Introduction to Databases and SQL

ЛЕКЦИЯ 4

Темы занятия

Нормализация

Добавление строк в таблицу

Выборка информации – простейшие варианты

Фильтрация

Сортировка

Уникальность строк и ограничение длины выборки

Нормализация

Нормализация – процесс преобразования таблиц базы данных в одну из нормальных форм.

Нормальная форма – набор требований к таблице, характеризующих таблицу с точки зрения избыточности.

Короче: **нормализация – изменение структуры БД для устранения избыточности данных.**

Первая нормальная форма (1NF)

Таблица находится в **первой нормальной форме**, если каждое её поле *атомарно*: значения в поле не может быть разделено на фрагменты, имеющие самостоятельный смысл.

Первая нормальная форма (1NF)

Что не так с этой таблицей?

Непорядок с колонкой `Phones` – не ясно, сколько там телефонов, и каким должен быть размер колонки.

ID	PersonName	Phones
1	Alex	333-4444
2	Mary	555-7777, 111-2222
3	John	
4	Paul	789-0123, 096-7654, 888-1111

Первая нормальная форма (1NF)

Приведём таблицу к первой нормальной форме.

ID	PersonName
1	Alex
2	Mary
3	John
4	Paul

PersonID	Phone
1	333-4444
2	555-7777
2	111-2222
4	789-0123
4	096-7654
4	888-1111

Вторая нормальная форма (2NF)

Таблица находится во **второй нормальной форме**, если она находится в 1NF и любая не ключевая колонка зависит от всего первичного ключа, а не от части ключа.

Хорошая новость: если первичный ключ атомарный и таблица в 1NF, то такая таблица уже находится в 2NF.

Вторая нормальная форма (2NF)

Первичным ключом таблицы является пара колонок (`Module`,`Lecture`). Но значения `ModuleName` зависят только от значений `Module`.

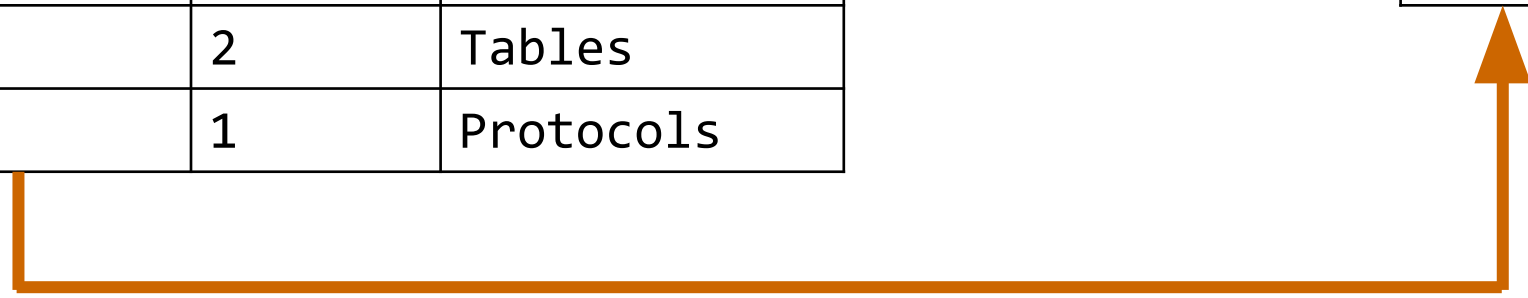
Module	Lecture	ModuleName	LectureName
1	1	Programming	Intro
1	2	Programming	JavaScript
2	1	SQL	Intro
2	2	SQL	Tables
3	1	Networks	Protocols

Вторая нормальная форма (2NF)

Приведём таблицу ко второй нормальной форме, выделив значения `ModuleName` в отдельную таблицу.

Module	Lecture	LectureName
1	1	Intro
1	2	JavaScript
2	1	Intro
2	2	Tables
3	1	Protocols

Module	ModuleName
1	Programming
2	SQL
3	Networks



Третья нормальная форма (3NF)

Таблица находится в **третьей нормальной форме**, если она находится в 2NF и любая не ключевая колонка зависит от первичного ключа **и только** от первичного ключа.

Третья нормальная форма (3NF)

Первичный ключ – колонка `ID`. Значения в колонке `Position` зависят только от колонки `PositionCode`.

ID	Name	PositionCode	Position
1	Alex	DEV	Developer
2	Mary	QA	QA Engineer
3	John	QA	QA Engineer
4	Paul	DEV	Developer

Третья нормальная форма (3NF)

Приведём таблицу к третьей нормальной форме, выделив `Position` в отдельную таблицу.

ID	Name	PositionCode
1	Alex	DEV
2	Mary	QA
3	John	QA
4	Paul	DEV

Code	Position
DEV	Developer
QA	QA Engineer

Денормализация

Денормализация – намеренное приведение структуры базы в состояние, не соответствующее критериям нормализации.

Зачем денормализовать? Минимум две причины:

1. Повышение производительности запросов
2. Сохранение исторических данных

Таблица Persons

В дальнейших примерах презентации будет использоваться таблица `Persons` (данные сотрудников):

Persons

	Column Name	Data Type	Allow Nulls
🔑	ID	int	☐
	FirstName	varchar(50)	☐
	LastName	varchar(50)	☑
	Department	char(2)	☐
			☐

Добавление строк в таблицу

Добавление строк выполняется при помощи инструкции `INSERT`. Указывается имя таблицы. В простейшем варианте в скобках задаётся значения всех полей добавляемой строки через запятую:

```
INSERT INTO Persons
VALUES (10, 'Alex', 'Volosevich', 'TC');
```

*) для `IDENTITY`-колонки значение не указывается.

Добавление строк в таблицу

Простейший вариант `INSERT` обладает недостатками:

- порядок значений в скобках должен соответствовать порядку колонок таблицы;
- не ясно, как (не)вставлять значения для колонок с `DEFAULT` и `NULL`.

Добавление строк в таблицу

При использовании INSERT можно после имени таблицы перечислить в скобках колонки, значения для которых указываются в списке VALUES:

```
INSERT INTO Persons(ID, Department, FirstName)
VALUES (11, 'TC', 'Alex');
```

Добавление множества строк

T-SQL позволяет при помощи одной инструкции `INSERT` добавить в таблицу несколько строк:

```
INSERT INTO Persons(ID, FirstName, LastName, Department)
VALUES (1, 'Anna', 'Klimenok', 'QA'),
       (2, 'Olga', 'Chekan', 'QA'),
       (3, 'Olga', 'Naumik', 'QA'),
       (4, 'Alexey', NULL, 'TC'),
       (5, 'Oleg', NULL, 'TC'),
       (6, 'Sergey', 'Pavlov', 'DV');
```

Выборка информации

Инструкция `SELECT` возвращает набор данных (*выборку*), удовлетворяющих заданным условиям.

В простейшем варианте `SELECT` извлекает информацию из всех колонок и всех строк одной указанной таблицы:

```
SELECT * FROM Persons
```

Указание колонок таблицы

Вместо * можно перечислить через запятую колонки таблицы, из которых будет формироваться выборка (это называется *проекция*). Колонки можно указывать в любом порядке или даже повторять:

```
SELECT FirstName, ID, ID FROM Persons
```

Псевдонимы колонок

При выборке для колонки можно указать *псевдоним*, и данные попадут в выборку под указанным именем:

```
SELECT ID, FirstName AS Name FROM Persons
```

В T-SQL при задании псевдонимов можно не писать `AS` (просто пробел поставить) или вместо `AS` ставить `=` (в этом случае псевдоним пишем слева):

```
SELECT ID, Name = FirstName FROM Persons
```

Псевдоним таблицы

Если выборка производится из нескольких таблиц, у которых есть колонки с одинаковыми именами, то на колонку нужно ссылаться так:

имя_таблицы.имя_колонки

Для удобства можно использовать псевдоним таблицы:

```
SELECT P.ID FROM Persons AS P
```

Операции с данными при выборке

При выборке можно осуществить операции с данными:

```
SELECT ID * 10, FirstName,
       GETDATE(), 5 FROM Persons
```

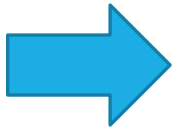
	(No column name)	FirstName	(No column name)	(No column name)
1	10	Anna	2016-04-08 12:44:44.287	5
2	20	Olga	2016-04-08 12:44:44.287	5
3	30	Olga	2016-04-08 12:44:44.287	5
4	40	Alexey	2016-04-08 12:44:44.287	5
5	50	Oleg	2016-04-08 12:44:44.287	5
6	60	Sergey	2016-04-08 12:44:44.287	5

Что же делает SELECT?

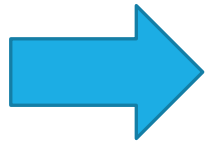
SELECT, по сути, выполняет *преобразование данных*. Мы указываем, как получить одну строку выборки, а SELECT повторяет наши указания для всех строк.

```
SELECT X=A, B, Y=C+D, Z=10 FROM Tbl
```

A	B	C	D
1	10	-1	0
2	20	-2	1
3	30	-3	0
4	40	-4	1



```
X = A
B = B
Y = C+D
Z = 10
```



X	B	Y	Z
1	10	-1	10
2	20	-1	10
3	30	-3	10
4	40	-3	10

Фильтрация

Данные в источнике для выборки можно отфильтровать при помощи предложения `WHERE`, которое записывается после `SELECT`-части и содержит условие-фильтр (это условие называется *предикатом*):

```
SELECT FirstName, LastName FROM Persons
WHERE Department = 'QA'
```

Построение предиката

При построении предиката используются операции сравнения, логические операции AND, OR, NOT, операции IN (NOT IN) и BETWEEN (NOT BETWEEN):

```
SELECT FirstName, LastName FROM Persons
WHERE (Department <> 'QA' AND ID >= 5);

SELECT FirstName, LastName FROM Persons
WHERE ID IN (1, 3, 5);

SELECT FirstName, LastName FROM Persons
WHERE ID BETWEEN 2 AND 4;
```

Сравнение с NULL

Чтобы сравнить значение с `NULL`, используются операции `IS NULL` и `IS NOT NULL`:

```
SELECT FirstName, LastName FROM Persons
WHERE LastName IS NULL
```

```
SELECT FirstName, LastName FROM Persons
WHERE LastName IS NOT NULL
```

Сравнение строк с шаблоном

Используя оператор `LIKE`, строки можно сравнивать с шаблоном. В шаблоне `_` означает один произвольный символ, а `%` – набор любых символов:

```
SELECT FirstName, LastName FROM Persons
WHERE FirstName LIKE 'Ol%'      -- Oleg, Olga

SELECT FirstName, LastName FROM Persons
WHERE FirstName LIKE 'Ol_a'     -- Olga
```

Сортировка

Строки в выборке можно отсортировать, используя предложение `ORDER BY`:

```
SELECT FirstName, LastName FROM Persons
WHERE Department = 'QA'
ORDER BY LastName
```

Сортировка

После `ORDER BY` указывает колонка или выражение, по которому производится сортировка. Колонку можно указать с помощью имени или псевдонима. И эта колонка не обязана быть упомянута в `SELECT`.

Можно указать несколько колонок. Тогда выборка сортируется по первой колонке, затем упорядоченный набор сортируется по второй колонке и так далее.

Сортировка по нескольким колонкам

Сортируем данные из `Persons` по имени, а при совпадении имён – по фамилии:

```
SELECT FirstName, LastName FROM Persons
ORDER BY FirstName, LastName
```

	FirstName	LastName
1	Alexey	NULL
2	Anna	Klimenok
3	Oleg	NULL
4	Olga	Chekan
5	Olga	Naumik
6	Sergey	Pavlov

Сортировка

После имени колонки можно задать направление сортировки: `ASC` – по возрастанию значений (это работает по умолчанию), или `DESC` – по убыванию:

```
SELECT FirstName, LastName FROM Persons
ORDER BY FirstName DESC, LastName ASC
```

	FirstName	LastName
1	Sergey	Pavlov
2	Olga	Chekan
3	Olga	Naumik
4	Oleg	NULL
5	Anna	Klimenok
6	Alexey	NULL

Требование уникальности

Указание `DISTINCT` **сразу после** ключевого слова `SELECT` приводит к удалению повторяющихся строк из выборки:

```
SELECT DISTINCT FirstName FROM Persons
```

	FirstName
1	Alexey
2	Anna
3	Oleg
4	Olga
5	Sergey

Ограничение длины выборки

Количество строк выборки можно ограничить, указав предложение TOP:

```
SELECT TOP 2 ID, FirstName FROM Persons
```

T-SQL позволяет задать в TOP **процент** от общего числа строк выборки:

```
SELECT TOP 25 PERCENT ID, FirstName FROM Persons
```

Ограничение длины выборки

В T-SQL есть опция `WITH TIES` – **не разрывать** набор по отсортированным значениям. Её можно применять только вместе с `ORDER BY`:

```
SELECT TOP 2 WITH TIES Department FROM Persons
ORDER BY Department
```

	Department
1	DV
2	QA
3	QA
4	QA

Последовательность ограничений

При наличии различных ограничений они работают так:

1. Фильтрация
2. Удаление дубликатов
3. Сортировка
4. Ограничение по длине выборки

Нюанс: после применения `DISTINCT` получаем новый набор. И если после этого сортируем, то надо использовать только столбцы, упомянутые в `SELECT`.