

Tools & Models for Data Science

Introduction to Relational Databases

Chris Jermaine & Risa Myers

Rice University



What is a Database?

- A collection of data
- Plus, a set of programs for managing that data

Back in the Day...

- The dominant data model was the network or navigational model (60's and 70's)
- Data were a set of records with pointers between them
- Much DB code was written in COBOL
- Big problem was lack of physical data independence
 - Code was written for specific storage model
 - Want to change storage? Modify your code
 - Want to index your data? Modify your code
 - Led to very little flexibility
 - Your code locked you into a physical database design!

Some People Realized This Was a Problem

- By 1970, EF Codd (IBM) was looking at the so-called relational model
 - Landmark 1970 paper, “A relational model of data for large shared data banks”
 - Led to the 1981 Turing Award
 - Highest honor a computer scientist receives
 - Analogous to a Nobel Prize
- Idea: data stored in “relations”
 - A relation is a table of tuples or records
 - Attributes of a tuple have no sub-structure (are atomic)
- No pointers!

Databases are Still Important

- 2014, Michael Stonebraker
 - “For fundamental contributions to the concepts and practices underlying modern database systems”
 - Led to the 2014 Turing Award
 - INGRES relational database system in 1974, based on Codd’s papers
 - Inspired numerous other database systems, including Postgres

Key RDBMS Features

- Querying
- ACID properties
 - Atomicity
 - Consistency
 - Isolation
 - Durability

Querying in the Relational Model

- Querying is done via a “relational calculus”
- Declarative
 - You give a mathematical description of the tuples you want
 - System figures out how to get those for you
- ? Why is this good?

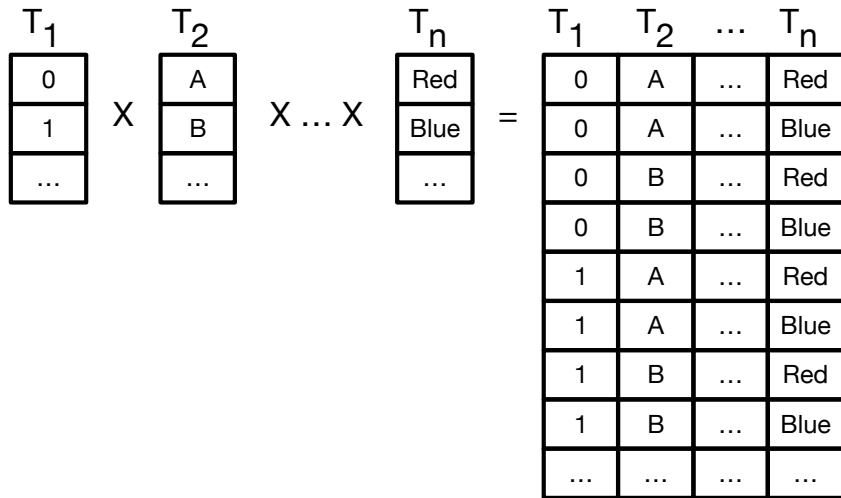
Querying in the Relational Model

- Querying is done via a “relational calculus”
- Declarative
 - You give a mathematical description of the tuples you want
 - System figures out how to get those for you
- ? Why is this good?
 - Data independence!
 - Your code has no data access specifications
 - You can change the physical organization with no code re-writes

- All data are stored in tables, or relations
- A relation schema consists of:
 - A relation name (e.g., LIKES)
 - A set of (attribute_name, attribute_type) pairs
 - Each pair is referred to as an “attribute”
 - Or sometimes as a “column”
 - Usually denoted using
LIKES (DRINKER string, COFFEE string)
 - Or simply LIKES (DRINKER, COFFEE)

- A relation schema defines a set of sets
 - Specifically, if T_1, T_2, \dots, T_n are the n attribute types
 - Where each T_i is a set of possible values
 - Ex: string is all finite-length character strings
 - Ex: integer is all numbers from -2^{31} to $2^{31} - 1$
 - Then a realization of the schema (aka a “relation”) is a subset of
 - $T_1 \times T_2 \times \dots \times T_n$
 - where \times is the Cartesian product operator

Attribute Types Forming a Relation



However, not all tuple combinations actually occur

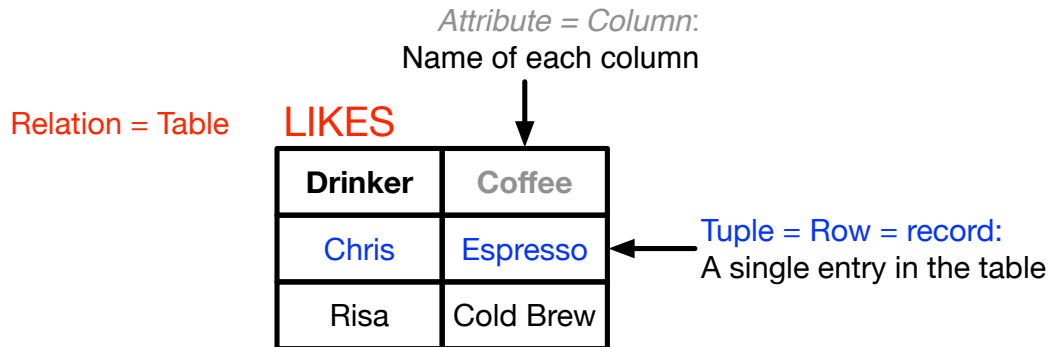
A Relation (continued)

- So for the relation schema
LIKES (DRINKER string, COFFEE string)
- A corresponding relation might be

$\{('Chris', 'Espresso'), ('Risa', 'Cold Brew')\}$

- This is also referred to as a “table”
- The entries in the relation are referred to as
 - “rows”
 - “tuples”
 - “records”

Relational Terminology



- In the relational model, given $R(A_1, A_2, \dots, A_n)$
- A set of attributes $K = \{K_1, \dots, K_m\}$ is a KEY of R if:
 - For any valid realization R' of R ...
 - For all t_1, t_2 in R' ...
 - If $t_1[K_1] = t_2[K_1]$ and $t_1[K_2] = t_2[K_2]$ and ... $t_1[K_m] = t_2[K_m]$...
 - Then it must be the case that $t_1 = t_2$
- ? Note: every relation schema SHOULD have a key... why?

- ? What is a key for
STUDENT (NETID, FNAME, LNAME, AGE, COLLEGE)?

? What is a key for
LIKES (DRINKER, COFFEE)?

- What is a key for
LIKES (DRINKER, COFFEE)?
What is the relation about?
Is it about a drinker's favorite style of coffee?
Or about the first person to drink a coffee?
Or about what styles of coffee does each drinker like?
Context matters!

Keys (continued)

- A relation schema can have many keys
- Those that are minimal are **CANDIDATE KEYS**
 - “Minimal” means no subset is a key
- One is typically designated as the **PRIMARY KEY**
- This key typically determines the order in which the data are stored
- Denoted with an underline
 - STUDENT (NETID, FNAME, LNAME, AGE, COLLEGE)

The relational model does not have pointers that connect different relations

? Why not?

The relational model does not have pointers that connect different relations

Why not?

- 1 Not nice mathematically
 - Mathematical elegance key goal in model design
- 2 Difficult implementation
 - Move an object? All pointers are invalid!
 - Solution: Use a centralized look-up table
 - Expensive
 - Complicated
 - Problem still exists

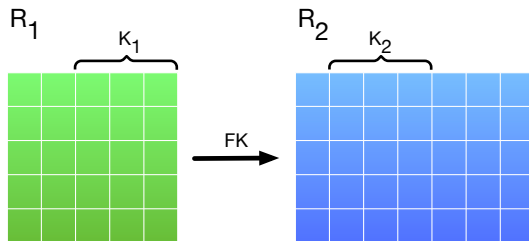
Solution: Foreign Keys

- But we still need some notion of between-tuple references
 - LIKES (DRINKER, COFFEE)
 - DRINKER (DRINKER, FNAME, LNAME)
 - Clearly, LIKES.DRINKER refers to DRINKER.DRINKER
- Accomplished via the idea of a FOREIGN KEY

Foreign Keys (continued)

- Given a relation schema: R_1, R_2
 - We say a set of attributes K_1 from R_1 is a foreign key to a set of attributes K_2 from R_2 if...
 - (1) K_2 is a candidate key for R_2 , and...
 - (2) For any valid realizations R'_1, R'_2 of R_1, R_2 ...
 - For each tuple $t_1 \in R'_1$, it MUST be the case that there exists $t_2 \in R'_2$ s.t...
 - $t_1[K_{1,1}] = t_2[K_{2,1}]$ and $t_1[K_{1,2}] = t_2[K_{2,2}]$ and ... $t_1[K_{1,m}] = t_2[K_{2,m}]$

? Intuitively, what does this mean?



Foreign Key Interpretation

Intuitively, what does this mean?

- The foreign key must be an attribute or set of attributes that uniquely identify a record in another table
- AND that combination of attribute values must be present in the other table
- The foreign key may consist of
 - More than one attribute
 - Attributes with different names in each table

? Which way is the foreign key?

PERSON

| NETID | FIRSTNAME | LASTNAME |
|-------|-----------|----------|
| cmj4 | Chris | Jermaine |
| rbm2 | Risa | Myers |

LIKES

| DRINKER | COFFEE |
|---------|------------|
| cmj4 | Espresso |
| rbm2 | Cold Brew |
| cmj4 | Chai Latte |

- In other words
 - The target must be a candidate key
 - There are no dangling pointers
- ? Why is this a requirement?

Foreign Key Interpretation

- In other words
 - The target must be a candidate key
 - There are no dangling pointers
- Why is this a requirement?
 - To prevent inconsistencies
 - To match to a single target
- The database enforces these requirements via
 - Cascading deletes
 - Failed inserts

Queries/Computations in the Relational Model

- The original query language was the RELATIONAL CALCULUS
 - Fully declarative programming language
 - Mostly theoretical/math – not actually implemented
 - Helps you understand how to write SQL
- next was the RELATIONAL ALGEBRA
 - Imperative
 - Define a set of operations over relations
 - An RA program is then a sequence of those operations
 - This is the “abstract machine” of RDBs
 - Helps you understand query performance

- Today we use SQL
 - Heavily influenced by RC
 - Has aspects of RA
 - More complex than either of them!

Overview of Relational Calculus

- RC is a variant on first order logic
- You say: “Give me all tuples t where $P(t)$ holds”
- $P(t)$ is a predicate in first order logic

- First order logic allows predicates
 - Predicate: A function that evals to true/false
 - “It’s raining on day X” or *Raining*(X)
 - “It’s cloudy on day X” or *Cloudy*(X)
- Build more complicated predicates using logical operations over them
 - and (\wedge) (both)
 - or (\vee) (either or both)
 - not (\neg) (flip truth value)
 - implies (\rightarrow)
 - if and only if (\leftrightarrow)

Predicates: And

$Raining(X) \wedge Cloudy(X)$ Evaluates to TRUE if both:

- It is raining on day X
and
- It is cloudy on day X

AND Truth Table

| p | q | $p \wedge q$ |
|----------|----------|--------------------------------|
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | F |

$Raining(X) \wedge Cloudy(X)$ Evaluates to TRUE if either:

- It is raining on day X
or
- It is cloudy on day X

OR Truth Table

| p | q | $p \vee q$ |
|----------|----------|------------------------------|
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |

Predicates: Implies

$Raining(X) \rightarrow Cloudy(X)$

Evaluates to TRUE if either:

- It is not raining on day X
or
- It is raining and cloudy on day X
- \rightarrow is like a logical “if-then”
- State is FALSE if it is raining and not cloudy
- If it's not raining, we don't care about whether or not it's cloudy

IMPLICATION

Truth Table

| p | q | $p \rightarrow q$ |
|----------|----------|-------------------------------------|
| T | T | T |
| T | F | F |
| F | T | T |
| F | F | T |

Predicates: If and Only If

Raining(X) \leftrightarrow Cloudy(X) Evaluates to TRUE if both:

- Raining(X) \rightarrow Cloudy(X)
AND
- Cloudy(X) \rightarrow Raining(X)

IFF Truth Table

| p | q | p \rightarrow q | q \rightarrow p | p \leftrightarrow q |
|----------|----------|-------------------------------------|-------------------------------------|---|
| T | T | T | T | T |
| T | F | F | T | F |
| F | T | T | F | F |
| F | F | T | T | T |

First Order Logic

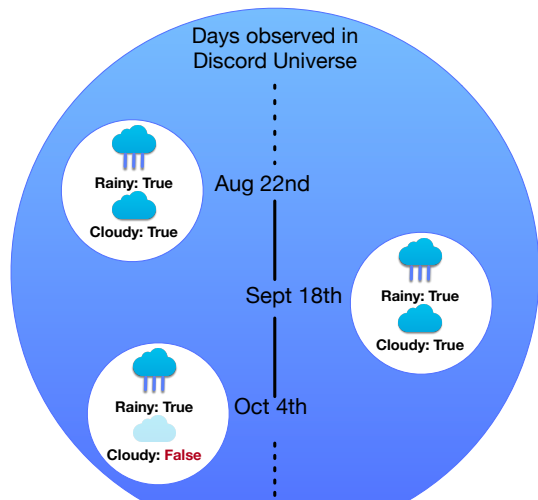
- Just predicates and logical ops?
 - You've got predicate logic
- But when you add quantification
 - \forall, \exists
 - You've got first order logic

Universal Quantification

- Asserts that a predicate is true all of the time
- Example:
 - $\forall(X)(\text{Raining}(X) \rightarrow \text{Cloudy}(X))$
 - Zero-argument predicate (takes no params)
 - Asserts that it only rains when it is cloudy
 - Note: idea of universe of discourse is key!
 - X is a variable. It ranges over the entire universe of discourse
 - Plug the value of that variable into the predicate. If it is always true, then the predicate is true

Universal Quantification Example 1

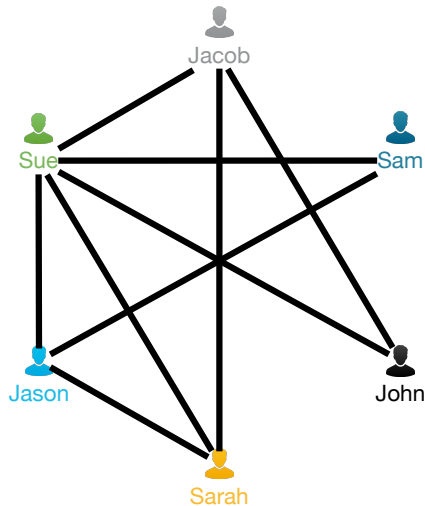
- $\forall(X)(\text{Raining}(X) \rightarrow \text{Cloudy}(X))$
- Given the following Universe of Discourse, is our predicate T or F?



| Day | Rainy | Cloudy | $\text{Raining}(X) \rightarrow \text{Cloudy}(X)$ |
|-----------|-------|--------|--|
| | | | |
| | | | |
| | | | |
| \forall | | | |

Universal Quantification Example 2

- $\forall(X)(\text{Friends}(X, Y))$
- This is a predicate over Y
- Asserts that person Y is friends with everyone
- Can be T or F, depending on what's in the U of D



- Asserts that a predicate **can** be satisfied
- Example:
 - $\exists(X)(\text{Raining}(X) \wedge \neg \text{Cloudy}(X))$
 - Asserts that it is possible for it to rain when it is not cloudy
 - aka a “sun shower”

Important Equivalence

- $\forall(X)(P(X))$ is equivalent to...
- $\neg\exists(X)(\neg P(X))$
 - Ex: $\neg\exists(X,Y)(\text{Friends}(X,Y) \wedge \text{Friends}(X,Z) \wedge \text{Friends}(Y,Z))$
 - Can be changed to:
 - $\forall(X,Y)(\neg(\text{Friends}(X,Y) \wedge \text{Friends}(X,Z) \wedge \text{Friends}(Y,Z)))$
 - Or $\forall(X,Y)(\neg\text{Friends}(X,Y) \vee \neg\text{Friends}(X,Z) \vee \neg\text{Friends}(Y,Z))$
- Which is easier?
 - Often easier to reason about \exists compared to \forall
 - Can be hard to conceptualize an assertion that something is true over every item in the entire universe!
 - In fact, SQL does not even have \forall
- ? What is the name of rule applied to distribute the negation?

Wrap up

- 1 What is the fundamental benefit of the relational model?
 - 2 Why does it matter?
- ? How can we use what we learned today?
 - ? What do we know now that we didn't know before?