UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2003

BEng Honours Degree in Computing Part III
MEng Honours Degree in Information Systems Engineering Part IV
BSc Honours Degree in Mathematics and Computer Science Part III
MSci Honours Degree in Mathematics and Computer Science Part III
MSc in Advanced Computing
PhD
for Internal Students of the Imperial College of Science, Technology and Medicine

*This paper is also taken for the relevant examinations for the Associateship of the City and Guilds of London Institute*
*This paper is also taken for the relevant examinations for the Associateship of the Royal College of Science*

# PAPER C312=I4.4

# ADVANCED DATABASES

Wednesday 7 May 2003, 10:00
Duration: 120 minutes

*Answer THREE questions*

Paper contains 4 questions
Calculators not required

## SECTION A

1a The following is a schema of a relational database, together with a *sample* of the data:

Tables: EMPLOYEE (Empno, Salary) cardinality = 200 tuples
EMP_PROJECT (Projno, Empno) cardinality = 2000 tuples

| Empno | Salary |
|---|---|
| 123 | 30,000 |
| 124 | 40,000 |
| 135 | 30,500 |
| 146 | 30,000 |
| 180 | 40,000 |
| 234 | 40,000 |
| 500 | 30,000 |
| ⋮ | ⋮ |

| Projno | Empno |
|---|---|
| 1 | 123 |
| 1 | 135 |
| 2 | 234 |
| 2 | 146 |
| 1 | 500 |
| 2 | 180 |
| 3 | 124 |
| 3 | 500 |
| ⋮ | ⋮ |

i) Using the data above, draw an EER diagram representing the two tables and their relationship. Make sure to state all assumptions you have made.

ii) Illustrate how this would look in an Object-Oriented (John Hughes) schema (ignoring operations).

b A relational DBMS is running on a tiny embedded computer, which contains a hard disk, and only enough memory to hold 100 tuples and ALL indexes.

The database consists of the two tables EMPLOYEE and EMP_PROJECT from part (a). For the query below, answer the following questions (assume keys are NOT automatically indexed and tuples for both tables are the same size):

| Query: | SELECT SALARY<br>FROM EMPLOYEE, EMP_PROJECT<br>WHERE EMPLOYEE.EMPNO = EMP_PROJECT.EMPNO; |
|---|---|
| Answer Output | 20 Tuples are output as a result of this query. |

i) Which of Nested-Loop or Sort-Merge join algorithms would you suggest the DBMS use and why? Illustrate your answer in terms of disk reads and tuple comparisons stating all assumptions clearly.

ii) How would a single additional index on one table speed this up? Illustrate the speed up potential in terms of disk reads and tuple comparisons, stating all assumptions clearly.

*The two parts carry, respectively, 40% and 60% of the marks*

2 Below is an example of two banking transactions that can run on Greedy Bank Plc's computing system.

$H_1 = r_1(z), [z = z + 100], w_1(z), r_1(y), r_1(s), [s = s + z], w_1(s), c_1$

$H_2 = r_2(y), [y = y - 100], r_2(z), [z = z + 100], w_2(y), w_2(z), c_2$

a Using histories notation, describe a *concurrent* execution, $H_{3a}$, of the transactions $H_1$ and $H_2$ using conservative 2PL.

b Using histories notation, describe a *concurrent* execution, $H_{3b}$, of transactions $H_1$ and $H_2$ using 2PL where locks are obtained only as necessary.

c If $H_1$ obtains the CPU first which of the two histories $H_{3a}$ or $H_{3b}$ cause the transactions to be waiting on each other and by how much? Assume each lock/unlock/read/write/calculation/commit takes a single unit of time and the computer can only process one command at a time.

d The table below shows the Recovery Manager log found when a database is executing a recovery that includes a cache consistent checkpoint record. State which REDO and UNDO actions the recovery manager will execute, and restore the branch table to a consistent state.

| | |
|---|---|
| LOG | $b_3$ |
| LOG | $b_4$ |
| UNDO | $w_4[b_{67}, cash = 36005.00]$ |
| REDO | $w_4[b_{67}, cash = 34005.00]$ |
| LOG | $b_1$ |
| UNDO | $w_1[b_{56}, cash = 94340.00]$ |
| REDO | $w_1[b_{56}, cash = 84340.00]$ |
| UNDO | $w_4[b_{34}, cash = 10900.67]$ |
| REDO | $w_4[b_{34}, cash = 8900.67]$ |
| LOG | $b_2$ |
| UNDO | $w_2[b_{67}, cash = 34005.00]$ |
| REDO | $w_2[b_{67}, cash = 36005.00]$ |
| LOG | $c_4$ |
| LOG | $c_2$ |
| UNDO | $w_1[b_{67}, cash = 36005.00]$ |
| REDO | $w_1[b_{67}, cash = 18900.00]$ |
| LOG | $b_3$ |
| UNDO | $w_3[b_{67}, cash = 18900.00]$ |
| REDO | $w_3[b_{67}, cash = 37005.00]$ |

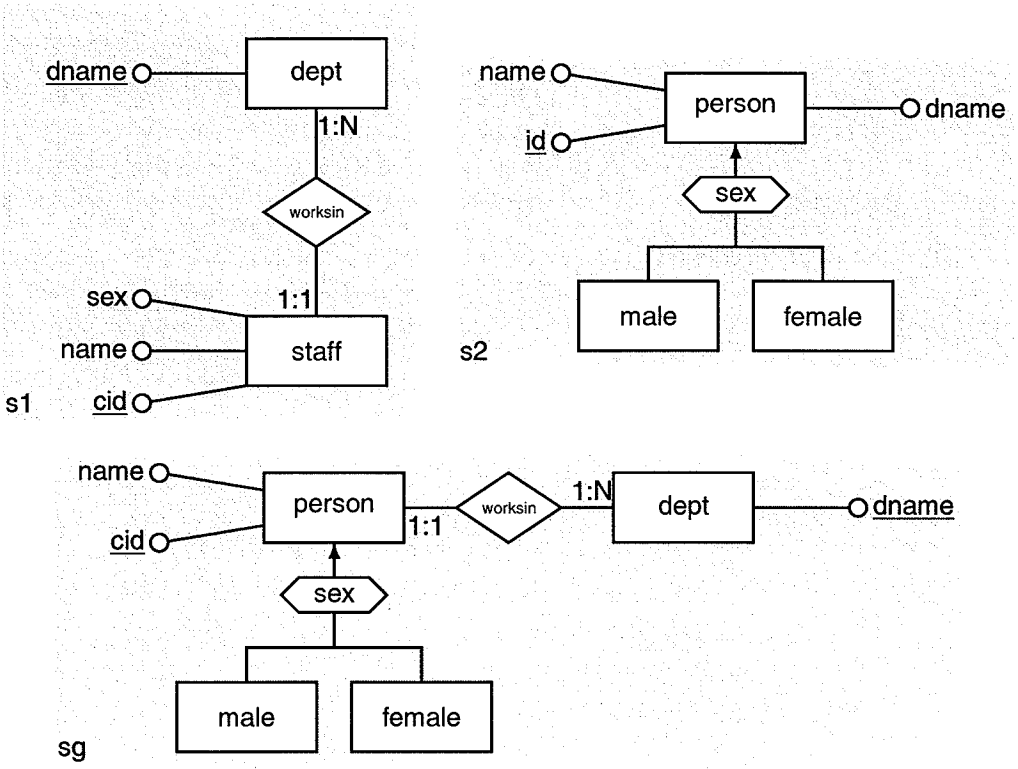
| Branch | |
|---|---|
| Sortcode | Cash |
| 56 | 84340.00 |
| 34 | 10900.67 |
| 67 | 36005.00 |
| 77 | 40000.00 |

e Suppose the system crashes while we are recovering from a previous crash. How would this affect the recovery of the database using the Recovery Manager log described in (d)?

*The five parts carry, respectively, 15%, 15%, 20%, 30%, and 20% of the marks*

## SECTION B

3a The diagram below shows two local schemas s1 and s2 and a global schema sg. Each member of staff in s1 is identified by their college identifier cid, and is associated to the department dept in which they work, which is identified by its name. The same set of staff are held in s2 in the person entity, and identified by the same college identifier held in the attribute id, and have an attribute dname recording the name of the department in which they work.




Your answer should clearly enumerate the primitive transformations steps during the integration, using notation such as:

addEntity(⟪special⟫, {X | salesman(X) ∧
salesman_manages_customer(X, Y) ∧ customer_fax(Y, '*123456*')})

i) Write the transformation steps that are necessary to convert s1 into sg

ii) Write the transformation steps that are necessary to convert s2 into sg

b The following are three transactions which operate on the data shown in `account` table below, stated as a sequence of read and write operations. The `details` table holds extra information about accounts, which the `number` field is a foreign key to `account`

**account**

| number | balance | holder_name | type |
|---|---|---|---|
| 0 | -70 | Prithweesh | current |
| 1 | 100 | Tina | current |
| 2 | 50 | Bundna | current |
| 3 | 20 | Bundna | savings |
| 4 | 197,800,456 | Imperial College | current |
| 5 | 200 | Prithweesh | savings |

**details**

| number | address | comment |
|---|---|---|
| 2 | 6 Railway Avenue, London SW20 | Account used for charity work |
| 4 | Exhibition Road, London SW7 | Main college account |

$H_1 = r_1[a_2], r_1[a_3], w_1[a_3]$
$H_2 = r_2[a_0], r_2[a_1], r_2[a_2], r_2[a_3], r_2[a_4], r_2[a_5]$
$H_3 = r_3[a_4], w_3[a_4], r_3[a_2], w_3[a_2]$

i) Write down a concurrent execution involving all of $H_1, H_2$ and $H_3$ which results in a deadlock, and draw the distributed waits-for graph (WFG) if the data is fragmented with accounts 0–2 on $S_1$ and 3–5 on $S_2$.

ii) Suppose most transactions are like $H_1$ (dealing only with accounts held by one person). Name the fragmentation technique and give the distribution of tables that could be applied to reduce the number of EXT arcs in the distributed WFG.

iii) Suppose it was found that the transactions of the form below had conflicts with transactions of the form of $H_3$ (which makes transfers between two accounts by updating the `balance` field). Name the fragmentation technique and give the definition for the distribution of `account` between the two servers that reduces the number of spawn arcs to EXT nodes in the distributed WFG.

```
SELECT number FROM account WHERE holder_name='...'
```

iv) Describe using the relational algebra how the fragmentation of `details` would be defined, if the records are to be always kept on the same server as records of `account` to which they correspond.

v) For the SQL query below, give the *optimised* relational algebra query tree if fragmentation of `account` were as defined as in your answer to (iii), combined with the fragmentation of `details` in your answer to (iv).

```
SELECT holder_name, comment FROM account,details
WHERE details.number<=2 AND account.number=details.number
```

*The two parts carry, respectively, 36% and 64% of the marks*

4a The XML file and associated element schema below is meant to comply to the following XML Schema, but the XML file contains errors. The XML file and the XML Schema have each line prefixed by a number which you can reference in the answer.

```
01  ⟨parts_suppliers_db⟩
02    ⟨part colour="10" pno="100" price="20.2"/⟩
03    ⟨part colour="red" pno="101" price="cheap"/⟩
04    ⟨part colour="yellow" pno="101" price="11"/⟩
05    ⟨supplier sno="20A" town="London"⟩
06      ⟨supplies⟩100⟨/supplies⟩
07      ⟨supplies⟩102⟨/supplies⟩
08    ⟨/supplier⟩
09    ⟨supplier sno="50J" town="Oslo"⟩
10      ⟨supplies⟩100⟨/supplies⟩
11      ⟨supplies⟩101⟨/supplies⟩
12    ⟨/supplier⟩
13  ⟨/parts_suppliers_db⟩
```

part

| id | pid | ord | colour | pno | price |
|---|---|---|---|---|---|
| 2 | 1 | 1 | 10 | 100 | 20.2 |
| 6 | 1 | 2 | red | 101 | cheap |
| 10 | 1 | 3 | yellow | 101 | 11 |

supplier

| id | pid | ord | sno | town |
|---|---|---|---|---|
| 14 | 1 | 4 | 20A | London |
| 21 | 1 | 5 | 50J | Oslo |

supplies

| id | pid | ord | supplies |
|---|---|---|---|
| 17 | 14 | 1 | 100 |
| 19 | 14 | 2 | 102 |
| 24 | 21 | 1 | 100 |
| 26 | 21 | 2 | 101 |

```
14  ⟨xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"⟩
15    ⟨xsd:element name="parts_suppliers_db"⟩
16      ⟨xsd:complexType⟩
17        ⟨xsd:choice maxOccurs="unbounded"⟩
18          ⟨xsd:element name="part" type="part"/⟩
19          ⟨xsd:element name="supplier" type="supplier"/⟩
20        ⟨/xsd:choice⟩
21      ⟨/xsd:complexType⟩
22      ⟨xsd:key name="part_pk"⟩
23        ⟨xsd:selector xpath="part"/⟩
24        ⟨xsd:field xpath="@pno"/⟩
25      ⟨/xsd:key⟩
26      ⟨xsd:key name="supplier_pk"⟩
27        ⟨xsd:selector xpath="supplier"/⟩
28        ⟨xsd:field xpath="@sno"/⟩
29      ⟨/xsd:key⟩
30      ⟨xsd:keyref name="supplier_supplies_fk" refer="part_pk"⟩
31        ⟨xsd:selector xpath="supplier/supplies"/⟩
32        ⟨xsd:field xpath="."/⟩
33      ⟨/xsd:keyref⟩
34    ⟨/xsd:element⟩
35    ⟨xsd:complexType name="supplier"⟩
36      ⟨xsd:choice maxOccurs="unbounded"⟩
37        ⟨xsd:element name="supplies" type="xsd:integer"/⟩
38      ⟨/xsd:choice⟩
39      ⟨xsd:attribute name="sno" type="xsd:string"/⟩
40      ⟨xsd:attribute name="town" type="xsd:string"/⟩
41    ⟨/xsd:complexType⟩
42    ⟨xsd:complexType name="part"⟩
43      ⟨xsd:attribute name="pno" type="xsd:integer"/⟩
44      ⟨xsd:attribute name="colour" type="xsd:string"/⟩
45      ⟨xsd:attribute name="price" type="xsd:float"/⟩
46    ⟨/xsd:complexType⟩
47  ⟨/xsd:schema⟩
```

i) Describe how the XML Schema integrity rules are being broken by three errors in the XML file. You answer should include the line number of the rule

being broken and the line number of the element(s) and attribute(s) that break the rule.

ii) Describe using the relational algebra how the following relational tables could be populated from the element schema of the XML file. Note that the two attributes of part_supplier are also foreign keys pointing at the corresponding attributes of part and supplier. (You may use the notation $\pi_{x\ as\ y}$ to project an attribute $x$ as having name $y$).

rdb_part(pno,colour,price)
rdb_supplier(sno,town)
rdb_part_supplier(pno,sno)

iii) What aspect of the original XML file could not be created by an application that used the element schema to recreate the file?

b The following shows a temporal database, represented in the temporal structure, for the valid time records of a stock table, recording the share price of companies identified by code.

| stock | |
|---|---|
| code | price |
| BAA | 162 |
| ENR | 1000 |
| ZEN | 1500 |

$t = 0$

| stock | |
|---|---|
| code | price |
| BAA | 160 |
| ENR | 10 |
| HMV | 210 |
| ZEN | 1450 |

$t = 1$

| stock | |
|---|---|
| code | price |
| BAA | 162 |
| HMV | 270 |
| ZEN | 1473 |
| ORG | 600 |

$t = 2$

| stock | |
|---|---|
| code | price |
| BAA | 164 |
| HMV | 272 |
| ZEN | 1480 |
| ORG | 450 |

$t = 3$

For each of the following three temporal relation algebra queries, list the output relation run at times stated in each part, and suggest what the query is intended to do (*e.g.* 'Finds all people who have left the company'). Note that ◆ is the *sometime in the past* operator, ■ is the *always in the past* operator, ● is the *previous time* operator, and $\overset{S}{\times}$ is the *since product* operator. Also note that stock1 and stock2 are aliases for the stock table.

i) at $t = 4$: $\blacksquare \pi_{\text{code}}$ stock

ii) at $t = 3$: $\pi_{\text{stock1.code}}\, \sigma_{\text{stock1.code=stock2.code} \wedge \text{stock1.price<stock2.price}}$ (stock1 × ◆stock2)

iii) at $t = 4$: $(\pi_{\text{code}} \bullet \text{stock}) -$
$\blacklozenge \pi_{\text{stock1.code}}\, \sigma_{\text{stock1.code=stock2.code} \wedge \text{stock1.price<stock2.price}}$ (stock1 × ●stock2)

iv) Give the temporal relation algebra query that finds all the stocks that have traded continuously since ENR last traded.

*The two parts carry, respectively, 50% and 50% of the marks*