Benchmarks for Temporal Databases

C. Vassilakis - A. Sotiropoulou

How the proposed benchmarks for temporal databases can be incorporated to TOOBIS

1.0 TSQL2 Benchmark

1.1 Schema for TSQL2 Benchmark

Three interfaces will be defined according to the TSQL2 Benchmark:

- The `Employee` interface
- The `Department` interface, and
- The `Skill` interface

The following TODL statements may be used for the definition of the above interfaces[1]

```
enum GenderType {male, female};
interface Skill
(extent Skills,
 key name)
{
          attribute String name;
}
```

1. Note that the time varying nature of the different attributes/relationships is defined in the appropriate report from TSQL2 Committee

```
interface Employee
(extent Employees,
 key name)
{
          readonly attribute String id;[1]
          attribute String name valid;
          attribute Long salary valid
               granularity month;
          attribute GenderType gender;
          attribute Instant granularity day d_birth;
          relationship Department belongsInDept
               valid granularity day
               inverse Department::hasEmployee;
          relationship Department managerInDept
               valid granularity day
               inverse Department::hasManager;
          relationship Set<Skill> hasSkills
               valid granularity day;
}
```

```
interface Department
(extent Departments,
 key name)
{
          attribute String name;
          attribute Long budget valid granularity day;
          relationship Set<Employee> hasEmployee
               valid granularity day
               inverse Employee::belongsInDept;
          relationship Employee hasManager
               valid granularity day
               inverse Employee::managerInDept;
}
```

1. This is required due to the nature of the queries. This attribute will be set through the “constructor” method and there will be no operation allowing this value to change.

1.2 Queries - Explicit-attribute Output

1.2.1 Class O1.S1 (Duration, Interval, Computing)

Query Q 2.1.1: Which departments had managers who served for the shortest continuous period?

```
select d.name from Departments as d
where min(select duration(valid(m)) from
                valid d.hasManager as m) =
           min(select duration(valid(m1))
                from Departments as d1,
                valid d1.hasManager as m1)
```

Result: `Query type is bag<string>`

Query Q 2.1.2: Who worked continuously in the Book department for at least as long as Di did?

```
select e.name from Employees as e
where exists d in valid e.belongsInDept:
           duration(valid(d)) >=
           max(select duration(valid(d1))
                from Employees as e1,
                     valid e1.belongsInDept as d1
                where e1.name = “Di” and
                     d1.name = “Book”)
```

Result: `Query type is bag<string>`

Query Q 2.1.3: Who worked continuously in the Toy department for at least as long as Di did?

```
select e.name from Employees as e
where exists d in valid e.belongsInDept:
           (duration(valid(d)) >=
           max(select duration(valid(d1))
                from Employees as e1,
                     (valid e1.belongsInDept) as d1
                     where d1->name = “Toy”))
```

Result: `Query type is bag<string>`

Query Q 2.1.4: Who worked continuously in a department longer than their current manager worked in their (that) department?

```
select distinct e.name from Employees as e
where exists d in valid e.belongsInDept:
                duration(valid(d)) >
           sum(select duration(valid(d1))
                from valid
                e.belongsInDept.hasManager.belongsInDept
                     as d1
                where d1.name = d.name)
```

Result: `Query type is set<string>`

Query Q 2.1.5: Who had the same salary for the longest continuous time period?

```
select e.name from Employees as e
where max(select duration(valid(s))
               from valid e.salary as s) =
          max(select duration(valid(s1))
               from Employees as e1, valid
                    e1.salary as s1)
```

Result: `Query type is bag<string>`

Query Q 2.1.6: Who worked for a manager in a department for a period as long as that manager managed the department?

```
select distinct e1.name from Employees as e,
          valid e.belongsInDept as d,
          (valid d.hasManager)[valid(d)] as m
group by e as e1, d as d1, m.value as m2
having sum(select duration(valid(x.d))
               from partition as x) >=
          sum(select duration(valid(m1))
               from valid d1.hasManager as m1
               where m1.name = m2.name)
```

Result: `Query type is set<string>`

Query Q 2.1.7: Which managers served continuously longer than some other manager?

```
select distinct d.hasManager.name
from Departments as d,
          (valid d.hasManager) as mgr
where exists (select * from Departments as d1,
                    valid d1.hasManager as mgr1
               where duration(valid(mgr)) >
                    duration(valid(mgr1)))
```

Result: `Query type is set<string>`

1.2.2 Class O1.S2 (Duration, Interval, Other)

Query Q 2.2.1: Which employees had the same salary for a single period of at least three years?

```
select e.name
from Employees as e,
          (valid e.salary) as sal
where duration(valid(sal)) >=
          interval "3" granularity Year
```

Result: `Query type is bag<bag<string>>`

Query Q 2.2.2: Who worked for the same manager for at least five years continuously?

```
select e.name from Employees as e
where exists m1 in valid[select m.value from
          valid e.belongsInDept as d,
          (valid d.hasManager)[valid(d)] as m:m.VT]:
          duration(valid(m1)) > interval "5"
               granularity Year
```

Result: Error: The historical equivalent of this class is not defined (Employee - Employee_Historical_State)

Query Q 2.2.3: Which employees have stayed in the same department throughout the past 5 years?

```
select e.name from Employees as e
where exists d in valid e.belongsInDept: valid(d)
          contains period(now() - interval "5"
          granularity Year, now())
```

Result: `Query type is bag<string>`

Query Q 2.2.4: For each department which has had the same managers and budget for the last 18 months, list its current name, manager and budget.

```
select d.name, d.hasManager, d.budget
from Departments as d
where exists m in valid d.hasManager: valid(m)
               contains period(now()- interval "18"
               granularity Month, now()) and
          exists b in valid d.budget: valid(b)
               contains period(now() - interval "18"
               granularity Month, now())
```

Result: `Query type is bag<struct {name: string, hasManager: Employee, budget: integer}>`

Query Q 2.2.5: Who has worked in the Toy department and has earned at least 40K throughout the last two years?

```
select e.name from Employees as e
where exists d in valid e.belongsInDept: valid(d)
          contains period(now() - interval "2"
               granularity Year, now()) and
          40000 <= all (select s.value
               from (valid e.salary)[period(now() -
               interval "2" granularity Year, now())]
                    as s)
          and d.name = "Toy"
```

Result: `Query type is bag<string>`

Query Q 2.2.6: Who had at least three raises in a continuous five-year period?

```
select e.name from Employees as e,
          (1 .. count(valid e.salary) - 2) as counter
where count(valid e.salary) > 4 and
          (end(valid((valid e.salary)[counter+3])) -
          begin(valid((valid e.salary)[counter])))  <
               interval "5" granularity Year and
          (valid e.salary)[counter] <
               (valid e.salary)[counter+1] and
          (valid e.salary)[counter+1] <
               (valid e.salary)[counter+2] and
          (valid e.salary)[counter+2] <
               (valid e.salary)[counter+3]
```

Result: `Query type is bag<string>`

Query Q 2.2.7: Who had the most raises in a continuous five-year period?

```
select p.id
from Employees as p,
          (partition valid as interval "1" granularity
               Year trailing interval "4" granularity
               Year)(valid p.salary)  as five_year_sal
where count(select *
          from 1 .. count(five_year_sal.partition)-1
               as i
          where (five_year_sal.partition)[i].value <
               (five_year_sal.partition)[i+1].value) =
          max(select count(select *
               from 1 .. count(fys.partition)-1 as i1
               where (fys.partition)[i1].value <
                    (fys.partition)[i1+1].value)
               from Employees p1,
                    (partition valid as interval "1"
                         granularity Year trailing
                         interval "4" granularity Year)
                         (valid p1.salary) as fys)
```

Result: `Query type is bag<string>`

1.2.3 Class O1.S3 (Duration, Element, Computed)

Query Q 2.3.1: Who worked in the Toy department for at least as long as DI worked there?

```
select e.name from Employees as e
where sum(select duration(valid(d))
               from valid e.belongsInDept as d
               where d.name = "Toy") >=
          sum(select duration(valid(d1))
               from Employees as e1,
                    valid e1.belongsInDept as d1
               where d1.name = "Toy" and e1.id = "DI")
```

Result: `Query type is bag<string>`

Query Q 2.3.2: Who worked in a department longer than their current manager worked in that department?

```
select distinct f.name from Employees as e,
          valid e.belongsInDept as d
group by e as f, d as g
having sum(select duration(valid(x.d))
               from partition as x) >
          sum(select duration(valid(d1))
               from valid
               f.belongsInDept.hasManager.belongsInDept
                    as d1
               where d1.name = g.name)
```

Result: `Query type is set<string>`

Query Q 2.3.3: Which managers managed which departments, longer than Di managed the Toy department?

```
select mx.name as Manager, dx.name as Department
from Departments as d,
          valid d.hasManager as m
group by d as dx, m as mx
having sum(select duration(valid(x.m))
               from partition as x) >
          sum(select duration(valid(m1))
               from Departments as d1,
                    valid d1.hasManager as m1
               where d1.name = "Toy" and m1.name = "Di")
```

Result: `Query type is bag<struct {Manager: string, Department: string}>`

Query Q 2.3.4: Who had the same salary for the longest total time?

```
select ex.name from Employees as e,
          valid e.salary as s
group by e as ex,s as sx
having sum(select duration(valid(x.s))
               from partition as x) =
          max(select sum(select duration(valid(x1.s1))
                         from partition as x1)
               from Employees as e1,
                    valid e1.salary as s1
               group by e1 as e1x, s1 as s1x)
```

Result: `Query type is bag<string>`

Query Q 2.3.5: Which departments had managers who served for the shortest total time?

```
select d.name from Departments as d
where exists m in valid d.hasManager:
        sum(select duration(valid(m1))
                        from valid d.hasManager as m1
                        where m1.id = m.id) =
        min(select sum(select duration(valid(x1.m2))
                                from partition as x1)
                from Departments as d1,
                        valid d1.hasManager as m2
                group by d1 as d1x, m2 as m2x)
```

Result: `Query type is bag<string>`

Query Q 2.3.6: List all employees currently in the Book department who received salaries of over 40K longer than ED did.

```
select e.name from Employees as e
where e.belongsInDept.name = "Book" and
          sum(select duration(valid(s))
               from valid e.salary as s
               where s > 40000) >
          sum(select duration(valid(s1))
               from Employees as e1,
                    valid e1.salary as s1
               where e1.id = "ED" and s1 > 40000)
```

Result: `Query type is bag<string>`

Query Q 2.3.7: Who worked in the Toy department for at least as long as the total time that the Toy department was NOT managed by ED?

```
select e.name from Employees as e
where sum(select duration(valid(d))
               from valid e.belongsInDept as d
               where d.name = "Toy") >=
          sum(select duration(valid(m))
               from Departments as d1,
                    valid d1.hasManager as m
               where d1.name = "Toy" and m.id != "ED")
```

Result: `Query type is bag<string>`

Query Q 2.3.8: Find the names of employees that have been in a department named Toy for a shorter period than has DI.

```
select e.id
from Employees as e
where sum(select duration(valid(d))
          from valid e.belongsInDept as d
          where d.name = "Toy") <=
          sum(select duration(valid(d1))
              from Employees as e1,
                   valid e1.belongsInDept as d1
              where e1.id = "DI" and d1.name = "Toy")
```

Result: `Query type is bag<string>`

Query Q 2.3.9: Find the current name and department for the employees which made $40K for a longer period than DI did.

```
select e.name, e.belongsInDept
from Employees as e
where (exists sal in valid e.salary: sal >= 40000) and
          sum(select duration(valid(s))
          from valid e.salary as s
          where s >= 40000)>
          sum(select duration(valid(s1))
              from Employees as e1,
                   valid e1.salary as s1
              where e1.id = "DI" and
                   s1 >= 40000)
```

Result: `Query type is bag<struct {name: string, belongsInDept: Department}>`

1.2.4 Class O1.S4 (Duration, Element, Other)

Query Q 2.4.1: Who managed the Book department for at least two years?

```
select e.id
from Employees as e
where (exists mgr in valid e.managerInDept:
          mgr.name = "Book") and
          sum(select duration(valid(m))
          from valid e.managerInDept as m
          where m.name = "Book") >=
          interval "2" granularity Year
```

Result: `Query type is bag<string>`

Query Q 2.4.2: Which employees had the same salary for at least three years?

```
select distinct ex.id
from Employees as e,
          valid e.salary as s
group by e as ex, s as sx
having sum(select duration(valid(x.s))
          from partition as x) >=
          interval "3" granularity Year
```

Result: `Query type is set<string>`

Query Q 2.4.3: Who worked for the same manager for at least five years?

```
select e.name from Employees as e
where exists m1 in valid[select m.value
                    from valid e.belongsInDept as d,
                         (valid d.hasManager)[valid(d)]
                              as m:m.VT]:
          duration(valid(m1)) > interval "5"
               granularity Year
```

Result:

```
Error: The historical equivalent of this class is not
defined (Employee - Employee_Historical_State)
```

Query Q 2.4.4: Who worked in a department for less than 6 months total?

```
select e.name from Employees as e
where exists (select dx from valid e.belongsInDept as d
               group by d as dx
               having sum(select duration(valid(x.d))
                              from partition as x) <
               interval "6" granularity Month)
```

Result: `Query type is bag<string>`

Query Q 2.4.5: Who worked for the same manager for total time of at least five years?

```
select e1.id
from Employees as e,
          valid e.belongsInDept as d,
          (valid d.hasManager)[valid(d)] as m
group by e as e1, m.value as mgr
having sum(select duration(x.m.VT)
          from partition as x) >=
          interval "5" granularity Year
```

Result: `Query type is bag<string>`

1.2.5 Class O1.S5 (Other, Event, Computed)

Query Q 2.5.1: Find ED’s skills when he joined the Book department.

```
select (valid e.hasSkills)[valid at begin(valid(d))]
from Employees as e,
          valid e.belongsInDept as d
where e.id = “ED” and d.name = “Book”
```

Result: `Query type is bag<set<Skill>>`

Query Q 2.5.2: Find the name and the budget of ED’s departments when he joined them.

```
select d.name, (valid d.budget)[begin(valid(d))] as bud
from Employees as e, valid e.belongsInDept as d
where e.id = “ED”
```

Result: `Query type is bag<struct {name: string, bud: integer}>`

Query Q 2.5.3: For each employee who was in the Toy department when it opened, find all data and skills that were valid at the time.

```
select emp.id,
        (valid emp.name)[valid at
        begin(valid((valid d.hasEmployee)[0]))]
                as n,
        (valid emp.salary)[valid at
        begin(valid(((valid d.hasEmployee)[0])))]
                as s,
        emp.gender, emp.d_birth,
        (valid emp.belongsInDept)[valid at
        begin(valid(((valid d.hasEmployee)[0])))]
                as dept,
        (valid emp.managerInDept)[valid at
        begin(valid(((valid d.hasEmployee)[0])))]
                as man,
        (valid emp.hasSkills)[valid at
        begin(valid(((valid d.hasEmployee)[0])))]
                as skills
from Departments as d,
        ((valid d.hasEmployee)[0]) as emp
where d.name = “Toy”
```

Result: `Query type is bag<struct {id: string, n: string, s: integer, gender: integer, d_birth: instant granularity day calendar Gregorian, dept: Department, man: Department, skills: set<Skill>}>`

Query Q 2.5.4: Find the names valid when the budget of the Toy department was decreased of the employees who had been working in the Toy department before the budget was decreased.

```
select (select (valid e.name)[begin(valid((valid
                      d.budget)[i]))]
           from (valid d.hasEmployee)[begin(valid((valid
                      d.budget)[i]))] as e) as names
from Departments as d,
           (0 .. (count(valid d.budget)-2)) as i
where d.name = "Toy" and
           (valid d.budget)[i] > (valid d.budget)[i+1]
```

Result: `Query type is bag<struct {names: bag<string>}>`

Query Q 2.5.5: Find ED's skills when his salary increased from \$30K to \$40K.

```
flatten(select (valid e.hasSkills)[valid at
begin(valid(s))]
from Employees as e, valid e.salary as s
where e.id = "ED" and s = 40000 and
           exists s1 in valid e.salary:
                (s1 = 30000 and
                end(valid(s1)) = begin(valid(s))))
```

Result: `Query type is set<Skill>`

1.2.6 Class O1.S6 (Duration, Element, Other)

Query Q 2.6.1: Find the name, current budget and manager of the Toy department.

```
select d.name, d.budget, d.hasManager
from Departments as d
where d.name = "Toy"
```

Result: `Query type is bag<struct {name: string, budget: integer, hasManager: Employee}>`

Query Q 2.6.2: Find the skills for which ED became qualified after 1/1/83.

```
element(   select (select s.value
                from (valid e.hasSkills)[valid at
                     period(instant "1983-01-01"
                          granularity Day, now())] as s
                except
                select s1.value
                from (valid e.hasSkills)[valid at
                     period(instant "beginning",
                          instant "1983-01-01"
                          granularity Day)] as s1)
           from Employees as e where e.id = "ED")
```

Result: `Query type is bag<set<Skill>>`

Query Q 2.6.3: Find DI’s salary on her 25th birthday.

```
select (valid e.salary) [valid at (e.d_birth +
                         interval "25"
                         granularity Year)]
from Employees as e
where e.id = "DI"
```

Result: `Query type is bag<integer>`

Query Q 2.6.4: Find the departments ED worked in before and not after 1/1/88.

```
select d.name from Employees as e,
          valid e.belongsInDept as d
where e.id = "ED" and valid(d) precedes
          instant "1988-01-01" granularity Day and
          not exists d1 in valid e.belongsInDept:
               (d1.name = d.name and instant
               "1988-01-01"
               granularity Day precedes valid(d1))
```

Result: `Query type is bag<string>`

Query Q 2.6.5: Find the date of birth and current name of the women who were working in the Toy department on 1/1/83.

```
select e.d_birth, e.name
from Employees as e
where e.gender = 0 and
          (valid e.belongsInDept)[instant "1983-01-01"
          granularity Day].name = "Toy"
```

Result: `Query type is bag<struct {d_birth: instant granularity day calendar Gregorian, name: string}>`

Query Q 2.6.6: Who worked in their current department for a longer time than their current manager worked in that department?

```
select e.id
from Employees as e
where sum(select duration(valid(d))
          from valid e.belongsInDept as d
          where d.name = e.belongsInDept.name) >
          sum(select duration(valid(d1))
               from Employees as e1,
                    valid e1.belongsInDept as d1
               where e1.id =
e.belongsInDept.hasManager.id and
                    d1.name = e.belongsInDept.name)
```

Result: `Query type is bag<string>`

1.2.7 Class O1.S7 (Other, Interval, Computed)

Query Q 2.7.1: Find the names of all employees that changed department while DI was working in a department called Toy.

```
select e.name from Employees as e
where exists d in valid e.belongsInDept :
          exists(select * from Employees as e1,
                    valid e1.belongsInDept as d1
               where e1.id = “DI” and d1.name = “Toy”
                    and begin(valid(d)) overlaps
                         valid(d1))
```

Result: `Query type is bag<string>`

Query Q 2.7.2: Which of all the skills ever recorded did ED not acquire while working in the Book department?

```
select sk.name from Skills as sk
except
flatten(select (select bs.name
               from (valid e.hasSkills)[i] except
                    (valid e.hasSkills)[i-1] as bs
               where (valid e.belongsInDept)[begin(
                    valid((valid e.hasSkills)[i]))].
                    name = “Book”)
          from Employees as e,
               2 .. count(valid e.hasSkills) as i
          where e.id = “ED” and
          count(valid e.hasSkills) >= 2)
```

Result: `Query type is bag<string>`

Query Q 2.7.3: Of the skills at some time possessed by ED, list those he did not acquire while he was working in the Book department.

```
select s.name
from Employees as e1,
          valid e1.hasSkills as skills,
          skills as s
except
flatten(select (select bs.name
               from (valid e.hasSkills)[i] except
                    (valid e.hasSkills)[i-1] as bs
               where (valid e.belongsInDept)[begin(
                    valid((valid e.hasSkills)[i]))].
                         name = “Book”)
          from Employees as e,
               2 .. count(valid e.hasSkills) as i
          where e.id = “ED” and
          count(valid e.hasSkills) >= 2)
```

Result: `Query type is bag<string>`

Query Q 2.7.4: For any employee who re-acquired a skill, find the name of the employee when a skill was re-acquired.

```
select (valid e.name)[valid at begin(valid(s))]
from Employees as e,
          valid e.hasSkills as s, s as as1,
          valid e.hasSkills as s1,
          valid e.hasSkills as s2
where begin(valid(s)) > begin(valid(s1)) and
          begin(valid(s1)) > begin(valid(s2)) and
          as1 in s2 and not (as1 in s1)
```

Result: `Query type is bag<string>`

Query Q 2.7.5: Find the gender and name at the time of all employees that started working in some department before ED acquired the skill Driving for the second time.

```
select e.gender as g,
          (valid e.name)[valid at begin(valid(d))] as n
from Employees as e, valid e.belongsInDept as d,
          Employees as e1, valid e1.hasSkills as s,
          s as ask
where e1.id = "ED" and ask.name = "Driving" and
          begin(valid(d)) precedes valid(s) and
          count(select s1
               from valid e1.hasSkills as s1,
                    s1 as as1
               where as1.name = "Driving" and
                    valid(s1) precedes valid(s)) = 1
```

Result: `Query type is bag<struct {g: integer, n: string}>`

Query Q 2.7.6: Who worked in a department for a given manager for at least the period when that manager managed the department?

```
select e.id
from Employees as e,
          valid e.belongsInDept as d,
          (valid d.hasManager)[valid(d)] as m
where not exists (select *
               from Departments as d1,
                    valid d1.hasManager as m1
               where d1.name = d.name and
                    m1.id = m.value.id and
                    not exists d2 in valid
                         e.belongsInDept: valid(d2)
                         contains valid(m1))
```

Result: `Query type is bag<string>`

Query Q 2.7.7: What was the highest salary earned by ED before changing his name to Edward?

```
max(select s from Employees as e,
               valid e.salary as s
          where e.id = "ED" and begin(valid(s))
               precedes
               min(select valid(n)
                    from valid e.name as n
                    where n = "Edward"))
```

Result: `Query type is integer`

1.2.8 Class O1.S8 (Other, Interval, Other)

Query Q 2.8.1: Find the name and skill pairs sometime possessed by people who worked in the Book or Toy department last year.

```
select ns.name, ns.skill
from Employees as e,
          tstruct(name: valid e.name,
               skill: valid e.hasSkills) as ns
where exists d in (valid e.belongsInDept)[period(now() -
               interval "2" granularity Year, now() -
               interval "1" granularity Year)]:
          d.value.name = "Book" or d.value.name = "Toy"
```

Result: `Query type is bag<struct {name: string, skill: set<Skill>}>`

Query Q 2.8.2: Find the current name and skills of all people who worked for the Book or Toy department last year.

```
select e.name, e.hasSkills
from Employees as e
where exists d in (valid e.belongsInDept)[period(now() -
               interval "2" granularity Year, now() -
               interval "1" granularity Year)] :
          d.value.name = "Toy" or d.value.name = "Book"
```

Result: `Query type is bag<struct {name: string, hasSkills: set<Skill>}>`

Query Q 2.8.3: Find their names (when they reported to DI) for all people who reported to DI before last year.

```
select (valid e.name)[valid at mgr.VT]
from Employees as e,
          valid e.belongsInDept as d,
          (valid d.hasManager)[valid(d)] as mgr
where mgr.value.id = "DI" and
          mgr.VT precedes now() - interval "2"
               granularity Year
```

Result: `Query type is bag<list struct {value: string, VT: period}>`

Query Q 2.8.4: Find the current manager of anyone who acquired a skill between 1983 and 1987 inclusive.

```
select e.belongsInDept.hasManager
from Employees as e
where count(valid e.hasSkills) > 1 and
          exists i in 2 .. count(valid e.hasSkills) :
               (period "[1983-1-1, 1988-1-1)"
               granularity Day contains begin(valid(
               (valid e.hasSkills)[i])) and
               count((valid e.hasSkills)[i] except
                    (valid e.hasSkills)[i-1]) > 0)
```

Result: `Query type is bag<Employee>`

Query Q 2.8.5: Find the name current of anyone who lost a skill in the last four years.

```
select e.name
from Employees as e
where count(valid e.hasSkills) > 1 and
          exists i in 2 .. count(valid e.hasSkills) :
               (period(now() - interval "4" granularity
               Year, now()) granularity Year contains
               begin(valid((valid e.hasSkills)[i])) and
               count((valid e.hasSkills)[i-1] except
                    (valid e.hasSkills)[i]) > 0)
```

Result: `Query type is bag<string>`

Query Q 2.8.6: Find the current name and department of anyone who changed their name or salary between July 1987 and June 1988 inclusive.

```
select e.name, e.belongsInDept
from Employees as e
where exists (select *
          from (valid e.salary)
               [period "[1987-07, 1988-08)"] as s1,
               (valid e.salary)
               [period "[1987-07, 1988-08)"] as s2
          where s1.value != s2.value)
or exists (select *
          from (valid e.name)
               [period "[1987-07, 1988-08)"] as n1,
               (valid e.name)
               [period "[1987-07, 1988-08)"] as n2
          where n1.value != n2.value)
```

Result: `Query type is bag<struct {name: string, belongsInDept: Department}>`

Query Q 2.8.7: Which employees stayed at their first salary for less than one year?

```
select e.name
from Employees as e
where duration(valid((valid e.salary)[0])) <
          interval "1" granularity Year
```

Result: `Query type is bag<string>`

Query Q 2.8.8: List the names and current managers and budgets of all departments with budgets of less than 200K during any period between January 1, 1985 and December 31, 1989.

```
select d.name as name, d.hasManager as manager,
          d.budget
from Departments as d
where exists b in ((valid d.budget)
          [period "[1985-01-01,1990-01-01)"]):
          b.value < 200000
```

Result: `Query type is bag<struct {name: string, manager: Employee, budget: integer}>`

Query Q 2.8.9: Who worked in the Toy department at some point and earned at least 40K throughout the last two years?

```
select e.name
from Employees as e,
          snapshot e.belongsInDept as dep_name
where ("Toy" = dep_name.name) and
          (for all s in (valid e.salary)
          [period(now()-interval "2" granularity Year,
          now())] : s.value >= 40000)
```

Result: `Query type is bag<string>`

1.2.9 Class O1.S9 (Other, Element, Computed)

Query Q 2.9.1: Find the names of departments that always had a budget greater than $90K during the times when managed by someone named Di.

```
select d.name
from Departments as d
where for all bm in tstruct(budget: valid d.budget,
                             manager: valid d.hasManager):
        (bm.budget > 90000 or
        bm.manager.name = "Di")
```

Result: `Query type is bag<string>`

Query Q 2.9.2: Find ED's salaries when he worked in the same department as DI.

```
select s.value as salary
from flatten(select (valid e1.salary)[commonPeriod.VT]
          from Employees as e1,
               Employees as e2,
               tstruct(EdDept: valid e1.belongsInDept,
                    DiDept: valid e2.belongsInDept)
                    as commonPeriod
          where e1.id = "ED" and e2.id = "DI") as s
```

Result: `Query type is bag<struct {salary: integer}>`

Query Q 2.9.3: Find the names of the departments that ED worked in while earning $40K.

```
select d.value.name as DeptName
from flatten(select (valid e.belongsInDept)[valid(s)]
             from Employees as e, valid e.salary as s
             where e.id = "ED" and s = 40000) as d
```

Result: `Query type is bag<struct {DeptName: string}>`

Query Q 2.9.4: Find ED's names after he left the Toy department.

```
select n.value as name
from flatten (select (valid e.name)
               [period(end(valid(d)), now())]
          from Employees as e,
               valid e.belongsInDept as d
          where e.id = "ED" and
               d.name = "Toy") as n
```

Result: `Query type is bag<struct {name: string}>`

Query Q 2.9.5: Find the skills that ED possessed sometime when he worked in the Toy department.

```
flatten(select sd.skills
          from Employees as e,
               tstruct(skills: valid e.hasSkills,
                    dept: valid e.belongsInDept) as sd
          where e.id = "ED"
               and sd.dept.name = "Toy")
```

Result: `Query type is set<Skill>`

Query Q 2.9.6: What new skills did ED obtain after he changed his name to Edward?

```
select (select ns.name
          from valid e.hasSkills as nsk, nsk as ns
          where cp precedes valid(nsk)) except
          (select os.name
          from valid e.hasSkills as osk, osk as os
          where begin(valid(osk)) precedes cp)
from (select e
          from Employees as e
          where e.id = "ED") as e,
          (select begin(valid(n))
               from valid e.name as n
               where n = "Edward") as cp
```

Result: `Query type is bag<string>`

Query Q 2.9.7: What where Toy's departmental budgets when it had a manager named Di?

```
select bm.budget as budget, n.VT as when
from Departments as d,
          tstruct(budget: valid d.budget,
               manager: valid d.hasManager) as bm,
          (valid bm.manager.name)[bm.VT] as n
where d.name = "Toy" and n.value = "Di"
```

Result: `Query type is bag<struct {budget: integer, when: instant granularity Second calendar Gregorian}>`

1.2.10 Class O1.S10 (Other, Element, Other)

Query Q 2.10.1: Which managers managed which departments between January 1, 1982 and December 31, 1989?

```
select d.name as DeptName, m as Manager
from Departments as d, valid d.hasManager as m
where valid(m) overlaps period "[1982-1-1, 1990-1-1)"
```

Result: `Query type is bag<struct {DeptName: string, Manager: Employee}>`

1.3 Queries - Valid-time Output

1.3.1 Class O2.S1 (Duration, Interval, Computed)

Query Q 3.1.1: Find the times when persons with a shorter employment in the Toy department than DI were employed in the Book department.

```
select valid(d)
from Employees as e,
          valid e.belongsInDept as d
where sum(select duration(valid(d1))
          from valid e.belongsInDept as d1
          where d1.name = "Toy") <
          sum(select duration(valid(d2))
               from Employees as e1,
                    valid e1.belongsInDept as d2
               where e1.id = "DI" and
                    d2.name = "Toy") and
          d.name = "Book"
```

Result: `Query type is bag<period granularity Day calendar Gregorian>`

Query Q 3.1.2: Find the employment periods of persons that made 40K for a longer time than DI made 40K.

```
select valid(d)
from Employees as e, valid e.belongsInDept as d
where sum(select duration(valid(s))
          from valid e.salary as s
          where s >= 40000) >
          sum(select duration(valid(s1))
               from Employees as e1,
                    valid e1.salary as s1
               where e1.id = "DI" and s1 >= 40000)
```

Result: `Query type is bag<period granularity Day calendar Gregorian>`

Query Q 3.1.3: Find the starting times in the Book department of persons which possessed the Filing skill for a longer time than DI.

```
select begin(valid(inBook))
from Employees as e, valid e.belongsInDept as inBook
where inBook.name = "Book" and
          sum(select duration(valid(s))
               from valid e.hasSkills as s
               where exists aSkill in s:
                    aSkill.name = "Filling") >
          sum(select duration(valid(s1))
               from Employees as e1,
                    valid e1.hasSkills as s1
               where e1.id = "DI" and
                    exists aSkill in s1:
                         aSkill.name = "Filling")
```

Result: `Query type is bag<instant granularity Day calendar Gregorian>`

Query Q 3.1.4: Return the times when persons employed a shorter time than DI acquired a skill.

```
select (begin(valid(newSkill)))
from Employees as e,
          1 .. (count(valid e.hasSkills) - 1) as i,
          (valid e.hasSkills) as newSkill
where count((valid e.hasSkills)[i+1] except
              (valid e.hasSkills)[i]) > 0 and
          sum(select duration(valid(d))
              from valid e.belongsInDept as d) >
          sum(select duration(valid(d1))
              from Employees as e1,
                   valid e1.belongsInDept as d1
              where e1.id = "DI")
```

Result: `Query type is bag<instant granularity Day calendar Gregorian>`

Query Q 3.1.5: Find the employment periods of persons employed shorter time than DI.

```
select valid(d)
from Employees as e, valid e.belongsInDept as d
where sum(select duration(valid(d))
          from valid e.belongsInDept as d1) <
          sum(select duration(valid(d2))
              from Employees as e2,
                   valid e2.belongsInDept as d2
              where e2.id = "DI")
```

Result: `Query type is bag<period granularity Day calendar Gregorian>`

Query Q 3.1.6: When did someone get a raise more quickly than DI got her first raise?

```
select begin(valid((valid e.salary)[i]))
from Employees as e,
          2 .. (count(valid e.salary)-1) as i
where count(valid e.salary) >= 2 and
          (valid e.salary)[i] > (valid e.salary)[i-1]
          and (begin(valid((valid e.salary)[i])) -
              begin(valid((valid e.salary)[i-1]))) <
          (select begin(valid((valid e1.salary)[i1])) -
              begin(valid((valid e1.salary)[i1-1]))
          from Employees as e1,
              2 .. (count(valid e1.salary) -1) as i1
          where e1.name = "DI" and
              (valid e1.salary)[i1] >
              (valid e1.salary)[i1-1]
          order by i)[1]
```

Result: `Query type is bag<instant granularity Month calendar Gregorian>`

Query Q 3.1.7: What was the longest period when no one was hired or left unemployed?

```
max(select duration(p)
    from periods(period_set(period(instant "1970-01-01",
                         now())) granularity Day -
                period_set(select begin(valid (e))
                             from Departments as d,
                                 valid d.hasEmployee as e)
                         granularity Day) as p)
```

Result: `Query type is interval granularity Day calendar Gregorian`

Note: We suppose that we are interested in time periods after the year 1970.

Query Q 3.1.8: What was the longest period when no one received a raise?

```
max(select duration(p)
from periods(period_set(period(instant "1970-01-01",
                         now())) granularity day -
               period_set(select begin(valid((valid
                              e.salary)[i]))
                         from Employees as e,
                         2 .. count(valid e.salary) as i
                         where count(valid e.salary) > 1
                              and (valid e.salary)[i] >
                              (valid e.salary)[i-1])
           granularity day) as p)
```

Result: `Query type is interval granularity day calendar Gregorian`

Query Q 3.1.9: When was the longest period when a department was without a manager?

```
max(select duration(valid(m))
           from Departments as d,
                valid d.hasManager as m
           where m = nil)
```

Result: `Query type is interval granularity Day calendar Gregorian`

1.3.2 Class O2.S2 (Duration, Element, Other)

Query Q 3.2.1: Find employment periods in the Toy department for persons that have worked there for at least 8 years.

```
select valid(dept)
from Employees as e,
          valid e.belongsInDept as dept
where dept.name = "Toy" and
          sum(select duration(valid(d))
               from valid e.belongsInDept as d
               where d.name = "Toy") >=
          interval "8" granularity Year
```

Result: `Query type is bag<period granularity Day calendar Gregorian>`

Query Q 3.2.2: Find the starting times of managers which managed a department for at least 5 years.

```
select begin(valid(mgr))
from Departments as d, valid hasManager as mgr
where sum(select duration(valid(m))
          from valid d.hasManager as m
          where m.id = mgr.id) > interval "5"
granularity Year
```

Result: `Query type is bag<instant granularity Day calendar Gregorian>`

Query Q 3.2.3: Find the rehiring dates of employees with a gap in employment that exceeds 1 month.

```
select begin(valid((valid e.belongsInDept)[i]))
from Employees as e,
          2 .. (count(valid e.belongsInDept)-1) as i
where count(valid e.belongsInDept) >= 2 and
          (begin(valid((valid e.belongsInDept)[i])) -
          end(valid((valid e.belongsInDept)[i-1]))) >
               interval "1" granularity Month
```

Result: `Query type is bag<instant granularity Day calendar Gregorian>`

Query Q 3.2.4: Find the times when persons possessed skills that they lost and regained more than 1 year later.

```
select valid(s)
from Employees as e, valid e.hasSkills as s
where exists aSkill in s:
          (exists s1 in valid e.hasSkills:
          (begin(valid(s1)) > (end(valid(s)) +
          interval "1" granularity Year) and
          aSkill in s1 and
          not (exists s2 in valid e.hasSkills:
               (begin(valid(s2)) < end(valid(s)) and
               begin(valid(s1)) > end(valid(s2)) and
               aSkill in s2))))
```

Result: `Query type is bag<period granularity Day calendar Gregorian>`

Query Q 3.2.5: Find budget periods that exceed 2 years.

```
select valid(budgets)
from Departments as d,
          (valid d.budget) as budgets
where duration(valid(budgets)) >
          interval "2" granularity Year
```

Result: `Query type is bag<period granularity Day calendar Gregorian>`

Query Q 3.2.6: When did no one's salary change for at least six months?

```
select p
from periods(period(instant "1970-01-01" granularity
Day,
        now()) granularity Day -
        period_set(select begin(valid(s))
                        from Employees as e,
                              valid e.salary as s)
                granularity Day) as p
where duration(p) > interval "6" granularity Month
```

Result: `Query type is bag<period granularity Day calendar Gregorian>`

1.3.3 Class O2.S3 (Duration, Element, Computed)

Query Q 3.3.1: When did somebody have the same salary for the longest continuous period?

```
select valid(sal)
from Employees as e,
          (valid e.salary) as sal
where duration(valid(sal)) =
               max(select duration(valid(sal1))
                    from Employees as e1,
                         valid e1.salary as sal1)
```

Result: `Query type is bag<period granularity Month calendar Gregorian>`

Query Q 3.3.2: When did anybody work for a manager in a department for as long as that manager managed that department?

```
select valid(d)
from Employees as e, valid e.belongsInDept as d
where exists m in (valid d.hasManager)[valid(d)]:
          not exists (select *
               from valid d.hasManager as m1
               where m1 = m.value and
                    not exists (select *
                    from valid e.belongsInDept as d1
                    where d1 = d and
                         (valid(d1) contains m.VT)))
```

Result: `Query type is bag<period granularity Day calendar Gregorian>`

Query Q 3.3.3: When did someone manage the Toy department for longer than DI did?

```
select valid(ToyMgr)
from Departments as d, valid d.hasManager as ToyMgr
where d.name = “Toy” and
          sum(select duration(valid(m1))
               from valid d.hasManager as m1
               where m1.id = ToyMgr.id) >
          sum(select duration(valid(m2))
               from valid d.hasManager as m2
               where m2.id = “DI”)
```

Result: `Query type is bag<period granularity Day calendar Gregorian>`

Query Q 3.3.4: When did anyone have a skill longer than ED had Driving?

```
select valid(skills)
from Employees as e, valid e.hasSkills as skills,
          Skills as s
where sum(select duration(valid(skills1))
          from valid e.hasSkills as skills1
          where s in skills1) >
          sum(select duration(valid(skills2))
               from Employees as e1,
                    valid e1.hasSkills as skills2
               where e1.id = “ED” and
                    exists s2 in skills2:
                         s2.name = “Driving”)
```

Result: `Query type is bag<period granularity Day calendar Gregorian>`

1.3.4 Class O2.S4 (Duration, Element, Other)

1.3.5 Class O2.S5 (Other, Event, Computed)

1.3.6 Class O2.S6 (Other, Event, Other)

Query Q 3.6.1: When did anybody have at least the skills that DI currently has?

```
select valid(s)
from Employees as e, valid e.hasSkills as s
where count(element(select e1.hasSkills
          from Employees as e1
          where e1.id = "DI") except s) = 0
```

Result: `Query type is bag<period granularity Day calendar Gregorian>`

1.3.7 Class O2.S7 (Other, Interval, Computed)

Query Q 3.7.1: When did the Toy budget decrease?

```
select begin(valid((valid d.budget)[i]))
from Departments as d,
          1 .. (count(valid d.budget) - 1) as i
where d.name = "Toy" and
          count(valid d.budget) >= 2 and
          (valid d.budget)[i-1] > (valid d.budget)[i]
```

Result: `Query type is bag<instant granularity Day calendar Gregorian>`

Query Q 3.7.2: When did an employee change name?

```
select begin(valid((valid e.name)[i]))
from Employees as e,
          1 .. (count(valid e.name) - 1)  as i
where count(valid e.name) >= 2 and
          (valid e.name)[i-1] != (valid e.name)[i]
```

Result: `Query type is bag<instant granularity Second calendar Gregorian>`

Query Q 3.7.3: When did the salary of an employee increase while the employee was a manager?

```
select begin(valid((valid e.salary)[i]))
from Employees as e, 1 .. (count(valid e.salary)-1) as i
where count(valid e.salary) >= 2 and
          (valid e.salary)[i-1] < (valid e.salary)[i]
          and exists d in Departments:
               ((valid d.hasManager)
               [begin(valid((valid e.salary)[i]))]).id
                    = e.id
```

Result: `Query type is bag<instant granularity Month calendar Gregorian>`

Query Q 3.7.4: Find the periods during which DI earned $40K and was manager of the Toy department.

```
select valid(s) intersect valid(m)
from Employees as e, Departments as d,
          valid e.salary as s, valid d.hasManager as m
where e.id = "DI" and d.name = "Toy" and
          s = 40000 and valid(s) overlaps valid(m)
          and m.id = e.id
```

Result: `Query type is bag<period granularity Month calendar Gregorian>`

Query Q 3.7.5: Find the acquisition dates of the skills ED acquired before or during the year he joined the Toy department.

```
select distinct begin(valid((valid e.hasSkills)[i]))
from Employees as e,
          1 .. (count(valid e.hasSkills)-1) as i
where e.id = "ED" and
          count(valid e.hasSkills) >= 2 and
          year(begin(valid((valid e.hasSkills)[i]))) <=
               min(select year(begin(valid(d)))
                    from Employees as e1,
                         valid e1.belongsInDept as d
                    where e1.id = "ED" and
                         d.name = "Toy") and
          count((valid e.hasSkills)[i] except
                    (valid e.hasSkills)[i-1]) > 0
```

Result: `Query type is set<instant granularity Day calendar Gregorian>`

1.3.8 Class O2.S8 (Other, Interval, Other)

Query Q 3.8.1: Find the beginning of a continuous period in which ED was named Edward, in which he had a constant salary, and which includes the year 1989.

```
select begin(valid(n) intersect valid(s))
from Employees as e, valid e.name as n,
          valid e.salary as s
where e.id = "ED" and n = "Edward" and
          (valid(n) intersect valid(s)) contains
          period "[1989-01-01, 1990-01-01)"
          granularity Year
```

Result: `Query type is bag<instant granularity Second calendar Gregorian>`

Query Q 3.8.2: Find the dates, before or after the years 1984 and 1985, when ED acquired a skill.

```
select begin(valid((valid e.hasSkills)[i]))
from Employees as e,
          1 .. (count(valid e.hasSkills)-1) as i
where e.id = "ED" and
          count(valid e.hasSkills) >= 2 and
          begin(valid((valid e.hasSkills)[i])) precedes
          instant "1984-01-01" granularity Day and
          instant "1985-12-31" granularity Day precedes
          begin(valid((valid e.hasSkills)[i])) and
          count(((valid e.hasSkills)[i]) except
               ((valid e.hasSkills)[i-1])) > 0
```

Result: `Query type is bag<instant granularity Day calendar Gregorian>`

Query Q 3.8.3: Find ED's unemployment periods when he was not exactly 30 years old.

```
period_set(element(select
period(begin(valid(first(valid  e.belongsInDept))),
        max(set(now(), end(valid(last(valid
e.belongsInDept))))))
        from Employees as e
        where e.id = "ED"))
except period_set(select valid(d)
        from Employees as e1,
                valid e1.belongsInDept as d
        where e1.id = "ED")
except period_set(element(select period(e2.d_birth +
        interval "30" granularity Year, e2.d_birth+
        interval "31" granularity Year)
        from Employees as e2
        where e2.id = "ED"))
```

Result: `Query type is period set granularity Second calendar Gregorian`

Query Q 3.8.4: Find the time periods when DI worked in the department in which she has been working during all of 1987.

```
select valid(d)
from Employees as e, valid e.belongsInDept as d
where e.id = "DI" and
          d in (select d1
               from valid e.belongsInDept as d1
               where valid(d1) contains
                    period "[1987-01-01, 1988-01-01)"
                         granularity Day)
```

Result: `Query type is bag<period granularity Day calendar Gregorian>`

Query Q 3.8.5: Find all the dates, between 1/1/83 and 12/31/85, when the Toy department budget changed.

```
select begin(valid((valid d.budget)[i]))
from Departments as d,
          1 .. (count(valid d.budget)-1) as i
where count(valid d.budget) >= 2 and
          (valid d.budget)[i] !=
               (valid d.budget)[i-1] and
          period "[1983-01-01, 1986-01-01)"
               granularity Day contains
          begin(valid((valid d.budget)[i]))
```

Result: `Query type is bag<instant granularity Day calendar Gregorian>`

1.3.9 Class O2.S9 (Other, Element, Computed)

Query Q 3.9.1: At what times did an employee simultaneously possess at least the same skills that DI possessed?

```
select joinedSkills.VT
from Employees as e,
          tstruct(someSkills: valid e.hasSkills,
               DISkills: element(select
                    valid e1.hasSkills
                    from Employees as e1
                    where e1.id = "DI")) as joinedSkills
where count(joinedSkills.DISkills except
          joinedSkills.someSkills) = 0 and
          e.id != "DI"
```

Result: `Query type is bag<period granularity Gregorian calendar Day>`

Query Q 3.9.2: When was the budget for Toy department more than 100K?

```
select valid(budg)
from Departments as d,
          (valid d.budget) as budg
where d.name = "Toy" and budg > 100000
```

Result: `Query type is bag<period granularity Day calendar Gregorian>`

Query Q 3.9.3: What was the last continuous period when ED was named Edward and had Driving, Filing and Typing as skills simultaneously?

```
(select ns.VT
from Employees as e,
          tstruct(name: valid e.name,
               skills: valid e.hasSkills) as ns
where e.id = "ED" and ns.name = "Edward" and
          count(set("Filing", "Driving", "Typing")
               except
               (select s.name from ns.skills as s)) = 0
order by begin(ns.VT) desc)[0]
```

Result: `Query type is period granularity Gregorian calendar Second`

Query Q 3.9.4: When did DI earn less than ED?

```
select joinedSalary.VT
from Employees as e1, Employees as e2,
          tstruct(DISal: valid e1.salary,
               EDSal: valid e2.salary) as joinedSalary
where e1.id = "DI" and e2.id = "ED" and
          joinedSalary.DISal < joinedSalary.EDSal
```

Result: `Query type is bag<period granularity Gregorian calendar Month>`

Query Q 3.9.5: When did ED work in Toy department while the department was managed by DI?

```
select mgr.VT
from Employees as e,
          valid e.belongsInDept as d,
          (valid d.hasManager)[valid(d)] as mgr
where e.id = "ED" and
          d.name = "Toy" and
          mgr.value.id = "DI"
```

Result: `Query type is bag<instant granularity Day calendar Gregorian>`

Query Q 3.9.6: When did an employee currently named Edward have driving skills?

```
select valid(skills)
from Employees as e, valid e.hasSkills as skills
where e.name = "Edward" and exists s in skills :
          s.name = "Driving"
```

Result: `Query type is bag<period granularity Day calendar Gregorian>`

1.3.10 Class O2.S10 (Other, Element, Other)

1.4 Queries - Explicit-attribute and Valid-time output

1.4.1 Class O3.S1 (Duration, Interval, Computed)

Query Q 4.1.1: Who, and when, were continuously employed in the Toy department shorter than DI was continuously employed in the Toy department?

```
select e.id, valid(d) as dx
from Employees as e, valid e.belongsInDept as d
where d.name = “Toy” and duration(valid(d)) < any
          (select duration(valid(d1))
          from Employees as e1,
               valid e1.belongsInDept as d1
          where e1.id = “DI” and d1.name = “Toy”)
```

Result: `Query type is bag<struct {id: string, dx: period granularity Day calendar Gregorian}>`

Query Q 4.1.2: Who, and when, were continuously employed in the Toy department shorter than DI was continuously employed in the Toy department, and what their gender and date of birth?

```
select e.id, valid(d) as dt, e.gender, e.d_birth
from Employees as e, valid e.belongsInDept as d
where d.name = “Toy” and duration(valid(d)) < any
          (select duration(valid(d1))
          from Employees as e1,
               valid e1.belongsInDept as d1
          where e1.id = “DI” and d1.name = “Toy”)
```

Result: `Query type is bag<struct {id: string, dt: period granularity Day calendar Gregorian, gender: integer, d_birth: instant granularity day calendar Gregorian}>`

Query Q 4.1.3: Who were continuously employed in the Toy department shorter than DI was continuously employed in the Toy department, and when did this employment start?

```
select e.id, begin(valid(d)) as dt
from Employees as e, valid e.belongsInDept as d
where d.name = “Toy” and duration(valid(d)) < any
          (select duration(valid(d1))
          from Employees as e1,
               valid e1.belongsInDept as d1
          where e1.id = “DI” and d1.name = “Toy”)
```

Result: `Query type is bag<struct {id: string, dt: instant granularity Day calendar Gregorian}>`

Query Q 4.1.4: Return the name on 01-Jan.-1984 along with the date 01-Jan.-1984 for each employee who was continuously employed in the Toy department shorter than Di was continuously employed in that department.

```
select (valid e.name)[instant "1984-1-1"
          granularity Day] as name, instant "1984-1-1"
          granularity Day as when
from Employees as e, valid e.belongsInDept as d
where d.name = "Toy" and duration(valid(d)) < any
          (select duration(valid(d1))
          from Employees as e1,
               valid e1.belongsInDept as d1
          where e1.id = "DI" and d1.name = "Toy")
```

Result: `Query type is bag<struct {name: string, when: instant granularity Day calendar Gregorian}>`

1.4.2 Class O3.S2 (Duration, Interval, Other)

Query Q 4.2.1: When was the Toy department's budget constant and greater than $175K for more than one year, and what was the budget?

```
select valid(b) as vb, b
from Departments as d, valid d.budget as b
where d.name = "Toy" and
          duration(valid(b)) > interval "1"
               granularity Year and
          b > 175000
```

Result: `Query type is bag<struct {vb: period granularity Day calendar Gregorian, b: integer}>`

Query Q 4.2.2: When was the Toy department's budget constant and greater than $175K for more than one year, and who was the manager for that time?

```
select valid(b) as vb, (select m.value.id as mvi
               from (valid d.hasManager)[valid(b)]
                         as m)
          as vbs
from Departments as d, valid d.budget as b
where d.name = "Toy" and b > 175000 and
          duration(valid(b)) > interval "1"
               granularity Year
```

Result: `Query type is bag<struct {vb: period granularity Day calendar Gregorian, vbs: bag<struct {mvi: string}>}>`

Note: The result schema of this query is bag<struct<period, set<string>>> i.e. each period is associated with a set of id, since during the 1-year period, the department may have changed its manager.

Query Q 4.2.3: Who managed a department with a budget that exceeded $175K and then held constant for one year, and when did that occur?

```
select m.id, valid(m) as when
from Departments as d, (valid d.hasManager) as m
where exists b in (valid d.budget):
          (b > 175000 and
          period(begin(valid(m)),
               begin(valid(m))+interval "1"
               granularity Year) contains valid(b))
```

Result: `Query type is bag<struct {id: string, when: period granularity Day calendar Gregorian>`

Query Q 4.2.4: What departments were in continuous operation (and when) longer than the duration between ED's and DI's birth dates?

Note: We consider that a department is operational if it has at least one employee.

```
select d.name, valid(e) as when
from Departments as d, valid d.hasEmployee as e
where count(e) > 0 and
          duration(valid(e)) >
               abs(element(select e1.d_birth
                    from Employees as e1
                    where e1.id = "ED") -
                    element(select e2.d_birth
                    from Employees as e2
                    where e2.id = "DI"))
```

Result: `Query type is bag<struct {name: string, when: period granularity Day calendar Gregorian}>`

Query Q 4.2.5: Who worked in one department for at least two years continuously and what were the periods of employment in that department?

```
select e.id, valid(d) as vd
from Employees as e, valid e.belongsInDept as d
where (exists d1 in valid e.belongsInDept:
          ((duration(valid(d1)) >
               interval "2" granularity Year)
          and d1.name = d.name))
```

Result: `Query type is bag<struct {id: string, vd: period granularity Day calendar Gregorian}>`

1.4.3 Class O3.S3 (Duration, Element, Computed)

Query Q 4.3.1: Who, when, and for which department did anybody work for as long as the length of time that department's budget was below 200K?

```
select e.id, d.name, valid(d) as vd
from Employees as e, valid e.belongsInDept as d
where sum(select duration(valid(d1))
          from valid e.belongsInDept as d1
          where d1.name = d.name) >
          sum(select duration(valid(b))
               from valid d.budget as b
               where b < 200000)
```

Result: `Query type is bag<struct {id: string, name: string, vd: period granularity Day calendar Gregorian}>`

Query Q 4.3.2: Who and when did anybody work in a department longer than their current manager worked in that department?

```
select e.id, valid(d) as vd
from Employees as e, valid e.belongsInDept as d
where sum(select duration(valid(d1))
          from valid e.belongsInDept as d1
          where d1.name = d.name) >
          sum(select duration(valid(d2))
               from Employees as e1,
                    valid e1.belongsInDept as d2
               where e1.id =
                    e.belongsInDept.hasManager.id
                    and d2.name = d.name)
```

Result: `Query type is bag<struct {id: string, vd: period granularity Day calendar Gregorian}>`

Query Q 4.3.3: For all employees who managed any departments at least as long as DI managed the Toy department, list their names, their gender, their departments and their salary histories during that time.

```
select (valid e.name)[valid(m)] as vname, e.gender,
          (valid e.belongsInDept)[valid(m)] as vdept,
          (valid e.salary)[valid(m)] as vsal
from Employees as e, valid e.managerInDept as m
where sum(select duration(valid(m1))
          from valid e.managerInDept as m1
          where m1.name = m.name) >
          sum(select duration(valid(m2))
               from Employees as e1,
                    valid e.managerInDept as m2
               where e1.id = “DI” and
                    m2.name = “Toy”)
```

Result: `Query type is bag<struct {vname: list struct {value: string, VT: period granularity Second calendar Gregorian}, gender: integer, vdept: list struct {value:`

Department, VT: period granularity Day calendar Gregorian}, vsal: list struct {value: integer, VT: period granularity Month calendar Gregorian}}>

Query Q 4.3.4: For departments that had a manager that served for the shortest total time, list the department name, the shortest-serving manager(s) and the times when those managers served the department.

```
select d.name, m.id, valid(m) as vm
from Departments as d, valid d.hasManager as m
where sum(select duration(valid(m1))
          from valid d.hasManager as m1
          where m1.id = m.id) =
          min(select sum(select duration(valid
(x.mgr1))
                   from partition as x)
               from Departments as d1,
                   valid d1.hasManager as mgr1
               group by mgr1.id, d1.name)
```

Result: Query type is bag<struct {name: string, id: string, vm: period granularity Day calendar Gregorian}>

Query Q 4.3.5: For all departments which had budgets of at least 200K for a longer total time than budgets of less than 200K, list their names, budgets and times when the budgets were not below 200K.

```
select d.name, (valid d.budget)[valid(b)] as vbb,
valid(b) as vb
from Departments as d, valid d.budget as b
where b >= 200000 and
          sum(select duration(valid(b1))
               from valid d.budget as b1
               where b1 >= 200000) >
          sum(select duration(valid(b2))
               from valid d.budget as b2
               where b2 < 200000)
```

Result: Query type is bag<struct {name: string, vbb: list struct {value: integer, VT: period granularity Day calendar Gregorian}, vb: period granularity Day calendar Gregorian}>

Query Q 4.3.6: What skills did ED hold for at least as long as the total time during his employment that he did not have the Driving skill, and when did he have those skills?

```
select longSkill, (select x.aSkill.skillPeriod
                from partition as x) as longSkillPeriod
from Employees as e,
        (select s.name as skillName,
                valid(s1) as skillPeriod
        from valid e.hasSkills as s1, s1 as s)
                as aSkill
where e.id = "ED"
group by aSkill.skillName as longSkill
having sum(select duration(x.aSkill.skillPeriod)
        from partition as x) >
        (element(select
                end(valid((valid
                        e1.belongsInDept)[count
                        (valid e1.belongsInDept)-1])) -
                begin(valid((valid
                        e1.belongsInDept)[0]))
        from Employees as e1
        where e1.id = "ED")) -
        sum(select duration(valid(s2))
                from Employees as e2,
                        valid e2.hasSkills as s2
                where not exists someSkill in s2:
                        someSkill.name = "Driving")
```

Result: `Query type is bag<struct {longSkill: string, longSkillPeriod: bag<period granularity Day calendar Gregorian>}>`

1.4.4 Class O3.S4 (Duration, Element, Other)

Query Q 4.4.1: Find the oldest employee who was a Typist on 12/31/85, and the times when that employee had been a Typist so far.

```
(select e.id, (select valid(s1)
               from valid e.hasSkills as s1
               where exists oneSkill in s1:
                    oneSkill.name = "Typist") as when
from Employees as e
where "Typist" in (select s.name
                    from (valid e.hasSkills)
                         [instant "1985-12-31"] as s)
order by e.d_birth asc)[0]
```

Result: `Query type is struct {id: string, when: bag<period granularity Day calendar Gregorian>}`

Query Q 4.4.2: For employees in the Toy department who had worked less than DI in that department as of 1/1/85, find their names on 1/1/85 and the time difference on 1/1/85.

```
select (valid e.name)[instant "1985-1-1"] as name,
          EmpDuration - DIDuration as TimeDiff
from Employees as e,
          set(sum(select duration(d.VT)
               from (valid e.belongsInDept)
                    [period(instant "beginning",
                         instant "1985-1-1")] as d
               where d.value.name = "Toy"))
                    as EmpDuration,
          set(sum(select duration(d1.VT)
               from Employees as e1,
                    (valid e1.belongsInDept)
                    [period(instant "beginning",
                         instant "1985-1-1")] as d1
               where e1.id = "DI" and
                    d1.value.name = "Toy"))
                    as DIDuration
where DIDuration > EmpDuration
```

Result: `Query type is bag<struct {name: string, TimeDiff: interval granularity Day calendar Gregorian}>`

Query Q 4.4.3: Find the current employees who worked at least during 1987, and the times in 1987 during which they worked.

```
select e.id, (select d.VT
                from (valid e.belongsInDept)[period
                       "[1987-01-01, 1988-01-01)"] as d)
                as times
from Employees as e
where period_set(select d1.VT
                from (valid e.belongsInDept)
                      [period "[1987-01-01, 1988-01-01)"
                                granularity Day] as d1)
        granularity Day =
        period_set(period "[1987-01-01, 1988-01-01)")
                granularity Day
```

Result: `Query type is bag<struct {id: string, times: bag<period granularity Day calendar Gregorian>}>`

1.4.5 Class O3.S5 (Other, Event, Computed)

Query Q 4.5.1: List the names and ages of all employees at the time they received their first salary increment.

```
select (valid e.name)[firstInc] as Name,
          firstInc - e.d_birth as Age
from Employees as e,
          (set(min(select begin(valid((valid
                    e.salary)[i]))
               from 1..(count(valid e.salary)-1) as i
               where count(valid e.salary) >= 2 and
                    (valid e.salary)[i-1] <
                    (valid e.salary)[i]))
          union set(instant ”forever”)) as firstInc
where firstInc != instant “forever”
```

Result: `Query type is bag<struct {Name: string, Age: interval granularity Month calendar Gregorian}>`

Query Q 4.5.2: List the name and salary histories up until their 25th birthday of all female employees.

```
select (valid e.name)[period (e.d_birth, e.d_birth +
               interval “25” granularity Year)] as name,
          (valid e.salary)[period (e.d_birth, e.d_birth
               + interval “25” granularity Year)]
          as salary
from Employees as e
where e.gender = 0
```

Result: `Query type is bag<struct {name: list struct {value: string, VT: period granularity Second calendar Gregorian}, salary: list struct {value: integer, VT: period granularity Month calendar Gregorian}}>`

Query Q 4.5.3: When and who ever changed their names?

```
select e.id, (select begin(valid((valid e.name)[i]))
          as b
               from 1..(count(valid e.name)-1) as i
               where count(valid e.name) >=2 and
                    (valid e.name)[i] !=
                    (valid e.name)[i-1]) as s1
from Employees as e
where count(distinct(snapshot e.name)) > 1
```

Result: `Query type is bag<struct {id: string, s1: bag<struct {b: instant granularity Second calendar Gregorian}>}>`

Query Q 4.5.4: How old was ED and what skills did he have at the time he changed his name to Edward?

```
select changeDate - e.d_birth as age,
           (valid e.hasSkills)[changeDate] as skills
from Employees as e,
           (select begin(valid((valid e.name)[i]))
           from 1..(count(valid e.name)-1) as i
           where count(valid e.name) >=2 and
                (valid e.name)[i] = "Edward" and
                (valid e.name)[i-1] != "Edward")
                     as changeDate
where e.id = "ED"
```

Result: `Query type is bag<struct {age: interval granularity Second calendar Gregorian, skills: set<Skill>}>`

Query Q 4.5.5: When did ED acquire the Driving skill, and what other skills did he have at the time?

```
select beginDriving, (valid e.hasSkills)[beginDriving]
as skills
from Employees as e,
           set(min(select begin(valid(s))
                from valid e.hasSkills as s
                where exists aSkill in s:
                     aSkill.name = "Driving"))
                          as beginDriving
where e.id = "ED"
```

Result:
`Query type is bag<struct {beginDriving: instant granularity Day calendar Gregorian, skills: set<Skill>}>`

Query Q 4.5.6: Who was the first female manager of the Toy department, and when did she become manager of that department for the first time?

```
(select m.id, begin(valid(m)) as when
from Departments as d, valid d.hasManager as m
where d.name = "Toy" and m.gender = 0
order by begin(valid(m)) asc)[0]
```

Result: `Query type is struct {id: string, when: instant granularity Day calendar Gregorian}`

1.4.6 Class O3.S6 (Other, Event, Other)

Query Q 4.6.1: Find the name and salary histories of employees whose date-of-birth was after 1/1/56.

```
select valid e.name, valid e.salary
from Employees as e
where instant "1956-01-01" precedes e.d_birth
```

Result: `Query type is bag<struct {name: attribute string valid granularity Second calendar Gregorian, salary: attribute integer valid granularity Month calendar Gregorian}>`

Query Q 4.6.2: Find the name and salary histories of employees who were called “Ed” after 1/1/88

```
select valid e.name, valid e.salary
from Employees as e
where exists n in (valid e.name)[period(instant
          "1988-01-01", instant "forever")]:
               n.value = "Ed"
```

Result: `Query type is bag<struct {name: attribute string valid granularity Second calendar Gregorian, salary: attribute integer valid granularity Month calendar Gregorian}>`

Query Q 4.6.3: Find the name and salary histories since their latest pay raise of employees whose latest pay raise was in 1985.

```
select (valid e.name)[period(latestRaise, now())] as n1,
          (valid e.salary)[period(latestRaise, now())]
as sal1
from Employees as e,
          set(max(select begin(valid(
                    (valid e.salary)[i]))
               from 2..count(valid e.salary) as i
               where (valid e.salary)[i] >
                    (valid e.salary)[i-1]))
                         as latestRaise
where count(valid e.salary) > 2 and
          year(latestRaise) = 1985
```

Result:

```
Query type is bag<struct {n1: list struct {value:
string, VT: period granularity Second calendar
Gregorian}, sal1: list struct {value: integer, VT:
period granularity Month calendar Gregorian}}>
```

Query Q 4.6.4: Find the name and salary histories of employees whose latest pay raise occurred after the date-of-birth of every other employee.

```
select valid e.name, valid e.salary
from Employees as e
where max(select e1.d_birth from Employees as e1)
          precedes
          max(select begin(valid((valid e.salary)[i]))
               from 1..(count(valid e.salary)-1) as i
               where count(valid e.salary) >= 2 and
                    (valid e.salary)[i] >
                    (valid e.salary)[i-1])
```

Result: `Query type is bag<struct {name: attribute string valid granularity Second calendar Gregorian, salary: attribute integer valid granularity Month calendar Gregorian}>`

Query Q 4.6.5: Find the name and salary histories of employees whose latest pay raise occurred on the date-of-birth of some other employee.

```
select valid e.name, valid e.salary
from Employees as e
where max(select begin(valid((valid e.salary)[i]))
          from 1..(count(valid e.salary)-1) as i
          where count(valid e.salary) >=2 and
               (valid e.salary)[i] >
               (valid e.salary)[i-1]) =
          any(select e1.d_birth from Employees as e1)
```

Result: `Query type is bag<struct {name: attribute string valid granularity Second calendar Gregorian, salary: attribute integer valid granularity Month calendar Gregorian}>`

Query Q 4.6.6: Who and when had at least the skills that DI currently has?

```
select e.id, (valid(s)) as when
from Employees as e, valid e.hasSkills as s
where count(element(select e1.hasSkills
               from Employees as e1
               where e1.id = "DI")
          except s) = 0
```

Result: `Query type is bag<struct {id: string, when: period granularity Day calendar Gregorian}>`

1.4.7 Class O3.S7 (Other, Interval, Computed)

Query Q 4.7.1: For the time before ED first worked in the Toy department, find the salary paid, and the time it was paid, of any employee who was in the Toy department at a time before Ed worked there.

```
select e.id, (valid e.salary)[period(instant
"beginning",
          EdJoinToy - interval "1" granularity Day)] as
sal
from Employees as e,
          set(min(select begin(valid(d1))
               from Employees as e1,
                    valid e1.belongsInDept as d1
               where e1.id = "ED" and
                    d1.name = "Toy")) as EdJoinToy
where exists d in valid e.belongsInDept:
          (d.name = "Toy" and
          begin(valid(d)) precedes EdJoinToy)
```

Result:
```
Query type is bag<struct {id: string, sal: list struct
{value: integer, VT: period granularity Month calendar
Gregorian}}>
```

Query Q 4.7.2: Find the greatest salary under $50K paid to ED and the times during which it was paid.

```
(select targetSal, (select valid(x.s) as vxs from
                         partition as x) as when
from Employees as e,
          valid e.salary as s
where e.id = "ED" and s < 50000
group by s as targetSal
order by targetSal desc)[0]
```

Result: `Query type is struct {targetSal: integer, when: bag<struct {vxs: period granularity Month calendar Gregorian}>}`

Query Q 4.7.3: Find ED's salary history.

```
select valid e.salary
from Employees as e
where e.id = "ED"
```

Result: `Query type is bag<attribute integer valid granularity Month calendar Gregorian>`

Query Q 4.7.4: For employees that were drivers and simultaneously made less than $40K, find the names, salaries and times during which this occurred.

```
select (valid e.name)[sal_skill.VT] as name,
          (valid e.salary)[sal_skill.VT] as sal,
sal_skill.VT
from Employees as e,
          tstruct(aSal: valid e.salary,
               Skills: valid e.hasSkills) as sal_skill
where sal_skill.aSal < 40000 and
          (exists aSkill in sal_skill.Skills :
               aSkill.name = "Driving")
```

Result: `Query type is bag<struct {name: list struct {value: string, VT: period granularity Second calendar Gregorian}, sal: list struct {value: integer, VT: period granularity Month calendar Gregorian}, VT: period granularity Gregorian calendar Month}>`

Query Q 4.7.5: For the Toy department when ED worked there, find the budgets and associated times.

```
select (valid d.budget)[EdsLabour] as vbudg, EdsLabour
from Departments as d,
          (select valid(es)
          from valid d.hasEmployee as es
          where exists e in es:
               e.id = "ED") as EdsLabour
where d.name = "Toy"
```

Result: `Query type is bag<struct {vbudg: list struct {value: integer, VT: period granularity Day calendar Gregorian}, EdsLabour: period granularity Day calendar Gregorian}>`

1.4.8 Class O3.S8 (Other, Interval, Other)

Query Q 4.8.1: For all employees that have been in the Book or Toy departments sometime during the last two years, find their current names and their skills together with the times when they were valid.

```
select e.name, valid e.hasSkills
from Employees as e
where exists d in (valid e.belongsInDept) [period(now()-
          interval "2" granularity Year, now())]:
          d.value.name = "Book" or d.value.name = "Toy"
```

Result: `Query type is bag<struct {name: string, hasSkills: relationship set<Skill> valid granularity Day calendar Gregorian}>`

Query Q 4.8.2: Find the current names of all people who reported to DI before last year along with the time when they reported to her.

```
select e.name, mgr.VT
from Employees as e,
          valid e.belongsInDept as d,
          (valid d.hasManager)[valid(d)] as mgr
where mgr.value.id = "DI" and
          begin(mgr.VT) precedes
               now() - interval "1" granularity Year
```

Result: `Query type is bag<struct {name: string, VT: period granularity Day calendar Gregorian}>`

Query Q 4.8.3: Find the manager and time when a skill was acquired between 1983 and 1987 (inclusive) by anyone who acquired a skill between these times.

```
select (valid(valid e.belongsInDept)[acquireSkill].
                hasManager)[acquireSkill] as manager,
            acquireSkill
from Employees as e,
            (select begin(valid((valid e.hasSkills)[i]))
            from 2..count(valid e.hasSkills) as i
            where count((valid e.hasSkills)[i] except
                        (valid e.hasSkills)[i-1]) > 0)
                as acquireSkill
where count(valid e.hasSkills) > 1 and
            acquireSkill overlaps period
            "[1983-01-01,1988-01-01)" granularity Day
```

Result: `Query type is bag<struct {manager: Employee, acquireSkill: instant granularity Day calendar Gregorian}>`

Query Q 4.8.4: For anyone who had two raises in the period March 1982 and March 1985 (inclusive), find the current name and the dates when raises occurred during the aforementioned times.

```
select e.name, raiseDates
from Employees as e,
            set(select begin(valid((valid e.salary)[i]))
                from 1..(count(valid e.salary)-1) as i
                where count(valid e.salary) >=2 and
                      (valid e.salary)[i] >
                      (valid e.salary)[i-1] and
                      begin(valid((valid e.salary)[i]))
                      overlaps period("1982-3", "1985-4")
                          granularity Month)
                              as raiseDates
where count(raiseDates) = 2
```

Result: `Query type is bag<struct {name: string, raiseDates: bag<instant granularity Month calendar Gregorian>}>`

1.4.9 Class O3.S9 (Other, Element, Computed)

Query Q 4.9.1: Find the salary history associated with the name Ed when it was associated with a person that had the Driving skill.

```
flatten(select (valid e.salary)[ns.VT]
from Employees as e,
            tstruct(name: valid e.name,
                skills: valid e.hasSkills) as ns
where ns.name = "Ed" and
            exists aSkill in ns.skills :
                aSkill.name = "Driving")
```

Result: `Query type is set<struct {value: integer, VT: period granularity Month calendar Gregorian}>`

Query Q 4.9.2: Find the salary history, during the periods in which ED had a driving skill, of the employees who earned less than $50K throughout 1989.

```
select e.id, (valid e.salary)[EdDriving] as salary
from Employees as e,
          (select valid(s)
          from Employees as e1,
               valid e1.hasSkills as s
          where e1.id = "ED" and
               exists aSkill in s:
               aSkill.name = "Driving") as EdDriving
where not exists aSal in (valid e.salary)[period
          "[1989-01-01, 1990-01-01)" granularity Year]:
          aSal.value > 50000
```

Result: `Query type is bag<struct {id: string, when: list struct {value: integer, VT: period granularity Month calendar Gregorian}}>`

Query Q 4.9.3: Find the name and salary histories of male employees when they were directed by a woman.

```
select (valid e.name)[femaleMgr] as fname,
          (valid e.salary)[femaleMgr] as fsalary
from Employees as e,
          (select m.VT
          from valid e.belongsInDept as d,
               (valid d.hasManager)[valid(d)] as m
          where m.value.gender = 0) as femaleMgr
where e.gender = 1
```

Result:

```
Query type is bag<struct {{fname: list struct {value:
string, VT: period granularity Second calendar
Gregorian}, fsalary: list struct {value: integer, VT:
period granularity Month calendar Gregorian}}>
```

Query Q 4.9.4: Find the department, the current manager name, and the periods when a department manager earned more than one third of the departmental budget.

```
select d.name as dept, d.hasManager.name as manager,
          (select sal.VT
          from tstruct(mgr: valid d.hasManager,
               budg: valid d.budget) as mb,
               (valid mb.mgr.salary)[mb.VT] as sal
          where sal.value > (mb.budg/3)) as
               highPaidManagerPeriods
from Departments as d
```

Result:
`Query type is bag<struct {name: string, manager: string, highPaidManagerPeriods: bag<period granularity Month calendar Gregorian>}>`

Note: This query returns a tuple for each department; for some departments the field highlyPaidManagerPeriods may be an empty set.

Query Q 4.9.5: Find the name and the salary history of the employees in the periods they earned as much as their managers (distinct from themselves).

```
select (valid e.name)[bigSal] as name,
          (valid e.salary)[bigSal] as sal
from Employees as e,
          (select ms.VT
          from tstruct(dept: valid e.belongsInDept,
                  sal: valid e.salary) as ds,
               (valid ds.dept.hasManager)[ds.VT] as m,
               (valid m.value.salary)[m.VT] as ms
          where ms.value <= ds.sal and
               m.value.id != e.id) as bigSal
```

Result:
`Query type is bag<struct {name: list struct {value: string, VT: period granularity Second calendar Gregorian}, sal: list struct {value: integer, VT: period granularity Month calendar Gregorian}}>`

Query Q 4.9.6: When did one person earn a lower salary than another younger person, and who were those persons?

```
select e1.id as younger, e2.id as older, joinedSal.VT
from Employees as e1, Employees as e2,
          tstruct(sal1: valid e1.salary,
               sal2: valid e2.salary) as joinedSal
where e2.d_birth precedes e1.d_birth and
          joinedSal.sal2 < joinedSal.sal1
```

Result:
`Query type is bag<struct {younger: string, older: string, VT: period granularity Gregorian calendar Month}>`

Query Q 4.9.7: When and who had the same salary for the longest continuous period of time?

```
select e.id, valid(s) as when
from Employees as e, valid e.salary as s
where duration(valid(s)) =
          max(select duration(valid(s1))
               from Employees as e1,
                    valid e1.salary as s1)
```

Result: `Query type is bag<struct {id: string, when: period granularity Month calendar Gregorian}>`

Query Q 4.9.8: List DI's skill and salary histories during the time she was a manager.

```
select (valid e.hasSkills)[DIMgr] as skills,
          (valid e.salary)[DIMgr] as salaries
from Employees as e,
          (select valid(m)
          from Departments as d,
               valid d.hasManager as m
          where m.id = "DI") as DIMgr
where e.id = "DI"
```

Result: `Query type is bag<struct {skills: list struct {value: set<Skill>, VT: period granularity Day calendar Gregorian}, salaries: list struct {value: integer, VT: period granularity Month calendar Gregorian}}>`

Query 4.9.9: List the names and salary histories of all employees when they were managers and earned at least 36K.

```
select (valid e.name)[targetPeriod] as mname,
          (valid e.salary)[targetPeriod] as msal
from Employees as e,
          (select valid(m) intersect valid(s)
          from Departments as d,
               valid d.hasManager as m,
               valid e.salary as s
          where s >= 36000 and m.id = e.id and
               valid(m) overlaps valid(s))
                    as targetPeriod
```

Result:
`Query type is bag<struct {mname: list struct {value: string, VT: period granularity Second calendar Gregorian}, msal: list struct {value: integer, VT: period granularity Month calendar Gregorian}}>`

1.4.10 Class O3.S10 (Other, Element, Other)

Query Q 4.10.1: Find the budget history in the period from 1/1/82 to 12/31/84 and from 1/1/87 till now of all departments ED ever worked in.

```
select (valid d.budget)[period
               "[1982-01-01, 1985-01-01)"] as budg1,
          (valid d.budget)[period
               "[1987-01-01, now())"] as budg2
from Departments as d
where exists es in valid d.hasEmployee:
          (exists e in es: e.id = "ED")
```

Result: `Query type is bag<struct {budg1: list struct {value: integer, VT: period granularity Day calendar Gregorian}, budg2: list struct {value: integer, VT: period granularity Day calendar Gregorian}}>`

Query Q 4.10.2: Find the name and the budget history in 1984 and 1987 of the department being directed by Di.

```
select d.name, (valid d.budget)[period
                   "[1984-01-01, 1985-01-01)"]
                        as budg1,
           (valid d.budget)[period
                "1987-01-01, 1988-01-01)"] as budg2
from Departments as d
where d.hasManager.name = "Di"
```

Result:
Query type is bag<struct {name: string, budg1: list struct {value: integer, VT: period granularity Day calendar Gregorian}, budg2: list struct {value: integer, VT: period granularity Day calendar Gregorian}}>

Query Q 4.10.3: Find the name of the department where ED working at the beginning of both of the years 1986 and 1987 and the periods ED worked there.

```
select d.name,
           (select valid(d1)
           from  valid e.belongsInDept as d1
           where d.name = d1.name) as EDPeriod
from Employees as e,
           (select (valid e.belongsInDept)
                [instant "1986-01-01"]
           from set(1) as dummy
           where ((valid e.belongsInDept)
                     [instant "1986-01-01"]).name =
                ((valid e.belongsInDept)
                     [instant "1986-01-01"]).name) as d
where e.id = "ED"
```

Result: Query type is bag<struct {name: string, EDPeriod: bag<period granularity Day calendar Gregorian>}>

Query Q 4.10.4: Find the current name of the manager ED had on both 1984's Christmas and his 27th birthday, and the dates the manager started as a manager.

```
select mgr.name,
          (select begin(valid(m))
          from Departments as d,
               valid d.hasManager as m
          where m.id = mgr.id) as beginMgr
from Employees as e,
          (select (valid(valid e.belongsInDept)
                    [instant "1984-12-25"].hasManager)
                    [instant "1984-12-25"]
          from set(1) as dummy
          where (valid(valid e.belongsInDept)
                    [instant "1984-12-25"].hasManager)
                    [instant "1984-12-25"].id =
               (valid(valid e.belongsInDept)
                    [e.d_birth+interval "27" granularity
                         Year].hasManager)
                    [e.d_birth+interval "27" granularity
                         Year].id) as mgr
where e.id = "ED"
```

Result: `Query type is bag<struct {name: string, beginMgr: bag<instant granularity Day calendar Gregorian>}>`

Query Q 4.10.5: Find the department name, the then manager, the modification dates and the new values of the budget for every budget change that occurred in 1984, 1986 and 1988.

```
select d.name, (valid d.hasManager)[changeDate]
          as Manager,
          (valid d.budget)[changeDate] as Budget,
          changeDate
from Departments as d,
          (select begin(valid((valid d.budget)[i]))
          from 1..(count(valid d.budget)-1) as i
          where count(valid d.budget) >= 2 and
               (valid d.budget)[i] !=
               (valid d.budget)[i-1] and
               year(begin(valid((valid d.budget)[i])))
                    in set(1984, 1986, 1988))
               as changeDate
```

Result: `Query type is bag<struct {name: string, Manager: Employee, Budget: integer, changeDate: instant granularity Day calendar Gregorian}>`

2.0 Kalua and Robertson benchmark

2.1 The Event History database

As described in the Kalua and Robertson paper, the Event History database is used for social sciences. As described in the paper under question there is one “relation” - this paper too deals with temporal query languages based on relational databases - **Employment** that contains all the needed information. This relation can be one class in our model and can be defined as:

```
enum sex_enum {female, male};
interface Employement
(extent Employements,
key name)
{
          attribute String name;
          attribute Instant graularity day dob;
          attribute sex_enum sex;
          attribute String mstatus
               valid granularity day;
          attribute String occupation
               valid granularity day;
          attribute String residence
               valid granularity day;
}
```

2.1.1 Queries on the Event History Database

QA1: What were the names and marital statuses of managers during the periods when they were managers?

```
select e.name as name,
          flatten(select (valid e.mstatus)[valid(m)]
               from valid e.occupation as m
               where m = “Manager”) as mstatus
from Employements as e
where “Manager” in (valid e.occupation)
```

Result: `Query type is bag<struct {name: string, mstatus: set<struct {value: string, VT: period granularity Month calendar Gregorian}>}>`

QA2: What were the marital statuses of those who lived with their parents?

```
select e.name as name,
          flatten(select (valid e.mstatus)[valid(wp)]
               from valid e.residence as wp
               where wp = “with parents”) as mstatus
from Employements as e
where “with parents” in (valid e.residence)
```

Result: `Query type is bag<struct {name: string, mstatus: set<struct {value: string, VT: period granularity Month calendar Gregorian}>}>`

QA3: List the names and sex of all employees who have ever been divorced.

```
select e.name as name, e.sex as sex
from Employements as e
where "Divorced" in (valid e.mstatus)
```

Result: `Query type is bag<struct {name: string, sex: char}>`

QA4: List the names, marital status histories and residence histories of all managers before they became managers.

```
select e.name as name, (valid e.mstatus)[beforeMgr]
          as mstatus, (valid e.residence)[beforeMgr]
          as residence
from Employements as e,
          set(period(instant "beginning",
               min(select begin(valid(occup))
                    from valid e.occupation as occup
                    where occup = "Manager"))) as
                         beforeMgr
where "Manager" in (valid e.occupation)
```

Result: `Query type is bag<struct {name: string, mstatus: list struct {value: string, VT: period granularity Month calendar Gregorian}, residence: list struct {value: string, VT: period granularity Month calendar Gregorian}}>`

QA5: List all people who lived with their parents along with their jobs during those periods.

```
select e.name as name,
          flatten(select (valid
                         e.occupation)[valid(wp)]
               from valid e.residence as wp
               where wp = "With parents") as occupation
from Employements as e
where "With parents" in (valid e.residence)
```

Result: `Query type is bag<struct {name: string, occupation: set<struct {value: string, VT: period granularity Month calendar Gregorian}>}>`

QA6: List all people who never lived with their parents along with their sex and their residence histories.

```
select e.name as name, e.sex as sex,
          valid e.residence as residence
from Employements as e
where not("With parents" in (valid e.residence))
```

Result: `Query type is bag<struct {name: string, sex: char, residence: attribute string valid granularity Month calendar Gregorian}>`

QA7: What jobs did the widows do?

```
select e.name as name,
          flatten(select (valid e.occupation)[valid(w)]
               from valid e.mstatus as w
               where w = "widow") as occupation
from Employements as e
where "widow" in (valid e.mstatus)
```

Result: `Query type is bag<struct {name: string, occupation: set<struct {value: string, VT: period granularity Month calendar Gregorian}>}>`

QA8: List all people who ever worked as programmers and who lived with their parents at some time, along with their employment and residence histories.

```
select e.name as name, valid e.occupation as occupation,
          valid e.residence as residence
from Employements as e
where "Programmer" in (valid e.occupation) and
          "With parents" in (valid e.residence)
```

Result:

```
Query type is bag<struct {name: string, occupation:
attribute string valid granularity Month calendar
Gregorian, residence: attribute string valid granularity
Month calendar Gregorian}>
```

QA9: What jobs did the divorced men do?

```
select e.name as name,
          flatten(select (valid e.occupation)[valid(m)]
               from valid e.mstatus as m
               where m = "divorced") as occupation
from Employements as e
where "divorced" in (valid e.mstatus)
```

Result:

```
Query type is bag<struct {name: string, occupation:
set<struct {value: string, VT: period granularity Month
calendar Gregorian}>}>
```

QA10: List all people who lived with their parents while married or divorced, giving the times when this occurred.

```
select e.name as name,
          (struct(mstatus: mr.mstatus,
                   VT: mr.VT))
          as mstatus
from Employements as e,
          (select struct(mstatus: mrs.mstatus,
                          VT: mrs.VT)
          from tstruct(mstatus: valid e.mstatus,
                          residence: valid e.residence)
                          as mrs
          where mrs.mstatus in ("married", "divorced")
               and mrs.residence = "with parents")
               as mr
```

Result:

```
Query type is bag<struct {name: string, mstatus: struct
{mstatus: string, VT: period granularity Gregorian
calendar Month}}>
```

<u>Note</u> This query yields the more "reasonable" result:

Ken Wits		

The described result is returned by the following query:

```
select e.name as name,
          (select struct(mstatus: ms, VT: valid(ms)) as
mstatus_str
          from valid e.mstatus as ms
          where ms in ("divorced", "married")
               and exists r in valid e.residence:
                    (valid(r) overlaps valid(ms)
               and r = "with parents")) as mstatus,
          (select struct(residence: r1, VT: valid(r1))
               as residence_str
          from valid e.residence as r1
          where r1 = "with parents" and
               exists ms1 in valid e.mstatus:
                    (valid (r1) overlaps valid(ms1) and
                    ms1 in ("divorced", "married")))
           as res_str
from Employements as e
where exists mr in tstruct(mstatus: valid e.mstatus,
                          residence: valid e.residence):
          (mr.mstatus in ("married", "divorced")
          and mr.residence = "with parents")
```

Result:

```
Query type is bag<struct {name: string, mstatus:
bag<struct {mstatus_str: struct {mstatus: string, VT:
period granularity Month calendar Gregorian}}>, res_str:
```

```
bag<struct {residence_str: struct {residence: string,
VT: period granularity Month calendar Gregorian}}>}>
```

QA11: Retrieve all spells defined by Marital Status and Residence for Kim Bruce and Ken Witts.

```
select e.name as name,
          (select struct(spell: mr.VT,
                        mstatus: mr.mstatus,
                        residence: mr.residence)
          from tstruct(mstatus: valid e.mstatus,
                        residence: valid e.residence)
                    as mr) as spells
from Employements as e
where e.name in ("Kim Bruce", "Ken Witts")
```

Result:

```
Query type is bag<struct {name: string, spells:
bag<struct {spell: period granularity Gregorian calendar
Month, mstatus: string, residence: string}>}>
```

QA12: List the names and sex of all those who ever lived with their parents during the period of survey, along with their residence histories during the periods that they did not live with their parents.

```
select e.name as name, e.sex as sex,
          (select r as residence,valid(r) as VT
          from valid e.residence as r
          where r != "with parents") as residence
from Employements as e
where "with parents" in (valid e.residence)
```

Result:

```
Query type is bag<struct {name: string, sex: char,
residence: bag<struct {residence: string, VT: period
granularity Month calendar Gregorian}>}>
```

QA13: List all women who re-married within a year after a divorce, along with their marital histories.

```
select e.name as name, valid e.mstatus as mstatus
from Employements as e
where e.sex = 'F' and
          exists ms1 in valid e.mstatus:
               (ms1 = "divorce"
          and exists ms2 in (valid e.mstatus)
                    [valid(ms1) + interval "1"
                         granularity Year]:
               ms2.value = "married")
```

Result: `Query type is bag<struct {name: string, mstatus: attribute string valid granularity Month calendar Gregorian}>`

QA14: List all people who re-married after they were divorced, along with their marital histories.

```
select e.name as name, valid e.mstatus as mstatus
from Employements as e
where exists ms1 in valid e.mstatus:
          (ms1 = "divorced" and
          exists ms2 in valid e.mstatus:
               (ms2 = "married" and
               (valid(ms1)) precedes
                    begin(valid(ms2))))
```

Result: `Query type is bag<struct {name: string, mstatus: attribute string valid granularity Month calendar Gregorian}>`

QA15: List all people who changed jobs within a year after divorcing, along with their new jobs.

```
select e.name as name,
          (select struct(mstatus: ms,
                         VT: valid(ms)) as mstatus,
                    struct(occupation: occup,
                         VT: valid(occup)) as occupation
          from valid e.mstatus as ms,
               valid e.occupation as occup
          where ms = "divorced" and
               (begin(valid(occup)) - begin(valid(ms)))
                    < interval "1" granularity Year)
                    as shortChange
from Employements as e
where exists (select *
               from valid e.mstatus as ms1,
                    valid e.occupation as occup1
               where ms1 = "divorced" and
                    ((begin(valid(occup1)) -
                    begin(valid(ms1))) < interval
                    "1" granularity Year))
```

Result: `Query type is bag<struct {name: string, shortChange: bag<struct {mstatus: struct {mstatus: string, VT: period granularity Month calendar Gregorian}, occupation: struct {occupation: string, VT: period granularity Month calendar Gregorian}}>}>`

<u>Note:</u> The result schema for this query is such that it can accomodate for people with more than one divorces, followed closely by job changes

QA16: Calculate the number of divorces per year for the four years preceding January 1988.

```
select Year as Year, count(partition) as divorced
from Employements as e, valid e.mstatus as ms
where ms = "divorced" and
          year(begin(valid(ms))) in (1984 .. 1987)
group by year(begin(valid(ms))) as Year
```

Result: `Query type is bag<struct {Year: integer, divorced: integer}>`

Note: This query returns data only for the years in which at least one divorce was issued. Data for all the years are returned by the following query:

```
select year as year,
          count(select *
               from Employements as e,
                    valid e.mstatus as ms
               where year(begin(valid(ms))) = year)
                    as divorced
from 1984 .. 1987 as year
```

Result: `Query type is bag<struct {year: integer, divorced: integer}>`

QA17: Calculate the total work-months of employment per calendar year for men from 1982 through 1984.

```
select begin(year) as year,
          sum(select duration(x.VT intersect year)
               from partition as x
               where x.value != "None")
from (select e.occupation
          from Employements as e
          where e.sex = 'M')
group by (partition valid as interval "1"
          granularity Year) as
          year
having year(begin(year)) in (1982 .. 1984)
```

Result:

```
Error: Syntax error: syntax error|as
```

QA18: Calculate the total work-months of unemployment per calendar year from 1982 through 1984.

```
select year as year,
          sum(select duration(x.VT intersect year)
               from partition as x
               where x.value = "None")
from (select e.occupation from Employees as e)
group by partition valid as interval "1" granularity
year calendar
          as year
having year(begin(year)) in (1982 .. 1984)
```

Result:
```
Error: Syntax error: syntax error|as
```

2.2 The University Database

This database keeps personal and professional data for faculty members and the departments they work for. There following objects are defined:

```
enum sex {male, female};
interface FDepartment
(extent FDepartments
 key FDept)
{
          attribute String FDept;
          relationship Person Secretary valid
               inverse Person::isSecretary;
          relationship Person Head valid
               inverse Person::isHead;
}


interface Faculty
(extent Faculties
 key FName)
{
          attribute String FName;
          attribute sex FSex;
          attribute String MStatus valid;
          attribute short no_dependents valid;
          relationship Department inDept valid;
          attribute String Rank valid;
          attribute long Salary valid;
          state relationship Set<Publication>
               publications
               inverse Publication::written_by;
          relationship Department isSecretary valid
               inverse Department::Secretary;
          relationship Department::isHead valid
               inverse Department::Head
}

interface Publication
(extent Publications)
{
          attribute String journal;
          attribute String issue;
          attribute Instant granularity month
               effective_time;
          relationship Person written_by
               inverse Person::publications;
}
```

Three issues are worth noting about the University database:

1. The database schema consists of the three extents, rather than four; more specifically, the “Pers_Data” and “Professional” extents (relations) have been merged into a single extent named “Persons”
2. Although the Temporal Object Data Model (TODM) provides inverse relationships for valid time data and these relationships could be used in some queries, we prefered not to use them, so that the functionality of the language could be evaluated, rather than the richness of the data model. However, we have used the ability to use path expressions so as to access properties of the objects pointed to by relationships.

   We consider as “forward” direction of the relationship the traversal path from the extent using the object as a foreign key, while the other direction is the “inverse” one.

   For instance, the relationship

   Department::relationship Person Secretary valid

   is considered to be the “forward” direction, while the relationship

   Person::relationship Department isSecretary valid

   is considered to be the “inverse” direction
3. Some queries have been rendered more complex than they could actually be, while trying to match the result schema proposed in the paper for queries. For example, query B1 could have been formulated as follows:

   ```
   select p.fname, s as salary, valid(s) as when
   from Persons as p, valid p.salary as s
   where s >= 50000
   ```

   This formulation is quite simpler than the one used in the answers section but the resulting schema depicted in the following table is more “relational-like”, instead of the more “object-oriented” result suggested in the paper:

Fname	Salary	When
Bob Grass	51000	[1991-8, NOW)
Don Irsay	51000	[1989-08, 1991-01)
Don Irsay	56000	[1991-01, NOW)

2.2.1 Queries on the University Database

QB1: List all faculty who have ever earned a salary of at least 50000

```
select pd.fname as fname, pd.salary as salary
from (select p.FName as fname,
             (select s as Salary,
                  valid(s) as salary_period
             from valid p.Salary as s
             where s >= 50000) as salary
          from Faculties as p) as pd
where count(pd.salary) > 0
```

Result: `Query type is bag<struct {fname: string, salary: bag<struct {Salary: integer, salary_period: period granularity Month calendar Gregorian}>}>`

QB2: Which secretaries have worked in more than one department, where and when?

```
select secretary,
      (select x.ds.department as dept, x.ds.VT as VT
      from partition as x) as department
from (select d.FDept as department,
               s as secretary, valid(s) as VT
          from FDepartments as d,
               valid d.Secretary as s) as ds
group by secretary
having count(select distinct x1.ds.department
               from partition as x1) > 1
```

Result: `Query type is bag<struct {secretary: Faculty, department: bag<struct {dept: string, VT: period granularity Month calendar Gregorian}>}>`

QB3: What departments were headed by Dick Bond and Tim Young and who were their secretaries?

```
select head as head,
          (select x.hd.department as department,
               x.hd.VT as when,
               x.hd.secretary as secretary
          from partition as x) as HeadData
from (select h.FName as head,
          d.FDept as department,
          valid(h) as VT,
          (valid d.Secretary)[valid(h)] as secretary
          from FDepartments as d, valid d.Head as h
          where h.FName in ("Dick Bond", "Tim Young"))
          as hd
group by hd.head as head
```

Result: `Query type is bag<struct {head: string, HeadData: bag<struct {department: string, when: period granularity Month calendar Gregorian, secretary: list struct {value: Faculty, VT: period granularity Month calendar Gregorian}}>}>`

<u>Note</u>: The result schema of the query associates with each head name a set of departments and with each department a set of pairs <secretary, when>. This allows for persons who were in head of more than one departments and each department may have changed secretary.

QB4: Which departments have been headed by the same person during two or more distinct periods, who and when?

```
select x.fdept as fdept, x.headData as Head
from (select d.FDept as fdept,
          (select h.FName as head, valid(h) as when
          from valid d.Head as h
          where exists d1 in valid d.Head:
               (((d1.inDept).FDept = d.FDept) and
               (begin(valid(h)) precedes
                    begin(valid(d1))))) as headData
          from FDepartments as d)
          as x
where count(x.headData) > 0
```

Result: `Query type is bag<struct {fdept: string, Head: bag<struct {head: string, when: period granularity Month calendar Gregorian}>}>`

QB5: When, and of which department was Ann Byron head?

```
select h.FName as head, d.FDept as dept,
          valid(h) as when
from FDepartments as d, valid d.Head as h
where h.FName = "Ann Byron"
```

Result: `Query type is bag<struct {head: string, dept: string, when: period granularity Month calendar Gregorian}>`

Note: The result schema of this query is:

Head	Dept	When

i.e. the name “Ann Byron” is not associated with a set of pairs <dept, when>, but occurs in each result tuple. This can be easily fixed by using the “group by” clause and adding an exterior level query, just as in query B3.

QB6: Which secretaries have worked in the same department during two or more distinct periods? Where, when and under whom?

```
select s.FName as secretary, d.FDept as dept,
          (select h.FName as HeadName, valid(h) as when
          from valid d.Head as h
          where exists s1 in valid d.Secretary:
               (s.FName = s1.FName and
               valid(s1) overlaps valid(h))) as Head
from FDepartments as d,
          valid d.Secretary as s
where begin(valid(s)) = min(select begin(valid(s2))
                         from valid d.Secretary as s2
                         where s2.FName = s.FName)
          and begin(valid(s)) != max(select
                              begin(valid(s3))
                         from valid d.Secretary as s3
                         where s3.FName = s.FName)
```

Result: `Query type is bag<struct {secretary: string, dept: string, Head: bag<struct {HeadName: string, when: period granularity Month calendar Gregorian}>}>`

QB7: List all faculty who published in the same journal at least twice, along with the journal issues and publication dates.

```
select FName as fname, journal as journal,
          (select x.issue as issue,
               x.effective_time as effective_time
               from Publications as x)
                    as Publication_Data
from Publications as p
group by written_by as FName, journal as journal
having count(partition) > 1
```

Result: `Query type is bag<struct {fname: Faculty, journal: string, Publication_Data: bag<struct {issue: string, effective_time: instant granularity month calendar Gregorian}>}>`

QB8: When did the associate professors attain this rank?

```
select p.FName as assoc_name,
          begin(valid(r)) as date_promoted
from Faculties as p, valid p.FRank as r
where r = "Assoc."
```

Result: `Query type is bag<struct {assoc_name: string, date_promoted: instant granularity Month calendar Gregorian}>`

QB9: List all assistant professors who got promoted in the last three years, along with their department and salary immediately before their promotion.

```
select p.FName as promoted_assist,
          begin(valid(r)) as date_of_promotion,
          ((valid p.inDept)[begin(valid(r)) -
               interval “1” granularity Month]).FDept
                    as Dept,
          (valid p.Salary)[begin(valid(r)) -
               interval “1” granularity Month]
               as salary
from Faculties as p,
          valid p.FRank as r
where r = “Assoc.” and
          now() - interval “3” granularity Year
               precedes begin(valid(r))
```

Result: `Query type is bag<struct {promoted_assist: string, date_of_promotion: instant granularity Month calendar Gregorian, Dept: string, salary: integer}>`

QB10: List the names, departments and salaries of all associate professors at the time Bob Gross got promoted from associate to full.

```
select p.FName,
          (valid p.inDept)[promotionDate] as dept,
          (valid p.Salary)[promotionDate] as salary
from Faculties as p,
          (select begin(valid(r))
          from Faculties as p1,
               valid p1.FRank as r
          where p1.FName = “Bob Gross” and r = “Full”)
               as promotionDate
where (valid p.FRank)[promotionDate] = “Assoc.”
```

Result: `Query type is bag<struct {FName: string, dept: FDepartment, salary: integer}>`

QB11: List all faculty who were promoted after being in a department for less than 3 years, along with their department and rank while in that department.

```
select p.FName as fname,
          struct(deptName: d.FDept,
                    when: valid(d)) as fdept,
          (select r as rank, valid(r) as when
          from valid p.FRank as r
          where exists d1 in valid p.inDept:
               (valid(r) overlaps valid(d1) and
               d1.FDept = d.FDept)) as rank_info
from Faculties as p, valid p.inDept as d
where exists r1 in valid p.FRank:
          (r1 != first(valid p.FRank) and
          begin(valid(r1)) - begin(valid(d)) <
               interval “3” granularity Year)
```

Result: `Query type is bag<struct {fname: string, fdept: struct {deptName: string, when: period granularity Month calendar Gregorian}, rank_info: bag<struct {rank: string, when: period granularity Month calendar Gregorian}>}>`

QB12: Which faculty stayed at the associate rank for at least six years?

```
select p.FName as six_year_assoc, valid(r) as when
from Faculties as p, valid p.FRank as r
where r = “Assoc.” and
          duration(valid(r)) >= interval “6”
               granularity Year
```

Result: `Query type is bag<struct {six_year_assoc: string, when: period granularity Month calendar Gregorian}>`

QB13: Who have been full professors for the last four years? and what have been their department and salary histories during this period?

```
select p.FName as profsLast4Years,
          (valid p.inDept)[period(instant “now” -
           interval “4” granularity Year, instant
          “now”) granularity Month]
               as dept,
          (valid p.Salary)[period(instant “now” -
          interval “4” granularity Year, instant “now”)
          granularity Month]
               as salary
from Faculties as p
where exists r in valid p.FRank :
          (r = “Full” and valid(r) contains
               period(instant “now” - interval “4”
          granularity Year, instant “now”)
          granularity Month)
```

Result: `Query type is bag<struct {profsLast4Years: string, dept: list struct {value: FDepartment, VT: period granularity Month calendar Gregorian}, salary: list struct {value: integer, VT: period granularity Month calendar Gregorian}}>`

QB14: For all current full professors, list their marital status and salary histories since January 1990.

```
select p.FName as fname,
          (valid p.MStatus)[period(instant “1990-01”
               granularity Month, instant “now”)
               granularity Month] as mstatus,
          (valid p.Salary)[period(instant “1990-01”
               granularity Month, instant “now”)
               granularity Month] as salary
from Faculties as p
where p.FRank = “Full”
```

Result:
`Query type is bag<struct {fname: string, mstatus: list struct {value: string, VT: period granularity Month calendar Gregorian}, salary: list struct {value: integer, VT: period granularity Month calendar Gregorian}}>`

QB15: Who got promoted from assistant to full professor while at least one other faculty in the university remained at the associate rank? When did this happen and what departments were they in at the time?

```
select p.FName as fname,
          (select d.FDept as deptName,
               valid(d) as when
          from valid p.inDept as d
          where valid(d) contains promotionDate)
               as fdept,
          (select r as rank, valid(r) as when
          from valid p.FRank as r
          where r in set("Assist.", "Full")) as rank
from Faculties as p,
          (select begin(valid(r))
          from valid p.FRank as r
          where count(valid p.FRank) > 2 and
               r = "Full") as promotionDate
where exists(select *
          from Faculties as p1
          where exists r1 in valid p1.FRank:
               (r1 = "Assoc." and valid(r1) contains
                    promotionDate))
```

Result: `Query type is bag<struct {fname: string, fdept: bag<struct {deptName: string, when: period granularity Month calendar Gregorian}>, rank: bag<struct {rank: string, when: period granularity Month calendar Gregorian}>}>`

QB16: Which faculty lost their spouses while still employed by the university and how long did they stay widowed?

```
select p.FName as widowed_faculty,
          duration(valid(m)) as mourning_period
from Faculties as p, valid p.MStatus as m
where m = "Widowed" and
          exists d in valid p.inDept:
               valid(d) contains begin(valid(m))
```

Result: `Query type is bag<struct {widowed_faculty: string, mourning_period: interval granularity Month calendar Gregorian}>`

QB17: Which faculty have ever served as associate professors and ever earned at least 40000, along with their rank and salary histories?

```
select p.FName as fname,
          valid p.FRank as rank,
          valid p.Salary as salary
from Faculties as p
where "Assoc." in (valid p.FRank) and
          (exists s in valid p.Salary:
               s >= 40000)
```

Result: `Query type is bag<struct {fname: string, rank: attribute string valid granularity Month calendar Gregorian, salary: attribute integer valid granularity Month calendar Gregorian}>`

QB18: Which secretaries have worked under more than one head in the same department? Where, when and under whom?

```
select secretary as secretary, dept as dept,
          (select x1.dhs.head as head,
               x1.dhs.when as when
          from partition as x1) as head
from (select d.FDept as dept, hs.head.FName as head,
               hs.secretary.FName as secretary,
               hs.VT as when
          from FDepartments as d,
               tstruct(head: valid d.Head,
                    secretary: valid d.Secretary) as hs)
          as dhs
group by dhs.secretary as secretary, dhs.dept as dept
having count(select distinct x.dhs.head
          from partition as x) > 1
```

Result: `Query type is bag<struct {secretary: string, dept: string, head: bag<struct {head: string, when: period granularity Gregorian calendar Month}>}>`

QB19: Which secretaries have worked in the same department and under the same head during two or more distinct periods? Where, when and under whom?

```
select Secretary as secretary, dept as dept,
          (select x.hsd.headName as headName,
               x.hsd.when as when
          from partition as x) as Head
from (select d.FDept as dept, hs.Head.FName as headName,
               hs.Secretary.FName as Secretary,
               hs.VT as when
          from FDepartments as d,
               tstruct(Head: valid d.Head,
                    Secretary: valid d.Secretary) as hs)
               as hsd
group by hsd.headName as headName,
          hsd.Secretary as Secretary,
          hsd.dept as dept
having count(partition) > 1
```

Result: `Query type is bag<struct {secretary: string, dept: string, Head: bag<struct {headName: string, when: period granularity Gregorian calendar Month}>}>`

QB20: Which faculty have earned at least 40000 while serving as an associate professor, along with their salaries during that period?

```
select fs.fname as fname, fs.salary as salary
from (select p.FName as fname,
               (select rs.salary as salary,
                        rs.VT as when
               from tstruct(rank: valid p.FRank,
                            salary: valid p.Salary)
                    as rs
               where rs.rank = "Assoc." and
                    rs.salary > 40000) as salary
          from Faculties as p) as fs
where count(fs.salary) > 0
```

Result: `Query type is bag<struct {fname: string, salary: bag<struct {salary: integer, when: period granularity Gregorian calendar Month}>}>`

QB21: Which faculty have earned at least 40000 while serving as an associate professor, along with their rank and salary histories up to the time they ceased to be associate professors?

```
select p.FName as fname,
          (valid p.FRank)[period(instant “beginning”,
           ceaseDate) granularity Month] as rank,
          (valid p.Salary)[period(instant “beginning”,
          ceaseDate) granularity Month] as salary
from Faculties as p,
          (select end(valid(r))
          from valid p.FRank as r
          where r = “Assoc.”) as ceaseDate
where exists rs in tstruct(rank: valid p.FRank,
                          salary: valid p.Salary) :
          (rs.rank = “Assoc.” and
          rs.salary >= 40000)
```

Result: `Query type is bag<struct {fname: string, rank: list struct {value: string, VT: period granularity Month calendar Gregorian}, salary: list struct {value: integer, VT: period granularity Month calendar Gregorian}}>`

QB22: When and in which department did Cheri Best work under Mike Webb?

```
select d.FDept as cheri_mike_dept, h.VT as when
from Faculties as p,
          valid p.inDept as d,
          (valid d.Head)[valid(d)] as h
where p.FName = “Cheri Best” and
          h.value.FName = “Mike Webb”
```

Result: `Query type is bag<struct {cheri_mike_dept: string, when: period granularity Month calendar Gregorian}>`

QB23: Which departments have been headed by assistant professors at one point or another?

```
select da.dept as dept,
          da.assist_prof_head as assist_prof_head
from (select d.FDept as dept,
               (select h.FName as headName,
                    r.VT as when
               from valid d.Head as h,
                    (valid h.FRank)[valid(h)] as r
               where r.value = “Assoc.”)
          as assist_prof_head
          from FDepartments as d) as da
where count(da.assist_prof_head) > 0
```

Result: `Query type is bag<struct {dept: string, assist_prof_head: bag<struct {headName: string, when: period granularity Month calendar Gregorian}>}>`

QB24: What were the salaries of assistant professors with exactly one dependent?

```
select os.one_depend_assists as one_depend_assists,
          os.salary as salary
from (select p.FName as one_depend_assists,
               (select ds.salary as salary,
                    ds.VT as when
               from tstruct(no_dependents: valid
                              p.no_dependents,
                         salary: valid p.Salary) as ds
               where ds.no_dependents = 1) as salary
          from Faculties as p) as os
where count(os.salary) > 0
```

Result: `Query type is bag<struct {one_depend_assists: string, salary: bag<struct {salary: integer, when: period granularity Gregorian calendar Month}>}>`

QB25: List all faculty who got promoted while single, along with their department, rank and salary immediately before their promotion.

```
select p.FName as fname,
          date_of_promotion as date_of_promotion,
          (valid p.inDept)[date_of_promotion -
               interval "1" granularity Month] as dept,
          (valid p.FRank)[date_of_promotion -
               interval "1" granularity Month] as rank,
          (valid p.Salary)[date_of_promotion -
               interval "1" granularity Month] as salary
from Faculties as p,
          (select begin(valid(r))
          from valid p.FRank as r
          where
          (valid p.MStatus)[begin(valid(r))] =
                    "Single" and
               r != (valid p.FRank)[1])
                    as date_of_promotion
```

Result: `Query type is bag<struct {fname: string, date_of_promotion: instant granularity Month calendar Gregorian, dept: FDepartment, rank: string, salary: integer}>`

QB26: What publication submissions were made by faculty while serving as full professors?

```
select p.written_by.FName, p.journal, p.issue,
          p.effective_time
from Publications as p
where (valid p.written_by.FRank)
          [begin(p.effective_time)] = "Full"
```

Result: `Query type is bag<struct {FName: string, journal: string, issue: string, effective_time: instant granularity month calendar Gregorian}>`

QB27: What were the salaries of the other faculty in Philosophy when Randy Wells was head of department?

```
select fs.fname as fname, fs.salary as salary
from (select p.FName as fname,
               flatten(select (valid p.Salary)
                         [valid(d) intersect when]
                    from valid p.inDept as d,
                         (select valid(head)
                         from FDepartments as d1,
                              valid d1.Head as head
                         where d1.FDept = "Philosophy"
                              and Head.FName =
                              "Randy Wells")
                                   as when
                    where d.FDept = "Philosophy")
                         as salary
          from Faculties as p
          where p.FName != "Randy Wells") as fs
where count(fs.salary) > 0
```

Result: `Query type is bag<struct {fname: string, salary: set<struct {value: integer, VT: period granularity Month calendar Gregorian}>}>`

QB28: What is the publication record of current full professors?

```
select p.FName,
          (select pub
          from Publications as pub
          where pub.written_by.FName = p.FName)
               as publi
from Faculties as p
where p.FRank = "Full"
```

Result: `Query type is bag<struct {FName: string, publi: bag<Publication>}>`

QB29: Which female faculty changed their rank *and* marital status in the same year and what rank and marital status information supports this retrieval?

```
select mr.fname as fname, mr.mstatus as mstatus,
          mr.rank as rank
from (select p.FName as fname,
               (select m as mstatus, valid(m) as when
               from valid p.MStatus as m
               where (valid(m) !=
                    valid((valid p.MStatus)[1]))
                    and exists r1 in valid p.FRank:
                         ((r1 != first(valid p.FRank))
                          and
                         year(begin(valid(r1))) =
                              year(begin(valid(m))))))
                    as mstatus,
               (select r as rank, valid(r) as when
               from valid p.FRank as r
               where (valid(r) !=
                    valid((valid p.FRank)[1]))
                    and exists m1 in valid p.MStatus:
                         ((valid(m1) !=
                         valid((valid p.MStatus)[1]))
                         and year(begin(valid(m1))) =
                              year(begin(valid(r)))))
                    as rank
          from Faculties as p
          where p.FSex = 'F') as mr
where count(mr.mstatus) > 0
```

Result: `Query type is bag<struct {fname: string, mstatus: bag<struct {mstatus: string, when: period granularity Month calendar Gregorian}>, rank: bag<struct {rank: string, when: period granularity Month calendar Gregorian}>}>`

QB30: Which faculty got promoted while they had fewer than three publications? And what were their ranks and publications through the time of their third publication?

```
select p.FName as fname,
          (select pub.journal as journal,
               pub.effective_time as when
          from Publications as pub
          where pub.written_by.FName = p.FName and
               begin(pub.effective_time) precedes
                    third_pub_time) as journal,
          (valid p.FRank)[period(instant "beginning",
                    third_pub_time)
          granularity Month]
               as rank
from Faculties as p,
          (select begin(pubs[3])
               from set(select
                         begin(pub1.effective_time)
                         from Publications as pub1
                         where pub1.written_by.FName =
                              p.FName
                         order by begin
                              (pub1.effective_time))
                    as pubs
               where count(pubs) >= 3) as third_pub_time
where exists r in valid p.FRank:
          r != first(valid p.FRank) and
          count(select *
               from publications as pub2
               where pub2.written_by.FName = p.FName
                    and begin(pub2.effective_time) <
                         third_pub_time) < 3
```

Result: `Query type is bag<struct {fname: string, journal: bag<struct {journal: string, when: instant granularity month calendar Gregorian}>, rank: list struct {value: string, VT: period granularity Month calendar Gregorian}}>`

QB31: What have been the highest salaries paid by each department? Who earned these, when and at what rank?

```
select dept as dept,
          (select p.FName,
               (valid p.FRank)[x.ds.VT] as rank,
               x.ds.salary as salary,
               x.ds.VT as when
          from partition as x
          where x.ds.salary = max(select x1.ds.salary
                                   from partition as x1))
          as highest_salary
from Faculties as p,
          tstruct(dept: valid p.inDept,
                  salary: valid p.Salary) as ds
group by ds.dept as dept
```

Result: `Query type is bag<struct {dept: FDepartment, highest_salary: bag<struct {FName: string, rank: list struct {value: string, VT: period granularity Month calendar Gregorian}, salary: integer, when: period granularity Gregorian calendar Month}>}>`

Note: The result schema of the query associates with each dept a set of data describing the highest salaries. Moreover, within each record describing the highest salary, the rank is a set, allowing this for changes of the rank, while the person was receiving a constant salary.

QB32: What have been the highest salaries paid to associate professors by each department? Who earned these and when?

```
select dept as dept,
          (select x.p.FName as fname,
               x.rsd.salary as salary,
               x.rsd.VT as when
          from partition as x
          where x.salary = max(select x1.salary
                               from partition as x1))
               as assoc_high_salary
from Faculties as p, tstruct(salary: valid p.Salary,
                         rank: valid p.FRank,
                         dept: valid p.inDept) as rsd
where rs.rank = "Assoc."
group by rsd.dept
```

Result:
`*:abort`

QB33: Tabulate the total number of faculty publications by rank.

```
select rank as rank, count(partition) as no_pubs
from (select (valid p.FRank)[begin(pub.effective_time)]
          from Faculties as p,
               Publications as pub
          where pub.written_by.FName = p.FName)
          as pubRank
group by pubRank as rank
```

Result: `Query type is bag<struct {rank: string, no_pubs: integer}>`

QB34: Tabulate the total number of faculty publications by gender.

```
select sex as sex, count(partition) as no_pubs
from (select p.FSex
          from Faculties as p,
               Publications as pub
          where pub.written_by.FName = p.FName)
          as pubSex
group by pubSex as sex
```

Result: `Query type is bag<struct {sex: char, no_pubs: integer}>`

QB35: What was the composition of Computer Science faculty by rank as of January 1992?

```
select rank as rank, count(partition) as no_faculty
from Faculties as p
where ((valid p.inDept)[instant "1992-01" granularity
               Month]).FDept = "Computer Science"
group by (valid p.FRank)[instant "1992-01" granularity
               Month] as rank
```

Result: `Query type is bag<struct {rank: string, no_faculty: integer}>`

Note: This query will not return data for rank "Assoc."

2.3 The Clinical Database

Contains information about Manic Depressive patients. There is only one object used and it is defined as follows:

```
enum sex_enum {male, female};
```

```
interface manic_depr
(extent ManicDeprs
key patient)
{
          attribute String patient;
          attribute sex_enum sex;
          attribute String episode_phase valid;
          attribute String treatment valid;
}
```

2.3.1 Queries on the Clinical Database

QC1: List all bipolar patients who experienced some form of mania as well as depression within a month of each other, along with any supporting information.

```
select p.patient as patient, p.episode as episode
from (select md.patient as patient,
               (select e as episode, valid(e) as when
               from valid md.episode_phase as e
               where (e like "*Mania" or
                         e like "*Depression") and
                    exists e1 in valid md.episode_phase:
                         ((e1 like "*Mania" or
                              e1 like "*Depression")
                         and e[length(e)-5: length(e)]
                               !=
                              e1[length(e1)-
                                   5:length(e1)]
                          and
                         abs(begin(valid(e1)) -
                              end(valid(e))) < interval
                              "1" granularity Month or
                         abs(begin(valid(e)) -
                              end(valid(e1))) < interval
                              "1" granularity Month))
                    as episode
          from ManicDeprs as md) as p
where count(p.episode) > 0
```

Result: `Query type is bag<struct {patient: string, episode: bag<struct {episode: string, when: period granularity Second calendar Gregorian}>}>`

QC2: List all patients who had the same episode of mania or depression recur within a month after the last one.

```
select p.patient as patient, p.episode_phase as episode
from (select md.patient as patient,
               (select e as episode_phase,
               valid(e) as when
               from valid md.episode_phase as e
               where exists e1 in valid
                    md.episode_phase:
                    (e1 = e and
                    begin(valid(e1)) != begin(valid(e))
                    and (abs(begin(valid(e1)) -
                         (end(valid(e))))) <
                         interval "1"
                         granularity Month
                    or abs(begin(valid(e)) -
                         (end(valid(e1)))) <
                         interval "1"
                         granularity Month)) as
                    episode_phase
          from ManicDeprs as md) as p
where count(p.episode_phase) > 0
```

Result: `Query type is bag<struct {patient: string, episode: bag<struct {episode_phase: string, when: period granularity Second calendar Gregorian}>}>`

QC3: For those patients who experienced severe mania, what treatment were they on at the time? And when did they experience this?

```
select md.patient as patient, valid(e) as when,
          (valid md.treatment)[begin(valid(e))]
               as treatment
from ManicDeprs as md, valid md.episode_phase as e
where e = "Severe Mania"
```

Result: `Query type is bag<struct {patient: string, when: period granularity Second calendar Gregorian, treatment: string}>`

QC4: Who experienced *new* bouts of mania or depression while on lithium treatment?

```
select md.patient as p_id,
          struct(episode: e, when: valid(e))
               as new_episode,
          struct(treatment: t, when: valid(t))
               as treatment
from ManicDeprs as md,
          valid md.episode_phase as e,
          valid md.treatment as t
where (valid(e) != valid((valid md.episode_phase)[1]))
           and
          (e != "Normal") and (t = "Lithium") and
          valid(t) contains begin(valid(e))
```

Result: `Query type is bag<struct {p_id: string, new_episode: struct {episode: string, when: period granularity Second calendar Gregorian}, treatment: struct {treatment: string, when: period granularity Second calendar Gregorian}}>`

QC5: For each patient, tabulate the total number of bouts of mania and the total number of bouts of depression.

```
select md.patient as patient,
          count(select e from valid md.episode_phase
                   as e
                   where e like "*Mania")
               as no_bouts_manic,
          count(select e1 from valid md.episode_phase
                   as e1
                   where e1 like "*Depression")
               as no_bouts_depressive
from ManicDeprs as md
```

Result: `Query type is bag<struct {patient: string, no_bouts_manic: integer, no_bouts_depressive: integer}>`

QC6: Tabulate the combined number of *new* bouts of mania or depression by treatment type.

```
select treatment,
          count(partition) as no_new_bouts
from (select t
          from ManicDeprs as md,
               valid md.episode_phase as e,
               valid md.treatment as t
          where (valid(e) !=
               valid((valid md.episode_phase)[1]))
               and (e != "Normal") and
               (valid(t) contains begin(valid(e))))
          as new_bouts
group by new_bouts as treatment
```

Result: `Query type is bag<struct {treatment: string, no_new_bouts: integer}>`

QC7: Tabulate the combined total number of days when the patient had bouts of mania or depression of any degree by treatment type.

```
select treatment as treatment,
          sum(select duration(x.et.VT)
               from partition as x) as days_md_bouts
from ManicDeprs as md,
          tstruct(episode: valid md.episode_phase,
                    treatment: valid md.treatment) as et
where et.episode != “Normal”
group by et.treatment as treatment
```

Result: `Query type is bag<struct {treatment: string, days_md_bouts: interval granularity Gregorian calendar Second}>`

QC8: What was the total number of days that each patient was “normal” during the six-month period?

```
select md.patient as patient,
          sum(select duration(e.VT)
               from (valid md.episode_phase)
               [period(instant “now” -
                    interval “6” granularity Month,
                    instant “now”)
                    granularity Month] as e
               where e.value = “Normal”)
          as total_days_normal
from ManicDeprs as md
```

Result: `Query type is bag<struct {patient: string, total_days_normal: interval granularity Second calendar Gregorian}>`

QC9: For each patient, tabulate the total number of days when they were manic and the total number of days when they were depressive during the six-month period.

```
select md.patient as patient,
          sum(select duration(e.VT)
               from (valid md.episode_phase)[period(
                    instant “now” -
                    interval “6” granularity Month,
                    instant “now”)
                    granularity Month] as e
               where e.value like “*Mania”)
               as tot_day_manic,
          sum(select duration(e1.VT)
               from (valid md.episode_phase)[period(
                    instant “now” -
                    interval “6” granularity Month,
                    instant “now”)
                    granularity Month] as e1
               where e1.value like “*Depression”)
                    as tot_days_depressive
from ManicDeprs as md
```

Result: `Query type is bag<struct {patient: string, tot_day_manic: interval granularity Second calendar Gregorian, tot_days_depressive: interval granularity Second calendar Gregorian}>`

QC10: What were the longest and shortest periods of “normality” in days for each patient?

```
select md.patient as patient,
          max(select duration(valid(e))
               from valid md.episode_phase as e
               where e = “Normal”) as max_normal_period,
          min(select duration(valid(e1))
               from valid md.episode_phase as e1
               where e1 = “Normal”) as min_normal_period
from ManicDeprs as md
```

Result: `Query type is bag<struct {patient: string, max_normal_period: interval granularity Second calendar Gregorian, min_normal_period: interval granularity Second calendar Gregorian}>`

3.0 Glaxo Queries

GLAXO 1. In the visit1, the inclusion criteria 8 of the VerificationOfEligibility container cannot be true if the number of corticoid uses is > 10 or < 2 in IllnessHIstory container, and reversely.

```
select p.name
from patients as p,
      (valid (p.on.illnessHistory)) as ill,
      (valid p.on.verificationOfEligibility) as elig
where ((ill.value[0]).corticoidsUses4 <= 10 and
       (ill.value[0]).corticoidsUses4 >= 2 and
       ((elig.value[0]).oralCorticoid3 = 0 or
        (elig.value[0]).oralCorticoid3 = 1))
```

Glaxo 2. If an aggravation is detected in the SummaryOfVisit of visit 2, check that the date of TrialEnd is between visit1 and visit2, and that at least one concomitant treatment has started between visit1 and visit2 + 1 day

```
select p.name
from patients as p, p.on.visiSummary as sum,
          p.on.physicalExam as phy,
          p.on.trialEnd as tend,
          p.on.concomitantTreatment as treat
where ((valid sum.value)[0]).criteriaAggravation = 1 and
          period(valid(valid phy.value)[0]),
               valid(valid phy.value)[1]))
          contains tend.value.dateOfTrialEnd0
          and exists ct in treat:
          period(valid(valid phy.value)[0]),
               valid(valid phy.value)[1]) contains
          ct.value.beginingDate5
```

Glaxo 3. What is the time to reach an increase of 10% of VEMS, with regards to J0?

```
select p.name, period(valid(valid
          p.on.functionalRespiratoryTest.value)[0]),
          min(valid(frt)))
from patients as p,
          valid p.on.functionalRespiratoryTest.value
               as frt
where frt.theoreticalVems >= 1.1 * ((valid
          p.on.functionalRespiratoryTest.value)[0].
          theoreticalVems
```

Glaxo 4. What is the average value of MaximumExpiratoryDebit (am and pm) week by week

```
select p.name, (select n.amMaximumExpiratoryDebit
               from weekly.partition as n) as amAvg,
               (select n.pmMaximumExpiratoryDebit
               from weekly.partition as n) as pmAvg
from patients as p,
          (valid p.periodicIndividualNotebook.value)
          (partition valid as interval "7" day)
               as weekly
where p.periodicIndividualNotebook != nil
```

4.0 Delta Queries

Contains the following interfaces

```
interface ProductGroup
(extent ProductGroups)
{
          attribute String attName;
          relationship ProductGroup relPartOf
               inverse ProductGroup::relComposedOf;
          relationship Set<ProductGroup> relComposedOf
               inverse ProductGroup::relPartOf;
          relationship Set<Product> relHasProducts
               inverse Product::relBelongs;
};
interface Product
(extent Products key attProductCode)
{
          attribute String attProductCode;
          attribute String attName;
          attribute String attType;
          attribute String attStatus valid event
               granularity Day;
          attribute Short attDuration valid state
               granularity Day;
          attribute Float attPrice valid state
               granularity Day;
          relationship ProductGroup relBelongs
               inverse ProductGroup::relHasProducts;
};
```

4.1 Queries

Delta 1. Show the name, the price and the period during which this price was valid, for each product, ordered by name and group

```
select p.attName as pName, pr as price,
          valid(pr) as time
from Products p, valid p.attPrice as pr
order by p.attName, p.relBelongs.attName
```

Delta 2. Display the name, the price and the period during which this price was valid, for each product whose price changed the last X months

```
select p.attName as pName, pr as price,
          valid(pr) as time
from Products p, valid p.attPrice as pr
where begin(valid(pr)) >
          now() - interval($1) granularity Month
```

Delta 3. Display the name, the price and the period during which this price was valid, for each product whose price changed during the last X months Y times

```
select p.attName as pName, pr as price,
          valid(pr) as time
from Products p, valid p.attPrice as pr
where count(select *
          from valid p.attPrice as pr1
          where valid(pr1) >
          now() - interval($1) granularity Month) > $2
```