

Extracting Queries by Static Analysis of Transparent Persistence

Ben Wiedermann and William R. Cook
The University of Texas at Austin

Databases

Programming Languages

Databases



Programming Languages

Databases

Print name and manager's name
of every employee
whose salary > \$65,000

Programming Languages

Database APIs (JDBC, etc)

Print name and manager's name
of every employee
whose salary > \$65,000

Transparent Persistence

```
String query = "from Employee e  
left join fetch e.manager  
where e.salary > 65000";  
List result = session.createQuery(query);  
for (Employee e : result.list()) {  
    print(e.name + e.manager.name);  
}
```

Query string

Runtime type errors

Hard to parameterize

Programmer burden

Subtle dependency

Programmer burden

```
String query = "from Employee e
               left join fetch e.manager
               where e.salary > 65000";
List result = session.createQuery(query);
for (Employee e : result.list()) {
    print(e.name + e.manager.name);
}
```

Send to database

Optimized search

Good performance

Transparent persistence

Static typing

Parameterization

No programmer burden

```
for (Employee e : root.employees) {  
    if (e.salary > 65000) {  
        print (e.name + e.manager.name);  
    }  
}
```

Linear search

“Record-at-a-time”

No DB optimization

Poor performance

```
for (Employee e : root.employees) {  
    if (e.salary > 65000) {  
        print (e.name + e.manager.name);  
    }  
}
```

Database APIs

Transparent Persistence

- Not true integration
 - Not type-safe
 - Burden on programmer
 - Good performance
-

- Better integration
- Type safe
- Relieves programmer burden
- Poor performance

Query Extraction

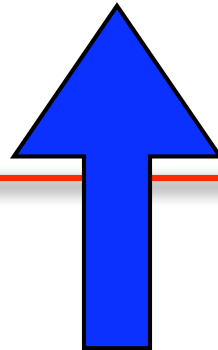
- Good performance

&

- Better integration
- Type safe
- Relieves programmer burden

```
String query = "from Employee e
                left join fetch e.manager
                where e.salary > 65000";
List result = session.createQuery(query);
for (Employee e : result.list()) {
    print(e.name + e.manager.name);
}
```

Query Extraction

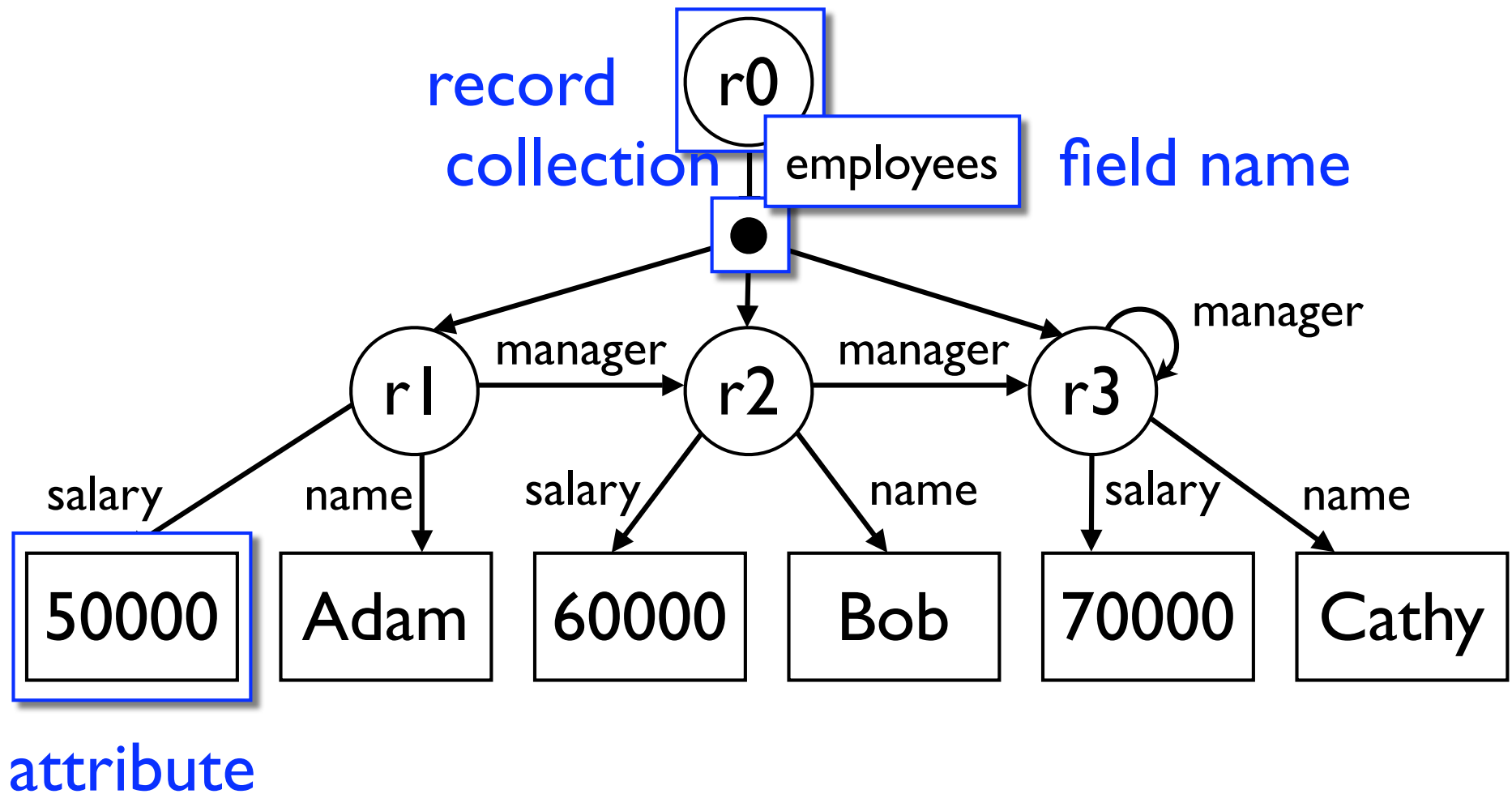


```
for (Employee e : root.employees) {
    if (e.salary > 65000) {
        print (e.name + e.manager.name);
    }
}
```

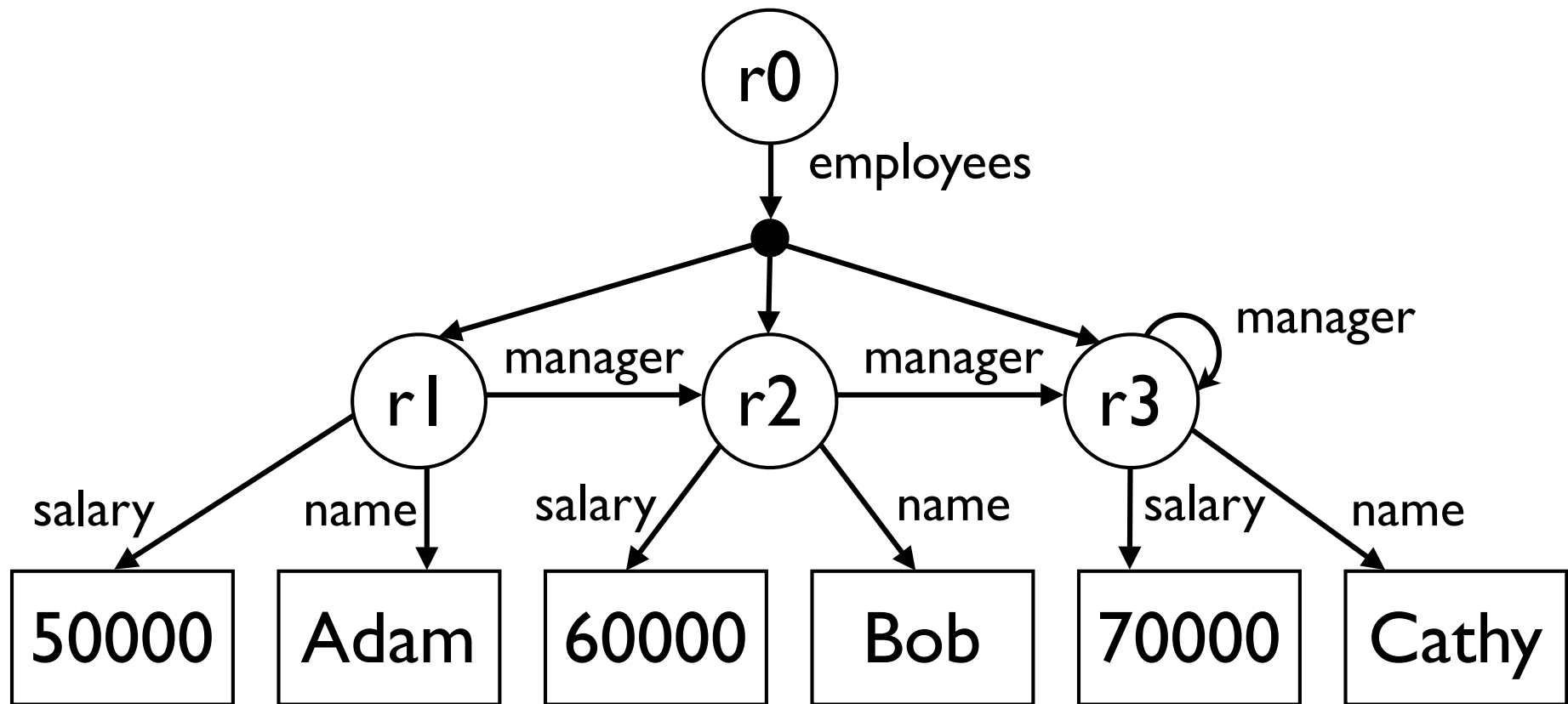
Approach

- Identify subset of the database used by program
 - Traversals from root define shape of query
 - Identify conditions under which data is used
- Current Assumptions
 - Program = transaction, one root access
 - Query result has same structure as database

Object View of Database



Object View of Database



Retrieve subgraph program requires

Simple Study Language

- Transparent persistence
 - Access through variable `root`
- Imperative
 - `x := y + z;`
 - No persistent updates
- Iteration of persistent collections
 - `for e in root.employees {...}`
- No procedures

Extracting Traversal Paths

```
for (Employee e : root.employees) {  
    if (e.salary > 65000) {  
        print (e.name + e.manager.name);  
    }  
}
```

employees

employees.l

employees.l.salary

employees.l.name

employees.l.manager

employees.l.manager.name

Abstract Interpretation

- Run program on “abstract data” instead of real data
 - Example: replace integers with {Neg, Zero, Pos}
 - In our case: replace **data** (v) with **paths** (π)
- Summarize possible executions by merging
 - conditional branches
 - loops

Over-approximation

employees

employees.t

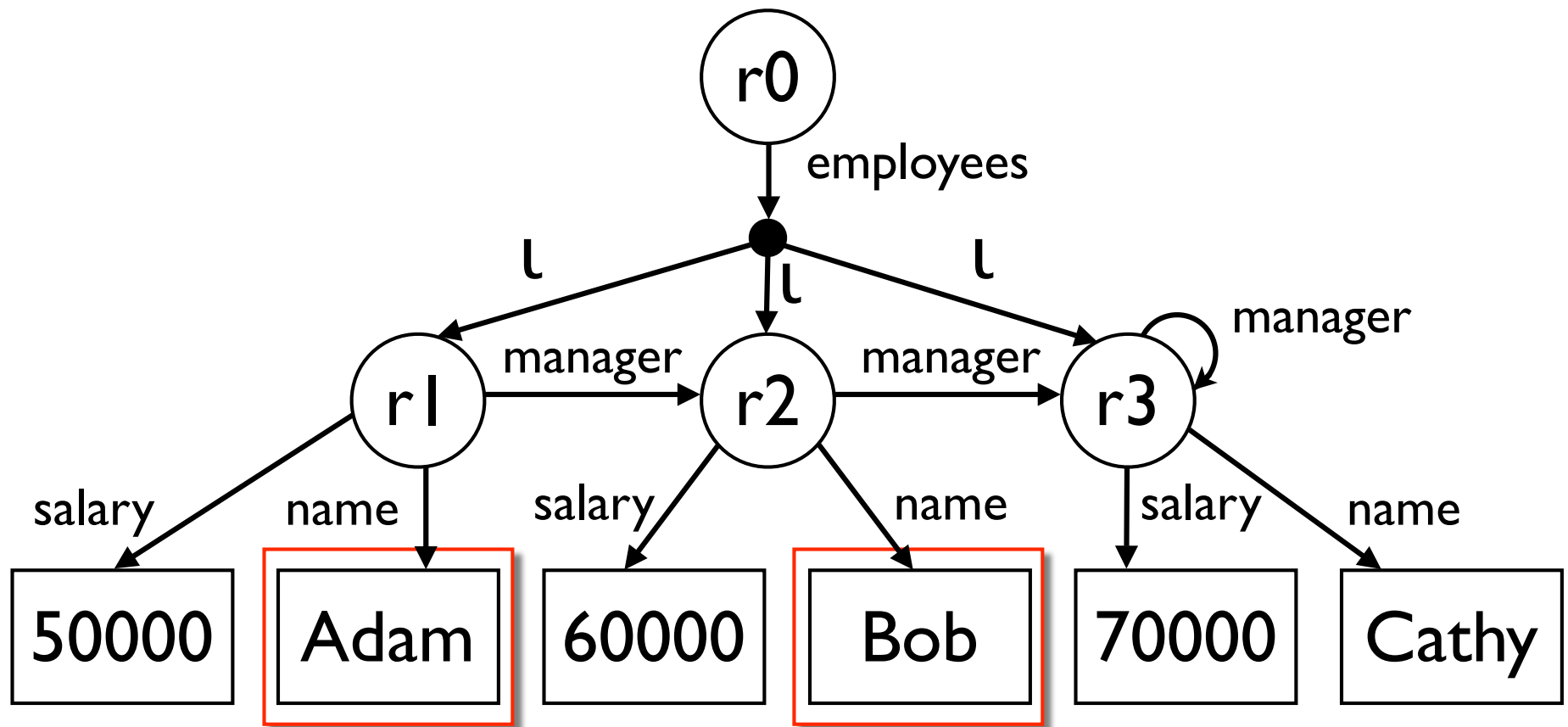
employees.t.salary

employees.t.name

employees.t.manager

employees.t.manager.name

Over-approximation



Need more precise approximation

Conditional Paths

```
for (Employee e : root.employees) {  
C= if (e.salary > 65000) {  
    print (e.name + e.manager.name);  
}}  
}
```

C

Query Condition Restrictions

- Condition is an expression over paths
- Database can execute operations
- Database can evaluate condition
 - Independent of collection order
 - Loop nesting mirrors relationships

Order Independence: 1 Loop

for l_1 in p
 $x := E[l_1]$;
 if $C[l_1, x]$ then S ;

for l_1 in p
 $x := E[l_1] + x$;
 if $C[l_1, x]$ then S ;

No loop-carried dependence

Order Independence: >1 Loop

```
for  $l_1$  in  $p$ 
  if  $C_1[l_1]$  then  $S_1$ ;
for  $l_2$  in  $p$ 
  if  $C_2[l_2]$  then  $S_2$ ;
```

```
for  $l_1$  in  $p$ 
  if  $C_1[l_1]$  then
     $x := E[l_1]$ ;
  for  $l_2$  in  $p$ 
    if  $C_2[l_2, x]$  then  $S$ ;
```

Refer to at most 1 element from a given iteration

Loop Nesting Mirrors Relationships

```
for l1 in p  
  for l2 in l1.f  
    if C[l1, l2] then S;
```

```
for l1 in p1  
  for l2 in p2  
    if C[l1, l2] then S;
```

Nested iterations extend container paths

Abstract Interpretation

- Domain: Path \times Condition
- Field traversal generates new path(s)
- Merge conditional branches
- Merge assignments
- Attach query conditions to paths

Example

```
for (Employee e : root.employees) {  
    if (e.salary > 65000) {  
        print (e.name + e.manager.name);  
    }  
}
```

employees	employees.l.name [C]
employees.l	employees.l.manager [C]
employees.l.salary	employees.l.manager.name [C]

C = employees.l.salary > 65000

Control / Data Flow

```
for (Employee e : root.employees) {  
    if (e.salary > 65000) {  
        print (e.name + e.manager.name);  
    }  
}
```

Control

Data

employees

employees.l

employees.l.salary

employees.l.name [C]

employees.l.manager [C]

employees.l.manager.name [C]

$C = \text{employees.l.salary} > 65000$

Abstract Interpretation

- Domain: Path x Condition x Dependence
- Field traversal generates new path(s)
- Attach query conditions to paths
- Merge conditional branches
- Merge assignments
- Record data dependence

Data Dependence in Loops

```
for e in p
  if C
    x:=x+1;
```

- Data dependence on p
- Attach condition C to p

Example

```
for (Employee e : root.employees) {  
    if (e.salary > 65000) {  
        print (e.name + e.manager.name);  
    }  
}
```

employees

employees.l.name [C]★

employees.l

employees.l.manager [C]★

employees.l.salary

employees.l [C]★

employees.l.manager.name [C]★

C = employees.l.salary > 65000

Query Creation

- Query retrieves subgraph
- Query must express structure
- Object Query Language (OQL)

```
employees                employees.l.name [C]★  
employees.l             employees.l.manager [C]★  
employees.l.salary      employees.l.manager.name [C]★  
employees.l [C]★
```

C = employees.l.salary > 65000

Query Creation

```
struct (employees = (  
  select struct (salary = e.salary,  
                name = e.name,  
                manager = struct(name =  
                                e.manager.name))  
  from employees as e  
  where e.salary > 65000)
```

```
employees                employees.l.name [C]★  
employees.l              employees.l.manager [C]★  
employees.l.salary       employees.l.manager.name [C]★  
employees.l [C]★
```

```
C = employees.l.salary > 65000
```

Program Creation

```
qs = " struct (employees = (  
    select struct (salary = e.salary,  
                  name = e.name,  
                  manager = struct(name =  
                                e.manager.name) )  
    from employees as e  
    where e.salary > 65000)"  
  
result = session.executeQuery(qs);  
  
for (Employee e : root.employees) {  
    if (e.salary > 65000) {  
        print (e.name + e.manager.name);  
    }  
}
```

Program Creation

```
qs = " struct (employees = (  
    select struct (salary = e.salary,  
                  name = e.name,  
                  manager = struct(name =  
                                e.manager.name) )  
    from employees as e  
    where e.salary > 65000)"  
  
result = session.executeQuery(qs);  
  
for (Employee e : result.employees) {  
    if (e.salary > 65000) {  
        print (e.name + e.manager.name);  
    }  
}
```

Related Work

- Shape Analysis for Data Access
 - [Vitenberg, Kvilekval, and Singh \[ECOOP04\]](#)
 - [Kvilekval and Singh \[DOA04\]](#)
- Queries as First-Class Program Values
 - [Bierman, Meijer, and Schulte \[ECOOP05\]](#)
 - [Cook and Rai \[ICSE05\]](#)
 - [Willis, Pearce, and Noble \[ECOOP06\]](#)
- Analyzing Query Strings
 - [Gould, Su, and Devanbu \[ICSE04\]](#)

Future Work

- Interprocedural analysis
- Multiple queries
- Persistent update
- More expressive queries
 - Aggregation
 - Compiler optimizations
- New target query languages

Databases



Programming Languages

σ

∞

i

λ

\cdot

Query Extraction

