu-Tang	1993	<text>
Id2	Run the Jewels 2	2015	<text>
Id3	Liquid Swords	1995	<text>
Id4	We Got It from Here	2016	<text>



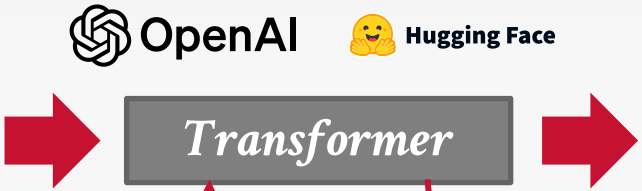
Embeddings

```
Id1 → [0.32, 0.78, 0.30, ...]
Id2 → [0.99, 0.19, 0.81, ...]
Id3 → [0.01, 0.18, 0.85, ...]
Id4 → [0.19, 0.82, 0.24, ...]
          ⋮
```

VECTOR DATA MODEL

Album(id, name, year, lyrics)

id	name	year	lyrics
Id1	Enter the Wu-Tang	1993	<text>
Id2	Run the Jewels 2	2015	<text>
Id3	Liquid Swords	1995	<text>
Id4	We Got It from Here	2016	<text>



Embeddings

```
Id1 → [0.32, 0.78, 0.30, ...]
Id2 → [0.99, 0.19, 0.81, ...]
Id3 → [0.01, 0.18, 0.85, ...]
Id4 → [0.19, 0.82, 0.24, ...]
⋮
```

Query

Find albums with lyrics about **running from the police** and released after **2005**

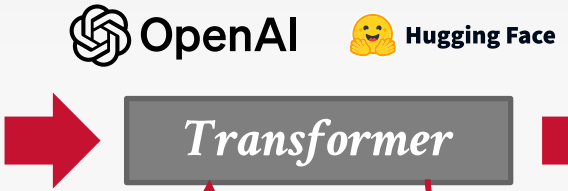
[0.02, 0.10, 0.24, ...]

Vector Index

VECTOR DATA MODEL

Album(id, name, year, lyrics)

id	name	year	lyrics
Id1	Enter the Wu-Tang	1993	<text>
Id2	Run the Jewels 2	2015	<text>
Id3	Liquid Swords	1995	<text>
Id4	We Got It from Here	2016	<text>



Embeddings

```
Id1 → [0.32, 0.78, 0.30, ...]
Id2 → [0.99, 0.19, 0.81, ...]
Id3 → [0.01, 0.18, 0.85, ...]
Id4 → [0.19, 0.82, 0.24, ...]
⋮
```

Query

Find albums with lyrics about **running from the police** and released after **2005**

[0.02, 0.10, 0.24, ...]

year > 2005

Vector Index

CONCLUSION

Databases are the most important and beautiful software in all of computer science.

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.