Carnegie Mellon University

Relational Algebra

Lecture #02

Database Systems
15-445/15-645
Fall 2017

Andy Pavlo
Computer Science Dept.
Carnegie Mellon Univ.

ADMINISTRIVIA

HW1 is due Wed Sep 13th.

We will release the first project on Mon Sep 11th.
It is due Wed Sep 27th.

TODAY'S AGENDA

More Relational Model

Relational Algebra

Relational Calculus

Alternative Data Models

BEFORE THE RELATIONAL MODEL

Database applications were difficult to build and maintain.

Tight coupling between logical and physical layers.

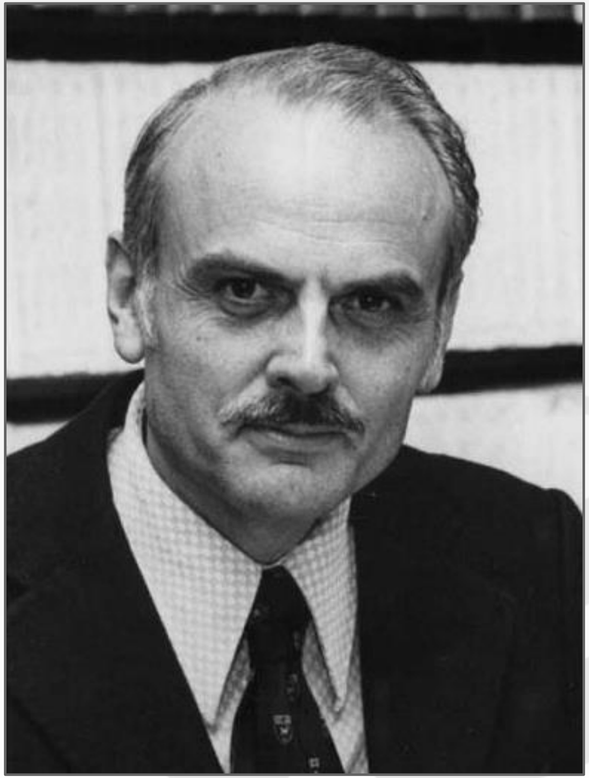
You have to (roughly) know what queries your app would execute before you deployed the database.

BEFORE THE RELATIONAL MODEL

Database applications were difficult to build and maintain.

Tight coupling between logical and physical layers.

You have to (roughly) know what queries your app would execute before you deployed the database.



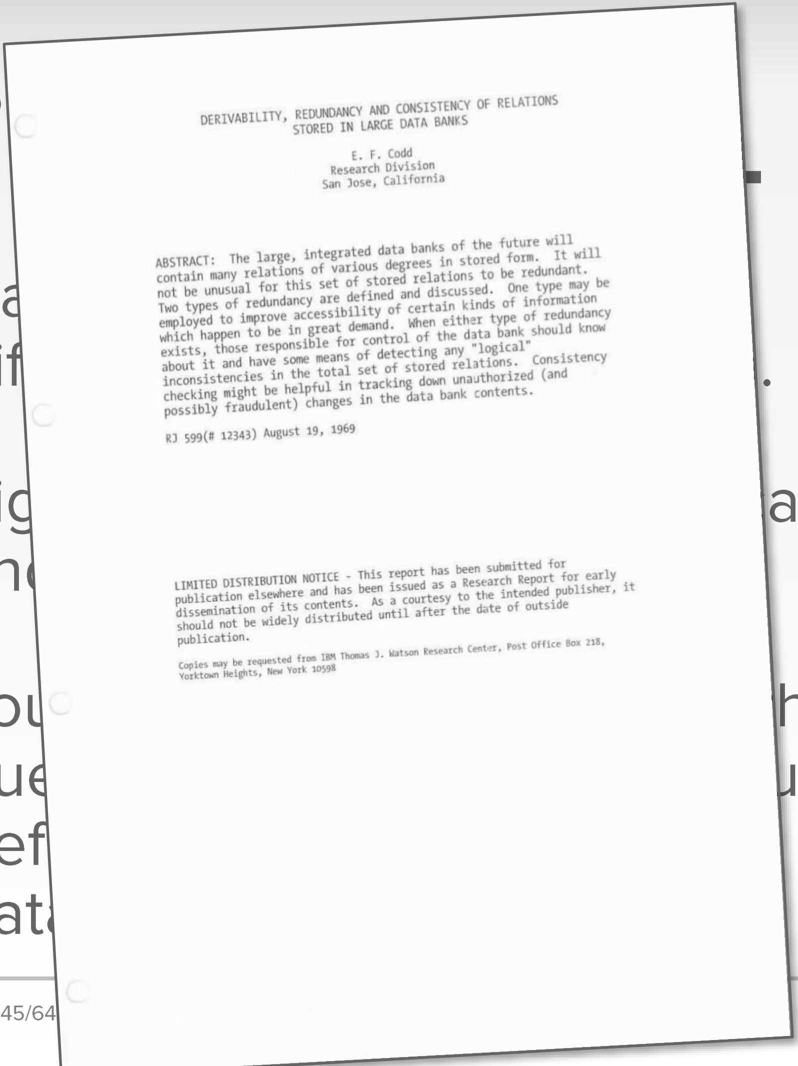
Edgar F. Codd

B
R

Da
dif .

Tig al
an

You hat
que ute
bef
data



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

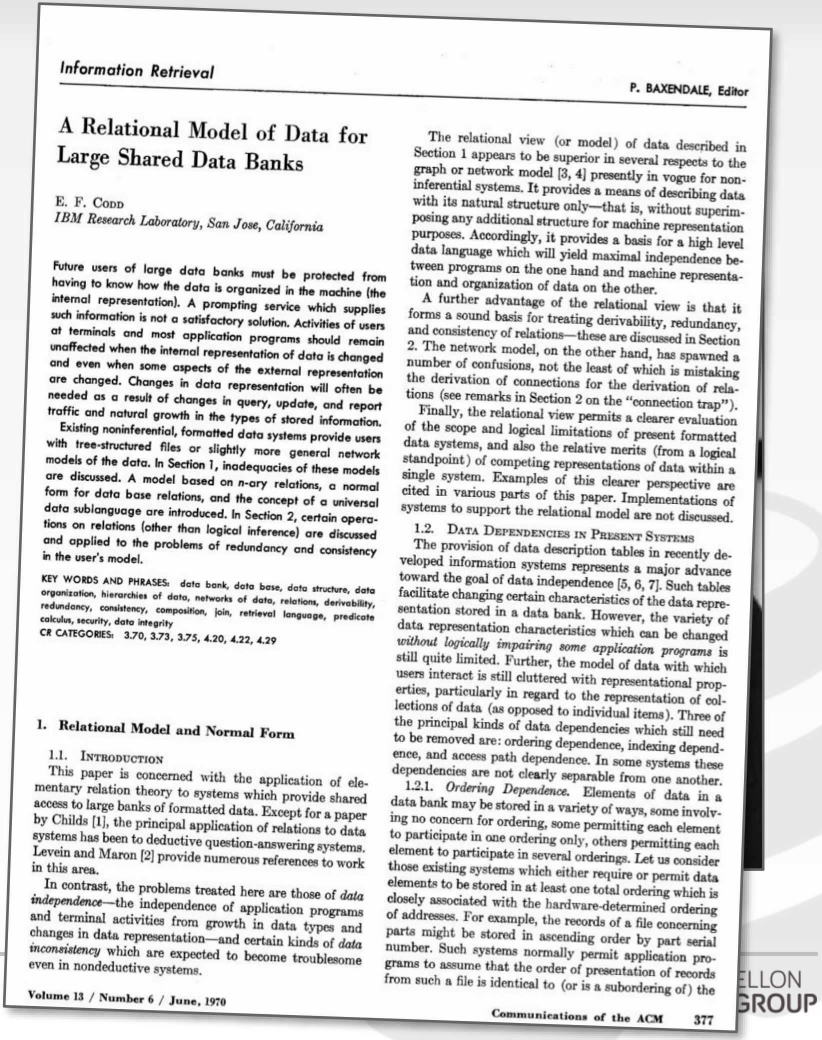
E. F. Codd
Research Division
San Jose, California

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



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.

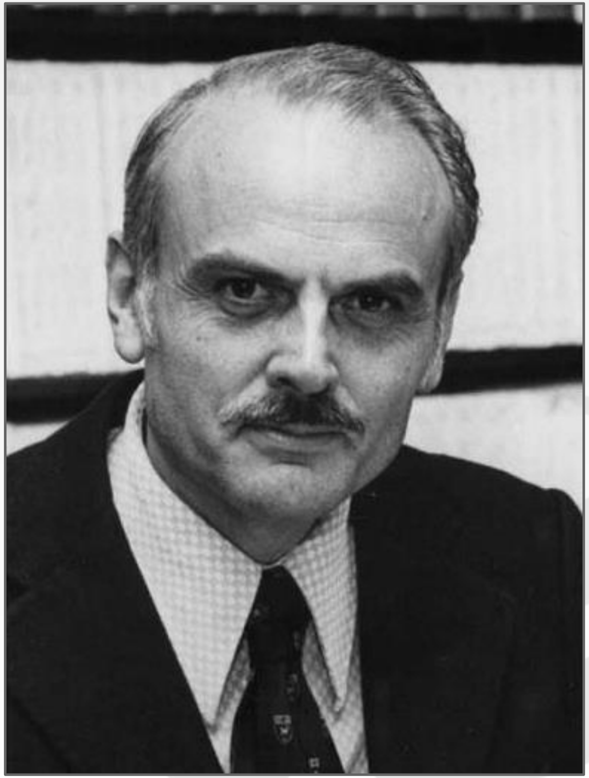
1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing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

RELATIONAL MODEL

Proposed in 1970 by Ted Codd.

Database abstraction to avoid this maintenance:

→ Store database in simple data structures.
→ Access data through high-level language.
→ Physical storage left up to implementation.



Edgar F. Codd

DATA MODELS

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

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

DATA MODELS

Relational

Key/Value

Graph

Document

Column-family

Array / Matrix

Hierarchical

Network

RELATIONAL MODEL

Structure: The definition of relations and their contents.

Integrity: Ensure the database's contents satisfy constraints.

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

RELATIONAL MODEL

A relation is 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.

RELATIONAL MODEL

A relation is unordered set that contain the relationship of attributes that represent entities.

A tuple is a set of attribute values (also known as its domain) in the relation.

- → Values are (normally) atomic/scalar.
- → The special value NULL is a member of every domain.

`Artist(name, year, country)`

name	year	country
Wu Tang Clan	1992	USA
Notorious B.I.G.	1992	USA
Ice Cube	1989	USA

***n*-ary Relation**

=

Table with *n* columns

RELATIONAL MODEL: CANDIDATE KEYS

Every relation has at least one candidate key that uniquely identifies a single tuple.

Candidate keys:

→ (name)
→ (name, year)
→ (name, country)
→ (name, year, country)

`Artist(name, year, country)`

name	year	country
Wu Tang Clan	1992	USA
Notorious B.I.G.	1992	USA
Ice Cube	1989	USA

RELATIONAL MODEL: CANDIDATE KEYS

Every relation has at least one candidate key that uniquely identifies a single tuple.

Candidate keys:

→ ~~(name)~~
→ (name, year)
→ (name, country)
→ (name, year, country)

`Artist(name, year, country)`

name	year	country
Wu Tang Clan	1992	USA
Notorious B.I.G.	1992	USA
Ice Cube	1989	USA
Ice Cube	2017	India

RELATIONAL MODEL: CANDIDATE KEYS

Every relation has at least one candidate key that uniquely identifies a single tuple.

Candidate keys:

→ ~~(name)~~
→ ~~(name, year)~~
→ (name, country)
→ (name, year, country)

Artist(name, year, country)

name	year	country
Wu Tang Clan	1992	USA
Notorious B.I.G.	1992	USA
Ice Cube	1989	USA
Ice Cube	2017	India
Ice Cube	2017	USA

RELATIONAL MODEL: CANDIDATE KEYS

Every relation has at least one candidate key that uniquely identifies a single tuple.

Candidate keys:

→ ~~(name)~~
→ ~~(name, year)~~
→ ~~(name, country)~~
→ ~~(name, year, country)~~

`Artist(name, year, country)`

name	year	country
Wu Tang Clan	1992	USA
Notorious B.I.G.	1992	USA
Ice Cube	1989	USA
Ice Cube	2017	India
Ice Cube	2017	USA
Ice Cube	2017	USA

RELATIONAL MODEL: CANDIDATE KEYS

Every relation has at least one candidate key that uniquely identifies a single tuple.

Candidate keys:

→ (id, name, year, country)

→ (id)

→ (id, name)

→ (id, year)

→ (id, country)

→ *All other combinations…*

Artist(id, name, year, country)

id	name	year	country
101	Wu Tang Clan	1992	USA
102	Notorious B.I.G.	1992	USA
103	Ice Cube	1989	USA
104	Ice Cube	2017	India
105	Ice Cube	2017	USA
106	Ice Cube	2017	USA

RELATIONAL MODEL: PRIMARY KEY

A relation's primary key is a candidate key that is deemed more "important" than other candidate keys.

Some DBMSs automatically create an internal primary key for a table if you do not define one.

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

id	name	year	country
101	Wu Tang Clan	1992	USA
102	Notorious B.I.G.	1992	USA
103	Ice Cube	1989	USA
104	Ice Cube	2017	India
105	Ice Cube	2017	USA
106	Ice Cube	2017	USA

RELATIONAL MODEL: FOREIGN KEYS

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

Artist(id, name, year, country)

id	name	year	country
101	Wu Tang Clan	1992	USA
102	Notorious B.I.G.	1992	USA
103	Ice Cube	1989	USA

Album(id, name, artists, year)

id	name	artists	year
11	Enter the Wu Tang	101	1993
22	St.Ides Mix Tape	???	1994

RELATIONAL MODEL: FOREIGN KEYS

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

Artist(id, name, year, country)

id	name	year	country
101	Wu Tang Clan	1992	USA
102	Notorious B.I.G.	1992	USA
103	Ice Cube	1989	USA

ArtistAlbum(artist_id, album_id)

artist_id	album_id
101	11
101	22
103	22

Album(id, name, year)

id	name	year
11	Enter the Wu Tang	1993
22	St.Ides Mix Tape	1994

DATA MANIPULATION LANGUAGES (DML)

How to store and retrieve information from a database.

Procedural:

→ The query specifies the (high-level) strategy the DBMS should use to find the desired result.

← **Relational Algebra**

Non-Procedural:

→ 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.

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.

→ Can rearrange attributes' ordering.

→ Can manipulate the values.

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 of the input relations.

Syntax: **(R U 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 U S)

a_id	b_id
a1	101
a2	102
a3	103
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)

a_id	b_id
a3	103
a4	104
a5	105

(R ⋈ S)

a_id	b_id
a3	103

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

RELATIONAL ALGEBRA: JOIN TYPES

Cross Join

→ Same thing as Cartesian product

Inner Join

→ Each tuple in the first relation must have a corresponding match in the second relation.

Outer Join

→ Each tuple in one relation does not need to have a corresponding match in the other relation.

RELATIONAL ALGEBRA: INNER JOINS (1)

Natural Join **($R \bowtie S$)**

→ Match tuples in **R** and **S** where the shared attributes are equivalent.

Theta / Equi Join **($R \bowtie_{\theta} S$)**

→ Match tuples using some arbitrary join predicate defined by **θ**.

→ If **θ** is just an equality predicate, then it is called an "Equi Join".

RELATIONAL ALGEBRA: INNER JOINS (1)

Natural Join $(R \bowtie S)$

→ Match tuples in **R** and **S** where the shared attributes are equivalent.

Theta / Equi Join $(R \bowtie_\theta S)$

→ Match tuples using some arbitrary join predicate defined by θ.

→ If θ is just an equality predicate, then it is called an "Equi Join".

$(R \bowtie S)$

a_id	b_id
a3	103

$(R \bowtie_{(R.a\_id=S.a\_id \land R.b\_id=S.b\_id)} S)$

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

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

RELATIONAL ALGEBRA: INNER JOINS (1)

Natural Join $(R \bowtie S)$

→ Match tuples in **R** and **S** where the shared attributes are equivalent.

$(R \bowtie S)$

a_id	b_id
a3	103

Theta / Equi Join $(R \bowtie_{\theta} S)$

→ Match tuples using some arbitrary join predicate defined by **θ**.

→ If **θ** is just an equality predicate, then it is called an "Equi Join".

$(R \bowtie_{(R.a\_id=S.a\_id \land R.b\_id=S.b\_id)} S)$

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

```
SELECT * FROM R INNER JOIN S
```

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

RELATIONAL ALGEBRA: INNER JOINS (1)

Natural Join $(R \bowtie S)$

→ Match tuples in **R** and **S** where the shared attributes are equivalent.

Theta / Equi Join $(R \bowtie_{\theta} S)$

→ Match tuples using some arbitrary join predicate defined by **θ**.

→ If **θ** is just an equality predicate, then it is called an "Equi Join".

$(R \bowtie S)$

a_id	b_id
a3	103

$(R \bowtie_{(R.a\_id=S.a\_id \land R.b\_id=S.b\_id)} S)$

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

```
SELECT * FROM R INNER JOIN S
```

```
SELECT * FROM R JOIN S
```

```
SELECT * FROM R, S
 WHERE R.a_id = S.a_id
   AND R.b_id = S.b_id;
```

RELATIONAL ALGEBRA: INNER JOINS (2)

Semi Join $(R \ltimes S)$

→ Generate relation that contains tuples of **R** that match with tuples in **S**.

→ Same as natural join except that output relation only contains tuples from the first relation.

Anti Join $(R \triangleright S)$

→ Generate relation that contains tuples of **R** that do not match with any tuple in **S**.

$(R \ltimes S)$

R.a_id	R.b_id
a3	103

```
SELECT R.* FROM R
 WHERE EXISTS(
   SELECT * FROM S
    WHERE R.a_id = S.a_id
      AND R.b_id = S.b_id);
```

$(R \triangleright S)$

R.a_id	R.b_id
a1	101
a2	102

```
SELECT R.* FROM R
 WHERE NOT EXISTS(
   SELECT * FROM S
    WHERE R.a_id = S.a_id
      AND R.b_id = S.b_id);
```

RELATIONAL ALGEBRA: OUTER JOINS

Left Outer Join (R⟕S)

→ Generate all combinations of tuples in **R** and **S** that are equal on their shared attributes, in addition to tuples in **R** that have no matching tuples in **S**.

Right Outer Join (R⟖S)

→ Same as LEFT OUTER but with the input relations reversed.

Full Outer Join (R⟗S)

→ Union of LEFT OUTER and RIGHT OUTER.

(R ⟕ S)

R.a_id	R.b_id	S.a_id	S.b_id
a1	101	NULL	NULL
a2	102	NULL	NULL
a3	103	a3	103

```
SELECT *
  FROM R LEFT OUTER 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 still defines the high-level steps of how to compute a query.

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

A better approach is just state the high-level query you want

→ Retrieve the joined tuples from R and S where b_id equals 102".

RELATIONAL CALCULUS

Tuple Relational Calculus

→ Uses variables whose permitted values are tuples of that relation.

Domain Relational Calculus

→ Uses domain variables to select attributes instead of whole tuples.

Both are equivalent to relational algebra.

Relational Algebra

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

Tuple Relational Calculus

$$\{\texttt{t} \mid \texttt{t} \in \texttt{R} \land \texttt{t.a\_id='a2'} \land \texttt{t.b\_id>102}\}$$

Domain Relational Calculus

$$\begin{aligned}\{\texttt{<a\_id,b\_id>} \mid \texttt{<a\_id,b\_id>} \in \texttt{R} \\ \land\ \texttt{a\_id='a2'} \\ \land\ \texttt{b\_id>102}\}\end{aligned}$$

DATA MODELS

Relational

Key/Value

Graph

Document

Column-family

← **NoSQL**

Array / Matrix

Hierarchical

Network

NON-RELATIONAL DATA MODELS: KEY-VALUE

Store records in an **associative array** that maps a key to a value.

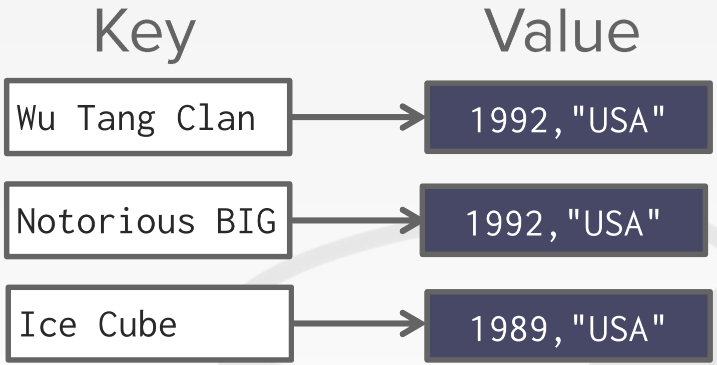
→ Sometimes called a **dictionary** or **hash table**.

The contents of a record's value is often opaque to DBMS.

→ It is up to the application to interpret its contents.

→ Some systems allow for compound values.

`Artist: name→(year,country)`



Key | Value

Wu Tang Clan → 1992,"USA"

Notorious BIG → 1992,"USA"

Ice Cube → 1989,"USA"

RocksDB · amazon DynamoDB · redis · LMDB · AEROSPIKE · ORACLE BERKELEY DB

NON-RELATIONAL DATA MODELS: GRAPH

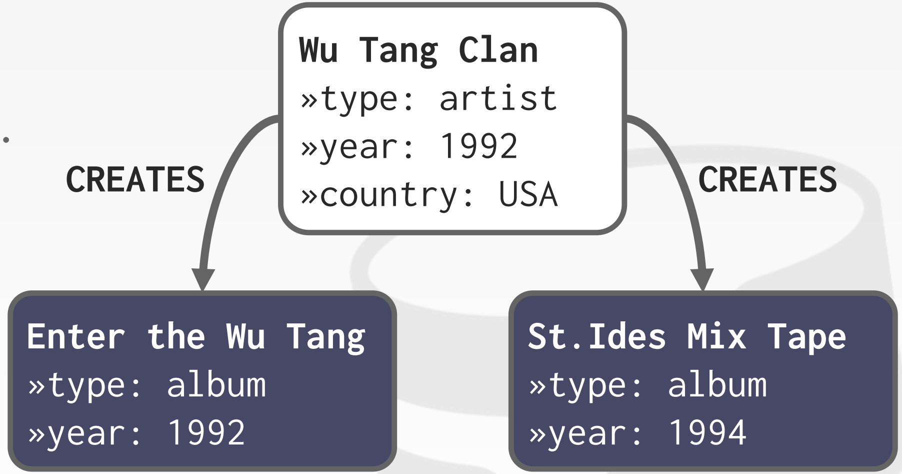
Represent the database as a collection of nodes and edges.

→ Each node/edge can also be annotated with additional properties.

No standard query language:

→ SPARQL, Gremlin, Cypher

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



neo4j INFINITEGRAPH Stardog

NON-RELATIONAL DATA MODELS: DOCUMENT

A document is a self-contained record that contains the description of its data attributes and their corresponding values.

```
{
 "name":    "Wu Tang Clan",
 "year":    1992,
 "country": "USA"
}
```

```
{
 "name":    "Ice Cube",
 "year":    1989,
 "country": "USA",
 "city":    "Los Angeles"
}
```

NON-RELATIONAL DATA MODELS: DOCUMENT

A document is a self-contained record that contains the description of its data attributes and their corresponding values.

Support nesting together all of the documents for a single entity.

→ This is called **denormalization** in the relational model.

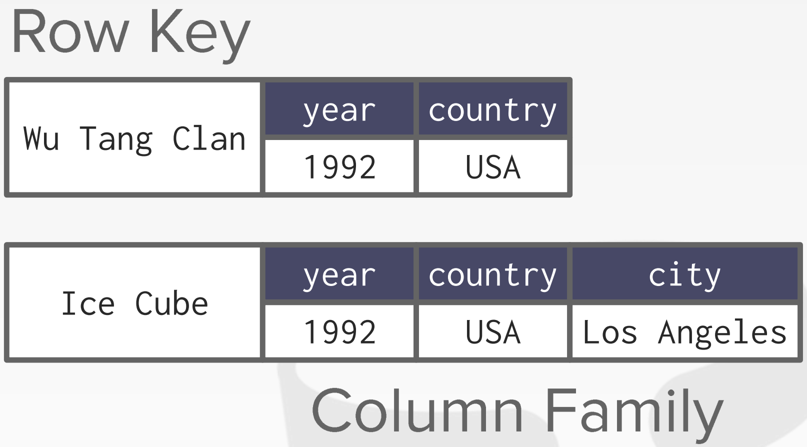
```
{
  "name":     "Wu Tang Clan",
  "year":     1992,
  "country": "USA",
  "albums": [
    { "name": "Enter the Wu Tang",
      "year": 1993,
      "tracks": [
        "Bring da Ruckus",
        "Shame on a *****",
        "Clan in da Front",
        ⋮
      ] }
  ]
}
```

MarkLogic CrateDB mongoDB CouchDB RAVENDB Couchbase

NON-RELATIONAL DATA MODELS: COLUMN-FAMILY

Hybrid data model that maps a key to a column family.

Each column family contains any number of rows that each has one or more column names and values.



Row Key

Wu Tang Clan	year	country
	1992	USA

Ice Cube	year	country	city
	1992	USA	Los Angeles

Column Family

CONCLUSION

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

Aggregations + Group By

Output Control / Redirection

Nested Queries

Common Table Expressions

Window Functions