Chapter 7 – The Relational Data Model, Relational Constraints, and the Relational Algebra

Origins of the Relational Model

- Developed by British computer scientist E.F. (Ted) Codd of IBM in a seminal paper in 1970 (*A Relational Model for Large Shared Data Banks*, Communications of the ACM, June 1970)
- Considered ingenious but impractical in 1970
- Conceptually simple
- Computers lacked power to implement the relational model
- Today, microcomputers can run sophisticated relational database software

Relational Model Concepts

- **Domain**: A set/universe of *atomic* values, where by "atomic" we mean indivisible (i.e., cannot be broken down into component parts).

  Examples of domains

  - 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 --> "Keerthy",   Sex --> Male,   IQ --> 786 }
  ```

- **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.
- **Relational Schema**: used for describing (the structure of) a relation. E.g., $R(A_1, A_2, \ldots, A_n)$ 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)`

  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.
- **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 Elmasri& Navathe define them) make implicit use of the order of the attributes. Hence, Elmasri& Navathe 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.
- **Interpretation of a Relation**: Each relation can be viewed as a **predicate** and each tuple an assertion that that predicate is satisfied (i.e., has value **true**) for the combination of values in it. In other words, each tuple represents a fact.

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.

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.

Superkey of a relation is subsets of attributes, for which no two tuples can have the same combination of values. 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 (i.e., any *minimal superkey)*

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.

Entity Integrity, Referential Integrity, and Foreign Keys

Entity Integrity Constraint: entity integrity constraint states that no primary key value can be null.

Referential Integrity Constraint: it is specified between two relations and is used to maintain the consistency among tuples of two relations. Referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation. For ex, the attribute DNO of EMPLOYEE gives the department number for which each employee works; hence, its value in every EMPLOYEE tuple must match the DNUMBER value of some tuple in the DEPARTMENT relation.

Foreign Key: A set of attributes FK in relation schema R1 is a foreign key of R1 that references relation R2 if it satisfies following two rules

1. The attributes in FK have the same domain(s) as the primary key attributes PK of R2;the attributes FK are said to reference or refer to the relation R2
2. A value of FK in a tuple t1 of the current state r1(R1) either occurs as a value of PK for some tuple t2 in the current state r2(R2) or is null. In the former case ,we

have t1[FK]=t2[PK], and we say that the tuple t1 references or refers to the tuple t2. R1 is called **referencing relation** and R2 is called **referenced relation.**

The conditions for a foreign key, given above, specify a referential integrity constraint between the 2 relation schemas R1 and R2.

Schema and State of a relation

- The **Schema** is a description of a Relation:
  - Denoted by R(A1, A2, ...An)
  - R is the **name** of the relation
  - The **attributes** of the relation are A1, A2, ..., An
- The **relation state r(R)** refers to the present set of data/tuples in the relation.

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.

Relational Algebra:

A brief introduction

- Relational algebra and relational calculus are formal languages associated with the relational model.
- Informally, relational algebra is a (high-level) procedural language and relational calculus a non-procedural language.
- However, formally both are equivalent to one another.
- A language that produces a relation that can be derived using relational calculus is relationally complete.
- Relational algebra operations work on one or more relations to define another relation without changing the original relations.
- Both operands and results are relations, so output from one operation can become input to another operation.
- Allows expressions to be nested, just as in arithmetic. This property is called closure.
- 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: σ (sigma))
- PROJECT (symbol: π (pi))
- RENAME (symbol: ρ (rho))

Relational Algebra Operations from Set Theory

- UNION (∪)
- INTERSECTION (∩)
- 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

Relational Algebra Operations:

Unary Relational Operations: SELECT

- The SELECT operation (denoted by σ (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:

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

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

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

In general, the *select* operation is denoted by $\sigma_{\text{<selection condition>}}(R)$ where

- the symbol σ (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 σ <selection condition>(R) produces a relation S that has the same schema (same attributes) as R
- SELECT is commutative:

$$\sigma_{\text{<condition1>}}(\sigma_{\text{<condition2>}}(R)) = \sigma_{\text{<condition2>}}(\sigma_{\text{<condition1>}}(R))$$

- Because of commutative property, a cascade (sequence) of SELECT operations may be applied in any order:

$$\sigma_{\text{<cond1>}}(\sigma_{\text{<cond2>}}(\sigma_{\text{<cond3>}}(R)) = \sigma_{\text{<cond2>}}(\sigma_{\text{<cond3>}}(\sigma_{\text{<cond1>}}(R)))$$

- A cascade of SELECT operations may be replaced by a single selection with a conjunction of all the conditions:

$$\sigma_{\text{<cond1>}}(\sigma_{\text{<cond2>}}(\sigma_{\text{<cond3>}}(R)) = \sigma_{\text{<cond1>AND<cond2>AND<cond3>}}(R)))$$

- 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

Unary Relational Operations: PROJECT

- PROJECT Operation is denoted by π (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:

$$\pi_{\text{LNAME, FNAME, SALARY}}(\text{EMPLOYEE})$$

- The general form of the *project* operation is:

$$\pi_{\text{<attribute list>}}(R)$$

  - π (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*

PROJECT Operation Properties

- The number of tuples in the result of projection π<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

$\pi_{<list1>}(\pi_{<list2>}(R)) = \pi_{<list1>}(R)$ as long as <list2> contains the attributes in <list1>

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 (see JOIN operation later)
- The general RENAME operation ρ can be expressed by any of the following forms:
  - $\rho_{S(B1, B2, \ldots, Bn)}(R)$ changes both:
    - the relation name to S, *and*
    - the column (attribute) names to B1, B2, …..Bn
  - $\rho_{S}(R)$ changes:
    - the *relation name* only to S
  - $\rho_{(B1, B2, \ldots, Bn)}(R)$ changes:
    - the *column (attribute) names* only to B1, B2, …..Bn
- For convenience, we also use a *shorthand* for renaming attributes in an intermediate relation:
  - If we write:
    - RESULT ← $\pi_{FNAME, LNAME, SALARY}$ (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) ←

$\rho_{RESULT(F.M.L.S.B,A,SX,SAL,SU,DNO)}(DEP5\_EMPS)$

The 10 attributes of DEP5_EMPS are *renamed* to F, M, L, S, B, A, SX, SAL, SU, DNO, respectively

Relational Algebra Operations from Set Theory

UNION Operation

- Binary operation, denoted by ∪
- The result of R ∪ 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:

$$DEP5\_EMPS \leftarrow \sigma_{DNO=5}(EMPLOYEE)$$
$$RESULT1 \leftarrow \pi_{SSN}(DEP5\_EMPS)$$
$$RESULT2(SSN) \leftarrow \pi_{SUPERSSN}(DEP5\_EMPS)$$
$$RESULT \leftarrow RESULT1 \cup RESULT2$$

The union operation produces the tuples that are in either RESULT1 or RESULT2 or both

- Type Compatibility of operands is required for the binary set operation UNION ∪, (also for INTERSECTION ∩, 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 R1∪R2 (also for R1∩R2, or R1–R2, see next slides) has the same attribute names as the *first* operand relation R1 (by convention)

INTERSECTION Operation

- 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"

DIFFERENCE Operation

- 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"

Some properties of UNION, INTERSECT, and DIFFERENCE

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

CARTESIAN 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 nR tuples (denoted as |R| = nR), and S has nS tuples, then R x S will have nR * nS 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):

$$\text{FEMALE\_EMPS} \leftarrow \sigma_{\text{SEX='F'}}(\text{EMPLOYEE})$$
$$\text{EMPNAMES} \leftarrow \pi_{\text{FNAME, LNAME, SSN}}(\text{FEMALE\_EMPS})$$
$$\text{EMP\_DEPENDENTS} \leftarrow \text{EMPNAMES } \mathbf{X} \text{ DEPENDENT}$$

EMP_DEPENDENTS will contain every combination of EMPNAMES and DEPENDENT whether or not they are actually related

- To keep only combinations where the DEPENDENT is related to the EMPLOYEE, we add a SELECT operation as follows

Example (meaningful):

FEMALE_EMPS ← σ SEX='F'(EMPLOYEE)

EMPNAMES ← π FNAME, LNAME, SSN (FEMALE_EMPS)

EMP_DEPENDENTS ← EMPNAMES X DEPENDENT

ACTUAL_DEPS ← σ SSN=ESSN (EMP_DEPENDENTS)

RESULT ← π FNAME, LNAME, DEPENDENT_NAME (ACTUAL_DEPS)

RESULT will now contain the name of female employees and their dependents

JOIN Operation

- 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*.

Example: Suppose that we want to retrieve the name of the manager of each department.

- To get the manager's name, we need to combine each DEPARTMENT tuple with the EMPLOYEE tuple whose SSN value matches the MGRSSN value in the department tuple.
- We do this by using the join operation.

DEPT_MGR ← DEPARTMENT $\bowtie_{\text{MGRSSN=SSN}}$ EMPLOYEE

- MGRSSN=SSN is the join condition
- Combines each department record with the employee who manages the department
- The join condition can also be specified as DEPARTMENT.MGRSSN= EMPLOYEE.SSN

Consider the following JOIN operation:

R(A1, A2, . . ., An) $\bowtie_{\text{R.Ai=S.Bj}}$ 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—r from R and s from S, but *only if they satisfy the join condition* r[Ai]=s[Bj]
- Hence, if R has nR tuples, and S has nS tuples, then the join result will generally have *less than* nR * nS tuples.
- Only related tuples (based on the join condition) will appear

Some properties of JOIN

- The general case of JOIN operation is called a Theta-join:
- The join condition is called *theta*
- *Theta* can be any general Boolean expression on the attributes of R and S; for example:
  R.Ai<S.Bj AND (R.Ak=S.Bl OR R.Ap<S.Bq)
- Most join conditions involve one or more equality conditions "AND"ed together; for example:
  R.Ai=S.Bj AND R.Ak=S.Bl AND R.Ap=S.Bq

EQUIJOIN Operation

- 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.
  - The JOIN seen in the previous example was an EQUIJOIN.

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)

DIVISION Operation

- The division operation is applied to two relations
- R(Z) ÷ S(X), where X is a subset of Z. Let Y = Z - X 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 tR appear in R with tR [Y] = t, and with
- tR [X] = ts *for every tuple* ts 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

(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) $T \leftarrow R \div S$.

Additional Relational Operations: Aggregate Functions and Grouping

- A type of request that cannot be expressed in the basic relational algebra is to specify mathematical **aggregate functions** on collections of values from the database.
- Examples of such functions include retrieving the average or total salary of all employees or the total number of employee tuples.
- These functions are used in simple statistical queries that summarize information from the database tuples.
- Common functions applied to collections of numeric values include

  SUM, AVERAGE, MAXIMUM, and MINIMUM.

- The COUNT function is used for counting tuples or values.
- Use of the Aggregate Functional operation $\mathcal{F}$
  - $\mathcal{F}_{\text{MAX Salary}}(\text{EMPLOYEE})$ retrieves the maximum salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{MIN Salary}}(\text{EMPLOYEE})$ retrieves the minimum Salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{SUM Salary}}(\text{EMPLOYEE})$ retrieves the sum of the Salary from the EMPLOYEE relation

- $\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}(\text{EMPLOYEE})$ computes the count (number) of employees and their average salary
  - Note: count just counts the number of rows, without removing duplicates
- The previous examples all summarized one or more attributes for a set of tuples
- Grouping can be combined with Aggregate Functions

Example: For each department, retrieve the DNO, COUNT SSN, and AVERAGE SALARY

- A variation of aggregate operation $\mathcal{F}$ allows this:
  - Grouping attribute placed to left of symbol
  - Aggregate functions to right of symbol

$$\text{DNO}\ \mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}(\text{EMPLOYEE})$$

Above operation groups employees by DNO (department number) and computes the count of employees and average salary per department

The OUTER JOIN Operation

- In NATURAL JOIN and EQUIJOIN, tuples without a *matching* (or *related*) tuple are eliminated from the join result
  - Tuples with null in the join attributes are also eliminated
  - This amounts to loss of information.

- A set of operations, called OUTER joins, can be used when we want to keep all the tuples in R, or all those in S, or all those in both relations in the result of the join, regardless of whether or not they have matching tuples in the other relation.
- The left outer join operation keeps <u>every tuple</u> in the first or left relation R in R ⟕ S; if no matching tuple is found in S, then the attributes of S in the join result are filled or "padded" with null values.
- A similar operation, right outer join, keeps every tuple in the second or right relation S in the result of R ⟖ S.
- A third operation, full outer join, denoted by ⟗ keeps all tuples <u>in both the left and the right relations</u> when no matching tuples are found, padding them with null values as needed.

OUTER UNION Operation

- The outer union operation was developed to take the union of tuples from two relations if the relations are *not type compatible*.
- This operation will take the union of tuples in two relations R(X, Y) and S(X, Z) that are **partially compatible**, meaning that only some of their attributes, say X, are type compatible.

    - The attributes that are type compatible are represented only once in the result, and those attributes that are not type compatible from either relation are also kept in the result relation T(X, Y, Z).
- Example: An outer union can be applied to two relations whose schemas are STUDENT(Name, SSN, Department, Advisor) and INSTRUCTOR(Name, SSN, Department, Rank).
    - Tuples from the two relations are matched based on having the same combination of values of the shared attributes— Name, SSN, Department.
    - If a student is also an instructor, both Advisor and Rank will have a value; otherwise, one of these two attributes will be null.
    - The result relation STUDENT_OR_INSTRUCTOR will have the following attributes:

STUDENT_OR_INSTRUCTOR (Name, SSN, Department, Advisor, Rank)

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 $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$