CSE544Database Management Systems

Lecture 3:
Data Models,
SQL Beyond Relations

Announcements

- HW1 due by Sunday
- HW2 is posted
- Lecture on Feb 19 moved to Feb 21, same room
- Final project presentations confirmed: March 14, 2pm-9:30pm
- Two parts, you will be scheduled in one

Outline

- Discuss Data Models
- SQL Beyond Relations

References

- M. Stonebraker and J. Hellerstein. What Goes Around Comes Around. In "Readings in Database Systems" (aka the Red Book). 4th ed.

Data Model Motivation

- Applications need to model real-world data
- User somehow needs to define the data
- Data model enables a user to define the data using high-level constructs without worrying about many low-level details of how data will be stored on disk

Early Proposal 1: IMS*

- What is it?

* IBM Information Management System

Early Proposal 1: IMS*

- **Hierarchical data model**

- **Record**
  - **Type**: collection of named fields with data types
  - **Instance**: must match type definition
  - Each instance has a **key**
  - Record types arranged in a **tree**

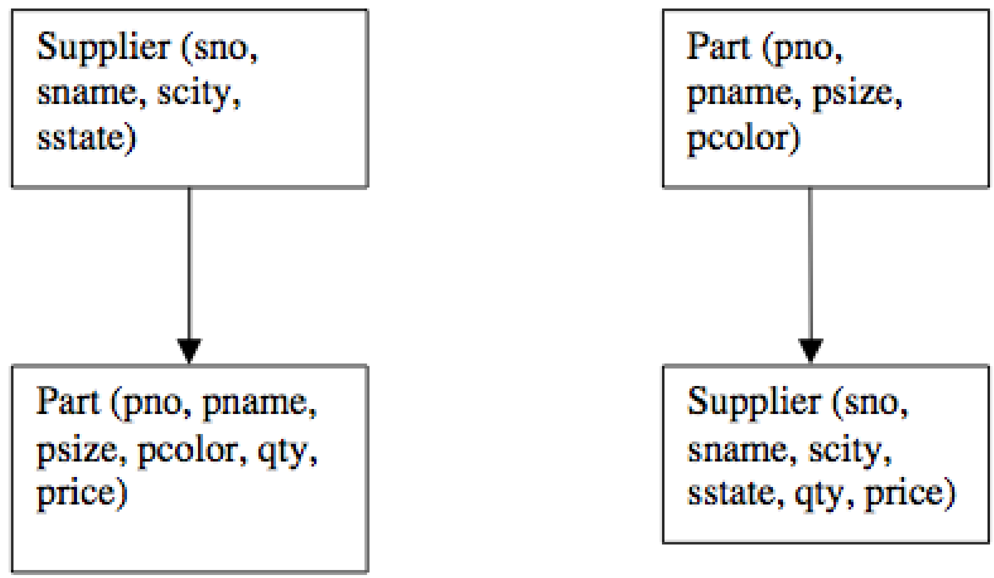
- **IMS database** is collection of instances of record types organized in a tree

* IBM Information Management System

IMS Example

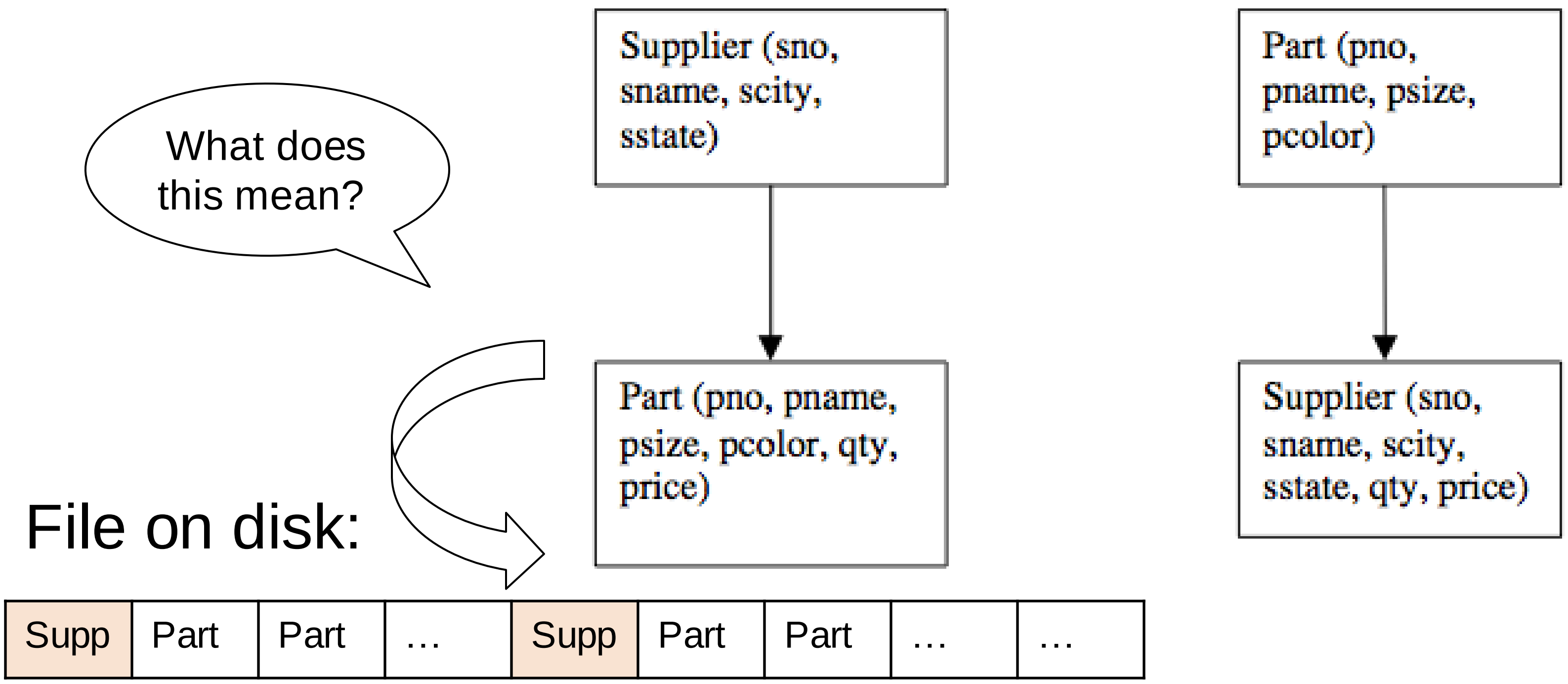
- Figure 2 from “What goes around comes around”

What does this mean?



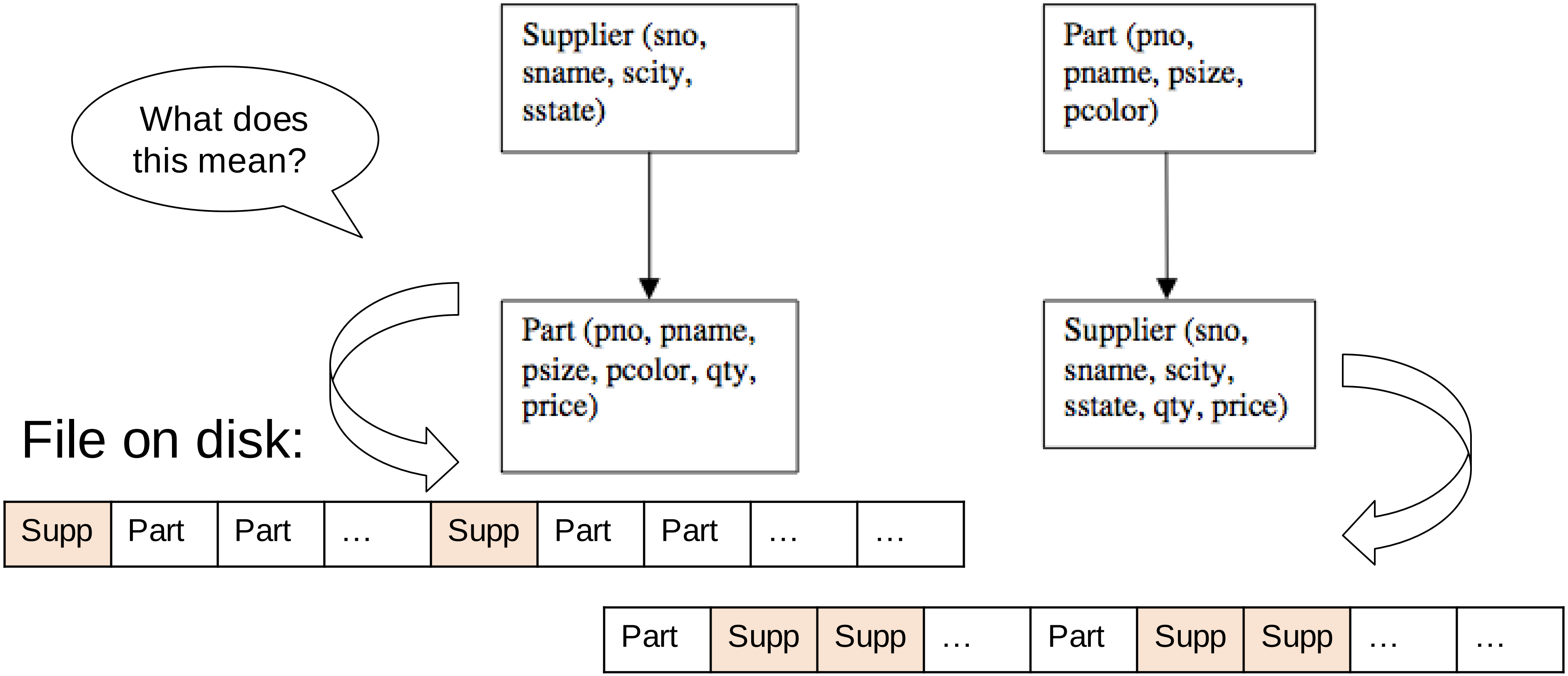
IMS Example

- Figure 2 from “What goes around comes around”



IMS Example

- Figure 2 from “What goes around comes around”



IMS Limitations

IMS Limitations

- **Tree-structured data model**
  - Redundant data; existence depends on parent

IMS Limitations

- **Tree-structured data model**
  - Redundant data; existence depends on parent

- **Record-at-a-time** user interface
  - User must specify algorithm to access data

IMS Limitations

- **Tree-structured data model**
  - Redundant data; existence depends on parent
- **Record-at-a-time** user interface
  - User must specify algorithm to access data
- **Very limited physical independence**
  - Phys. organization limits possible operations
  - Application programs break if organization changes
- **Some logical independence but limited**

Data Manipulation Language: DL/1

- Each record has a hierarchical sequence key (HSK)

- HSK defines semantics of commands:
  - get_next; get_next_within_parent

- **DL/1 is a record-at-a-time language**
  - Programmers construct algorithm, worry about optimization

Data storage

- Root records
  - Stored sequentially (sorted on key)
  - Indexed in a B-tree using the key of the record
  - Hashed using the key of the record
- Dependent records
  - Physically sequential
  - Various forms of pointers
- Selected organizations restrict DL/1 commands
  - No updates allowed due to sequential organization
  - No “get-next” for hashed organization

Data Independence

- **Physical data independence**: Applications are insulated from changes in **physical storage details**

- **Logical data independence**: Applications are insulated from changes to **logical structure of the data**

Lessons from IMS

- Physical/logical data independence needed
- Tree structure model is restrictive
- Record-at-a-time programming forces user to do optimization

Early Proposal 2: CODASYL

What is it?

Early Proposal 2: CODASYL

What is it?

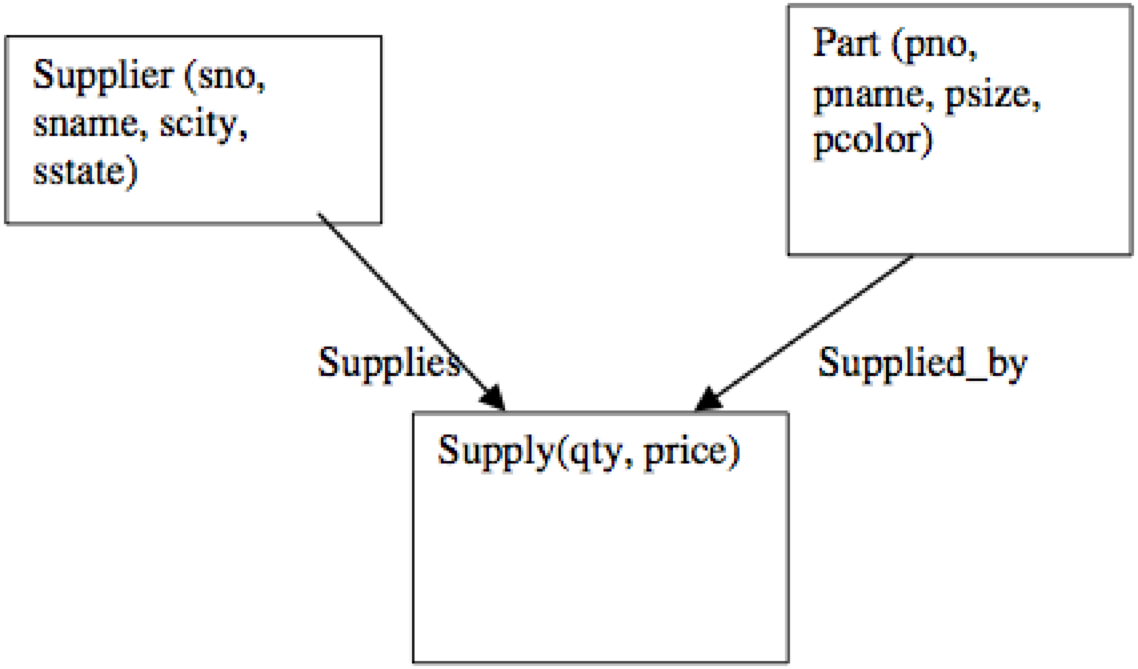
- **Networked data model**

- Primitives are also **record types** with **keys**
- Record types are organized into **network**
- Multiple parents; arcs = “sets”
- More flexible than hierarchy

- **Record-at-a-time** data manipulation language

CODASYL Example

- Figure 5 from “What goes around comes around”



CODASYL Limitations

- No data independence: application programs break if organization changes
- Record-at-a-time: “navigate the hyperspace”

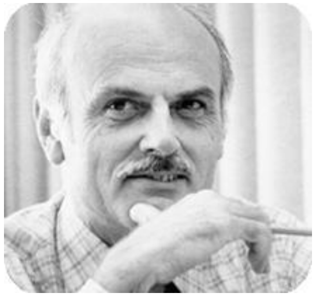
The Programmer as Navigator

by Charles W. Bachman

Relational Model Overview

Ted Codd 1970

- What was the motivation? What is the model?



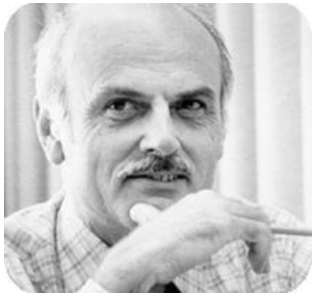
Relational Model Overview

Ted Codd 1970

- Motivation: logical and physical data independence

- Store data in a **simple data structure** (table)
- Access data through **set-at-a-time** language
- **No physical storage proposal**

Relational Database: A Practical Foundation for Productivity



Great Debate

- Pro relational
  - What were the arguments?

- Against relational
  - What were the arguments?

- How was it settled?

Great Debate

- Pro relational
  - CODASYL is too complex
  - No data independence
  - Record-at-a-time hard to optimize
  - Trees/networks not flexible enough

- Against relational
  - COBOL programmers cannot understand relational languages
  - Impossible to implement efficiently

- Ultimately settled by the market place

Data Independence

How it is achieved today:

- Physical independence: SQL to Plan
- Logical independence: Views in SQL

Physical Data Independence

- In SQL we express ***What*** data we want to retrieve

- The optimizers figures out ***How*** to retrieve it

```
Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)
```

Query Plan

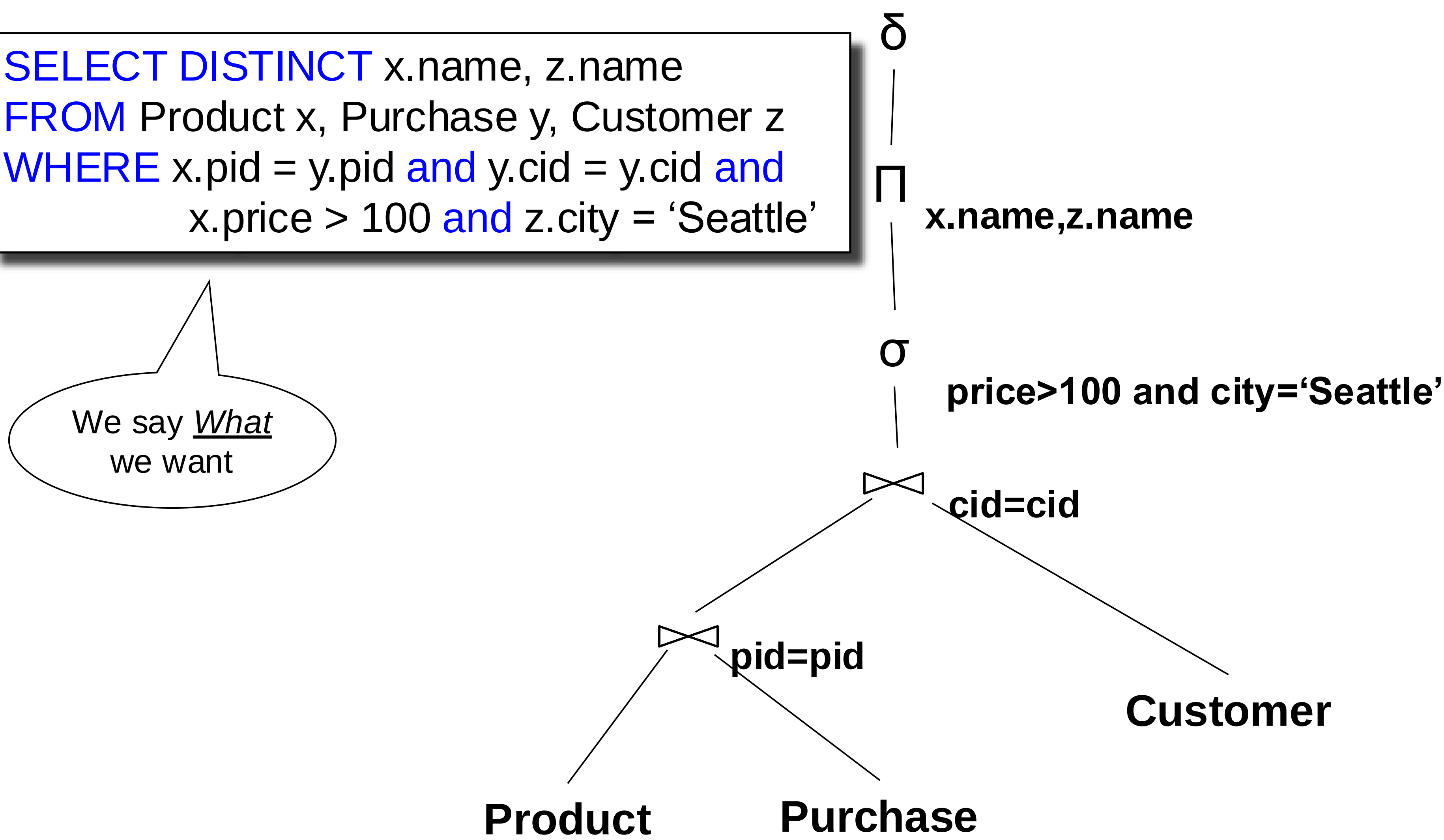
```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
         x.price > 100 and z.city = 'Seattle'
```

We say *What* we want

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

Logical Query Plan

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

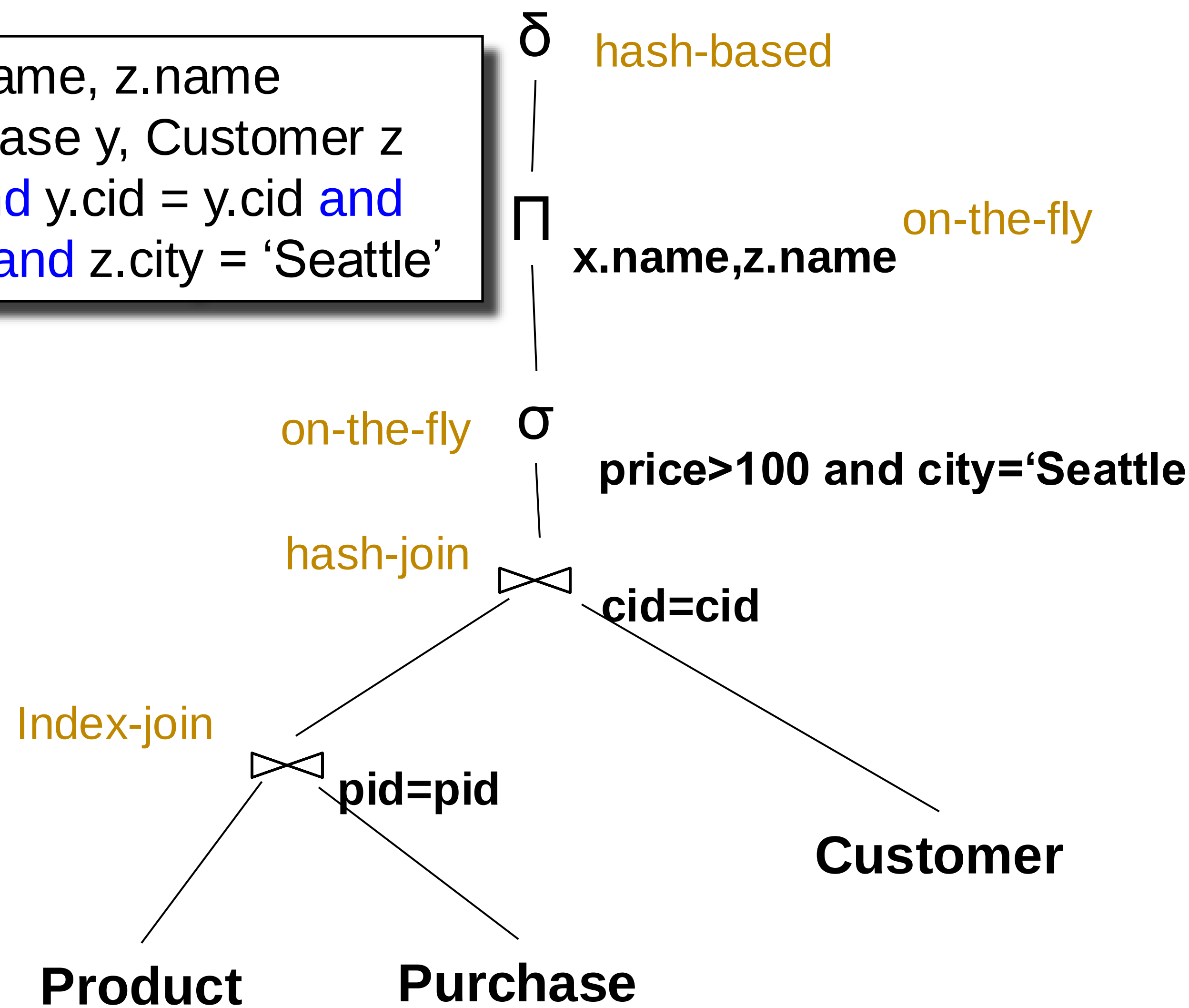


Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

Physical Query Plan

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

We say *What* we want



Says *How* to get it

Logical Data Independence

A View is a Relation defined by a SQL query

It can be used as any relation

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

View Example

View definition:

```sql
CREATE VIEW Big_Parts AS
        SELECT * FROM Part
        WHERE psize > 10;
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

View Example

View definition:

```
CREATE VIEW Big_Parts AS
        SELECT * FROM Part
        WHERE psize > 10;
```

Virtual table: `Big_Parts(pno,pname,psize,pcolor)`

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

View Example

View definition:

```
CREATE VIEW Big_Parts AS
        SELECT * FROM Part
        WHERE psize > 10;
```

Virtual table: `Big_Parts(pno,pname,psize,pcolor)`

Querying the view:

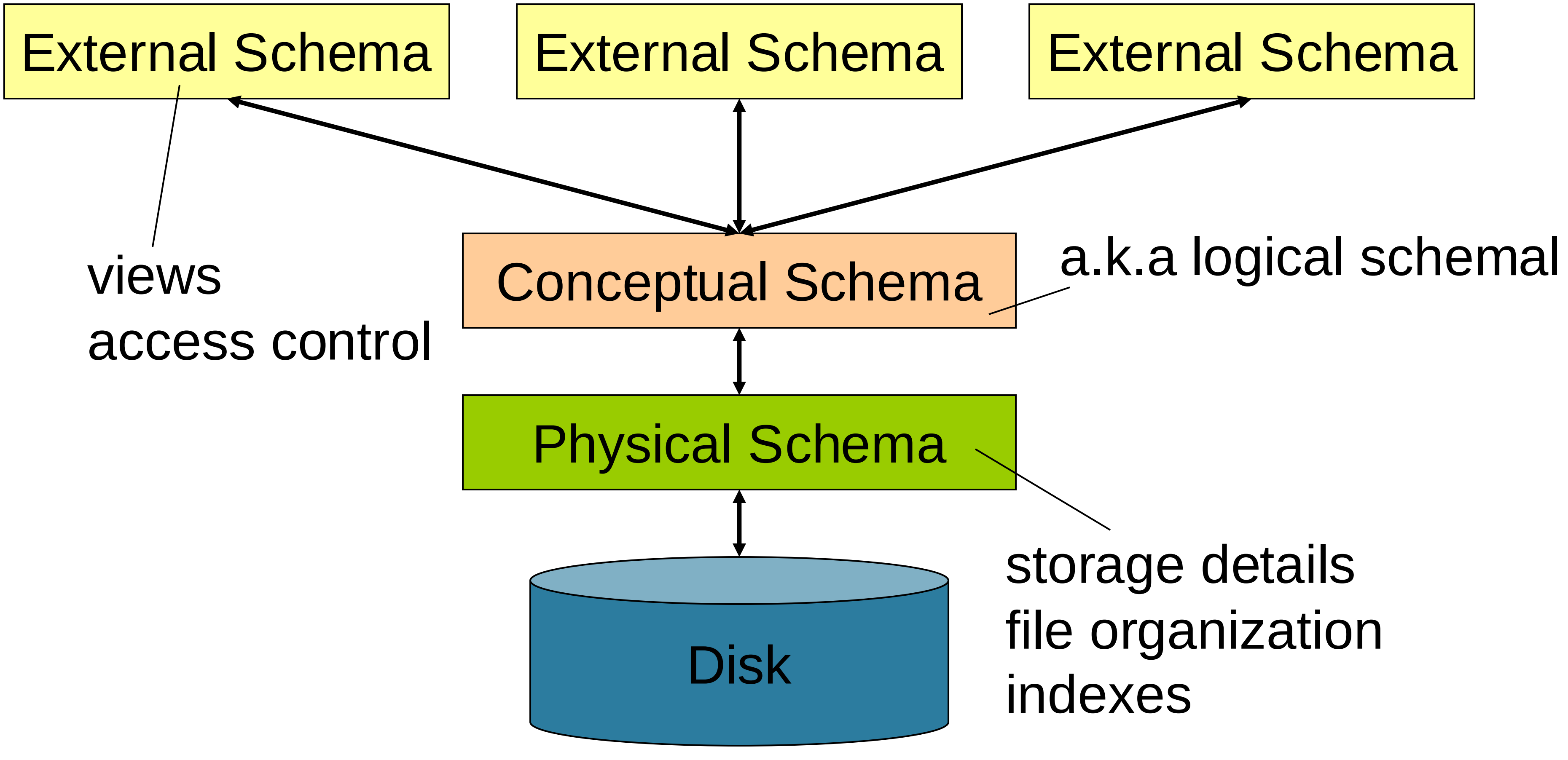
```
SELECT *
FROM Big_Parts
WHERE pcolor='blue';
```

Two Types of Views

- Virtual views:
  - Default in SQL, and what Stonebraker means in the paper
  - CREATE VIEW xyz AS …
  - Computed at query time

- Materialized views:
  - Some SQL engines support them
  - CREATE MATERIALIZED VIEW xyz AS
  - Computed at definition time

Levels of Abstraction



Recap: Data Independence

- **Physical data independence**: Applications are insulated from changes in **physical storage details**

- **Logical data independence**: Applications are insulated from changes to **logical structure of the data**

Outline

- Discuss Data Models
- SQL Beyond Relations

SQL Beyond Relations

- Sparse tensors
- Graphs
- Recursion

Sparse Tensors

- "Tensor" = a multidimensional array
  E.g.   t[i,j,k]

- A "sparse" tensor: many entries are 0

- Sparse tensors can naturally and efficiently be represented in SQL

Sparse Matrix

$$A = \begin{bmatrix} 5 & 0 & -2 \\ 0 & 0 & -1 \\ 0 & 7 & 0 \end{bmatrix}$$

How can we represent it as a relation?

Sparse Matrix

$$A = \begin{bmatrix} 5 & 0 & -2 \\ 0 & 0 & -1 \\ 0 & 7 & 0 \end{bmatrix}$$

Row	Col	Val
1	1	5
1	3	-2
2	3	-1
3	2	7

Matrix Multiplication in SQL

$C = A \cdot B$

Matrix Multiplication in SQL

$$C = A \cdot B$$

$$C_{ik} = \sum_j A_{ij} \cdot B_{jk}$$

Matrix Multiplication in SQL

$$C = A \cdot B$$

$$C_{ik} = \sum_{j} A_{ij} \cdot B_{jk}$$

```
SELECT A.row, B.col, sum(A.val*B.val) as val
FROM A, B
WHERE A.col = B.row
GROUP BY A.row, B.col;
```

Discussion

- Matrix multiplication = join + group-by
- Many operations can be written in SQL
- E.g. try at home: write in SQL

  $$Tr(A \cdot B \cdot C)$$

  where the trace is defined as:

  $$Tr(X) = \sum_i X_{ii}$$

- Surprisingly, $A + B$ is a bit harder…

Matrix Addition in SQL

$$C = A + B$$

Matrix Addition in SQL

$$C = A + B$$

```
SELECT A.row, A.col, A.val + B.val as val
FROM   A, B
WHERE  A.row = B.row and A.col = B.col
```

Matrix Addition in SQL

$$C = A + B$$

```
SELECT A.row, A.col, A.val + B.val as val
FROM   A, B
WHERE  A.row = B.row and A.col = B.col
```

Why is this wrong?

Solution 1: Outer Joins

$$C = A + B$$

```
SELECT



FROM A full outer join B ON A.row = B.row and A.col = B.col;
```

Solution 1: Outer Joins

$$C = A + B$$

```
SELECT



 (CASE WHEN A.val is null THEN 0 ELSE A.val END) +
 (CASE WHEN B.val is null THEN 0 ELSE B.val END)  as val
FROM A full outer join B ON A.row = B.row and A.col = B.col;
```

Solution 1: Outer Joins

$$C = A + B$$

```
SELECT
 (CASE WHEN A.row is null THEN B.row ELSE A.row END) as row,

 (CASE WHEN A.val is null THEN 0 ELSE A.val END) +
 (CASE WHEN B.val is null THEN 0 ELSE B.val END)  as val
FROM A full outer join B ON A.row = B.row and A.col = B.col;
```

Solution 1: Outer Joins

$$C = A + B$$

```
SELECT
 (CASE WHEN A.row is null THEN B.row ELSE A.row END) as row,
 (CASE WHEN A.col is null THEN B.col ELSE A.col END) as col,
 (CASE WHEN A.val is null THEN 0 ELSE A.val END) +
 (CASE WHEN B.val is null THEN 0 ELSE B.val END)  as val
FROM A full outer join B ON A.row = B.row and A.col = B.col;
```

Solution 2: Group By

$$C = A + B$$

?

Solution 2: Group By

$$C = A + B$$

```
SELECT m.row, m.col, sum(m.val)
FROM (SELECT * FROM A
            UNION ALL
         SELECT * FROM B) as m
GROUP BY m.row, m.col;
```

SQL Beyond Relations

- Sparse tensors
- Graphs
- Recursion

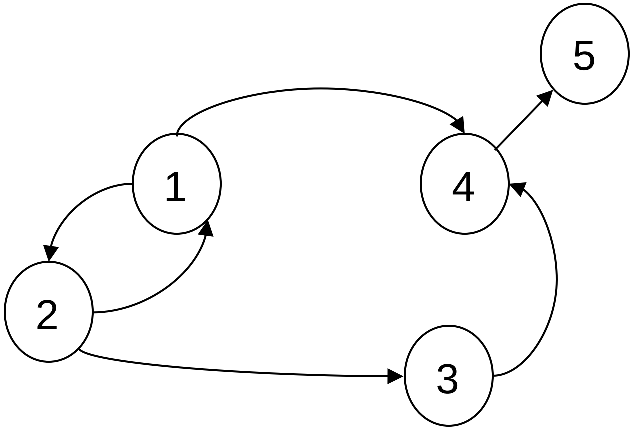
Graph Databases

- Graph databases systems: niche category specialized for large graphs
- Neo4J, Neptune, PathQueries,…
- SQL with Property Graphs: SQL/PGQ

Can use plain vanilla SQL for graphs

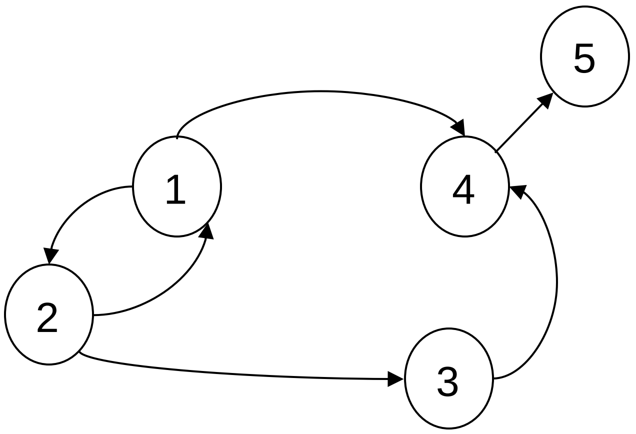
Graph Databases

A graph:



Graph Databases

A graph:



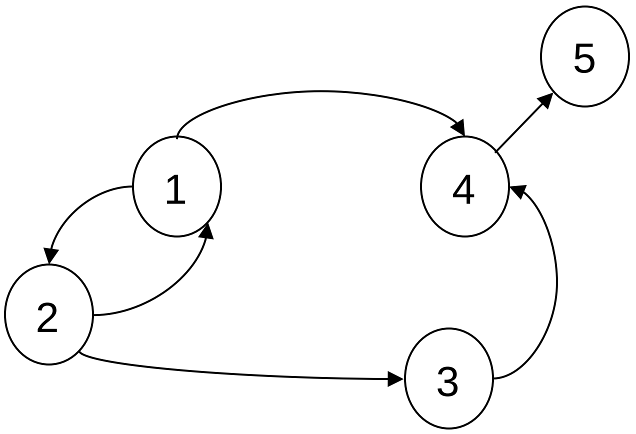
A relation:

Edge

src	dst
1	2
2	1
2	3
1	4
3	4
4	5

Graph Databases

A graph:



A relation:

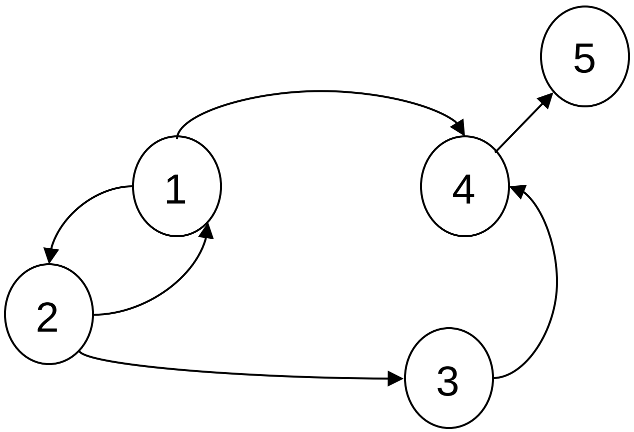
Edge

src	dst
1	2
2	1
2	3
1	4
3	4
4	5

Find nodes at distance 2: $\{(x,z) | \exists y\ Edge(x,y) \wedge Edge(y,z)\}$

Graph Databases

A graph:



A relation:

Edge

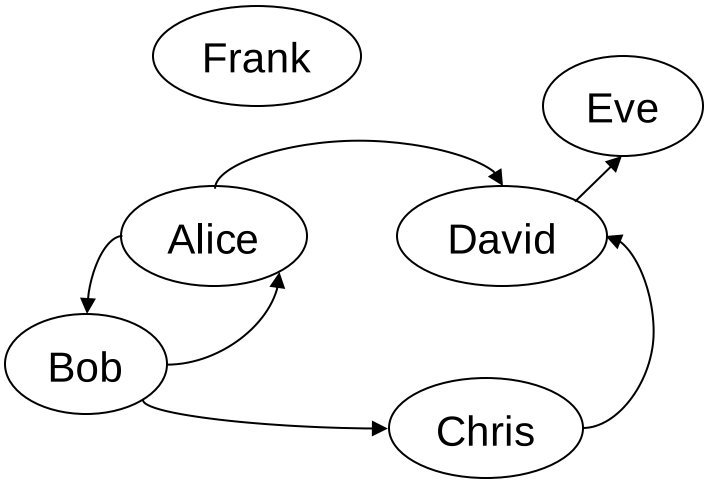
src	dst
1	2
2	1
2	3
1	4
3	4
4	5

Find nodes at distance 2: $\{(x, z) | \exists y\ Edge(x, y) \wedge Edge(y, z)\}$

```
SELECT DISTINCT e1.src as X, e2.dst as Z
FROM Edge e1, Edge e2
WHERE e1.dst = e2.src;
```

Other Representation

Representing nodes separately; needed for “isolated nodes” e.g. Frank



Node

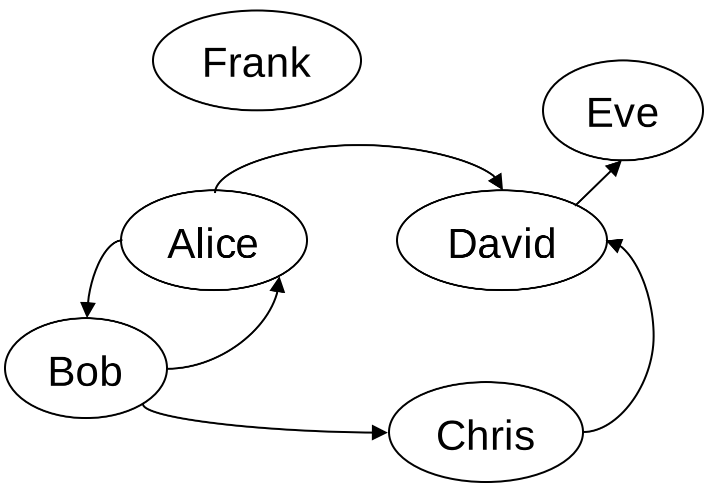
src
Alice
Bob
Chris
David
Eve
Frank

Edge

src	dst
Alice	Bob
Bob	Alice
Bob	Chris
Alice	David
Chris	David
David	Eve

Other Representation

Representing nodes separately; needed for “isolated nodes” e.g. Frank



Node

src
Alice
Bob
Chris
David
Eve
Frank

Edge

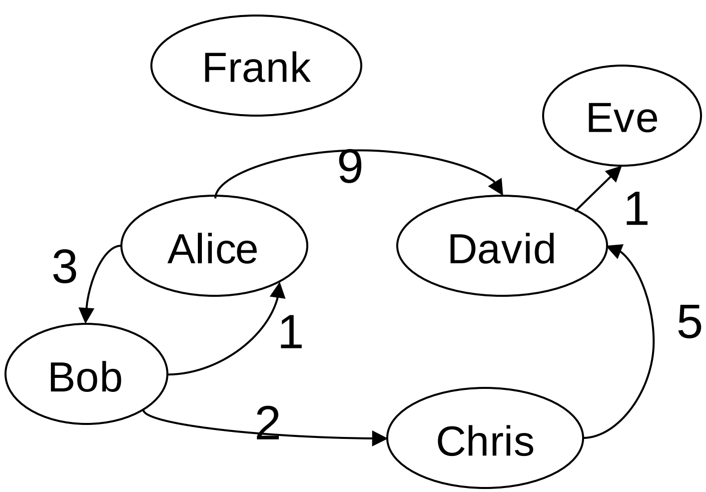
src	dst
Alice	Bob
Bob	Alice
Bob	Chris
Alice	David
Chris	David
David	Eve

Compute the number of children of each node

```
SELECT n.src, count(e.src)
FROM Node n
LEFT OUTER JOIN Edge e
WHERE n.src = e.src
GROUP BY n.src
```

Other Representation

Adding edge labels
Adding node labels…



Node

src
Alice
Bob
Chris
David
Eve
Frank

Edge

src	dst	weight
Alice	Bob	3
Bob	Alice	1
Bob	Chris	2
Alice	David	9
Chris	David	5
David	Eve	1

Discussion

- Graphs are naturally represented using relations
- SQL can be used for graph patters:
  - Pairs of nodes at distance 4
  - Triples (x,y,z) that form a triangle
  - Etc
- To find/travers paths, we need recursion. Next.

SQL Beyond Relations

- Sparse tensors
- Graphs
- Recursion

Recursion

- The SQL fragment we studied can be translated to Relational Algebra and optimized well
- This fragment is missing recursion
- We discuss the extension of SQL with recursion (SQL'99)

Warning

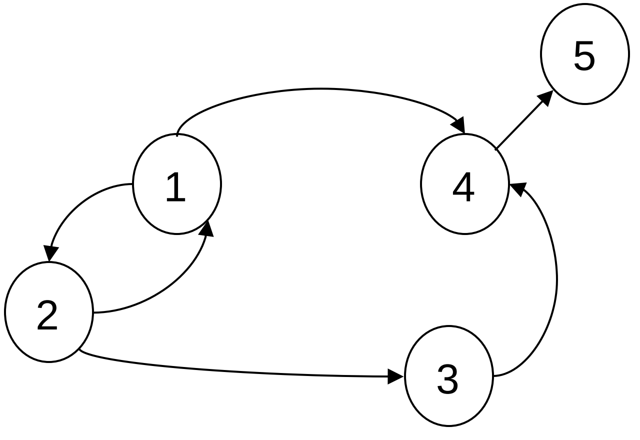
- SQL Recursion: notoriously bad design
- Right design: datalog (in a few weeks)
- Until then, fasten your seat-belts!

WITH RECURSIVE

```
WITH RECURSIVE TBL AS (
        SELECT…FROM…                    -- non-recursive rule
          UNION
        SELECT … FROM …[TBL]…)            -- recursive rule
SELECT … FROM …[TBL] …
```

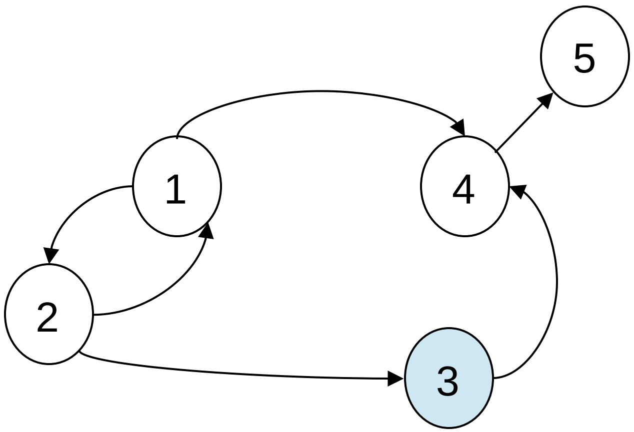
Example

Find all nodes x that have a path to node 3



Example

Find all nodes x that have a path to node 3

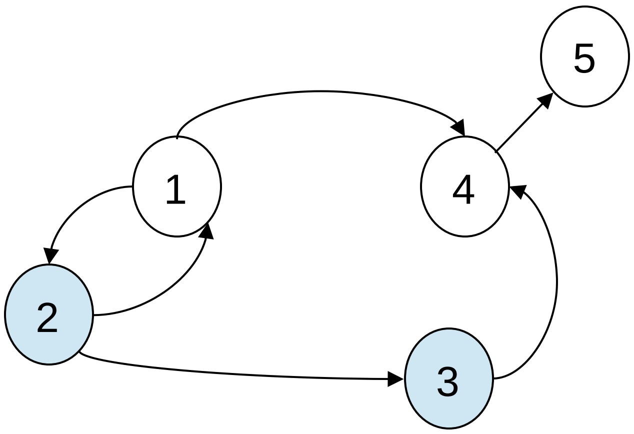


3 itself

```
SELECT 3 as v;
```

Example

Find all nodes x that have a path to node 3



3 itself

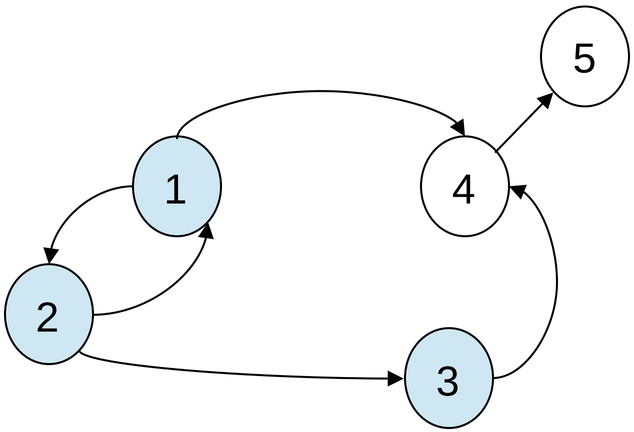
```
SELECT 3 as v;
```

Nodes connected to 3

```
… UNION
SELECT x.src as v
FROM Edge x
WHERE x.dst = 3;
```

Example

Find all nodes x that have a path to node 3



3 itself

```
SELECT 3 as v;
```

Nodes connected to 3

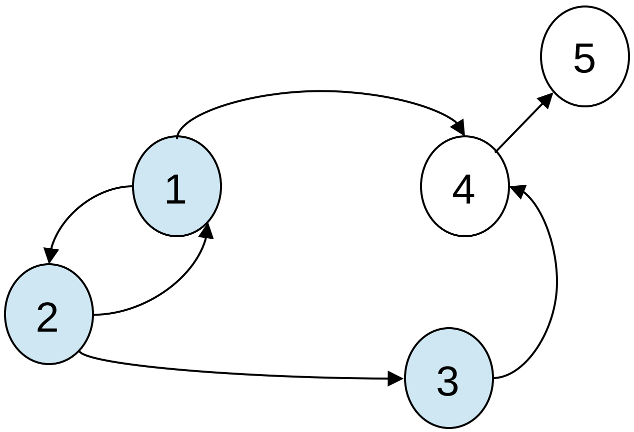
```
… UNION
SELECT x.src as v
FROM Edge x
WHERE x.dst = 3;
```

Nodes connected to them

```
… UNION
SELECT x.src as v
FROM Edge x, Edge y
WHERE x.dst=y.src
      and y.dst = 3;
```

Example

Find all nodes x that have a path to node 3



3 itself

```
SELECT 3 as v;
```

Nodes connected to 3

```
… UNION
SELECT x.src as v
FROM Edge x
WHERE x.dst = 3;
```

Nodes connected to them

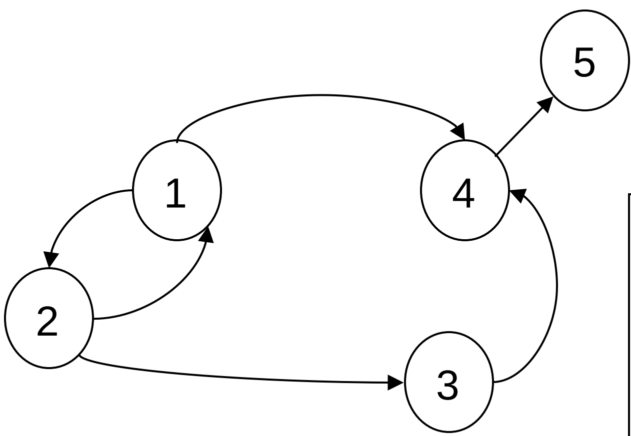
```
… UNION
SELECT x.src as v
FROM Edge x, Edge y
WHERE x.dst=y.src
      and y.dst = 3;
```

Cannot answer the query in the SQL fragment studied so far

. . .

Example

Find all nodes x that have a path to node 3



Can answer with recursion!

```
WITH RECURSIVE
  Answ AS (SELECT 3 as v
                  UNION
                 SELECT x.src as v
                 FROM Edge x, Answ a
                 WHERE x.dst = a.v)
SELECT * FROM Answ;
```

Semantics

- Recursive query is computed bottom-up

- Initially TBL = non-recursive query
- Repeatedly evaluate the recursive query, add new tuples to TBL
- Stop when no more change
- Finally, evaluate the main query

Semantics

```
WITH RECURSIVE TBL AS (
            SELECT…FROM…                       -- non-recursive rule
               UNION
            SELECT … FROM …[TBL]…)             -- recursive rule
SELECT … FROM …[TBL] …
```

ΔTBL_0 := SELECT…FROM… -- non-recursive rule

$TBL_0 := \Delta TBL_0$

$t := 0$

REPEAT $t := t+1$

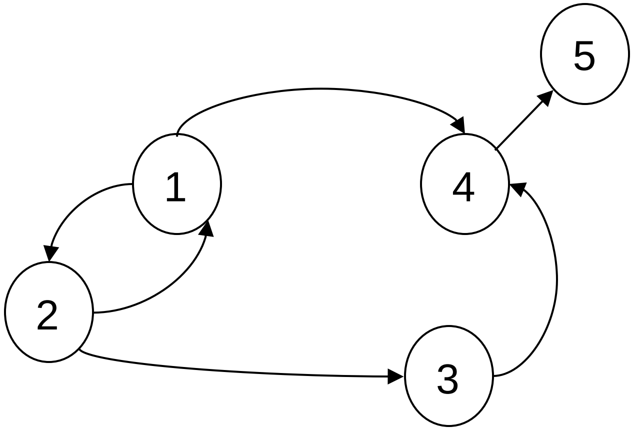
  ΔTBL_t := SELECT … FROM …[ΔTBL_{t-1}]… -- recursive rule

  $TBL_t := TBL_{t-1}$ UNION ΔTBL_t

UNTIL no more change

Example

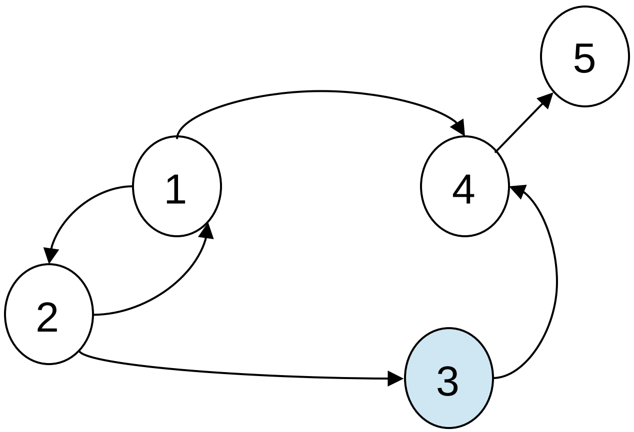
Find all nodes x that have a path to node 3




```
WITH RECURSIVE
   Answ AS (SELECT 3 as v
                  UNION
               SELECT x.src as v
               FROM Edge x, Answ a
               WHERE x.dst = a.v)
SELECT * FROM Answ;
```

Example

Find all nodes x that have a path to node 3

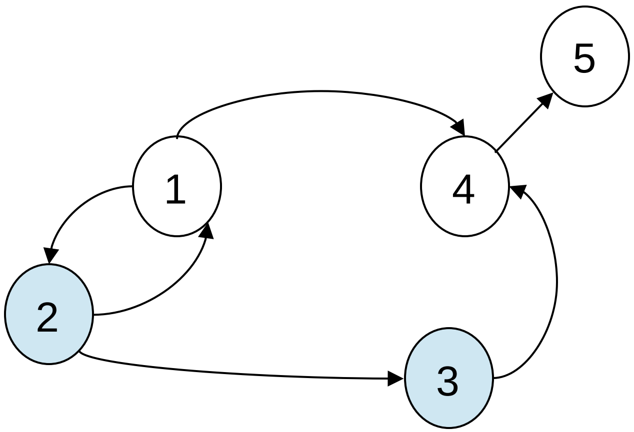



```
WITH RECURSIVE
  Answ AS (SELECT 3 as v
                  UNION
                SELECT x.src as v
                FROM Edge x, Answ a
                WHERE x.dst = a.v)
SELECT * FROM Answ;
```

t	$\Delta Answ_t$	$Answ_t$
0	3	3

Example

Find all nodes x that have a path to node 3

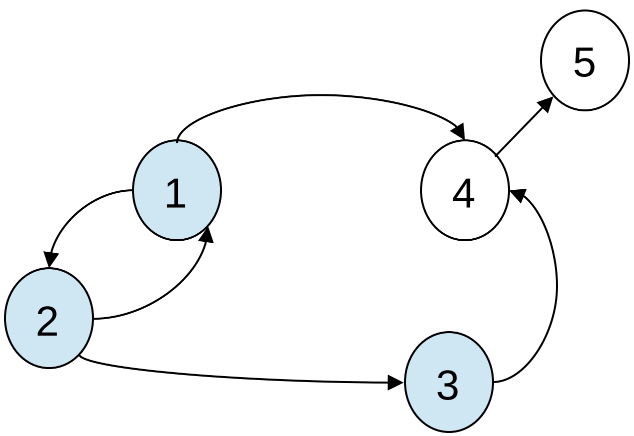



```
WITH RECURSIVE
  Answ AS (SELECT 3 as v
              UNION
           SELECT x.src as v
           FROM Edge x, Answ a
           WHERE x.dst = a.v)
SELECT * FROM Answ;
```

t	$\Delta Answ_t$	$Answ_t$
0	3	3
1	2	3, 2

Example

Find all nodes x that have a path to node 3

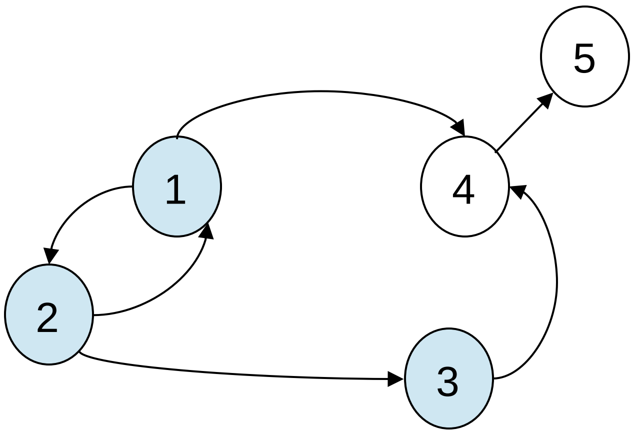



```
WITH RECURSIVE
   Answ AS (SELECT 3 as v
               UNION
             SELECT x.src as v
             FROM Edge x, Answ a
             WHERE x.dst = a.v)
SELECT * FROM Answ;
```

t	$\Delta Answ_t$	$Answ_t$
0	3	3
1	2	3, 2
2	1	3,2,1

Example

Find all nodes x that have a path to node 3




```
WITH RECURSIVE
  Answ AS (SELECT 3 as v
              UNION
           SELECT x.src as v
           FROM Edge x, Answ a
           WHERE x.dst = a.v)
SELECT * FROM Answ;
```

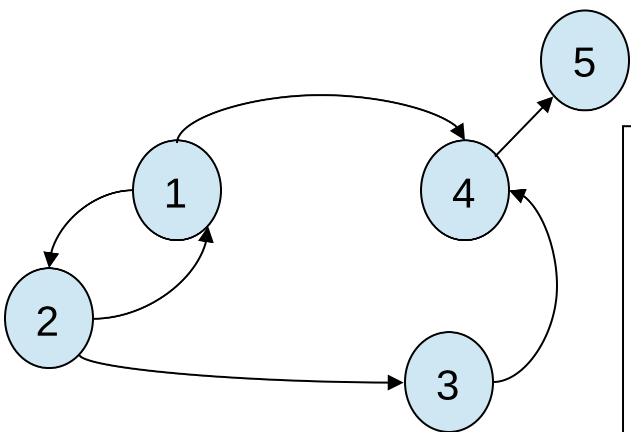
t	$\Delta Answ_t$	$Answ_t$
0	3	3
1	2	3, 2
2	1	3,2,1
3	2	3,2,1

Limitations

- Strict syntax:
  - One non-recursive rule
  - UNION one recursive rule
- May use UNION ALL, but that is often leads to non-termination
- No aggregates in the recursion
- Recursive relation may occur only once

Strict Syntax

Find all nodes x that have undirected a path to 3




```
WITH RECURSIVE
  Answ AS (SELECT 3 as v
                  UNION
                    SELECT x.src as v
                    FROM Edge x, Answ a
                    WHERE x.dst = a.v
                  UNION
                    SELECT x.dst as v
                    FROM Answ a, Edge x
                    WHERE a.v = x.src
)
SELECT * FROM Answ;
```

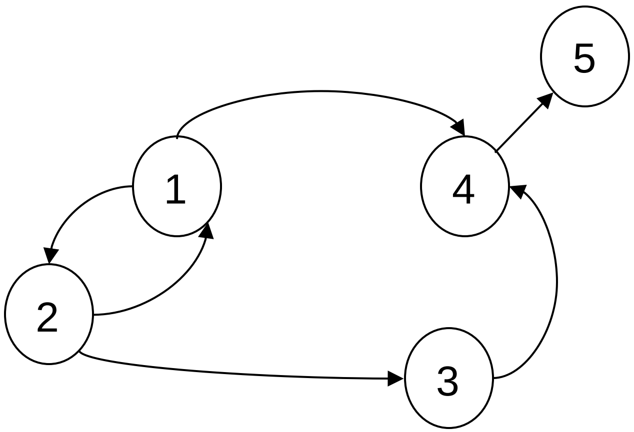
Backwards

Forward

Syntax Error

Union All is Dangerous

Find all nodes x that have a path to node 3



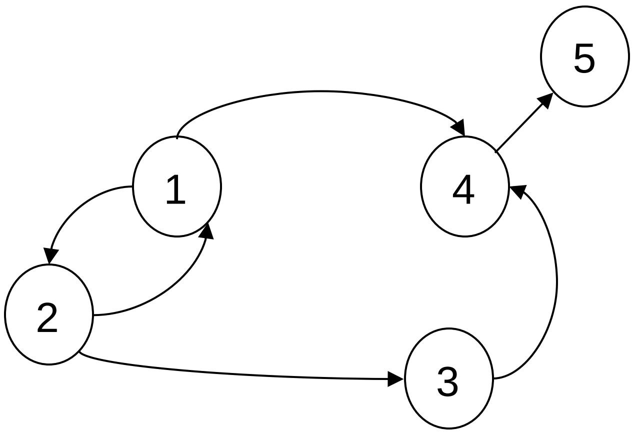
t	$\Delta Answ_t$	$Answ_t$
0	3	3
1	2	3, 2
2	1	3,2,1
3	2	3,2,1,2
4	1	3,2,1,2,1
5	2	3,2,1,2,1,2

```
WITH RECURSIVE
  Answ AS (SELECT 3 as v
                  UNION ALL
                SELECT x.src as v
                FROM Edge x, Answ a
                WHERE x.dst = a.v)
SELECT * FROM Answ;
```

Does not terminate

No Aggregates in Recursion

Find all nodes x find the shortest path to node 3



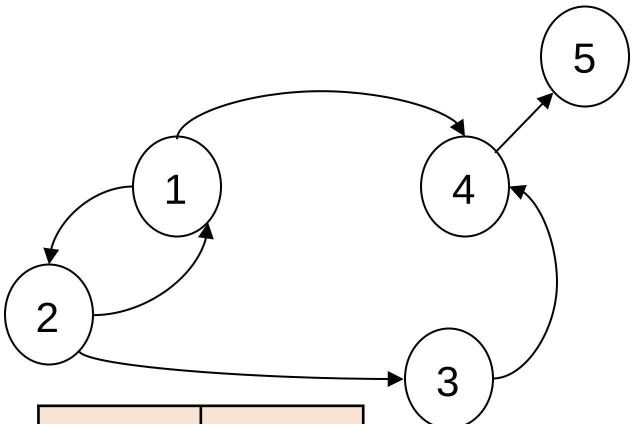
v	l
3	0
2	1
1	2

```
WITH RECURSIVE
   Answ AS (SELECT 3 as v, 0 as l
                   UNION
            SELECT x.src as v, 1+a.l as l
            FROM Edge x, Answ a
            WHERE x.dst = a.v
                   and a.l < (SELECT count(*)
                              FROM Edge))
SELECT v, min(l) as l
FROM Answ
GROUP BY l;
```

Debugging

Debugging

Find all nodes x that have a path to node 3




```
WITH RECURSIVE
  Answ AS (SELECT 3 as v, 0 as t
                 UNION
               SELECT x.src as v, a.t+1 as t
               FROM Edge x, Answ a
               WHERE x.dst = a.v and a.t<5)
SELECT * FROM Answ ORDER BY t;
```

v	t
3	0
2	1
1	2
2	3
1	4
2	5

t	$\Delta Answ_t$	$Answ_t$
0	3	3
1	2	3, 2
2	1	3,2,1
3	2	3,2,1
4	1	3,2,1
…		

Knight's (Shortest) Path

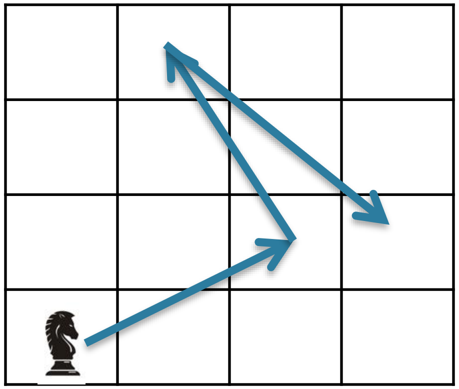
- Given a chess board, check which positions can the knight reach starting from the bottom-left position
- Variations:
  - The board is m x n, for various m,n
  - The board has obstructions
  - We may want to also compute the length of the shortest path

Knight's (Shortest) Path

Graph:
 vertices = board

Board

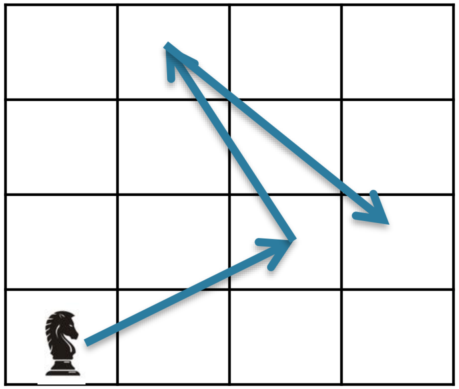
x	y
1	1
1	2
…	…
2	1
…	…
10	10



Knight's (Shortest) Path

Graph:
  vertices = board
  edges = (+2,+1), (+2,-1), …

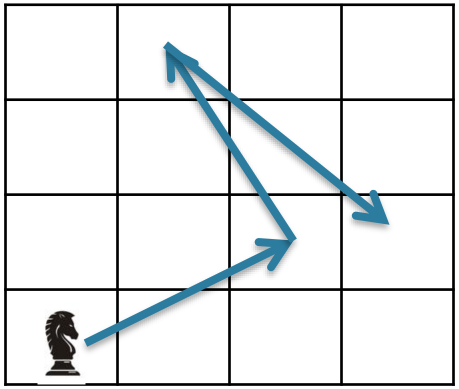
```
create table board as
  select x as x, y as y
  from generate_series(1,10) x,
       generate_series(1,10) y;
```



Knight's (Shortest) Path

Graph:
  vertices = board
  edges = …

```
create table board as
  select x as x, y as y
  from generate_series(1,10) x,
       generate_series(1,10) y;
```



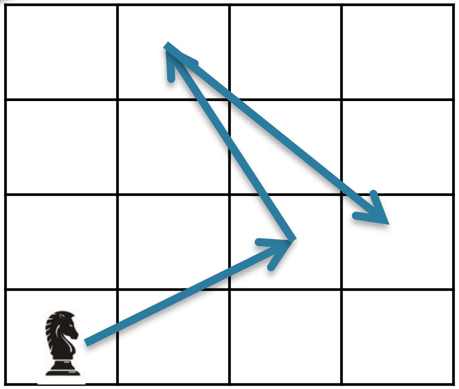
Edge

xsrc	ysrc	xdst	ydst
1	1	3	2
1	1	2	3
…	…		

Knight's (Shortest) Path

Better:
use delta's

Graph:
  vertices = board
  edges = (+2,+1), (+2,-1), …

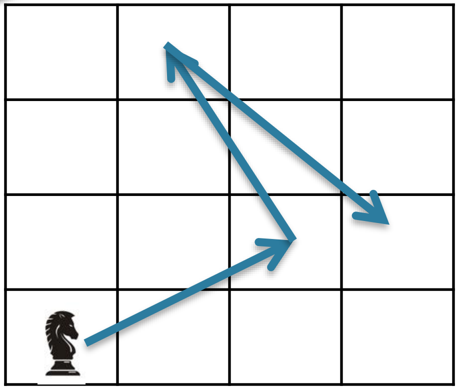


```
create table board as
  select x as x, y as y
  from generate_series(1,10) x,
       generate_series(1,10) y;
```

Knight’s (Shortest) Path

Better:
use delta’s

Graph:
  vertices = board
  edges = (+2,+1), (+2,-1), …



```
create table board as
  select x as x, y as y
  from generate_series(1,10) x,
       generate_series(1,10) y;
```

```
create table move (dx int, dy int);
insert into move values
  (1,2), (2,1), (-1,2), (2,-1),
  (1,-2), (-2,1), (-1,-2), (-2,-1);
```

Knight's (Shortest) Path

Compute all positions reachable from (1,1) on a 5 x 5 board
(The answer is boring: all of them. But we extend next.)

Knight's (Shortest) Path

Compute all positions reachable from (1,1) on a 5 x 5 board
(The answer is boring: all of them. But we extend next.)

```
with recursive reach as
  (



                                        )
select r.x, r.y
from reach r
order by r.x, r.y;
```

Knight's (Shortest) Path

Compute all positions reachable from (1,1) on a 5 x 5 board
(The answer is boring: all of them. But we extend next.)

```
with recursive reach as
  (select 1 as x, 1 as y
    union



                                                    )
select r.x, r.y
from reach r
order by r.x, r.y;
```

Knight's (Shortest) Path

Compute all positions reachable from (1,1) on a 5 x 5 board
(The answer is boring: all of them. But we extend next.)

```
with recursive reach as
  (select 1 as x, 1 as y
     union
    select                  as x,                 as y
    from reach r, move m
    where
                                                      )
select r.x, r.y
from reach r
order by r.x, r.y;
```

Knight's (Shortest) Path

Compute all positions reachable from (1,1) on a 5 x 5 board
(The answer is boring: all of them. But we extend next.)

```
with recursive reach as
  (select 1 as x, 1 as y
     union
   select                 as x,                 as y
   from reach r, move m
   where 1 <= r.x + m.dx and r.x + m.dx <= 5
     and 1 <= r.y + m.dy and r.y + m.dy <= 5)
select r.x, r.y
from reach r
order by r.x, r.y;
```

Knight's (Shortest) Path

Compute all positions reachable from (1,1) on a 5 x 5 board
(The answer is boring: all of them. But we extend next.)

```
with recursive reach as
  (select 1 as x, 1 as y
     union
   select r.x + m.dx as x, r.y + m.dy as y
   from reach r, move m
   where 1 <= r.x + m.dx and r.x + m.dx <= 5
     and 1 <= r.y + m.dy and r.y + m.dy <= 5)
select r.x, r.y
from reach r
order by r.x, r.y;
```

Knight’s (Shortest) Path

For n=2,10, check if the knight can reach the top-right position on an n x n board

Knight's (Shortest) Path

For n=2,10, check if the knight can reach the top-right position on an n x n board

```
create table n as
 select n as n
 from generate_series(2,10) n;
```

Knight's (Shortest) Path

For n=2,10, check if the knight can reach the top-right position on an n x n board

```
create table n as
 select n as n
 from generate_series(2,10) n;
```

```
with recursive reach as
   (select                1 as x, 1 as y
     union
    select        r.x + m.dx as x, r.y + m.dy as y
    from       reach r, move m
    where
           1 <= r.x + m.dx and r.x + m.dx <= n.n
     and 1 <= r.y + m.dy and r.y + m.dy <= n.n)
select
```

Knight's (Shortest) Path

For n=2,10, check if the knight can reach the top-right position on an n x n board

```
create table n as
 select n as n
 from generate_series(2,10) n;
```

```
with recursive reach as
   (select n.n as n, 1 as x, 1 as y from n n
     union
    select       r.x + m.dx as x, r.y + m.dy as y
    from        reach r, move m
    where
            1 <= r.x + m.dx and r.x + m.dx <= n.n
      and 1 <= r.y + m.dy and r.y + m.dy <= n.n)
select
```

Knight's (Shortest) Path

For n=2,10, check if the knight can reach the top-right position on an n x n board

```
create table n as
 select n as n
 from generate_series(2,10) n;
```

```
with recursive reach as
   (select n.n as n, 1 as x, 1 as y from n n
     union
    select n.n, r.x + m.dx as x, r.y + m.dy as y
    from n n, reach r, move m
    where n.n = r.n
      and 1 <= r.x + m.dx and r.x + m.dx <= n.n
      and 1 <= r.y + m.dy and r.y + m.dy <= n.n)
select
```

Knight's (Shortest) Path

For n=2,10, check if the knight can reach the top-right position on an n x n board

```
create table n as
 select n as n
 from generate_series(2,10) n;
```

```
with recursive reach as
   (select n.n as n, 1 as x, 1 as y from n n
     union
    select n.n, r.x + m.dx as x, r.y + m.dy as y
    from n n, reach r, move m
    where n.n = r.n
      and 1 <= r.x + m.dx and r.x + m.dx <= n.n
      and 1 <= r.y + m.dy and r.y + m.dy <= n.n)
select r.n
from reach r
where r.x=r.n and r.y = r.n
order by r.n;
```

Knight's (Shortest) Path

For each position find the shortest path from (1,1), on a 5x5 board

```
with recursive reach as
  (select 1 as x, 1 as y, 0 as l
     union
    select r.x + m.dx as x, r.y + m.dy as y, r.l+1 as l
    from reach r, move m
    where 1 <= r.x + m.dx and r.x + m.dx <= 5
     and 1 <= r.y + m.dy and r.y + m.dy <= 5
     and r.l <= 25)
select r.x, r.y, min(r.l)
from reach r
group by r.x, r.y
order by r.x, r.y;
```

Knight’s (Shortest) Path

Now the board has obstructions show in the file board.csv

```
10,---X-X--X-
09,--X---X---
08,---X------
07,----X--X--
06,----------
05,X--X----X-
04,-X-X--X---
03,--X-X-XX--
02,-X-----X--
01,---X------
```

Knight's (Shortest) Path

Now the board has obstructions show in the file board.csv

```
10,---X-X--X-
09,--X---X---
08,---X------
07,----X--X--
06,----------
05,X--X----X-
04,-X-X--X---
03,--X-X-XX--
02,-X-----X--
01,---X------
```

```
create table board_raw (row int, cols text);
copy board_raw
from '……/board.csv'
delimiter ',';
```

Knight’s (Shortest) Path

Now the board has obstructions show in the file board.csv

```
10,---X-X--X-
09,--X---X---
08,---X------
07,----X--X--
06,----------
05,X--X----X-
04,-X-X--X---
03,--X-X-XX--
02,-X-----X--
01,---X------
```

```
create table board_raw (row int, cols text);
copy board_raw
from ‘……/board.csv'
delimiter ',';
```

```
create table board as
  (select b.row as x, y as y
   from board_raw b,
         generate_series(1,length(b.cols)) as y
   where substr(b.cols, y, 1) = '-');
```

Knight’s (Shortest) Path

Now the board has obstructions show in the file board.csv

```
10,---X-X--X-
09,--X---X---
08,---X------
07,----X--X--
06,----------
05,X--X----X-
04,-X-X--X---
03,--X-X-XX--
02,-X-----X--
01,---X------
```

```
create table board_raw (row int, cols text);
copy board_raw
from ‘……/board.csv'
delimiter ',';
```

String functions: look them up

```
create table board as
  (select b.row as x, y as y
   from board_raw b,
        generate_series(1,length(b.cols)) as y
   where substr(b.cols, y, 1) = '-');
```

Knight's (Shortest) Path

Now the board has obstructions show in the file board.csv

```
10,---X-X--X
09,--X---X--
08,---X-----
07,----X--X-
06,---------
05,X--X----X
04,-X-X--X--
03,--X-X-XX-
02,-X-----X-
01,---X-----
```

```
with recursive
  mrows as (select max(b.x) as m from board b),
  ncols as (select max(b.y) as n from board b),

select
```

Size of the board

Knight’s (Shortest) Path

Now the board has obstructions show in the file board.csv

```
10,---X-X--X
09,--X---X--
08,---X-----
07,----X--X-
06,---------
05,X--X----X
04,-X-X--X--
03,--X-X-XX-
02,-X-----X-
01,---X-----
```

```
with recursive
   mrows as (select max(b.x) as m from board b),
   ncols as (select max(b.y) as n from board b),
   reach as
   (select 1 as x, 1 as y
     union
    select r.x + m.dx as x, r.y + m.dy as y
    from                    reach r, move m, mrows mr, ncols nc
    where 1 <= r.x + m.dx and r.x + m.dx <= mr.m
     and 1 <= r.y + m.dy and r.y + m.dy <= nc.n
                                                            )
select
```

Knight's (Shortest) Path

Now the board has obstructions show in the file board.csv

```
10,
09
08,
07,--
06,---------
05,X--X----X
04,-X-X--X--
03,--X-X-XX-
02,-X-----X-
01,---X-----
```

Check that destination is not obstructed

```
recursive
  as (select max(b.x) as m from board b),
  s (select max(b.y) as n from board b),
  as
  lect 1 as x, 1 as y
  union
  select r.x + m.dx as x, r.y + m.dy as y
  from board dest, reach r, move m, mrows mr, ncols nc
  where 1 <= r.x + m.dx and r.x + m.dx <= mr.m
    and 1 <= r.y + m.dy and r.y + m.dy <= nc.n
    and r.x + m.dx = dest.x and r.y + m.dy = dest.y)
select
```

Knight's (Shortest) Path

Now the board has obstructions show in the file board.csv

```
10,---X-X--X
09,--X---X--
08,---X-----
07,----X--X-
06,---------
05,X--X----X
04,-X-X--X--
03,--X-X-XX-
02,-X-----X-
01,---X-----
```

```
with recursive
   mrows as (select max(b.x) as m from board b),
   ncols as (select max(b.y) as n from board b),
   reach as
   (select 1 as x, 1 as y
      union
    select r.x + m.dx as x, r.y + m.dy as y
    from board dest, reach r, move m, mrows mr, ncols nc
    where 1 <= r.x + m.dx and r.x + m.dx <= mr.m
      and 1 <= r.y + m.dy and r.y + m.dy <= nc.n
      and r.x + m.dx = dest.x and r.y + m.dy = dest.y)
select
```

Knight's (Shortest) Path

Now the board has obstructions show in the file board.csv

```
10,---X-X--X
09,--X---X--
08,---X-----
07,----X--X-
06,---------
05,X--X----X
04,-X-X--X--
03,--X-X-XX-
02,-X-----X-
01,---X-----
```

```
with recursive
   mrows as (select max(b.x) as m from board b),
   ncols as (select max(b.y) as n from board b),
   reach as
   (select 1 as x, 1 as y
      union
    select r.x + m.dx as x, r.y + m.dy as y
    from board dest, reach r, move m, mrows mr, ncols nc
    where 1 <= r.x + m.dx and r.x + m.dx <= mr.m
      and 1 <= r.y + m.dy and r.y + m.dy <= nc.n
      and r.x + m.dx = dest.x and r.y + m.dy = dest.y)
select r.x, r.y
from reach r;
```

Summary

- Although limited, recursion increases the expressive power of SQL
- HW2 asks you to solve several puzzles by using vanilla SQL; some (but not all) puzzles require recursion