SJDB – A Simple Java Database

SJDB supports a limited subset of the relational algebra, consisting of the following operators only:

- cartesian product
- select with a predicate of the form attr=val or attr=attr
- project
- equijoin with a predicate of the form attr=attr
- scan (an operator that reads a named relation as a source for a query plan)

In addition, all attributes on all relations will be strings; there are no other datatypes available. Attributes also have globally unique names (there may not be two attributes of the same name on different relations), and self-joins on relations are not permitted.

The sjdb package contains the following classes and interfaces:

Relation	an unnamed relation, contains attributes
NamedRelation	a named relation
Attribute	an attribute on a relation
Predicate	a predicate for use with a join or select operator
Operator	abstract superclass for all operators
UnaryOperator	abstract superclass for all operators with a single child
Scan	an operator that feeds a named relation into a query plan
Select	an operator that selects certain tuples in its input, via some predicate
Project	an operator that projects certain attributes from its input
BinaryOperator	abstract superclass for all operator with two children
Product	an operator that performs a cartesian product over its inputs
Join	an operator that joins its inputs, via some predicate
Catalogue	a directory and factory for named relations and their attributes
CatalogueException	a failure to retrieve relations or attributes from the catalogue
CatalogueParser	a utility class that reads a serialised catalogue from file
QueryParser	a utility class that reads a query and builds a canonical query plan
PlanVisitor	an interface that when implemented performs a depth-first plan traversal
Inspector	a utility class that traverses an annotated plan and prints out the estimates
SJDB	class containing main()
Test	an example of the test harnesses used for marking

The SJDB class contains a main() method with skeleton code for reading catalogues and queries.

The system provides basic statistical information about the relations and attributes in the database, as below. These are stored on the relations and attributes themselves, and not in the catalogue.

- the number of tuples in each relation
- the value count (number of distinct values) for each attribute

A sample serialised catalogue (cat.txt) and queries (q1.txt, etc) are available in sjdb/data.

Test Harness Notes

The file Test.java in the SJDB distribution contains an example of the test harness that I will be using to mark your submissions. This example test harness manually constructs both plans and catalogues as follows:

```
package sjdb;

import java.io.*;
import java.util.ArrayList;
import sjdb.DatabaseException;

public class Test {
	private Catalogue catalogue;

	public Test() {
	}

	public static void main(String[] args) throws Exception {
		Catalogue catalogue = createCatalogue();
		Inspector inspector = new Inspector();
		Estimator estimator = new Estimator();

		Operator plan = query(catalogue);
		plan.accept(estimator);
		plan.accept(inspector);

		Optimiser optimiser = new Optimiser(catalogue);
		Operator planopt = optimiser.optimise(plan);
		planopt.accept(estimator);
		planopt.accept(inspector);
	}

	public static Catalogue createCatalogue() {
		Catalogue cat = new Catalogue();
		cat.createRelation("A", 100);
		cat.createAttribute("A", "a1", 100);
		cat.createAttribute("A", "a2", 15);
		cat.createRelation("B", 150);
		cat.createAttribute("B", "b1", 150);
		cat.createAttribute("B", "b2", 100);
		cat.createAttribute("B", "b3", 5);

		return cat;
	}

	public static Operator query(Catalogue cat) throws Exception {
		Scan a = new Scan(cat.getRelation("A"));
		Scan b = new Scan(cat.getRelation("B"));

		Product p1 = new Product(a, b);

		Select s1 = new Select(p1, new Predicate(new Attribute("a2"), new Attribute("b3")));

		ArrayList<Attribute> atts = new ArrayList<Attribute>();
		atts.add(new Attribute("a2"));
		atts.add(new Attribute("b1"));

		Project plan = new Project(s1, atts);

		return plan;
	}

}
```

As can be seen in this test harness, I use the **Inspector** class (provided with the SJDB sources) to print out a human-readable version of your query plans – your query plans must be able to accept this visitor without throwing exceptions. Your estimator and optimiser need not (and should not) produce any data on stdout (you should use the Inspector for this when testing).

Note also that you should manually construct plans that contain joins in order to test your Estimators.

Estimators and Optimisers that do not run without errors will be marked by inspection only, and will consequently receive a reduced mark.

Cost Estimation

As described in lectures, the following parameters are used to estimate the size of intermediate relations:

- T(R), the number of tuples of relation R
- V(R,A), the value count for attribute A of relation R (the number of distinct values of A)

Note that, for any relation R, $V(R, A) \leq T(R)$ for all attributes A on R.

Scan

T(R) (the same number of tuples as in the NamedRelation being scanned)

Product

$T(R \times S) = T(R)T(S)$

Projection

$T(\pi_A(R)) = T(R)$ (assume that projection does not eliminate duplicate tuples)

Selection

For predicates of the form attr=val:

$T(\sigma_{A=c}(R)) = T(R)/V(R,A)$, $V(\sigma_{A=c}(R),A) = 1$

For predicates of the form attr=attr:

$T(\sigma_{A=B}(R)) = T(R)/\max(V(R,A),V(R,B))$, $V(\sigma_{A=B}(R),A) = V(\sigma_{A=B}(R),B) = \min(V(R,A), V(R,B)$

Join

$T(R \bowtie_{A=B} S) = T(R)T(S)/\max(V(R,A),V(S,B))$, $V(R \bowtie_{A=B} S,A) = V(R \bowtie_{A=B} S,B) = \min(V(R,A), V(S,B))$

(assume that A is an attribute of R and B is an attribute of S)

Note that, for an attribute C of R that is not a join attribute, $V(R \bowtie_{A=B} S,C) = V(R,C)$

(similarly for an attribute of S that is not a join attribute)

Further Reading

For further information on cost estimation, see §16.4 of Database Systems: The Complete Book