Advanced Database Management Systems

Lecture 4 – Chapter 5
The Relational Data Model

Edgar F. Codd

Information Retrieval

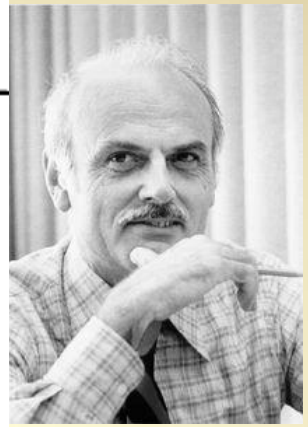
A Relational Model of Data for Large Shared Data Banks

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

Communications of the ACM 13[6] June 1970



Ted Codd proposed the ***relational data model*** in 1970.

He received the ACM Turing Award in 1981.

Relational Data Model

- Core of majority of modern databases
- Virtually all business relies on some form of relational database
- Solid theoretical/mathematical foundation
- Simple but robust implementation

Models, Schemas and States

- A *data model* defines the constructs available for defining a schema
  - defines possible schemas
- A *schema* defines the constructs available for storing the data
  - defines database structure
  - limits the possible database states
- A *database state* (or *instance*) is all the data at some point in time
  - the database content

Models, Schemas and States

- *data model*
  - fixed by the DBMS
- *schema*
  - defined by the DB designer
  - generally fixed once defined *
- *database state*
  - changes over time due to user updates

* schema modifications are possible once the database is populated, but this generally causes difficulties

The Relational Data Model

- All data is stored in relations
  - relations are sets, but generally viewed as 2D tables
- DB schema = a set of relation specifications
  - the specification of a particular relation is called a *relation schema*
- DB state = the data stored in the relations
  - the data in a particular relation is called a *relation state* (or *relation instance* or simply *relation*)

***Principle of Uniform Representation*:**
The entire content of a relational database is represented in one and only one way: namely, as attribute values within tuples within relations.

RDM Schemas

External View

External View

External View

relation specifications

Conceptual Schema

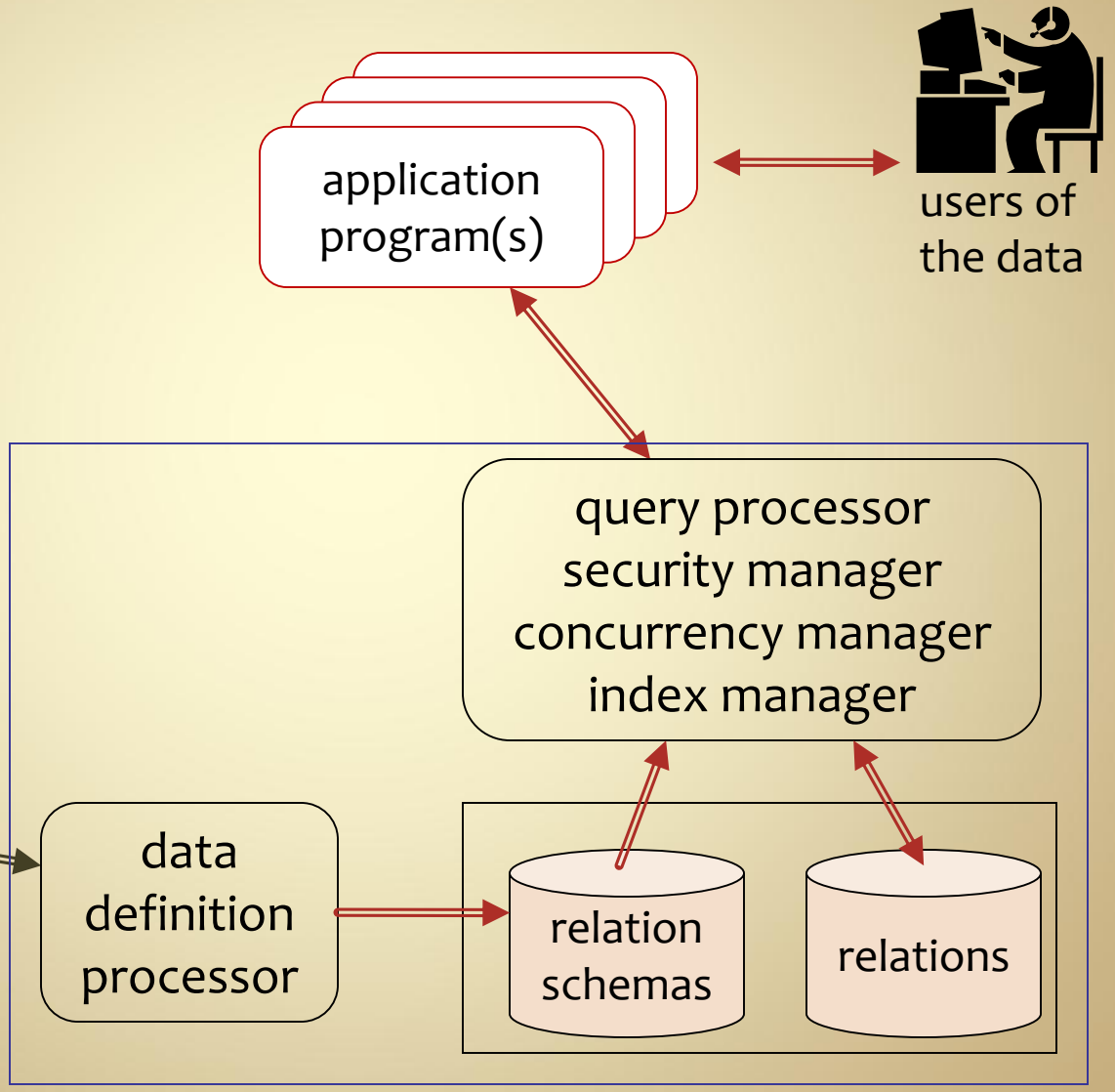
Internal Schema

mapping from relations to storage layout (files)

Relational Data Definition

database designer enters the definition of relation schemas

SQL DDL = relation definition language (CREATE TABLE)



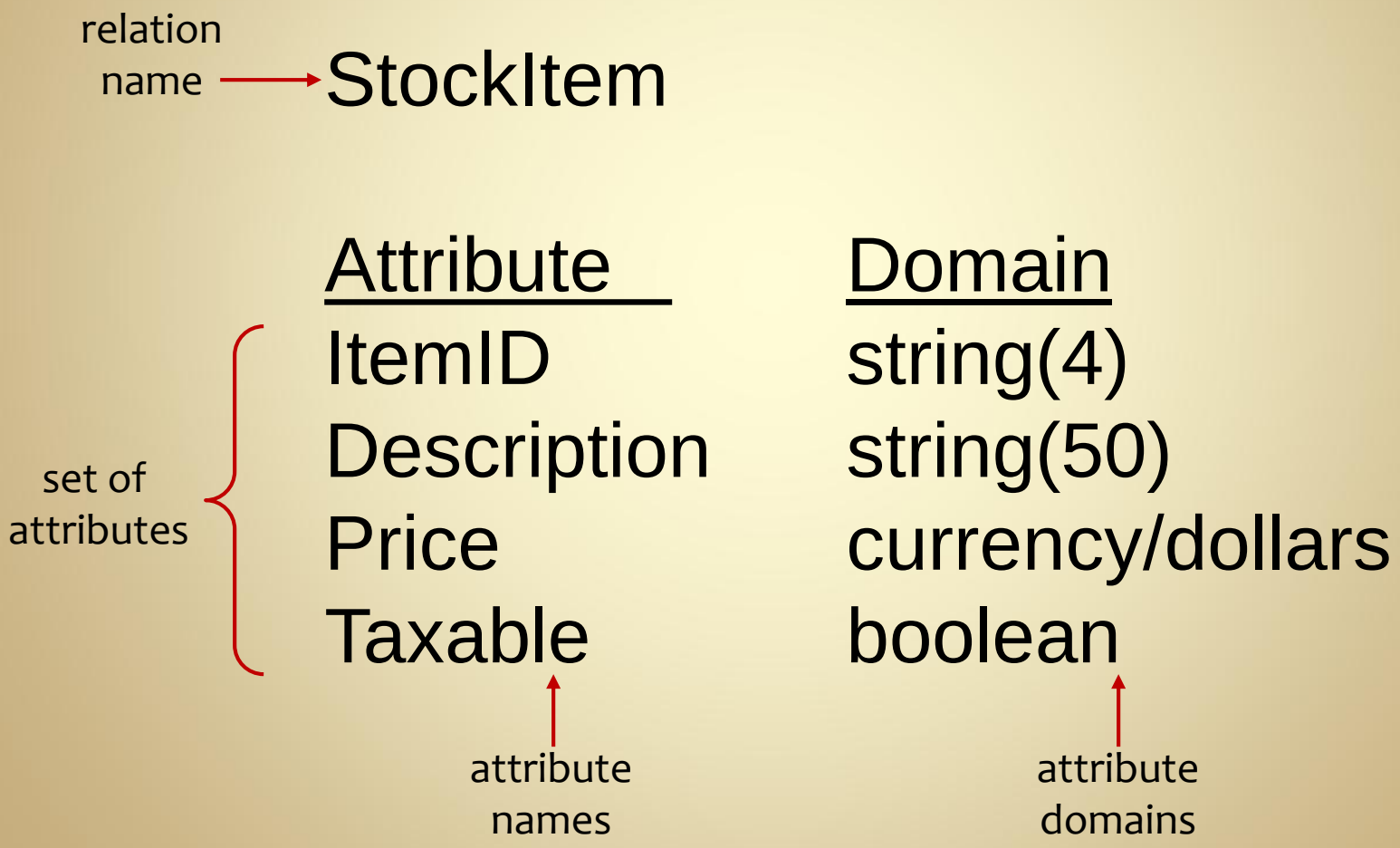
Relation Schemas and Relation Instances

Relation Schemas

- A relation is defined by
  a *name* and
  a *set of attributes*

- Each attribute has a *name* and a *domain*
  - a domain is a set of possible values
  - *types* are domain names
  - all domains are sets of atomic values –
    RDM does not allow complex data types
  - domains may contain a special **null** value

Example Relation Schema



Definition: Relation Schema

- Relation Schema

$$R(A_1, A_2, \ldots, A_n)$$

  - R is the relation name
  - $A_1 \ldots A_n$ are the attribute names

- Domains are denoted by

$$dom(A_i)$$

- degree = the number of attributes

Example Relation Schema

STOCKITEM(ItemId, Description, Price, Taxable)

dom(ItemId) = string(4)
dom(Description) = string(50)
dom(Price) = currency/dollars
dom(Taxable) = boolean

degree of STOCKITEM = 4

Definition: Relation

- A relation is denoted by

$$r(R)$$

  - R is the name of the relation schema for the relation

- A relation is a set of tuples

$$r(R) = (t_1, t_2, \ldots , t_m)$$

Definition: Relation

- Each tuple is an ordered list of n values

  $t = < v_1, v_2, \ldots , v_n >$

  - n is the degree of R

- Each value in the tuple must be in the domain of the corresponding attribute

  $v_i \in dom(A_i)$

- Alternate notations:

  i^{th} value of tuple t is also referred to as

  $v_i = t[A_i]$ or $v_i = t.A_i$

Example Relation

r(STOCKITEM) =
{ < I119, "Monopoly", $29.95, true >,
  < I007, "Risk", $25.45, true >,
  < I801, "Bazooka Gum", $0.25, false > }

t_2 = < I007, "Risk", $25.45, true >

t_2[Price] = t_2.Price = $25.45

t_2[Price] ϵ dom(Price) = currency/dollars

Characteristics of Relations

- A relation is a *set*
  - tuples are unordered
  - no duplicate tuples
- Attribute values within tuples are ordered
  - values are matched to attributes by position
  - alternate definition defines a tuple as a set of (name,value) pairs, which makes ordering of tuple unnecessary (we won't use this definition)

Characteristics of Relations

- Values in tuples are *atomic*
  - atomic = non-structured
    (similar to primitive types in C++)
  - implication:
    no nested relations or other complex data structures
- If domain includes *null* values,
  null may have many interpretations
  - "does not exist"
  - "not applicable"
  - "unknown"

Theory vs. Reality

- The theoretical data model is mathematical:
  - a relation is a ***set* of tuples**
  - this is Codd's definition

- In the real-world, the model is practical:
  - efficiency concerns
  - excepted standard: SQL
  - a relation is a table, not a set
  - a relation may have order and duplicates

SQL: Relation States

- A relation is viewed as a table
- The attributes define the columns of the table
- Each row in the table holds related values for each attribute
  - a row often represents a conceptual entity (object)
- Values in each column must come from the domain of the attribute
  - the values are instances of the attribute type

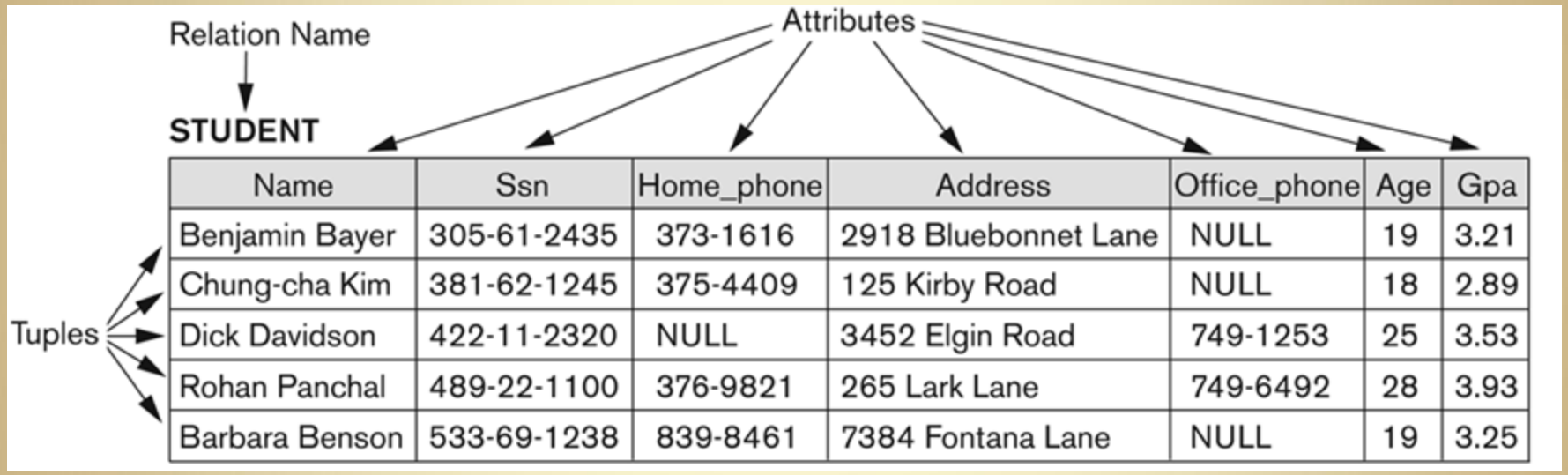
Relation: Table Representation

Each row collects related attribute values

StockItem			
ItemId	**Description**	**Price**	**Taxable**
I119	Monopoly	$29.95	True
I007	Risk	$25.45	True
I801	Bazooka Gum	$0.25	False

Column values all come from the same domain

Example Relation



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

Example Schema

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

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Example State

Constraints

Constraints

- Constraints are restrictions on legal relation states
  - they add further semantics to the schema
- Domain constraints $v_i \in dom(A_i)$
  - values for an attribute must be from the domain associated with the attribute
- Non-null constraints
  - the domain of some attributes may not include null, implying that a value for that attribute is required for all tuples

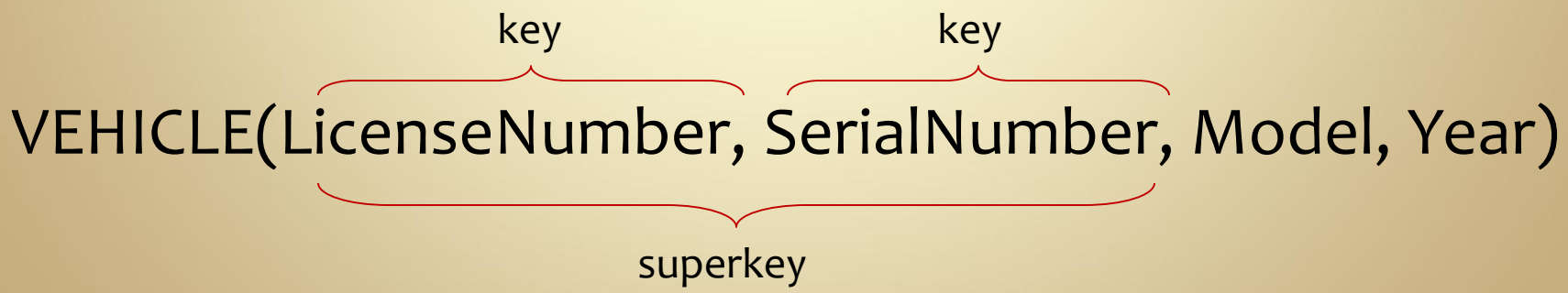
Key Constraints

- By definition, all tuples in a relation are unique
- Often, we want to restrict tuples further such that some subset of the attributes is unique for all tuples
- Example: in the StockItem relation, no ItemID should appear in more than one tuple
  - ItemID is called a **key** attribute

Keys and Superkeys

- Any subset of attributes
  that must be unique is called a *superkey*
- If no subset of the attributes of a superkey
  must also be unique,
  then that superkey is called a *key*

- Example:

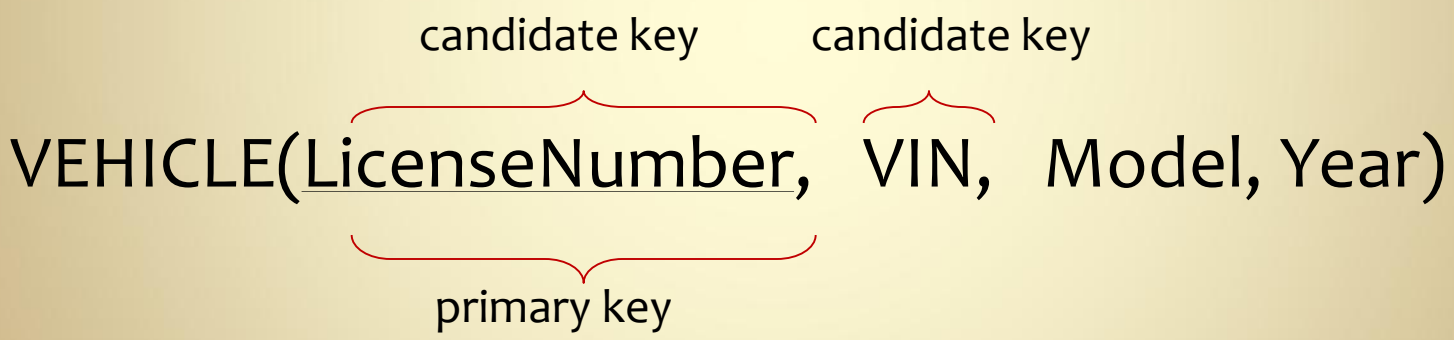


Candidate and Primary Keys

- If a relation has more than one key, each key is called a *candidate key*
- One candidate key must be chosen to be the *primary key*
- The primary key is the one that will be used to *identify* tuples
- If there is only one key, it is the primary key

Candidate and Primary Keys

- Primary keys are indicated
  by underlining the attributes that make up that key



Example Keys

STOCKITEM(ItemId, Description, Price, Taxable)

superkeys:

(ItemId), (Description), (ItemId, Description)

keys:

(ItemId), (Description)

candidate keys:

(ItemId), (Description)

primary key:

(ItemId)

(assumes that Description is unique for all items)

Integrity Constraints

- Entity integrity constraint
  - no primary key value can be null
  - the primary key is the tuple identifier
- Referential integrity constraint
  - references between relations must be valid
  - the *foreign key* of a referencing relation must exist as a primary key in the referenced relation

Example: Referential Integrity

STOCKITEM(ItemId, Description, Price, Taxable)
STORESTOCK(StoreId, Item, Quantity)

STORESTOCK[Item] *refers to* STOCKITEM[ItemID]

STORESTOCK[Item] is a *foreign key* referencing
the primary key STOCKITEM[ItemID]

Any value appearing in STORESTOCK[Item]
must appear in STOCKITEM[ItemID]

It must be true that
dom(STORESTOCK[Item]) = dom(STOCKITEM[ItemID])

Referential Integrity

- PK = primary key in R_2
- FK = foreign key in R_1
- $dom(R_1[FK]) = dom(R_2[PK])$
- constraint:

$$\text{if } v \in R_1[FK] \text{ then } v \in R_2[PK]$$

- note: FK is not necessarily a key of R_1

Example: Referential Integrity

STOCKITEM(ItemId, Description, Price, Taxable)
STORESTOCK(StoreId, Item, Quantity)
STORE(StoreID, Manager, Address, Phone)

- (StoreId, Item) is the primary key of STORESTOCK
- STORESTOCK[StoreId] is a foreign key referencing STORE
- STORESTOCK[Item] is a foreign key referencing STOCKITEM

Referential Integrity: Diagrammatic Representation

STOCKITEM(ItemId, Description, Price, Taxable)

PK

FK

STORESTOCK(StoreId, Item, Quantity)

FK

PK

STORE(StoreID, Manager, Address, Phone)

Referential Integrity: Textual Representation

STOCKITEM(ItemId, Description, Price, Taxable)
STORESTOCK(StoreId, Item, Quantity)
STORE(StoreID, Manager, Address, Phone)

constraints:
STORESTOCK[StoreId] refers to STORE[StoreID]
STORESTOCK[Item] refers to STOCKITEM[ItemId]

Referential Integrity: Example State

r(STORESTOCK) =

{ < "S002", "I065", 120 >,
< "S047", "I954", 300 >,
< "S002", "I954", 198 > }

StoreId is a foreign key but not a key

all values in FK exist in PK

r(STORE) =

{ < "S002", "Tom", "112 Main", "999-8888" >,
< "S047", "Sasha", "13 Pine", "777-6543" > }

Referential Integrity: Constraint Violation

r(STORESTOCK) =

{ < "S002", "I065", 120 >,
< "S047", "I954", 300 >,
< "S333", "I954", 198 > }

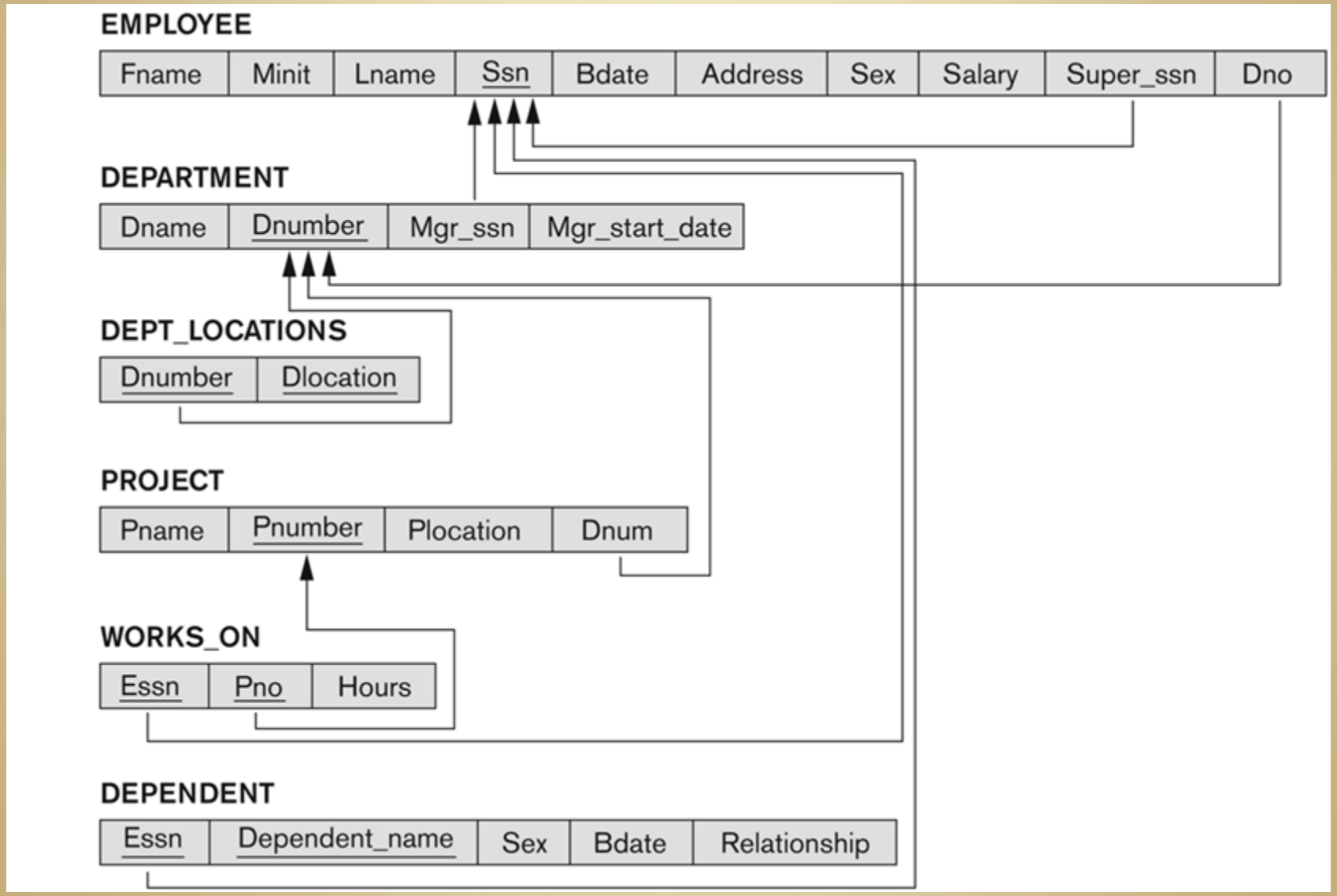
StoreId "S333" does not exist in PK:
this is an illegal database state

r(STORE) =

{ < "S002", "Tom", "112 Main", "999-8888" >,
< "S047", "Sasha", "13 Pine", "777-6543" > }

Both relation states are legal, but the database state is illegal.

Schema with FKs



State Change and Constraint Enforcement

Causes of Constraint Violations

- What can cause a referential integrity constraint violation?
  - inserting a tuple in R_1 with an illegal FK
  - modifying a tuple in R_1 to have an illegal FK
  - deleting a tuple in R_2 that had the PK referenced by some FK in R_1
- How can a referential integrity constraint be enforced?
  - reject the operation that attempts to violate it (may cause other operations to be rejected … transactions)
    or
  - repair the violation, by cascading inserts or deletes

Data Manipulation Operations

There are three ways to modify the value of a relation:

- Insert: add a new tuple to R
- Delete: remove an existing tuple from R
- Update: change the value of an existing tuple in R

Delete and Update both require some way to identify an existing tuple (a selection)

Inserting Tuples

r_1(STORESTOCK) = { < "S002", "I065", 120 >, < "S047", "I954", 300 >, < "S333", "I954", 198 > }

insert < "S047", "I099", 267 >

r_2(STORESTOCK) = { < "S002", "I065", 120 >, < "S333", "I954", 198 >, < "S047", "I099", 267 >, < "S047", "I954", 300 > }

any constraint violations?

Deleting Tuples

r_2(STORESTOCK) = { < "S002", "I065", 120 >, < "S333", "I954", 198 >, < "S047", "I099", 267 >, < "S047", "I954", 300 > }

delete tuples with Item = "I954"

r_3(STORESTOCK) = { < "S002", "I065", 120 >, < "S047", "I099", 267 > }

Updating Tuples

r_3(STORESTOCK) = { < "S002", "I065", 120 >, < "S047", "I099", 267 > }

change the **Quantity** of tuples
with **StoreID** = "S002" and **Item** = "I954" to 250

r_3(STORESTOCK) = { < "S002", "I065", 250 >, < "S047", "I099", 267 > }

Analyzing State Changes

- Any update can be viewed as (delete and insert)

update: < "S002", "I065", 120 > to < "S002", "I065", 250 >

is equivalent to

delete: < "S002", "I065", 120 >

insert: < "S002", "I065", 250 >

- Any database state change can be viewed as a set of deletes and inserts on individual relations

- This makes the analysis of potential constraint violations a well defined problem

Enforcing Constraints

- *constraint enforcement*:
  ensuring that no invalid database states can exist

- *invalid state*: a state in which a constraint is violated

- Possible ways to enforce constraints:
  - reject any operation that causes a violation, or
  - allow the violating operation and then attempt to correct the database

Constraint Violating Operations

- To automate constraint enforcement the operations that can cause violations need to be identified

	insert	delete	update
domain, non-null	yes	no	yes
key	yes	no	yes
entity integrity	yes	no	yes
referential integrity	yes/FK	yes/PK	yes/FK/PK

Correcting Constraint Violations

violation	correction
domain, non-null	ask user to enter a valid value or use a default value
key	ask user to enter a unique key or generate a unique key
entity integrity	ask user to enter a unique key or generate a unique key
referential integrity FK insertion	force an insert in the PK (may cascade)
referential integrity PK deletion	propagate delete to FK (may cascade)

Summary: Relational Schemas

- A relational schema consists of
  a set of relation schemas
  and a set of constraints

- Relation schema
  - list of attributes: name and domain constraint
  - superkeys: key constraints
  - primary key: entity integrity constraint

- Foreign keys: referential integrity constraints
  - defined between relation schemas

AIRPORT

Airport_code	Name	City	State

FLIGHT

Flight_number	Airline	Weekdays

FLIGHT_LEG

Flight_number	Leg_number	Departure_airport_code	Scheduled_departure_time
		Arrival_airport_code	Scheduled_arrival_time

LEG_INSTANCE

Flight_number	Leg_number	Date	Number_of_available_seats	Airplane_id
Departure_airport_code	Departure_time	Arrival_airport_code	Arrival_time	

FARE

Flight_number	Fare_code	Amount	Restrictions

AIRPLANE_TYPE

Airplane_type_name	Max_seats	Company

CAN_LAND

Airplane_type_name	Airport_code

AIRPLANE

Airplane_id	Total_number_of_seats	Airplane_type

SEAT_RESERVATION

Flight_number	Leg_number	Date	Seat_number	Customer_name	Customer_phone

Schema for Airline Database

NEXT UP

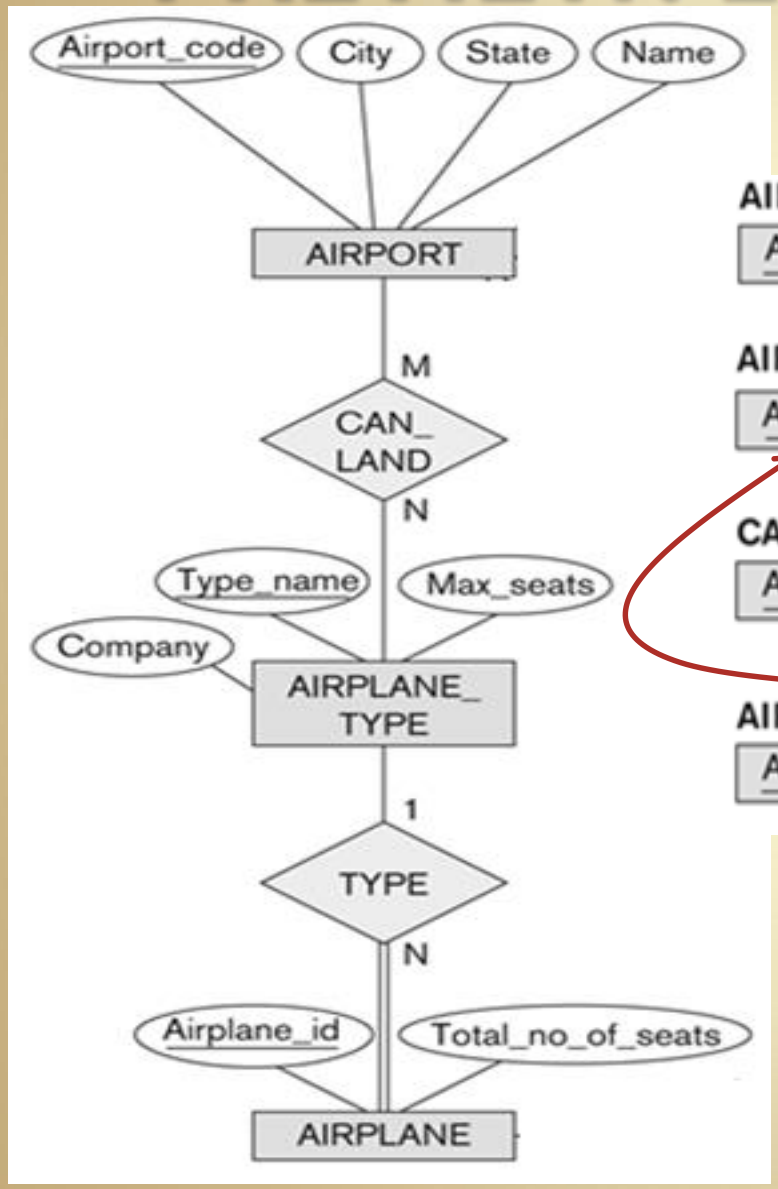
- skip ahead to Chapter 7:

  Translating ER Schemas to Relational Schemas

- then back to Chapter 6:

  The Relational Algebra: operations on relations

PREVIEW: ER to Relational



AIRPORT

Airport_code	Name	City	State

AIRPLANE_TYPE

Airplane_type_name	Max_seats	Company

CAN_LAND

Airplane_type_name	Airport_code

AIRPLANE

Airplane_id	Total_number_of_seats	Airplane_type