**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System**

**Theory Assignment # 6**

**Submitted by:**

PRANESH DHUNJU SHRESTHA  
013BSCCSIT026

**Submitted to:**

|  |  |
| --- | --- |
| **Er. Sanjay Kumar Yadav**  Lecturer  St. Xavier’s College |  |

**Date of Submission:** September 10, 2015

* **RELATIONAL ALGEBRA**

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows −

* Select
* Project
* Union
* Set different
* Cartesian product
* Rename
* **JOIN**

JOIN is used to combine related tuples from two relations:

* In its simplest form the JOIN operator is just the cross product of the two relations.
* As the join becomes more complex, tuples are removed within the cross product to make the result of the join more meaningful.
* JOIN allows you to evaluate a join condition between the attributes of the relations on which the join is undertaken.

The notation used is

 R JOINjoin condition S

**JOIN Example**

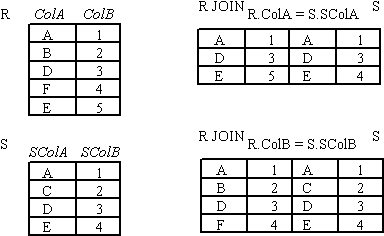
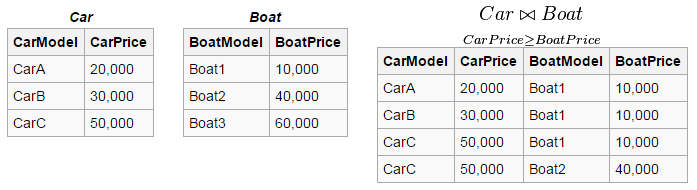


Figure : JOIN

### Θ-JOIN

Consider tables Car and Boat which list models of cars and boats and their respective prices. Suppose a customer wants to buy a car and a boat, but she does not want to spend more money for the boat than for the car. The θ-join on the [relation](https://en.wikipedia.org/wiki/Relation_(database)) CarPrice ≥ BoatPrice produces a table with all the possible options. When using a condition where the attributes are equal, for example Price, then the condition may be specified as Price=Price or alternatively (Price) itself.



* **NATURAL JOIN**

Invariably the JOIN involves an equality test, and thus is often described as an equi-join. Such joins result in two attributes in the resulting relation having exactly the same value. A `natural join' will remove the duplicate attribute(s).

* In most systems a natural join will require that the attributes have the same name to identify the attribute(s) to be used in the join. This may require a renaming mechanism.
* If you do use natural joins make sure that the relations do not have two attributes with the same name by accident.
* **OUTER JOIN**

Notice that much of the data is lost when applying a join to two relations. In some cases this lost data might hold useful information. An outer join retains the information that would have been lost from the tables, replacing missing data with nulls.

There are three forms of the outer join, depending on which data is to be kept.

* LEFT OUTER JOIN - keep data from the left-hand table
* RIGHT OUTER JOIN - keep data from the right-hand table
* FULL OUTER JOIN - keep data from both tables

**OUTER JOIN example**

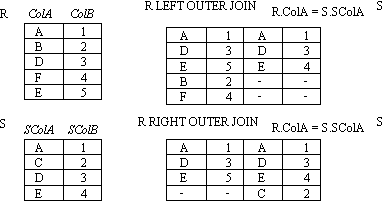
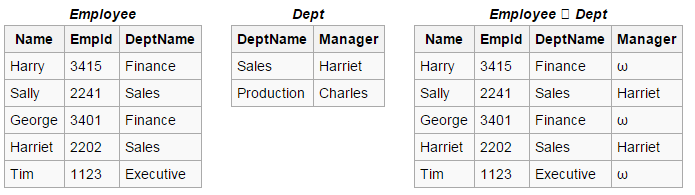


Figure : OUTER JOIN (left/right)

#### LEFT OUTER JOIN (⟕)

The left outer join is written as R ⟕ S where R and S are [relations](https://en.wikipedia.org/wiki/Relation_(database)). The result of the left outer join is the set of all combinations of tuples in R and S that are equal on their common attribute names, in addition (loosely speaking) to tuples in R that have no matching tuples in S.

For an example consider the tables Employee and Dept and their left outer join:

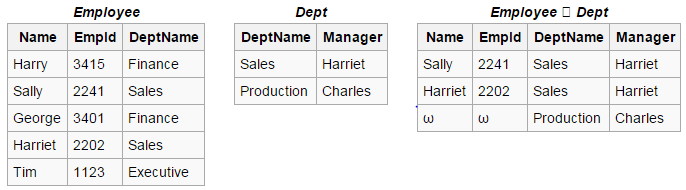


#### RIGHT OUTER JOIN (⟖)

The right outer join behaves almost identically to the left outer join, but the roles of the tables are switched.

The right outer join of [relations](https://en.wikipedia.org/wiki/Relation_(database)) R and S is written as R ⟖ S. The result of the right outer join is the set of all combinations of tuples in R and S that are equal on their common attribute names, in addition to tuples in S that have no matching tuples in R.

For example consider the tables Employee and Dept and their right outer join:



**INNER JOIN**

An inner join requires each record in the two joined tables to have matching records, and is a commonly used join operation in [applications](https://en.wikipedia.org/wiki/Application_software) but should not be assumed to be the best choice in all situations. Inner join creates a new result table by combining column values of two tables (A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied by matching non-NULL values, column values for each matched pair of rows of A and B are combined into a result row.

The result of the join can be defined as the outcome of first taking the [Cartesian product](https://en.wikipedia.org/wiki/Cartesian_product) (or [Cross join](https://en.wikipedia.org/wiki/Join_(SQL)#Cross_join)) of all records in the tables (combining every record in table A with every record in table B) and then returning all records which satisfy the join predicate. Actual SQL implementations normally use other approaches, such as [hash joins](https://en.wikipedia.org/wiki/Hash_join) or [sort-merge joins](https://en.wikipedia.org/wiki/Sort-merge_join), since computing the Cartesian product is slower and would often require a prohibitively large memory space to store.

## RENAME OPERATION (Ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** ρ.

**Notation** − ρ x (E)

Where the result of expression **E** is saved with name of **x**.

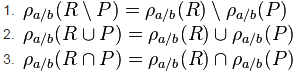
#### Basic rename properties

Successive renames of a variable can be collapsed into a single rename. Rename operations which have no variables in common can be arbitrarily reordered with respect to one another, which can be exploited to make successive renames adjacent so that they can be collapsed.



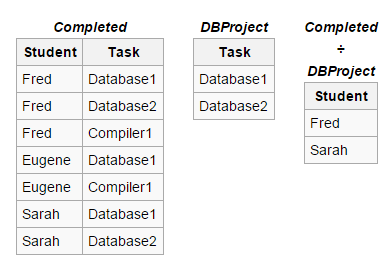
#### Rename and set operators

Rename is distributive over set difference, union, and intersection.



### DIVISION OPERATION(÷)

The division is a binary operation that is written as R ÷ S. The result consists of the restrictions of tuples in R to the attribute names unique to R, i.e., in the header of R but not in the header of S, for which it holds that all their combinations with tuples in S are present in R. For an example see the tables Completed, DBProject and their division:



If DBProject contains all the tasks of the Database project, then the result of the division above contains exactly the students who have completed both of the tasks in the Database project.

More formally the semantics of the division is defined as follows:



where {a1,...,an} is the set of attribute names unique to R and t[a1,...,an] is the restriction of t to this set. It is usually required that the attribute names in the header of S are a subset of those of R because otherwise the result of the operation will always be empty.

The simulation of the division with the basic operations is as follows. We assume that a1,...,an are the attribute names unique to R and b1,...,bm are the attribute names of S. In the first step we project R on its unique attribute names and construct all combinations with tuples in S:

T := πa1,...,an(R) × S

In the prior example, T would represent a table such that every Student (because Student is the unique key / attribute of the Completed table) is combined with every given Task. So Eugene, for instance, would have two rows, Eugene -> Database1 and Eugene -> Database2 in T.

In the next step we subtract R from T [relation](https://en.wikipedia.org/wiki/Relation_(database)):

U := T − R

Note that in U we have the possible combinations that "could have" been in R, but weren't. So if we now take the projection on the attribute names unique to R then we have the restrictions of the tuples in R for which not all combinations with tuples in S were present in R:

V := πa1,...,an(U)

So what remains to be done is take the projection of R on its unique attribute names and subtract those in V:

W := πa1,...,an(R) − V

* **ASSIGNMENT OPERATION**

• You’ve probably gotten a sense, particularly with division, that relational algebra feels

a lot like programming: there are many steps to some expressions, with intermediate

or temporary relations along the way.

3

• For this very reason, we have the assignment operation, which works a lot like assignments

in a programming language. It is notated with the left-pointing arrow ←:

variable ← E

where E is any relational algebra expression.

• The assignment operation is more of a notational convenience rather than a real relational

operation — it helps human beings with writing out complex relational expressions

in steps so that they can be more easily understood.

• Revisiting the breakdown of the division operation, we can use assignment to rewrite

(1) this way:

temp1 ← ΠR−S(r)

temp2 ← ΠR−S((temp1 × s) − ΠR−S,S(r))

r ÷ s = temp1 − temp2

* **ADDITIONAL OPERATION**

• “Additional operations” refer to relational algebra operations that can be expressed in terms of the fundamentals — select, project, union, set-difference, cartesian-product, and rename.

• The compositions of these operations are so lengthy, yet so common, that we define new operations for them, based on the fundamentals. Kind of a mathematical “syntactic sugar.”

* **SET-INTERSECTION**

• The set-intersection operation is a binary operation on relations r and s that is denoted by the traditional intersection symbol, ∩. r ∩ s results in all tuples t such that (t ∈ r) ∧ (t ∈ s). 1

• Set-intersection is defined in terms of set-difference: r ∩ s = r − (r − s)

• Thus, set-intersection must follow the same compatibility rules as set-difference: same arity, corresponding domains.

* **NATURAL-JOIN**

• The natural-join operation is a binary operation on relations r(R) and s(S) that is denoted by the symbol ./. Intuitively, a natural-join “matches” the tuples of r with the tuples of s based on attributes that are both in r and s.

• If we take the relational schemas R and S as sets of attributes, then we can define “attributes that are in both r and s” as R ∩ S = {A1, A2, . . . , An}. With that, we can formally define r ./ s as: r ./ s = ΠR ∪ S(σr.A1 = s.A1 ∧ r.A2 = s.A2 ∧ ... ∧ r.An = s.An (r × s))

• Note that R ∪ S removes duplicate attribute names, so r ./ s will only have one attribute Ak ∀Ak ∈ R ∩ S.

• Natural join is associative — that is, (a ./ b) ./ c = a ./ (b ./ c).

• When r and s do not have any common attributes — i.e., R ∩ S = ∅ — then r ./ s = r × s.

* **REFERENCE :**

[1] <http://www.tutorialspoint.com/dbms/relational_algebra.htm>

[2] http://myweb.lmu.edu/dondi/share/db/relational3.pdf