Пустое значение NULL и значения по умолчанию

Когда мы знакомились с запросами INSERT, то использовали такой пример:

```
insert into Tasks
(Task, Description, Tasklist_ID)
values
('Записаться на ТО', 'Записаться на техобслуживание машины', 5),
('Купить стол', 'Купить письменный стол в комнату сына', 5)
```

В этом примере в таблице **Tasks** создаются две записи, в которых указанные значения сохраняются в трёх полях: Task, Description и Tasklist_ID.

Но в таблице Tasks пять полей, а не три. Полю Task_ID, как мы говорили, база данных присваивает значение автоматически. А что происходит с оставшимся полем Done? Какие значения появятся в этом поле в двух добавленных записях?

Для таких случаев в базах данных изначально было предусмотрено пустое значение, для которого в SQL есть специальное ключевое слово: **NULL**. По умолчанию (то есть если явно не заданы другие правила) в новых записях полям, не указанным в запросе INSERT, присваивается значение NULL.

Важно понимать: NULL не является ни нулевым значением, ни пустой строкой, ни значением FALSE для полей логического типа. Поскольку NULL означает фактическое отсутствие какого-либо значения, все знакомые нам операторы сравнения не будут работать с NULL. Чтобы использовать NULL в условиях отбора записей, нужно использовать специальные конструкции **IS NULL** и **IS NOT NULL**, например:

```
where Done is NULL
```

для отбора записей, в которых полю Done содержит пустое значение, или

```
where Done is not NULL
```

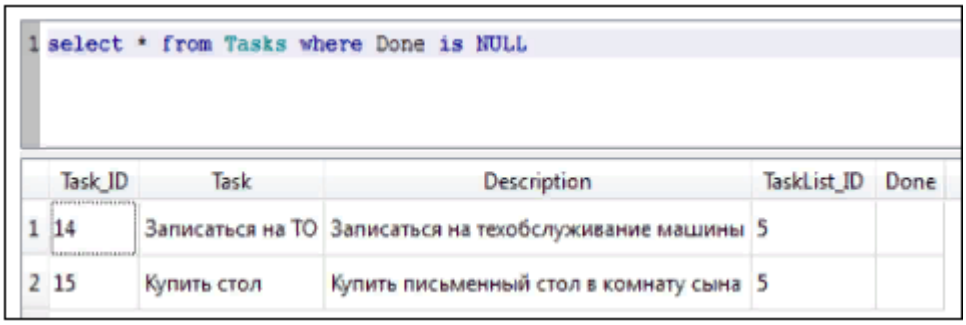
для отбора записей, в которых поле Done содержит любое непустое значение.

Мы можем проверить это с использованием приведенного выше примера. Выполните этот запрос с учебной базой данных:

```
insert into Tasks
(Task, Description, Tasklist_ID)
values
('Записаться на ТО', 'Записаться на техобслуживание машины', 5),
('Купить стол', 'Купить письменный стол в комнату сына', 5)
```

Теперь проверьте, какие записи содержат поле Done с пустыми значениями с помощью такого запроса:

```
select * from Tasks where Done is NULL
```



Мы можем убедиться, что все остальные известные нам условия отбора не работают с пустыми значениями. Например, выполните такие запросы:

```
select * from Tasks where Done=NULL
```

```
select * from Tasks where Done in (NULL)
```

Они не вернут ни одной записи.

```
0 Rows returned from: select * from Tasks where Done in (NULL) (took 2ms)
```

Следующий запрос также проигнорирует записи с пустыми значениями, хоть и вернёт записи с непустыми:

```
select * from Tasks where Done in (0, 1, NULL)
```

Эти особенности работы с NULL очень часто приводят к недоразумениям и ошибкам. Но их можно избежать, запретив использование NULL для отдельных или для всех полей базы данных.

В SQL можно явно указать, что значения NULL для данного поля запрещены. В описании такого поля при создании таблицы нужно добавить ключевые слова NOT NULL.

```
create table Tasks(
Task_ID integer,
Task text,
Description text,
TaskList_ID integer NOT NULL,
Done integer NOT NULL)
```

В этом примере запрет использования NULL задан для полей Tasklist_ID и Done. Но что произойдёт, если мы теперь попытаемся добавить в эту таблицу запись, в которой не указаны значения этих полей?

Давайте посмотрим. Создайте с помощью SQLite Database Browser новую базу данных для экспериментов с помощью команды File / New Database. Укажите имя файла — например, test-not-null.sqlite. Сразу после создания программа откроет диалоговое окно с предложением создать новую таблицу. Нажмите кнопку «отмена»: мы создадим новую таблицу сами, с помощью команды SQL.

Перейдите на вкладку Execute SQL, скопируйте в окно запроса приведенную выше команду и нажмите F5 для запуска его выполнения. Запрос выполнится и создаст новую таблицу Tasks.

```
Query executed successfully: create table Tasks(
  Task_ID INTEGER,
  Task TEXT,
  Description TEXT,
  TaskList_ID INTEGER NOT NULL,
  Done INTEGER NOT NULL) (took 1ms)
```

Теперь попробуйте добавить в эту таблицу записи с помощью запроса, который мы использовали раньше при изучении пустого значения NULL:

```
insert into Tasks
(Task, Description, Tasklist_ID)
values
('Записаться на ТО', 'Записаться на техобслуживание машины', 5),
('Купить стол', 'Купить письменный стол в комнату сына', 5)
```

Движок базы данных откажется выполнять этот запрос, так как в нём не указаны значения для поля Done, а присваивать этому полю пустые значения мы запретили:

```
NOT NULL constraint failed: Tasks.Done: insert into Tasks
(Task, Description, Tasklist_ID)
values
('Записаться на ТО', 'Записаться на техобслуживание машины', 5),
('Купить стол', 'Купить письменный стол в комнату сына', 5)
```

Теперь, чтобы добавлять записи в таблицу Tasks, мы всегда должны явно указывать значение поля Done.

Но давайте вспомним смысл этого поля: оно принимает значение 0 для невыполненных задач, и 1 для выполненных. Очевидно, когда мы добавляем новую задачу в список, она ещё не выполнена, а значит, это поле в новой записи всегда должно содержать значение 0. Можно ли как-то указать базе данных, чтобы она автоматически присваивала значение 0 этому полю во всех новых записях?

Ответ: да, в SQL есть и такая возможность. В описании поля можно указать значение, которое должно присваиваться ему по умолчанию (то есть в случаях, когда мы не указываем его явно). Для этого используется ключевое слово **DEFAULT**:

```
Done integer default 0
```

Ключевое слово DEFAULT может комбинироваться с указанием NOT NULL. В этом случае описание поля будет выглядеть так:

```
Done integer not NULL default 0
```

А полностью команда создания таблицы с этим полем может быть представлена так:

```
create table Tasks(
Task_ID integer,
Task text,
Description text,
TaskList_ID integer not NULL,
Done integer not NULL default 0)
```

Удаление и изменение таблиц

Удаление таблицы

Чтобы проверить, как работает ключевое слово DEFAULT, нам нужно создать таблицу Tasks с использованием приведенной выше команды. Но в нашей базе данных для экспериментов уже есть таблица Tasks, а в одной базе данных, как мы помним, не может быть двух таблиц с одинаковыми именами.

Можно было бы, конечно, создать ещё одну экспериментальную базу данных. Но мы лучше используем этот конфликт с пользой, познакомившись с командой удаления таблиц, которую мы уже упоминали:

- DROP TABLE

Эта команда полностью удаляет таблицу базы данных во всеми данными, которые в ней хранились. Команда имеет очень простую структуру:

```
drop table {имя таблицы}
```

Для таблицы Tasks эта команда выглядит так:

```
drop table Tasks
```

Выполните эту команду в учебной базе данных Tasks. Все данные таблицы будут потеряны, но вы всегда можете заново скачать файл учебной базы данных для продолжения экспериментов.

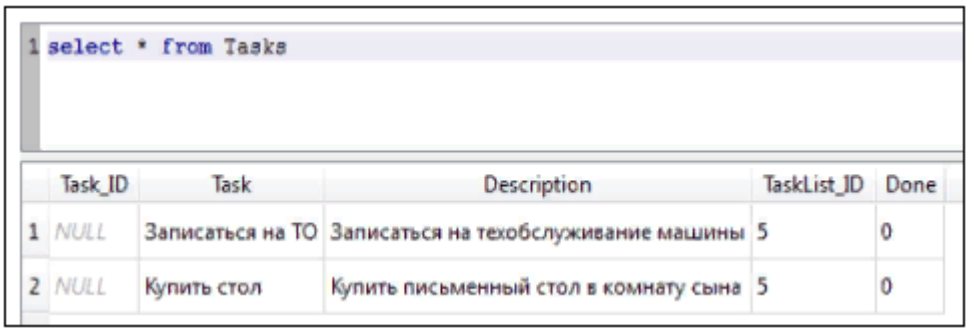
Теперь мы можем выполнить команду создания таблицы с новыми настройками поля Done:

```
create table Tasks(
Task_ID integer,
Task text,
Description text,
TaskList_ID integer not NULL,
Done integer not NULL default 0)
```

Посмотрим, как теперь сработает запрос на добавление записей, в котором не указаны значения этого поля:

```
insert into Tasks
(Task, Description, Tasklist_ID)
values
('Записаться на ТО', 'Записаться на техобслуживание машины', 5),
('Купить стол', 'Купить письменный стол в комнату сына', 5)
```

Выполним этот запрос и посмотрим на содержимое таблицы Tasks (у вас, конечно, уже не вызывает проблем моментальное составление запроса, возвращающего все записи таблицы):



```
select * from Tasks
```

	Task_ID	Task	Description	TaskList_ID	Done
1	NULL	Записаться на ТО	Записаться на техобслуживание машины	5	0
2	NULL	Купить стол	Купить письменный стол в комнату сына	5	0

Как видим, поле Done в обеих записях получило значение 0, как и было указано. Но значения NULL теперь появились в поле Task_ID, потому что в нашей команде создания таблицы указан только тип этого поля без каких-либо дополнительных ограничений!

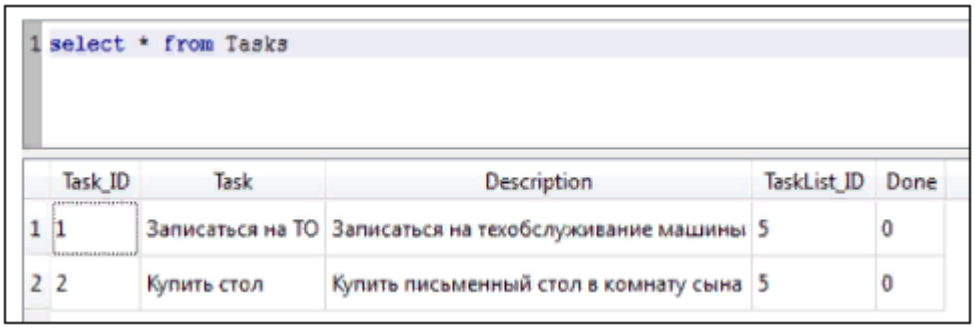
Чтобы исправить эту ситуацию, снова удалим таблицу Tasks командой DROP TABLE и создадим её заново с помощью доработанной команды CREATE TABLE:

```
create table Tasks(
Task_ID integer primary key autoincrement,
Task text,
Description text,
TaskList_ID integer not NULL,
Done integer not NULL default 0)
```

В описании поля Task_ID появились ключевые слова PRMARY KEY AUTOINCREMENT, на которые мы уже обращали внимание. Значение слов PRIMARY KEY мы разберём позже в нашем курсе. А слово AUTOINCREMENT в описании поля означает, что значение этого поля должно автоматически увеличиваться при добавлении каждой новой записи.

Попробуем снова выполнить запрос на добавление двух записей и посмотрим на результат.

```
insert into Tasks
(Task, Description, Tasklist_ID)
values
('Записаться на ТО', 'Записаться на техобслуживание машины', 5),
('Купить стол', 'Купить письменный стол в комнату сына', 5)
```



```
select * from Tasks
```

	Task_ID	Task	Description	TaskList_ID	Done
1	1	Записаться на ТО	Записаться на техобслуживание машины	5	0
2	2	Купить стол	Купить письменный стол в комнату сына	5	0

Теперь с записями всё в порядке: поле Done получило значение по умолчанию 0, а поле Task_ID заполняется последовательно увеличивающимися значениями.

Изменение таблицы

В предыдущей главе мы удалили целую таблицу, чтобы создать её заново с изменённым полем Task_ID. При удалении таблицы удаляются безвозвратно и все содержащиеся в ней данные. Но что делать, если мы не хотим терять данные? Можно ли внести изменения в структуру таблицы, не удаляя её?

SQL предоставляет такую возможность, для этого используется третья команда для работы с таблицами:

- ALTER TABLE

С её помощью можно добавлять, изменять и удалять отдельные поля таблицы. В обобщённом виде эту команду можно представить в трёх формах.

Для добавления поля:

```
alter table {имя таблицы} add column {описание поля}
```

Для изменения поля:

```
alter table {имя таблицы} alter column {описание поля}
```

Для удаления поля:

```
alter table {имя таблицы} drop column {имя поля}
```

Ключевое слово COLUMN во всех трёх вариантах является необязательным.

Команда на изменение поля Task_ID могла бы выглядеть так:

```
alter table Tasks alter column Task_ID integer primary key autoincrement
```

К сожалению, движок SQLite поддерживает команду ALTER TABLE в очень усечённом виде: он позволяет только добавлять к таблице новые поля. Поэтому мы не смогли воспользоваться этой возможностью.

Использование NULL. Создание и удаление таблиц — тест

Какой из запросов вернёт все записи таблицы Tasks, в которых поле Description имеет значение NULL?

Выберите один или несколько ответов:

- a. select * from Tasks where Description is NULL
- b. select * from Tasks where Description = 'NULL'
- c. select * from Tasks where Description in (NULL)
- d. select * from Tasks where Description = NULL

Какой из запросов вернёт все записи таблицы Categories, в которых поле Category имеет значение, отличное от NULL?

Выберите один или несколько ответов:

- a. select * from Categories where Category not in (NULL)
- b. select * from Categories where Category <> NULL
- c. select * from Categories where Category != NULL
- d. select * from Categories where Category is not NULL

Какой из предложенных вариантов описания поля Done при создании таблицы Tasks нужно использовать, чтобы этому полю по умолчанию присваивалось значение 0?

Выберите один или несколько ответов:

- a. Done integer default NULL
- b. Done integer default 0
- c. Done integer NULL
- d. Done integer (0)

Какой запрос удалит таблицу Tasks из базы данных?

Выберите один ответ:

- a. drop Tasks
- b. delete from Tasks
- c. delete table Tasks
- d. drop table Tasks