

CS4604 Final Examination

December 11, 2006

Please enter the following information:

Name:

ID:

GOOD LUCK!

Please do not write below this line.

Question	Maximum Score	Score
1	30	
2	7	
3	11	
4	12	
5	20	
6	20	
7*	20	
Total	100	

* Extra credit

1. ($3 \times 10 = 30$ points)

- (a) If relation R has n tuples and relation S has m tuples, what is the maximum number of tuples that $R \cap S$ can contain? Assume set-theoretic semantics.

Solution: The answer is $\min(m, n)$. A number of students answered “ n , when S contains no tuples in R ”. I deducted two points for that answer.

- (b) An E/R diagram has three entity sets A , B , and C . It also contains three relationships: R between A and B ; S between B and C ; and T between C and A . The relationships do not contain any attributes. Which of the relationships is redundant, i.e., the tuples contained in it can be inferred from the tuples in the other two relationships?

Solution: All may be needed, since different entities may participate in each relationship. Some students said T is redundant. How can that be? The situation is completely symmetric? If T is redundant, then why are R and S not redundant?

- (c) The scenario is the same as in the previous question. In addition, you know that each relationship is one-one. Which relationships are redundant?

Solution: The one-one constraints do not make any relationship redundant. Once again, different tuples may belong to each relationship.

- (d) How many non-trivial multi-valued dependencies are possible in a two attribute relation?

Solution: Zero. Any multi-valued dependency in this relation must involve both two attributes. Hence, it is trivial.

- (e) What is the difference between “DROP R;” and “DELETE FROM R;”?

Solution: DROP R; deletes the entire table from the database, whereas DELETE FROM R; deletes *all* the tuples in the table but retains the table definition in the database. A number of students said that the second query deletes one or some tuple from R , which is incorrect. I deducted one point for that error. Note that DELETE FROM R; is a valid SQL query.

- (f) When is an attribute-based CHECK constraint checked by an RDBMS?

Solution: I was looking for a specific answer: whenever an update to the table containing that attribute changes the value of the attribute. I deducted two points for generic answers such as “when a change is made to the database” or “inserts and updates”.

- (g) Rewrite $\sigma_C(R \times S)$ as another expression in relational algebra.

Solution: $R \bowtie_C S$

- (h) A relation R has attributes A , B , and C . Fill in the blank so that the query returns each tuple in R exactly once. `SELECT * FROM R _____;`

Solution: I goofed on this question. I meant to ask for a query that returns each *distinct* tuple in R exactly once. In that case, the answer is GROUP BY A, B, C. For the question I actually asked, you can leave the blank empty. I did deduct two points when students completed the query using illegal SQL.

- (i) Circle **true** or **false**. The result of the query
`(SELECT ProfessorPID FROM Teach) EXCEPT (SELECT PID FROM Professors);`
can contain the same tuple more than once.

Solution: False, because SQL converts relations to sets upon seeing a “set”-like query such as EXCEPT, UNION, or INTERSECT.

- (j) Who lives in a pineapple under the sea?

Solution: SpongeBob SquarePants, of course! Everyone got this one correct, since it was not a serious question.

2. (3 + 4 = 7 points) Suppose a relation R has attributes A_1, A_2, \dots, A_n . As a function of n , how many superkeys does R have if

(a) R has exactly one key made up of two attributes $\{A_1, A_2\}$.

Solution: A superkey is any set of attributes that includes a key. A key itself is a superkey. Therefore, to form a superkey, we need A_1, A_2 , and any subset of the remaining $n - 2$ attributes. As we have discussed a few times in class, the number of such subsets is 2^{n-2} .

To count the number of subsets of k objects (including the empty set), keep in mind that for each object we have two choices: whether to include the object in a subset or not. Therefore, there are $2 \times 2 \times \dots \times 2$ subsets (2 multiplied by itself k times), which is 2^k .

A few students thought that a superkey is a key, which led them to the wrong answer.

I tried to award partial credit depending on how close I felt you came to the correct answer.

(b) R has precisely two keys $\{A_1\}$ and $\{A_2\}$.

Solution: Reasoning as in the previous question, there are 2^{n-1} superkeys that contain $\{A_1\}$ and 2^{n-1} superkeys that contain $\{A_2\}$. However, in each of these two sets of superkeys, we have counted the set of superkeys that contain both A_1 and A_2 . So, to get the right number of superkeys, we have to subtract 2^{n-2} from $2^{n-1} + 2^{n-1}$, yielding $3 \times 2^{n-2}$ superkeys. I took off two points if you forgot to account for overcounting.

3. (5 + 3 + 3 = 11 points) In relational algebra, the outerjoin operator is like the natural join operator, except that it allows dangling tuples to be padded with nulls in the result. Specifically, $R \overset{\circ}{\bowtie} S$ contains all tuples in $R \bowtie S$ and any tuples in R or S that do not join with any tuples in the other relation. Each such dangling tuple is padded with appropriate NULL values in $R \overset{\circ}{\bowtie} S$. Here is an example:

	<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><th colspan="2">R</th></tr> <tr><th>A</th><th>B</th></tr> <tr><td>1</td><td>2</td></tr> <tr><td>2</td><td>3</td></tr> <tr><td>2</td><td>4</td></tr> </table>	R		A	B	1	2	2	3	2	4	<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><th colspan="2">S</th></tr> <tr><th>B</th><th>C</th></tr> <tr><td>2</td><td>10</td></tr> <tr><td>2</td><td>8</td></tr> <tr><td>3</td><td>15</td></tr> <tr><td>8</td><td>3</td></tr> </table>	S		B	C	2	10	2	8	3	15	8	3	<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><th colspan="3">$R \overset{\circ}{\bowtie} S$</th></tr> <tr><th>$A$</th><th>$B$</th><th>$C$</th></tr> <tr><td>1</td><td>2</td><td>10</td></tr> <tr><td>1</td><td>2</td><td>8</td></tr> <tr><td>2</td><td>3</td><td>15</td></tr> <tr><td>2</td><td>4</td><td>NULL</td></tr> <tr><td>NULL</td><td>8</td><td>3</td></tr> </table>	$R \overset{\circ}{\bowtie} S$			A	B	C	1	2	10	1	2	8	2	3	15	2	4	NULL	NULL	8	3
R																																														
A	B																																													
1	2																																													
2	3																																													
2	4																																													
S																																														
B	C																																													
2	10																																													
2	8																																													
3	15																																													
8	3																																													
$R \overset{\circ}{\bowtie} S$																																														
A	B	C																																												
1	2	10																																												
1	2	8																																												
2	3	15																																												
2	4	NULL																																												
NULL	8	3																																												

- (a) Write a relational algebra expression equivalent to $R \overset{\circ}{\bowtie} S$ (for the relations R and S in the example), using the relational algebra operations we learnt in class. You may have to invent some notation for “padding with NULLs”. If you do so, specify the notation clearly.

Solution: The definition of the outerjoin operator in the question, almost has the answer. Compute the natural join. Remove the tuples in the natural join from each relation (after an appropriate projection). Pad the remaining tuples in each relation with NULLs and add them to the natural join. Here is a solution:

$$R \bowtie S \cup ((R - \pi_{A,B}(R \bowtie S) \times \text{NULL}_C) \cup (\text{NULL}_A \times (S - \pi_{B,C}(R \bowtie S))),$$

where NULL_A stands for a relation with one attribute A and one row that has a NULL value for that attribute.

- (b) Suppose A is the key for R , R contains n tuples, and S contains m tuples. What is the maximum number of tuples that $R \bowtie S$ can contain? An RDBMS can use such reasoning to estimate the size of a natural join.

Solution: The answer is mn , since each tuple of R may join with each tuple of S on attribute B , e.g., when all tuples have the same value for B .

The question had a typo. I wanted to make B the key for R . In that case, the answer is m . Since A is a key for R , each tuple in S can pair up with at most one tuple of R . Therefore, $R \bowtie S$ can contain at most the number of tuples in S .

- (c) For the same conditions as the previous problem, what is the maximum number of tuples that $R \overset{\circ}{\bowtie} S$ can contain?

Solution: The answer is mn . Some students wrote a smaller number here than in the previous question. How can that be correct, when the outerjoin has more tuples than the natural join, by definition?

4. (3 + 2 + 7 = 12 points) The relation `Teach`(`CourseName`, `CourseNumber`, `Department`, `ProfessorPID`) satisfies the following constraint expressed in relational algebra:

$$\sigma_C(\rho_T(\text{Teach}) \times \rho_U(\text{Teach})) = \emptyset,$$

where the condition C is

(`T.CourseNumber` = `U.CourseNumber`) AND (`T.Department` = `U.Department`)
AND (`T.CourseName` <> `U.CourseName`).

- (a) Write this constraint in another notation (not SQL, or relational algebra, or English, or Spanish, or ...).

Solution: `CourseNumber Department` \rightarrow `CourseName`. We have discussed in class how to write an FD as a constraint in relational algebra. The book also discusses this point.

- (b) What is the key for `Teach`?

Solution: Using the algorithm for using FDs to find keys leads us inexorably to the conclusion that the key is

{`CourseNumber`, `Department`, `ProfessorPID`}.

- (c) `Teach` has some redundancy. Write a sequence of SQL commands that will *replace* `Teach` with one or more other relations that do not have any redundancy.

Solution: I wanted you to realise that `Teach` is not in BCNF and that it can be brought into BCNF by decomposing it into two relations. I wanted the SQL commands that would create these relations, insert the right tuples from `Teach` into these relations, and remove `Teach`. Here is Tim Driscoll's solution, a modified little and without types for the attributes:

```
CREATE TABLE Courses(CourseNumber, Department, CourseName);
INSERT INTO Courses (SELECT CourseNumber, Department, CourseName FROM Teach);
ALTER TABLE Teach DROP COLUMN CourseName;
```

Most students elected to create two new tables, corresponding to the results of decomposition. I deducted three points if you did not insert tuples into the new tables. I deducted three points if you only created one (correct) table. I deducted one point if you did not drop the `Teach` table.

5. (20 points) This question involves using SQL to do some simple data mining. You have recently been appointed the Chief Information Officer of a small departmental store that has ambitions of taking on Wal-Mart. You know that Wal-Mart has been very successful in using data mining to figure out exactly what their customers want (even if the customers do not know themselves), so you set yourself the task of figuring out the purchasing patterns of your customers. Your company's database has one table with the following schema: `Purchases(Transaction, Item)`. A transaction consists of all the items a customer purchases in one visit to the store. Each transaction gets a unique identifier, which is stored in the `Transaction` attribute of the `Purchases` relation. For each item bought in that transaction, `Purchases` contains a tuple for that transaction-item pair. Therefore, if a customer buys k items in a visit to the store, `Purchases` contains k tuples representing that transaction.

Write an SQL query whose result has three attributes—`Item1`, `Item2`, and `Count`—and satisfies the following conditions:

- (a) the `Count` value is at least 2000 in each tuple in the result,
- (b) in each tuple, the number of transactions in which customers buy `Item1` is at least as large as the number of transactions in which customers buy `Item2`, and
- (c) the result contains all pairs of items (i_1, i_2) such that there are at least 2000 transactions where a customer purchased both items. Such pairs of items are examples of *frequent itemsets* in data mining.

As an example, suppose `(Diapers, Beer, 2174)` is a tuple in the result. This tuple means that in 2174 transactions where a customer purchased diapers, the customer also purchased beer. Keep in mind that there may be other transactions where the customer purchased diapers but not beer and vice-versa. You may find it convenient to use `VIEW`s to answer this query (*but you do not have to*).

Solution: The key is to join `Purchases` with itself on the `Transaction` attribute. Then, you obtain a set of all pairs of items bought in the same transaction (with the ID of the shared transaction). After this, grouping by the item pair and checking for at least 2000 shared transactions in the `HAVING` clause does the trick. You also have to ensure that `item1` is purchases in more transactions than `item2`. Here is the SQL query that Alaina Ambrose came up with:

```
CREATE VIEW PurchaseCount AS
(SELECT Item, COUNT (Transaction) AS cnt FROM Purchases GROUP BY Item);
SELECT P.item AS item1, Q.item AS item2, COUNT (P.transaction) AS count
FROM Purchases AS P, Purchases AS Q, PurchaseCount AS PC, PurchaseCount AS QC
WHERE P.Item = PC.Item AND Q.item = QC.item AND P.cnt >= Q.cnt
      AND P.Transaction = Q.Transaction AND P.item <> Q.item
GROUP BY P.item, Q.item
HAVING COUNT (P.transaction) >= 2000;
```

I deducted five points if you did not satisfy condition (b) (in the query above, the check `P.cnt >= Q.cnt` enforces this condition. A number of students did not guarantee that they returned item pairs where both items were purchased in at least 2000 transactions; their query only ensured that the sum of the number of transactions in which a customer purchased `item1` and the number of transactions (possibly different) in which a customer purchased `item2` was 2000. I deducted five points for this error.

Some students mis-read the question to imply that they needed to write a separate query for each of the three parts. I was as generous with partial credit as I could be.

6. ($4 \times 5 = 20$ points) Query optimisers implemented in many RDBMSs take a given SQL query, convert it into an equivalent representation in relational algebra, and use equivalences between different relational algebra expressions to convert the initial query into one that can be executed faster.

In the next five questions, you are given two relational algebra expressions separated by an equivalence sign (\equiv). For each question, write **True** if the expressions are equivalent and **False** if they are not. If the expressions are not equivalent, state a condition under which they become equivalent. Take care to ensure that the conditions are as general as possible.

- (i) $\sigma_C(\pi_{A_1, A_2, \dots, A_n}(R)) \equiv \pi_{A_1, A_2, \dots, A_n}(\sigma_C(R))$, provided that every attribute involved in C belongs to the set $\{A_1, A_2, \dots, A_n\}$.

Solution: True.

- (ii) $\sigma_C(R \times S) \equiv \sigma_C(R) \times S$.

Solution: False, but is true if C involves only attributes that are in R .

- (iii) $\pi_{A_1, A_2, \dots, A_n}(R - S) \equiv \pi_{A_1, A_2, \dots, A_n}(R) - \pi_{A_1, A_2, \dots, A_n}(S)$.

Solution: False, since projecting first may make some tuples in R and S be identical, when the complete unprojected tuples are not. However, the condition is true if $\{A_1, A_2, \dots, A_n\}$ are all the attributes of R (and S). Eric Hultman had a nice criterion: $\{A_1, A_2, \dots, A_n\}$ is a superkey for R and for S . In this case, all the other attributes of R and S are determined by $\{A_1, A_2, \dots, A_n\}$. Therefore, projecting into them before or after the intersection does not matter.

- (iv) $\pi_{A_1, A_2, \dots, A_n}(R \cup S) \equiv \pi_{A_1, A_2, \dots, A_n}(R) \cup \pi_{A_1, A_2, \dots, A_n}(S)$.

Solution: True

- (v) $\sigma_{C \text{ AND } D}(R) \equiv \sigma_C(\sigma_D(R))$.

Solution: True

7. (Extra credit, 20 points) Consider the relations

Branch(BranchName, Assets, City)

Customer(CustomerName, Address, City).

Account(AccountNumber, BranchName, CustomerName, Balance).

The following query finds the assets and names of all banks that have depositors living in Blacksburg and have a balance of more than \$100,000:

$$\pi_{\text{Asset, BranchName}}(\sigma_{\text{Customer.City='Blacksburg' AND Balance}>100000}(\text{Customer} \bowtie \text{Account} \bowtie \text{Branch}))$$

Use the equivalence rules from the previous question (or any others you can come up with) to rewrite this query into something that be executed much faster. Your ultimate goal is to ensure that each relation used in a natural join contains the smallest possible number of rows and attributes. Convert the query in steps; in each step, apply one equivalence rule. Mention the rule, and write the new relational algebra expression that results from applying the rule.

Solution: The trick is to split the condition into two and apply the two pieces individually to the appropriate relation. I gave full credit if you did so. You can also move the projections as much into the query as possible, keeping only the necessary attributes. Here is the final query:

$$\pi_{\text{Assets,BranchName}}((\pi_{\text{CustomerName}}(\sigma_{\text{City='Blacksburg'}}(\text{Customer}))) \bowtie \pi_{\text{BranchName,CustomerName}}(\sigma_{\text{Balance}>100000}(\text{Accounts}))) \\ \bowtie \pi_{\text{Assets,BranchName}}(\text{Branch}))$$

Some students added the condition $\sigma_{\text{City='Blacksburg'}}(\text{Branch})$ to their solution. However, this condition is not in the original query; a customer living in Blacksburg may have a large balance in a branch not located in Blacksburg. I deducted two points for that error.

Scratch space. Not graded.

Scratch space. Not graded.

Scratch space. Not graded.

Scratch space. Not graded.