Self-correcting test bank for SQL basics

Gabriella Baksa-Haskó[1]

[1] Corvinus University of Budapest
gabriella.hasko@uni-corvinus.hu

Abstract. Learning SQL needs a lot of practice. Because of the richness of this language it is quite difficult to check and correct these practice exercises. Until now we gave the possibility to our students to upload their solutions into the Moodle and with large workforce, we gave them feedback. My new solution is a flipped method. Students do not upload their solutions in SQL but only answer a question in connection with the result. For example: how many rows are there in the result or what is the name in the first row. It is important to have a sample database with very diversified data so different logically wrong solutions should lead to different wrong answers. The practical quiz in the Moodle will give an immediate feedback including the possible reason for the wrong answer. We can build the database analyzing the typical wrong solutions uploaded by the students in the previous year. We have to deal with the logical errors only because the DBMS checks the syntactics during the work. This test bank will help the students with immediate feedback during their practice but will not be proper for examinations because the correct answers can be guessed in other ways as well.

Keywords: SQL, test bank, practicing

1. Introduction

SQL is a structured language for creating queries in a relational database management system [1]. In the learning process repetition is very important. On the lessons we can explain the commands and can try a few examples. For deeper learning, students have to practice at home a lot.

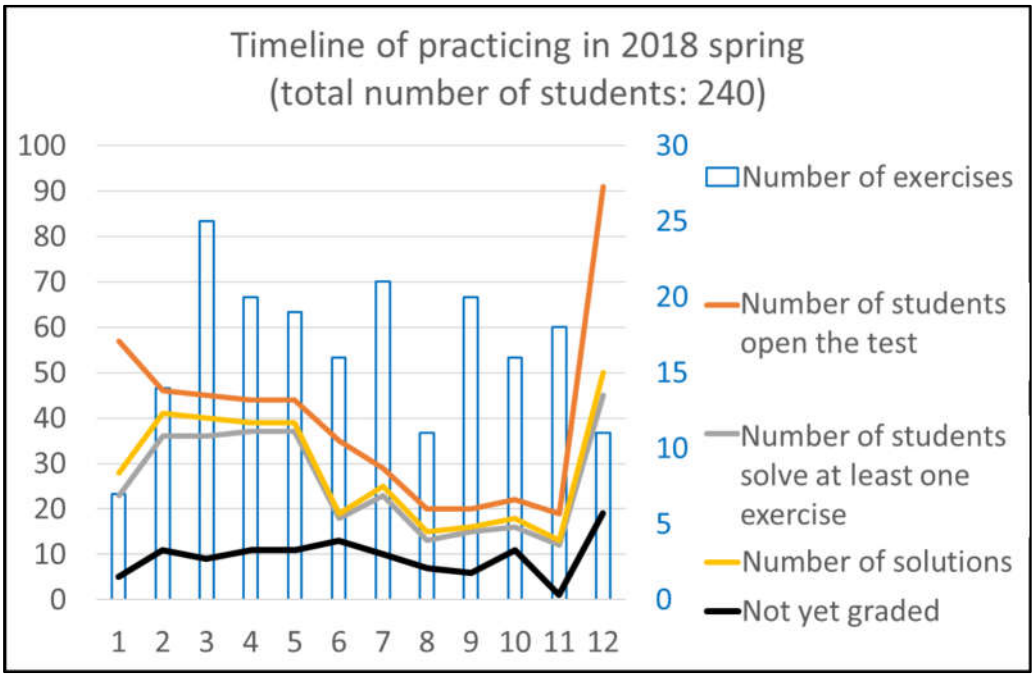


Figure 1: Practical exercises and student activity in 2018 spring semester

Our problems in 2018 spring semester:

- Limited number of exercises
- Students did not have immediate feedback (more days delay)
- Giving feedback was a time-consuming task for the teachers (5-10 minutes/solutions)
- Less than 10% of the students practiced regularly

As shown on Figure 1, number of practicing exercises varied between 7 and 25 and they were the same exercises as we solved on the lessons. In the examined semester there were 240 students on the course but much less than half of them even opened the exercises at home. Those, who solved at least one exercise remained under 20% of the participants.

On Figure 2 we can see on purple columns the average delay of the feedback (left vertical axis). For example, the two students who submitted their solutions on the first day, waited more than 6 days for the feedback. On green columns we can check the time spent with correcting the solutions (right vertical axis).

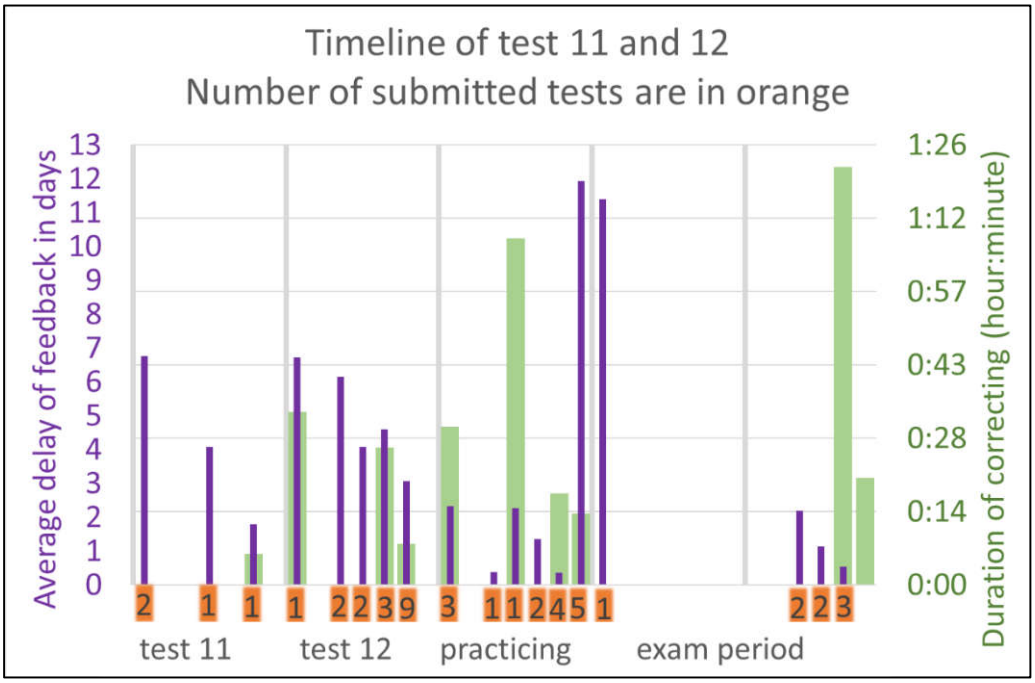


Figure 2: Solutions and correction of test 11 and test 12

2. Possible solutions

Our course is supported by Moodle learning system. In Moodle we can create multiple choice questions, so the students can get immediate feedback [2]. But this kind of exercises are not efficient in learning a practical subject. If we give the possible answers students can choose the correct one more easily and do not have to write queries on their own.

If we use open questions in the tests, we have to find a solution for immediate feedback. In 2018 spring semester we asked the sql codes as answer, but for automatic correction we need a very complex text

analyzation method, or we have to give all the possible correct answers. Regarding the variety of the sql language it is a difficult task.

Our flipped solution is the following: we create a sample database filled with specially constructed data: there are lot of equal values and lot of similar values with small changes. In the exercises students have to write queries but, in the quiz, they have to answer a question in connection with the result of their solution. For example: How many records are there in the result or what is the ID in the second row. It is important to avoid the following problem: in some cases, logically wrong solutions can give accidently the good answer. That is why we have to find those questions where the typically wrong solutions give different answers.

3. The process of creating the questions

By analyzing the exercises used in the last semester and the solutions uploaded by the students we could find the typical mistakes of the basic question types. After creating all the questions variations of the basic question types, we have to select the proper ones. Proper questions are those where you cannot give the correct answer accidently, or even better where we can detect the error type from the wrong answer. Obviously more complex questions have more variations but more error possibilities as well, so the number of proper questions will not grow with the complexity.

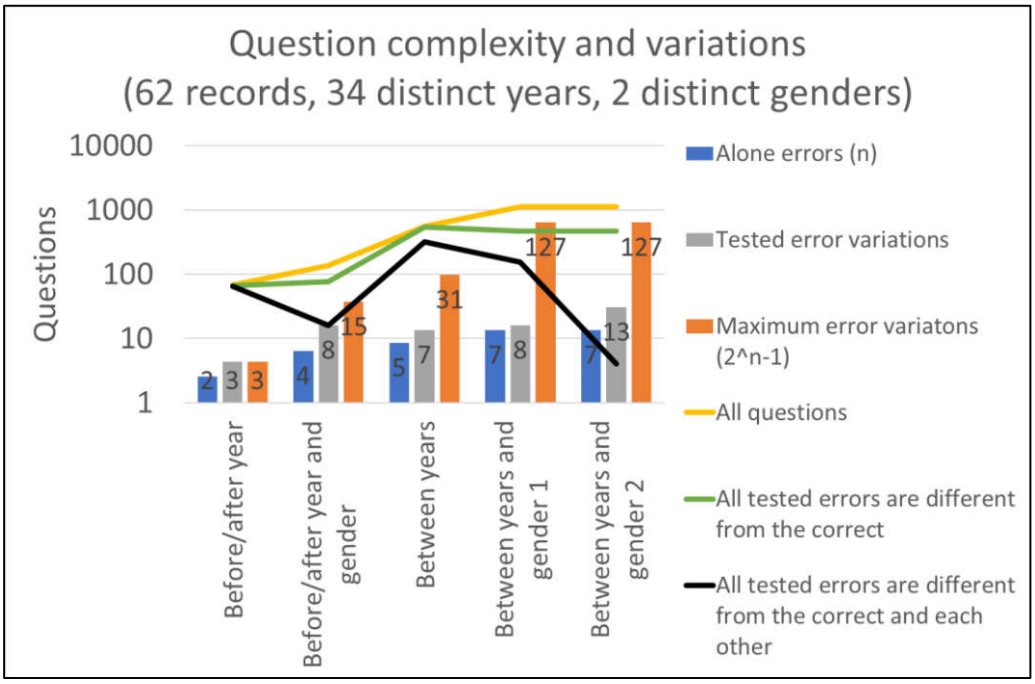
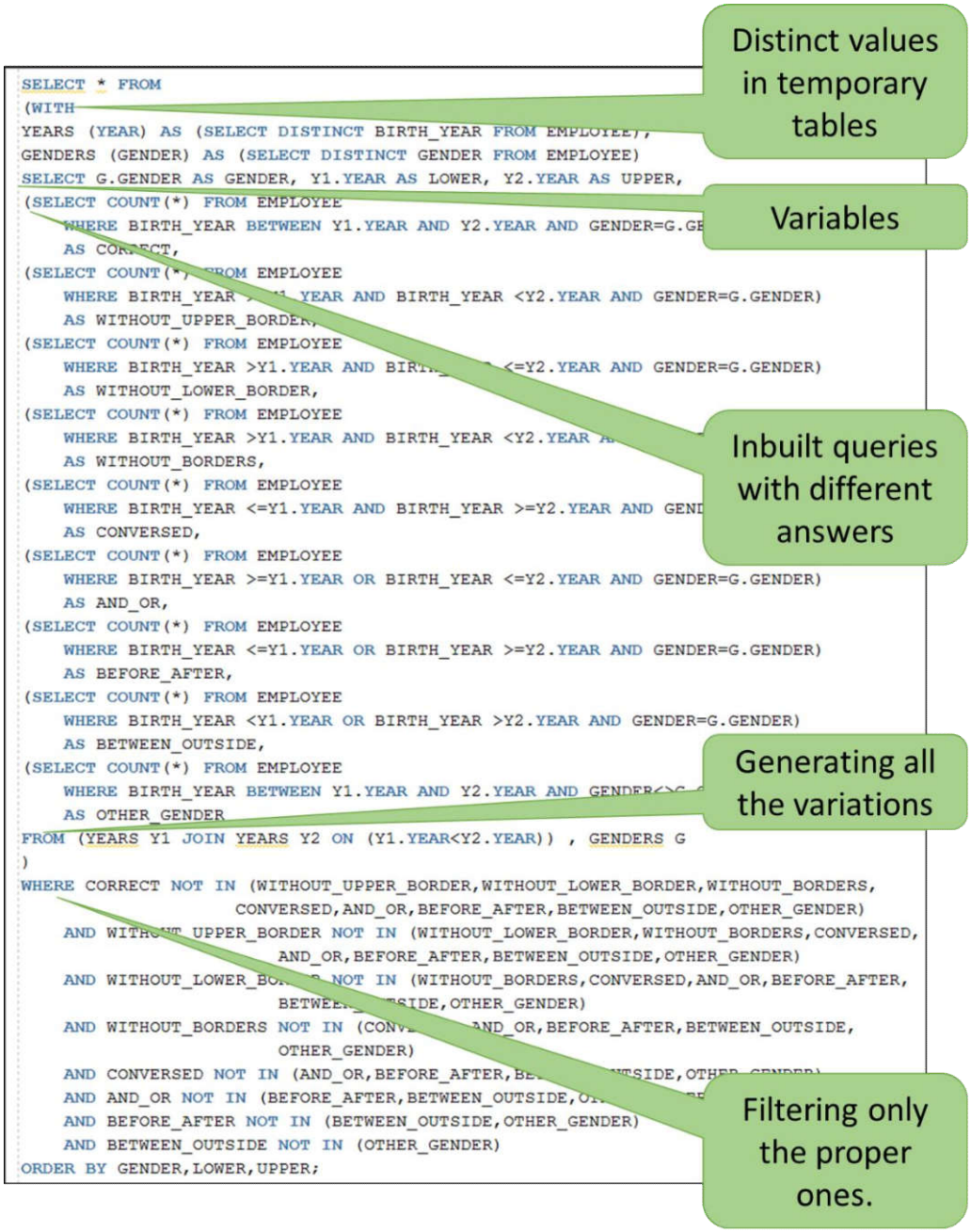


Figure 3: Number of questions according to the complexity of the question

On Figure 3 you can check the numbers of an example. The sample table is about employees with their id, name, birth year and gender. There are 62 records, but only 34 different birth year. The blue columns show the number of typical errors. For example, the exercise is: Write a query to have those workers who was born before 1990. How many records do you have as result? Students can make the following mistakes, write birth_year<=1990 or birth_year>1990 instead of birth_year<1990 in the condition. Or they can combine the mistakes: birth_year>=1990. The maximum error variations are 2^n-1 if n is the number

of the alone mistakes. We have to compromise and check only a few, more common combinations to be able to find enough proper questions.

For each basic question type we can find the proper questions by a sql query (Figure 4).




```
SELECT * FROM
(WITH
YEARS (YEAR) AS (SELECT DISTINCT BIRTH_YEAR FROM EMPLOYEE),
GENDERS (GENDER) AS (SELECT DISTINCT GENDER FROM EMPLOYEE)
SELECT G.GENDER AS GENDER, Y1.YEAR AS LOWER, Y2.YEAR AS UPPER,
(SELECT COUNT(*) FROM EMPLOYEE
    WHERE BIRTH_YEAR BETWEEN Y1.YEAR AND Y2.YEAR AND GENDER=G.GE
    AS CORRECT,
(SELECT COUNT(*) FROM EMPLOYEE
    WHERE BIRTH_YEAR >Y1.YEAR AND BIRTH_YEAR <Y2.YEAR AND GENDER=G.GENDER)
    AS WITHOUT_UPPER_BORDER,
(SELECT COUNT(*) FROM EMPLOYEE
    WHERE BIRTH_YEAR >Y1.YEAR AND BIRTH_YEAR <=Y2.YEAR AND GENDER=G.GENDER)
    AS WITHOUT_LOWER_BORDER,
(SELECT COUNT(*) FROM EMPLOYEE
    WHERE BIRTH_YEAR >Y1.YEAR AND BIRTH_YEAR <Y2.YEAR A
    AS WITHOUT_BORDERS,
(SELECT COUNT(*) FROM EMPLOYEE
    WHERE BIRTH_YEAR <=Y1.YEAR AND BIRTH_YEAR >=Y2.YEAR AND GEND
    AS CONVERSED,
(SELECT COUNT(*) FROM EMPLOYEE
    WHERE BIRTH_YEAR >=Y1.YEAR OR BIRTH_YEAR <=Y2.YEAR AND GENDER=G.GENDER)
    AS AND_OR,
(SELECT COUNT(*) FROM EMPLOYEE
    WHERE BIRTH_YEAR <=Y1.YEAR OR BIRTH_YEAR >=Y2.YEAR AND GENDER=G.GENDER)
    AS BEFORE_AFTER,
(SELECT COUNT(*) FROM EMPLOYEE
    WHERE BIRTH_YEAR <Y1.YEAR OR BIRTH_YEAR >Y2.YEAR AND GENDER=G.GENDER)
    AS BETWEEN_OUTSIDE,
(SELECT COUNT(*) FROM EMPLOYEE
    WHERE BIRTH_YEAR BETWEEN Y1.YEAR AND Y2.YEAR AND GENDER<>G.
    AS OTHER_GENDER
FROM (YEARS Y1 JOIN YEARS Y2 ON (Y1.YEAR<Y2.YEAR)) , GENDERS G
)
WHERE CORRECT NOT IN (WITHOUT_UPPER_BORDER,WITHOUT_LOWER_BORDER,WITHOUT_BORDERS,
                      CONVERSED,AND_OR,BEFORE_AFTER,BETWEEN_OUTSIDE,OTHER_GENDER)
    AND WITHOUT_UPPER_BORDER NOT IN (WITHOUT_LOWER_BORDER,WITHOUT_BORDERS,CONVERSED,
                          AND_OR,BEFORE_AFTER,BETWEEN_OUTSIDE,OTHER_GENDER)
    AND WITHOUT_LOWER_BORDER NOT IN (WITHOUT_BORDERS,CONVERSED,AND_OR,BEFORE_AFTER,
                          BETWEEN_OUTSIDE,OTHER_GENDER)
    AND WITHOUT_BORDERS NOT IN (CONVERSED,AND_OR,BEFORE_AFTER,BETWEEN_OUTSIDE,
                          OTHER_GENDER)
    AND CONVERSED NOT IN (AND_OR,BEFORE_AFTER,BETWEEN_OUTSIDE,OTHER_GENDER)
    AND AND_OR NOT IN (BEFORE_AFTER,BETWEEN_OUTSIDE,OTHER_GENDER)
    AND BEFORE_AFTER NOT IN (BETWEEN_OUTSIDE,OTHER_GENDER)
    AND BETWEEN_OUTSIDE NOT IN (OTHER_GENDER)
ORDER BY GENDER,LOWER,UPPER;
```

Figure 4: SQL query for finding the proper questions

In the first part of the code we select the distinct values into temporary tables. In the select list we show the current values of the variables in the question and in subqueries all the examined solutions. In the

form clause we generate all the possible variations (sometimes we need cartesian product, sometimes join). Finally, we put this whole query into a subquery and write a condition to filter only the proper ones.

This query will give the variables of the question, the correct answer and the different wrong answers. We need only one further step the create the gift code from these numbers to be able to import the questions into the Moodle.

1	1948	1981	16	14	15	13	0	61	9	5	36
1	1949	1981	15	13	14	12	0	60	11	7	35
1	1957	1981	13	11	12	10	0	51	21	18	26
1	1958	1981	12	10	11	9	0	51	22	19	26
1	1960	1981	11	9	10	8	0	51	25	20	26
1	1965	1981	10	8	9	7	0	45	30	27	20
1	1967	1981	9	7	8	6	0	43	35	30	18
1	1972	1981	8	6	7	5	0	36	42	38	11
1	1973	1981	7	5	6	4	0	35	44	40	10
1	1974	1981	6	4	5	3	0	34	46	42	9
1	1978	1981	5	3	4	2	0	31	53	46	6
2	1947	1954	9	6	8	5	0	62	37	33	3
2	1947	1960	13	11	12	10	0	62	32	29	6
2	1947	1961	15	13	14	12	0	62	30	27	6
2	1947	1966	19	17	18	16	0	62	26	23	7

Figure 5: A few proper questions created by the sql query shown in the previous figure

4. Application

If we create a test bank following this method, we can use it to give feedback to the students during practice. Because we ask only results this solution is system independent, can be used with any kind of database management system. This kind of test bank cannot be used for DDL codes and advanced, very complex queries, only for the basics. It is even inadequate in exam situation because the correct answers can be guessed in other ways as well. In our university we will test our test bank in the 2019 spring semester with more than 200 students.

References

[1] R. Elmasri, S. Navathe: Fundamentals of Database Systems Sixth Edition, Addison-Wesley Publishing Company, USA 2010.

[2] Moodle: Multiple Choice question type https://docs.moodle.org/35/en/Multiple_Choice_question_type

[3] Moodle: GIFT format https://docs.moodle.org/35/en/GIFT_format