Week 2-2

SQL and Query Processing

Big Data

Prof. Hwanjo Yu
POSTECH

Interpreting complicated SQL

```
SELECT binid,
        round(avg(cast(Fluo as float)),3) as Fluo,
        round(avg(cast(Oxygen as float)),3) as Oxygen,
        round(avg(cast(Nitrate_uM as float)),3) as Nitrate_uM,
        round(avg(cast(longitude as float)),3) as longitude,
        round(avg(cast(latitude as float)),3) as latitude
FROM (
        SELECT *,
                cast(floor(ts) + floor((ts - floor(ts))*24*60/binsize) * binsize / (24*60) as datetime) as binid
        FROM (
                SELECT *,
                        cast(timestamp as float) as ts,
                        5.0 as binsize
                FROM Tokyo_4_merged_data_time
        ) x
) bins
GROUP BY binid
ORDER BY binid ASC
```

Interpreting complicated SQL

```
SELECT binid,
        round(avg(cast(Fluo as float)),3) as Fluo,
        round(avg(cast(Oxygen as float)),3) as Oxygen,
        round(avg(cast(Nitrate_uM as float)),3) as Nitrate_uM,
        round(avg(cast(longitude as float)),3) as longitude,
        round(avg(cast(latitude as float)),3) as latitude
FROM (
        SELECT *,
                cast(floor(ts) + floor((ts - floor(ts))*24*60/binsize) * binsize / (24*60) as datetime) as binid
        FROM (
                SELECT *,
                        cast(timestamp as float) as ts,
                        5.0 as binsize
                FROM Tokyo_4_merged_data_time
        ) x
) bins
GROUP BY binid
ORDER BY binid ASC
```

Interpreting complicated SQL

```
SELECT x.strain, x.chr, x.region as snp_region, x.start_bp as snp_start_bp, x.end_bp as snp_end_bp,
        w.start_bp as nc_start_bp, w.end_bp as nc_end_bp, w.category as nc_category,
        CASE WHEN (x.start_bp >= w.start_bp AND x.end_bp <= w.end_bp)
                    THEN x.end_bp - x.start_bp + 1
              WHEN (x.start_bp <= w.start_bp AND w.start_bp <= x.end_bp)
                    THEN x.end_bp - w.start_bp + 1
              WHEN (x.start_bp <= w.end_bp AND w.end_bp <= x.end_bp)
                    THEN w.end_bp - x.start_bp + 1
        END  AS len_overlap
 FROM hotspots_deserts x JOIN table_noncoding_positions w ON x.chr = w.chr
 WHERE (x.start_bp >= w.start_bp AND x.end_bp <= w.end_bp)
        OR (x.start_bp <= w.start_bp AND w.start_bp <= x.end_bp)
        OR (x.start_bp <= w.end_bp AND w.end_bp <= x.end_bp)
 ORDER BY x.strain, x.chr ASC, x.start_bp ASC
```

Interpreting complicated SQL

```
SELECT x.strain, x.chr, x.region as snp_region, x.start_bp as snp_start_bp, x.end_bp as snp_end_bp,
    w.start_bp as nc_start_bp, w.end_bp as nc_end_bp, w.category as nc_category,
    CASE WHEN (x.start_bp >= w.start_bp AND x.end_bp <= w.end_bp)
            THEN x.end_bp - x.start_bp + 1
        WHEN (x.start_bp <= w.start_bp AND w.start_bp <= x.end_bp)
            THEN x.end_bp - w.start_bp + 1
        WHEN (x.start_bp <= w.end_bp AND w.end_bp <= x.end_bp)
            THEN w.end_bp - x.start_bp + 1
    END  AS len_overlap
FROM hotspots_deserts x JOIN table_noncoding_positions w ON x.chr = w.chr
WHERE (x.start_bp >= w.start_bp AND x.end_bp <= w.end_bp)
    OR (x.start_bp <= w.start_bp AND w.start_bp <= x.end_bp)
    OR (x.start_bp <= w.end_bp AND w.end_bp <= x.end_bp)
ORDER BY x.strain, x.chr ASC, x.start_bp ASC
```

POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Interpreting complicated SQL

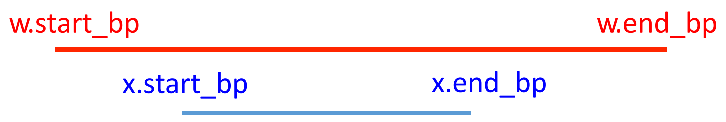
```
SELECT x.strain, x.chr, x.region as snp_region, x.start_bp as snp_start_bp, x.end_bp as snp_end_bp,
       w.start_bp as nc_start_bp, w.end_bp as nc_end_bp, w.category as nc_category,
       len_overlap(x.start_bp, x.end_bp, w.start_bp, w.end_bp)
FROM hotspots_deserts x JOIN table_noncoding_positions w ON x.chr = w.chr
WHERE (x.start_bp >= w.start_bp AND x.end_bp <= w.end_bp)
      OR (x.start_bp <= w.start_bp AND w.start_bp <= x.end_bp)
      OR (x.start_bp <= w.end_bp AND w.end_bp <= x.end_bp)
ORDER BY x.strain, x.chr ASC, x.start_bp ASC
```

Interpreting complicated SQL

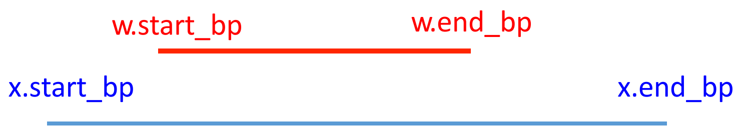
```
SELECT x.strain, x.chr, x.region as snp_region, x.start_bp as snp_start_bp, x.end_bp as snp_end_bp,
        w.start_bp as nc_start_bp, w.end_bp as nc_end_bp, w.category as nc_category,
        len_overlap(x.start_bp, x.end_bp, w.start_bp, w.end_bp)
 FROM hotspots_deserts x JOIN table_noncoding_positions w ON x.chr = w.chr
 WHERE (x.start_bp >= w.start_bp AND x.end_bp <= w.end_bp)
        OR (x.start_bp <= w.start_bp AND w.start_bp <= x.end_bp)
        OR (x.start_bp <= w.end_bp AND w.end_bp <= x.end_bp)
 ORDER BY x.strain, x.chr ASC, x.start_bp ASC
```

Interpreting complicated SQL

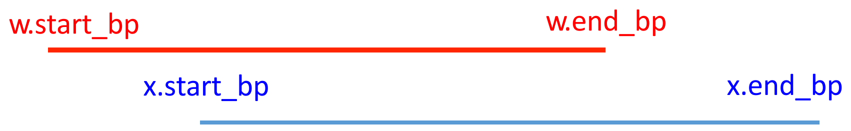
(x.start_bp >= w.start_bp AND x.end_bp <= w.end_bp)



OR (x.start_bp <= w.start_bp AND w.start_bp <= x.end_bp)



OR (x.start_bp <= w.end_bp AND w.end_bp <= x.end_bp)



Interpreting complicated SQL

```
SELECT x.strain, x.chr, x.region as snp_region, x.start_bp as snp_start_bp, x.end_bp as snp_end_bp,
        w.start_bp as nc_start_bp, w.end_bp as nc_end_bp, w.category as nc_category,
        len_overlap(x.start_bp, x.end_bp, w.start_bp, w.end_bp)
 FROM hotspots_deserts x JOIN table_noncoding_positions w ON x.chr = w.chr
 WHERE overlaps(x.start_bp, x.end_bp, w.start_bp, w.end_bp)
 ORDER BY x.strain, x.chr ASC, x.start_bp ASC
```

Interpreting complicated SQL

```
SELECT x.strain, x.chr, x.region as snp_region, x.start_bp as snp_start_bp, x.end_bp as snp_end_bp,
      w.start_bp as nc_start_bp, w.end_bp as nc_end_bp, w.category as nc_category,
      len_overlap(x.start_bp, x.end_bp, w.start_bp, w.end_bp)
 FROM hotspots_deserts x JOIN table_noncoding_positions w ON x.chr = w.chr
 WHERE overlaps(x.start_bp, x.end_bp, w.start_bp, w.end_bp)
 ORDER BY x.strain, x.chr ASC, x.start_bp ASC
```

```
SELECT x.strain, x.chr, x.region as snp_region, x.start_bp as snp_start_bp, x.end_bp as snp_end_bp,
      w.start_bp as nc_start_bp, w.end_bp as nc_end_bp, w.category as nc_category,
      len_overlap(x.start_bp, x.end_bp, w.start_bp, w.end_bp)
 FROM hotspots_deserts x, table_noncoding_positions w
 WHERE x.chr = w.chr AND overlaps(x.start_bp, x.end_bp, w.start_bp, w.end_bp)
 ORDER BY x.strain, x.chr ASC, x.start_bp ASC
```

User-defined functions

- As a user, you can write a function, register it in the database, call it from SQL, set permissions on it

- Scalar functions

```
SELECT myfunc(r.a, r.b)…
WHERE yourfunc(r.c, r.d) < 5
```

- Aggregate functions

```
SELECT x, concat(r.s) …
GROUP BY x
```

- Table functions

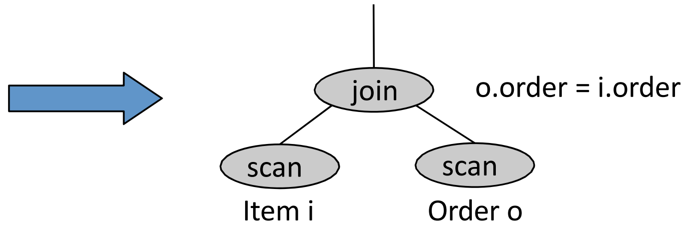
```
SELECT … FROM tablefunc(a,b)
```

User-defined function support

- PostgreSQL (and Greenplum)
  - SQL, PL/pgSQL, Python, C/C++, R,
- Microsoft SQL Server
  - SQL, T-SQL, C# or any CLR language
- Oracle
  - SQL, PL-SQL, Java, C/C++, Python, others
- SQLite
  - NONE!! (sorry)

Same logical expression, different physical algorithms

```
SELECT  *
FROM    Order o, Item i
WHERE  o.order =  i.order
```



Option 1: Nested loop join

```
for each record i in Item:
  for each record o in Order:
     if o.order = i.order:
        return matching pair
```

Which is faster?

Option 2: Hash join

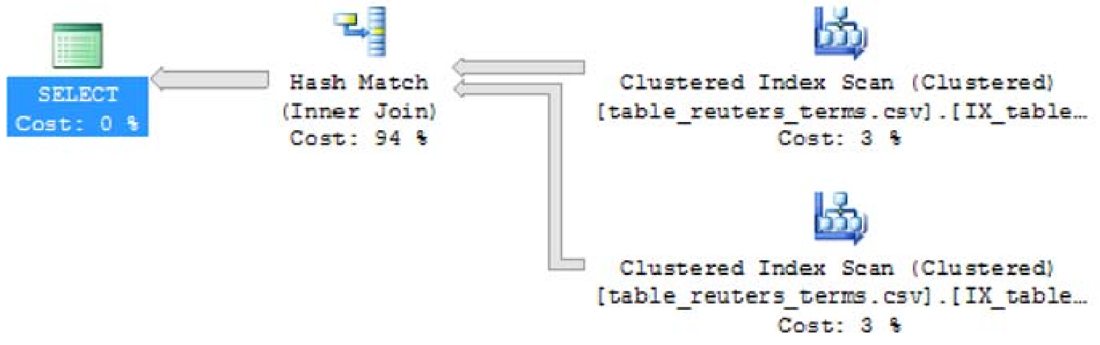
```
for each record i in Item:
   insert into hashtable

for each record o in Order:
  lookup corresponding records in hashtable
  return matching pairs
```

Exposing the algebra: Microsoft SQL server

"Find pairs of terms that co-occur in documents"

```sql
select a.term_id, b.term_id
from [billhowe].[reuters] a, [billhowe].[reuters] b
where a.doc_id = b.doc_id
  and a.term_id != b.term_id
```

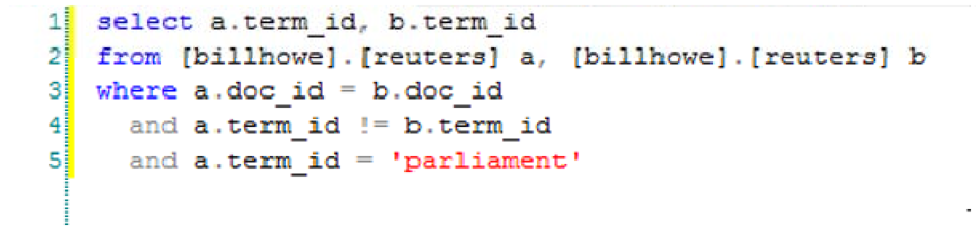
EXPLAIN


SELECT
Cost: 0 %
Hash Match
(Inner Join)
Cost: 94 %
Clustered Index Scan (Clustered)
[table_reuters_terms.csv].[IX_table…
Cost: 3 %
Clustered Index Scan (Clustered)
[table_reuters_terms.csv].[IX_table…
Cost: 3 %

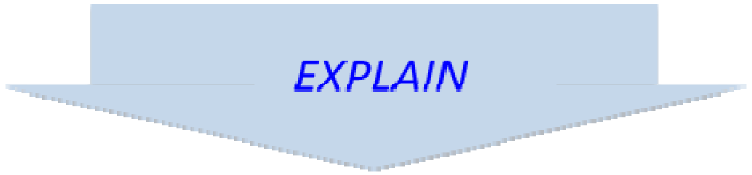
POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

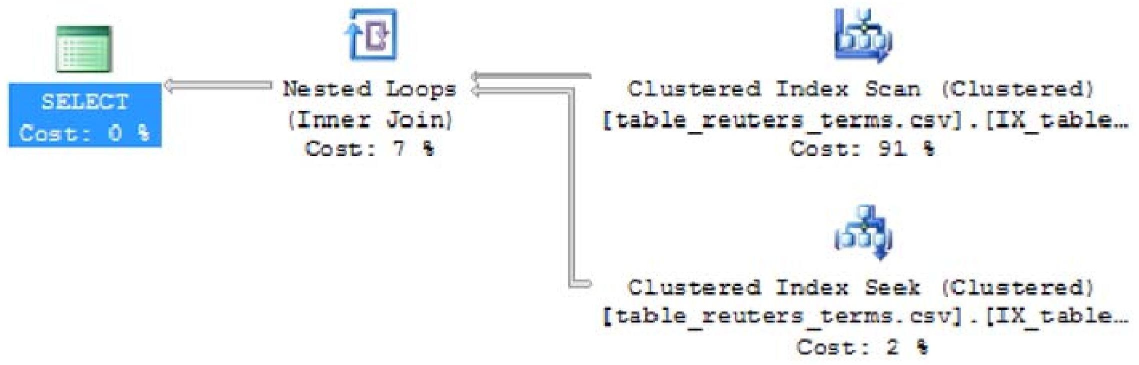
Exposing the algebra: Microsoft SQL server

“Find pairs of terms that co-occur ‘parliament’ in documents”



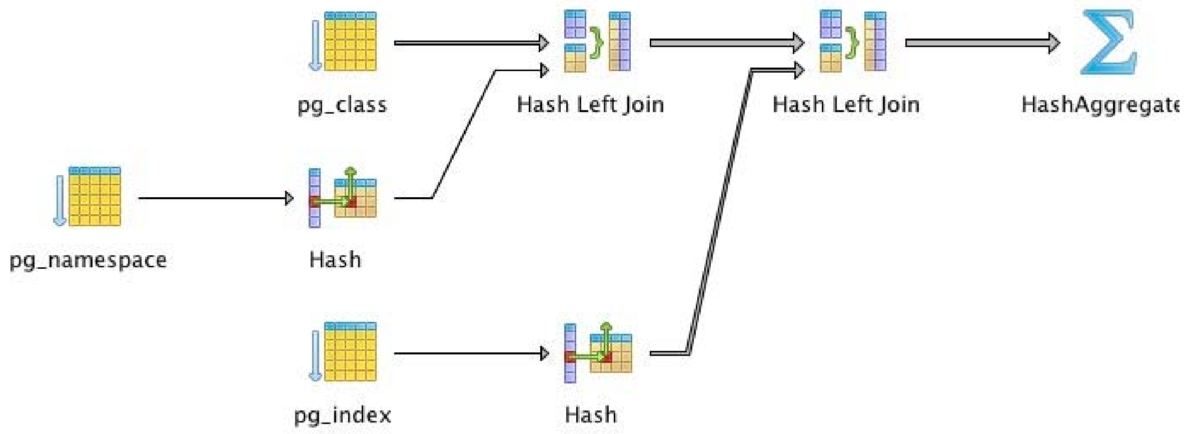
```
select a.term_id, b.term_id
from [billhowe].[reuters] a, [billhowe].[reuters] b
where a.doc_id = b.doc_id
  and a.term_id != b.term_id
  and a.term_id = 'parliament'
```



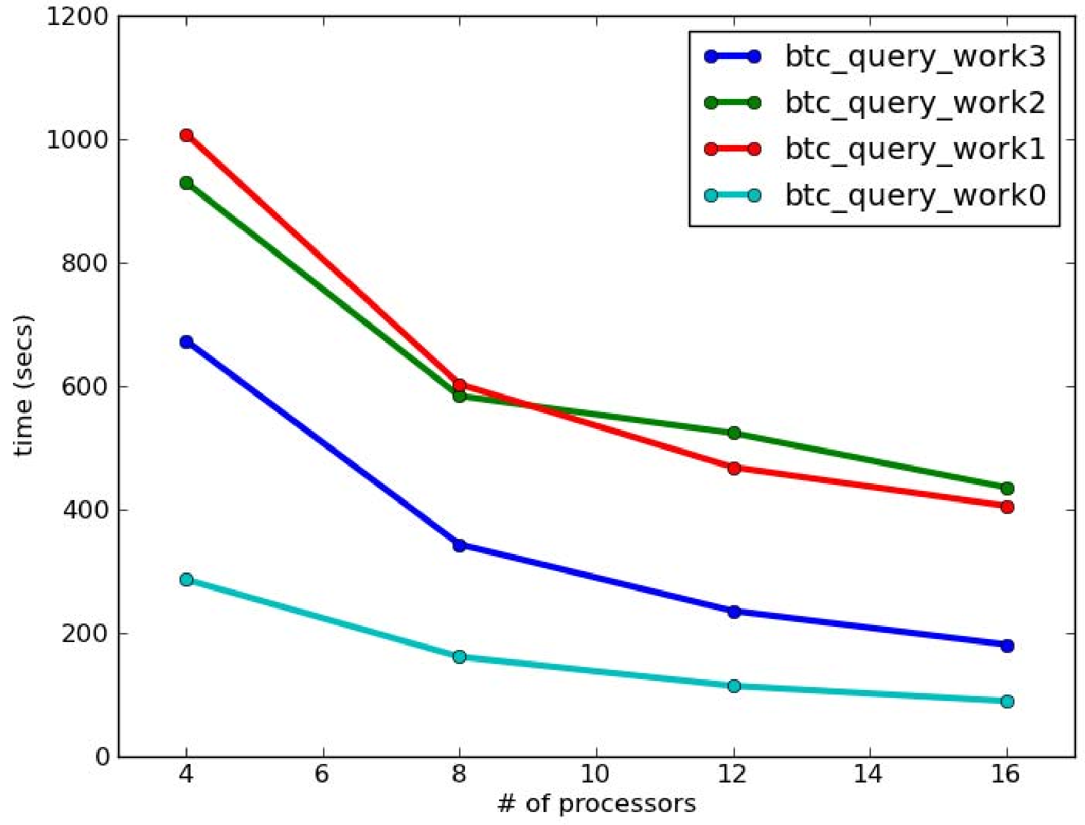

SELECT
Cost: 0 %
Nested Loops
(Inner Join)
Cost: 7 %
Clustered Index Scan (Clustered)
[table_reuters_terms.csv].[IX_table…
Cost: 91 %
Clustered Index Seek (Clustered)
[table_reuters_terms.csv].[IX_table…
Cost: 2 %

Exposing the algebra: PostgreSQL

EXPLAIN SELECT ….



Algebraic optimization matters



BTC 2010 Dataset

3B quads
623 GB processed

Equivalent logical expressions; different costs

$$\sigma_{p=knows}(R) \bowtie_{o=s} (\sigma_{p=holdsAccount}(R) \bowtie_{o=s} \sigma_{p=accountHompage}(R))$$

right associative

$$(\sigma_{p=knows}(R) \bowtie_{o=s} \sigma_{p=holdsAccount}(R)) \bowtie_{o=s} \sigma_{p=accountHompage}(R)$$

left associative

$$\sigma_{p1=knows\ \&\ p2=holdsAccount\ \&p3=accountHompage}(R \times R \times R)$$

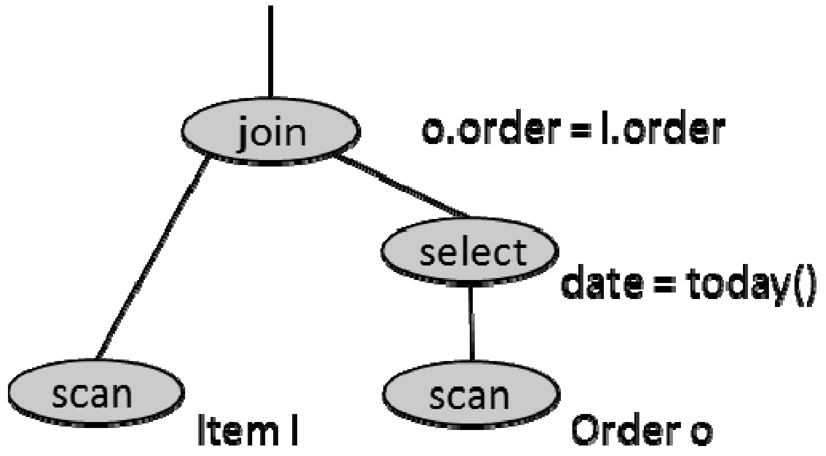
cross product

Key idea: Declarative languages

Order(order, date, account)
Item(order, part)

Find all orders from today, along with the items ordered

```
SELECT  *
  FROM  Order o,  Item i
 WHERE  o.order = i.order
   AND  o.date = today()
```



SQL is the "WHAT" not the "HOW"

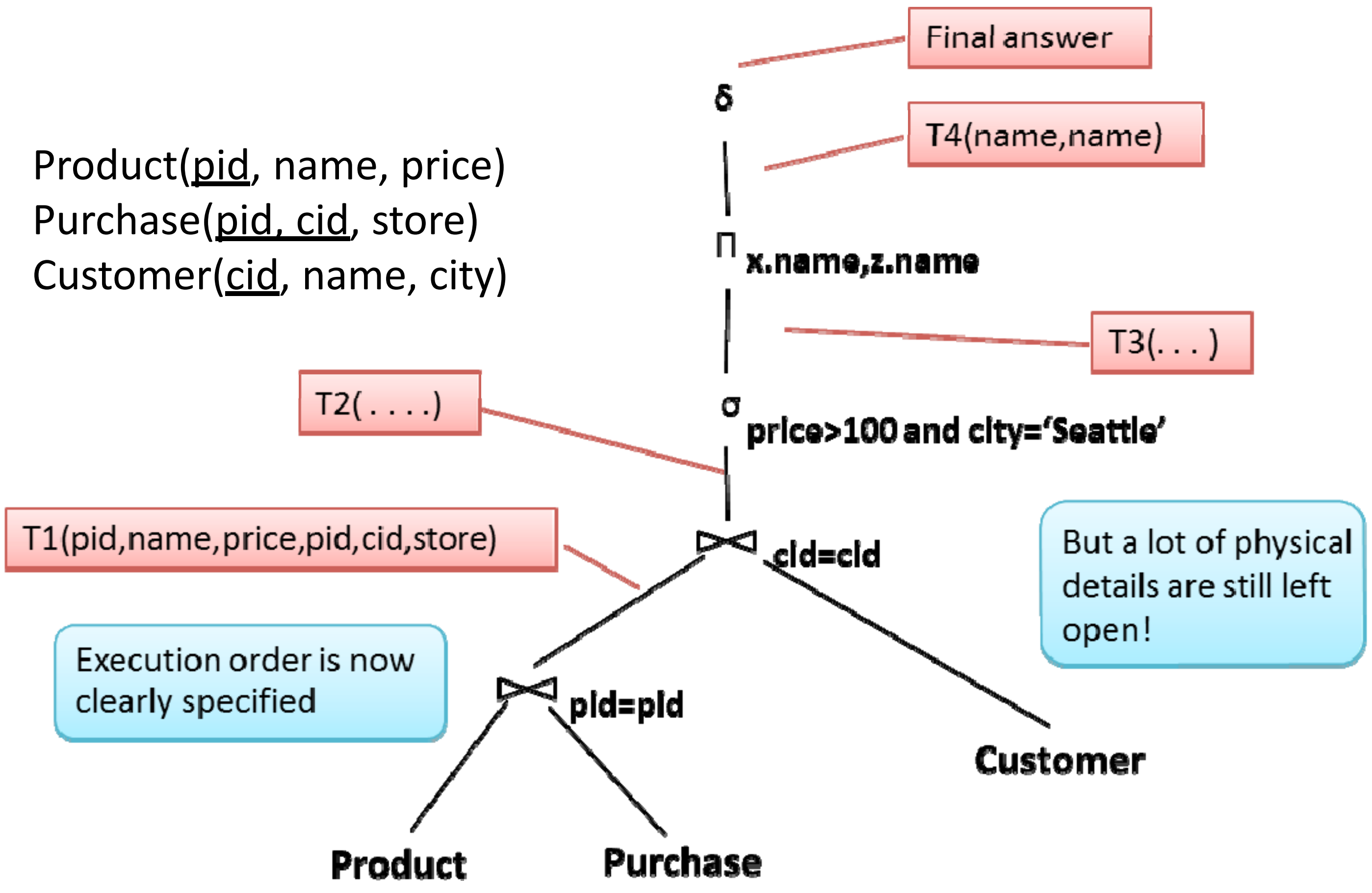
Product(<u>pid</u>, name, price)
Purchase(<u>pid, cid</u>, store)
Customer(<u>cid</u>, name, city)

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
            x.price > 100 and z.city = 'Seattle'
```

It's clear WHAT we want, unclear HOW to get it

Relational algebra

Product(<u>pid</u>, name, price)
Purchase(<u>pid, cid</u>, store)
Customer(<u>cid</u>, name, city)

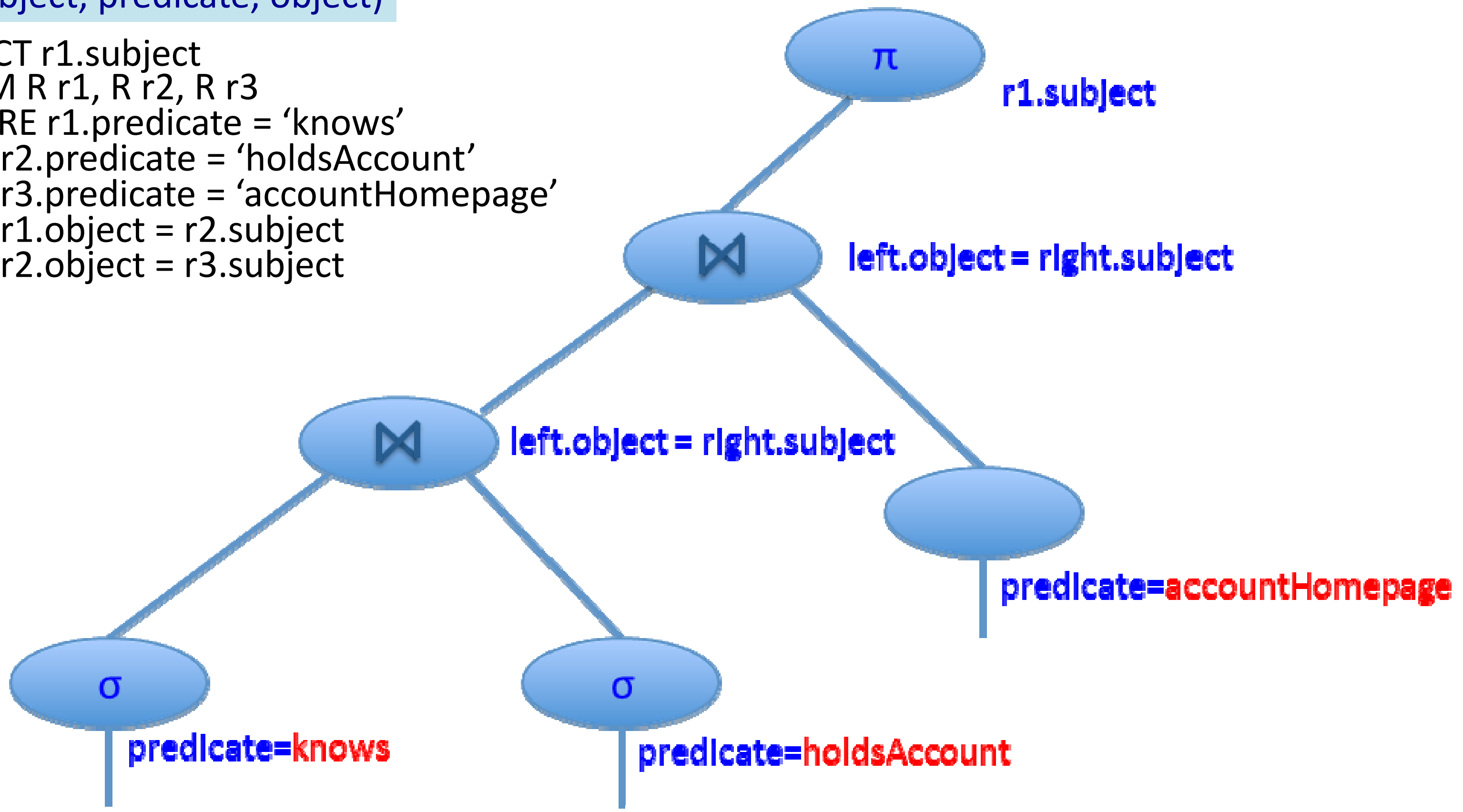


Final answer
δ
T4(name,name)
Π x.name,z.name
T3(. . .)
T2(. . . .)
σ price>100 and city='Seattle'
T1(pid,name,price,pid,cid,store)
⋈ cid=cid
But a lot of physical details are still left open!
Execution order is now clearly specified
⋈ pid=pid
Customer
Product
Purchase

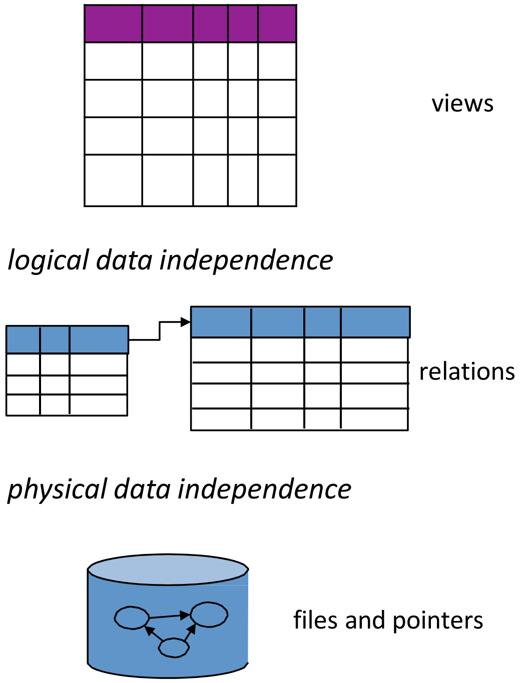
Another example

R(subject, predicate, object)

```
SELECT r1.subject
FROM R r1, R r2, R r3
WHERE r1.predicate = ‘knows’
AND r2.predicate = ‘holdsAccount’
AND r3.predicate = ‘accountHomepage’
AND r1.object = r2.subject
AND r2.object = r3.subject
```


π
r1.subject
⋈
left.object = right.subject
⋈
left.object = right.subject
σ
predicate=knows
σ
predicate=holdsAccount
predicate=accountHomepage

Key idea: “Logical data independence”



```
 SELECT  *
   FROM  my_sequences
```

```
 SELECT  seq
   FROM  ncbi_sequences
  WHERE  seq = ‘GATTACGATATTA’;
```

```
f  =  fopen  (‘table_file’);
fseek  (10030440);
while  (True)  {
    fread  (&buf,  1,  8192,  f);
    if  (buf  ==  GATTACGATATTA) {
        . . .
```

What are views?

- A view is just a query with a name
- We can use the view just like a real table

Why can we do this?

Because we know that every query returns a relation:
We say that the language is “algebraically closed”

View example

A view is a relation defined by a query

Purchase(customer, pid, store)
Product(<u>pid, price</u>)

StorePrice(store, price)

```
CREATE VIEW  StorePrice AS
SELECT x.store, y.price
FROM   Purchase x, Product y
WHERE x.pid = y.pid
```

This is like a new table
StorePrice(store,price)

View example

Customer(<u>cid</u>, name, city)
Purchase(customer, product, store)
Product(<u>pname</u>, price)

StorePrice(store, price)

How to Use a View?

- A "high end" store is a store that sold some product over 1000. For each customer, find all the high end stores that they visit. Return a set of (customer- name, high-end-store) pairs.

```
SELECT DISTINCT z.name, u.store
FROM Customer z, Purchase u, StorePrice v
WHERE z.cid = u.customer
AND u.store = v.store
AND v.price > 1000
```

Key idea: Indexes

- Databases are especially, but not exclusively, effective at “Needle in Haystack” problems:
  - Extracting small results from big datasets
  - Your query will finish, regardless of dataset size.
  - Indexes are easily built and automatically used when appropriate

```
CREATE     INDEX seq_idx ON   sequence(seq);

SELECT     seq
FROM       sequence
WHERE      seq=‘GATTACGATATTA’;
```

SQL exercise

- Install SQLite (Search Internet) in your favorite machine
- create table Documents(DocID int, Term text, Count int);
- .mode csv
- .import Documents.csv Documents (You need to make csv data files.)

Documents.csv

1	data	4
1	base	3
1	system	5
1	fall	6
1	semester	2
2	data	1
2	base	2
2	structure	3
2	network	4
2	algorithm	5
3	ds	2
3	sd	3
3	final	4
3	mid	5
4	system	1
4	ds	4
4	sd	3
4	pl	5
4	data	2
5	base	2
5	structure	2
5	network	3
5	fall	4
6	data	2
6	base	4

SQL exercise

Documents(DocID int, Term text, Count int);

1. Write a SQL query that is equivalent to the following relational algebra expression.
   - $\pi_{\text{Term}}(\sigma_{\text{Docid=2}}(\text{Documents}))\ \text{U}\ \pi_{\text{Term}}(\sigma_{\text{count=3}}(\text{Documents}))$
   - Try both union and union all to see the difference
   - Try “or” instead of “union” and compare results with using union
2. Write a SQL query to count the number of documents containing the word “data”.
3. Write a SQL query to find all documents that have more than 3 terms.
4. Write a SQL query to count the number of documents that contain both the word “data” and “base”.
5. Write a SQL query to compute the similarity of every pair of documents in Documents. The similarity here is computed by summing the same term counts of two documents. For example, the similarity of Doc1 <’a’:2, ‘b’:1, ‘c’:3> and Doc2 <’b’:2, ‘c’:1, ‘d’:4> is 1*2 (‘b’) + 3*1 (‘c’) = 5. (Hint: to avoid computing the similarity of both (Doc1, Doc2) and (Doc2, Doc1), add a condition of the form a.DocID < b.DocID.)

We didn’t cover…

- Design of Relational Databases
  - Drawing ER Diagram, e.g. 1-to-1, many-to-1, many-to-many
  - Relational Normalization, e.g. 1st, 2nd, 3rd normal forms
- Query Processing
  - Indexing methods, e.g. B+tree, Hash, R-tree, …
  - Data organization in disk, e.g. directory, block, fragmentation
  - Disk-based sorting, e.g. multiway merge sort
- Query Optimization
  - Query rewriting, Query execution plan tree, …
- Transaction management
  - ACID properties, two-phase locking, …
- Recovery
  - Logging, WAL, …

1 semester of course