GeekBrains

Базы данных

Оконные функции, индексы, работа в графическом клиенте

SQLite 3.31.1
SQLiteStudio 3.2.1

На этом уроке

1. Научимся работать с оконными функциями.
2. Изучим, как работают индексы и когда надо их использовать.
3. Узнаем, как работать в графическом клиенте SQLiteStudio.

Оглавление

Оконные функции

Оконные функции позволяют выполнить действия над наборами строк, объединяя их по некоторому определённому признаку.

Синтаксис применения оконных функций

Для работы с оконными функциями используется ключевое слово OVER. Перед OVER указываем, какую функцию надо использовать. После OVER в скобках требуется определить правило для разделения строк выборки на части (окна).

```
'Функция' OVER ('Окно')
```

Вариант запроса с использованием функций агрегирования и группировки

Чтобы увидеть отличия в использовании оконных функций от уже знакомых нам функций агрегирования, рассмотрим примеры решения одной задачи двумя способами.

Например, нам надо для каждого ученика определить количество курсов, которые он закончил, а также минимальную оценку из всех курсов. Сначала решим эту задачу на основе знаний, полученных ранее. Используем функции агрегирования COUNT и MIN и группируем данные по пользователю:

```
SELECT
  students.id,
  students.surname,
  students.name,
  COUNT(stream_id) AS courses_passed,
  MIN(grade) AS min_grade
  FROM students
    LEFT JOIN grades
      ON grades.student_id = students.id
  GROUP BY students.id
  ORDER BY students.surname, students.name;
```

Рассмотрим подробнее, как работает это запрос. К таблице учеников присоединяем таблицу оценок посредством левого внешнего объединения. Это сделано, чтобы в отчёт попали и те ученики, которые пока не закончили ни одного курса.

В качестве условия объединения используем значение идентификатора ученика. Далее группируем данные по значению идентификатора пользователя, так как нам надо получить аналитику в разрезе учеников. В части запроса SELECT определим, что у нас попадает в отчёт — фамилия и имя ученика, количество пройденных курсов и минимальная оценка по всем курсам. Важно, что для определения количества пройденных курсов мы считаем идентификаторы потоков COUNT(stream_id). Если считать строки COUNT(*), то получим неверный результат для учеников, которые пока не завершили ни одного курса. Выполним запрос и проверим результат:

```
sqlite> SELECT
```

```
   ...>   students.id,
   ...>   students.surname,
   ...>   students.name,
   ...>   COUNT(stream_id) AS courses_passed,
   ...>   MIN(grade) AS min_grade
   ...>   FROM students
   ...>     LEFT JOIN grades
   ...>       ON grades.student_id = students.id
   ...>   GROUP BY students.id
   ...>   ORDER BY students.surname, students.name;
id          surname     name        courses_passed  min_grade
----------  ----------  ----------  --------------  ----------
3           Васильева   Ирина       0
1           Иванов      Игорь       3               4.9
2           Павлова     Анастасия   1               5
sqlite>
```

Вариант запроса на функциях агрегирования, используемых в качестве оконных

Теперь решим эту задачу с использованием оконных функций.

```
SELECT DISTINCT
  students.id,
  students.surname,
  students.name,
  COUNT(stream_id) OVER(PARTITION BY students.id) AS courses_passed,
  MIN(grade) OVER(PARTITION BY students.id) AS min_grade
  FROM students
    LEFT JOIN grades
      ON grades.student_id = students.id
  ORDER BY students.surname, students.name;
```

Если выполним этот запрос, то получим результат, полностью аналогичный предыдущему варианту с группировкой:

```
sqlite> SELECT DISTINCT
   ...>   students.id,
   ...>   students.surname,
   ...>   students.name,
   ...>   COUNT(stream_id) OVER(PARTITION BY students.id) AS courses_passed,
   ...>   MIN(grade) OVER(PARTITION BY students.id) AS min_grade
   ...>   FROM students
   ...>     LEFT JOIN grades
   ...>       ON grades.student_id = students.id
   ...>   ORDER BY students.surname, students.name;
id          surname     name        courses_passed  min_grade
----------  ----------  ----------  --------------  ----------
```

```
3           Васильева   Ирина       0
1           Иванов      Игорь       3               4.9
2           Павлова     Анастасия   1               5
sqlite>
```

Проанализируем, чем отличаются эти два варианта. В обоих случаях применяются уже знакомые нам функции COUNT и MIN, но в первом запросе они используются в качестве функций агрегирования, а во втором — как оконные функции. Важно, что во втором запросе нет группировки, вместо этого для каждой функции определено собственное правило (окно), как надо делить строки данных перед применением функции.

Выражение OVER(PARTITION BY students.id) определяет разделение данных на отдельные наборы строк (окна) по значению идентификатора ученика. Другими словами, для каждого значения идентификатора ученика определяется отдельное окно, к строкам которого затем применится функция, заданная перед ключевым словом OVER.

Другое важное отличие решения на оконных функциях — применение DISTINCT. DISTINCT используется, если требуется вывести только уникальные комбинации значений, в нашем случае — только уникальные комбинации данных по ученикам. Чтобы понять, почему это необходимо, разберём вывод запроса без DISTINCT:

```
sqlite> SELECT
   ...>   students.id,
   ...>   students.surname,
   ...>   students.name,
   ...>   COUNT(stream_id) OVER(PARTITION BY students.id) AS courses_passed,
   ...>   MIN(grade) OVER(PARTITION BY students.id) AS min_grade
   ...>   FROM students
   ...>     LEFT JOIN grades
   ...>       ON grades.student_id = students.id
   ...>   ORDER BY students.surname, students.name;
id          surname     name        courses_passed  min_grade
----------  ----------  ----------  --------------  ----------
3           Васильева   Ирина       0
1           Иванов      Игорь       3               4.9
1           Иванов      Игорь       3               4.9
1           Иванов      Игорь       3               4.9
2           Павлова     Анастасия   1               5
sqlite>
```

Без DISTINCT мы получили в результирующей выборке 5 строк. Разберёмся, почему именно столько, и посмотрим данные таблицы оценок.

```
sqlite> SELECT * FROM grades;
student_id  stream_id   grade
```

```
----------  ----------  -------------
1           1           4.9
2           1           5
1           3           5
1           2           4.9
sqlite>
```

В таблице оценок четыре записи. Для каждой записи запрос с оконными функциями выводит запрошенные данные — три строки для ученика с идентификатором 1 и одну строку для ученика с идентификатором 2. Чтобы этот момент стал понятнее, добавим в вывод идентификатор потока:

```
sqlite> SELECT
   ...>   grades.stream_id,
   ...>   students.id,
   ...>   students.surname,
   ...>   students.name,
   ...>   COUNT(stream_id) OVER(PARTITION BY students.id) AS courses_passed,
   ...>   MIN(grade) OVER(PARTITION BY students.id) AS min_grade
   ...>   FROM students
   ...>     LEFT JOIN grades
   ...>       ON grades.student_id = students.id
   ...>   ORDER BY students.surname, students.name;
stream_id   id          surname     name        courses_passed  min_grade
----------  ----------  ----------  ----------  --------------  ----------
            3           Васильева   Ирина       0
1           1           Иванов      Игорь       3               4.9
2           1           Иванов      Игорь       3               4.9
3           1           Иванов      Игорь       3               4.9
1           2           Павлова     Анастасия   1               5
sqlite>
```

Пятая строка, которая относится к ученице Ирине Васильевой, получается в результате использования внешнего объединения с таблицей students. У этой ученицы нет завершённых потоков.

Для получения искомой информации дублирующиеся данные не нужны, в нашем случае это данные ученика Игоря Иванова, поэтому очень часто в запросах с использованием оконных функций применяется DISTINCT.

В выводе последнего запроса мы также видим окна, по которым производятся вычисления оконными функциями. Окна у нас определяются значением идентификатора ученика id, то есть одна строка с id равным 3 — первое окно, три строки с id равным 1 — второе окно, одна строка с id равным 2 — третье окно.

Вернём DISTINCT и вынесем определение окна посредством ключевого слова WINDOW. Финальная версия запроса выглядит следующим образом:

```
SELECT DISTINCT
  students.id,
  students.surname,
  students.name,
  COUNT(stream_id) OVER(w_students) AS courses_passed,
  MIN(grade) OVER(w_students) AS min_grade
  FROM students
    LEFT JOIN grades
      ON grades.student_id = students.id
  WINDOW w_students AS (PARTITION BY students.id)
  ORDER BY students.surname, students.name;
```

Встроенные оконные функции SQLite

В качестве оконных функций используются не только функции агрегирования, которые применяются в качестве оконных, но и функции, применяемые только на окнах. Рассмотрим применение оконных функций ROW_NUMBER и FIRST_VALUE.

Функция ROW_NUMBER возвращает порядковый номер строки в пределах окна, а FIRST_VALUE возвращает первое значение столбца в пределах окна.

Изменим немного запрос, рассмотренный выше:

```
SELECT DISTINCT
  ROW_NUMBER() OVER(w_students) AS row_number,
  Students.id AS student_id,
  students.surname,
  students.name,
  grade,
  FIRST_VALUE(grade) OVER(w_students) AS first_value
  FROM students
    LEFT JOIN grades
      ON grades.student_id = students.id
  WINDOW w_students AS (PARTITION BY students.id)
  ORDER BY students.surname, students.name;
```

Выполним запрос и оценим результаты:

```
sqlite> SELECT DISTINCT
   ...>   ROW_NUMBER() OVER(w_students) AS row_number,
   ...>   Students.id AS student_id,
   ...>   students.surname,
   ...>   students.name,
   ...>   grade,
   ...>   FIRST_VALUE(grade) OVER(w_students) AS first_value
   ...>   FROM students
   ...>     LEFT JOIN grades
```

```
   ...>       ON grades.student_id = students.id
   ...>   WINDOW w_students AS (PARTITION BY students.id)
   ...>   ORDER BY students.surname, students.name;
row_number  student_id  surname     name        grade           first_value
----------  ----------  ----------  ----------  -------------   -----------
1           3           Васильева   Ирина
1           1           Иванов      Игорь       4.9             4.9
2           1           Иванов      Игорь       4.9             4.9
3           1           Иванов      Игорь       5               4.9
1           2           Павлова     Анастасия   5               5
sqlite>
```

Работу оконных функций ROW_NUMBER и FIRST_VALUE можно хорошо увидеть на данных второго окна ученика с идентификатором равным 1 (выделено красным). В столбце row_number для этого окна используется порядковая нумерация строк от 1 до 3, а в столбце first_value проставлено первое значение оценки — 4.9.

Применение оконных функций существенно расширяет возможности работы с данными, иногда в литературе оконные функции называются аналитическими.

Индексы

Рассмотрим, как происходит выборка данных на примере простого запроса — используем базу данных учеников.

```
sqlite> .header on
sqlite> .mode column
sqlite> SELECT * FROM students WHERE surname = 'Иванов';
id          surname     name
----------  ----------  ----------
1           Иванов      Игорь
sqlite>
```

Чтобы вернуть найденную строку как результат, СУБД должна найти файл, где хранятся данные таблицы учеников, прочитать данные с носителя, загрузить их в оперативную память и затем перебирать каждую строку таблицы, проверяя совпадение по условию поиска — сравнивать фамилию каждого ученика с заданной. После перебора всех строк те данные, которые соответствуют условию поиска, вернутся клиенту.

Как видим, такой метод перебора прекрасно работает в нашем случае, но представим, что требуется выполнить такой запрос в базе данных, содержащей миллионы записей в таблице учеников. Задача станет уже далеко не такой простой, ведь СУБД понадобится сначала загрузить большой объём

данных с носителя в оперативную память, а затем проверить каждую из миллионов строк методом простого перебора.

Мы можем предположить, и наши предположения будут верными, что такой подход к поиску данных в реальных базах данных не может быть быстрым и эффективным. Пользователи будут получать значительные задержки во времени между выполнением запроса и получением результата. Но по опыту работы с различными приложениями мы знаем, что, как правило, отклик на наши действия приходят достаточно быстро. Очевидно, что в реальных системах применяется какой-то другой способ поиска данных, а не просто перебираются все строки таблицы.

Представим, что данные об учениках хранятся в файле базы данных в упорядоченном виде по значению фамилии, аналогично тому, как нам покажет выборка с сортировкой:

```
sqlite> SELECT * FROM students ORDER BY surname;
id          surname     name
----------  ----------  ----------
3           Васильева   Ирина
1           Иванов      Игорь
2           Павлова     Анастасия
...
sqlite>
```

Если строки таблицы отсортированы по фамилии ученика, мы можем применить более эффективные методы поиска, например, метод двоичного поиска. Другие названия: бинарный поиск, метод деления на два.

Суть метода двоичного поиска — весь первоначальный массив данных, не забываем, что строки отсортированы по фамилии учеников, мы делим на две равные части по количеству строк, а затем сравниваем, в какую из частей попадает наше искомое значение фамилии. Это легко сделать, так как среднее значение известно.

После того как мы определим, в какой из частей следует искать подходящие строки, отбросим другую часть, а требуемую часть снова поделим на два набора данных. Продолжим деление до тех пор, пока не появятся подходящие записи. Применение такого алгоритма намного эффективнее, чем метод простого перебора всех строк.

Проблема в том, что в различных запросах потребуется искать данные и по значениям других полей, например, по имени пользователя. Нельзя хранить одни и те же данные отсортированными по нескольким столбцам, это невозможно физически. Но мы можем хранить отсортированные значения столбцов отдельно от данных таблицы. Например, сохраним отсортированные фамилии учеников. Эти данные будут выглядеть аналогично данным, которые мы получаем при выполнении выборки значений столбца с сортировкой:

```
sqlite> SELECT surname FROM students ORDER BY surname;
```

```
surname
----------
Васильева
Иванов
Павлова
...
sqlite>
```

Если в такой структуре рядом со значением столбца хранить указатель на место в файле базы данных, где находится соответствующая строка, то мы сможем применить алгоритм поиска делением на два и получить из файла таблицы только те строки, которые соответствуют искомым значениям. У нас также получится создать такие структуры для любого количества столбцов. Это и будет решением проблемы с поиском данных. Такие структуры называются индексами.

Индекс — структура базы данных, в которой хранятся отсортированные значения столбца или нескольких столбцов, с указателями на место в физическом файле, где хранится соответствующая строка таблицы.

Создание индексов

Рассмотрим команду создания индекса:

```
CREATE INDEX 'Имя индекса' ON 'Имя таблицы'('Имена столбцов');
```

Мы можем дать произвольное имя индексу, но один из распространённых вариантов — комбинация имени таблицы и имени столбца, для которых индекс создаётся, плюс добавляется суффикс _idx (сокращение от index). Рассмотрим, как создать индекс на столбец фамилии таблицы учеников.

```
CREATE INDEX students_surname_idx ON students(surname);
```

Выполним команду создания индекса:

```
sqlite> CREATE INDEX students_surname_idx ON students(surname);
sqlite>
```

Если теперь посмотреть структуру таблицы учеников, то мы также увидим созданный индекс.

```
sqlite> .schema students
CREATE TABLE students (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  surname TEXT NOT NULL,
  name TEXT NOT NULL
```

```
);
CREATE INDEX students_surname_idx ON students(surname);
sqlite>
```

После того как индекс создан, во всех запросах, где выполняется поиск или сортировка по значению фамилии ученика, выполнится поиск по индексу students_surname_idx.

Уникальные индексы

Если данные столбца не повторяются, то создаётся уникальный индекс. В этом случае в команду надо добавить ключевое слово UNIQUE. Например, имеет смысл создать уникальный индекс на столбец номера потока, входящего в таблицу потоков.

```
CREATE UNIQUE INDEX streams_number_uq ON streams(number);
```

В таком случае для имени индекса мы используем суффикс _uq (сокращение от unique).

Составные индексы

Индексы иногда создаются на несколько столбцов сразу, такие индексы называются составными. Например, если при работе с данными часто выполняется запрос с поиском и по фамилии и по имени ученика, то формируется составной индекс на столбцы фамилии и имени.

```
CREATE INDEX students_surname_name_idx ON students(surname, name);
```

Неявные индексы

В некоторых случаях СУБД SQLite создаёт индексы автоматически. Если столбец таблицы объявлен первичным ключом, внешним ключом, или на столбец определено ограничение уникальности значений, то СУБД построит индексы на такие столбцы без участия пользователя.

Когда требуется применять индексы

Если наличие индексов даёт такие существенные преимущества, то возникает вопрос — почему тогда не создавать индексы на все столбцы всех таблиц автоматически? Вероятно, в этом случае поиск по значениям всех столбцов будет максимально быстрым?

К сожалению, такой подход не будет эффективным. Дело в том, что на создание индексов, их хранение и поддержание в актуальном состоянии требуются ресурсы — процессорное время, место на носителях информации, оперативная память системы. Если создать индексы на все столбцы всех

таблиц, то возникнет ситуация, когда затраты на обслуживание индексов перевесят преимущества, которые индексы дают для поиска.

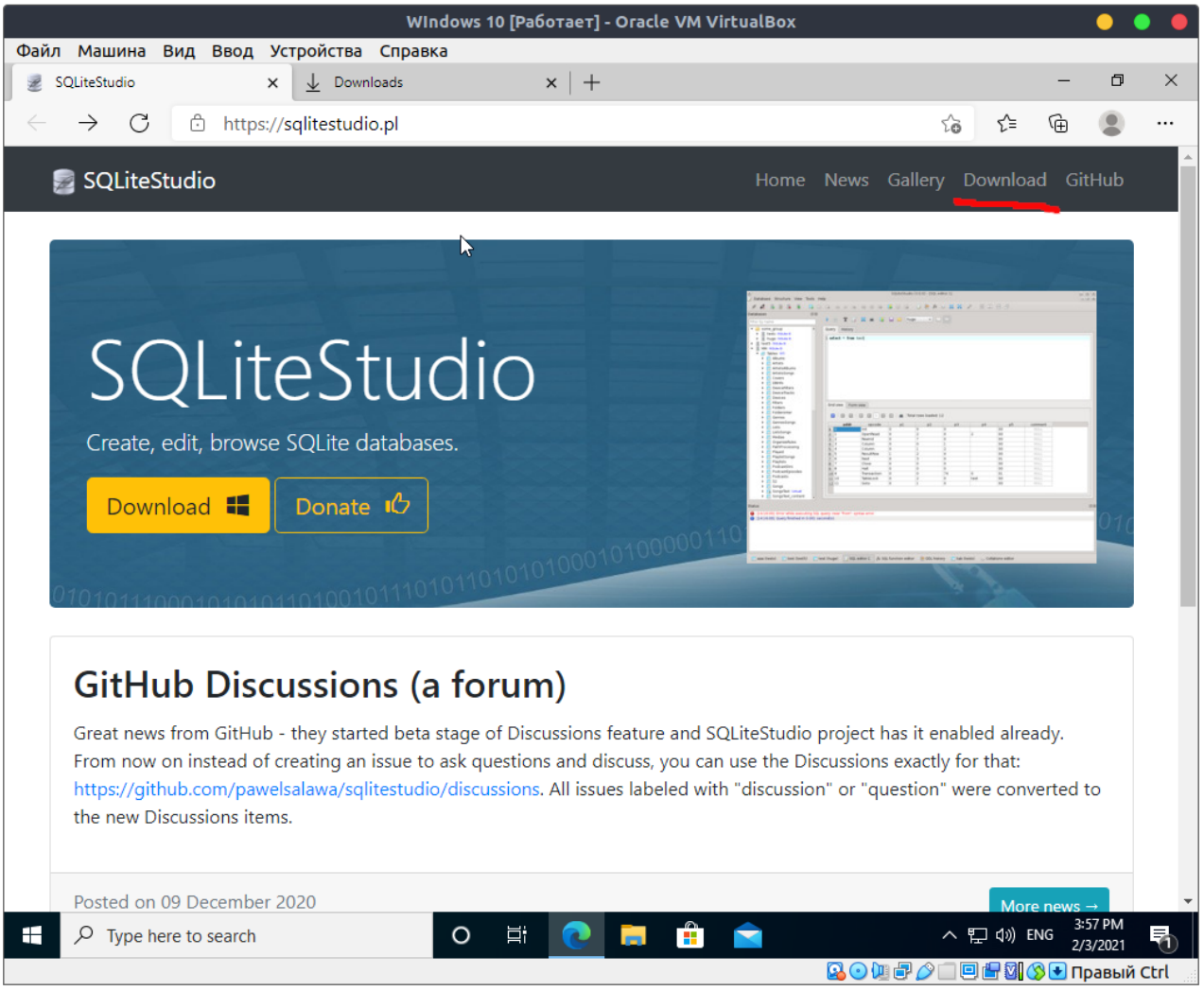
Когда надо создавать индексы? Общепринятым подходом считается создавать индексы на этапе эксплуатации системы, когда уже понятно, какие запросы выполняются наиболее часто и значительно нагружают базу данных. Для ускорения работы таких запросов на столбцы поиска и/или сортировки создаются соответствующие индексы.

Работа в графическом клиенте SQLiteStudio

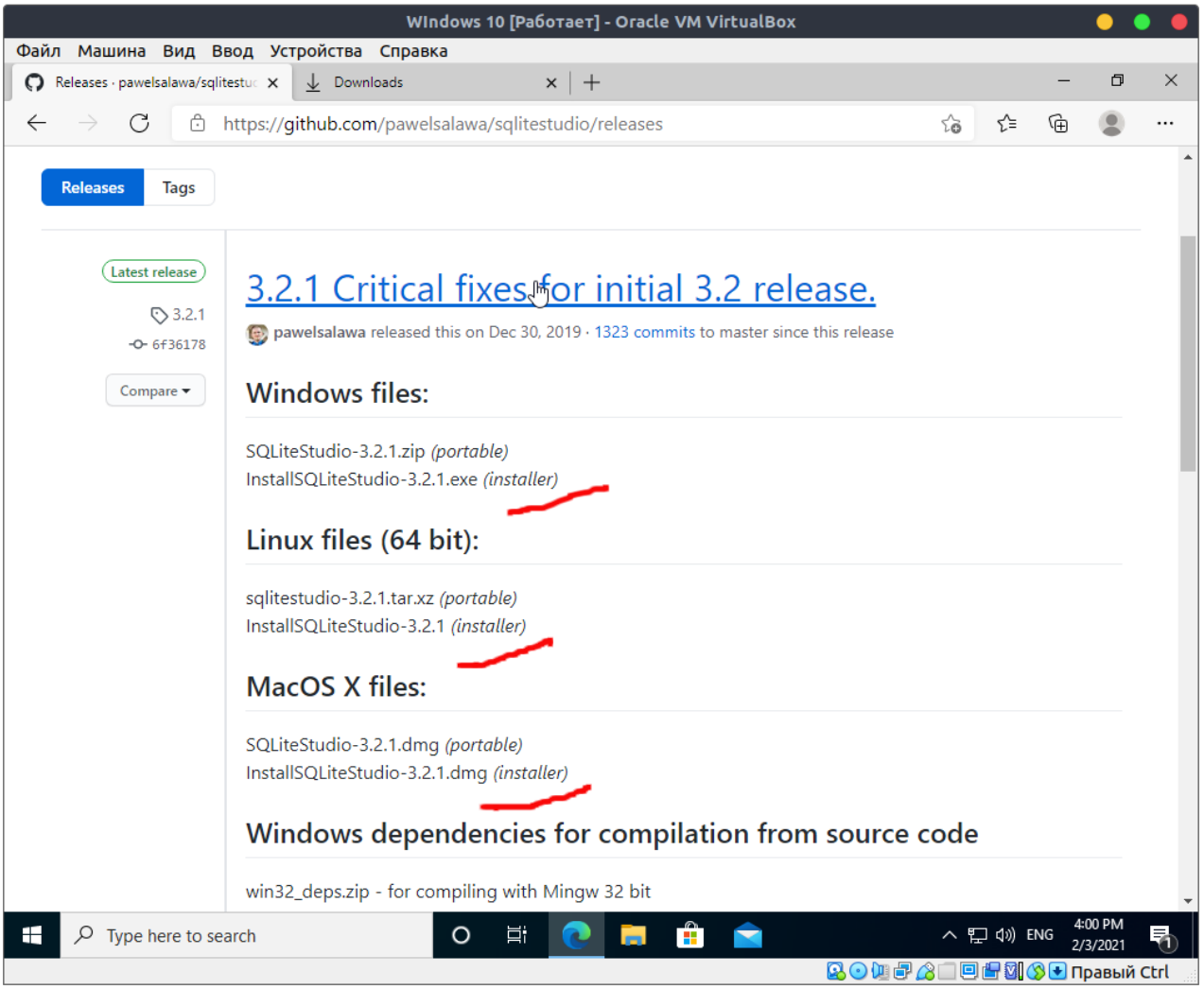
Для работы с SQLite используются различные графические программы-клиенты. Один из самых популярных решений — SQLiteStudio. Программа устанавливается с официального сайта.

Установка SQLiteStudio

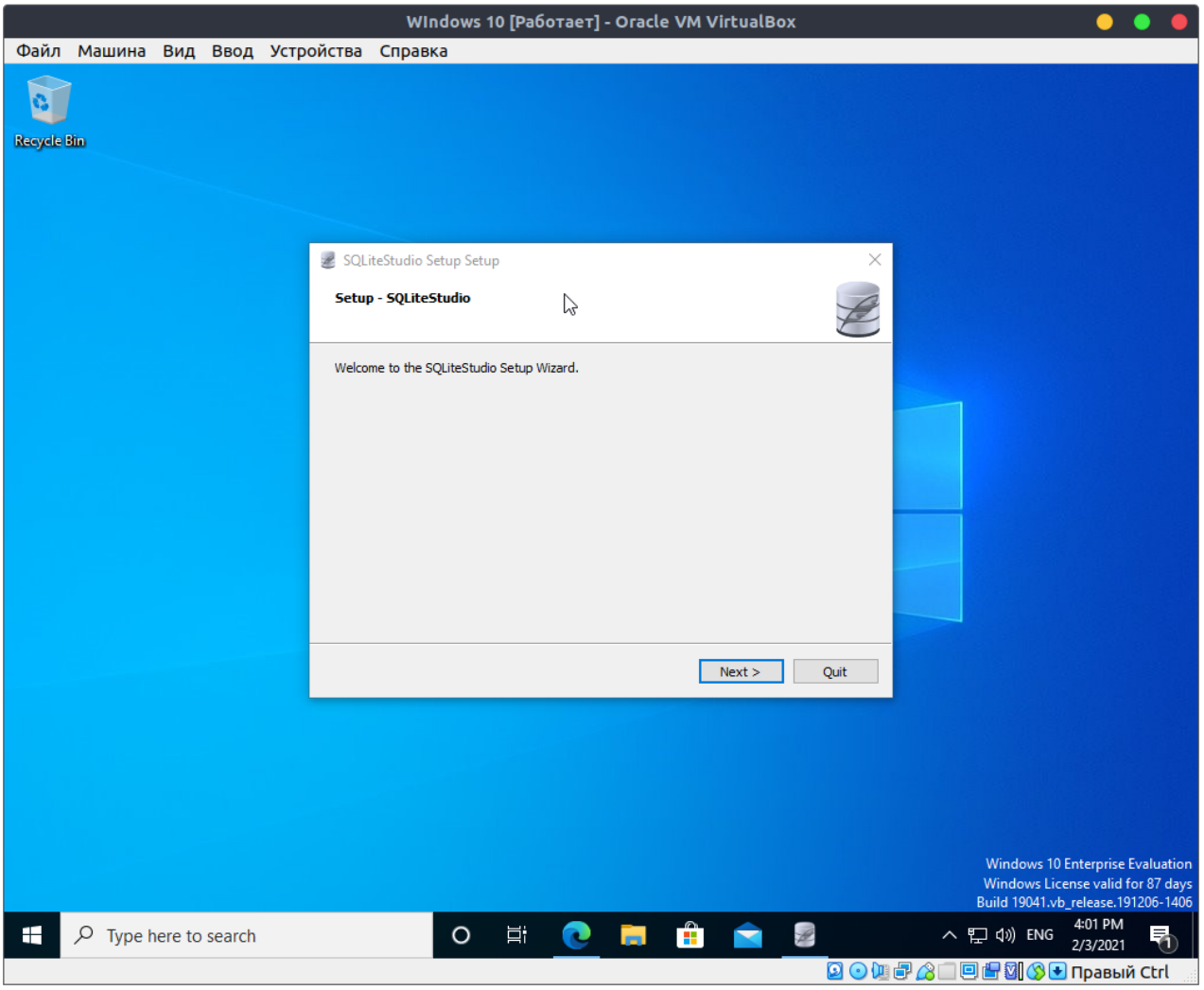
На стартовой странице проекта кликаем в правом верхнем углу Download.



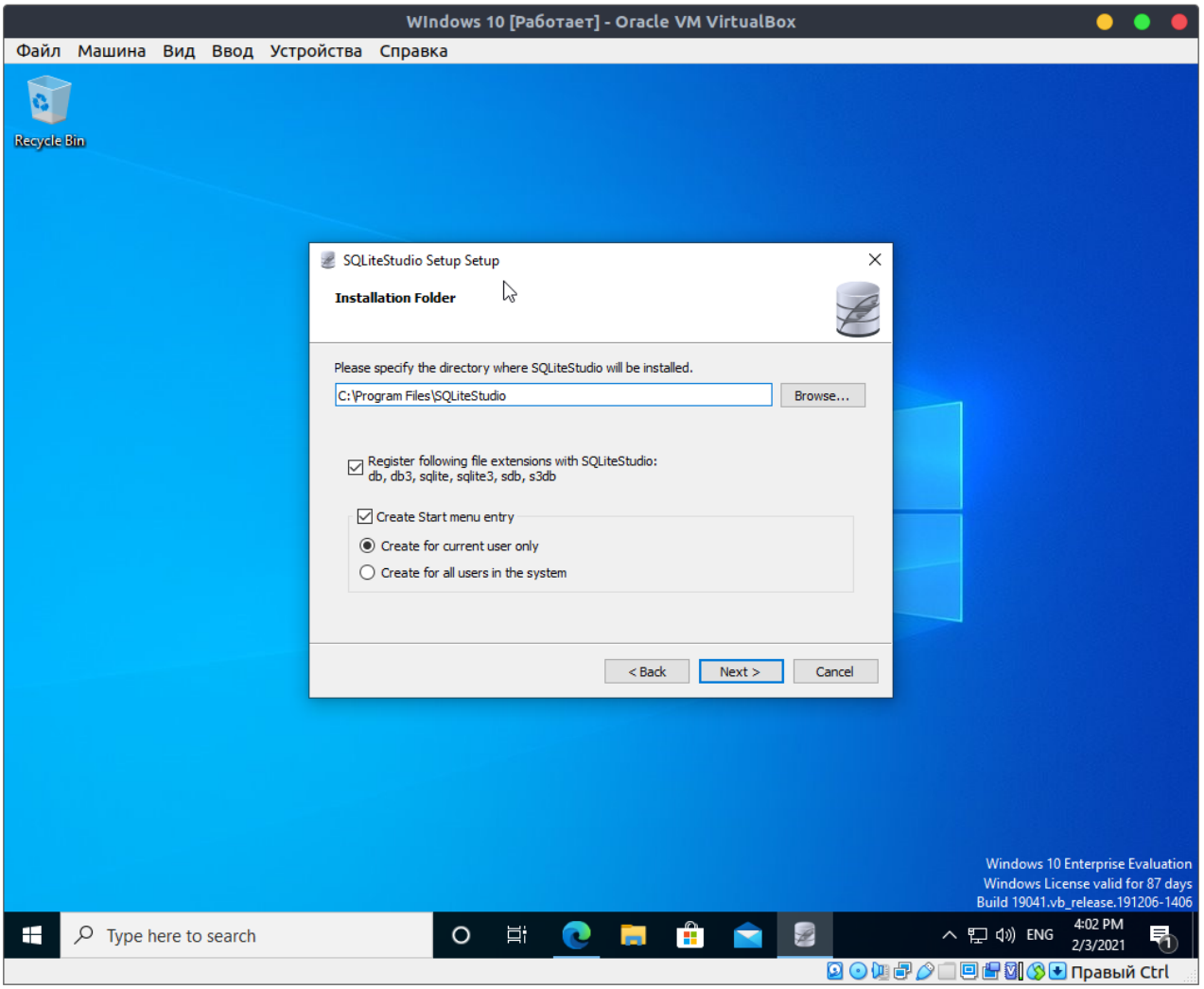
Далее на странице загрузок выбираем установщик (installer) для своей операционной системы.



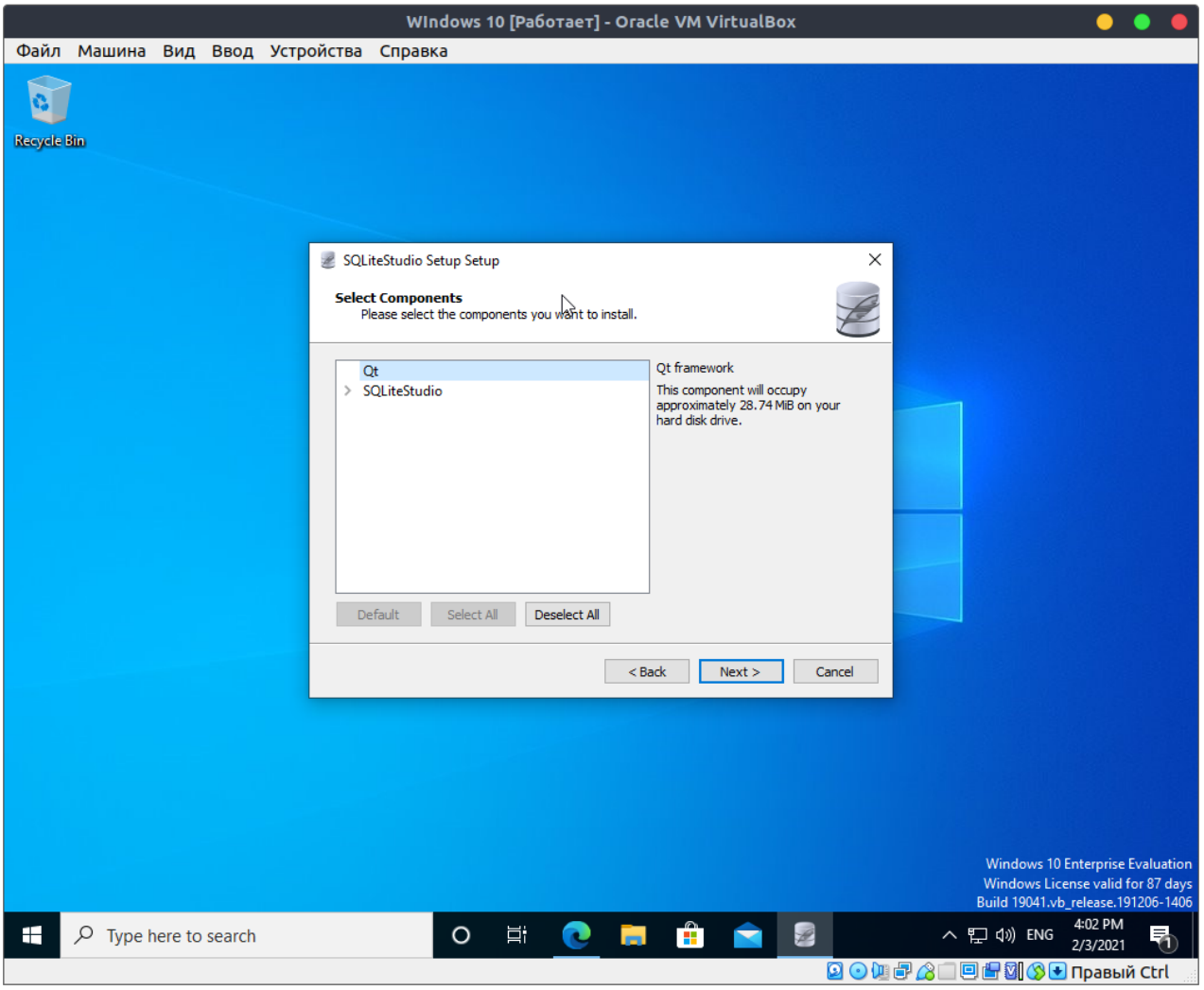
После загрузки файла запускаем его — на ОС Linux предварительно надо сделать файл установщика исполняемым. После приветствия нажимаем Next.



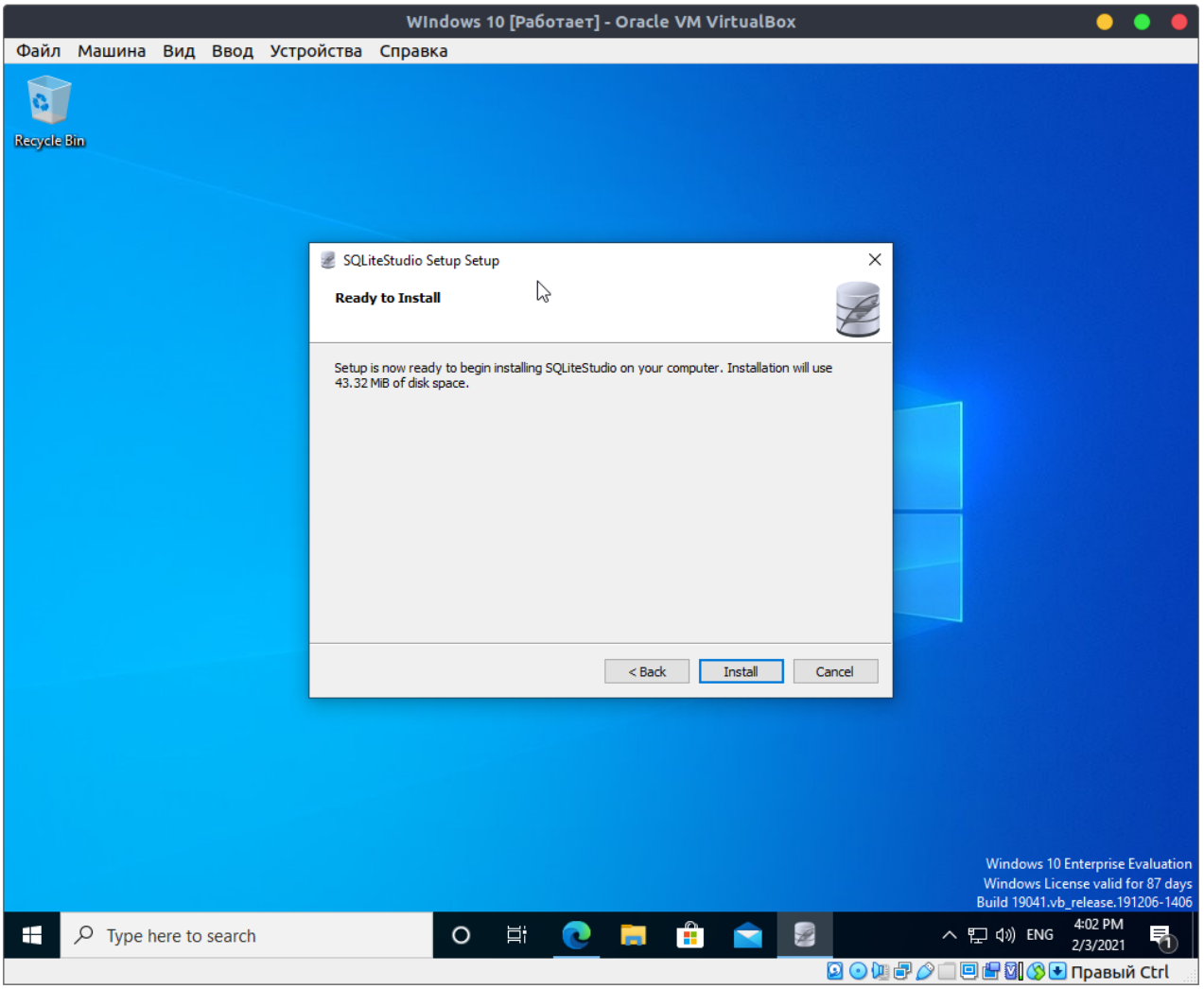
Далее выбираем путь установки и ассоциацию с типами файлов. Если менять эти значения не надо, то нажимаем Next.



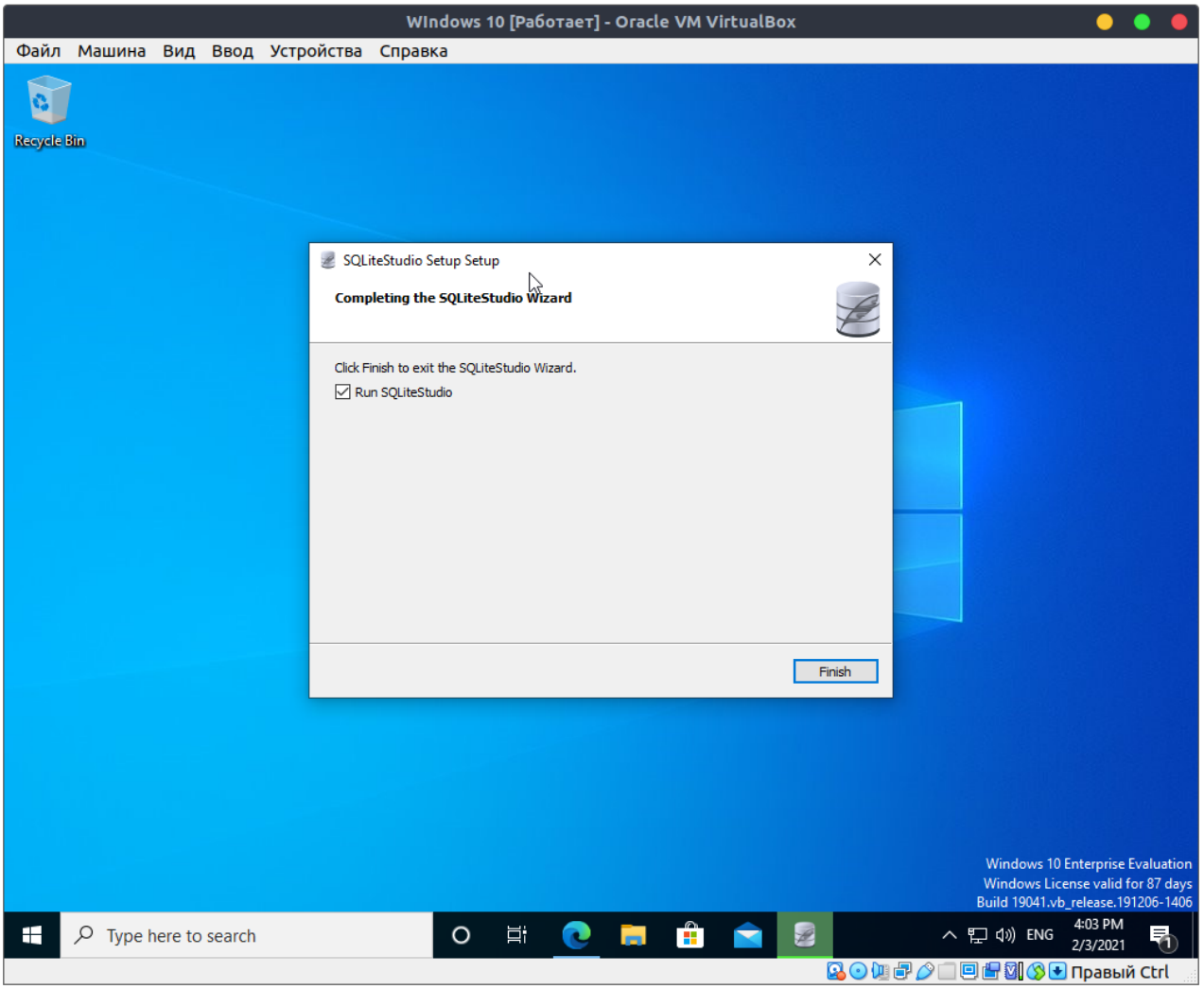
На следующем шаге выбираем требуемые компоненты или нажимаем Next, чтобы установка прошла с предопределёнными настройками.



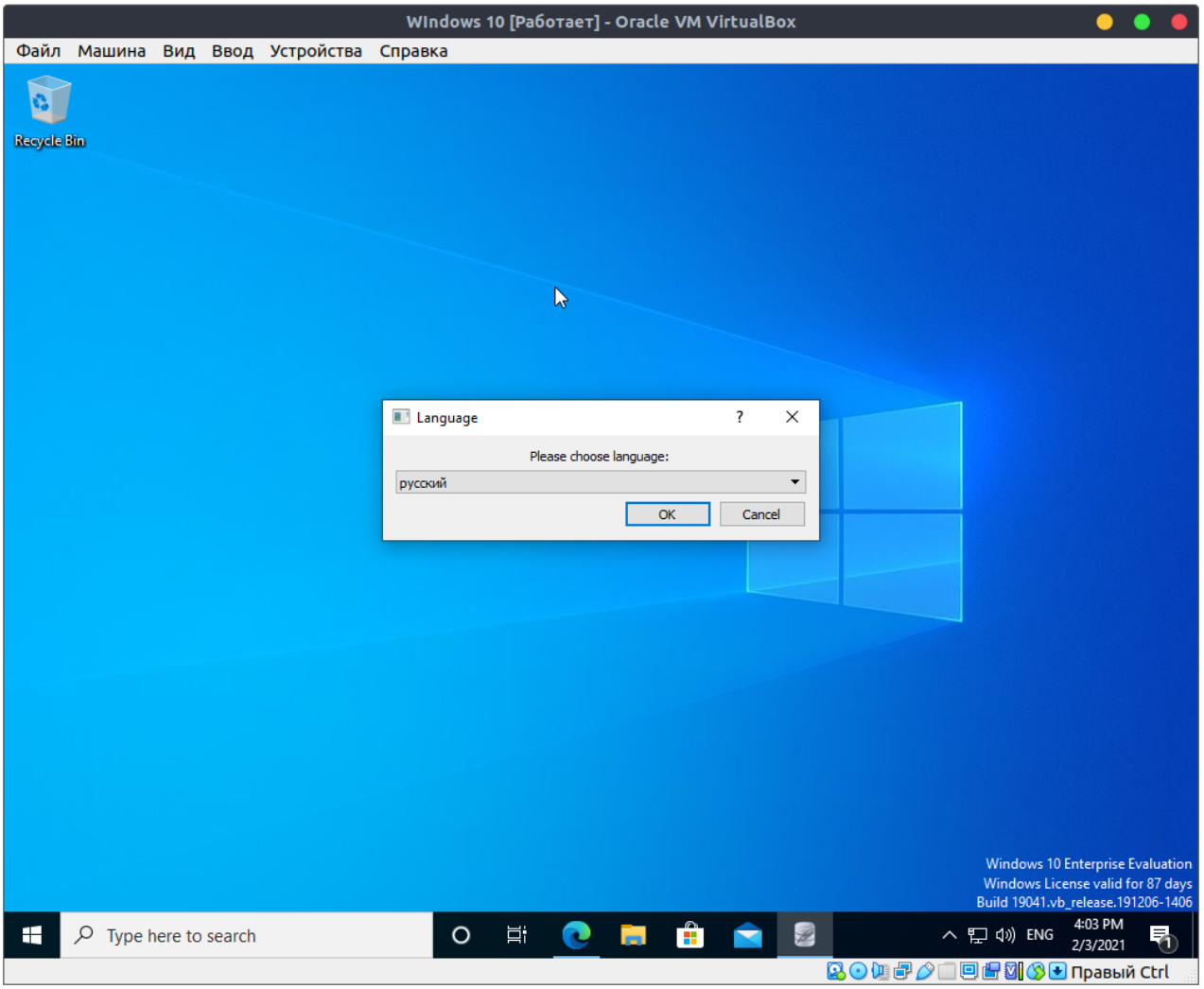
Подтверждаем установку, нажимаем Install.



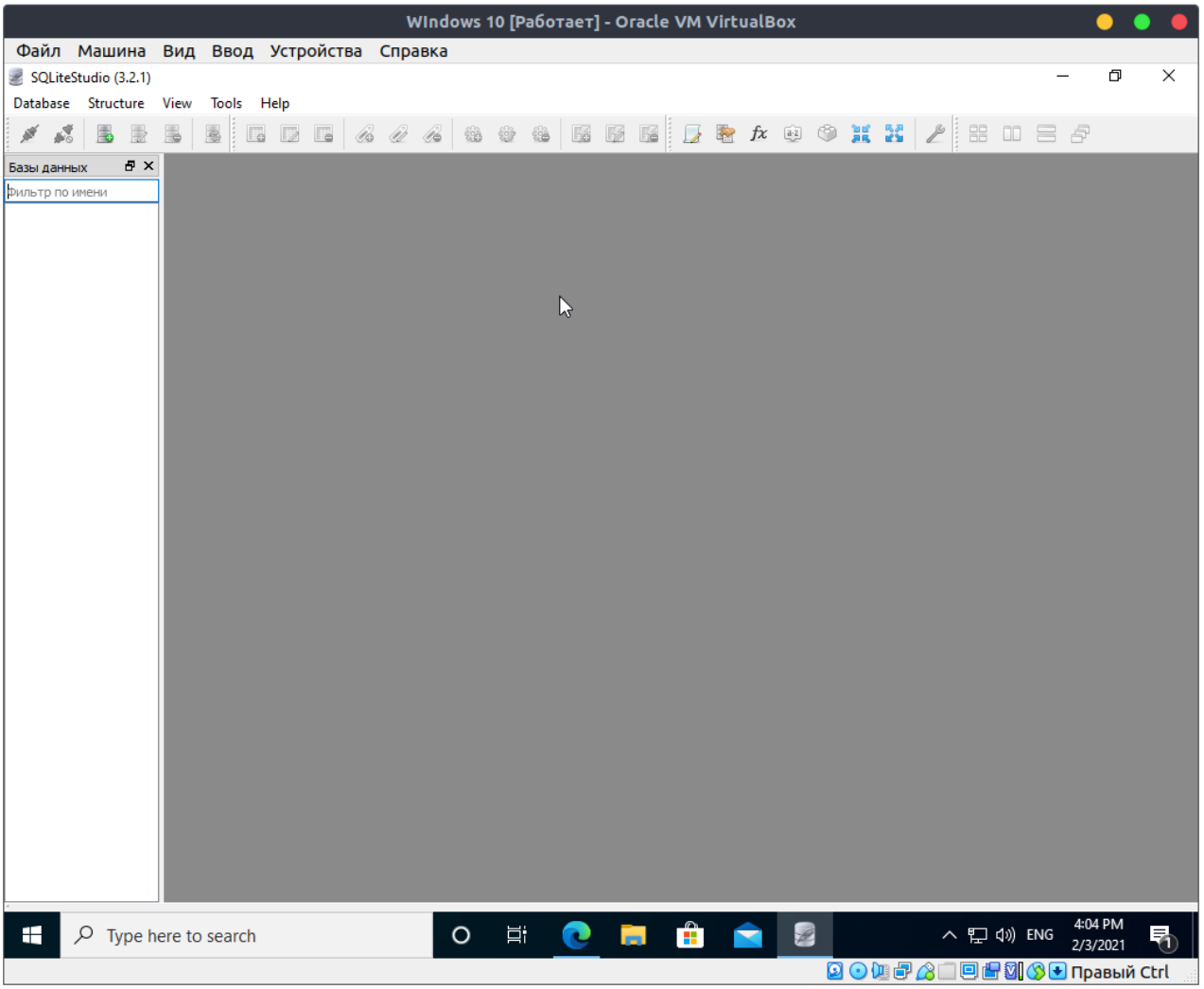
После окончания установки кликаем Finish.



Запустится программа SQLiteStudio, а затем потребуется выбрать язык интерфейса.

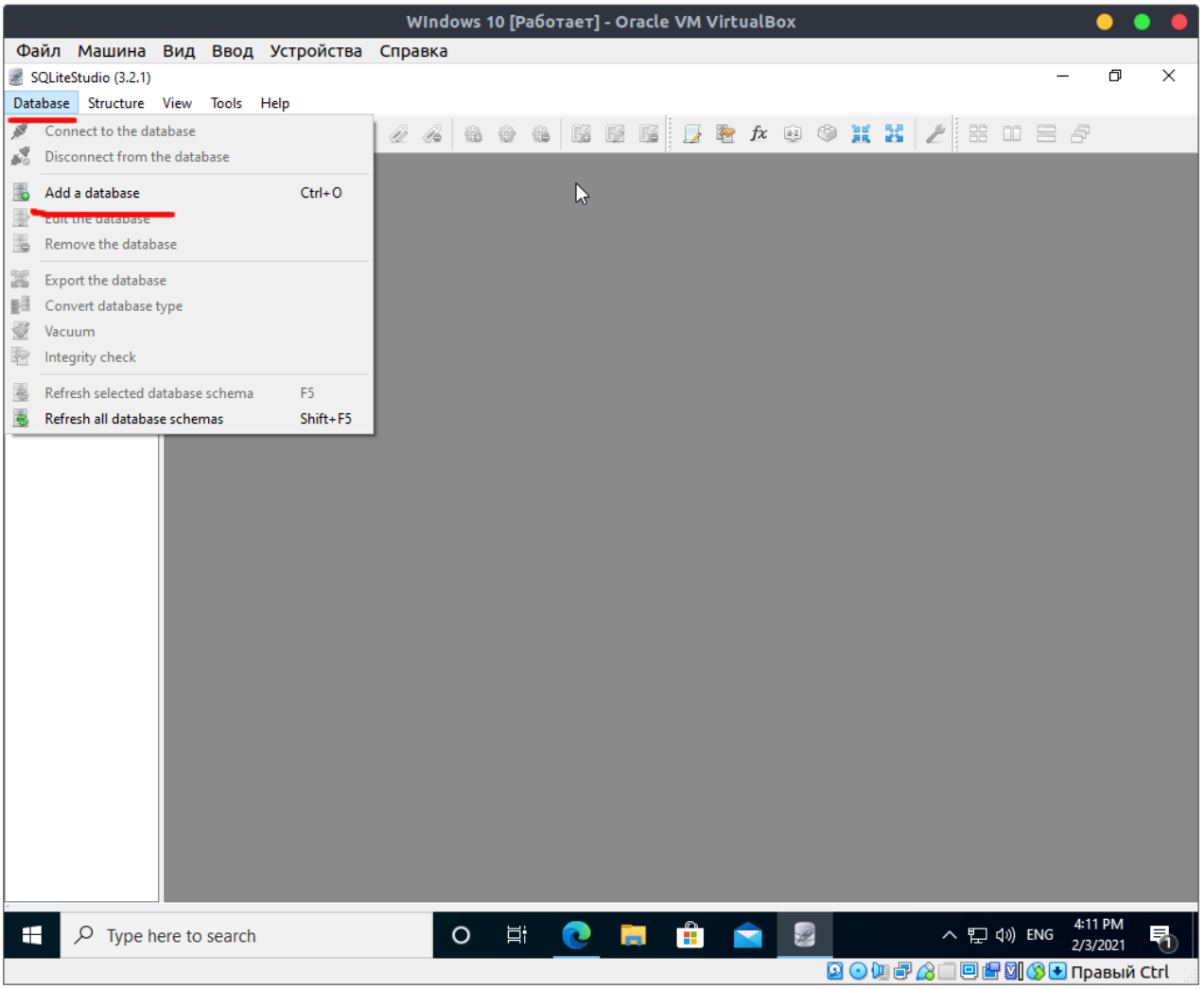


После выбора языка программа будет готова к использованию.

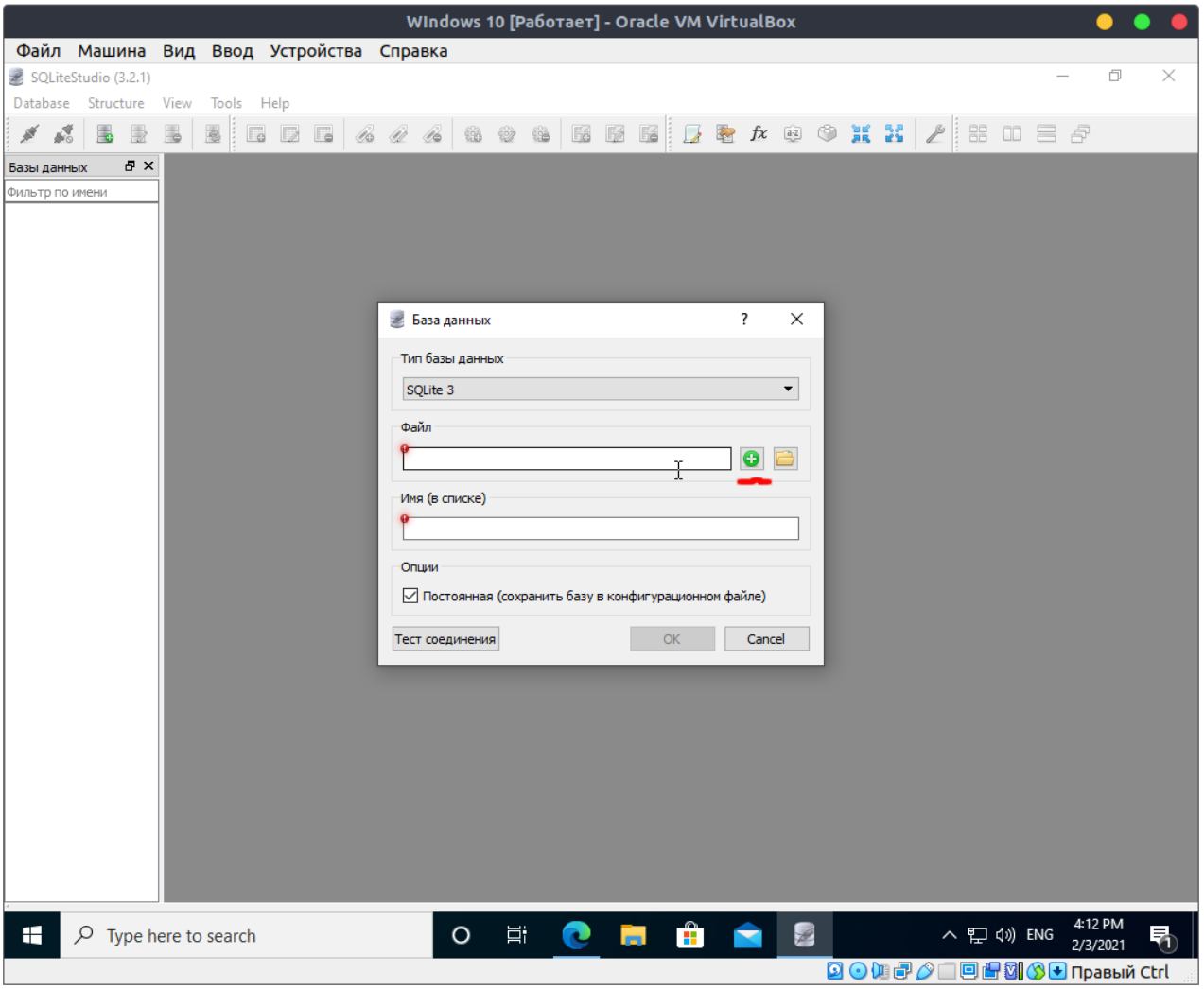


Создать базу данных

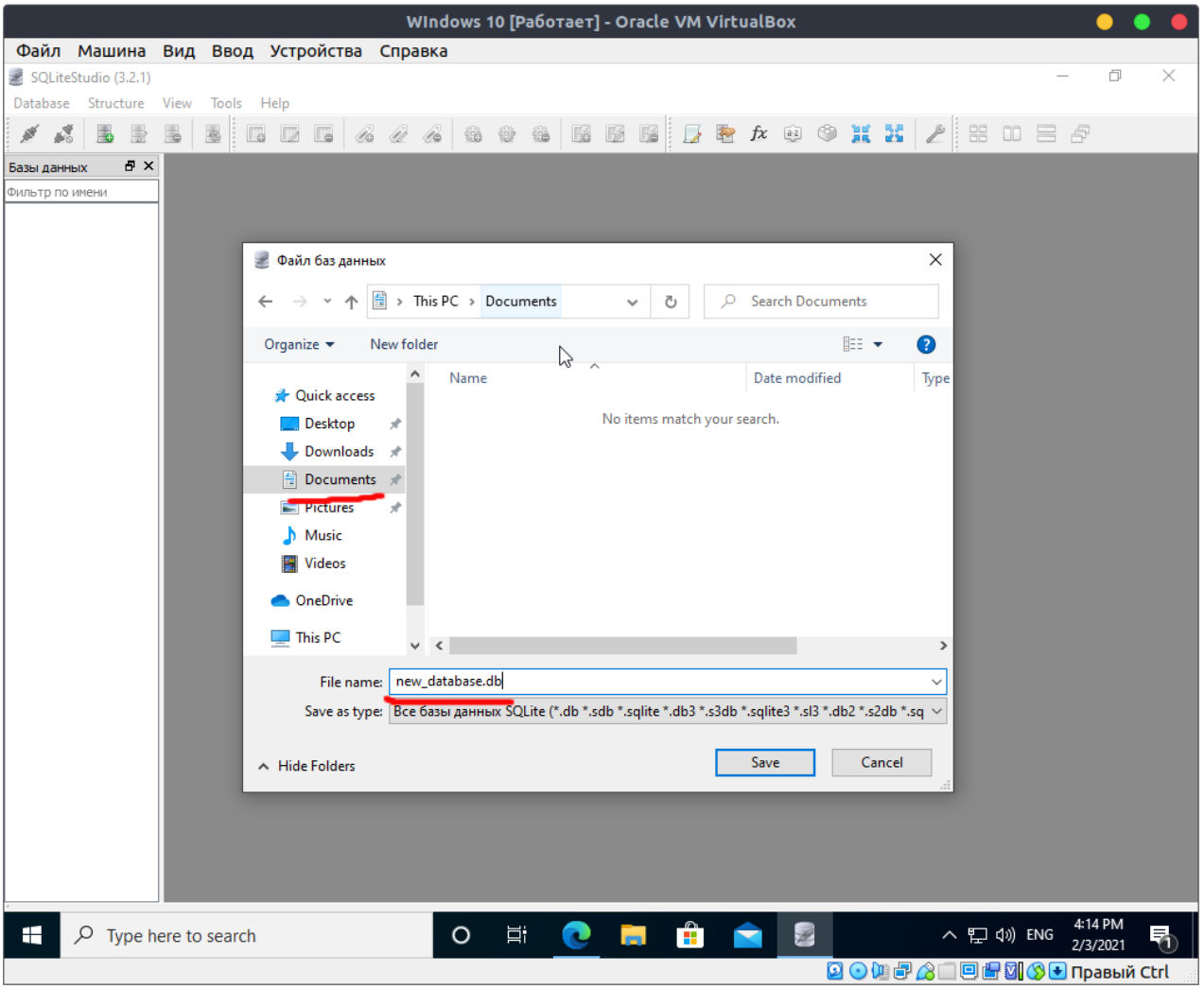
Чтобы создать новую базу данных, надо зайти в меню Database и выбрать пункт Add a database.



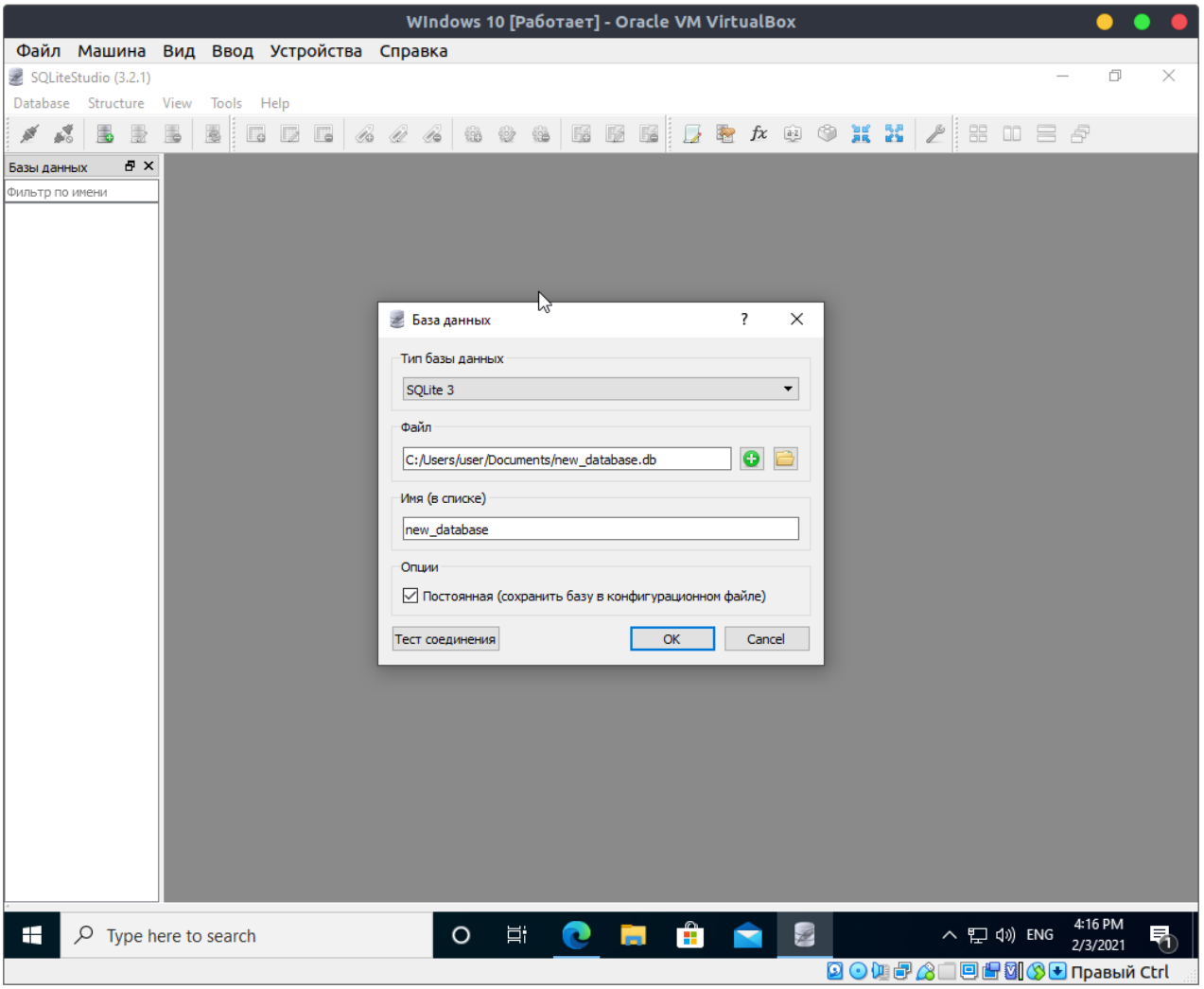
После этого потребуется выбрать имя файла базы данных и имя самой БД. Нажимаем кнопку «+» справа от поля «Файл».



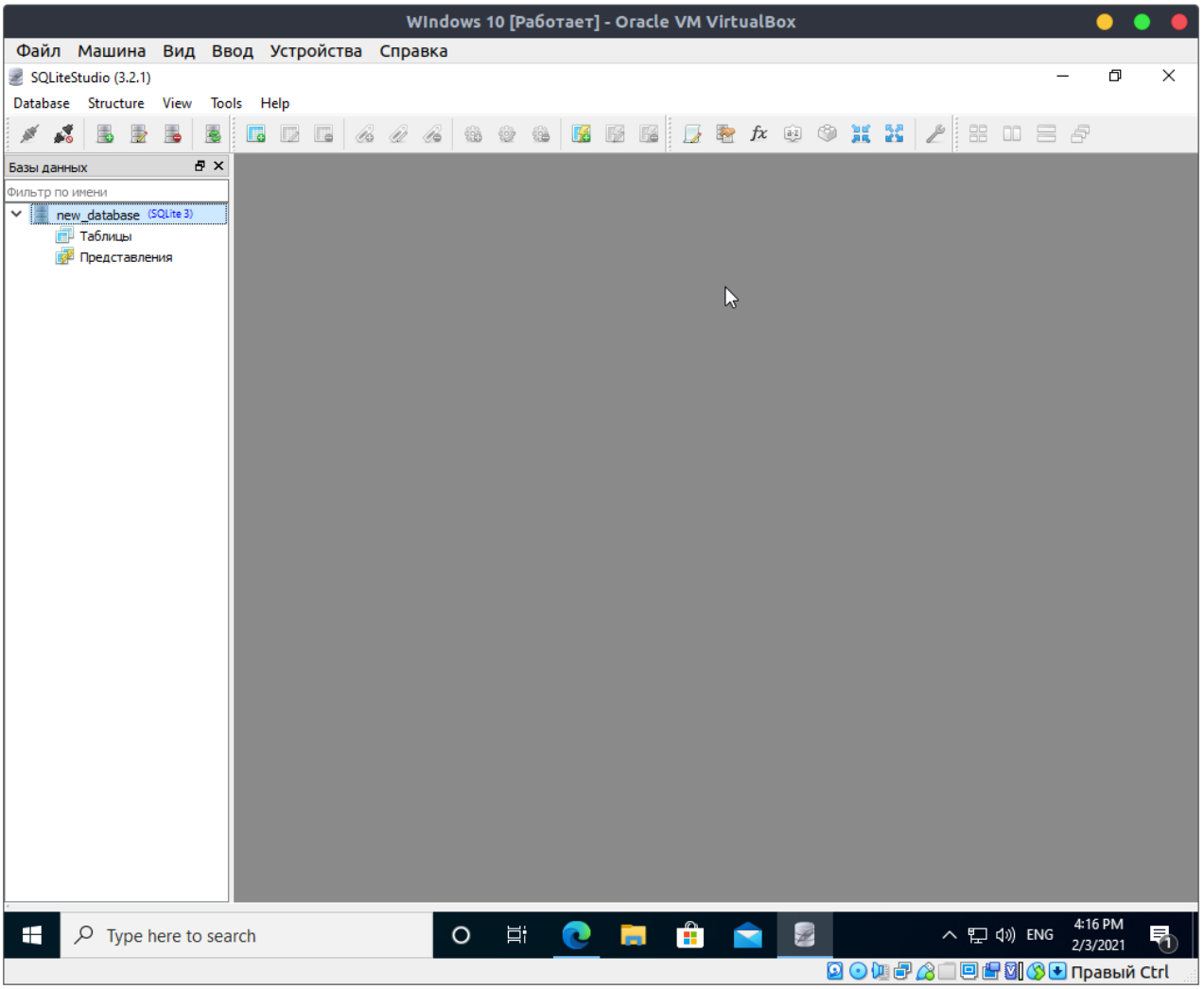
Выбираем папку, где будет храниться файл базы данных, и его имя. После ввода значений нажимаем Save.



Подтверждаем выбор в окне «Базы данных», жмём «Ок».

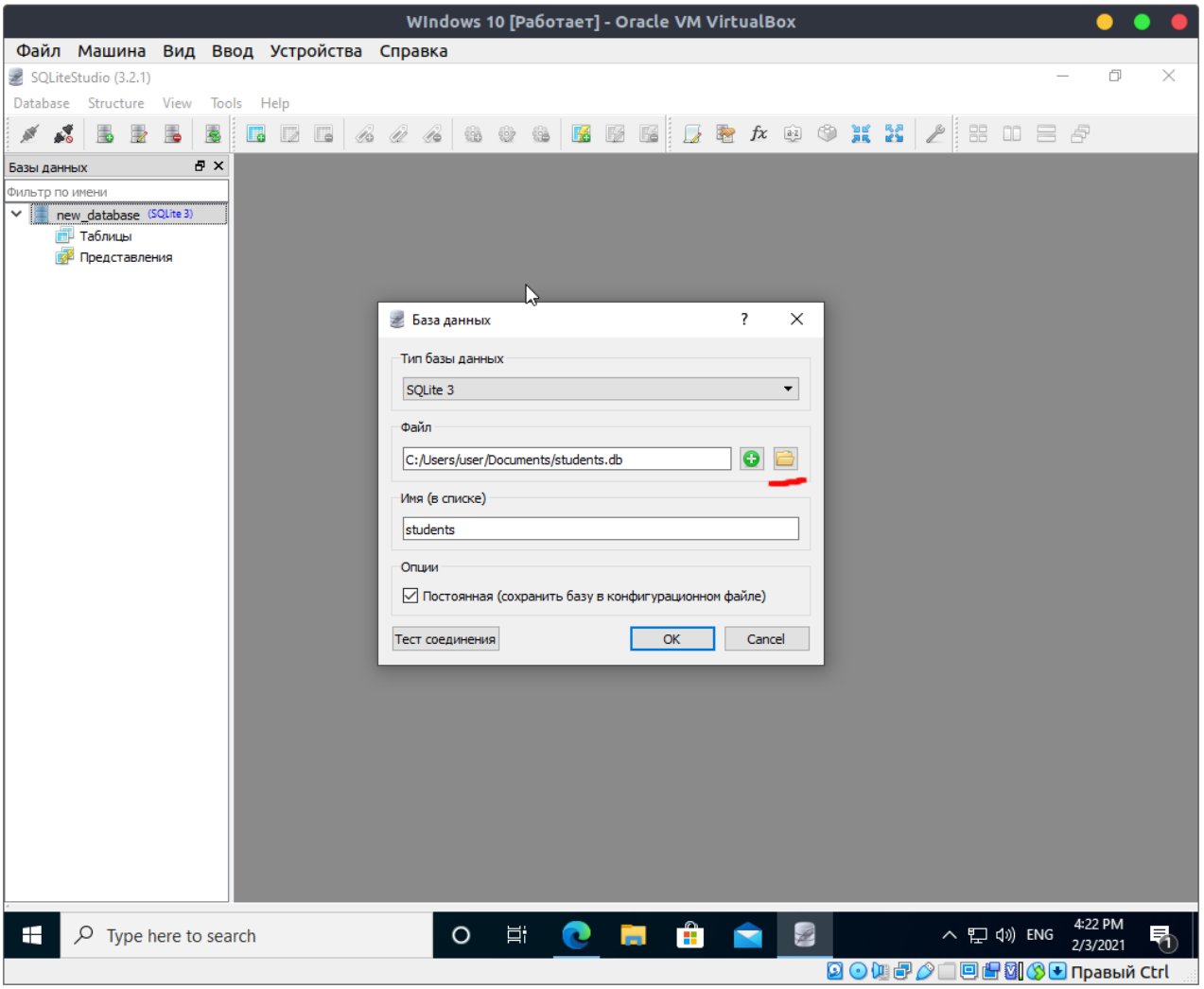


Новая база данных создана, она появилась в левой панели интерфейса.

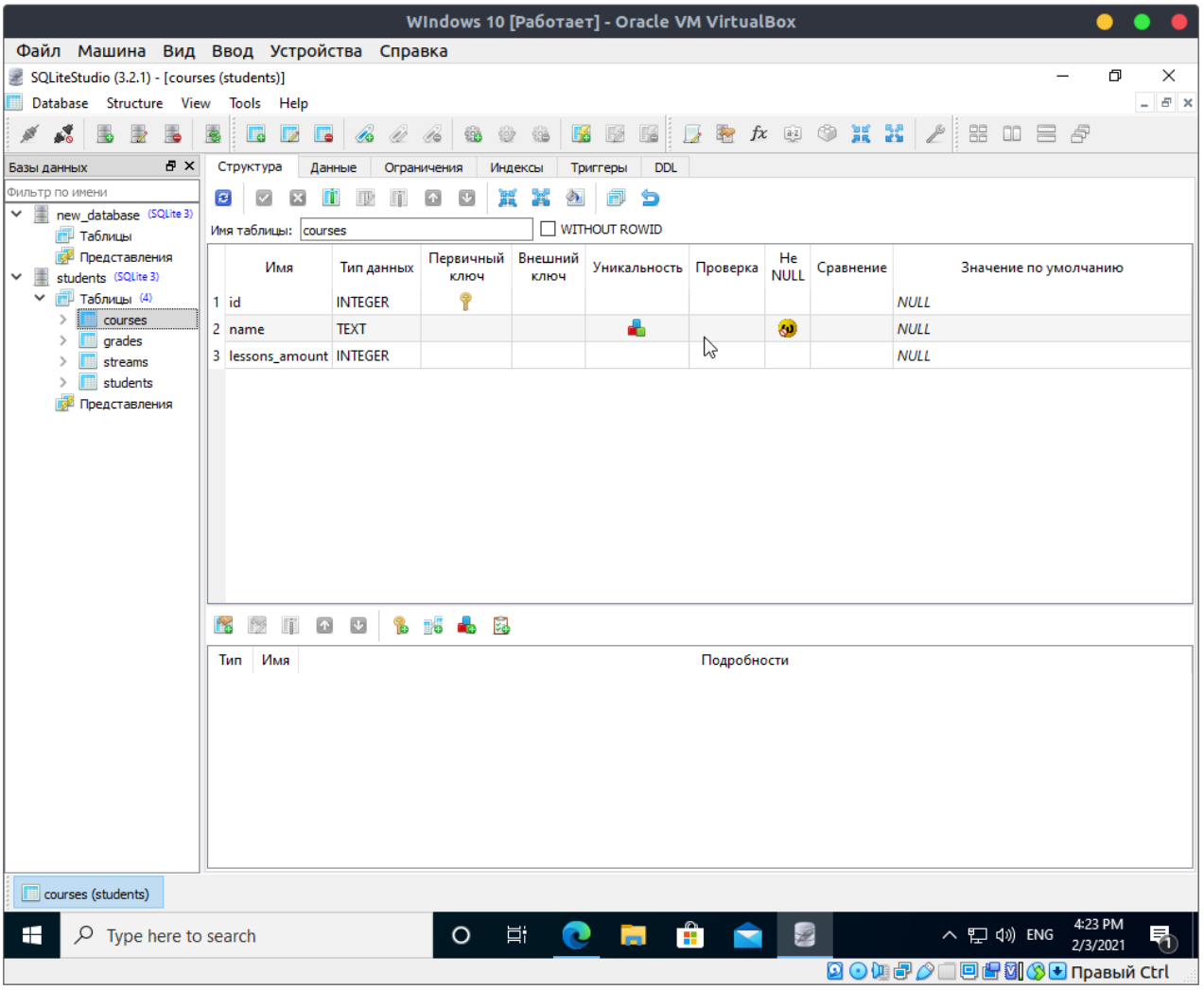


Открыть существующую базу данных

Чтобы открыть существующую базу данных, надо также зайти в меню Database, выбрать Add a database, аналогично тому, как мы делали выше, и в окне файла БД выбрать подходящий файл с данными.

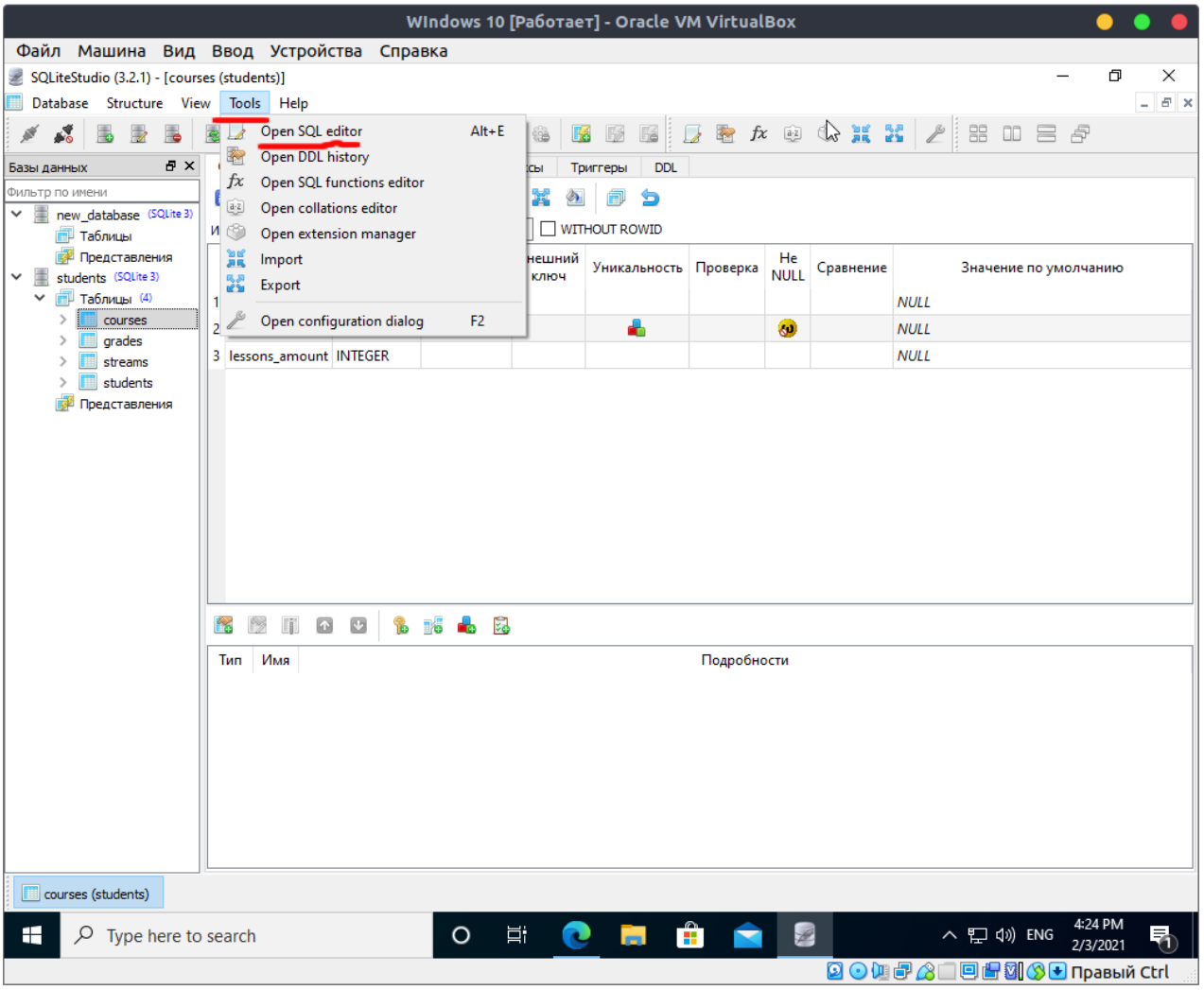


После подтверждения база данных появится в левом меню приложения.

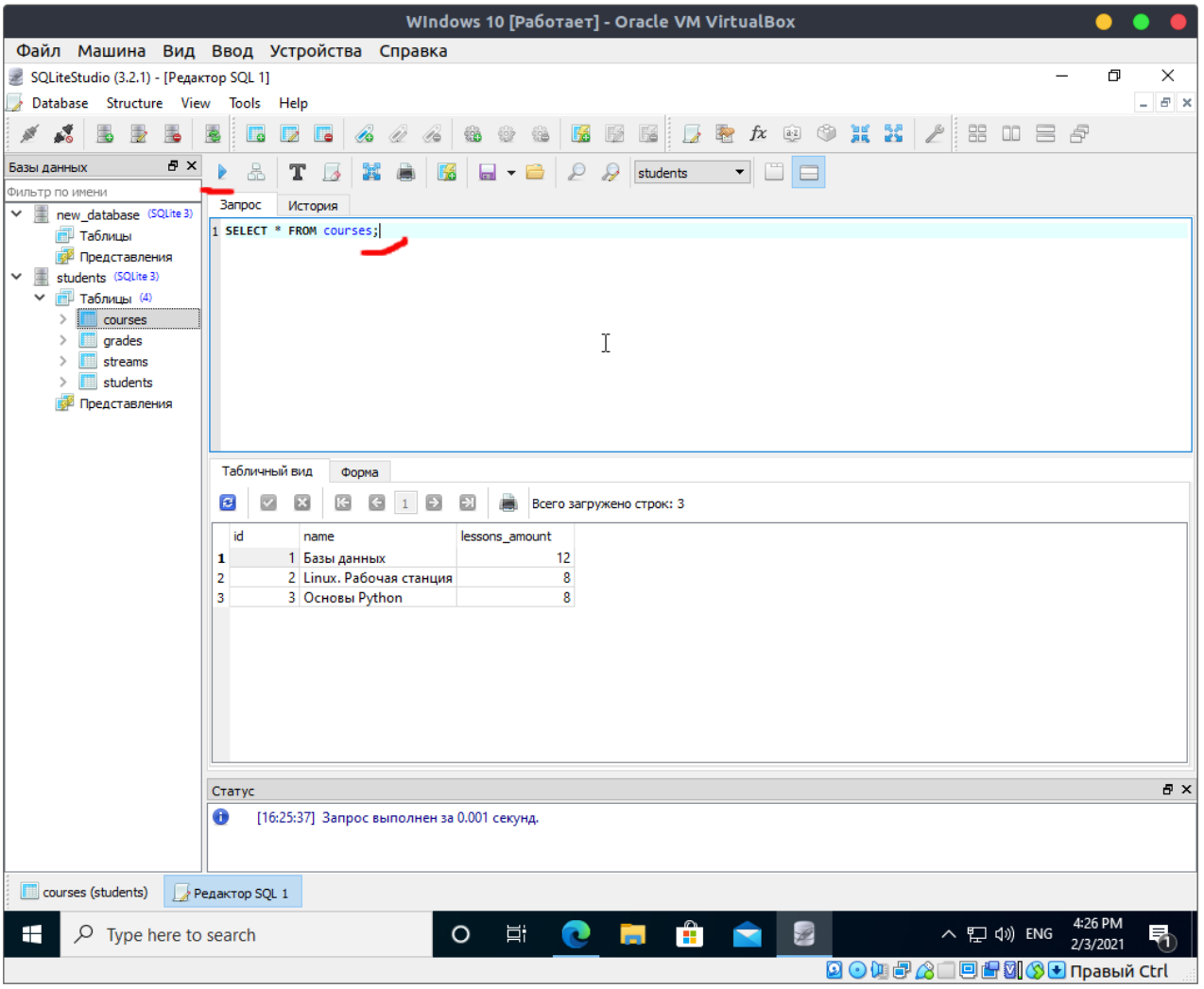


Выполнение команд

Посредством графического клиента SQLiteStudio выполняются операции с данными. Рассмотрим, как выполняется запрос на получение данных. В меню Tools выбираем пункт Open SQL editor.



В открывшемся окне редактора вводим подходящую команду, устанавливаем курсор после точки с запятой и нажимаем значок треугольника — выполняем запрос.



Практическое задание

Работаем с базой данных учителей teachers.db. Для каждого задания требуется сдать только код, который выполняется для получения результата, в текстовом файле. В качестве отчёта к четвёртому заданию надо приложить скриншот.

1. Найдите общее количество учеников для каждого курса. В отчёт выведите название курса и количество учеников по всем потокам курса. Решите задание с применением оконных функций.
2. Найдите среднюю оценку по всем потокам для всех учителей. В отчёт выведите идентификатор, фамилию и имя учителя, среднюю оценку по всем проведённым потокам. Учителя, у которых не было потоков, также должны попасть в выборку. Решите задание с применением оконных функций.
3. Какие индексы надо создать для максимально быстрого выполнения представленного запроса?

```
SELECT
  surname,
  name,
  number,
  performance
FROM academic_performance
  JOIN teachers
    ON academic_performance.teacher_id = teachers.id
  JOIN streams
    ON academic_performance.stream_id = streams.id
WHERE number >= 200;
```

4. Установите SQLiteStudio, подключите базу данных учителей, выполните в графическом клиенте любой запрос.
5. **Дополнительное задание.** Для каждого преподавателя выведите имя, фамилию, минимальное значение успеваемости по всем потокам преподавателя, название курса, который соответствует потоку с минимальным значением успеваемости, максимальное значение успеваемости по всем потокам преподавателя, название курса, соответствующий потоку с максимальным значением успеваемости. Выполните задачу с использованием оконных функций.

Глоссарий

Двоичный поиск — бинарный поиск, метод деления на два — алгоритм поиска, основанный на последовательном разделении набора данных на половины и определении, в какой из половин находится искомое значение.

Индекс — структура базы данных, которая содержит упорядоченные значения столбца или нескольких столбцов, а также ссылки на место в файле БД, где хранятся соответствующие строки таблицы.

Оконные функции — позволяют выполнить действия над наборами строк, которые объединяются по некоторому определённому признаку.

Дополнительные материалы

1. Статья «Учимся применять оконные функции».
2. Статья «Unetway. Индексы».

Используемые источники

1. Документация SQLite, Windows Functions.

2. Документация SQLite, Create Index.

3. Менеджер баз данных SQLiteStudio.