Electronics and
Computer Science

UNIVERSITY OF
Southampton

Coursework Specification

Module:	COMP3211 Advanced Databases		
Assignment:	Database Programming Exercise	**Weighting:**	25 %
Lecturer:	Dr Nicholas Gibbins and Dr George Konstantinidis		
Deadline:	16:00 Wed 5 May 2021	**Feedback:**	Fri 21 May 2021

Instructions

In this assignment, you will build a query optimiser for SJDB, a simple RDBMS. Your optimiser should accept a canonical query plan (a project over a series of selects over a cartesian product over the input named relations) and aim to construct a left-deep query plan which minimises the sizes of any intermediate relations. To help you implement your optimiser, it is strongly suggested that you follow the two phases below:

Phase 1: Estimator.java

Before implementing an optimiser for query plans, you must first estimate the cost of the query plans.

In the first phase, you must create a class **Estimator** that implements the **PlanVisitor** interface and performs a depth-first traversal of the query plan. On each operator, the Estimator should create an instance of Relation (bearing appropriate Attribute instances and tuple counts) and attach to the operator as its output.

Some operators may require you to revise the value counts for the attributes on the newly created output relations (for example, a select of the form attr=val will change the number of distinct values for that attribute to 1). Note also that an attribute on a relation may not have more distinct values than there are tuples in the relation.

Page 5 of this coursework specification lists the formulae that you should use to calculate the sizes of the output relations, and to revise the attribute value counts. The supplied distribution of SJDB includes a skeleton for Estimator, including an implementation of the visit(Scan) method.

Note that in case you have to perform rounding of floats or doubles into integers for you estimations, you should use the ceiling function (approximate to the greater integer).

Phase 2: Optimiser.java

In the second phase, you must create a class **Optimiser** that will take a canonical query plan as input, and produce an optimised query plan as output. The optimised plan should not share any operators with the canonical query plan; all operators should be created afresh.

In order to demonstrate your optimiser, you should be able to show your cost estimation and query optimisation classes in action on a variety of inputs. The SJDB zip file contains a sample catalogue and queries. In addition, the SJDB class (see page 3) contains a main() method with sample code for reading a serialised catalogue from file and a query from stdin.

Note

You should **not** need to modify any of the provided classes or interfaces as part of your submission (aside from Estimator and Optimiser that you will create), but if you think that you have a justifiable reason for doing so, please contact George for permission first. You should **not** use external libraries (such as Guava etc.). Your project should be able to compile by running javac *.java inside the sjdb folder and **not depend on outside code or libraries**. To test this, you can copy your sjdb directory to any university machine (e.g. uglogin.ecs.soton.ac.uk) and test that it compiles there.

Late submissions will be penalised at 10% per working day.
No work can be accepted after feedback has been given.
You should expect to spend up to 37.5 hours on this assignment.
Please note the University regulations regarding academic integrity.

Submission

Please submit ONLY your Estimator.java and Optimiser.java files using the C-Bass electronic hand-in system (http://handin.ecs.soton.ac.uk/) by 4pm on the due date.

Late submissions will be penalised at 10% per working day and no work can be accepted after feedback has been given.

You should expect to spend up to 37.5 hours on this assignment, and you should note the University regulations regarding academic integrity:
http://www.calendar.soton.ac.uk/sectionIV/academic-integrity-statement.html

Relevant Learning Outcomes

1. The internals of a database management system
2. The issues involved in developing database management software
3. Demonstrate how a DBMS processes, optimises and executes a query
4. Implement components of a DBMS

Marking Scheme

Criterion	Description	Outcomes	Total
Cost Estimation	Implementation of the cost estimator	1,2,3,4	50 %
Push Selections	Move select operators down	1,2,3,4	10 %
Join Creation	Combine select operators with cartesian products	1,2,3,4	15 %
Push Projections	Move project operators down	1,2,3,4	10 %
Join Ordering	Reorder joins	1,2,3,4	15 %

Note that partial credit will be given for incomplete solutions; for example, an optimiser that moves some (but not all) selections down the query plan will still receive part of the total mark for the relevant assessment criterion.

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:

Estimator	a skeleton for the Estimator
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
DatabaseException	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 distribution contains an example of the test harness that we 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, we 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) ≤ 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