JQL : The Java Query Language

Slides created by Darren Willis, David J Pearce, James Noble; used with their permission

Victoria
UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

JQL Object Querying

JQL is a simple extension for Java.

JQL adds support for **Object Querying.**

Object Querying helps us pick sets of objects from other sets.

JQL can simplify working with collections.

JQL Object Querying

Without JQL:

```
for(Object a : collection1)
  for(Object b : collection2)
    if(a.equals(b))
      results.add(new Object[]{a,b});
```

With JQL:

```
selectAll(Object a = collection1,
          Object b = collection2
          :a.equals(b));
```

JQL Hypothetical IM Client

A Hypothetical Instant Messaging Client

Session

`String name;`
 	
`send_msg(Msg);`
`recv_msg(Msg);`

Window

`String name;`
`int id;`
 	
`show_msg(Msg);`
`enter_msg();`

JQL Querying : Nested Loop

We're ***joining*** the sets of Sessions and Windows.

An easy way – the nested loop join:

```
ArrayList results = new Arraylist();
for(Session s : sessions)
  for(Window w : windows)
   if(w.name.equals(s.name))
      results.add(new Object[w,s]);
```

This takes **O(|*sessions*| * |*windows*|)** time.

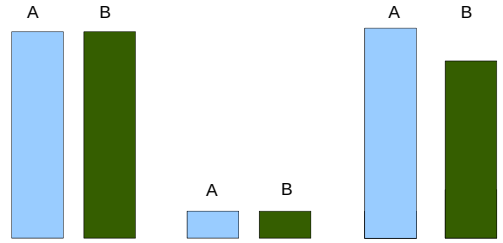
JQL Querying : Hash Join

Another join technique, from databases:

```
ArrayList results = new ArrayList();
Hashtable nameIndex = new Hashtable();
for(Session s : sessions)
  nameIndex.put(s.name,s);
}
for(Window w : windows){
  results.add(names.get(w.name));
}
```

(This is a simplified version)

JQL Joining Dilemma



JQL JQL for Collection Operations

```
results = selectAll(Session s = sessions,
                    Window w = windows:
                    s.name.equals(w.name));
```

Let the Query Evaluator take care of the details!

JQL Domain Variable Sources

Passing in a Domain Variable source

```
selectAll(Window w = windows :w.ID == 5)
```

Domain Variable Definition without a source

```
selectAll(Window w : w.ID == 5)
```

JQL Query Expressions

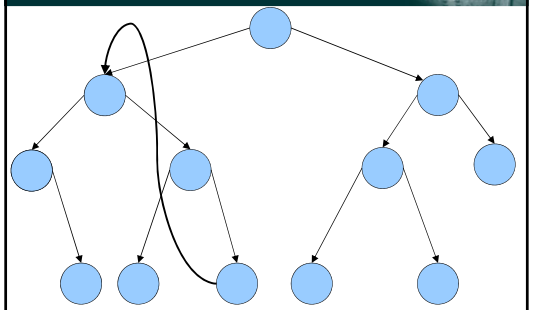
```
w.ID == 5

w.ID == 5 && w.name == s.name

w.ID > 5 && w.ID < 10 && w.name
  == s.name && s.isActive() &&
  ...
```

JQL Query Expressions

- Method calls are allowed
- Side effects are possible
- These expressions do not short circuit

So what else can we do with queries?

JQL BinTree Aliasing



JQL A Query Invariant

```
assert null == selectA(BinTree a, BinTree b :
               (a != b && a.left == b.left)  ||
               (a != b && a.right == b.right)||
                a.left == b.right);
```

Uses **Object Tracking** to check all BinTrees.

JQL Object Tracking

```
public class GroupChat{
   private List particpants;

   public SomeClass(){
   participants = new ArrayList();
   ...
   }
   public void addChatter(String name){
     ...
     participants.add(new Session(name));
     ...
   }
}
```

Object Tracker

JQL Expression Ordering

Queries get broken down along **&&**s

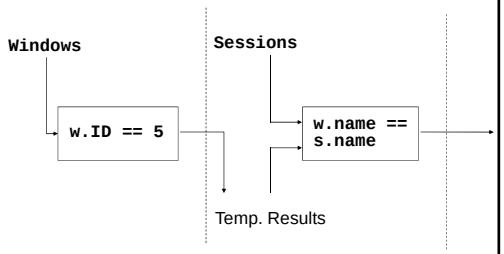
```
w.ID == 5 && w.name == s.name
```

Becomes two subqueries:

```
w.ID = 5
w.name == s.name
```

JQL arranges these into a *query pipeline*.

JQL A Query Pipeline




```
w.ID == 5 && w.name == s.name
```

JQL Ordering the Pipeline

There are two factors to consider:

•Cost
- -Very dependent on input sizes
- - Input sizes are dependent on selectivity…

•Selectivity
- - The proportion of results rejected by the stage

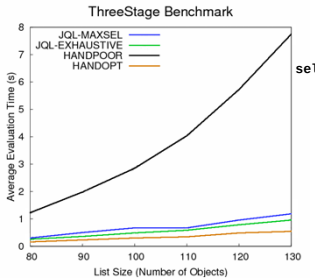
- We estimate it based on the expression used:
  ==, < >, !=, aMethod()
- Or, we can test it with sampling

JQL Configuring Join Order

Two strategies to decide Join Order:

- Exhaustive Search
  - Good results but expensive
  - Requires searching n! possible orderings
- Maximum Selectivity Heuristic
  - Less optimal order, less overhead.

JQL Querying Performance



Four versions of:

```
selectAll(Integer a=as,
          Integer b=bs,
          Integer c=cs,
          Integer d=ds:
          a == b &&
          b != c &&
          c < d);
```

JQL HANDOPT Implementation

```
HashMap<Integer,ArrayList<Integer>> map;
map = new
   HashMap<Integer,ArrayList<Integer>>(
   );
for(Integer i1 : array1) {
    ArrayList<Integer> grp =
       map.get(i1);
    if(grp == null) {
        grp = new ArrayList<Integer>();
        map.put(i1,grp);
    }
    grp.add(i1);
}
ArrayList<Object[]> matches = new
   ArrayList<Object[]>();
Collections.sort(array4);
for(Integer i2 : array2) {
    int b=i2;
    ArrayList<Integer> grp =
       map.get(i2);
    if(grp != null) {
```

```
for(Integer i1 : grp) {  int a=i1;
    for(Integer i3 : array3) {
        int c=i3;
        if(b != c) {
        for(int
x=array4.size();x!=0;x=x1){
        int d=array4.get(x-1);
        if(c<d){
            Object[] t = new Object[4];
            t[0]=a;
            t[1]=b;
            t[2]=c;
            t[3]=d;
            matches.add(t);
        } else { break; }
                }
        }
    }
}
return matches;
```

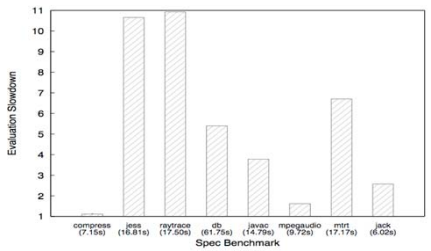
JQL Performance Discussion

```
selectAll(Integer a=as, Integer b=bs,
          Integer c=cs, Integer d=ds :
          a==b && b!=c && c < d);
```

JQL's performance is pretty good!

The average programmer is more likely to code HANDPOOR than HANDOPT.

JQL Object Tracker Performance



Object tracker overhead varies greatly.

JQL Other Querying Systems

Cω/LINQ(C#)
Provide similar querying capabilities.

Query-Based Debuggers
QBD, DQBD by Lencevicius et al.
Used querying and object tracking for debugging.

Relational Databases
Much prior work in query optimisation.

JQL In Development/Consideration

Caching
Both simple, and incrementalised

Other join methods
Again, from databases – Merge, Sort, etc

Tool Support
Eclipse/Netbeans/etc

JQL Conclusions

- Queries are neat
- They are a powerful, simple, useful abstraction.
- JQL provides efficient and easy support for querying in Java.
- For more information on JQL:
  `www.mcs.vuw.ac.nz/~darren/jql/`