Подборка SQL-заданий

Запрос №1: жанры с максимальным числом заказанных экземпляров

База данных

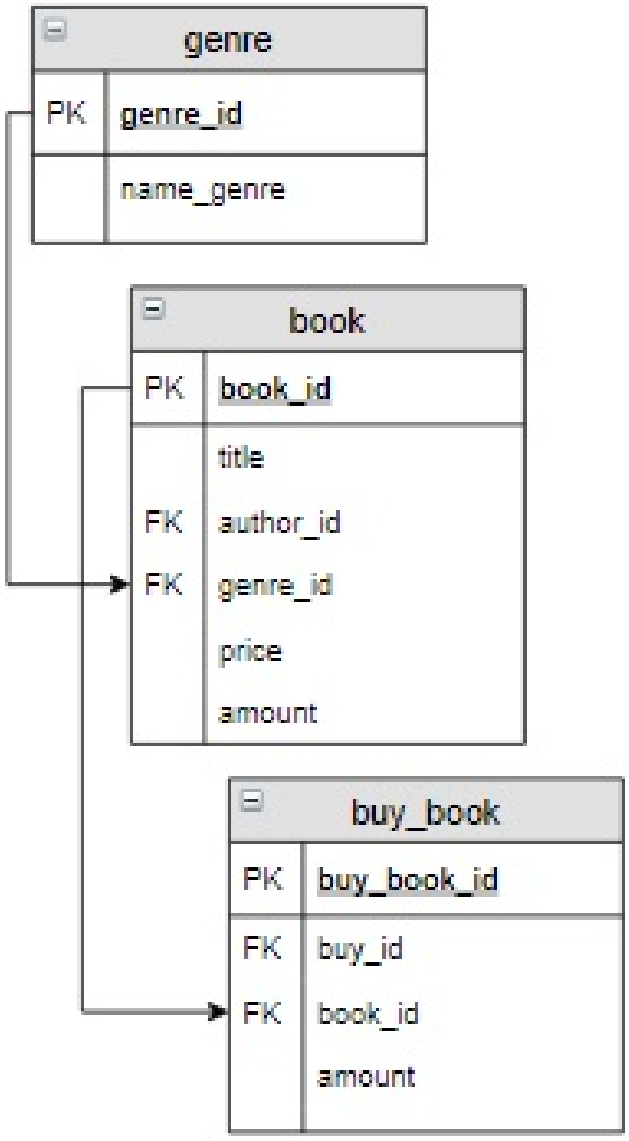

Рис. 1: Фрагмент схемы: `genre` — `book` — `buy_book`

Формулировка

Вывести жанр (или жанры), в котором было заказано больше всего экземпляров книг, указать это количество. Последний столбец назвать Количество.

Решение

```
SELECT name_genre, SUM(buy_book.amount) AS Количество
FROM genre
    INNER JOIN book USING (genre_id)
    INNER JOIN buy_book USING (book_id)
GROUP BY name_genre
HAVING Количество = (
    SELECT SUM(buy_book.amount) AS Колво
    FROM genre
        INNER JOIN book USING (genre_id)
        INNER JOIN buy_book USING (book_id)
    GROUP BY name_genre
    LIMIT 1
);
```

Объяснение

1. Соединяем таблицы `genre` → `book` → `buy_book`.
2. Считаем `SUM` количества экземпляров по каждому жанру.
3. Через **HAVING** оставляем только те жанры, у которых сумма совпала с подзапросом.

Запрос №2: сравнение ежемесячной выручки за два года

База данных

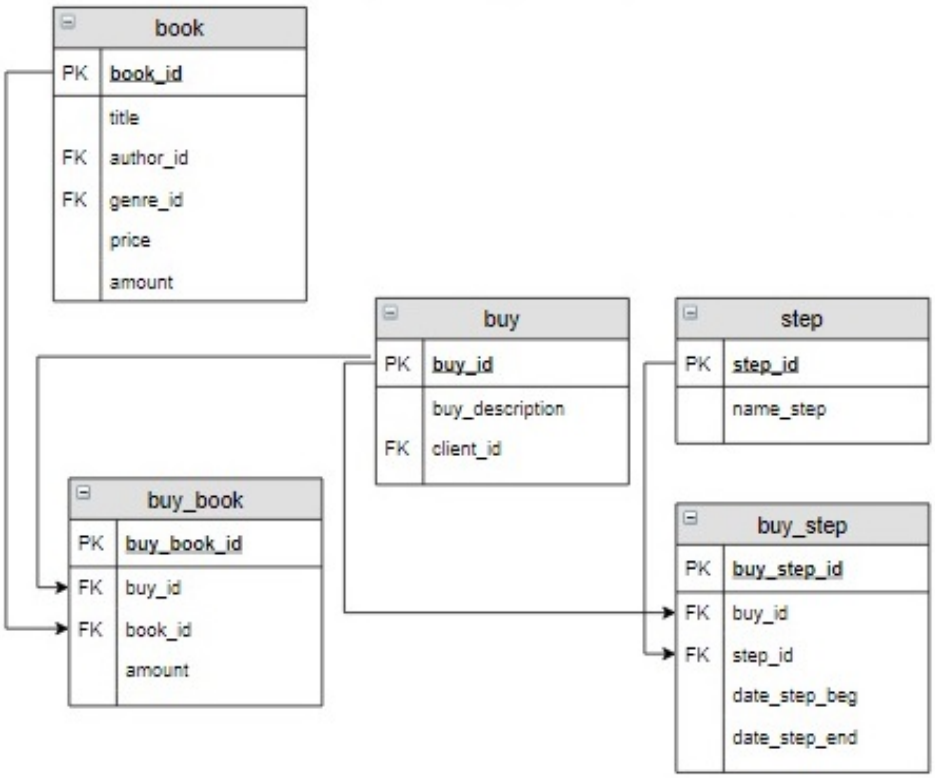

Рис. 2: Схема: `book`, `buy`, `buy_book`, `step`, `buy_step`

Формулировка

Сравнить ежемесячную выручку от продажи книг за текущий и предыдущий годы. Вывести Год, Месяц, Сумма. Отсортировать сначала по возрастанию месяцев, затем по возрастанию лет.

Решение

```
SELECT YEAR(date_payment) AS Год,
       MONTHNAME(date_payment) AS Месяц,
       SUM(price * amount) AS Сумма
FROM buy_archive
GROUP BY YEAR(date_payment), MONTHNAME(date_payment)

UNION ALL

SELECT YEAR(date_step_end) AS Год,
       MONTHNAME(date_step_end) AS Месяц,
```

```
       SUM(book.price * buy_book.amount) AS Сумма
FROM buy_step
    INNER JOIN buy_book USING (buy_id)
    INNER JOIN book USING (book_id)
WHERE step_id = 1
  AND date_step_end IS NOT NULL
GROUP BY YEAR(date_step_end), MONTHNAME(date_step_end)

ORDER BY Месяц, Год;
```

Объяснение

1. Исторические продажи берутся из `buy_archive` по `date_payment`.
2. Текущие продажи — из `buy_step` для шага оплаты (`step_id = 1`), сумма считается по цене книги и количеству.
3. Итоги объединяются через **UNION ALL**, группируются и сортируются по месяцу, затем году.

Запрос №3: студенты с максимальным результатом попыток

База данных

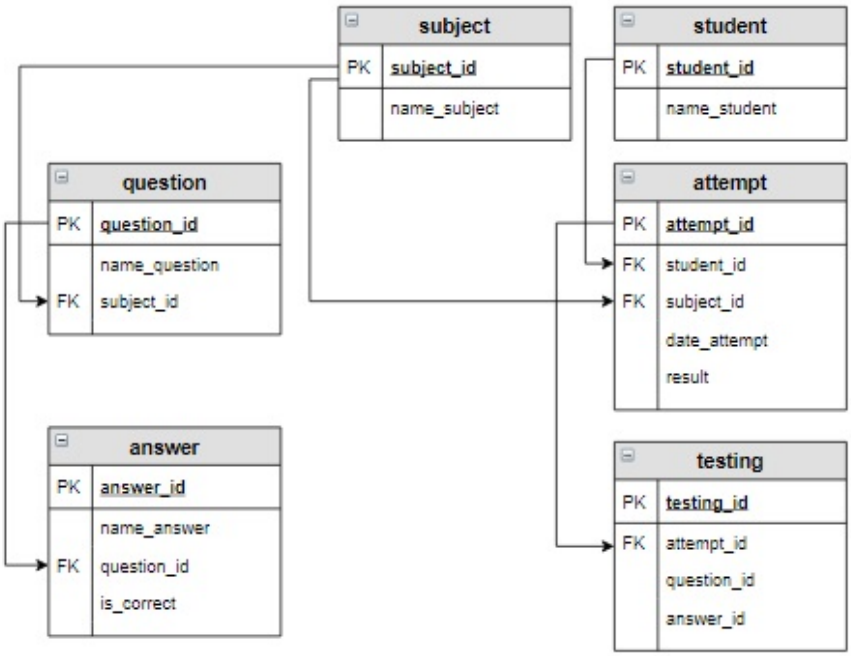

Рис. 3: Схема: `student`, `attempt` и связанные таблицы

Формулировка

Вывести студентов (различных студентов), имеющих максимальные результаты попыток. Отсортировать по фамилии студента в алфавитном порядке.

Решение

```
SELECT DISTINCT name_student, result
FROM student
    INNER JOIN attempt USING (student_id)
WHERE result = (
    SELECT result
    FROM attempt
    ORDER BY result DESC
    LIMIT 1
)
ORDER BY name_student;
```

Объяснение

1. Соединяем `student` с `attempt` по `student_id`.
2. Во вложенном запросе берём максимальный `result` как первую строку из сортировки по убыванию.
3. Через фильтр `WHERE` оставляем только попытки с этим максимальным значением.
4. `DISTINCT` убирает дубли, если у одного студента несколько попыток с одинаковым максимумом.

Запрос №4: успешность по каждому вопросу

База данных

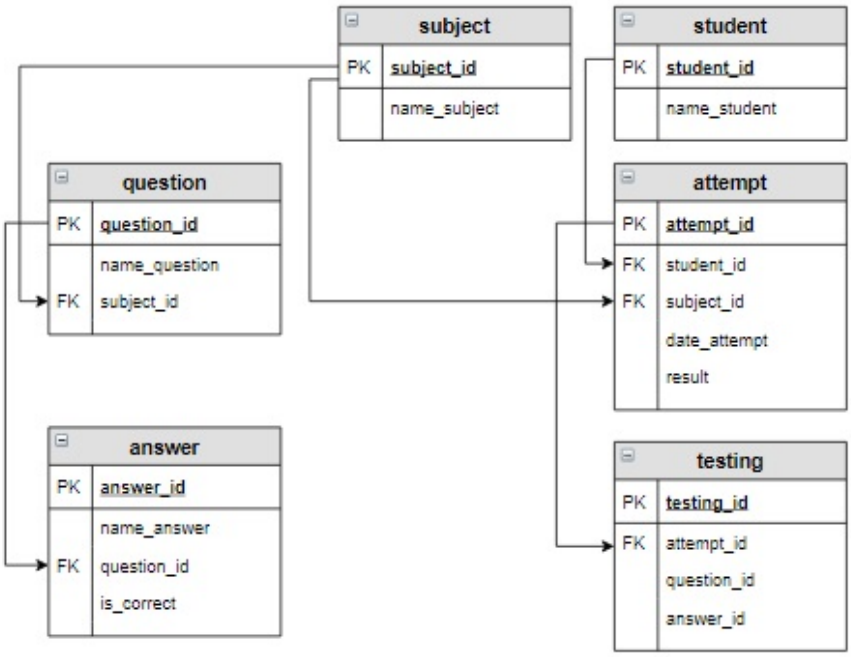

Рис. 4: Схема: `subject`, `question`, `answer`, `testing`

Формулировка

Для каждого вопроса вывести процент успешных решений — отношение количества верных ответов к общему количеству ответов, округлённое до двух знаков. Включить дисциплину, укороченный текст вопроса (столбец Вопрос), а также вычисляемые столбцы Всего_ответов и Успешность. Отсортировать: по названию дисциплины, затем по убыванию успешности, затем по тексту вопроса по алфавиту. Текст вопроса обрезать до 30 символов и добавить многоточие "...".

Решение

```
SELECT name_subject,
       CONCAT(SUBSTR(name_question, 1, 30), "...") AS Вопрос,
       COUNT(testing.question_id) AS Всего_ответов,
       ROUND((SUM(is_correct) / COUNT(testing.question_id)) * 100, 2) AS Успешность
FROM testing
    LEFT JOIN answer  USING (answer_id)
    INNER JOIN question ON testing.question_id = question.question_id
    INNER JOIN subject USING (subject_id)
GROUP BY name_subject, name_question
```

```
ORDER BY name_subject, Успешность DESC, name_question;
```

Объяснение

1. `testing` хранит выбранные ответы; через `LEFT JOIN answer` берём признак `is_correct` (0/1) для каждой записи.
2. **Всего_ответов**: `COUNT(testing.question_id)` — число ответов на вопрос.
3. **Успешность**: доля верных ответов $\frac{\sum \texttt{is\_correct}}{\texttt{COUNT}} \times 100$, округление `ROUND(..., 2)`.
4. В выборку добавляем предмет из `subject` и сокращённый текст вопроса.

Комментарии

- `LEFT JOIN answer` позволяет корректно считать количество ответов даже при отсутствующем `answer` (если такое возможно); `SUM` игнорирует `NULL`.
- `GROUP BY` выполнен по исходному `name_question`; отображаем обрезанную версию через `CONCAT+SUBSTR`.
- Порядок сортировки: дисциплина ↑, успешность ↓, вопрос (по алфавиту) ↑.

Запрос №5: формирование вспомогательной таблицы applicant

База данных

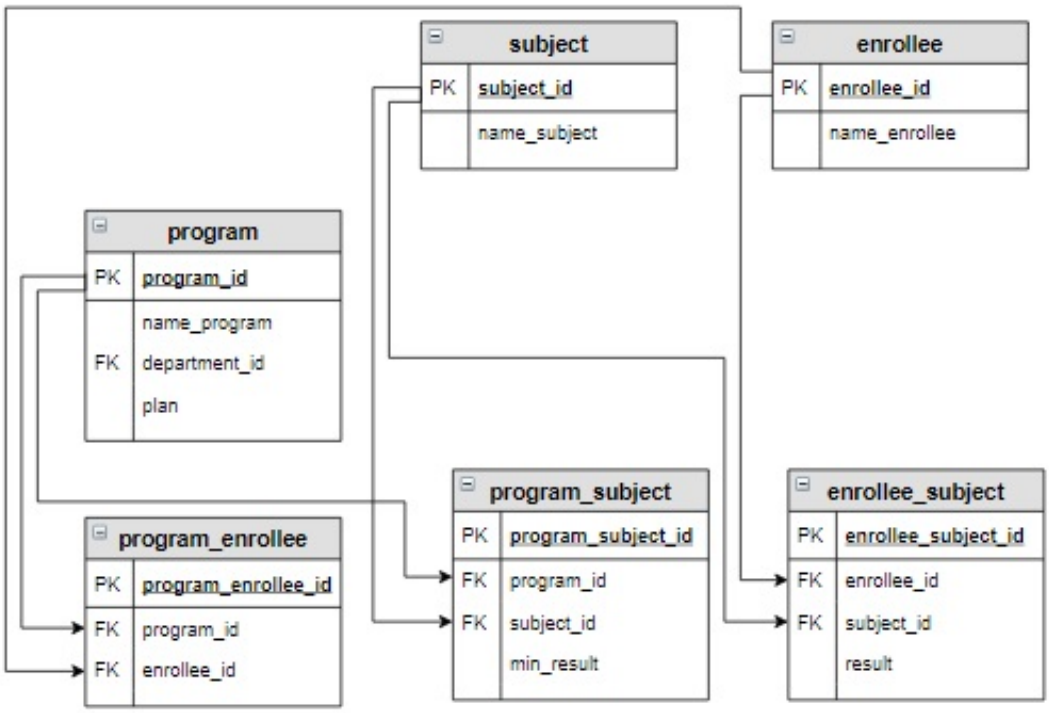

Рис. 5: Схема для заданий 5–6: `program`, `program_enrollee`, `program_subject`, `enrollee_subject`, `subject`, `enrollee`

Формулировка

Создать вспомогательную таблицу `applicant`, куда включить `program_id`, `enrollee_id` и сумму баллов абитуриентов (столбец itog), в отсортированном сначала по `program_id`, а потом по убыванию суммы виде.

Решение

```
CREATE TABLE applicant AS
SELECT program_id,
       program_enrollee.enrollee_id,
       SUM(result) AS itog
FROM program_enrollee
    LEFT JOIN program_subject USING (program_id)
    LEFT JOIN enrollee_subject
           ON program_enrollee.enrollee_id = enrollee_subject.enrollee_id
          AND program_subject.subject_id   = enrollee_subject.subject_id
GROUP BY program_id, enrollee_id
```

```
ORDER BY program_id, itog DESC;
```

Объяснение

1. Из `program_enrollee` берём пары программа–абитуриент.
2. Через связи с `program_subject` и `enrollee_subject` собираем баллы по требуемым предметам.
3. Суммируем результаты по каждой паре и записываем в новую таблицу `applicant`.

Комментарии

- Порядок строк в физической таблице не гарантируется: `ORDER BY` в `CREATE TABLE ... SELECT` влияет на результирующий набор, но не «фиксирует» порядок хранения. Для просмотра используйте `SELECT ... ORDER BY`.
- Если каких-то предметов нет у абитуриента, `LEFT JOIN` даёт `NULL`; сумма по реально найденным предметам всё равно считается корректно.

Запрос №6: удаление абитуриентов, не прошедших минимумы

База данных

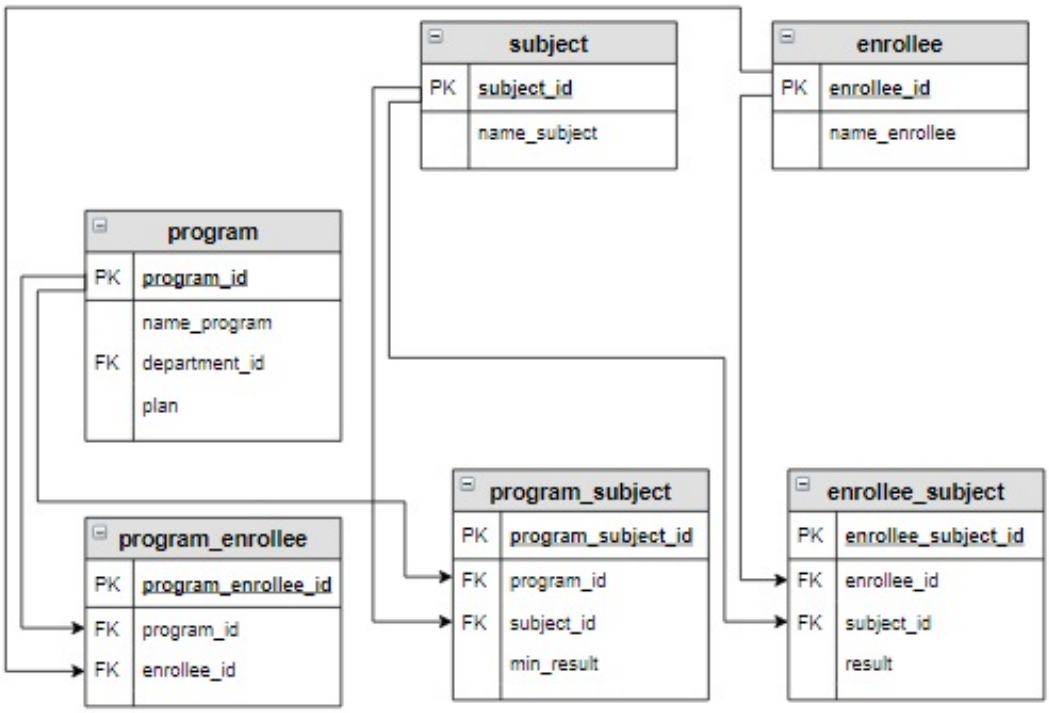

Рис. 6: Схема для заданий 5–6 (та же, что выше)

Формулировка

Из таблицы `applicant` удалить записи, если абитуриент на выбранную программу не набрал минимального балла хотя бы по одному предмету.

Решение

```
DELETE FROM applicant
USING applicant
    INNER JOIN program_enrollee USING (enrollee_id)
    LEFT JOIN program_subject
           ON program_enrollee.program_id = program_subject.program_id
    LEFT JOIN enrollee_subject
           ON program_enrollee.enrollee_id = enrollee_subject.enrollee_id
          AND program_subject.subject_id   = enrollee_subject.subject_id
WHERE result <= min_result
  AND applicant.program_id = program_enrollee.program_id;
```

Объяснение

1. Для каждой записи `applicant` находим связанные предметы программы и результаты абитуриента.
2. Если по какому-либо предмету `result` $\leq$ `min_result`, соответствующая запись `applicant` удаляется.

Комментарии

- Синтаксис `DELETE ... USING ... JOIN` — диалект MySQL/MariaDB; в других СУБД потребуется эквивалент с подзапросом.
- Условие `applicant.program_id = program_enrollee.program_id` гарантирует сопоставление в рамках той же программы.

Запрос №7: добавление бонусов к итоговым баллам

База данных

Рис. 7: Схема для задания 7: `achievement`, `enrollee_achievement`

Формулировка

Повысить итоговые баллы в `applicant` на суммы дополнительных баллов абитуриентов.

Решение

```
UPDATE applicant, (
    SELECT enrollee_id,
           IF(SUM(bonus) IS NULL, 0, SUM(bonus)) AS Bonus
    FROM enrollee
        LEFT JOIN enrollee_achievement USING (enrollee_id)
        LEFT JOIN achievement          USING (achievement_id)
    GROUP BY enrollee_id
) AS bonusnaya
SET itog = itog + Bonus
WHERE bonusnaya.enrollee_id = applicant.enrollee_id;
```

Объяснение

1. Во вложенном запросе считаем суммарный `bonus` по каждому `enrollee_id` (`LEFT JOIN` учитывает отсутствие достижений).
2. Обновляем `applicant.itog`, прибавляя рассчитанный бонус соответствующего абитуриента.

Комментарии

- В MySQL эквивалентно можно использовать `IFNULL(SUM(bonus), 0)` вместо `IF(SUM(bonus) IS NULL, 0, ...)`.
- Обновляется только сумма `itog`; структура таблицы `applicant` не меняется.

Запрос №8: заполнение step_keyword по вхождению ключевых слов

База данных

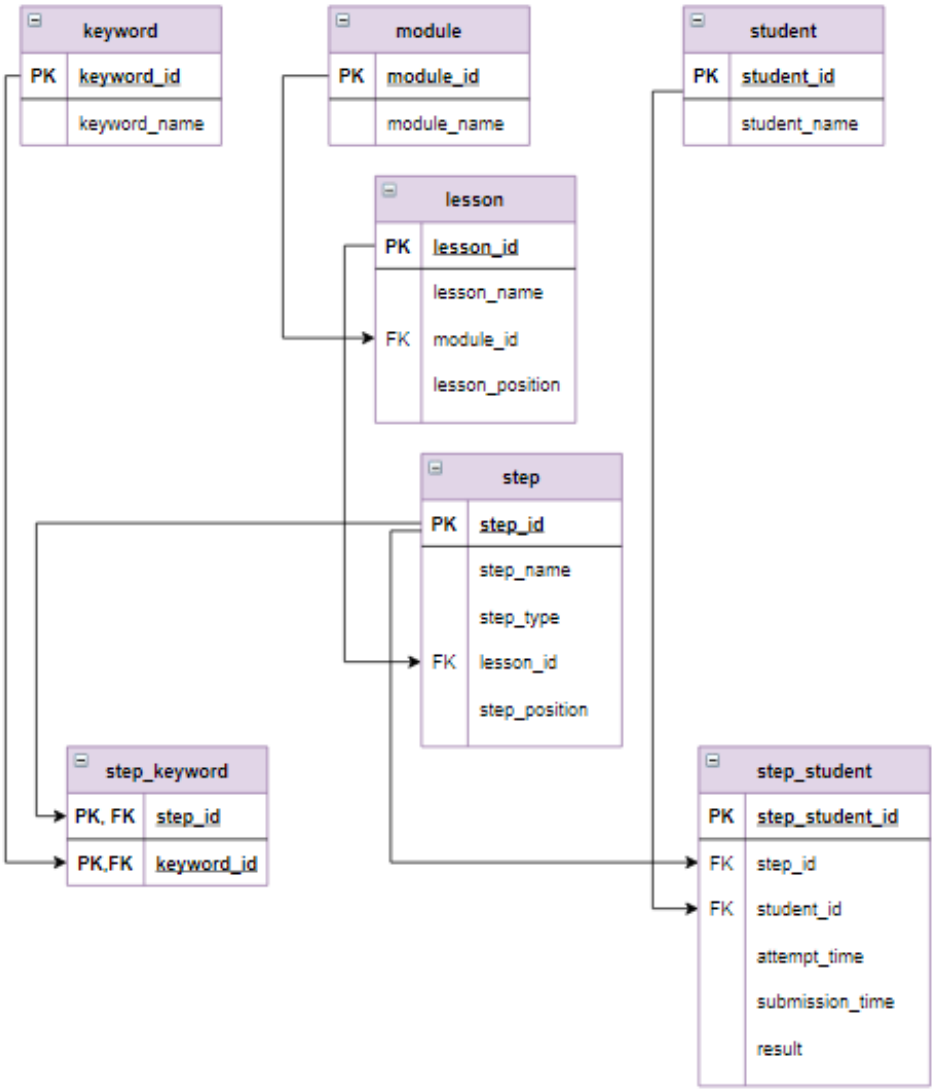

Рис. 8: Схема: `module/lesson/step`, `student`, `step_student`, `keyword`, `step_keyword`

Формулировка

Заполнить таблицу `step_keyword` так: если ключевое слово содержится в названии шага, добавить строку с id шага и id ключевого слова.

Решение

```
INSERT INTO step_keyword (step_id, keyword_id)
SELECT s.step_id, k.keyword_id
```

```
FROM step s
JOIN keyword k
ON (
    INSTR(CONCAT(' ', s.step_name, ' '), CONCAT(' ', k.keyword_name, ' ')) > 0
    OR INSTR(CONCAT(' ', s.step_name, ' '), CONCAT(' ', k.keyword_name, ',')) > 0
    OR INSTR(CONCAT(' ', s.step_name, ' '), CONCAT(' ', k.keyword_name, '(')) > 0
    );
```

Объяснение

1. Для каждой пары шаг–ключевое слово проверяется вхождение слова в название шага.
2. Добавочные пробелы/знаки (пробел, запятая, скобка) позволяют матчить *слово целиком*.

Комментарии

- `INSTR` регистрозависим/независим в зависимости от коллации; при необходимости использовать нижний регистр в обеих частях.
- Если в названии шага одно слово встретится несколько раз, вставится всё равно одна строка (так как выбирается одна пара `step_id, keyword_id`).

Запрос №9: распределение студентов по группам по числу решённых шагов

База данных

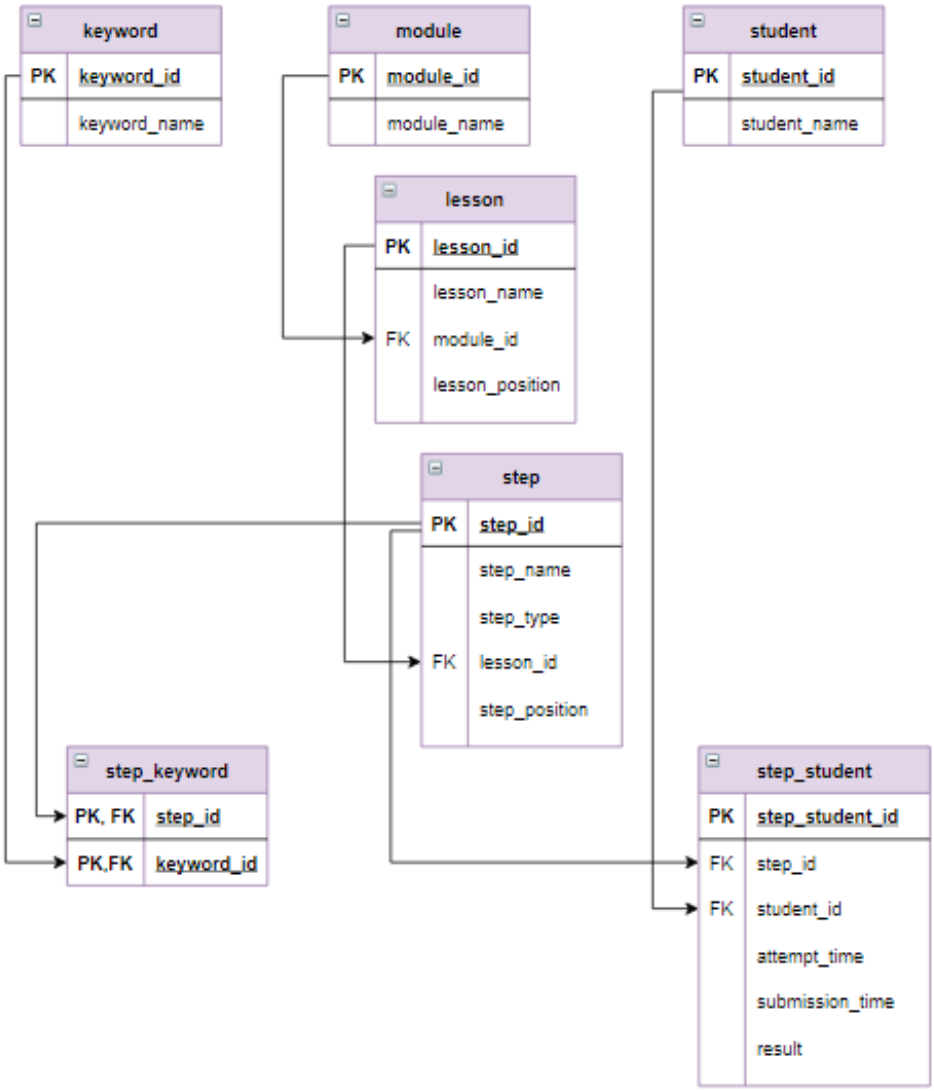

Рис. 9: Схема: `student`, `step_student`

Формулировка

Посчитать, сколько студентов относится к каждой группе (I, II, III, IV) по их количеству решённых шагов. Столбцы: Группа, Интервал, Количество. Указать границы интервала.

Решение

```
SELECT Группа,
CASE
```

```
    WHEN Группа = "I"   THEN "от 0 до 10"
    WHEN Группа = "II"  THEN "от 11 до 15"
    WHEN Группа = "III" THEN "от 16 до 27"
    WHEN Группа = "IV"  THEN "больше 27"
END AS Интервал, COUNT(*) AS Количество
FROM (
    SELECT student_name, rate,
    CASE
        WHEN rate <= 10 THEN "I"
        WHEN rate <= 15 THEN "II"
        WHEN rate <= 27 THEN "III"
        ELSE "IV"
    END AS Группа
    FROM (
      SELECT student_name, COUNT(*) AS rate
      FROM (
        SELECT student_name, step_id
        FROM student
        INNER JOIN step_student USING(student_id)
        WHERE result = "correct"
        GROUP BY student_name, step_id
      ) query_in
      GROUP BY student_name
      ORDER BY 2
    ) query_in_1
) query_in_2
GROUP BY Группа;
```

Объяснение

1. Для каждого студента считаем число уникальных шагов с результатом `"correct"`.
2. По числу шагов присваиваем группу и агрегируем количество студентов в каждой группе.

Комментарии

- Интервалы заданы в `CASE`; при изменении границ достаточно поправить выражение.

Запрос №10: успешность шагов с учётом отсутствующих правильных/неправильных попыток

База данных

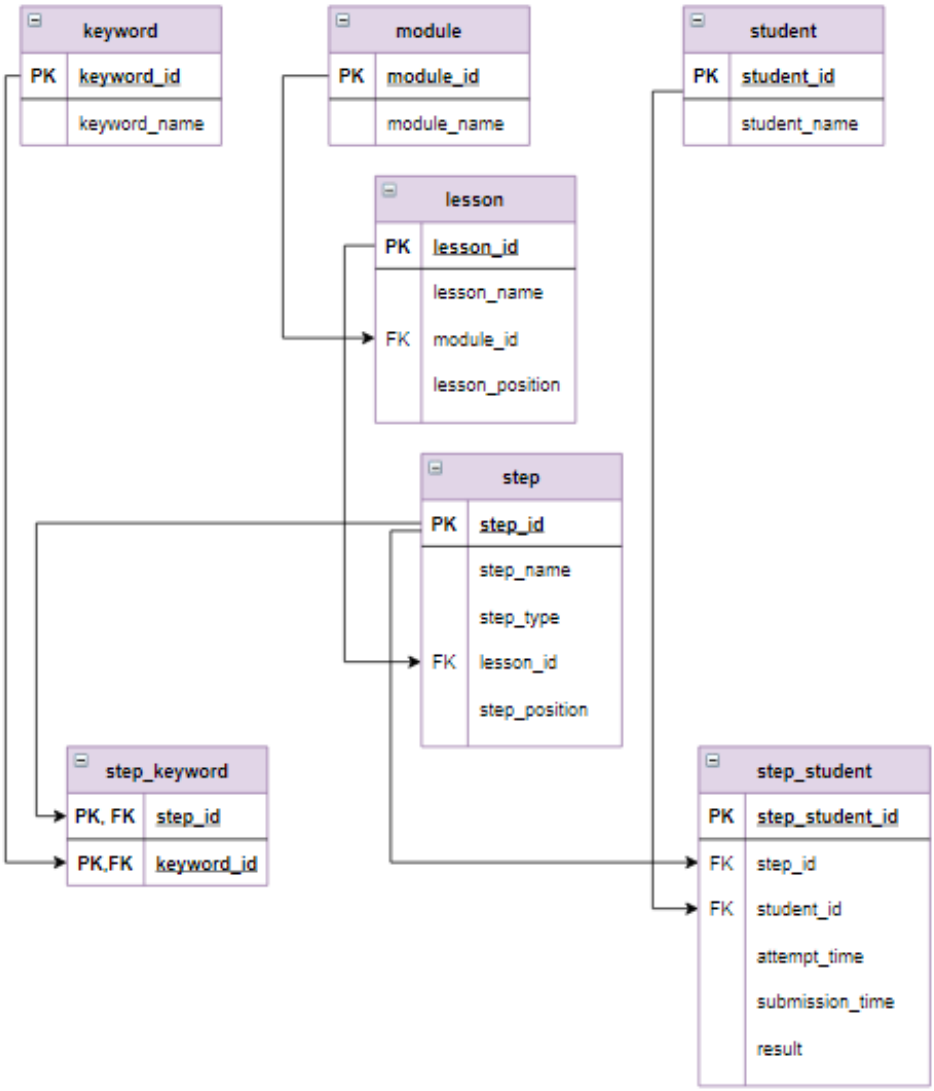

Рис. 10: Схема: `step`, `step_student`

Формулировка

Исправить запрос так, чтобы для шагов без неверных ответов ставить 100%, а без верных — 0%. Отсортировать по возрастанию успешности, затем по алфавиту названия шага.

Решение

```
WITH get_count_correct (st_n_c, count_correct) AS (
  SELECT step_name, COUNT(*)
```

```
  FROM step INNER JOIN step_student USING (step_id)
  WHERE result = "correct"
  GROUP BY step_name
),
get_count_wrong (st_n_w, count_wrong) AS (
  SELECT step_name, COUNT(*)
  FROM step INNER JOIN step_student USING (step_id)
  WHERE result = "wrong"
  GROUP BY step_name
)
SELECT st_n_c AS Шаг,
       ROUND(IFNULL(count_correct, 0) /
             (IFNULL(count_correct, 0) + IFNULL(count_wrong, 0)) * 100) AS Успешнос
   ть
FROM get_count_correct
LEFT JOIN get_count_wrong ON st_n_c = st_n_w
UNION
SELECT st_n_w AS Шаг,
       ROUND(IFNULL(count_correct, 0) /
             (IFNULL(count_correct, 0) + IFNULL(count_wrong, 0)) * 100) AS Успешнос
   ть
FROM get_count_correct
RIGHT JOIN get_count_wrong ON st_n_c = st_n_w
ORDER BY 2, 1;
```

Объяснение

1. Считаются по отдельности числа верных и неверных попыток на шаг.
2. `LEFT/RIGHT JOIN` и `IFNULL` обеспечивают корректные 100% или 0% при отсутствии одной из сторон.

Комментарии

- Сортировка: успешность ↑, затем шаг по алфавиту.

Запрос №11: прогресс пользователей по курсу и выдача сертификатов

База данных

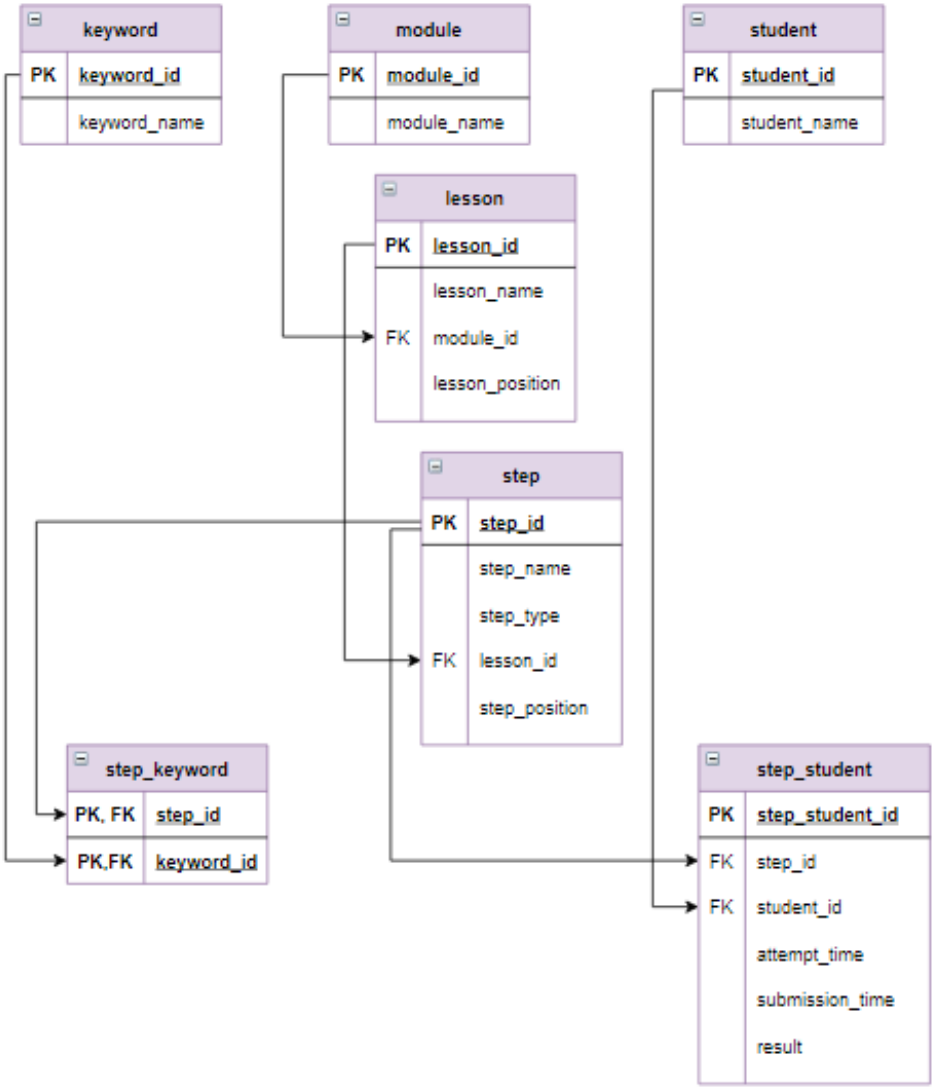

Рис. 11: Схема: `student`, `step_student`

Формулировка

Посчитать прогресс каждого пользователя как отношение числа верно пройденных шагов к общему количеству различных шагов в `step_student`, в процентах (округлить до целого). 100% — «Сертификат с отличием», $\geq$80% — «Сертификат», иначе пусто. Отсортировать по прогрессу $\downarrow$, затем по имени $\uparrow$.

Решение

```
SET @step_sount := (
    SELECT COUNT(DISTINCT step_id)
    FROM step_student
);

WITH student_cor (st_n_cor, count_cor) AS (
    SELECT student_name, COUNT(DISTINCT step_id)
    FROM step_student
    JOIN student USING (student_id)
    WHERE result = "correct"
    GROUP BY student_name
)
SELECT
    st_n_cor AS Студент,
    ROUND((count_cor / @step_sount) * 100, 0) AS Прогресс,
    CASE
        WHEN ROUND((count_cor / @step_sount) * 100, 0) = 100 THEN "Сертификат с отл
    ичием"
        WHEN ROUND((count_cor / @step_sount) * 100, 0) >= 80  THEN "Сертификат"
        ELSE ""
    END AS Результат
FROM student_cor
ORDER BY 2 DESC, 1;
```

Объяснение

1. Вычисляется общее число уникальных шагов и сохраняется в переменную.
2. Для каждого студента считаются уникальные «правильные» шаги и переводятся в проценты.
3. По значению процента формируется текст результата.

Комментарии

- Переменная `@step_sount` используется в расчёте для всех строк.

Запрос №12: среднее время прохождения урока

База данных

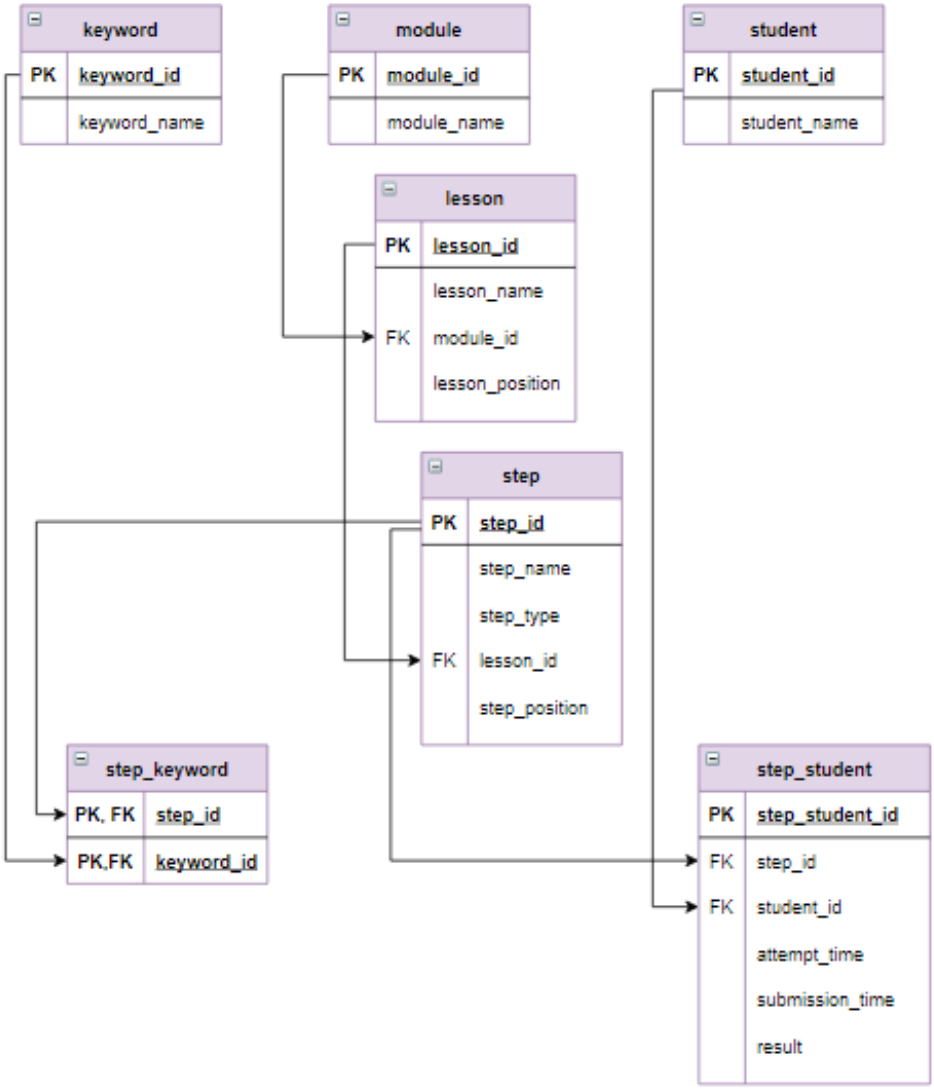

Рис. 12: Схема: `module/lesson/step`, `step_student`

Формулировка

Посчитать среднее время выполнения урока (в часах, с точностью до двух знаков) по алгоритму из условия; вывести Номер, Урок, Среднее_время. Отсортировать по возрастанию времени.

Решение

```
WITH get_time_per_step (student_id, step_id, time_per_step) AS (
    SELECT student_id, step_id, SUM(submission_time - attempt_time)
```

```
    FROM step
    JOIN step_student USING (step_id)
    WHERE submission_time - attempt_time < 4 * 3600
    GROUP BY student_id, step_id
),
get_time_per_lesson (student_id, lesson_id, time_per_lesson) AS (
    SELECT student_id, lesson_id, SUM(time_per_step) / 3600
    FROM lesson
    JOIN step USING (lesson_id)
    JOIN get_time_per_step USING (step_id)
    GROUP BY student_id, lesson_id
),
get_avg_time (lesson_id, avg_time_per_lesson) AS (
    SELECT lesson_id, SUM(time_per_lesson)/COUNT(student_id)
    FROM get_time_per_lesson
    GROUP BY lesson_id
)
SELECT
    ROW_NUMBER() OVER (ORDER BY (avg_time_per_lesson)) AS Номер,
    CONCAT(module_id, '.', lesson_position, ' ', lesson_name) AS Урок,
    ROUND(avg_time_per_lesson, 2) AS Среднее_время
FROM get_avg_time
JOIN lesson USING (lesson_id)
JOIN module USING (module_id);
```

Объяснение

1. Суммируется «нормальное» время попыток по шагу (длительностью < 4 часов).
2. Для каждого студента складывается время по шагам урока; затем усредняется по студентам.
3. Формируется человекочитаемое имя урока и нумерация строк.

Комментарии

- Все вычисления выполняются в секундах; к часам переводится в `get_time_per_lesson`.

Запрос №13: относительный рейтинг в модуле

База данных

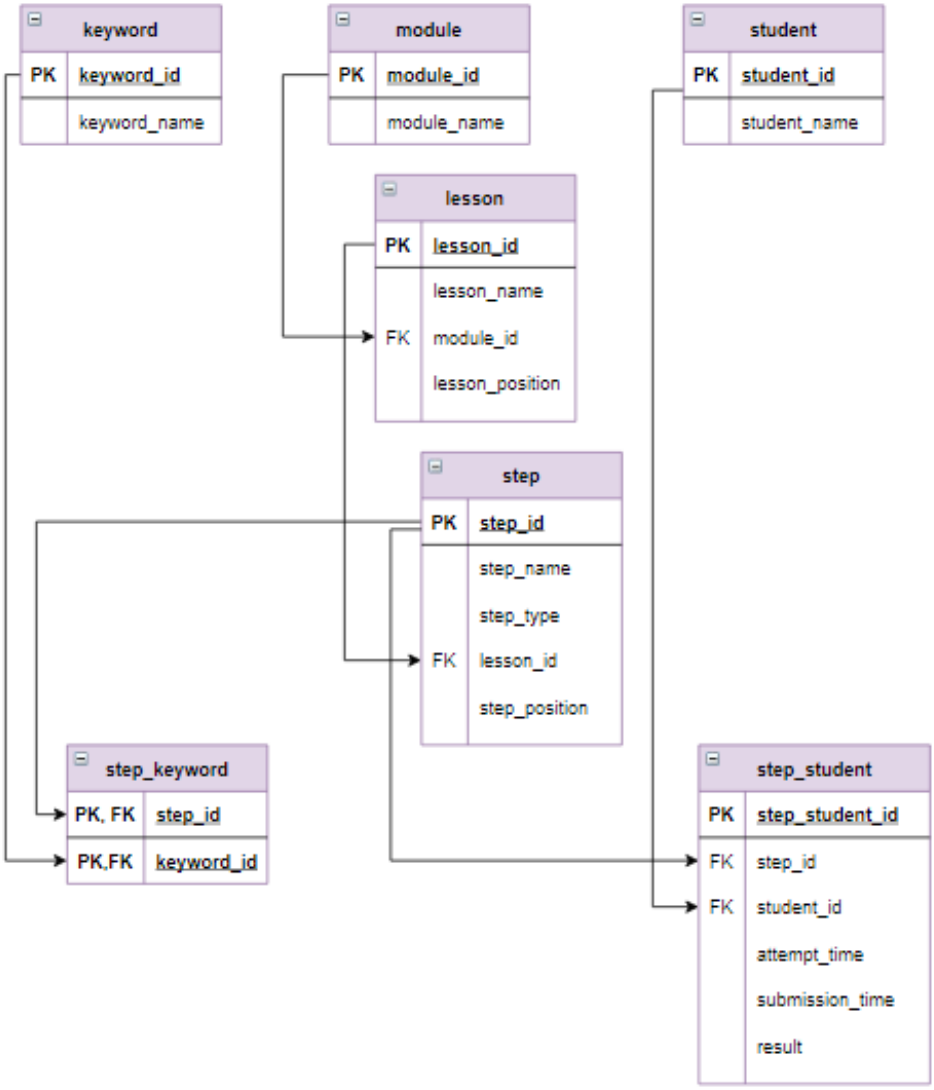

Рис. 13: Схема: `student`, `step_student`, `step`, `lesson`, `module`

Формулировка

Вычислить рейтинг каждого студента относительно максимумов по модулю (в %). Вывести Модуль, Студент, Пройдено_шагов, Относительный_рейтинг. Сортировка: модуль ↑, рейтинг ↓, студент ↑.

Решение

```
WITH get_rate(mod_id, student_id, rate) AS (
   SELECT module_id, student_id, COUNT(DISTINCT step_id)
```

```
   FROM step_student INNER JOIN student USING (student_id)
                     INNER JOIN step USING (step_id)
                     INNER JOIN lesson USING (lesson_id)
   WHERE result = "correct"
   GROUP BY module_id, 2
),
get_module_max(mod_id, student_id, max_value) AS (
    SELECT mod_id AS Модуль, student_id,
           MAX(rate) OVER (PARTITION BY mod_id) AS Максимум_по_модулю
    FROM get_rate
)
SELECT
    gr.mod_id AS Модуль,
    student_name AS Студент,
    rate AS Пройдено_шагов,
    ROUND((rate / max_value) * 100, 1) AS Относительный_рейтинг
FROM get_rate gr
INNER JOIN get_module_max gmm
  ON gr.student_id = gmm.student_id AND gr.mod_id = gmm.mod_id
INNER JOIN student s ON gr.student_id = s.student_id
ORDER BY 1, 4 DESC, 2;
```

Объяснение

1. Для каждой пары модуль–студент считается число уникальных пройденных шагов.
2. По окну внутри модуля берётся максимум, затем считается доля от максимума.

Комментарии

- Округление рейтинга — до одного знака: `ROUND(..., 1)`.

Запрос №14: подробности по попыткам студента_59

База данных

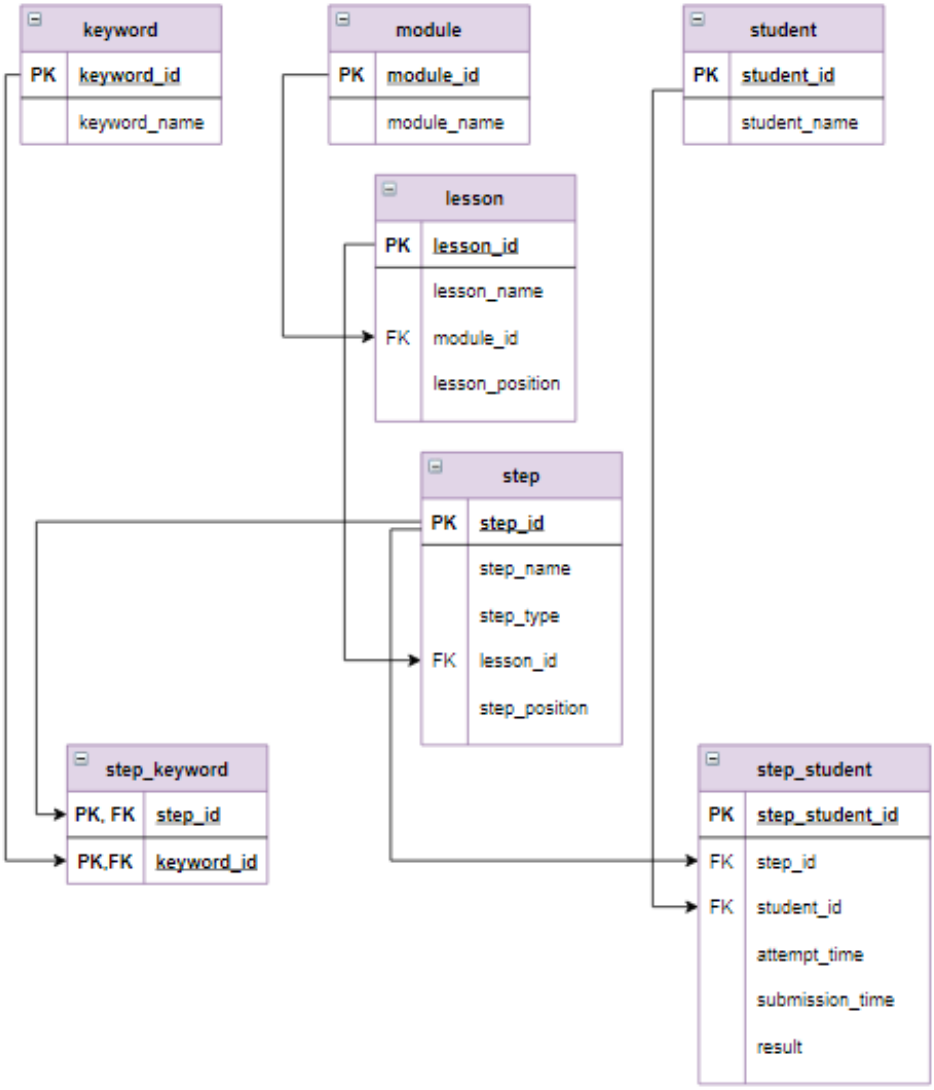

Рис. 14: Схема: `student`, `step_student`, `step`, `lesson`, `module`

Формулировка

Для `student_59` вывести по всем попыткам: информацию о шаге (*module.lesson.step*), порядковый номер попытки (по времени отправки), результат, время попытки (заменить на среднее по пользователю, если попытка > 1 часа), относительное время попытки в %.

Решение

```
WITH
get_number AS (
```

```
    SELECT
        ss.student_id,
        l.module_id,
        l.lesson_id,
        st.step_id,
        CONCAT(l.module_id, '.', l.lesson_position, '.', st.step_position) AS
    step_inf,
        ss.submission_time,
        ss.attempt_time,
        ss.result,
        ROW_NUMBER() OVER (ORDER BY ss.student_id) AS get_number_id
    FROM step_student ss
    JOIN step st USING (step_id)
    JOIN lesson l USING (lesson_id)
    JOIN student s USING (student_id)
    WHERE s.student_name = 'student_59'
),
get_avg_time AS (
    SELECT student_id, ROUND(AVG(submission_time - attempt_time), 0) AS avg_time
    FROM get_number
    WHERE (submission_time - attempt_time) < 3600
    GROUP BY student_id
),
student_new_time AS (
    SELECT
        gn.student_id,
        gn.get_number_id,
        gn.step_inf,
        IF((gn.submission_time - gn.attempt_time) > 3600,
           gat.avg_time,
           gn.submission_time - gn.attempt_time) AS step_time
    FROM get_number gn
    JOIN get_avg_time gat USING (student_id)
),
get_sum_time AS (
    SELECT student_id, step_inf, SUM(step_time) AS sum_time
    FROM student_new_time
    GROUP BY student_id, step_inf
)
SELECT
    s.student_name AS Студент,
    gn.step_inf AS Шаг,
```

```
    DENSE_RANK() OVER (
        PARTITION BY gn.step_id ORDER BY gn.submission_time
    ) AS Номер_попытки,
    gn.result AS Результат,
    SEC_TO_TIME(snt.step_time) AS Время_попытки,
    ROUND(snt.step_time * 100 / gst.sum_time, 2) AS Относительное_время
FROM get_number gn
LEFT JOIN get_sum_time gst USING (step_inf)
LEFT JOIN student_new_time snt USING (get_number_id)
LEFT JOIN student s ON gn.student_id = s.student_id
ORDER BY gn.step_id, Номер_попытки;
```

Объяснение

1. Для каждой попытки считаются длительности; слишком длинные заменяются на среднее по пользователю.
2. Складывается суммарное время по шагу; относительная доля считается как часть от суммы.

Комментарии

- Все расчёты ведутся в секундах; в формат времени переводится только при выводе.

Запрос №15: группы по способу прохождения шагов

База данных

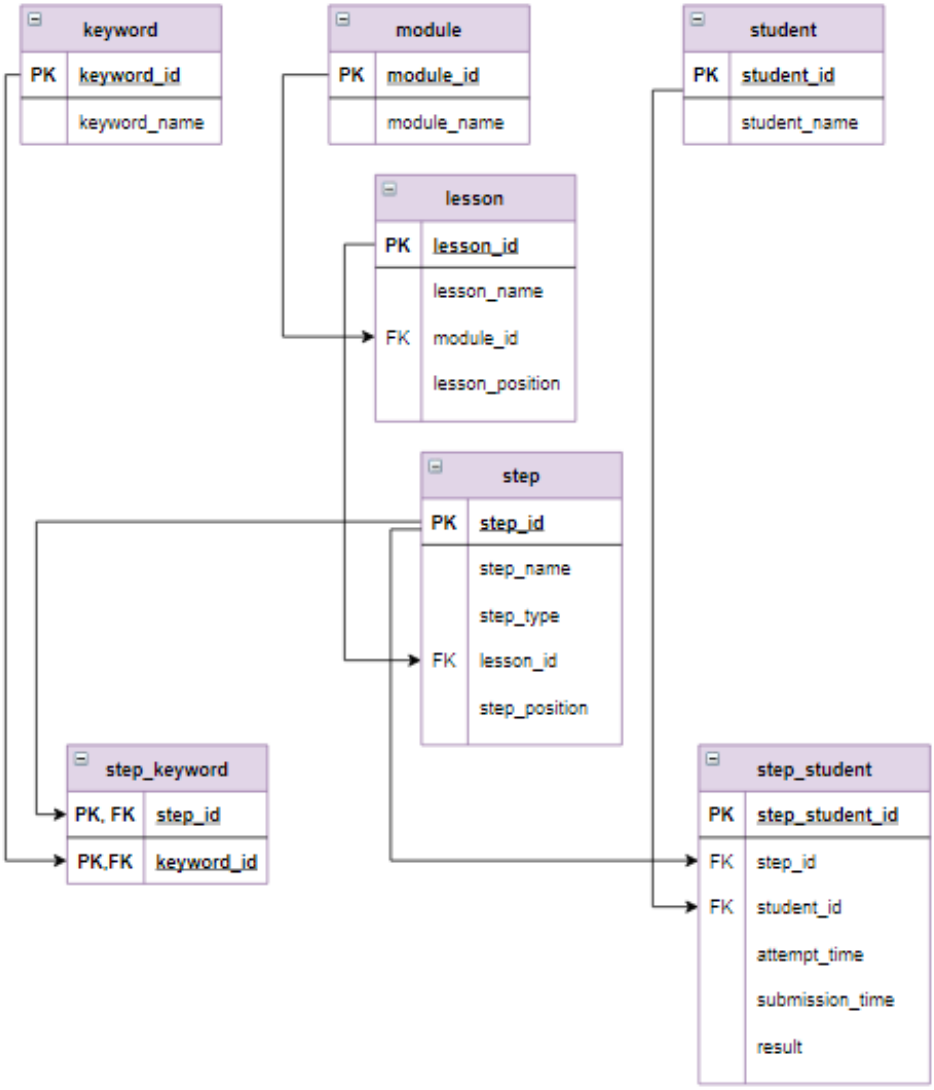

Рис. 15: Схема: `student`, `step_student`

Формулировка

Выделить группы обучающихся по поведению на шагах: I — после верной делают неверную, II — больше одной верной по шагу, III — все попытки по шагу неверные. Вывести Группа, Студент, Количество_шагов. Сортировка: группа ↑, количество ↓, имя ↑.

Решение (вариант 1)

```
WITH 2_step_students (student_id, step_id) AS (
```

```
    SELECT DISTINCT student_id, step_id
    FROM step_student
    GROUP BY student_id, step_id
    HAVING COUNT(result) > 1
),
student_steps_time_ord (student_id, step_id, result, sort_order) AS (
    SELECT student_id, step_id, result,
           ROW_NUMBER() OVER (
             PARTITION BY student_id, step_id
             ORDER BY FROM_UNIXTIME(submission_time)
           ) AS sort_order
    FROM step_student
    WHERE (student_id, step_id) IN (SELECT student_id, step_id FROM 2_step_students
   )
),
first_group (student_id, step_id, group_id) AS (
    SELECT student_id, step_id,
           CASE
             WHEN result = "correct"
              AND LEAD(result) OVER (
                PARTITION BY student_id, step_id ORDER BY sort_order
              ) <> result
             THEN "I" ELSE NULL END AS group_id
    FROM student_steps_time_ord
),
second_group (student_id, step_id, group_id) AS (
    SELECT student_id, step_id, "II" AS group_id
    FROM step_student
    WHERE result = "correct"
    GROUP BY student_id, step_id
    HAVING COUNT(result) > 1
),
third_group (student_id, step_id, group_id) AS (
    SELECT student_id, step_id, "III" AS group_id
    FROM step_student
    GROUP BY student_id, step_id
    HAVING SUM(result = "wrong") = COUNT(*)
)
SELECT group_id AS Группа, student_name AS Студент, count_step AS Количество_шагов
FROM (
    SELECT group_id, student_id, COUNT(DISTINCT step_id) AS count_step
    FROM first_group WHERE group_id = "I"
```

```
    GROUP BY student_id, group_id
    UNION ALL
    SELECT group_id, student_id, COUNT(DISTINCT step_id) AS count_step
    FROM second_group
    GROUP BY student_id, group_id
    UNION ALL
    SELECT group_id, student_id, COUNT(DISTINCT step_id) AS count_step
    FROM third_group
    GROUP BY student_id, group_id
) AS union_groups
JOIN student USING (student_id)
ORDER BY 1 ASC, 3 DESC, 2 ASC;
```

Решение (вариант 2)

```
WITH attempts AS (
  SELECT
      student_name,
      step_id,
      SUM(result = 'correct')
          OVER (PARTITION BY student_name, step_id) AS correct_count,
      (LAG(result) OVER (
           PARTITION BY student_name, step_id
           ORDER BY submission_time
       ) = 'correct' AND result = 'wrong') AS has_wrong_after_correct
  FROM student
  INNER JOIN step_student USING(student_id)
)
SELECT
    CASE
        WHEN has_wrong_after_correct = 1 THEN 'I'
        WHEN correct_count >= 2 THEN 'II'
        WHEN correct_count = 0 THEN 'III'
    END AS Группа,
    student_name AS Студент,
    COUNT(DISTINCT step_id) AS Количество_шагов
FROM attempts
GROUP BY Группа, student_name
HAVING Группа IS NOT NULL
ORDER BY Группа, Количество_шагов DESC, Студент;
```

Объяснение

1. Вариант 1 строит группы через набор CTE и оконные функции `LEAD`/`ROW_NUMBER`.
2. Вариант 2 использует одно CTE с `LAG` и оконной суммой правильных попыток.

Комментарии

- Оба решения выдают требуемые три группы с нужной сортировкой результата.