Module 2

The Relational Data Model and Relational Database Constraints and Relational Algebra Origins.

Relational Model Concepts

- **Domain**: A (usually named) set/universe of *atomic* values, where by "atomic" we mean simply that, from the point of view of the database, each value in the domain is indivisible (i.e., cannot be broken down into component parts).
  Examples of domains (some taken from page 147):
  - USA_phone_number: string of digits of length ten
  - SSN: string of digits of length nine
  - Name: string of characters beginning with an upper case letter
  - GPA: a real number between 0.0 and 4.0
  - Sex: a member of the set { female, male }
  - Dept_Code: a member of the set { CMPS, MATH, ENGL, PHYS, PSYC, ... }

These are all *logical* descriptions of domains. For implementation purposes, it is necessary to provide descriptions of domains in terms of concrete **data types** (or **formats**) that are provided by the DBMS (such as String, int, boolean), in a manner analogous to how programming languages have intrinsic data types.

- **Attribute**: the *name* of the role played by some value (coming from some domain) in the context of a **relational schema**. The domain of attribute A is denoted dom(A).
- **Tuple**: A tuple is a mapping from attributes to values drawn from the respective domains of those attributes. A tuple is intended to describe some entity (or relationship between entities) in the miniworld.
  As an example, a tuple for a PERSON entity might be
  { Name --> "Rumpelstiltskin", Sex --> Male, IQ --> 143 }
- **Relation**: A (named) set of tuples all of the same form (i.e., having the same set of attributes). The term **table** is a loose synonym. (Some database purists would argue that a table is "only" a physical manifestation of a relation.)
- **Relational Schema**: used for describing (the structure of) a relation. E.g., $R(A_1, A_2, ..., A_n)$ says that R is a relation with *attributes* $A_1, ... A_n$. The **degree** of a relation is the number of attributes it has, here *n*.
  Example: STUDENT(Name, SSN, Address)
  (See Figure 2.1, page 149, for an example of a STUDENT relation/table having several tuples/rows.)
  One would think that a "complete" relational schema would also specify the domain of each

attribute.

- **Relational Database**: A collection of **relations**, each one consistent with its specified relational schema.

Characteristics of Relations

Ordering of Tuples: A relation is a *set* of tuples; hence, there is no order associated with them. That is, it makes no sense to refer to, for example, the 5th tuple in a relation. When a relation is depicted as a table, the tuples are necessarily listed in *some* order, of course, but you should attach no significance to that order. Similarly, when tuples are represented on a storage device, they must be organized in *some* fashion, and it may be advantageous, from a performance standpoint, to organize them in a way that depends upon their content.

Ordering of Attributes: A tuple is best viewed as a mapping from its attributes (i.e., the names we give to the roles played by the values comprising the tuple) to the corresponding values. Hence, the order in which the attributes are listed in a table is irrelevant. (Note that, unfortunately, the set theoretic operations in relational algebra (at least how E&N define them) make implicit use of the order of the attributes. Hence, E&N view attributes as being arranged as a sequence rather than a set.)

Values of Attributes: For a relation to be in *First Normal Form*, each of its attribute domains must consist of atomic (neither composite nor multi-valued) values. Much of the theory underlying the relational model was based upon this assumption. Chapter 10 addresses the issue of including non-atomic values in domains. (Note that in the latest edition of C.J. Date's book, he explicitly argues against this idea, admitting that he has been mistaken in the past.)

The **Null** value: used for *don't know*, *not applicable*.

Interpretation of a Relation: Each relation can be viewed as a **predicate** and each tuple in that relation can be viewed as an assertion for which that predicate is satisfied (i.e., has value **true**) for the combination of values in it. In other words, each tuple represents a fact. Example (see Figure 2.1): The first tuple listed means: There exists a student having name Benjamin Bayer, having SSN 305-61-2435, having age 19, etc.

Keep in mind that some relations represent facts about entities (e.g., students) whereas others represent facts about relationships (between entities). (e.g., students and course sections).

The **closed world assumption** states that the only true facts about the miniworld are those represented by whatever tuples currently populate the database.

Relational Model Notation:

- $R(A_1, A_2, ..., A_n)$ is a relational schema of degree n denoting that there is a relation R having as its attributes $A_1, A_2, ..., A_n$.
- By convention, Q, R, and S denote relation names.

- By convention, *q*, *r*, and *s* denote relation states. For example, *r(R)* denotes one possible state of relation *R*. If *R* is understood from context, this could be written, more simply, as *r*.
- By convention, *t*, *u*, and *v* denote tuples.
- The "dot notation" *R.A* (e.g., STUDENT.Name) is used to qualify an attribute name, usually for the purpose of distinguishing it from a same-named attribute in a different relation (e.g., DEPARTMENT.Name).

Relational Model Constraints and Relational Database Schemas

Constraints on databases can be categorized as follows:

- **inherent model-based:** Example: no two tuples in a relation can be duplicates (because a relation is a set of tuples)
- **schema-based:** can be expressed using DDL; this kind is the focus of this section.
- **application-based:** are specific to the "business rules" of the miniworld and typically difficult or impossible to express and enforce within the data model. Hence, it is left to application programs to enforce.

Elaborating upon **schema-based constraints**:

Domain Constraints:

Each attribute value must be either **null** (which is really a *non-value*) or drawn from the domain of that attribute. Note that some DBMS's allow you to impose the **not null** constraint upon an attribute, which is to say that that attribute may not have the (non-)value **null**.

Key Constraints:

A relation is a *set* of tuples, and each tuple's "identity" is given by the values of its attributes. Hence, it makes no sense for two tuples in a relation to be identical (because then the two tuples are actually one and the same tuple). That is, no two tuples may have the same combination of values in their attributes.

Usually the miniworld dictates that there be (proper) subsets of attributes for which no two tuples may have the same combination of values. Such a set of attributes is called a **superkey** of its relation. From the fact that no two tuples can be identical, it follows that the set of all attributes of a relation constitutes a superkey of that relation.

A **key** is a *minimal superkey*, i.e., a superkey such that, if we were to remove any of its attributes, the resulting set of attributes fails to be a superkey.

Example: Suppose that we stipulate that a faculty member is uniquely identified by *Name* and *Address* and also by *Name* and *Department*, but by no single one of the three attributes mentioned. Then *{ Name, Address, Department }* is a (non-minimal) superkey and each of *{ Name, Address }* and *{ Name, Department }* is a key (i.e., minimal superkey).

Candidate key: any key! (Hence, it is not clear what distinguishes a key from a candidate key.)

Primary key: a key chosen to act as the means by which to identify tuples in a relation. Typically, one prefers a primary key to be one having as few attributes as possible.

Relational Databases and Relational Database Schemas

A **relational database schema** is a set of schemas for its relations together with a set of **integrity constraints**.

A **relational database state/instance/snapshot** is a set of states of its relations such that no integrity constraint is violated.

Entity Integrity, Referential Integrity, and Foreign Keys

Entity Integrity Constraint:

In a tuple, none of the values of the attributes forming the relation's primary key may have the (non-)value **null**. Or is it that at least one such attribute must have a non-null value? In my opinion, E&N do not make it clear.

Referential Integrity Constraint:

A **foreign key** of relation *R* is a set of its attributes intended to be used (by each tuple in *R*) for identifying/referring to a tuple in some relation *S*. (*R* is called the *referencing* relation and *S* the *referenced* relation.) For this to make sense, the set of attributes of *R* forming the foreign key should "correspond to" some superkey of *S*. Indeed, by definition we require this superkey to be the primary key of *S*.

This constraint says that, for every tuple in *R*, the tuple in *S* to which it refers must actually be in *S*. Note that a foreign key may refer to a tuple in the same relation and that a foreign key may be part of a primary key (indeed, for weak entity types, this will always occur). A foreign key may have value **null** (necessarily in all its attributes??), in which case it does not refer to any tuple in the referenced relation.

Semantic Integrity Constraints:

application-specific restrictions that are unlikely to be expressible in DDL. Examples:

- salary of a supervisee cannot be greater than that of her/his supervisor
- salary of an employee cannot be lowered

Update Operations and Dealing with Constraint Violations

For each of the *update* operations (Insert, Delete, and Update), we consider what kinds of constraint violations may result from applying it and how we might choose to react.

Insert:

- domain constraint violation: some attribute value is not of correct domain
- entity integrity violation: key of new tuple is **null**

- key constraint violation: key of new tuple is same as existing one
- referential integrity violation: foreign key of new tuple refers to non-existent tuple Ways of dealing with it: reject the attempt to insert! Or give user opportunity to try again with different attribute values.

Delete:

- referential integrity violation: a tuple referring to the deleted one exists. Three options for dealing with it:
- Reject the deletion
- Attempt to **cascade** (or **propagate**) by deleting any referencing tuples (plus those that reference them, etc., etc.)
- modify the foreign key attribute values in referencing tuples to **null** or to some valid value referencing a different tuple

Update:

- Key constraint violation: primary key is changed so as to become same as another tuple's
- referential integrity violation:
    - foreign key is changed and new one refers to nonexistent tuple
    - primary key is changed and now other tuples that had referred to this one violate the constraint

Transactions:

This concept is relevant in the context where multiple users and/or application programs are accessing and updating the database concurrently. A transaction is a logical unit of work that may involve several accesses and/or updates to the database (such as what might be required to reserve several seats on an airplane flight). The point is that, even though several transactions might be processed concurrently, the end result must be as though the transactions were carried out sequentially. (Example of simultaneous withdrawals from same checking account.) **Relational Model Concepts**

- The model was first introduced by Tod Codd of IBM Research in 1970.
- It uses the concept of a mathematical relation. Hence, the database is a collection of relations.
- A relation can be thought as a table of values, each row in the table represents a collection of related data values.
- In relational model terminology, a row is called a *tuple*, a column header is called an *attribute*.
- A **relation schema** *R*, denoted by $R(A_1,...,A_n)$, is made up of a relation name *R* and a list of attributes $A_1,...,A_n$.
- The domain of A_i is denoted by $dom(A_i)$.
- A relation schema that describes a relation *R* is called the *name* of this relation.
- The **degree** of a relation is the number of attributes in its relation schema

- For example, a relation schema of order 7

STUDENT(Name, SSN, HPhone, Address, WPhone, Age, GPA) describes students. The relation student can be shown as follows:

Fig: The attributes and tuples of a relation STUDENT

Relation name

Attributes

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 Relation

- A *tuple* can be considered as a set of (<attribute>, <value>) pairs. Thus the following two tuples are *identical*:
- t1 = <(Name, B. Bayer),(SSN, 305-61-2435),(HPhone, 3731616),
- (Address, 291 Blue Lane),(WPhone, null),(Age, 23),(GPA, 3.25)>
- t2 = <(HPhone, 3731616),(WPhone, null),(Name, B. Bayer),(Age, 23), (Address, 291 Blue Lane),(SSN, 305-61-2435),(GPA, 3.25)>
- *Tuple* ordering is not a part of relation, that is the following relation is *identical* to that of Table 7.1.

The relation STUDENT with a different order of tuples

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

Relational Model Notation

- A relation schema *R* of degree *n* is denoted by $R(A_1,...,A_n)$
- An *n*-tuple *t* in a relation r(R) is dented by $t = < v_1,...,v_n >$, where
  - v_i is the value corresponding to attribute A_i .
  - Both $t[A_i]$ and $t.A_i$ refer to the value A_i .
- The letters *Q*, *R*, *S* denote relation names.
- The letters *q*, *r*, *s* denote relation states.
- The letters *t*, *u*, *v* denote tuples.
- An attribute *A* can be qualified with the relation name *R* using the dot notation *R.A* -for example, STUDENT.Name or STUDENT.Age.

The Relational Algebra

Operations to manipulate relations.

Used to specify retrieval requests (queries). Query result is in the form of a relation

Relational Operations:

σSELECT and πPROJECT operations.

Set operations: These include UNION U, INTERSECTION | |, DIFFERENCE -, CARTESIAN PRODUCT X.

JOIN operations $\bowtie$.

Other relational operations: DIVISION, OUTER JOIN, AGGREGATE FUNCTIONS.

σ SELECT and πPROJECT

SELECT operation (denoted by σ):

Selects the tuples (rows) from a relation R that satisfy a certain *selection condition* c Form of the operation: σ_c . The condition c is an arbitrary Boolean expression on the attributes of R . Resulting relation has the *same attributes* as R. Resulting relation includes each tuple in r(R) whose attribute values satisfy the condition c

Examples:

$\sigma_{DNO=4}$(EMPLOYEE)

$\sigma_{SALARY>30000}$(EMPLOYEE)

$\sigma_{(DNO=4 \text{ AND } SALARY>25000) \text{ OR } DNO=5}$(EMPLOYEE)

PROJECT operation (denoted by): π

Keeps only certain attributes (columns) from a relation R specified in an *attribute list* L .Form of operation: π_L(R) . Resulting relation has only those attributes of R specified in L .The PROJECT operation eliminates duplicate tuples in the resulting relation so that it remains a mathematical set (no duplicate elements). Duplicate tuples are eliminated by the operation. π

Example: $\pi_{SEX,SALARY}$(EMPLOYEE)

If several male employees have salary 30000, only a single tuple <M, 30000> is kept in the resulting relation.

Figure 7.8 Results of SELECT and PROJECT operations.
(a) $\sigma_{(DNO=4 \text{ AND } SALARY>25000) \text{ OR } (DNO=5 \text{ AND } SALARY>30000)}(EMPLOYEE)$.
(b) $\pi_{LNAME, FNAME, SALARY}(EMPLOYEE)$. (c) $\pi_{SEX, SALARY}(EMPLOYEE)$

(a)

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
Franklin	T	Wong	333445555	1955-12-08	638 Voss,Houston,TX	M	40000	888665555	5
Jennifer		Wallace	987654321	1941-06-20	291 Berry,Bellaire,TX	F	43000	888665555	4
Ramesh		Narayan	666884444	1962-09-15	975 FireOak,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

Set Operations

Binary operations from mathematical set theory:

UNION: $R_1 \cup R_2$, **INTERSECTION**: $R_1 \cap R_2$, **SET DIFFERENCE**: $R_1 - R_2$, **CARTESIAN PRODUCT**: $R_1 \times R_2$.

∪ ∩ For , , -, the operand relations R1(A1, A2, ..., An) and R2(B1, B2, ..., Bn) must have the same number of attributes, and the domains of corresponding attributes must be compatible; that is, dom(Ai) = dom(Bi) for i=1, 2, ..., n. This condition is called union compatibility. The resulting relation for ∪, ∩, or - has the same attribute names as the first operand relation R1 (by convention).

Figure 7.11 Illustrating the set operations union, intersection, and difference. (a) Two union compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∪ INSTRUCTOR. (d) STUDENT − INSTRUCTOR. (e) INSTRUCTOR − STUDENT.

(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

CARTESIAN PRODUCT

$R(A_1, A_2, ..., A_m, B_1, B_2, ..., B_n) \quad R_1(A_1, A_2, ..., A_m) \text{ X } R_2 (B_1, B_2, ..., B_n)$

A tuple t exists in R for each combination of tuples t_1 from R_1 and t_2 from R_2 such that:

$t[A_1, A_2, ..., A_m] = t_1$ and $t[B_1, B_2, ..., B_n] = t_2$

If R_1 has n_1 tuples and R_2 has n_2 tuples, then R will have n_1*n_2 tuples.

CARTESIAN PRODUCT is a *meaningless operation* on its own. It can *combine related tuples* from two relations *if followed by the appropriate SELECT operation*.

Example: Combine each DEPARTMENT tuple with the EMPLOYEE tuple of the manager.

DEP_EMP DEPARTMENT X EMPLOYEE

DEPT_MANAGER $_{\text{MGRSSN=SSN}}$(DEP_EMP)

Figure 7.12 An illustration of the CARTESIAN PRODUCT operation.

FEMALE_EMPS	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle,Spring,TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry,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

JOIN Operations

THETA JOINS: Similar to a CARTESIAN PRODUCT followed by a SELECT. The condition c is called a *join condition*.

$R(A_1, A_2, ..., A_m, B_1, B_2, ..., B_n) \quad R_1(A_1, A_2, \bowtie, A_m) \quad {}_c R_2 (B_1, B_2, ..., B_n)$

EQUIJOIN: The join condition c includes one or more *equality comparisons* involving attributes from R_1 and R_2. That is, c is of the form:

$(A_i=B_j)$ AND ... AND $(A_h=B_k)$; $1 \leq i,h \leq m$, $1 \leq j,k \leq n$

In the above EQUIJOIN operation:

$A_i, ..., A_h$ are called the **join attributes** of R_1

$B_j, ..., B_k$ are called the **join attributes** of R_2

Example of using EQUIJOIN:

Retrieve each DEPARTMENT's name and its manager's name:

$T \leftarrow DEPARTMENT \bowtie_{MGRSSN = SSN} EMPLOYEE$

$RESULT \leftarrow \pi_{DNAME,FNAME,LNAME} (T)$

NATURAL JOIN (*):

In an EQUIJOIN $R \leftarrow R_1 \bowtie_c R_2$, the join attribute of R_2 appear *redundantly* in the result relation R. In a NATURAL JOIN, the *redundant join attributes* of R_2 are *eliminated* from R. The equality condition is *implied* and need not be specified.

$R \leftarrow R_1 *_{\text{(join attributes of R1),(join attributes of R2)}} R_2$

Example: Retrieve each EMPLOYEE's name and the name of the DEPARTMENT he/she works for:

$T \leftarrow EMPLOYEE *_{(DNO),(DNUMBER)} DEPARTMENT$

$RESULT \leftarrow \pi_{FNAME,LNAME,DNAME} (T)$

If the join attributes *have the same names* in both relations, they *need not be specified* and we can write $R \leftarrow R_1 * R_2$.

Example: Retrieve each EMPLOYEE's name and the name of his/her SUPERVISOR:

$SUPERVISOR(SUPERSSN,SFN,SLN) \leftarrow \pi_{SSN,FNAME,LNAME}(EMPLOYEE)$

$T \leftarrow EMPLOYEE * SUPERVISOR$

$RESULT \leftarrow \pi_{FNAME,LNAME,SFN,SLN}(T)$

Figure 7.14 An illustration of the NATURAL JOIN operation. (a) PROJ_DEPT ← PROJECT * DEPT. (b) DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS.

(a)

PROJ_DEPT	PNAME	PNUMBER	PLOCATION	DNUM	DNAME	MGRSSN	MGRSTARTDATE
	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	MGRSSN	MGRSTARTDATE	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

Note: In the *original definition* of NATURAL JOIN, the join attributes were *required* to have the same names in both relations.

There can be a *more than one set of join attributes* with a *different meaning* between the same two relations. For example:

JOIN ATTRIBUTES RELATIONSHIP

EMPLOYEE.SSN= EMPLOYEE *manage* DEPARTMENT.MGRSSN the DEPARTMENT EMPLOYEE.DNO= EMPLOYEE *works for* DEPARTMENT.DNUMBER the DEPARTMENT

Example: Retrieve each EMPLOYEE's name and the name of the DEPARTMENT he/she works for:

T← EMPLOYEE $\bowtie_{DNO=DNUMBER}$ DEPARTMEN

RESULT ← $\pi_{FNAME,LNAME,DNAME}$(T)

A relation can have a *set of join attributes* to join it with *itself* :

JOIN ATTRIBUTES	RELATIONSHIP
EMPLOYEE(1).SUPERSSN= EMPLOYEE(2).SSN	EMPLOYEE(2) *supervises* EMPLOYEE(1)

One can *think of this* as joining *two distinct copies* of the relation, although only one relation actually exists In this case, *renaming* can be useful.

Example: Retrieve each EMPLOYEE's name and the name of his/her SUPERVISOR:

SUPERVISOR(SSSN,SFN,SLN) ← $\pi_{SSN,FNAME,LNAME}$(EMPLOYEE)

T←EMPLOYEE $\bowtie_{SUPERSSN=SSSN}$ SUPERVISOR

RESULT ← $\pi_{FNAME,LNAME,SFN,SLN}$(T)

Complete Set of Relational Algebra Operations:

All the operations discussed so far can be described as a sequence of *only* the operations SELECT, PROJECT, UNION, SET DIFFERENCE, and CARTESIAN PRODUCT.

Hence the set { σ , π, , - , X } is called a *complete set* of relational algebra operations. Any query language *equivalent to* these operations is called **relationally complete**.

For database applications, additional operations are needed that were not part of the *original* relational algebra. These include:

1. Aggregate functions and grouping.
2. OUTER JOIN and OUTER UNION. **Additional**

Relational Operations AGGREGATE FUNCTIONS ( ) $\Im$

Functions such as SUM, COUNT, AVERAGE, MIN, MAX are often applied to sets of values or sets of tuples in database applications

<grouping attributes> $\Im_{\text{<function list>}}$(R) The grouping attributes are optional

Example 1: Retrieve the average salary of all employees (no grouping needed):

ρ (AVGSAL) $\leftarrow \Im_{\text{AVERAGE SALARY}}$ (EMPLOYEE)

Example 2: For each department, retrieve the department number, the number of employees, and the average salary (in the department):

ρ (DNO,NUMEMPS,AVGSAL) $\Im$ DNO$\leftarrow_{\text{COUNT SSN, AVERAGE SALARY}}$ (EMPLOYEE)

DNO is called the *grouping attribute* in the above example

Figure 7.16 An illustration of the AGGREGATE FUNCTION operation. (a) R(DNO, NO_OF_EMPLOYEES, AVERAGE_SAL) $\leftarrow$ $_{\text{DNO}}\Im_{\text{COUNT SSN,AVERAGE SALARY}}$ (EMPLOYEE). (b) $_{\text{DNO}}\Im_{\text{COUNT SSN,AVERAGE SALARY}}$(EMPLOYEE). (c) $\Im_{\text{COUNT SSN,AVERAGE SALARY}}$(EMPLOYEE).

(a)

	DNO	NO_OF_EMPLOYEES	AVERAGE_SAL
	5	4	33250
	4	3	31000
	1	1	55000

OUTER JOIN

In a regular EQUIJOIN or NATURAL JOIN operation, tuples in R_1 or R_2 that do not have matching tuples in the other relation *do not appear in the result*

Some queries require all tuples in R_1 (or R_2 or both) to appear in the result When no matching tuples are found, **null**s are placed for the missing attributes **LEFT OUTER JOIN**: R_1 X R_2 lets every tuple in R_1 appear in the result **RIGHT OUTER JOIN**: R_1 X R_2 lets every tuple in R_2 appear in the result **FULL OUTER JOIN**: R_1 X R_2 lets every tuple in R_1 or R_2 appear in the result