Date

Chapter 6:

Relational Algebra and Relational Calculus

Relational Algebra

→ The basic set of operations for the relational model is the relational algebra. These operations enable a user to specify basic retrieval requests as relational algebra expressions.

→ Relational algebra operations are used in DBMS query optimization. SQL is converted to relational operations for optimization and processing.

→ Input to a relational algebra operation is one or more relations. The result of an operation is a new relation, derived from the input relations.

→ Relational Algebra consists of several groups of operations.

1) Unary Relational Operations (one input table)
   - SELECT (σ)
   - PROJECT (π)
   - RENAME (ρ)

2) Relational Algebra Operations from Set Theory
   - UNION ($\cup$)
   - INTERSECTION ($\cap$)
   - DIFFERENCE ($-$)
   - CARTESIAN PRODUCT ($\times$)

3) Binary Relational Operations (two input tables)
   - JOIN
   - DIVISION

4) Additional Relational Operations
   - OUTER JOINS
   - OUTER UNION
   - AGGREGATE FUNCTIONS (SUM, MIN, MAX, AVG, COUNT)

PROJECT (π)

→ The PROJECT operation selects certain columns from the table and discards the other columns.

→ The result of the PROJECT operation can be visualized as a vertical partition of the

Date

relation into two relations: one which contains needed columns and other that contains the discarded columns.

→ The PROJECT operation removes any duplicate tuples, so the result of the PROJECT operation is a set of distinct tuples and hence a valid relation. This is called duplicate elimination.

→ The number of tuples in a relation resulting from a PROJECT operation is always less than or equal to the number of tuples in R.

If the projection list is a superkey of R (it includes some key of R), the resulting relation has the same number of tuples as R.

→ Commutativity does not hold on PROJECT.

$$\pi_{<list1>}(\pi_{<list2>}(R)) = \pi_{<list1>}(R)$$

as long as <list2> contains the attributes in <list1> otherwise L.H.S is incorrect.

→ General form of PROJECT operation is

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

- π used to represent PROJECT
- <attribute list> is desired list of attributes in relation R

Example:

List each employee's first and last name and salary:

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

→ $\pi_{sex, salary}(EMPLOYEE)$ corresponds to the following:

```
SELECT DISTINCT Sex, Salary
FROM EMPLOYEE
```

SELECT (σ)

→ The SELECT operation is used to choose a subset of the tuples from a relation that satisfies a selection condition.

→ The selection works as a filter. It keeps only those tuples that satisfy the qualifying condition while the other tuples are discarded.

Mujahid
CamScanner

Date

→ In general, the SELECT operation is denoted by

$$\sigma_{\langle \text{selection condition} \rangle} (R)$$

- σ is used to represent the SELECT operation.
- the selection condition is a Boolean expression specified on the attributes of R.

→ The SELECT operator is unary, it is applied to single relation. Moreover, the selection operation is applied to each tuple individually, hence selection condition cannot involve more than one tuple.

→ SELECT operation is commutative that is

$$\sigma_{\langle cond1 \rangle} (\sigma_{\langle cond2 \rangle} (R))$$
$$=$$
$$\sigma_{\langle cond2 \rangle} (\sigma_{\langle cond1 \rangle} (R))$$

Because of commutativity, a cascade (sequence) of SELECT operations may be applied in any order.

→ We can always combine a cascade (or sequence) of SELECT operations into a single SELECT operation with a conjunctive AND condition; that is

$$\sigma_{\langle cond1 \rangle} (\sigma_{\langle cond2 \rangle} (\ldots (\sigma_{\langle condn \rangle} (R))))$$
$$=$$
$$\sigma_{\langle cond1 \rangle \text{ AND } \langle cond2 \rangle \text{ AND } \ldots \text{ AND } \langle condn \rangle} (R)$$

→ In SQL, the SELECT condition is typically specified in the WHERE clause of query as given.

Example:

$$\sigma_{Dno = 4 \text{ AND } Salary > 25000} (EMPLOYEE)$$

would correspond to SQL as

```
SELECT   *
FROM     EMPLOYEE
WHERE    Dno = 4 AND
         Salary > 25000;
```

→ 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.

Date

Relational Algebra Expression

→ To apply several relational algebra operations, we

- Either write the operations as a single relational algebra expression by nesting the operations in parenthesis.
- Or, we apply one operation at a time and create intermediate result relations. In this case, we must give names to the relations that hold the intermediate results.

Example:

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

- Single relational algebra expression:

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

- Sequence of operations, giving a name to each

$$DEP5_EMP \leftarrow \sigma_{DNO=5}(Employee)$$

$$RESULT \leftarrow \pi_{FNAME, LNAME, SALARY}(DEP5_EMP)$$

- SQL query:

```
SELECT   FNAME, LNAME, SALARY
FROM     Employee
WHERE    DNO = 5;
```

RENAME (ρ)

→ The RENAME is used to rename the attributes of a relation or the relation name or both.

→ It is useful when a query requires multiple operations also in some cases such as JOIN.

→ 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 names to B1, B2, B3, ..., BN.
- $\rho_{S}(R)$ changes only the relation name to S.

Date

- $\rho_{(B1, B2, \ldots BN)}(R)$ changes the column names only to B1, B2, ..., BN.

→ We also use shorthand for renaming an intermediate relation:

- If we write:

$$R(FN, LN, SAL) \leftarrow \pi_{FNAME, LNAME, SALARY}(EMPS)$$

R will rename FNAME to FN, LNAME to LN, and SALARY to SAL.

- If we write:

$R(F, M, L, S, B, A, SX, SAL, SU, D) \leftarrow EMP$

then EMP is renamed to R and the 10 attributes of EMP are renamed to F, M, L, S, B, A, SX, SAL, SU, D respectively.

→ In SQL, RENAME is achieved by using AS.

Example:

$$\rho_{FN, LS, SAL}\left(\pi_{FNAME, LNAME, SALARY}\left(\sigma_{DNO=5}(EMP)\right)\right)$$

The corresponding SQL query:

```
SELECT  FNAME AS FN, LNAME AS LN, SALARY AS SAL
FROM    EMP
WHERE   DNO = 5;
```

UNION (U)

→ UNION is a binary operation denoted by U. The result of R U S, is a relation that includes all tuples that are either in R or in S or in both R & S. Duplicate tuples are eliminated.

→ The two operand relations R and S must be "type compatible" which means:

- R and S must have same number of attributes.
- Each pair of corresponding attributes of relations have same domain

Example:

→ To retrieve the SSNs of all employees who either work in department 5 (Result 1) or directly supervise an employee who works in department 5 (Result 2).

We can use the UNION operator in this case as:

$DEP5_EMPS \leftarrow \sigma_{DNO=5}(EMPLOYEE)$

$RESULT1 \leftarrow \pi_{SSN}(DEP5_EMPS)$

$RESULT2 \leftarrow \pi_{SUPERSSN}(DEP5_EMPS)$

$RESULT \leftarrow RESULT1 \cup RESULT2$

Date

Type Compatibility

→ Type Compatibility of operands is required for the binary set operation UNION ∪, INTERSECTION ∩, SET DIFFERENCE −.

→ R1 (A1, A2, ..., RN) and R2 (B1, B2, ..., BN) are type compatible if:

- they have the same number of attributes (columns).
- the domains of corresponding attributes are type compatible (i.e. $dom(A_i) = dom(B_i)$ for $i = 1, 2, \ldots, n$).

→ The resulting relation for R1 ∪ R2 has the same attribute names as the first operand relation R1.

INTERSECTION (∩)

→ INTERSECTION is a binary operation denoted by ∩. The result of operation R∩S, is a relation that includes all Tuples that are in both R and S.

→ The attribut names in the result will be the same as the attribut names in R.

→ The two operand relations R and S must be "type compatible".

SET DIFFERENCE (−)

→ SET DIFFERENCE is a binary operation 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 of R−S will be the same as the attribute names in R.

→ The two operand relations R and S must be "type compatible".

→ SET DIFFERENCE operation is not commutative; that is, in general

$$R - S \neq S - R$$

Properties

→ Both union and intersection operations are commutative, that is

$R \cup S = S \cup R$

$R \cap S = S \cap R$

Date

→ Both union and intersection can be treated as n-ary operations applicable to any no. of relations as both are associative operations, that is

$R \cup (S \cup T) = (R \cup S) \cup T$

$R \cap (S \cap T) = (R \cap S) \cap T$

CATESIAN PRODUCT

→ Cartesian product is used to combine tuples from two relations in a combinatorial fashion.

→ The cross product denoted by $R(A_1, A_2, \ldots, A_N) \times S(B_1, B_2, \ldots, B_m)$ will result in a relation Q that will have the no. of attributes (columns) given as

$m + n$

and the number of tuples (n) in Q will be

$n_R * n_S$

→ The resulting relation state has one tuple for each combination of tuples — one from R and one from S.

→ R and S do not have to be "type compatible".

Example:

FEMALE_EMP ← $\sigma_{sex = 'F'}$ (EMP)

EMPNAME ← $\pi_{FNAME, LNAME, SSN}$ (FEMALE_EMP)

EMP_DEPDNT ← EMPNAME × DEPENDENT

EMP_DEPDNT will contain every combination of tuples from EMPNAMES and DEPENDENT, whether or not tuples are related.

⇒ In SQL query, we achieve cross product by leaving WHERE-clause

```
SELECT *
FROM EMP, DEPT
```

or can also be obtained by using CROSS JOIN.

```
SELECT *
FROM (DEPT CROSS JOIN EMP);
```

JOINS

→ The Join operation denoted by ⋈ is used to combine related tuples from two relations into single "longer" tuples.

→ The JOIN operation can be specified as a CARTESIAN PRODUCT operation followed by a

Date

SELECT operation.

→ The general form of a JOIN operation on two relations $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots, B_m)$ is

$$R \bowtie_{\langle \text{join condition} \rangle} S$$

→ The result of the JOIN condition is a relation Q with $n+m$ attributes $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots B_m)$ in that order.

→ In JOIN, only combinations of tuples satisfying the join condition appear in the result, whereas in CARTESIAN PRODUCT all combinations of tuples are included in the result.

Example:

Suppose that we want to retrieve the name of the manager of each department. To get the manger's name, we need to combine each department tuple with employee tuple whose SSN value matches the Mgr_SSN value in the department tuple. We do this by using JOIN operation and then projecting the result over the necessary attributes

$$DEPT_MGR \leftarrow DEPT \bowtie_{Mgr_SSN = SSN} EMP$$

$$RESULT \leftarrow \pi_{Fname, Lname, Fname}(DEPT_MGR)$$

Properties:

→ 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.

→ General case of JOIN operation is called a Theta-join:

$$R \bowtie_{theta} S$$

→ The join condition is called theta.

→ Theta can be any general boolean expression on the attributes of R and S; for example

$R.A_i < S.B_j$

AND

$(R.A_k = S.B_i$ OR $R.A_p < S.B_q)$

→ Most join conditions involve one or more equality conditions "AND"ed together. for example

$R.A_i = S.B_j$

AND

$R.A_k = S.B_i$

AND

$R.A_p = S.B_q$

Mujahid

CamScanner

Date

EQUIJOIN

→ The most common use of JOIN involves join conditions with equality comparisons only. Such a JOIN, where the only comparison operator used is called an EQUIJOIN.

→ In the result of an EQUIJOIN we always have one or more pairs of attributes that have identical values in every tuple.

NATURAL JOIN

→ A type of join that combines rows from two or more tables based on columns with the same name and compatible data types in both tables is called a NATURAL JOIN.

→ It is denoted by * and was created to get rid of the second attribute in an EQUI-JOIN result.

→ 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 above is not the case, a renaming operation is applied first.

Example:

Suppose we want to combine each PROJECT tuple with the DEPARTMENT tuple that controls the project. In the following example, first we rename the Dnumber attribute of DEPARTMENT to Dnum — so that it has the same name as the Dnum attribute in PROJECT — and then we apply NATURAL JOIN:

$$PROJ_DEPT \leftarrow PROJECT * \rho_{(Dname, Dnum, Mgr_ssn, Mgr_start_date)}(DEPARTMENT)$$

The same query can be done in two steps by creating an intermediate table DEPT as:

$$DEPT \leftarrow \rho_{(Dname, Dnum, Mgr_SSN, Mgr_start_date)}(DEPARTMENT)$$

$$PROJ_DEPT \leftarrow PROJECT * DEPT$$

The attribute Dnum is the join attribute for NATURAL JOIN operation, bcz it is only attribute with same name in both relations.

Date

INNER vs OUTER JOIN

→ All the join operations so far are known as INNER JOINS.

→ A tuple in one relation that does not have any matching tuples in the other relation (based on join condition) does not appear in the join result when using INNER JOIN.

→ If the user wants every tuple to appear at least once in the join result, another form of join called OUTER JOIN can be used.

Complete Set of Relational Operations

→ The set of relational algebra operations $\{\sigma, \pi, \cup, \rho, -, \times\}$ is a complete set, that is, any of the other original algebra operations can be expressed as a sequence of operations from this set.

Example:

the INTERSECTION operation can be expressed by using UNION and MINUS as given:

$$R \cap S \equiv (R \cup S) - ((R-S) \cup (S-R))$$

→ Also, a JOIN operation can be specified as a CARTESIAN PRODUCT followed by a SELECT operation, given as:

$$R \bowtie_{\langle condition \rangle} S \equiv \sigma_{\langle condition \rangle} (R \times S)$$

DIVISION

→ Given two relations $R(X, Y)$ with attributes X and Y and $S(Y)$ with attribute Y (where Y has same domain both relations). The division of R by S, denoted as $R \div S$, returns a relation with attribute X that contains all tuples x such that for every tuple y in S, there exists a tuple (x, y) in R.

Example:

→ Retrieve the names of employees who work on all the projects that "John Smith" works on.

→ First retrieve the list of project numbers that 'John Smith' works on in the intermediate relation SMITH_PNOS.

Date

$SMITH \leftarrow \sigma_{Fname = 'JOHN' \text{ AND } Lname = 'SMITH'} (EMPLOYEE)$

$SMITH_PNOS \leftarrow \pi_{PNO} (WORKS_ON \bowtie_{ESSN = SSN} SMITH)$

→ Next, create a relation that includes a tuple <Pno, Essn> whenever the employee whose SSN is ESSN works on the project whose number is PNO in the intermediate relation SSN_PNO.

$SSN_PNOS \leftarrow \pi_{ESSN, PNO} (WORKS_ON)$

→ Finally, apply the DIVISION operation which gives the desired ESSN:

$SSNS(SSN) \leftarrow SSN_PNOS \div SMITH_PNOS$

$RESULT \leftarrow \pi_{Fname, Lname} (SSN * EMPLOYEE)$

QUERY TREES

→ A query tree is a tree data structure that corresponds to a relational algebra expression. It represents the input relations of the query as leaf nodes of the tree, and represents the relational algebra operations as internal nodes.

→ Execution of the query tree consists of executing an internal node operation whenever its operands (represented by its child nodes) are available, then replacing that internal node by the relation that results from executing the operation. The execution terminates when the root node is executed and produces the result relation for the query.

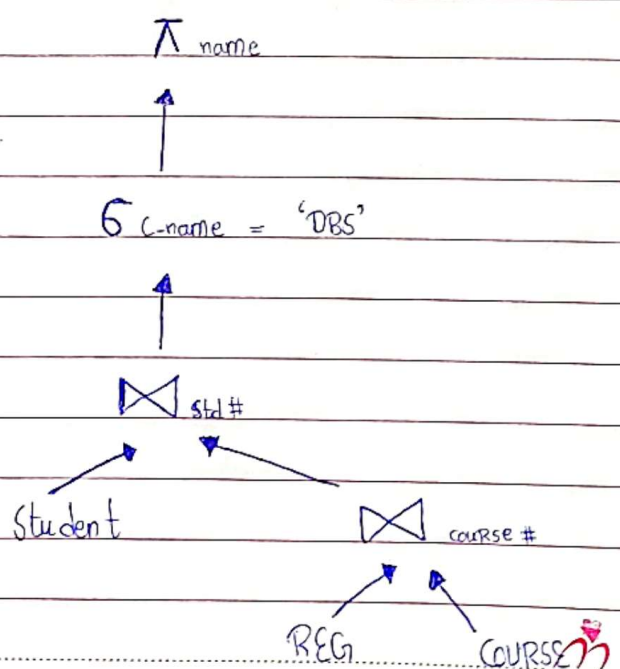
Example:

List the names of the Students registered in Database course.

- The query is given as:

$\pi_{name} (\sigma_{c\text{-}name = 'DBS'} (STD \bowtie_{Std\#} REG \bowtie_{Course\#} COURSE))$

- The query tree for above query is given as:



Mujahid
CamScanner

Date

Aggregate Functions

→ A type of query that cannot be expressed in the basic relational algebra is to specify mathematical aggregate functions on collections of values from the database.

Example:

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, MAX, and MIN. The COUNT function is used for counting tuples or values.

→ An operation $\mathcal{F}$ (aggregate function) can be added to the relational algebra for such queries

Aggregate Function Operation

→ Uses of Aggregate function Operation are as follows:

- $\mathcal{F}_{MAX_SAL}$ (EMPLOYEE) retrieves the maximum salary value from the EMPLOYEE relation.
- $\mathcal{F}_{MIN_SAL}$ (EMPLOYEE) retrieves the minimum salary value from the EMPLOYEE relation.
- $\mathcal{F}_{SUM_SAL}$ (EMPLOYEE) retrieves the sum of all the salary values from EMPLOYEE relation.
- $\mathcal{F}_{COUNT_SSN,\ AVG_SAL}$ (EMPLOYEE) computes the count (number) of employees and their average salary.
- COUNT (*) just counts the number of rows.

Using Grouping with Aggregation

→ It involves grouping the tuples in a relation by the value of their attributes and then applying an aggregate function independently to each group.

Example:

→ To group EMPLOYEE tuples by Dno, so that each group includes the tuples for employees working

Date

in the same department. We can list each Dno value along with the average salary of employees within the department and the number of employees who work in the department.

→ A variation of aggregate function $\mathcal{F}$ allows this:

- Grouping attribute placed to left of symbol.
- Aggregate function to right of the symbol.

Thus,

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

- Above operation groups employees by DNO and computes the count of employees and average salary per department.

Recursive Closure Operation

→ This operation is applied to a recursive relationship b/w tuples of the same type, such as the relationship b/w an employee and a supervisor. This relationship is described by the foreign key Super_SSN of the EMPLOYEE relation and it relates each employee tuple (in the role of supervisee) to another employee tuple (in the role of supervisor).

Example:

- An example of a recursive operation is to retrieve all supervisees of an employee e at all levels — that is, all employees e', ~~all employees~~ directly supervised by e, all employees e'' directly supervised by each employee e'; all employees e''' directly supervised by each employee e'', and so on.

→ An operation called the transitive closure of relations has been proposed to compute the recursive relationship as far as the recursion proceeds.

Outer Join

→ A set of operations, called outer joins, were developed for the case where the user wants to keep the tuples in R, or all those in S, or all those in both relations in the result of

Date

JOIN, regardless of whether or not they have matching tuples in the other relation.

Example:

- Suppose we want a list of all employee names as well as the name of the departments they manage if they happen to manage a department; if they do not manage one, we can indicate it with a NULL value. We can apply an operation LEFT Outer JOIN, denoted by ⟕, to retrieve the results as:

$$TEMP \leftarrow (EMPLOYEE \text{ ⟕}_{SSN = Mgrssn} DEPART)$$
$$RESULT \leftarrow \pi_{Fname, Lname, Minit, Dname}(TEMP)$$

→ **LEFT OUTER JOIN** operation keeps every tuple 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.

→ **RIGHT OUTER JOIN** denoted by ⟖ keeps every tuple in the second, or right, relation S in the result of R ⟖ S.

→ **FULL OUTER JOIN**, denoted by ⟗, keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with NULL values as needed.

Outer Union

→ The OUTER UNION operation was developed to take the union of tuples from two relations that have some common attributes, but are not union (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 union compatible. The attributes that are union compatible are

Date

represented only once in the result, and those attributes that are not union compatible from either relation are also kept in the result relation T(X,Y,Z). It is therefore, the same as a FULL OUTER JOIN on the common attributes.

Example:

- An OUTER **UNION** can be applied to two relations whose schemas are

STUDENT (Name, Ssn, Dept, Advisor)
INSTRUCTOR (Name, Ssn, Dept, Rank)

- Tuples from the two relations are matched based on having the same combination of values of the shared attributes — Name, Ssn, Department. The resulting relation STUDENT_OR_INSTRUCTOR, will be as:

STUDENT_OR_INSTRUCTOR
(Name, Ssn, Dept, Advisor, Rank)