The Relational Model

Lecture 3

Outline

1. Context
2. Model Concepts
3. Relational Constraints
4. Update Operations
5. Transactions

Database Design and Implementation Process

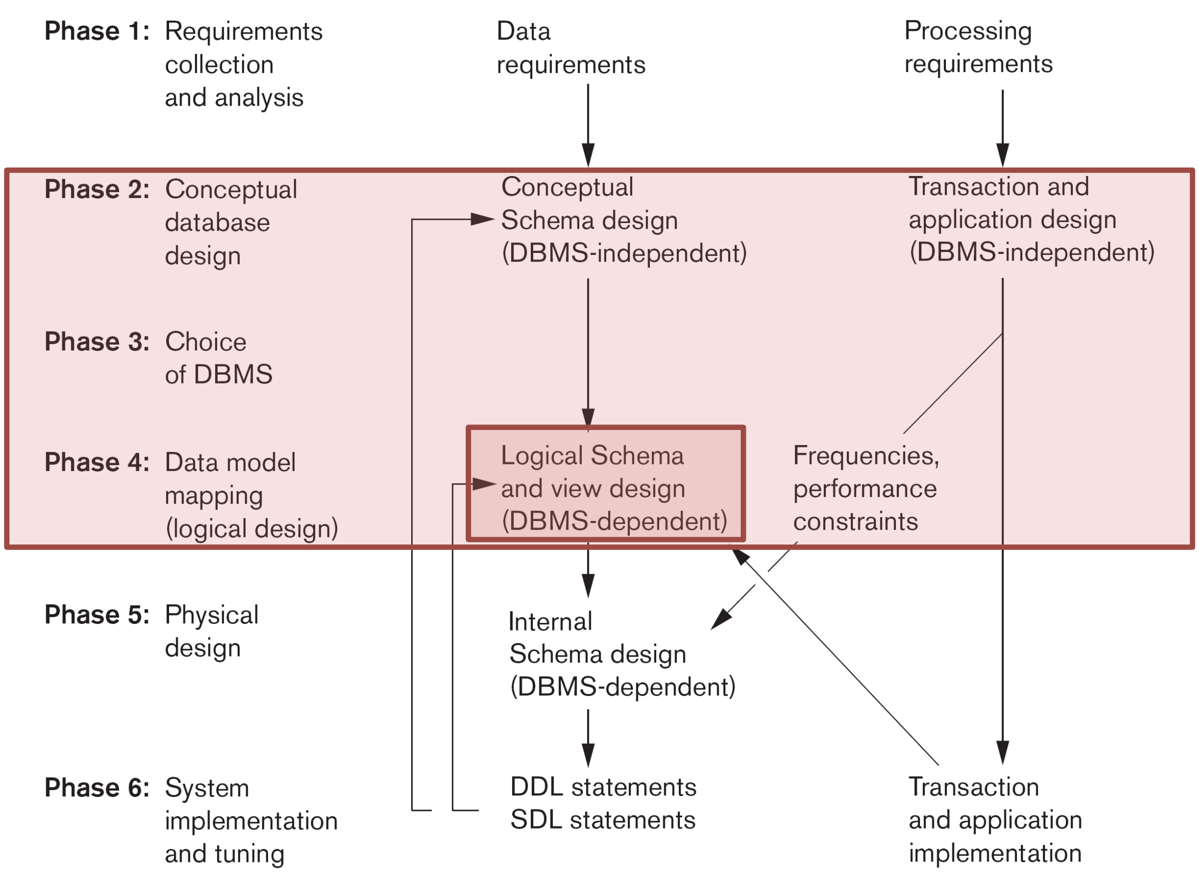
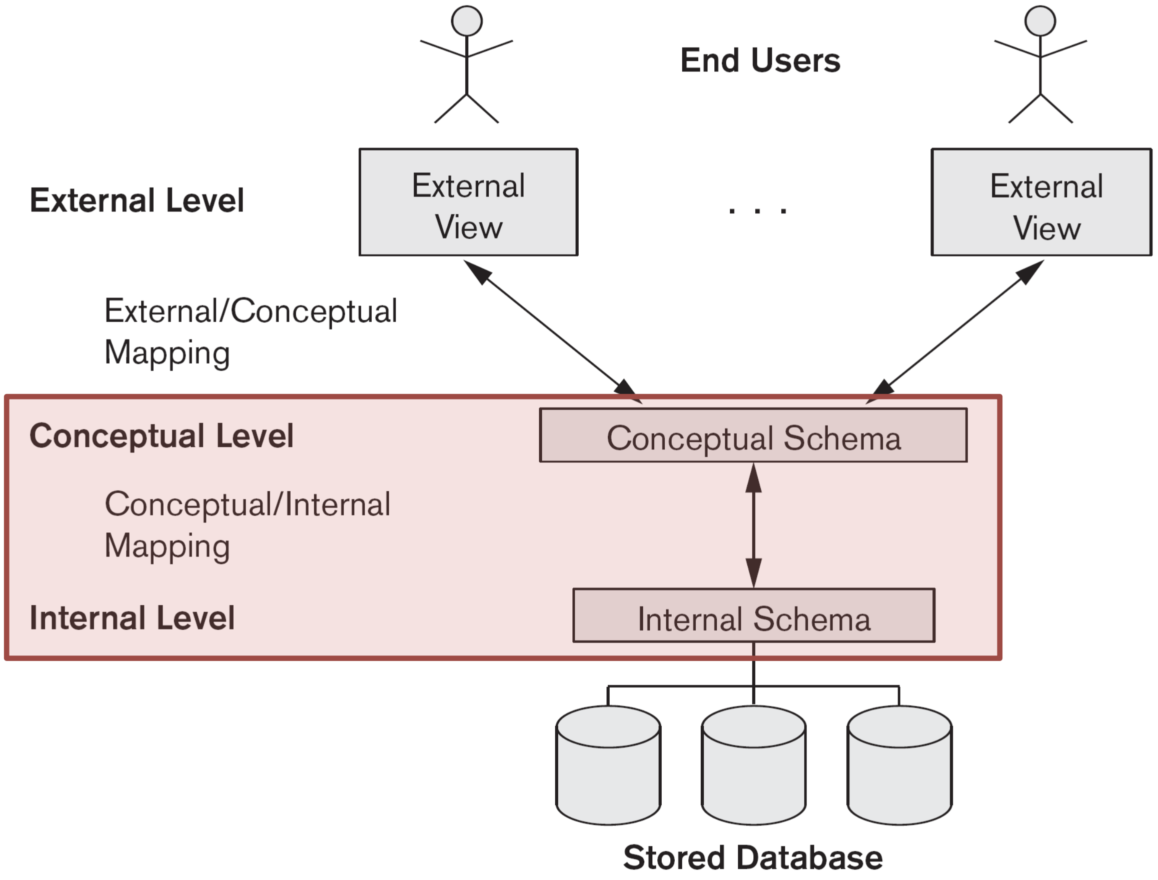

Data content, structure, and constraints
Database applications
Phase 1: Requirements collection and analysis
Data requirements
Processing requirements
Phase 2: Conceptual database design
Conceptual Schema design (DBMS-independent)
Transaction and application design (DBMS-independent)
Phase 3: Choice of DBMS
Phase 4: Data model mapping (logical design)
Logical Schema and view design (DBMS-dependent)
Frequencies, performance constraints
Phase 5: Physical design
Internal Schema design (DBMS-dependent)
Phase 6: System implementation and tuning
DDL statements
SDL statements
Transaction and application implementation

Figure 10.1
Phases of database design and implementation for large databases.

Data Models

Figure 2.2
The three-schema architecture.



End Users

External Level

External View

. . .

External View

External/Conceptual Mapping

Conceptual Level

Conceptual Schema

Conceptual/Internal Mapping

Internal Level

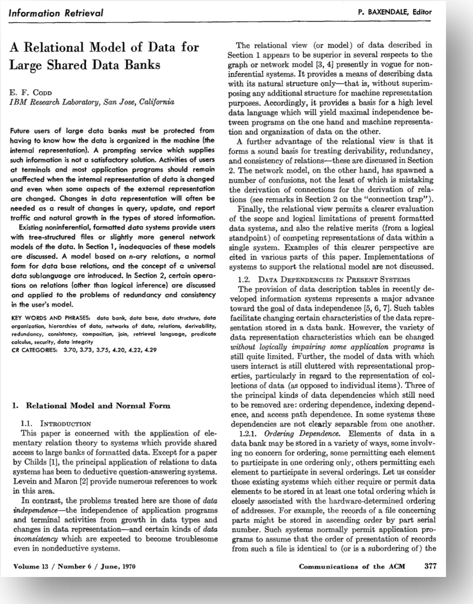
Internal Schema

Stored Database

The Relational Model

Codd, Edgar F. "A relational model of data for large shared data banks." *Communications of the ACM* 13.6 (**1970**): 377-387.

"Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation)… 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… In Section 1, inadequacies of [existing] 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."



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 non-inferential 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.

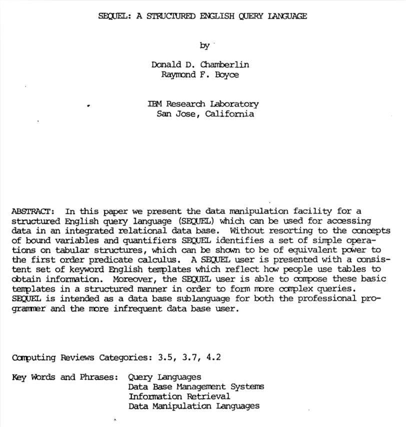
1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing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

Volume 13 / Number 6 / June, 1970 Communications of the ACM 377

FYI: SQL

Chamberlin, Donald D., and Raymond F. Boyce. "SEQUEL: A structured English query language." *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*. ACM, **1974**.

"In this paper we present the data manipulation facility for a structured English query language (SEQUEL) which can be used for accessing data in an integrated relational data base. Without resorting to the concepts of bound variables and quantifiers SEQUEL identifies a set of simple operations on tabular structures, which can be shown to be of equivalent power to the first order predicate calculus. A SEQUEL user is presented with a consistent set of keyword English templates which reflect how people use tables to obtain information. Moreover, the SEQUEL user is able to compose these basic templates in a structured manner in order to form more complex queries. SEQUEL is intended as a data base sublanguage for both the professional programmer and the more infrequent data base user."



SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE

by

Donald D. Chamberlin
Raymond F. Boyce

IBM Research Laboratory
San Jose, California

ABSTRACT: In this paper we present the data manipulation facility for a structured English query language (SEQUEL) which can be used for accessing data in an integrated relational data base. Without resorting to the concepts of bound variables and quantifiers SEQUEL identifies a set of simple operations on tabular structures, which can be shown to be of equivalent power to the first order predicate calculus. A SEQUEL user is presented with a consistent set of keyword English templates which reflect how people use tables to obtain information. Moreover, the SEQUEL user is able to compose these basic templates in a structured manner in order to form more complex queries. SEQUEL is intended as a data base sublanguage for both the professional programmer and the more infrequent data base user.

Computing Reviews Categories: 3.5, 3.7, 4.2

Key Words and Phrases: Query Languages
Data Base Management Systems
Information Retrieval
Data Manipulation Languages

Motivation

- A **declarative** method for specifying data and queries
- A **formal** mathematical basis for database systems
- A foundation for efficient and usable database systems

Model Concepts

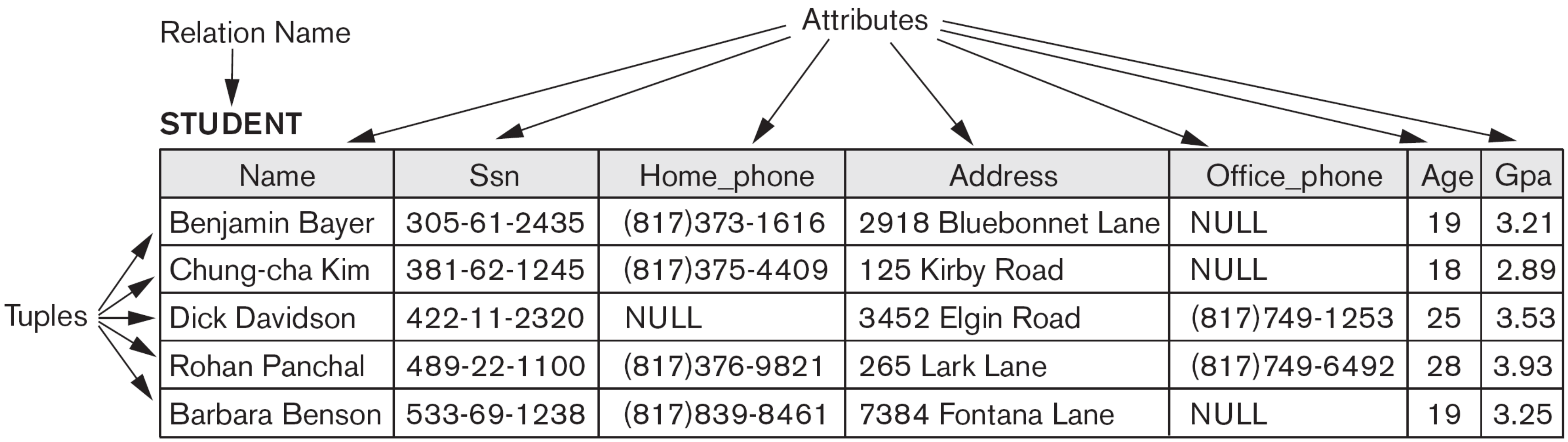
- A database is a set of named **relations** (tables) and a set of **integrity constraints**
  - Database is in a **valid state** if it satisfies all integrity constraints (else **invalid state**)

- The schema of an *n*-ary relation is an ordered list of n **attributes** (columns)
  - Mathematically equivalent as a set

- Each attribute has a **domain** (type) of atomic values
  - Related to the 1NF assumption

- The state of the relation is a set of ***n*-tuples** (rows), each an ordered list of values in the corresponding domain, or **NULL**
  - Mathematically a subset of the Cartesian product of the attribute domains; related to the closed-world assumption
  - Actual implementations loosen the definition to a *bag* of tuples (and hence allow duplicate rows, more later)

Example Relation


Relation Name
Attributes
Tuples

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Figure 3.1
The attributes and tuples of a relation STUDENT.

STUDENT(Name, Ssn, Home_phone, Address, Office_phone, Age, Gpa)

dom(Name) = Names
dom(Ssn) = Social_security_numbers
…

Ordering of Tuples

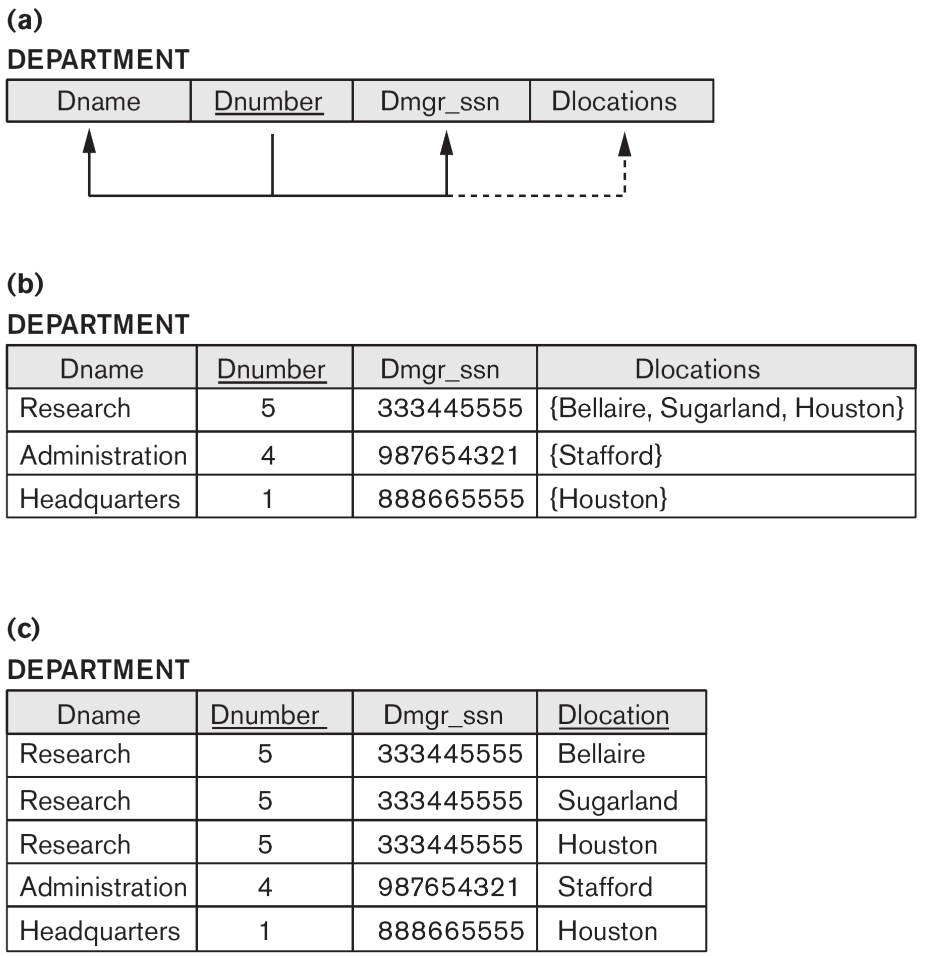
- A relation is formally defined as a set of tuples; thus, there is no inherent order
- The physical representation *will* have an ordering, but the relation definition sets no preference as to this ordering
- As we will discuss later in physical design, indexes may establish an ordering for purposes of query efficiency

Values in Tuples

- Each value must be atomic – no composite or multi-valued attributes
  - Composite: simple component attributes
  - Multi-valued: separate relations

- NULL
  - Several possible meanings (e.g. unknown, not available, does not apply, undefined)
  - Best to avoid, else deal with caution (esp. during queries with filtration/aggregation)

Violation of Atomic (1)



(b)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

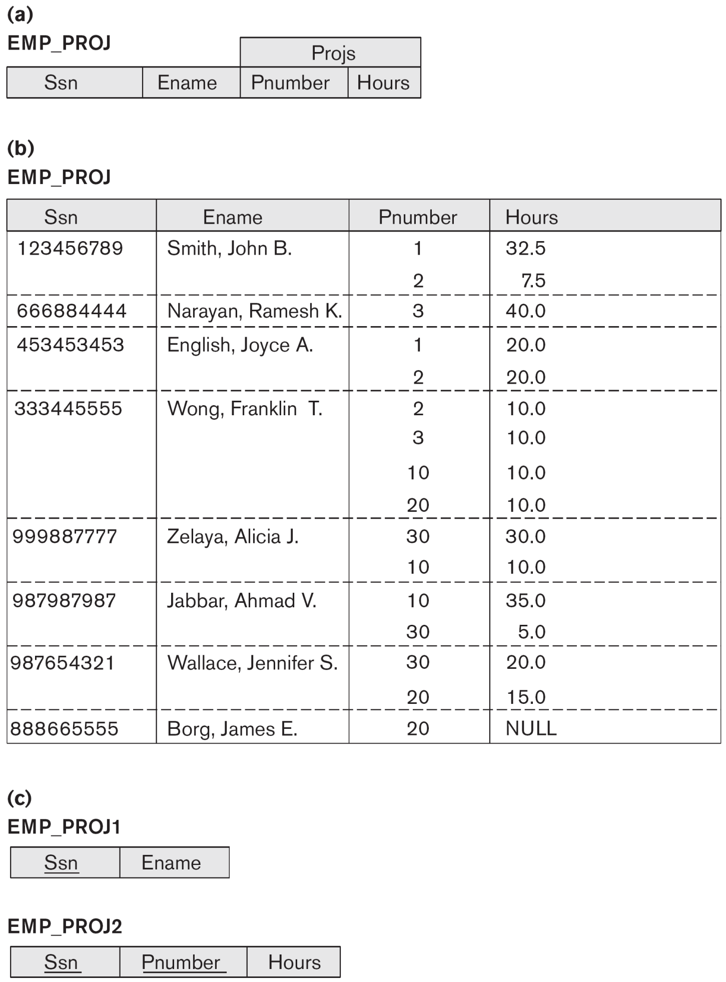
(c)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Figure 15.9
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Sample state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

Violation of Atomic (2)



(a)

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

(b)

EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1 2	32.5 7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1 2	20.0 20.0
333445555	Wong, Franklin T.	2 3 10 20	10.0 10.0 10.0 10.0
999887777	Zelaya, Alicia J.	30 10	30.0 10.0
987987987	Jabbar, Ahmad V.	10 30	35.0 5.0
987654321	Wallace, Jennifer S.	30 20	20.0 15.0
888665555	Borg, James E.	20	NULL

(c)

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

Figure 15.10
Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a *nested relation* attribute PROJS. (b) Sample extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

Types of Relational Constraints

Categories of restrictions on data that can be specified on a relational database:

1. Inherent in the data model (implicit)
2. **Schema-based (explicit)**
3. Application-based (or trigger/assertions)
4. Data dependencies

   Relates to “goodness” of database design; we will revisit in *normalization*

Schema-Based Constraints

- Domain constraints
- Key constraints
- Constraints on NULLs
- Entity integrity
- Referential integrity

Domain Constraints

- Within each tuple, the value of each attribute *A* must be an atomic value from the domain dom(*A*)
- More later on standard data types in SQL

Keys

A **key** is a set of attributes that satisfies two properties:

1. Two distinct tuples in any state of the relation cannot have identical values for all the attributes of the key (termed *superkey*)
2. We cannot remove any attributes from the key and still have the uniqueness constraint hold (termed *minimal superkey*)

A relation may have multiple keys (each is a **candidate key**). Relations commonly have a **primary key** (underlined; typically small number of attributes, used to *identify* tuples), and may also have some number of additional **unique keys**.

Constraints on NULLs

- Schema must dictate whether or not a NULL value is allowed for each attribute
- No primary key value can be NULL (**entity integrity constraint**)

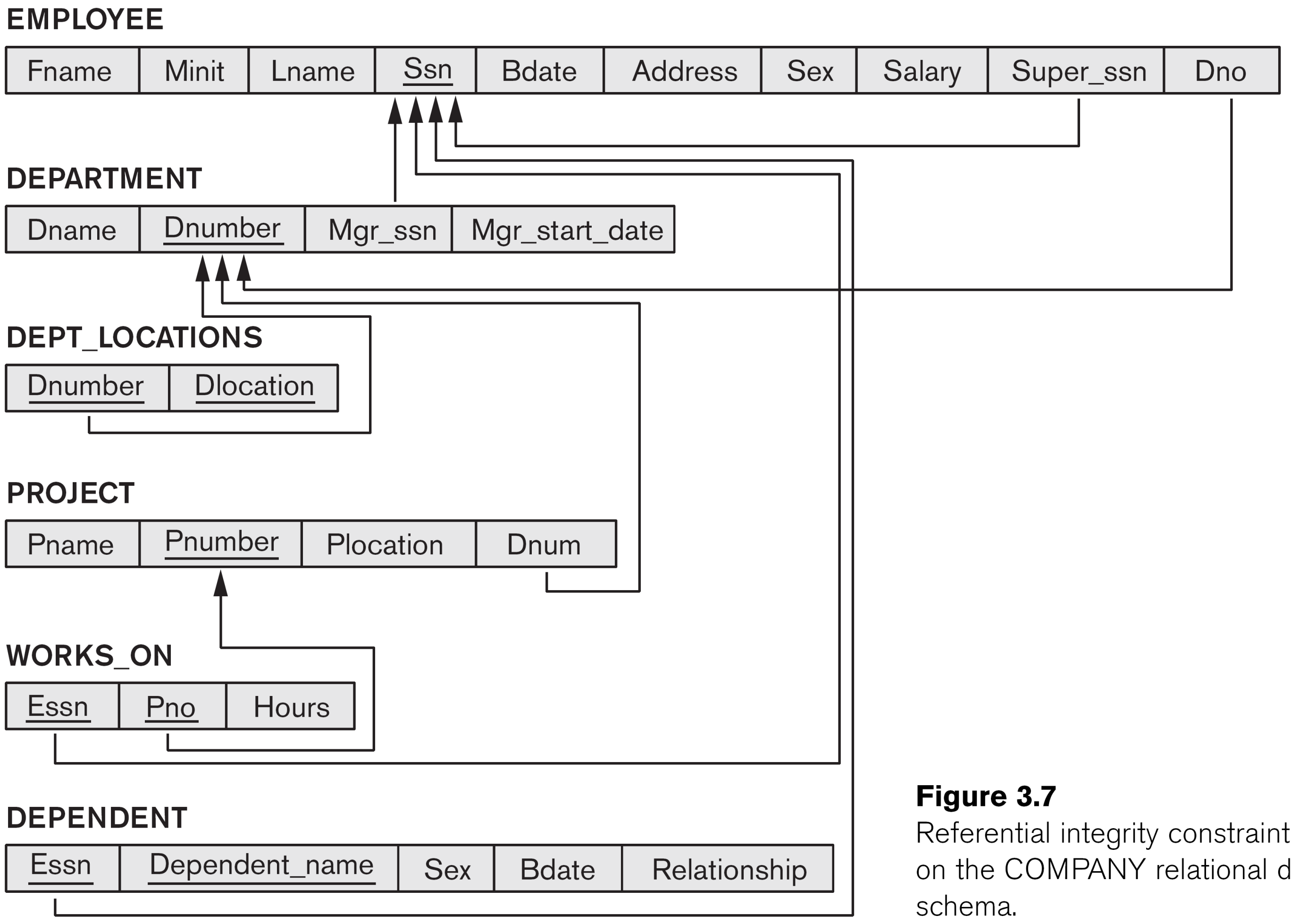
Referential Integrity

All tuples in one relation must refer to an *existing* tuple in some relation (or NULL); indicated by directed arc

A **foreign key** (FK) in R1 *references* R2 if…

1. The attributes in FK have the same domain(s) as the primary key attribute(s) PK of R2
2. A value of FK in a tuple t_1 either is NULL or occurs as a value of PK for some tuple t2 (t_1 **refers to** t_2)

COMPANY



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Figure 3.7
Referential integrity constraints displayed on the COMPANY relational database schema.

Update Operations

- Insert
- Delete
- Update

We now examine how these can violate various types of constraints and the resulting actions that can be taken.

Insert

- Domain
  - An attribute value does not appear in the corresponding domain (including NULL)
- Key
  - A key value already exists in another tuple
- Entity
  - Any part of the primary key is NULL
- Referential Integrity
  - Any value of any foreign key refers to a tuple that does not exist in the referenced relation

Typical action: reject insertion

Delete

- Referential Integrity
  - Tuple being deleted is referenced by foreign keys from other tuples

Possible actions:

- Reject deletion
- Cascade (propagate deletion)
- Set default/NULL referencing attribute values (careful with primary key)

Update

- If neither part of primary key nor foreign key, need only check…
  - Domain
- Modifying primary key…
  - Like Delete + Insert
- Modifying foreign key…
  - Like Insert

Actions typically similar to Delete with separate options.

Transaction

- Sequence of database operations, including retrieval and update(s)

- Treated as atomic unit of work
  - Must leave the database in a valid state