

# The Relational Data Model

SWEN 304  
Trimester 2, 2017

Lecturer: Dr Hui Ma  
**Engineering and Computer Science**



# Outline

- Basic terms and concepts:
  - Relation Schema, Attribute, Domain
  - Relation, Tuple
  - Relational database schema
  - Relational database instance
- Relational integrity constraints
  - Domain constraints, Attribute constraints
  - Key constraint, Unique constraint
  - Interrelation constraints
- Constraint violation: database updates
- Reading: Chapter of the Relational Data Model

# Pre-Relational Database Systems

- Network and hierarchical database systems
  - Emerged in late sixties of the twentieth century
- Deficiencies of the **network** and **hierarchical** database systems:
  - **Complex** data structures (hence hard to comprehend and use),
  - No **separation** between logical and physical data description (hence program data dependency)
  - **Navigational** programming languages (low programming productivity)

# The Relational Model of Data

- Introduced in 1970 by E. F. Codd
- Provides a very simple way of storing, manipulating and retrieving information
- The relational data model (RDM) represents the database as a collection of relations
- Finite **relations** in the mathematical sense are sets of tuples (or records)
- Well-defined concepts and easy to understand
- Clear separation of the (syntactical) schema level and the (semantic) instance level

# The Relational Model of Data

- The use of relations enables a purely logical treatment of data management tasks
- The use of relations enables physical data independence
  - All physical structure concepts (storage extents, pointers, entry point records, hashing algorithms, access tree structures etc.) are hidden from users and programmers
- **Declarative** language for database querying and updating
- The RDM is the de facto standard for commercial database systems

# Relation Schema

- A **Relation Schema** is denoted by  $N(A_1, A_2, \dots, A_n)$ 
  - $N$  is the **name** of the relation
  - $A_1, A_2, \dots, A_n$  are the **attributes** of the relation
- Each attribute has a **domain**  $D$  or a set of valid values
- The **degree** (or **arity**) of the relation is the number of attributes  $n$  of relation schema  $N$
- **Example:** SUPPLIER (Supplier\_no, Name, Address)
  - SUPPLIER is the relation name
  - Defined over the three attributes: Supplier\_no, Name, Address, i.e.  $n=3$

# Attribute

- A **property** of a set of similar UoD objects, e.g.
  - Id, Fname, Dept, (semantically defined attributes)
  - $A, B, \dots, X, Y$  (semantically un-interpreted attributes)
- The attribute name is used to interpret the meaning of the data elements corresponding to that attribute
- Some notational conventions
  - {Fname + Lname, Fname + Major } instead of {{Fname, Lname }, {Fname, Major }}

# Domain

- Domain is a set of values, e.g. *STRING*, *DATE*
- It can be defined by **type specification**

$$D = \{d_i \mid i = 1, \dots, n\}$$

with  $D$  as domain name and  $d_i$  as a domain element that satisfies a constraint

- Example:

$$CourseIdDom = \{'SWEN304', 'MATH114', 'STAT193', \dots\}$$

// set of character strings starting by four capital letters followed by three digits,

- There is a domain  $D$  associated with each attribute  $A$ , denoted by  $dom(A) = D$
- Example:  $dom(Lname) = STRING$ ,

$$dom(Course\_id) = CourseIdDom$$



# Tuple

- A **tuple**  $t$  over a relation schema  $N(A_1, A_2, \dots, A_n)$  is an ordered list of values, denoted
  - $t = \langle v_1, \dots, v_n \rangle$       or       $t = (v_1, \dots, v_n)$
- Each value is derived from an appropriate *domain* or is a *null* value ( $\omega$ ).
- Example: a row in the SUPPLIER relation
  - $t = \langle 247, \text{'Feed The Crowds'}, \text{'Bumpytown'} \rangle$
  - this is called a 3-tuple as it has 3 values
  - a tuple (row) in the SUPPLIER relation.
- Tuple  $t$  is also sometime represented as
  - $t = \{(A_1, v_1), \dots, (A_n, v_n)\}$

with  $(A_n, v_n)$  as (attribute, value) pairs

# Relation

- Let  $R = \{A_1, \dots, A_n\}$  be a set of attributes and  $\text{dom}(A_i) = D_i, i = 1, \dots, n$ ,
- a **relation**  $r$  over  $R$  is a finite set of ( $n$ -)tuples  $t_i$ ,

$$r = \{t_1, \dots, t_n\}$$

STUDENT			
Id	Lname	Fname	Major
300111	Smith	Susan	COMP
300121	Bond	James	MATH
300132	Smith	Susan	COMP

$t$  —

- It is common to use table notation for relations,
  - the attributes of  $R$  correspond to the column heads
  - the  $n$ -tuple correspond to the rows
  - the order of the rows in such a table is not important

# Relation Schema and Its Instances

- A **relation** is an instance of the relation schema  $N(A_1, \dots, A_n)$ , denoted by  $r(N)$ , or simply  $r$  if it satisfies all constraints of  $N$
- A **relational variable**  $\rho(N)$  of the type  $N$  is the place holder of relation  $r(N)$ 
  - The relational variable  $\rho(N)$  (denoted in sequel and SQL simply by  $N$ ) contains an instance of the relation schema  $N$  in each moment of time
  - It is the **current** instance of our relation schema  $N(A_1, \dots, A_n)$  in the database

# Relation Schema, Variable, and Instances

## ■ Relation Schema:

STUDENT(Lname, Fname, Id, Major)

- $dom(Lname) = STRING, dom(Fname) = STRING$
- $dom(Id) = STRING, dom(Major) = STRING$

## ■ Instances:

$\rho(STUDENT)$ at time 1			
Id	Lname	Fname	Major
300111	Smith	Susan	COMP
300121	Bond	James	MATH
300132	Smith	Susan	COMP

$\rho(STUDENT)$ at time 2			
Id	Lname	Fname	Major
300111	Smith	Susan	COMP
300121	Bond	James	MATH
300132	Smith	Susan	COMP
300135	John	Cecil	MATH

# Questions

1. Suppose you are given a set of tuples,  $\{t_1, t_2, t_3\}$ , where each  $t_i$  is a tuple over the same set of attributes  $R$ 
  - How many different relations over  $R$  can be built by using subsets of this set of tuples?
2. Suppose you are given a set of 100 tuples over the same set of attributes  $R$ 
  - How many different relations over  $R$  can be built by using subsets of this set of tuples?

# Restrictions

- Let  $R = \{A_1, \dots, A_n\}$  be the set of attributes of a relation schema  $N$  and  $r(N) = \{t_1, \dots, t_n\}$
- **Restriction** of a tuple  $t$  onto  $\{A_k, \dots, A_m\} \subseteq \{A_1, \dots, A_n\}$ , denoted as  $t[A_k, \dots, A_m]$ , refers to a sublist of values  $(v_k, \dots, v_m)$  in  $t = (v_1, \dots, v_n)$ , for  $1 \leq k$  and  $m \leq n$
- Example: STUDENT = {Id, Lname, Fname, Major}
  - $t = (300121, \text{Bond}, \text{James}, \text{MATH})$
  - $t[Lname] = \langle \text{Bond} \rangle,$
  - $t[Fname, Major] = \langle \text{James}, \text{Math} \rangle$
- Restriction of a relation  $r$  onto a set of attributes  $\{A_k, \dots, A_m\}$ , is denoted by
 
$$r(N)[A_k, \dots, A_m] = \{t[A_k, \dots, A_m] \mid t \in r\}$$

# A Question

- Given a relation

STUDENT			
Id	Lname	Fname	Major
300111	Smith	Susan	COMP
300121	Bond	James	MATH
300132	Smith	Susan	COMP
300135	John	Cecil	MATH

- What is  $r(\text{STUDENT})[\text{Lname}, \text{Major}]$ ?

a)

Lname	Major
Smith	COMP
Bond	MATH
Smith	COMP
John	MATH

b)

Lname	Major
Bond	MATH
Smith	COMP
John	MATH

c)

Lname	Major
Smith	COMP
Bond	MATH
John	MATH

# Definition Summary

- Given a relation schema  $R(A_1:D_1, A_2:D_2, \dots, A_n:D_n)$ 
  - $R$  is the **name** of the relation
  - $A_1, A_2, \dots, A_n$  are the **attributes** of the relation
  - $D_i$  is the **domain** of attribute  $A_i$ ,  $dom(A_i) = D_i$
- For convenience we sometimes omit the domain assignment from a relation schema
- Relation  $r(R)$ : a specific **state** (or "value" or "population") of  $R$  is a *set of tuples* (rows)
  - $r(R) = \{t_1, t_2, \dots, t_n\}$  where each  $t_i$  is an n-tuple
  - $t_i = \langle v_1, v_2, \dots, v_n \rangle$  where each  $v_j$  is *element-of*  $dom(A_j)$
- $r(R) \subset dom(A_1) \times dom(A_2) \times \dots \times dom(A_n)$



## Example

- Let  $R(A_1, A_2)$  be a relation schema:
- Let  $dom(A_1) = \{0,1\}$ ,  $dom(A_2) = \{a,b,c\}$
- Then:  $dom(A_1) \times dom(A_2)$  is all possible combinations:  
 $\{ \langle 0,a \rangle, \langle 0,b \rangle, \langle 0,c \rangle, \langle 1,a \rangle, \langle 1,b \rangle, \langle 1,c \rangle \}$
- The relation state  $r(R) \subset dom(A_1) \times dom(A_2)$
- Example:  $r(R)$  could be  $\{ \langle 0,a \rangle, \langle 0,b \rangle, \langle 1,c \rangle \}$ 
  - this is one possible state (or “population” or “extension”)  $r$  of the relation  $R$ , defined over  $A_1$  and  $A_2$
  - It has three 2-tuples:  $\langle 0,a \rangle, \langle 0,b \rangle, \langle 1,c \rangle$
  - How many different states (instances) can be?

# Exercises

- Consider schema STUDENT(Id, Lname, Fname, Major)
- 1. Suppose each attribute (e.g. Lname) can get 100 different values
  - a) How many different individual records of the STUDENT schema construct (e.g. (007007, Bond, James, Comp), or (010101, Wong, Sue, Math), or (007007, Mogin, Pavle, Comm)) can be made?
  - b) How many different student records can be made if we introduce a constraint that each student record has to possess a unique *Id* value?
- 2. Suppose each attribute (e.g. Lname) can get only 2 different values, and there is no restriction on *Id* values
  - c) How many different sets of records (instances) can be made?

# Constraints

- Constraints are **conditions** that must hold on **all** valid relation states
- Fundamental to databases
- Real world has **constraints** on what is possible
- A database is an abstraction of the real world  
⇒ should reflect these constraints
- We cannot ensure that the database is correct, but we can ensure that it is **meaningful**
- Constraints are derived from the semantics of the UoD (rules of behaviour, **business rules**)

# Relation Schema Constraints

- The basic relation schema constraints are:
  - Domain constraint
  - Attribute constraint
  - Key constraint, entity integrity constraint
  - Unique constraint
  - Referential integrity constraint
- Some other relational data model constraints, like data dependencies, e.g. functional dependencies, will be covered later in the course

# Domain Constraint

- Specification of a domain:

*Domain\_Name(Basic data type, Maximum length, Condition (range of values | format))*

- e.g. *Age (Integer,  $0 \leq d$  and  $d < 150$ )*
  - Domain name: *Age*
  - Basic data type: *Integer*
  - Maximum length: *not applicable*
  - Condition:  $0 \leq d$  and  $d < 150$  (range of values)
- e.g. *Phone (Char, 12, (999) 999-999)*
  - Domain name: *Phone*
  - Basic data type: *Char*
  - Maximal length: *12*
  - Condition: *(999) 999-999* (format)

# Attribute Constraint

- Domain constraints restrict the attribute values, but may not be sufficient
- Attribute constraints can further restrict attribute values
- Generally, the **attribute constraint** of an attribute  $A$  within a relation schema  $N$  is defined as

$$(Dom(N, A), Range(N, A), Null(N, A))$$

- $Dom(N, A)$  associates attribute  $A$  in  $N$  with a domain via domain name  $D$
- $Range(N, A)$  is used to further restrict the range of allowable attribute  $A$  values in the relations over  $N$
- $Null(N, A)$  specifies whether attribute  $A$  may or may not have a null value in any instance over  $N$

# Attribute Constraint Examples

- Relation schema: STUDENT
  - Attribute: FName
  - $Dom(STUDENT, FName) = STRING$
  - $Range(STUDENT, FName) = none$
  - $Null(STUDENT, FName) = N // not null$
  
- Relation schema: GRADES
  - Attribute: Grade
  - $Dom(GRADES, Grade) = STRING$
  - $Range(GRADES, Grade)$ : one of {'A+', 'A', 'A-', 'B+', 'B', 'B-', 'C+', 'C'}
  - $Null(GRADES, Grade) = Y // yes, null value allowed$

# Relation Schema Key

- A relation is a set of tuples, hence all tuples have to be **distinct**
- Let  $N(A_1, \dots, A_n)$  be a relation schema and  $X = \{A_k, \dots, A_m\} \subseteq \{A_1, \dots, A_n\}$ ,  $X$  is a **relation schema key** of  $N$ , if

1°  $(\forall r(N))(\forall u, v \in r(N))(u[X] = v[X] \Rightarrow u = v)$  (**unique**)

2°  $(\forall Y \subset X)(\neg 1^\circ)$  (**minimal**)

3°  $(\forall r(N))(\forall t \in r(N))(\forall A \in X)(t[A] \neq \omega)$  (**not null**)

- A relation schema key is not allowed to have a null value as the key value is used to identify the individual tuples.
- A relation schema key is also called a **minimal key** or a **key**



# Primary Key and Entity Integrity

- A relation schema may have more than one key  $K$
- One of the relation schema keys  $K$  is designated as a **primary key**  $K_p$ 
  - a key used in UoD for identification most frequently
- **Entity integrity constraint:** no primary key values can be null
- Examples:
  - STAFF(IRDNo, Staff\_id, Fname, Lname, DoB )
    - $K = \{ \text{IRDNo}, \text{Staff\_id} \}$
    - $K_p = \{ \text{Staff\_id} \}$
  - STUDENTNT(Id, Fname, Lname, Major) (the primary key is underlined)

# Superkey, Unique Constraint

- A **superkey** is a superset of a minimal key

$$1^{\circ} (\forall r(N)) (\forall u, v \in r(N)) (u[X] = v[X] \Rightarrow u = v) \\ \text{(unique)}$$

$$3^{\circ} (\forall r(N)) (\forall t \in r(N)) (\forall A \in X) (t[A] \neq \omega) \text{ (not null)}$$

- A **unique constraint** is a constraint that satisfies the condition

$$1^{\circ} (\forall r(N)) (\forall u, v \in r(N)) (u[X] = v[X] \Rightarrow u = v) \\ \text{(unique)}$$

- Can be null
- May not be minimal (but it is preferred to make it minimal)

# Example

- Consider the relation GRADES
- Suppose for each course a student may have at most one grade. What is the relation scheme key?

GRADES		
Id	Course_id	Grade
300111	SWEN304	A+
300111	COMP301	A
300111	MATH314	A
300121	COMP301	B
300132	COMP301	C
300121	SWEN304	ω
300132	SWEN304	ω

# Example

- Consider the relation GRADES
- Suppose for each course a student may have more than one grade, but at most one in a given term. What is the relation scheme key?

GRADES			
Id	Course_id	Term	Grade
300111	SWEN304	1501	A+
300111	COMP301	1501	A
300111	MATH214	1602	A
300121	COMP301	1502	B
300111	MATH214	1502	D
300132	COMP301	1602	C
300121	SWEN304	1602	B
300132	MATH214	1702	ω

# An Example: Inconsistent Relations

STUDENT			
<u>Id</u>	Lname	Fname	Major
300111	Smith	Susan	COMP
300121	Bond	James	MATH
300132	Smith	Susan	COMP
300132	John	Cecil	MATH

COURSE			
<u>Course id</u>	Cname	Points	Dept
SWEN304	DB sys	15	Engineering
COMP301	softEng	20	Engineering
COMP301	softDesign	20	Engineering
MATH214	DisMat	15	Mathematics

# Redefining Some Terms

- **Relation schema**  $N(R, C)$ 
  - $N$  is the name,  $R$  is the set of attributes,  $C$  is the set of constraints
  
- A **tuple**  $t$  over  $R$ :
  - the set of pairs  $t = \{(A_1, a_1), \dots, (A_n, a_n)\}$ , where  $A_i \in R$ ,  $a_i \in \text{dom}(A_i)$ , and  $n = |R|$  is  $\text{Degree}(r(N))$
  
- Relation schema **instance** (relation)  $r(N)$  over  $R$ :
  - a finite **set of n-tuples** that satisfies all constraints

# Relational Databases

- A single relation is not adequate for many database applications
  - “Flat-file” databases are single relation schema databases.
- Typically, we need **many** relation schemas
  - for each kind of entity we are interested in
  - for each interaction/relationship between kinds of entities
  - for multi-valued properties of entities
- Need to **connect** the relation schemas
  - “Foreign keys”
- Need to ensure integrity of the **whole** database
  - “**Referential Integrity**” constraints

# Relational Database Schema

- Relational database schema  $N(S, IC)$ 
  - $N$  is the name,
  - $S = \{N_1(R_1, C_1), \dots, N_k(R_k, C_k)\}$  is a set of relation schemas, and
  - $IC$  is a set of **interrelation** constraints
- Example:
  - Name  $N = \text{UNIVERSITY}$
  - The set of relation schemas
 
$$S = \{\text{STUDENT}(\{\text{Id}, \text{FName}, \text{LName}, \text{Major}\}, \{\text{Id}\}),$$

$$\text{GRADES}(\{\text{Id}, \text{Course\_id}, \text{Grade}\}, \{\text{Id} + \text{Course\_id}\}),$$

$$\text{COURSE}(\{\text{Pname}, \text{Points}, \text{Course\_id}, \text{Dept}\}, \{\text{Course\_id}\})\}$$



# Foreign Key

- Relation schemas are **connected** to each other by (**primary key**, **foreign key**) pairs
- Example:  
 TEXTBOOK({Title, ISBN, Pcode, Pnum }, {ISBN })  
 COURSE({Pcode, Pnum, Pname }, {Pcode + Pnum })
- Instances are connected by (*primary key*, *foreign key*) values
- A foreign key **may** contain null value

TEXTBOOK			
Title	<u>ISBN</u>	Pcod	Pnum
COD	1111	COMP	203
FDBS	2222	SWEN	ω

COURSE		
<u>Pcode</u>	<u>Pnum</u>	Pname
COMP	203	CO
SWEN	304	DBS

# Foreign Key (A General Definition)

- Let  $X = \{A_1, \dots, A_m\}$  be the primary key of  $N_1(R_1, C_1)$ , and let  $Y = \{B_1, \dots, B_m\}$  be a subset of  $R_2$  in  $N_2(R_2, C_2)$
- $Y = \{B_1, \dots, B_m\}$  is a foreign key in  $N_2(R_2, C_2)$  with regard to  $X$  in  $N_1$ , if:
  - $(\forall i \in \{1, \dots, m\})(\text{Dom}(N_2, B_i) \subseteq \text{Dom}(N_1, A_i))$   
(Domain compatibility),
  - $(\forall i \in \{1, \dots, m\})(\text{Range}(N_2, B_i) \Rightarrow \text{Range}(N_1, A_i))$   
(Attribute compatibility)

# Referential Integrity

- Referential integrity is also called a **foreign key constraint**, denoted as

$$N_2[Y] \subseteq N_1[X] \text{ or } N_2.Y < N_1.X$$

- Relations  $r(N_2)$  and  $r(N_1)$  satisfy **referential integrity**  $N_2[Y] \subseteq N_1[X]$  if the set of not null elements of the restriction  $r(N_2)[Y]$  is contained in the restriction  $r(N_1)[X]$ 
  - $N_2$ : the **referencing** relation schema
  - $N_1$ : the **referenced** relation schema
- It is used to maintain **consistency** among tuples of two relations, connected by a (*primary key, foreign key*) pair

# Referential Integrity – A Formal Definition

- Relations  $r(N_2)$  and  $r(N_1)$  satisfy referential integrity  $N_2[Y] \subseteq N_1[X]$  if

$$(\forall u \in r(N_2))(\exists v \in r(N_1))(u[Y] = v[X] \vee (\exists i \in \{1, \dots, m\})(u[B_i] = \omega))$$

- Either tuples  $u$  and  $v$  are equal on  $X$  and  $Y$  values, or there exists at least one attribute in  $Y$  whose  $u$  value is null

# Referential Integrity Constraints

- A set of **referential integrity** constraints forms the most important subset of the relational database schema constraints set  $IC$ .
- Very often, referential integrities are the only interrelation constraints considered
- Example
  - Database schema name: UNIVERSITY
  - A set of relation schemas:

$$S = \{STUDENT, GRADES, COURSE\}$$

$$IC = \{GRADES[Id] \subseteq STUDENT[Id], \\ GRADES[Course\_id] \subseteq COURSE[Course\_id] \}$$

# An Example – Consistent relations

STUDENT			
Id	Lname	Fname	Major
300111	Smith	Susan	COMP
300121	Bond	James	MATH
300143	Bond	Jenny	MATH
300132	Smith	Susan	COMP

COURSE			
Course_id	Cname	Points	Dept
SWEN304	DB sys	15	Engineering
COMP301	softEng	20	Engineering
MATH214	DisMat	15	Mathematics

GRADES		
Id	Course_id	Grade
300111	SWEN304	A+
300111	COMP301	A
300111	MATH214	A
300121	COMP301	B
300132	COMP301	C
300121	SWEN304	B+
300132	SWEN304	C+

# An Example – Inconsistent relations

STUDENT			
Id	Lname	Fname	Major
300111	Smith	Susan	COMP
300121	Bond	James	MATH
300143	Bond	Jenny	
300132	Smith	Susan	COMP

COURSE			
Course_id	Cname	Points	Dept
SWEN304	DB sys	15	Engineering
COMP301	softEng	20	Engineering
MATH214	DisMat	15	Mathematics

GRADES		
Id	Course_id	Grade
300111	SWEN304	A+
300111	COMP301	A
300111	MATH114	A
300121	COMP301	B
300132	COMP301	C
300121	SWEN304	B+
300138	SWEN304	C+

- To avoid inconsistencies with reality we first need to observe the data dependencies hold in reality and make them explicit (specify them)

# A Common Pitfall – Foreign key

- Consider the following relation schemas:

PERSON({Name, Birthday, Address}, {Name + Birthday})

STUDENT({StudentID, Name, DoB, Email}, {StudentID})

- We define a foreign key on STUDENT:

STUDENT [Name, DoB]  $\subseteq$  PERSON[Name, Birthday]

- This is **NOT equivalent** to: (Why?)

STUDENT[Name]  $\subseteq$  PERSON[Name], and

STUDENT[DoB]  $\subseteq$  PERSON[Birthday]

PERSON		
Name	Birthday	Address
Grampa Simpson	01.01.1900	16 Park Ave
Apu Nahasapeemapetilon	29.02.1961	98 Ada St.

STUDENT		
ID	Name	Birthday
007	Apu Nahasapeemapetilon	01.01.1900
008	Grampa Simpson	29.02.1961



# Other Types of Constraints

- Semantic Integrity Constraints:
  - based on application semantics and cannot be expressed by the model per se
  - Example: “the max number of courses a student can enroll in one year”
- A **constraint specification** language may have to be used to express these
- SQL-99 allows triggers and **ASSERTIONS** to express for some of these

# Relational Database Instance

- A database schema  $DBS$  as a **complex** data type defines a finite, but very large number of different database instances
- An instance of the relational database schema  $N(S, IC)$  is

$$db = \{r(N_1), \dots, r(N_k)\}$$

such that:

- Each  $r(N)$  is an instance of a relation schema  $N(R, C)$  in  $S$ , and
- $db$  satisfies all constraints in  $IC$

# Relational Database Instance

- A **database variable**  $db_s$  of the type  $DBS$  contains a set of relational variables  $\{\rho(N_1), \dots, \rho(N_k)\}$
- A relational variable  $\rho(N_1)$  contains an instance of the relation schema  $N$  in each moment of time
- In the sequel
  - a database variable has the same name as the database schema itself
  - each relation variable has the same name as the corresponding relation schema

# Database Schema and Its Instances

- Example:
  - $N = \text{BOOKSHOP}$
- $S = \{ \text{SUPPLIER}(\underline{\text{supplier\_no}}, \text{name}, \text{address}),$   
 $\text{ARTICLE}(\underline{\text{article\_no}}, \text{short\_name}, \text{number\_on\_stock},$   
 $\text{price}),$   
 $\text{OFFER}(\underline{\text{supplier\_no}}, \underline{\text{article\_no}}, \underline{\text{date}}, \text{price}) \}$
- $IC = \{ \text{OFFER} [\text{supplier\_no}] \subseteq \text{SUPPLIER} [\text{supplier\_no}]$   
 $\text{OFFER} [\text{article\_no}] \subseteq \text{ARTICLE} [\text{article\_no}]\}$

# An Instance of The BOOKSHOP Database

SUPPLIER		
<u>supplier_no</u>	name	address
247	Feed The Crowds	Bumptytown
640	Save A Penny	Noroofsville
⋮		

ARTICLE			
<u>article_no</u>	short_name	price	number_on_stock
0815	Weetbricks	5.99	249
0816	Weetkicks	5.99	249
⋮			

OFFER			
<u>supplier_no</u>	<u>article_no</u>	<u>date</u>	price
247	0815	28.02.	3.95
247	0816	28.02.	6.95
247	0815	29.02.	3.93
640	0815	28.02.	3.94
⋮			

# Key Constraints

- You are given a relation schema  $N(R, C)$  and an instance  $r(N)$
- Suppose  $C$  does not contain any key specification
- Inferring keys from instances is very **hard** if possible at all, since there are so many of them
- By analyzing instances and  $Null(N, A)$  constraints, you can only conclude which subsets of  $R$  can **not** be a key
- Also, from instances you may infer which key constraints are **not violated** by instances

# Find Key Constraints not Violated in $r(N)$

a) Suppose  $Null(N, A) = N$  for all attributes except  $F$  in  $N_2$

$r(N_1) =$

$A$	$B$	$C$	$D$
$a_1$	$b_1$	$c_1$	$d_1$
$a_2$	$b_2$	$c_2$	$d_2$
$a_3$	$b_3$	$c_3$	$d_3$
$a_4$	$b_3$	$c_4$	$d_3$
$a_5$	$b_1$	$c_5$	$d_3$

$r(N_2) =$

$A$	$B$	$C$	$D$	$E$	$F$
$a_1$	$b_1$	$c_1$	$d_1$	$e_1$	$f_1$
$a_1$	$b_2$	$c_1$	$d_2$	$e_1$	$f_2$
$a_2$	$b_1$	$c_2$	$d_1$	$e_2$	$f_3$
$a_1$	$b_3$	$c_3$	$d_1$	$e_1$	$\omega$
$a_3$	$b_1$	$c_1$	$d_3$	$e_2$	$f_4$

b) Suppose now  $Null(N_1, C) = Y$  and  $Null(N_2, D) = Y$  and  $Null(N_2, F) = Y$ , and there are some null values in the corresponding columns

# Procedure

1. Produce a **power** set of the set of relation schema attributes
  2. Check subsets for key constraint satisfaction, starting from the subsets with **lower** cardinality
  3. If a subset satisfies a key constraint, all its **supersets** will also satisfy, and therefore do not need to be checked
- Results ( $SatKey(N)(r(N))$ ) key constraint of relation schema  $N$  not violated in  $r(N)$
  - a):
    - $SatKey(N_1)(r(N_1)) = \{A, C\},$
    - $SatKey(N_2)(r(N_2)) = \{AB, CD, BCE, BDE\}$
  - b):
    - $SatKey(N_1)(r(N_1)) = \{A\},$
    - $SatKey(N_2)(r(N_2)) = \{AB, BCE\}$



# Relational Database Operations

- Database Management System must implement **update** operations:
  - **insert**,
  - **delete**, and
  - **modify**
  
- Database Management System must implement **retrieval** operations:
  - query language
  - Need a well defined language

# DB Updates and Constraints

- No update operation should leave a database in an inconsistent state (with violated constraints)
- A DBMS must take the actions necessary to prevent a constraint violation:
  - **reject**: do not allow the operation
  - **cascade**: propagate the operation by making necessary consequential changes
  - **set null**, or **set default**: reset other values to maintain consistency

# Inserts and Constraint Violations

- **Inserting** a new tuple could **violate**
  - Attribute/domain constraints  
(a value is not of the right type or within the required range)
  - Uniqueness constraints  
(the values of the key attributes duplicate another tuple)
  - Not Null constraints  
(an attribute has the value null when it shouldn't)
  - Referential Integrity constraints  
(the values of the attributes of a foreign key do not match any tuple in the other relation)
- **Response:**
  - **Reject** the operation – there is no change that the DBMS system could safely make to resolve the inconsistency

# Deletes and Constraint Violations

- **Deleting** a tuple can only **violate** a **referential integrity constraint**:
  - If a tuple  $t$  is referred to by foreign keys in some tuples  $t_1, t_2, \dots, t_n$  in other relations, then deleting  $t$  will make  $t_1, t_2, \dots, t_n$  inconsistent.
  - Example:
    - Delete a student record from the database, and all their grade records will refer to nothing
- There are several options:
  - **Reject** the deletion
  - **Set null / set default**: insert null or a default value in the foreign key attributes of tuples in other relation(s) that refer to  $t$  (can't do set null if foreign key attributes are NOT NULL)
  - **Cascade**: delete tuples in other relation(s) that refer to  $t$  (appropriate only if the other tuples "existentially depend" on  $t$ )

# Modify and Constraint Violations

- **Modifying/updating** the values of attributes in a tuple may **violate** constraints
  - Attribute/domain constraints  
Response: **reject** (like insert)
  - Key constraints (if attribute is part of a key)  
Response: treat as a **delete followed by an insert**
  - Referential integrity constraints (if attribute is part of a foreign key).  
Response: **reject** (like insert), or **cascade**, or **set null**, or **set default** (like delete)

# A Question for You

- Consider the following database instance

TEXTBOOK			
Title	<u>ISBN</u>	Pcod	Pnum
COD	1111	COMP	203
FDBS	2222	SWEN	ω

COURSE		
<u>Pcode</u>	<u>Pnum</u>	Pname
COMP	203	CO
SWEN	304	DBS

- Should a DBMS reject the following update operation: (Y/N)?

```
UPDATE TEXTBOOK SET PNum = 304 WHERE ISBN = 2222;
```

- Should a DBMS reject the following update operation: (Y/N)?

```
UPDATE TEXTBOOK SET PNum = 304 WHERE ISBN = 1111;
```

# DB Updates and Constraints

Update operation	Domain / Attribute constraint	Key / Entity integrity constraint,	Referential integrity
<b>insert</b>	reject	reject	reject
<b>delete</b>	no violation	no violation	reject, cascade, set null, set default
<b>modify</b>	reject	reject	reject, cascade, set null, set default

# Summary

- Basic concepts of the relational data model:
  - **Domain** (set of values) – data type,
  - **Attribute** (property of a set of similar UoD objects),
  - **Relation schema**
- Relation schema constraints:
  - **Domain** constraint,
  - **Attribute** constraint,
  - **Key** constraint and unique constraint
- A **relational database schema** – a set of **relation schemas** and a set of **interrelation constraints**
- The **referential integrity** is the most important interrelation constraint, it links tuples of two relations
- A **relational database** is a set of such relation instances that satisfy all relational and interrelation constraints
- No update operation should leave a database in an inconsistent state (with violated constraints)