Содержание

Структура учебной базы данных

Рассмотрим процесс построения учебной базы данных, которая ляжет в основу большинства примеров. Дадим её вербальное описание.

> База данных автомастерская. В базе данных должны учитываться: дата, наименование, стоимость оказанной услуги, мастер, специализация мастера, владелец автомобиля, марка автомобиля, год выпуска автомобиля регистрационный номер автомобиля.

Для разработки структуры БД воспользуемся стандартной процедурой.

Определение типов сущностей

Перечислим все сущности, которые присутствуют в вербальном описании БД. Определим, являются ли эти сущности сильными или слабыми по следующему принципу:

Сильная сущность — независимая сущность.

Слабая сущность — зависимая сущность.

В результате для учебной базы данный получим:

- дата оказания услуги (слабая сущность)
- наименование услуги (слабая сущность)
- стоимость услуги (слабая сущность)
- мастер (**сильная сущность**)
- специализация мастера (слабая сущность)
- владелец автомобиля (**сильная сущность**)
- марка автомобиля (слабая сущность)
- регистрационный номер автомобиля (слабая сущность)
- год выпуска автомобиля (слабая сущность)

В связи с тем, что в вербальном описании часть сущностей могут присутствовать в неявном виде, проведём дополнительный анализ и выявим:

- услуга (**сильная сущность**)
- автомобиль (**сильная сущность**)
- журнал оказанных услуг (**сильная сущность**)

Даже если на данном этапе не будет выявлено ни одной неявной сущности, они будут выявлены позже (если таковые вообще имеются).

Важно отметить, что понятие «сильная» и «слабая» сущность всегда должны рассматриваться в некотором контексте. Скажем, если в вербальном определении речь идет о «владельце автомобиля», то «владелец» — это слабая сущность, так как это одно из свойств автомобиля; автомобиль в данном случае рассматривается как сильная сущность. Но если описание учебной базы данных дополнить фразой «у владельца автомобиля есть номер телефона», то в этом случае «владелец» является сильной сущностью, а «номер телефона» — слабой сущностью, то есть свойством «владельца автомобиля». В этом случае в рамках одной базы данных одна и та же сущность рассматривается и как сильная, и как слабая. Эта сущность должна быть отмечена как сильная.

Определение типов связей

Чаще всего рассматриваются только бинарные связи, то есть связи, которые объединяют пары сущностей. Обычно для построения связи используется следующий приём: все перечисленные выше сущности связываются между собой каким-нибудь глаголом. При этом не обязательно, что для каждой пары сущностей может быть образована связь.

Связь 1:1 (один к одному) возникает, когда одной родительской сущности соответствует одна дочерняя сущность, и, наоборот: одной дочерней сущности соответствует одна родительская сущность.

Связь 1:M (один ко многим) возникает, когда одной родительской сущности соответствует несколько дочерних сущностей, но не наоборот: одной дочерней сущности соответствует одна родительская сущность.

Связь M:M (много ко многим) возникает, когда одной родительской сущности соответствует несколько дочерних сущностей, и, наоборот: одной дочерней сущности соответствует несколько родительских сущностей.

Выпишем связи, которые образуются в учебной базе данных.

- у услуги есть стоимость (1:1)
- у услуги есть название (1:1)
- услуга оказывается владельцу (M:M)
- у мастера есть ФИО (1:1)
- у мастера есть специализация (1:M)
- мастер оказывает услуги (M:M)
- у автомобиля есть цвет (1:M)
- у автомобиля есть марка (1:M)
- у автомобиля есть год выпуска (1:1)
- у автомобиля есть регистрационный номер (1:1)
- у владельца есть ФИО (1:1)
- у владельца есть адрес (1:1)
- у владельца есть номер телефона (1:1)
- у владельца есть дата рождения (1:1)

При проведении связей могут быть выявлены дополнительные сущности, которые могут сделать разрабатываемую БД более функциональной. Такими сущностями, например, являются: ФИО владельца, дата рождения владельца. Это позволит, скажем, поздравлять клиентов с днем рождения.

Поясним на конкретных примерах, почему в учебной базе данных проставлены те или иные типы связей.

- «у услуги есть название»: у услуги есть одно-единственное название, название соответствует одной-единственной услуге.
- «у автомобиля есть марка»: у автомобиля есть одна-единственная марка, но есть много автомобилей той же самой марки.
- «услуга оказывается владельцу»: владельцу оказывается много разных услуг, конкретная услуга может оказываться нескольким владельцам.

- «у автомобиля есть год выпуска». Рассмотрим эту связь подробнее. В принципе, здесь может быть проведена связь «один ко многим» по аналогии с маркой автомобиля, т.к. у автомобиля есть конкретный год выпуска, но есть много автомобилей с этим годом выпуска. Почему же в этом случае организуется связь «один к одному», а не «один ко многим? Дело в том, что марки автомобиля могут и должны быть вынесены в отдельную таблицу. В случае с годом выпуска в организации дополнительной таблицы нет смысла, поэтому используется связь один к одному.

Определение атрибутов и связывание их с типами сущностей

Атрибут — свойство сущности.

На одном из предыдущих этапов уже определено, какие из сущностей являются сильными, а какие слабыми. На основании этого распределения сущности должны образовать группы по следующему принципу — каждая слабая сущность становится атрибутом (свойством) сильной сущности. Из каждой сильной сущности формируется таблица, поля которой представляют собой атрибуты этой сильной сущности. Каждый атрибут может обладать дополнительными характеристиками.

Во-первых, необходимо выделить, какие из атрибутов являются простыми, а какие составными.

Атрибут называется простым (атомарным), если он не может быть разделён на составные части.

Составной атрибут может быть разделён на отдельные компоненты, которые в свою очередь могут быть рассмотрены как простые атрибуты.

Важно отметить, что составной атрибут является нежелательным при проектировании БД. Дело в том, что при заполнении значений составного атрибута достаточно сложно гарантировать, что пользователь будет строго следовать порядку перечисления компонент. Например, рассмотрим такой составной атрибут, как ФИО. Вполне очевидно, что сначала должна идти фамилия, затем имя и в последнюю очередь отчество. Однако практически невозможно отследить, что человек правильно внесёт эти данные. В результате нельзя будет, например, правильно отсортировать всех сотрудников по алфавиту. Невозможность контролирования порядка следования составных частей может привести к более серьезной проблеме: в базе данных рано или поздно появится дублирующуюся информация. Другими словами, может получиться несколько строк ссылающихся на один и тот же объект, например, «Иванов Иван Иванович» и «Иван Иванович Иванов». С точки зрения базы данных это два разных человека, а с точки зрения пользователя — один и тот же. Перечисленных проблем легко избежать, если разбить составной атрибут на необходимое количество простых.

Во-вторых, необходимо выделить потенциальные многозначные атрибуты.

Однозначный атрибут — атрибут, который содержит одно значение для одной сущности. Большинство атрибутов являются однозначными для каждого отдельного экземпляра этой сущности.

Многозначный атрибут — это атрибут, содержащий несколько значений для каждого экземпляра сущности.

Многозначные атрибуты является нежелательными — по тем же причинам, что и составные. Ярким примером многозначного атрибута является атрибут «телефонный номер», т.к. номер может быть рабочий, домашний, сотовый. При этом не совсем понятно, каким образом все эти значения могут быть помещены в единственное поле. Самым простым способом устранения многозначного атрибута является разбивание многозначного атрибута на столько однозначных атрибутов, сколько потенциальных значений он может принять.

В-третьих, могут быть выделены производные атрибуты, т.е. атрибуты, которые могут быть получены из имеющихся по определённому алгоритму. Иногда, если алгоритм вычисления значения производного атрибута является достаточно долгим и дорогостоящим, целесообразно внести некоторую избыточности в структуру базы данных с целью повышения общей производительности.

Теперь определим типы используемых данных. Можно выделить следующие основные типы данных:

- строка (символ)
- битовый
- числа (целые, с фиксированной точкой, с плавающей точкой)
- деньги
- дата-время

Получим следующее разбиение на таблицы:

владелец

● ФИО	(составной)		[строка]
● пол	(простой)		[строка]
● адрес	(составной)	(многозначный)	[строка]
● дата рождения	(составной)		[дата]
● телефон	(составной)	(многозначный)	[строка]

услуга

● название	(простой)	[строка]
● цена	(простой)	[деньги]

специальность

● название	(простой)	[строка]

журнал оказанных услуг

● дата оказания услуги	(составной)	[дата-время]
● время, на оказание услуги	(составной)	[дата-время]

автомобиль

● регистрационный номер	(составной)	[строка]
● год выпуска	(простой)	[дата]
● цвет автомобиля	(простой)	[строка]
● марка автомобиля	(простой)	[строка]

марка автомобиля

● название	(простой)	[строка]

мастер

- ФИО (составной) [строка]
- специальность (простой) [строка]

Определение доменов атрибутов

Домен — это набор допустимых значений для одного или нескольких атрибутов.

Другими словами, домен определяет все потенциальные значения, которые могут быть присвоены атрибуту. Одна из главных причин выделения доменов состоит в том, что домены помогают обеспечивать логическую целостность базы данных.

Приведём небольшой пример. Пол человека может быть записан или как «муж»/«жен», или как «м»/«ж». Для того, чтобы обеспечить однозначность написания, определяется домен. При попытке внести значение, которое не включено в домен, будет выдаваться ошибка о невозможности внесения данных.

Для учебной базы данных доменами могут быть выбраны:

- пол владельца
- дата рождения владельца
- год выпуска автомобиля
- дата оказания услуги
- время, затраченное на оказание услуги

Определение первичных ключей

Следующий важный момент при проектировании базы данных состоит в определении потенциальных и первичных ключей.

Потенциальным ключом называется атрибут или набор атрибутов, которые уникальным образом идентифицируют отдельные экземпляры типа сущности.

Потенциальный ключ обладает двумя свойствами:

- уникальность: ключ единственным образом идентифицирует экземпляр сущности;
- неприводимость: никакое допустимое подмножество ключа не обладает свойством уникальности.

Примером потенциального ключа могут служить табельный номер или номер паспорта.

Первичным ключом называется некоторый выбранный потенциальный ключ сущности.

При выборе первичного ключа среди нескольких потенциальных можно руководствоваться следующими рекомендациями:

- используйте потенциальный ключ с минимальным набором атрибутов;
- используйте тот потенциальный ключ, который имеет минимальную вероятность потери уникальности значений в будущем;
- по возможности используйте числовой потенциальный ключ;
- используйте потенциальный ключ, значения которого имеют минимальную длину (в случае текстовых атрибутов);
- остановите свой выбор на потенциальном ключе, с которым будет проще всего работать (с точки зрения пользователя).

Если сильная сущность не содержит ни одного подходящего потенциального ключа, то потенциальный ключ всегда может быть искусственным образом добавлен в виде некоторого уникального числового кода.

Перечислим первичные ключи из рассматриваемого примера.

- Для владельца это **код_владельца**
- Для автомобиля это **регистрационный_номер**
- Для услуги это **код_услуги**
- Для специальности это **код_специальности**
- Для журнала оказанных услуг это **номер_чека**
- Для марки автомобиля это **код_марки**
- Для мастера это **табельный_номер**

Определение внешних ключей

Внешний ключ — это атрибут или множество атрибутов внутри сущности, которое соответствует потенциальному ключу некоторой (может быть, той же самой) сущности.

Внешние ключи позволяют определять логическое связывание таблиц. Суть связывания состоит в установлении соответствия полей связи родительской и дочерней таблиц.

Перечислим внешние ключи с указанием таблиц, на которые они ссылаются.

В таблице **автомобили** три внешних ключа:

- **код_владельца** (таблица **владельцы**)
- **код_марки** (таблица **марки автомобилей**)
- **код_цвета** (таблица **цвета**)

В таблице **журнал_оказанных_услуг** тоже три внешних ключа:

- **регистрационный_номер** (таблица **автомобили**)
- **табельный_номер** (таблица **мастера**)
- **код_услуги** (таблица **услуги**)

И, наконец, в таблице **мастера** есть один внешний ключ **код_специальности**, ссылающийся на таблицу **специальности**.

Для удобства обычно принимают следующую схему именования таблиц и их столбцов: таблицы в своём название чаще всего содержат имена существительные во множественном числе, а столбцы — имена существительные в единственном числе.

Создание диаграммы «сущность-связь»

При создании диаграммы «сущность-связь» чаще всего используется какая-либо CASE-среда.

CASE-технология (Computer-Aided Software/System Engineering) представляет собой совокупность методологий анализа, проектирования, разработки и сопровождения сложных систем и поддерживается комплексом взаимосвязанных средств автоматизации.

CASE-технология обычно определяется как методология проектирования информационных систем плюс инструментальные средства, позволяющие наглядно моделировать предметную

область, анализировать ее модель на всех этапах разработки и сопровождения информационной системы и разрабатывать приложения для пользователей.

CASE-технологии обеспечивают всех участников проекта, включая заказчиков, единым, строгим, наглядным и интуитивно понятным графическим языком, позволяющим получать обозримые компоненты с простой и ясной структурой.

При этом проектируемые объекты (программы или структура базы данных) представляются двумерными схемами (которые проще в использовании, чем многостраничные описания), позволяющими заказчику участвовать в процессе разработки, а разработчикам — общаться с экспертами предметной области, разделять деятельность системных аналитиков, проектировщиков и программистов, обеспечивая легкость сопровождения и внесения изменений в систему.

На рисунке приведена диаграмма, разработанная средствами MS Visio 2003.

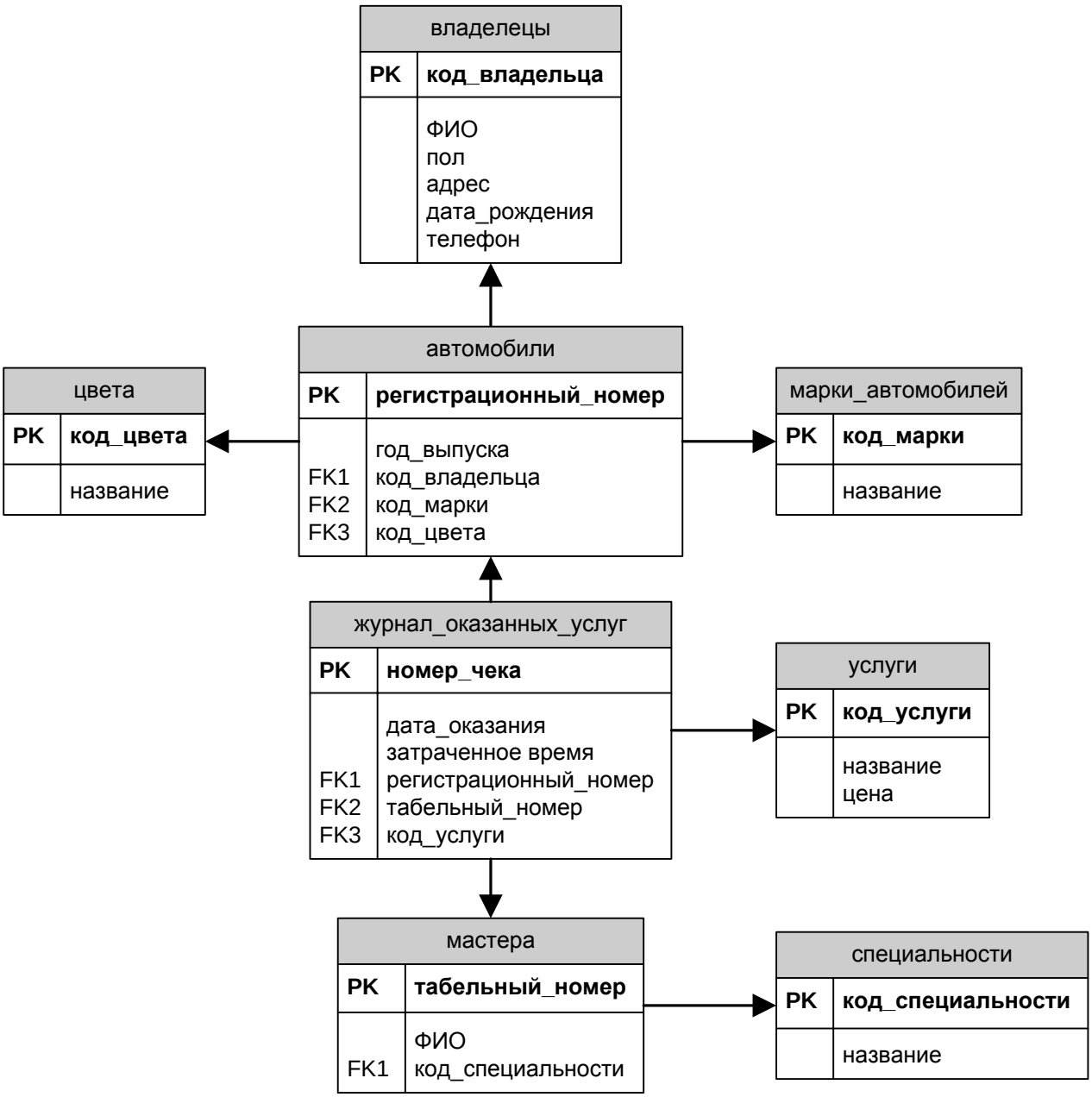

Рисунок 1. Диаграмма "сущность-связь" учебной базы данных

Создание таблиц

Прежде чем приступить к заполнению таблиц, необходимо их создать. Сделать это можно одним из двух способов.

Первый способ заключается в использовании визуальных инструментов, которые предлагаются некоторыми средами разработки баз данных.

Второй способ заключается в самостоятельном написании запросов по созданию таблиц. Остановимся на этом подробнее.

Один из возможных синтаксисов запроса на создание таблицы выглядит следующим образом:

```
create table <имя таблицы>
(
  <имя столбца 1> <тип данных> <ограничения>,
  <имя столбца 2> <тип данных> <ограничения>,
  ...,
  <имя столбца n> <тип данных> <ограничения>,
  <дополнительное ограничение 1>,
  <дополнительное ограничение 2>,
  ...,
  <дополнительное ограничение n>
)
```

Перечислим основные ограничения, которые могут использоваться при создании таблиц:

- обязательное для заполнения поле (**`not null`**);
- поле, значение которого должно удовлетворять условию (**`check`**);
- уникальное поле, значение которого не может повторяться (**`unique`**);
- ограничение первичного ключа (**`CONSTRAINT <имя ограничения> PRIMARY KEY (<название столбца, на который накладывается ограничение>)`**);
- ограничение внешнего ключа (**`CONSTRAINT <имя ограничения> FOREIGN KEY (<название столбца, на который накладывается ограничение>) REFERENCES <имя таблицы на которую ссылается столбец>`**).

Также перечислим основные типы данных, которые чаще всего используются при создании таблиц:

- целое число (**`int`**)
- вещественное число (**`float`**)
- целое число с фиксированным количеством знаков после запятой (**`decimal`**)
- деньги (**`money`**)
- дата и время (**`datetime`**)
- строка фиксированной длины (**`char`**)
- строка переменной длины (**`varchar`**)

Приведём примеры использования этого синтаксиса при создании учебной базы данных.

Прежде чем начать, следует обратить внимание на правильный порядок создания таблиц. Например, нельзя создавать таблицу с внешними ключами до того, как будут созданы таблицы, на которые они ссылаются. Запишем запросы на создание таблиц учебной базы данных в нужном порядке.

Таблица **владельцы**:

```
create table clients
(
  id int not null,
  fio varchar(64) not null,
  sex char(1) not null check (sex in ('м', 'ж')),
  address varchar(64),
  phone varchar(16),
  birth date,
  CONSTRAINT clients_pk PRIMARY KEY (id)
)
```

Таблица **цвета**:

```
create table colors
(
  id int not null,
  title varchar(64) unique not null,
  CONSTRAINT colors_pk PRIMARY KEY (id)
)
```

Таблица **марки_автомобилей**:

```
create table brands
(
  id int not null,
  title varchar(64) unique not null,
  CONSTRAINT brands_pk PRIMARY KEY (id)
)
```

Таблица **услуги**:

```
create table services
(
  id int not null,
  title varchar(64) unique not null,
  price decimal(10, 2) not null check (price>0),
  CONSTRAINT services_pk PRIMARY KEY (id)
)
```

Таблица **специальности**:

```
create table professions
(
  id int not null,
  title varchar(64) unique not null,
  CONSTRAINT professions_pk PRIMARY KEY (id)
)
```

Таблица **мастера**:

```
create table masters
(
  id int not null,
  fio varchar(64) not null,
  profession_id int not null,
  CONSTRAINT masters_pk PRIMARY KEY (id),

  CONSTRAINT profession_fk FOREIGN KEY
    (profession_id) REFERENCES professions
)
```

Таблица **автомобили**:

```
create table cars
(
  id varchar(12) not null,
  p_date date not null,
  client_id int not null,
  color_id int not null,
  brand_id int not null,
  CONSTRAINT cars_pk PRIMARY KEY (id),

  CONSTRAINT client_fk FOREIGN KEY
    (client_id) REFERENCES clients,
  CONSTRAINT color_fk FOREIGN KEY
    (color_id) REFERENCES colors,
  CONSTRAINT brand_fk FOREIGN KEY
    (brand_id) REFERENCES brands
)
```

Таблица **журнал_оказанных_услуг**:

```
create table cashbook
(
  id int not null,
  s_date date not null,
  s_time int not null check (s_time>0),
  car_id varchar(12) not null,
  service_id int not null,
  master_id int not null,
  CONSTRAINT cashbook_pk PRIMARY KEY (id),

  CONSTRAINT car_fk FOREIGN KEY
    (car_id) REFERENCES cars,
  CONSTRAINT service_fk FOREIGN KEY
    (service_id) REFERENCES services,
  CONSTRAINT master_fk FOREIGN KEY
    (master_id) REFERENCES masters
)
```

Наполнение таблиц данными

Для наполнения таблиц данными может использоваться либо визуальный интерфейс, предоставляемый средой разработки базы данных, либо специальные операторы языка SQL.

Добавление данных

Общий синтаксис оператора добавления данных в таблицу:

```
INSERT INTO <имя таблицы>
(
  <имя столбца 1>,
  <имя столбца 2>,
  ...,
  <имя столбца n>
)
VALUES
(
  <значение 1>,
  <значение 2>,
  ...,
  <значение n>
)
```

Модификация данных

Общий синтаксис оператора модификации данных в таблице:

```
UPDATE <имя таблицы> SET
<имя столбца 1> = <выражение 1>,
<имя столбца 2> = <выражение 2>,
...,
<имя столбца n> = <выражение n>
WHERE <условие отбора модифицируемых записей>
```

Удаление данных

Общий синтаксис оператора удаления данных из таблицы:

```
DELETE FROM <имя таблицы>
WHERE <условие отбора удаляемых записей>
```

Пример заполнения таблиц учебной базы данных

таблица «clients»					
id	fio	sex	address	phone	birth
1	"Иванов И.И."	"м"	"Ульяновск"	"+7-999-888-77-66"	"1970-03-15"
2	"Петров П.П."	"м"	"Ульяновск"	"+7-999-555-44-33"	"1975-05-19"
3	"Сидоров С.С."	"м"	"Ульяновск"	"+7-999-222-11-11"	"1974-08-12"
4	"Краснова Д.С."	"ж"	"Самара"	"+7-888-333-22-22"	"1984-12-11"
5	"Потанина М.Н."	"ж"	"Саратов"	"+7-777-666-33-22"	"1965-01-23"
6	"Ковров А.П.	"м"	"Уфа"	"+7-666-555-33-33"	"1982-02-15"
7	"Маринина И.П."	"ж"	"Пенза"	"+7-444-888-99-99"	"1978-07-05"

таблица «colors»	
id	Title
1	"красный"
2	"зелёный"
3	"синий"
4	"чёрный"
5	"белый"
6	"жёлтый"
7	"серый"

таблица «brands»	
id	title
1	"Rolls-Royce"
2	"Lamborghini"
3	"Maybach"
4	"Maserati"
5	"Bugatti"
6	"Bentley"
7	"Ferrary"

таблица «cars»				
id	p_date	client_id	color_id	brand_id
"x666xx99rus"	"1990-01-01"	2	2	3
"a199aa73rus"	"1980-01-01"	1	1	7
"a464oo65rus"	"2003-01-01"	6	6	5
"в641ав53rus"	"2007-01-01"	7	7	5
"e123ee73rus"	"2000-01-01"	4	3	2
"p542pe33rus"	"2009-01-01"	6	7	5
"т378тт66rus"	"2008-01-01"	7	7	6
"т564не63rus"	"2005-01-01"	5	5	4
"у777уу77rus"	"1985-01-01"	3	4	1

Masters		
id	fio	profession_id
1	"Кириллов А.А."	1
2	"Таралин Б.В."	1
3	"Сухоруков В.Г."	2
4	"Борисов А.Д."	3
5	"Малинин К.Е."	4
6	"Пупкин И.А."	5
7	"Чугунов А.В."	6
8	"Веселов Т.К."	7

professions	
id	title
1	"специалист по электрике"
2	"специалист по дискам/покрышкам"
3	"специалист по ремонту кузова"
4	"специалист по звукоизоляции"
5	"специалист по работе с двигателем"
6	"специалист по покраске"
7	"специалист по чистке салона"

services		
id	title	price
1	"замена электрики"	3000.00
2	"балансировка колеса"	200.00
3	"замена колеса"	100.00
4	"рихтовка кузова"	5000.00
5	"звукоизоляция салона"	2000.00
6	"ремонт двигателя"	10000.00
7	"смена масла"	4000.00
8	"окраска кузова"	20000.00
9	"чистка салона"	600.00

cashbook					
id	s_date	s_time	car_id	service_id	master_id
1	"2009-01-10"	1	"a199aa73rus"	1	1
2	"2009-01-17"	2	"x666xx99rus"	1	2
3	"2009-02-03"	1	"y777yy77rus"	2	3
4	"2009-02-04"	10	"e123ee73rus"	7	6
5	"2009-03-02"	5	"x666xx99rus"	5	5
6	"2009-03-02"	3	"e123ee73rus"	6	6
7	"2009-03-02"	1	"e123ee73rus"	5	5

Построение запросов

Общий синтаксис построения запросов таков:

```
SELECT [ALL | DISTINCT]
{* | [имя_столбца [AS новое_имя]] [,...n]}
FROM имя_таблицы [AS] [псевдоним] [,...n]
[WHERE <критерии поиска>]
[GROUP BY  имя_столбца [,...n]]
[HAVING <критерии выбора групп>]
[ORDER BY имя_столбца [,...n]]
```

Ключевое слово **DISTINCT** необходимо указывать в том случае, если требуется удалить все дублирующиеся строки из результата выполнения запроса.

Ключевое слово **ALL** используется по умолчанию и применяется в том случае, если нужны все строки, которые будут получены в результате выполнения запроса.

После ключевого слова **SELECT** идёт перечень столбцов, которые должны фигурировать в результате выполнения запроса. Каждый столбец может получить псевдоним (для чего необходимы псевдонимы, будет объяснено ниже). Если требуется получить полный перечень столбцов из таблиц, которые используются в запросе, указывается специальная мнемоника «*****», на место которой в момент выполнения запроса будет подставлен полный перечень столбцов.

После ключевого слова **FROM** идёт перечень таблиц, которые участвуют в запросе. Каждая таблица тоже может получить псевдоним.

После ключевого слова **WHERE** записывается перечень условий, согласно которым будет производиться отбор строк в результате выполнения запроса.

Результат выполнения запроса может быть сгруппирован; условия группировки указываются после ключевого слова **GROUP BY**.

Аналогично ключевому слову **WHERE**, ключевое слово **HAVING** используется для указывания дополнительных условий на данные, которые должны получиться в результате выполнения запроса. Однако, в отличие от **WHERE**, после ключевого слова **HAVING** указываются условия на отбор групп данных.

После ключевого слова **ORDER BY** указывается перечень столбцов, по которым будет осуществляться сортировка, а также принцип организации этой сортировки. Для сортировки в прямом порядке используется ключевое слово **ASC**, а для сортировки в обратном порядке — ключевое слово **DESC**.

Важно учитывать порядок, в котором выполняются вышеперечисленные части SQL запроса:

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

Знание этого порядка помогает понять логику выполнения запроса и облегчить процесс оптимизации запроса.

Приведём примеры самых простых запросов.

Пусть мы хотим получить полный перечень всех клиентов, которые пользуются услугами автомастерской. Сделать это можно при помощи следующего запроса:

Пример 1. Получить полный перечень клиентов, которые пользуются услугами автомастерской.

```
SELECT * FROM clients
```

Усложним задание и напишем запрос, который должен выводить ФИО и адрес клиента. Для этого в запросе нужно указать те столбцы, которые нам необходимы.

Пример 2. Получить список фамилий и адресов клиентов, которые пользуются услугами автомастерской.

```
SELECT fio, address FROM clients
```

Теперь продемонстрируем пример использования ключевых слов **DISTINCT** и **ALL**. Когда ни одно из ключевых слов не указано, по умолчанию используется **ALL**. Если мы захотим написать запрос, который будет выводить перечень городов, в которых живут клиенты автомастерской, то следующий запрос несколько раз выведет «Ульяновск»:

Пример 3. Получить перечень городов, в которых живут клиенты, пользующиеся услугами автомастерской (неправильная реализация).

```
SELECT ALL address FROM clients
```

В этом случае мы дали команду на выбор всех строк из таблицы **clients** из столбца **address**. Если же нужен перечень городов без повторений, то следует использовать ключевое слово **DISTINCT**.

Пример 4. Получить перечень городов, в которых живут клиенты, пользующиеся услугами автомастерской (правильная реализация).

```
SELECT DISTINCT address FROM clients
```

При выполнении этого запроса будут получены только уникальные записи, что и требовалось в вербальной формулировке запроса.

Запросы с условиями

С помощью ключевого слова **WHERE** можно определять, какие строки данных из приведенных в списке **FROM** таблиц появятся в результате запроса. Рассмотрим пять основных типов условий поиска.

Сравнение

В языке SQL можно использовать следующие операторы сравнения:

=	Равенство
<	Меньше
>	Больше
<=	меньше или равно
>=	больше или равно
<>	не равно

Если нужно более сложное условие, оно может быть составлено при помощи логических операторов **AND**, **OR**, **NOT** и круглых скобок, определяющих очерёдность выполнения действий.

Пример 5. Вывести список услуг, цена на которые не превосходит 1000 рублей.

```
SELECT title FROM services where price<=1000
```

Пример 6. Вывести список услуг цена, на которые больше или равна 2000 и меньше или равна 10000.

```
SELECT title, price
FROM services
where price>=2000 AND price<=10000
```

Пример 7. Вывести список клиентов, проживающих в Ульяновске или Саратове.

```
SELECT fio FROM clients
where
address='Ульяновск' OR address='Саратов'
```

Диапазон

Оператор **BETWEEN** используется для поиска значения внутри некоторого интервала, задаваемого своими минимальным и максимальным значениями, при этом граничные значения включаются в диапазон поиска.

Пример 8. Вывести список услуг, цена на которые больше или равна 2000 и меньше или равна 10000.

```
SELECT title, price
FROM services
where price BETWEEN 2000 AND 10000
```

Пример 9. Вывести список услуг, цена на которые не лежит в диапазоне от 2000 до 10000.

```
SELECT title, price
FROM services
where price NOT BETWEEN 2000 AND 10000
```

Принадлежность множеству

Оператор **IN** используется для проверки того, равняется ли значение в ячейке таблицы одному из значений в предоставленном списке. При помощи оператора **IN** может быть достигнут тот же результат, что и в случае применения оператора **OR**, однако оператор **IN** выполняется быстрее.

Пример 10. Вывести список клиентов, проживающих в Ульяновске или Саратове.

```
SELECT fio FROM clients
where address IN ('Ульяновск', 'Саратов')
```

Оператор **IN** очень удобен в запросах с отрицательным условием:

Пример 11. Вывести список клиентов, проживающих не в Ульяновске и не в Саратове.

```
SELECT fio FROM clients
where address NOT IN ('Ульяновск', 'Саратов')
```

Список вариантов может быть не только статическим, т.е. жёстко заданным на момент составления запроса, но и динамическим, т.е. формироваться при помощи подзапроса в момент выполнения запроса (см. раздел «Подзапросы»).

Соответствие шаблону

С помощью оператора **LIKE** можно сравнить выражение с заданным шаблоном, в котором допускается использование символов-заменителей:

- Символ **%** — вместо этого символа может быть подставлено любое количество произвольных символов.
- Символ _ заменяет один символ строки.
- [] — вместо символа строки будет подставлен один из возможных символов, указанных в квадратных скобках.
- [^] — вместо соответствующего символа строки будут подставлены все символы, кроме указанных в квадратных скобках.

Пример 12. Вывести список автомобилей, у которых первой цифрой номера является цифра 7 или цифра 3.

```
SELECT id FROM cars
where id LIKE '_[37]%'
```

Значение NULL

Оператор **IS NULL** используется для сравнения текущего значения со значением **NULL** — специальным значением, указывающим на отсутствие любого значения. **NULL** — это не то же самое, что знак пробела (**пробел** — допустимый символ) или ноль (**0** — допустимое число). **NULL** отличается и от строки нулевой длины (пустой строки).

Пример 13. Вывести список клиентов, у которых не указан номер телефона.

```
SELECT fio FROM clients
where phone IS NULL
```

Пример 14. Вывести список клиентов, у которых указан номер телефона.

```
SELECT fio, phone FROM clients
where phone IS NOT NULL
```

Сортировка результатов

При выполнении SQL запроса результирующие строки никак не упорядочены, кроме того, один и тот же запрос, будучи запущенным несколько раз подряд, может выдавать строки в разном порядке. Поэтому, если требуется, чтобы результат выполнения запроса был отсортирован по некоторому столбцу (группе столбцов), нужно использовать ключевое слово **ORDER BY**.

Пример 15. Вывести список клиентов, у которых указан номер телефона, при этом отсортировав записи по фамилии.

```
SELECT fio, phone FROM clients
where phone IS NOT NULL
ORDER BY fio ASC
```

Пример 16. Вывести список клиентов, сгруппировав их по городу проживания и отсортировав сначала по городу, а внутри города — по фамилии.

```
SELECT address, fio FROM clients
ORDER BY address ASC, fio ASC
```

Пример 17. Вывести список предоставляемых услуг, отсортировав их по цене (начиная с самых дорогостоящих).

```
SELECT title, price FROM services
ORDER BY price DESC
```

Соединение и объединение таблиц в запросе

Все SQL запросы, которые нам встречались до текущего момента, обращались в предложении **FROM** только к одной таблице. А как можно построить запрос, который будет обращаться к столбцам, расположенным в разных таблицах? Для этого в запросе необходимо указать принцип соединения таблиц. Ведь если этого не сделать, то соединение таблиц будет строиться по принципу декартова произведения, а декартово произведение содержит в себе слишком много лишней информации.

Рассмотрим соединение таблиц по принципу декартового произведения.

Пусть нам необходимо составить таблицу, в которой бы первый столбец содержал фамилии владельцев, а второй — регистрационные номера их автомобилей. Эти данные разнесены по разным таблицам: **cars** и **clients**.

Если мы запишем запрос на выборку в следующем виде:

```
SELECT fio, id FROM cars, clients
```

то первое, с чем мы столкнёмся, будет сообщение о том, что этот запрос в принципе не сможет быть выполнен. Первая причина его ошибочности заключается в том, что столбец **id** является неоднозначным. Он присутствует в обеих таблицах, и интерпретатор SQL в момент исполнения запроса не сможет самостоятельно сделать выбор в пользу одной из таблиц и выдаст ошибку. Решить эту проблему можно за счёт использования полного формата именования столбцов:

```
<имя таблицы>.<имя столбца>
```

Для себя можно сформулировать достаточно простое правило: использование полного формата именования столбцов является более желательным, т.к. устраняет неоднозначности и неопределённости, и, кроме всего прочего, это упрощает процесс чтения SQL запроса.

При использовании полного формата именования столбцов запрос будет выглядеть следующим образом:

```
SELECT clients.fio, cars.id FROM cars, clients
```

Однако это не решит всех проблем. Дело в том, что если после предложения **FROM** имена таблиц перечислены через запятую без указания принципа их соединения, то они будут соединены по принципу декартового произведения, то есть к каждой строке первой таблицы будут по очереди приписаны все столбцы второй таблицы. В терминах нашего примера это будет означать, что к первому клиенту будут приписаны регистрационные номера всех автомобилей, затем ко второму клиенту будут приписаны регистрационные номера всех автомобилей и т.д. В результате будет получена таблица, количество строк в которой будет равняться произведению количеств строк двух исходных таблиц, а количество столбцов в результирующей таблице будет равнять сумме количеств столбцов исходных таблиц. Конечно же, ненужные строчки могут быть отсечены при помощи предложения **WHERE**:

```
SELECT clients.fio, cars.id
FROM cars, clients
WHERE cars.client_id=clients.id
```

В результате выполнения этого запроса останутся только те строки, где фамилия клиента указана напротив регистрационного номера его автомобиля. Но, несмотря на то, что в результате будет небольшое количество строк, на момент соединения таблиц в предложении **FROM** количество строк соединения может быть довольно велико. Если бы в базе данных было 1000 клиентов и 1000 автомобилей, то в результате было бы 1000 строк. А вот в

соединении до выполнения предложения **WHERE** было бы 1000000 строк, что достаточно много даже для современных компьютеров. Поэтому в чистом виде декартово произведение используется достаточно редко, а вместо него применяется внутреннее или внешнее соединение таблиц.

Внутреннее соединение

Для внутреннего соединения таблиц используется ключевое слово **INNER JOIN**. Запишем общий формат использования конструкции **INNER JOIN**:

```
SELECT … FROM
<имя таблицы 1> INNER JOIN <имя таблиц 2>
ON <условие соединения таблиц>
```

В простейшем случае условие соединения таблиц выглядит как равенство соответствующих столбцов.

Пример 18. Получить информацию о владельцах и регистрационных номерах их автомобилей (этот пример аналогичен приведённому выше, но строится не на основе декартового произведения).

```
SELECT clients.fio, cars.id
FROM cars INNER JOIN clients
ON cars.client_id=clients.id
```

В результате выполнения операции внутреннего соединения в соединение попадут только те строки, которые удовлетворяют условию соединения таблиц.

А как быть в том случае, если соединить нужно не две, а три таблицы? Принцип используется тот же, но при соединении большего количества таблиц становится важен порядок их следования. В середине должна оказаться таблица, которая будет связываться с двумя остальными.

Пример 19. Получить информацию о владельцах, регистрационных номерах и марках их автомобилей.

```
SELECT clients.fio, cars.id, brands.title
FROM clients INNER JOIN
(cars INNER JOIN brands ON cars.brand_id=brands.id)
ON cars.client_id=clients.id
```

В этом примере таблица **cars** склеивается с таблицей **clients**, плюс к этому таблица **cars** склеивается с таблицей **brands**. Поэтому она (таблица **cars**) должна оказаться посередине, ведь мы не можем напрямую соединить таблицы **clients** и **brands**. Иными словами сначала произойдёт соединение таблиц **cars** и **brands**, потому что их соединение осуществляется в скобочках, а затем к полученному соединению будет подклеена таблица **clients**. Альтернативный синтаксис записи этого запроса выглядит следующим образом:

```
SELECT clients.fio, cars.id, brands.title
FROM clients
INNER JOIN cars ON cars.client_id=clients.id
INNER JOIN brands ON cars.brand_id=brands.id
```

Но порядок следования таблиц важен и в этом случае, так, второе предложение **INNER JOIN** не может появиться раньше первого.

Перед тем, как перейти к понятию внешнего соединения таблиц, рассмотрим ещё один пример.

Пример 20. Получить информацию о датах обращения в автосервис владельцев автомобилей, при этом указав регистрационный номер и марку автомобиля.

```
SELECT
cashbook.s_date, clients.fio, cars.id, brands.title
FROM cars
INNER JOIN cashbook ON cashbook.car_id=cars.id
INNER JOIN clients ON cars.client_id=clients.id
INNER JOIN brands ON cars.brand_id=brands.id
```

В результате будут выведены сведения только от тех клиентах и автомобилях, которым хотя бы раз оказывались некоторые услуги. Если ни одна услуга ни разу не оказывалась, то при внутреннем соединении строка, содержащая информацию об автомобиле, будет проигнорирована, потому как ей не будет найдено соответствие в таблице, содержащей информацию об указанных услугах.

Внешнее соединение

Если же мы хотим вывести и тех клиентов, которым ни разу ни одна услуга не оказывалась, то сделать это можно при помощи внешнего соединения. Чем отличается внешнее соединение от внутреннего? Во внутреннем соединении все таблицы являются равноправными, и не важно, как рассматривать операцию соединения: как приклеивание таблицы номер два к таблице номер один или как приклеивание таблицы номер один к таблице номер два. В случае же внешнего соединения одна из таблиц считается ведущей, а другая подчинённой. Поэтом рассматривают левое внешнее соединение, реализуемое при помощи конструкции **`LEFT JOIN`**, и правое внешнее соединение, реализуемое при помощи конструкции **`RIGHT JOIN`**. В первом случае главной является первая таблица, во втором случае вторая.

Важным отличием внешнего соединения от внутреннего является тот факт, что в соединение попадут все строки главной таблицы, даже если им не будет найдено соответствие в подчинённой таблице. А что же тогда будет выведено на том месте, где должны появиться связанные данные из второй таблицы? Там появится в прямом смысле слова «ничего», т.е. значение **`NULL`**.

Пример 21. Получить информацию о датах обращения в автосервис владельцев автомобилей, при этом указав, регистрационный номер и марку автомобиля. Вывести информацию о владельце, даже если он ни разу не обращался в автосервис.

```
SELECT
cashbook.s_date, clients.fio, cars.id, brands.title
FROM cars
LEFT JOIN cashbook ON cashbook.car_id=cars.id
INNER JOIN clients ON cars.client_id=clients.id
INNER JOIN brands ON cars.brand_id=brands.id
```

Использование ключевого слова UNION

Ключевое слово **UNION** тоже используется для объединения, но не таблиц, а результатов выполнения запросов на выборку. При этом объединяемые части должны быть совместимы, т.е. иметь одинаковое количество полей с совпадающими типами данных. Так как результатом выполнения запроса на выборку данных является таблица, то этот результат также может именоваться таблицей наравне с исходными таблицами. Тогда в этой терминологии можно сказать, что объединением двух таблиц является таблица, содержащая все строки, которые имеются в первой таблице, во второй таблице или в обеих таблицах сразу.

Пример 22. Вывести список клиентов, проживающих в Ульяновске или Саратове.

```
SELECT fio FROM clients
where address='Ульяновск'
UNION
SELECT fio FROM clients
where address='Саратов'
```

В этом примере в первой части запроса формируется список клиентов, проживающих в Ульяновске, во второй части запроса формируется список клиентов, проживающих в Саратове, затем обе эти части объединяются в одну общую таблицу при помощи ключевого слова **UNION**. Объединение может быть корректно проведено, т.к. количество столбцов и их тип совпадают.

Разумеется, приведённый выше пример может быть написан при помощи более простых средств, но возможности конструкции **UNION** пока нельзя продемонстрировать в полном объеме — для этого требуется изучить построение вычисляемых полей.

Построение вычисляемых полей

Во всех запросах, рассмотренных выше, в результат выполнения запроса на выборку столбцы данных попадали в неизменном виде. Но вполне естественно, что при написании сложных запросов нам понадобится производить над ними некоторые вычисления, а также формировать новые столбцы. Все эти средства обеспечиваются языком SQL.

В общем случае для создания вычисляемого поля в предложении **SELECT** следует указать некоторое выражение языка SQL. В этих выражениях применяются арифметические операции сложения, вычитания, умножения и деления, а также встроенные функции языка SQL. При построении сложных выражений могут понадобиться скобки.

Ещё одним важным моментом, который возникает при построении вычисляемых полей, являются названия этих полей. Если в запросе на выборку присутствует столбец в неизменном виде, то в результате выполнения запроса этот столбец будет называться так же, как и в исходной таблице. А как быть, если этого столбца раньше не было, и он был сформирован в результате выполнения запроса? В этом случае необходимо вспомнить, что при определении общего синтаксиса SQL запроса было введено такое понятие как псевдоним. При помощи псевдонимов можно давать имена не только вновь сформированным столбцам, но и переименовывать существующие столбцы. В каком случае может понадобиться переименование существующего столбца? Например, нужно построить запрос, в котором должна фигурировать фамилия клиента (**fio**) и фамилии мастера (**fio**). При написании запроса можно разделить названия этих столбцов за счёт использования полного синтаксиса именования столбцов, но в результирующей таблице получится два разных столбца с одинаковыми названиями. Некоторые СУБД могут обеспечить автоматическое переименование столбцов в подобных ситуациях, например, первый столбец будет называться (**fio1**), а второй столбец (**fio2**), но это не всегда удобно. Поэтому лучше это сделать самостоятельно, указав соответствующий псевдоним.

Пример 23. Для каждой оказанной услуги указать дату оказания, количество затраченных часов, стоимость одного часа работы, итоговую стоимость и регистрационный номер автомобиля, над которым проводилась работа.

```
SELECT
cashbook.car_id AS регистрационный_номер_машины,
services.title AS вид_услуги,
cashbook.s_date AS дата_оказания_услуги,
cashbook.s_time AS затраченное_время_в_часах,
services.price AS цена_одного_часа,
services.price*cashbook.s_time AS итого
FROM services INNER JOIN cashbook ON
services.id=cashbook.service_id
```

Важно ещё упомянуть тот факт, что при именовании столбцов использование символа пробела и символов национальных алфавитов является нежелательным, но допустимым. Для того чтобы содержащее пробелы имя столбца интерпретировалось как единое целое, оно может заключаться в квадратные скобки или символы кавычек.

Использование агрегатных функций

С помощью итоговых (агрегатных, или обобщающих) функций в рамках SQL-запроса можно получить обобщающие статистические сведения о множестве отобранных значений.

Пользователю доступны следующие основные итоговые функции:

- **COUNT(Выражение)** — возвращает количество записей в выходном наборе SQL-запроса;
- **MIN/MAX(Выражение)** — возвращает наименьшее и наибольшее из множества значений в некотором поле запроса;
- **AVG(Выражение)** — возвращает среднее значение множества значений, хранящихся в определенном поле отобранных запросом записей.
- **SUM(Выражение)** — возвращает сумму множества значений, содержащихся в определенном поле отобранных запросом записей.

Чаще всего в качестве выражения выступают отдельные столбцы, но также могут записываться и более сложные выражения, содержащие в качестве аргументов имена столбцов, принадлежащих различным таблицам.

Функции **MIN**, **MAX** и **COUNT** применимы как к числовым, так и к нечисловым полям. Функции **AVG** и **SUM** могут применяться только к числовым полям.

Если до применения обобщающей функции необходимо исключить дублирующиеся значения, следует перед именем столбца в определении функции поместить ключевое слово **DISTINCT**. Оно не имеет смысла для функций **MIN** и **MAX**, однако его использование может повлиять на результаты выполнения остальных функций.

Очень важно отметить, что итоговые функции могут использоваться только в списке предложения **SELECT** и в составе предложения **HAVING**. Ещё одно правило использования итоговых функций состоит в том, что все столбцы, которые входят в предложение **SELECT** и не входят в итоговую функцию, должны войти в предложение **GROUP BY**, обеспечивая объединение данных в группы. Обратное правило звучит несколько иначе: в предложение **GROUP BY** могут входить столбцы, которые не входят в предложение **SELECT**.

Пример 24. Определить первого по алфавиту сотрудника автомастерской.

```
SELECT MIN(fio) AS первый_по_алфавиту
FROM masters
```

При использовании функции **COUNT** можно использовать мнемонику *, т.к. в некоторых случаях неважно, на каком столбце вычисляется эта функция. В этом примере имеет значение только количество строк в результирующем запросе, поэтому имя столбца может быть не указано.

Пример 25. Определить, сколько раз в автосервисе оказывались услуги автомобилю с регистрационным номером "x666xx99rus".

```
SELECT COUNT(*) AS количество_обращений
FROM cashbook
WHERE car_id='x666xx99rus'
```

Явное указание имени столбца для функции **COUNT** может быть актуальным только при использовании внешнего объединения таблиц, где количество значений в каждом столбце может быть различно.

Пример 26. Определить среднюю стоимость услуг, оказываемых в автосервисе.

```
SELECT AVG(price) AS средняя_цена
FROM services
```

Следующий пример показывает, каким образом аргументом итоговой функции может быть не отдельный столбец, а более сложное выражение.

Пример 27. Определить, на какую сумму были оказаны услуги автомобилю с регистрационным номером "x666xx99rus".

```
SELECT
SUM(cashbook.s_time*services.price) AS сумма
FROM
cashbook INNER JOIN services
ON cashbook.service_id=services.id
WHERE car_id='x666xx99rus'
```

Ограничения на группировку данных

Запрос, в котором присутствует ключевое слово **GROUP BY**, называется группирующим запросом, поскольку в нем группируются данные, полученные в результате выполнения операции **SELECT**. Для каждой отдельной группы создается единственная суммарная строка.

Стандарт SQL требует, чтобы предложение **SELECT** и фраза **GROUP BY** были тесно связаны между собой. При наличии в операторе **SELECT** ключевого слова **GROUP BY** каждый элемент списка в предложении **SELECT** должен иметь единственное значение для всей группы.

Если совместно с **GROUP BY** используется предложение **WHERE**, то оно обрабатывается первым, а группированию подвергаются только те строки, которые удовлетворяют условию поиска.

Ещё одной подсказкой для использования ключевого слова **GROUP BY** являются следующие формулировки в вербальном описании запроса: «для каждого…», «для каждой…»

Пример 28. Для каждого клиента определить количество автомобилей, которыми он владеет.

```
SELECT
clients.fio AS ФИО,
COUNT(*) AS количество_автомобилей
FROM
clients INNER JOIN cars
ON clients.id=cars.client_id
GROUP BY clients.fio
```

Столбец **clients.fio** не входит в итоговую функцию, но он входит в предложение **SELECT**, кроме всего прочего, в тексте запроса используется ключевая фраза «для каждого клиента», следовательно, этот столбец должен войти в предложение **GROUP BY**.

С целью сокращения длины запроса могут использоваться псевдонимы не только для имён столбцов, но и для имён таблиц (это не единственное применение механизма псевдонимов таблиц — его основное назначение в именовании динамически сформированных таблиц; этот вопрос будет рассмотрен в следующем разделе). Перепишем пример 28, используя псевдонимы для таблиц **cars** и **clients**.

Пример 29. Для каждого клиента определить количество автомобилей, которыми он владеет.

```
SELECT
CL.fio AS ФИО,
count(*) AS количество_автомобилей
FROM
clients AS CL
INNER JOIN cars AS CR
ON CL.id=CR.client_id
GROUP BY CL.fio
```

Возможно, в этом запросе выгода от сокращения записи и не видна, но в действительно сложных запросах сокращения заметно упрощают чтение текста запроса.

Пример 30. Определить, на какую сумму каждому автомобилю были оказаны услуги.

```
SELECT
C.car_id AS регистрационный_номер_автомобиля,
SUM(C.s_time*S.price) AS сумма
FROM
cashbook AS C INNER JOIN services AS S
ON C.service_id=S.id
GROUP BY C.car_id
```

Отметим, что запрос из примера 27, в котором рассчитывалась сумма оказанных услуг для конкретного автомобиля, и запрос из примера 30 очень похожи.

Пример 31. Определить, на какую сумму каждому автовладельцу были оказаны услуги.

```
SELECT
CL.fio AS ФИО,
SUM(CS.s_time*S.price) AS сумма
FROM
cashbook AS CS
INNER JOIN services AS S ON CS.service_id=S.id
INNER JOIN cars AS CR ON CS.car_id=CR.id
INNER JOIN clients AS CL ON CL.id=CR.client_id
GROUP BY CL.fio
```

Несмотря на возрастающую сложность запроса и количество объединяемых таблиц, общий принцип построения остаётся прежним: всё, что не вошло в итоговую функцию, должно войти в предложение **GROUP BY**.

Пример 32. Определить выручку автомастерской за каждый месяц.

```
SELECT MONTH(CS.s_date) AS месяц,
SUM(CS.s_time*S.price) AS сумма
FROM
cashbook AS CS
INNER JOIN services AS S ON CS.service_id=S.id
GROUP BY MONTH(CS.s_date)
```

Для получения номера месяца используется функция **MONTH()**, которая возвращает целое число в диапазоне от **1** до **12**. Аналогично могут использоваться функции **YEAR()** и **DAY()**, которые возвращают соответственно номер года и номер дня.

Несмотря на то, что столбец **MONTH(CS.s_date)** получил псевдоним **месяц**, им нельзя пользоваться внутри этого запроса; и в предложении **GROUP BY** приходится записать то же самое выражение, которое использовалось для построения вычисляемого столбца.

При помощи **HAVING** отбираются все предварительно сгруппированные посредством **GROUP BY** блоки данных, удовлетворяющие заданным в **HAVING** условиям.

Условия в **HAVING** отличаются от условий в **WHERE**:

- **HAVING** исключает из результирующего набора данных группы с результатами агрегированных значений;
- **WHERE** исключает из расчета агрегатных значений по группировке записи, не удовлетворяющие условию;
- в условии поиска **WHERE** нельзя задавать агрегатные функции.

Пример 33. Определить клиентов, которые владеют более чем одним автомобилем.

```
SELECT
clients.fio AS ФИО,
COUNT(*) AS количество_автомобилей
FROM
clients INNER JOIN cars
ON clients.id=cars.client_id
GROUP BY clients.fio
HAVING COUNT(*)>1
```

В этом примере мы вновь сталкиваемся с той же проблемой, что и в примере 32: несмотря на то, что для вновь сформированного столбца был обозначен псевдоним, мы не можем использовать этот псевдоним в предложении **HAVING** и вынуждены повторить запись выражения, которое лежит в основе создания столбца.

Пример 34. Определить месяцы, в которые выручка автомастерской не превышала 10000.

```
SELECT
MONTH(CS.s_date) AS месяц,
SUM(CS.s_time*S.price) AS сумма
FROM
cashbook AS CS
INNER JOIN services AS S ON CS.service_id=S.id
GROUP BY MONTH(CS.s_date)
HAVING SUM(CS.s_time*S.price)<=10000
```

Подзапросы

Очень часто получить необходимые данные при помощи одного запроса не представляется возможным. В этих случаях используется такой механизм, как вложенные запросы, или подзапросы.

Внутренний подзапрос представляет собой такой же оператор **SELECT**, как и внешний. Внешний оператор **SELECT** имеет доступ к результатам, полученным во внутреннем запросе, а внутренний запрос может обращаться не только к данным своих таблиц, но и к таблицам, которые используются во внешнем запросе.

Подзапрос — это инструмент создания временной таблицы, содержимое которой извлекается и обрабатывается внешним оператором.

Текст подзапроса должен быть заключен в скобки. К подзапросам применяются следующие правила и ограничения:

- Внутренние запросы обязательно должны быть помещены после оператора сравнения.
- Ключевое слово **ORDER BY** не используется, хотя и может присутствовать во внешнем подзапросе.
- Список в предложении **SELECT** внутреннего запроса должен состоять из имен отдельных столбцов или составленных из них выражений — за исключением случая, когда в подзапросе присутствует ключевое слово **EXISTS**.
- По умолчанию имена столбцов в подзапросе относятся к таблице, имя которой указано в предложении **FROM** внутреннего запроса. Однако, допускается ссылка и на столбцы таблицы, указанной во фразе **FROM** внешнего запроса.

Подзапрос может встречаться в одной из следующих секций:

- в предложении **SELECT**,
- в предложении **WHERE**,
- в предложении **HAVING**,
- в предложении **FROM**.

Подзапросы, возвращающие единичные значения

Выделяют скалярные подзапросы, возвращающие единичные значения, и табличные подзапросы, возвращающие множественные значения. Рассмотрим скалярные подзапросы.

Пример 35. Определить самого молодого владельца автомобиля.

```
SELECT fio
FROM clients
WHERE birth=
(SELECT MIN(birth) FROM clients)
```

В этом запросе сначала будет выполнен внутренний подзапрос, который вычисляет минимальный возраст клиента. Но нам нужно не само значение минимального возраста, а фамилия клиента, который обладает минимальным возрастом. Фамилия определяется уже на уровне внешнего запроса.

Наиболее часто встречающаяся ошибка при написании подобных запросов состоит в том, что студенты сортируют клиентов по возрасту и выбирают самую верхнюю строчку из отсортированной таблицы. На самом деле они совершают две грубые ошибки. Первая

состоит в том, что операция поиска минимума или максимума имеет гораздо меньшую сложность по сравнению с операцией сортировки. Вторая состоит в том, что может существовать несколько строк таблицы, удовлетворяющих условию минимума. Если произвести сортировку и взять в ответ только верхнюю строку, то остальные строки будут утеряны.

Пример 36. Определить услуги, стоимость которых превосходит среднее, также вывести превышение над средним.

```
SELECT title AS название_услуги,
price AS цена,
price-(SELECT AVG(price) FROM services) AS
превышение
FROM services
WHERE price>
(SELECT AVG(price) FROM services)
```

В этом примере подзапрос на вычисление среднего арифметического встречается два раза: один раз в предложении **SELECT** как часть арифметического выражения для вычисления превышения над средним арифметическим, а другой раз в предложении **WHERE** как часть условия для отбора только тех строк, которые отвечают поставленному условию.

Пример 37. Определить даты, когда оказывалась самая дорогая услуга.

```
SELECT DISTINCT
s_date
FROM cashbook
WHERE service_id=
(
  SELECT id FROM services
  WHERE price=
  (SELECT MAX(price) FROM services)
)
```

Пример 37 демонстрирует, что уровень вложенности запросов может быть больше, чем один. Подзапрос второго уровня определяет стоимость самой дорогой услуги. Подзапрос первого уровня определяет **id** самой дорогой услуги. И, наконец, запрос самого верхнего уровня определяет те дни, когда оказывалась услуга с выбранным **id**. Важным нюансом в написании этого запроса является использование ключевого слова **DISTINCT**, т.к. в один и тот же день самая дорогая услуга может оказываться более одного раза.

Пример 38. Определить клиентов, которым оказывалась самая дорогая услуга.

```
SELECT DISTINCT
CL.fio
FROM
clients AS CL INNER JOIN cars AS CR
ON CL.id=CR.client_id
```

```
INNER JOIN cashbook AS CS
ON CS.car_id=CR.id
INNER JOIN services AS S
ON S.id=CS.service_id
WHERE service_id=
(
  SELECT id FROM services
  WHERE price=
  (SELECT MAX(price) FROM services)
)
```

Как и в примере 37, в примере 38 необходимо использовать ключевое слово **DISTINCT**, т.к. одному и тому же клиенту самая дорогая услуга может оказываться более одного раза. Наконец, рассмотрим пример 39.

Пример 39. Определить клиентов, для которых средняя цена услуг (её необходимо вывести на экран), которыми они пользуются, выше аналогичного показателя для полного перечня доступных услуг.

```
SELECT CL.fio AS ФИО,
AVG(S.price) AS средняя_цена_услуги
FROM
clients AS CL INNER JOIN cars AS CR
ON CL.id=CR.client_id
INNER JOIN cashbook AS CS
ON CS.car_id=CR.id
INNER JOIN services AS S
ON S.id=CS.service_id
GROUP BY CL.fio
HAVING AVG(S.price)>
(SELECT AVG(price) FROM services)
```

Подзапросы, возвращающие множественные значения

Во многих случаях значение, подлежащее сравнению в предложениях **WHERE** или **HAVING**, представляет собой не одно, а несколько значений. Вложенные подзапросы генерируют непоименованное промежуточное отношение — временную таблицу. Оно может использоваться только в том месте, где появляется в подзапросе.

Применяемые к подзапросу операции основаны на тех операциях, которые, в свою очередь, применяются к множеству, а именно:

- **{WHERE | HAVING} выражение [NOT] IN (подзапрос);**
- **{WHERE | HAVING} выражение оператор_сравнения {ALL | ANY} (подзапрос);**
- **{WHERE | HAVING } [NOT] EXISTS (подзапрос);**

Несмотря на то, что временный запрос является неименованной таблицей, мы можем дать ей имя, которое будет доступно в рамках запроса более высокого уровня.

Пример 40. Определить клиентов, которым были оказаны услуги на сумму, превышающую 10000.

```
SELECT T.ФИО
FROM
(SELECT
CL.fio AS ФИО,
SUM(CS.s_time*S.price) AS сумма
FROM
cashbook AS CS
INNER JOIN services AS S ON CS.service_id=S.id
INNER JOIN cars AS CR ON CS.car_id=CR.id
INNER JOIN clients AS CL ON CL.id=CR.client_id
GROUP BY CL.fio) AS T
WHERE T.сумма>10000
```

В примере 40 результат выполнения подзапроса содержит более одной строки и более одного столбца, т.е. этот результат можно интерпретировать как таблицу. Для того, чтобы с этой таблицей можно было работать, дадим ей имя. Это может быть сделано при помощи псевдонима. В этом примере временная таблица получает псевдоним **T,** и в рамках запроса верхнего уровня можно обращаться к её столбцам на основании тех псевдонимов, которые им были даны на уровне внутреннего запроса.

Напомним, что оператор **IN** используется для сравнения некоторого значения со списком значений, при этом проверяется, входит ли значение в предоставленный список или сравниваемое значение не является элементом представленного списка. Этот оператор уже рассматривался ранее, но список потенциальных значений был статичный. Рассмотрим случаи, когда этот список может генерироваться динамически при помощи вложенных подзапросов.

Пример 41. Определить клиентов, которым оказывалась самая дорогая услуга.

```
SELECT fio FROM clients
WHERE clients.id IN
(
  SELECT client_id FROM cashbook
  INNER JOIN services
  ON cashbook.service_id=services.id
  INNER JOIN cars
  ON cashbook.car_id=cars.id
  WHERE price=
  (SELECT MAX(price) FROM services)
)
```

Запрос из примера 41 уже рассматривался в примере 38, но там он был реализован более сложным способом. А вот пример 42 без конструкции **IN** решить уже будет затруднительно.

Пример 42. Определить клиентов, которым оказывалась самая дорогая услуга и которым никогда не оказывалась самая дешёвая услуга.

```
SELECT fio FROM clients
WHERE clients.id IN
(
  SELECT client_id
  FROM cashbook
  INNER JOIN services
  ON cashbook.service_id=services.id
  INNER JOIN cars
  ON cashbook.car_id=cars.id
  WHERE price=
  (SELECT MAX(price) FROM services)
)
AND
clients.id NOT IN
(
  SELECT client_id
  FROM cashbook
  INNER JOIN services
  ON cashbook.service_id=services.id
  INNER JOIN cars
  ON cashbook.car_id=cars.id
  WHERE price=
  (SELECT MIN(price) FROM services)
)
```

Ключевые слова **ANY** и **ALL** могут использоваться с подзапросами, возвращающими один столбец чисел.

Если подзапросу будет предшествовать ключевое слово **ALL**, условие сравнения считается выполненным только тогда, когда оно выполняется для всех значений в результирующем столбце подзапроса.

Если записи подзапроса предшествует ключевое слово **ANY**, то условие сравнения считается выполненным, если оно выполняется хотя бы для одного из значений в результирующем столбце подзапроса.

Если в результате выполнения подзапроса получено пустое значение, то для ключевого слова **ALL** условие сравнения будет считаться выполненным, а для ключевого слова **ANY** — невыполненным.

Пример 43. Определить автовладельцев, которым были оказаны услуги на самую большую сумму.

```
SELECT
CL.fio AS ФИО
FROM
cashbook AS CS
INNER JOIN services AS S
ON CS.service_id=S.id
INNER JOIN cars AS CR
ON CS.car_id=CR.id
INNER JOIN clients AS CL
ON CL.id=CR.client_id
GROUP BY CL.fio
HAVING SUM(CS.s_time*S.price)>=ALL
(
  SELECT
  SUM(CS.s_time*S.price) AS сумма
  FROM
  cashbook AS CS
  INNER JOIN services AS S
  ON CS.service_id=S.id
  INNER JOIN cars AS CR
  ON CS.car_id=CR.id
  INNER JOIN clients AS CL
  ON CL.id=CR.client_id
  GROUP BY CL.fio
)
```

В результате выполнения подзапроса формируется столбец сумм, которые были уплачены каждым из автовладельцев. Следует обратить внимание на то, что сама фамилия не участвует в предложение **SELECT**, но участвует в предложении **GROUP BY**. Отметим также, что в запросе верхнего уровня итоговая функция не присутствует в предложении **SELECT**, но присутствует в предложении **HAVING**.

Важен также тот факт, что для сравнения используется операция «больше или равно», а не операция «больше». Если применить операцию «больше», то запрос всегда будет возвращать пустое множество, т.к. не может быть клиентов, которые потратили больше себя.

Подобный принцип построения запроса может быть не самым эффективным, т.к. при его выполнении расчёт и группировка итоговых сумм будет производиться дважды: первый раз во внутреннем подзапросе, а второй раз в запросе верхнего уровня. Кроме того, сам текст запроса содержит достаточно большие повторяющиеся куски. В целом эта задача могла бы быть более элегантно решена при помощи пользовательских просмотров.

Рассмотрим еще один пример.

Пример 44. Определить автовладельцев, которым были оказаны услуги на сумму, большую, чем сумма, уплаченная хотя бы одним жителем Ульяновска.

```
SELECT
CL.fio AS ФИО
FROM
cashbook AS CS
INNER JOIN services AS S
ON CS.service_id=S.id
INNER JOIN cars AS CR
ON CS.car_id=CR.id
INNER JOIN clients AS CL
ON CL.id=CR.client_id
GROUP BY CL.fio
HAVING
SUM(CS.s_time*S.price)>ANY
(
  SELECT
  SUM(CS.s_time*S.price) AS сумма
  FROM
  cashbook AS CS
  INNER JOIN services AS S
  ON CS.service_id=S.id
  INNER JOIN cars AS CR
  ON CS.car_id=CR.id
  INNER JOIN clients AS CL
  ON CL.id=CR.client_id
  WHERE CL.address='Ульяновск'
  GROUP BY CL.fio
)
```

В этом примере во внутреннем запросе формируется столбец сумм, которые были уплачены жителями Ульяновска, а во внешнем запросе выбираются те, для которых этот показатель является более высоким хотя бы в одной позиции.

Ключевые слова **EXISTS** и **NOT EXISTS** предназначены для использования только совместно с подзапросами. Результат их обработки представляет собой логическое значение **TRUE** или **FALSE**. Для ключевого слова **EXISTS** результат равен **TRUE** в том и только в том случае, если в возвращаемой подзапросом результирующей таблице присутствует хотя бы одна строка. Если результирующая таблица подзапроса пуста, результатом обработки операции **EXISTS** будет значение **FALSE**.

Для ключевого слова **NOT EXISTS** используются правила обработки, обратные по отношению к ключевому слову **EXISTS**. Поскольку по ключевым словам **EXISTS** и **NOT EXISTS** проверяется лишь наличие строк в результирующей таблице подзапроса, то эта таблица может содержать произвольное количество столбцов.

Пример 45. Определить автовладельцев, которым была оказана хотя бы одна услуга.

```
SELECT fio
FROM clients
WHERE EXISTS
(SELECT * FROM cashbook
INNER JOIN cars
ON cars.id=cashbook.car_id
WHERE client_id=clients.id)
```

Пример 46. Определить автовладельцев, которым не было оказано ни одной услуги.

```
SELECT fio
FROM clients
WHERE NOT EXISTS
(SELECT * FROM cashbook
INNER JOIN cars
ON cars.id=cashbook.car_id
WHERE client_id=clients.id)
```

В этих двух примерах используется приём, при котором из внутреннего запроса осуществляется обращение к столбцам запроса верхнего уровня. Во внутреннем подзапросе происходит обращение к таблице **cashbook** и делается попытка выбрать из неё все строчки, относящиеся к некоторому фиксированному текущему клиенту. Затем в запросе верхнего уровня фиксируется следующий клиент, и для него повторяется процесс выполнения внутреннего запроса.