Dr. Stefan Brass
School of Information Sciences
University of Pittsburgh

July 26, 2001

INFSCI 2711 "Database Analysis and Design" — Example I for Final Exam: Solutions —

General Remarks

- The average was 22.2 points out of 36. The maximum reached was 34 points. Every student got 8 bonus points, which moves the average to 30.2.
- Many students said that the time was not sufficient, that they had no idea how the questions would look like, and that query optimization was treated too shortly before the exam in class.
- In retrospecitive, I also regret the system which gives no credit for partially correct answers. If I do "multiple choice" again, I will make sure that only one answer in every exercise has to be checked.
- Of course, I will try to make the final exam for this class better (easier).
- Students also said that they learnt a lot going over this exam again in class. So the questions were actually interesting.

Exercise 1 (SQL Errors) 6 Points

a)
```
SELECT ID, TITLE
FROM   FILM
WHERE  ID NOT IN(SELECT C.ID FROM CRITIQUE C)
```

☒ Correct

A common mistake was to mark this as incorrect because no tuple variable is declared for `FILM` and `ID` is used in the outer query without tuple variable. However, it is legal to leave out the tuple variable name in the `FROM` clause, then the table name `FILM` is used as tuple variable name, too. Also, attribute references need only a tuple variable when they would otherwise be ambiguous. But note that the tuple variable `C` declared in the subquery is not available in the outer query (this is like Pascal's block structure). Therefore, `ID` in the outer query can only refer to `FILM`.

b)
```
SELECT F.ID, F.TITLE
FROM   FILM F, CRITIQUE C
WHERE  F.ID = C.ID AND C.SOURCE IS NULL
```

☒ Wrong, Reason: If there is no critique/review, there will be no join partner. `C.SOURCE` can never be null, since it is part of the key. An outer join would work here.

c)
```
SELECT F.ID, F.TITLE
FROM   FILM F
WHERE  NOT EXISTS(SELECT * FROM CRITIQUE C)
```

☒ Wrong, Reason: This is an uncorrelated subquery, which under `NOT EXISTS` is almost always wrong. Here, if there is at least one `CRITIQUE`, no matter for what film, the output of the query will be empty.

d)
```
SELECT F.ID, F.TITLE
FROM   FILM F
WHERE  F.ID NOT IN (SELECT C.ID FROM CRITIQUE C
                    WHERE GRADE <> 'A')
```

☒ Wrong, Reason: This checks that the film has no reviews with other grades than “A”, but it also prints films without any reviews. It was required that there is at least one review with grade “A”.

e)
```
SELECT F.ID, F.TITLE
FROM   FILM F, CRITIQUE C
WHERE  F.ID = C.ID
AND    C.GRADE = 'A'
HAVING COUNT(DISTINCT GRADE) = 1
```

☒ Wrong, Reason: This is in fact screwed up more than I intended. Syntactically, a `GROUP BY` is missing. You are not allowed to use `F.ID` and `F.TITLE` in the `SELECT`-list (outside aggregation functions) unless they are listed under `GROUP BY`. But even then, reviews with other grades than “A” are simply ignored.

f)
```
SELECT F.ID, F.TITLE
FROM   FILM F, CRITIQUE C
WHERE  F.ID = C.ID AND C.GRADE = 'A'
AND    NOT EXISTS(SELECT * FROM CRITIQUE X
                  WHERE X.ID = F.ID AND X.GRADE <> 'A')
```

☒ Correct

Exercise 2 (Disks and Buffering) 6 Points

a) When the DBMS sends a read request for a single block of 2 KByte to the disk, the total access time consists of three parts. Which one is on average the longest?

☒ Seek time

Seek time is about 10ms, latency time about 4ms, and transfer time (from the disk surface into the disk buffer or also into the computer) is less than 1ms.

b) How many distinct accesses (requiring a seek and the transfer of a few blocks) can a disk carry out per second (approximately)?

☒ 50

Some students checked 500 or more. But this would leave only 2ms for each access, whereas the seek time is already around 10ms. It is true that modern disks might allow a bit more than 50.

c) Suppose your system is short in main memory, but the operating system supports paging (virtual memory, it can swap out pages or entire processes). Would it then improve the DBMS performance to further increase the size of the buffer cache?

☒ The performance will even decrease.

In this case, “double paging” will occur: The DBMS does some paging, but its buffer frames are also paged out by the operating system. This is bad, since the DBMS loses control over the main memory. It has more knowledge about future accesses than the DBMS, but all intelligent buffer replacement strategies have no effect if the buffers are anyway paged out by the OS. In addition, suppose that a database block in a buffer frame was not changed. When the DBMS decides to reuse the buffer frame, it does not have to write the block back to the disk. But the OS must write the block to its swap file, because it doesn’t know that the block is also stored in the database file.

d) What is a good cache hit ratio?

[✓] Above 90%, probably even higher that 95%.

The Oracle Performance Tuning book by Gurry/Corrigan says (p. 470) that one should aim at 95% for OLTP applications and 85% for batch applications and that one should buy more memory if the hit ratio is below 60% and there are no other ways to improving it. The Oracle8 Tuning manual also mentions that one should do something if the hit ratio is less than 60% or 70%, and that one can be satisfied if it is above 90%. I apologize that we haven’t talked about these numbers in class. I though that from the definition of the hit ratio (about which we have talked) it follows “the larger, the better”.

e) Suppose you do RAID level 0+1 (striping and mirroring). You use blockwise striping over 4 disks, and have mirrored the data on another 4 disks. Compared to the number of block read accesses a single disk allows, how would this number increase for the RAID system under optimal circumstances (i.e. equal distribution of requests, perfect load balancing)?

[✓] It would be 8 times higher.

Many students checked 4 times. However, for read accesses, the mirror disks can also be used (whereas write accesses must be done on both copies). The handouts state that reading long files does not become faster for mirroring. However, this looks only at a single request. With mirroring, requests can be executed in parallel.

f) What information is temporarily stored in the buffer cache (in Oracle)? As always, there might be more than one correct answer.

[✓] Blocks containing table data.
[✓] Blocks containing index data.

All data from the data files. Some students checked that also blocks containing program code of the DBMS itself would be contained in this buffer. This is not correct. Probably I should have made clearer that the DBMS buffer cache has nothing to do with the operating system cache. Note that in Oracle parsed SQL queries, PL/SQL procedures, and data dictionary rows are cached not in the database buffer cache, but in the “shared pool”.

Exercise 3 (Access Paths) 6 Points

a) Suppose you accidentally deleted a cluster. Will your application programs still run (maybe slower)?

[✓ They will not run.]

A cluster is not a redundant data structure like an index, but instead contains the actual table rows like a heap file. In a heap file, rows are stored whereever there is space, in a cluster, rows are stored in the block assigned to their value for the cluster attribute. If you delete the cluster, the tables in it will also be gone. However, Oracle will not let you drop a cluster still containing tables (unless you add (“`INCLUDING TABLES`”).

b) Consider again the table `FILM(ID, TITLE, YEAR, DIRECTOR, OSCARS)`.
Suppose you have created a B-tree index with the command

```
CREATE INDEX I_FILM_DIR_YEAR ON FILM(DIRECTOR, YEAR)
```

Which of the following conditions can be evaluated using the index (and not doing a full index scan)? As always, there can be more than one correct answer.

- [✓ `DIRECTOR = 'Lucas, George'`]
- ☐ `YEAR > 1970`
- [✓ `DIRECTOR = 'Lucas, George' AND YEAR > 1980`]
- ☐ `UPPER(DIRECTOR) LIKE 'LU%'`

Note that “`YEAR > 1970`” can not be evaluated using this index, because the main sorting criterion for the index entries is “`DIRECTOR`”. In general, you can use an index on `(A,B)` like an index on “`(A)`”, but you cannot use it like an index on “`(B)`”. “`UPPER(DIRECTOR) LIKE 'LU%'`” also cannot be evaluated using the index. Oracle would have to do four index lookups: `'LU%'`, `'Lu%'`, `'lU%'`, and `'lu%'` (and this number increases exponentially with the length of the search string). In general, whenever you apply a datatype function to an attribute in Oracle, this will make indexes unusable for this attribute.

c) Suppose you have to access 5% of the rows of a large table (significantly larger than your buffer cache). Each block contains about 50 rows. You do not know anything about the distribution of the rows among the blocks (i.e. assume that they are equally distributed). Which access method is more efficient?

- ☒ Full Table Scan

If each block contains 50 rows, 5% are still 2.5 rows on average in every block. So it is very likely that you must anyway fetch every block, and it is much faster to do so in the order in which they are stored on the disk. In addition, since the exercise mentions that the table is larger than your buffer cache, it might even happen that via the index you have to read the same block more than once.

d) Suppose you sometimes want to find films with many Oscars, e.g. have a condition like `OSCARS >= 8` (which is satisfied only by a small fraction of the films). You find that a full-table scan is rather slow. When a new film is entered into the database, the number of Oscars is normally 0, and only later updated when Oscars are granted to the film. Which access method would you propose?

- ☐ Hash Cluster
- ☐ Index Cluster
- ☒ B-tree Index
- ☐ When the column is updated, no index can be created.

Clusters are not possible here, because if a row in a cluster is updated on the cluster column, this will immediately lead to a migrated row. Migrated rows should really be avoided. One book even says "at all costs". A B-tree can be updated. Of course, if there are very many updates and insertions, but only few queries, one must ask whether the index is good. But here it was stated that the response time for this type of query is not acceptable and that something must be done.

e) Suppose you have stored the table `ROLE` in a hash cluster accessed by the film ID. How many block accesses does this query need?

```
SELECT * FROM ROLE WHERE ID = 2456
```

A single row will be not more than 100 Bytes long (shorter than a block).

- ☒ Normally one, but theoretically is no upper limit (0 if in cache).

If there are very many roles stored for a single film, a block might overflow. Also, nothing is said about the hash function. A bad hash function could map all rows to a single block. But I agree that this example was not good, especially since if the

`ID` are assigned consecutive numbers, it would be easy to have a good hash function. The problem with more tuples than expected for a single film remains however.

f)
- ☒ The index is smaller than the table, thus a full scan of the index is faster than a full table scan (useful for index-only QEPs).
- ☒ The partial rows stored in the index are more or less clustered by `ID`.
- ☐ This index enforces the key constraint. No other index on `(ID,SOURCE)` is needed.
- ☐ This index is useful for the condition `GRADE = 'A'`.

The index entries are stored in sorted sequence, and `ID` is the main sorting criterion. So entries with the same value for `ID` will be near to each other (although a block boundery might be in between, and the blocks might not be stored in consecutive disk locations). Some students reported that they were confused by this use of the word "cluster". Note that the index would enforce the key `(ID, SOURCE, GRADE)`, but this key is weaker than `(ID, SOURCE)`. The index is not useful for conditions only on `GRADE`, because this is not the main sorting criterion of the index entries.

Exercise 4 (Heap Files) 1+4+1=6 Points

a) Suppose you delete 50% of the rows stored in a heap file. How long will a full table scan need afterwards in comparison to the time it needed before (in Oracle)?

- ☒ As long as before.

In Oracle, full table scans go always to the high water mark, i.e. all blocks which ever contained rows are read. But even without this very simple solution, it might be that every block contains still contains at least one row. Because we want to guarantee stable ROWIDs, we cannot move rows to other blocks to compactify the disk storage.

b) Every row needs the following space:

- 3 Bytes for the header
- 1 length byte and 4 data bytes for the `ID`. Note that Oracle stores 6 digits in $6/2 + 1 = 4$ bytes.
- 1 length byte and on average 20 data bytes for `TITLE`.
- 1 length byte and 3 data bytes for `YEAR`
- 1 length byte and on average 10 data bytes for `DIRECTOR`.
- 2 Bytes for the row directory entry.

These are 46 Bytes. If the column `OSCARS` is not null, in addition 1 length Byte and 2 data Bytes are needed, giving 49 Byte in total. With 50000 rows of each kind, we get a total of 4750000 Bytes. The block size is 2048, but there is an overhead of 90 Bytes for the block header and 205 Bytes for the space reserve (`PCTFREE`). So each block can only store 1753 Bytes. Thus, we need 4750000/1753=2710 blocks. Students who added 1 block for the segment header got one extra point. The `INITIAL` size must be specified in Bytes (KB/MB), not in blocks. So the right answer is "`INITIAL 5420K`". It is also acceptable to compute first the average number of rows per block (1753/47.5 = 36.9) and then compute the number of blocks as 100000/36.9 = 2710. If you round to 36, you get 2778 blocks.

c) Suppose you expect that the table will grow in one year to 200 000 rows (i.e. double its size). Suppose further that you have sufficient disk space. If you consider the time needed for full table scans, what would be the best solution?

[✓] Define an `INITIAL` extent large enough for 200 000 rows

☐ Define an `INITIAL` extent for the current number of rows and `NEXT` extents for the growth in each quarter. In this way, full table scans done now are not penalized for the future growth.

Full table scans anyway go only to the high water mark. So blocks which do not yet contain rows (and never contained rows) are not read. So there is no penalty for making the extent larger than needed except that the additional disk space is already marked as used. (Well, the distance to other tables stored on the same disk might be a bit larger. But this is a complex matter and normally ignored.) But there is the benefit that even in one year the table will still be stored in a single extent (without need to recreate it).

Exercise 5 (Data Dictionary) 6 Points

a) Write an SQL query which lists tables (not indexes) stored in more than one extent.

```
SELECT SEGMENT_NAME
FROM   USER_SEGMENTS
WHERE  EXTENTS > 1
AND    SEGMENT_TYPE = 'TABLE'
```

An alternative is:

```
SELECT   SEGMENT_NAME
FROM     USER_EXTENTS
WHERE    SEGMENT_TYPE = 'TABLE'
GROUP BY SEGMENT_NAME
HAVING   COUNT(*) > 1
```

A lot of students tried to join `TABS` and `USER_EXTENTS` via `TABLESPACE_NAME`. However, a tablespace contains multiple tables (it is an abstraction of a disk). For student accounts, all your tables are stored in a single tablespace. Then this will simply be a cartesian product.

b) Write an SQL query which lists tables which were never analyzed (i.e. have no statistics) or which were analyzed before `'01-JAN-99'`.

```
SELECT TABLE_NAME
FROM   TABS
WHERE  LAST_ANALYZED < '01-JAN-99'
OR     LAST_ANALYZED IS NULL
```

Note that you had to know here that all your tables are represented in `TABS`, not only the ones which are analyzed. However, if a table was never analyzed (or the statistics were deleted), attributes like `NUM_ROWS` (data gathered via the analysis) and of course `LAST_ANALYZED` are null. Some students wrote `LAST_ANALYZED = NULL`, which would work in SQL Server, but not in Oracle and not in Standard-conform databases. In the standard, any comparison with a null value (except via `IS NULL`), gives the third truth value "unknown" instead of true or false.

c) Write an SQL query which lists tables where the storage capacity (disk space) below the high water mark is utilized to less than 20%. To simplify the task a bit, you can assume that every block has 1753 Bytes of available space (2048 Byte minus 90 Byte for the header and 205 Byte for `PCTFREE=10`). Use the number or rows and average row length contained in `TABS`. Add 2 Bytes to the average row length for the row directory entry. Do not count the segment header block, neither for the used space nor for the allocated space.

```
SELECT TABLE_NAME
FROM   TABS
WHERE  NUM_ROWS * (AVG_ROW_LEN+2)/1753 < 0.2 * BLOCKS
```

The left side of the "<" computes the number of needed blocks. The right side computes 20% of the blocks which are actually used. Of course there are many equivalent formulations, e.g.

```
NUM_ROWS * (AVG_ROW_LEN+2)/(1753*BLOCKS) < 0.20
```

Note that SQL has no "percent" notation. Some students wrote on the right hand side "20%". Also note that the parentheses around (1753*BLOCKS) are actually needed here.

Exercise 6 (Query Optimization) 1+1+1+3=6 Points

a) Consider the following query:

```
SELECT TITLE
FROM   FILM
WHERE  YEAR < 1980
AND    DIRECTOR = 'Lucas, George'
```

Which of the following access paths would the rule-based optimizer choose? Here, only one answer is correct (the highest ranked among the available paths).

- ☐ Full Table Scan
- ☐ Unique Index on FILM(ID)
- ☐ Index on FILM(YEAR)
- ☒ Index on FILM(DIRECTOR)

The unique index on FILM(ID) cannot be used since there is no matching condition in the query. Then the next best access path is the index on FILM(DIRECTOR). This falls in the category "single column index", whereas using the index on FILM(YEAR) would be an "unbounded range scan". A full table scan comes at the very end of the priority list of the rule-based optimizer.

b) Consider this query:

```
SELECT F.TITLE
FROM   FILM F, ROLE R, CRITIQUE C
WHERE  F.ID = R.ID
AND    C.ID = F.ID
AND    R.ACTOR_NAME = 'Ford, Harrison'
AND    C.GRADE = 'A'
```

Consider the QEP constructed by the rule-based optimizer which starts by accessing ROLE. Which table would the rule-based optimizer join first with ROLE?

☒ `FILM`
☐ `CRITIQUE`

Once `ROLE` is accessed, `R.ID` is known, so we can use the index on `FILM(ID)` because of the condition "`F.ID = R.ID`". So we could do an index join here. There is actually no condition which would allow use an index on `C.ID`. Oracle does not automatically derive "`C.ID = R.ID`". But even if we had this condition, using the index on `ROLE(ID, ROLE_NAME)` would be ranked much lower than using the index on `FILM(ID)`. The reason is that we don't have the `ROLE_NAME)` here, so using this index counts as a bounded range search.

c) Suppose we want to compute directors who have made films over a period of 30 years:

```
SELECT X.DIRECTOR
FROM   FILM X, FILM Y
WHERE  X.DIRECTOR = Y.DIRECTOR
AND    X.YEAR - Y.YEAR >= 30
```

Suppose that the only index which exists is on `FILM(ID)`. Which kind of join would the rule-based optimizer of Oracle use (if it starts with accessing `X`)?

☐ Proper Nested Loop Join (i.e. with full table scan on the right)
☒ Merge Join
☐ Index Join with the index on `FILM(ID)`

Oracle uses as join condition here only `X.DIRECTOR = Y.DIRECTOR`. Many students checked nested loop join, because if you consider the full join condition (both parts of the "`AND`"), it is not an equality condition. But all the nested loop join does is to compute the cartesian product, and to check the join condition afterwards. Then it is better to do the merge join using only `X.DIRECTOR = Y.DIRECTOR`, and then to check the other part `X.YEAR - Y.YEAR >= 30` afterwards.

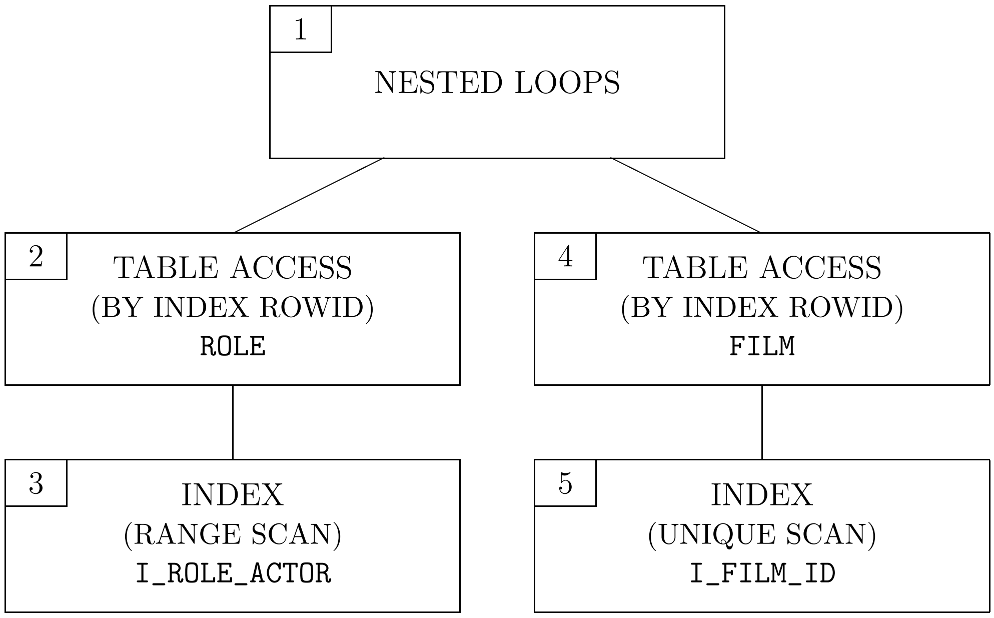
d) Consider the following query:

```
SELECT TITLE
FROM   FILM F, ROLE R
WHERE  YEAR < 1980 AND F.ID = R.ID
AND    ACTOR_NAME = 'Ford, Harrison'
```

The following indexes exist:

- Unique indexes on the keys, i.e. on `FILM(ID)` and on `ROLE(ID, ROLE_NAME)`.
- Index on `ROLE(ACTOR_NAME)`.

Please draw the Oracle QEP which the rule-based optimizer constructs starting with access to ROLE. You can use the tree notation with interconnected boxes or use one line for every operation with indentation to clarify the structure. You do not have to assign numbers to every node.


1
NESTED LOOPS
2
TABLE ACCESS
(BY INDEX ROWID)
ROLE
3
INDEX
(RANGE SCAN)
I_ROLE_ACTOR
4
TABLE ACCESS
(BY INDEX ROWID)
FILM
5
INDEX
(UNIQUE SCAN)
I_FILM_ID