Introduction to Data Management CSE 344

Lecture 14: Datalog (cont.)

(Ch 5.3–5.4)

Announcements

- HW3 was due yesterday.

- HW4 will be up today. Due next Tuesday.
  - No coding, Datalog and RC on paper.

- WQ4 is due next Monday
  - it will be useful review for the midterm
  - finish it early if you have time

- Midterm on Friday, July 21h, in class…
  - All the web quizes are open if that helps you study

Announcements

- Change in Syllabus

	Old	New
HW	30	30
Web Quiz	15	15
Participation	5	5
Midterm	20	**25**
Final	30	**25**

Final will be 1h and similar format to midterm. Focus on second half of class

UW DB News

University of Washington Database Group Follow
Jul 18 · 6 min read

Introducing Cosette

Today we are thrilled to announce our official 1.0 release of Cosette, a SQL solver for *automatically checking semantic equivalences of SQL queries*. With Cosette, one can easily verify the correctness of SQL rewrite rules, find errors in buggy SQL rewrites, building auto-graders for SQL assignments, developing SQL optimizers, busting "fake SQLs," etc.

https://medium.com/@uwdb/introducing-cosette-527898504bd6

Midterm

- Content
  - Lectures 1 through 13 (Monday)
  - HW 1–3, WQ 1–4
- Closed book. No computers, phones, watches, etc.!
- Can bring one letter-sized piece of paper with notes, but…
  - test will not be about memorization
  - formulas provided for join algorithms & selectivity
  - can ask me during test about anything you could look up
- Similar in format & content to CSE 344 17wi midterm
  - Previous midterms on course webpage

Midterm Concept Review I

- relational data model
  - set semantics vs bag semantics
  - primary & secondary keys
  - foreign keys
  - schemas

- SQL
  - CREATE TABLE
  - SELECT-FROM-WHERE (SFW)
  - joins: inner vs outer, natural
  - group by & aggregation
  - ordering
  - CREATE INDEX

Midterm Concept Review II

- relational queries
  - languages for writing them:
    - standard relational algebra
    - datalog (even without recursion)
    - SQL (even without grouping / aggregation)
  - monotone queries are a proper subset
  - SFW queries (i.e., w/out subqueries) are monotone

- Given an English problem statement you should be able to write a query in:
  - Relational Algebra , Relational Calculus
  - Datalog , SQL

Midterm Concept Review III

- types of indexes
  - B+ tree vs hash
    - hash indexes use at most 2 disk accesses
    - B+ tree can be used for < predicates
    - B+ tree index on (X,Y) also allows searching for X=a matches
  - clustered vs non-clustered
    - selectivity above 1-2% => not helped by non-clustered indexes

- cost-based query optimization
  - consider choices over logical and physical query plans
    - most important choice in latter is choice of join algoirthm
    - those include nested loop, sorted merge, hash, and indexed joins
  - primary goal of the optimizer is to avoid really bad plans

Today

- More Datalog
- Midterm Review

What is Datalog?

- Another query language for relational model
  - Simple and elegant
  - Initially designed for *recursive* queries
  - Some companies use datalog for data analytics
    - e.g. LogicBlox
  - Increased interest due to recursive analytics

- We discuss only *recursion-free* or *non-recursive* datalog and add negation

Why Do We Learn Datalog?

- Datalog can be translated to SQL
  – Helps to express complex queries
- Increase in datalog interest due to recursive analytics
- A query language that is closest to mathematical logic
  – Good language to reason about query properties
  – Can show that:
  1. Non-recursive datalog & RA have **equivalent power**
  2. Recursive datalog is strictly more powerful than RA
  3. Extended RA & SQL92 is strictly more powerful than datalog

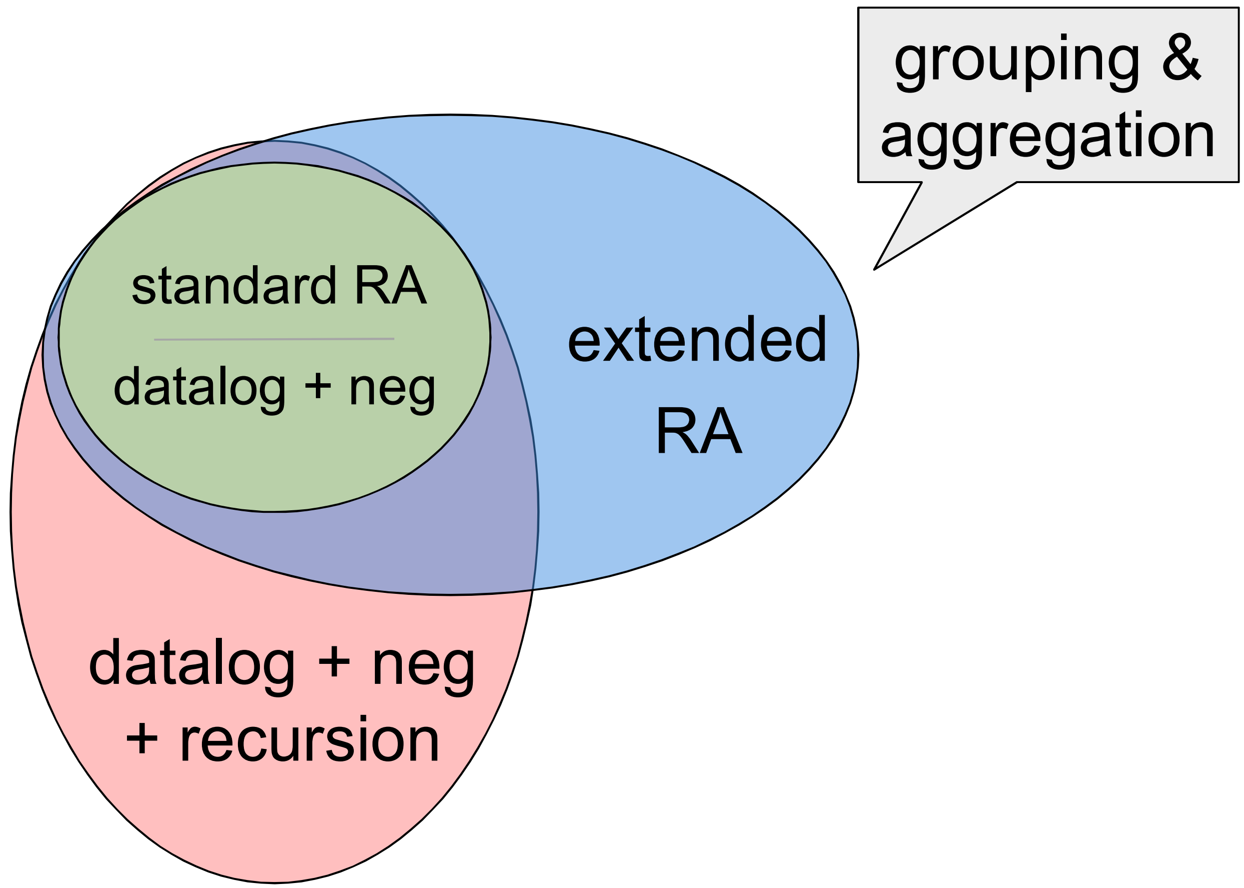
Datalog vs Relational Algebra

- Every expression in standard relational algebra can be expressed as a Datalog query

- But operations in the extended relational algebra (grouping, aggregation, and sorting) have no corresponding features in the version of datalog that we discussed today

- Similarly, datalog can express recursion, which relational algebra cannot

Datalog vs Relational Algebra



Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Datalog with negation

Find all actors who only acted in 1994.

This is wrong. Why?

```
A(x) :- Actor(x,_,_) Cast(x,m) Movie(m,_,1994)
```

not1994(x) :- A(x,_,_) Cast(x,m) Movie(m,_,y)
y != 1994

```
nonAns(x) :- Actor(x,_,_) Cast(x,m) Movie(m,_,y),y!=1994
A(x) :-  Actor(x,_,_) , not nonAns(x)
```

Friend(name1, name2)
Enemy(name1, name2)

More Examples

Find all of Joe's friends who do not have any friends except for Joe:

```
JoeFriends(x) :- Friend('Joe',x)
NonAns(x) :- Friend(y,x), y != ‘Joe’
A(x) :- JoeFriends(x), not NonAns(x)
```

Datalog Summary

- facts (extensional relations - EDBs) and rules (intensional relations - IDBs)
  - rules can use relations, arithmetic, union, intersect, …
- As with SQL, existential quantifiers are easier
  - use negation to handle universal

Using what we have learned

How to write a complex SQL query:

- Write it in RC
- Translate RC to datalog
- Translate datalog to SQL

Take shortcuts when you know what you're doing

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

$Q(x) = \exists y.\ Likes(x, y) \wedge \forall z.(Serves(z,y) \Rightarrow Frequents(x,z))$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

$$Q(x) = \exists y.\ Likes(x, y) \wedge \forall z.(Serves(z,y) \Rightarrow Frequents(x,z))$$

P ⇒ Q same as ¬P ∨ Q

∀x P(x) same as ¬∃x ¬P(x)

Step 1: Replace ∀ with ∃ using de Morgan's Laws

¬(¬P∨Q) same as P∧ ¬ Q

$$Q(x) = \exists y.\ Likes(x, y) \wedge \neg\exists z.(Serves(z,y) \wedge \neg Frequents(x,z))$$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

$P \Rightarrow Q$ same as $\neg P \lor Q$

$Q(x) = \exists y.\ Likes(x, y) \land \forall z.(Serves(z,y) \Rightarrow Frequents(x,z))$

$\forall x\ P(x)$ same as $\neg \exists x\ \neg P(x)$

Step 1: Replace ∀ with ∃ using de Morgan's Laws

$\neg(\neg P \lor Q)$ same as $P \land \neg Q$

$Q(x) = \exists y.\ Likes(x, y) \land \neg \exists z.(Serves(z,y) \land \neg Frequents(x,z))$

Step 2: Make sure the query is domain independent

$Q(x) = \exists y.\ Likes(x, y) \land \neg \exists z.(Likes(x,y) \land Serves(z,y) \land \neg Frequents(x,z))$

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog$^{\neg}$ to SQL

$Q(x) = \exists y.\ Likes(x, y) \wedge \neg\ \exists z.(Likes(x,y) \wedge Serves(z,y) \wedge \neg Frequents(x,z))$

H(x,y)

Step 3: Create a datalog rule for each subexpression;
(shortcut: only for “important” subexpressions)

```
H(x,y)   :- Likes(x,y),Serves(z,y), not Frequents(x,z)
Q(x)     :- Likes(x,y), not H(x,y)
```

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog¬ to SQL

```
H(x,y)   :- Likes(x,y),Serves(z,y), not Frequents(x,z)
Q(x)     :- Likes(x,y), not H(x,y)
```

Step 4: Write it in SQL

```
SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
  (SELECT * FROM Likes L2, Serves S
   WHERE … …)
```

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog¬ to SQL

```
H(x,y)   :- Likes(x,y), Serves(z,y), not Frequents(x,z)
Q(x)     :- Likes(x,y), not H(x,y)
```

Step 4: Write it in SQL

```
SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
  (SELECT * FROM Likes L2, Serves S
   WHERE L2.drinker=L.drinker and L2.beer=L.beer
         and L2.beer=S.beer
         and not exists (SELECT * FROM Frequents F
                         WHERE F.drinker=L2.drinker
                           and F.bar=S.bar))
```

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog¬ to SQL

```
H(x,y)  :- ~~Likes(x,y)~~,Serves(z,y), not Frequents(x,z)
Q(x)    :- Likes(x,y), not H(x,y)
```

Unsafe rule

Improve the SQL query by using an unsafe datalog rule

```
SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
  (SELECT * FROM Serves S
   WHERE L.beer=S.beer
        and not exists (SELECT * FROM Frequents F
                        WHERE F.drinker=L.drinker
                              and F.bar=S.bar))
```

Summary: all these formalisms are equivalent!

- We have seen these translations:
  - RA → datalog¬
  - RC → datalog¬
- Practice at home, and read *Query Language Primer:*
  - Nonrecursive datalog¬ → RA
  - RA → RC
- Summary:
  - RA, RC, and non-recursive datalog¬ + negation can express the same class of queries, called Relational Queries

Query Optimizer Summary

- **Input**: A logical query plan
- **Output**: A good physical query plan
- **Basic query optimization algorithm**
  - Enumerate alternative plans (logical and physical)
  - Compute estimated cost of each plan
    - Compute number of I/Os
    - Optionally take into account other resources
  - Choose plan with lowest cost
  - This is called cost-based optimization

Cost of Join Algorithms

- Nested Loop
  - B(R) + B(R)B(S)
- Nested Loop (with index)
  - If index on S is clustered: B(R) + T(R)B(S)/V(S,A)
  - If index on S is unclustered: B(R) + T(R)T(S)/V(S,A)
- Hash Join
  - B(R) + B(S)
  - uses more disk space when B(R) > M

Review: Physical Query

We only care about Disk I/O operations

Supplier (sid, sname ,scity, sstate)

Supply (pno, sid, quanty)

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = ‘Seattle’
    and x.sstate = ‘WA’
```

T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 11

T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

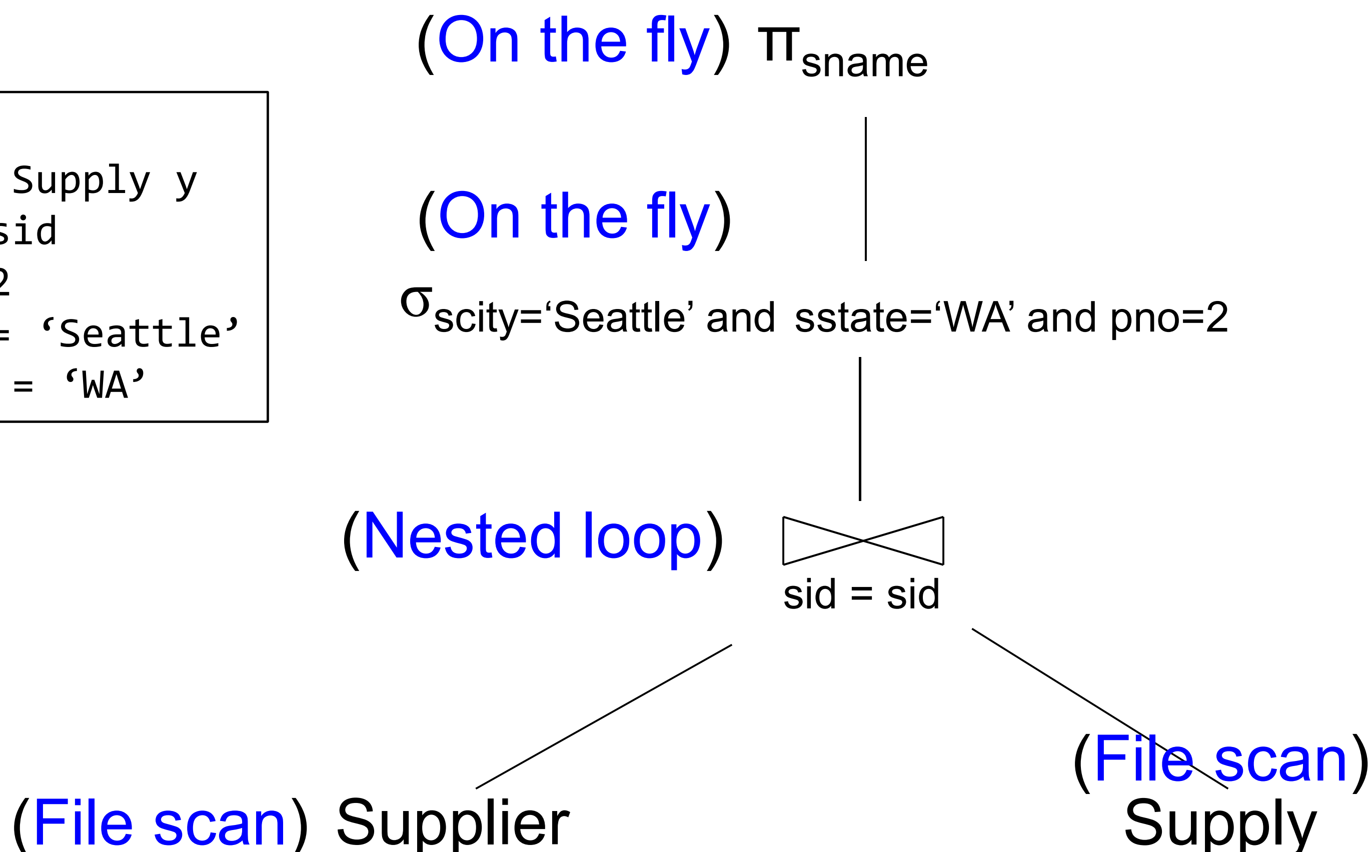
V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 11

Physical Query: Naive Plan

$\pi_{sname}(\sigma_{scity=\text{'Seattle'} \text{ and } sstate=\text{'WA'} \text{ and } pno=2}(\text{Supplier} \bowtie \text{Supply}))$

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```



T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 11

Physical Query: Naive Plan

(On the fly) π_{sname}

(On the fly) $\sigma_{scity='Seattle' \text{ and } sstate='WA' \text{ and } pno=2}$

(Nested loop) ⋈ sid = sid

Supplier (File scan)

Supply (File scan)

Selection and project on-the-fly
→ No additional cost.

Total cost of plan is thus cost of join:
= B(Supplier)+B(Supplier)*B(Supply)
= 100 + 100 * 100
= 10,100 I/Os

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

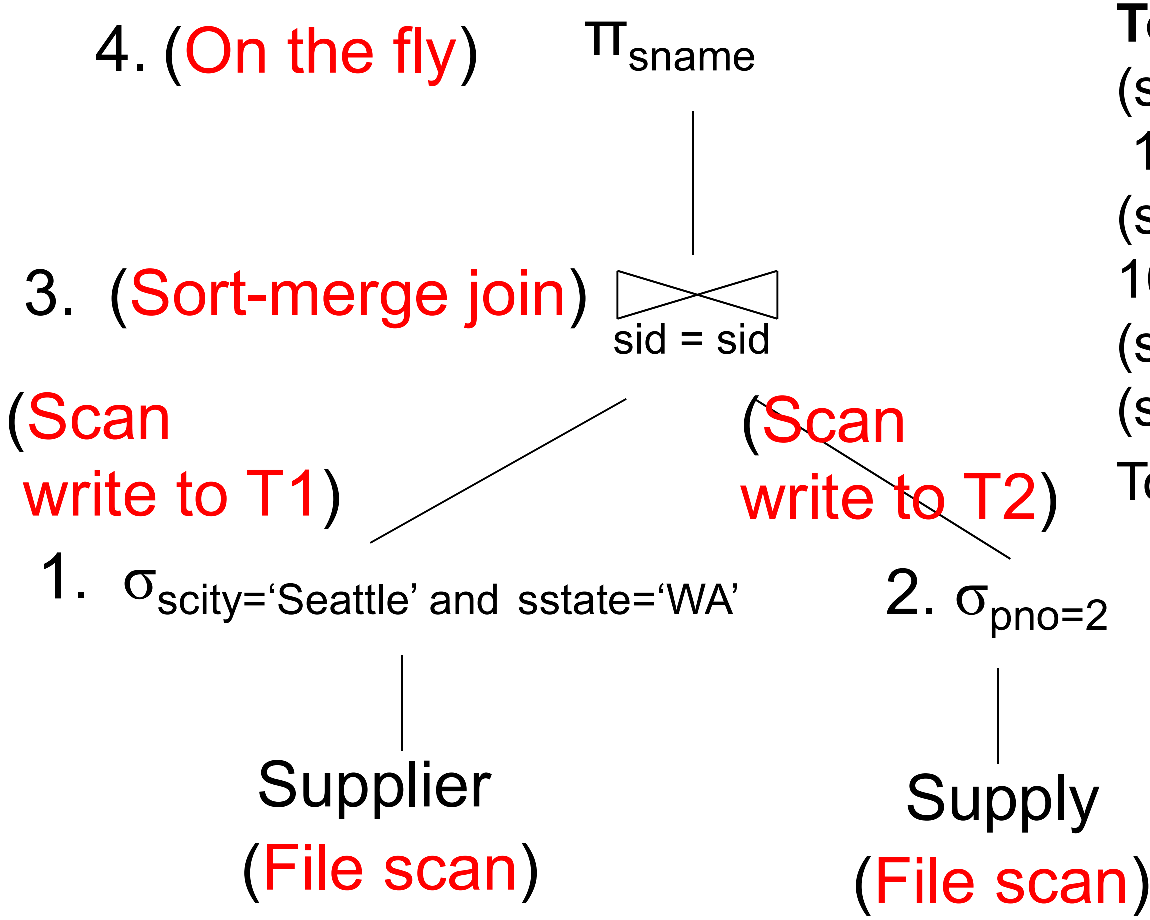
T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 11

Physical Query: Optimized



Total cost

(step 1)

100 + 1000 * 1/20 * 1/10 ~= 100

(step 2)

100 + 100 * 1/2500 ~= 100

(step 3) 2

(step 4) 0

Total cost ≈ **204 I/Os**

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 11

Physical Query: Using Indexes

(On the fly) 4. π_{sname}

B(R) V(R,a) + T(R) V(R,a)

(On the fly)

3. $\sigma_{scity=\text{'Seattle' and sstate='WA'}}$

2. ⋈ sid = sid (Index nested loop)

4 reads ⇓ one per tuple

(Use hash index)

10000*1/2500 = 4 tuples

1 Block

1. $\sigma_{pno=2}$

Supply
(Index on pno)
Assume: clustered

Supplier
(Index on sid)
Clustering does not matter

Total cost
= 1 (or 2) (step 1.)
+ 4 (step 2.)
+ 0 (step 3.)
+ 0 (step 4.)
Total cost ≈ **5 I/Os (or 6)**

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```