The Relational Algebra and The Relational Calculus

Acknowledgement:
Most of these slides have been adapted from slides made available by Pearson/Addison Wesley to accompany Elmasri and Navathe's textbook Fundamentals of Database Systems

Chapter Outline

- Relational Algebra
  - Unary Relational Operations
  - Binary Relational Operations From Set Theory
  - Other Binary Relational Operations
  - Additional Relational Operations
- Examples of Queries
- Relational Calculus
  - Tuple Relational Calculus

Relational Algebra Overview

- 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)

Relational Algebra Overview (2)

- 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 Operations

- 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 ($\cup$), INTERSECTION ($\cap$), DIFFERENCE or MINUS (–)
    - CARTESIAN PRODUCT ($\times$)
  - Binary Relational Operations
    - JOIN ($\bowtie$ – several variations of JOIN exist)
    - DIVISION ($\div$)
  - Additional Relational Operations
    - OUTER JOINS ($\mathbb{M}$)
    - AGGREGATE FUNCTIONS (these compute summary of information; for example: SUM, COUNT, AVG, MIN, MAX)

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:

$$\sigma_{SALARY > 30,000} (EMPLOYEE)$$

Unary Relational Operations: SELECT (2)

- 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

Unary Relational Operations: SELECT (3)

- SELECT Operation Properties
  - The SELECT operation $\sigma_{\text{<selection condition>}}(R)$ produces a relation S that has the same schema (same attributes) as R
  - SELECT σ is commutative:
    - $\sigma_{\text{<condition1>}}(\sigma_{\text{< condition2>}}(R)) = \sigma_{\text{<condition2>}}(\sigma_{\text{< condition1>}}(R))$
  - Because of commutativity 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:

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

Unary Relational Operations: PROJECT (2)

- 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*
  - This is because the result of the *project* operation must be a *set of tuples* and mathematical sets *do not allow* duplicate elements

Unary Relational Operations: PROJECT (3)

- PROJECT Operation Properties
  - The number of tuples in the result of projection $\pi_{<list>}(R)$ is always less than 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
  - A cascade of PROJECT operations may be replaced by a single projection with the final list
    - $\pi_{<list1>}(\pi_{<list2>}(R)) = \pi_{<list1>}(R)$ as long as <list2> contains the attributes in <list1>

Relational Algebra Expressions

- We may want to apply several relational algebra operations one after the other
  - Either we can write the operations as a single **relational algebra expression** by nesting the operations, or
  - We can apply one operation at a time and create **intermediate result relations**
- In the latter case, we must give names to the relations that hold the intermediate results

Relational Algebra Expressions (2)

- Example: Retrieve the first name, last name, and salary of all employees who work in department number 5
  - We can write a single relational algebra expression as follows:

    $\pi_{FNAME, LNAME, SALARY}(\sigma_{DNO=5}(EMPLOYEE))$

  - Or we can explicitly show the sequence of operations, giving a name to each intermediate relation:

    $DEP5\_EMPS \leftarrow \sigma_{DNO=5}(EMPLOYEE)$

    $RESULT \leftarrow \pi_{FNAME, LNAME, SALARY}(DEP5\_EMPS)$

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 (see Cartesian Product and JOIN operations later)

Unary Relational Operations: RENAME (2)

- 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 attribute names to B1, B2, …, Bn
      - Some attributes may have their names maintained
  - $\rho_S(R)$ changes:
    - the relation name only to S
  - $\rho_{(B1,\ B2,\ \ldots,\ Bn\ )}(R)$ changes:
    - the attribute names only to B1, B2, …, Bn

Unary Relational Operations: RENAME (3)

- For convenience, we also use a shorthand for renaming attributes in an intermediate relation:
  - RESULT(FNAME, LNAME, SAL) $\leftarrow \pi_{\text{FNAME, LNAME, SALARY}}$ (DEP5_EMPS)
- Other authors propose a simplified notation for RENAME
  - RESULT $\leftarrow \pi_{\text{FNAME, LNAME, SALARY}\rightarrow\text{SAL}}$ (DEP5_EMPS)
  - RESULT $\leftarrow \rho_{\text{SALARY}\rightarrow\text{SAL}} (\pi_{\text{FNAME, LNAME, SALARY}}$ (DEP5_EMPS))

Binary Relational Operations from Set Theory: UNION

- UNION is a 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)

Binary Relational Operations from Set Theory: Type Compatibility

- Type Compatibility of operands is required for the binary set operation UNION ∪, (also for INTERSECTION ∩, and SET DIFFERENCE –, see next slides)
- 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)

Binary Relational Operations from Set Theory: INTERSECTION

- 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 two operand relations R and S must be “type compatible”

Binary Relational Operations from Set Theory: SET DIFFERENCE

- 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 two operand relations R and S must be “type compatible”

Binary Relational Operations from Set Theory: Some Properties

- 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 set difference operation is not commutative; that is, in general
  - $R - S \neq S - R$

Binary Relational Operations from Set Theory: CARTESIAN PRODUCT

- CARTESIAN (or CROSS) PRODUCT (×)
  - This operation is used to combine tuples from two relations in a combinatorial fashion
  - Denoted by R(A1, A2, ..., An) × 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×S will have $n_R * n_S$ tuples
  - The two operands do NOT have to be "type compatible"

Binary Relational Operations from Set Theory: CARTESIAN PRODUCT

- Generally, CROSS PRODUCT is not a meaningful operation
  - However, the sequence of CARTESIAN PRODUCT followed by SELECT is used quite commonly to identify and select related tuples from two relations
  - This sequence of operations is very important for any relational database with more than a single relation, because it allows us combine related tuples from various relations
  - This sequence is so useful that it was embedded into a single operation: JOIN
    - There are many variations of the JOIN operation

Other Binary Relational Operations: JOIN

- The JOIN Operation (denoted by ⋈)
  - The general form of a join operation on two relations R(A1, A2, ..., An) and S(B1, B2, ..., Bm) is:

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

  - R and S are relations (that might result from general relational algebra expressions)
  - The join condition is a Boolean (conditional) expression specified on the attributes in the cartesian product between relations R and S
    - The join condition has the same form of a select condition (i.e., it may include conjunctions and/or disjunctions of terms, etc.), where the terms usually have the form Ai θ Bi
    - If θ is always =, the operation is called an EQUIJOIN
      - Otherwise, it is called as THETA JOIN (or θ-JOIN)

Other Binary Relational Operations: JOIN (2)

- Given a JOIN R(A1, A2, ..., An) $\bowtie_{R.Ai=S.Bj}$ 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 n_R tuples, and S has n_S tuples, then the join result will generally have *less than* $n_R * n_S$ tuples.
  - Only related tuples (based on the join condition) will appear in the result

Other Binary Relational Operations: NATURAL JOIN

- NATURAL JOIN Operation (denoted by *)
  - Another variation of JOIN called NATURAL JOIN 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

Other Binary Relational Operations: DIVISION

- The DIVISION operation is applied to two relations
  - R(Z) ÷ S(X), where X is a subset of Z
  - It is usually employed to represent queries that require the notion of *for all*
- Let Y = Z – X (and hence Z = X ∪ 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

Summary of the Relational Operations

Table 8.1 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 $R_1 * R_2$

Summary of the Relational Operations (2)

Table 8.1 Operations of Relational Algebra

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

Complete Set of Relational Operations

- The set of operations including SELECT σ, PROJECT π , UNION $\cup$, DIFFERENCE –, RENAME ρ, and CARTESIAN PRODUCT $\times$ is called a *complete set* because any other relational algebra expression can be expressed by a combination of these five operations.
- For example:
  - $R \cap S = (R \cup S) - ((R - S) \cup (S - R))$
  - $R \bowtie_{\text{<join condition>}} S = \sigma_{\text{<join condition>}} (R \times S)$

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
    - COUNT, SUM, AVERAGE, MAXIMUM, and MINIMUM

Additional Relational Operations: Aggregate Functions and Grouping

- Aggregate functions using the operator $\mathcal{I}$ (*script F*) as follows:

  <grouping attributes> $\mathcal{I}$ <function list> (R)

  - The grouping attributes are attributes from relation R
  - The function list includes aggregation functions involving attributes from R
    - Examples: AVG(weight), COUNT(*), SUM(balance+overdraft_limit), COUNT(DISTINCT(location))
  - The resulting relation has as **attributes** the grouping attributes (if any) plus one attribute for storing the result of each aggregate function and as **tuples** one tuple for each distinct group with the corresponding aggregates
    - Important: Nulls are skipped in the computation of aggregation functions

Additional Relational Operations: OUTER JOIN

- 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
- 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
- There are three variations:
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN

Additional Relational Operations: OUTER JOIN (2)

- The LEFT OUTER JOIN operation (⟕)
  - Keeps every tuple in the first or left relation R in R⟕$_{\text{<join condition>}}$ 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
- The RIGHT OUTER JOIN (⟖)
  - Keeps every tuple in the second or right relation S in the result of R⟖$_{\text{<join condition>}}$ S, padding with null values as needed
- The FULL OUTER JOIN (⟗)
  - R⟗$_{\text{<join condition>}}$ S keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed

Examples of Queries

1) Retrieve the name and address of all employees who work for the ‘Research’ department
2) Retrieve the names of department managers who have no dependents and the name of the department each of them manage
3) Retrieve the social security numbers and salaries of all employees who either work in department (number) 5 or directly supervise an employee who works in department 5
4) Retrieve the names of employees who work on a project controlled by department 5
5) Retrieve the names of employees who work on all the projects controlled by department 5
6) Retrieve the names of the employees whose salary is the greatest in the department they work, per sex
7) Retrieve the names of all projects which do not have women working on them
8) Retrieve the names of all projects whose the total salary cost is greater than or equal to $100,000, given that the salary cost of an employee in a project is the fraction of his/her salary according to the number of hours per week he/she works on the project (assume employee positions of 40 hours per week), and that the total salary cost of a project is calculated based on the salary cost of all employees who work on the project

A Database State for Company

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

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** or **declarative** language while 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 (conversely, the domain relational calculus is based on domain 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

$$\{t \mid COND(t)\}$$

  - 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)

Tuple Relational Calculus (2)

- 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