Самоучитель

Курс SQL
Базы данных
ORACLE

Илья Хохлов

Илья Хохлов

Самоучитель. Курс SQL. Базы данных. ORACLE

«Автор»
1985

Хохлов И. Л.

Самоучитель. Курс SQL. Базы данных. ORACLE /
И. Л. Хохлов — «Автор», 1985

Систематичный и последовательный курс изучения языка SQL с первых шагов до уровня специалиста с трёхлетним стажем. Весь материал сопровождается практическими примерами и самостоятельными задачами, с вариантами их решений для самоконтроля. Книга написана простым и понятным языком.

Содержание

Илья Хохлов
Самоучитель. Курс SQL. Базы данных. ORACLE

Введение

Здравствуй, уважаемый читатель!

Вероятно, ты уже немного знаешь, что такое SQL и для чего он нужен. Или, по крайней мере, немного про него слышал. Скорее всего, это сейчас как раз тот навык, который тебе нужно улучшить в связи с должностью, которую ты сейчас занимаешь или которую собираешься получить. Возможно, также, что ты решил поменять что-то в своей жизни и решил попробовать себя в айти, и SQL нужен тебе как раз для этого.

В любом случае, в рамках этой книги я постараюсь максимально передать правильное понимание языка SQL и свой опыт. А начнём с тобой вообще с нуля. С самого начала. Полностью изучив книгу и проделав практические задачи, которые я подготовил для тебя, ты должен получить примерно три года мощной практики работы с базами данных при решении не только типичных, но самых изощрённых задач! Как если бы ты сам через все это прошёл, набил шишки и всему бы, в конечном итоге, научился. Понимаю, что звучит немного неправдоподобно, как книга может загрузить в тебя три года опыта? И я бы на твоём месте, вероятно, имел тень сомнения и это нормально! Но все, что я собрал здесь и в каком виде я собираюсь тебе это преподнести – это мощная программа знаний, навыки передачи которой я совершенствую уже примерно пятнадцать лет! Я научился объяснять SQL! Мои ученики, некоторые из которых только начали свой путь в айти и изначально не имели собственного опыта работы с базами данных, пройдя мой курс обучения, успешно проходили собеседования в айти компаниях по знаниям языка SQL и показывали лучшую компетентность чем даже те, кто в действительности непосредственно до собеседования уже имел опыт работы с базами данных и писал запросы. Все это так, но в действительности, дело было, конечно, в самих учениках! И я в конце говорю им об этом! И тебе говорю! Когда ты действительно готов освоить новый навык или его улучшить, то ты сделаешь это. И не важно через какой источник.

Когда человек готов – учитель найдётся!

(Поговорка)

Совсем коротко про язык SQL: что это такое, и для чего он нужен. Сейчас везде базы данных: сайты, мобильные приложения, различные CRM и ERP системы или другие системы учета, или автоматизации чего-либо, а также популярные программы вроде 1С тоже работают с базами данных. Базы данных во всех сферах нашей жизни. Вообще во всех: вся информация обрабатывается и хранится в базах данных. А SQL – это единственный (!) язык работы с базами данных! Теперь понял, что ты сейчас на пороге изучения единственного ключа, без которого невозможно войти в мир айти? Если ты уже в нем, то, вероятно, дошёл до двери, где без SQL дальше никак.

Вот я и объяснил очень кратко: SQL – это команды, с помощью которых из базы данных извлекается любая информация или кладётся туда. С помощью этих команд базы данных и создаются, и создаются объекты внутри баз данных, например, таблицы. Это было совсем общее объяснение!

Как будет проходить наш курс: вначале я более подробно расскажу, что такое SQL. Объясню, что такое база данных и что такое система управления базами данных (СУБД). Затем мы установим СУБД ORACLE на твой компьютер (или ноутбук), создадим в ней первую базу

данных и загрузим в нее таблицы с тестовыми данными. Они будут необходимы для отработки практических навыков. Всего в книге 15 блоков с практическими задачами. Для некоторых задач я указал каким именно способом я предлагаю их решить, для некоторых – нет. Это значит, что жду от тебя, что ты сам выберешь оптимальный способ решения этой задачи. Звездочками отмечены задачи повышенной сложности. Их решение не обязательно. Но если ты смог решить такую задачу каким-нибудь способом, то это, разумеется, **превосходно**! Постарайся, пожалуйста, сначала решать практические задачи максимально самостоятельно. Так будет больше пользы! Если не будет получаться справиться с какой-либо задачей, то на следующей странице ты найдешь решение. Часто я буду не только показывать тебе ответ для самоконтроля или для понимания, как нужно было решить задачу, но буду и подробно объяснять путь решения. В любом случае, сначала максимально решай каждую задачу сам! Дай своему мозгу проложить дорогу к решению. Лучше еще раз перечитай урок или отложи решение на завтра, но по возможности, старайся решать сам. И так, урок за уроком, мы пройдем курс!

Об авторе

В каждой книге принято немного писать об авторе, чтобы было представление о том, кто её написал, об образовании и опыте автора. Поэтому ниже немного напишу о себе.

Меня зовут Илья Хохлов. Я эксперт в области информационных технологий и баз данных, предприниматель. Являюсь ведущим разработчиком программного обеспечения в собственной компании «Прайм Софт». Мы автоматизируем бизнес российских и зарубежных компаний.

Я являюсь автором курса «SQL. Базы данных. ORACLE», которому обучаю уже 15 лет.

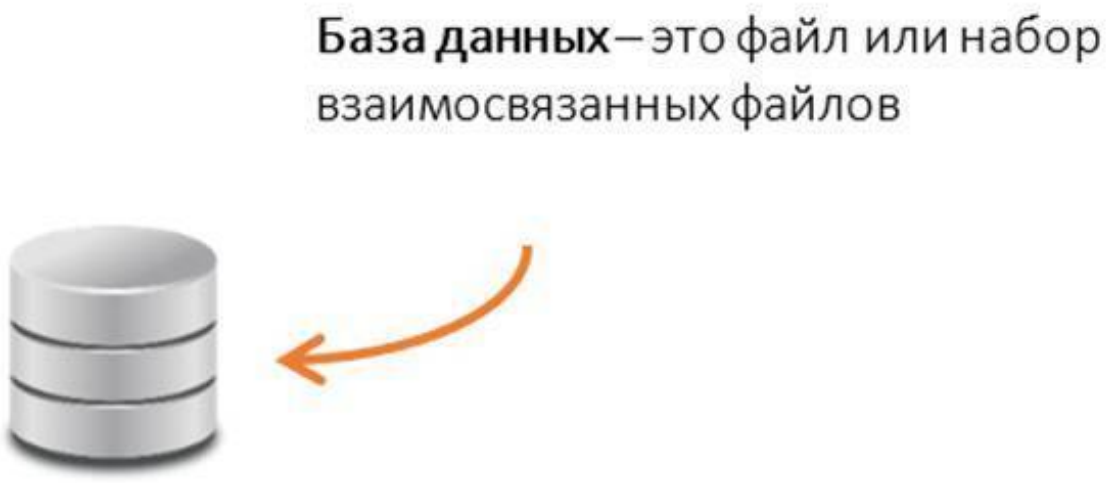
Имею два высших образования, большой опыт работы в айти компаниях, как российских, так и зарубежных. Начал карьеру айти еще в 2005 году. Я работал программистом на последнем курсе в ВУЗе, в котором и учился, а также параллельно пробовал свои навыки преподавания айти технологий в небольшом проф. колледже, в качестве подработки. С 2006 по 2008 года работал штатным программистом в энергосбытовой сфере в Тверской области. С 2008 года по 2017 года – в двух айти-компаниях: Integrator IT и DiaSoft, и потом пять лет ведущим разработчиком в одном из лидирующих московских банков. С 2018 по 2022 года участвовал в проектах с компаниями Status Pro GmbH, Orga-Soft GmbH и MPS-Solutions в Германии.

Буду рад, если найдешь меня, также и в социальных сетях или подпишешься на мой канал в Телеграме, Youtube или Яндекс.Дзене.

Telegram, где мы решаем SQL задачи https://t.me/sql_oracle_databases
Youtube https://youtube.com/c/PrimeSoft
Яндекс.Дзен https://zen.yandex.ru/iliahohlov

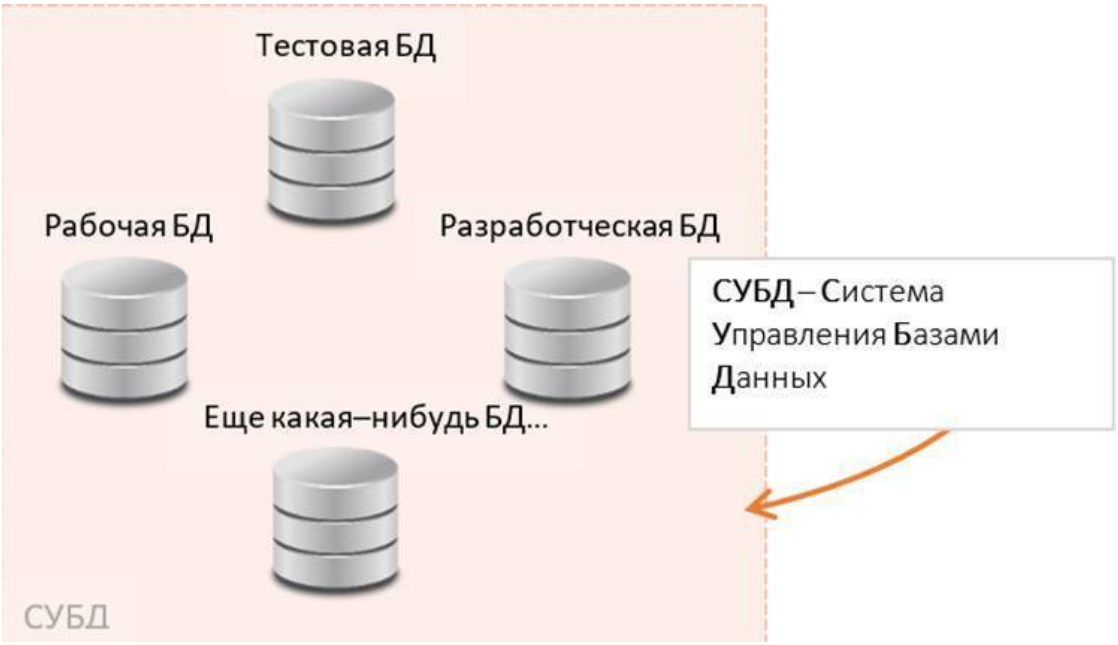
1. Реляционные базы данных

Вначале, давай разберем что такое вообще **б**аза **д**анных (сокращенно – БД). База данных – это, попросту говоря, файл, находящийся на компьютере, на сервере (на главном компьютере) или набор взаимосвязанных файлов. Пока, для простоты, будем понимать, что база данных – это некоторый файл.



Что внутри файла базы данных? В основном – таблицы с данными! Но не только. Объектами баз данных могут быть также и представления, пользователи, роли, хранимые программные объекты (триггеры, процедуры, функции), сиквенсы (счетчики) и еще много всего. Обо всем по порядку!

Итак, база данных – это файл, содержащий в себе таблицы с данными и другие объекты, необходимые для работы **и**нформационной **с**истемы (ИС). Баз данных (файлов) на одном компьютере (сервере) может быть несколько. Всеми файлами (базами данных) управляет система.



Именно с помощью системы производится чтение данных из баз данных (выбираются данные из одной или нескольких таблиц), производится изменение, добавление или удаление данных. Эта система называется **С**истемой **У**правления **Б**азами **Д**анных (СУБД).

Никакой пользователь не может напрямую работать с файлом базы данных. Только с помощью системы он заходит в нужную базу данных, получает доступ к расположенным в ней таблицам, видит доступные ему объекты базы данных и выбирает сведения из какой–либо таблицы.

Ниже список популярных производителей **С**истем **У**правления **Б**азами **Д**анных:

В базах данных вся информация хранится в таблицах. Давай рассмотрим пример некоторой простой таблицы – таблицы Сотрудники:

Таблица **Сотрудники**

ID	NAME	BIRTHDATE
1	Иванов Иван Иванович	12.07.1981
2	Петрова Надежда Анатольевна	04.04.1977
3	Никитин Афанасий Константинович	28.09.1979
4	Первый Николай Николаевич	03.01.1982
5	Орешкина Дарья Васильевна	30.05.1983

С первого взгляда все понятно, верно? В этой таблице три столбца – ID, NAME и BIRTHDATE.

В столбце ID хранится уникальный идентификатор каждого сотрудника. Он же может являться табельным номером. У каждого сотрудника этот номер свой.

В столбце NAME хранится Фамилия Имя Отчество (ФИО) сотрудников. В компании, которую мы уже начали рассматривать в рамках наших уроков, полные однофамильцы допустимы. И чтобы различать сотрудников в кадровом, бухгалтерском и других учетах, каждому сотруднику и присваивают уникальный идентификатор (табельный номер) – столбец ID.

В столбце BIRTHDATE – дата рождения каждого сотрудника.

Столбцы таблицы в терминологии баз данных называются **полями**. В приведенном выше примере у таблицы Сотрудники три поля.

Строки таблицы называются ее **записями**. Когда мы будем писать запросы, те строки данных, которые будет возвращать запрос SELECT, тоже будут называться записями. Ты, наверняка, слышал, как общаются между собой разработчики или аналитики, пишущие запросы: «сколько записей вернул твой запрос»? Это и имелось ввиду: сколько строк данных удалось получить запросом (командой выборки данных из базы данных). У таблиц и запросов строки данных имеют еще одно название – кортежи. Этот термин был введен одним из разработчиков языка SQL, но на практике термин **кортеж** не прижился.

Столбец или набор столбцов, с помощью которых можно явно определить только одну строчку (запись) в таблице, называется **первичным ключом** (**P**rimary **K**ey, или сокращенно PK). В нашем примере, это специально созданный (для идентификации) столбец ID.

При проектировании системы эквайринга (приеме денежных оплат) через терминалы (которые мы можем встречать в супермаркетах или на улице) или через банк, в таблице приема платежей у каждой операции должен быть свой уникальный идентификатор (он печатается на чеке). Чтобы, в случае не зачисления средств, можно было обратиться в службу поддержки и назвать номер платежа. Хотя, современные информационные системы умеют и без того быстро находить проводившиеся оплаты с помощью номера телефона, в счет которого совершался платеж, или с помощью номера лицевого счета, который итак всегда известен плательщику.

В каждой таблице, где требуется иметь возможность сослаться на конкретную строчку таблицы (например, в таблице платежей – на конкретный платеж, в таблице клиентов – на конкретного клиента), должен быть первичный ключ.

Надеюсь, теперь стало ясно что такое первичный ключ и для чего он нужен. В большинстве случаев – это отдельный (один) столбец. Но также первичным ключом может быть два столбца (два поля)! И более. Тогда сочетание значений в этих столбцах должно быть всегда уникально в пределах целой таблицы. И сочетание значений одной строки не может повторяться для другой строки. Чтобы лучше понять для чего же такое может понадобиться, рассмотрим следующий пример.

Некоторая фирма, имеющая несколько филиалов, в конце каждого месяца собирает аналитику по деятельности компании: среднесписочная численность сотрудников в каждом филиале, общая сумма денежных поступлений и сумма денежных расходов по филиалу.

Таблица **Аналитика по филиалам**

ID_FILIAL	PERIOD	AVG_PERS_COUNT	SUM_IN	SUM_OUT
1	Январь 2018	31	30 000 000	11 000 000
2	Январь 2018	25	28 000 890	10 500 009
3	Январь 2018	33	31 000 500	11 000 008

Комбинация значений двух столбцов ID_FILIAL и PERIOD уникальна, и может быть только у одной строки, поэтому разработчики таблицы «Аналитика по филиалам» установили первичным ключом именно сочетание этих двух полей.

В графе ID_FILIAL указывается идентификатор филиала, по которому файл. сохраняются итоги, а в графе PERIOD указывается месяц и год, за который сохраняются итоги. Далее в столбцах AVG_PERS_COUNT, SUM_IN и SUM_OUT указывается среднее количество сотрудников, сумма поступлений и расходов, соответственно.

В нашем примере, за январь 2018 года, была внесена информация по трем филиалам. Проходит месяц и теперь вносится информация за февраль 2018.

Таблица **Аналитика по филиалам**

ID_FILIAL	PERIOD	AVG_PERS_COUNT	SUM_IN	SUM_OUT
1	Январь 2018	31	30 000 000	11 000 000
2	Январь 2018	25	28 000 890	10 500 009
3	Январь 2018	33	31 000 500	11 000 008
1	Февраль 2018	32	34 000 000	12 000 000
2	Февраль 2018	25	28 570 900	10 900 000
3	Февраль 2018	34	31 100 020	11 000 108

Если смотреть значение только в графе ID_FILIAL, то мы обнаружим, что оно повторяется. Конечно, ведь данные по одному филиалу вносились каждый месяц. Значение идентификатора филиала повторяется, но в сочетании со значением столбца PERIOD – оно уникально! Со значением ID_FILIAL = 1 и PERIOD = Январь 2018 есть только одна строчка. Также как и со значением ID_FILIAL = 3 и PERIOD = Февраль 2018.

Так как при проектировании таблицы, так и задумали, что по каждому филиалу за один месяц информация будет только в одной строке, программисты установили сочетание столбцов ID_FILIAL и PERIOD первичным ключом. Это очень просто можно настроить на любой таблице и в одной из следующих глав мы рассмотрим, как это делается. Первичный ключ, если его устанавливать на таблице, помимо логики, дает еще массу преимуществ: теперь ORACLE (или другая СУБД, если ты работаешь не в ORACLE) будет контролировать, чтобы никто не смог добавить строку в таблицу с повторяющейся комбинацией ID_FILIAL и PERIOD. Кроме того, первичный ключ – это еще и самый быстрый способ получения из таблицы информации: если кто–то захочет получить сведения из таблицы по ID_FILIAL = 1 и за Апрель 2019 (например), то вне зависимости столько бы много строчек в таблице не было, СУБД сразу даст результат. Сразу – это значит за мгновение, даже если в таблице будут миллионы строк. Об оптимизации запросов, ключах и индексах у нас будет отдельная тема и мы подробно рассмотрим, как писать запросы так, чтобы они всегда быстро работали!

Теперь рассмотрим еще один термин, который нам нужно понять – **внешний ключ** (Foreign Key, или сокращенно FK).

В компании, базу данных которой мы рассматриваем, помимо таблицы «Сотрудники», есть еще таблица «Автомобили сотрудников».

В таблице «Автомобили сотрудников» есть 4 столбца: столбец ID – сквозной идентификатор каждого учетного автомобиля, столбец ID_PERS – идентификатор сотрудника, владельца автомобиля, столбец NOMER – государственный регистрационный номер автомобиля и столбец MARKA – марка автомобиля.

В таблице «Автомобили сотрудников» столбец ID будет являться собственным первичным ключом. А в столбце ID_PERS указаны только такие значения, которые есть в таблице «Сотрудники» в графе ID. Столбец (или набор столбцов), значения которого ссылаются на первичный ключ другой таблицы, называется внешним ключом (или **f**oreign **k**ey, сокращенно FK). Чтобы было еще понятнее, Foreign key лучше перевести не как «внешний ключ», а как «чужой ключ», то есть ключ не своей таблицы, а другой. Стало понятнее?

Из рисунка выше видно, что Audi A4 принадлежит сотруднику с идентификатором 1, то есть Иванову Ивану. Два автомобиля BMW X3 и Ford Mondeo у сотрудника Петрова Надежда Анатольевна. Fiat Panda принадлежит Афанасию Константиновичу и т.д. Сотрудник Первый

Николай Николаевич не имеет ни одного автомобиля, так как в таблице «Автомобили сотрудников» нет ни одной записи (ни одной строки), где бы в столбце ID_PERS было значение 4.

Разработчики создали в таблице «Автомобили сотрудников» столбец ID_PERS и сознательно настроили так, чтобы он был внешним ключом, то есть значения в нем могли бы быть только такими, которые есть в таблице «Сотрудники» в столбце ID. Теперь СУБД будет сама контролировать, запись с каким значением для графы ID_PERS добавляется в таблицу «Автомобили сотрудников». Чтобы нельзя было добавить строчку в таблицу, указав идентификатор владельца такого, которого нет. Это и есть главное назначение внешнего ключа.

В таблице «Автомобили сотрудников» значение графы ID_PERS ссылается на идентификатор конкретного сотрудника. По этому идентификатору всегда можно получить все сведения о сотруднике: Фамилию Имя Отчество, в каком отделе он работает, на какой должности и т.д. В базе данных еще есть много таблиц, и почти каждая из них имеет столбец, который ссылается на другую таблицу. Например, в таблице Сотрудников, помимо прочих, может быть еще и столбец, указывающий на идентификатор филиала, в котором работает сотрудник. По идентификатору филиала в таблице Филиалов можно найти конкретную строчку с названием филиала и другой сопутствующей информацией: адресом, телефоном и т.д.

В примере, который мы разобрали выше с Сотрудниками и Автомобилями сотрудников, у каждого сотрудника может быть несколько автомобилей. Может быть! А может и не одного. Но если база данных построена так, что согласно связи, одной строчке в одной таблице потенциально может относиться несколько строчек другой таблицы, то такая связь называется **один–ко–многим**. Еще бывают связи **один–к–одному** и **многие–ко–многим**. Немного подробнее поговорим об этом попозже.

Итак, раз в базе данных почти все таблицы как–то относятся к другим таблицам – «Автомобили сотрудников» к «Сотрудникам», «Филиалы» к «Сотрудникам» и т.д. – такую базу данных называют **реляционной** (*от англ. relations – отношения*).

Сейчас практически все базы данных имеют реляционную модель. То есть модель данных, построенную на отношениях.

2. Группы команд языка SQL

Вопрос на собеседовании „Какие команды DML Вы знаете?“ не поставит нас в тупик, а удивит: насколько простое в этой компании собеседование!

Все команды языка SQL разделяются на 4 группы:

DML (Data Manipulation Language) – язык манипуляции данными. Набор из четырех основных команд, для работы непосредственно с информацией, хранящейся в таблицах. С помощью этих команд можно: выбирать из таблицы (чтение), вставлять новые строчки с информацией в таблицу (например, добавлять новые товары в таблицу товаров, добавлять нового сотрудника в таблицу сотрудников), редактировать что–то в строчках данных и удалять строки из таблицы. Помимо этих четырех команд работы с данными, есть еще одна команда – MERGE. Это операция также добавляет строчки в таблицу, но, если записи с такими же ключевыми значениями уже в целевой таблице есть, то MERGE их обновит;

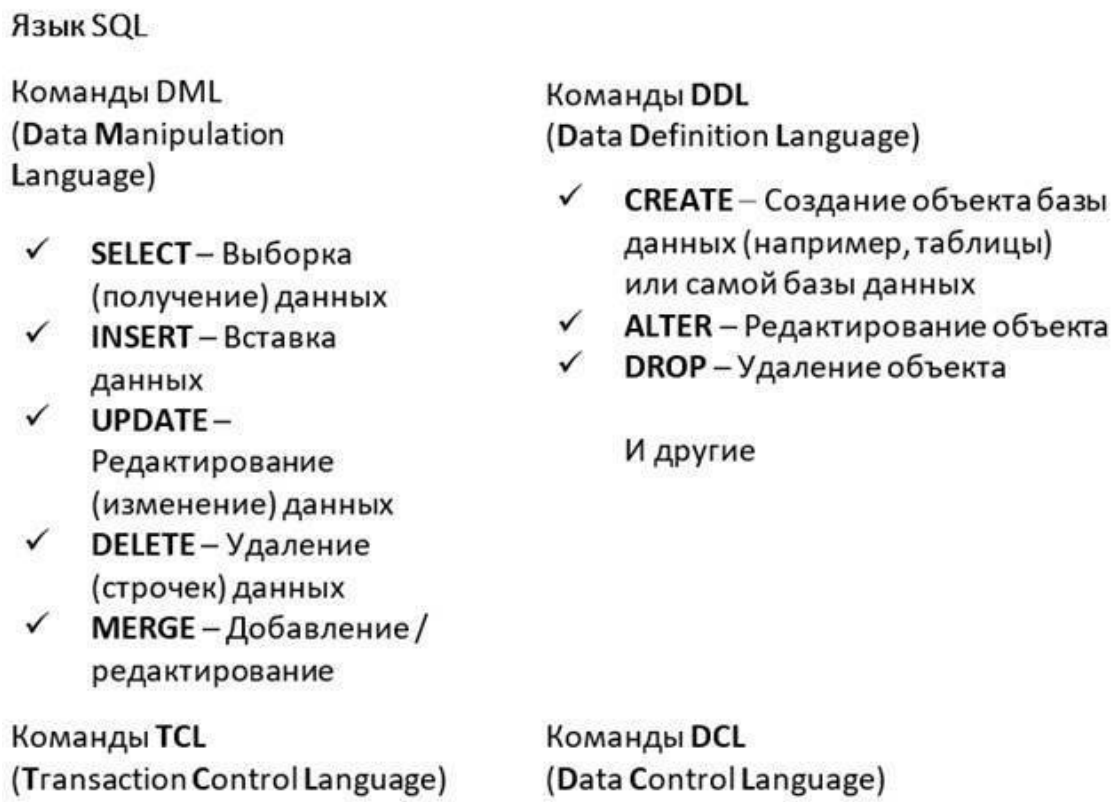
DDL (Data Definition Language) – язык определения данных. Перед тем как строчки с данными добавлять в таблицу, надо сначала создать саму таблицу в базе данных. Вот для этого и нужны команды DDL: создание таблиц и других объектов базы данных, их редактирование и удаление;

TCL (Transaction Control Language) – язык управления транзакциями;

DCL (Data Control Language) – язык контроля доступа к данным.

К группе команд DML относятся команды: SELECT – выбрать/прочитать информацию из таблицы/таблиц, INSERT – вставить новые строчки с данными, UPDATE – изменить, хранящиеся в таблице данные, команда DELETE – удалить строчки с данными, и команда MERGE – вставить/обновить данные в таблице.

К группе команд DDL относятся команды: CREATE – создание новых объектов в базе данных; ALTER – изменение уже существующих объектов, например, расширение таблицы, то есть добавление в нее столбца, для хранения новых сведений; DROP – удаление объекта из базы данных, например, таблицы целиком. Существует еще несколько команд, которые мы рассмотрим позже.



В группе команд TCL управления транзакциями всего две команды: COMMIT и ROLLBACK. Первая подтверждает проведенные изменения, а вторая откатывает. Понятие транзакций и более подробную работу с ними мы рассмотрим в отдельной главе.

К командам контроля доступа к данным DCL относятся команды: GRANT – предоставление привилегий на определенные действия к определенным объектам для определенных пользователей, ролей или для всех; REVOKE – снятие привилегий на определенные действия к определенным объектам с определенный пользователей, ролей или всех. Например, с помощью этих команд, можно дать некоторому пользователю базы данных права на вставку данных в таблицу, которую мы создали. Или, с помощью них, мы можем предоставить права, например, на чтение информации с нашей таблицы, к примеру, сразу всем. То есть к этой группе команд относятся команды, с помощью которых можно давать права на объект базы данных или наоборот, запрещать кому–то делать что–то с таблицей или другим объектом базы данных.

Контрольные вопросы №1

В этой главе мы достаточно изучили теории, хорошо разобрались с группами команд языка SQL и теперь необходимо закрепить полученные знания, чтобы четко понимать к какой группе команд относится та или иная команда. Для этого предлагаю ответить на следующие практические вопросы:

1. Какие команды DML ты запомнил?

2. Нам необходимо добавить новое лекарство в нашу базу данных в таблицу–справочник лекарственных средств. Какую команду SQL мы должны использовать? (Ответ должен быть, например, команда UPDATE или DELETE, или другая).

3. В таблице заказов необходимо отредактировать количество проданного товара для одной из покупок, сделанных минуту назад. Клиент решил купить больше товара! Необходимо изменить значение в некоторой строке. Какая команда SQL будет выполнена?

4. Для выполнения изменения законодательства нашей компании обязательно нужно будет в базе данных хранить дополнительные сведения о товарах. Для этого потребуется добавить два новых столбца в таблицу товаров. Какая команда SQL будет выполнена?

5. Необходимо удалить ошибочно заведенного клиента, то есть удалить строку из таблицы клиентов. Какая это команда SQL?

6. Необходимо для одного из клиентов в столбце «Дата закрытия» удалить дату, так как руководство приняло решение возобновить работу с клиентом. Какая команда SQL?

7. Для расширения бизнеса потребуется одна новая таблица для хранения операций по скидочным картам. Какая команда SQL должна быть выполнена чтобы создать новую таблицу в базе данных?

8. Нужно получить остаток по счету некоторого клиента. Какая команда SQL будет выполнена?

9. Необходимо вывести список последних десяти операций по счету клиента. Какая команда SQL?

10. В базе данных создали новую таблицу. Необходимо предоставить возможность выбирать данные из этой таблицы всем пользователям. Какую команду SQL необходимо выполнить, чтобы предоставить права на выполнение команды SELECT из этой новой таблицы?

Ответы на контрольные вопросы на следующей странице.

Ответы на контрольные вопросы №1

1. Какие команды DML ты запомнил?
Ответ: команды DML: SELECT, INSERT, UPDATE, DELETE и MERGE.

2. Нам необходимо добавить новое лекарство в нашу базу данных в таблицу–справочник лекарственных средств. Какую команду SQL мы должны использовать? (Ответ должен быть, например, команда UPDATE или DELETE, или другая).
Ответ: команда INSERT.

3. В таблице заказов необходимо отредактировать количество проданного товара для одной из покупок, сделанных минуту назад. Клиент решил купить больше товара! Необходимо изменить значение в некоторой строке. Какая команда SQL будет выполнена?
Ответ: команда UPDATE.

4. Для выполнения изменения законодательства нашей компании обязательно нужно будет в базе данных хранить дополнительные сведения о товарах. Для этого потребуется добавить два новых столбца в таблицу товаров. Какая команда SQL будет выполнена?
Ответ: команда ALTER.

5. Необходимо удалить ошибочно заведенного клиента, то есть необходимо будет удалить строку из таблицы клиентов. Какая это команда SQL?
Ответ: команда DELETE.

6. Необходимо для одного из клиентов в столбце «Дата закрытия» удалить дату, так как руководство приняло решение возобновить работу с клиентом. Какая команда SQL?
Ответ: UPDATE, так как это не команда удаления строки целиком, а команда изменения ее значения в определенном столбце с даты, на пустое значение.

7. Для расширения бизнеса потребуется одна новая таблица для хранения операций по скидочным картам. Какая команда SQL должна быть выполнена чтобы создать новую таблицу в базе данных?
Ответ: команда CREATE.

8. Нужно получить остаток по счету некоторого клиента. Какая команда SQL будет выполнена?
Ответ: команда SELECT.

9. Необходимо вывести список последних десяти операций по счету клиента. Какая команда SQL?
Ответ: команда SELECT.

10. В базе данных создали новую таблицу. Необходимо предоставить возможность выбирать данные из этой таблицы всем пользователям базы данных. Какую команду SQL необходимо выполнить, чтобы предоставить права на возможность выполнения команды SELECT из этой новой таблицы?
Ответ: команда GRANT.

3. Структура команды SELECT

SELECT (*с англ. «выбрать»*) – это команда получения информации из базы данных и преобразование ее к любому удобному виду. С помощью этой команды можно выбирать данные из одной таблицы или сразу из нескольких (позже мы узнаем, что получать данные можно не только из таблиц). Получаемый результат можно сортировать, группировать, анализировать.

SELECT – это самая часто используемая команда языка SQL. С помощью нее можно получать как табличные данные (например, список клиентов с подробными сведениями о них, топ самых продаваемых товаров за прошлый год, или список доступных банковских продуктов для клиента), так и какую–либо обобщающую информацию – одним значением (например, доступный баланс на банковской карте или количество друзей друга/подруги в социальной сети, или даже оставшееся количество мест в любимом отеле на интересующую дату). Любые данные в любом виде из базы данных получает команда SELECT.

Синтаксис команды SELECT максимально прост. Чтобы выбрать какую–либо информацию из таблицы нужно написать:

```
SELECT <Столбец1>, <Столбец2>, ... <СтолбецN>
FROM <Имя_таблицы>
```

Итак, чтобы выбрать информацию из некоторой таблицы, нужно написать слово **SELECT**, потом какие именно столбцы интересуют (через запятую), потом слово **FROM** и далее имя таблицы.

Любой запрос можно писать хоть весь в одну строку, хоть разбивать его на несколько строк. Если запрос получается большой и сложный, то, чтобы он был более легко читаем, его принято разбивать на несколько строк. Постепенно мы будем привыкать к хорошему стилю написания SQL–кода.

Теперь попробуем написать первую команду выборки данных:

```
SELECT ID, Name FROM Persons
```

Приведенный выше запрос выберет данные из таблицы Persons. Покажет информацию из столбцов ID и Name. Получим результат вида:

ID	NAME
1	Иванов Иван Иванович
2	Петрова Надежда Анатольевна
3	Никитин Афанасий Константинович
4	Первый Николай Николаевич
5	Орешкина Дарья Васильевна

Давай выведем еще и даты рождения сотрудников:

```
SELECT ID, Name, BirthDate FROM Persons
```

ID	NAME	BIRTHDATE
1	Иванов Иван Иванович	12.07.1981
2	Петрова Надежда Анатольевна	04.04.1977
3	Никитин Афанасий Константинович	28.09.1979
4	Первый Николай Николаевич	03.01.1982
5	Орешкина Дарья Васильевна	30.05.1983

Чтобы полученный результат SQL–запроса упорядочить по одному или нескольким столбцам (сортировка данных), нужно в предложение добавить еще блок ORDER BY (*с англ. «упорядочить по»*):

```
SELECT <Столбец1>, <Столбец2>, ... <СтолбецN>
 FROM <Имя_таблицы>
ORDER BY <Столбец1>, ...
```

Выборка информации из таблицы без условий, то есть всех строк данных (!) не часто бывает нужна и поэтому, почти всегда, на выбираемые строки из таблицы накладывают условие или условия, чтобы отбирать только подходящие условиям данные. Это делается с помощью блока WHERE. Именно в блоке WHERE пишутся одно или несколько (комбинация) условий, для определения отбираемых данных. Его место в запросе SELECT:

```
SELECT <Столбец1>, <Столбец2>, … <СтолбецN>
 FROM <Имя_таблицы>
WHERE <Условие или условия отбора данных из таблицы>
```

Блок ORDER BY всегда пишется в самом конце SQL–запроса!

Если к выводимым данным необходимо добавить данные еще из другой одной или нескольких таблиц (присоединить к выводящимся данным данные из других таблиц), то после того, как мы написали слово FROM и имя основной таблицы, мы можем присоединять дополнительные таблицы с помощью слова JOIN:

```
SELECT <Столбец1>, <Столбец2>, … <СтолбецN>
 FROM <Имя_таблицы>
  JOIN –ы (Соединения дополнительных таблиц)
WHERE <Условие или условия отбора данных из таблицы>
ORDER BY <Столбец1>, …
```

Ничего себе, сколько всего, скажешь ты, как это можно все запомнить и понять?! Каждый блок мы разберем по–отдельности и каждому уделим достаточно внимания!

Но и это еще не все. Есть еще одна возможность команды SELECT – это группировка.

Получаемые данные можно группировать по одному или нескольким признакам одновременно.

Для того чтобы указать по одинаковым значениям в каком столбце необходимо данные группировать, нужно команду SELECT дополнить блоком GROUP BY (*с англ. «группировать по»*) и затем написать имя столбца, по которому необходима группировка. И теперь все строки получаемого набора данных будут группироваться по одинаковому значению в этом столбце.

Также, может понадобиться в конце года, например, отобрать «любимых» клиентов нашей организации для того, чтобы поздравить их с наступающими праздниками и сделать некоторый приятный бонус. Любимыми являются клиенты, у которых сумма заказов за прошедший год более 500.000 рублей.

Для решения подобной задачи, вначале, из таблицы «Продаж» мы отберем все сделки за прошедший год. Для выборки данных, согласно этому условию, воспользуемся блоком WHERE. Итак, когда данные будут извлечены из таблицы, за нужный нам промежуток времени, мы можем увидеть, что ни один из единичных заказов не больше 500.000 р. То есть ни в одной полученной строке в столбце «Сумма сделки» значение не больше 500.000 р. Но, если сгруппировать полученные данные по каждому клиенту (иными словами строки с одинаковым значением Клиента слить в одну, подсчитав Сумму сделок по каждому клиенту), то может получиться, что некоторые клиенты, суммарно за год, хорошо превышают этот порог.

До выполнения группировки мы видели в полученной таблице данных **каждую** строчку продаж, затем мы все данные сгруппировали по клиентам и стали видеть **по каждому клиенту** теперь **только одну** строку с общим итогом по нему.

Например, все выбранные продажи «Клиенту А» сгруппировались в одну строку, подсчитав сумму продаж ему, а все продажи «Клиенту Б» в другую строку, также с итогом по нему. И так по каждому клиенту. Теперь мы видим итоги с суммами продаж за год с группировкой по клиентам. Так как «Клиент А», например, в течение года каждый месяц делал заказы на 100.000 р. Поэтому, после группировки всех сумм его заказов, мы получим 1.200.000 р (100.000 × 12 месяцев).

Подробнее про группировку и ее мощные сопутствующие возможности, мы рассмотрим в отдельной главе. У нас будет сразу несколько уроков, связанных с группировкой, чтобы хорошо разобрать эту тему.

Место слова GROUP BY в предложении SELECT:

```
SELECT <Столбец1>, <Столбец2>, ... <СтолбецN>
  FROM <Имя_таблицы>
    JOIN –ы (Соединения дополнительных таблиц)
WHERE <Условие или условия отбора данных из таблицы>
GROUP BY <Признак группировки>
ORDER BY <Столбец1>, ...
```

После группировки всех продаж за год в общем отчете, тем не менее, еще остается много клиентов, которые обращались в нашу компанию один или два раза, и, что самое главное, общая сумма их заказов не превышает порог «любимых клиентов». И таких клиентов много. Руководство нашей компании не хотело бы вручную из полученных итогов отбирать «Любимых клиентов». Чтобы оставить только нужные данные на основе получаемых сгруппированных итогов, мы воспользуемся опцией «HAVING» блока GROUP BY.

Важно понять, что только после группировки по клиентам мы смогли получить итоговую сумму заказов за год по каждому клиенту (до этого мы имели изначальную таблицу, где в строках были указаны стоимости единичных сделок), и, чтобы на основе уже этой полученной суммы (сгруппированной суммы) отфильтровать результирующий набор клиентов, мы можем применить HAVING.

На собеседованиях часто можно встретить такой вопрос: в чем разница между WHERE и HAVING? И теперь мы знаем ответ: WHERE выполняет первичный отбор данных из таблицы (таблиц) (в нашем примере, мы сначала отобрали данные продаж только за прошедший год), а HAVING отсеивает уже на основе сгруппированной информации.

То есть после того, как будут получены суммы по клиентам, в результирующем наборе останутся только те клиенты, у которых эти суммы более интересующего нас значения.

Конечно, мы можем отбирать строчки из таблицы заказов тех, где «Сумма сделки» больше, например, определенной. Но у нас задача была другая. Нам необходимо было получить клиентов, сумма заказов за год которых превысила 500.000 р. Поэтому мы применили сначала WHERE, для первичного отбора строчек данных из таблицы «Заказов» тех, которые относятся к прошедшему году, и затем воспользовались группировкой GROUP BY по клиентам с подсчетом «Сумм сделок» по каждому их них с опцией HAVING, чтобы на основе сгруппированной (агрегированной, то есть обобщенной) информации сделать еще одну фильтрацию данных.

HAVING следует писать после GROUP BY:

```
SELECT <Столбец1>, <Столбец2>, ... <СтолбецN>
  FROM <Имя_таблицы>
    JOIN –ы (Соединения дополнительных таблиц)
WHERE <Условие или условия отбора данных из таблицы>
GROUP BY <Признак группировки>
HAVING <Условия отбора на основе данных группировки>
ORDER BY <Столбец1>, ...
```

И это уже полная структура одного предложения SELECT. Полный список ключевых слов, которые можно применять при выборке данных. Из всех перечисленных ключевых слов обязательными являются только SELECT и FROM. Запросы могут быть даже без WHERE и без сортировки – ORBER BY. Главное, что всегда нужно указывать, – это какие столбцы отбирать и откуда.

Конечно, мы будем применять еще и кейсы, и подзапросы, но это все будет строиться на основе структуры, которая приведена выше. Поэтому, ее нужно запомнить.

Для Гуру: в СУБД MS SQL Server и MySQL даже FROM не обязателен при выводе данных, но это исключение и применяется при решении специфических задач. Объясню тебе про это на уроке про псевдотаблиц.

4. Написание простых запросов получения данных

4.1. Выборка некоторых или всех столбцов из таблицы

Дорогой читатель, в начале следующей главы мы разберем как установить ORACLE и создать базу данных на своем компьютере. А также, мы познакомимся с программой SQL Developer, одним из самых распространенных средств работы с базами данных ORACLE. Пройдя по ссылке, ты скачаешь скрипт и загрузишь его в свою новую, пока пустую, базу данных. После прогрузки скрипта, у тебя появятся таблицы с тестовыми (учебными) данными. И все это за несколько простых шагов!

Теперь у тебя будет фактически подготовленное рабочее (учебное) место!

В конце каждой главы для тебя подготовлены практические задачи! Их нужно постараться сделать максимально самостоятельно. К некоторым задачам будут даваться рекомендации к выполнению, к некоторым – нет. Это значит, что их можно будет решить любым способом. Некоторые задачи можно будет решить только комбинацией методов. Большинство задач – это стандартные задачи, которые решают специалисты по SQL, а некоторые – нестандартные. С помощью них, ты научишься нестандартно и более глубоко понимать SQL. Если ты в течение часа не смог решить некоторую задачу, ее можно отложить и попробовать вернуться к ней, например, попозже или завтра! Многие мои ученики иногда так справлялись с достаточно трудными запросами, и, на второй день, почти всегда говорили, что смогли взглянуть на задачу под другим углом.

После списка задач к каждой главе ты найдешь решения к задачам. Мы подробно вместе прорешаем каждую задачу. Но не нужно этим пользоваться сразу, если у тебя не получается решить задачу. Желательно, ответами пользоваться минимум завтра. Ты получишь больше пользы, если сможете решить задачу сам, пусть и дольше.

В предыдущей главе мы рассмотрели общую структуру любого предложения SELECT. Работая постоянно с запросами, через довольно короткое время, мы запомним назначение и расположение каждого блока и еще чуть позже, будем правильно и максимально эффективно их использовать! Теперь, в качестве примеров, составим несколько простых SQL–запросов.

Напишем запрос, выбирающий сотрудников из таблицы Persons, который отображал бы их Фамилию Имя Отчество, Дату рождения и идентификатор филиала, в котором они работают. Фамилия Имя Отчество лежит в колонке NAME, Дата рождения – в колонке BIRTHDATE и идентификатор филиала – в графе FilialID:

```
SELECT NAME, BIRTHDATE, FilialID
  FROM Persons
```

Выполняем запрос и получаем результат:

	NAME	BIRTHDATE	FILIALID
1	Иванов Иван Иванович	12.04.81	1
2	Петров Павел Сергеевич	16.09.82	1
3	Караваева Людмила Сергеевна	19.09.82	1
4	Черных Александр Александрович	22.01.72	1
5	Лимонадова Анна Васильевна	12.02.77	2
6	Белый Олег Викторович	30.03.86	2
7	Финина Алла Павловна	01.03.84	2
8	Некрасов Дмитрий Валерьевич	06.03.82	2
9	Гостин Сергей Олегович	12.04.82	3

Как видим, вывелись именно запрошенные столбцы из таблицы. И еще, первый столбец, – сквозная нумерация строк возвращаемых данных. Мы его не запрашивали нашей SQL–командой. И на самом деле, это не ORACLE нам его вернул вместе с возвращаемыми запросом данными, а сама программа через которою мы работаем в базе данных (в нашем случае программа SQL Developer) добавила нам его для удобства. Далее не будем обращать на него внимание.

Запрос вернул 21 запись (21 строку). В целях экономии места в книге, мы иногда будем отображать не все возвращаемые данные.

4.2. Использование условий при получении данных. Ключевое слово WHERE

Теперь доработаем запрос, пусть он выведет только сотрудников, где в графе FilialID равно 2:

```
SELECT NAME, BIRTHDATE, FilialID
  FROM Persons
 WHERE FilialID = 2
```

Как видим из получаемых данных, во втором филиале у нас работает 4 сотрудника!

	NAME	BIRTHDATE	FILIALID
1	Лимонадова Анна Васильевна	12.02.77	2
2	Белый Олег Викторович	30.03.86	2
3	Финина Алла Павловна	01.03.84	2
4	Некрасов Дмитрий Валерьевич	06.03.82	2

СУБД ORACLE, выполняя команду SELECT, выбирает для нас такие строчки из всей таблицы Persons, где в колонке FilialID значение равно двум!

Какие символы, помимо знака равно, можно использовать в условиях:

= Равно

<> Не равно

!= Не равно

> Больше

< Меньше

>= Больше или равно

<= Меньше или равно

Как мы видим, если в SQL запросе нужно выбрать данные с условием на неравенство, то мы можем написать как **<>**, так и **!=**.

Следующей командой выберем сотрудников, работающих не во втором филиале:

```
SELECT NAME, BIRTHDATE, FilialID
  FROM Persons
 WHERE FilialID <> 2
```

Выведутся данные (всего 17 строк, для экономии места вот первых 10):

	NAME	BIRTHDATE	FILIALID
1	Иванов Иван Иванович	12.04.81	1
2	Петров Павел Сергеевич	16.09.82	1
3	Караваева Людмила Сергеевна	19.09.82	1
4	Черных Александр Александрович	22.01.72	1
5	Гостин Сергей Олегович	12.04.82	3
6	Розовая Лидия Георгиевна	14.05.82	3
7	Бодров Иван Антонович	17.06.88	3
8	Симонова Ольга Петровна	07.07.79	3
9	Красавина Лидия Борисовна	05.08.78	3
10	Первый Иван Иванович	08.09.77	3

4.3. Сортировка данных. Блок ORDER BY

Теперь полученные данные мы можем еще и упорядочить по Фамилии Имени Отчеству. Для этого допишем блок ORDER BY.

```
SELECT NAME, BIRTHDATE, FilialID
  FROM Persons
 WHERE FilialID <> 2
 ORDER BY NAME
```

В блоке ORDER BY (*с англ. «упорядочить по»*) указали графу NAME, так как согласно значению в этом столбце нам необходимо было упорядочить строки. В результате получаем следующую таблицу с данными:

	NAME	BIRTHDATE	FILIALID
1	Бабкина Надежда Григорьевна	03.12.82	1
2	Бодров Иван Антонович	17.06.88	3
3	Гостин Сергей Олегович	12.04.82	3
4	Иванов Иван Иванович	12.04.81	1
5	Иванова Тамара Николаевна	15.05.77	1
6	Караваева Людмила Сергеевна	19.09.82	1
7	Красавина Лидия Борисовна	05.08.78	3
8	Линина Мария Семеновна	19.10.81	1
9	Ломоносов Игорь Павлович	09.09.80	1
10	Первый Иван Иванович	08.09.77	3
11	Петров Павел Сергеевич	16.09.82	1
12	Розовая Лидия Георгиевна	14.05.82	3
13	Сидоров Иван Денисович	29.10.79	3
14	Симонова Ольга Петровна	07.07.79	3
15	Сливкина Наталья Эдуардовна	08.09.77	1
16	Черных Александр Александрович	22.01.72	1
17	Шилова Анна Ивановна	19.11.80	3

Как видим, строки упорядочены (отсортированы) по Фамилии Имени Отчеству. Точно также, если нам нужно было бы расставить сотрудников не в алфавитном порядке согласно их ФИО, а, например, согласно их дате рождения, то в боке ORDER BY указали бы BIRTHDATE.

Всякий раз, указывая значение столбца, по которому ведется сортировка строк, мы можем сортировать как в прямом порядке, так и в обратном. Для сортировки строк в обратном порядке нужно сразу после имени столбца написать слово DESC. И все!

Выведем сотрудников третьего филиала, упорядоченных по Фамилии Имени Отчеству в обратном порядке:

```
SELECT NAME, BIRTHDATE, FilialID
  FROM Persons
 WHERE FilialID = 3
 ORDER BY NAME DESC
```

Получаем:

	NAME	BIRTHDATE	FILIALID
1	Шилова Анна Ивановна	19.11.80	3
2	Симонова Ольга Петровна	07.07.79	3
3	Сидоров Иван Денисович	29.10.79	3
4	Розовая Лидия Георгиевна	14.05.82	3
5	Первый Иван Иванович	08.09.77	3
6	Красавина Лидия Борисовна	05.08.78	3
7	Гостин Сергей Олегович	12.04.82	3
8	Бодров Иван Антонович	17.06.88	3

А теперь выведем тех же самых сотрудников, но в прямом порядке. Не в обратном! Для этого нужно просто убрать DESC. Или вместо DESC написать ASC.

```
SELECT NAME, BIRTHDATE, FilialID
  FROM Persons
 WHERE FilialID = 3
 ORDER BY NAME ASC
```

ASC и DESC – это такие «флажки» для ORACLE, указывающий на то, в каком направлении требуется упорядочить данные. Если ASC после столбца не указывать, то СУБД итак поймет, что данные нужно упорядочить в прямом порядке, ведь DESC–то нету! Так как не писать ASC проще, чем писать его, его использование уходит в прошлое. Его уже почти никто не использует. Но, если Вы придете работать в компанию, которая существует уже много лет, и Вам нужно будет в рамках некоторой задачи доработать отчет, вернее его запрос, на основе которого формируются данные, и если Вы там увидите ASC, то теперь Вы будет знать, что это означает! Язык SQL, подобно любому человеческому языку, также стремится к простоте. Отбрасывая то, что можно опустить, не использовать. И при этом сохраняя однозначность выполнения.

В процессе изучения языка SQL, мы встретим еще несколько «отмирающих» слов, неиспользование которых не является ошибкой. Эти слова попросту не несут в себе дополнительного смысла, и присутствие их даёт такой же результат, как и отсутствие.

4.4. Выборка данных по нескольким условиям. Использование AND и OR. Приоритеты операторов

Теперь рассмотрим, как сочетать несколько условий в одном блоке WHERE. Выведем сотрудников, работающих в филиале 2 или 3, упорядоченных по ФИО. По сути, нужно вывести все строчки из таблицы Persons, в которых в столбце FilialID значение равно 2 или 3. Если в строчке в графе FilialID = 2 – показываем такую строку! Если 3 – тоже показываем!

```
SELECT NAME, BIRTHDATE, FilialID
  FROM Persons
 WHERE FilialID = 2 OR FilialID = 3
 ORDER BY NAME
```

Результат:

	NAME	BIRTHDATE	FILIALID
1	Белый Олег Викторович	30.03.86	2
2	Бодров Иван Антонович	17.06.88	3
3	Гостин Сергей Олегович	12.04.82	3
4	Красавина Лидия Борисовна	05.08.78	3
5	Лимонадова Анна Васильевна	12.02.77	2
6	Некрасов Дмитрий Валерьевич	06.03.82	2
7	Первый Иван Иванович	08.09.77	3
8	Розовая Лидия Георгиевна	14.05.82	3
9	Сидоров Иван Денисович	29.10.79	3
10	Симонова Ольга Петровна	07.07.79	3
11	Финина Алла Павловна	01.03.84	2
12	Шилова Анна Ивановна	19.11.80	3

Одна из типичных ошибок, которую совершают начинающие специалисты SQL, они иногда пишут так:

```
...
WHERE FilialID = 2 OR 3
...
```

И потом удивляются, почему запрос не может «отработать»? Так писать неправильно, так как ORACLE (или другая СУБД) не сможет однозначно понять, что имелось ввиду. ORACLE, получая такой запрос, «думает»: нужно вывести строчки в которых в столбце FilialID значение равно двум или… три. И тут не понятно, что – «три»?! Значение в каком столбце должно быть равно трем? На эту тему есть старинный анекдот:

Летят Петька и Василий Иванович в самолёте. Василий Иванович:

– *Петька, приборы?*
– *Девять!*
Летят дальше. Через некоторое время Василий Иванович снова:
– *Петька, приборы?!*
– *Девять!*
– *Что «девять»–то?*
– *А что «приборы»?*
Всегда нужно указывать и второе условие (то, что после OR) полностью! Правильно так:

```
...
WHERE FilialID = 2 OR FilialID = 3
...
```

В SQL запросе, для комбинации условий, мы использовали OR. С помощью него из таблицы будет выведена всякая строчка в случае, если выполняется одно, либо другое условие. То есть, минимум одно из них. ORACLE, пробегая по всей таблицы (Persons) и решая какую строчку выбрать нам в результирующий набор, будет сначала пробовать первое условие. То есть смотреть значение «2» ли в поле FilialID. Если нет, то может в поле FilialID значение «3»? Если да, то строчка будет выбрана, и мы ее увидим.

В следующем примере выведем строчки таблицы, соответствующие одновременно двум условиям. Для наглядности, добавим еще и вывод идентификатора департамента сотрудника:

```
SELECT NAME, BIRTHDATE, FilialID, DepartamentID
  FROM Persons
 WHERE FilialID = 1 AND DepartamentID = 2
```

Из таблицы Persons отберутся такие строчки, в которых в столбце FilialID значение равно 1 и одновременно в этой же строке в столбце DepartamentID значение равно 2. Результат:

	NAME	BIRTHDATE	FILIALID	DEPARTAMENTID
1	Караваева Людмила Сергеевна	19.09.82	1	2
2	Черных Александр Александрович	22.01.72	1	2

В одном запросе можно одновременно использовать AND и OR:

```
SELECT NAME, BIRTHDATE, FilialID, DepartamentID
  FROM Persons
 WHERE FilialID = 1 AND DepartamentID = 2 OR DepartamentID = 3
```

Беглым взглядом не понятно, какие данные хотел отобрать разработчик: то ли сотрудников, работающих в филиале 1 и в департаменте 2 или 3, то ли сотрудников, работающих в первом филиале и втором департаменте и еще в департаменте 3 и в не важно каком филиале. На самом деле, OR имеет больший вес, чем AND, и поэтому разделит условие на два:

отберутся сотрудники с филиала 1 и с одновременным отнесением ко второму департаменту, а также те, у кого просто указан третий департамент. Чтобы данный SQL–код был более наглядным, рекомендуется использовать скобки, чтобы расставить приоритеты выполнения условий. Например, если требуется вывести сотрудников только первого филиала с департаментов 2 и 3, можно написать так:

```
SELECT NAME, BIRTHDATE, FilialID, DepartamentID
  FROM Persons
 WHERE FilialID = 1 AND (DepartamentID = 2 OR DepartamentID = 3)
```

Получим данные:

	NAME	BIRTHDATE	FILIALID	DEPARTAMENTID
1	Петров Павел Сергеевич	16.09.82	1	3
2	Караваева Людмила Сергеевна	19.09.82	1	2
3	Черных Александр Александрович	22.01.72	1	2

Чтобы вывести значения со всех столбцов таблицы необязательно их все перечислять в секции SELECT, для этого достаточно поставить звездочку:

```
SELECT *
  FROM Persons
```

5. Подготовка рабочего места

5.1. Скачивание и установка СУБД ORACLE

На сегодняшний день ORACLE занимает одно из лидирующих мест на рынке производителей систем управления базами данных и, к счастью, предоставляет свой продукт для скачивания в некоммерческих целях абсолютно бесплатно. То есть начинающие специалисты могут бесплатно установить на свой домашний компьютер СУБД и попрактиковаться, например, в составлении запросов.

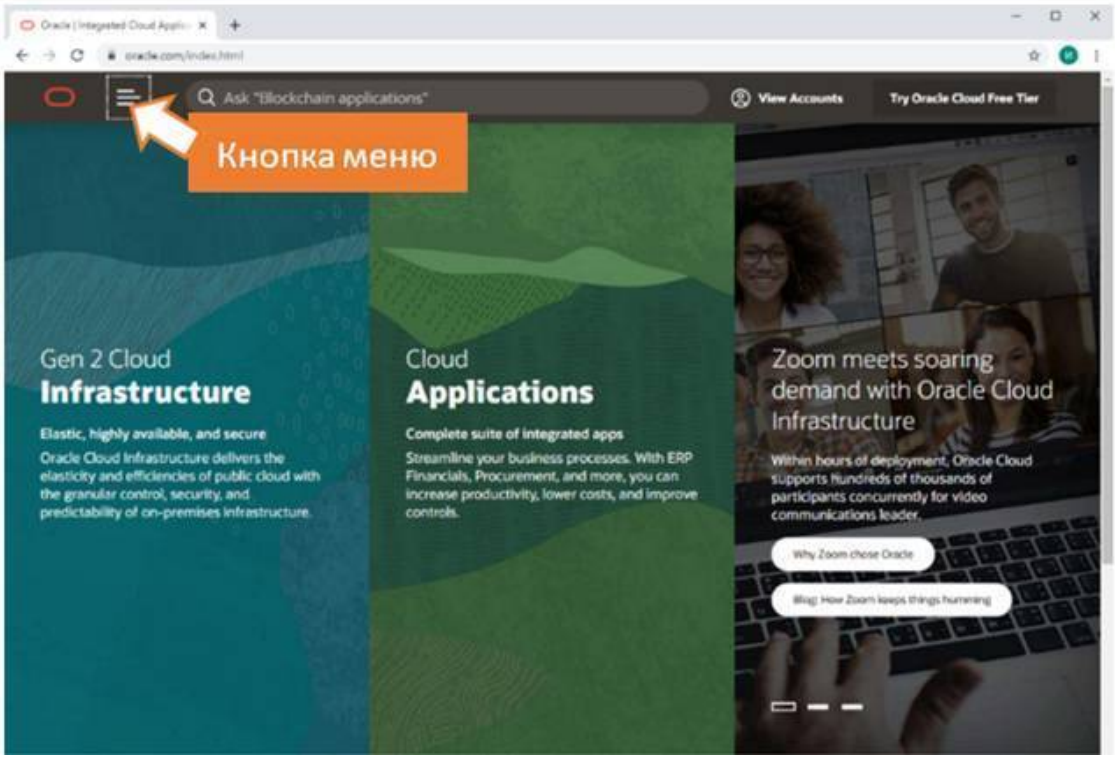
Для того, чтобы в максимальной степени овладеть знаниями языка SQL, крайне необходимо сейчас установить на наш домашний компьютер или ноутбук ORACLE, программу SQL Developer и закачать тестовую базу данных, на которой мы будем выполнять практические задачи каждого урока! Это очень важно! Сейчас мы подготовим рабочее место.

Если у тебя будет что-то не получаться во время скачивания СУБД ORACLE с официального сайта или во время ее установки, то зайди ко мне по ссылке ниже – я подготовил для тебя инструкцию по максимально простому скачиванию и установке: https://prime-soft.biz/how-to-install

Ниже я опишу полный процесс скачивания и установки ORACLE с официального сайта, но, если не будет получаться, то обязательно воспользуйся ссылкой выше. Я помогу тебе!

Музыкант никогда не сможет научиться играть на музыкальном инструменте, если не будет практиковаться на нем, так и мы – не сможем стать специалистами SQL, если только прочитаем книгу! Обязательно нужна практика.

Сперва заходим на сайт https://oracle.com и нажимаем на кнопку меню в виде трех горизонтальных полос (на момент написания книги она находится слева сверху).



В открывшемся меню нажимаем Oracle Database.

На открывшейся странице нажимаем Download Oracle Database 19c (Загрузить Oracle версию 19c), даже если решим устанавливать другую версию. Переходим в список последних версий СУБД Oracle, доступных для закачивания для разных операционных систем. Ты должен знать какая у тебя версия операционной системы и какой битности: 32 или 64. Нам не важно какую версию СУБД ты установишь, 19.3, 18 или даже 12. Или, может быть, на момент чтения этой книги уже выйдет более новая версия Oracle. Главное, чтобы она подходила к твоей операционной системе. Например, у нас операционная система Windows 10 x64, тогда, мы можем загрузить, например, версию Oracle 19.3. Нажимаем на кнопку с надписью «Zip» напротив нашей версии ОС. В появившемся окне перед загрузкой, нужно установить галочку «I reviewed and accept the Oracle License Agreement» – это наше соглашение о неиспользовании ORACLE в коммерческих целях и дополнительная информация. И нажимаем загрузить.

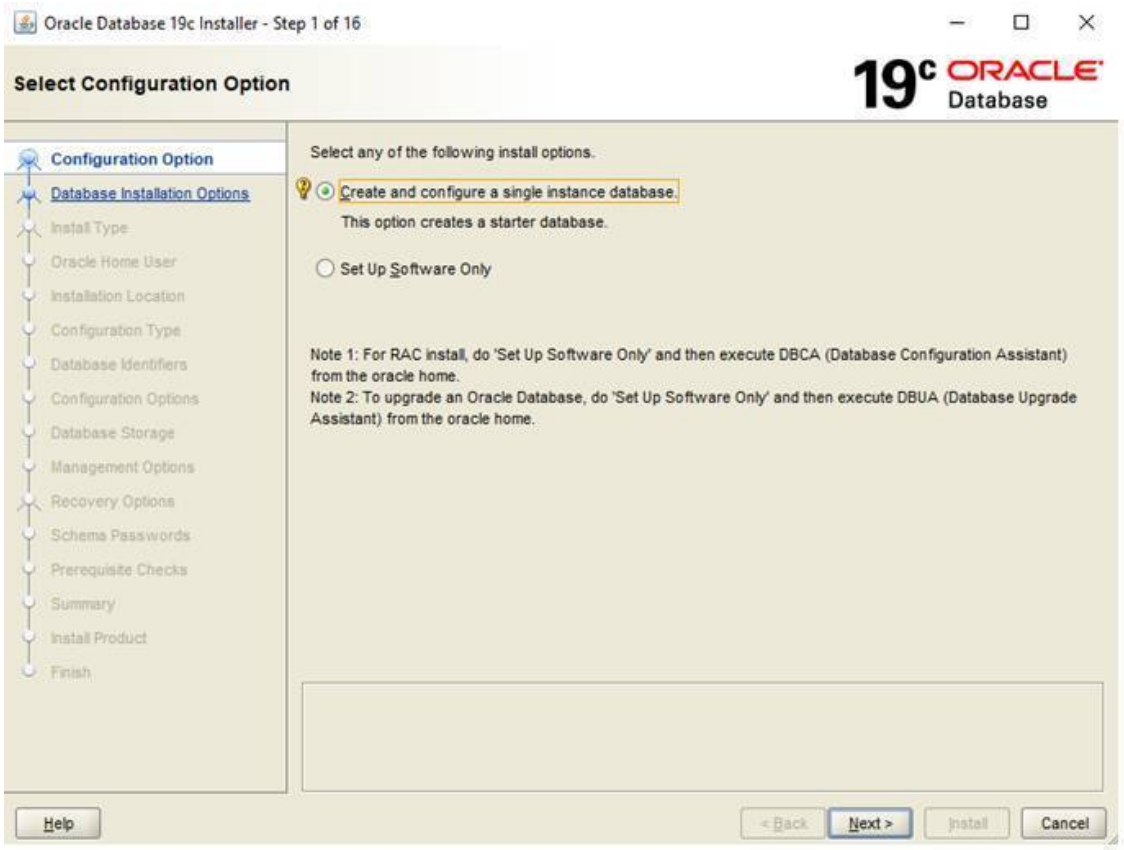
И последний шаг, который является обязательным уже несколько лет. На момент скачивания СУБД ORACLE, у тебя должен быть личный кабинет на сайте ORACLE. Если его нет, то ORACLE предложит его создать в несколько шагов. Из обязательного, тебе потребуется подтвердить адрес электронной почты и, если хочешь, при регистрации, можно установить флажок о желании получать приглашения на различные бесплатные обучающие вебинары от Oracle! Больше ничего и это займет еще несколько минут. Если регистрация в личном кабинете уже есть, нужно будет ввести адрес электронной почты и пароль, который ты ранее придумал.

Все готово, мы скачали Oracle на наш компьютер в виде стандартного zip–архивчика. Теперь его нужно распаковать. Распаковать лучше в каталог, путь до которого будет содержать только латинские буквы и цифры, то есть без пробелов и кириллицы. Например: «D:\Downloads \ORACLE» или «D:\ORACLE».

Есть еще один важный момент, который нужно учесть перед установкой Oracle. Проверь, пожалуйста, как называется твой компьютер. Каково его сетевое имя. Если он называется подобно «Мария ПК» или «Компьютер Андрея», то вначале его нужно переименовать и перезагрузить компьютер. Имя тоже должно быть дано латинскими буквами, без пробелов и спецсимволов, например: «MariaPC» или «Andry».

В каталоге, куда была распакована СУБД, найди файл setup.exe и запусти его. Сделать это лучше от имени администратора, то есть не двойным кликом, а щелкнув правой кнопкой мыши и выбрав из контекстного меню: «Запустить с правами администратора».

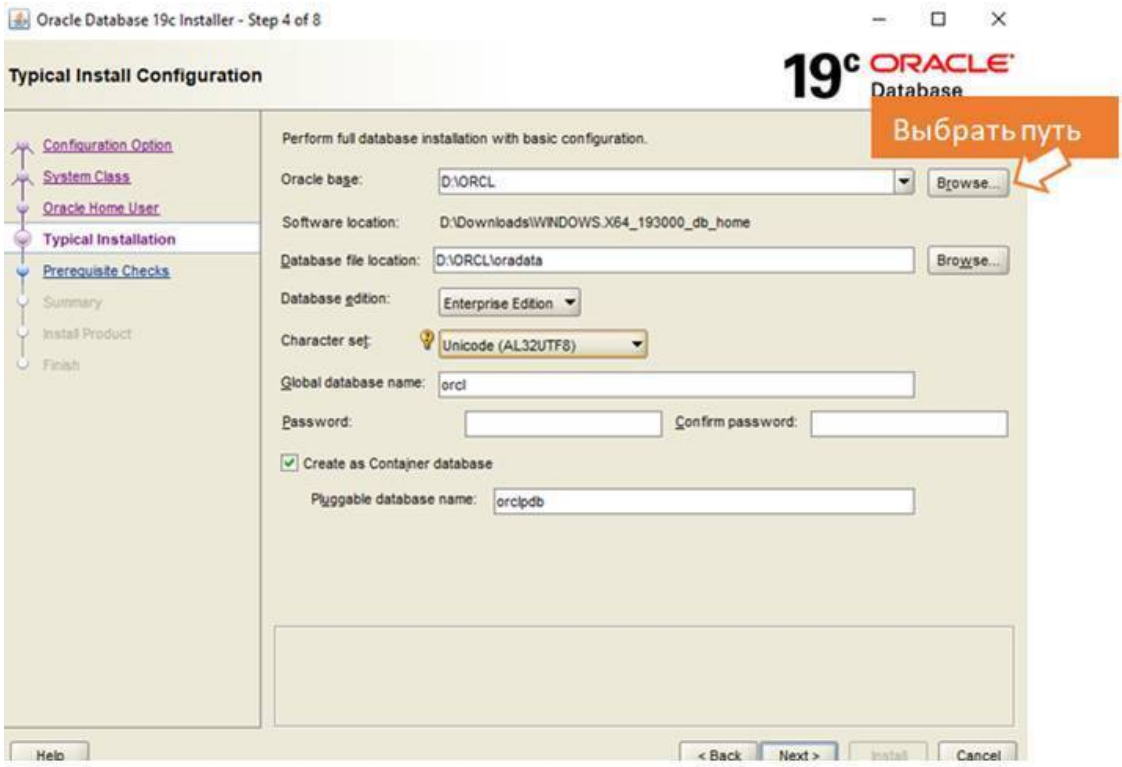
Через несколько мгновений, появится экран установки Oracle. В первом окне нужно выбрать пункт «Create and configure a single database» («Создать и конфигурировать базу данных»). Это самый первый пункт и далее нажать «Next» («Далее»).



В открывшемся втором шаге мастера установки из двух пунктов «Desktop class» и «Server class» необходимо выбрать первый. Будет установлена версия СУБД с конфигурациями, максимально подходящими для домашнего компьютера или ноутбука. «Server class» используется на отдельных серверах, где устанавливаемой СУБД разрешается использовать всю мощность целевой машины. Нажимаем «Next».

На третьем шаге выбираем «Use Virtual Account» и также нажимаем «Next».

На шаге 4 необходимо сначала выбрать путь установки Oracle.



Сделать это можно кликом по верхней кнопке «Browse» («Обзор»). В открывшемся окне выбора пути установки Oracle нужно выбрать пустой каталог, который ты заранее создал (или можно создать папку непосредственно в окне выбора пути установки). Важно, чтобы полный путь до этой папки не содержал символов кириллицы, спецсимволов и пробелов. Полный путь установки может быть таким: «D:\ORCL» или «C:\ORACLE», или, например, «D:\OracleBase». Путь установки ORACLE не может быть вида: «D:\Новая папка».

Когда мы выберем путь установки СУБД Oracle в верхней строке, путь расположения непосредственно создаваемой базы данных в нижней строке изменится автоматически.

В выпадающем списке «Character set» («Набор символов») сейчас нужно выбрать «Unicode AL32UTF8».

В текстовом окне «Global database name» нужно указать имя базы данных. Можно назвать, например, «work», «study», «mybase» или оставить предлагаемое Ораклом значение по–умолчанию – «orcl».

Далее необходимо установить самый главный пароль системного пользователя ORACLE. Его нельзя забывать. Для нашей базы данных мы можем сейчас выбрать самый простой пароль, даже 123, и в поле «Confirm» еще раз его продублировать. Нажимаем «Next» и получаем сообщение, что пароль администратора ORACLE не соответствует рекомендациям ORACLE. Конечно, ведь он такой простой и короткий. Но мы работаем не с реальными данными какой–либо компании, а устанавливаем ORACLE для тренировок. Поэтому, простота пароля нас волновать не должна и Оракл на вопрос о том, действительно ли мы хотим продолжить с таким легким паролем системного администратора, мы отвечаем «Yes».

На следующем шаге Oracle проверяет настройки операционной системы и оценивает производительность нашего компьютера (или ноутбука) и, если для комфортной работы СУБД будет каких–либо параметров недостаточно, то Oracle даст об этом знать. Например, может сообщить, что недостаточно оперативной памяти. В этом случае, в окне устанавливаем галочку «Ignore All» («Игнорировать все замечания») и нажимаем появившуюся кнопку «Next».

На этом, практически последнем шаге, Оракл показывает, что сейчас будет выполнено. Нажимаем заветную кнопку «Install» («Установить»).

Во время установки могут появиться дополнительные окна с запросом разрешений, например, от брандмауэра операционной системы. В появляющихся окнах нужно всегда нажимать «Разрешить».

Через некоторое время установка будет завершена. Поздравляю! Мы сделали первую одну из самых сложных задач по подготовке нашего собственного рабочего места – мы установили СУБД Oracle!

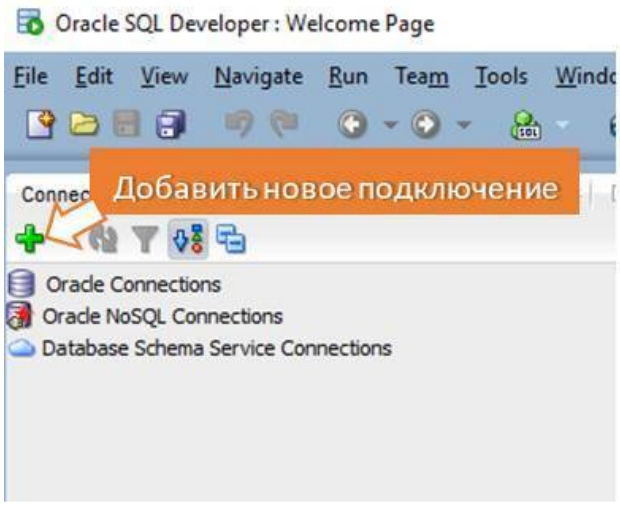
5.2. Установка программы SQL Developer и подключение к базе данных

Программа SQL Developer – стандартная программа производства фирмы Oracle для работы с базами данных СУБД Oracle. После установки этой программы, с помощью нее мы подключимся к нашей СУБД и зайдем на первую и единственную пока базу данных. Иными словами, с помощью нее мы будем видеть объекты базы данных, к которой подключились (например, таблицы), а также выполнять SQL запросы. Ввиду наличия всего нужного функционала и отсутствия дополнительной платы, эта программа стала одной из самых популярных для работы с базами данных Oracle. Примерно такой же популярностью пользуются и программы сторонних разработчиков, предназначенные для работы с Оракловыми базами данных: PL/SQL Developer и TOAD.

В последних версиях СУБД Oracle, программа SQL Developer не идет в комплекте поставки – ее нужно устанавливать дополнительно. Но, если ты устанавливаешь СУБД Oracle версию 12, то программа SQL Developer будет установлена автоматически. Ты можешь найти ее в меню «Пуск».

Итак, если ты установил Oracle новой версии, то, скорее всего, программа SQL Developer у тебя отсутствует. Ты можешь проверить это. Попробовать найти ее в меню Пуск. Отсутствует? Тогда устанавливаем ее дополнительно! Снова идем на сайт Oracle: https://www.oracle.com/tools/downloads/sqldev–downloads.html и выбираем программу для нашей операционной системы. Если у нас шестидесяти четырёх битная Windows 10, то и нажимаем на кнопку «Download» напротив нее. Снова соглашаемся с лицензионными соглашениями и загружаем архив с программой. Загрузив zip–архив, его осталось только распаковать в нужный каталог.

После распаковки архива с программой в удобный каталог, можно сразу запустить SQL Developer! При первом запуске, он установит необходимые настройки в операционную систему и сразу можно начинать пользоваться. После того, как SQL Developer полностью запустится, нужно создать новое подключение. Создав и сохранив новое подключение, далее будем его только выбирать, не нужно будет его больше настраивать. Для создания нового подключения, нажимаем кнопку как на рисунке ниже.



В открывшемся окне заполняем поля:

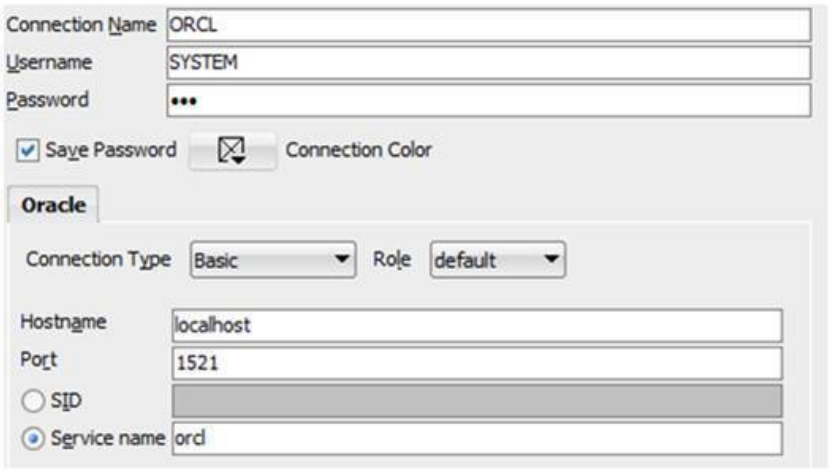
«Connection name» («Название подключения») – любое понятное Вам название, например, «Моя база»;

«Username» («Имя пользователя») – имя пользователя базы данных (см. ниже);

«Password» («Пароль») – пароль пользователя базы данных.

В качестве имени пользователя сейчас можно указать SYSTEM (это системный пользователь–администратор) и пароль, который мы придумали при установке СУБД.

Далее можем установить галочку «Save password» («Сохранить пароль»), чтобы, в последствии, подключаясь каждый раз к нашей базе данных, не вписывать пароль.



В предыдущей главе мы установили ORACLE. По–умолчанию, СУБД создает трех системных пользователей–администраторов, наделенных нужными правами: SYSTEM, SYS и SYSMAN. В последствии мы сможем сами создавать на нашей базе данных дополнительные новые учетные записи (пользователей) и давать им нужные привилегии. Так обычно и делается. Никто не работает в системных учетных записях, кроме самих администраторов баз данных, так как системные учетные записи имеют слишком много привилегий. Системные учетные записи нужны для обслуживания базы данных, выполнения важных административных настроек, управления. Так, например, с помощью них можно даже удалить всю базу данных целиком. Обычные пользователи таких привилегий не имеют. Заполняем дальше:

«Connection type» («Тип подключения») – выберем «Basic»;

«Role» («Роль») – полномочия, с которыми мы будем подключаться к базе данных, выберем «default» (то есть «по–умолчанию»);

«Hostname» («Имя сервера») – имя компьютера или адрес в Интернете, где расположен сервер базы данных. В нашем случае, сервер – это и есть наш локальный компьютер, поэтому напишем localhost (как на фото выше);

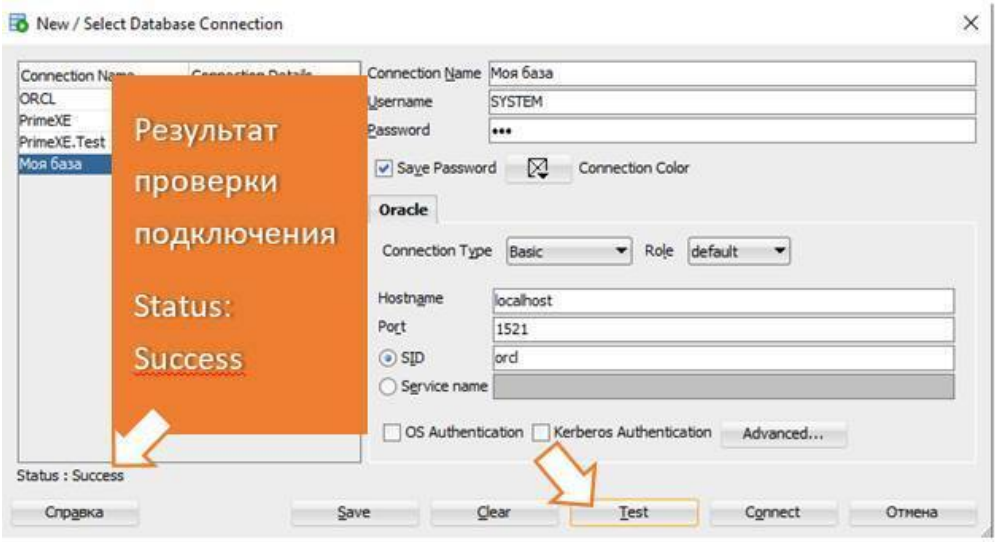
«Port» («Порт») – порт сервера (если ты мало знаком с айти, то порт – это, в нашем случае, как бы «номер сетевой двери», в которой Оракл ожидает прием команд извне для выполнения их в базе данных) – установим стандартный порт 1521, так как мы его при установке СУБД не меняли;

«SID» или «Service name» – можно установить переключатель в любой из двух положений и указать имя базы данных, которое мы дали при установке. В нашем случае это «orcl».

После того, как указали все настройки подключения, нажимаем кнопку «Test», чтобы проверить подключение. В результате мы должны увидеть «Status: Success» («Состояние: Успех»). Теперь нажимаем кнопку «Save» («Сохранить»), чтобы сохранить новое подключение и потом кнопку «Connect» («Подключиться»).

Теперь у нас установлена одна из самых востребованных систем управления базами данных, специалисты в области которой имеют высокооплачиваемую зарплату – это СУБД ORACLE! У нас также уже установлена программа SQL Developer, через которую мы будем работать с нашей базой данных и даже настроено подключение к нашей пока пустой базе данных.

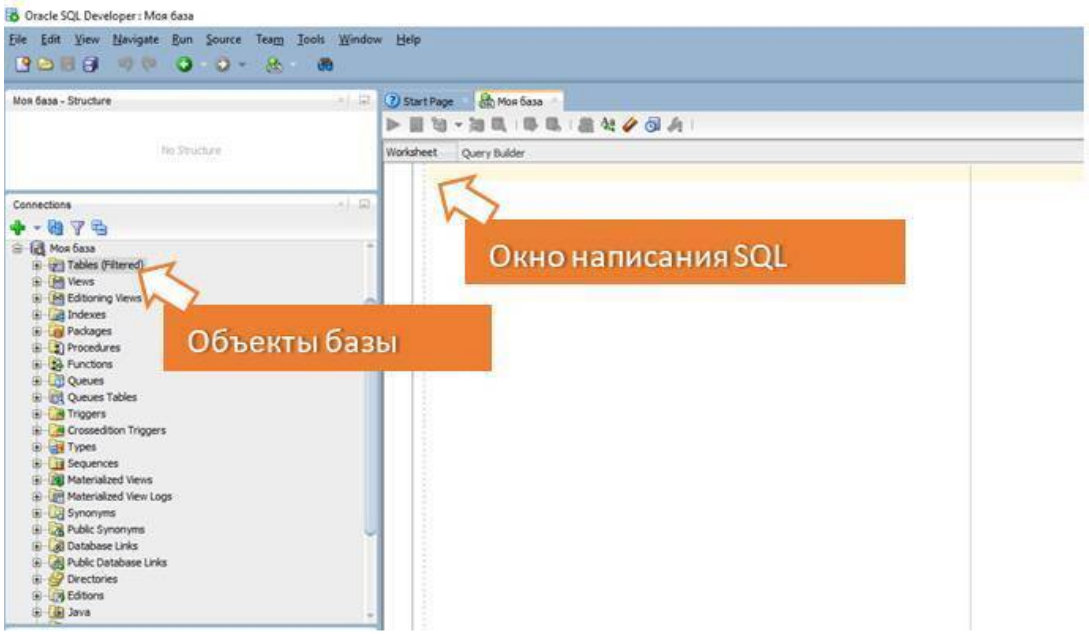
Осталось только закачать в нашу пока пустую базу данных тестовые данные.



5.3. Создание тестовой базы данных

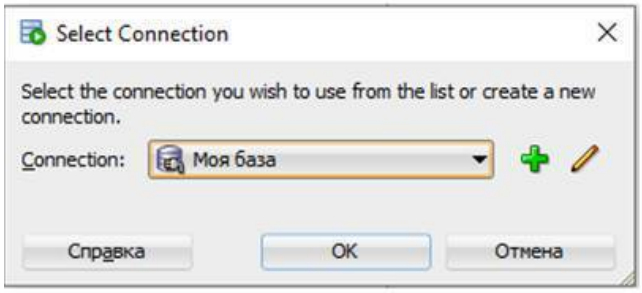
Итак, с помощью кнопки «Connect» мы соединились с нашей базой данных, иначе говоря, зашли в нее.

Слева, в дереве объектов базы данных активного подключения, мы можем видеть все объекты базы данных, разделенные по группам (папкам). Например, таблицы расположены в группе «Tables».



Большое окно, занимающее почти весь экран – это окно написания SQL–запросов. Здесь мы будем проводить почти все наше рабочее время и создавать исключительно шедевры на языке SQL! Можно одновременно работать с более чем одним запросом. Например, составляя и продумывая один большой SQL–запрос, мы можем открыть еще одно SQL–окно и написать некоторый другой запрос на проверку некоторых сторонних данных.

Чтобы открыть новое (или первое) SQL окно (если мы его закрыли), нужно сверху в главном меню нажать «Tools» («Инструменты») – «SQL Worksheet» («SQL окно»). Или, по–умолчанию, Alt+F10. Далее, в появившемся окне, нужно выбрать к какой базе будем писать SQL запросы – выберем наше единственное подключение и нажмем «OK».



Если бы у нас было бы несколько подключений, то в одном SQL–окне мы могли бы работать с одной базой данных, например, с тестовой, а во втором SQL–окне мы могли бы работать

с боевой базой данных. Получив запросами данные из разных баз данных в разных окнах, мы могли бы их, например, сравнивать. Хотя для этого можно было бы использовать более удобные средства.

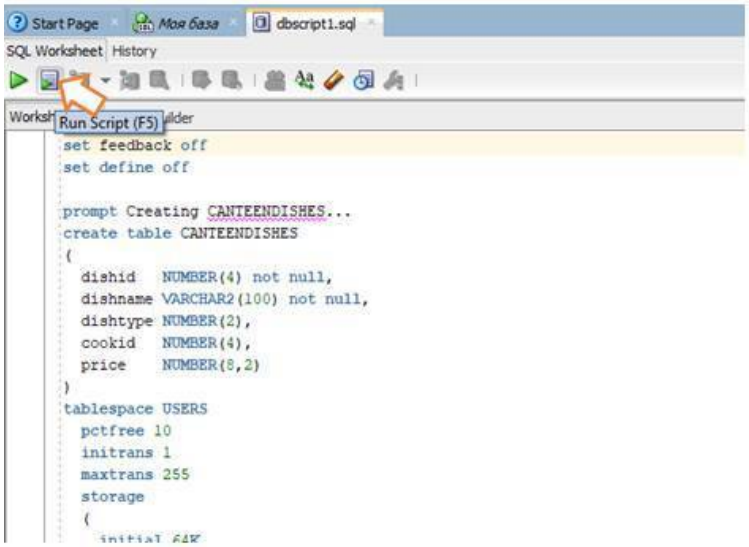
Открыть новое SQL–окно можно, также, нажав кнопку на панели инструментов.

Приступим к загрузке в нашу базу данных тестовых данных. Вначале закачиваем скрипт на компьютер:

https://prime–soft.biz/download/dbscript1.sql

Теперь нажимаем «File» («Файл») – «Open» («Открыть») или одноименную кнопку на панели инструментов (в виде открывающейся папки) и выбираем скаченный файл. В открывшемся окне нажимаем кнопку на панели инструментов «Run Script» («Выполнить скрипт»).

После выполнения скрипта наша база данных будет готова!



В группе объектов «Tables» («Таблицы») в дереве объектов базы данных появятся новые объекты. Если хотим видеть их прямо сейчас, а не дожидаться перезапуска программы SQL Developer, – нужно щелкнуть правой кнопкой мыши по объектам «Tables» («Таблицы») и выбрать «Refresh» («Обновить»).

В группе объектов «Tables» («Таблицы») мы видим сейчас достаточно много объектов и большая часть из них системные, то есть они не были добавлены нашим скриптом и мы с ними работать не будем. Почему же тогда они отображаются? Потому, что мы сейчас используем системную учетную запись. Мы администраторы. По–хорошему, уже бизнес (тестовые) объекты (таблицы с данными) мы могли прогрузить в отдельной схеме данных, а не в системном пользователе, чтобы слева в дереве объектов не отображались «лишние» таблицы. Но мы итак уже проделали слишком много работы для подготовки нашего рабочего места. Поэтому, пока остановимся на том, что будем работать (практиковаться, учиться) из–под пользователя SYSTEM. И ничего, что мы видим «лишние» таблицы в дереве объектов. Мы просто скоро

запомним некоторый набор таблиц, которые мы будем использовать и с которыми будем практиковаться при написании SQL–запросов. Системные таблицы трогать не будем!

Мы установили ORACLE, программу SQL Developer, настроили (создали и сохранили) подключение к установленной СУБД и еще загрузили тестовые данные для выполнения практических задач! Мы подготовили рабочее место!

Сейчас самое время сделать небольшую паузу. Заслуженно отдохнуть. Со следующей главы мы продолжим изучать SQL, и теперь, в конце каждой главы, мы будем выполнять самостоятельные практические задачи! Итак, начинаем наш курс!

6. Операторы сравнения IN, LIKE и BETWEEN

Помимо основных операторов сравнения (больше, меньше, равно, меньше или равно, больше или равно) в языке SQL есть операторы, упрощающие выборку данных по диапазону или множеству. Например, если нужно из таблицы банковских операций отобрать те, у которых код операции 12, 23, 48, 49, 54 или один из еще некоторого множества чисел, то, чтобы не перебирать все эти значения при сравнении со столбцом через оператор OR, можно делать запросы вида:

```
SELECT *
  FROM Oper
 WHERE OPER_CODE IN (12, 23, 48, 49, 54, 106, 110, 125)
```

Отберутся операции, у которых в столбце «OPER_CODE» значение входит в перечисленное множество, иначе говоря, одно из них. Символ звездочки после слова SELECT указывает на то, что будут выбраны все столбцы таблицы «Oper».

Отберем сотрудников, работающих в первом филиале в первом, втором или третьем департаменте.

```
SELECT *
  FROM Persons
 WHERE FilialID = 1
   AND DepartamentID IN (1, 2, 3)
```

Надеюсь, что применение оператора IN достаточно хорошо стало понятно. Теперь рассмотрим следующий оператор. В начале выберем сотрудника, зная его точное ФИО:

```
SELECT NAME, BIRTHDATE, FilialID
  FROM Persons
 WHERE NAME = 'Караваева Людмила Сергеевна'
```

Получаем результат:

	NAME	BIRTHDATE	FILIALID
1	Караваева Людмила Сергеевна	19.09.82	1

Выбралась конкретная строчка из таблицы, у которой в столбце NAME строгое соответствие запрашиваемому текстовому значению.

Для того, чтобы искать не по точному соответствию, а по фрагменту или маске, можно использовать оператор LIKE (*like – это не только с англ. «нравиться», но и «как», в смысле «похож»*).

```
SELECT *
  FROM Persons
 WHERE NAME LIKE '%Анна%'
```

В результате из таблицы сотрудников будут отобраны Анны, а точнее те строчки, у которых в столбце NAME есть фрагмент «Анна». Значки процентов в начале и конце слова означают, что в этом месте может быть еще текст. То есть слово «Анна» должно быть в NAME, но оно может сопровождаться в начале и в конце еще текстом. И такие совпадения будут отобраны. Если в начале или в конце текста не будет, а также, если NAME состоит целиком только из «Анна», то такие строчки тоже будут отобраны. Результат:

	PERSONID	NAME	BIRTHDATE	DEPARTAMENTID	FILIALID
1	5	Лимонадова Анна Васильевна	12.02.77	3	2
2	16	Шилова Анна Ивановна	19.11.80	2	3

Попробуем отобрать всех сотрудников с именем «Иван»:

```
SELECT *
  FROM Persons
 WHERE NAME LIKE '%Иван%'
```

Сколько много данных вернулось! Похоже, здесь есть что–то лишнее:

	PERSONID	NAME	BIRTHDATE	DEPARTAMENTID	FILIALID
1	1	Иванов Иван Иванович	12.04.81	1	1
2	11	Бодров Иван Антонович	17.06.88	2	3
3	14	Первый Иван Иванович	08.09.77	1	3
4	15	Сидоров Иван Денисович	29.10.79	2	3
5	16	Шилова Анна Ивановна	19.11.80	2	3
6	17	Иванова Тамара Николаевна	15.05.77	4	1

Мы видим, что помимо сотрудников с именем «Иван» отобрались еще и те, у кого в столбце NAME есть этот фрагмент не в составе имени, а является частью фамилии или отчества. Что же делать? Мы можем в качестве фрагмента поиска указать « Иван » с пробелами в начале и конце! Таким образом это будет означать, что слева и справа есть еще слова, отделенные от «Иван» пробелами – это фамилия и отчество.

```
SELECT *
  FROM Persons
 WHERE NAME LIKE '% Иван %'
```

Теперь всех отобранных сотрудников точно зовут «Иван»:

	PERSONID	NAME	BIRTHDATE	DEPARTAMENTID	FILIALID
1	1	Иванов Иван Иванович	12.04.81	1	1
2	11	Бодров Иван Антонович	17.06.88	2	3
3	14	Первый Иван Иванович	08.09.77	1	3
4	15	Сидоров Иван Денисович	29.10.79	2	3

Если бы нам нужно было найти всех сотрудников, чья фамилия начинается со слова «Иван», то оператору LIKE необходимо было быть дать значение:

```
SELECT *
  FROM Persons
 WHERE NAME LIKE 'Иван%'
```

Найдутся сотрудники, у которых в графе NAME значение начинается с «Иван»:

	PERSONID	NAME	BIRTHDATE	DEPARTAMENTID	FILIALID
1	1	Иванов Иван Иванович	12.04.81	1	1
2	17	Иванова Тамара Николаевна	15.05.77	4	1

Помимо знака «%» при поиске с LIKE можно использовать символ нижнего подчеркивания – «_». Он означает обязательно один любой символ (буква, цифра, символ). Например, при поиске автомобиля по регистрационному номеру, мы хотим найти тот у которого в номере буква «в», «а» и «а», но между первой и второй буквой идут три цифры, тогда мы можем использовать «маску» поиска, указав после первой «в» три нижних подчеркивания:

```
SELECT *
  FROM PersonCars
 WHERE CARREGNUMBER LIKE 'в___аа%'
```

Так как в таблице «PersonCars» («Автомобили сотрудников») в столбце CARREGNUMBER регистрационный номер указан с кодом региона и страны, то после букв «а» мы указали «%». Отобранные автомобили имеют номера, начинающиеся на букву «в» и соответствующие маске поиска:

	PERSONID	CARREGNUMBER	CARNAME
1	3	в575аа 69 RUS	Тойота Камри
2	14	в700аа 179 RUS	Фольксваген Пассат B5

По–умолчанию, поиск с LIKE, а также по точному совпадению, регистрозависимый, то есть, если в таблице PersonCars был бы автомобиль, с большими буквами в регистрационном номере, то мы бы его не отобрали. Упустили. Существуют настройки СУБД, позволяющие делать поиск регистронезависимым, но это, почти всегда, не практикуется. Чуть позже мы сами немного доработаем наш запрос, и он будет работать на любых базах регистронезависимо. Мы

будем находить строчки из таблицы, соответствующие маске, и будет не важно, какими буквами написан текст в сравниваемом столбце – мы научимся всегда отбирать нужные данные!

Теперь посмотрим, как можно отбирать данные, где значение в столбце входит в некоторый диапазон. Например, может потребоваться отобрать платежи, совершенные за некоторый временной промежуток. Или мы можем отобрать сотрудников, чьи даты рождения попадают в определенный диапазон. По–простому, мы можем написать так:

```
SELECT *
  FROM Persons
 WHERE BIRTHDATE >= '01.01.1980'
   AND BIRTHDATE <= '31.12.1989'
```

Отберутся все строчки из Persons, для которых в столбце BIRTHDATE значение больше или равно начальной дате и одновременно меньше или равно конечной дате. Каждая отбираемая строчка будет проверена одновременно на два условия. Чтобы упростить выбор данных за диапазон, мы можем использовать оператор BETWEEN:

```
SELECT *
  FROM Persons
 WHERE BIRTHDATE BETWEEN '01.01.1980' AND '31.12.1989'
```

Выглядит проще, не так ли? Теперь имя столбца в условие пишется только один раз. Отберутся строки, для которых дата рождения между «01.01.1980» и «31.12.1989».

Чтобы воспользоваться оператором BETWEEN, его нужно написать сразу после названия столбца, значение которого необходимо принимать для сравнения, затем первую границу диапазона, потом AND и последнюю границу диапазона:

```
<Имя_столбца> BETWEEN <Начальная_граница> AND <Конечная_граница>
```

Отлично! Теперь мы можем выбирать данные с условием за диапазон. Первое, что нужно поправить, это правильную подачу дат в SQL–запрос. Почему данная вставка дат не корректная и к чему это может привести? Если мы попробуем выполнить на нашей базе данных один из запросов в котором есть даты в условии, приведенных выше, то мы даже можем получить ошибку от Oracle. И это будет правильно! Сейчас в запрос в кавычках мы пишем дату, но то, что пишется в кавычках для СУБД – это текст. Согласно полученному от нас SQL–запросу, Oracle «понимает», что необходимо отбирать данные, опираясь на столбец, в котором лежат даты и что нас интересуют такие строчки из таблицы, где дата в определенном диапазоне (дат). Чтобы понять, входит ли дата в строке в определенный диапазон (дат), необходимо сначала текст, содержащий начальную и конечную дату диапазона, преобразовать в даты. И Oracle до выполнения запроса, выполняет неявное преобразование текста в даты. Такое преобразование текста в дату называется неявным, так как оно осуществляется без нашего явного на то указания. Но Ораклу нужно выполнить запрос чтобы сравнить даты рождения сотрудников и понять, попадают ли они в диапазон, вот он и выполняет такое преобразование!

Как писать запросы с включенными в них датами, мы узнаем из следующей темы.

7. Преобразование данных

7.1 Функция to_date

Итак, мы поняли, что при написании дат в SQL–запросах в виде текста (то есть дат в кавычках, так как все, что в кавычках – это текст), перед выполнением SQL–запросов, даты из текста будут автоматически распознаны. Текст будет неявно преобразован в дату. Неявно, означает, что без явной нашей команды на преобразование. ORACLE, на основании своего понимания как должна выглядеть дата, делает ее извлечение из текста.

Например, в SQL запросе указана дата вида: '01.03.1980'. На одном компьютере, ORACLE может «понять», что первые две цифры – это номер дня, затем, после точки, идет номер месяца и в конце указан четырехсимвольный номер года. На другом компьютере СУБД распознает дату с предшествующим месяцем, затем с днем. То есть, наоборот.

В первом случае, будут отобраны данные за первое марта, во втором случае – за третье января. Такое использование дат в SQL запросах крайне опасно. Особенно в описанном выше случае – ORACLE «смог» из текста распознать дату, но мы не узнали, что он распознал ее неправильно. СУБД не выдала ошибку. На основе полученных данных мог строиться анализ и приниматься важное решение.

Если бы в запрос передавалась дата '21.01.1980', а Оракл ожидал бы в начале месяц, а потом день, то при выполнении запроса, мы бы получили ошибку о некорректном номере месяца. Получение явной ошибки от Оракла – это лучше, чем ее скрытое наличие. Теперь самое время научиться вписывать даты в запрос правильно.

Для того, чтобы правильно вписать дату в SQL–запрос, необходимо ее явно преобразовать из текста в дату с указанием того, как она должна быть преобразована. Что из написанного день, а что – месяц. Для этого воспользуемся функцией Oracle – to_date.

В функции to_date вначале нужно указать какой текст должен быть преобразован в дату, а затем нужно указать маску (формат) преобразования.

```
to_date(<текст>, <формат>)
```

Пример использования функции to_date:

```
SELECT *
  FROM Persons
 WHERE BIRTHDATE BETWEEN to_date('01.01.1980', 'dd.mm.yyyy')
                     AND to_date('31.12.1989', 'dd.mm.yyyy')
```

Приведенный выше запрос выбирает из таблицы Persons сотрудников, родившихся в 80–е.

Напишем запрос, выводящий все заказы в платной столовой (расположенной на пятом этаже в нашей компании), сделанные 10 марта 2019 года:

```
SELECT *
  FROM PersonCanteenOrder
 WHERE DATEORDER = to_date('10.03.2019', 'dd.mm.yyyy')
```

Получаем результат (на скриншоте ниже показаны не все отобранные данные – в целях экономии места):

	PERSONID	DATEORDER	DISHID
1	3	10.03.19	40
2	4	10.03.19	5
3	4	10.03.19	10
4	4	10.03.19	19
5	4	10.03.19	24
6	4	10.03.19	30
7	17	10.03.19	9
8	17	10.03.19	28
9	17	10.03.19	3

В колонке PersonID идентификатор сотрудника, сделавшего заказ. В колонке DateOrder дата заказа, и в колонке DishID – идентификатор заказанного блюда.

7.2. Функция to_char

Еще одна функция преобразования Oracle. В отличии от рассмотренной предыдущей функции, функция to_char делает обратное – преобразовывает в текст. Имеет два главный входных параметра: первый параметр – число или дата, которую нужно преобразовать в текст; и второй параметр – формат преобразования (как именно нужно преобразовать в текст). Есть еще и третий необязательный параметр, который мы пока опустим – это языковые параметры.

Сначала рассмотрим преобразование даты в текст. Проиллюстрируем преобразование даты в текст на примере дат рождения сотрудников. Напишем запрос, выводящий дату рождения, например, в формате месяц прописью и год цифрой. На такое способна только функция to_char!

```
SELECT NAME,
       BIRTHDATE,
       to_char(BIRTHDATE, 'Month YYYY')
  FROM Persons
```

Получим результат:

	NAME	BIRTHDATE	TO_CHAR(BIRTHDATE,'MONTHYYYY')
1	Иванов Иван Иванович	12.04.81	Апрель 1981
2	Петров Павел Сергеевич	16.09.82	Сентябрь 1982
3	Караваева Людмила Сергеевна	19.09.82	Сентябрь 1982
4	Черных Александр Александрович	22.01.72	Январь 1972
5	Лимонадова Анна Васильевна	12.02.77	Февраль 1977
6	Белый Олег Викторович	30.03.86	Март 1986
7	Финина Алла Павловна	01.03.84	Март 1984
8	Некрасов Дмитрий Валерьевич	06.03.82	Март 1982
9	Гостин Сергей Олегович	12.04.82	Апрель 1982
10	Розовая Лидия Георгиевна	14.05.82	Май 1982
11	[illegible]	[illegible]	[illegible]

После ключевого слова SELECT мы указали вывод данных столбца NAME, BIRTHDATE и преобразованного столбца BIRTHDATE с помощью функции to_char. Функция привела выводимые даты к запрашиваемому во втором параметре формату – месяц прописью и год числом. Если в качестве формата преобразование указать буквенное обозначение месяца, написанное в нижнем регистре, то функция to_char выведет результат с именем месяца также в таком же регистре:

```
SELECT NAME,
       BIRTHDATE,
       to_char(BIRTHDATE, 'month YYYY')
  FROM Persons
```

Получим:

	NAME	BIRTHDATE	TO_CHAR(BIRTHDATE,'MONTHYYYY')
1	Иванов Иван Иванович	12.04.81	апрель 1981
2	Петров Павел Сергеевич	16.09.82	сентябрь 1982
3	Караваева Людмила Сергеевна	19.09.82	сентябрь 1982
4	Черных Александр Александрович	22.01.72	январь 1972
5	Лимонадова Анна Васильевна	12.02.77	февраль 1977
6	Белый Олег Викторович	30.03.86	март 1986
7	Финина Алла Павловна	01.03.84	март 1984
8	Некрасов Дмитрий Валерьевич	06.03.82	март 1982
9	Гостин Сергей Олегович	12.04.82	апрель 1982
10	Розовая Лидия Георгиевна	14.05.82	май 1982
11	Бодров Иван Антонович	17.06.88	июнь 1988
12	Симонова Ольга Петровна	07.07.79	июль 1979

Если укажем в верхнем регистре, то и получим тоже в верхнем.
Значение месяца мы также можем вывести числом, если укажем это в формате:

```
SELECT NAME,
       BIRTHDATE,
       to_char(BIRTHDATE, 'MM YYYY')
  FROM Persons
```

Получим:

	NAME	BIRTHDATE	TO_CHAR(BIRTHDATE,'MMYYYY')
1	Иванов Иван Иванович	12.04.81	04 1981
2	Петров Павел Сергеевич	16.09.82	09 1982
3	Караваева Людмила Сергеевна	19.09.82	09 1982
4	Черных Александр Александрович	22.01.72	01 1972
5	Лимонадова Анна Васильевна	12.02.77	02 1977
6	Белый Олег Викторович	30.03.86	03 1986
7	Финина Алла Павловна	01.03.84	03 1984

И теперь придумаем совсем хитрый формат:

```
SELECT NAME,
       BIRTHDATE,
       to_char(BIRTHDATE, 'DD.MM/YY')
  FROM Persons
```

Выведем день, через точку, месяц и потом через слэш номер года, состоящий из двух символов:

	NAME	BIRTHDATE	TO_CHAR(BIRTHDATE,'DD.MM/YY')
1	Иванов Иван Иванович	12.04.81	12.04/81
2	Петров Павел Сергеевич	16.09.82	16.09/82
3	Караваева Людмила Сергеевна	19.09.82	19.09/82
4	Черных Александр Александрович	22.01.72	22.01/72
5	Лимонадова Анна Васильевна	12.02.77	12.02/77
6	Белый Олег Викторович	30.03.86	30.03/86
7	Финина Алла Павловна	01.03.84	01.03/84

Как видим, мы можем выводить дату практически в любом удобном (нужном) для нас виде. При составлении формата, можем использовать дополнительные символы: пробел, тире, слеш (косая черта), скобки и т.д. Можем выводить, также, и отдельную информацию о дате, например, день недели:

```
SELECT NAME,
       BIRTHDATE,
       to_char(BIRTHDATE, 'Day')
  FROM Persons
```

Результат:

	NAME	BIRTHDATE	TO_CHAR(BIRTHDATE,'DAY')
1	Иванов Иван Иванович	12.04.81	Воскресенье
2	Петров Павел Сергеевич	16.09.82	Четверг
3	Караваева Людмила Сергеевна	19.09.82	Воскресенье
4	Черных Александр Александрович	22.01.72	Суббота
5	Лимонадова Анна Васильевна	12.02.77	Суббота
6	Белый Олег Викторович	30.03.86	Воскресенье
7	Финина Алла Павловна	01.03.84	Четверг

В таблице ниже список основных элементов даты (и времени), которые можно использовать при составлении формата для функции to_char.

Элемент формата даты	Значение
D	День недели (1–7)
DAY	Название дня недели
DD	День месяца (1 – 31)
DDD	День года (1 – 366)
DY	Сокращенное название дня
IW	Неделя года (1 – 52), (1 – 53)
HH, HH12	Час дня (1–12)
HH24	Час дня (0–23)
MI	Минута (0–59)
MM	Месяц (1 – 12), Январь = 1, Декабрь = 12
MONTH	Название месяца
MON	Сокращенное название месяца
Q	Квартал года (1 – 4) С января по март – первый квартал.
SS	Секунды (0–59)
W	Неделя месяца (1–5)
YYY, YY, Y	Последние три, две или одна цифра года.

Теперь нам ни одна задача не страшна! Но это еще не все! Функция to_char может использоваться для преобразования числа в текст, а не только даты. После преобразования числа в текст, для него уже не будут доступны арифметические операции, но зато смотреться будет красиво.

Преобразование числа в текст делается, в основном, для его форматирования, красивого отображения, например, с пробелами–разделителями групп разрядов. Или можно всегда отображать значение с определенным количеством чисел после запятой. Для целого числа, например, после запятой будут отображаться нули. По мере развития отчетных систем, выводящих данные на печать или в отчеты, надобность форматирования чисел самой СУБД при выдаче данных, потеряла актуальность. Сами отчетные системы умеют отображать числа с любым форматированием и для них главное – само число, то есть сами данные. Тем не менее, для полноты темы, ниже мы рассмотрим использование функции to_char для преобразования числа в текст.

Выведем названия блюд столовой, их цены, а также преобразованные значения цен:

```
SELECT DishName, Price, to_char(Price,
'990D99')
```

Получим:

	DISHNAME	PRICE	TO_CHAR(PRICE,'990D99')
1	Компот из яблок	20	20,00
2	Компот из абрикосов	28	28,00
3	Компот из слив	25	25,00
4	Мин.вода "Бон аква"	18	18,00
5	Мин.вода "Шишкин лес"	12	12,00
6	Сок "J7" в ассортименте	20	20,00
7	Суп гороховый	50	50,00
8	Щи	50	50,00
9	Борщ	50	50,00
10	Суп овощной	45	45,00
11	Крем-суп из шампиньонов	55	55,00
12	Суп из морепродуктов	48	48,00

Самое первое на что нужно обратить внимание в маске, это на разделитель целой и дробной части – D (*англ. delimiter – разделитель*). Далее слева и справа от разделителя мы видим цифры «0» и «9». «0» обозначает обязательное число. То есть в ценах блюд столовой до запятой должно быть минимум одно число. Даже если это будет 0. Например, цена 0 рублей 50 копеек. «9» обозначает необязательное число. В нашей маске после разделителя указаны два необязательных числа. При стоимости товара в ровно 20 рублей, после запятой нет чисел. Они бы были если товар стоил 20 рублей 55 копеек, но мы, исходя из того, какие цены у нас есть в столовой, заложили в формат, что после разделителя целой и дробной части, дальше чисел может не быть (в случае целых чисел). И поставили «9» и «9», показывая Оракулу, что нам нужны здесь два знака, но их в значении цены может не быть. Благодаря этому Оракл вывел цену в третьем столбце со знаками после запятой, даже, в случае целых чисел.

Рассмотрим еще один пример. Выведем суммы заработных плат, перечисляемых сотрудникам:

```
SELECT Sum, to_char(Sum, '999G990D99')
  FROM PersonPayments
```

Получаем:

	SUM	TO_CHAR(SUM,'999G990D99')
1	55000,5	55 000,50
2	80000	80 000,00
3	70000	70 000,00
4	45800,5	45 800,50
5	57000,1	57 000,10
6	75000,5	75 000,50
7	90000	90 000,00
8	50000	50 000,00
9	70000	70 000,00
10	45800,5	45 800,50
11	57000	57 000,00
12	75000,5	75 000,50

Спереди разделителя указано только одно обязательное число. Остальные все числа не обязательные. Через 3 цифры вставлен разделитель групп разрядов – буква «G». Это визуальный пробел для облегчения понимания размера числа.

7.3. Функция to_number

Функция to_number эквивалент функции to_char при преобразовании числа в текст, но делает обратное – преобразовывает текст в число. Пример:

```
SELECT to_number('1234.56',
'999999.99')
```

Получаем:

	TO_NUMBER('1234.56','999999.99')
1	1234,56

8. Правильное обращение с NULL («пустыми» значениями)

Это еще одна очень важная тема! Ее знание не просто так спрашивают на собеседованиях! Выберем из таблицы CanteenDishes («Блюда столовой») строчки. Для повторения темы сортировки, заодно, упорядочим выбираемые блюда по цене в обратном порядке:

```
SELECT *
  FROM CanteenDishes
 ORDER BY Price DESC
```

Получаем:

	DISHID	DISHNAME	DISHTYPE	COOKID	PRICE
1	28	Тирамиссу	5	(null)	140,5
2	18	Гуляш из говядины	3	21	110,6
3	17	Рыба жаренная	3	19	102,8
4	14	Бефстроганоф	3	19	98
5	29	Творожная запеканка	5	20	96,4
6	20	Стейк из семги	3	18	88,2
7	16	Бедро куриное	3	20	80
8	19	Котлета жаренная	3	18	78,2
9	33	Пирожное вишневое	5	(null)	68,5
10	26	Плов	4	21	68,4
11	38	Салат греческий	6	19	60,4
12	27	Яблочный штрудель	5	18	60,2
13	37	Сельдь под шубой	6	18	59,6
14	25	Картофель по-русски	4	21	58,2
15	11	Крем-суп из шампиньонов	2	21	55

Здесь нужно обратить внимание на столбец «CookID» («Идентификатор повара»). Из всего ассортимента блюд нашей столовой, есть блюда, идентификатор повара у которых не заполнен. Значение в графе CookID пустое. Для таких строчек в этом столбце программа SQL Developer выводит нам «*(null)*». Важным моментом здесь является то, как выбирать данные, опираясь на этот столбец.

Представим, что перед нами стоит задача вывести все блюда, которые не готовятся в нашей столовой, то есть те, где не указан повар (CookID). Например, пирожное Тирамиссу повара нашей платной столовой сами не готовят. Оно закупное из продовольственной базы. Или, например, питьевая минеральная вода (не попала на скриншоте выше, не уместилась, так как отображены не все строчки). Ее также привозят в больших объемах в маленьких бутылочках и продают в столовой.

Итак, чтобы отобрать строчки из таблицы, где значение в это графе пустое, мы **не** можем написать запрос вида:

```
SELECT *
  FROM CanteenDishes
 WHERE CookID = NULL
```

Вернее можем, но при попытке выполнить его, получим пустой набор данных. Все из–за того, что при сравнении с NULL операторы «равно» и «не равно» не допустимы. Результат всегда будет «Ложь».

Чтобы выбрать блюда столовой с проверкой на NULL в столбце CookID, необходимо использовать оператор IS или IS NOT. Корректируем предыдущий пример:

```
SELECT *
  FROM CanteenDishes
 WHERE CookID IS NULL
```

И вот они, данные:

	DISHID	DISHNAME	DISHTYPE	COOKID	PRICE
1	4	Мин.вода "Бон аква"	1	(null)	18
2	5	Мин.вода "Шишкин лес"	1	(null)	12
3	6	Сок "J7" в ассортименте	1	(null)	20
4	28	Тирамиссу	5	(null)	140,5
5	30	Булочка с маком	5	(null)	48,9
6	33	Пирожное вишневое	5	(null)	68,5

Мы отобрали все блюда, у которых в графе CookID пусто (NULL). Если бы нам нужно было, наоборот, вывести блюда, у которых CookID указан, то мы бы написали:

```
SELECT *
  FROM CanteenDishes
 WHERE CookID IS NOT NULL
```

Получить данные по блюдам, у которых указан CookID мы можем, также и запросом:

```
SELECT *
  FROM CanteenDishes
 WHERE CookID > 0
```

Строчки с пустым CookID не попадут в результирующий набор данных. Данная команда даст результат (мы увидим список блюд, у которых указан идентификатор повара), так как в нашем запросе значение графы CookID сравнивается с 0, а не с NULL. И поэтому можно

обойтись и без IS или IS NOT. Только если нам нужно отобрать строки с пустым CookID, то тут, конечно, без IS NULL не обойтись.

С NULL не только нельзя сравнивать с помощью операторов «равно» и «не равно», но и нельзя применять арифметические операции, иначе результатом будет NULL. Ниже один из любимых вопросов на собеседовании по SQL на понимание принципов работы с NULL:

Выберите верные утверждения (возможно несколько вариантов):

1) NULL <> 1

2) NULL <> NULL

3) NULL = NULL

4) NULL IS NOT NULL

5) NULL IS NULL

Правильным вариантом является только номер 5.

Обрати внимание, что в заголовке этой темы, я во фразе «пустые» значения слово «пустые» взял в кавычки. Это для того, чтобы показать тебе, что хоть при наличии значения NULL в каком-то столбце некоторой записи пусто, NULL обычно трактуется не сколько как пустота, сколько как неопределенность. Вот поэтому при прямом сравнении NULL = NULL будет ответ ложь (false), так как одна неопределенность не может быть равна другой неопределенности, ведь мы не знаем, что скрывается ни под одной из неопределенностей.

Практические задачи №1

1. Написать запрос, выводящий список сотрудников организации, упорядоченный по ФИО.

2. Написать запрос, выводящий список сотрудников организации, упорядоченный по дате рождения. Чтобы сначала вывелись самые молодые сотрудники.

3. Вывести список филиалов.

4. Вывести список блюд столовой, цена которых больше 80 рублей.

5. Вывести список сотрудников, работающих в филиале 1 или 2.

6. Вывести список сотрудников, работающих в первом филиале, родившихся не ранее 01.01.1980.

7. Вывести список сотрудников, фамилия которых начинается на букву «И».

8. Вывести список блюд столовой, цена которых в диапазоне от 70 до 100 рублей. Результат упорядочить по цене, по убыванию.

9. Вывести список блюд столовой, которые готовятся в собственной столовой организации, а не закупаются. У таких строчек указан повар (в столбце CookID присутствует идентификатор повара). Результат отсортировать по наименованию блюда.

Решения практических задач №1 на следующей странице.

Решение практических задач №1

1. Написать запрос, выводящий список сотрудников организации, упорядоченный по ФИО.

```
SELECT *
  FROM Persons
 ORDER BY NAME
```

2. Написать запрос, выводящий список сотрудников организации, упорядоченный по дате рождения. Чтобы сначала вывелись самые молодые сотрудники.

```
SELECT *
  FROM Persons
 ORDER BY BIRTHDATE DESC
```

3. Вывести список филиалов.

```
SELECT *
  FROM Filial
```

4. Вывести список блюд столовой, цена которых больше 80 рублей.

```
SELECT *
  FROM CanteenDishes
 WHERE Price > 80
```

5. Вывести список сотрудников, работающих в филиале 1 или 2.

```
SELECT *
  FROM Persons
 WHERE FilialID IN (1, 2)
```

6. Вывести список сотрудников, работающих в первом филиале, родившихся не ранее 01.01.1980.

```
SELECT *
  FROM Persons
 WHERE FilialID = 1
   AND BIRTHDATE >= to_date('01.01.1980',
'dd.mm.yyyy')
```

7. Вывести список сотрудников, фамилия которых начинается на букву «И».

```
SELECT *
  FROM Persons
 WHERE NAME LIKE 'И%'
```

8. Вывести список блюд столовой, цена которых в диапазоне от 70 до 100 рублей. Результат упорядочить по цене, по убыванию.

```
SELECT *
  FROM CanteenDishes
 WHERE Price BETWEEN 70 AND 100
 ORDER BY Price DESC
```

9. Вывести список блюд столовой, которые готовятся в собственной столовой организации, а не закупаются. У таких строчек указан повар (в столбце CookID присутствует идентификатор повара). Результат отсортировать по наименованию блюда.

```
SELECT *
  FROM CanteenDishes
 WHERE CookID IS NOT NULL
 ORDER BY DishName
```

Ещё больше практики и интересных задач в моем Telegram канале: https://t.me/sql_oracle_databases

9. Соединения таблиц с помощью JOIN

9.1. Что такое соединения. Назначение соединений

Почти всегда при выборке из базы данных недостаточно информации одной таблицы. Часто нужно выводящиеся данные одной таблицы дополнять сведениями из другой.

При выполнении запроса SELECT из таблицы продаж, среди прочих, мы получим столбцы «артикул реализованного товара» и «количество». В отчете, для которого мы составляем SQL–запрос, необходимо отображать не только артикул, но еще и название товара и даже единицу измерения. Но что же делать, эти сведения отсутствуют в таблице продаж. Эти данные лежат в совсем другой таблице – в таблице «товары»!

Для того, чтобы в результате выполнения запроса получить больше данных о товаре, (чтобы выводился не только его артикул), но еще и наименование и единица измерения, мы к данным из таблицы «продаж» добавим данные из таблицы «товары». Каждая выводящаяся строчка из таблицы продаж дополнится столбцами из таблицы «товары», того товара, артикул которго указан в выводящейся строчке из таблицы «продаж».

Для того, чтобы к данным, выводящимся из одной таблицы, уметь добавлять информацию из другой, в языке SQL существуют соединения (джоины) таблиц. Рассмотрим типы соединений.

9.2. LEFT JOIN. Левое внешнее соединение

При таком типе соединиения, данные из таблицы, которая указана в SQL–запросе левее (относительно фразы LEFT JOIN) будут выведены все. Запомнить – просто!

К этим данным будут присоединены данные второй таблицы, расположенной в запросе правее. Поясню на примере. Пусть нам нужно вывести сотрудников и их автомобили. Так как цель задачи в первую очередь вывести сотрудников и во втором столбце вывести их автомобили (при их ниличии), то выборка будет именно из таблицы сотрудников. И уже к выводящимся сотрудникам приджоиним их автомобили. Давайте рассмотрим таблицы:

Таблица **СОТРУДНИКИ**

ID	NAME	BIRTHDATE
1	Иванов Иван Иванович	12.07.1981
2	Петрова Надежда Анатольевна	04.04.1977
3	Никитин Афанасий Константинович	28.09.1979
4	Первый Николай Николаевич	03.01.1982
5	Орешкина Дарья Васильевна	30.05.1983

Таблица **АВТОМОБИЛИ СОТРУДНИКОВ**

ID	ID_PERS	NOMER	NAME
1	1	А123БВ	Audi A4
2	2	Т345АУ	BMW X3
3	2	Б777БА	Ford Mondeo
4	3	С111ПП	Fiat Panda
5	5	В345АА	Mercedes AMG

Необходимо подготовить отчет о **Сотрудниках и их авто**

ФИО	Марка

В таблице «Автомобили сотрудников» столбец «ID» – это всего лишь сквозной идентификатор, первичный ключ (номер машины по-порядку). Он нас сейчас не интересует. А вот столбец «ID_PERS» – это внешний ключ, ссылающийся на таблицу «Сотрудники». Он нам сейчас будет нужен. Другими словами, в графе «ID_PERS» указан идентификатор сотрудника, которому принадлежит автомобиль. Согласно нему видно, что BMW X3 и Ford Mondeo принадлежат Петровой Надежде Анатольевне. А у Первого Николая Николаевича, сотрудника с идентификатором 4, нет ни одного автомобиля.

Конец ознакомительного фрагмента.