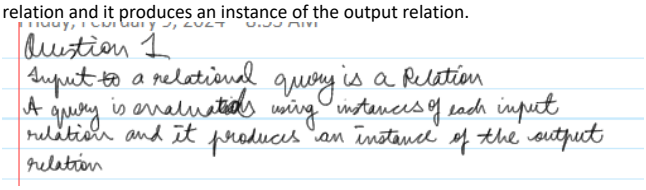
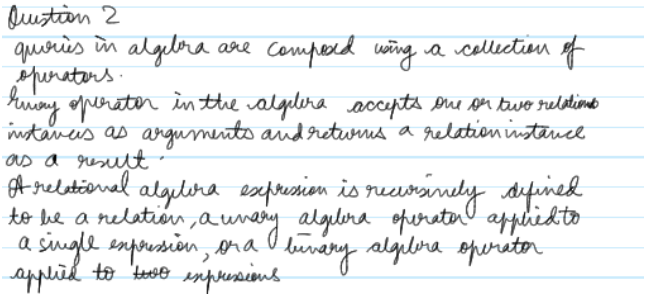
Review questions

Monday, February 12, 2024 1:02 PM

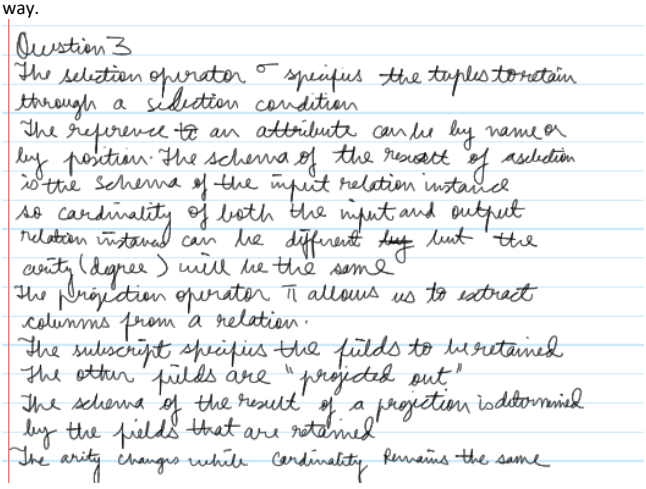
QUESTION 1
The inputs and outputs of a query are relations. A query is evaluated using instances of each input relation and it produces an instance of the output relation.



QUESTION 2
A fundamental property is that every operator in the algebra accepts (one or two) relation instances as arguments and returns a relation instance as the result. This property makes it easy to compose operators to form a complex query—a relational algebra expression is recursively defined to be a relation, a unary algebra operator applied to a single expression, or a binary algebra operator applied to two expressions.



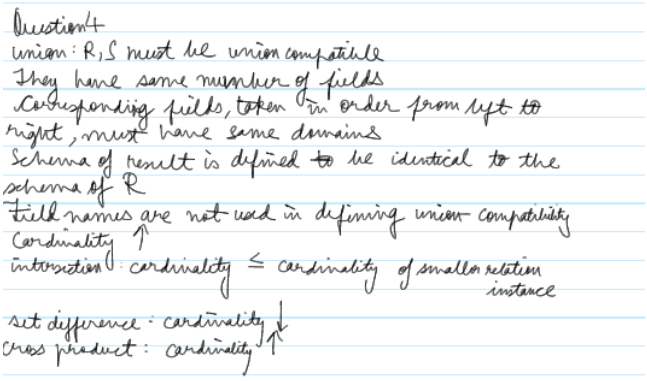
QUESTION 3
The selection operator σ specifies the tuples to retain through a selection con□dition. In general, the selection condition is a Boolean combination (i.e., an expression using the logical connectives □ and □) of terms that have the form attribute op constant or attribute1 op attribute2, where op is one of the com□parison operators <, <=, =, =, >=, or >. The reference to an attribute can be by position (of the form .i or i) or by name (of the form .name or name). The schema of the result of a selection is the schema of the input relation instance. The projection operator π allows us to extract columns from a relation;
The schema of the result of a projection is determined by the fields that are projected in the obvious way.



QUESTION 4
Union: R ∪ S returns a relation instance containing all tuples that occur in either relation instance R or relation instance S (or both). R and S must be union-compatible, and the schema of the result is defined to be identical to the schema of R. Two relation instances are said to be union-compatible if the following conditions hold: – they have the same number of the fields, and – corresponding fields, taken in order from left to right, have the same domains. Note that field names are not used in defining union-compatibility. For convenience, we will assume that the fields of R ∪ S inherit names from R, if the fields of R have names
Intersection: R ∩ S returns a relation instance containing all tuples that occur in both R and S. The relations R and S must be union-compatible, and the schema of the result is defined to be identical to the schema of R
Set-difference: R–S returns a relation instance containing all tuples that occur in R but not in S. The relations R and S must be union-compatible, and the schema of the result is defined to be identical to the schema of R. Cross-product: R × S returns a relation instance whose schema contains all the fields of R (in the same order as they appear in R) followed by all the fields of S (in the same order as they appear

Choose File | No file chosen

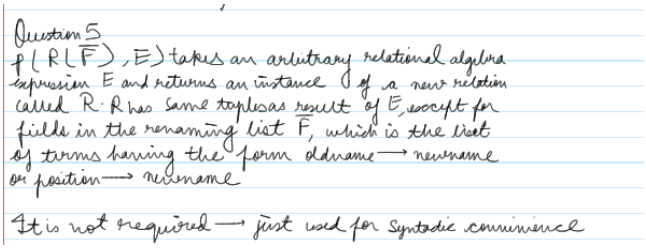
The cross-product of R and S contains one tuple r, s (the concatenation of tuples r and s) for each pair of tuples r ∈ R, s ∈ S. The cross-product opertion is sometimes called Cartesian product.



QUESTION 5

It is therefore convenient to be able to give names explicitly to the fields of a relation instance that is defined by a relational algebra expression. In fact, it is often convenient to give the instance itself a name so that we can break a large algebra expression into smaller pieces by giving names to the results of subexpressions. We introduce a renaming operator ρ for this purpose. The expression ρ(R(F), E) takes an arbitrary relational algebra expression E and returns an instance of a (new) relation called R. R contains the same tuples as the result of E and has the same schema as E, but some fields are renamed. The field names in relation R are the same as in E, except for fields renamed in the renaming list F, which is a list of terms having the form oldname → newname or position → newname. For ρ to be well-defined, references to fields (in the form of oldnames or positions in the renaming list) may be unambiguous and no two fields in the result may have the same name. Sometimes we want to only rename fields or (re)name the relation; we therefore treat both R and F as optional in the use of ρ. (Of course, it is meaningless to omit both.)

It is customary to include some additional operators in the algebra, but all of them can be defined in terms of the operators we have defined thus far. (In fact, the renaming operator is needed only for syntactic convenience, and even the ∩ operator is redundant; R∩S can be defined as R−(R−S).)

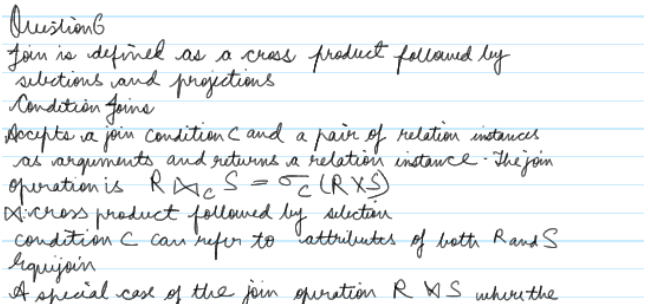


QUESTION 6

n accepts a join condition c and a pair of relation instances as arguments and returns a relation instance. The join condition is identical to a selection condition in form. The operation is defined as follows: R c S = σc(R × S) Thus is defined to be a cross-product followed by a selection. Note that the condition c can (and typically does) refer to attributes of both R and S. The reference to an attribute of a relation, say, R, can be by position (of the form R.i) or by name (of the form R.name).

A common special case of the join operation R S is when the join condition consists solely of equalities (connected by ∧) of the form R.name1 = S.name2, that is, equalities between two fields in R and S. In this case, obviously, there is some redundancy in retaining both attributes in the result. For join conditions that contain only such equalities, the join operation is refined by doing an additional projection in which S.name2 is dropped. The join operation with this refinement is called equijoin. The schema of the result of an equijoin contains the fields of R (with the same names and domains as in R) followed by the fields of S that do not appear in the join conditions. If this set of fields in the result relation includes two fields that inherit the same name from R and S, they are unnamed in the result relation

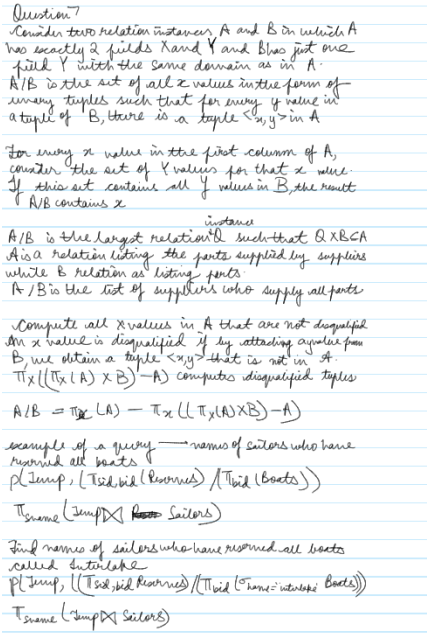
A further special case of the join operation R S is an equijoin in which equalities are specified on all fields having the same name in R and S. In this case, we can simply omit the join condition; the default is that the join condition is a collection of equalities on all common fields. We call this special case a natural join, and it has the nice property that the result is guaranteed not to have two fields with the same name.



join condition consists solely of equalities connected by ∧ of the form R.name1 = S.name2
The join operation is defined by doing an additional projection in which S.name2 is dropped
The schema of the result consists of fields of R with the same names and domains as in R followed by the fields of S that do not appear in the join conditions.
Natural join
Omit the join condition → the default is that the join condition is a collection of equalities on all common fields

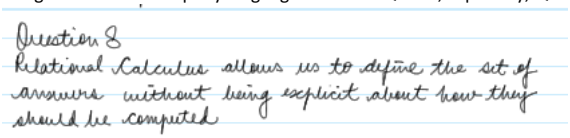
QUESTION 7

Consider two relation instances A and B in which A has (exactly) two fields x and y and B has just one field y, with the same domain as in A. We define the division operation A/B as the set of all x values (in the form of unary tuples) such that for every y value in (a tuple of) B, there is a tuple x,y in A. Another way to understand division is as follows. For each x value in (the first column of) A, consider the set of y values that appear in (the second field of) tuples of A with that x value. If this set contains (all y values in) B, the x value is in the result of A/B. An analogy with integer division may also help to understand division. For integers A and B, A/B is the largest integer Q such that Q ∗ B ≤ A. For relation instances A and B, A/B is the largest relation instance Q such that Q × B ⊆ A. It helps to think of A as a relation listing the parts supplied by suppliers and of the B relations as listing parts. A/Bi computes suppliers who supply all parts listed in relation instance Bi.



QUESTION 8

Relational calculus is an alternative to relational algebra. In contrast to the algebra, which is procedural, the calculus is nonprocedural, or declarative, in that it allows us to describe the set of answers without being explicit about how they should be computed. Relational calculus has had a big influence on the design of commercial query languages such as SQL and, especially, Query-by-Example (QBE).

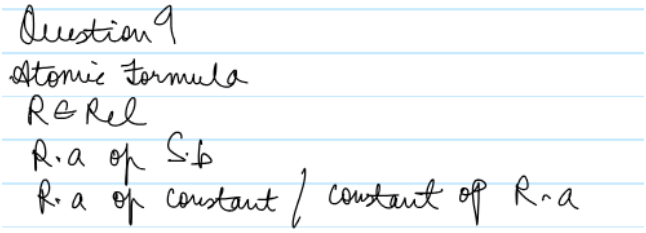


QUESTION 9

A tuple variable is a variable that takes on tuples of a particular relation schema as values. That is, every value assigned to a given tuple variable has the same number and type of fields. A tuple relational calculus query has the form { T | p(T) }, where T is a tuple variable and p(T) denotes a formula that describes T; we will shortly define formulas and queries rigorously. The result of this query is the set of all tuples t for which the formula p(T) evaluates to true with T = t. The language for writing formulas p(T) is thus at the heart of TRC and essentially a simple subset of first-order logic.
A formula is recursively defined to be one of the following, where p and q are themselves formulas and p(R) denotes a formula in which the variable R appears:
any atomic formula ¬p, p ∧ q, p ∨ q, or p ⇒ q ∃R(p(R)), where R is a tuple variable ∀R(p(R)), where R is a tuple variable In the last two clauses, the quantifiers ∃ and ∀ are said to bind the variable R. A variable is said to be free in a formula or subformula (a formula contained in a larger formula) if the (sub)formula does not contain an occurrence of a quantifier that binds it.
We do not define types of variables formally, but the type of a variable should be clear in most cases, and the important point to note is that comparisons of values having different types should always fail. (In discussions of relational calculus, the simplifying assumption is often made that there is a single

domain of constants and this is the domain associated with each field of each relation.) A TRC query is defined to be expression of the form {T | p(T)}, where T is the only free variable in the formula p.

A query is evaluated on a given instance of the database. Let each free variable in a formula F be bound to a tuple value. For the given assignment of tuples to variables, with respect to the given database instance, F evaluates to (or simply 'is') true if one of the following holds: F is an atomic formula R ∈ Rel, and R is assigned a tuple in the instance of relation Rel. F is a comparison R.a op S.b, R.a op constant, or constant op R.a, and the tuples assigned to R and S have field values R.a and S.b that make the comparison true. F is of the form ¬p and p is not true, or of the form p ∧ q, and both p and q are true, or of the form p ∨ q and one of them is true, or of the form p ⇒ q and q is true whenever4 p is true. F is of the form ∃R(p(R)), and there is some assignment of tuples to the free variables in p(R), including the variable R, 5 that makes the formula p(R) true. F is of the form ∀R(p(R)), and there is some assignment of tuples to the free variables in p(R) that makes the formula p(R) true no matter what tuple is assigned to R.



QUESTION 10

A tuple variable is a variable that takes on tuples of a particular relation schema as values. That is, every value assigned to a given tuple variable has the same number and type of fields. A tuple relational calculus query has the form { T | p(T) }, where T is a tuple variable and p(T) denotes a formula that describes T; we will shortly define formulas and queries rigorously. The result of this query is the set of all tuples t for which the formula p(T) evaluates to true with T = t. The language for writing formulas p(T) is thus at the heart of TRC and essentially a simple subset of first-order logic.

Relational calculus is an alternative to relational algebra. In contrast to the algebra, which is procedural, the calculus is nonprocedural, or declarative, in that it allows us to describe the set of answers without being explicit about how they should be computed. Relational calculus has had a big influence on the design of commercial query languages such as SQL and, especially, Query-by-Example (QBE). The variant of the calculus we present in detail is called the tuple relational calculus (TRC). Variables in TRC take on tuples as values.

QUESTION 11

Consider the query {S | ¬(S ∈ Sailors)}. This query is syntactically correct. However, it asks for all tuples S such that S is not in (the given instance of)

Sailors. The set of such S tuples is obviously infinite, in the context of infinite domains such as the set of all integers. This simple example illustrates an unsafe query. It is desirable to restrict relational calculus to disallow unsafe queries.

For a calculus formula Q to be considered safe, at a minimum we want to ensure that, for any given I, the set of answers for Q contains only values in Dom(Q, I). While this restriction is obviously required, it is not enough. Not only do we want the set of answers to be composed of constants in Dom(Q, I), we wish to compute the set of answers by examining only tuples that contain constants in Dom(Q, I)! This wish leads to a subtle point associated with the use of quantifiers ∀ and ∃: Given a TRC formula of the form ∃R(p(R)), we want to find all values for variable R that make this formula true by checking only tuples that contain constants in Dom(Q, I). Similarly, given a TRC formula of the form ∀R(p(R)), we want to find any values for variable R that make this formula false by checking only tuples that contain constants in Dom(Q, I). We therefore define a safe TRC formula Q to be a formula such that: 1. For any given I, the set of answers for Q contains only values that are in Dom(Q, I). 2. For each subexpression of the form ∃R(p(R)) in Q, if a tuple r (assigned to variable R) makes the formula true, then r contains only constants in Dom(Q, I). 3. For each subexpression of the form ∀R(p(R)) in Q, if a tuple r (assigned to variable R) contains a constant that is not in Dom(Q, I), then r must make the formula true.

QUESTION 12

We presented two formal query languages for the relational model. Are they equivalent in power? Can every query that can be expressed in relational algebra also be expressed in relational calculus? The answer is yes, it can.

If a query language can express all the queries that we can express in relational algebra, it is said to be relationally complete. A practical query language is expected to be relationally complete; in addition, commercial query languages typically support features that allow us to express some queries that cannot be expressed in relational algebra