Purdue University
Computer Science Department
CS 448: Introduction to Database Systems
Prof: Bharat Bhargava

CS 448 – Homework #3
Due 10/14/2016
Reference: Fundamentals of Database Systems, Elmasri and Navathe, 6th Edition

Question 0
Write your “homework sn” from Blackboard on the top of the first page of your submission. It has the format XXPX.

Question 1
17.7 (p.625)

Question 2
17.20 (p.626)

Question 3
17.25 (p.626)

Question 4
18.12 (p.671)

Question 5
18.24 (p.673)

Question 6
19.13 (p.724) Do the work for Q1 (p.100) and Q1A (p.101) and do part b) of the question.

Question 7
15.2 (p.537)

Question 8
15.5 (p.537)

Question 9
15.24 (p.538)

those tuples for which the condition evaluates to TRUE after substituting their corresponding attribute values—are selected.

Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:  SELECT  Fname, Lname, Address
     FROM    EMPLOYEE, DEPARTMENT
     WHERE   Dname='Research' AND Dnumber=Dno;
```

In the WHERE clause of Q1, the condition Dname = 'Research' is a **selection condition** that chooses the particular tuple of interest in the DEPARTMENT table, because Dname is an attribute of DEPARTMENT. The condition Dnumber = Dno is called a **join condition**, because it combines two tuples: one from DEPARTMENT and one from EMPLOYEE, whenever the value of Dnumber in DEPARTMENT is equal to the value of Dno in EMPLOYEE. The result of query Q1 is shown in Figure 4.3(b). In general, any number of selection and join conditions may be specified in a single SQL query.

A query that involves only selection and join conditions plus projection attributes is known as a **select-project-join** query. The next example is a select-project-join query with *two* join conditions.

Query 2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

```
Q2:  SELECT  Pnumber, Dnum, Lname, Address, Bdate
     FROM    PROJECT, DEPARTMENT, EMPLOYEE
     WHERE   Dnum=Dnumber AND Mgr_ssn=Ssn AND
             Plocation='Stafford';
```

The join condition Dnum = Dnumber relates a project tuple to its controlling department tuple, whereas the join condition Mgr_ssn = Ssn relates the controlling department tuple to the employee tuple who manages that department. Each tuple in the result will be a *combination* of one project, one department, and one employee that satisfies the join conditions. The projection attributes are used to choose the attributes to be displayed from each combined tuple. The result of query Q2 is shown in Figure 4.3(c).

4.3.2 Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

In SQL, the same name can be used for two (or more) attributes as long as the attributes are in *different relations*. If this is the case, and a multitable query refers to two or more attributes with the same name, we *must* **qualify** the attribute name with the relation name to prevent ambiguity. This is done by *prefixing* the relation name to the attribute name and separating the two by a period. To illustrate this, suppose that in Figures 3.5 and 3.6 the Dno and Lname attributes of the EMPLOYEE relation were

called Dnumber and Name, and the Dname attribute of DEPARTMENT was also called Name; then, to prevent ambiguity, query Q1 would be rephrased as shown in Q1A. We must prefix the attributes Name and Dnumber in Q1A to specify which ones we are referring to, because the same attribute names are used in both relations:

```
Q1A:  SELECT   Fname, EMPLOYEE.Name, Address
      FROM     EMPLOYEE, DEPARTMENT
      WHERE    DEPARTMENT.Name='Research' AND
               DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;
```

Fully qualified attribute names can be used for clarity even if there is no ambiguity in attribute names. Q1 is shown in this manner as is Q1′ below. We can also create an *alias* for each table name to avoid repeated typing of long table names (see Q8 below).

```
Q1′:  SELECT   EMPLOYEE.Fname, EMPLOYEE.LName,
               EMPLOYEE.Address
      FROM     EMPLOYEE, DEPARTMENT
      WHERE    DEPARTMENT.DName='Research' AND
               DEPARTMENT.Dnumber=EMPLOYEE.Dno;
```

The ambiguity of attribute names also arises in the case of queries that refer to the same relation twice, as in the following example.

Query 8. For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
Q8:   SELECT   E.Fname, E.Lname, S.Fname, S.Lname
      FROM     EMPLOYEE AS E, EMPLOYEE AS S
      WHERE    E.Super_ssn=S.Ssn;
```

In this case, we are required to declare alternative relation names E and S, called **aliases** or **tuple variables**, for the EMPLOYEE relation. An alias can follow the keyword **AS**, as shown in Q8, or it can directly follow the relation name—for example, by writing EMPLOYEE E, EMPLOYEE S in the FROM clause of Q8. It is also possible to **rename** the relation attributes within the query in SQL by giving them aliases. For example, if we write

```
EMPLOYEE AS E(Fn, Mi, Ln, Ssn, Bd, Addr, Sex, Sal, Sssn, Dno)
```

in the FROM clause, Fn becomes an alias for Fname, Mi for Minit, Ln for Lname, and so on.

In Q8, we can think of E and S as two *different copies* of the EMPLOYEE relation; the first, E, represents employees in the role of supervisees or subordinates; the second, S, represents employees in the role of supervisors. We can now join the two copies. Of course, in reality there is *only one* EMPLOYEE relation, and the join condition is meant to join the relation with itself by matching the tuples that satisfy the join condition E.Super_ssn = S.Ssn. Notice that this is an example of a one-level recursive query, as we will discuss in Section 6.4.2. In earlier versions of SQL, it was not possible to specify a general recursive query, with an unknown number of levels, in a

15.2. Discuss insertion, deletion, and modification anomalies. Why are they considered bad? Illustrate with examples.

15.3. Why should NULLs in a relation be avoided as much as possible? Discuss the problem of spurious tuples and how we may prevent it.

15.4. State the informal guidelines for relation schema design that we discussed. Illustrate how violation of these guidelines may be harmful.

15.5. What is a functional dependency? What are the possible sources of the information that defines the functional dependencies that hold among the attributes of a relation schema?

15.6. Why can we not infer a functional dependency automatically from a particular relation state?

15.7. What does the term *unnormalized relation* refer to? How did the normal forms develop historically from first normal form up to Boyce-Codd normal form?

15.8. Define first, second, and third normal forms when only primary keys are considered. How do the general definitions of 2NF and 3NF, which consider all keys of a relation, differ from those that consider only primary keys?

15.9. What undesirable dependencies are avoided when a relation is in 2NF?

15.10. What undesirable dependencies are avoided when a relation is in 3NF?

15.11. In what way do the generalized definitions of 2NF and 3NF extend the definitions beyond primary keys?

15.12. Define Boyce-Codd normal form. How does it differ from 3NF? Why is it considered a stronger form of 3NF?

15.13. What is multivalued dependency? When does it arise?

15.14. Does a relation with two or more columns always have an MVD? Show with an example.

15.15. Define fourth normal form. When is it violated? When is it typically applicable?

15.16. Define join dependency and fifth normal form.

15.17. Why is 5NF also called project-join normal form (PJNF)?

15.18. Why do practical database designs typically aim for BCNF and not aim for higher normal forms?

Exercises

15.19. Suppose that we have the following requirements for a university database that is used to keep track of students' transcripts:

a. The university keeps track of each student's name (Sname), student number (Snum), Social Security number (Ssn), current address (Sc_addr) and

phone (Sc_phone), permanent address (Sp_addr) and phone (Sp_phone), birth date (Bdate), sex (Sex), class (Class) ('freshman', 'sophomore', ... , 'graduate'), major department (Major_code), minor department (Minor_code) (if any), and degree program (Prog) ('b.a.', 'b.s.', ... , 'ph.d.'). Both Ssn and student number have unique values for each student.

b. Each department is described by a name (Dname), department code (Dcode), office number (Doffice), office phone (Dphone), and college (Dcollege). Both name and code have unique values for each department.

c. Each course has a course name (Cname), description (Cdesc), course number (Cnum), number of semester hours (Credit), level (Level), and offering department (Cdept). The course number is unique for each course.

d. Each section has an instructor (Iname), semester (Semester), year (Year), course (Sec_course), and section number (Sec_num). The section number distinguishes different sections of the same course that are taught during the same semester/year; its values are 1, 2, 3, ..., up to the total number of sections taught during each semester.

e. A grade record refers to a student (Ssn), a particular section, and a grade (Grade).

Design a relational database schema for this database application. First show all the functional dependencies that should hold among the attributes. Then design relation schemas for the database that are each in 3NF or BCNF. Specify the key attributes of each relation. Note any unspecified requirements, and make appropriate assumptions to render the specification complete.

15.20. What update anomalies occur in the EMP_PROJ and EMP_DEPT relations of Figures 15.3 and 15.4?

15.21. In what normal form is the LOTS relation schema in Figure 15.12(a) with respect to the restrictive interpretations of normal form that take *only the primary key* into account? Would it be in the same normal form if the general definitions of normal form were used?

15.22. Prove that any relation schema with two attributes is in BCNF.

15.23. Why do spurious tuples occur in the result of joining the EMP_PROJ1 and EMP_ LOCS relations in Figure 15.5 (result shown in Figure 15.6)?

15.24. Consider the universal relation $R = \{A, B, C, D, E, F, G, H, I, J\}$ and the set of functional dependencies $F = \{ \{A, B\}\rightarrow\{C\}, \{A\}\rightarrow\{D, E\}, \{B\}\rightarrow\{F\}, \{F\}\rightarrow\{G, H\}, \{D\}\rightarrow\{I, J\} \}$. What is the key for R? Decompose R into 2NF and then 3NF relations.

15.25. Repeat Exercise 15.24 for the following different set of functional dependencies $G = \{\{A, B\}\rightarrow\{C\}, \{B, D\}\rightarrow\{E, F\}, \{A, D\}\rightarrow\{G, H\}, \{A\}\rightarrow\{I\}, \{H\}\rightarrow\{J\} \}$.

Hashing provides very fast access to an arbitrary record of a file, given the value of its hash key. The most suitable method for external hashing is the bucket technique, with one or more contiguous blocks corresponding to each bucket. Collisions causing bucket overflow are handled by chaining. Access on any nonhash field is slow, and so is ordered access of the records on any field. We discussed three hashing techniques for files that grow and shrink in the number of records dynamically: extendible, dynamic, and linear hashing. The first two use the higher-order bits of the hash address to organize a directory. Linear hashing is geared to keep the load factor of the file within a given range and adds new buckets linearly.

We briefly discussed other possibilities for primary file organizations, such as B-trees, and files of mixed records, which implement relationships among records of different types physically as part of the storage structure. We reviewed the recent advances in disk technology represented by RAID (Redundant Arrays of Inexpensive (or Independent) Disks), which has become a standard technique in large enterprises to provide better reliability and fault tolerance features in storage. Finally, we reviewed three currently popular options in enterprise storage systems: storage area networks (SANs), network-attached storage (NAS), and iSCSI storage systems.

Review Questions

17.1. What is the difference between primary and secondary storage?

17.2. Why are disks, not tapes, used to store online database files?

17.3. Define the following terms: *disk, disk pack, track, block, cylinder, sector, interblock gap, read/write head.*

17.4. Discuss the process of disk initialization.

17.5. Discuss the mechanism used to read data from or write data to the disk.

17.6. What are the components of a disk block address?

17.7. Why is accessing a disk block expensive? Discuss the time components involved in accessing a disk block.

17.8. How does double buffering improve block access time?

17.9. What are the reasons for having variable-length records? What types of separator characters are needed for each?

17.10. Discuss the techniques for allocating file blocks on disk.

17.11. What is the difference between a file organization and an access method?

17.12. What is the difference between static and dynamic files?

17.13. What are the typical record-at-a-time operations for accessing a file? Which of these depend on the current file record?

17.14. Discuss the techniques for record deletion.

17.15. Discuss the advantages and disadvantages of using (a) an unordered file, (b) an ordered file, and (c) a static hash file with buckets and chaining. Which operations can be performed efficiently on each of these organizations, and which operations are expensive?

17.16. Discuss the techniques for allowing a hash file to expand and shrink dynamically. What are the advantages and disadvantages of each?

17.17. What is the difference between the directories of extendible and dynamic hashing?

17.18. What are mixed files used for? What are other types of primary file organizations?

17.19. Describe the mismatch between processor and disk technologies.

17.20. What are the main goals of the RAID technology? How does it achieve them?

17.21. How does disk mirroring help improve reliability? Give a quantitative example.

17.22. What characterizes the levels in RAID organization?

17.23. What are the highlights of the popular RAID levels 0, 1, and 5?

17.24. What are storage area networks? What flexibility and advantages do they offer?

17.25. Describe the main features of network-attached storage as an enterprise storage solution.

17.26. How have new iSCSI systems improved the applicability of storage area networks?

Exercises

17.27. Consider a disk with the following characteristics (these are not parameters of any particular disk unit): block size $B = 512$ bytes; interblock gap size $G = 128$ bytes; number of blocks per track = 20; number of tracks per surface = 400. A disk pack consists of 15 double-sided disks.

a. What is the total capacity of a track, and what is its useful capacity (excluding interblock gaps)?

b. How many cylinders are there?

c. What are the total capacity and the useful capacity of a cylinder?

d. What are the total capacity and the useful capacity of a disk pack?

e. Suppose that the disk drive rotates the disk pack at a speed of 2400 rpm (revolutions per minute); what are the transfer rate (*tr*) in bytes/msec and the block transfer time (*btt*) in msec? What is the average rotational delay (*rd*) in msec? What is the bulk transfer rate? (See Appendix B.)

f. Suppose that the average seek time is 30 msec. How much time does it take (on the average) in msec to locate and transfer a single block, given its block address?

We introduced the concept of a logical index and compared it with the physical indexes we described before. They allow an additional level of indirection in indexing in order to permit greater freedom for movement of actual record locations on disk. We also reviewed some general issues related to indexing, and commented on column-based storage of relations, which has particular advantages for read-only databases. Finally, we discussed how combinations of the above organizations can be used. For example, secondary indexes are often used with mixed files, as well as with unordered and ordered files.

Review Questions

18.1. Define the following terms: *indexing field*, *primary key field*, *clustering field*, *secondary key field*, *block anchor*, *dense index*, and *nondense (sparse) index*.

18.2. What are the differences among primary, secondary, and clustering indexes? How do these differences affect the ways in which these indexes are implemented? Which of the indexes are dense, and which are not?

18.3. Why can we have at most one primary or clustering index on a file, but several secondary indexes?

18.4. How does multilevel indexing improve the efficiency of searching an index file?

18.5. What is the order p of a B-tree? Describe the structure of B-tree nodes.

18.6. What is the order p of a B^+-tree? Describe the structure of both internal and leaf nodes of a B^+-tree.

18.7. How does a B-tree differ from a B^+-tree? Why is a B^+-tree usually preferred as an access structure to a data file?

18.8. Explain what alternative choices exist for accessing a file based on multiple search keys.

18.9. What is partitioned hashing? How does it work? What are its limitations?

18.10. What is a grid file? What are its advantages and disadvantages?

18.11. Show an example of constructing a grid array on two attributes on some file.

18.12. What is a fully inverted file? What is an indexed sequential file?

18.13. How can hashing be used to construct an index?

18.14. What is bitmap indexing? Create a relation with two columns and sixteen tuples and show an example of a bitmap index on one or both.

18.15. What is the concept of function-based indexing? What additional purpose does it serve?

18.16. What is the difference between a logical index and a physical index?

18.17. What is column-based storage of a relational database?

Exercises

18.18. Consider a disk with block size $B = 512$ bytes. A block pointer is $P = 6$ bytes long, and a record pointer is $P_R = 7$ bytes long. A file has $r = 30{,}000$ EMPLOYEE records of *fixed length*. Each record has the following fields: Name (30 bytes), Ssn (9 bytes), Department_code (9 bytes), Address (40 bytes), Phone (10 bytes), Birth_date (8 bytes), Sex (1 byte), Job_code (4 bytes), and Salary (4 bytes, real number). An additional byte is used as a deletion marker.

a. Calculate the record size R in bytes.

b. Calculate the blocking factor bfr and the number of file blocks b, assuming an unspanned organization.

c. Suppose that the file is *ordered* by the key field Ssn and we want to construct a *primary index* on Ssn. Calculate (i) the index blocking factor bfr_i (which is also the index fan-out fo); (ii) the number of first-level index entries and the number of first-level index blocks; (iii) the number of levels needed if we make it into a multilevel index; (iv) the total number of blocks required by the multilevel index; and (v) the number of block accesses needed to search for and retrieve a record from the file—given its Ssn value—using the primary index.

d. Suppose that the file is *not ordered* by the key field Ssn and we want to construct a *secondary index* on Ssn. Repeat the previous exercise (part c) for the secondary index and compare with the primary index.

e. Suppose that the file is *not ordered* by the nonkey field Department_code and we want to construct a *secondary index* on Department_code, using option 3 of Section 18.1.3, with an extra level of indirection that stores record pointers. Assume there are 1,000 distinct values of Department_code and that the EMPLOYEE records are evenly distributed among these values. Calculate (i) the index blocking factor bfr_i (which is also the index fan-out fo); (ii) the number of blocks needed by the level of indirection that stores record pointers; (iii) the number of first-level index entries and the number of first-level index blocks; (iv) the number of levels needed if we make it into a multilevel index; (v) the total number of blocks required by the multilevel index and the blocks used in the extra level of indirection; and (vi) the approximate number of block accesses needed to search for and retrieve all records in the file that have a specific Department_code value, using the index.

f. Suppose that the file is *ordered* by the nonkey field Department_code and we want to construct a *clustering index* on Department_code that uses block anchors (every new value of Department_code starts at the beginning of a new block). Assume there are 1,000 distinct values of Department_code and that the EMPLOYEE records are evenly distributed among these values. Calculate (i) the index blocking factor bfr_i (which is also the index fan-out fo); (ii) the number of first-level index entries and the number of first-level index blocks; (iii) the number of levels needed if we make it into a multilevel index; (iv) the total number of blocks

required by the multilevel index; and (v) the number of block accesses needed to search for and retrieve all records in the file that have a specific Department_code value, using the clustering index (assume that multiple blocks in a cluster are contiguous).

g. Suppose that the file is *not* ordered by the key field Ssn and we want to construct a B^+-tree access structure (index) on Ssn. Calculate (i) the orders p and p_{leaf} of the B^+-tree; (ii) the number of leaf-level blocks needed if blocks are approximately 69 percent full (rounded up for convenience); (iii) the number of levels needed if internal nodes are also 69 percent full (rounded up for convenience); (iv) the total number of blocks required by the B^+-tree; and (v) the number of block accesses needed to search for and retrieve a record from the file—given its Ssn value—using the B^+-tree.

h. Repeat part g, but for a B-tree rather than for a B^+-tree. Compare your results for the B-tree and for the B^+-tree.

18.19. A PARTS file with Part# as the key field includes records with the following Part# values: 23, 65, 37, 60, 46, 92, 48, 71, 56, 59, 18, 21, 10, 74, 78, 15, 16, 20, 24, 28, 39, 43, 47, 50, 69, 75, 8, 49, 33, 38. Suppose that the search field values are inserted in the given order in a B^+-tree of order $p = 4$ and $p_{leaf} = 3$; show how the tree will expand and what the final tree will look like.

18.20. Repeat Exercise 18.19, but use a B-tree of order $p = 4$ instead of a B^+-tree.

18.21. Suppose that the following search field values are deleted, in the given order, from the B^+-tree of Exercise 18.19; show how the tree will shrink and show the final tree. The deleted values are 65, 75, 43, 18, 20, 92, 59, 37.

18.22. Repeat Exercise 18.21, but for the B-tree of Exercise 18.20.

18.23. Algorithm 18.1 outlines the procedure for searching a nondense multilevel primary index to retrieve a file record. Adapt the algorithm for each of the following cases:

a. A multilevel secondary index on a nonkey nonordering field of a file. Assume that option 3 of Section 18.1.3 is used, where an extra level of indirection stores pointers to the individual records with the corresponding index field value.

b. A multilevel secondary index on a nonordering key field of a file.

c. A multilevel clustering index on a nonkey ordering field of a file.

18.24. Suppose that several secondary indexes exist on nonkey fields of a file, implemented using option 3 of Section 18.1.3; for example, we could have secondary indexes on the fields Department_code, Job_code, and Salary of the EMPLOYEE file of Exercise 18.18. Describe an efficient way to search for and retrieve records satisfying a complex selection condition on these fields, such as (Department_code = 5 AND Job_code = 12 AND Salary = 50,000), using the record pointers in the indirection level.

19.6. How many different join orders are there for a query that joins 10 relations?

19.7. What is meant by *cost-based query optimization*?

19.8. What is the difference between *pipelining* and *materialization*?

19.9. Discuss the cost components for a cost function that is used to estimate query execution cost. Which cost components are used most often as the basis for cost functions?

19.10. Discuss the different types of parameters that are used in cost functions. Where is this information kept?

19.11. List the cost functions for the SELECT and JOIN methods discussed in Section 19.8.

19.12. What is meant by semantic query optimization? How does it differ from other query optimization techniques?

Exercises

19.13. Consider SQL queries Q1, Q8, Q1B, and Q4 in Chapter 4 and Q27 in Chapter 5.

a. Draw at least two query trees that can represent *each* of these queries. Under what circumstances would you use each of your query trees?

b. Draw the initial query tree for each of these queries, and then show how the query tree is optimized by the algorithm outlined in Section 19.7.

c. For each query, compare your own query trees of part (a) and the initial and final query trees of part (b).

19.14. A file of 4096 blocks is to be sorted with an available buffer space of 64 blocks. How many passes will be needed in the merge phase of the external sort-merge algorithm?

19.15. Develop cost functions for the PROJECT, UNION, INTERSECTION, SET DIFFERENCE, and CARTESIAN PRODUCT algorithms discussed in Section 19.4.

19.16. Develop cost functions for an algorithm that consists of two SELECTs, a JOIN, and a final PROJECT, in terms of the cost functions for the individual operations.

19.17. Can a nondense index be used in the implementation of an aggregate operator? Why or why not?

19.18. Calculate the cost functions for different options of executing the JOIN operation OP7 discussed in Section 19.3.2.

19.19. Develop formulas for the hybrid hash-join algorithm for calculating the size of the buffer for the first bucket. Develop more accurate cost estimation formulas for the algorithm.