The Relational Data Model

COM 3563: Database Implementation

Avraham Leff

Yeshiva University

avraham.leff@yu.edu

COM3563: Fall 2020

Today's Lecture: Overview

1. The Relational Data Model
2. Implications Of Relational Model
3. Keys
4. Foreign Keys
5. Look Ahead

Many figures taken from lectures created by Professor Zvi Kedem, NYU Courant (used with permission), and from lectures by Professor Greiner (used with permission)

The Relational Data Model

Introduction

- Google “learn sql” and marvel at the sheer number of results that come back
- Unfortunately, much of that material focuses on a “cookbook approach”
- In this course, you’ll learn how the power of SQL derives from the relational data model

> The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations

-
- Please don’t run away ☺
- By understanding “relational semantics”, you will be a much better database programmer
- Key point: a relational database is not simply “tables, columns & rows”

Relations

- A relation is a mathematical concept based on the ideas of sets
  - First proposed by Dr. Codd of IBM Research in *A Relational Model for Large Shared Data Banks* Communications of the ACM, June 1970
  - Fair to say that this paper caused a major revolution in the field of database management (earned Dr. Codd the ACM Turing Award (1981)
- Relation: basic "data type" of relational databases
- When interacting with a relational database instance, relations are the basic "programming variable"
- Initially we'll assume that relations are sets
  - Later, we'll allow relations to be multisets
  - Meaning: *duplicates are allowed* (unlike sets)

Relations Are Sets (I)

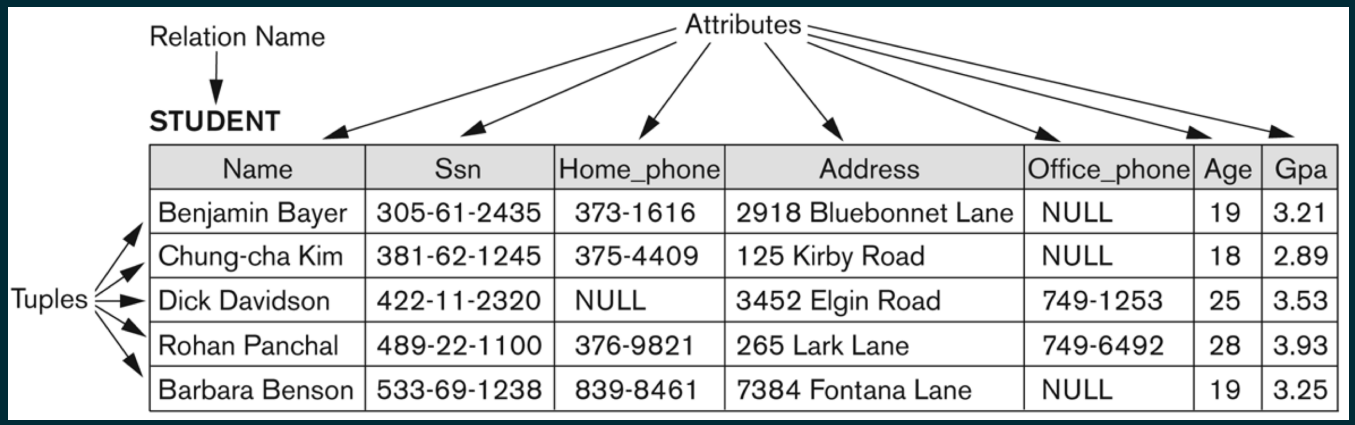
- The term “relation” does not pertain to relationships between tables
  - Although we’ll see that this concept is also crucial to the relational data model
- Rather:

  In mathematics, an n-ary relation on n sets, is any subset of Cartesian product of the n sets (i.e., a collection of n-tuples) …

  Heterogeneous n-ary relations are used in the semantics of predicate calculus, and in relational databases.

  Source
- This definition needs to be unpacked …slowly ☺

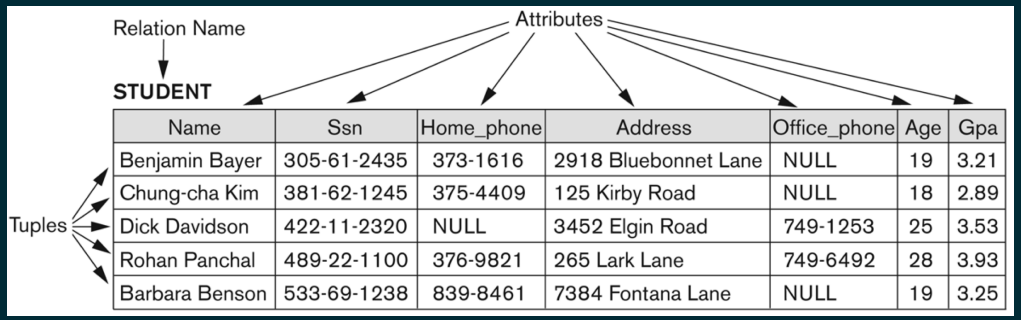
Relations Are Sets (II)



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

- A relation is a set of tuples (see Figure)
- Tuples in the "student" relation are "7-tuples"
  - Meaning: each tuples is an ordered set of values, with each value derived from a specified domain (next slide)
  - Because the domains are often obvious, we usually focus on the value, and ignore the domain: this is a mistake!
- Make sure that you see how each of these tuples *"is any subset of Cartesian product of the n sets (i.e., a collection of n-tuples)"*
  - There is a set of valid student names (the "name" domain)
  - There is a set of valid social-security-numbers (the "Ssn" domain)
  - etc

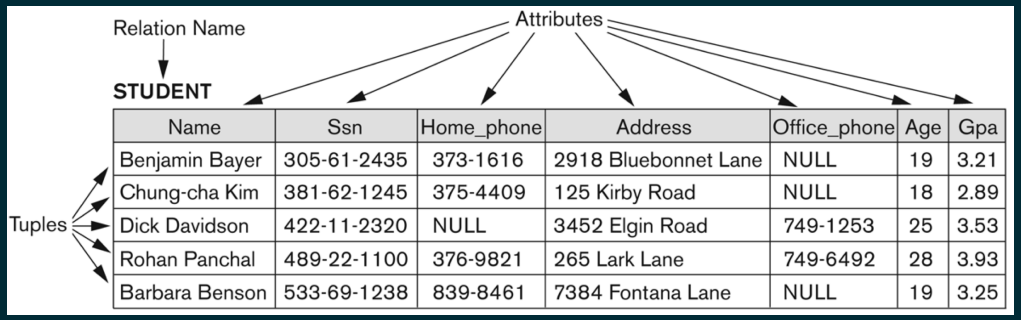
Tuples & Domains (I)



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

- OK: we now know that a tuple is an ordered set of values, each value derived from a domain ...
- Q: but what is a domain?
- A: a domain is a definition of a set of permitted values
  - Example: *"USAPhoneNumbers"* is the set of 10 digit phone numbers valid in the USA
    - Note: this is not the set of integers!
    - Note: this is not the set of Strings!
- Think of a domain as a programming language "data-type"
  - But more sophisticated: because it can include a "format"
  - Example: *"USAPhoneNumbers"* may have this format: *(ddd)ddd-dddd* where each *d* is a decimal digit

Tuples & Domains (II)



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

- Because multiple tuple values may be associated with the same domain, we need a way to distinguish them
  - Example: The domain *Date* may be used to define both "Invoice date" and "Payment date"
  - Can't use "order" because tuples are sets ☺
- Instead: associate tuple values with both a domain and a attribute name
  - Example: use the attribute names *Invoice-date* and *Payment-date* to distinguish the two Date-valued tuple values
  - Attribute names convey the semantics of the tuple value

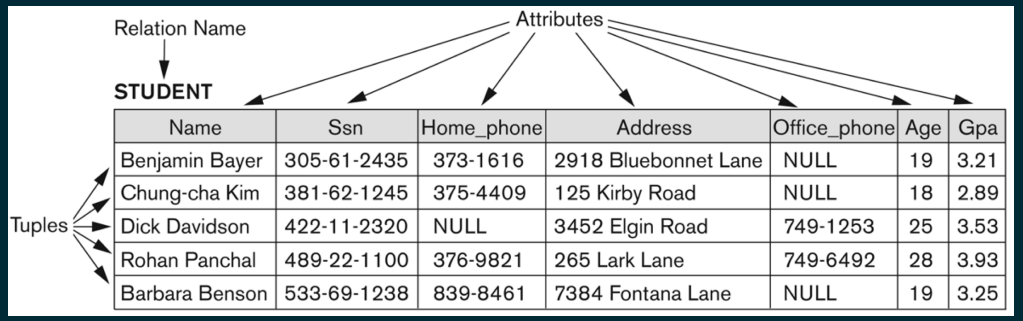
Relation Instances

- Think of individual tuples as being "facts" about the specified domains
  - Predicates whose values are "true"
  - Example: *"there is a student with the specified name and given social-security name ..."*
- It follows that a relation is only a "point-in-time" statement!
  - Example: the student has a GPA of 3.82 now, but that may no longer be true at the end of the semester ☺
- We therefore must distinguish between relation instances and associated relation schema
  - A relation instance is a specific set of tuples (per our previous definitions)
  - A relation schema is the logical definition of the relation
    - It applies to all instances of the relation, and is valid "across time"

Relational Schemas

- Think of a relation schema as "tuples without the values"
  - Just the ordered sequence of $\{attributename, domain\}$ pairs
- Analogy: a Java Integer type is to the Integer value 3 as a relational instance is to a relation schema
- Alternatively: a schema corresponds to the header of a database table
  - A relation instance is the populated table
  - The instance corresponds to a relation's state (a point in time statement)

Don’t Lose Sight of the Big Picture



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

Informal Terms		Formal Terms
Table		Relation
Column Header		Attribute
All possible Column Values		Domain
Row		Tuple
Table Definition		Schema of a Relation
Populated Table		State of the Relation

- This slide tells you that it’s OK to think of a relational database as a collection of tables, where each table contains a bunch of rows, and each row consists of a number of columns
- OK: but only if you keep the set-concept fundamentals in mind
  - Otherwise, you simply won’t see the “big picture” of how things work ☹

Implications Of Relational Model

Implications of “Relations Are Sets”: I

S	A	B
	a	2
	a	56
	b	2

S	A	B
	a	56
	a	2
	b	2
	a	56
	a	2
	a	56

- Relations are sets of tuples
- So: these two relations are equal
- Your SQL queries will have to decide whether you want to “remove duplicates”
- Your SQL queries will have to decide whether you want tuples presented in a a given order

Implications of “Relations Are Sets”: II

S	A	B
	a	2
	a	56
	b	2

S	B	A
	56	a
	2	a
	2	b
	56	a
	2	a
	56	a

- Attribute position in a tuple is irrelevant
  - Because attributes are named
- So: these two relations are equal
- Your SQL queries will have to decide the order in which you present tuple attributes

Implications of “Relations Are Sets”: III

- Attribute values are required to be atomic
  - Meaning: indivisible
- Classic example: phone-numbers
  - Cannot associate a person with a set of phone numbers
  - Cannot even treat a single phone number as “really” consisting of an “area-code” + “seven digits” (you will not be able to query on these sub-parts)
- You get to define the appropriate semantics
  - But if you want “queries by area-code”, then you must define the appropriate domains for your relation

Touching Lightly On The Topic of NULL

- Another important implication of the “pure” relational model . . .
- The concept of an attribute having the value of NULL makes no sense ☹
  - To paraphrase Yoda:

  Do. Or do not. There is no try.

  - For a given tuple: *“A tuple attribute has a value. Or the tuple attribute doesn’t exist. There is no NULL”*
- We’ll talk a bit more about this later in the lecture and a lot more in another lecture ☺

Most Important Consequence Of Set-Based Relational Model (I)

- To appreciate the importance of having a set-based data model, we must look ahead to the topic of SQL
- You probably already know that we use SQL to query and manipulate relational databases
  - That part is true ☺
- You possibly think of SQL as a “programming language” (like Java)
  - That part is false ☺
- In a nutshell
  - SQL is based on relational algebra (with many extensions)
  - Relational algebra is the topic of Chapter 2.6 which you’re not responsible for
- “Pure” relational algebra uses mathematical notation with Greek letters
  - SQL is a “commercial” version of relational algebra
  - Some necessary extensions, some unnecessary (& therefore irritating) ones

Most Important Consequence Of Set-Based Relational Model (II)

- Many database textbooks first present the relational algebra, then proceed to SQL
- I consider this to be a waste of your time ☺
  - In real life™ you will only be using SQL to interact with relational databases
- OTOH: SQL semantics are fundamentally that of relational algebra
  - So: we'll cover relational algebra (in a couple of lectures) using SQL ☺
- For now: a quick explanation of how the set-based relational data model is closely tied to relational algebra

Sets And Operations On Them

- If A, B, and C are sets, then we have the operations
- $\cup$ Union, $A \cup B = \{ x \mid x \in A \ \vee \ x \in B \}$
- $\cap$ Intersection, $A \cap B = \{ x \mid x \in A \ \wedge \ x \in B \}$
- $-$ Difference, $A - B = \{ x \mid x \in A \ \wedge \ x \notin B \}$
- In mathematics, difference is frequently denoted by a symbol similar to a backslash: $A \setminus B$
- $\times$ Cartesian product, $A \times B = \{ (x,y) \mid x \in A \ \wedge \ y \in B \}$, $A \times B \times C = \{ (x,y,z) \mid x \in A \ \wedge \ y \in B \ \wedge \ z \in C \}$, etc.
- The above operations form an ***algebra***, that is you can perform operations on results of operations, such as $(A \cap B) \times (C \times A)$ and such operations ***always produce sets***
- So you can write expressions and not just programs!

Relations in Relational Algebra

- Relations are sets of ***tuples***, which we will also call ***rows***, drawn from some domains
- These domains ***do not*** include NULLs
- Relational algebra deals with relations (which look like tables with fixed number of columns and varying number of rows)
- We assume that each domain is linearly ordered, so for each *x* and *y* from the domain, one of the following holds
  - $x < y$
  - $x = y$
  - $x > y$
- Frequently, such comparisons will be meaningful even if *x* and *y* are drawn from different columns
  - For example, one column deals with income and another with expenditure: we may want to compare them

Operations on relations

- There are several fundamental operations on relations
- We will describe them in turn:
  - Projection
  - Selection
  - Cartesian product
  - Union
  - Difference
  - Intersection (technically not fundamental)
- A very important property: ***Any operation on relations produces a relation***
- This is why we call our structure an ***algebra*** (closed under operations)

Keys

Relational Model: Constraints (I)

- The relational data model has a reputation for having a "strict structure" (as compared to other database models)
- This reputation is well deserved ☺
- Constraints are a very good example
- As we shall discuss many times, a production system needs as many constraints as possible!
  - Otherwise: application programmers have to implement the identical semantics in their code
  - Multiple times, usually badly ☹
  - Far better to use robust, polished middleware!
- Constraints determine which values are permitted or not permitted in the database

Relational Model: Constraints (II)

Three types of constraints

- Data model represents implicit constraints
  - Example: relational database doesn't allow attributes to be "list-valued"
- Schema-based constraints are explicit
  - DBA specifies them when defining a relation's schema
  - We'll drill down on this type of constraint in the next slide
- Application constraints must be defined (and enforced) by the programmer because they're beyond the expressive power of the middleware
  - If they're useful, capitalism may drive that type of constraint into the middleware ☺
  - Example: *"Our department requires that, in every semester, at least one course is taken in a different department"* ☺
  - Note: SQL-99 introduced the CREATE TRIGGER and CREATE ASSERTION to express some of these semantic constraints

Schema-Based Constraints: Introduction

- By specifying any of these (explicit, schema-based) constraints, the DBA specifies conditions that must hold for all valid relation states
  - Note: in work for this course, you are the DBA!
- Some examples:
  - Key constraints (next slides)
  - Entity integrity constraints (after we discuss "keys")
  - Referential integrity constraints (at the end of today's lecture)
- Another schema-based constraint: the domain constraint
  - *"Every value in a tuple must be from the domain of its attribute"*
  - One exception: the infamous NULL value (if the DBA specifies that this is OK)

Keys & Relational Model

- A key is used to specify a specific tuple
- The relational model is oblivious to concept of keys
  - Set theory doesn't say anything about how you identify set elements ☺
- So why do relational databases include "keys"?
- For efficiency!
- Contrast `java.util.Set` API to `java.util.Map`

Very important point: when defining the key, you must consider the universe of all potential relation data (for this relation schema), not just what the table currently contains

Key Semantics: To Uniquely Identify Tuples

Person	FN	LN	Grade	YOB
	John	Smith	8	1976
	Lakshmi	Smith	9	1981
	John	Smith	8	1976
	John	Yao	9	1992

- So: to say anything useful about keys for a given relational schema, we have to know quite a bit about the schema
- In this example, we might be "told"
  - "Any two tuples that are equal on both FN and LN are equal"
    - This is an assertion: the combination of a first and last name uniquely identifies a Person
    - This is not just an assertion about this relation instance: it's an assertion about every instance of this schema
    - This assertion is unlikely to be true ☺

Superkeys versus Key

Person	FN	LN	Grade	YOB
	John	Smith	8	1976
	Lakshmi	Smith	9	1981
	John	Smith	8	1976
	John	Yao	9	1992

- We can then say that **(FN, LN)** is a superkey of Person
- A superkey is a combination of attributes that can be uniquely used to identify a tuple
  - Depending on the schema, many possible combinations may have this property
- Here we're "told" that *neither FN nor LN by themselves suffice to identify a Person tuple*
- We therefore say that **(FN, LN)** are a key (not *only* a superkey)
  - A "key" is a minimal set of attributes needed to identify a tuple

Multi-Attribute Keys versus Single-Attribute

Pay	Grade	Salary
	8	128
	9	139
	7	147

- We're "told" that any two tuples that are equal on **Grade** are equal on all other attributes
  - HR insists that this is true ☺
- So: **(Grade)** is a key
- But: **(Salary)** is not a key
  - (Assuming we've been "told" that people may have the same salary but not the same pay grade)

- Purpose of key is to uniquely identify a tuple:
  - So don't worry about the number of key attributes
  - You use as many attributes as are needed to provide the required semantics

Time For Some Definitions

- A set of columns in a relation is a superkey *iff* any two tuples that are equal on the elements of these columns are completely equal
  - Terminology: superkey has a "super-set" ($\supseteq$) of a key's attributes
- A relation always has at least one superkey
  - Because: the set of all the attributes is a superkey
  - Any two tuples that are equal on all attributes are "completely equal" ☺
- A key is a minimal superkey
  - That is: removing any attribute from the set used by that key creates something that cannot be used as a key

Implications of These Definitions

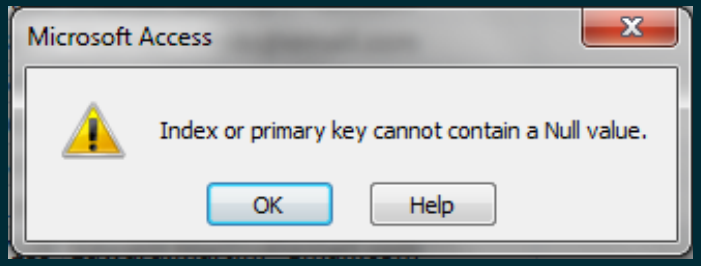
- A relation always has at least one key
  - Start with any superkey and remove unnecessary attributes
- There may be more than one key
- Exactly one key is chosen as the relation's primary key
  - Completely arbitrary choice
- Remaining keys are sometimes called candidate keys
  - They were "candidates" for the primary key, but were not chosen ☹

The NULL Issue

Warning: this is a contentious topic, no time to do justice to it now …

- NULL is the placeholder value used to denote "this tuple does not have a value for this attribute"
  - Not just an SQL artifact: introduced by Codd himself
- "Does not have a value" is not the same as "has the value of zero" (or the equivalent for a given domain)
  - Means: we do not know the value
  - Cannot be emphasized enough ☺
- A predicate comparison of a value to NULL therefore can return neither "true" nor "false"
  - Only: "unknown"
  - Q: Do you see what this has to do with three-valued logic?
  - A: Because, when comparing attribute values, we potentially have three states to evaluate
  - More in a later lecture!

NULL And Primary Keys



- Because a (primary) key uniquely identifies a relation's tuples, the corresponding attributes are not allowed to be NULL
- We refer to this property (supplied by relational databases) as entity integrity
  - Note: if a primary key has several attributes, none of these attributes may be assigned the NULL value
  - The DBA may also constrain other attributes to forbid NULL values
- Reminder: do not confuse database NULL with e.g., Java "object reference is null"!
- Java NULL actually is a value
- It has its own set of contentious issues, but not the one we're discussing now ☺

Foreign Keys

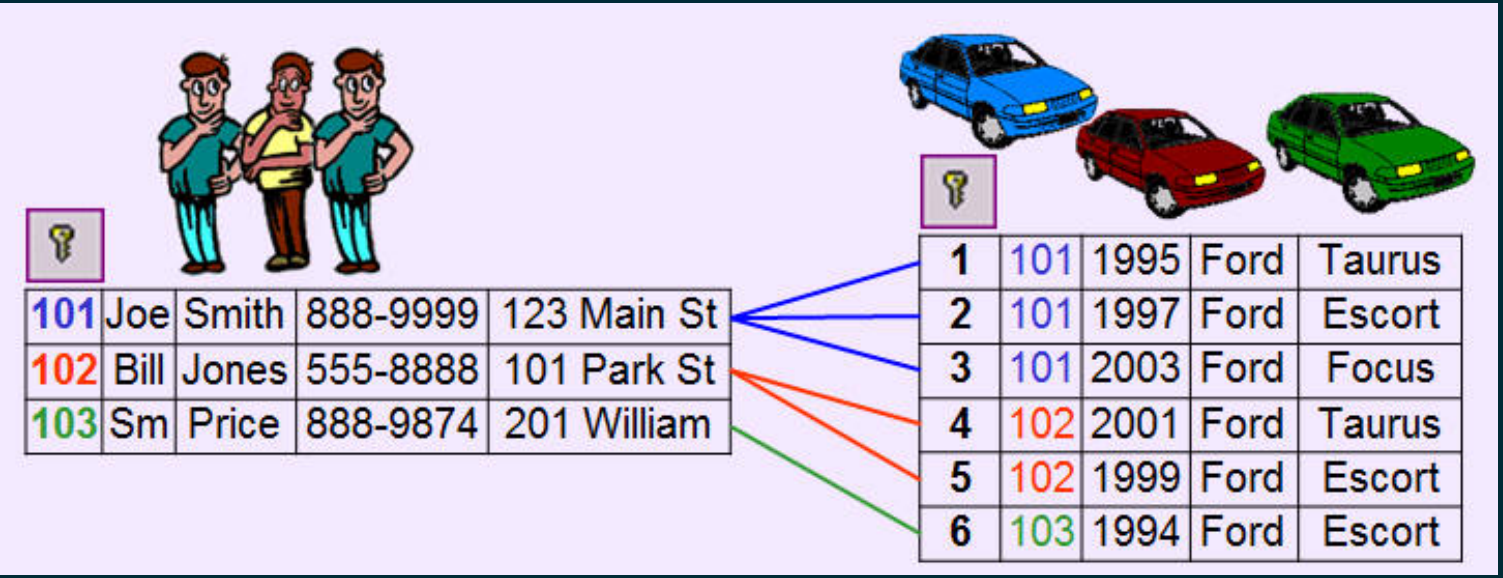
Do you agree with the following definition?

Definition

A relational database is a set of tables with values drawn from specified domains, specified using a relational schema

- This definition conforms to what we've been discussing
- But ignores the important issue of defining relationships between relations

Relations Can Reference Other Relations



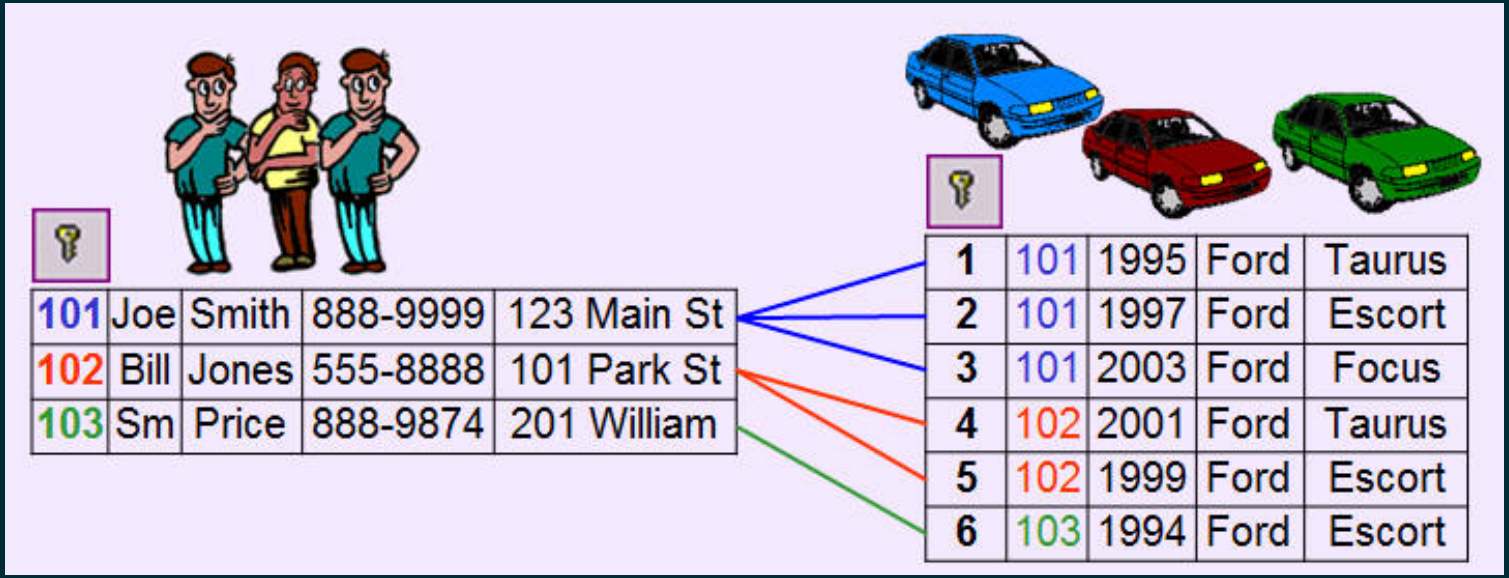
101	Joe	Smith	888-9999	123 Main St
102	Bill	Jones	555-8888	101 Park St
103	Sm	Price	888-9874	201 William

1	101	1995	Ford	Taurus
2	101	1997	Ford	Escort
3	101	2003	Ford	Focus
4	102	2001	Ford	Taurus
5	102	1999	Ford	Escort
6	103	1994	Ford	Escort

Auto Mechanic Example:

- Relation on the left is our *Customer* data
  - Leftmost column in *Customer* table is *Id*
  - This is a primary key attribute
- Relation on the right is our *Customer Automobile* data
  - Note the column with colored values: *CustomerId*
  - *CustomerId* is not a “stand-alone” attribute

Foreign Keys



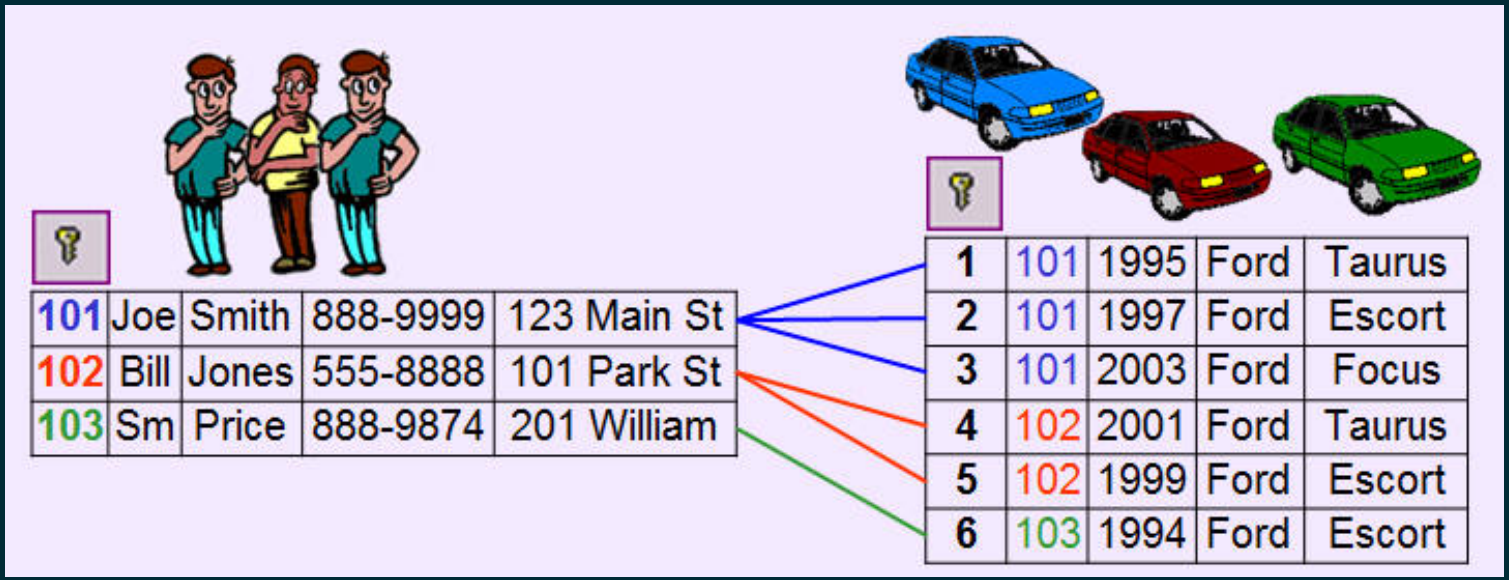
101	Joe	Smith	888-9999	123 Main St
102	Bill	Jones	555-8888	101 Park St
103	Sm	Price	888-9874	201 William

1	101	1995	Ford	Taurus
2	101	1997	Ford	Escort
3	101	2003	Ford	Focus
4	102	2001	Ford	Taurus
5	102	1999	Ford	Escort
6	103	1994	Ford	Escort

- *Customer Automobile* includes among its attributes the primary key of another relation (*Customer*)
- *CustomerId* is termed a foreign key referencing *Customer*
- *Customer Automobile* is a referencing relation of the foreign key
- *Customer* is the referenced relation of the foreign key

Referential Integrity Constraint

Values appearing in the *referencing relation* must appear in some tuple of the *referenced relation*



101	Joe	Smith	888-9999	123 Main St
102	Bill	Jones	555-8888	101 Park St
103	Sm	Price	888-9874	201 William

1	101	1995	Ford	Taurus
2	101	1997	Ford	Escort
3	101	2003	Ford	Focus
4	102	2001	Ford	Taurus
5	102	1999	Ford	Escort
6	103	1994	Ford	Escort

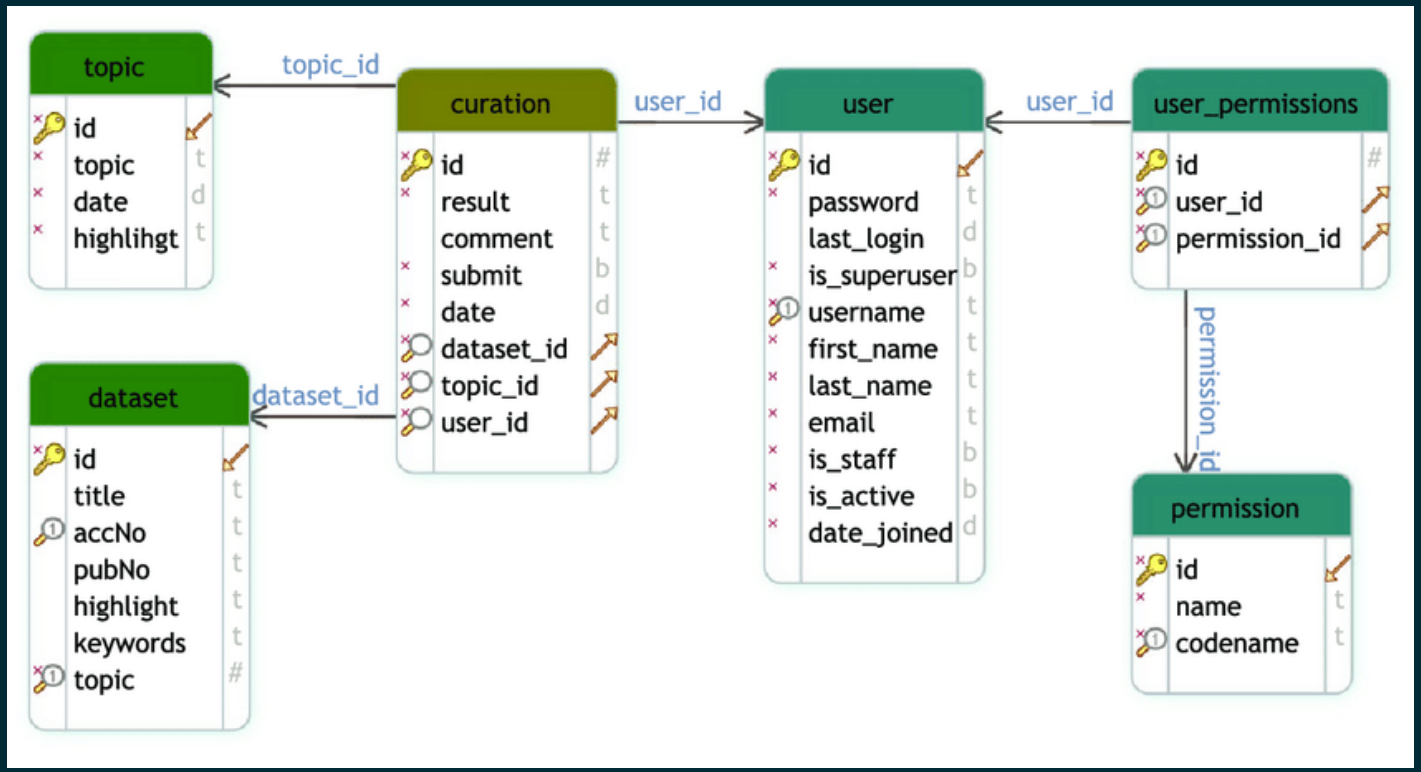
- The referential integrity constraint is sometimes referred to as the foreign key constraint
  - Example: it's illegal to insert a tuple with the value of 42 for the *CustomerId* attribute into the *Customer Automobile* relation

Definition

A relational database consists of

- A set of relations
- A set of binary, “many-to-one mappings” between them
  - Corresponding to the tuples in various referencing relations that map to a single tuple in the referenced relation
- (Conversely) A set of binary “one-to-many mappings” between the “parent-to-child” relations
  - Example: all relations that include a “customer id” attribute
  - Name doesn’t matter: it’s the fact that DBA declared that there is a foreign key relationship

Schema Diagrams



- Each relation appears as a box, relation name on top
- Attribute names inside the box
- Note: non-graphical version indicate primary key attribute (or attributes) by underlining the relevant attributes
- Foreign key dependencies appear as arrows from referencing relation to primary key of referenced relation

Look Ahead

Is There Anything Wrong With This Relation?

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

- You can infer the relational schema from this table
  - Each tuple contains some “Professor” information and some “Department” information
- Given the previous discussion, how many problems can you detect?

Schema Problems (I)

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

If a department has more than one instructor, the building name and budget get repeated multiple times. Updates to the building name and budget may get performed on some of the copies but not others, resulting in an inconsistent state where it is not clear what is the actual building name and budget of a department.

Example: consider the {Physics, Watson, 70000} tuples

Schema Problems (II)

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

A department needs to have at least one instructor in order for building and budget information to be included in the table. Nulls can be used when there is no instructor, but null values are rather difficult to handle.

Schema Problems (III)

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

If all instructors in a department are deleted, the building and budget information are also lost. Ideally, we would like to have the department information in the database irrespective of whether the department has an associated instructor or not, without resorting to null values.

Database Normalization

The problems we just observed are addressed in the important topic of database normalization

- We want our schemas to
  - Store information without redundancy
  - Store information such that the information we need can be retrieved easily
- These two goals are (somewhat) in conflict with one another
- Good solutions require understanding how to use relational operators correctly together with normalization theory
- We will cover “normalization theory” later in the course

Today's Lecture: Wrapping it Up

The Relational Data Model

Implications Of Relational Model

Keys

Foreign Keys

Look Ahead

Before Our Next Class

- **Readings:** Textbook Chapter 2 through 2.4