МИНИСТЕРСТВО НАУКИ ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент информационных и компьютерных систем

Методические указания к выполнению лабораторных работ «Основы SQL» по дисциплине «Разработка баз данных»

по направлению (09.03.03) «Прикладная информатика»

Владивосток
2022

Одобрено научно-методическим советом департамента
УДК 004.65(076)
ББК 16.3р30-2я73
К 785

Методические указания по выполнению лабораторных работ «Основы SQL» по дисциплине «Разработка баз данных» разработаны в соответствии с требованиями федерального государственного образовательного стандарта высшего образования по направлению подготовки 09.03.03 Прикладная информатика (уровень бакалавриата). В методических указаниях содержится краткое описание операторов SQL, технология выполнения лабораторных работ по данной тематике.

Методические указания обсуждены на заседании департамента «Компьютерных и информационных системы», протокол № 5 от «19» января 2022 г.

Директор департамента д.т.н., профессор Пустовалов Е.В.

Составили: Красюк Л.В., ст. преподаватель Департамента ИКС
Бедрина С.Л., к.э.н., доцент Департамента ИКС
Елсукова Е.А., ст. преподаватель Департамента ИКС

ОГЛАВЛЕНИЕ

1 ЦЕЛИ И ЗАДАЧИ ЛАБОРАТОРНОЙ РАБОТЫ

Цель лабораторной работы:

1. Изучить основные операторы языка SQL.
2. Используя конструкции языка SQL создать базу данных, таблицы базы данных, представления.

ЗАДАНИЕ

1) На основе анализа предметной области и даталогической модели БД выполнить физическое проектирование базы данных в СУБД MS SQL.

2) Используя операторы манипулирования данными создать представления для решения пользовательских задач.

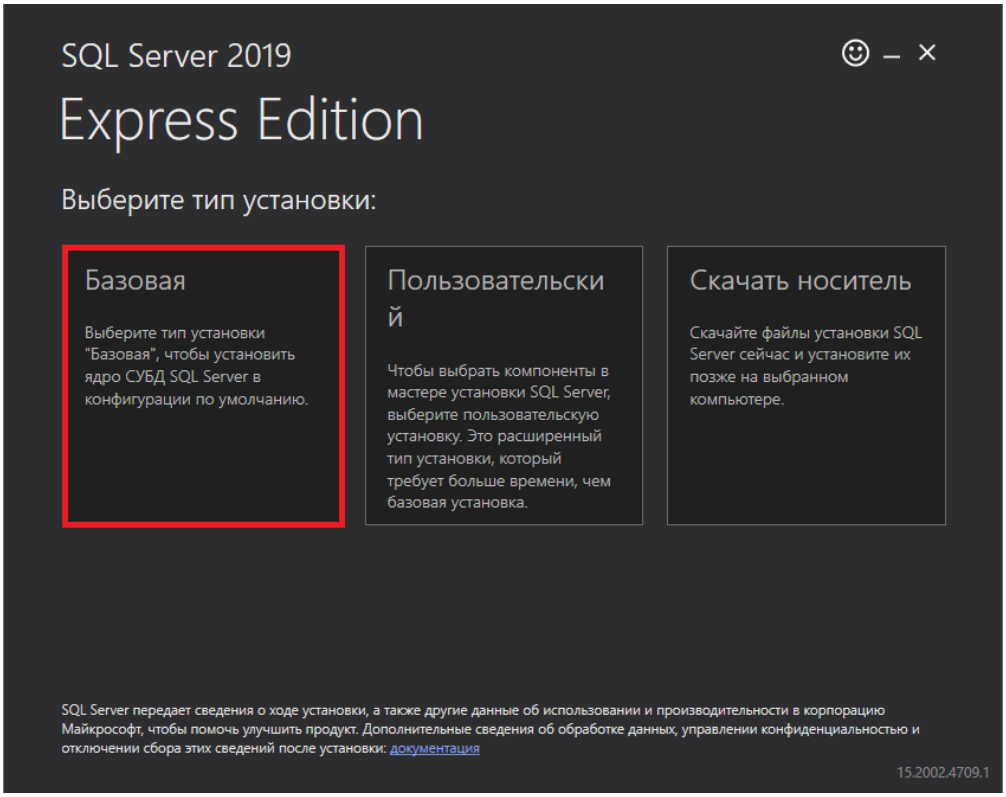
ХОД РАБОТЫ

1. Осуществить установку MS SQL Server Express.
2. Создать базу данных computer.
3. С помощью операторов SQL реализовать алгоритмы решения задач.
4. Подготовить отчет о проделанной работе. Структура отчета:

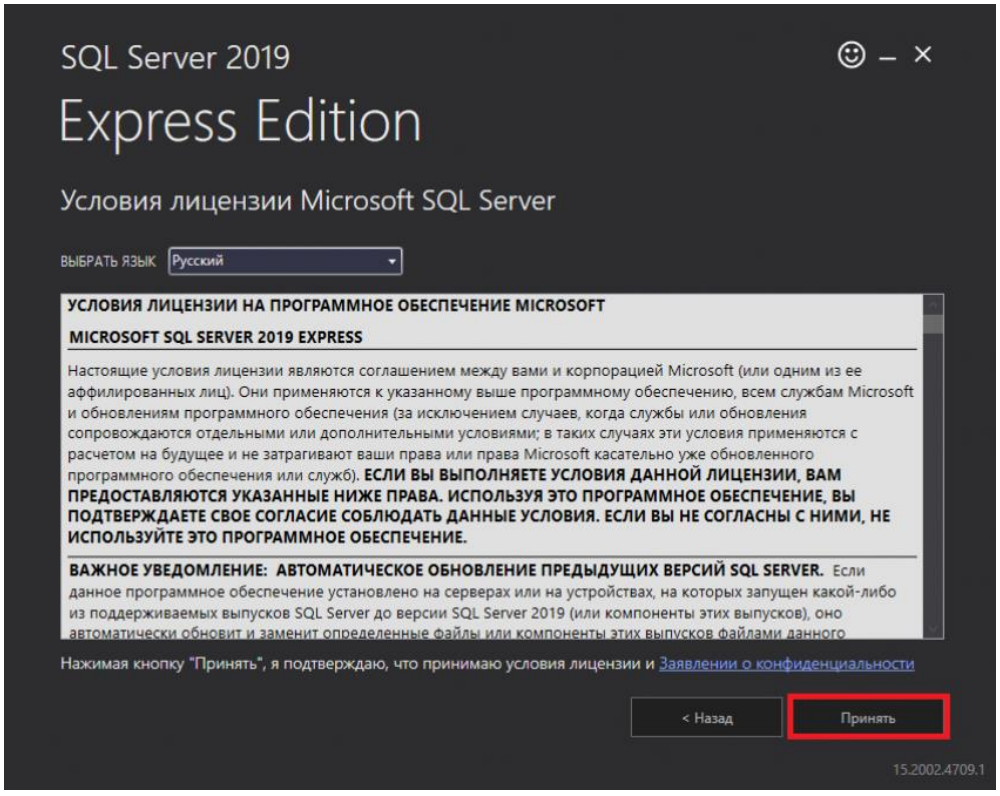
- титульный лист;
- задание;
- описание процесса создания БД; алгоритмы решения задач;
- заключение.

2 ИНСТРУКЦИИ ПО УСТАНОВКЕ

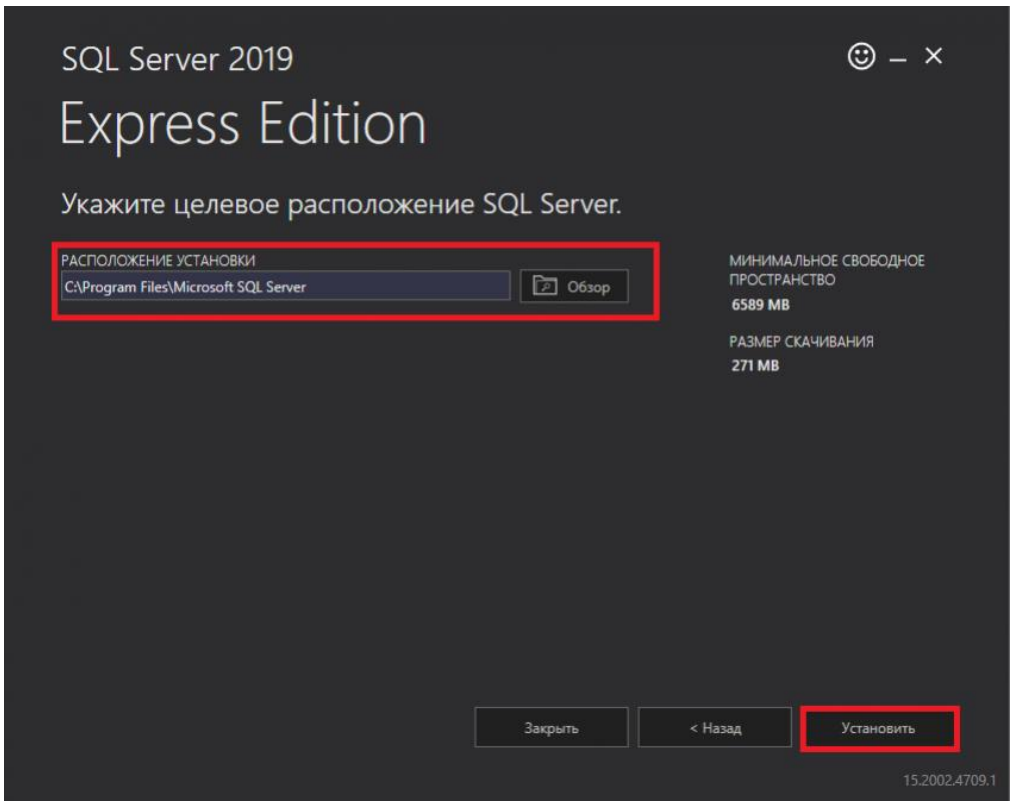
1. Скачать актуальную версию SQL Server Express Вы можете на официальном сайте Microsoft (бесплатно).
2. После скачивания необходимо запустить исполняемый файл.
3. Выбрать тип установки БАЗОВАЯ.



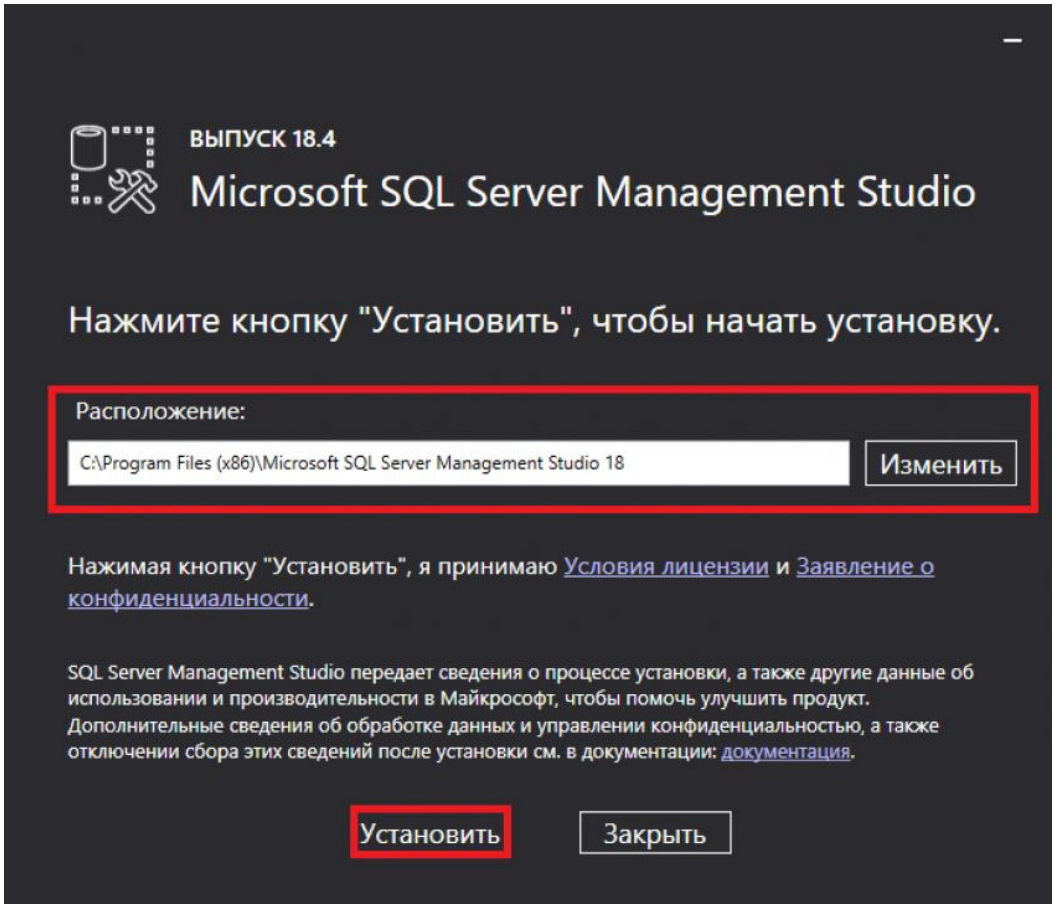
4. Принять Условия лицензии.



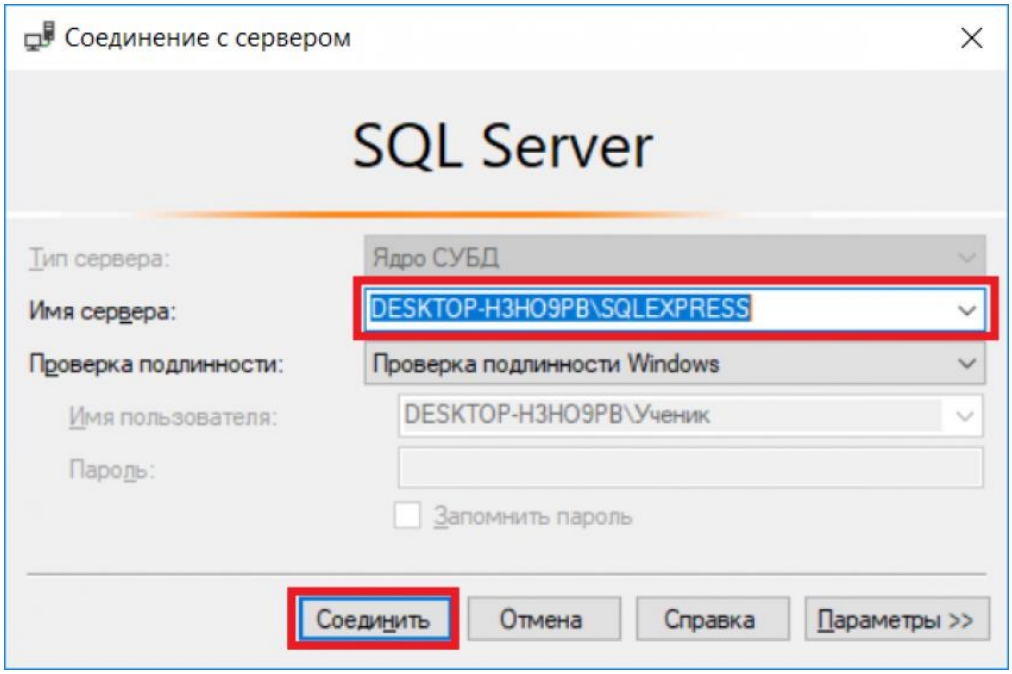
5. Указать целевое расположение SQL Server и нажать на кнопку «Установить».



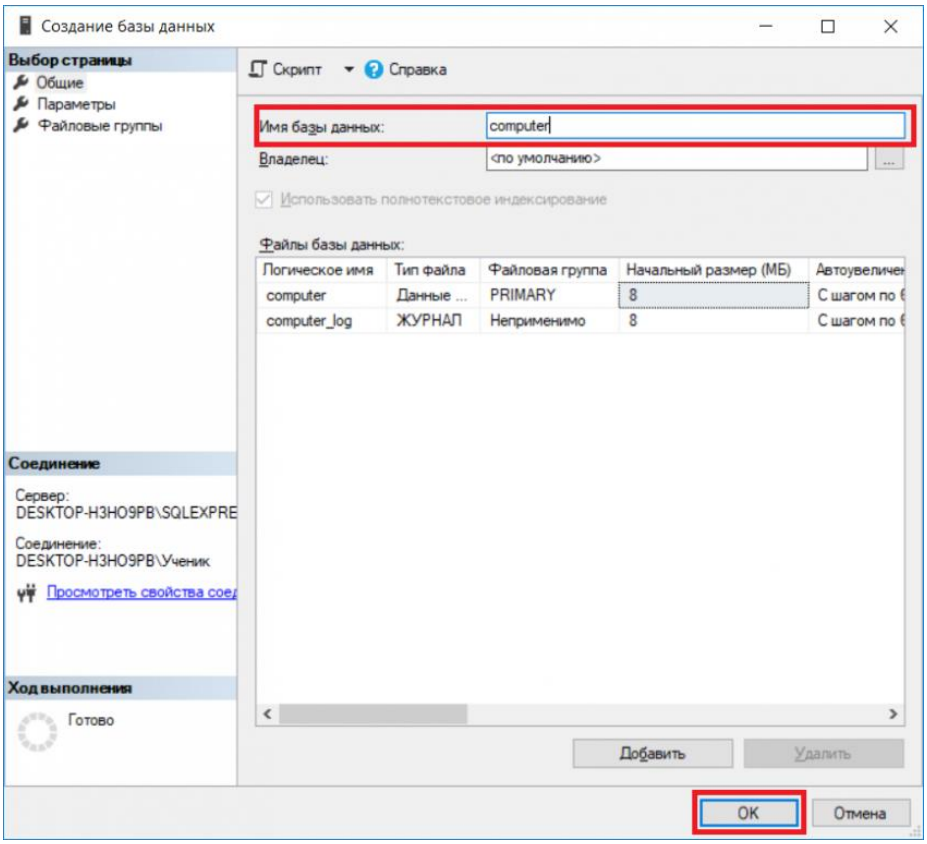
6. Скачать актуальную версию SQL Server Management Studio Вы можете на официальном сайте Microsoft (бесплатно).
7. После скачивания необходимо запустить исполняемый файл.
8. Нажать кнопку «Установить».



9. После установки необходимо запустить SSMS и ввести имя локального сервера. Имя сервера вводится по принципу: **<имя_компьютера>**\SQLEXPRESS.



10. После этого необходимо создать несколько пустых баз данных. Кликнуть правой кнопкой мыши по «Базы данных» в дереве ядра СУБД, после этого выбрать пункт «Создать базы данных».



11. Таким образом необходимо создать базу данных:

1. computer (**База данных «Компьютерная фирма»)**

Схема БД состоит из четырех таблиц (рис.1):

1. Product(maker, model, type)

2. PC(code, model, speed, ram, hd, cd, price)
3. Laptop(code, model, speed, ram, hd, screen, price)
4. Printer(code, model, color, type, price)

Таблица Product представляет производителя (maker), номер модели (model) и тип (PC - ПК, Laptop - портативный компьютер или Printer - принтер). Предполагается, что в этой таблице номера моделей уникальны для всех производителей и типов продуктов. В таблице PC для каждого номера модели, обозначающего ПК, указаны скорость процессора - speed (МГерц), общий объем оперативной памяти - ram (Мбайт), размер диска - hd (в Гбайт), скорость считывающего устройства - cd (например, '4x') и цена - price. Таблица Laptop аналогична таблице PC за исключением того, что вместо скорости CD-привода содержит размер экрана - screen (в дюймах). В таблице Printer для каждой модели принтера указывается, является ли он цветным - color ('y', если цветной), тип принтера - type (лазерный - Laser, струйный - Jet или матричный - Matrix) и цена - price.

Схема данных

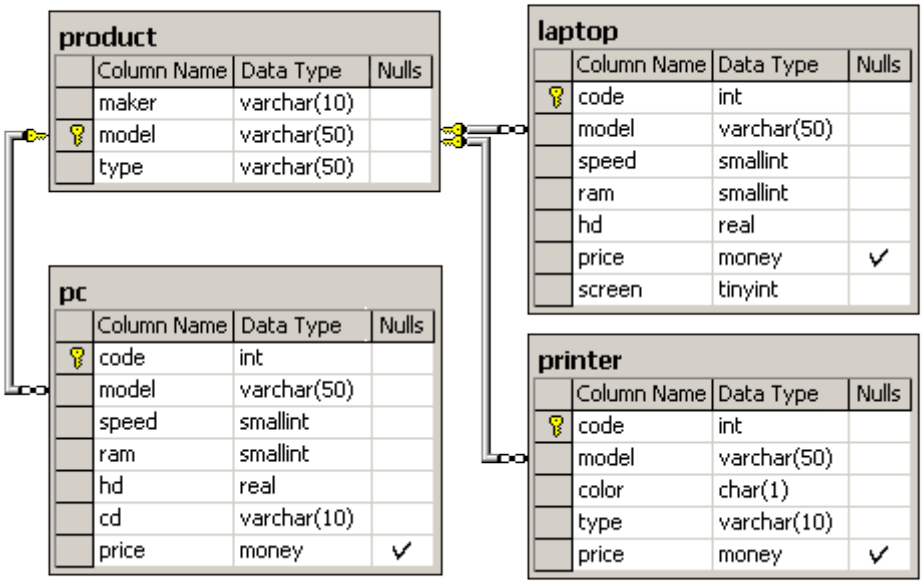


Рисунок 1 - Схема базы данных «Компьютерная фирма»

Дополнительную информацию можно извлечь из представленной на рис. 1 логической схемы данных. Таблицы по типам продукции (ПК, портативные компьютеры и принтеры) содержат внешний ключ (model) к таблице Product. Связь «один-ко-многим» означает, что в каждой из этих таблиц может отсутствовать модель, имеющаяся в таблице Product. С другой стороны, модель с одним и тем же номером может встречаться в такой таблице несколько раз, причем даже с полностью идентичными техническими характеристиками, так как первичным ключом здесь является столбец code. Последнее требует пояснения, так как разные люди вкладывают в понятие модели разный смысл. В рамках данной схемы считается, что модель — это единство производителя и технологии. Например, одинаковые модели могут комплектоваться технологически идентичными накопителями, но разной емкости, скажем, 60 и 80 Гбайт. В частности, это означает, что допустимо присутствие в таблице PC двух ПК с одинаковыми номерами модели, но по разной цене.

На языке предметной области данная схема может означать, что в таблице Product содержится информация обо всех известных поставщиках рассматриваемой продукции и моделях, которые они поставляют, а в остальных таблицах находятся имеющиеся в наличии

(или продаже) модели. Поэтому вполне возможна ситуация, когда имеется поставщик (maker) с моделями, ни одной из которых нет в наличии.

3 ОСНОВЫ ЯЗЫКА SQL

SQL является инструментом, предназначенным для обработки и чтения данных, содержащихся в компьютерной базе данных. SQL – это сокращенное название структурированного языка запросов *(Structured Query Language)*. SQL основывается на реляционной алгебре.

SQL в его чистом (базовом) виде является информационно-логическим языком, а не языком программирования. Вместе с тем стандарт языка спецификацией SQL/*PSM* предусматривает возможность его процедурных расширений, с учетом которых язык уже вполне может рассматриваться в качестве языка программирования.

SQL используется для реализации всех функциональных возможностей, которые СУБД предоставляет пользователю, а именно:

- *Организация данных*. SQL дает пользователю возможность изменять структуру представления данных, а также устанавливать отношения между элементами базы данных.
- *Чтение данных*. SQL дает пользователю или приложению возможность читать из базы данных содержащиеся в ней данные и пользоваться ими.
- *Обработка данных*. SQL дает пользователю или приложению возможность изменять базу данных, т.е. добавлять в нее новые данные, а также удалять или обновлять уже имеющиеся в ней данные.
- *Управление доступом*. С помощью SQL можно ограничить возможности пользователя по чтению и изменению данных и защитить их от несанкционированного доступа.
- *Совместное использование данных*. SQL координирует совместное использование данных пользователями, работающими параллельно, чтобы они не мешали друг другу.
- *Целостность данных*. SQL позволяет обеспечить целостность базы данных, защищая ее от разрушения из-за несогласованных изменений или отказа системы.

Таким образом, SQL является достаточно мощным языком для взаимодействия с СУБД.

Каждый оператор SQL начинается с глагола, представляющего собой ключевое слово, определяющее, что именно делает этот оператор (SELECT, INSERT, DELETE...). В операторе содержатся также предложения, содержащие сведения о том, над какими данными производятся операции. Каждое предложение начинается с ключевого слова, такого как FROM, WHERE и др. Структура предложения зависит от его типа – ряд предложений содержит имена таблиц или полей, некоторые могут содержать дополнительные ключевые слова, константы или выражения.

3.1 Операторы SQL

Основу языка SQL составляют операторы, условно разбитые не несколько групп по выполняемым функциям.

Можно выделить следующие группы операторов (перечислены не все операторы SQL):

Операторы DDL (Data Definition Language) - операторы определения объектов базы данных

Data Definition Language содержит операторы, позволяющие создавать, изменять и уничтожать базы данных и объекты внутри них (таблицы, представления и др.).

- CREATE SCHEMA - создать схему базы данных.
- DROP SHEMA - удалить схему базы данных.
- CREATE TABLE - создать таблицу.
- ALTER TABLE - изменить таблицу.
- DROP TABLE - удалить таблицу.
- CREATE DOMAIN - создать домен.
- ALTER DOMAIN - изменить домен.
- DROP DOMAIN - удалить домен.
- CREATE COLLATION - создать последовательность.
- DROP COLLATION - удалить последовательность.
- CREATE VIEW - создать представление.
- DROP VIEW - удалить представление.

Синтаксис некоторых операторов DDL представлен в таблице 1.

Таблица 1 - Основные операторы работы с базой данных

Оператор	**Синтаксис**
Оператор создания базы данных CREATE DATABASE	CREATE DATABASE имя_БД ON PRIMARY (NAME=логическое имя, FILENAME=’физическое имя (адрес и имя файла данных с расширением mdf)’, SIZE=размер, MAXSIZE=максимальный размер (UNLIMITED), FILEGROWTH=инкремент_увеличения_файлов) LOG ON (NAME=логическое имя, FILENAME=’ ’физическое имя (адрес и имя файла журнала транзакций с расширением ldf)’, SIZE=размер, MAXSIZE=максимальный размер (UNLIMITED), FILEGROWTH=инкремент_увеличения_файлов)
Оператор изменения базы данных ALTER DATABASE	ALTER DATABASE имя_базы_данных ADD FILE (NAME=логическое имя, FILENAME=’физическое имя’, SIZE=размер, MAXSIZE=максимальный размер (UNLIMITED), FILEGROWTH=инкремент_увеличения_файлов) TO FILEGROUP имя_группы_файлов
Оператор сжатия базы данных DBCC SHRINKDATABASE	DBCC SHRINKDATABASE (имя_базы_данных, процент_сжатия, (NOTRUNCATE (Место в базе данныхсвобождается за счет файлов данных. Освобожденное место отдается ОС. Параметр процент_сжатия игнорируется.)|TRUNCATEONLY (Все неиспользуемое

	место отдается в распоряжение системы. Параметр процент_сжатия игнорируется.))
Оператор удаления базы данных DROP DATABASE	DROP DATABASE имя_базы_данных

Операторы DML (Data Manipulation Language) - операторы манипулирования данными

Data Manipulation Language содержит операторы, позволяющие выбирать, добавлять, удалять и модифицировать данные. Обратите внимание на то, что эти операторы не обязаны завершать транзакцию, внутри которой они вызваны.

- SELECT - отобрать строки из таблиц.
- INSERT - добавить строки в таблицу.
- UPDATE - изменить строки в таблице.
- DELETE - удалить строки в таблице.
- COMMIT - зафиксировать внесенные изменения.
- ROLLBACK - откатить внесенные изменения.

Синтаксис операторов работы с данными представлен в таблице 2.

Таблица 2 - Основные операторы работы с данными

Оператор	Описание
SELECT	Применяется для выбора данных
INSERT	Применяется для добавления строк к таблице
DELETE	Применяется для удаления строк из таблицы
UPDATE	Применяется для изменения данных

Иногда оператор SELECT относят к отдельной категории, называемой Data Query Language (DQL).

Операторы защиты и управления данными

- CREATE ASSERTION - создать ограничение
- DROP ASSERTION - удалить ограничение
- GRANT - предоставить привилегии пользователю или приложению на манипулирование объектами
- REVOKE - отменить привилегии пользователя или приложения

Кроме того, есть группы операторов установки параметров сеанса, получения информации о базе данных, операторы статического SQL, операторы динамического SQL.

Наиболее важными для пользователя являются операторы манипулирования данными (DML).

3.2 Синтаксис оператора SELECT

Оператор SELECT является фактически самым важным для пользователя и самым сложным оператором SQL. Он предназначен для выборки данных из таблиц, т.е. он, собственно, и реализует одно их основных назначение базы данных - предоставлять информацию пользователю.

Оператор SELECT всегда выполняется над некоторыми таблицами, входящими в базу данных.

Большинство операторов SELECT описывают четыре главных свойства результирующего набора:

- столбцы, которые должны войти в результирующий набор;

- таблицу, из которой извлекаются данные для формирования результирующего набора;
- условия, которым должны соответствовать строки исходной таблицы, чтобы попасть в результирующий набор;
- последовательность упорядочения строк в результирующем наборе.

Оператор SELECT состоит из трех основных и дополнительных разделов и имеет следующий синтаксис:

```
SELECT список_столбцов, формул
FROM список_таблиц|список_представлений
WHERE условие_поиска
ORDER BY имя_столбца|список_номеров_столбцов (способ_сортировки (ASC|DESC))
GROUP BY (ALL) неагрегатное_выражение(я)
HAVING условие_поиска
```

Для предотвращения дублирования возвращаемых данных можно использовать ключевое слово DISTINCT. В этом случае оператор SELECT будет иметь следующий синтаксис:

```
SELECT DISTINCT список_столбцов, формул
FROM список_таблиц|список_представлений
WHERE условие_поиска
```

Равенство строк проверяется для всех столбцов из списка_столбцов. Значения NULL интерпретируются как одинаковые, поэтому возвращается только одна строка с NULL.

Оператор SELECT INTO

С помощью оператора SELECT INTO можно создать новую таблицу, основанную на результате выполнения запроса. Созданная таблица состоит из столбцов, перечисленных в списке столбцов оператора SELECT INTO.

```
SELECT список_столбцов, формул
INTO имя_новой_таблицы
FROM список_таблиц|список_представлений
WHERE условие_поиска
```

На самом деле в базах данных могут быть не только постоянно хранимые таблицы, а также временные таблицы и так называемые представления. Представления - это просто хранящиеся в базе данные SELECT-выражения. С точки зрения пользователей представления - это таблица, которая не хранится постоянно в базе данных, а "возникает" в момент обращения к ней. С точки зрения оператора SELECT и постоянно хранимые таблицы, и временные таблицы и представления выглядят совершенно одинаково. Конечно, при реальном выполнении оператора SELECT системой учитываются различия между хранимыми таблицами и представлениями, но эти различия скрыты от пользователя.

Результатом выполнения оператора SELECT всегда является таблица. Таким образом, по результатам действий оператор SELECT похож на операторы реляционной алгебры. Любой оператор реляционной алгебры может быть выражен подходящим образом сформулированным оператором SELECT. Сложность оператора SELECT определяется тем, что он содержит в себе все возможности реляционной алгебры, а также дополнительные возможности, которых в реляционной алгебре нет.

3.3 Базовые предикаты

Предикаты представляют собой выражения, принимающие истинностное значение. Они могут представлять собой как одно выражение, так и любую комбинацию из

неограниченного количества выражений, построенную с помощью булевых операторов AND, OR или NOT. Кроме того, в этих комбинациях может использоваться SQL-оператор IS, а также круглые скобки для конкретизации порядка выполнения операций.

Предикат в языке SQL может принимать одно из трех значений TRUE (истина), FALSE (ложь) или UNKNOWN (неизвестно). Исключение составляют следующие предикаты: IS NULL (отсутствие значения), EXISTS (существование), UNIQUE (уникальность) и MATCH (совпадение), которые не могут принимать значение UNKNOWN.

Правила комбинирования всех трех истинностных значений легче запомнить, обозначив TRUE как 1, FALSE как 0 и UNKNOWN как 1/2 (где-то между истинным и ложным значениями).

AND с двумя истинностными значениями дает минимум этих значений. Например, TRUE AND UNKNOWN будет равно UNKNOWN.

OR с двумя истинностными значениями дает максимум этих значений. Например, FALSE OR UNKNOWN будет равно UNKNOWN.

Отрицание истинностного значения равно 1 минус данное истинностное значение. Например, NOT UNKNOWN будет равно UNKNOWN.

Логические операторы при отсутствии скобок, как и арифметические операторы, выполняются в соответствии с их старшинством. Одноместная операция NOT имеет наивысший приоритет. В этом легко убедиться, если выполнить следующие два запроса.

```
-- модели, не являющиеся ПК
-- второй предикат ничего не меняет, т.к. он добавляет условие,
-- уже учтенное в первом предикате
SELECT maker, model, type
FROM Product
WHERE NOT type='PC' OR type='Printer';
```

```
-- модели производителя A, которые не являются ПК
SELECT maker, model, type
FROM Product
WHERE NOT type='PC' AND maker='A';
```

Поменять порядок выполнения логических операторов можно при помощи скобок:

```
-- модели, не являющиеся ПК или принтером, т.е. модели ноутбуков в нашем случае
SELECT maker, model, type
FROM Product
WHERE NOT (type='PC' OR type='Printer');
```

```
-- модели, которые не являются ПК, выпускаемыми производителем A
SELECT maker, model, type
FROM Product
WHERE NOT (type='PC' AND maker='A');
```

Следующий приоритет имеет оператор AND. Сравните результаты следующих запросов.

```
-- модели ПК, выпускаемые производителем A, и любые модели производителя B
SELECT maker, model, type
FROM Product
WHERE type='PC' AND maker='A' OR maker='B';
-- модели ПК, выпускаемые производителем A или производителем B
```

```
SELECT maker, model, type
FROM Product
WHERE type='PC' AND (maker='A' OR maker='B');
```

Предикат в предложении WHERE выполняет реляционную операцию ограничения, т.е. строки, появляющиеся на выходе предложения FROM ограничиваются теми, для которых предикат дает значение TRUE.

Если cond1 и cond2 являются простыми условиями, то ограничение по предикату

cond1 AND cond2

эквивалентно пересечению ограничений по каждому из предикатов.

Ограничение по предикату

cond1 OR cond2

эквивалентно объединению ограничений по каждому из предикатов.

Ограничение по предикату

NOT cond1

эквивалентно взятию разности, когда от исходного отношения вычитается ограничение по предикату cond1.

3.4 Примеры применения базовых предикатов

Задача: Получить информацию о моделях ПК производителя A.

В данном примере:

cond1: maker = 'A' ,

cond2: type = 'pc'.

cond1 **AND** cond2

```
SELECT * FROM product
WHERE maker = 'A' AND type = 'pc';
```

Пересечение:

```
SELECT * FROM product
WHERE maker = 'A'
INTERSECT
SELECT * FROM product
WHERE type = 'pc';
```

Задача: Получить информацию о моделях производителей A и B.

Здесь:

cond1: maker = 'A' ,

cond2: maker = 'B'.

cond1 **OR** cond2

```
SELECT * FROM product
WHERE maker = 'A' OR maker = 'B';
```

Объединение

```
SELECT * FROM product
WHERE maker = 'A'
UNION
SELECT * FROM product
WHERE maker = 'B';
```

В свою очередь, условия condX могут не быть простыми. Например:

Задача: Получить информацию о моделях ПК производителей A и B.

```
SELECT * FROM product
WHERE (maker = 'A' OR maker = 'B') AND type = 'pc';
```

можно выразить через пересечение

```
SELECT * FROM product
WHERE maker = 'A' OR maker = 'B'
INTERSECT
SELECT * FROM product
WHERE  type = 'pc';
```

а его эквивалентную форму

```
SELECT * FROM product
WHERE (maker = 'A' AND type = 'pc')
    OR (maker = 'B' AND type = 'pc');
```

через объединение

```
SELECT * FROM product
WHERE maker = 'A' AND type = 'pc'
UNION
SELECT * FROM product
WHERE maker = 'B' AND type = 'pc';
```

3.5 Предикаты сравнения

Предикат сравнения представляет собой два выражения, соединяемых оператором сравнения. Имеется шесть традиционных операторов сравнения: =, >, <, >=, <=, <>.

Данные типа NUMERIC (числа) сравниваются в соответствии с их алгебраическим значением.

Данные типа CHARACTER STRING (символьные строки) сравниваются в соответствии с их алфавитной последовательностью. Если a1a2…an и в1 в…вn — две последовательности символов, то первая «меньше» второй, если a1 < в1, или a1 = в1 и a2 < в2 и т. д. Считается также, что a1, a2…an < в1 в2…вm, если n < m и a1, a2…an = в1, в2…вn, то есть если первая строка является префиксом второй. Например, ‘folder’ < ‘for’, так как первые две буквы этих строк совпадают, а третья буква строки ‘folder’ предшествует третьей букве строки ‘for’. Также справедливо неравенство ‘bar’ < ‘barber’, поскольку первая строка является префиксом второй.

Данные типа DATETIME (дата/время) сравниваются в хронологическом порядке. Данные типа INTERVAL (временной интервал) преобразуются в соответствующие типы, а затем сравниваются как обычные числовые значения типа NUMERIC.

Задача: Получить информацию о компьютерах, имеющих частоту процессора не менее 500 МГц и цену ниже $800

```
SELECT *
FROM PC
WHERE speed >= 500 AND
 price < 800;
```

Задача: Получить информацию обо всех принтерах, которые не являются матричными и стоят меньше $300

```
SELECT *
FROM printer
WHERE NOT (type = 'matrix') AND
 price < 300;
```

3.6 Предикат BETWEEN

Синтаксис:

```
BETWEEN::=
<Проверяемое выражение> [NOT] BETWEEN
 <Начальное выражение> AND <Конечное выражение>
```

Предикат BETWEEN проверяет, попадают ли значения проверяемого выражения в диапазон, задаваемый пограничными выражениями, соединяемыми служебным словом AND. Естественно, как и для предиката сравнения, выражения в предикате BETWEEN должны быть совместимы по типам.

Предикат

```
exp1 BETWEEN exp2 AND exp3
```

равносилен предикату

```
exp1 >= exp2 AND exp1 <= exp3
```

А предикат

```
exp1 NOT BETWEEN exp2 AND exp3
```

равносилен предикату

```
NOT (exp1 BETWEEN exp2 AND exp3)
```

Если значение предиката exp1 BETWEEN exp2 AND exp3 равно TRUE, в общем случае это отнюдь не означает, что значение предиката exp1 BETWEEN exp3 AND exp2 тоже будет TRUE, так как первый будет интерпретироваться как предикат:

```
exp1 >= exp2 AND exp1 <= exp3
```

а второй как:

```
exp1 >= exp3 AND exp1 <= exp2
```

Пример. Найти модель и частоту процессора компьютеров стоимостью от $400 до $600:

```
SELECT model, speed

FROM PC

WHERE price BETWEEN 400 AND 600;
```

3.7 Предикат IN

Синтаксис:

```
IN::=

 <Проверяемое выражение> [NOT] IN (<подзапрос>)

 | (<выражение для вычисления значения>,...)
```

Предикат IN определяет, будет ли значение проверяемого выражения обнаружено в наборе значений, который либо явно определен, либо получен с помощью табличного подзапроса. Здесь табличный подзапрос - это обычный оператор SELECT, который создает одну или несколько строк для одного столбца, совместимого по типу данных со значением проверяемого выражения. Если целевой объект эквивалентен хотя бы одному из указанных в предложении IN значений, истинностное значение предиката IN будет равно TRUE. Если для каждого значения X в предложении IN целевой объект <> X, истинностное значение будет равно FALSE. Если подзапрос выполняется, и результат не содержит ни одной строки

(пустая таблица), предикат принимает значение FALSE. Когда не соблюдается ни одно из упомянутых выше условий, значение предиката равно UNKNOWN.

Пример. Найти модель, частоту процессора и объем жесткого диска тех компьютеров, которые комплектуются накопителями 10 или 20 Гбайт:

```
SELECT model, speed, hd
FROM PC
WHERE hd IN (10, 20);
```

Пример. Найти модель, частоту процессора и объем жесткого диска компьютеров, которые комплектуются накопителями 10 Гбайт или 20 Гбайт и выпускаются производителем A:

```
SELECT model, speed, hd
FROM PC
WHERE hd IN (10, 20) AND
 model IN (SELECT model
 FROM product
 WHERE maker = 'A'
 );
```

3.8 Предикат LIKE

Синтаксис:

```
LIKE::=
 <Выражение для вычисления значения строки>
 [NOT] LIKE <Выражение для вычисления значения строки>
 [ESCAPE <символ>]
```

Предикат LIKE сравнивает строку, указанную в первом выражении, для вычисления значения строки, называемого проверяемым значением, с образцом, который определен во втором выражении для вычисления значения строки. В образце разрешается использовать два трафаретных символа:

— символ подчеркивания (_), который можно применять вместо любого единичного символа в проверяемом значении;

— символ процента (%) заменяет последовательность любых символов (число символов в последовательности может быть от 0 и более) в проверяемом значении.

Если проверяемое значение соответствует образцу с учетом трафаретных символов, то значение предиката равно TRUE. В таблице 1 приводится несколько примеров написания образцов.

Таблица 1 – Примеры написания образцов оператора LIKE

Образец	**Описание**
'abc%'	Любые строки, которые начинаются с букв «abc»
'abc_'	Строки длиной строго 4 символа, причем первыми символами строки должны быть «abc»
'%z'	Любая последовательность символов, которая обязательно заканчивается символом «z»
'%Rostov%'	Любая последовательность символов, содержащая слово «Rostov» в любой позиции строки
'% % %'	Текст, содержащий не менее 2-х пробелов, например, "World Wide Web"

Пример. Найти все корабли, имена классов которых заканчиваются на букву ‘o’

```
SELECT *
FROM Ships
WHERE class LIKE '%o' ;
```

Истинностное значение предиката LIKE присваивается в соответствии со следующими правилами:

— если либо проверяемое значение, либо образец, либо управляющий символ есть NULL, истинностное значение равно UNKNOWN;

— в противном случае, если проверяемое значение и образец имеют нулевую длину, истинностное значение равно TRUE;

— в противном случае, если проверяемое значение соответствует шаблону, то предикат LIKE равен TRUE;

— если не соблюдается ни одно из перечисленных выше условий, предикат LIKE равен FALSE.

3.9 Использование значения NULL в условиях поиска

Предикат:

```
IS [NOT] NULL
```

позволяет проверить отсутствие (наличие) значения в полях таблицы. Использование в этих случаях обычных предикатов сравнения может привести к неверным результатам, так как сравнение со значением NULL дает результат UNKNOWN (неизвестно).

Так, если требуется найти записи в таблице PC, для которых в столбце price отсутствует значение (например, при поиске ошибок ввода), можно воспользоваться следующим оператором:

```
SELECT *
FROM PC
WHERE price IS NULL;
```

Характерной ошибкой является написание предиката в виде:

```
WHERE price = NULL
```

Этому предикату не соответствует ни одной строки, поэтому результирующий набор записей будет пуст, даже если имеются изделия с неизвестной ценой. Это происходит потому, что сравнение с NULL-значением согласно предикату сравнения оценивается как UNKNOWN. А строка попадает в результирующий набор только в том случае, если предикат в предложении WHERE есть TRUE. Это же справедливо и для предиката в предложении HAVING.

3.10 Переименование столбцов и вычисления в результирующем наборе

Имена столбцов, указанные в предложении SELECT, можно переименовать. Это делает результаты более читабельными, поскольку имена полей в таблицах часто сокращают с целью упрощения набора. Ключевое слово AS, используемое для переименования, согласно стандарту можно и опустить, так как оно неявно подразумевается.

Например, запрос:

```
SELECT ram AS Mb, hd Gb
FROM PC
WHERE cd = '24x';
```

переименует столбец ram в Mb (мегабайты), а столбец hd в Gb (гигабайты). Этот запрос возвратит объемы оперативной памяти и жесткого диска для тех компьютеров, которые имеют 24-скоростной CD-ROM.

Иногда бывает необходимо выводить поясняющую информацию рядом с соответствующим значением. Это можно сделать, добавив строковое выражение как дополнительный столбец. Например, запрос:

```
SELECT ram, 'Mb' AS ram_units, hd, 'Gb' AS hd_units
FROM PC
WHERE cd = '24x';
```

3.11 Получение итоговых значений

Как узнать количество моделей ПК, выпускаемых тем или иным поставщиком? Как определить среднее значение цены на компьютеры, имеющие одинаковые технические характеристики? На эти и многие другие вопросы, связанные с некоторой статистической информацией, можно получить ответы при помощи итоговых (агрегатных) функций. Стандартом предусмотрены следующие агрегатные функции, представленные в таблице 2:

Таблица 2 – Агрегатные функции

Название	**Описание**
COUNT(*)	Возвращает количество строк источника записей
COUNT	Возвращает количество значений в указанном столбце
SUM	Возвращает сумму значений в указанном столбце
AVG	Возвращает среднее значение в указанном столбце
MIN	Возвращает минимальное значение в указанном столбце
MAX	Возвращает максимальное значение в указанном столбце

Все эти функции возвращают единственное значение. При этом функции COUNT, MIN и MAX применимы к данным любого типа, в то время как SUM и AVG используются только для данных числового типа. Разница между функцией COUNT(*) и COUNT(имя столбца | выражение) состоит в том, что вторая (как и остальные агрегатные функции) при подсчете не учитывает NULL-значения.

Пример: Найти минимальную и максимальную цену на персональные компьютеры

```
SELECT MIN(price) AS Min_price,
    MAX(price) AS Max_price
FROM PC;
```

Пример: Найти имеющееся в наличии количество компьютеров, выпущенных производителем A

```
SELECT COUNT(*) AS Qty
FROM PC
WHERE model IN(SELECT model
 FROM Product
 WHERE maker = 'A'
 );
```

Пример: Если же нас интересует количество различных моделей, выпускаемых производителем A, то запрос можно сформулировать следующим образом (пользуясь тем фактом, что в таблице Product номер модели - столбец model - является первичным ключом и, следовательно, не допускает повторений)

```
SELECT COUNT(model) AS Qty_model
FROM Product
WHERE maker = 'A';
```

Для того чтобы при получении статистических показателей использовались только уникальные значения, при аргументе агрегатных функций можно применить параметр DISTINCT. Другой параметр - ALL - задействуется по умолчанию и предполагает подсчет всех возвращаемых (не NULL) значений в столбце.

Если же нам требуется получить количество моделей ПК, производимых каждым производителем, то потребуется использовать предложение GROUP BY, синтаксически следующего после предложения WHERE.

3.12 Предложение GROUP BY

Предложение GROUP BY используется для определения групп выходных строк, к которым могут применяться агрегатные функции (COUNT, MIN, MAX, AVG и SUM). Если это предложение отсутствует, и используются агрегатные функции, то все столбцы с именами, упомянутыми в SELECT, должны быть включены в агрегатные функции, и эти функции будут применяться ко всему набору строк, которые удовлетворяют предикату запроса. В противном случае все столбцы списка SELECT, не вошедшие в агрегатные функции, должны быть указаны в предложении GROUP BY. В результате чего все выходные строки запроса разбиваются на группы, характеризуемые одинаковыми комбинациями значений в этих столбцах. После чего к каждой группе будут применены агрегатные функции. Следует иметь в виду, что для GROUP BY все значения NULL трактуются как равные, то есть при группировке по полю, содержащему NULL-значения, все такие строки попадут в одну группу.

Если при наличии предложения GROUP BY, в предложении SELECT отсутствуют агрегатные функции, то запрос просто вернет по одной строке из каждой группы. Эту возможность, наряду с ключевым словом DISTINCT, можно использовать для исключения дубликатов строк в результирующем наборе.

Рассмотрим простой пример:

```
SELECT model, COUNT(model) AS Qty_model,
    AVG(price) AS Avg_price
FROM PC
GROUP BY model;
```

В этом запросе для каждой модели ПК определяется их количество и средняя стоимость. Все строки с одинаковыми значениями model (номер модели) образуют группу, и на выходе SELECT вычисляются количество значений и средняя цена для каждой группы.

Существует несколько определенных правил выполнения агрегатных функций.

1. Если в результате выполнения запроса не получено ни одной строки (или ни одной строки для данной группы), то исходные данные для вычисления любой из агрегатных функций отсутствуют. В этом случае результатом выполнения функций **COUNT** будет нуль, а результатом всех других функций — **NULL**.

Данное свойство может дать не всегда очевидный результат. Рассмотрим, например, такой запрос:

```
SELECT 1 a WHERE
EXISTS(SELECT  MAX(price)
         FROM PC
         WHERE price<0);
```

Подзапрос в предикате EXISTS возвращает одну строку с NULL в качестве значения столбца. Поэтому, несмотря на то, что ПК с отрицательными ценами нет в базе данных, запрос в примере вернет 1.

2. Аргумент агрегатной функции не может сам содержать агрегатные функции (функция от функции). То есть в простом запросе (без подзапросов) нельзя, скажем, получить максимум средних значений.

3. Результат выполнения функции COUNT есть целое число (INTEGER). Другие агрегатные функции наследуют типы данных обрабатываемых значений.

4. Если при выполнении функции SUM будет получен результат, превышающий максимально возможное значение для используемого типа данных, возникает ошибка.

Итак, агрегатные функции, включенные в предложение SELECT запроса, не содержащего предложения GROUP BY, исполняются над всеми результирующими строками этого запроса. Если же запрос содержит предложение GROUP BY, каждый набор строк, который имеет одинаковые значения столбца или группы столбцов, заданных в предложении GROUP BY, составляют группу, и агрегатные функции выполняются для каждой группы отдельно.

3.13 Предложение HAVING

Если предложение WHERE определяет предикат для фильтрации строк, то предложение HAVING применяется после группировки для определения аналогичного предиката, фильтрующего группы по значениям агрегатных функций. Это предложение необходимо для проверки значений, которые получены с помощью агрегатной функции не из отдельных строк источника записей, определенного в предложении FROM, а из групп таких строк. Поэтому такая проверка не может содержаться в предложении WHERE.

Пример: Получить количество ПК и среднюю цену для каждой модели, средняя цена которой менее $800

```
SELECT model, COUNT(model) AS Qty_model,
   AVG(price) AS Avg_price
FROM PC
GROUP BY model
HAVING AVG(price) < 800;
```

Заметим, что в предложении HAVING нельзя использовать псевдоним (Avg_price), используемый для именования значений агрегатной функции в предложении SELECT. Дело в том, что предложение SELECT, формирующее выходной набор запроса, выполняется предпоследним перед предложением ORDER BY. Ниже приведен порядок обработки предложений в операторе SELECT:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

Этот порядок не соответствует синтаксическому порядку общего представления оператора SELECT, который ближе к естественному языку:

```
SELECT [DISTINCT | ALL]{*
 | [<выражение для столбца> [[AS] <псевдоним>]] [,…]}
FROM <имя таблицы> [[AS] <псевдоним>] [,…]
[WHERE <предикат>]
[[GROUP BY <список столбцов>]
[HAVING <условие на агрегатные значения>] ]
[ORDER BY <список столбцов>]
```

Следует отметить, что предложение HAVING может использоваться и без предложения GROUP BY. При отсутствии предложения GROUP BY агрегатные функции применяются ко всему выходному набору строк запроса, т.е. в результате мы получим всего одну строку, если выходной набор не пуст.

Таким образом, если условие на агрегатные значения в предложение HAVING будет истинным, то эта строка будет выводиться, в противном случае мы не получим ни одной строки. Рассмотрим такой пример.

Пример: Найти максимальную, минимальную и среднюю цену на персональные компьютеры.

```
SELECT MIN(price) AS min_price,
MAX(price) AS max_price, AVG(price) avg_price
FROM PC;
```

Пример: Найти максимальную, минимальную и среднюю цену на персональные компьютеры при условии, что средняя цена не превышает $600.

```
SELECT MIN(price) AS min_price,
MAX(price) AS max_price, AVG(price) avg_price
FROM PC
HAVING AVG(price) <= 600;
```

3.14 Сортировка и NULL-значения

Если столбец, по которому выполняется сортировка, допускает NULL-значения, то при использовании SQL Server следует иметь в виду, что при сортировке по возрастанию NULL-значения будут идти в начале списка, а при сортировке по убыванию - в конце.

Создадим копию таблицы PC с именем PC_, где существует строка, со значением NULL в столбце price:

```
INSERT INTO PC_
VALUES(13,2112,600,64,8,'24x',NULL);
```

Почему это важно? Дело в том, что при поиске экстремальных значений часто используют метод, основанный на сортировке. Рассмотрим, например, такую задачу.

Пример: Найти модели ПК, имеющих минимальную цену.

```
SELECT TOP 1 WITH ties model
FROM PC_
ORDER BY price;
```

Конструкция WITH TIES используется для того, чтобы вывести все модели с наименьшей ценой, если их окажется несколько. Однако в результате мы получим модель 2112, цена которой неизвестна, в то время как должны получить модели 1232 и 1260, имеющих действительно минимальные цены. Мы их и получим, если исключим из рассмотрения модели с неизвестными ценами:

```
SELECT TOP 1 WITH ties model
FROM PC_
WHERE price IS NOT NULL
ORDER BY price;
```

Но тут появляется еще одна проблема, связанная с дубликатами. Поскольку могут быть два или больше ПК модели 1232 с минимальной ценой, то все они будут выводиться в результирующем наборе. DISTINCT без указания в списке столбцов предложения SELECT тех, по которым выполняется сортировка, использовать мы не можем, о чем и сообщает ошибка, если мы попытаемся это сделать

```
SELECT DISTINCT TOP 1 WITH ties model
FROM PC_
WHERE price IS NOT NULL
ORDER BY price;
```

3.15 Использование в запросе нескольких источников записей

В предложении FROM допускается указание нескольких таблиц. Простое перечисление таблиц через запятую практически не используется, поскольку оно соответствует реляционной операции, которая называется декартовым произведением.

Поэтому перечисление таблиц, как правило, используется совместно с условием соединения строк из разных таблиц, указываемым в предложении WHERE.

Явные операции соединения

В предложении FROM может быть указана явная операция соединения двух и более таблиц. Среди ряда операций соединения, описанных в стандарте языка SQL, многими серверами баз данных поддерживается только операция соединения по предикату. Синтаксис соединения по предикату имеет вид:

```
FROM <таблица 1>
 [INNER]
 {{LEFT | RIGHT | FULL } [OUTER]} JOIN <таблица 2>
[ON <предикат>]
```

Соединение может быть либо внутренним (INNER), либо одним из внешних (OUTER). Служебные слова INNER и OUTER можно опускать, поскольку внешнее соединение однозначно определяется его типом — LEFT (левое), RIGHT (правое) или FULL (полное), а просто JOIN будет означать внутреннее соединение.

Предикат определяет условие соединения строк из разных таблиц. При этом INNER JOIN означает, что в результирующий набор попадут только те соединения строк двух таблиц, для которых значение предиката равно TRUE. Как правило, предикат определяет эквисоединение по внешнему и первичному ключам соединяемых таблиц, хотя это не обязательно.

Пример: Найти производителя, номер модели и цену каждого компьютера, имеющегося в базе данных.

```
SELECT maker, Product.model AS model_1,
            PC.model AS model_2, price
FROM Product INNER JOIN
 PC ON PC.model = Product.model
ORDER BY maker, model_2;
```

Внешнее соединение LEFT JOIN означает, что помимо строк, для которых выполняется условие предиката, в результирующий набор попадут все остальные строки из первой таблицы (левой). При этом отсутствующие значения столбцов из правой таблицы будут заменены NULL-значениями.

Пример: Привести все модели ПК, их производителей и цену.

```
SELECT maker, Product.model AS model_1, pc.model AS model_2, price
FROM Product LEFT JOIN
 PC ON PC.model = Product.model
WHERE type = 'pc'
ORDER BY maker, PC.model;
```

Соединение RIGHT JOIN обратно соединению LEFT JOIN, то есть в результирующий набор попадут все строки из второй таблицы, которые будут соединяться только с теми строками из первой таблицы, для которых выполняется условие соединения. В нашем случае левое соединение

```
Product LEFT JOIN PC ON PC.model = Product.model
```

будет эквивалентно правому соединению

```
PC RIGHT JOIN Product ON PC.model = Product.model
```

Наконец, при полном соединении (FULL JOIN) в результирующую таблицу попадут не только те строки, которые имеют одинаковые значения в сопоставляемых столбцах, но и все остальные строки исходных таблиц, не имеющие соответствующих значений в другой таблице. В этих строках все столбцы той таблицы, в которой не было найдено соответствия, заполняются NULL-значениями. То есть полное соединение представляет собой комбинацию левого и правого внешних соединений. Так, запрос для таблиц A и B, приведенных в начале:

```
SELECT A.*, B.*
FROM A FULL JOIN
  B ON A.a = B.c;
```

Заметим, что это соединение симметрично, то есть A FULL JOIN B эквивалентно B FULL JOIN A. Обратите также внимание на обозначение A.*, что означает вывести все столбцы таблицы A.

Пример: Найти производителей, которые выпускают принтеры, но не ПК, или выпускают ПК, но не принтеры.

Воспользуемся формулой. Полное соединение производителей ПК и производителей принтеров даст нам как тех, кто производит что-то одно, так и тех, кто производит и то, и другое.

```
SELECT * FROM
(SELECT DISTINCT maker FROM Product WHERE type='pc') m_pc
FULL JOIN
(SELECT DISTINCT maker FROM Product WHERE type='printer') m_printer
ON m_pc.maker = m_printer.maker;
```

Теперь вычтем из результата тех, кто производит и то, и другое (внутреннее соединение):

```
SELECT m_pc.maker m1, m_printer.maker m2  FROM
(SELECT maker FROM Product WHERE type='pc') m_pc
FULL JOIN
(SELECT maker FROM Product WHERE type='printer') m_printer
ON m_pc.maker = m_printer.maker
EXCEPT
SELECT * FROM
(SELECT maker FROM Product WHERE type='pc') m_pc
INNER JOIN
(SELECT maker FROM Product WHERE type='printer') m_printer
ON m_pc.maker = m_printer.maker;
```

Попутно убрали из этого решения избыточные DISTINCT, поскольку EXCEPT выполнит исключение дубликатов. Это единственный полезный тут урок, т.к. операцию взятия разности (EXCEPT) можно заменить простым предикатом:

```
WHERE m_pc.maker IS NULL OR m_printer.maker IS NULL
```

или даже так

```
m_pc.maker + m_printer.maker IS NULL
```

ввиду того, что конкатенация с NULL-значением дает NULL.

```
SELECT * FROM
(SELECT DISTINCT maker FROM Product WHERE type='pc') m_pc
FULL JOIN
(SELECT DISTINCT maker FROM Product WHERE type='printer') m_printer
ON m_pc.maker = m_printer.maker
WHERE m_pc.maker IS NULL OR m_printer.maker IS NULL;
```

Наконец, чтобы представить результат в один столбец, воспользуемся функцией COALESCE:

```
SELECT COALESCE(m_pc.maker, m_printer.maker) FROM
(SELECT DISTINCT maker FROM Product WHERE type='pc') m_pc
FULL JOIN
(SELECT DISTINCT maker FROM Product WHERE type='printer') m_printer
ON m_pc.maker = m_printer.maker
WHERE m_pc.maker IS NULL OR m_printer.maker IS NULL;
```

3.16 Объединение

Для объединения запросов используется служебное слово **UNION**:

```
<запрос 1>
UNION [ALL]
<запрос 2>
```

Предложение UNION приводит к появлению в результирующем наборе всех строк каждого из запросов. При этом, если определен параметр ALL, то сохраняются все дубликаты выходных строк, в противном случае в результирующем наборе присутствуют только уникальные строки. Заметим, что можно связывать вместе любое число запросов. Кроме того, с помощью скобок можно задавать порядок объединения.

Операция объединения может быть выполнена только при выполнении следующих условий:

— количество выходных столбцов каждого из запросов должно быть одинаковым;

— выходные столбцы каждого из запросов должны быть совместимы между собой (в порядке их следования) по типам данных;

— в результирующем наборе используются имена столбцов, заданные в первом запросе;

— предложение ORDER BY применяется к результату соединения, поэтому оно может быть указано только в конце всего составного запроса.

Пример: Найти номера моделей и цены ПК и портативных компьютеров.

```
SELECT model, price
FROM PC
UNION
SELECT model, price
FROM Laptop
ORDER BY price DESC;
```

3.17 Пересечение и разность

В стандарте языка SQL имеются предложения оператора SELECT для выполнения операций пересечения и разности результатов запросов-операндов. Этими предложениями являются INTERSECT [ALL] (пересечение) и EXCEPT [ALL] (разность), которые работают аналогично предложению UNION. В результирующий набор попадают только те строки, которые присутствуют в обоих запросах (INTERSECT) или только те строки первого запроса, которые отсутствуют во втором (EXCEPT). При этом оба запроса, участвующих в операции, должны иметь одинаковое число столбцов, и соответствующие столбцы должны иметь одинаковые (или неявно приводимые) типы данных. Имена столбцов результирующего набора формируются из заголовков первого запроса.

Если не используется ключевое слово ALL (по умолчанию подразумевается DISTINCT), то при выполнении операции автоматически устраняются дубликаты строк.

Если указано ALL, то количество дублированных строк подчиняется следующим правилам (n1 - число дубликатов строк первого запроса, n2 - число дубликатов строк второго запроса):

— INTERSECT ALL: min(n1, n2)

— EXCEPT ALL: n1 - n2, если n1>n2.

3.18 Предикат EXISTS

Синтаксис:

```
EXISTS::=
 [NOT] EXISTS (<табличный подзапрос>)
```

Предикат EXISTS принимает значение TRUE, если подзапрос содержит любое количество строк, иначе его значение равно FALSE. Для NOT EXISTS все наоборот. Этот предикат никогда не принимает значение UNKNOWN.

Обычно предикат EXISTS используется в зависимых (коррелирующих) подзапросах. Этот вид подзапроса имеет внешнюю ссылку, связанную со значением в основном запросе. Результат подзапроса может зависеть от этого значения и должен оцениваться отдельно для каждой строки запроса, в котором содержится данный подзапрос. Поэтому предикат EXISTS может иметь разные значения для разных строк основного запроса.

Пример на пересечение: Найти тех производителей портативных компьютеров, которые также производят принтеры.

```
SELECT DISTINCT maker
FROM Product AS lap_product
WHERE type = 'laptop' AND
 EXISTS (SELECT maker
 FROM Product
 WHERE type = 'printer' AND
 maker = lap_product.maker
 );
```

3.19 Реляционное деление

Рассмотрим следующую задачу. Определить производителей, которые выпускают модели всех типов. Ключевым словом здесь является «всех», т.е. производитель в таблице Product должен иметь модели каждого типа, т.е. и PC, и Laptop, и Printer. Как раз для решения подобных задач в реляционную алгебру Коддом была введена специальная операция реляционного деления (DIVIDE BY).

С помощью этой операции наша задача решается очень просто:

```
Product[maker, type] DIVIDE BY Product[type]
```

Здесь квадратными скобками обозначается операция взятия проекции на соответствующие атрибуты. Операция реляционного деления избыточна, т.е. она может быть выражена через другие операции реляционной алгебры.

3.20 Разность

Если взять операцию разности ВСЕХ имеющихся типов моделей и типов у конкретного производителя, то результирующая выборка не должна содержать строк.

```
SELECT DISTINCT maker
FROM Product Pr1
WHERE 0 = (SELECT COUNT(*) FROM
(SELECT type FROM Product
 EXCEPT
 SELECT type FROM Product Pr2
 WHERE Pr2.maker = Pr1.maker
```

```
) X );
```

Этот запрос можно написать короче, если воспользоваться тем свойством, что истинностное значение предиката ALL есть TRUE, если подзапрос не возвращает строк:

```
SELECT DISTINCT maker
FROM Product Pr1
WHERE type = ALL
(SELECT type FROM Product
 EXCEPT
 SELECT type FROM Product Pr2
 WHERE Pr2.maker = Pr1.maker
);
```

Для искомых производителей список типов в предикате ALL будет пуст (предикат равен TRUE). В остальных случаях он будет содержать типы моделей, отсутствующие у производителя из внешнего запроса, поэтому операция сравнения (равенство "=") для всех его моделей даст FALSE.

3.21 Оператор CASE

Оператор CASE может быть использован в одной из двух синтаксических форм записи:

1-я форма:

```
CASE <проверяемое выражение>
 WHEN <сравниваемое выражение 1>
 THEN <возвращаемое значение 1>
 …
 WHEN <сравниваемое выражение N>
 THEN <возвращаемое значение N>
 [ELSE <возвращаемое значение>]
END
```

2-я форма:

```
CASE
 WHEN <предикат 1>
 THEN <возвращаемое значение 1>
 …
 WHEN <предикат N>
 THEN <возвращаемое значение N>
 [ELSE <возвращаемое значение>]
END
```

Все предложения WHEN должны иметь одинаковую синтаксическую форму, то есть нельзя смешивать первую и вторую формы. При использовании первой синтаксической формы условие WHEN удовлетворяется, как только значение проверяемого выражения станет равным значению выражения, указанного в предложении WHEN. При использовании второй синтаксической формы условие WHEN удовлетворяется, как только предикат принимает значение TRUE. При удовлетворении условия оператор CASE возвращает значение, указанное в соответствующем предложении THEN. Если ни одно из условий WHEN не выполнилось, то будет использовано значение, указанное в предложении ELSE. При отсутствии ELSE, будет возвращено NULL-значение. Если удовлетворены несколько условий, то будет возвращено значение предложения THEN первого из них, так как остальные просто не будут проверяться.

Пусть требуется вывести список всех моделей ПК с указанием их цены. При этом если модель отсутствует в продаже (ее нет в таблице PC), то вместо цены вывести текст «Нет в наличии».

Чтобы заменить NULL-значения нужным текстом, можно воспользоваться оператором CASE:

```
SELECT DISTINCT product.model,
 CASE
 WHEN price IS NULL
 THEN 'Нет в наличии'
 ELSE CAST(price AS CHAR(20))
 END price
FROM Product LEFT JOIN
 PC ON Product.model = PC.model
WHERE product.type = 'pc';
```

3.22 Операторы модификации данных

Язык манипуляции данными (DML — Data Manipulation Language) помимо оператора SELECT, осуществляющего извлечение информации из базы данных, включает операторы, изменяющие состояние данных. Эти операторы представлены в таблице 3:

Таблица 3 – Операторы модификации данных

Оператор	**Функция**
INSERT	Добавление записей (строк) в таблицу БД
UPDATE	Обновление данных в столбце таблицы БД
DELETE	Удаление записей из таблицы БД

Оператор INSERT

Оператор INSERT вставляет новые записи в таблицу. При этом значения столбцов могут представлять собой литеральные константы, либо являться результатом выполнения подзапроса. В первом случае для вставки каждой строки используется отдельный оператор INSERT; во втором случае будет вставлено столько строк, сколько возвращается подзапросом.

Синтаксис оператора следующий:

```
INSERT INTO <имя таблицы>[(<имя столбца>,...)]
{VALUES (<значение столбца>,…)}
| <выражение запроса>
| {DEFAULT VALUES}
```

При этом порядок значений должен соответствовать порядку, заданному оператором CREATE TABLE для таблицы, в которую вставляются строки.

Заметим, что при вставке строки в таблицу проверяются все ограничения, наложенные на данную таблицу. Это могут быть ограничения первичного ключа или уникального индекса, проверочные ограничения типа CHECK, ограничения ссылочной целостности. В случае нарушения какого-либо ограничения вставка строки будет отклонена.

Рассмотрим теперь случай использования подзапроса.

```
INSERT INTO Product_D
SELECT *
FROM Product
WHERE type = 'PC';
```

или

```
INSERT INTO Product_D
SELECT maker, model, type
FROM Product
WHERE type = 'PC';
```

или

```
INSERT INTO Product_D(maker, model, type)
SELECT maker, model, type
FROM Product
WHERE type = 'PC';
```

Оператор UPDATE

Оператор UPDATE изменяет имеющиеся данные в таблице. Команда имеет следующий синтаксис:

```
UPDATE <имя таблицы>
SET {<имя столбца> = {<выражение для вычисления значения столбца>
| NULL
| DEFAULT},...}
[ {WHERE <предикат>}]
```

С помощью одного оператора могут быть заданы значения для любого количества столбцов. Однако в одном и том же операторе UPDATE можно вносить изменения в каждый столбец указанной таблицы только один раз. При отсутствии предложения WHERE будут обновлены все строки таблицы.

Если столбец допускает NULL-значение, то его можно указать в явном виде. Кроме того, можно заменить имеющееся значение на значение по умолчанию (DEFAULT) для данного столбца.

Ссылка на «выражение для вычисления значения столбца» может относиться к текущим значениям в изменяемой таблице. Например, мы можем уменьшить все цены портативных компьютеров на 10 процентов с помощью следующего оператора:

```
UPDATE Laptop
SET price = price*0.9;
```

Для вычисления значений столбцов допускается также использование подзапросов. Например, требуется укомплектовать все портативные компьютеры самыми быстрыми процессорами из имеющихся в наличии. Тогда можно написать:

```
UPDATE Laptop
SET speed = (SELECT MAX(speed)
FROM Laptop
);
```

Пример: Пусть требуется указать «No PC» (нет ПК) в столбце type для тех моделей ПК из таблицы Product, для которых нет соответствующих строк в таблице PC.

Решение посредством соединения таблиц можно записать так:

```
UPDATE Product
SET type = 'No PC'
FROM Product pr LEFT JOIN
 PC ON pr.model=PC.model
WHERE type = 'pc' AND
 PC.model IS NULL;
```

Оператор DELETE

Оператор DELETE удаляет строки из временных или постоянных базовых таблиц, представлений или курсоров, причем в двух последних случаях действие оператора

распространяется на те базовые таблицы, из которых извлекались данные в эти представления или курсоры. Оператор удаления имеет простой синтаксис:

```
DELETE FROM <имя таблицы >
[WHERE <предикат>];
```

Если предложение WHERE отсутствует, удаляются все строки из таблицы или представления (представление должно быть обновляемым). Более быстро эту операцию (удаление всех строк из таблицы) можно в Transact-SQL также выполнить с помощью команды

```
TRUNCATE TABLE <имя таблицы>
```

Однако есть ряд особенностей в реализации команды TRUNCATE TABLE, которые следует иметь в виду:

— не журнализируется удаление отдельных строк таблицы; в журнал записывается только освобождение страниц, которые были заняты данными таблицы;

— не отрабатывают триггеры, в частности, триггер на удаление;

— команда неприменима, если на данную таблицу имеется ссылка по внешнему ключу, и даже если внешний ключ имеет опцию каскадного удаления.

— значение счетчика (IDENTITY) сбрасывается в начальное значение.

Пример: Требуется удалить из таблицы Laptop все портативные компьютеры с размером экрана менее 12 дюймов.

```
DELETE FROM Laptop
WHERE screen < 12;
```

Все блокноты можно удалить с помощью оператора

```
DELETE FROM Laptop;
```

или

```
TRUNCATE TABLE Laptop;
```

Transact-SQL расширяет синтаксис оператора DELETE, вводя дополнительное предложение FROM:

```
FROM <источник табличного типа>
```

При помощи источника табличного типа можно конкретизировать данные, удаляемые из таблицы в первом предложении FROM. При помощи этого предложения можно выполнять соединения таблиц, что логически заменяет использование подзапросов в предложении WHERE для идентификации удаляемых строк. Поясним сказанное на примере.

Пример: Пусть требуется удалить те модели ПК из таблицы Product, для которых нет соответствующих строк в таблице PC.

Используя стандартный синтаксис, эту задачу можно решить следующим запросом:

```
DELETE FROM Product
WHERE type = 'pc' AND
 model NOT IN (SELECT model
 FROM PC
 );
```

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Кузнецов, С.Д. Введение в модель данных SQL [Электронный ресурс]/ Кузнецов С.Д.— Электрон. текстовые данные.— М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 350 с.— Режим доступа: http://www.iprbookshop.ru/73664.html.— ЭБС «IPRbooks»

2. Кузнецов, С.Д. Введение в реляционные базы данных [Электронный ресурс]/ Кузнецов С.Д.— Электрон. текстовые данные.— М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 247 с.— Режим доступа: http://www.iprbookshop.ru/73671.html.— ЭБС «IPRbooks»

3. Маркин, А.В. Программирование на SQL: учебное пособие для среднего профессионального образования/ А. В. Маркин. — Москва: Издательство Юрайт, 2020. — 435 с. — (Профессиональное образование). — ISBN 978-5-534-11093-7. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/456926

4. Стасышин, В.М. Практикум по языку SQL : учебное пособие / Стасышин В.М., Стасышина Т.Л.. — Новосибирск : Новосибирский государственный технический университет, 2016. — 60 с. — ISBN 978-5-7782-2937-2. — Текст : электронный // IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/91668.html. — Режим доступа: для авторизир. Пользователей.