

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования  
«Витебский государственный технологический университет»

## **БАЗЫ ДАННЫХ**

Методические указания по выполнению расчетно-графической работы  
для студентов специальности 1-40 05 01-01 «Информационные системы и  
технологии (в проектировании и производстве)»

Витебск  
2018

УДК 7.01 (075.8) (14 пт)

Составители:

**В.Е. Казаков, Д.В. Черненко**

Рекомендовано к изданию редакционно-издательским  
советом УО «ВГТУ», протокол № 8 от 30.11.2012.

**Базы данных** : методические указания по выполнению расчетно-графической работы / сост. **В.Е. Казаков, Д.В. Черненко** – Витебск : УО «ВГТУ», 2018. – 22 с.

Методические указания являются руководством по выполнению расчетно-графической работы по дисциплине «Базы данных», определяют порядок выбора студентом темы работы, общие требования, предъявляемые к расчетно-графической работе, освещают последовательность ее подготовки, требования к структуре и содержанию.

Предназначены для студентов дневной и заочной на базе ССУЗ формы обучения специальности 1-40 05 01-01 «Информационные системы и технологии (в проектировании и производстве)»

**УДК 7.01 (075.8) (12 пт)**

© УО «ВГТУ», 2018 (12 пт)

Рассмотрим выполнение всех этапов РГР на примере проектирования и создания базы данных «Издательский отдел», содержащей информацию об авторах (фамилия, контактный телефон и т.д.), книгах (название, автор, издательство и т.д.) и издательствах (название, город и т.д.).

Таблицы 1 - Представим данные хранимые в БД в виде следующей:

Данные	Описание
1	2
Author	Автор
Book	Книга
Publish	Издательство

Проектирование БД начнем с нормализации.

Нормализация (normalization) – группировка и/или распределение атрибутов по отношениям с целью устранения аномалий операций с БД, обеспечения целостности данных и оптимизации модели БД.

Нормальных форм девять, но, на практике, вы не увидите базу данных в пятой или даже в четвертой нормальной форме. Редко кто нормализует отношение до нормальной формы Бойса-Кодда (усиленная третья нормальная форма, ее частный случай). Обычно нормализацию проводят до третьей нормальной формы.

### **Первая нормальная форма (1НФ)**

Отношение находится в 1НФ, если все его атрибуты являются атомарными, т.е. не имеют компонентов. Можно сказать, что таблица в 1НФ, так как:

- устранены повторяющиеся группы в отдельных таблицах;
- созданы отдельные таблицы для каждого набора связанных данных;
- каждый набор связанных данных идентифицирован с помощью первичного ключа.

В одной таблице не может использоваться несколько полей для хранения похожих данных.

Ключевым свойством первой нормальной формы является атомарность, т.е. значение в ячейки таблицы может быть только одно. Однако у авторов

Иванов И.В. и Петрова О.С. в таблице 1 в столбце Title указано два значения. Поэтому эта таблица, не находится в 1НФ.

Таблицы 2 – Таблица авторов не в 1НФ.

Author	Title	Tel
1	2	3
Иванов И.В.	Рассказы, Роман	(29)322-33-44
Петрова О.С	Стихи, Проза	(33)524-12-15
Пупкин С.П.	Повесть	(29)761-20-89

Приведем нашу таблицу к 1НФ.

Таблицы 3 – Таблица авторов в 1НФ.

Author	Tel	Title	Publish	City
1	2	3	4	5
Иванов И.В.	(29)322-33-44	Рассказы	Мир книги	Минск
Иванов И.В.	(29)322-33-44	Роман	Книжный пресс	Москва
Петрова О.С	(33)524-12-15	Стихи	Опечатка	Витебск
Петрова О.С	(33)524-12-15	Проза	Книжный пресс	Москва
Пупкин С.П.	(29)761-20-89	Повесть	Опечатка	Витебск

Таблица 3 находится в 1НФ, поскольку в каждой ячейки содержится только одно логическое значение, то есть соблюдено свойство атомарности. Добились мы этого путем дублирования данных, следовательно, внесли избыточность в базу данных.

Избавиться от избыточности в базе данных нам поможет вторая нормальная форма.

### Вторая нормальная форма (2НФ)

Отношение находится во 2НФ, если оно находится в 1НФ, и при этом любой атрибут, не входящий в состав первичного ключа (ПК), функционально полно зависит от ПК. К тому же 2НФ не может обойтись без ПК. Можно сказать, что таблица находится в 2НФ, так как:

- созданы отдельные таблицы для наборов значений, относящихся к нескольким записям;
- эти таблицы были связаны с помощью внешнего ключа.

Первые два столбца автор и его номер телефона, а также два последних издательство и город связаны между собой функционально. Выявив функциональные связи, можно создать таблицы-справочники на основе этих связей.

Разобьем нашу таблицу на три: spr\_Author, spr\_Publish, tab\_Book. Выполнив такое преобразование, мы получили отношение, находящееся во

2НФ, избавились от избыточности данных и упростили работу по обслуживанию базы данных.

Представлять полученную структуру удобнее в виде ER-диаграмм.

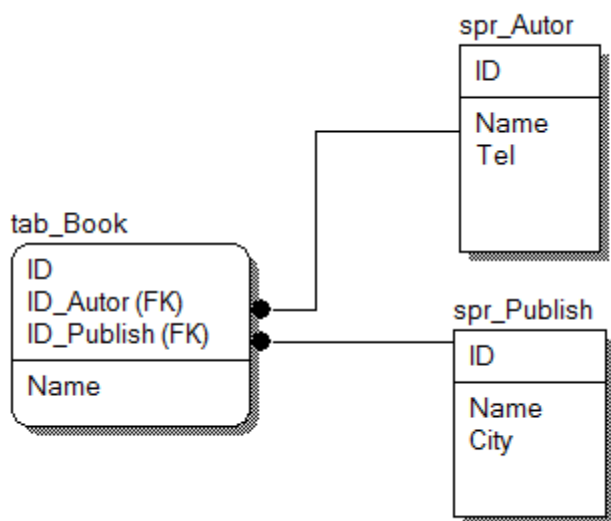


Рисунок 1 – ER-диаграмма отношения находящегося во 2НФ

### Третья нормальная форма (3НФ)

Переменная отношения находится в 3НФ, когда она находится во 2НФ, и отсутствуют транзитивные функциональные зависимости не ключевых атрибутов от ключевых, т.е. любой столбец таблицы должен зависеть только от ключевого столбца. Она расширяет две предыдущие нормальные формы, неся в себе два правила:

- таблица должна соответствовать второй нормальной форме;
- все столбцы, не входящие в полный первичный ключ, должны зависеть от него и не должны зависеть друг от друга.

Во 2НФ в таблице Publish поле City логически связано с полем Name. Для того чтобы наше отношение находилось в 3НФ нам нужно разрушить эту связь. Это можно сделать путем разбиения существующей таблицы на два справочника. Первый справочник – справочник издательств, второй справочник – городов.

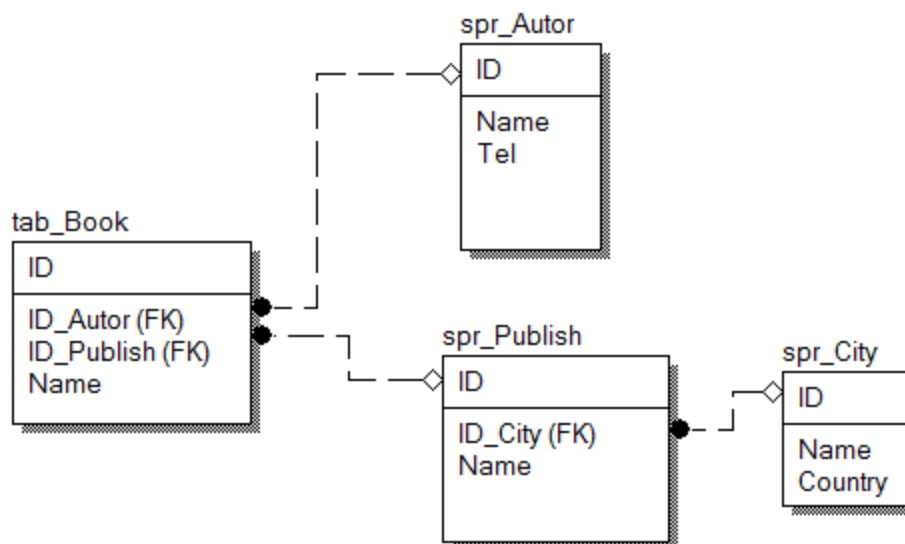


Рисунок 2 – ER-диаграмма отношения находящегося в 3НФ

Придя к 3НФ мы устранили избыточность данных в БД и аномалии, возникающие при добавлении и удалении данных. Взамен получили более сложную структуру, а значит, более сложные запросы к БД и как следствие более низкую скорость работы.

### Нормальная форма Бойса-Кодда(НФБК)

Отношение находится в НФБК, если оно находится в 3НФ и в нем отсутствуют зависимости атрибутов первичного ключа от не ключевых атрибутов возникает при условии, что отношение имеет два (или более) возможных ключа, которые являются составными и имеют общий атрибут.

### Четвертая нормальная форма (4НФ)

Отношение находится в 4НФ, если оно находится в 3НФ и в нем отсутствуют многозначные зависимости, не являющиеся функциональными зависимостями.

Существует несколько основных типов операторов SQL.

1. DDL (Data Definition Language) – язык определения данных. Он дает возможность создавать различные объекты БД и переопределять их структуру, например, создавать или удалять таблицы. Примеры операторов:

CREATE ALTER DROP RENAME

2. DML (Data Manipulation Language) – язык манипуляции данными. Он дает возможность манипулировать данными внутри объектов реляционной БД. Примеры операторов:

INSERT UPDATE DELETE

3. DQL (Data Query Language) – язык запросов к данным. Он используется для построения запросов к реляционным БД. Примеры операторов:

SELECT

4. DCL (Data Control Language) – язык управления данными. Он позволяет осуществлять контроль над возможностью доступа пользователей к данным внутри БД, а также для назначений пользователям привилегий доступа.

5. DAL (Data Administration Language) – язык администрирования данных. Он дает возможность выполнять аудит и анализ операций внутри БД.

6. Команды управления транзакциями.

Несмотря на то, что стандарт языка SQL определяется ANSI (американским национальным институтом стандартов) и ISO (международной организацией по стандартизации), разработчики коммерческих СУБД без уведомления расширяют SQL ANSI, интегрируют дополнительные возможности в этот язык. Поэтому в настоящее время существует большое количество диалектов SQL

### Создание таблиц

Таблицы создаются командой CREATE TABLE. Эта команда создает пустую таблицу. Команда CREATE TABLE определяет имя таблицы и саму таблицу в виде описания набора имен столбцов, указанных в определенном порядке. Она также определяет типы данных и размеры столбцов. Каждая таблица должна иметь, по крайней мере, один столбец.

Синтаксис команды:

CREATE TABLE <имя\_таблицы>

(

<имя\_поля1> <тип\_поля1> [NOT NULL][UNIQUE | PRIMARY KEY]  
[REFERENCES <имя\_мастер\_таблицы> [(<имя\_связанного\_поля>)]],

...

[CONSTRAINT <имя\_ограничения>  
PRIMARY KEY (<имя\_поля1>[, ...]) |  
FOREIGN KEY (<имя\_поля1>[, ...]) REFERENCES <имя\_связанной\_таблицы>  
(<имя\_связанного\_поля1>[, ...]),... ]

Удалить можно только пустую таблицу. Заполненная таблица с находящимися в ней строками не может быть удалена, т. е. таблица перед удалением должна быть очищена. Команда на удаление таблицы имеет следующий вид:

DROP TABLE <имя\_таблицы>;

С помощью команды CREATE TABLE создаются поля (часть команды до ключевого слова CONSTRAINT) и ограничения таблицы (часть команды после ключевого слова CONSTRAINT).

Следует обратить внимание на то, что внешние ключи могут быть созданы непосредственно при описании поля (конструкция REFERENCES). В этом случае также создаётся ограничение, но такой способ его создания называют неявным. Несмотря на различные способы описания, каждое из создаваемых ограничений является отдельным, связанным с таблицей, объектом. Имена для неявно описанных ограничений задаются самим ядром СУБД.

### Типы данных

Понятие тип данных в БД полностью адекватно понятию типа данных в языках программирования.

Тип данных определяет:

- каков размер области памяти, занимаемой объектом;
- как интерпретируется эта память;
- какие действия можно выполнять с этим объектом.

В SQL используются следующие основные типы данных, форматы которых могут несколько различаться для разных СУБД:

а) символьные типы данных – содержат буквы, цифры и специальные символы:

- CHAR или CHAR(n) – символьные строки фиксированной длины. Длина строки определяется параметром n;

б) целые типы данных – поддерживают только целые числа (дробные части и десятичные точки не допускаются):

- INTEGER или INT – целое, для хранения которого отводится, как правило, 4 байта;
- SMALLINT – короткое целое (2 байта);

в) вещественные типы данных – описывают числа с дробной частью:

- FLOAT и SMALLFLOAT – числа с плавающей точкой;



д) дата и время – используются для хранения даты, времени и их комбинаций:

- DATETIME – тип данных для хранения моментов времени (год + месяц + день + часы + минуты + секунды + доли секунд);

е) двоичные типы данных – позволяют хранить данные любого объема в двоичном коде (оцифрованные изображения, исполняемые файлы и т.д.);

- BINARY, • BYTE, • BLOB.

### **Рассмотрим следующий пример:**

Разработать запрос для создания структуры базы данных, приведённой на рисунке. В начале создадим таблицы справочник городов spr\_City и справочник авторов spr\_Author

```
CREATE TABLE spr_City (ID Counter, Name char (50), Country char (3),  
PRIMARY KEY (ID));
```

```
CREATE TABLE spr_Author (ID Counter, Name char (50), Tel char (25),  
PRIMARY KEY (ID));
```

На втором этапе создадим справочник издательств spr\_Publish связанных со справочником городов с помощью внешнего ключа ID\_City

```
CREATE TABLE spr_Publish (ID Counter, Name char (50), ID_City Int,  
PRIMARY KEY (ID),  
FOREIGN KEY (ID_City) REFERENCES spr_City (ID));
```

На заключительном этапе создадим таблицу книг tab\_Book связанную внешними ключами со справочником издательств spr\_Publish - ID\_Publish и со справочником авторов spr\_Author - ID\_Author

```
CREATE TABLE tab_Book (ID Counter, Name char (50), Cena float,  
ID_Author Int, ID_Publish Int,
```

PRIMARY KEY (ID),

FOREIGN KEY (ID\_Publish) REFERENCES spr\_Publish (ID),

FOREIGN KEY (ID\_Author) REFERENCES spr\_Author (ID));

Последовательно выполнив все запросы, строго в указанной последовательности, получим набор таблиц входящих в нашу БД связанных между собой связями согласно разработанной ранее даталогической модели.

### **Ввод значений полей**

Данные заносятся в поля и исключаются из них с помощью трех команд языка манипулирования данными (Data Manipulation Language - DML: INSERT (вставить), UPDATE (обновить) и DELETE (удалить). В SQL их часто называют командами обновления (*update commands*).

Все строки в SQL вводятся с использованием команды модификации INSERT. Предложение INSERT имеет один из следующих форматов:

```
INSERT INTO < имя таблицы >  
[(имя_поля [,имя_поля] ...)]  
VALUES ( < значение > [,<значение >] ... );
```

или

```
INSERT INTO <имя таблицы>  
VALUES (<значение>[,<значение>] ... );
```

Рассмотрим следующий пример:

```
INSERT INTO spr_City VALUES (1,"Витебск","РБ");
```

```
INSERT INTO spr_City VALUES (2,"Минск","РБ");
```

```
INSERT INTO spr_City VALUES (3,"Москва","РФ");
```

```
INSERT INTO spr_Author VALUES (1," Иванов И.В.", "+375(29)322-33-44");
```

```
INSERT INTO spr_Author VALUES (2," Петрова О.С ", "+375(33)524-12-15");
```

```
INSERT INTO spr_Author VALUES (3," Пупкин С.П.", "+375(29)761-20-89");
```

```
INSERT INTO spr_Publish VALUES (1,"Мир книги", 2);
```

```
INSERT INTO spr_Publish VALUES (2,"Книжный пресс", 3);
```

```
INSERT INTO spr_Publish VALUES (3,"Опечатка", 1);
```

```
INSERT INTO tab_Book VALUES (1,"Рассказы", 5.20, 1, 1);
```

```
INSERT INTO tab_Book VALUES (2,"Роман", 10.15, 1, 2);
```

```
INSERT INTO tab_Book VALUES (3,"Стихи", 3.55, 2, 3);
```

```
INSERT INTO tab_Book VALUES (4,"Проза", 6.78, 2, 2);
```

```
INSERT INTO tab_Book VALUES (5,"Повесть", 8.35, 3, 3);
```

### **Извлечение информации из таблицы**

Результатом запроса на выборку данных из базы является таблица, а также перечисление и скалярное значение как частные случаи таблицы с одним полем и несколькими записями и одним полем и одной записью соответственно. Запрос на выборку – это описание того, как должна быть создана результирующая таблица.

Выборка информации из базы данных осуществляется с помощью команды SELECT языка SQL.

Синтаксис команды SELECT:

```
SELECT <список_вывода>  
FROM <источник_выборки>  
[ WHERE <условие_WHERE> ]  
[ GROUP BY <список_группировки> ]  
[ HAVING <условие_HAVING> ]  
[ ORDER BY <список_сортировки>];
```

<список\_вывода> состоит из выражений, описывающих содержимое столбцов результирующей таблицы.

<источник\_выборки> является конструкцией, описывающей таблицу, из которой будет производиться выборка.

<условие\_WHERE> – это логическое выражение, определяющее записи источника выборки, которые попадут в результирующую таблицу.

<список\_группировки> – это список выражений, на основе которых производится группировка записей результирующей таблицы.

<условие\_HAVING> – это логическое выражение, определяющее записи сгруппированной таблицы, которые попадут в результирующую таблицу.

<список\_сортировки> – это список выражений, на основе которых производится сортировка записей результирующей таблицы.

Рассмотрим основные приёмы, используемые при составлении запросов.

### **Фильтрация**

Чаще всего требуется работать с подмножеством строк таблицы. Поэтому все SQL выражения для работы с данными включают блок where, где

размещаются всевозможные фильтры. Блок having, включает фильтрацию, относящиеся к группам данных.

К операторам, используемым в условиях, относятся:

- Операторы сравнения, такие как =, !=, <, >, <>
- Логические операторы

Таблицы 4 – Таблица логических операторов.

Логический оператор	Действие
1	2
ALL	TRUE, если весь набор сравнений дает результат TRUE
AND	TRUE, если оба булевых выражения дают результат TRUE
ANY	TRUE, если хотя бы одно сравнение из набора дает результат TRUE
BETWEEN	TRUE, если операнд находится внутри диапазона
EXISTS	TRUE, если подзапрос возвращает хотя бы одну строку
IN	TRUE, если операнд равен одному выражению из списка или одной или нескольким строкам, возвращаемым подзапросом
LIKE	TRUE, если операнд совпадает с шаблоном
NOT	Обращает значение любого другого булевого оператора
OR	TRUE, если любое булево выражение равно TRUE
SOME	TRUE, если несколько сравнений из набора дают результат TRUE

Для того чтобы работа фильтра была максимально эффективна в запросе необходимо обязательно использовать логические операторы.

В качестве примера выберем имена авторов, чей телефон имеет код 29.

```
SELECT Name
FROM spr_Author
WHERE (Tel like "(29)*");
```

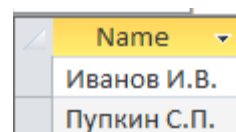


Рисунок 3 – Результат выполнения запроса

### Вычисляемое поле результирующей таблицы

В языке SQL как и в любом языке программирования существует возможность проводить различные вычисления. В качестве примера выведем таблицу цен на книги в различной валюте по установленному курсу:

```
SELECT Name, round(Cena/1.9556, 2) as [Цена в USD],  
        round(Cena/2.3897, 2) as [Цена в EUR],  
        round(Cena/3.4693*100, 2) as [Цена в RUB]  
FROM tab_Book
```

Name	Цена в USD	Цена в EUR	Цена в RUB
Рассказы	2.66	2.18	149.89
Роман	5.19	4.25	292.57
Стихи	1.82	1.49	102.33
Проза	3.47	2.84	195.43
Повесть	4.27	3.49	240.68

Рисунок 4 – Результат выполнения запроса

### Объединение нескольких таблиц в источнике выборки

Большинство запросов обращены к двум, трем или более таблицам. Запросы, которые позволяют это сделать, в SQL называются объединениями. В основе этого механизма лежит теория множеств позволяющая найти в разных множествах соответствующие объекты. Этот механизм в языке SQL реализует оператор JOIN.

Разберём пример. Необходимо вывести название книги, издательство и город где она издана. В нашей БД таблицы соединяются с помощью внутреннего объединения с использованием своих внешних ключей.

Рассмотрим конструкцию такого объединения.

Блок 1 на рисунке – это объединение двух таблиц tab\_Book и spr\_Publish. Результат объединения также является таблицей, содержащей все поля из исходных объединяемых таблиц.

Блок 2 на рисунке – это конструкция объединения таблицы spr\_City с объединением, полученным в блоке 1.

В результате получим следующий запрос на выборку:

```
SELECT tab_Book.Name, spr_Publish.Name, spr_City.Name
FROM
```

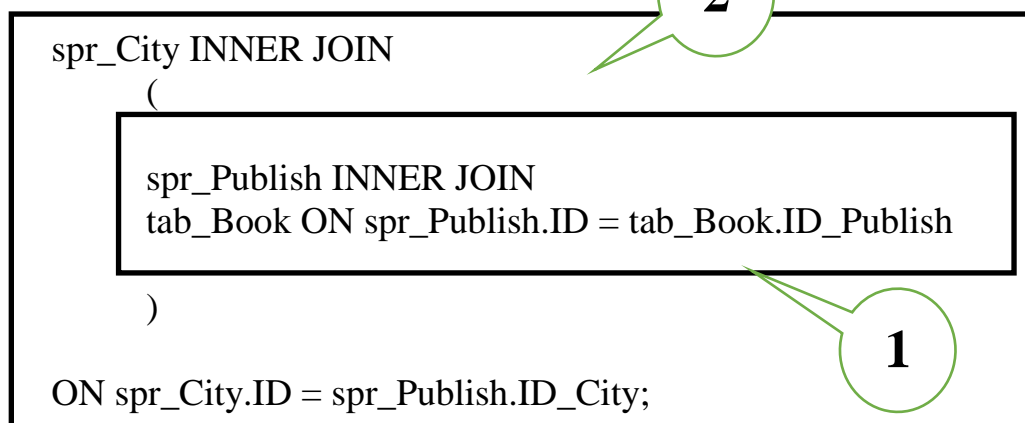


Рисунок 5 – Структура запроса – объединения.

tab_Book.Name	spr_Publish.Name	spr_City.Name
Рассказы	Мир книги	Минск
Роман	Книжный пресс	Москва
Проза	Книжный пресс	Москва
Стихи	Опечатка	Витебск
Повесть	Опечатка	Витебск

Рисунок 6 – Результат выполнения запроса

Часто при объединении в результирующей таблице появляются повторяющиеся записи. Для их исключения используем ключ DISTINCT:

```
SELECT DISTINCT <список_вывода> FROM <источник_выборки>
```

## Группировка

Группировка данных позволяет разделить все данные на логические наборы, благодаря чему становится возможным выполнение статистических вычислений отдельно в каждой группе. Для объединения результатов выборки по одному или нескольким столбцам используется оператор SQL GROUP BY.

Поясним работу этого оператора на примере выборки средней цены книги по издательству.

```
SELECT spr_Publish.Name as [Издательство],
       Avg(tab_Book.Cena) as [Средняя цена]
```

```
FROM spr_Publish INNER JOIN tab_Book
    ON spr_Publish.ID = tab_Book.ID_Publish
GROUP BY spr_Publish.Name;
```

Выполним группировку по названию издательства, а среднюю цену определим с помощью функции Avg.

В результате работы запроса получим таблицу с данными:

Издательство	Средняя цена
Книжный пресс	8,465
Мир книги	5,2
Опечатка	5,95

Рисунок 6 – Результат выполнения запроса

### Фильтрация сгруппированной таблицы

Так же, как для фильтрации строк в таблице, можно осуществить фильтрацию по сгруппированным данным. Для этого в языке SQL используют оператор HAVING

Возьмем предыдущий пример и добавим фильтрацию по группам, для которых средняя цена меньше восьми.

```
SELECT spr_Publish.Name AS Издательство,
    Avg(tab_Book.Cena) AS [Средняя цена]
FROM spr_Publish INNER JOIN tab_Book
    ON spr_Publish.ID = tab_Book.ID_Publish
GROUP BY spr_Publish.Name
HAVING Avg(tab_Book.Cena)<8;
```

Выполнив запрос, получим следующую таблицу:

Издательство	Средняя цена
Мир книги	5,2
Опечатка	5,95

Рисунок 7 – Результат выполнения запроса

## Простые подзапросы

**Подзапрос** — это запрос на выборку данных, вложенный в другой запрос. Простые подзапросы характеризуются тем, что они формально никак не связаны с содержащими их внешними запросами. Это позволяет сначала выполнить подзапрос, результат которого затем используется для выполнения внешнего запроса.

Простые подзапросы можно разделить на три категории:

- подзапросы, возвращающие единственное значение;

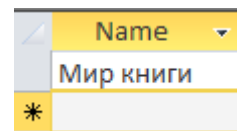
Тип возвращаемой подзапросом таблицы определяет, как можно ее использовать и какие операторы можно применять. Сам подзапрос действует как временная таблица, областью видимости для которой является выражение. В качестве примера используем подзапрос для связи двух таблиц.

```
SELECT Name
```

```
FROM spr_Publish
```

```
WHERE ID_City=(SELECT ID FROM spr_City where Name='Минск');
```

Выбрав с помощью подзапроса первичный ключ из справочника `spr_City` по внешнему ключу `ID_City` таблицы `spr_Publish` выбираем нужные данные.



Name
Мир книги
*

Рисунок 8 – Результат выполнения запроса

- подзапросы, возвращающие список значений из одного столбца таблицы;

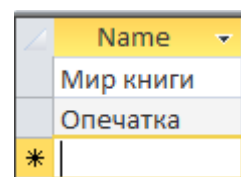
Подзапрос может возвращать и несколько записей. Чтобы в этом случае в условии `WHERE` внешнего запроса можно было использовать операторы сравнения, требующие единственного значения, используются кванторы, такие как `ALL` (все) и `SOME` или `ANY` (некоторый).

В примере показано как с помощью подзапроса связать две таблицы.

```
SELECT Name FROM spr_Publish
```

```
WHERE ID_City=
```

```
SOME(SELECT ID FROM spr_City WHERE  
Country="РБ");
```



Name
Мир книги
Опечатка
*

Рисунок 9 – Результат выполнения запроса



В качестве оператора сравнения может быть использован любой допустимый оператор сравнения. Данный запрос должен вернуть все записи таблицы spr\_Publish, для которых значение в поле ID\_City равно хотя бы для какого-нибудь одного значения поля ID справочника spr\_City.

Запросы с квантором ALL имеет аналогичную структуру. Они должны возвращать список всех значений, для которых оператор сравнения истинен для любого значения из списка возвращаемого подзапросом.

### Подзапрос в качестве источника выборки

Подзапрос можно вставлять в оператор FROM. При этом подзапрос будет выступать в качестве источника данных. В таком запросе таблице, возвращаемой подзапросом в операторе FROM, можно присвоить псевдоним, а во внешнем запросе использовать данные этой таблицы, используя псевдоним.

```
SELECT (Tab.Name+ Tab.Tel) as Визитка
```

```
FROM(SELECT * FROM spr_Author WHERE (Tel like "*(29)*")) Tab;
```

Результат выборки данного запроса имеет вид:

Визитка	
Иванов И.В.	+375(29)322-33-44
Пупкин С.П.	+375(29)761-20-89
*	

Рисунок 10 – Результат выполнения запроса

### Соотнесённые подзапросы

Подзапрос также можно использовать в секции Where. Здесь в отличие от предыдущих вариантов связанный (коррелированный) подзапрос выполняется каждый раз для каждой строки основного запроса.

Например, следующий запрос использует связанный подзапрос для подсчета количества публикаций у каждого автора. Затем основной запрос выбирает тех авторов, у которых больше одной публикации.

```
SELECT a.Name  
FROM spr_Author a  
WHERE 1<(SELECT COUNT(*) FROM [tab_Book] b WHERE b.ID_Author =  
a.Id);
```

Name
Иванов И.В.
Петрова О.С
*

Рисунок 11 – Результат выполнения запроса

Ссылка на a.Id в самом конце подзапроса - это то, что делает этот подзапрос связанным. Чтобы подзапрос мог выполняться, основной запрос должен поставлять значения для a.Id. В данном случае основной запрос извлекает из таблицы spr\_Author все строки и выполняет по одному подзапросу для всех авторов, передавая в него соответствующий Id автора при каждом выполнении. Если подзапрос возвращает значение большее одного, условие фильтрации выполняется и строка добавляется в результирующий набор.

### Использование объединения UNION

Результирующая таблица, полученная в результате выполнения запроса с использованием объединения UNION, будет представлять собой вертикальное объединение двух и более запросов (т.е. объединение записей).

Основные требования к запросам, входящим в объединение UNIT:

- Результирующие таблицы запросов должны иметь одинаковое количество полей;
- Соответствующие поля результирующих таблиц должны быть совместимы для объединения, т. е. иметь совместимый тип.

В качестве примера выведем сведения о книгах, расположив их в три ценовые категории: 'эконом', 'бюджет', 'супер' с ценами до 5 и свыше 10 соответственно.

```
SELECT Name, Cena, 'эконом' AS Статус
FROM tab_Book WHERE Cena<5
UNION
SELECT Name, Cena, 'бюджет' AS Статус
FROM tab_Book WHERE Cena>=5 and Cena<=10
UNION
SELECT Name, Cena, 'супер' AS Статус
FROM tab_Book WHERE Cena>10
ORDER BY Cena
```

Обратите внимание на поле «Статус»: каждый из запросов содержит в нём соответствующую константу.

Раздел ORDER BY в данном случае используется для сортировки сразу всех записей результирующей таблицы.

Итоговая результирующая таблица выглядит так:

Name	Цена	Статус
Стихи	3,55	эконом
Рассказы	5,2	бюджет
Проза	6,78	бюджет
Повесть	8,35	бюджет
Роман	10,15	супер

Рисунок 12 – Результат выполнения запроса

## СУБД MS SQL Server

MS SQL Server является клиент-серверной СУБД. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия, а также для web-приложений.

### Разработка скрипта создания базы данных

В качестве стандартного языка управления базами данных в MS SQL Server используется язык Transact SQL (T-SQL). Основное его отличие от диалекта Jet-SQL состоит в возможности пакетного исполнения команд. Такой пакет команд называют скриптом. В языке T-SQL также имеются управляющие конструкции (циклы, ветвления), с помощью которых можно организовать команды в определённый алгоритм.

Непосредственно для создания базы данных используется команда:

```
CREATE DATABASE <имя_базы_данных>
```

Создадим новую базу данных PubDep\_DB с помощью команды

```
CREATE DATABASE PubDep_DB
```

Рассмотрим пример скрипта, добавляющего группу связанных таблиц в уже существующую базу данных. Напомним, что для создания таблиц используется команда CREATE TABLE, а для удаления DROP TABLE.

Таблицу можно рассматривать как компонент, состоящий из полей, ограничений и триггеров. Ограничения – это специальные объекты базы данных, связанные с таблицами, предназначенные для создания первичных и внешних ключей, а также для задания выражений, контролируемых вводимые в таблицу данные (ограничения CHECK в данных указаниях не рассматриваются).

С помощью команды CREATE TABLE создаются поля (часть команды до ключевого слова CONSTRAINT) и ограничения таблицы (часть команды после ключевого слова CONSTRAINT).

Следует обратить внимание на то, что внешние ключи могут быть созданы непосредственно при описании поля (конструкция REFERENCES). В этом случае также создаётся ограничение, но такой способ его создания называют неявным. Несмотря на различные способы описания, каждое из создаваемых ограничений является отдельным, связанным с таблицей, объектом. Имена для неявно описанных ограничений задаются самим ядром СУБД.

Для удобства отладки необходимо включать в скрипт команды очистки базы данных от ранее созданных объектов, которые могли остаться от предыдущих запусков скрипта. Такая процедура обычно содержит проверку наличия объекта и команду его удаления, в случае, если он находится в базе данных.

Для проверки наличия объекта используется системная таблица «sysobjects», в которой имеется по одной записи для каждого объекта базы данных (таблицы, хранимой процедуры, ограничения, триггера и т. д.).

В таблице приведено описание некоторых наиболее полезных полей системной таблицы «sysobjects».

Таким образом, для запуска скрипта не потребуется вручную подготавливать базу данных.

Таблица 5 Поля таблицы «sysobjects»

Поле	Тип данных	Описание
1	2	3
name	nvarchar(128)	Имя объекта
Id	Int	Идентификатор объекта

Окончание таблицы 5

1	2	3
type	char(2)	Тип объекта. Может принимать одно из значений, идентифицирующих тип объекта: C = ограничение CHECK, D = ограничение DEFAULT, F = ограничение FOREIGN KEY, K = ограничение PRIMARY KEY, P = хранимая процедура, S = системная таблица, TR = триггер, U = пользовательская таблица, V = представление.
parent_obj	Int	Идентификатор хозяина объекта. Например: хозяином триггера является таблица, для которой он был создан.
crdate	datetime	Дата и время создания объекта.

В качестве примера разработаем скрипт создания структуры базы данных, на основе даталогической модели разработанной ранее.

```
USE PubDep_DB
```

```
GO
```

```
if exists ( SELECT name FROM sysobjects
```

```
WHERE name = 'tab_Book' AND type = 'U' )
```

```
DROP TABLE tab_Book
```

```
GO
```

```
if exists ( SELECT name FROM sysobjects
```

```
WHERE name = 'spr_Author' AND type = 'U' )
```

```
DROP TABLE spr_Author
```

```
GO
```

```
if exists ( SELECT name FROM sysobjects
```

```

        WHERE name = 'spr_Publish' AND type = 'U' )

DROP TABLE spr_Publish

GO

if exists ( SELECT name FROM sysobjects

        WHERE name = 'spr_City' AND type = 'U' )

DROP TABLE spr_City

GO


CREATE TABLE spr_City (ID int, Name char (50), Country char (3),
PRIMARY KEY (ID));

GO

CREATE TABLE spr_Author (ID int, Name char (50), Tel char (25),
PRIMARY KEY (ID));

GO

CREATE TABLE spr_Publish (ID int, Name char (50), ID_City Int,
PRIMARY KEY (ID),
FOREIGN KEY (ID_City) REFERENCES spr_City (ID));

GO

CREATE TABLE tab_Book (ID int, Name char (50), Cena float,
        ID_Author Int, ID_Publish Int,
PRIMARY KEY (ID),
FOREIGN KEY (ID_Publish) REFERENCES spr_Publish (ID),
FOREIGN KEY (ID_Author) REFERENCES spr_Author (ID));

GO

```

В первую очередь используем команду USE PubDep\_DB для подключения к базе данных. Это избавляет от необходимости следить за тем, какая база данных в данный момент является активной.

Нужно обратить внимание на последовательность размещения команд удаления и создания таблиц.

Для успешного выполнения скрипта необходимо сначала удалить таблицу с которой связаны ограничения, ссылающиеся на другие таблицы.

Однако имеются и другие варианты. Если вначале скрипта удалить все связанные с таблицами объекты (в том числе и ограничения), то последовательность удаления таблиц уже не будет играть никакой роли.

Аналогичным образом создадим скрипт, заполняющий таблицы базы данных информацией на основе созданных ранее запросов использующих команду INSERT INTO.

Пример подобного скрипта приведен ниже

```
USE PubDep_DB
```

```
GO
```

```
INSERT INTO spr_City
```

```
(ID, Name, Country)
```

```
VALUES (1,'Витебск','РБ'),
```

```
      (2,'Минск','РБ'),
```

```
      (3,'Москва','РФ');
```

```
GO
```

```
INSERT INTO spr_Author
```

```
(ID, Name, Tel)
```

```
VALUES (1,' Иванов И.В.', '+375(29)322-33-44'),
```

```
      (2,' Петрова О.С ', '+375(33)524-12-15'),
```

```
      (3,' Пупкин С.П.', '+375(29)761-20-89');
```

```
GO
```

```
INSERT INTO spr_Publish
```

```
(ID, Name, ID_City)
```

```
VALUES (1,'Мир книги', 2),
       (2,'Книжный пресс', 3),
       (3,'Опечатка', 1);
```

GO

```
INSERT INTO tab_Book
```

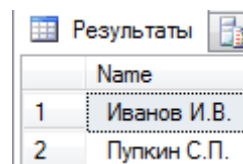
```
(ID , Name, Cena, ID_Author, ID_Publish)
```

```
VALUES (1,'Рассказы', 5.20, 1, 1),
       (2,'Роман', 10.15, 1, 2),
       (3,'Стихи', 3.55, 2, 3),
       (4,'Проза', 6.78, 2, 2),
       (5,'Повесть', 8.35, 3, 3);
```

GO

Для тестирования созданной с помощью скриптов базы данных используем разработанные ранее запросы.

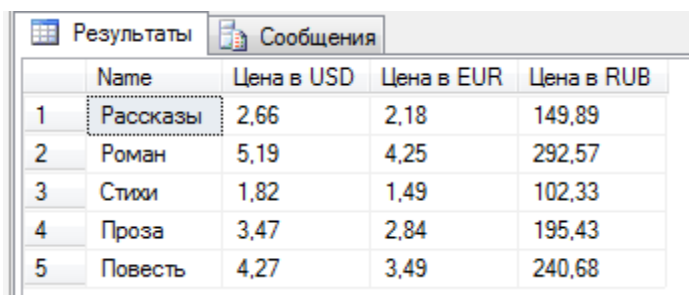
```
SELECT Name
FROM PubDep_DB.dbo.spr_Author
WHERE (Tel LIKE '%(29)%');
```



Результаты	
	Name
1	Иванов И.В.
2	Пупкин С.П.

Рисунок 13 – Результат выполнения запроса

```
SELECT Name, round(Cena/1.9556, 2) as [Цена в USD],
       round(Cena/2.3897, 2) as [Цена в EUR],
       round(Cena/3.4693*100, 2) as [Цена в RUB]
FROM tab_Book
```



Результаты		Сообщения		
	Name	Цена в USD	Цена в EUR	Цена в RUB
1	Рассказы	2,66	2,18	149,89
2	Роман	5,19	4,25	292,57
3	Стихи	1,82	1,49	102,33
4	Проза	3,47	2,84	195,43
5	Повесть	4,27	3,49	240,68

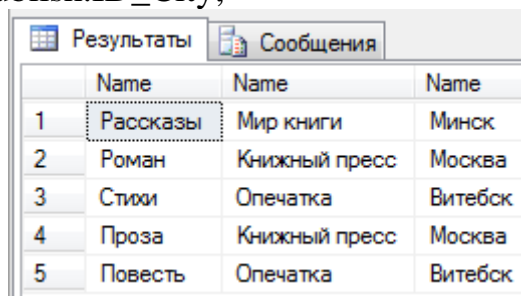
Рисунок 14 – Результат выполнения запроса



```

SELECT tab_Book.Name, spr_Publish.Name, spr_City.Name
FROM spr_City INNER JOIN (spr_Publish INNER JOIN
    tab_Book ON spr_Publish.ID = tab_Book.ID_Publish)
ON spr_City.ID = spr_Publish.ID_City;

```



	Name	Name	Name
1	Рассказы	Мир книги	Минск
2	Роман	Книжный пресс	Москва
3	Стихи	Опечатка	Витебск
4	Проза	Книжный пресс	Москва
5	Повесть	Опечатка	Витебск

Рисунок 14 – Результат выполнения запроса

Аналогичным образом следует выполнить все запросы на выборку разработанные ранее. Если результат выполнения запросов аналогичен предыдущим результатам, то делаем вывод, что и базы данных аналогичны, а тестирование прошло успешно.

## Задания

Разработать систему, использующую БД и отвечающую следующим требованиям:

1. Создать логическую модель БД, проведя анализ функциональных зависимостей атрибутов отношений, определить ключи, обосновывая их выбор, указать связи между отношениями, провести нормализацию отношений до 3НФ. (схема БД должна быть в пояснительной записке)
2. Разработать таблицы (не менее 5), указав типы данных, ограничения, выдачу сообщений о некорректном вводе, создание списков для атрибутов, подстановку данных для подчиненных таблиц

Создать запрос с использованием возможностей:

- фильтрации
- вычисляемого поля результирующей таблицы
- объединения нескольких таблиц в источнике выборки
- Группировки
- Фильтрация сгруппированной таблицы
- подзапроса, возвращающего единственное значение
- подзапроса в качестве источника выборки
- соотнесённых подзапросов
- Использование объединения UNION

Таблицы 6 – Таблица тем для БД по вариантам.

ВАРИАНТ	Название базы данных
1	2
1.	СТРАХОВАЯ КОМПАНИЯ
2.	ГОСТИНИЦА
3.	Ломбард
4.	Реализация готовой продукции
5.	Бюро по трудоустройству
6.	Нотариальная контора
7.	Фирма по продаже запчастей
8.	Курсы повышения квалификации

Окончание таблицы 6

1	2
9.	ТУРИСТИЧЕСКАЯ ФИРМА
10.	ГРУЗОВЫЕ ПЕРЕВОЗКИ
11.	УЧЕТ ТЕЛЕФОННЫХ ПЕРЕГОВОРОВ
12.	УЧЕТ ВНУТРИОФИСНЫХ РАСХОДОВ
13.	Библиотека
14.	ПРОКАТ АВТОМОБИЛЕЙ
15.	СДАЧА В АРЕНДУ ТОРГОВЫХ ПЛОЩАДЕЙ
16.	ПЛАТНАЯ ПОЛИКЛИНИКА
17.	Общеобразовательная школа
18.	УЧЕТ ТЕЛЕКОМПАНИЕЙ СТОИМОСТИ ПРОШЕДШЕЙ В ЭФИРЕ РЕКЛАМЫ
19.	Парикмахерская
20.	Распределение дополнительных обязанностей
21.	Техническое обслуживание станков
22.	ВЕДЕНИЕ ЗАКАЗОВ
23.	ИНТЕРНЕТ-МАГАЗИН
24.	ЮВЕЛИРНАЯ МАСТЕРСКАЯ
25.	РАСПРЕДЕЛЕНИЕ УЧЕБНОЙ НАГРУЗКИ
26.	ЗАНЯТОСТЬ АКТЕРОВ ТЕАТРА
27.	ИНВЕСТИРОВАНИЕ СВОБОДНЫХ СРЕДСТВ
28.	КОМБИНАТ ПИТАНИЯ
29.	АВТОПАРК
30.	СУДОХОДСТВО

Более подробно с заданием по варианту студент может ознакомиться на электронном ресурсе [sdo.vstu.by](http://sdo.vstu.by) для предмета «Базы данных» специальности 1-40 05 01-01 «Информационные системы и технологии»

## Список рекомендуемых литературных источников и веб-ресурсов

1. Дейт, К. Введение в системы баз данных / К. Дейт. – 6-издание. – Киев : Диалектика, 1998. – 784 с.
2. Грабер, М. Справочное руководство по SQL / М. Грабер. – Москва : Лори, 1997. – 291 с.
3. Клайн К. SQL. Справочник. 2-е издание. – Москва : КУДИЦ-ОБРАЗ, 2006 – 832 с.
4. Материалы сайта <http://office.microsoft.com/ru-ru/access/>
5. Материалы сайта [www.microsoft.com/sqlserver/](http://www.microsoft.com/sqlserver/)
6. Артёмов, Д. В. Microsoft SQL Server 7.0: установка, управление, оптимизация / Д. В. Артёмов, Г. В. Погульский. – Москва : Издательский отдел «Русская редакция», 1998. – 488 с.
7. Бойко, В.В. Проектирование баз данных информационных систем. / В. В. Бойко, В. М. Савинков. – Москва : Финансы и статистика, 1989. – 351 с.
8. Кириллов, В. В. Структурированный язык запросов (SQL) / В. В. Кириллов. – Санкт-Петербург : ИТМО, 1994. – 80 с.
9. Мейер, М. Теория реляционных баз данных / М. Мейер. – Москва : Мир, 1987 – 608 с.
10. Microsoft Access 2002. Русская версия. Шаг за шагом : практ. пособ. / пер. с англ. – Москва : Издательство ЭКОМ, 2002. – 352 с.
11. Материалы сайта [https://studopedia.su/13\\_104185\\_podzaprosi-vozvrashchayushchie-edinstvennoe-znachenie.html](https://studopedia.su/13_104185_podzaprosi-vozvrashchayushchie-edinstvennoe-znachenie.html)

Учебное издание

## БАЗЫ ДАННЫХ

Методические указания по выполнению расчетно-графической работы

Составители:

**Казаков Вадим Евгеньевич**  
**Черненко Дмитрий Владимирович**

Редактор *Н.В. Медведева*  
Корректор *Н.В. Медведева*  
Компьютерная верстка *Т.Г. Трусова*

---

Подписано к печати \_\_\_\_\_. Формат \_\_\_\_\_. Усл. печ. листов \_\_\_\_\_.  
Уч.-изд. листов \_\_\_\_\_. Тираж \_\_\_\_\_ экз. Заказ № \_\_\_\_\_.

Учреждение образования «Витебский государственный технологический университет»  
210035, г. Витебск, Московский пр., 72.

Отпечатано на ризографе учреждения образования

«Витебский государственный технологический университет».

Свидетельство о государственной регистрации издателя, изготовителя,  
распространителя печатных изданий № 1/172 от 12 февраля 2014 г.

Свидетельство о государственной регистрации издателя, изготовителя,  
распространителя печатных изданий № 3/1497 от 30 мая 2017 г.