Relational Model Concepts

- The relational Model of Data is based on the concept of a Relation.
- A Relation is a mathematical concept based on the **ideas of sets**.
- The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations.

Relation

- RELATION: A table of values
  - A relation may be thought of as a **set of rows**.
  - A relation may alternately be thought of as a **set of columns**.
  - Each row represents a fact that corresponds to a real-world **entity** or **relationship**.
  - Each row has a value of an item or set of items that uniquely identifies that row in the table.
  - Sometimes row-ids or sequential numbers are assigned to identify the rows in the table.
  - Each column typically is called by its column name or column header or attribute name.

Schema of a Relation

- A **Relation** may be defined in multiple ways.
- The **Schema** of a Relation: *R* (A1, A2, ...An)

  Relation schema *R* is defined over **attributes** A1, A2, ...An

  For Example -

  CUSTOMER (Cust-id, Cust-name, Address, Phone#)

  Here, CUSTOMER is a relation defined over the four attributes Cust-id, Cust-name, Address, Phone#, each of which has **a domain** or **a set of valid values**. For example, the domain of Cust-id is 6 digit numbers.

Tuples

- A **tuple** is **an ordered set of values**
- Each value is derived from an appropriate domain.
- Each row in the CUSTOMER table may be referred to as a tuple in the table and would consist of four values.
- **<632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">** is a tuple belonging to the CUSTOMER relation.
- **A relation** may be regarded as a ***set of tuples*** (rows).
- **Columns** in a table are also called **attributes** of the relation.
-

Domains

- A **domain** has a logical definition: e.g., "USA_phone_numbers" are the set of 10 digit phone numbers valid in the U.S.
- A domain may have a data-type or a format defined for it. The USA_phone_numbers may have a format: (ddd)-ddd-dddd where each d is a decimal digit. E.g., Dates have various formats such as monthname, date, year or yyyy-mm-dd, or dd mm,yyyy etc.
- An attribute designates the **role** played by the domain. E.g., the domain Date may be used to define attributes "Invoice-date" and "Payment-date".

FORMAL DEFINITIONS

The relation is formed over the cartesian product of the sets; each set has values from a domain; that domain is used in a specific role which is conveyed by the attribute name.

For example, attribute Cust-name is defined over the domain of strings of 25 characters. The role these strings play in the CUSTOMER relation is that of the name of customers.

Formally,

Given $R(A_1, A_2, ..., A_n)$

$r(R) \subset dom(A_1) \times dom(A_2) \times ....X\ dom(A_n)$

R: schema of the relation

r of R: a specific "value" or population of R.

R is also called the intension of a relation

r is also called the extension of a relation

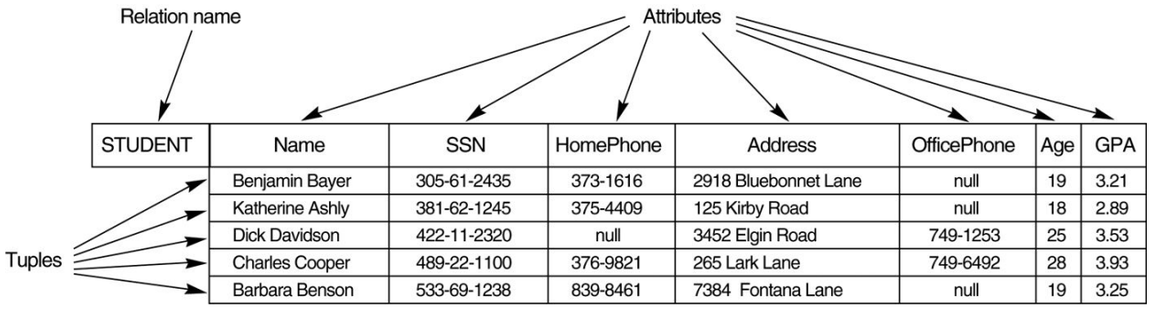
Let S1 = {0,1}

Let S2 = {a,b,c}

Let $R \subset S1 \times S2$

Then for example: r(R) = {<0,a> , <0,b> , <1,c> }

is one possible "state" or "population" or "extension" r of the relation R, defined over domains S1 and S2. It has three tuples.



STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	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
	Charles Cooper	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

CHARACTERISTICS OF RELATIONS

- Ordering of tuples in a relation r(R): The tuples are *not* considered to be ordered, even though they appear to be in the tabular form.
- Ordering of attributes in a relation schema R (and of values within each tuple): We will consider the attributes in R(A_1, A_2, ..., A_n) and the values in t=<v_1, v_2, ..., v_n> to be *ordered* .
  (However, a more general *alternative definition* of relation does not require this ordering).
- Values in a tuple: All values are considered *atomic* (indivisible). A special null value is used to represent values that are unknown or inapplicable to certain tuples.

- component values of a tuple t by $t[A_i] = v_i$ (the value of attribute A_i for tuple t).
  Similarly, $t[A_u, A_v, ..., A_w]$ refers to the subtuple of t containing the values of attributes **A_u, A_v, ..., A_w, respectively.**

STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21

Relational Integrity Constraints

- **Constraints are *conditions* that must hold on *all* valid relation instances. There are three main types of constraints:**
  1. **Key constraints**
  2. **Entity integrity constraints**
  3. **Referential integrity constraints**

Key Constraints

- Superkey of R: A set of attributes SK of R such that no two tuples *in any valid relation instance r(R)* will have the same value for SK. That is, for any distinct tuples t1 and t2 in r(R), t1[SK] ≠ t2[SK].
- Key of R: A "minimal" superkey; that is, a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey.
  Example: The CAR relation schema:
  CAR(State, Reg#, SerialNo, Make, Model, Year)
  has two keys Key1 = {State, Reg#}, Key2 = {SerialNo}, which are also superkeys. {SerialNo, Make} is a superkey but *not* a key.
- If a relation has *several* candidate keys, one is chosen arbitrarily to be the primary key. The primary key attributes are *underlined*.

Figure 7.4 The CAR relation with two candidate keys: LicenseNumber and EngineSerialNumber.

CAR	LicenseNumber	EngineSerialNumber	Make	Model	Year
	Texas ABC-739	A69352	Ford	Mustang	96
	Florida TVP-347	B43696	Oldsmobile	Cutlass	99
	New York MPO-22	X83554	Oldsmobile	Delta	95
	California 432-TFY	C43742	Mercedes	190-D	93
	California RSK-629	Y82935	Toyota	Camry	98
	Texas RSK-629	U028365	Jaguar	XJS	98

Entity Integrity

- **Relational Database Schema**: A set S of relation schemas that belong to the same database. S is the *name* of the **database**.

$S = \{R_1, R_2, ..., R_n\}$

- **Entity Integrity**: **The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of r(R).** This is because primary key values are used to *identify* the individual tuples.

t[PK] ≠ null for any tuple t in r(R) for any R

- Note: Other attributes of R may be similarly constrained to disallow null values, even though they are not members of the primary key.

Referential Integrity

- A constraint involving *two* relations (the previous constraints involve a *single* relation).
- Used to specify a *relationship* among tuples in two relations: the **referencing relation** and the **referenced relation**.
- Tuples in the *referencing relation* R_1 have attributes FK (called **foreign key** attributes) that reference the primary key attributes PK of the *referenced relation* R_2. A tuple t_1 in R_1 is said to **reference** a tuple t_2 in R_2 if $t_1[FK] = t_2[PK]$.
- A referential integrity constraint can be displayed in a relational database schema as a directed arc from R_1.FK to R_2.PK

Referential Integrity Constraint

Statement of the constraint

The value in the foreign key column (or columns) FK of the the **referencing relation** R_1 can be either:

(1) a value of an existing primary key value of the corresponding primary key PK in the **referenced relation** R_2, or..

(2) a null.

In case (2), the FK in R_1 should not be a part of its own primary key.

Figure 7.5 Schema diagram for the COMPANY relational database schema; the primary keys are underlined.

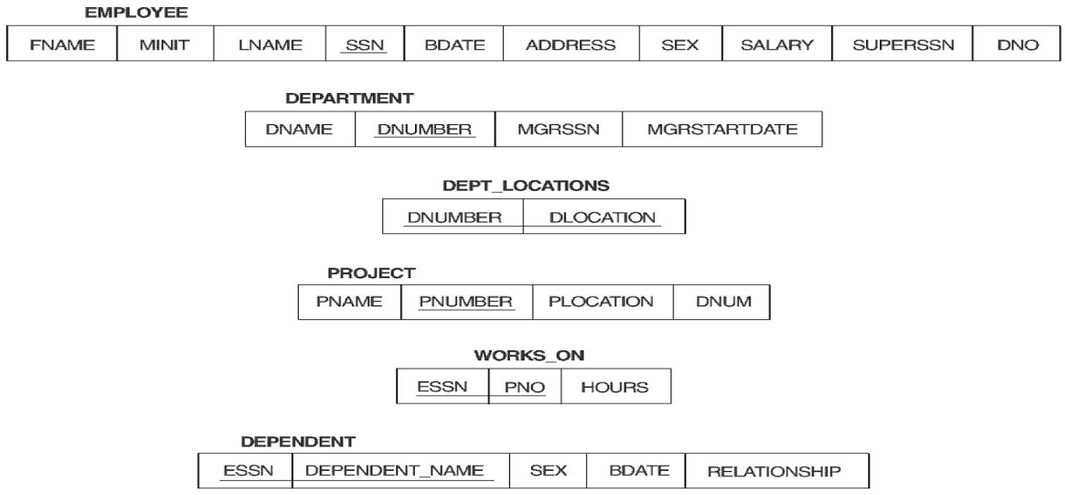


Figure 7.6 One possible relational database state corresponding to the COMPANY schema.

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
John		Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin		Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia		Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer		Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh		Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce		English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad		Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James		Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS

DNUMBER	DLOCATION
	Houston
	Stafford
	Bellaire
	Sugarland

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

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

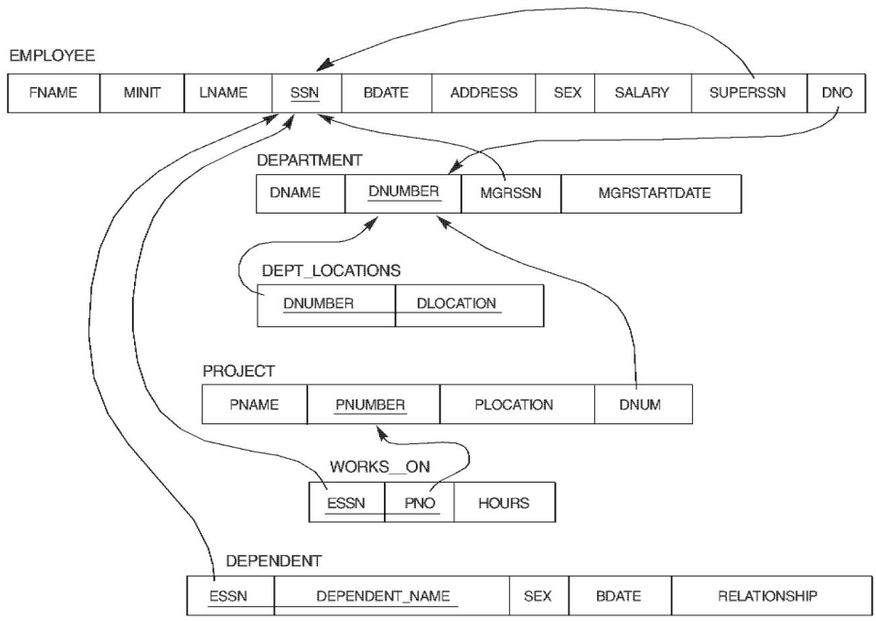
PROJECT

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

Figure 7.7 Referential integrity constraints displayed on the COMPANY relational database schema diagram.



Update Operations on Relations

- INSERT a tuple.
- DELETE a tuple.
- MODIFY a tuple.

- Integrity constraints should not be violated by the update operations.
- Several update operations may have to be grouped together.
- Updates may *propagate* to cause other updates automatically. This may be necessary to maintain integrity constraints.
- In case of integrity violation, several actions can be taken: (e.g. Tables: **Employee** and **Work_On**)
  Cancel the operation that causes the violation (REJECT option)
  Perform the operation but inform the user of the violation
  Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
  Execute a user-specified error-correction routine

Q. Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course:
STUDENT(SSN, Name, Major, Bdate)
COURSE(Course#, Cname, Dept)
ENROLL(SSN, Course#, Quarter, Grade)
BOOK_ADOPTION(Course#, Quarter, Book_ISBN)
TEXT(Book_ISBN, Book_Title, Publisher, Author)
Draw a relational schema diagram specifying the foreign keys for this schema.

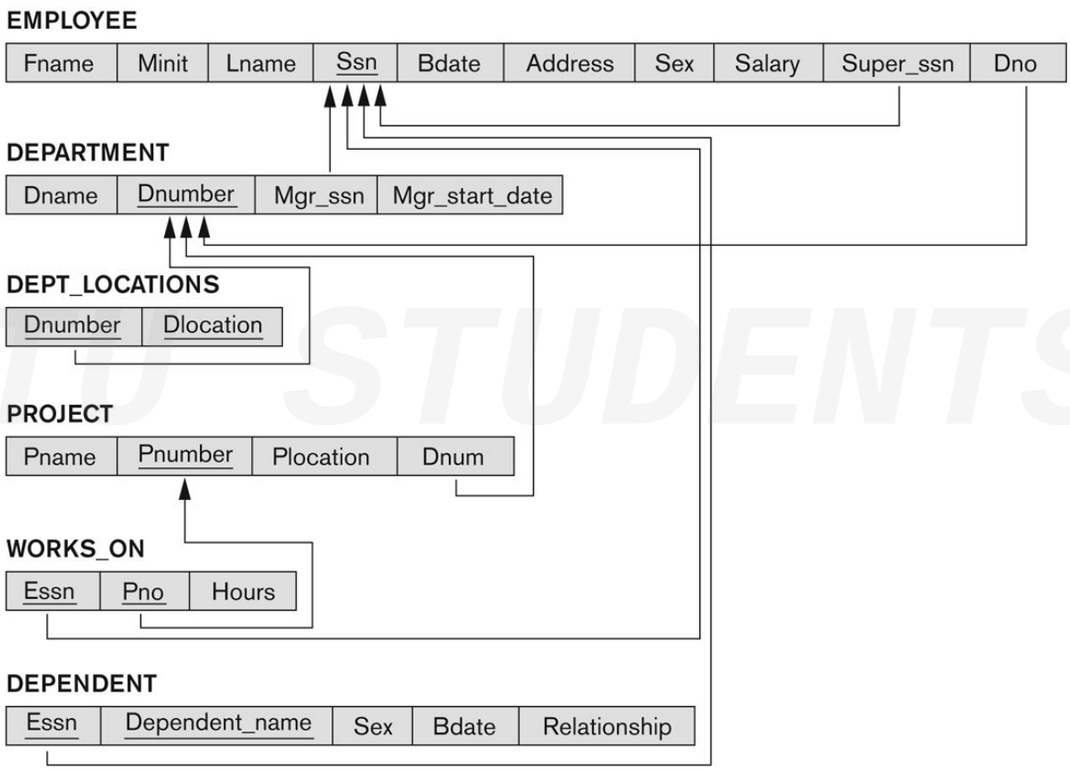
Relational Algebra

- Relational algebra is the basic set of operations for the relational model
- These operations enable a user to specify **basic retrieval requests** (or **queries**)
- The result of an operation is a *new relation*, which may have been formed from one or more *input* relations
- This property makes the algebra "closed" (all objects in relational algebra are relations)
- The **algebra operations** thus produce new relations
- These can be further manipulated using operations of the same algebra
- A sequence of relational algebra operations forms a **relational algebra expression**
- The result of a relational algebra expression is also a relation that represents the result of a database query (or retrieval request)
- Relational Algebra consists of several groups of operations

- Unary Relational Operations
  - SELECT (symbol: **s** (sigma))
  - PROJECT (symbol: **p** (pi))
  - RENAME (symbol: **ρ** (rho))
- Relational Algebra Operations From Set Theory
  - UNION (**È**), INTERSECTION (**Ç**), DIFFERENCE (or MINUS, –)
  - CARTESIAN PRODUCT (**x**)
- Binary Relational Operations
  - JOIN (several variations of JOIN exist)
  - DIVISION

- Additional Relational Operations
  - OUTER JOINS, OUTER UNION
  - AGGREGATE FUNCTIONS (These compute summary of information: for example, SUM, COUNT, AVG, MIN, MAX)

Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.



Unary Relational Operations: SELECT

- The SELECT operation (denoted by s (sigma)) is used to select a *subset* of the tuples from a relation based on a selection condition.
  - The selection condition acts as a filter
  - Keeps only those tuples that satisfy the qualifying condition
  - Tuples satisfying the condition are *selected* whereas the other tuples are discarded (*filtered out*)
- Examples:
  - Select the EMPLOYEE tuples whose department number is 4:

$s_{DNO = 4}$ (EMPLOYEE)

  - Select the employee tuples whose salary is greater than $30,000:

$s_{SALARY > 30,000}$ (EMPLOYEE**)**

- In general, the *select* operation is denoted by $s_{<selection\ condition>}$(R) where
  - the symbol s (sigma) is used to denote the *select* operator
  - the selection condition is a Boolean (conditional) expression specified on the attributes of relation R
  - tuples that make the condition true are selected
    - appear in the result of the operation
  - tuples that make the condition false are filtered out

- discarded from the result of the operation

- **SELECT Operation Properties**

  - The SELECT operation $s_{<\text{selection condition}>}(R)$ produces a relation S that has the same schema (same attributes) as R
  - SELECT s is commutative:
  - $s_{<\text{condition1}>}(s_{<\text{condition2}>}(R)) = s_{<\text{condition2}>}(s_{<\text{condition1}>}(R))$
  - Because of commutativity property, a cascade (sequence) of SELECT operations may be applied in any order:
  - $s_{<\text{cond1}>}(s_{<\text{cond2}>}(s_{<\text{cond3}>}(R)) = s_{<\text{cond2}>}(s_{<\text{cond3}>}(s_{<\text{cond1}>}(R)))$
  - A cascade of SELECT operations may be replaced by a single selection with a conjunction of all the conditions:
  - $s_{<\text{cond1}>}(s_{<\text{cond2}>}(s_{<\text{cond3}>}(R)) = s_{<\text{cond1}> \text{ AND } <\text{cond2}> \text{ AND } <\text{cond3}>}(R))$
  - The number of tuples in the result of a SELECT is less than (or equal to) the number of tuples in the input relation R

The following query results refer to this database state

Figure 5.6
One possible database state for the COMPANY relational database schema.

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

Unary Relational Operations: PROJECT

- PROJECT Operation is denoted by **p** (pi)
- This operation keeps certain *columns* (attributes) from a relation and discards the other columns.
  - PROJECT creates a vertical partitioning
    - The list of specified columns (attributes) is kept in each tuple
    - The other attributes in each tuple are discarded
- Example: To list each employee's first and last name and salary, the following is used:
  $p_{\text{LNAME, FNAME,SALARY}}(\text{EMPLOYEE})$
- The general form of the *project* operation is:
  $p_{<\text{attribute list}>}(R)$
  - p (pi) is the symbol used to represent the *project* operation
  - <attribute list> is the desired list of attributes from relation R.
- The project operation *removes any duplicate tuples*
  - This is because the result of the *project* operation must be a *set of tuples*
    - Mathematical sets *do not allow* duplicate elements.
- PROJECT Operation Properties
  - The number of tuples in the result of projection $p_{<\text{list}>}(R)$ is always less or equal to the

number of tuples in R
    - If the list of attributes includes a *key* of R, then the number of tuples in the result of PROJECT is *equal* to the number of tuples in R
- PROJECT is *not* commutative
    - $p_{<list1>}(p_{<list2>}(R)) = p_{<list1>}(R)$ as long as <list2> contains the attributes in <list1>

Figure 6.1
Results of SELECT and PROJECT operations. (a) $\sigma_{(Dno=4\ \textbf{AND}\ Salary>25000)\ OR\ (Dno=5\ \textbf{AND}\ Salary>30000)}(EMPLOYEE)$. (b) $\pi_{Lname, Fname, Salary}(EMPLOYEE)$. (c) $\pi_{Sex, Salary}(EMPLOYEE)$.

(a)

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
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

(b)

Lname	Fname	Salary
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

(c)

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

To retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a select and a project operation

- We can write a *single relational algebra expression* as follows:
    - $\mathbf{p}_{FNAME, LNAME, SALARY}(\mathbf{s}_{DNO=5}(EMPLOYEE))$
- OR We can explicitly show the *sequence of operations*, giving a name to each intermediate relation:
    - DEP5_EMPS ← $\mathbf{s}_{DNO=5}(EMPLOYEE)$
    - RESULT ← $\mathbf{p}_{FNAME, LNAME, SALARY}$ (DEP5_EMPS)

Unary Relational Operations: RENAME

- The RENAME operator is denoted by ρ (rho)
- In some cases, we may want to *rename* the attributes of a relation or the relation name or both
    - Useful when a query requires multiple operations
    - Necessary in some cases
    - The general RENAME operation ρ can be expressed by any of the following forms:
        $\rho_{S(B1, B2, ..., Bn)}(R)$ changes both:
        the relation name to S, *and*
        the column (attribute) names to B1, B1, ...Bn
        $\rho_{S}(R)$ changes:
        the *relation name* only to S
        $\rho_{(B1, B2, ..., Bn)}(R)$ changes:
        the *column (attribute) names* only to B1, B1, ...Bn
- For convenience, we also use a *shorthand* for renaming attributes in an intermediate relation:
    - If we write:
        - RESULT ← $\mathbf{p}_{FNAME, LNAME, SALARY}$ (DEP5_EMPS)
        - RESULT will have the *same attribute names* as DEP5_EMPS (same attributes as EMPLOYEE)
    - If we write:
        - RESULT (F, M, L, S, B, A, SX, SAL, SU, DNO)← $\mathbf{p}_{FNAME, LNAME, SALARY}$ (DEP5_EMPS)
- The 10 attributes of DEP5_EMPS are *renamed* to F, M, L, S, B, A, SX, SAL, SU, DNO, respectivel

(a)

Fname	Lname	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

(b)
TEMP

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
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

R

First_name	Last_name	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

Figure 6.2
Results of a sequence of operations.
(a) $\pi_{Fname, Lname, Salary}(\sigma_{Dno=5}(EMPLOYEE))$.
(b) Using intermediate relations and renaming of attributes.

- **UNION Operation**
  - Binary operation, denoted by È
  - The result of R È S, is a relation that includes all tuples that are either in R or in S or in both R and S
  - Duplicate tuples are eliminated
  - The two operand relations R and S must be "type compatible" (or UNION compatible)
  - R and S must have same number of attributes
  - Each pair of corresponding attributes must be type compatible (have same or compatible domains)

- Example:
  - To retrieve the social security numbers of all employees who either *work in department 5* (RESULT1 below) or *directly supervise an employee who works in department 5* (RESULT2 below)
  - We can use the UNION operation as follows:
    DEP5_EMPS ← $s_{DNO=5}$ (EMPLOYEE)
    RESULT1 ← p_{SSN}(DEP5_EMPS)
    RESULT2(SSN) ← $p_{SUPERSSN}$(DEP5_EMPS)
    RESULT ← RESULT1 È RESULT2
  - The union operation produces the tuples that are in either RESULT1 or RESULT2 or both

Figure 6.3
Result of the UNION operation RESULT ← RESULT1 ∪ RESULT2.

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

- Type Compatibility of operands is required for the binary set operation UNION È, (also for INTERSECTION Ç, and SET DIFFERENCE)
- R1(A1, A2, ..., An) and R2(B1, B2, ..., Bn) are type compatible if:
  - they have the same number of attributes, and
  - the domains of corresponding attributes are type compatible (i.e. dom(Ai)=dom(Bi) for i=1, 2, ..., n).
- The resulting relation for R1ÈR2 (also for R1ÇR2, or R1–R2, see next slides) has the same attribute names as the *first* operand relation R1 (by convention)

- INTERSECTION is denoted by ∧
- The result of the operation R ∧ S, is a relation that includes all tuples that are in both R and S
  - The attribute names in the result will be the same as the attribute names in R
- The two operand relations R and S must be "type compatible"

- **SET** DIFFERENCE (also called MINUS or EXCEPT) is denoted by –
- The result of R – S, is a relation that includes all tuples that are in R but not in S
  - The attribute names in the result will be the same as the attribute names in R
- The two operand relations R and S must be "type compatible"

(a) **STUDENT**

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

(b)

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Figure 6.4
The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT – INSTRUCTOR. (e) INSTRUCTOR – STUDENT.

- Notice that both union and intersection are *commutative* operations; that is
  - R È S = S È R, and R Ç S = S Ç R
- Both union and intersection can be treated as n-ary operations applicable to any number of relations as both are *associative* operations; that is
  - R È (S È T) = (R È S) È T
  - (R Ç S) Ç T = R Ç (S Ç T)
- The minus operation is not commutative; that is, in general

R – S ≠ S – R

CARTESIAN PRODUCT

- CARTESIAN (or CROSS) PRODUCT Operation
  - This operation is used to combine tuples from two relations in a combinatorial fashion.
  - Denoted by R(A1, A2, . . ., An) x S(B1, B2, . . ., Bm)
  - Result is a relation Q with degree n + m attributes:
    - Q(A1, A2, . . ., An, B1, B2, . . ., Bm), in that order.
  - The resulting relation state has one tuple for each combination of tuples—one from R and one from S.
  - Hence, if R has n_R tuples (denoted as $|R| = n_R$), and S has n_S tuples, then R x S will have $n_R * n_S$ tuples.
  - The two operands do NOT have to be "type compatible"
  - Generally, CROSS PRODUCT is not a meaningful operation

  Can become meaningful when followed by other operations
  Example (not meaningful):
  FEMALE_EMPS ← **s** $_{SEX='F'}$(EMPLOYEE)
  EMPNAMES ← **p** $_{FNAME, LNAME, SSN}$ (FEMALE_EMPS)
  EMP_DEPENDENTS ← EMPNAMES x DEPENDENT
  EMP_DEPENDENTS will contain every combination of EMPNAMES and DEPENDENT whether or not they are actually related

Figure 6.5
The CARTESIAN PRODUCT (CROSS PRODUCT) operation.

FEMALE_EMPS

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	1968-07-19	3321Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291Berry, Bellaire, TX	F	43000	888665555	4
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

EMPNAMES

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

EMP_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

ACTUAL_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...

RESULT

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

Binary Relational Operations: JOIN (denoted by ⋈)

- The sequence of CARTESIAN PRODECT followed by SELECT is used quite commonly to identify and select related tuples from two relations
- A special operation, called JOIN combines this sequence into a single operation
- This operation is very important for any relational database with more than a single relation, because it allows us *combine related tuples* from various relations
- The general form of a join operation on two relations R(A1, A2, . . ., An) and S(B1, B2, . . ., Bm) is:

R $\bowtie_{\text{<join condition>}}$ S

- where R and S can be any relations that result from general *relational algebra expressions*.

DEPT_MGR

Dname	Dnumber	Mgr_ssn	...	Fname	Minit	Lname	Ssn	...
Research	5	333445555	...	Franklin	T	Wong	333445555	...
Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
Headquarters	1	888665555	...	James	E	Borg	888665555	...

Figure 6.6
Result of the JOIN operation

EQUIJOIN

- The most common use of join involves join conditions with *equality comparisons* only
- Such a join, where the only comparison operator used is =, is called an EQUIJOIN.
  - In the result of an EQUIJOIN we always have one or more pairs of attributes (whose names need not be identical) that have identical values in every tuple.

NATURAL JOIN Operation

Another variation of JOIN called NATURAL JOIN — denoted by * — was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition.

because one of each pair of attributes with identical values is superfluous

The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, *have the same name* in both relations

If this is not the case, a renaming operation is applied first.

- Example: To apply a natural join on the DNUMBER attributes of DEPARTMENT

and DEPT_LOCATIONS, it is sufficient to write:

- DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS
- Only attribute with the same name is DNUMBER
- An implicit join condition is created based on this attribute:
- DEPARTMENT.DNUMBER=DEPT_LOCATIONS.DNUMBER
- Another example: Q ← R(A,B,C,D) * S(C,D,E)
- The implicit join condition includes *each pair* of attributes with the same name, "AND"ed together:
  - R.C=S.C AND R.D.S.D
- Result keeps only one attribute of each such pair:
  - Q(A,B,C,D,E)

(a)

PROJ_DEPT

Pname	Pnumber	Plocation	Dnum	Dname	Mgr_ssn	Mgr_start_date
ProductX	1	Bellaire	5	Research	333445555	1988-05-22
ProductY	2	Sugarland	5	Research	333445555	1988-05-22
ProductZ	3	Houston	5	Research	333445555	1988-05-22
Computerization	10	Stafford	4	Administration	987654321	1995-01-01
Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

(b)

DEPT_LOCS

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

Figure 6.7
Results of two NATURAL JOIN operations.
(a) PROJ_DEPT ← PROJECT * DEPT.
(b) DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS.

■ DIVISION Operation

The division operation is applied to two relations

R(Z) ¸ S(X), where X subset Z. Let Y = Z - X (and hence Z = X È Y); that is, let Y be the set of attributes of R that are not attributes of S.

The result of DIVISION is a relation T(Y) that includes a tuple t if tuples t_R appear in R with t_R [Y] = t, and with

t_R [X] = t_s *for every tuple* t_s in S.

For a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with *every* tuple in S.

(a)

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

(b)

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

Figure 6.8
The DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) *T* ← *R* ÷ *S*.

Table 6.1
Operations of Relational Algebra

Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{<\text{selection condition}>}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{<\text{attribute list}>}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{<\text{join condition}>} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{<\text{join condition}>} R_2$, OR $R_1 \bowtie_{(<\text{join attributes 1}>),\ (<\text{join attributes 2}>)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{<\text{join condition}>} R_2$, OR $R_1 *_{(<\text{join attributes 1}>),\ (<\text{join attributes 2}>)} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Relational Calculus

- A **relational calculus** expression creates a new relation, which is specified in terms of variables that range over rows of the stored database relations (in **tuple calculus**) or over columns of the stored relations (in **domain calculus**).
- In a calculus expression, there is *no order of operations* to specify how to retrieve the query result—a calculus expression specifies only what information the result should contain.
  - This is the main distinguishing feature between relational algebra and relational calculus.
  - Relational calculus is considered to be a **nonprocedural** language.
  - This differs from relational algebra, where we must write a *sequence of operations* to specify a retrieval request; hence relational algebra can be considered as a **procedural** way of stating a query.

Tuple Relational Calculus

- The tuple relational calculus is based on specifying a number of tuple variables.
- Each tuple variable usually ranges over a particular database relation, meaning that the variable may take as its value any individual tuple from that relation.
- A simple tuple relational calculus query is of the form
  - $\{t \mid \text{COND}(t)\}$
- where t is a tuple variable and COND (t) is a conditional expression involving t.
- The result of such a query is the set of all tuples t that satisfy COND (t).

- Example: To find the first and last names of all employees whose salary is above $50,000, we can write the following tuple calculus expression:
- $\{t.\text{FNAME}, t.\text{LNAME} \mid \text{EMPLOYEE}(t) \text{ AND } t.\text{SALARY}>50000\}$
- The condition EMPLOYEE(t) specifies that the range relation of tuple variable t is EMPLOYEE.
- The first and last name (PROJECTION $p_{\text{FNAME, LNAME}}$) of each EMPLOYEE tuple t that satisfies the condition t.SALARY>50000 (SELECTION $s_{\text{SALARY >50000}}$) will be retrieved.

The Existential and Universal Quantifiers

- Two special symbols called quantifiers can appear in formulas; these are the universal quantifier (") and the existential quantifier ($).
- Informally, a tuple variable t is bound if it is quantified, meaning that it appears in an (" t) or ($ t) clause; otherwise, it is free.
- If F is a formula, then so are ($ t)(F) and (" t)(F), where t is a tuple variable.
  - The formula ($ t)(F) is true if the formula F evaluates to true for some (at least one) tuple assigned to free occurrences of t in F; otherwise ($ t)(F) is false.
  - The formula (" t)(F) is true if the formula F evaluates to true for every tuple (in the universe) assigned to free occurrences of t in F; otherwise (" t)(F) is false.
- " is called the universal or "for all" quantifier because every tuple in "the universe of" tuples must make F true to make the quantified formula true.
- $ is called the existential or "there exists" quantifier because any tuple that exists in "the universe of" tuples may make F true to make the quantified formula true.

- The language **SQL** is based on tuple calculus. It uses the basic block structure to express the queries in tuple calculus:
  - SELECT <list of attributes>
  - FROM <list of relations>
  - WHERE <conditions>
- SELECT clause mentions the attributes being projected, the FROM clause mentions the relations needed in the query, and the WHERE clause mentions the selection as well as the join conditions.

  SQL syntax is expanded further to accommodate other operations

  Another language which is based on tuple calculus is **QUEL** which actually uses the range variables as in tuple calculus. Its syntax includes:

  RANGE OF <variable name> IS <relation name>

  Then it uses

  RETRIEVE <list of attributes from range variables>

  WHERE <conditions>

  This language was proposed in the relational DBMS INGRES.

The Domain Relational Calculus

- Another variation of relational calculus called the domain relational calculus, or simply, domain calculus is equivalent to tuple calculus and to relational algebra.
- The language called QBE (Query-By-Example) that is related to domain calculus was developed almost concurrently to SQL at IBM Research, Yorktown Heights, New York.
  - Domain calculus was thought of as a way to explain what QBE does.
- Domain calculus differs from tuple calculus in the type of variables used in formulas:
  - Rather than having variables range over tuples, the variables range over single values from domains of attributes.
- To form a relation of degree n for a query result, we must have n of these domain variables—one for each attribute.
- An expression of the domain calculus is of the form

$\{ x_1, x_2, \ldots, x_n \mid$

$COND(x_1, x_2, \ldots, x_n, x_{n+1}, x_{n+2}, \ldots, x_{n+m})\}$

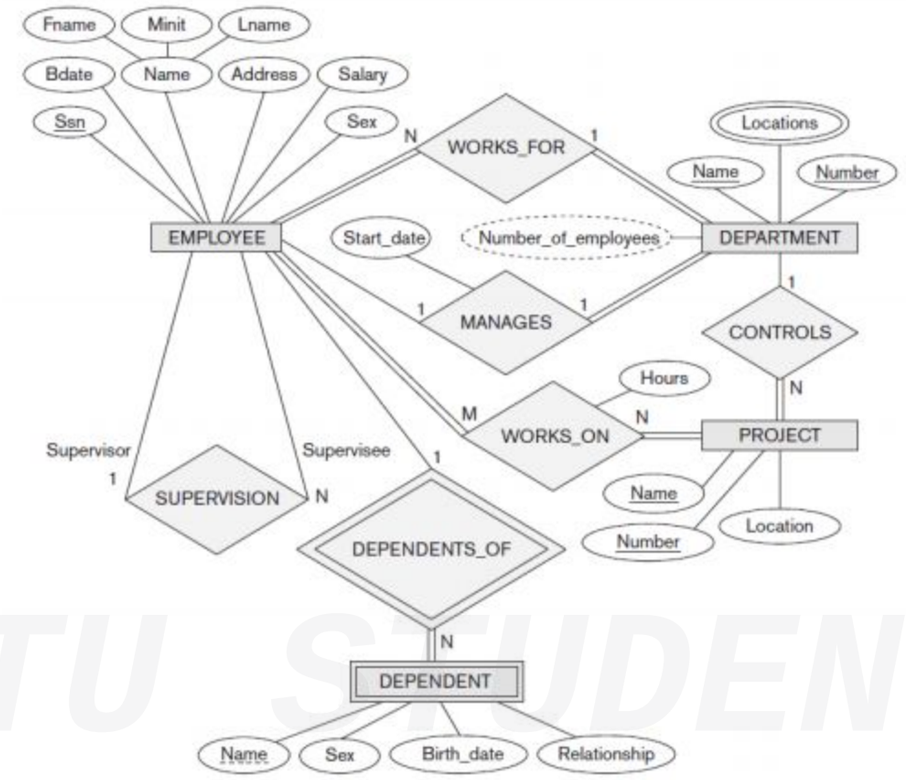
  - where $x_1, x_2, \ldots, x_n, x_{n+1}, x_{n+2}, \ldots, x_{n+m}$ are domain variables that range over domains (of attributes)
  - and COND is a condition or formula of the domain relational calculus.

QBE: A Query Language Based on Domain Calculus

- This language is based on the idea of giving an example of a query using example elements.
- An example element stands for a domain variable and is specified as an example value preceded by the underscore character.
- P. (called P dot) operator (for "print") is placed in those columns which are requested for the result of the query.
- A user may initially start giving actual values as examples, but later can get used to providing a minimum number of variables as example elements.

Relational Database Design Using ER-to-Relational Mapping

Figure 9.1
The ER conceptual schema diagram for the COMPANY database.



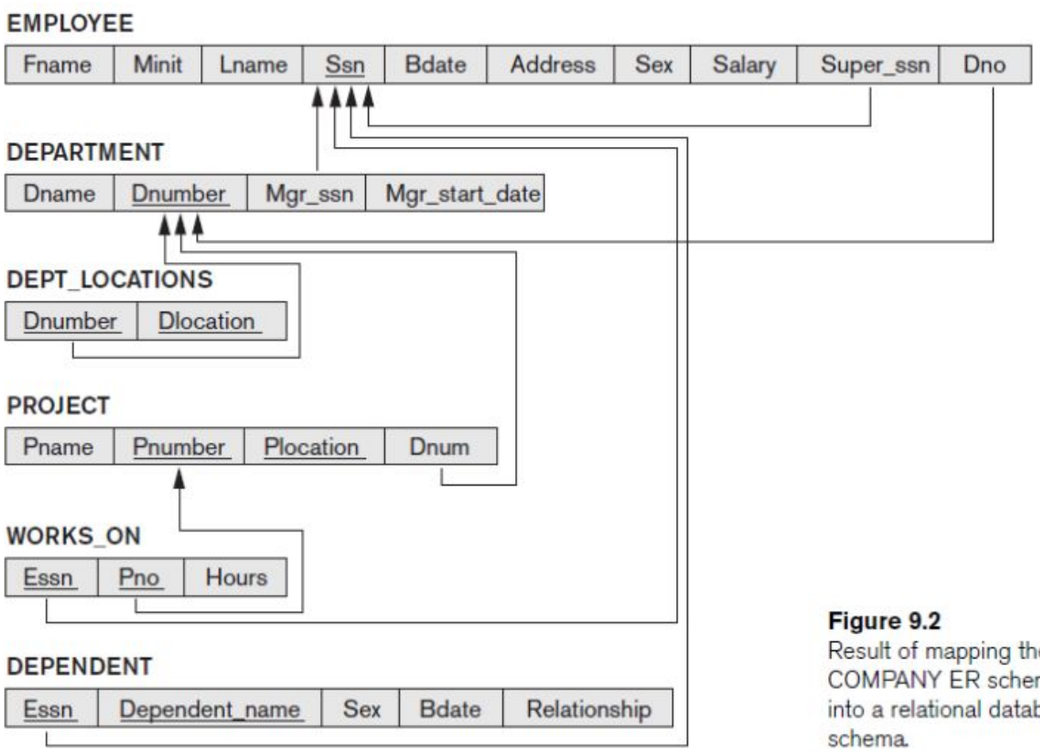


Figure 9.2
Result of mapping the COMPANY ER schema into a relational database schema.

ER-to-Relational Mapping Algorithm

COMPANY database example

Assume that the mapping will create tables with simple single-valued attributes

Step 1: Mapping of Regular Entity Types

For each regular entity type, create a relation R that includes all the simple attributes of E Called entity relations . Each tuple represents an entity instance.

Step 2: Mapping of Weak Entity Types

For each weak entity type, create a relation R and include all simple attributes of the entity type as attributes of R . Include primary key attribute of owner as foreign key attributes of R

Step 3: Mapping of Binary 1:1 Relationship Types

For each binary 1:1 relationship type . Identify relations that correspond to entity types participating in R . Possible approaches: Foreign key approach. Merged relationship approach.Crossreference or relationship relation approach

Step 4: Mapping of Binary 1:N Relationship Types.

For each regular binary 1:N relationship type. Identify relation that represents participating entity type at N-side of relationship type . Include primary key of other entity type as foreign key in S . Include simple attributes of 1:N relationship type as attributes of S

Alternative approach • Use the relationship relation (cross-reference) option as in the third option for binary 1:1 relationships

Step 5: Mapping of Binary M:N Relationship Types . For each binary M:N relationship type.Create a new relation S. Include primary key of participating entity types as foreign key attributes in S. Include any simple attributes of M:N relationship type

Step 6: Mapping of Multivalued Attributes

For each multivalued attribute ,Create a new relation .Primary key of R is the combination of A and K. If the multivalued attribute is composite, include its simple components

Step 7: Mapping of N-ary Relationship Types

For each n-ary relationship type R , Create a new relation S to represent R

• Include primary keys of participating entity types as foreign keys

• Include any simple attributes as attributes

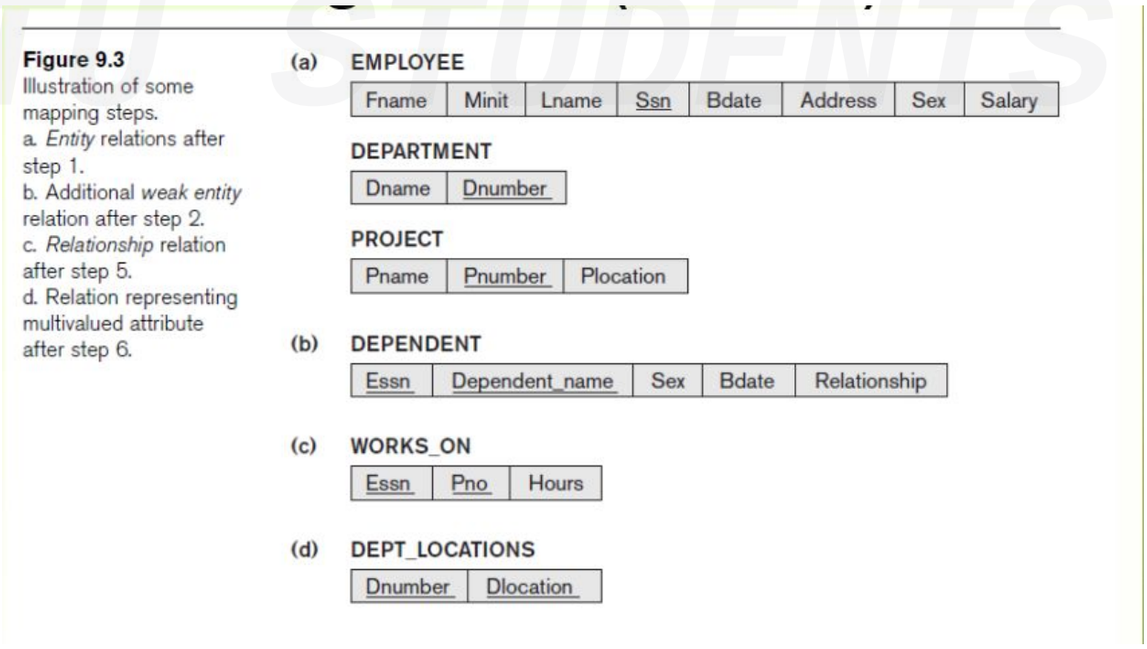


Figure 9.3
Illustration of some mapping steps.
a. *Entity* relations after step 1.
b. Additional *weak entity* relation after step 2.
c. *Relationship* relation after step 5.
d. Relation representing multivalued attribute after step 6.

Table 9.1 Correspondence between ER and Relational Models

ER MODEL	RELATIONAL MODEL
Entity type	*Entity* relation
1:1 or 1:N relationship type	Foreign key (or *relationship* relation)
M:N relationship type	*Relationship* relation and *two* foreign keys
n-ary relationship type	*Relationship* relation and *n* foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

Basic SQL

♣ SQL language ♣

Considered one of the major reasons for the commercial success of relational databases

SQL -Structured Query Language
Statements for data definitions, queries, and updates (both DDL and DML)

SQL Data Definition and Data Types
Terminology:
Table, row, and column used for relational model terms relation, tuple, and attribute

CREATE statement - Main SQL command for data definition
SQL schema - Identified by a schema name
Includes an authorization identifier and descriptors for each element
Schema elements include - Tables, constraints, views, domains, and other constructs.
Each statement in SQL ends with a semicolon.

CREATE SCHEMA statement
e.g; CREATE SCHEMA COMPANY AUTHORIZATION 'Jsmith';
Specify a new relation
Provide name
Specify attributes and initial constraints. Can optionally specify schema:

CREATE TABLE COMPANY.EMPLOYEE ... or CREATE TABLE EMPLOYEE...
The CREATE TABLE Command in SQL
Base tables (base relations) Relation and its tuples are actually created and stored as a file by the DBMS

Virtual relations. Created through the CREATE VIEW statement

Attribute Data Types and Domains in SQL
Basic data types

- Numeric data types •
- Integer numbers: INTEGER, INT, and SMALLINT
- Floating-point (real) numbers: FLOAT or REAL, and DOUBLE PRECISION
- Character-string data types • Fixed length: CHAR(n), CHARACTER(n) • Varying length: VARCHAR(n), CHAR VARYING (n), CHARACTER VARYING(n)
- Bit-string data types • Fixed length: BIT(n) • Varying length: BIT VARYING(n)
- Boolean data type • Values of TRUE or FALSE or NULL
- DATE data type • Ten positions • Components are YEAR, MONTH, and DAY in the form YYYY-MM-DD

- The SELECT-FROM-WHERE Structure of SQL Queries

SELECT <attributes>
FROM <table>
WHERE<condition>

EXAMPLES

QUERY Retrieve the birthdate and address of the employee(s) whose name is 'John B. Smith'

SELECT BDATE, ADDRESS FROM EMPLOYEE WHERE FNAME='John' AND MINIT='B' AND LNAME='Smith';

QUERY Retrieve the name and address of all employees who work for the 'Research' department.
SELECT FNAME, LNAME, ADDRESSλ FROM EMPLOYEE, DEPARTMENT WHERE DNAME='Research' AND DNUMBER=DNO;

Unspecified WHERE-Clause and Use of Asterisk (*)

SELECT * FROM EMPLOYEE WHERE DNO=5;

SELECT * FROM EMPLOYEE, DEPARTMENT WHERE DNAME='Research' AND DNO=DNUMBER;

QUERY Select all combinations of EMPLOYEE SSN and DEPARTMENT DNAME in the database.
SELECT * FROM EMPLOYEE, DEPARTMENT;

Tables as Sets in SQL QUERY
Retrieve the salary of every employee and all distinct salary values.

SELECT ALL SALARY FROM EMPLOYEE;
SELECT DISTINCT SALARYλ FROM EMPLOYEE;

Substring Comparisons, Arithmetic Operators, and Ordering
QUERY:Retrieve all employees whose address is in Houston, Texas.

SELECT FNAME, LNAME FROM EMPLOYEE WHERE ADDRESS LIKE '%Houston,TX%';

QUERY Find all employees who were born during the 1950s.

SELECT FNAME, LNAME λ FROM EMPLOYEE WHERE BDATE LIKE'_ _ 5 _ _ _ _ _ _ _';

QUERY Retrieve all employees in department 5 whose salary is between $30,000 and $40,000.

SELECT *FROM EMPLOYEE WHERE (SALARY BETWEEN 30000 AND 40000) AND DNO = 5;

QUERY Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, first name.

SELECT DNAME, LNAME, FNAME, PNAME λ FROM DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT WHERE DNUMBER=DNO AND SSN=ESSN AND PNO=PNUMBER ORDER BY DNAME, LNAME, FNAME;

Explicit Sets and NULLS in SQL
QUERY Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

SELECT DISTINCT ESSN λ FROM WORKS_ON WHERE PNO IN (1, 2, 3);

QUERY Retrieve the names of all employees who do not have supervisors.
SELECT FNAME, LNAME FROM EMPLOYEE WHERE SUPERSSN IS NULL;

Renaming Attributes and Joined Tables

QUERY For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor. S
ELECT E.LNAME AS EMPLOYEE_NAME, S.LNAME AS SUPERVISOR_NAME FROM EMPLOYEE AS E, EMPLOYEE AS S WHERE E.SUPERSSN=S.SSN;

Aggregate Functions and Grouping

QUERY

Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.

SELECT SUM (SALARY), MAX (SALARY), MIN (SALARY), λ AVG (SALARY) FROM EMPLOYEE;

QUERY

Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

SELECT SUM (SALARY), MAX (SALARY), MIN (SALARY), AVG (SALARY) FROM EMPLOYEE, DEPARTMENT WHERE DNO=DNUMBER AND DNAME='Research';

QUERY

Count the number of distinct salary values in the database.

SELECT COUNT (DISTINCT SALARY) FROM EMPLOYEE;

SELECT LNAME, FNAME FROM EMPLOYEE WHERE (SELECT COUNT (*) FROM DEPENDENT WHERE SSN=ESSN) >= 2;

QUERY

For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project.

SELECT PNUMBER, PNAME, COUNT (*) λ FROM PROJECT, WORKS_ON WHERE PNUMBER=PNO GROUP BY PNUMBER, PNAME HAVING COUNT (*) > 2;

Insert, Delete, and Update Statements in SQL

INSERT COMMAND

INSERT INTO EMPLOYEE VALUES ('Richard', 'K', 'Marini', '653298653', '1962-12-30','98 Oak Forest,Katy,TX', 'M', 37000, '987654321', 4);

INSERT INTO EMPLOYEE (FNAME, LNAME, DNO) VALUES ('Robert', 'Hatcher', 5);

CREATE TABLE DEPTS_INFO (DEPT_NAME VARCHAR(15), NO_OF_EMPS INTEGER, TOTAL_SAL INTEGER); INSERT INTO DEPTS_INFO (DEPT_NAME, λ NO_OF_EMPS, TOTAL_SAL) SELECT DNAME, COUNT (*), SUM (SALARY) FROM (DEPARTMENT JOIN EMPLOYEE ON DNUMBER=DNO) GROUP BY DNAME;

DELETE COMMAND

DELETE FROM EMPLOYEE WHERE LNAME='Brown';
DELETE FROM EMPLOYEE WHERE SSN='123456789';

The UPDATE Command

UPDATE PROJECT SET PLOCATION = 'Bellaire', DNUM = 5 WHERE PNUMBER=10;