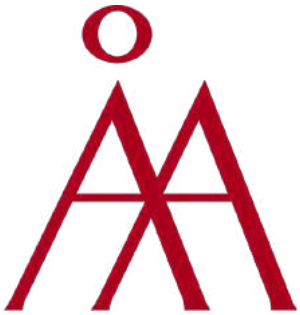


Åbo Akademi University

Advanced Courses in Databases

Learning Journal

Filipe Felício (2004623)

March 24, 2021

Contents

Chapter 1

Introduction

When I had my first databases course last semester in my home university (in Braga, Portugal) my individual emphasis (within the group work I had to do for that particular course) was more into the product management side when accurately translating the high-level business goals into databases' description, control and exploration requirements and later evolving to an EER Diagram and gradually into a Logic Model. I thoroughly enjoy the process of developing a product and in particular turning the clients' expectations into an appropriate software tool. That project was the basis for my professional life in the last months where I have been developing a REST API with Databases for the managements of processes of a real estate company.

This course helped me a lot to mature my fundamentals and really evolve my understatement from an developer/engineer standpoint on databases, relational algebra and SQL.

1.1 Team

The team for the exercises was composed of Filipe Felício (2004623) and Luís Araújo (2004623). The final learning journal was individual.

Chapter 2

Relational Algebra

This chapter was a review of what a previous learned.

2.1 Exercises

For the first question, you should use the relational algebra calculator available on https://dbis-uibk.github.io/relax. Choose the dataset Silberschatz - UniversityDB for this exercise.

2.1.1 1)a)

List the *course_id* and *title* of the courses in the department "Biology" that are worth 4 credit points.

$$\Pi_{course.course\_id,\ course.title}(\sigma_{course.dept\_name\ =\ 'Biology'\ \wedge\ course.credits\ \geq\ 4}(course))$$

Result Grid		
#	course_id	title
1	BIO-101	Intro. to Biology
2	BIO-301	Genetics

Review

In this exercise I made a minor mistake due to lack of attention and resulting bad interpretation of the exercise. Because of that instead of filtering by the courses that are worth 4 credits I filtered the ones that are worth at least 4 credits. It does not seem that it had any impact on the final result since I got two tuples as well. This is what the correct one looks like:

$$\Pi_{course.course\_id,\ course.title}(\sigma_{course.dept\_name\ =\ 'Biology'\ \wedge\ course.credits\ \ 4}(course))$$

2.1.2 1)b)

List the *course_id* and *title* of the courses that the instructor "Srinivasan" teaches.

$$join1 \leftarrow (instructor) \bowtie_{instructor.ID\ =\ teaches.ID} (teaches)$$
$$join2 \leftarrow (join1) \bowtie_{teaches.course\_id\ =\ course.course\_id} (course)$$

$where \leftarrow \sigma_{instructor.name\ =\ 'Srinivasan'}(join2)$
$\Pi_{(course.course\_id,\ course.title)}(where)$

Result Grid		
#	course_id	title
1	CS-101	Intro. to Computer Science
2	CS-315	Robotics
3	CS-347	Database System Concepts

Review

This exercise was correct.

2.1.3 1)c)

List the *student ID*, *name* and *dept_name* of all students that have taken courses held by the instructor "Srinivasan".

$join1 \leftarrow (instructor) \bowtie_{instructor.ID\ =\ teaches.ID} (teaches)$
$join2 \leftarrow (join1) \bowtie_{teaches.course\_id\ =\ course.course\_id} (course)$
$where \leftarrow \sigma_{instructor.name\ =\ 'Srinivasan'}(join2)$
$join3 \leftarrow (student) \bowtie_{student.ID\ =\ takes.ID\_id} (takes)$
$join4 \leftarrow (where) \bowtie_{teaches.course\_id\ =\ takes.course\_id} (join3)$
$\Pi_{(course.course\_id,\ course.title)}(join4)$

Result Grid			
#	ID	name	dept_name
1	128	Zhang	Comp. Sci.
2	12345	Shankar	Comp. Sci.
3	45678	Levy	Physics
4	54321	Williams	Comp. Sci.
5	76543	Brown	Comp. Sci.
6	98765	Bourikas	Elec. Eng.

Review

I believe that this exercise was correct, since I got 6 tuples and delivered on the requested collumns. I have made a typing mistake on the parameters of the SELECT and left the ones from the previous query when I was copying my work from the relational algebra calculator.

In addition, by analysing the provided solution, I realized that I have made one more join than necessary trying to tie the students and the instructor "Srinivasan". I have neglected that through the ID's in the table teaches and the table takes I could achieve the desired result with less joins, and by consequence did not need the table course.

This is what the solution looks like:

$join1 \leftarrow (takes) \bowtie_{takes.ID\ =\ student.ID} (student)$
$join2 \leftarrow (teaches) \bowtie_{teaches.course\_id\ =\ takes.course\_id} (join1)$
$join3 \leftarrow (instructor) \bowtie (join2)$
$where \leftarrow \sigma_{instructor.name\ =\ 'Srinivasan'}(join3)$
$\Pi_{(student.ID,\ student.name, student.dept\_name)}(where)$

2.1.4 1)d)

List the *name*, *dept_name* and *salary* of the instructors that have the highest salary.

$\tau_{[3],\ desc}(\pi_{instructor.name,\ instructor.dept\_name,\ instructor.salary}(instructor))$

Result Grid			
#	name	dept_name	salary
1	Einstein	Physics	95000
2	Brandt	Comp. Sci.	92000
3	Wu	Finance	90000
4	Gold	Physics	87000
...			
12	Mozart	Music	40000

Review

In this exercise I have made a mistake when I rank order teachers based on their salary instead of selecting the teachers that earned the highest salary. This is what a solution looks like:

$group\_by \leftarrow \gamma_{max(salary)\ ->\ maxsalary}(instructor)$
$join \leftarrow (instructor) \bowtie instructor.salary\ =\ maxsalary(group\_by)$
$\Pi_{(name,\ dept\_name,\ salary)}(join)$

2.1.5 1)e)

List the instructor *ID*, *name*, *department* and the total number of students that each instructor supervises.

$join \leftarrow (advisor) \bowtie advisor.s\_id\ =\ instructor.ID(instructor)$
$\gamma_{instructor.ID,\ instructor.name,\ instructor.dept\_name;\ count(advisor.s\_id)\ ->\ number\_students}(join)$

Result Grid				
#	ID	name	dept_name	number_students
1	45565	Katz	Comp. Sci.	2
2	10101	Srinivasan	Comp. Sci.	1
3	76543	Singh	Finance	1
4	22222	Einstein	Physics	2
5	98345	Kim	Elec. Eng.	2
6	76766	Crick	Biology	1

Review

This exercise was correct.

2.1.6 1)f)

List the *course_id*, *title* and *name* of the instructor for all courses that were held in the building 'Packard' in 2010.

$join1 \leftarrow (instructor) \bowtie_{instructor.ID\ =\ teaches.ID} (teaches)$
$join2 \leftarrow (join1) \bowtie_{teaches.course\_id\ =\ course\_id} course$
$join3 \leftarrow (join2) \bowtie_{instructor.dept\_name\ =\ department.dept\_name} department$
$where \leftarrow \sigma_{department.building\ =\ 'Packard'\ \wedge\ teaches.year\ =\ 2010}(join3)$
$\Pi_{course.course\_id,\ course.title,\ instructor.name}(where)$

Result Grid			
#	course_id	title	name
1	BIO-101	Intro. to Biology	Mozart
2	BIO-301	Genetics	Mozart
3	BIO-399	Computational Biology	Mozart
4	CS-101	Intro. to Computer Science	Mozart
...			
13	PHY-101	Physical Principles	Mozart

Review

In this exercise I have made a mistake by doing the where after joining the tables and not only in the department table. This is what the solution looks like:

$where \leftarrow \sigma_{department.building\ =\ 'Packard'\ \wedge\ teaches.year\ =\ 2010}(department)$
$join1 \leftarrow (instructor) \bowtie_{instructor.ID\ =\ teaches.ID} (teaches)$
$join2 \leftarrow (join1) \bowtie_{teaches.course\_id\ =\ course\_id} course$
$join3 \leftarrow (join2) \bowtie_{instructor.dept\_name\ =\ department.dept\_name} where$
$\Pi_{course.course\_id,\ course.title,\ instructor.name}(where)$

Chapter 3

SQL

This chapter was a review of what a previous learned.

For the second question you should use a MySQL database server, either the one available at `babbage2.abo.fi` or your own MySQL server.

First and foremost we import the desired database:

```
/Select the database 'university' in the SQL schema./
USE university;
```

3.1 Exercises

3.1.1 2)a)

List the *course_id* and *title* of the courses in the department "Finance" that are worth 4 credit points.

```
SELECT
    course_id, title
FROM
    course
WHERE
    dept_name = 'Finance' AND credits = 4;
```

Result Grid		
#	course_id	title
1	304	Music 2 New for your Instructor
2	319	World History
3	349	Networking
4	362	Embedded Systems
5	586	Image Processing
6	781	Compiler Design
7	922	Microeconomics

Review

This exercise was correct.

3.1.2 2)b)

List the *course_id* and *title* of the courses that the instructor "Dale" teaches.

```sql
SELECT DISTINCT
    course.course_id, title
FROM
    course
        JOIN
    teaches ON course.course_id = teaches.course_id
        JOIN
    instructor ON teaches.ID = instructor.ID
WHERE
    name = 'Dale';
```

Result Grid		
#	course_id	title
1	158	Elastic Structures
2	237	Surfing
3	496	Aquatic Chemistry
4	629	Finite Element Analysis
5	748	Tort Law
6	802	African History
7	893	Systems Software
8	927	Differential Geometry

Review

This exercise was correct. I could have used the command USING instead of the command ON since the name of both pairs of columns where I performed the join is the same. With that syntax this is what it looks like:

```sql
SELECT DISTINCT
    course.course_id, title
FROM
    course
        JOIN
    teaches USING (course_id)
        JOIN
    instructor USING ID
WHERE
    name = 'Dale';
```

3.1.3 2)c)

Which courses have the student 'Tuomisto' passed? The result should include the *course_id*, *course title*, *grade* and number of *credits* of the course.

```sql
SELECT
    course.course_id, title, grade, credits
FROM
    student
        JOIN
    takes ON student.ID = takes.ID
        JOIN
    course ON takes.course_id = course.course_id
WHERE
    name = 'Tuomisto';
```

Result Grid				
#	course_id	title	grade	credits
1	239	The Music of the Ramones	A+	4
2	345	Race Car Driving	A+	4
3	362	Embedded Systems	B-	4
4	400	Visual BASIC	A	4
5	408	Bankruptcy	C	3
6	545	International Practicum	B-	3
7	571	Plastics	C	4
8	612	Mobile Computing	B	3
9	692	Cat Herding	C	3
10	702	Arabic	C+	3
11	760	How to Groom your Cat	A	3
12	795	Death and Taxes	A	3
13	808	Organic Chemistry	B+	4
14	852	World History	C+	4
15	875	Bioinformatics	B	3
16	893	Systems Software	B	3
17	974	Astronautics	A+	3

Review

In this exercise, like in the previous one, I could have used the syntax USING instead of ON. Also I forgot to filter by grades above 'F' and remove the entries which had the value on the column 'grade' was null. This is what a solution with those changes looks like:

```
SELECT
    course.course_id, title, grade, credits
FROM
    student
        JOIN
    takes USING (ID)
        JOIN
    course USING (course_id)
WHERE
    name = 'Tuomisto' and grade <> 'F'
    and grade is not null;
```

3.1.4 2)d)

List all students that have taken courses by the instructor 'Morris'. The result should contain the student *ID* and *name*, and the *course_id* and course *title*, and be ordered alphabetically on student *name*.

```
SELECT
    student.id, student.name, course.course_id, course.title
FROM
    instructor
        JOIN
    teaches ON instructor.id = teaches.id
        JOIN
    course ON teaches.course_id = course.course_id
        JOIN
    takes ON teaches.course_id = takes.course_id
        JOIN
    student ON takes.id = student.id
WHERE
    instructor.name = 'Morris'
ORDER BY student.name;
```

Result Grid				
#	ID	name	course_id	title
1	5144	Abdellatif	795	Death and Taxes
2	5144	Abdellatif	313	International Trade
3	78858	Abdul-Rahman	242	Rock and Roll
4	20244	Abu-B	791	Operating Systems
5	83622	Achilles	696	Heat Transfer
...				
1475	38121	Zuyev	313	International Trade

In order to check if this query was correct I split it in two queries. The first get the courses that the instructor is 'Morris' *ID*'s and second get the requested information based on the *ID*s

```
SELECT DISTINCT
    course.course_id
FROM
    course
        JOIN
    teaches ON course.course_id = teaches.course_id
        JOIN
    instructor ON teaches.ID = instructor.ID
WHERE
    name = 'Morris';
```

Result Grid	
#	course_id
1	242
2	313
3	696
4	791
5	795

```
SELECT student.ID, name, course.course_id, course.title
FROM
    student
        JOIN
    takes ON student.ID = takes.ID
        JOIN
    course ON takes.course_id = course.course_id
WHERE
    course.course_id IN ('242','313','696','791','795')
ORDER BY name;
```

Result Grid				
#	ID	name	course_id	title
1	5144	Abdellatif	795	Death and Taxes
2	5144	Abdellatif	313	International Trade
3	78858	Abdul-Rahman	242	Rock and Roll
4	20244	Abu-B	791	Operating Systems
5	83622	Achilles	696	Heat Transfer
...				
1475	38121	Zuyev	313	International Trade

Review

This exercise was correct.

3.1.5 2)e)

List the *name*, *dept_name* and *salary* of the instructors that have the highest salary.

```
SELECT
    name, dept_name, salary
FROM
    instructor
ORDER BY salary DESC;
```

Result Grid			
#	name	dept_name	salary
1	Wieland	Pol. Sci.	124651.41
2	Voronina	Physics	121141.99
3	Mird	Marketing	119921.41
4	Sakurai	English	118143.98
5	Bietzk	Cybernetics	117836.50
...			
50	Lembr	Accounting	32241.56

Review

In this exercise I made a mistake where I rank order the teacher by salary instead of selecting the one that earned the most. One solution looks like this:

```
SELECT name, dept_name, salary
FROM instructor
WHERE salary = (SELECT MAX(salary) FROM instructor);
```

3.1.6 2)f)

List the instructor *ID*, *name*, *department* and the total number of students that each instructor supervises.

```
SELECT
    instructor.ID,
    name,
    dept_name,
    COUNT(takes.ID) AS number_students
FROM
    instructor
        JOIN
    teaches ON instructor.ID = teaches.ID
        JOIN
    takes ON teaches.course_id = takes.course_id
GROUP BY instructor.ID;
```

Later I found the table advisor that connects directly students and instructors, so where is an alternative way to do it:

```
SELECT
    instructor.ID,
    name,
    dept_name,
    COUNT(advisor.s_ID) AS number_students
FROM
    instructor
        JOIN
```

```
    advisor ON instructor.ID = advisor.i_ID
GROUP BY instructor.ID;
```

Result Grid				
#	ID	name	dept_name	number_students
1	14365	Lembr	Accounting	870
2	15347	Bawa	Athletics	266
3	19368	Wieland	Pol. Sci.	910
4	22591	DAgostino	Psychology	5391
5	25946	Liley	Languages	338
...				
31	99052	Dale	Cybernetics	3658

Review

This exercise was correct.

Chapter 4

File organization, Indexing and B+ trees

4.1 Exercises

Assume that the relation student is defined in the following way:

```
CREATE TABLE student (
    ID CHAR(5),
    name VARCHAR(20) NOT NULL,
    dept_name VARCHAR(20),
    tot_cred NUMERIC(3 , 0 ),
    PRIMARY KEY (ID),
    FOREIGN KEY (dept_name)
        REFERENCES department (dept_name)
);
```

4.1.1 1)

Draw a figure illustrating how the three following records of type student would be stored with the variable-length record representation. You can assume that length and offset values are stored as integer values using 2 bytes each (i.e. as values of type unsigned short int) and that the attribute tot_cred with the type numeric(3,0) is of fixed size and stored in 8 bytes.

ID	name	dept_name	tot_cred
78858	Abdul-Rahman	Psychology	49
11422	Saito	Physics	34
68150	Kim	Math	NULL

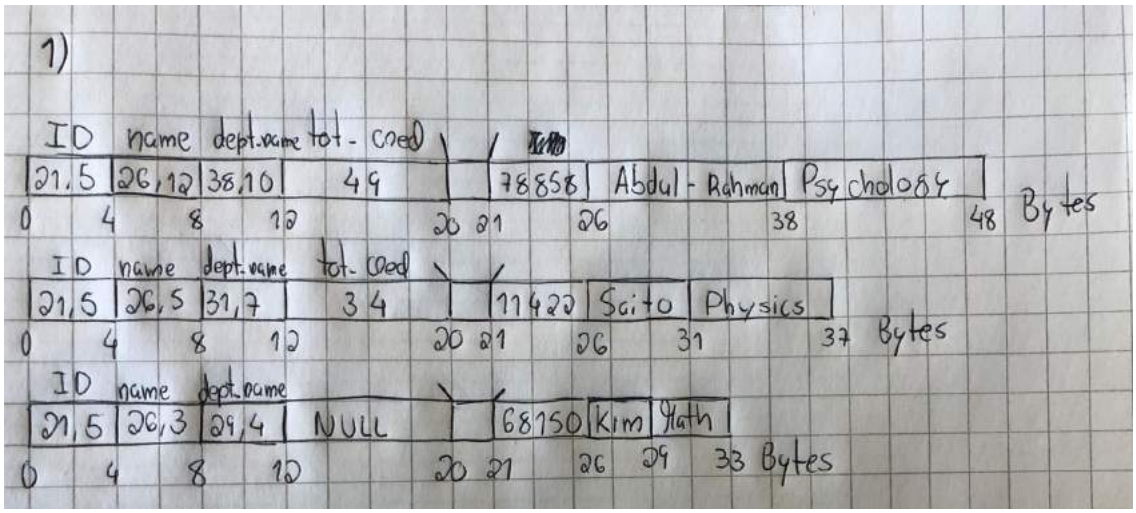


Figure 4.1: Exercise 1

Review

Here we got the ID wrong, being that it has a constant number of bytes that takes, it should be on the left part.

4.1.2 2)

Draw a figure describing how the above three records would be stored in a slotted page structure with a block size of 256 bytes, if they were inserted in the order they appear in the table. As before, we assume that lengths and offset values use 2 bytes.

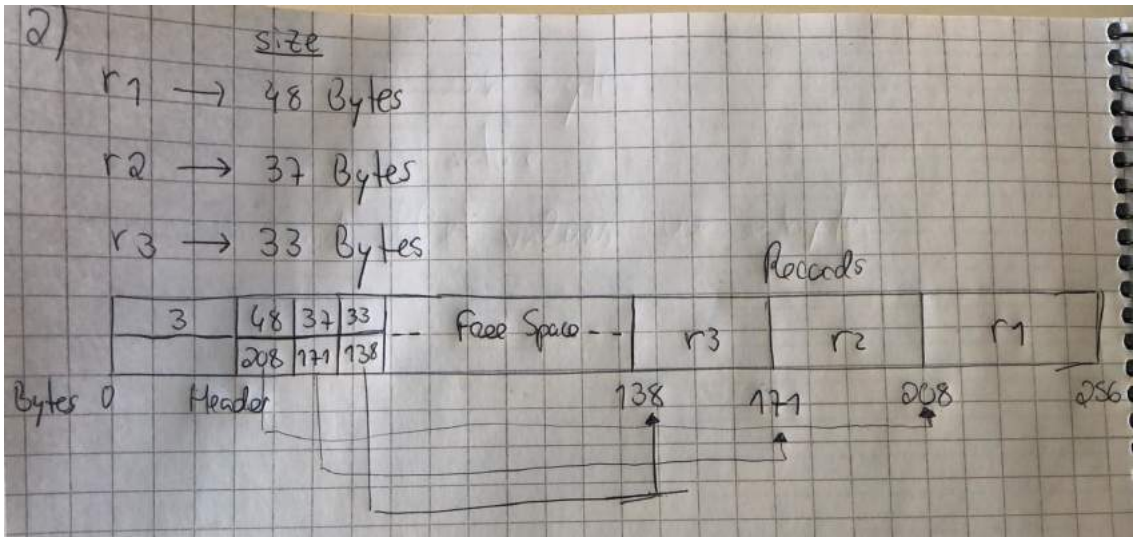


Figure 4.2: Exercise 2

Review

The mistake comes from the previous error.

4.1.3 3.a)

What is the difference between a primary index (also called a clustering index) and a secondary index (also called a non-clustering index)?

Answer: In a sequentially ordered file, the difference between a primary index and a secondary index is that the first's search key specifies the sequential order of the file and the search key of this index is usually but not necessarily the primary key, whilst the second's specifies a different order from the sequential order of the file.

Review

This exercise was correct.

4.1.4 3.b)

What is the difference between a dense index and a sparse index?

Answer: In a dense index, for every search-key value exists a index record, while in a sparse index only some search-keys have index-records and to locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for. There is a trade-off where on the first you need more space (because there is a index record for every search-key) but locating a record is more efficient and on the second you need less space but it is less efficient searching for a record.

Review

This exercise was correct.

4.1.5 3.c)

Is it possible in general to have two clustering indices on the same relation for different search keys?

Answer: In general, it is not possible to have two primary indices on the same relation for different keys because the tuples in a relation would have to be stored in different order to have same values stored together.

Review

This exercise was correct.

4.1.6 3.d)

Why are sparse secondary indices never used?

Answer: Secondary indices have to be dense, since the records in the file are not ordered on the search key of the index. Which means, secondary sparse indices are never used because all records in file are not ordered on the search key of the index.

Review

This exercise was correct.

4.1.7 3.e)

If indices make file accesses efficient, why don't we always define an index on all attributes in a relation?

Answer: It is impossible to define an index on all attributes of a relation because there will be conflict of search. A key attribute must be defined by an index. Example: The data of a particular instructor can be accessed by his/ her unique attribute eg. ID of the instructor. It's not possible to define his/ her ID, Salary, Department etc as an index or search key

Review

Indices need to be kept up-to-date when tuples are inserted, deleted or modified in the database, and this causes additional work for each index. Each index also requires additional storage space. Indices on keys other than the primary key (usually non-clustering indices) are also more likely to be updated. Primary key values seldom change, but other attributes may change.

4.1.8 4.a)

Construct a B+-tree with a fan-out of 4 for the following set of key values, inserted in the given order:
3, 4, 6, 12, 18, 20, 24, 26, 30, 32, 36, 40
For each insertion operation, try to predict what will happen with the leaf nodes and its parent node:

insert 3

Prediction: The tree starts empty so we just create a node (root node) and insert the 3.

Figure 4.3: insert 3

Result:

Figure 4.4: insert 4

insert 4

Prediction: We insert the 4 in the root node, right of the 3.

Figure 4.5: insert 4

Result:

Figure 4.6: insert 4

insert 5

Prediction: We insert the 6 in the root node, right of the 4.

Figure 4.7: insert 5

Result:

Figure 4.8: insert 5

insert 12

Prediction: We split the root node into 2 leaf nodes. On the left leaf we put the 3 and the 4. On the right leaf we put the 6 and the 12. On the new parent we just insert the 6.

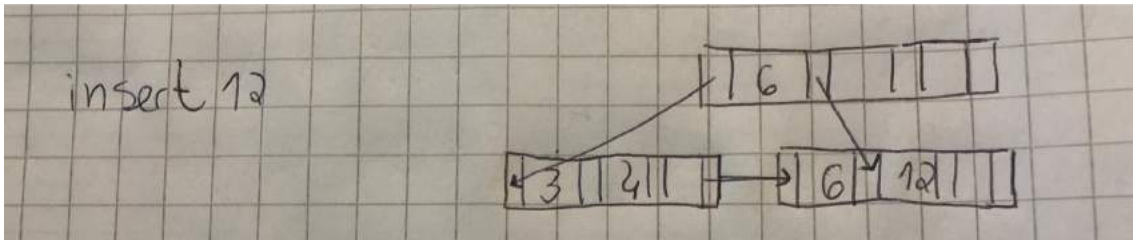


Figure 4.9: insert 12

Result:

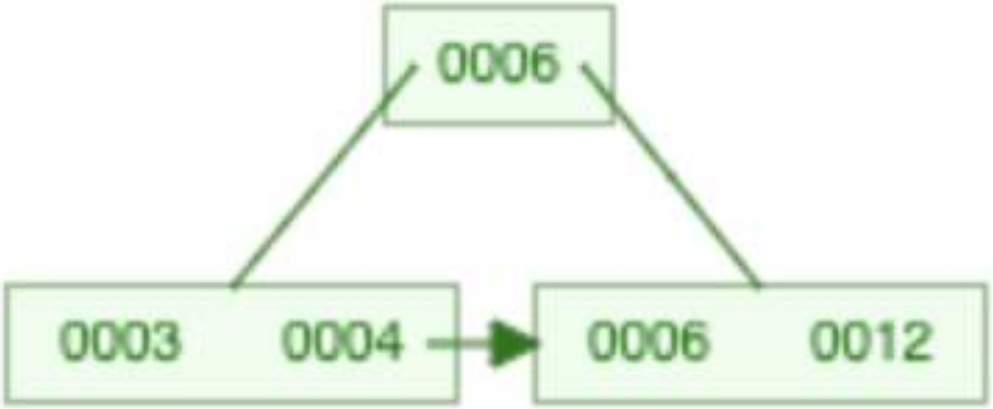


Figure 4.10: insert 12

insert 18

Prediction: We insert the 18 on the right leaf node on the right of the 12.

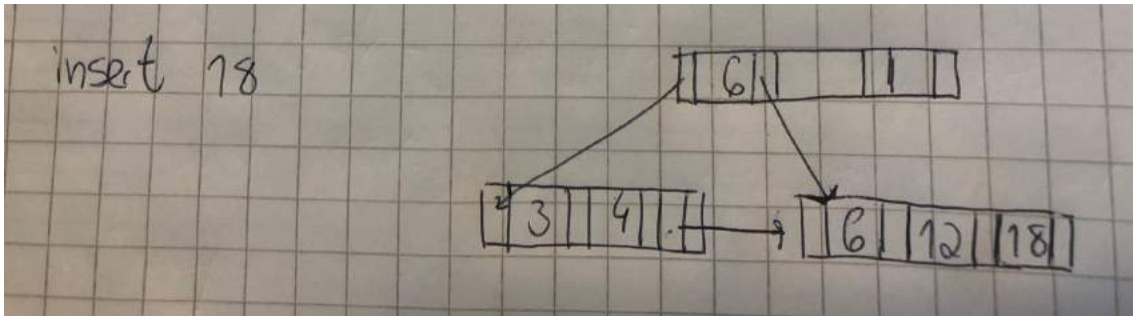


Figure 4.11: insert 18

Result:

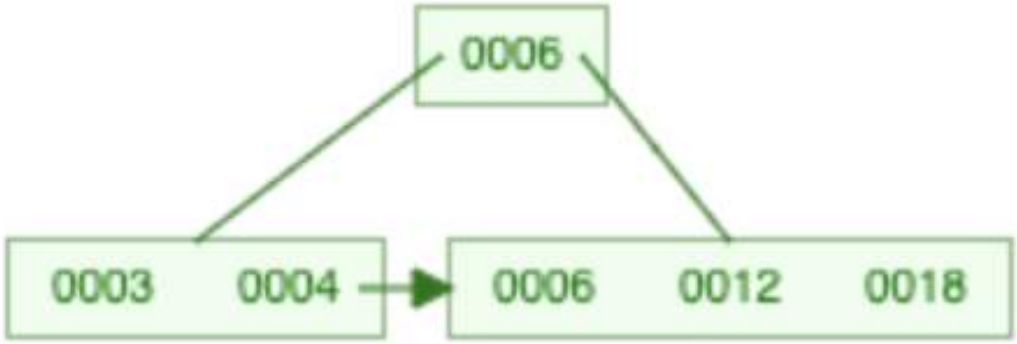


Figure 4.12: insert 18

insert 20

Prediction: We split the right node create a new leaf node. We insert the 18 on the parent node. We leave 6 and 12 on the middle leaf node and insert 18 and 20 on the new leaf node.

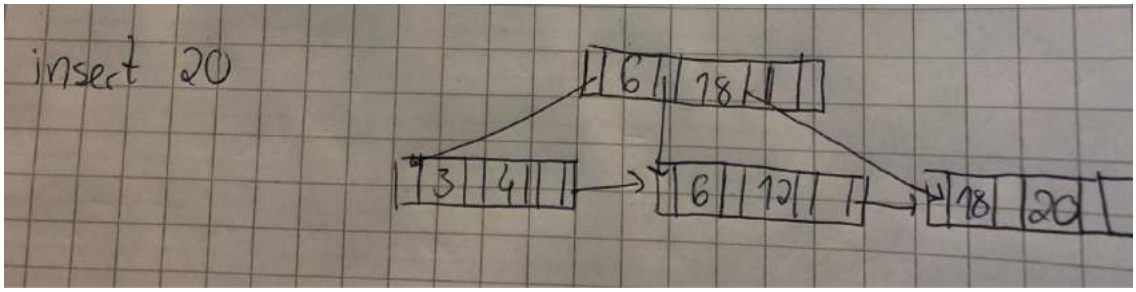


Figure 4.13: insert 20

Result:

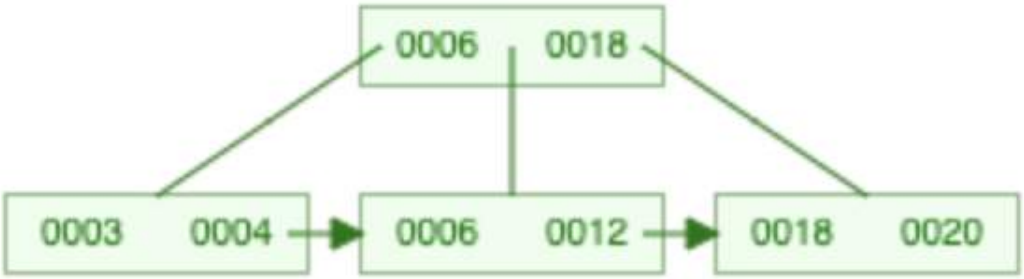


Figure 4.14: insert 20

insert 24

Prediction: We insert the 24 in the right leaf node, just by the 20.

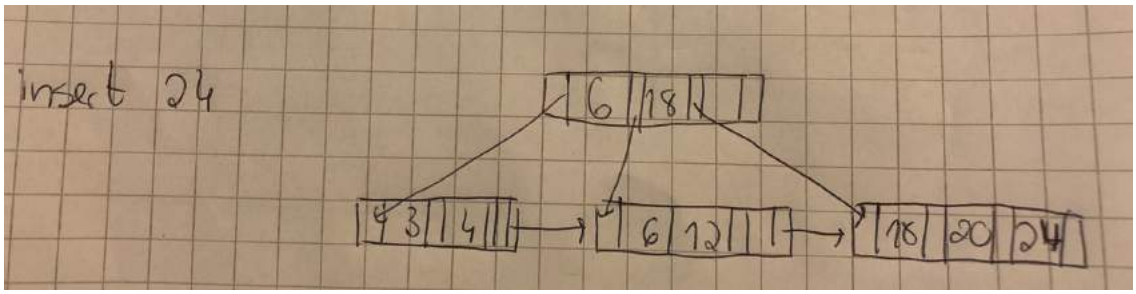


Figure 4.15: insert 24

Result:

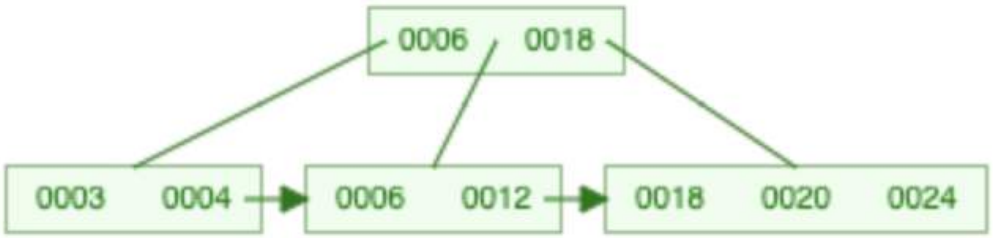


Figure 4.16: insert 24

insert 26

Prediction: We split the right leaf node, on the that node we leave the 18 and the 20. On the new leaf node we insert the 24 and 26. On the parent (root) node we insert the 24.

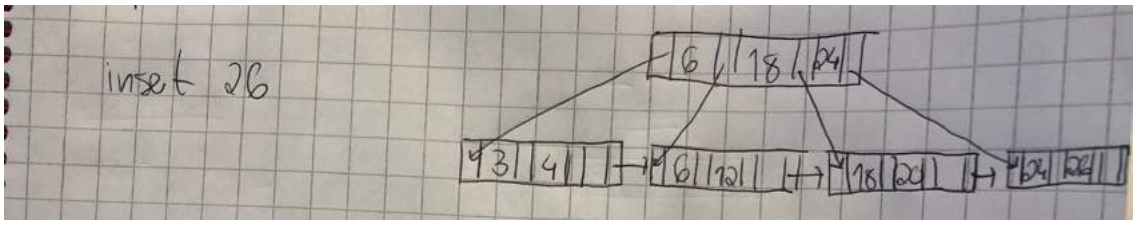


Figure 4.17: insert 26

Result:

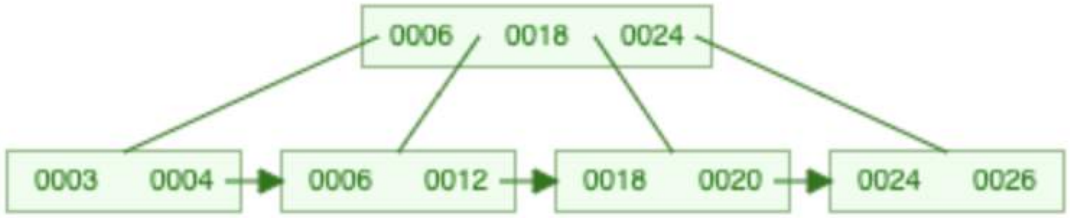


Figure 4.18: insert 26

insert 30

Prediction: We insert the 30 in the 4th leaf node (from the left to the right), just by the 30.

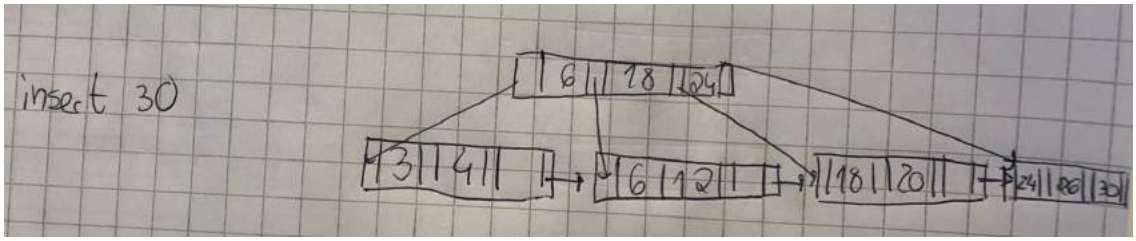


Figure 4.19: insert 30

Result:

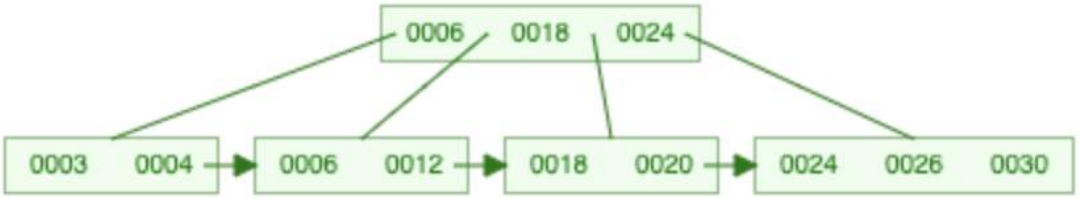


Figure 4.20: insert 30

insert 32

Prediction: We split the parent node (root), leaving only the 24. On the two child nodes (that are going to act as parents nodes to the leaf nodes) we are going to leave the 16 and 18 (on the left) and are going to insert the 30 on the right. On the leaf nodes we split the 4th leaf node (from the left to the right), leaving the 24 and the 26. On the new node we insert the 30 and the 32.

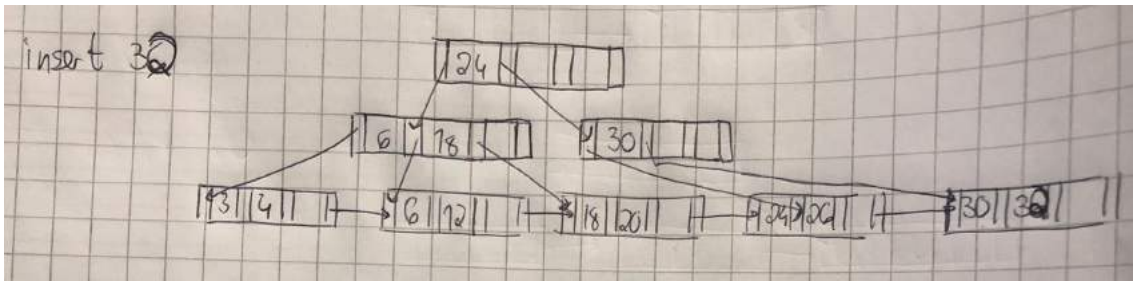


Figure 4.21: insert 32

Result:

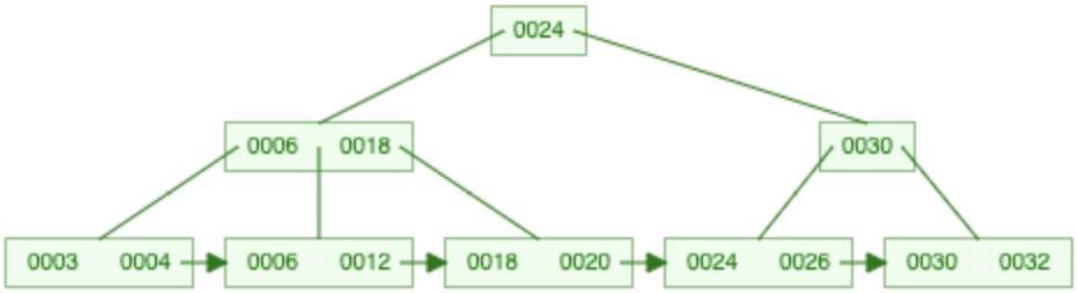


Figure 4.22: insert 32

insert 36

Prediction: We insert the 36 in the 5th leaf node (from the left to the right), just by the 32.

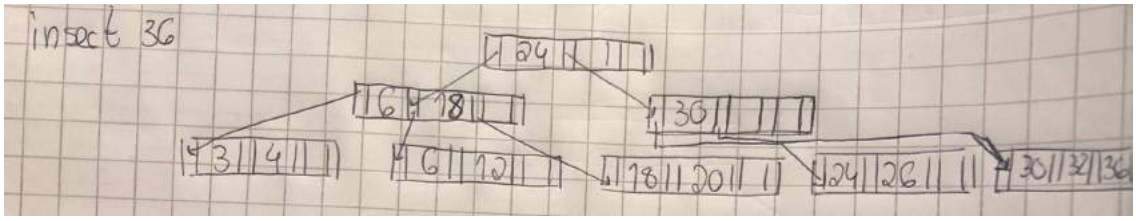


Figure 4.23: insert 36

Result:

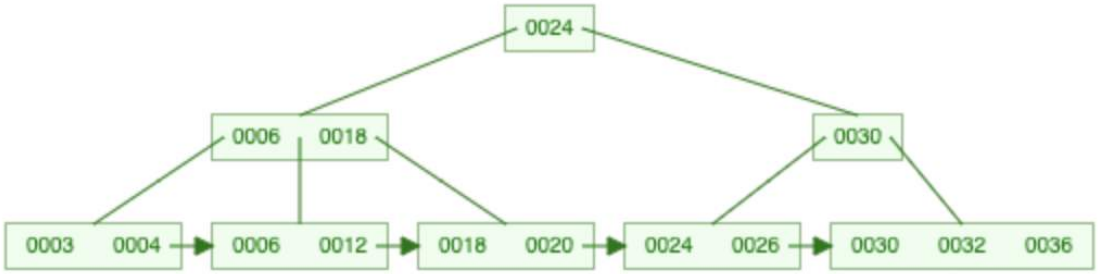


Figure 4.24: insert 36

insert 40

Prediction: We split the 5th leaf node (from the left to the right), leaving the 30 and the 32. On the new node we insert the 36 and the 40. On the parent node we insert the 36.

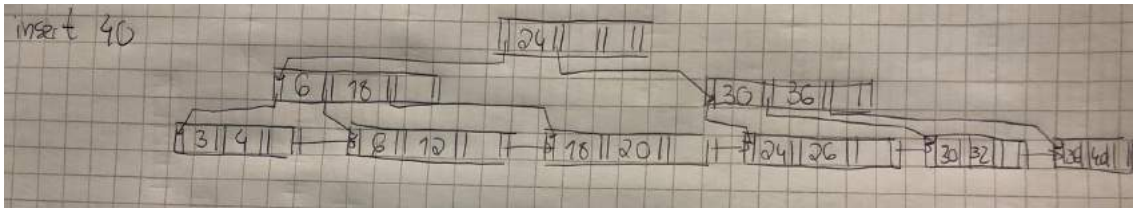


Figure 4.25: insert 40

Result:

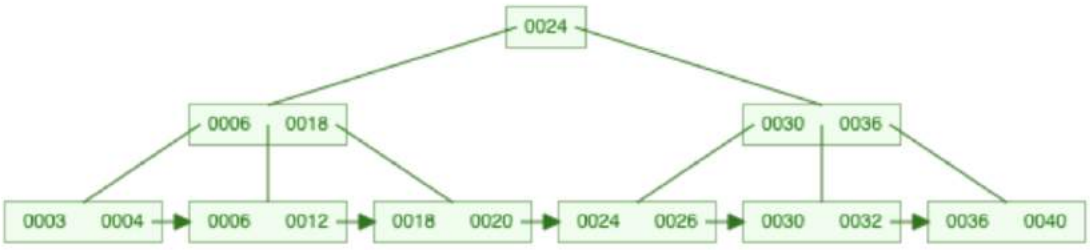


Figure 4.26: insert 40

4)b)

Construct a B+-tree with the same fan-out and the same set of nodes, but inserted in an other order:

18, 6, 36, 24, 32, 12, 3, 30, 4, 40, 20, 26

What are the main differences between these two B+-tree representations? For what kind of database operations do you think these index trees would be efficient?

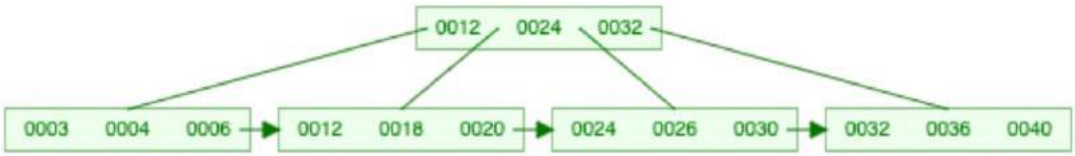


Figure 4.27

Answer: The most noticeable difference in the these two B+-tree representations is the number of levels and the occupancy of the nodes.

- For the operation of inserting the 1st tree will be more efficient because in the 2nd the root (parent) node is full meaning that a split inevitably will occur.
- For the operation of deleting the 2nd tree is more efficient due to the fact that in the first a deletion would imply a sibling merge because all nodes (except the root), if an element were to be removed, when added the number of nodes with a sibling could reach full occupancy.

Review

This exercise was correct.

5)

What will the occupancy of the leaf nodes be (i.e. the degree to which the leaf nodes are filled), if the index entries are inserted in ascending sorted order? What if the entries are inserted in descending order?

Answer: When talking about the occupancy leaf nodes in this data structure (B+-tree) the pattern that in notice after consecutive insertions of index entries in ascending sorted order is that in the leaf node most to right it stacks with search keys until that node is full and needs to split and in the order remaining nodes it stays with 2 elements. In the same scenario but with consecutive insertions of index entries in a descending sorted order we notice the same pattern in the leaf except that we observe this filling on the node most to left instead of the one most to the right.

Review

This exercise was correct.

Chapter 5

Hashing and Bitmaps

5.1 Exercises

5.1.1 1)

Assume that we have an extendable hash index on a file with an integer search key, and that the hash function is h(x) = x mod 8 (i.e. the hash value is in the range [0,7]) and a bucket can contain at most three records. Draw an illustration of how the extendable hash structure develops, step by step,when inserting records with the following search key values:
2, 3, 5, 7, 11, 17, 19, 23, 29, 31

Insert 2, 3 and 5

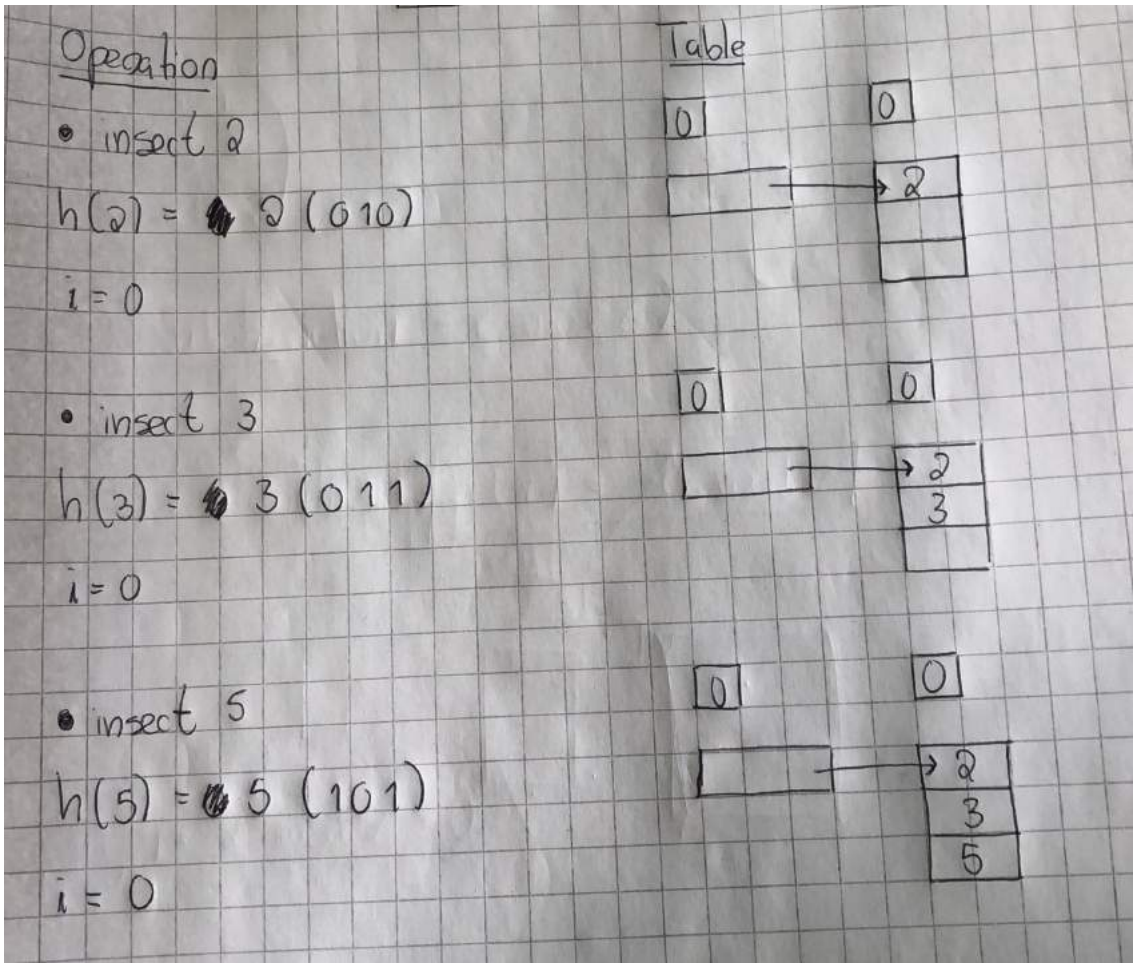


Figure 5.1: Insert 2, 3 and 5

Insert 7, 11 and 17

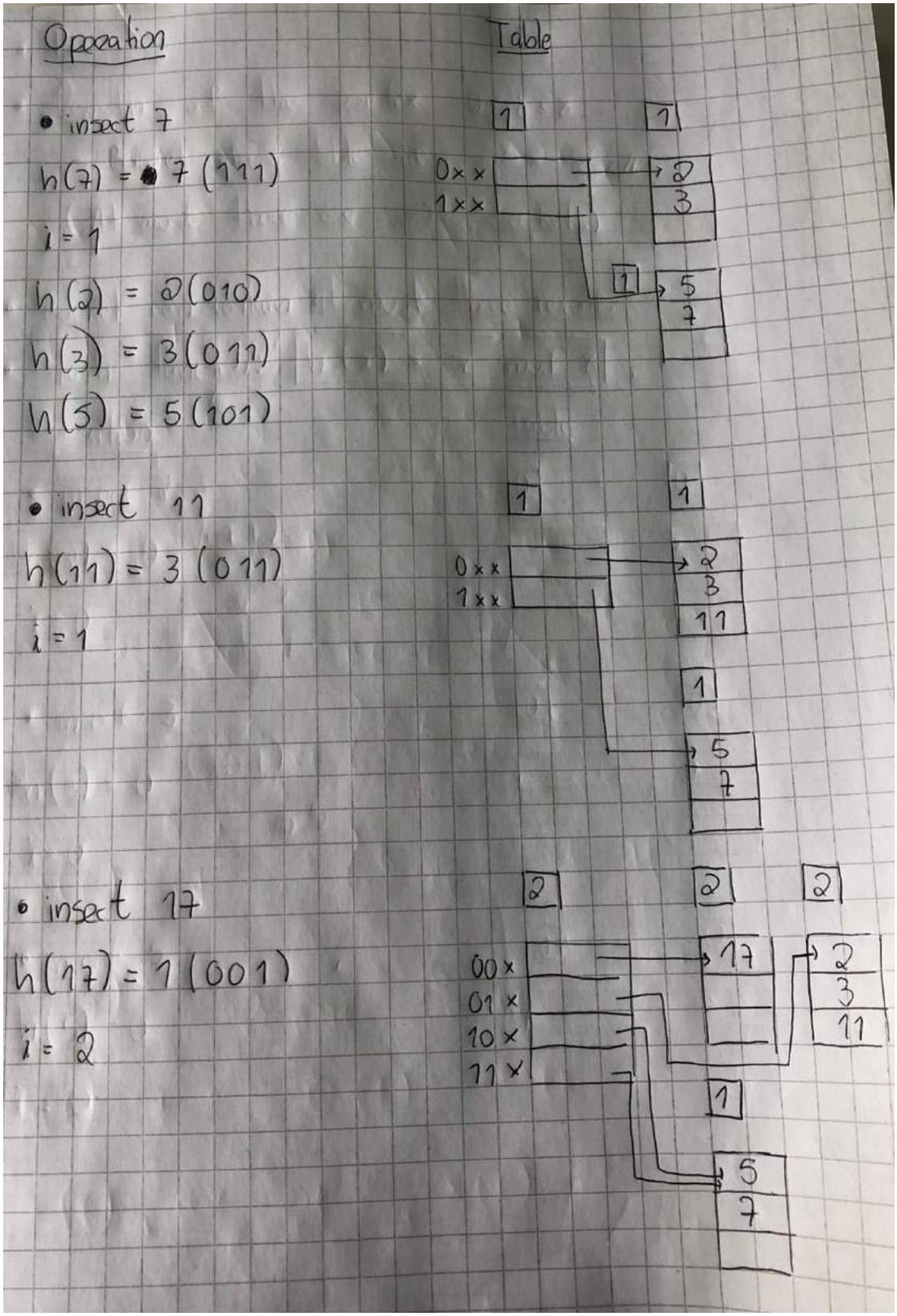


Figure 5.2: Insert 7, 11 and 17

Insert 19, 23 and 29

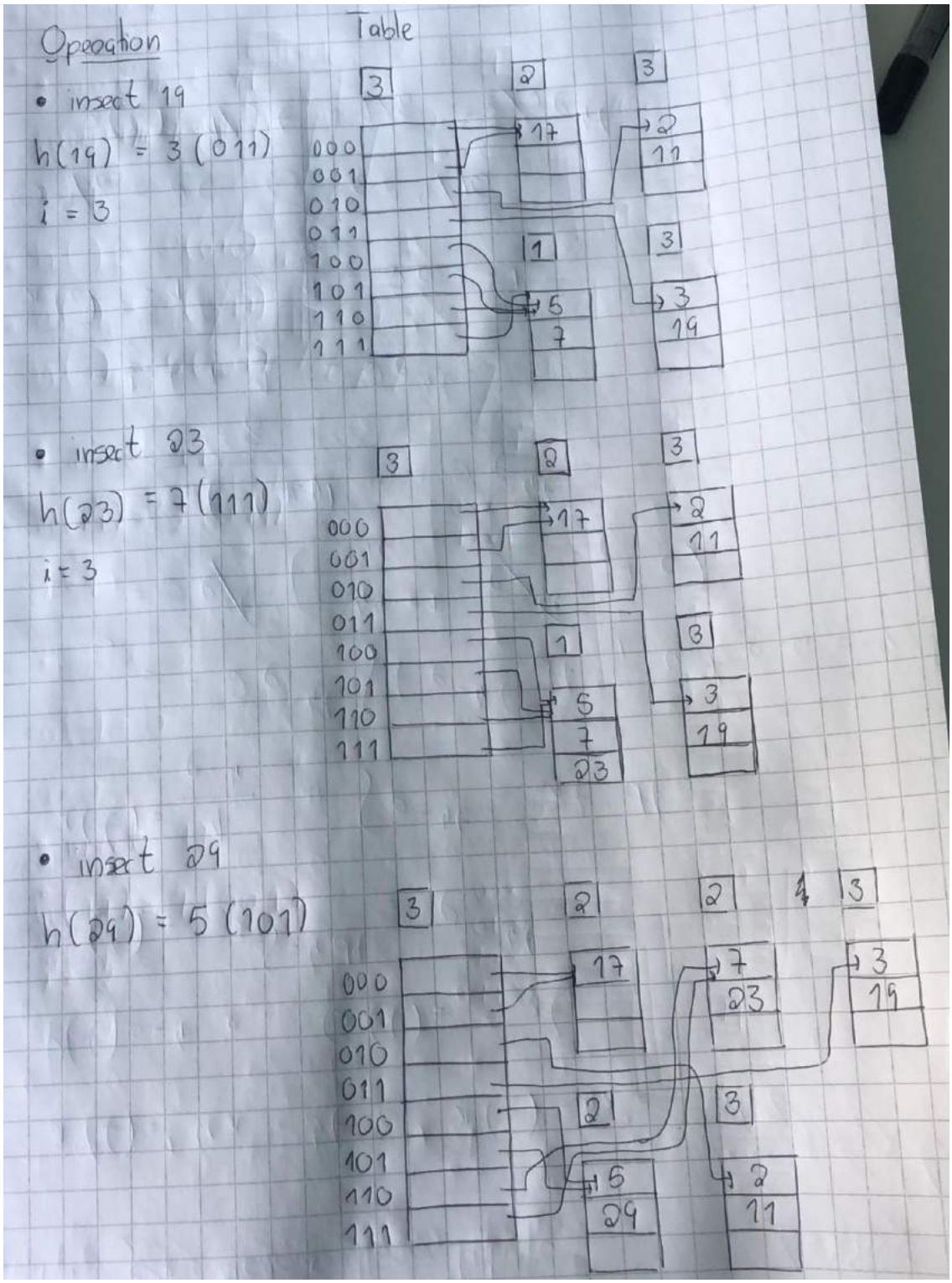


Figure 5.3: Insert 19, 23 and 29

Insert 31

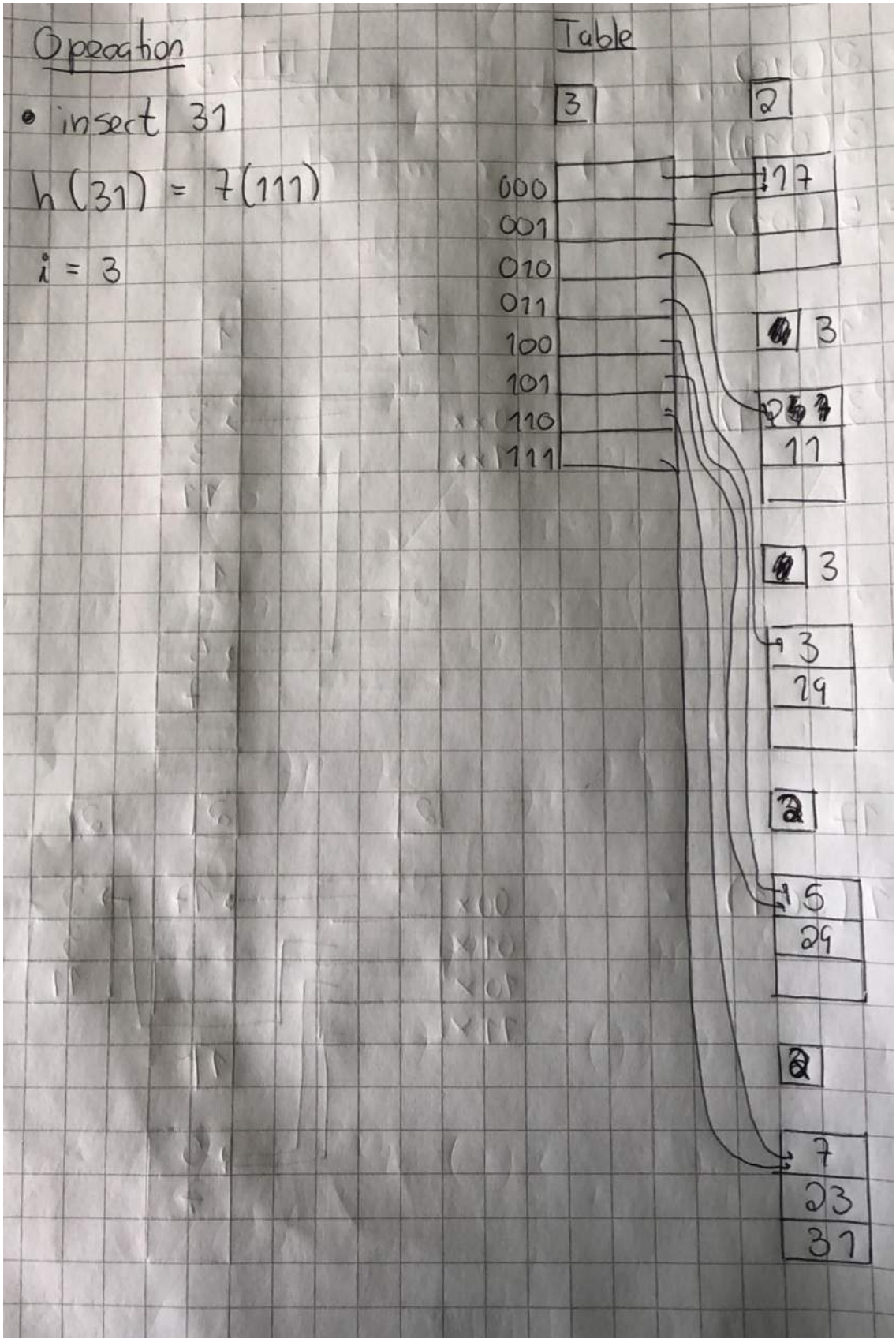


Figure 5.4: Insert 31

Review

This exercise is almost completely correct, the only mistake take by the team was not realising that when duplicating the bucket address table on the insertion of the value 19, instead of putting the value 11 on the index 011, by a lack of attention we putted on the index 010.

5.1.2 2)

What will happen if we have a file with an extendable hash index and insert a very large number of records all with the same search key? As an example, we could have an index on the family name in a phone number catalog, and insert a very large number of records with the key "Andersson". How will this affect the performance of the index?

Performance of the index does not degrade with the growth of file because the structure grows dynamically on demand

Review

This question was not consensual between the team, unfortunately the wrong answer prevailed, here is the correct answer:

All records with the same hash key will go to the same bucket, and after the bucket is full we have to create overflow buckets. The performance of the index will suffer, since for these records we have to use linear search through the overflow buckets.

5.1.3 3)

Assume we have a hash function defined as $h(x) = x^2 \ mod \ B$ where B is the number of buckets.

a) What is wrong with this hash function if B = 10?

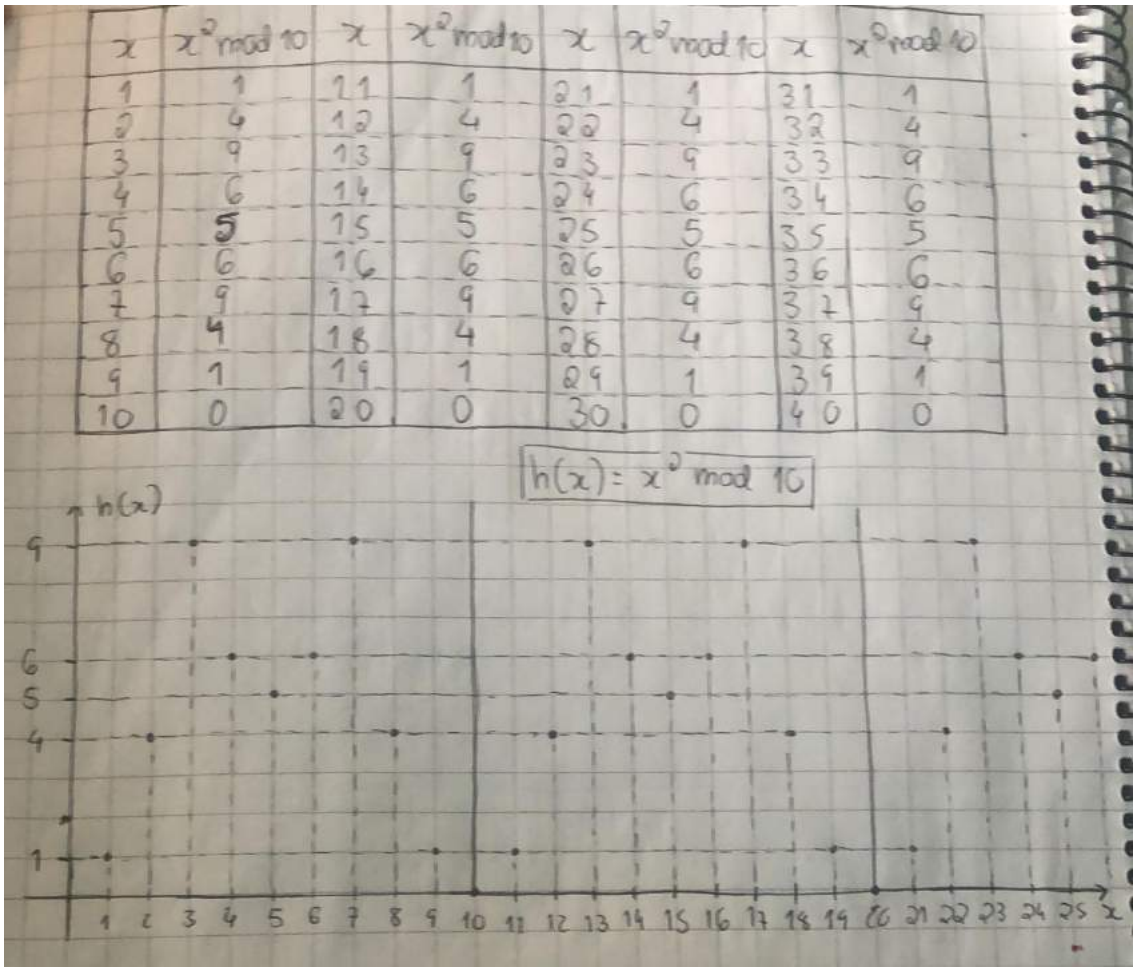


Figure 5.5

The problem with with this hash function if B = 10 is that it is a periodical function (of period 10), like we can see in Figure 5. Which this implies for the purpose of an hash table is that no matter what is the value x the result h(x) will always be between a set of results. In this particular case h(x) varies between 0, 1, 4, 5, 6 and 9, this results in the utilization of only this buckets and over time in a lot of overflow in these. For this reasons, this is a sub-optimal hash function.

Review

This exercise was correct.

b) Is the hash function we get with B = 16 better or worse?

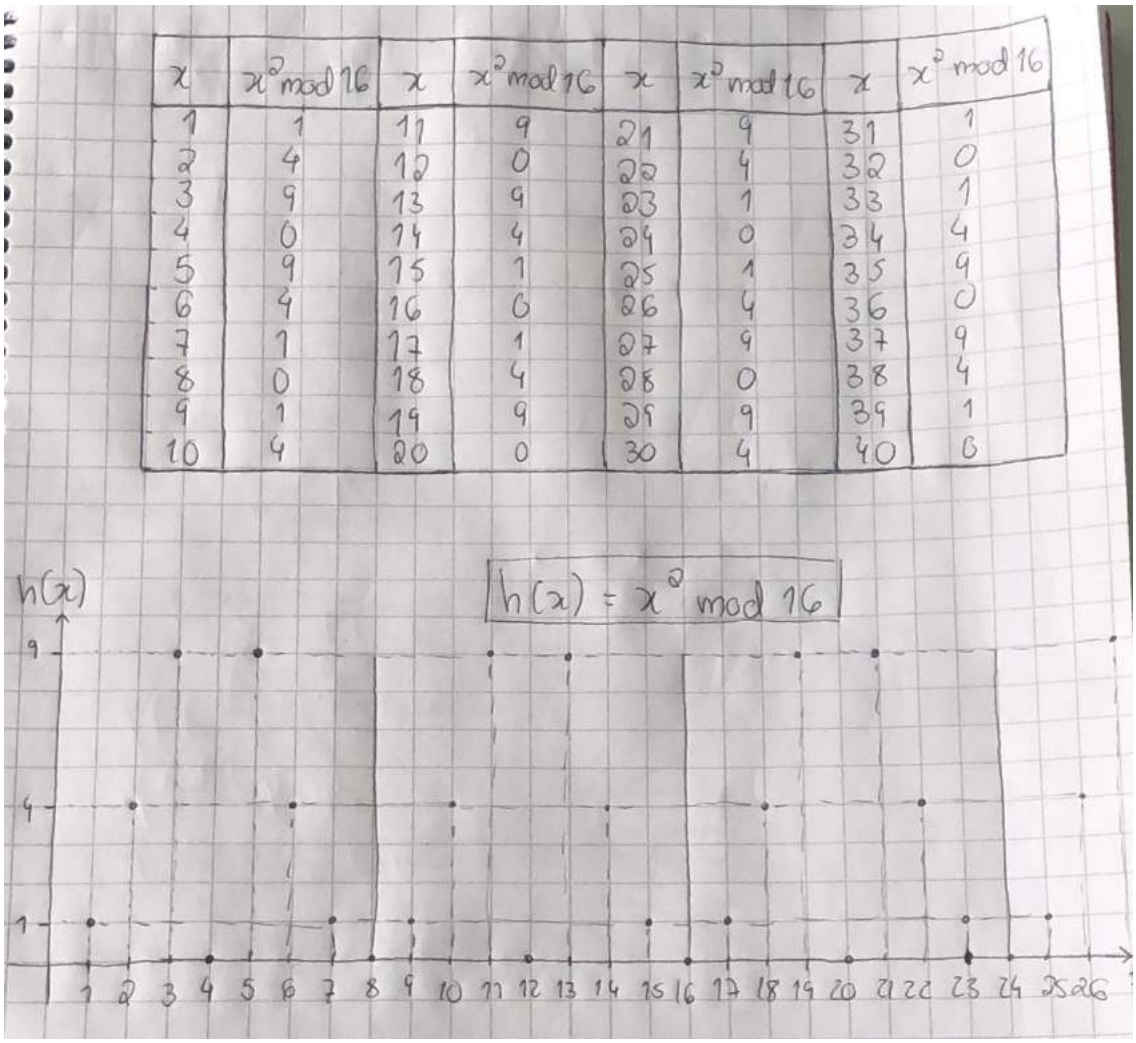


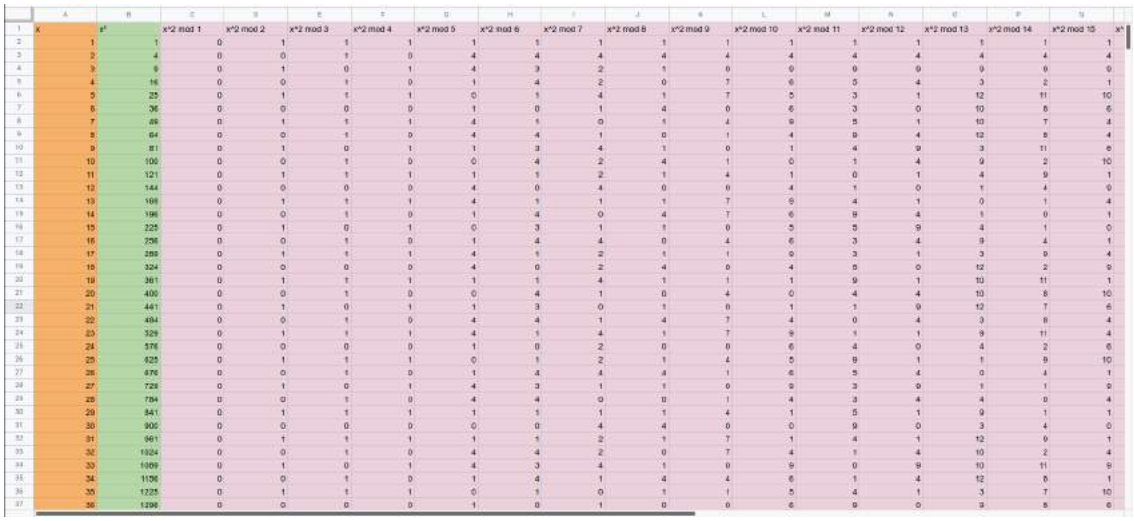
Figure 5.6

As we can see in Figure 6, the hash function we get with B = 16 is worse because is also a periodic function but with a smaller period - eight - and the results h(x) vary between a smaller set of values as a consequence: 0, 1, 4 and 9. The problems exposed above are even worse in this case.

Review

This exercise was correct.

c) Are there any values of B that give a useful hash function?



x	x^2	x^2 mod 1	x^2 mod 2	x^2 mod 3	x^2 mod 4	x^2 mod 5	x^2 mod 6	x^2 mod 7	x^2 mod 8	x^2 mod 9	x^2 mod 10	x^2 mod 11	x^2 mod 12	x^2 mod 13	x^2 mod 14	x^2 mod 15	x^1
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	0	0	1	0	4	4	4	4	4	4	4	4	4	4	4	4
3	9	0	1	0	1	4	3	2	1	0	9	9	9	9	9	9	9
4	16	0	0	1	0	1	4	2	0	7	6	5	4	3	2	1	
5	25	0	1	1	1	0	1	4	1	7	5	3	1	12	11	10	
6	36	0	0	0	0	1	0	1	4	0	6	3	0	10	8	6	
7	49	0	1	1	1	4	1	0	1	4	9	5	1	10	7	4	
8	64	0	0	1	0	4	4	1	0	1	4	9	4	12	8	4	
9	81	0	1	0	1	1	3	4	1	0	1	4	9	3	11	6	
10	100	0	0	1	0	0	4	2	4	1	0	1	4	9	2	10	
11	121	0	1	1	1	1	1	2	1	4	1	0	1	4	9	1	
12	144	0	0	0	0	4	0	4	0	0	4	1	0	1	4	9	
13	169	0	1	1	1	4	1	1	1	7	9	4	1	0	1	4	
14	196	0	0	1	0	1	4	0	4	7	6	9	4	1	0	1	
15	225	0	1	0	1	0	3	1	1	0	5	5	9	4	1	0	
16	256	0	0	1	0	1	4	4	0	4	6	3	4	9	4	1	
17	289	0	1	1	1	4	1	2	1	1	9	3	1	3	9	4	
18	324	0	0	0	0	4	0	2	4	0	4	5	0	12	2	9	
19	361	0	1	1	1	1	1	4	1	1	1	9	1	10	11	1	
20	400	0	0	1	0	0	4	1	0	4	0	4	4	10	8	10	
21	441	0	1	0	1	1	3	0	1	0	1	1	9	12	7	6	
22	484	0	0	1	0	4	4	1	4	7	4	0	4	3	8	4	
23	529	0	1	1	1	4	1	4	1	7	9	1	1	9	11	4	
24	576	0	0	0	0	1	0	2	0	0	6	4	0	4	2	6	
25	625	0	1	1	1	0	1	2	1	4	5	9	1	1	9	10	
26	676	0	0	1	0	1	4	4	4	1	6	5	4	0	4	1	
27	729	0	1	0	1	4	3	1	1	0	9	3	9	1	1	9	
28	784	0	0	1	0	4	4	0	0	1	4	3	4	4	0	4	
29	841	0	1	1	1	1	1	1	1	4	1	5	1	9	1	1	
30	900	0	0	0	0	0	0	4	4	0	0	9	0	3	4	0	
31	961	0	1	1	1	1	1	2	1	7	1	4	1	12	9	1	
32	1024	0	0	1	0	4	4	2	0	7	4	1	4	10	2	4	
33	1089	0	1	0	1	4	3	4	1	0	9	0	9	10	11	9	
34	1156	0	0	1	0	1	4	1	4	4	6	1	4	12	8	1	
35	1225	0	1	1	1	0	1	0	1	1	5	4	1	3	7	10	
36	1296	0	0	0	0	1	0	1	0	0	6	9	0	9	5	6	

Figure 5.7

For B = 1 (meaning 1 bucket), the hash function looks like this:

$h(x) = x^2 \bmod 1$

In this scenario, the hash function will have only on output: zero, here we have a hash function that output to all of the buckets available and at a equal rate so we can consider in this case a useful hash function.

For B = 2 (meaning 2 buckets), the hash function looks like this:

$h(x) = x^2 \bmod 2$

In this scenario, the hash function will have binary output: zero if x^2 is pair and one if it is impair, here we have a hash function that output to all of the buckets available and at a equal rate so we can consider in this case a useful hash function.

In any other scenario, it is not a useful hash function.

Review

This exercise was correct.

5.1.4 4)

Go through the tables in the university database and have a look at how the indices are created in the schema definitions (in the SQL file university.sql). Verify that the expected indices exist by using the command show index from table_name for each table.

a) Which tables have both a primary and a secondary index?

The tables that have both a primary and secondary index are:

- advisor
- course
- instructor
- prereq
- section
- student
- takes
- teaches

Review

This exercise was correct.

b) Which tables have a primary index on a composite key (i.e. a primary key that consists of more than one attribute)?

The tables that have a primary index on a composite key are:

- classroom
- prereq
- section
- takes
- teaches
- time_slot

Review

This exercise was correct.

c) Which tables have a secondary index on a composite key?

The tables that have a secondary index on a composite key are:

- section
- takes
- teaches

Review

This exercise was correct.

5.1.5 5)

a) Construct bitmap indices on the attributes dept_name and salary. The salary values should be divided into 4 ranges: 0–49999, 50000–59999, 60000–69999 and 70000 and above

Constructing the bitmap for the different departments, first selected them, and after we attributed the corresponded bit, 1 if the record was present in that position and 0 if it was not present.

Bitmaps for dept_name		
#	Dept_name	Bitmap
1	Comp.Sci.	100000100010
2	Finance	010000001000
3	Music	001000000000
4	Physics	000101000000
5	History	000010010000
6	Biology	000000000100
7	Elec. Eng.	000000000001

For the salary level bitmap, we first created a table were in each row we could distinguish the salary and the level corresponded.
After this first table been created, the final table was effortlessly created.

Record number	Salary (level)
0	65000 (L3)
1	90000 (L4)
2	40000 (L1)
3	95000 (L4)
4	60000 (L3)
5	87000 (L4)
6	75000 (L4)
7	62000 (L3)
8	80000 (L4)
9	72000 (L4)
10	92000 (L4)
11	80000 (L4)

Bitmaps for salary_level	
Level	Bitmap
L1	001000000000
L2	000000000000
L3	100010010000
L4	010101101111

Review

This exercise was correct.

b) Consider a SQL query that requests all instructors in the Finance department:

```
SELECT ID, name, salary
FROM instructor
WHERE dept_name = 'Finance' AND salary >= 80000;
```

Describe how this query could be answered using the above created bitmap indexes. Describe the intermediate steps and bit operations in the execution of the query.

To answer this query, using the bitmap indexes created, first we have to compute:
$dept\_name = 'Finance' \; and \; salary >= 80000$.

We can easily notice that the bitmap from the department 'Finance' is 010000001000.
One the other hand, it's not that simple to obtain the other part of the operation, $salary >= 80000$. Initially, we can identify that this operation is included on the level 4 of the salary_level bitmap, but, and since this level is determined by all the salaries higher than 70000, this level has also the salary between 7000 and 8000, which we need to be a little careful when getting the determined bitmap. So, with this knowledge, we can get the bitmap from the salary_level 4, 010101101111, and verify each position where the bit is 1. If the salary is lower than 80000 we change the bit to 0, otherwise the bit stays as 1. With this, we obtain the bitmap 010101001011.Then, we can determine the final intersection operation with this 2 bitmaps:
$010000001000\ AND\ 010101001011\ =\ 010000001000$

Finally, with this bitmap we can obtain the positions of the records needed to solve this query, which are positions number 1 and 8 of the table.

Review

This exercise was correct.

Chapter 6

Query processing

6.1 Exercises

6.1.1 1)

For each of the algorithms A1 – A8 that can be used to execute select-operations, design a SQL query on the university database that could be executed with this algorithm.

A1

```sql
SELECT *
FROM student
WHERE dept_name = 'Finance'
```

A2

```sql
FROM student
USE INDEX(PRIMARY)
WHERE ID = '26080';
```

A3

```sql
FROM student
USE INDEX(PRIMARY)
WHERE name = 'Skeen';
```

A4

```sql
FROM student
USE INDEX(dept_name)
WHERE name = 'Skeen';
```

A5

```sql
FROM student
USE INDEX(PRIMARY)
WHERE tot_cred > 2
```

A6

```
FROM student
USE INDEX(dept_name)
WHERE tot_cred > 2
```

A7

```
FROM student
USE INDEX(dept_name)
WHERE dept_name = 'Civil Eng.' AND tot_cred > 2
```

A8

```
FROM student
USE INDEX(PRIMARY,dept_name)
WHERE dept_name = 'Civil Eng.' AND tot_cred > 2
```

6.1.2 2)

Assume that we have a relation containing 22,000 records where all records have a fixed size of 160 bytes, and that the disk block size is 4 KB (4096 bytes). There is a B+-tree index on the primary key of the relation, and we can assume that the index is small enough to be kept in main memory at all times, so no disk accesses are needed for index lookup. How many disk seeks and block transfers can we estimate to be needed for:

a) A linear scan of all records in the file

In a linear scan, we will have to inspect each block of the file and test all records to see whether they satisfy the selection condition or not. So, it will be needed an initial seek to find the first block of the file, and then a read for all consecutive blocks. Therefore, let's calculate the number of existing blocks:

$22000 * 160 = 3520000\ bytes\ (Size\ of\ all\ records)$
$3520000\ /\ 4096 \approx 860\ blocks$

We can conclude that there are 860 blocks, which means that the linear scan will make 859 block transfers.

b) Accessing a record with a given value on the primary key

When accessing a record with a given value on the primary key, in a B+-tree index, the algorithm will make a seek and transfer operation, to fetch the block where the record is located. So, the number of seeks and transfers will be the same as the height of the B+-tree(h) plus one.

$h + 1\ where\ h = \log_2 860 \approx 10$
$10 + 1 = 11\ seeks\ and\ transfers$

Accordingly, the selection will make 11 disk seeks and 11 block transfers.

c) Accessing the records with a given value on a non-key attribute

To access the records with a given value on a non-key attribute, the algorithm will have to pass through the B+-tree, just as if the given value was on the primary key, but this time the search on the disk block will not be so straight forward. Therefore, the number of seeks will be the same as the height(h) of the tree plus one, but the number of block transfers will be the height of the tree plus the number of records(b) in a block disk.

$h + b \; where \; b = 4096 \; / \; 160 \approx 26$
$10 + 26 = 36 \; transfers$

So, there will be made 11 disk seeks and 36 block transfers.

d) Accessing all records with a value greater than a given value on the primary key

The difference between this selection and the previous one is that in this scan, the algorithm, after making his way through the B+-tree, it will not pass by the whole block disk, the selection will start from the point where the given value equals to the primary key. Because of this, the value can variate, depending on the given and primary key value, and so we will consider half of the size of the disk block as the number of the block transfers.

Finally, the selection will make 11 disk seeks and $10 + 13 = 23$ block transfers.

6.1.3 3)

The relation r(A, B, C) contains 20000 tuples and the relation s(C, D, E, F) contains 45000 tuples. The tuples in r are of size 128 Bytes and the tuples in s are of size 100 bytes. The block size is 4 KB (4096 Bytes). The primary keys of the relations are underlined, and the relations have a B+-tree primary index on these keys. We assume that index blocks of the B+-tree are kept in memory and do not need to be read from disk.
Explain how the following relational algebra expressions can be executed, and which of the algorithms A1 – A10 can be used. Give estimates for the number of disk seeks and transfers the queries need.

a) $\sigma_{C=23194} \; (r)$

Selecting c from the outer relation(r) where c = 23194. But r contains 20000 tuples as against 23194 tuples.
For this, we use algorithm A3. We retrieve all records of the relation using the index, and check each record whether the condition on a non-key attribute golds or not.
Since we have an equality on a non-key, there may be many records satisfying the criteria, but just as we assumed that the blocks of the index tree is held in memory, the cost become:
Number of seeks = 1
Number of transfers = 128

b) $\sigma_{C=23194} \; (s)$

Selecting c from the inner relation(s) where c = 23194. And s contains 4500 tuples.
For this, we use algorithm A2. The index is used to retrieve a single record that satisfies the corresponding equality condition. Sing the index is kept in memory the cost is only one seek and one block transfer.

c) $\sigma_{E='Turku'}\ (s)$ **assuming that there is a secondary B+-tree index on E in s**

Selecting E from inner relation(s) where E = 'Turku'. For this query, we use algorithm A1.
The selection is equality on a candidate key attribute, once the record is found, the file scan is stopped immediately. Only one record can satisfy the selection criteria. In this kind of selection operation, linear search can be applied regardless of selection condition. This type of search algorithm (called the index scans) must be on a search-key of an index.
Number of seeks = 1
Number of transfers = 100 / 2 = 50

d) $\sigma_{B\geq 1992}\ (r)$

Selecting B from outer relation(r) where B is greater than or equal to 1992.
For this query, we use A5 algorithm. We use the index to find the first tuple where B = 1992 and scan the relation sequentially from there. For $\sigma_{B\geq 1992}\ (r)$, we just need to scan the beginning of the tupple where B = 1992 to the last tuple where B ¿ 1992. If the system has information about where the relation is stored in disk no accesses to the index is needed, else, index is used to find the first block of the relation. For this selection operation:
Number of seeks = 1
Number of transfers = 128

Chapter 7

Query Optimization

7.1 Execution plans

Execution plans allow us to understand the coordination of the operations on the overall effect on the query performance, is an essential tool for query optimization.

7.2 Transformation of relational expressions

This was the chapter that I particularly enjoyed the most. Using equivalence rules we are able to transform relational expressions, using this rules we can start from expressions that correct in terms of giving the correct result and the later do the program calculation to transform them into efficient ones. It goes with one of my favorite areas in Software Engineering - Formal Methods.

7.3 Exercises

7.3.1 1)

Formulate SQL queries that answer the following questions and generate the execution plan for each of them. Study the execution plans and write verbal explanations of how the queries is executed. Include pictures of the execution plans, and a short explanation of how each query is executed.

a) How many credits have the student with name 'Conti' passed? The result should contain student ID, name of the student and the total number of credits.

```
SELECT
    student.ID, student.name, SUM(course.credits) AS total_cred
FROM
    student
        JOIN
    takes ON student.ID = takes.ID
        JOIN
    course ON takes.course_id = course.course_id
WHERE
    student.name = 'Conti'
GROUP BY (student.ID);
```

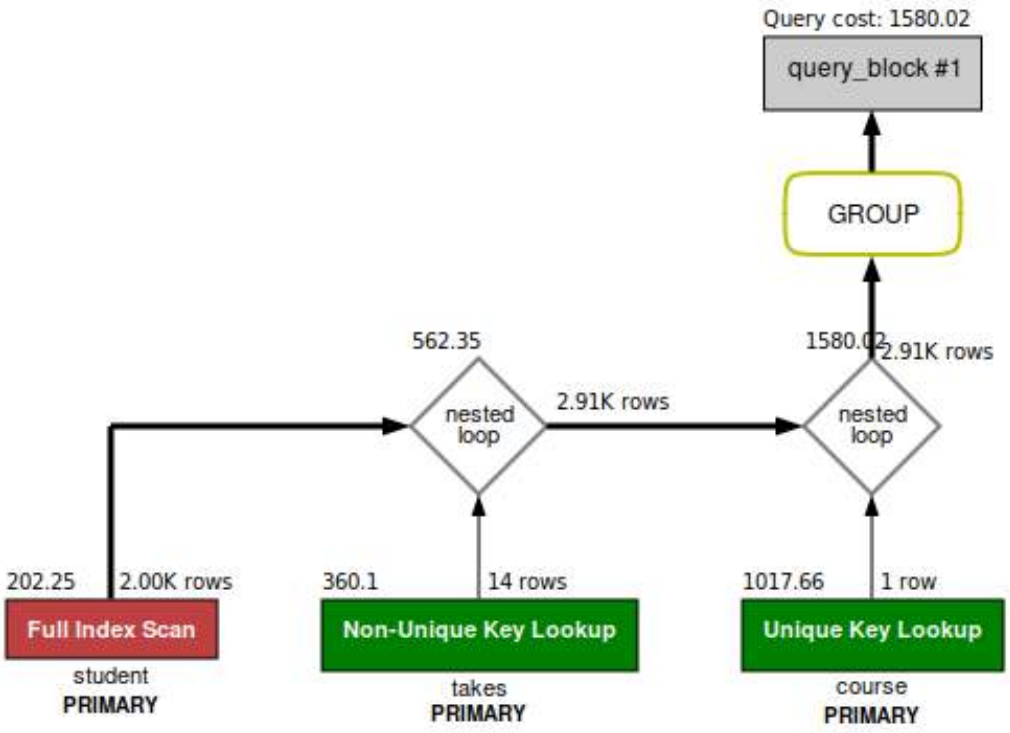


Figure 7.1: Execution Plan

Review

This exercise is correct.

b) Which courses have the student named 'Conti' taken? The result should include student ID,student name, course title, year the course was taken, grade of the course and the number of credit points of each course.

```
SELECT
    student.ID,
    student.name,
    course.title,
    takes.year,
    course.credits
FROM
    student
        JOIN
    takes ON student.ID = takes.ID
        JOIN
    course ON takes.course_id = course.course_id
WHERE
    student.name = 'Conti';
```

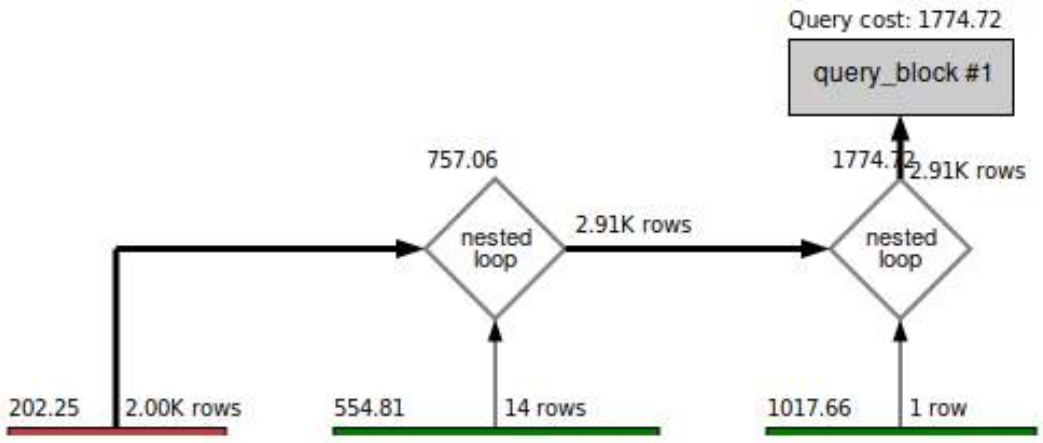


Figure 7.2: Execution Plan

Review

This exercise is correct.

c) Which courses have the student with ID '82301' taken? The result should include the student ID, student name, course title, year the course was taken, grade of the course and the number of credit points of each course. The result will be identical to the previous query since Conti has student ID '82301'. However, the execution plan will be different. Explain how it differs from the previous query.

```
SELECT
    student.ID,
    student.name,
    course.title,
    takes.year,
    course.credits
FROM
    student
        JOIN
    takes ON student.ID = takes.ID
        JOIN
    course ON takes.course_id = course.course_id
WHERE
    student.ID = '82301';
```

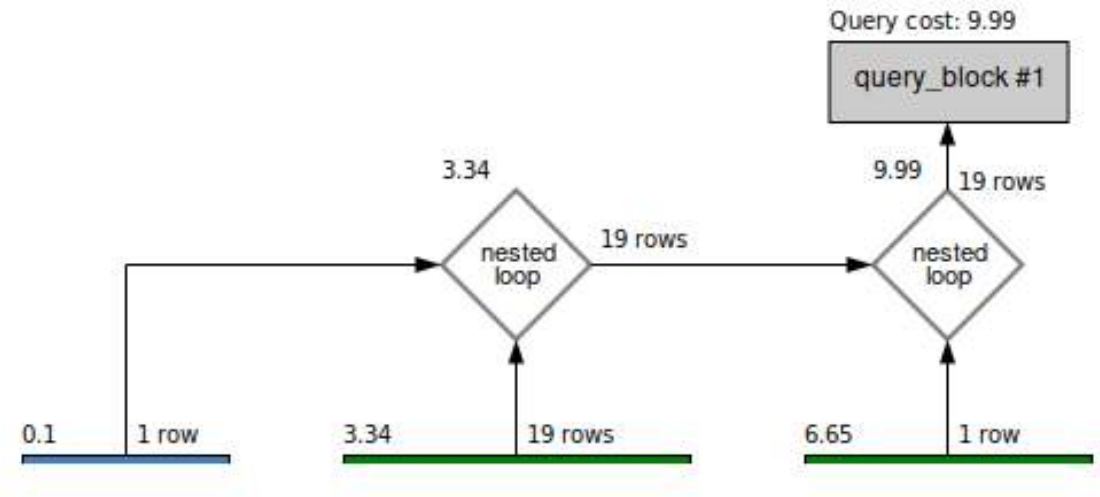


Figure 7.3: Execution Plan

Review

This exercise is correct.

d) Create a list of the students in the Biology department and the number of credit points each of these have passed. The result should include the student ID, the student name and the sum of credits points for the courses the student has passed.The result should be in descending order of the number of credit points the student has (that is, the student with most credit points first).

```
SELECT
    student.ID, COUNT(course.credits) AS total_cred
FROM
    student
        JOIN
    takes ON student.ID = takes.ID
        JOIN
    course ON takes.course_id = course.course_id
GROUP BY student.ID
ORDER BY total_cred DESC;
```

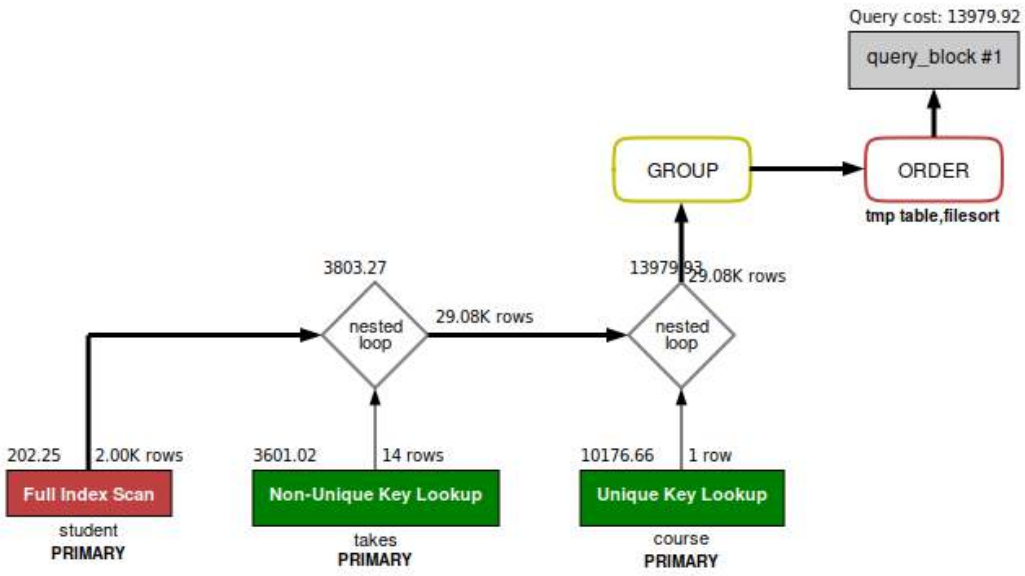


Figure 7.4: Execution Plan

Review

This exercise was correct.

e) For each instructor, for how many students does the instructor act as an advisor. The result should contain the ID of the instructor, the name and department of the instructor and the number of students this instructor is advisor for. The result should be sorted by the number of students.

```sql
SELECT
    instructor.ID,
    instructor.dept_name,
    COUNT(s_ID) AS num_students
FROM
    instructor
        JOIN
    advisor ON instructor.ID = i_ID
GROUP BY instructor.ID
ORDER BY num_students DESC;
```

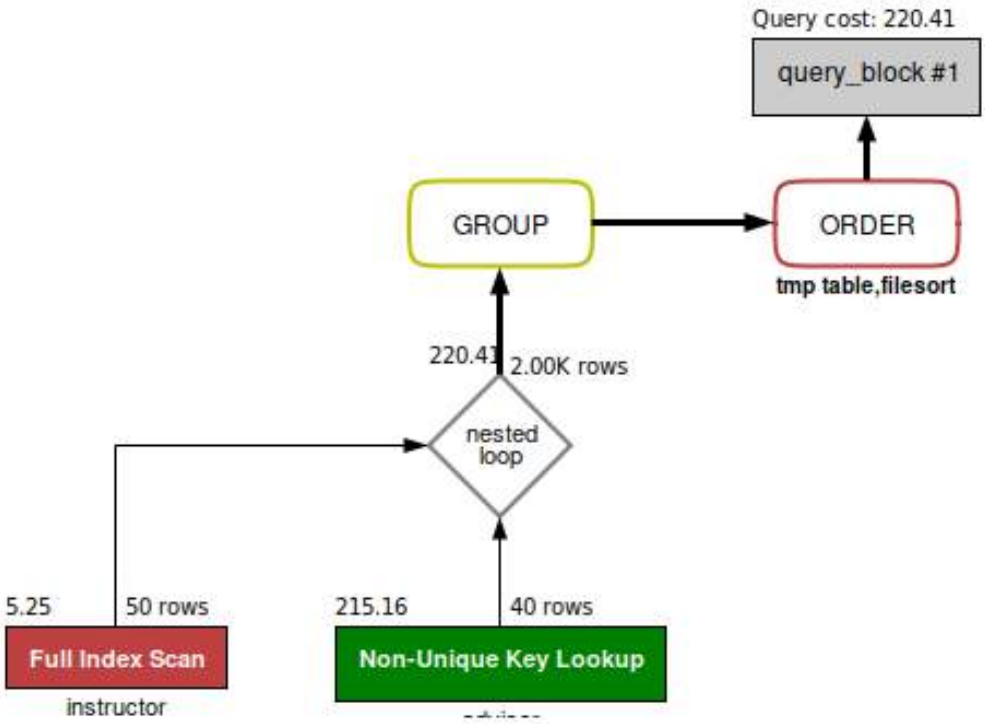


Figure 7.5: Execution Plan

Review

This exercise is correct.

f) Which department pays the largest total salary sum to its instructors?

```
SELECT
    instructor.dept_name, SUM(instructor.salary) AS total_salary
FROM
    instructor
GROUP BY instructor.dept_name
ORDER BY total_salary DESC
LIMIT 1;
```

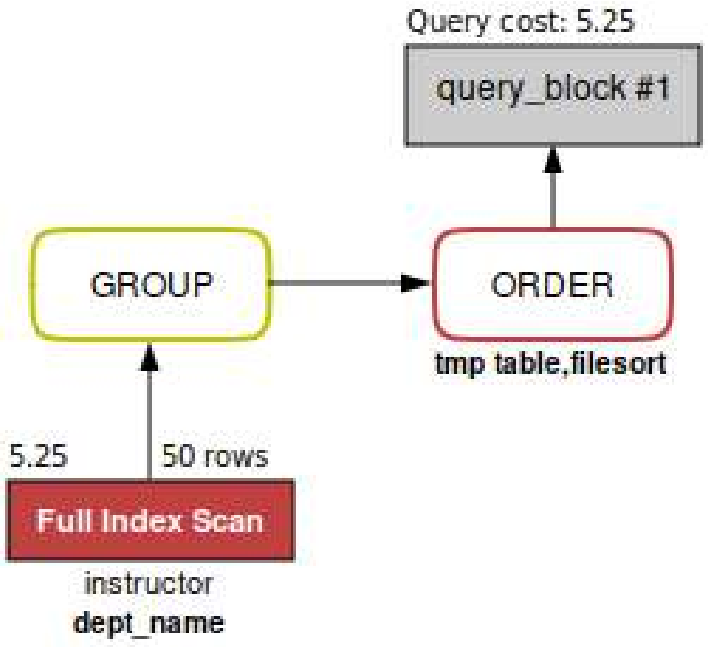


Figure 7.6: Execution Plan

Review

This exercise is correct.

7.3.2 2)

Choose two queries in question 1 above and change the order in which tables are joined by using STRAIGHT_JOIN so that the order becomes different than in the original formulation. Compare the execution plans to the original (optimized) plans and check how this join order affects the total cost of the query

d)

```
SELECT
    student.ID, COUNT(course.credits) AS total_cred
FROM
    student
        STRAIGHT_JOIN
    takes ON student.ID = takes.ID
        STRAIGHT_JOIN
    course ON takes.course_id = course.course_id
GROUP BY student.ID
ORDER BY total_cred DESC;
```

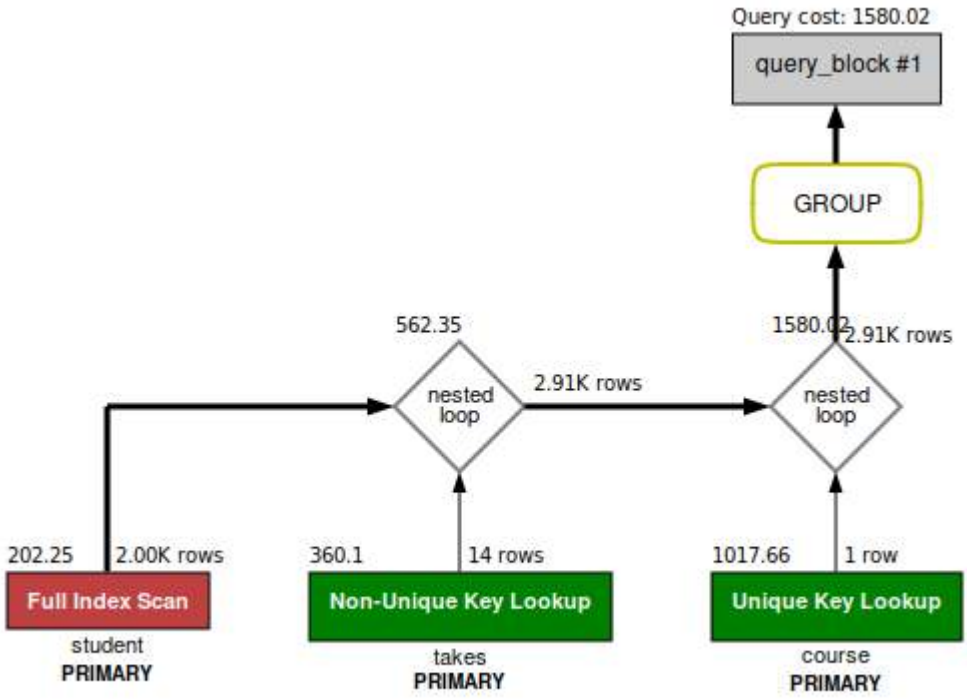


Figure 7.7: Execution Plan

Review

This exercise is correct.

e)

```sql
SELECT
    instructor.ID,
    instructor.dept_name,
    COUNT(s_ID) AS num_students
FROM
    instructor
        STRAIGHT_JOIN
    advisor ON instructor.ID = i_ID
GROUP BY instructor.ID
ORDER BY num_students DESC;
```

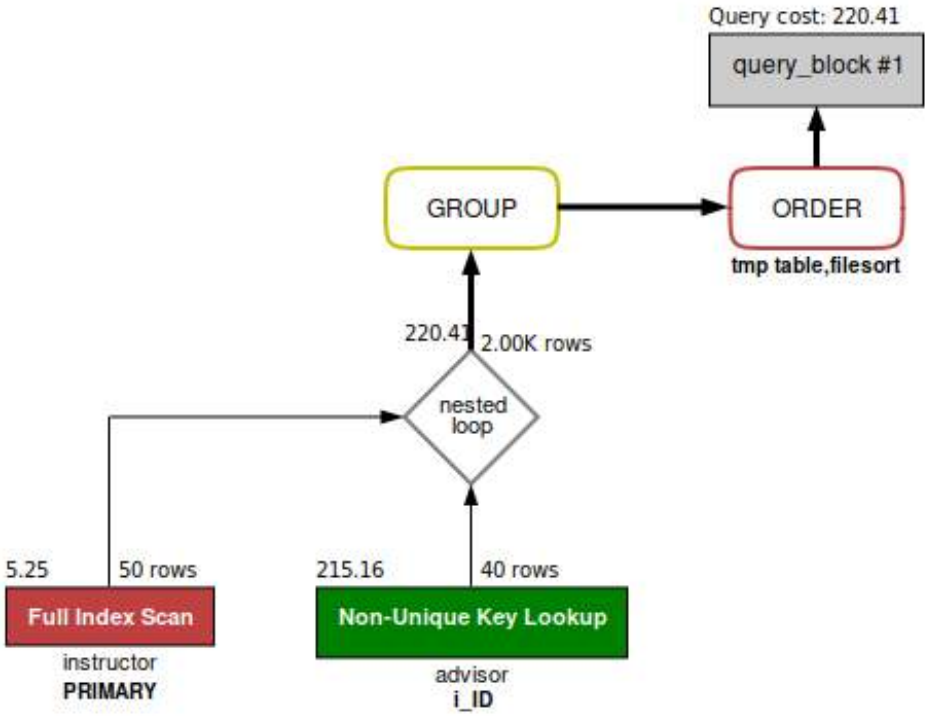


Figure 7.8: Execution Plan

Review

This exercise is correct.

7.3.3 3)

Assume we have a relation r = (A, B, C) and a relation s = (C, D, E) where the primary keys are underlined. Transform the following relational algebra expressions into equivalent expressions using the transformation rules for relational algebra expressions.

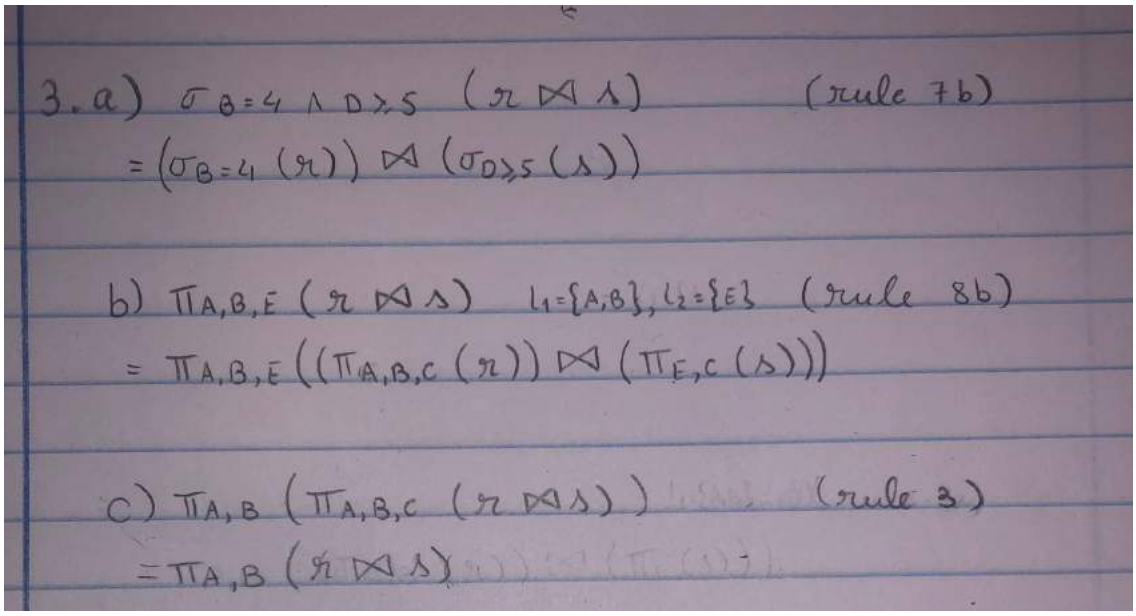


Figure 7.9: Exercise 3

Review

In the exercise c) we could have use rule 8b like in exercise b) to further transform.

7.3.4 4)

Consider the relations r = (A, B, C, D) and s = (D, E, F, G) where the primary keys are underlined. Transform the following relational algebra expressions into equivalent expressions, which can be executed more efficiently.

a)

The equivalent expression is executed more efficiently because it reduces the number of projections.

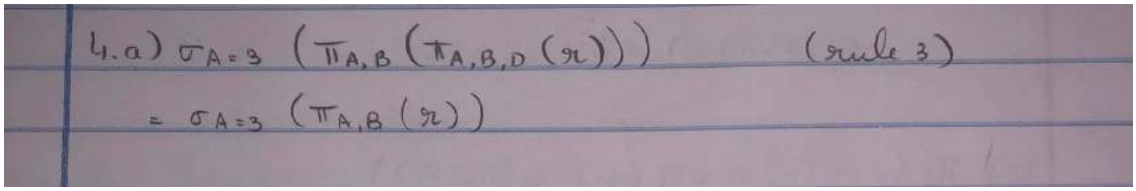


Figure 7.10: Exercise 4.a

Review

This exercise is correct.

b)

Since the join operator is a costly operation, minimizing this operation is a good way of increasing the efficiency of the query. The equivalent expression makes this operation more efficient because, before operating the join, it will first decrease the rows of the tables.

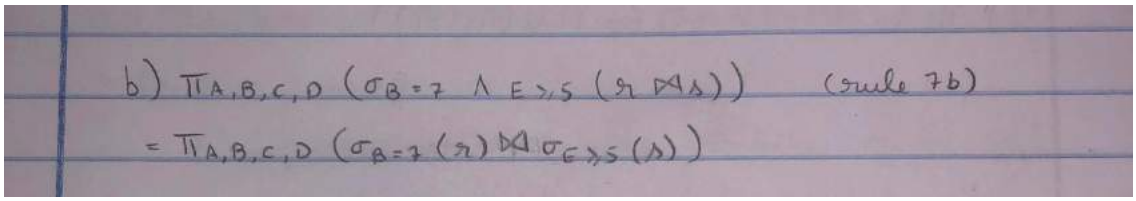


Figure 7.11: Exercise 4.b

Review

Here we should have used the rules 1, 7a and 8a by this order.

c)

For the same reason as the previous exercise, the equivalent expression is more efficient.

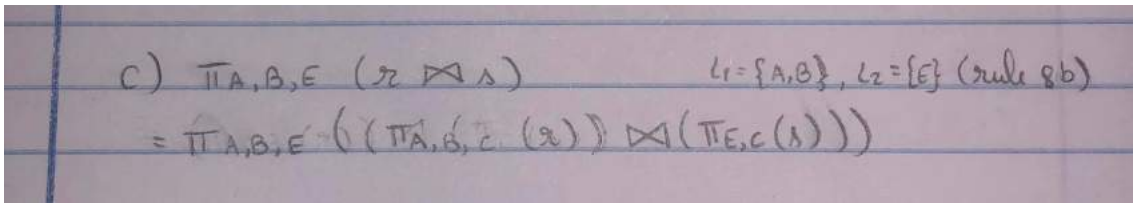


Figure 7.12: Exercise 4.c

Review

This exercise is correct.

d)

From the group's point of view, the equivalent expression obtained is more efficient. Once the attribute D is a primary key, the search for an attribute D will be very cost-efficient. So, from the first expression, we can observe that the select operation will pass through the whole $r \cup s$ table, while the equivalent expression will first select a primary key from the $r \cup s$ table, making this table smaller for the next selection.

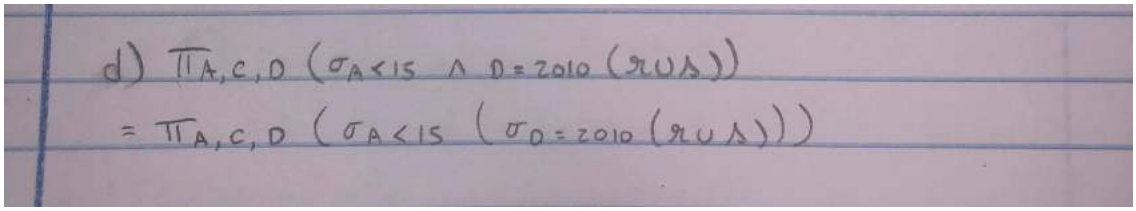


Figure 7.13: Exercise 4.d

Review

Here we should have used rule 12.

7.4 Cost-based optimization

What differentiates a programmer from a engineer is the ability of solving problems in a optimal way. Using the equivalence rules we learned prior, we can achieve optimal performance, in this chapter, we learned how to calculate the cost of JOINS.

7.5 Other optimizations

In this chapter, we also learned how to optimizations on other operations:

- **Nested Subqueries** Using Decorrelation;
- **Top-K Queries**;
- **Updates**;
- **Multiqueries**.

7.6 Cost estimation

We talked about how to estimate the size of SELECTS, JOINS, OUTER JOINS and Other Operations. Additionally, we talked about the estimation of the number of DISTINCT VALUES.

7.7 Exercises

7.7.1 1)

Assume that we have 3 tables r = (<u>A</u>, B, C), s = (<u>C</u>, D, E) and t = (<u>E</u>, F) where the primary keys of the relations are underlined. The table r has 1000 tuples, s has 1500 tuples and t has 750 tuples.

Estimate the size of the natural join rst and explain how you reason to calculate the estimate. Describe an efficient execution strategy for computing the join.

First, we join r and s and we joined them in C. C is a key in s and a foreign key in r, so each tuple of r joins exactly with one tuple in s:

$u = r \bowtie s$
$T(u) = T(r) = \ 1000 \ tuples$

After we join u with the table t on E, E is a key in t and a foreign key in u, so each tuple of u joins exactly with one tuple in t:

$T(u \bowtie t) = T(t) = \ 750 \ tuples$

Review

This exercise was correct.

7.7.2 2)

Assume we have the same tables r and s as in the previous question. We also know that the attribute B in the table r has a minimum value of 100 and a maximum value of 300. Transform the following expression into a more efficient form and estimate the size of its result

First, we transform the expression into a more efficient form:

$$\sigma_{B \leq 250}\ (r\ \bowtie\ s)$$
$$= (\sigma_{B \leq 250}\ r) \bowtie\ s$$

Then, we estimate the size of its result:

$$T(\sigma_{B \leq 250}\ (r\ \bowtie\ s)) = T(r\ \bowtie\ s)\ 250 - 100300 - 100 \equiv$$
$$\equiv T(\sigma_{B \leq 250}\ (r\ \bowtie\ s)) = T(r\ \bowtie\ s)\ 150200 \equiv$$
$$\equiv T(\sigma_{B \leq 250}\ (r\ \bowtie\ s)) = 0.75 \times 1000 \equiv$$
$$\equiv T(\sigma_{B \leq 250}\ (r\ \bowtie\ s)) = 750$$

Review

This exercise was correct.

7.7.3 3)

Assume that we have 3 tables r = (<u>A</u>, B, D), s = (<u>C</u>, D, F) and t = (<u>E</u>, F) where the primary keys of the relations are underlined and the tables have the same number of tuples as in the previous questions. We also know the following about the number of distinct values of the attributes in the tables: V(B, r) = 200, V(D, s) = 300 and V(F, t) = 50.
Transform the following expression into a more efficient form and estimate the size of its result

$$\sigma_{A=1023}\ (r\ \bowtie\ s\ \bowtie\ t)$$

For a more efficient expression, we changed the expression to $\sigma_{A=1023}\ (r\ \bowtie\ (\ s \bowtie\ t))$. With this expression, the first join to compute is $s \bowtie\ t$ which is smaller then $r \bowtie\ s$.
When intersecting s with t ($s \cap t$) we can find 50 tuples, and when intersection r with the result of the join of s with t ($r \cap (s \cap t)$, we end up with 300 + 50 = 350.

Review

On this exercise we made the transformation correctly, but instead of putting the DISTINCT VALUES factor into the equation we just calculated normally the size of its result. The solution looks like this:

$$T(r) = nsV(D, s) = 1500300 = 5\ T(r) = ntV(F, t) = 75050 = 15$$

The selection on A is pushed into the join, giving the expression:

$$(\sigma_{A=1023}\ r)\ \bowtie\ s\ \bowtie\ t$$

The size of the selection on A=1023 in r is 1, since A is the primary key and there can only be one tuple satisfying the condition.

$$T((\sigma_{A=1023}\ r)\ \bowtie\ s\ \bowtie\ t) = 1 * 5 * 15 = 75$$

7.7.4 4)

Estimate the number of tuples that the following queries will produce. Please also explain how the estimates are calculated.

a) $\sigma_{dept="Physics"}\ (student)$

Number of tuples (nr) = 10000
V(dept,student) = 10
Estimated number of tuples = $nrV(dept, student)$

. $= 1000010$

. $= 1000$

Review

This exercise was correct.

b)$\sigma_{credits \geq 120}\ (student)$

Number of tuples (nr) = 10000
v = 120
min(credits,student) = 0
max(credits,student) = 200
Estimated number of tuples = $nr * v\ -\ min(credits, student)(max(credits, student)\ -\ min(credits, student))$

. $= 10000\ * 120 - 0200 - 0$

. $= 6000$

Review

Despite the fact that the solution indicates otherwise, this exercise is correct according to the material given:
If min(A,r) and max(A,r) are available in the metadata catalog we estimate c,the number of tuples satisfying the condition as:

- c = $0\ if\ v < min(A, r)$ (no tuples have a value smaller than the minimum)
- c = $nr\ if\ v \geq max(A, r)$ (all tuples have a value smaller than the maximum)
- c = $nr * v\ -\ min(credits, student)(max(credits, student)\ -\ min(credits, student))$ otherwise

Considering that the last predicate is the one that verifies, the correct formula was chosen.

c)$\sigma_{year='2009'\ \wedge\ grade \neq 'F'}\ (takes)$

- **year = '2009'**
  Once the attribute "year" is a primary key the estimated number of tuples is 1.
- **grade ≠ 'F'**
  Number of tuples (nr) = 50000
  V(grade,takes) = 9
  Estimated number of tuples = $nr\ - nrV(year, takes)$

  . $= 50000\ -\ 500009$

. $= 44444,445$

- **year = '2009' $\wedge$ grade $\neq$ 'F'**
  Estimated number of tuples = nr * $s1 * s2nr^2$

  . $= 50000 * 1 * 44444,4550000^2$

  . $= 0,889$

Review

Despite the fact that the solution indicates otherwise, this exercise is correct according to the material given:

If the equality condition is on a key attribute, the size estimate is 1 (since key values are unique and V(A,r) = nr), and according to the enunciate "year" is a primary key, so the estimated number of tuples is 1.

Chapter 8

Conclusion

Now that the course comes to an end, my understatement of Databases evolved a lot. Accurately solving problems is a job of an engineer, using the things I gained by the course I develop this semester, I greatly increase the sophistication of the models and queries code I develop by having an in dept knowledge of the algorithms that are being each of the operators and the combinations of them.

This insights already have an impact on my every day life and the way I provide Databases solutions.

Bibliography

[1] Database System Concepts by Abraham Silberschatz, Henry F. Korth and S. Sudarshan, sixth edition

[2] Connolly, T., Begg, C., Database Systems: A Practical Approach to Design, Implementation, and Management, Addison-Wesley, Global Edition, 26 Sep2014. ISBN-10: 1292061189, ISBN-13: 978-1292061184.