ECE 650
Systems Programming & Engineering

Spring 2018

Relational Databases:
Tuples, Tables, Schemas, Relational Algebra

Tyler Bletsch

Duke University

Slides are adapted from Brian Rogers (Duke)

Overview

- Relational model - Ted Codd of IBM Research in 1970
  - “A Relational Model of Data for Large Shared Data Banks”
- Attractive for databases
  - Simplicity + mathematical foundation
- Based on mathematical relations
  - Theoretical basis in set theory and first order predicate logic
- Implemented in a large number of commercial databases
  - E.g. Oracle, PostgreSQL, Microsoft Access, etc.

Relational Model

- Represents database as a collection of *relations*
  - Think of a relation as a table of values
  - E.g.

Employee Table

Name	Position	Department	Phone #
Reynolds	Manager	Sales	555-555-5444
Smith	Engineer	Development	555-555-5555

- Relation as a table
  - Table name is called a ***relation***
  - Each row represents a collection of related data values (***tuple***)
  - Columns help interpret meaning of values in each row; also called an ***attribute***
    - All values in a column have the same *data type*
    - Data type of the values that can appear in column is called ***domain***

Definition Summary

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

Domain

- What is a domain
  - Set of atomic values
    - Each value in domain is indivisible from relational model view
  - Commonly specified as a data type; often domain given a name
- Examples (logical definitions):
  - USA_phone _numbers: set of 10-digit phone #'s valid in US
  - Local_phone_numbers: set of 7-digit phone #'s value in area code
  - Names: Set of names of persons
  - Grade_point_averages: Set of real numbers between 0 and 4
- Name, data type, format:
  - USA_phone_numbers is char string of form (ddd)ddd-dddd
    - Where d is a decimal digit and first 3 digits are a valid area code

Relation Schema

- Relation schema R denoted as R(A1, A2, …,An)
  - Made up of relation name R and list of attributes A1, A2, …, An
  - Attribute Ai
    - Names a role played by some domain D in relation schema R
    - D is the domain of Ai and is denoted by dom(Ai)
- Relation Schema describes a relation (named R)
- Degree of a relation is number of attributes n
- Example relation schema of degree 7:
  - STUDENT(Name, SSN, HomePhone, Address, OfficePhone, Age, GPA)

Relation

- A ***relation*** of a relation schema R is denoted by r(R)
  - Set of n-tuples: r = {t1, t2, …, tm}
  - Each n-tuple t is an ordered list of n values t = <v1, v2, …, vn>
    - Where each value vi is an element of dom(Ai) or NULL
    - The ith value in tuple t is referred to as t[Ai]

Employee Table

Attributes

Name	Position	Department	Phone #
Reynolds	Manager	Sales	555-555-5444
Smith	Engineer	Development	null

Relation name

Tuples

Relation (2)

- Stated another way
  - Relation r(R) is a mathematical relation of degree n on the domains dom(A1), dom(A2), …, dom(An)
  - Which is a subset of the Cartesian product of the domains of R
    - r(R) ⊆(dom(A1) x dom(A2) x … x dom(An))
    - Cartesian product specifies all possible combinations
    - Cardinality of domain D is |D|; # of tuples in Cartesian product is:
      - |dom(A1)| * |dom(A2)| * … * |dom(An)|
  - Current relation state:
    - Reflects only valid tuples that represent particular state of real world
    - Schemas are relatively static (change very infrequently)
    - But current relation state may change frequently
  - Possible for several attributes to have the same domain
    - But attributes indicate different roles of the domain
      - E.g. HomePhone vs. OfficePhone

Relational Model Notation

- Relation schema R of degree n is denoted by R(A1, A2, …, An)
- N-tuple t in a relation r(R) is denoted by t = <v1, v2, …, vn>
  - vi is the value corresponding to attribute Ai
  - t[Ai] refers to the value vi in t for Attribute Ai
- Letters Q, R, S denote relation names
- Letters q, r, s denote relation states
- Letters t, u, v denote tuples
- R.A denotes the relation name to which an attribute belongs
  - Since the same name may be used for attributes in different relations

Definition Summary

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

Relational Constraints

Relational Constraints: Restrictions on data that can be specified on a relational database schema

- Domain Constraints
- Key Constraints
- Constraints on NULL
- Entity Integrity Constraint
- Referential Integrity Constraint

Domain Constraints

- Value of each attribute A must be atomic value from dom(A)
- Data types include standard numeric types
  - Integer, long integer
  - Float, double-precision float
- Also characters, fixed-length and variable-length strings
- Others
  - Date, timestamp, money data types
  - Enumerated data types
- Will discuss more when we talk about SQL

Key Constraints (1)

- All tuples in a relation must be distinct
  – No two tuples can have same values for all attributes
- Superkey
  – Set of attributes where no two tuples can have the same values
  – Every relation has at least one default superkey (all attributes)
- Key
  – Superkey with property that removing any attribute from the set leaves a set that is not a superkey of the relation schema
  - Example
    - STUDENT(Name, SSN, HomePhone, Address, OfficePhone, Age, GPA)
      - Attribute set {SSN} is key (no 2 students can have same value)
      - Attribute set {SSN, Name, Age} is a superkey (but not a key)

Key Constraints (2)

- Value of key attribute uniquely identifies each tuple
- Set of attributes constituting a key is a property of the relation schema
  - Should hold on *every* relation state of the schema
  - Time-invariant: should hold even as tuples are added
- A relation schema may have more than one key
  - Each is called a candidate key; one is designated as **primary key**
  - Convention to underline the primary key of a relation schema

Owner	<u>LicenseNum</u>	EngineSerialNum	Make	Model	Year

Entity Integrity Constraint & NULL Constraints

- Entity Integrity Constraint
  - Primary key value cannot be NULL

- NULL may or may not be permitted for other attributes
- E.g. if Name attribute must have a valid, non-null value
  - It is said to be constrained to be NOT NULL

Relational Database

- Contains many relations
- Tuples in relations are related in various ways
- Relational database schema
  - Set of relation schemas S = {R1, R2, …, Rm}
  - Set of integrity constraints (IC)

Example Relational Database Schema

COMPANY = {EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}

EMPLOYEE

FNAME	MINIT	LNAME	<u>SSN</u>	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	<u>DNUMBER</u>	MGRSSN	MGRSTARTDATE

DEPT LOCATIONS

<u>DNUMBER</u>	<u>DLOCATION</u>

PROJECT

PNAME	<u>PNUMBER</u>	PLOCATION	DNUM

WORKS_ON

<u>ESSN</u>	<u>PNO</u>	HOURS

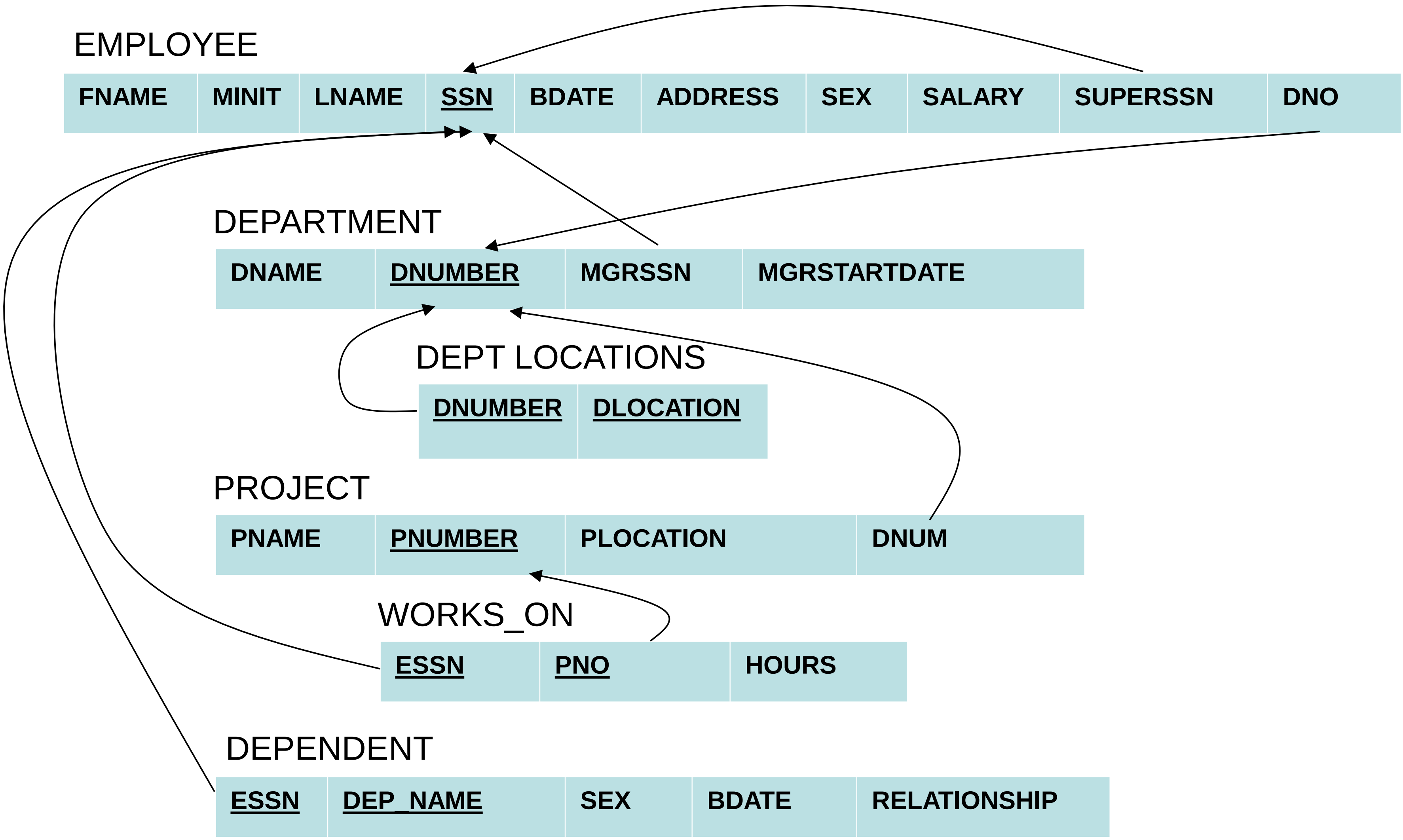
DEPENDENT

<u>ESSN</u>	<u>DEP_NAME</u>	SEX	BDATE	RELATIONSHIP

Referential Integrity Constraint

- Specified between 2 relations
- Maintains consistency among tuples of two relations
- Informally
  - Tuple in a relation that refers to another relation must refer to an existing tuple in that relation
  - Even more informally: you can refer to rows in other tables, but the thing you're referring to has to exist
- Formally
  - For ref integrity constraint between R1 & R2, define *foreign key*
  - Set of attributes FK in R1 is foreign key referencing R2 if:
    1. Attributes in FK have same domain(s) as the primary key attributes PK of R2 (attributes FK thus refer to the relation R2)
    2. A value of FK in tuple t1 of current state r1(R1) either occurs as a value of PK for some tuple t2 in r2(R2) or is NULL

Example Referential Integrity Constraints



Other Constraints

- Semantic Integrity Constraints
  - E.g. salary of employee should not exceed salary of supervisor
  - E.g. max hours an employee can work on all projects per week
  - Can be specified via a constraint specification language
    - Via mechanisms called triggers or assertions
- Transition Constraints
  - Deal with state changes in the database
  - E.g. tenure length of an employee can only increase
  - Specified using rules and triggers

Relational Model Operations

- Updates
  - Insert, delete, modify
  - Integrity constraints must not be violated

- Retrievals
  - Involve relational algebra operations

Insert

- Provides list of attribute values for new tuple t to be inserted into relation R
- Danger: could possibly violate several constraints
  - Domain: attribute value doesn't appear in corresponding domain
  - Key: key value in new tuple t already exists in another tuple
  - Entity: primary key of new tuple t is NULL
  - Referential: foreign key in t refers to a tuple that does not exist
- Example (see example COMPANY database)
  - Insert <'Cecilia', 'F', 'Kolonsky', null, '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, null, 4> into EMPLOYEE
    - Entity integrity constraint violation; insert is rejected

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

Delete

- Specify a deletion
  - Give a condition on the attributes of the tuple(s) of a relation
  - E.g. delete tuple with attributes matching given values
- Danger: Could violate referential integrity
  - If tuple being deleted is referenced by foreign keys in other tuples
- Options if a deletion causes a violation
  - Reject the deletion operation
  - Cascade the deletion
    - Delete tuples that reference the tuple being deleted
  - Modify the referencing attribute values
    - E.g. change them to NULL

Update

- Change values of attribute(s) in tuple(s) of a relation
- Specify a condition on the attributes of the relation to select tuple(s) to be modified
- E.g. update SALARY of EMPLOYEE tuple with SSN='999887777' TO 28000
- Danger?
  - Modifying a primary key: equivalent to delete + insert
  - Modifying a foreign key: check referential integrity
  - Non-keys: Usually valid to update, except must of course be of correct type

Relational Algebra Operations

- Data models must include a set of ops to manipulate data
- Relational Algebra
  - Basic set of relational model operations
- Ops allow users to specify basic data retrieval requests
  - Result of retrieval is a new relation
    - May have been formed from one or more other relations
  - Result relations can be further manipulated with further ops
- Sequence of relational algebra ops form an “expression”
- Relational algebra operations:
  - Set ops: union, intersection, set difference, Cartesian product
  - Ops specifically for relational databases: select, project, join

SELECT Operation

- Essentially a filter over a relation
  - Forms a new relation with only tuples matching a condition
  - Resulting relation has same degree & attributes as original relation
- $\sigma_{\text{<selection condition>}}(R)$
  - E.g. $\sigma_{(\text{DNO=4 AND SALARY > 50000})}(\text{EMPLOYEE})$
  - R is a relation
    - Could be a database relation or result of another select
  - Selection condition can compare (=, <, <=, >, >=, !=)
  - Selection condition clauses can be combined (AND, OR, NOT)
- SELECT operation applies independently to each tuple
  - Resulting number of tuples is less than or equal to original relation
- Note that SELECT is commutative
  - Chain of SELECT ops can be applied in any order

σ
sigma

PROJECT Operation

- PROJECT chooses certain columns of a relation
  - Recall SELECT chooses certain rows of a relation
  - Other columns are discarded
- $\pi_{<\text{attribute list}>}(R)$
  - E.g. $\pi_{\text{LNAME, FNAME, SALARY}}(\text{EMPLOYEE})$
  - Result has only attributes shown in list (in same order as listed)
  - If list only includes non-key attributes, there may be duplicates
    - Duplicate tuples are removed by PROJECT operation
- Commutativity does not hold for PROJECT operation

π
pi

Sequences of Operations & RENAME

- If we want to apply several ops one after the other
  - Can either write as a single expression (via nesting)
  - Or can apply one op at a time and save intermediate relations
- Example:
  - get {first name, last name, salary} of all employees in dept 5
  - $\pi_{LNAME,\ FNAME,\ SALARY}(\sigma_{DNO=5}\ (EMPLOYEE))$
    or
  - $DEP5\_EMPS = \sigma_{DNO=5}\ (EMPLOYEE)$
    $RESULT = \pi_{LNAME,\ FNAME,\ SALARY}(DEP5\_EMPS)$
- Can also use to rename attributes
  - Sometimes useful for UNION and JOIN as we'll see
  - $R(LASTNM,\ FIRSTNM,\ SALARY)= \pi_{LNAME,\ FNAME,\ SALARY}(TMP)$

Set Theoretic Ops

- UNION, INTERSECTION, SET DIFFERENCE
  - $\cup$, $\cap$, -
- Binary ops applied to two sets
- Relations must be *union compatible*
  - Have same degree n, and dom(Ai) = dom(Bi) for all 1<=i<=n
- Example:
  - Find SSN of all employees who work in dept 5 or supervise an employee in dept 5
  - DEP5_EMPS = $\sigma_{DNO=5}$ (EMPLOYEE)
  - RESULT1 = π_{SSN}(DEP5_EMPS)
  - RESULT2(SSN) = $\pi_{SUPERSSN}$(DEP5_EMPS)
  - RESULT = RESULT1 $\cup$ RESULT2

Cartesian Product

- Also called cross product or cross join (denoted by **×**)
- Combines tuples from 2 relations
  - Resulting relation has attributes of both original relations
- Commonly used followed by a SELECT
  - That matches attributes coming from both component relations
- Example:
  - For each female employee get a list of names of her dependents
  - FEMALE_EMPS = $\sigma_{SEX='F'}$ (EMPLOYEE)
  - EMPNAMES= $\pi_{FNAME,\ LNAME,\ SSN}$(FEMALE_EMPS)
  - EMP_DEPENDENTS = EMPNAMES × DEPENDENT
  - ACTUAL_DEPENDENTS = $\sigma_{SSN=ESSN}$ (EMP_DEPENDENTS)
  - RESULT= $\pi_{FNAME,\ LNAME,\ DEPENDENT\_NAME}$(ACTUAL_DEPENDENTS)

- Note: Cartesian product operation by itself doesn't make much sense, but it's an ingredient in JOINs (next slide)

JOIN Operation

- Useful to combined related tuples (denoted by $\bowtie$)
- Example:
  - Retrieve name of manager of each department
  - DEPT_MGR = DEPARTMENT $\bowtie_{MGRSSN=SSN}$ EMPLOYEE
  - RESULT = $\pi_{DNAME,\ LNAME,\ FNAME}$(DEP_MGR)
- Essentially does a Cartesian Product, then SELECT
  - General condition is: <cond> AND <cond> AND … AND <cond>
- Special case joins with specific names:
  - **Theta join:** When all cond are of form Ai θ Bj where Ai and Bj are attributes of R and S
  - **Equi join**: A Theta join where the operator is equality
  - **Natural join**: An Equi join where attributes Ai and Bj have the same name; automatically gets rid of second (superfluous) attribute