Writing Queries (Exercises)

Lionel Parreaux

(Adapted from slides by **Amir Shaikhha** and **Daniel Lupei**)

A little bit of logic

- De Morgan's Laws

  $\neg(p(a) \wedge p(b)) \Leftrightarrow \neg p(a) \vee \neg p(b)$

  $\neg(p(a) \vee p(b)) \Leftrightarrow \neg p(a) \wedge \neg p(b)$

- Generalized De Morgan's Laws

  $\neg(p(a_1) \wedge \ldots \wedge p(a_n)) \Leftrightarrow \neg p(a_1) \vee \ldots \vee \neg p(a_n)$

  $\neg(p(a_1) \vee \ldots \vee p(a_n)) \Leftrightarrow \neg p(a_1) \wedge \ldots \wedge \neg p(a_n)$,

  where $\mathbf{A}=\{a_1,\ldots,a_n\}$

A little bit of logic

- Since:

  $p(a_1)^\wedge\ldots^\wedge p(a_n) \Leftrightarrow \forall x\in \mathbf{A}\ (p(x))$

  $p(a_1)^\vee\ldots^\vee p(a_n) \Leftrightarrow \exists x\in \mathbf{A}\ (p(x))$,

- Generalized De Morgan's Laws become:

  $\neg(\forall x\in \mathbf{A}\ (p(x))) \Leftrightarrow \exists x\in \mathbf{A}\ (\neg p(x))$

  $\neg(\exists x\in \mathbf{A}\ (p(x))) \Leftrightarrow \forall x\in \mathbf{A}\ (\neg p(x))$

- and also:

  $\forall x\in \mathbf{A}\ (p(x)) \Leftrightarrow \neg(\ \exists x\in \mathbf{A}\ (\neg p(x))\ )$

  $\exists x\in \mathbf{A}\ (p(x)) \Leftrightarrow \neg(\ \forall x\in \mathbf{A}\ (\neg p(x))\ )$

A little bit of logic

- Set inclusion: $\mathbf{A} \subseteq \mathbf{B}$

  $\forall x \in \mathbf{A}\ (x \in \mathbf{B}) \Leftrightarrow \neg(\exists x \in \mathbf{A}\ (x \notin \mathbf{B})) \Leftrightarrow \neg(\exists x \in \mathbf{A}\ \neg(\exists y \in \mathbf{B}\ (x=y)))$

- Set equality: $\mathbf{A=B} \Leftrightarrow \mathbf{A} \subseteq \mathbf{B} \wedge \mathbf{B} \subseteq \mathbf{A}$

- Implication

  $p(a) \rightarrow p(b) \Leftrightarrow \neg p(a) \vee p(b)$

  $\neg\ (p(a) \rightarrow p(b)) \Leftrightarrow p(a) \wedge \neg p(b)$

Relational Calculus

- Tuple relational calculus
  - Variable *x* is associated to an entire tuple of relation R: $x \in R$
  - Access field *val*: *x.val*
  - Leads to SQL
- Domain relational calculus
  - Variable *val* is associated to a field of a tuple of relation R: $\langle val \rangle \in R$
  - Access field *val*: *val*
  - Leads to Datalog
- Read more in Ramakrishnan&Gehrke: 4.3 (pg.116)

A simple example

- Consider a graph given by a DB with schema:
  - Vertex(v)
  - Edge(v_1, v_2, size)
- Find all triplets of vertices that satisfy the triangle inequalities.
- IsTriangle($dist_{xy}$, $dist_{yz}$, $dist_{zx}$):

  $(dist_{xy} \leq dist_{yz} + dist_{zx})$ ^ $(dist_{yz} \leq dist_{zx} + dist_{xy})$ ^

  $(dist_{zx} \leq dist_{xy} + dist_{yz})$

A simple example (1): Quantified English

Find all triplets of vertices that satisfy the triangle inequalities.

Find all x is a Vertex, y is a Vertex, z is a vertex,
e_{xy} is an Edge between x and y,
e_{yz} is an Edge between y and z,
e_{zx} is an Edge between z and x,
such that IsTriangle(e_{xy}.size, e_{yz}.size, e_{zx}.size)

A simple example (1): Tuple relational calculus

Find all x is a Vertex, y is a Vertex, z is a vertex,
e_{xy} is an Edge between x and y,
e_{yz} is an Edge between y and z,
e_{zx} is an Edge between z and x,
such that IsTriangle(e_{xy}.size, e_{yz}.size, e_{zx}.size).

$\{x,y,z \mid x \in \text{Vertex} \wedge y \in \text{Vertex} \wedge z \in \text{Vertex} \wedge$
$e_{xy} \in \text{Edge} \wedge e_{yz} \in \text{Edge} \wedge e_{zx} \in \text{Edge} \wedge$
$(e_{xy}.v_1 = x.v) \wedge (e_{xy}.v_2 = y.v) \wedge$
$(e_{yz}.v_1 = y.v) \wedge (e_{yz}.v_2 = z.v) \wedge$
$(e_{zx}.v_1 = z.v) \wedge (e_{zx}.v_2 = x.v) \wedge$
$\text{IsTriangle}(e_{xy}.\text{size}, e_{yz}.\text{size}, e_{zx}.\text{size}) \}$

A simple example (1): SQL

$$\{x,y,z \mid x\in Vertex \wedge y\in Vertex \wedge z\in Vertex \wedge$$
$$e_{xy}\in Edge \wedge e_{yz}\in Edge \wedge e_{zx}\in Edge \wedge$$
$$(e_{xy}.v_1 = x.v) \wedge (e_{xy}.v_2 = y.v) \wedge$$
$$(e_{yz}.v_1 = y.v) \wedge (e_{yz}.v_2 = z.v) \wedge$$
$$(e_{zx}.v_1 = z.v) \wedge (e_{zx}.v_2 = x.v) \wedge$$
$$IsTriangle(e_{xy}.size,\ e_{yz}.size,\ e_{zx}.size)\ \}$$

SELECT x.v, y.v, z.v
FROM Vertex x, Vertex y, Vertex z, Edge e_{xy}, Edge e_{yz}, Edge e_{zx}
WHERE ($e_{xy}.v_1$= x.v) AND ($e_{xy}.v_2$= y.v) AND
($e_{yz}.v_1$= y.v) AND ($e_{yz}.v_2$= z.v) AND
($e_{zx}.v_1$= z.v) AND ($e_{zx}.v_2$= x.v) AND
IsTriangle(e_{xy}.size, e_{yz}.size, e_{zx}.size)

A simple example (3)

- Consider a graph given by a DB with schema:
    - Vertex(v)
    - Edge(v_1, v_2, size)
- **Check that all** connected triplets of vertices satisfy the triangle inequalities.

A simple example (3)

$\{ \forall x \in Vertex\ \forall y \in Vertex\ \forall z \in Vertex$
$\forall e_{xy} \in Edge\ \forall e_{yz} \in Edge\ \forall e_{zx} \in Edge$
$((e_{xy}.v_1 = x.v \wedge e_{xy}.v_2 = y.v \wedge$
$e_{yz}.v_1 = y.v \wedge e_{yz}.v_2 = z.v \wedge$
$e_{zx}.v_1 = z.v \wedge e_{zx}.v_2 = x.v) \rightarrow$
$IsTriangle(e_{xy}.size, e_{yz}.size, e_{zx}.size)) \}$

$\{\neg(\exists x \in Vertex\ \exists y \in Vertex\ \exists z \in Vertex$
$\exists e_{xy} \in Edge\ \exists e_{yz} \in Edge\ \exists e_{zx} \in Edge$
$\neg((e_{xy}.v_1 = x.v \wedge e_{xy}.v_2 = y.v \wedge$
$e_{yz}.v_1 = y.v \wedge e_{yz}.v_2 = z.v \wedge$
$e_{zx}.v_1 = z.v \wedge e_{zx}.v_2 = x.v) \rightarrow$
$IsTriangle(e_{xy}.size, e_{yz}.size, e_{zx}.size)) \}$

A simple example (3)

$\{\neg(\exists x \in Vertex\ \exists y \in Vertex\ \exists z \in Vertex$
$\quad \exists e_{xy} \in Edge\ \exists e_{yz} \in Edge\ \exists e_{zx} \in Edge$
$\quad\quad \neg((e_{xy}.v_1 = x.v \wedge e_{xy}.v_2 = y.v \wedge$
$\quad\quad\quad e_{yz}.v_1 = y.v \wedge e_{yz}.v_2 = z.v \wedge$
$\quad\quad\quad e_{zx}.v_1 = z.v \wedge e_{zx}.v_2 = x.v) \rightarrow$
$\quad\quad\quad IsTriangle(e_{xy}.size, e_{yz}.size, e_{zx}.size)\ )\ \}$

$\{\neg(\exists x \in Vertex\ \exists y \in Vertex\ \exists z \in Vertex$
$\quad \exists e_{xy} \in Edge\ \exists e_{yz} \in Edge\ \exists e_{zx} \in Edge$
$\quad\quad ((e_{xy}.v_1 = x.v \wedge e_{xy}.v_2 = y.v \wedge$
$\quad\quad\quad e_{yz}.v_1 = y.v \wedge e_{yz}.v_2 = z.v \wedge$
$\quad\quad\quad e_{zx}.v_1 = z.v \wedge e_{zx}.v_2 = x.v) \wedge$
$\quad\quad\quad \neg IsTriangle(e_{xy}.size, e_{yz}.size, e_{zx}.size)\ )\ \}$

A simple example (3)

$$\{\neg(\exists x{\in}\text{Vertex } \exists y{\in}\text{Vertex } \exists z{\in}\text{Vertex}$$
$$\exists e_{xy}{\in}\text{Edge } \exists e_{yz}{\in}\text{Edge } \exists e_{zx}{\in}\text{Edge}$$
$$((e_{xy}.v_1 = x.v \wedge e_{xy}.v_2 = y.v \wedge$$
$$e_{yz}.v_1 = y.v \wedge e_{yz}.v_2 = z.v \wedge$$
$$e_{zx}.v_1 = z.v \wedge e_{zx}.v_2 = x.v) \wedge$$
$$\neg\text{IsTriangle}(e_{xy}.\text{size}, e_{yz}.\text{size}, e_{zx}.\text{size}))\}$$

SELECT 1 FROM dummy
WHERE NOT EXISTS (
SELECT * FROM Vertex x, Vertex y, Vertex z,
Edge e_{xy}, Edge e_{yz}, Edge e_{zx}
WHERE ($e_{xy}.v_1$= x.v) AND ($e_{xy}.v_2$= y.v) AND
($e_{yz}.v_1$= y.v) AND ($e_{yz}.v_2$= z.v) AND
($e_{zx}.v_1$= z.v) AND ($e_{zx}.v_2$= x.v) AND
NOT IsTriangle(e_{xy}.size, e_{yz}.size, e_{zx}.size))

Exercise 1

- Find the students whose grades have only improved in time.

- Schema:
  - Student(sid)
  - Grades(sid, val, date)

Exercise 1 – Quantified English

Find the students whose grades have only improved in time.

Find all (x is Student) such that

for all (g_1 is a grade of x)(g_2 is a grade of x)

g_1.date > g_2.date implies g_1.val >g_2.val

Exercise 1 – Quantified English

Find all (x is Student) such that

for all (g_1 is a grade of x)(g_2 is a grade of x)

g_1.date > g_2.date implies g_1.val >g_2.val

Find all (x is Student) such that

there is no (g_1 is a grade of x)(g_2 is a grade of x)

not(g_1.date > g_2.date implies g_1.val >g_2.val **)**

Exercise 1 – Quantified English

Find all (x is Student) such that

there is no (g_1 is a grade of x)(g_2 is a grade of x)

not(g_1.date > g_2.date **implies** g_1.val >g_2.val **)**

Find all (x is Student) such that

there is no (g_1 is a grade of x)(g_2 is a grade of x)

s.t. g_1.date > g_2.date **and** g_1.val ≤g_2.val

Exercise 1 – Relational Claculus

Find all (x is Student) such that

there is no (g_1 is a grade of x)(g_2 is a grade of x)

s.t. g_1.date **≤** g_2.date **and** g_1.val >g_2.val

$\{ x \mid x \in Student \wedge$

$\neg(\exists g_1 \in Grades\ \exists g_2 \in Grades$

$(g_1.sid = x.sid) \wedge (g_2.sid = x.sid) \wedge$

$(g_1.date > g_2.date) \wedge (g_1.val \leq g_2.val))\ \}$

Exercise 1 – SQL

$\{ x \mid x \in Student \wedge$

$\neg(\exists g_1 \in Grades\ \exists g_2 \in Grades$

$(g_1.sid = x.sid) \wedge (g_2.sid = x.sid) \wedge$

$(g_1.date > g_2.date) \wedge (g_1.val \leq g_2.val)) \}$

SELECT x.sid FROM Student x

WHERE NOT EXISTS

(SELECT * FROM Grades g_1, Grades g_2

WHERE g_1.sid = x.sid AND g_2.sid = x.sid AND

g_1.date > g_2.date AND g_1.val ≤g_2.val)

Exercise 2

- Find the students who only take courses that are taken by all students.

- Schema:
  - Student(sid),
  - Course(cid),
  - Taken(cid, sid)

Exercise 2 – Quantified English

Find the students
who only take courses
that are taken by all students.

Find all (x is a Student) such that
for all (Courses c taken by x) we have that
c is taken by all students.

Exercise 2 – Quantified English

Find all (x is a Student) such that
for all (Courses c taken by x) we have that
c is taken by all students.

Find all (x is a Student) such that
for all (Courses c taken by x)**(Students y)** we have
c is taken by y.

Exercise 2 – Quantified English

Find all (x is a Student) such that
for all (Courses c taken by x)(Students y) we have
c is taken by y.

Find all (x is a Student) such that
for all (Courses c taken by x)(Students y)
there is a (Course c taken by y).

Exercise 2 – Quantified English

Find all (x is a Student) such that
for all (Courses c taken by x)(Students y)
there is a (Course c taken by y).

Find all (x is a Student) such that
there is no (Course c taken by x)(Student y)
for which there is **no** (Course c taken by y) **)**.

Exercise 2 – Relational Calculus

Find all (x is a Student) such that
there is no (Course c taken by x)(Student y)
for which there is no (Course c taken by y)).

$$\{x \mid x \in Student \wedge \neg(\exists t \in Taken\ \exists y \in Student\ (t.sid = x.sid) \wedge (\neg(\exists t' \in Taken\ (t'.sid = y.sid \wedge t'.cid = t.cid)))) \}$$

Exercise 2 – SQL

$\{x \mid x \in Student \wedge$

$\qquad \neg(\exists t \in Taken\ \exists y \in Student\ (t.sid = x.sid) \wedge$

$\qquad\qquad (\neg(\exists t' \in Taken\ (t'.sid = y.sid \wedge$

$\qquad\qquad\qquad t'.cid = t.cid\ )\ )\ )\ )\ \}$

```
SELECT x.sid FROM Student x
WHERE NOT EXISTS
   (SELECT * FROM Taken t, Student y
    WHERE t.sid = x.sid
    AND NOT EXISTS
        (SELECT * FROM Taken t'
          WHERE t'.sid = y.sid AND t'.cid = t.cid))
```

Building complex queries from smaller parts

- Express the query as a sequence of steps/views.
- Define the views.
- SQL, calculus, relational algebra are compositional: If the solution is not to use views, compose them into a single query.

Exercise 3

- Given schema:
    - A(x)
    - B(y)
    - F(x,y)
- Write a query that checks whether F represents a bijective function from A to B.

Exercise 3 – Quantified English

Q:= F represents a bijective function from A to B.

:= Q_A and Q_B and Q_F

Q_A := (A = {f.x | for all f in F})

Q_B := (B = {f.y | for all f in F})

Q_F := There are no (f,f', two mappings in F) such that (f.x = f'.x and f.y != f'.y) or

(f.x != f'.x and f.y = f'.y).

Exercise 3 – Q_A, Q_B

$Q_A := (A = \{f.x \mid \text{for all } f \text{ in } F\})$

$\quad := (A = \{f.x \mid \forall f \in F\})$

$\quad := (A \subseteq \{f.x \mid \forall f \in F\}) \wedge (\{f.x \mid \forall f \in F\} \subseteq A)$

$\quad := \neg(\exists a \in A\ \neg(\exists f \in F\ (a.x = f.x))) \wedge$

$\quad \neg(\exists f \in F\ \ \neg(\exists a \in A\ (f.x = a.x)))$

$Q_B := \neg(\exists b \in B\ \neg(\exists f \in F\ (b.y = f.y))) \wedge$

$\quad \neg(\exists f \in F\ \ \neg(\exists b \in B\ (f.y = b.y)))$

Exercise 3 – $Q_A \wedge Q_B$

$$
\begin{aligned}
Q_A \wedge Q_B := {} & \neg(\exists a \in A\ \neg(\exists f \in F\ (a.x=f.x))) \wedge \\
& \neg(\exists f \in F\ \ \neg(\exists a \in A\ (f.x=a.x))) \wedge \\
& \neg(\exists b \in B\ \neg(\exists f \in F\ (b.y=f.y))) \wedge \\
& \neg(\exists f \in F\ \ \neg(\exists b \in B\ (f.y=b.y))) \\
:= {} & \neg(\exists a \in A\ \neg(\exists f \in F\ (a.x=f.x))) \wedge \\
& \neg(\exists b \in B\ \neg(\exists f \in F\ (b.y=f.y))) \wedge \\
& \neg(\exists f \in F\ \ \neg(\exists a \in A\ (f.x=a.x)) \vee \\
& \qquad \neg(\exists b \in B\ (f.y=b.y)))
\end{aligned}
$$

Exercise 3 – Q_F

Q_F := There are no (f,f', two mappings in F) such that

(f.x = f'.x and f.y != f'.y) or

(f.x != f'.x and f.y = f'.y).

$:= \neg(\exists f \in F\ \exists f' \in F\ (f.x = f'.x \wedge f.y \ne f'.y) \vee$

$(f.x \ne f'.x \wedge f.y = f'.y))$

Exercise 3 – Relational Calculus

$$Q := \{ \neg(\exists a \in A\ \neg(\exists f \in F\ (a.x=f.x))) \wedge$$
$$\neg(\exists b \in B\ \neg(\exists f \in F\ (b.y=f.y))) \wedge$$
$$\neg(\exists f \in F\ \ \neg(\exists a \in A\ (f.x=a.x)) \vee \neg(\exists b \in B\ (f.y=b.y))) \wedge$$
$$\neg(\exists f \in F\ \exists f' \in F\ (f.x = f'.x \wedge f.y \mathrel{!=} f'.y) \vee$$
$$(f.x \mathrel{!=} f'.x \wedge f.y = f'.y)) \}$$

$$Q := \{ \neg(\exists a \in A\ \neg(\exists f \in F\ (a.x=f.x))) \wedge$$
$$\neg(\exists b \in B\ \neg(\exists f \in F\ (b.y=f.y))) \wedge$$
$$\neg(\exists f \in F\ \ \neg(\exists a \in A\ (f.x=a.x)) \vee \neg(\exists b \in B\ (f.y=b.y)) \vee$$
$$\exists f' \in F\ ((f.x=f'.x \wedge f.y \mathrel{!=} f'.y) \vee (f.x \mathrel{!=} f'.x \wedge f.y=f'.y))) \}$$

Exercise 3 – SQL

```
SELECT 1 FROM dummy
WHERE NOT EXISTS (SELECT * FROM A a
                  WHERE NOT EXISTS (SELECT * FROM F f
                                    WHERE a.x=f.x) )
AND NOT EXISTS ( SELECT * FROM B b
                 WHERE NOT EXISTS (SELECT * FROM F f
                                   WHERE b.y=f.y)  )
AND NOT EXISTS ( SELECT * FROM F f
                 WHERE NOT EXISTS (SELECT * FROM A a
                                   WHERE a.x=f.x)
                 OR NOT EXISTS (SELECT * FROM B b
                                WHERE b.y=f.y)
                 OR EXISTS (SELECT * FROM F f'
                            WHERE (f.x=f'.x AND f.y != f'.y)
                             OR       (f.x != f'.x AND f.y=f'.y)) );
```

A complicated query

- Schema:
  - “student” S(sid),
  - “course” C(cid),
  - “course taken” T(sid, cid)
- Find the students who take at least one course for which the size of the group of courses that are taken by the same group of students who also take this course is maximal.

A complicated query #2

- Find the students who take at least one course x for which the size of the group of (courses that are taken by the same group of students) who also take x is maximal.
- Ceq := (pairs of) courses that are taken by the same group of students.

```
{ (x,y) | forall s: Course(x) and Course(y) and
      (Taken(s, x) <=> Taken(s, y)) }
```

```
{ (x,y) | Course(x) and Course(y)
   and not exists s: not
      (((not Taken(s, x)) or Taken(s, y))  and
      (Taken(s, x) or (not Taken(s, y)))) }
```

A complicated query #3

Domain relational calculus:

```
Ceq := { (x,y) | Course(x) and Course(y)
    and (not exists s:
       (Taken(s, x) and (not Taken(s, y))))
    and (not exists s:
       (Taken(s, y) and (not Taken(s, x)))) }
```

A complicated query #4

Tuple relational calculus:

```
Ceq := { (c1.cid, c2.cid) | (c1 in Course) and (c2 in Course)
    and not exists (
       (t1 in Taken) and (t1.cid = c1.cid) and
       (not exists ((t2 in Taken) and
                    (t2.cid = c2.cid) and (t1.sid = t2.sid))))
    and not exists (
       (t2 in Taken) and (t2.cid = c2.cid) and
       (not exists ((t1 in Taken) and
                    (t1.cid = c1.cid) and (t1.sid = t2.sid))))
}
```

A complicated query #5

```
create view ceq as (
select c1.cid, c2.cid from Course c1, Course c2
where not exists (
   select * from Taken t1
   where t1.cid = c1.cid and not exists (
      select * from Taken t2
      where t2.cid = c2.cid and t1.sid = t2.sid))
and not exists (
   select * from Taken t2
   where t2.cid = c2.cid and not exists (
      select * from Taken t1
      where t1.cid = c1.cid and t1.sid = t2.sid))
)
```

A complicated query #6

- Find the students who take at least one course x for which the (size of the equivalence class of x with respect to Ceq) who also take x is maximal.
- Remember: Ceq is an equivalence relation – it is reflexive, symmetric, and transitive

- Assume the schema of Ceq is (c1, c2)
- size of the equivalence class of x with respect to Ceq:

```
create view grpsize as (
select c1, count(c2) from Ceq group by c1
)
```

A complicated query #7

- Find the students who take at least one course x for which grpsize(x, s) is such that s is the maximal groupsize).
- Schema grpsize(cid, size)
- Cwmaxgrpsize: courses with maximal group size.

```
create view Cwmaxgrpsize as
(select cid from grpsize where
size = (select max(size) from grpsize)
)
```

A complicated query #8

- Find the students who take at least one course x in cwmaxgrpsize.

```
select S.sid
from   Student S, Taken T,
       Cvmaxgrpsize C
where  S.sid = T.sid
and    T.cid = C.cid
```