Appendix D:
Relational Algebra and Relational Calculus

Introduction

In this appendix, we overview two formal languages underlying the relational model discussed in Chapter 6 and SQL discussed in Chapter 7: relational algebra and relational calculus.

Relational Algebra

Introduction

Relational algebra provides the theoretical foundation of relational databases and SQL and was created by Edgar F. Codd[1]. It consists of a set of operators on relations. Every operator uses one or more relations as its input or operand(s) and produces a new relation as its result. Relational algebra is procedural or prescriptive since one must explicitly specify what operators should be applied to which relations. In what follows, we distinguish between set operators and relational operators.

Set Operators

The set operators are UNION, INTERSECTION, DIFFERENCE, and PRODUCT. Each operator uses two relations as operands and produces a new relation as a result. In some cases, the operand relations need to be union compatible which implies that they have the same number of attribute types defined on the same domains. In what follows, we represent a relation as R and its cardinality or number of tuples as |R|. Let's now introduce the union compatible

[1] Codd, E.F., A Relational Model of Data for Large Shared Data Banks, Communications of the ACM, 13 (6): 377–387, 1970.

relations P and Q whereby P represents the suppliers located in ‘San Francisco’ and Q the suppliers that can supply product 100 as depicted in Figure 1.

P

SUPNR	SUPNAME	SUPSTATUS
16	Bart	90
18	Wilfried	60
20	Seppe	80

Q

SUPNR	SUPNAME	SUPSTATUS
10	Victor	80
16	Bart	90
22	Catherine	85
24	Sophie	75
26	Laura	92

Figure 1 Example relations P and Q.

UNION Operator

The UNION operator applied to two union-compatible relations P and Q results into a new relation $R = P \cup Q$ so:

$$R = \{t \mid t \in P \text{ OR } t \in Q\} \qquad \text{with } \max(|P|, |Q|) \leq |R| \leq |P| + |Q|$$

The resulting relation R consists of the tuples that either belong to P or Q, or both. Duplicate tuples are eliminated. When applied to our example relations in Figure 1, the resulting

relation R consists of all suppliers either in San Francisco or that can supply product number 100 as depicted in Figure 2.

P ∪ Q

SUPNR	SUPNAME	SUPSTATUS
16	Bart	90
18	Wilfried	60
20	Seppe	80
10	Victor	80
22	Catherine	85
24	Sophie	75
26	Laura	92

Figure 2 UNION operator.

Note that the cardinality of the resulting relation depends upon the number of duplicate tuples in P and Q.

INTERSECTION Operator

The INTERSECTION operator applied to two union-compatible relations P and Q results into a new relation $R = P \cap Q$ such that:

$$R = \{t \mid t \in P \text{ AND } t \in Q\} \qquad \text{with } 0 \leq |R| \leq |\min(|P|, |Q|)$$

The resulting relation R consists of the tuples that belong to both P and Q. When applied to our example relations in Figure 1, the resulting relation consists of all suppliers that are both located in San Francisco and can supply product 100 as depicted in Figure 3.

P ∩ Q

SUPNR	SUPNAME	SUPSTATUS
16	Bart	90

Figure 3 INTERSECTION operator.

DIFFERENCE Operator

The DIFFERENCE operator applied to two union-compatible relations P and Q results into a new relation R = P - Q such that:

$$R = \{t \mid t \in P \text{ AND } t \notin Q\} \qquad \text{with } 0 \leq |R| \leq |P|$$

The resulting relation R consists of the tuples that belong to P but not to Q. Note that when using the DIFFERENCE operator, the order of the operands is important since it concerns a non-commutative operation. When applied to our example relations in Figure 1, the resulting relation R consists of all suppliers located in San Francisco and cannot supply product number 100 as depicted in Figure 4.

P - Q

SUPNR	SUPNAME	SUPSTATUS
18	Wilfried	60
20	Seppe	80

Figure 4 DIFFERENCE operator.

PRODUCT Operator

The PRODUCT operator (also known as CARTESIAN PRODUCT, CROSS PRODUCT or CROSS JOIN operator) returns the Cartesian product of two relations. Consider the following two relations: a supplier relation P and a product relation Q. The Cartesian product of both relations consists

of all possible combinations of tuples from P and Q. The resulting relation $R = P \times Q$ consists of a set of tuples r = pq such that:

$$R = \{r = pq \mid p \in P \text{ and } q \in Q\} \text{ with } |R| = |P| \times |Q|$$

Hence, the resulting operations will have as many tuples as the number of combinations of supplier and product tuples. This is illustrated in Figure 5.

P

SUPNR	SUPNAME	SUPSTATUS
16	Bart	90
18	Wilfried	60
20	Seppe	80

Q

PRODNR	PRODNAME	PRODCOLOR
100	Hiking Shoes	black
200	Backpack	blue

P × Q

SUPNR	SUPNAME	SUPSTATUS	PRODNR	PRODNAME	PRODCOLOR
16	Bart	90	100	Hiking Shoes	black
16	Bart	90	200	Backpack	blue
18	Wilfried	60	100	Hiking Shoes	black
18	Wilfried	60	200	Backpack	blue
20	Seppe	80	100	Hiking Shoes	black
20	Seppe	80	200	Backpack	blue

Figure 5 PRODUCT operator.

Relational Operators

The relational operators are SELECT, PROJECT, RENAME, JOIN, and DIVISION. The SELECT, PROJECT and RENAME operators are unary operators since they only use one operand. The JOIN and DIVISION operators are binary operators since they use two operands.

SELECT Operator

The SELECT operator is used to return a subset of tuples of a relation based on a selection condition. The latter can consist of predicates combined using Boolean operators (AND, OR, NOT). Every predicate consists of a comparison between the values of two domain-compatible attribute types, or between an attribute type and a constant value from the domain of the attribute type.

The result of applying a SELECT operator with a selection condition S on a relation P can be represented as:

R=σ_S(P) with SELECT operator σ

Note that applying a SELECT operator to a relation results in a horizontal subset of the relation.

As an example, consider the following relation P (Figure 6).

P

SUPNR	SUPNAME	SUPCITY	SUPSTATUS
10	Victor	San Francisco	80
16	Bart	San Francisco	90
22	Catherine	Las Vegas	85
24	Sophie	New York	75
26	Laura	Orlando	92

Figure 6 Example relation.

The result of R=$\sigma_{SUPCITY=San\ Francisco}$(P) then becomes (Figure 7):

SUPNR	SUPNAME	SUPCITY	SUPSTATUS
10	Victor	San Francisco	80
16	Bart	San Francisco	90

Figure 7 SELECT operator.

PROJECT Operator

The PROJECT operator is used to project a relation on a list of attribute types. The result of applying a PROJECT operator with a list L on a relation P can be represented as:

R=π_L(P) with PROJECT operator π

Note that applying a PROJECT operator to a relation results in a vertical subset of the relation with |R| ≤ |P| since duplicate tuples will be eliminated.

As an example, consider the relation in Figure 6 again. The result of $\pi_{SUPNR,\ SUPNAME}$(P) then becomes (Figure 9):

SUPNR	SUPNAME
10	Victor
16	Bart
22	Catherine
24	Sophie
26	Laura

Figure 8 PROJECT operator.

RENAME Operator

The RENAME operator is used to rename an attribute type of a relation (in case, e.g., naming conflicts can occur). The result of applying a RENAME operator on a relation P to rename attribute type B to A can be represented as:

$R=\rho_{A|B}(P)$ with RENAME operator ρ

As an example, again consider the relation in Figure 6. The result of $R=\rho_{SUPLEVEL|SUPSTATUS}(P)$ then becomes (Figure 9):

SUPNR	SUPNAME	SUPCITY	SUPLEVEL
10	Victor	San Francisco	80
16	Bart	San Francisco	90
22	Catherine	Las Vegas	85
24	Sophie	New York	75
26	Laura	Orlando	92

Figure 9 RENAME operator.

JOIN Operator

The JOIN operator allows to combine tuples of two relations based on specific conditions, also called join conditions. The result of a join is the combined effect of a Cartesian product and selection.

The result of applying a JOIN operator to two relations P and Q with join condition j can be represented as follows:

$R = P \bowtie_j Q$ with JOIN operator $\bowtie$ and join condition j.

The resulting relation R consists of a set of combinations of pq tuples such that a combined tuple $r \in R$ is characterized by:

$R = \{r = pq \mid p \in P \text{ AND } q \in Q \text{ AND } j\}$ with $|R| \leq |P| \times |Q|$

An example join condition j could be that two tuples should have the same values for two corresponding attribute types.

Consider the example relations in Figure 10. The relation P is a SUPPLIER relation, and relation Q is a SUPPLIES relation indicating which suppliers can supply what products.

P

SUPNR	**SUPNAME**	**SUPCITY**	**SUPSTATUS**
10	Victor	San Francisco	80
16	Bart	San Francisco	90
22	Catherine	Las Vegas	85
24	Sophie	New York	75
26	Laura	Orlando	92

Q

SUPNR	PRODNR
10	100
10	200
22	100
22	120
22	180
26	100
26	160

Figure 10 Example relations.

Suppose we want a list of supplier data along with the products they can supply. This can be obtained with this join:

$$R = P \bowtie_{P.SUPNR=Q.SUPNR} Q$$

The result is shown in Figure 11.

$P \bowtie_{P.SUPNR=Q.SUPNR} Q$

SUPNR	SUPNAME	SUPCITY	SUPSTATUS	PRODNR
10	Victor	San Francisco	80	100
10	Victor	San Francisco	80	200
22	Catherine	Las Vegas	85	100
22	Catherine	Las Vegas	85	120
22	Catherine	Las Vegas	85	180
26	Laura	Orlando	92	100
26	Laura	Orlando	92	160

Figure 11 JOIN operator.

The above JOIN operator is also called a THETA join, where THETA (θ) refers to the comparison operator in the join condition. Depending upon THETA, these joins can be distinguished: GREATER-THAN-JOIN, LESS-THAN-JOIN, GREATER-THAN-OR-EQUAL-JOIN, LESS-THAN-OR-EQUAL-JOIN, and an EQUI-JOIN (if an equals occurs).

The NATURAL-JOIN is a variant of the EQUI-JOIN in which one of the shared join-attribute types is removed from the result. The join in Figure 11 is an example of a NATURAL-JOIN.

All join examples discussed thus far are INNER-JOINS since they do not include tuples with lacking corresponding join values. The OUTER-JOIN operator will also include these tuples in the result. A distinction can be made between a LEFT-OUTER-JOIN, RIGHT-OUTER-JOIN, and FULL-OUTER-JOIN.

A LEFT-OUTER-JOIN includes all tuples from the left relation (P) in the result, either matched with a corresponding tuple from the right relation (Q) based on the join condition or augmented with NULL values in case no match with a tuple from Q can be made. A LEFT-OUTER-JOIN can be represented as:

$$R = P \mathbin{⟕}_j Q$$

Let's illustrate this with an example. Suppose we would like to perform a LEFT-OUTER-JOIN between relations P and Q of Figure 10Figure 10. The result is displayed in Figure 12. Note the NULL values that have been added because suppliers Bart and Sophie cannot supply any products.

$$\mathbf{P} \mathbin{⟕}_{\mathbf{P.SUPNR=Q.SUPNR}} \mathbf{Q}$$

SUPNR	SUPNAME	SUPCITY	SUPSTATUS	PRODNR

10	Victor	San Francisco	80	100
10	Victor	San Francisco	80	200
16	Bart	San Francisco	90	NULL
22	Catherine	Las Vegas	85	100
22	Catherine	Las Vegas	85	120
22	Catherine	Las Vegas	85	180
24	Sophie	New York	75	NULL
26	Laura	Orlando	92	100
26	Laura	Orlando	92	160

Figure 12 LEFT-OUTER-JOIN operator.

A RIGHT-OUTER-JOIN includes all tuples from the right relation (Q) in the result, either matched with a corresponding tuple from the left relation (P) based on the join condition or augmented with NULL values in case no match with a tuple from P can be made. A RIGHT-OUTER-JOIN can be represented as:

$$R = P ⟖_j Q$$

A FULL-OUTER-JOIN includes all tuples from P and Q in the result either matched with the counterparts according to the join condition or augmented with NULL values in case no match can be found. A FULL-OUTER-JOIN can be represented as:

$$R = P ⟗_j Q$$

DIVISION Operator

The DIVISION operator is a binary operator that divides a relation P by a relation Q with $Q \subseteq P$. The result of a DIVISION operator can be represented as:

$$R = P \div Q \qquad \text{with DIVISION operator } \div$$

The resulting relation R includes all tuples that belong to P combined with every tuple in Q.

Note that the DIVISION operator is not directly implemented in SQL.

Consider the following relations (Figure 13).

P

SUPNR	PRODNR
10	100
10	200
22	100
24	100
26	100
26	200
28	200

Q

PRODNR
100
200

Figure 13 Example relations.

The result of P ÷ Q then becomes (Figure 14):

SUPNR
10
26

Figure 14 DIVISION operator.

Primitive Relational Algebra Operators and Derived Operators

The SELECTION (σ), PROJECTION (π), DIFFERENCE (-), PRODUCT ($\times$) and UNION ($\cup$) operators are often referred to as the five primitive operators since all other relational algebra operators can be derived from them (derived operators). Let's illustrate this with a few examples.

The effect of an INTERSECTION operator can be obtained by combining a UNION and DIFFERENCE operator:

$$P \cap Q \equiv (P \cup Q) - ((P - Q) \cup (Q - P))$$

The effect of a JOIN operator can be obtained by combining a PRODUCT and SELECT operator:

$$P \bowtie_j Q \equiv \sigma_j(P \times Q)$$

The effect of a DIVISION operator can be obtained by combining a PROJECT, PRODUCT and DIFFERENCE operator:

$$R_1 = \pi_L(P)$$

$$R_2 = \pi_L((Q \times R_1) - P)$$

$$P \div Q \equiv R_1 - R_2$$

Examples

Consider the following set of relations (primary keys are underlined; foreign keys are in italics):

SUPPLIER(SUPNR, SUPNAME, SUPADDRESS, SUPCITY, SUPSTATUS)

PRODUCT(PRODNR, PRODNAME, PRODTYPE, AVAILABLE_QUANTITY)

SUPPLIES(*SUPNR*, *PRODNR*, PURCHASE_PRICE, DELIV_PERIOD)

PURCHASE_ORDER(PONR, PODATE, *SUPNR*)

PO_LINE(*PONR*, *PRODNR*, QUANTITY)

Assume that we are interested in the suppliers that are either located in 'San Francisco' or can supply product number 100. This can be written as:

R1 = $\sigma_{SUPCITY=San\ Francisco}$(SUPPLIER)

R2 = $\sigma_{PRODNR=100}$(SUPPLIES)

R3 = π_{SUPNR}(R2)

R4 = R3 $\bowtie_{R3.SUPNR=SUPPLIER.SUPNR}$(SUPPLIER)

R5 = R1 ∪ R4

Assume now that we are interested in the name and city of the suppliers that can supply all products. This can be written as:

R1 = $\pi_{SUPNR,PRODNR}$(SUPPLIES)

R2 = π_{PRODNR}(PRODUCT)

R3 = R1 ÷ R2

R4 = R3 $\bowtie_{R3.SUPNR=SUPPLIER.SUPNR}$(SUPPLIER)

R5 = $\pi_{SUPNAME,SUPCITY}$(R4)

Relational Calculus

Relational calculus is a declarative alternative to relational algebra. It describes the set of answers without explicitly specifying how they should be computed. Codd's theorem states that relational algebra and relational calculus are equivalent in the sense that any relational

algebraic expression has a relational calculus equivalent and vice versa[2]. Two variants of relational calculus exist tuple relational calculus and domain relational calculus. The difference relates to the type of variables used in the predicates.

Tuple Relational Calculus

Tuple variables are the key building blocks of tuple relational calculus. A tuple variable refers to a tuple of a corresponding relation, also called range relation. A tuple relational calculus expression specifies a range relation for each tuple variable, a condition to select (combinations of) tuples, and a list of attribute types from which the values should be retrieved. A tuple relational calculus expression looks as follows:

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

whereby t represents the tuple variable, A_i the attribute type of which the value needs to be retrieved and COND(t) the range relation and extra conditions to select relevant tuples. Consider the following example:

$$\{l.SUPNR, l.SUPNAME \mid SUPPLIER(l) \text{ AND } l.SUPSTATUS > 85\}$$

This will select the supplier number and supplier name of all suppliers whose status is bigger than 85.

A condition is specified using a calculus formula, also called well-formed formula (wff). This can consist of various predicate calculus atoms combined using Boolean operators (AND, OR, NOT). The result of applying a formula to a tuple of a range relation can either be true, false or unknown. If the result is true, the tuple will be selected in the result.

[2] E. F. Codd, Relational completeness of data base sublanguages, in R. Rustin, (ed.) *Data Base Systems*, Proceedings of 6th Courant Computer Science Symposium, pp. 65-98, Prentice-Hall, 1972.

Relational calculus also includes quantifiers such as the existential quantifier (∃ or EXISTS) and universal quantifier (∀ or FOR ALL).

A formula with an existential quantifier (∃) evaluates to true if at least one tuple satisfies it. Assume we would be interested to find the numbers and names of all suppliers with at least one outstanding purchase order with date 27/02/2018. This can be formulated as:

{s.SUPNR, s.SUPNAME | SUPPLIER(s) AND (∃ po) (PURCHASE_ORDER(po) AND po.PODATE='27/02/2018' AND s.SUPNR=po.SUPNR)}

Two tuple variables are used: s with range relation SUPPLIER and po with range relation PURCHASE_ORDER. The condition po.PODATE='27/02/2018' is a selection condition (similar to the SELECT operator in relational algebra). The condition s.SUPNR=po.SUPNR is a join condition (similar to the JOIN operator in relational algebra). The existential operator (∃) ensures that as soon as a supplier has one outstanding purchase order with the given date, the expression evaluates to true and the supplier number and name will be returned.

A formula with a universal quantifier (∀) evaluates to true if every tuple from the range relation satisfies the conditions stated. Assume we would be interested to find the highest product number. This can be formulated as:

{p1.PRODNR | PRODUCT(p1) AND (∀p2) (PRODUCT(p2) AND p1.PRODNR ≥ p2.PRODNR)}

Two tuple variables are used with the same range relation: PRODUCT. The universal quantifier (∀) ensures that all tuples p2 have a lower product number than the product number of tuple p1.

Domain Relational Calculus

Instead of tuple variables, domain relational calculus defines domain variables that range over single values of domains of attribute types. A domain calculus expression for a relation of degree n looks as follows:

$$\{v_1, v_2, \ldots v_n, \mid COND(v_1, v_2, \ldots v_n) \}$$

where $v_1, v_2, \ldots v_n$ represent the domain variables that range over domains of attribute types, and COND represents the extra conditions.

Our earlier tuple relational calculus query selecting the supplier number and supplier name of all suppliers whose status is bigger than 85 would now look as follows:

$$\{ab \mid \exists(e) (SUPPLIER(abcde) \text{ AND } e > 85)\}$$

Note we defined five domain variables as a for the domain of SUPNR, b for the domain of SUPNAME, c for the domain of SUPADDRESS, d for the domain of SUPCITY and e for the domain of SUPSTATUS. The condition e > 85 is a selection condition.

Our second tuple relational calculus query to find the numbers and names of all suppliers with at least one outstanding purchase order with date 27/02/2018 can be formulated as:

$$\{ab \mid \exists(a) \exists(l) \exists(m) (SUPPLIER(abcde) \text{ AND } (PURCHASE\_ORDER(klm) \text{ AND } l = '27/02/2018' \text{ AND } a = m)\}$$

Note the selection condition l = '27/02/2018' and join condition a = m.

The existential quantifier (∃ or EXISTS) and universal quantifier (∀ or FOR ALL) are also supported in domain relational calculus. Our earlier query to retrieve the numbers and names

of all suppliers with at least one outstanding purchase order with date 27/02/2018 can now be formulated as:

{ab | (∃a) SUPPLIER(abcde) AND (∃r) (∃q) (PURCHASE_ORDER(pqr) AND q = '27/02/2018' AND a = r)}

Likewise, our query to find the highest product number can be formulated as:

{a | (∃a) PRODUCT(abcd) AND ((∀p) PRODUCT(pqrs) AND a ≥ p)}

Closing Thoughts

The QBE (Query-by-Example) language developed by IBM is an example of a language based on domain relational calculus. SQL is more closely related to relational algebra or tuple relational calculus. Note, however, this mapping is not perfect. Examples of queries that cannot be expressed in relational algebra or relational calculus include aggregations such as counting tuples or summing attribute values of tuples. Another key issue is that SQL results are represented as multisets which may include duplicates as opposed to sets used by relational algebra.