Министерство внутренних дел Российской Федерации

Тюменский институт повышения квалификации сотрудников МВД России

Д.Б. Панюшин, М.М. Сазонов

SQL ДЛЯ СОТРУДНИКОВ ПОДРАЗДЕЛЕНИЙ ОПЕРАТИВНО-РАЗЫСКНОЙ ИНФОРМАЦИИ

Учебное пособие

Тюмень
2017

УДК 343.102+025.4
ББК 67.99(2)91+32.973
П 16

Рекомендовано к изданию редакционно-издательским советом
Тюменского института повышения квалификации
сотрудников МВД России

Рецензенты:
доцент кафедры информационной безопасности учебно-научного комплекса
информационных технологий Московского университета МВД России
имени В.Я. Кикотя, кандидат технических наук *К.К. Борзунов*;
начальник кафедры информационного и технического обеспечения ОВД
Дальневосточного юридического института МВД России,
кандидат технических наук, доцент *П.Б. Скрипко*

Панюшин Д.Б., Сазонов М.М.

П 16 SQL для сотрудников подразделений оперативно-разыскной информации: учебное пособие. Тюмень: Тюменский институт повышения квалификации сотрудников МВД России, 2017. 62 с.
ISBN 978-5-93160-245-5

В учебном пособии дается систематизированное изложение информации по вопросам использования SQL-запросов в деятельности подразделений оперативно-разыскной информации при работе с реляционными и SQL-ориентированными базами данных.

Предназначено для слушателей, обучающихся по дополнительным профессиональным программам повышения квалификации сотрудников подразделений оперативно-разыскной информации МВД России, сотрудников подразделений оперативно-разыскной информации системы МВД России, преподавателей образовательных организаций МВД России.

УДК 343.102+025.4
ББК 67.99(2)91+32.973

ISBN 978-5-93160-245-5

Содержание

Предисловие 5
1. Основы реляционных баз данных 6
1.1. Структура реляционной базы данных 6
1.2. Связь между таблицами 7
Вопросы для самоконтроля 8
2. SQL: возникновение и особенности 9
2.1. Возникновение SQL, разработка стандарта 9
2.2. Типы данных 9
Вопросы для самоконтроля 11
3. Создание, удаление и изменение таблиц 12
3.1. Создание таблиц 12
3.2. Создание ограничений в таблицах 12
3.3. Создание индексов 15
3.4. Удаление таблиц 16
3.5. Внесение изменений в таблицы 17
Задание для самоконтроля 18
4. Выборка данных из таблиц 19
4.1. Команда SELECT 19
4.2. Просмотр столбцов таблицы 20
4.3. Изменение порядка столбцов 21
4.4. Устранение повторяющихся значений 22
4.5. Уточнение выборки 23
Задания для самоконтроля 24
5. Применение реляционных и булевых операторов 25
5.1. Реляционные операторы 25
5.2. Булевы операторы 25
Задания для самоконтроля 27
6. Использование специальных операторов 28
6.1. Оператор IN 28
6.2. Оператор BETWEEN 29
6.3. Оператор LIKE 29
6.4. Условие IS 30
6.5. Оператор NOT и специальные операторы 31
Задания для самоконтроля 32
7. Функции агрегирования 33
7.1. Функции агрегирования и их назначение 33
7.2. Использование функций агрегирования 33
7.3. Оператор COUNT 34
Задания для самоконтроля 35
8. Форматирование результатов запросов 36
8.1. Переименование столбцов 36
8.2. Объединение столбцов 36

8.3. Добавление текста в выборку 37
8.4. Сортировка выводимой информации 37
8.5. Сортировка выборки по множеству столбцов 38
Задания для самоконтроля 39
9. Выборки из нескольких таблиц 40
9.1. Особенности выборки из нескольких таблиц 40
9.2. Оператор INNER JOIN 40
9.3. Оператор LEFT OUTER JOIN 42
9.4. Оператор RIGHT OUTER JOIN 43
9.5. Оператор FULL OUTER JOIN 44
9.6. Соединение таблиц с дополнительным условием 45
9.7. Соединение более чем двух таблиц 45
Задания для самоконтроля 46
10. Вложенные запросы 47
10.1. Назначение подзапросов 47
10.2. Значения, получаемые из подзапросов 48
10.3. Применение многострочных операторов сравнения IN, ANY, ALL в подзапросах 48
10.4. Создание многоуровневых подзапросов 49
Задания для самоконтроля 51
11. Ввод, удаление и изменение значений полей 52
11.1. Ввод новых строк 52
11.2. Удаление строк 54
11.3. Изменение данных 55
Задания для самоконтроля 56
Список рекомендуемой литературы 57
Приложение № 1. Инфологическая модель базы данных, используемая при работе с учебным пособием 58
Приложение № 2. Таблицы и данные, используемые при работе с учебным пособием 59

Предисловие

Эффективное решение задач, стоящих перед оперативными подразделениями органов внутренних дел, невозможно без удовлетворения потребности каждого сотрудника в нужный момент получить систематизированную информацию по интересующему его вопросу. Удовлетворение этой потребности достигается за счет создания и использования информационных систем.

Большинство современных информационных систем построены на основе реляционной модели. В связи с этим каждый сотрудник специализированных подразделений, выполняющих информационно-аналитическое обеспечение оперативно-разыскной деятельности, для результативного решения стоящих перед ним задач обязан владеть знаниями структурированного языка запросов (далее – SQL) с целью обеспечения возможности работы с информацией, содержащейся в SQL-ориентированных базах данных.

В учебном пособии рассмотрены основы теории реляционных баз данных и синтаксиса SQL. Приведенные в учебном пособии примеры позволяют выявить особенности использования структурированного языка запросов для реляционных баз данных, используемых подразделениями оперативно-разыскной информации.

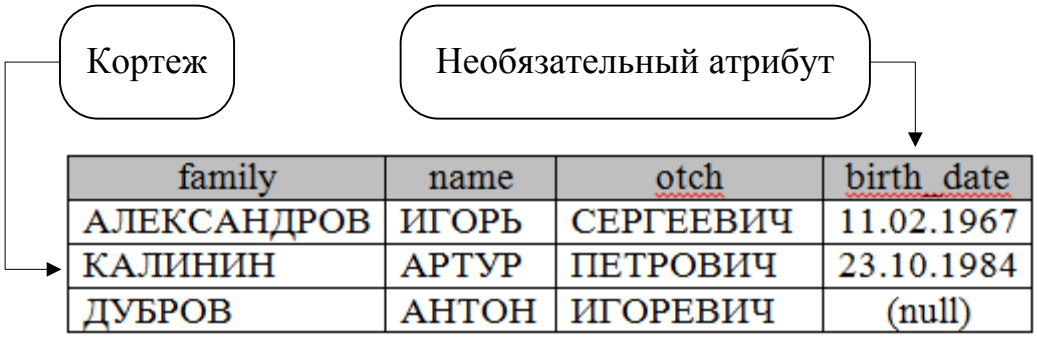
Изложенный в учебном пособии материал предназначен для дополнительного профессионального обучения сотрудников подразделений оперативно-разыскной информации системы МВД России, позволяет им систематизировать уже имеющиеся у них знания SQL и сформировать компетенции, необходимые для работы с реляционными или SQL-ориентированными базами данных. Учебное пособие состоит из одиннадцати тем. По каждой теме приводятся задания для закрепления рассмотренного материала.

Следует отметить, что хотя настоящее издание предназначено главным образом для сотрудников подразделений оперативно-разыскной информации системы МВД России, однако сфера его возможного практического использования не ограничивается подготовкой сотрудников только названной категории. Изучение представленного в пособии материала будет полезно также сотрудникам правоохранительных органов, планирующим использование SQL-ориентированных баз данных в своей служебной деятельности.

1. Основы реляционных баз данных

1.1. Структура реляционной базы данных

Реляционные базы данных основаны на реляционной модели данных. Термин «реляционный» означает, что такая модель данных основана на математическом понятии отношение (relation). В качестве неформального синонима термину «отношение» часто встречается слово «таблица». При использовании реляционной модели база данных представляет собой множество взаимосвязанных двумерных таблиц, каждая из которых содержит информацию об определенном типе сущности (ТРАНСПОРТ, ЛИЦО, УГОЛОВНОЕ ДЕЛО и т.д.)[1]. Каждая строка таблицы (запись, или кортеж) содержит данные об экземпляре сущности (например, лице, автомобиле, похищенной вещи), а столбцы таблицы содержат различные характеристики этих типов сущностей – атрибуты (например, ФАМИЛИЯ, ИМЯ, ОТЧЕСТВО, ДАТА РОЖДЕНИЯ, МЕСТО РОЖДЕНИЯ). Все атрибуты в таблице поименованы. Имя атрибута должно быть уникальным и позволять определять его (например, имя атрибута, характеризующего дату рождения, будет «birth_date»). Атрибут может быть обязательным и необязательным. Значение обязательного атрибута должно быть определено в момент внесения информации в базу данных. Если атрибут необязательный, то для таких случаев предусмотрено специальное значение – NULL, которое можно интерпретировать как «неизвестное значение» (рис. 1.1).



family	name	otch	birth_date
АЛЕКСАНДРОВ	ИГОРЬ	СЕРГЕЕВИЧ	11.02.1967
КАЛИНИН	АРТУР	ПЕТРОВИЧ	23.10.1984
ДУБРОВ	АНТОН	ИГОРЕВИЧ	(null)

Рис. 1.1. Фрагмент таблицы «Физические лица» (fl) в реляционной базе данных

Экземпляры сущностей одного типа обладают одинаковыми наборами атрибутов, но должны отличаться значениями хотя бы одного атрибута, для того чтобы быть идентифицируемыми. Подмножество атрибутов сущности, которые идентифицируют экземпляр сущности и не содержат меньшего подмножества атрибутов, идентифицирующих экземпляр сущности, называют *потенциальным ключом*. Все экземпляры сущности должны иметь хотя бы один потенциальный ключ. Экземпляр сущности может иметь несколько потенциальных ключей, в этом случае один из них

[1] Информационная система ЦОРИ ГУМВД России по Московской области насчитывает более 140 таблиц, содержащих информацию о типах сущностей и типах их связей.

может быть выбран в качестве *первичного ключа*, как правило, имеющего минимальный набор атрибутов или наименьший размер (физического хранения). Оставшиеся потенциальные ключи называют *альтернативными*. Первичный ключ может состоять из единственного атрибута, значения которого уникальны для каждого экземпляра сущности. Так, например, в базе данных не должно быть двух абонентских номеров сотовой связи с одинаковым сочетанием цифр, поэтому абонентский номер является первичным ключом. Такой первичный ключ называют *простым ключом*.

Если тип сущности не имеет атрибута, содержащего уникальное значение, то первичный ключ может быть составлен из нескольких атрибутов, совокупность значений которых гарантирует уникальность. Так, атрибуты ГОРОД, УЛИЦА, ДОМ, КВАРТИРА не могут быть по отдельности первичными ключами сущности АДРЕС, так как могут оказаться одинаковыми у двух и более адресов. Однако не должно быть двух адресов с одинаковым значением атрибутов ГОРОД, УЛИЦА, ДОМ, КВАРТИРА. Поэтому у сущности АДРЕС первичным ключом будет указанный выше набор атрибутов. Такой первичный ключ называется *составным ключом*.

Если несколько экземпляров сущностей одного типа не отличаются значениями атрибутов, то для их идентификации может быть добавлен еще один атрибут – порядковый или системный номер (id), который будет уникальным у каждого экземпляра сущности одного типа.

1.2. Связь между таблицами

Для связи отношений (таблиц) используются *внешние ключи*. Внешний (вторичный) ключ (Foreign key) – это атрибут отношения (подчиненной таблицы), который является копией первичного ключа (Primary key) другого отношения (главной таблицы) (рис. 1.2). Если связь является необязательной, то значение внешнего ключа может быть неопределенным (Null).

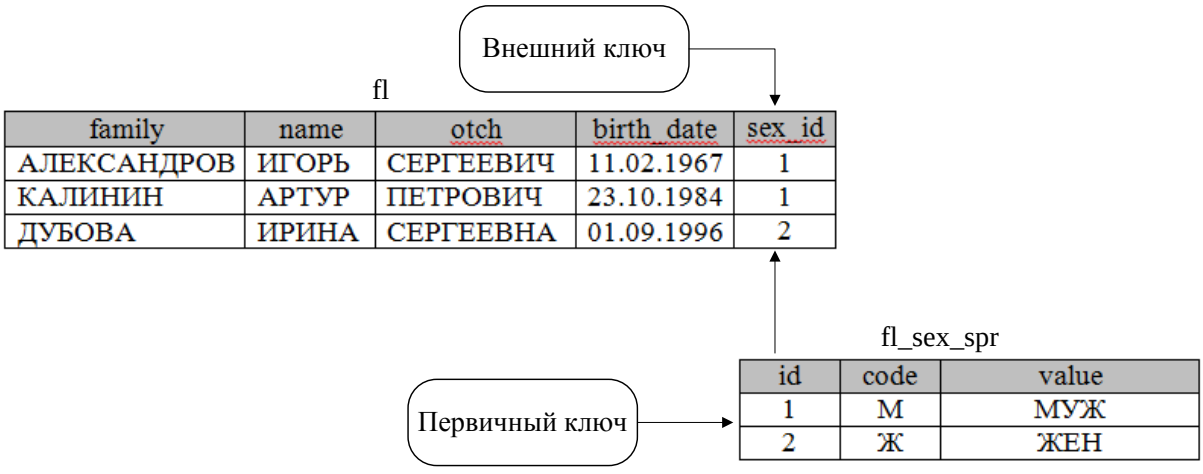


Рис. 1.2. Пример связи отношений «один-ко-многим»

Для создания связи отношений «многие ко многим» используется дополнительная таблица связей (links) (рис. 1.3).

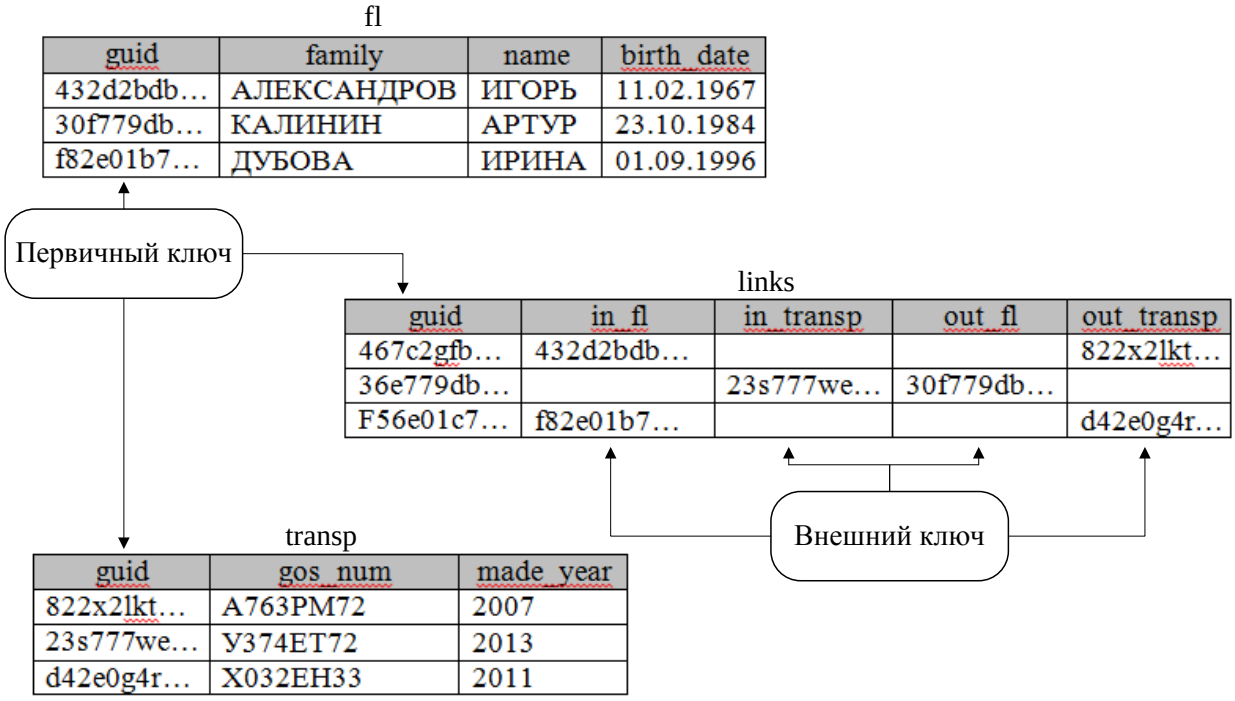


Рис. 1.3. Пример связи отношений «многие-ко-многим»

Таким образом, внешние ключи логически связывают экземпляры сущностей разных типов.

Вопросы для самоконтроля

1. Что составляет основу реляционных баз данных?
2. Что такое кортеж?
3. Определите обязательный атрибут.
4. Что такое потенциальный ключ?
5. Дайте понятие первичного ключа.
6. Что такое составной ключ?
7. Какое минимальное количество таблиц нужно для реализации связи вида «один-к-одному» для двух сущностей?
8. Какое минимальное количество таблиц нужно для реализации связи вида «один-ко-многим» для двух сущностей?
9. Какое минимальное количество таблиц нужно для реализации связи вида «многие-ко-многим» для двух сущностей?

2. SQL: возникновение и особенности

2.1. Возникновение SQL, разработка стандарта

В любой современной системе управления базами данных (далее – СУБД), поддерживающей реляционную модель, для обработки данных используется стандартизированный язык SQL, прообраз которого был создан в IBM (International Business Machines) в 1970-е годы для проверки возможностей модели данных Эдгара Кодда в рамках экспериментальной СУБД System R. Сначала язык назывался SEQUEL, но позднее был переименован в SQL.

Учитывая, что производители других реляционных СУБД разрабатывали свои варианты языка обработки данных, возникла необходимость стандартизации языка запросов в целях унификации программного обеспечения.

Первая версия стандарта языка была выпущена американским институтом стандартов (далее – ANSI) в 1986 году и через год одобрена международной организацией по стандартизации (ISO). В последующие годы в стандарт вносились изменения, и сегодня можно говорить о существовании семи вариантов стандарта, последний из которых SQL:2008

Однако, несмотря на разработку стандарта, различия в диалектах языка все еще остаются, поскольку в конкретной СУБД могут быть не реализованы или реализованы по-своему некоторые языковые средства, описанные в стандарте, не входящие в основную часть стандарта (Core). Кроме того, в диалектах языка могут присутствовать еще не стандартизированные возможности, т.к. производители, реагируя на потребности своих клиентов, обычно реализуют их гораздо быстрее, чем выходит следующая версия стандарта.

Целями SQL является определение типов данных, предоставление к ним доступа и их обработка. Особенностью SQL является то, что он ориентирован на работу с данными, представленными в виде взаимосвязанных таблиц. При этом при использовании SQL пользователю достаточно указать, с какими данными и из каких таблиц он хочет работать, не определяя процедуру обработки информации. СУБД самостоятельно установит местонахождение данных, индексы и даже наиболее эффективные последовательности операций для получения требуемых результатов.

2.2. Типы данных

В целях унификации работы с данными, хранящимися в столбцах таблиц реляционной базы данных, они были типизированы, т.е. отнесены к одному из типов данных, предопределяемых в SQL или определяемых пользователями путем применения соответствующих средств языка.

В SQL выделяют более 10 типов данных, однако в данном учебном пособии будут рассмотрены только базовые из них, используемые в СУБД «Oracle». При работе с другими СУБД следует учесть, что в них могут существовать свои типы данных.

Тип данных CHAR.

Тип данных CHAR обозначает, что атрибут состоит из символьных данных *фиксированной* длины. Длина значения атрибута задается в байтах от 1 до 255 (по умолчанию 1). Если вводимое значение короче фиксированной длины, оно дополняется необходимым количеством пробелов.

Тип данных VARCHAR2.

Тип данных VARCHAR2 обозначает, что атрибут состоит из символьных данных *переменной* длины. При создании таблицы со столбцом VARCHAR2 для него задается максимальная длина в байтах от 1 до 2000. Для хранения значений этого типа будет выделено столько байт, сколько нужно для записи конкретного значения, но не больше заданного числа *n,* которое указывается в скобках.

Следует помнить, что ограничения на длину значения атрибута закладывается в байтах. Таким образом, если набор символов базы данных использует однобайтовую схему кодирования символов, то число символов в столбеце совпадает с числом байтов. В мультибайтовой схеме кодирования[2] такого соответствия нет. Например, некоторые символы могут занимать один байт, другие – два байта, и т.д.

Тип данных LONG.

Тип данных LONG означает, что атрибут может состоять из символьных данных переменной длины до двух гигабайт.

Тип данных NUMBER.

Тип данных NUMBER используется для хранения нуля и положительных или отрицательных чисел с фиксированной точкой. Тип данных NUMBER характеризуется точностью и масштабом, которые указываются в скобках, через запятую. Точность – общее число цифр (может задаваться от 1 до 38); масштаб – число цифр справа от десятичной точки (может задаваться от –84 до 127). При указании отрицательного масштаба происходит округление указанного количество цифр слева от десятичной точки. Если точность не указана, столбец хранит значения так, как они задаются. Если не указан масштаб, он считается нулевым. В таблице 2.1 приведены примеры влияния показателя точности и масштаба на хранение типа данных NUMBER.

[2] Например, в некоторых языках с большим количеством символов (японском, китайском, корейском) или языках, где текст читается справа налево (арабский, иврит), для хранения одного символа требуется несколько байт.

Входные данные	Тип данных	Хранится как
654,321.89	NUMBER(*,1)	654321.9
654,321.89	NUMBER(8)	654321
654,321.89	NUMBER(8,2)	654321.89
654,321.89	NUMBER(8,1)	654321.9
654,321.89	NUMBER(5)	ошибка: превышена точность
654,321.89	NUMBER(6,-2)	654300

Табл. 2.1. Влияние показателя точности и масштаба на хранение числовых данных

Следует отметить, что приведенные типы данных схожи по своим свойствам, но не идентичны типам данных, используемым в стандарте ANSI. В таблице 2.2 приведено сравнение типов данных ANSI с типами данных ORACLE.

Тип данных ANSI SQL	**Тип данных ORACLE**
CHARACTER(n), CHAR(n)	CHAR(n)
NUMERIC(p,s), DECIMAL(p,s), DEC(p,s)	NUMBER(p,s)
INTEGER, INT, SMALLINT	NUMBER(38)
CHARACTER VARYING(n), CHAR VARYING(n)	VARCHAR2(n)

Табл. 2.2. Сравнение типов данных ANSI с типами данных ORACLE.

Тип данных DATE.

Тип данных DATE предназначен для хранения информации о дате и времени. Тип данных DATE запоминает век, год, месяц, день, часы, минуты и секунды. Данные дат в СУБД «Oracle» хранятся в фиксированных полях длиной семь байт, соответствующих веку, году, месяцу, дню, часу, минуте и секунде.

Тип данных BLOB.

Тип данных BLOB предназначен для хранения двоичных данных объемом до 4 ГБ.

Вопросы для самоконтроля

1. Назовите цель создания SQL.
2. Чем обусловлена необходимость стандартизации SQL.
3. Какие типы данных существуют в языке SQL?
4. Назовите отличия типа данных CHAR от VARCHAR2?
5. Как будет хранится в СУБД Oracle число 756.234 внесенное в поле с типом данных NUMBER (7,-3)?
6. Охарактеризуйте информацию, хранимую в поле с типом данных DATE?
7. Какой тип данных можно использовать для помещения в поле фотоизображения?

3. Создание, удаление и изменение таблиц

3.1. Создание таблиц

Таблицы создаются с помощью команды CREATE TABLE. Эта команда позволяет создать пустую таблицу – таблицу, не имеющую строк. При вводе команды CREATE TABLE необходимо определить имена таблицы и столбцов, входящих в ее состав. Кроме того, для каждого столбца указывается тип данных и его размер. Каждая таблица должна иметь, по крайней мере, один столбец.

При указании имени таблиц и столбцов (или каких-либо других объектов базы данных) нельзя использовать пробелы, поскольку они используются для разделения отдельных частей команд в SQL. Таким образом, для разделения слов в именах таблиц вместо пробелов часто используется символ подчеркивания (_).

Размер столбца зависит от типа данных и от той информации, которая планируется в нем размещаться. Если размер не указывается, то система установит значение автоматически.

Следующая команда позволяет создать таблицу lico_obmen:

```
CREATE TABLE lico_obmen
   (guid varchar2(64),
    family varchar2(250),
    name varchar2(250),
    otch varchar2(250));
```

Порядок столбцов в определении таблицы существенен, он определяет порядок, в котором задаются значения элементов строк.

3.2. *Создание ограничений в таблицах*

При создании таблица (или при изменении) дополнительно можно определить ограничения на значения, которые вводятся в поля, и SQL будет отвергать любое из них, если оно не соответствует определенному критерию. Существует два основных типа ограничения: ограничения на столбцы и на таблицу в целом. Разница между ними состоит в том, что ограничения на столбцы применимы только к отдельным столбцам, а ограничения на таблицу применимы к группам, состоящим из одного или нескольких столбцов.

Ограничения на столбец добавляются при определении столбца после указания типа данных и перед запятой. Ограничения на таблицу размещаются в конце определения таблицы, после определения последнего столбца, перед закрывающей круглой скобкой.

Например, часто используются так называемые обязательные для заполнения поля. Для создания такого поля необходимо запретить использование NULL значений в нем, использовав ключевое слово NOT NULL.

Наложим ограничение NOT NULL на столбец *guid* таблицы *lico_obmen*.

```
CREATE TABLE lico_obmen
    (guid varchar2(64) NOT NULL,
     family varchar2(250),
     name varchar2(250),
     otch varchar2(250));
```

Еще одним часто используемым ограничением является UNIQUE. Данное ограничение используется в том случае, когда нужно, чтобы все значения, введенные в столбец, отличались друг от друга. Например, в столбцах, используемых в качестве первичного ключа. Если при создании таблицы указывается ограничение UNIQUE для столбца, то будет отвергнута любая попытка ввести в поле этого столбца значение, уже содержащееся в другой строке. Команда по созданию ограничения UNIQUE на столбец *guid* таблицы *lico_obmen* выглядит следующим образом:

```
CREATE TABLE lico_obmen
    (guid varchar2(64) NOT NULL UNIQUE,
     family varchar2(250),
     name varchar2(250),
     otch varchar2(250));
```

Широкое применение при создании таблиц находит ограничение PRIMARY KEY, которое используется для создания первичного ключа. Ограничение PRIMARY KEY может быть применено к таблице или к множеству столбцов. По функциональным возможностям оно сходно с ограничением UNIQUE, за исключением того, что только один первичный ключ (состоящий из любого количества столбцов) может быть определен для одной таблицы. В первичном ключе недопустимо применение NULL-значений. Синтаксис команды создания таблицы *lico_obmen* с объявлением первичного ключа в поле *guid* выглядит следующим образом:

```
CREATE TABLE lico_obmen
    (guid varchar2(64) NOT NULL PRIMARY KEY,
     family varchar2(250),
     name varchar2(250),
     otch varchar2(250));
```

Если первичный ключ состоит из более чем одного поля, то можно применить ограничение PRIMARY KEY в качестве ограничения таблицы к нескольким столбцам.

```
CREATE TABLE lico_obmen
    (guid varchar2(64) NOT NULL,
     family varchar2(250),
     name varchar2(250),
     otch varchar2(250),
     CONSTRAINT      pk_lico_obmen
PRIMARY KEY (guid));
```

В целях обеспечения целостности в базе данных применяется ограничение FOREIGN KEYS. Внешний ключ означает, что значения из одной таблицы должны появиться в другой. Для создания FOREIGN KEYS необходимо наличие двух таблиц: первая, из которой заимствуются значения, называется parent table (родительской таблицей), вторая, содержащая FOREIGN KEY, значения в которую заимствуется, называется *child table* (дочерней таблицей). FOREIGN KEYS в дочерней таблице, как правило, ссылаются на PRIMARY KEY в родительской таблице. Ограничение FOREIGN KEYS определяется для таблицы. Например, в рассматриваемой нами базе данных таблица *fl_sex_spr* является родительской для таблицы *fl*, поскольку первичный ключ таблицы *fl_sex_spr* определенный для столбца *id*, должен повторятся во внешнем ключе таблицы *fl* определенном для столбца *sex_id*. Команда по созданию внешнего ключа для таблицы *lico_obmen* по столбцу *sex_id* будет выглядеть следующим образом:

```
CREATE TABLE lico_obmen
    (guid varchar2(64),
     family varchar2(250),
     name varchar2(250),
     otch varchar2(250),
     sex_id number,
CONSTRAINT fk_sex
   FOREIGN KEY (sex_id)
   REFERENCES fl_sex_spr (id));
```

В некоторых случаях при создании таблиц весьма эффективным представляется присвоение столбцу значений «по умолчанию». Это позволит существенно сократить время на ввод информации. Назначение значений по умолчанию DEFAULT также определяется как ограничение. Создадим в таблице *lico_obmen* столбец *create_date*, который по умолчанию будет заполняться текущей датой.

```
CREATE TABLE lico_obmen
    (guid varchar2(64) NOT NULL UNIQUE,
     family varchar2(250),
     name varchar2(250),
     otch varchar2(250),
     create_date date DEFAULT (sysdate));
```

Для предотвращения ввода в базу данных ошибочных сведений используется ограничение CHECK. Например, создадим в таблице *lico_obmen* столбец *birth_date*, в который, будет невозможно внести значения ранее 1895 года.

```
CREATE TABLE lico_obmen
    (guid varchar2(64) NOT NULL UNIQUE,
     family varchar2(250),
     name varchar2(250),
     otch varchar2(250),
     birth_date date CHECK
    (birth_date<to_date ('01.01.1895', 'dd.mm.yyyy')));
```

3.3. Создание индексов

Индекс (index) – это упорядоченный (в алфавитном или числовом порядке) список содержимого столбцов или группы столбцов в таблице. Таблицы могут иметь большое количество строк, и, поскольку строки задаются в любом произвольном порядке, поиск их по значению какого-либо из полей может занять достаточно много времени. Индексы предназначены для ускорения процесса извлечения информации из таблиц, за счет формирования из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы, что позволяет искать строки, удовлетворяющие критерию поиска. Ускорение работы с использованием индексов достигается в первую очередь за счет того, что индекс имеет структуру, оптимизированную для поиска.

Для оптимальной производительности запросов индексы обычно создаются на тех столбцах таблицы, которые часто используются в запросах. Для одной таблицы может быть создано несколько индексов. Однако, увеличение числа индексов замедляет операции добавления, обновления, удаления строк таблицы, поскольку при этом приходится обновлять сами индексы. Кроме того, индексы занимают дополнительный объем памяти, поэтому перед созданием индекса следует убедиться, что планируемый выигрыш в производительности запросов превысит дополнительную затрату ресурсов компьютера на сопровождение индекса.

Для создания индекса заранее должна быть создана таблица, содержащая столбцы, имена которых указаны в команде по созданию индекса. Имя индекса, определенное в команде, должно быть уникальным в базе данных, то есть оно не может использоваться для каких-либо других целей любым пользователем базы данных. Будучи однажды созданным, индекс является невидимым для пользователя.

Создадим индекс *in_family* для столбца *family* таблицы *fl*:

```
CREATE INDEX ind_fl_family ON fl (family);
```

При создании индекса на столбцы, на них может быть наложено ограничение UNIQUE. Создадим уникальный индекс на столбец *guid* таблицы *fl*:

```
CREATE UNIQUE INDEX ind_fl_guid ON fl (guid);
```

Эта команда будет отвергнута, если столбец *guid* уже имеет одинаковые значения. Уникальный индекс, создаваемый для нескольких столбцов, требует, чтобы комбинация значений во всех столбцах была уникальной.

Иногда возникает необходимость удалить индекс.

```
DROP INDEX ind_fl_guid;
```

Удаление индекса не изменяет содержимого поля (полей).

3.4. Удаление таблиц

Следует помнить, что перед удалением таблицы необходимо предварительно удалить из нее все данные, то есть сделать ее пустой. Таблица, имеющая строки, не может быть удалена. Для удаления таблицы *lico_obmen* необходимо написать следующую команду:

```
DROP TABLE lico_obmen;
```

После выполнения команды невозможно работать с объектом, имя которого было указано в команде DROP. Перед выполнением команды следует удостовериться, что эта таблица не содержит внешних ключей для какой-либо другой таблицы.

3.5. Внесение изменений в таблицы

Для внесения изменений в уже созданные таблицы применяется команда ALTER TABLE, которая позволяет переименовывать таблицы и добавлять, изменять или удалять столбцы в них.

Для добавления столбца *birth_year* в таблицу *lico_obmen* необходимо выполнить следующую команду:

```
ALTER TABLE lico_obmen
  ADD birth_year number;
```

При необходимости добавить несколько столбцов они указываются в разделе ADD через запятую.

Для изменения столбца используется оператор MODIFY.

```
ALTER TABLE lico_obmen
  MODIFY birth_year date;
```

При необходимости изменения нескольких столбцов они указываются в разделе MODIFY через запятую.

При необходимости удалить столбец из таблицы используется оператор DROP COLUMN.

```
ALTER TABLE lico_obmen
  DROP COLUMN birth_year;
```

Для переименования столбца используется оператор RENAME COLUMN … TO, где после RENAME COLUMN указывается старое название столбца, а после TO – новое. Переименуем столбец *otch* таблицы *lico_obmen* в *otchestvo*.

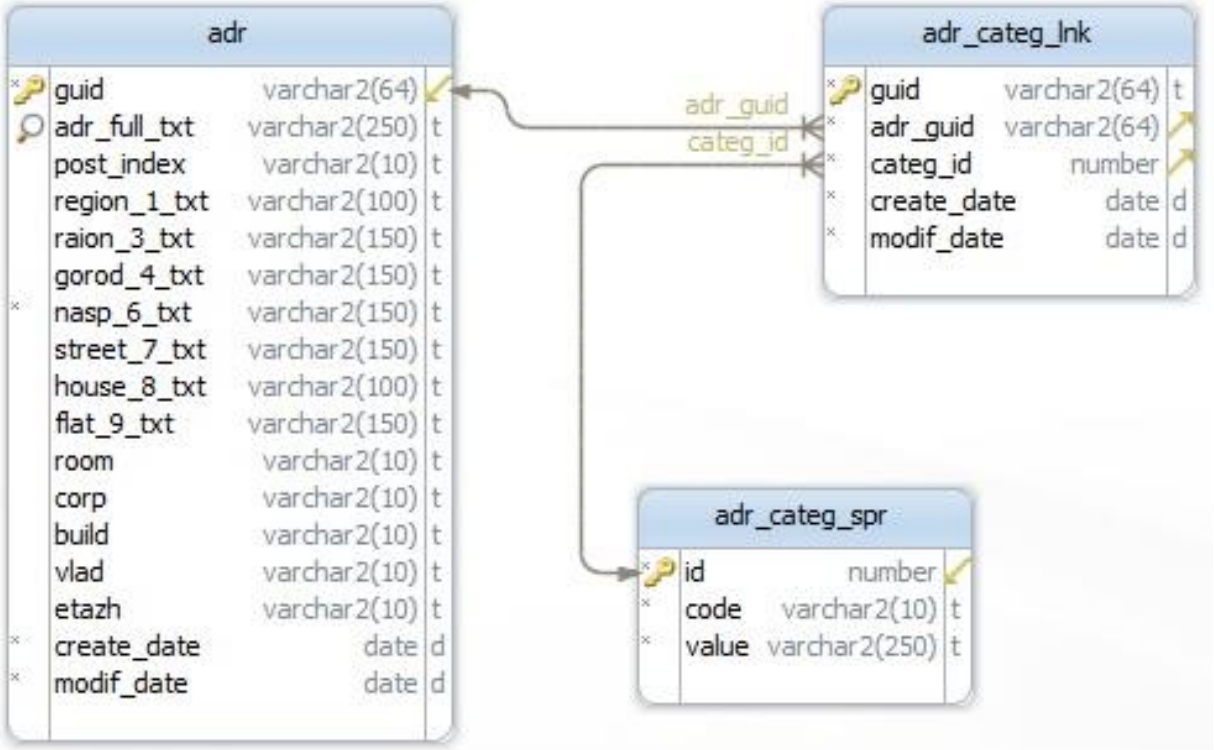
```
ALTER TABLE lico_obmen
  RENAME COLUMN otch TO otchestvo;
```

Для переименования всей таблицы используется оператор RENAME TO. Переименуем таблицу *lico_obmen* в *lico*.

```
ALTER TABLE lico_obmen
  RENAME TO lico;
```

Задание для самоконтроля

С помощью языка SQL-команд создайте следующие таблицы в СУБД «Oracle»:



4. Выборка данных из таблиц

4.1. Команда SELECT

Для выборки данных из таблиц используется команда SELECT, которую часто называют запросом, подразумевая запрос данных. Несмотря на кажущуюся простоту этой команды, ее можно расширять для выполнения очень сложных вычислений и поточной обработки данных.

Следует отметить, что SQL основан, в том числе, на реляционной алгебре, таким образом, в нем поддерживается аналогичное свойство замкнутости: в результате выполнения SELECT или любой другой операции с таблицами мы также получим таблицу.

Команда SELECT состоит из нескольких разделов, которые указываются в следующем порядке:

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

SELECT… FROM… [WHERE…] [GROUP BY…] [HAVING…] [ORDER BY…]

Раздел SELECT является обязательным, он определяет список столбцов таблицы-результата. SELECT может состоять из списка столбцов, разделенных запятой или из единственного символа «звездочка» (*), определяющего выбор всех столбцов.

Раздел FROM также является обязательным и задает список таблиц, из которых производится выборка.

После указания обязательных разделов указываются дополнительные.

В разделе WHERE указываются условия, которым должны соответствовать выбираемые строки.

Раздел GROUP BY применяется для разделения строк исходной таблицы или таблиц на группы.

Раздел HAVING работает аналогично разделу WHERE, но задает условие отбора не для одиночных записей, а для целых групп.

Раздел ORDER BY позволяет отсортировать строки таблицы-результата по заданному критерию.

Рассмотрим действие команды SELECT на примере. Получим содержимое столбцов *family*, *name*, *otch*, *birth_date* из таблицы *fl*, введя следующий запрос:

```
SELECT family, name, otch, birth_date
FROM fl;
```

Выборка данных по запросу показана на рис. 4.1. Результатом будут все данные из указанных столбцов таблицы *fl* и заголовки этих столбцов.

family	*name*	*otch*	*birth_date*
КАЛИНИН	АРТУР	ПЕТРОВИЧ	23.10.1984
ИВАНИЩЕВ	ГРИГОРИЙ		17.04.1995
ДУБРОВ	ЕВГЕНИЙ	ИГОРЕВИЧ	03.10.1986
ДУБРОВ	АНТОН	ИГОРЕВИЧ	01.09.1991
ГАРАНИН	ИВАН		14.01.1975
АЛЕКСАНДРОВ	ИГОРЬ	СЕРГЕЕВИЧ	11.02.1967
ДУБОВА	ИРИНА	СЕРГЕЕВНА	01.09.1996

Рис. 4.1. Выборка данных с помощью команды SELECT

Обзор каждой части получившего запроса:

SELECT – определяет, что представленная команда является запросом.

family, name, otch, birth_date – перечень столбцов, которые должны быть представлены в таблице-результате после выполнения запроса.

FROM – указывает на то, из какой таблицы осуществлять выборку. За ним следует пробел.

fl – имя таблицы, используемая как источник информации.

; – символ сообщает базе данных об окончании текста текущего запроса.

Запрос не упорядочивает и не сортирует выборку данных. Строки результата запроса выдаются в том виде, в котором они расположены в таблице. Порядок расположения записей при этом совершенно произволен.

Существует упрощенный вариант просмотра всех колонок таблицы. Для этого используется символ (*), заменяющий весь перечень столбцов.

```
SELECT *
FROM fl;
```

Результат выполнения запроса такой же, что и для указанного ранее, за исключением того, что в выборке будут присутствовать все столбцы таблицы *fl*.

4.2. Просмотр столбцов таблицы

Команда SELECT позволяет извлекать из таблицы указываемые пользователем столбцы. При этом в таблицу-результат можно не выводить столбцы, которые не нужны пользователю.

```
SELECT family, name, birth_date
FROM fl;
```

Например, по запросу, указанному выше, получаются выходные данные, включающие столбцы *family*, *name*, *birth_date*, и исключающие все остальные. Результат выборки представлен на рис. 4.2.

family	*name*	*birth_date*
КАЛИНИН	АРТУР	23.10.1984
ИВАНИЩЕВ	ГРИГОРИЙ	17.04.1995
ДУБРОВ	ЕВГЕНИЙ	03.10.1986
ДУБРОВ	АНТОН	01.09.1991
ГАРАНИН	ИВАН	14.01.1975
АЛЕКСАНДРОВ	ИГОРЬ	11.02.1967
ДУБОВА	ИРИНА	01.09.1996

Рис. 4.2. Выбор столбцов

4.3. Изменение порядка столбцов

Столбцы таблицы по умолчанию упорядочены, но их можно извлекать в любом удобном порядке. Оператор (*) извлекает столбцы в порядке по умолчанию, но если перечислить столбцы самостоятельно, они выводятся в том порядке, который был указан.

В таблице *fl* укажем следующий порядок столбцов: сначала «дата рождения» (*birth_date)*, за ним – столбец «фамилия» (*family)*, затем – «имя» (*name)* и «отчество» (*otch)*:

```
SELECT birth_date, family, name, otch
FROM fl;
```

Выборка данных представлена на рис. 4.3.

birth_date	*family*	*name*	*otch*
23.10.1984	КАЛИНИН	АРТУР	ПЕТРОВИЧ
17.04.1995	ИВАНИЩЕВ	ГРИГОРИЙ	
03.10.1986	ДУБРОВ	ЕВГЕНИЙ	ИГОРЕВИЧ
01.09.1991	ДУБРОВ	АНТОН	ИГОРЕВИЧ
14.01.1975	ГАРАНИН	ИВАН	
11.02.1967	АЛЕКСАНДРОВ	ИГОРЬ	СЕРГЕЕВИЧ
01.09.1996	ДУБОВА	ИРИНА	СЕРГЕЕВНА

Рис. 4.3. Упорядоченные столбцы

4.4. Устранение повторяющихся значений

В предыдущих разделах мы рассмотрели возможность получения информации из определенных столбцов таблицы. Так, если нам необходимо узнать, какие страны встречаются в таблице *fl*, необходимо ввести следующий запрос:

```
SELECT birth_country_txt
FROM fl;
```

Аналогичный результат даст использование оператора ALL:

```
SELECT ALL birth_country_txt
FROM fl;
```

Получится выборка, представленная на рис. 4.4.

birth_country_txt
РОССИЯ
УКРАИНА
ГРУЗИЯ
РОССИЯ
РОССИЯ
АРМЕНИЯ
РОССИЯ

Рис. 4.4. Общий SELECT

В целях исключения повторных значений из таблицы-результата используется оператор DISTINCT.

Для получения такого же списка, но без повторений, нужно изменить запрос, внеся оператор DISTINCT:

```
SELECT DISTINCT birth_country_txt
FROM fl;
```

Получится выборка, представленная на рис. 4.5.

birth_country_txt
РОССИЯ
УКРАИНА
ГРУЗИЯ
АРМЕНИЯ

Рис. 4.5. SELECT без дублей

Оператор DISTINCT не игнорирует пустые значения. Таким образом, при использовании DISTINCT таблица-результат будет содержать NULL в качестве отдельного значения.

4.5. Уточнение выборки

Для определения условий, которым должны соответствовать выбираемые строки в таблице-результате используется оператор WHERE.

Например, если необходимо узнать даты рождения, фамилии, имена и отчества на всех лиц с фамилией «Дубров» из таблицы (fl), то необходимо ввести следующий запрос:

```
SELECT family, name, otch, birth_date
FROM fl
WHERE family = 'ДУБРОВ';
```

Получится выборка, представленная на рис. 4.6.

family	name	otch	birth_date
ДУБРОВ	АНТОН	ИГОРЕВИЧ	01.09.1991
ДУБРОВ	ЕВГЕНИЙ	ИГОРЕВИЧ	03.10.1986

Рис. 4.6. SELECT с оператором WHERE

Следует обратить внимание, что столбец, используемый оператором WHERE, может не входить в таблицу-результат. Числовые поля используются в работе оператора WHERE с некоторыми отличиями от символьных, а именно, при указании значения поля не нужно заключать значение в одиночные кавычки. Если необходимо узнать государственные номера автомобилей, произведенных в 2015 году, необходимо ввести следующий запрос:

```
SELECT gos_num
FROM transp
WHERE made_year = 2015;
```

Получится выборка, представленная на рис. 4.7.

gos_num
M001MM72

Рис. 4.7. Использование оператора WHERE с числовым типом данных

Задания для самоконтроля

1. Проведите выборку всех данных из таблицы *fl*.

2. Выберите из таблицы *fl* лиц с фамилией Калинин.

3. Выведите Фамилию, Имя, Отчество лиц с фамилией Гаранин из таблицы *fl*.

4. Выясните vin-номера автомобилей 2003 года выпуска из таблицы *transp*.

5. Применение реляционных и булевых операторов

5.1. Реляционные операторы

Реляционные операторы – это математические символы, которые задают определенное сравнение двух значений, применимые к числовым величинам. SQL определяет и работает со следующими операторами сравнения: равно (=), больше чем (>), меньше чем (<), больше или равно (>=), меньше или равно (<=), неравно (<> или !=). Если необходимо найти vin транспортных средств (*transp*), произведенных после 2015 года, следует использовать сравнение «больше», применимое к столбцу «год выпуска» (*made_year*).

```
SELECT vin
FROM transp
WHERE made_year > 2015;
```

Получится выборка, представленная на рис. 5.1.

vin
ZFA18800004473122

Рис. 5.1. Использование оператора (>)

5.2. Булевы операторы

SQL может обрабатывать основные булевы операторы. Булевы выражения – это те выражения, в отношении которых можно применить два состояния: «истина» или «ложь». Булевы операторы объединяют одно или несколько состояний «истина/ложь» и получают общее значение «истина/ложь». Стандартные операторы данной категории – это AND, OR, NOT (таблица 5.1).

Оператор	Принцип действия
AND	Выдает истину, если оба элемента сравнения истинны
OR	Выдает истину, если хотя бы один элемент сравнения истинен
NOT	Применим к одному элементу, меняет его статус на диаметрально противоположный: «истина» – «ложь»

Табл. 5.1. Булевы операторы

Если необходимо вычислить лиц *(fl)* с фамилией «ДУБРОВ» моложе 1987 года рождения, то необходимо ввести следующий запрос:

```
SELECT family, name, otch, birth_year, birth_date
FROM fl
WHERE birth_year> 1987
AND family = 'ДУБРОВ';
```

Получится выборка, представленная на рис. 5.2.

family	name	otch	birth_year	birth_date
ДУБРОВ	АНТОН	ИГОРЕВИЧ	1991	01.09.1991

Рис. 5.2. Использование оператора AND

При использовании оператора OR в выборку будут включены сведения обо всех лицах, которые либо моложе 1980 года, либо с фамилией «ДУБРОВ».

```
SELECT family, name, otch, birth_year, birth_date
FROM fl
WHERE birth_year> 1987
OR family = 'ДУБРОВ';
```

Получится выборка, представленная на рис. 5.3.

family	name	otch	birth_year	birth_date
ИВАНИЩЕВ	ГРИГОРИЙ		1995	17.04.1995
ДУБРОВ	АНТОН	ИГОРЕВИЧ	1991	01.09.1991
ДУБРОВ	ЕВГЕНИЙ	ИГОРЕВИЧ	1986	03.10.1986
ДУБОВА	ИРИНА	СЕРГЕЕВНА	1996	01.09.1996

Рис. 5.3. Использование оператора OR

Использование оператора NOT позволяет получить противоположное значение выражения. Пример запроса с использованием NOT:

```
SELECT family, name, otch, birth_year, birth_date
FROM fl
WHERE birth_year> 1987
AND NOT family = 'ДУБРОВ';
```

Получится выборка, представленная на рис. 5.4.

family	name	otch	birth_year	birth_date
ИВАНИЩЕВ	ГРИГОРИЙ		1995	17.04.1995
ДУБОВА	ИРИНА	СЕРГЕЕВНА	1996	01.09.1996

Рис. 5.4. Использование оператора NOT

Были выбраны все лица не по фамилии Дубров моложе 1987 года рождения. Иванищев и Дубова моложе 1987 года, и их фамилия не Дубров, значит, они соответствуют обоим условиям. Оператор NOT должен предшествовать значению, которое он должен изменить, и не может располагаться перед сравнением. Некорректно вводить «*…AND family NOT = 'ДУБРОВ';»*. Запрос выдаст ошибку обработки.

Задания для самоконтроля

1. Используя реляционные операторы, выберите всех лиц из таблицы *fl*, кроме лиц с фамилией Александров.
2. Используя булевы операторы, выберите всех лиц из таблицы *fl* с фамилиями Александров и Гаранин.
3. Выберите автомобили до 2012 года выпуска из таблицы *transp*, используя реляционные операторы.
4. Выберите автомобили, выпущенные до 2004 года и после 2014 года, из таблицы *transp*, используя булевы и реляционные операторы.

6. Использование специальных операторов

6.1. Оператор IN

Оператор IN определяет множество значений, в которое искомое значение может входить или не входить. Если необходимо вычислить лиц (fl), фамилии (family) которых «ДУБРОВ» и «ИВАНИЩЕВ», то можно ввести следующий запрос:

```
SELECT family, name, otch, birth_date
FROM fl
WHERE family = 'ДУБРОВ'
OR family ='ИВАНИЩЕВ';
```

Но есть более простой способ извлечь эту информацию:

```
SELECT family, name, otch, birth_date
FROM fl
WHERE family IN ('ДУБРОВ', 'ИВАНИЩЕВ');
```

Выборка, полученная и в первом и во втором случае, будет идентична. Результат представлен на рис. 6.1.

family	name	otch	birth_date
ИВАНИЩЕВ	ГРИГОРИЙ		17.04.1995
ДУБРОВ	АНТОН	ИГОРЕВИЧ	01.09.1991
ДУБРОВ	ЕВГЕНИЙ	ИГОРЕВИЧ	03.10.1986

Рис. 6.1. Использование оператора IN

Оператор IN определяет множество, каждое значение которого перечисляется в круглых скобках, и если значений несколько, то значения разделяются запятой. Если в поле, которое находится слева от оператора IN в запросе, есть одно из перечисленных значений, то в таблице-результате будут представлены все строки, соответствующие условию. Одиночные кавычки слева и справа от значения не нужны, если элементы имеют числовой тип. Можно найти всех лиц 1967 и 1984 годов рождения.

```
SELECT family, name, otch, birth_date
FROM fl
WHERE birth_year IN (1967, 1984);
```

Получится выборка, представленная на рис. 6.2.

family	name	otch	birth_date
АЛЕКСАНДРОВ	ИГОРЬ	СЕРГЕЕВИЧ	11.02.1967
КАЛИНИН	АРТУР	ПЕТРОВИЧ	23.10.1984

Рис. 6.2. Использование оператора IN с числовым типом данных

6.2. Оператор BETWEEN

Оператор BETWEEN используется с такими типами данных, как DATE и NUMBER. BETWEEN задает границы максимального и минимального значения, в которые должно попасть искомое значение.

Используется оператор BETWEEN вместе с ключевым словом AND, которое разделяет два крайних значения указываемого диапазона. Если необходимо вычислить лиц (fl), год рождения (birth_year) которых находится между 1992 и 1995 годами, то нужно ввести следующий запрос:

```
SELECT family, name, otch, birth_date
FROM fl
WHERE birth_date BETWEEN
TO_DATE ('01.01.1995', 'dd.mm.yyyy')
AND TO_DATE ('31.12.2012', 'dd.mm.yyyy');
```

Получится выборка, представленная на рис. 6.3.

family	name	otch	birth_date
ИВАНИЩЕВ	ГРИГОРИЙ		17.04.1995
ДУБОВА	ИРИНА	СЕРГЕЕВНА	01.09.1996

Рис. 6.3. Использование оператора BETWEEN

6.3. Оператор LIKE

Оператор LIKE используется с такими типами данных как VARCHAR2 и CHAR. Оператор LIKE позволяет найти значение, если оно полностью не известно пользователю. Для поиска используются специальные символы, которыми заменяют неизвестные символы. Существует два вида специальных символов, используемых с оператором LIKE:

– «подчеркивание» (_) заменяет один символ. Например, образцу «_АЛИНИН» будут соответствовать «КАЛИНИН» или «МАЛИНИН», но не будет соответствовать «ТААЛИНИН». Если необходимо вычислить лиц (fl), фамилия (family) которых «КАЛИНИН», но неизвестно точно, «КАЛИНИН» или «КАЛЕНИН» то нужно ввести следующий запрос:

```
SELECT family, name, otch, birth_date
FROM fl
WHERE family LIKE 'КАЛ_НИН';
```

Получится выборка, представленная на рис. 6.4.

family	name	otch	birth_date
КАЛИНИН	АРТУР	ПЕТРОВИЧ	23.10.1984

Рис. 6.4. Использование оператора LIKE со специальным символом (_)

– «процент» (%) заменяет последовательность символов неопределенной длины (нулевой в том числе). Например, образцу «К%Н» будут соответствовать «КАЛИНИН», «КОНАРИН», «КЕВРАН», но не «КАЛИНИНА». Если необходимо вычислить лиц (fl), фамилия (family) которых начинается на «А», то нужно ввести следующий запрос:

```
SELECT family, name, otch, birth_date
FROM fl
WHERE family LIKE 'А%';
```

Получится выборка, представленная на рис. 6.5.

family	name	otch	birth_date
АЛЕКСАНДРОВ	ИГОРЬ	СЕРГЕЕВИЧ	11.02.1967

Рис. 6.5. Использование оператора LIKE со специальным символом (%)

Оператор LIKE весьма полезен при поиске фамилии или других данных, правильное или полное значение которых неизвестно. Обратите внимание, что специальный символ (_) заменяет один символ, и чтобы заменить им несколько символов, необходимо использовать его во всех местах пропуска. Специальный символ (%) заменяет неограниченное количество символов, поэтому нет необходимости использовать его дважды в одном месте пропуска.

6.4. Условие IS

Зачастую в таблицах встречаются записи с пустыми значениями полей, потому что значения поля просто нет. SQL указывает в таких полях значение NULL. Значение NULL не может быть адекватно обработано таким традиционным оператором сравнения, как равенство (=). Для работы с NULL используется особый оператор IS. Например, если необходимо вы-

числить лиц (fl), отчество которых неизвестно и поле *otch*, соответственно, не заполнено (NULL), то нужно ввести следующий запрос:

```
SELECT family, name, otch, birth_date
FROM fl
WHERE otch IS NULL;
```

Получится выборка, представленная на рис. 6.6.

family	name	otch	birth_date
ГАРАНИН	ИВАН		14.01.1975
ИВАНИЩЕВ	ГРИГОРИЙ		17.04.1995

Рис. 6.6. Использование условия IS NULL

6.5. Оператор NOT и специальные операторы

Специальные операторы могут использоваться с булевым оператором NOT. Если необходимо исключить NULL-значения из выборки, то оператор NOT используется для того, чтобы условие приняло диаметрально противоположное значение:

```
SELECT family, name, otch
FROM fl
WHERE birth_region_txt IS NOT NULL;
```

Получится выборка, представленная на рис. 6.7.

family	name	otch
КАЛИНИН	АРТУР	ПЕТРОВИЧ
ДУБРОВ	АНТОН	ИГОРЕВИЧ
ГАРАНИН	ИВАН	
ДУБОВА	ИРИНА	СЕРГЕЕВНА

Рис. 6.7. Использование операторов NOT и NULL

Допустимо и такое использование оператора:

```
SELECT family, name, otch, birth_date
FROM fl
WHERE NOT family IS NULL;
```

Оператор NOT можно также использовать с оператором IN:

```
SELECT family, name, otch, birth_date
FROM fl
WHERE family NOT IN ('КАЛИНИН', 'ИВАНИЩЕВ',
'ДУБРОВ');
```

Получится выборка, представленная на рис. 6.8.

family	name	otch	birth_date
ГАРАНИН	ИВАН		14.01.1975
АЛЕКСАНДРОВ	ИГОРЬ	СЕРГЕЕВИЧ	11.02.1967
ДУБОВА	ИРИНА	СЕРГЕЕВНА	01.09.1996

Рис. 6.8. Использование операторов NOT и IN

Данный запрос будет эквивалентен следующему запросу:

```
SELECT family, name, otch, birth_date
FROM fl
WHERE NOT family IN ('КАЛИНИН', 'ИВАНИЩЕВ',
'ДУБРОВ');
```

Аналогичным образом можно использовать NOT LIKE и NOT BETWEEN.

Задания для самоконтроля

1. Используя специальные операторы, выберите всех лиц из таблицы *fl*, родившихся в промежутке между 1985 и 1994 годами.
2. Используя специальные операторы, выберите всех лиц из таблицы *fl*, фамилия которых начинается на букву «Д».
3. Используя специальные операторы, выберите всех лиц из таблицы *fl*, у которых отсутствует ИНН.
4. Используя специальные операторы, выберите всех лиц из таблицы *fl*, в ИНН которых встречаются цифры «66».
5. Используя специальные операторы, найдите автомобили, в государственном номере которых встречаются буквы «ОТ».

7. Функции агрегирования

7.1. Функции агрегирования и их назначение

Функциями агрегирования называются функции, которые позволяют определить количество записей в таблице, количество значений в столбце таблицы, найти минимальное, максимальное и среднее значение для столбца таблицы, вычислить сумму данных для столбца. Таким образом, агрегирующие функции обеспечивают получение некоторой обобщенной информации, которая может быть помещена в одно поле таблицы-результата. Функции агрегирования представлены в таблице 7.1.

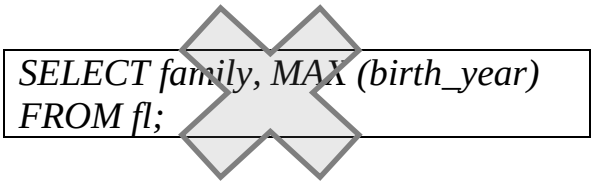
Функция	Описание
COUNT	определение количества строк или определенных значений
MAX	вычисление наибольшего значения без учета неопределенных значений
MIN	вычисление наименьшего значения без учета неопределенных значений
AVG	вычисление среднего из определенных значений
SUM	вычисление арифметической суммы определенных значений

Табл. 7.1. Функции агрегирования

7.2. Использование функций агрегирования

Функции агрегирования указываются перед именем столбцов в запросе SELECT. AVG и SUM используют только цифровые значения. COUNT, MAX и MIN – используют цифровые и символьные значения. Эти операции существенно отличаются от обычной выборки тем, что при применении функций агрегирования результат обработки будет содержать только одно значение. Именно поэтому нельзя запрашивать информацию из нескольких столбцов, если к ним применяются функции агрегирования.

Пример ошибочного применения функции агрегирования:



Поля и функции агрегирования не могут выбираться запросом одновременно без дополнительных условий. Синтаксически правильно будет использовать следующий пример.

```
SELECT family, birth_year
FROM fl
WHERE birth_year in
    (SELECT MAX (birth_year)
    FROM fl);
```

7.3. Оператор COUNT

Функцией оператора COUNT является подсчет количества значений в определенном столбце или количества строк в таблице. Когда подсчитываются значения по столбцу, в операторе применяется DISTINCT для подсчета числа различных значений определенного поля без повторов. Можно использовать его, например, для подсчета количества лиц, имеющих в пользовании определенные автомобили, или количество лиц, проживавших по определенному адресу. Если нам нужно узнать количество записей в таблице fl на данный момент, нужно ввести следующий запрос:

```
SELECT COUNT (guid)
FROM fl;
```

Получится выборка, представленная на рис. 7.1.

COUNT(guid)
7

Рис. 7.1 Использование оператора COUNT

Для того чтобы подсчитать общее количество строк в таблице, необходимо использовать оператор COUNT (*) вместо имени поля:

```
SELECT COUNT (*)
FROM fl;
```

Оператор COUNT (*) включает и NULL-значения, и повторяющиеся значения. Оператор DISTINCT в данном случае не применяется. В результате получается число, превышающее результат обработки оператора COUNT для отдельных полей, в котором исключены повторяющиеся строки и NULL-значения.

Функции агрегирования могут также работать с аргументом ALL, который необходимо разместить перед именем поля. Использование аргумента ALL включит все дубликаты в результат обработки. Главное различие между аргументами ALL и * при использовании оператора COUNT заключается в том, что ALL не считает поля с NULL содержимым.

По аналогии с оператором COUNT используются SUM, AVG, MAX, MIN.

Задания для самоконтроля

1. Используя функции агрегирования, посчитайте количество автомобилей из таблицы (transp), выпущенных в 2003 году.
2. Используя функции агрегирования, выберите самый старый автомобиль из таблицы (transp).
3. Используя функции агрегирования, посчитайте количество лиц, внесенных в базу в марте 2017 года.
4. Используя функции агрегирования, посчитайте количество лиц, информация об ИНН которых внесена в базу данных.

8. Форматирование результатов запросов

8.1. Переименование столбцов

SQL предоставляет пользователю возможность переименовать столбец в таблице-результате. Например, для составления отчета мы решили получить таблицу, в которой имена столбцов будут написаны по-русски.

```
SELECT family AS Фамилия, name AS Имя, otch AS
Отчество FROM fl;
```

Получится таблица-результат, представленная на рис. 8.1.

Фамилия	***Имя***	***Отчество***
КАЛИНИН	АРТУР	ПЕТРОВИЧ
ИВАНИЩЕВ	ГРИГОРИЙ	
ДУБРОВ	ЕВГЕНИЙ	ИГОРЕВИЧ
ДУБРОВ	АНТОН	ИГОРЕВИЧ
ГАРАНИН	ИВАН	
АЛЕКСАНДРОВ	ИГОРЬ	СЕРГЕЕВИЧ
ДУБОВА	ИРИНА	СЕРГЕЕВНА

Рис. 8.1. Переименование столбцов

8.2. Объединение столбцов

Слияния выходных данных двух столбцов в единое целое, называется конкатенацией и обозначается символами (||). Например, объединим фамилию, имя и отчество в один столбец.

```
SELECT family ||' '|| name ||' '|| otch as ФИО FROM fl;
```

Результатом запроса будет таблица-результат, представленная на рис. 8.2.

ФИО
КАЛИНИН АРТУР ПЕТРОВИЧ
ИВАНИЩЕВ ГРИГОРИЙ
ДУБРОВ ЕВГЕНИЙ ИГОРЕВИЧ
ДУБРОВ АНТОН ИГОРЕВИЧ
ГАРАНИН ИВАН
АЛЕКСАНДРОВ ИГОРЬ СЕРГЕЕВИЧ
ДУБОВА ИРИНА СЕРГЕЕВНА

Рис. 8.2. Объединение столбцов

8.3. Добавление текста в выборку

Для удобства представления выводимой информации иногда возникает потребность вставить в таблицу-результат какой-либо текст, что существенно упрощает задачу по формированиям отчетов, сводя к минимуму механический труд по оформлению выходных данных. Например, добавим в таблицу (fl) выходной столбец «года рождения»:

```
SELECT family ||' '|| name ||' '|| otch ||' '|| birth_year ||' '||
'года рождения' as Информация
FROM fl;
```

Получится выборка, представленная на рис. 8.3.

Информация
КАЛИНИН АРТУР ПЕТРОВИЧ 1984 года рождения
ИВАНИЩЕВ ГРИГОРИЙ 1995 года рождения
ДУБРОВ ЕВГЕНИЙ ИГОРЕВИЧ 1986 года рождения
ДУБРОВ АНТОН ИГОРЕВИЧ 1991 года рождения
ГАРАНИН ИВАН 1975 года рождения
АЛЕКСАНДРОВ ИГОРЬ СЕРГЕЕВИЧ 1967 года рождения
ДУБОВА ИРИНА СЕРГЕЕВНА 1996 года рождения

Рис. 8.3. Добавление текста в выборку

8.4. Сортировка выводимой информации

Записи в таблицах при представлении не отсортированы по умолчанию и выводятся в порядке их размещения в базе. Для организации пользовательских сортировок выводимой информации применяется оператор ORDER BY, который упорядочивает информацию по значениям одного или нескольких столбцов. При применении оператора ORDER BY можно задать последовательность по возрастанию (ASC) или по убыванию (DESC) для каждого столбцов. По умолчанию установлена последовательность по возрастанию значений, т.е. запросы «…ORDER BY family» и «…ORDER BY family ASC» будут выдавать идентичные результаты в выборках. Запрос на выборку лиц (fl), отсортированных по фамилии, в порядке обратном алфавитному, выглядят так:

```
SELECT family, name, otch, birth_date, birth_year
FROM fl
ORDER BY family DESC;
```

Получится выборка, представленная на рис. 8.4.

family	name	otch	birth_date	birth_year
КАЛИНИН	АРТУР	ПЕТРОВИЧ	23.10.1984	1984
ИВАНИЩЕВ	ГРИГОРИЙ		17.04.1995	1995
ДУБРОВ	ЕВГЕНИЙ	ИГОРЕВИЧ	03.10.1986	1986
ДУБРОВ	АНТОН	ИГОРЕВИЧ	01.09.1991	1991
ДУБОВА	ИРИНА	СЕРГЕЕВНА	01.09.1996	1996
ГАРАНИН	ИВАН		14.01.1975	1975
АЛЕКСАНДРОВ	ИГОРЬ	СЕРГЕЕВИЧ	11.02.1967	1967

Рис. 8.4. Сортировка выборки по убыванию поля «Фамилия» (family)

8.5. Сортировка выборки по множеству столбцов

Сортировки можно провести одновременно по нескольким столбцам, например, не только по фамилии (family), но и по имени (name):

```
SELECT family, name, otch, birth_date
FROM fl
ORDER BY family DESC, name DESC;
```

Получится выборка, представленная на рис. 8.5.

family	name	otch	birth_date
КАЛИНИН	АРТУР	ПЕТРОВИЧ	23.10.1984
ИВАНИЩЕВ	ГРИГОРИЙ		17.04.1995
ДУБРОВ	ЕВГЕНИЙ	ИГОРЕВИЧ	03.10.1986
ДУБРОВ	АНТОН	ИГОРЕВИЧ	01.09.1991
ДУБОВА	ИРИНА	СЕРГЕЕВНА	01.09.1996
ГАРАНИН	ИВАН		14.01.1975
АЛЕКСАНДРОВ	ИГОРЬ	СЕРГЕЕВИЧ	11.02.1967

Рис. 8.5. Сортировка выборки по нескольким полям

Если сортируемое поле отсутствует в выборке, оператор ORDER BY не сможет его упорядочить. Поэтому все поля, к которым будет применена сортировка, должны быть включены в выборку.

Если в столбце, который подлежит упорядочиванию, присутствуют NULL-значения, то они находятся или в конце, или перед значениями этого поля – это зависит от используемой СУБД.

Задания для самоконтроля

1. Используя сортировку в порядке, обратном алфавитному, выберите лиц из таблицы *fl*, фамилии которых начинаются на буквы «К», «И», «Д».

2. Используя символы конкатенации, сделайте выборку автомобилей из таблицы *transp* по столбцам «государственный номер» и «vin». Строка в таблице-результате должна выглядеть следующим образом «Автомобиль с государственным номером Е584ОТ72 имеет vin-номер KL1UF756E6B19 5928».

9. Выборки из нескольких таблиц

9.1. Особенности выборки из нескольких таблиц

SQL позволяет соединять несколько таблиц для выборки информации из них. С помощью соединений связывается информация, которая содержится в разных таблицах. При создании запроса на выборку данных из нескольких таблиц имена таблиц указываются после оператора FROM. Запрос при этом может быть адресован на любой столбец любой таблицы.

Следует подчеркнуть, что ранее при работе с командой SELECT, определяя список столбцов таблицы-результата мы не указывали имена таблиц. Это можно было не делать, поскольку запросы извлекали информацию только из одной таблицы. При извлечении информации из нескольких таблиц есть вероятность, что они могут содержать одинаковые по названию столбцы (например, guid в таблицах transp и fl.). Чтобы не допустить ошибки при формировании выборки из нескольких таблиц, необходимо всегда указывать не только имя столбца, но и таблицы.

Таким образом, имя столбца должно состоять из имени таблицы и имени столбца, которые разделяются точкой. Например:

– fl.family (Фамилия из таблицы «Физические лица»);
– fl.name (Имя из таблицы «Физические лица»);
– transp.gos_num (Государственный номер из таблицы «Транспорт»).

В SQL для проведения выборки из нескольких таблиц используется оператор JOIN. Существует 4 различных соединения (JOIN):

– INNER JOIN (простое соединение);
– LEFT OUTER JOIN (LEFT JOIN);
– RIGHT OUTER JOIN (RIGHT JOIN);
– FULL OUTER JOIN (FULL JOIN).

9.2. Оператор INNER JOIN

INNER JOIN – Наиболее часто применяемый оператор, он позволяет получить из нескольких таблиц только те строки, в которых выполняются условия соединения. Например, для того, чтобы увидеть пол каждого лица с непустым значением гендерной принадлежности, нужно ввести запрос:

```
SELECT fl.family, fl.name, fl.otch, fl.birth_date, fl_sex_spr.value
FROM fl
INNER JOIN fl_sex_spr
ON fl.sex_id = fl_sex_spr.id;
```

Получится выборка, представленная на рис. 9.1.

fl.family	fl.name	fl.otch	fl.birth_date	fl_sex_spr.value
КАЛИНИН	АРТУР	ПЕТРОВИЧ	23.10.1984	МУЖЧИНА
ИВАНИЩЕВ	ГРИГОРИЙ		17.04.1995	МУЖЧИНА
ДУБРОВ	ЕВГЕНИЙ	ИГОРЕВИЧ	03.10.1986	МУЖЧИНА
ГАРАНИН	ИВАН		14.01.1975	МУЖЧИНА
АЛЕКСАНДРОВ	ИГОРЬ	СЕРГЕЕВИЧ	11.02.1967	МУЖЧИНА
ДУБОВА	ИРИНА	СЕРГЕЕВНА	01.09.1996	ЖЕНЩИНА

Рис. 9.1. Объединение двух таблиц с помощью оператора INNER JOIN

В предыдущем запросе SQL выбирает строки из таблицы *fl* и таблицы *fl_sex_spr*, в которых значение поля *sex_id* таблицы *fl* соответствует значению поля *id* таблицы *fl_sex_spr*, и объединяет их.

Визуально можно представить это следующим образом (рис. 8.2.).

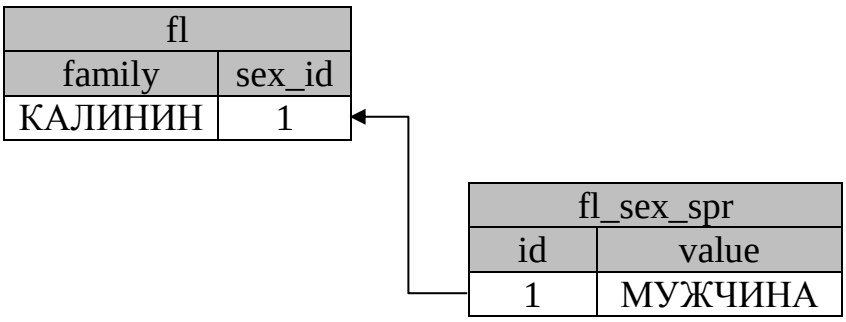


Рис. 9.2. Соединение таблиц

Схема соединения таблиц с помощью оператора INNER JOIN показана на рис. 9.3.

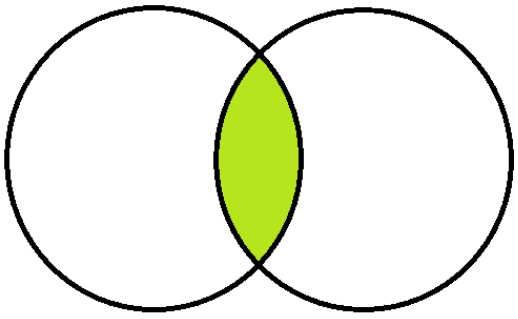


Рис. 9.3. INNER JOIN

Аналогичный результат можно получить и несколько другим способом.

```
SELECT fl.family, fl.name, fl.otch, fl.birth_date, fl_sex_spr.value
FROM fl, fl_sex_spr
WHERE fl.sex_id = fl_sex_spr.id;
```

9.3. Оператор LEFT OUTER JOIN

Оператор LEFT OUTER JOIN используется, когда нам надо получить информацию о всех строках из первой (левой) таблицы и только тех строках второй (правой) таблицы, в которых выполняются условия соединения. Например, для того чтобы выбрать всех лиц и узнать их гендерную принадлежность, нужно ввести запрос:

```
SELECT fl.family, fl.name, fl.otch, fl.birth_date, fl_sex_spr.value
FROM fl
LEFT OUTER JOIN fl_sex_spr
ON fl.sex_id = fl_sex_spr.id;
```

Получится выборка, представленная на рис. 9.4.

fl.family	fl.name	fl.otch	fl.birth_date	fl_sex_spr.value
КАЛИНИН	АРТУР	ПЕТРОВИЧ	23.10.1984	МУЖЧИНА
ИВАНИЩЕВ	ГРИГОРИЙ		17.04.1995	МУЖЧИНА
ДУБРОВ	ЕВГЕНИЙ	ИГОРЕВИЧ	03.10.1986	МУЖЧИНА
ДУБРОВ	АНТОН	ИГОРЕВИЧ	01.09.1991	null
ГАРАНИН	ИВАН		14.01.1975	МУЖЧИНА
АЛЕКСАНДРОВ	ИГОРЬ	СЕРГЕЕВИЧ	11.02.1967	МУЖЧИНА
ДУБОВА	ИРИНА	СЕРГЕЕВНА	01.09.1996	ЖЕНЩИНА

Рис. 9.4. Объединение двух таблиц

В предыдущем запросе SQL выбирает все строки из таблицы «физические лица» (fl) и объединяет их со строками в таблице *fl_sex_spr*. И если значение строки *fl.sex_id* заполнено, то в выборку попадает лицо и его гендерная принадлежность. Если значение строки *fl.sex_id* не заполнено, то в выборку попадает лицо без учета гендерной принадлежности.

Схема соединения таблиц с помощью оператора LEFT OUTER JOIN показана на рис. 9.5.

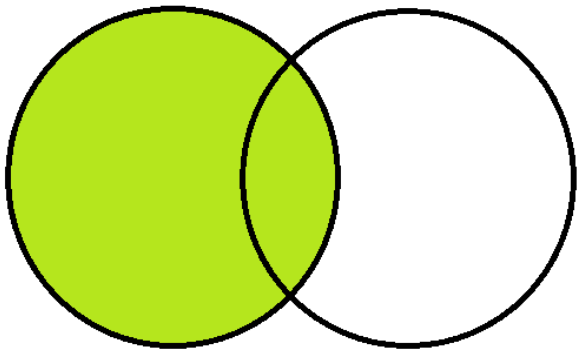


Рис. 9.5. LEFT OUTER JOIN

9.4. Оператор RIGHT OUTER JOIN

Оператор RIGHT OUTER JOIN используется, наоборот, когда нам надо получить информацию о всех строках из второй (правой) таблицы и только тех строках первой (левой) таблицы, в которых выполняются условия соединения. Например, при объединении таблицы «физические лица» (fl) и таблицы *fl_sex_spr* с помощью оператора RIGHT OUTER JOIN будет получен результат, представленный на рис. 9.6.

```
SELECT fl.family, fl.name, fl.otch, fl.birth_date, fl_sex_spr.value
FROM fl
RIGHT OUTER JOIN fl_sex_spr
ON fl.sex_id = fl_sex_spr.id;
```

fl.family	fl.name	fl.otch	fl.birth_date	fl_sex_spr.value
КАЛИНИН	АРТУР	ПЕТРОВИЧ	23.10.1984	МУЖЧИНА
ИВАНИЩЕВ	ГРИГОРИЙ		17.04.1995	МУЖЧИНА
ДУБРОВ	ЕВГЕНИЙ	ИГОРЕВИЧ	03.10.1986	МУЖЧИНА
ГАРАНИН	ИВАН		14.01.1975	МУЖЧИНА
АЛЕКСАНДРОВ	ИГОРЬ	СЕРГЕЕВИЧ	11.02.1967	МУЖЧИНА
ДУБОВА	ИРИНА	СЕРГЕЕВНА	01.09.1996	ЖЕНЩИНА
null	null	null	null	НЕИЗВЕСТЕН

Рис. 9.6. Объединение двух таблиц с помощью оператора RIGHT OUTER JOIN

Схема соединения таблиц с помощью оператора RIGHT OUTER JOIN показана на рис. 9.7.

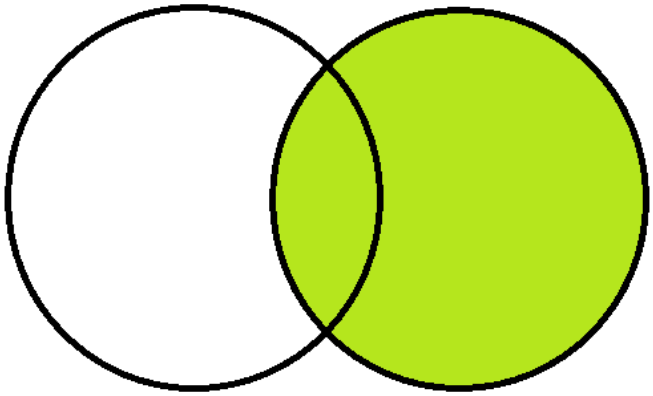


Рис. 9.7. RIGHT OUTER JOIN

9.5. Оператор FULL OUTER JOIN

Следующий оператор FULL OUTER JOIN используется для получения всех строк из первой таблицы и всех строк из второй таблицы. Полям строк, не подпадающих под условия объединения, присваиваются NULL-значения. При объединении таблиц *fl* и *fl_sex_spr* с помощью оператора FULL OUTER JOIN будет получен результат, представленный на рис. 9.8.

```
SELECT fl.family, fl.name, fl.otch, fl.birth_date, fl_sex_spr.value
FROM fl
FULL OUTER JOIN fl_sex_spr
ON fl.sex_id = fl_sex_spr.id;
```

fl.family	fl.name	fl.otch	fl.birth_date	fl_sex_spr.value
КАЛИНИН	АРТУР	ПЕТРОВИЧ	23.10.1984	МУЖЧИНА
ИВАНИЩЕВ	ГРИГОРИЙ		17.04.1995	МУЖЧИНА
ДУБРОВ	ЕВГЕНИЙ	ИГОРЕВИЧ	03.10.1986	МУЖЧИНА
ДУБРОВ	АНТОН	ИГОРЕВИЧ	01.09.1991	null
ГАРАНИН	ИВАН		14.01.1975	МУЖЧИНА
АЛЕКСАНДРОВ	ИГОРЬ	СЕРГЕЕВИЧ	11.02.1967	МУЖЧИНА
ДУБОВА	ИРИНА	СЕРГЕЕВНА	01.09.1996	ЖЕНЩИНА
null	null	null	null	НЕИЗВЕСТЕН

Рис. 9.8. Объединение двух таблиц с помощью оператора FULL OUTER JOIN

Схема соединения таблиц с помощью оператора RIGHT OUTER JOIN показана на рис. 9.9.

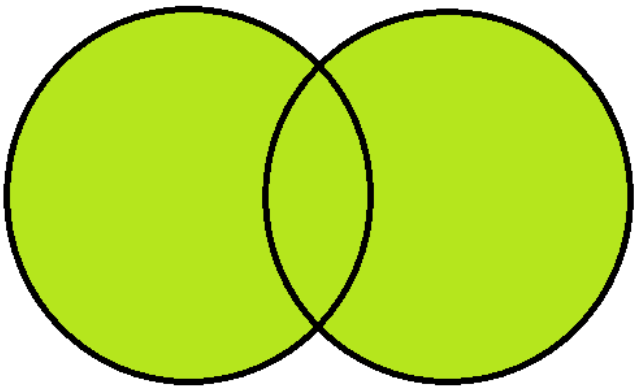


Рис. 9.9. FULL OUTER JOIN

9.6. Соединение таблиц с дополнительным условием

SQL позволяет не просто объединять таблицы, но и указывать при этом необходимые условия. Например, нам необходимо получить не только объединение двух таблиц (fl) и (fl_sex_spr), но и задать дополнительное условие, что информация должна быть представлена только о лицах женского пола.

```
SELECT fl.family, fl.name, fl.otch, fl.birth_date, fl_sex_spr.value
FROM fl
INNER JOIN fl_sex_spr
ON fl.sex_id = fl_sex_spr.id
WHERE fl_sex_spr.value = ' Женщина';
```

В результате выполнения этого запроса будет получен результат, представленный на рис. 9.10.

fl.family	fl.name	fl.otch	fl.birth_date	fl_sex_spr.value
ДУБОВА	ИРИНА	СЕРГЕЕВНА	01.09.1996	ЖЕНЩИНА

Рис. 9.10. Объединение таблиц с дополнительным условием

9.7. Соединение более чем двух таблиц

Если нужно установить связь между физическим лицом (fl) и транспортом (transp), то нам потребуется еще и таблица связи (link), через которую организованы связи объектов базы. Таким образом, мы можем получить информацию о транспортных средствах, с которыми связаны лица, внесенные в базу.

```
SELECT fl.family, fl.name, fl.otch, fl.birth_date,
transp.gos_num, transp.model_txt
FROM fl, transp, link
WHERE (fl.guid = link.in_fl
AND transp.guid = link. out_transp)
OR (fl.guid = link.out_fl
AND transp.guid = link. in_transp);
```

Получится выборка, представленная на рис. 9.11.

fl.family	fl.name	fl.otch	fl.birth_date	transp.gos_num	transp.model_txt
КАЛИНИН	АРТУР	ПЕТРОВИЧ	23.10.1984	M344OT77	M3
ИВАНИЩЕВ	ГРИГОРИЙ		17.04.1995	X637XX72	A6
ИВАНИЩЕВ	ГРИГОРИЙ		17.04.1995	O004OO72	525i
ДУБРОВ	ЕВГЕНИЙ	ИГОРЕВИЧ	03.10.1986	M001MM72	135i
ГАРАНИН	ИВАН		14.01.1975	P992TO72	2170
АЛЕКСАНДРОВ	ИГОРЬ	СЕРГЕЕВИЧ	11.02.1967	E584OT72	PASSAT

Рис. 9.11. Соединение трех таблиц

При соединении более двух таблиц также возможно создание дополнительных условий.

```
SELECT fl.family, fl.name, fl.otch, fl.birth_date,
transp.gos_num, transp.model_txt
FROM fl, transp, link
WHERE ((fl.guid = link.in_fl
AND transp.guid = link. out_transp)
OR (fl.guid = link.out_fl
AND transp.guid = link. in_transp))
AND fl.family = 'ИВАНИЩЕВ';
```

В результате выполнения запроса будет получен результат, представленный на рис. 9.12.

fl.family	fl.name	fl.otch	fl.birth_date	transp.gos_num	transp.model_txt
ИВАНИЩЕВ	ГРИГОРИЙ		17.04.1995	X637XX72	A6
ИВАНИЩЕВ	ГРИГОРИЙ		17.04.1995	O004OO72	525i

Рис. 9.12. Соединение трех таблиц с дополнительным условием

Задания для самоконтроля

1. Выберите лиц из таблицы fl, связанных с автомобилями марки BMW.

2. Выберите гендерную принадлежность лица, связанного с аккаунтом социальной сети ВКонтакте «rimski».

10. Вложенные запросы

10.1. Назначение подзапросов

SQL допускает создание запроса в запросе. Такой запрос называется вложенным или подзапросом. Вложенные запросы могут находиться в разделах WHERE, FROM или SELECT. В СУБД Oracle допускается использовать до 255 уровней подзапросов в предложении WHERE. Количество подзапросов в разделе FROM не ограничено.

Вложенный запрос в разделе WHERE, как правило, используется в ситуации, когда условие поиска нам не известно, и его необходимо предварительно установить. Например, если нам известна только фамилия (family) лица (fl), но мы не знаем его года рождения (*birth_year*), при этом нам необходимо выбрать всех лиц с таким же годом рождения, как у исходного лица, то нужно ввести запрос:

```
SELECT family, name, otch, birth_date
FROM fl
WHERE birth_year =
(SELECT birth_year
        FROM fl
        WHERE family = 'КАЛИНИН');
```

Получится выборка, представленная на рис.10.1.

family	name	otch	birth_date
КАЛИНИН	АРТУР	ПЕТРОВИЧ	23.10.1984

10.1. Использование подзапроса

Для того чтобы обработать внешний запрос, SQL должен сначала обработать внутренний запрос. В результате выбранной оказывается единственная строка, где *birth_year* = 1984. SQL образует из данного значения условие для внешнего запроса, которое теперь будет выглядеть так: «…WHERE *birth_year* = 1984». После этого выполняется внешний запрос как обычный. Внутренний запрос должен выбрать только один столбец, и тип данных выбранного столбца в обязательном порядке должен соответствовать типу значения, содержащемуся в условии внешнего запроса. Если бы было известно значение поля «год рождения» изначально, то можно было бы указать:

«…*birth_year* = 1984»

и тем самым освободиться от внутреннего запроса, однако использование внутреннего запроса делает данную процедуру более гибкой.

10.2. Значения, получаемые из подзапросов

Подзапрос в рассмотренном запросе возвращает только одно значение. Если вместо WHERE family = 'КАЛИНИН' подставить WHERE family = 'ДУБРОВ', то результатом становятся несколько значений. Оценка истинности условия в таком случае становится невозможной, что приводит к оценке запроса как ошибочного. Если используются внутренние запросы, основанные на реляционных операторах, необходимо знать наверняка, что результатом обработки подзапроса будет только одно значение. Если применяется внутренний запрос, не генерирующий значений, то это не ошибка, но в таком случае и внешний запрос не даст никаких результатов.

Для гарантии получения одного значения в результате выполнения внутреннего запроса можно использовать DISTINCT.

10.3. Применение многострочных операторов сравнения IN, ANY, ALL в подзапросах

При работе с внутренними запросами, в результате обработки которых получается несколько строк, применяются операторы IN, ANY, ALL.

Пример построения сложного запроса по выборке среди лиц (fl), находящихся в базе, ровесников Дубровых:

```
SELECT fl.family, fl.name, fl.otch, fl.birth_date
FROM fl
WHERE fl. birth_year IN
    (SELECT birth_year
    FROM fl
    WHERE family = 'ДУБРОВ');
```

Пример построения сложного запроса по выборке лиц (fl), находящихся в базе, которые старше Дубровых:

```
SELECT fl.family, fl.name, fl.otch, fl.birth_date
FROM fl
WHERE birth_year < ALL
    (SELECT birth_year
    FROM fl
    WHERE family = 'ДУБРОВ');
```

10.4. Создание многоуровневых подзапросов

При поиске информации в SQL-ориентированных базах данных зачастую приходится использовать многоуровневые сложные запросы. Приведем несколько примеров.

Установим данные лиц (fl) мужского пола (fl_sex_spr.value), которым принадлежит автомобиль (transp) модели PASSAT (transp.model_txt) (рис. 10.2).

```
SELECT fl.family, fl.name, fl.otch, fl.birth_date
FROM fl
WHERE fl.guid IN
    ((SELECT link.in_fl
    FROM link
    WHERE link.out_transp IN
        (SELECT transp.guid
        FROM transp
        WHERE transp.model_txt = 'PASSAT')),
    (SELECT link.out_fl
    FROM link
    WHERE link.in_transp IN
        (SELECT transp.guid
        FROM transp
        WHERE transp.model_txt = 'PASSAT')))
AND fl.sex_id IN
    (SELECT fl_sex_spr.id
    FROM fl_sex_spr
    WHERE fl_sex_spr.value LIKE 'M%');
```

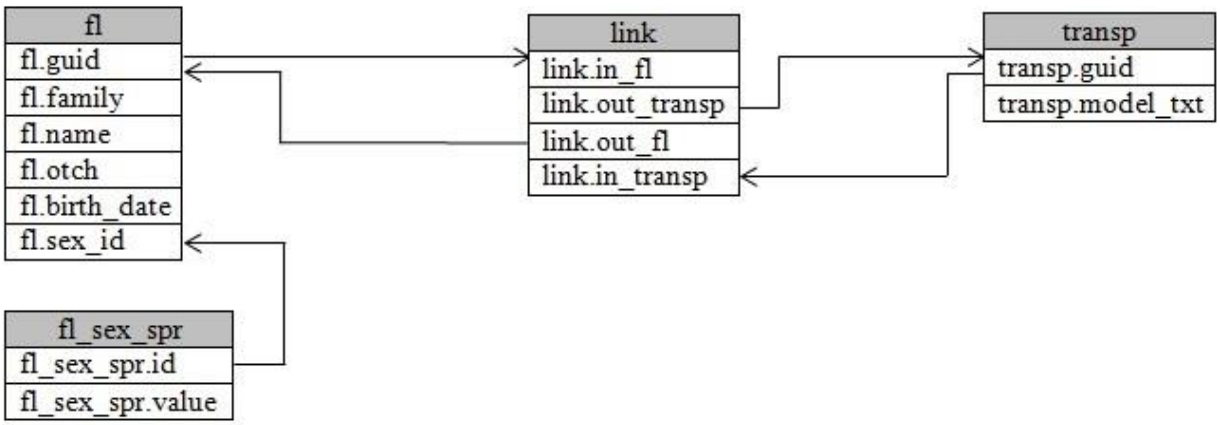


Рис. 10.2. Схема связи таблиц при выполнении сложного запроса

Установим данные лиц (fl) мужского пола (fl_sex_spr.value), которым принадлежит автомобиль (transp) марки ФОЛЬКСВАГЕН (transp_mark_spr.name_rus // transp_mark_spr.name_lat) коричневого цвета (transp_color_spr) (рис. 10.3).

```
SELECT fl.family, fl.name, fl.otch, fl.birth_date
FROM esd.fl
WHERE
fl.sex_id IN
    (SELECT fl_sex_spr.id
    FROM esd.fl_sex_spr
    WHERE fl_sex_spr.value LIKE 'M%')
AND fl.guid IN
    ((SELECT link.in_fl
    FROM esd.link
    WHERE link.out_transp IN
        (SELECT transp.guid
        FROM esd.transp
        WHERE transp.mark_id IN
            (SELECT transp_mark_spr.id
             FROM esd.transp_mark_spr
             WHERE transp_mark_spr.name_rus LIKE 'ФОЛЬКСВАГЕН'
             OR transp_mark_spr.name_lat LIKE 'VOLKSWAGEN')
             AND esd.transp.color_id in
                 (SELECT esd.transp_color_spr.id FROM esd.transp_color_spr
                  WHERE esd.transp_color_spr.value LIKE 'Коричневый'))),
    (SELECT link.out_fl
    FROM esd.link
    WHERE link.in_transp IN
        (SELECT transp.guid
        FROM esd.transp
        WHERE transp.mark_id IN
            (SELECT transp_mark_spr.id
             FROM esd.transp_mark_spr
             WHERE transp_mark_spr.name_rus LIKE 'ФОЛЬКСВАГЕН'
             OR transp_mark_spr.name_lat LIKE 'VOLKSWAGEN')
             AND esd.transp.color_id in
                 (SELECT esd.transp_color_spr.id FROM esd.transp_color_spr
                  WHERE esd.transp_color_spr.value LIKE 'Коричневый'))));
```

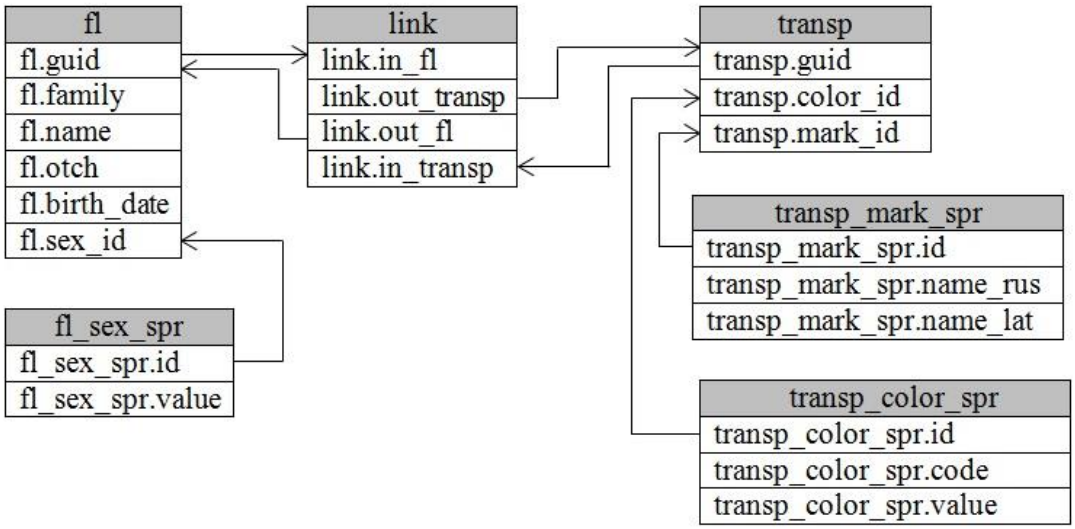


Рис. 10.3. Схема связей таблиц при выполнении сложного запроса

В предыдущих примерах показано, что нет предела глубине построения вложенных запросов. Соответственно, информацию можно собирать по крупицам из обширного диапазона таблиц.

Очень часто вложенные запросы целесообразно сочетать с соединением таблиц. Так, для выполнения последнего задания может быть составлен такой запрос:

```
SELECT fl.family, fl.name, fl.otch, fl.birth_date
FROM fl, fl_sex_spr
WHERE fl.sex_id=fl_sex_spr.id and fl_sex_spr.value='Мужчина'
AND fl.guid IN
      (SELECT esd.fl.guid
       FROM fl, transp, link, transp_mark_spr, transp_color_spr
        WHERE (((fl.guid=link.in_fl AND link.out_transp = transp.guid)
        OR (fl.guid=link.out_fl AND link.in_transp = transp.guid))
        AND transp.guid IN
                (SELECT transp.guid
                FROM transp, transp_mark_spr, transp_color_spr
                WHERE (transp.mark_id=transp_mark_spr.id
                AND transp.color_id=transp_color_spr.id
                AND (transp_mark_spr.name_rus LIKE 'ФОЛЬКСВАГЕН' OR
                transp_mark_spr.name_lat LIKE 'VOLKSWAGEN')
                 AND transp_color_spr.value = 'Коричневый'))))
```

Задания для самоконтроля

1. Выберите цвет автомобиля владельца электронной почты tinto@live.com.

2. Выберите даты рождения лиц, связанных с автомобилями, зарегистрированными в 77 регионе.

11. Ввод, удаление и изменение значений полей

11.1. Ввод новых строк

Добавление новых строк в таблицу осуществляется с помощью команды INSERT.

Предположим, что нам необходимо внести фамилию, имя и отчество нового лица в базу данных. Структура таблицы «физические лица» изображена на рис. 11.1.

guid	family	name	otch
3eqjak20…	ОТОРВИН	НИКОЛАЙ	ИВАНОВИЧ

Рис. 11.1 Таблица fl

Команда INSERT в данном случае будет иметь следующий синтаксис:

```
INSERT INTO esd.fl (guid, family, name, otch) VALUES
(SYS_GUID(), 'АРТЕМЬЕВ', 'ИВАН', 'ОЛЕГОВИЧ')
```

После применения команды и последующей отработки значений, таблица примет вид, показанный на рис. 11.2

guid	family	name	otch
8rs68zy9…	АРТЕМЬЕВ	ИВАН	ОЛЕГОВИЧ

11.2 Применение команды INSERT

Команда не генерирует выходных данных, но пользователь будет уведомлен о том, что данные внесены в таблицу. Имя таблицы должно быть задано предварительно, а каждое значение в списке должно иметь тип данных, соответствующий типу данных столбца, в который это значение должно быть вставлено. Порядок указания вводимых данных в разделе *VALUES* должен соответствовать порядку указания столбцов в разделе *INSERT INTO.*

Если необходимо вставить NULL-значение, нужно указать его как обычное значение. Допустим, значение поля *otch* для новой записи неизвестно. В этом случае можно вставить строку, указав NULL в столбце *otch*:

```
INSERT INTO esd.fl (guid, family, name, otch) VALUES
(SYS_GUID(), 'ГАРИН', 'ВИКТОР', NULL);
```

Так как NULL является специальным символом, а не обычным символьным значением, то указывается без одиночных кавычек.

guid	family	name	otch
027gp4w4…	ГАРИН	ВИКТОР	null

11.3 Добавление NULL-значений командой INSERT

При этом значениям полей, которые находятся в столбцах, не указанных в разделе INSERT INTO, будет присвоено значение NULL.

Для указания имен столбцов, в которые необходимо ввести значения, порядок столбцов в таблице неважен. Предположим, значения для таблицы fl берутся из файла, который вам выдали для ввода в базу данных. Предположим, что данные в этом файле представлены в следующем виде:

11.02.1978	ГОРИН	ЮРИЙ	АЛЕКСЕЕВИЧ

Для простоты понимания предпочтительно вводить значения в порядке, указанном в файле. Синтаксис будет следующим:

```
INSERT INTO fl (guid, birth_date, family, name, otch)
VALUES (SYS_GUID(), TO_DATE ('11.02.1975', 'dd.mm.yyyy'), 'ГОРИН',
'ЮРИЙ', 'АЛЕКСЕЕВИЧ');
```

Команду INSERT можно применять для того, чтобы извлечь значения из одной таблицы и разместить их в другой, воспользовавшись для этого командой SELECT. Для этого достаточно заменить предложение VALUES на соответствующий запрос. Допустим, кроме таблицы *fl* у нас есть таблица *lico_obmen*, которая используется для загрузки данных из файлов «как есть» и для последующего массового ввода информации. Выглядят они примерно так (рис. 11.4.):

fl
guid
family
name
otch

lico_obmen
guid
fam
imj
otch

Рис. 11.4. Таблицы fl и lico_obmen.

Команда для автоматического ввода информации из таблицы *lico_obmen* в таблицу *fl* будет иметь следующий синтаксис:

```
INSERT INTO fl (guid, family, name, otch)
SELECT * FROM lico_obmen;
```

Для отбора только определенных значений возможно использование раздела WHERE оператора SELECT, например:

```
INSERT INTO fl (guid, family, name, otch)
SELECT * FROM lico_obmen
WHERE fam like 'ИВАН%';
```

При использовании команды DELETE могут применяться подзапросы.

При копировании информации из одной таблицы в другую часто может возникнуть необходимость проверить, нет ли уже в базе вновь водимого лица.

```
INSERT INTO fl (guid, family, name, otch)
SELECT guid, fam, imj, otch
FROM lico_obmen
WHERE NOT EXISTS (SELECT *
        FROM fl
        WHERE fl.guid = lico_obmen.guid);
```

11.2. Удаление строк

Записи из таблицы можно удалить с помощью команды DELETE. При использовании этой команды удаляются только строки целиком, а не отдельные значения этих строк. Для удаления всех строк таблицы необходимо ввести:

```
DELETE  FROM fl;
```

В результате отработки команды, таблица становится пустой и ее можно удалить командой DROP TABLE. Чаще всего из таблицы требуется удалить только некоторые строки. Чтобы их определить, можно, как и для запросов, использовать уточнения. Например:

```
DELETE FROM fl
WHERE family = 'ИВАНОВ';
```

Данная команда удалит все строки со значением поля *family* «ИВАНОВ». Удаление записи по первичному ключу обеспечит безопасность остальных записей, так как удаление по фильтру первичного ключа гарантирует удаление только одной записи. При использовании команды DELETE могут применяться подзапросы.

11.3. Изменение данных

Изменять некоторые или все значения в уже существующей строке можно с помощью команды UPDATE. Команда UPDATE состоит из двух обязательных разделов, начинающихся с ключевых слов UPDATE и SET.

```
UPDATE…
SET…
```

Раздел UPDATE определяет таблицу, в которой будет происходить изменение данных. В разделе SET указываются столбцы и присваиваемые новые значения. Например, для изменения отчества всем лицам на «Иванович» нужно применить следующую команду:

```
UPDATE fl
    SET otch = 'ИВАНОВИЧ';
```

Команда, изменяющая отчество Оторвину Николаю Ивановичу с «Иванович» на «Николаевич», будет такой:

```
UPDATE fl
SET otch = 'НИКОЛАЕВИЧ'
WHERE family = 'ОТОРВИН'
AND name = 'НИКОЛАЙ'
AND otch = 'ИВАНОВИЧ';
```

Команда UPDATE позволяет обновить значения сразу в нескольких столбцах. В этом случае в разделе SET пары «столбец – значение» разделяются запятыми.

```
UPDATE fl
SET otch = 'НИКОЛАЕВИЧ',
    name = 'ВЛАДИМИР'
WHERE family = 'ОТОРВИН'
AND name = 'НИКОЛАЙ'
AND otch = 'ИВАНОВИЧ';
```

Для внесения значения NULL его можно просто указать без необходимости использования специального синтаксиса. Поэтому, если нужно установить отчества всех лиц равными NULL, необходимо ввести следующую команду:

```
UPDATE fl
SET otch = NULL;
```

Задания для самоконтроля

1. Вставьте в таблицу fl лицо с атрибутами: «Иванцов Станислав Игоревич, 10.12.1960 года рождения».
2. Измените Иванцову Станиславу Игоревичу отчество на «Владимирович».
3. Создайте связь Иванцова Станислава Владимировича с номером сотового телефона 8 912 346 5784.
4. Удалите информацию об Иванцове Станиславе Владимировиче и его связях из базы данных.

Список рекомендуемой литературы

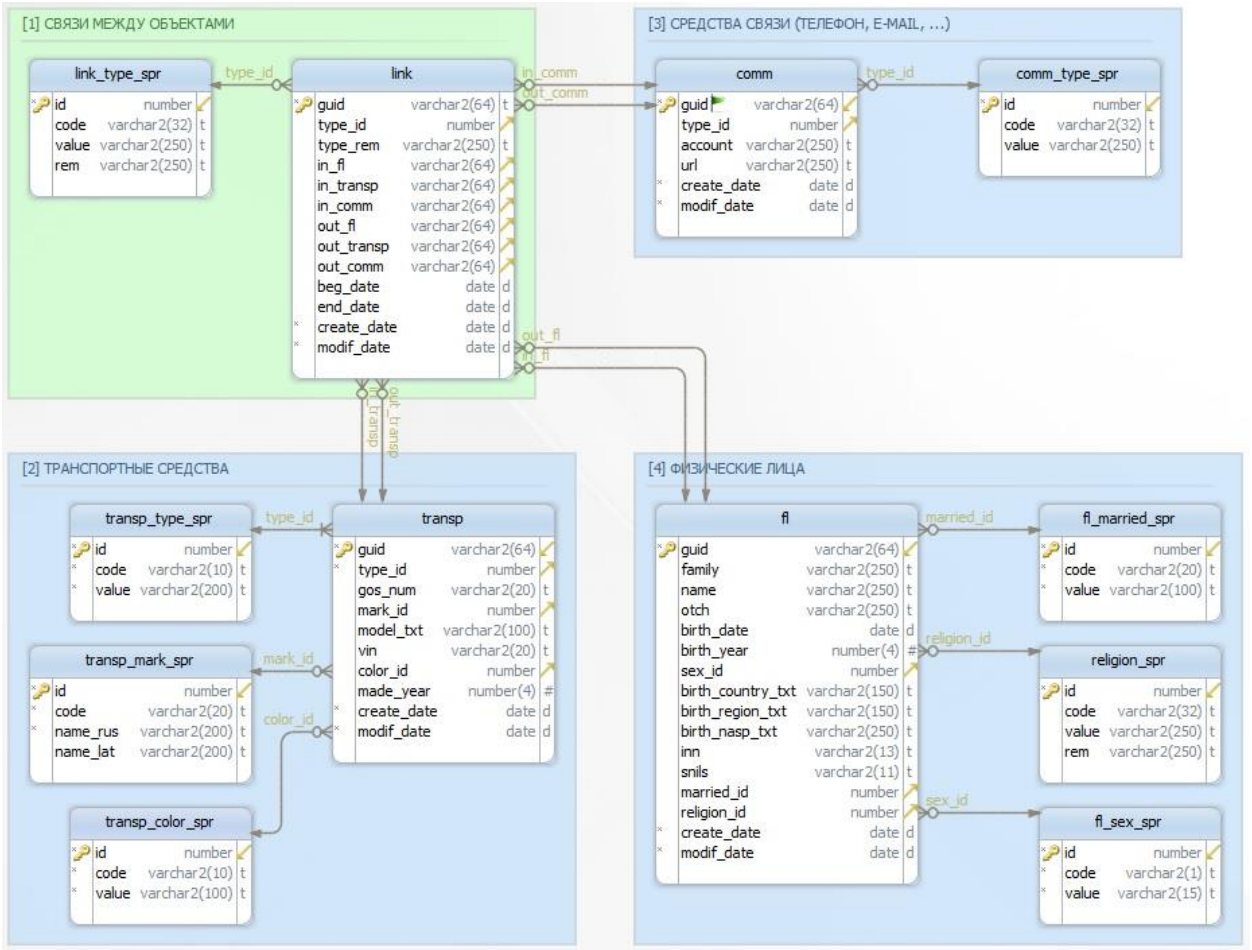
1. Oracle PL/SQL [Электронный ресурс]. URL: http://oracleplsql.ru (дата обращения: 16.03.2017).

2. Дейт К.Дж. Введение в системы баз данных / пер. с англ.; 8-е изд. М.: Вильяме, 2005. 1328 с.

3. Национальная платформа открытого образования. Курс «Управление данными [Электронный ресурс] / Санкт-Петербургский политехнический университет Петра Великого. URL: https://courses.openedu.ru/courses/course-v1:spbstu+DATAM+fall_2016/info (дата обращения: 12.12.2016).

4. Нестеров С.А. Базы данных: учебник и практикум для академического бакалавриата. М.: Юрайт, 2016. 230 с.

Инфологическая модель базы данных, используемая при работе с учебным пособием



Таблицы и данные, используемые при работе с учебным пособием

transp_color_spr

id	*code*	*value*
1	1	Белый
2	2	Желтый
3	3	Зеленый
4	4	Многоцветный
5	5	Коричневый
6	6	Красный
7	7	Черный
8	8	Серый
9	9	Синий
10	10	Оранжевый

transp_mark_spr

id	*code*	*name_rus*	*name_lat*
1	1	ЛЭНД РОВЕР	LAND ROVER
2	2	АУДИ	AUDI
3	3	БМВ	BMW
4	4	МЕРСЕДЕС-БЕНЦ	MERCEDES-BENZ
5	5	ОПЕЛЬ	OPEL
6	6	ПОРШ	PORSCHE
7	7	ФОЛЬКСВАГЕН	VOLKSWAGEN
8	8	ВАЗ	VAZ
9	9	ГАЗ	GAZ

transp_type_spr

id	*code*	*value*
1	1	Внедорожник
2	2	Кабриолет
3	3	Купэ
4	4	Пикап
5	5	С будкой
6	6	Седан
7	7	Универсал
8	8	Хэтчбэк

comm_type_spr

id	*code*	*value*
1	2	Стационарный телефон
2	3	Мобильный телефон
3	4	Ip-телефон
4	5	Электронная почта
5	6	Социальная сеть
6	7	Мессенджер
7	8	Интернет-сайт (содержащий блог, форум и т.п.)

link_type_spr

id	*code*	*value*	*rem*
1	1	Использует	
2	2	Владелец	
3	3	Звонит	
4	4	Рабочий телефон	
5	5	Управляет по доверенности	
6	6	Использует в работе	
7	7	Вписан в страховой полис	
8	8	Использует для совершения преступления	
9	9	Замечено на месте происшествия	
10	10	Зарегистрировано на ФЛ	

religion_spr

id	*code*	*value*
1	2	Ислам
2	3	Христианство
3	4	Иудаизм
4	5	Индуизм

fl_sex_spr

id	*code*	*value*
1	1	Мужчина
2	2	Женщина
3	3	Неизвестен

fl_married_spr

id	*code*	*value*
1	1	Холост (не замужем)
2	2	Женат (замужем)
3	3	Женат (замужем), живут раздельно
4	4	Разведен (разведена)
5	5	Вдовец (вдова)

transp

guid	*type_id*	*gos_num*	*mark_id*	*model_txt*	*vin*	*color_id*	*made_year*	*create_date*	*modif_date*
i1khaghr…	7	O004OO72	3	525i	SALFA28B57H011265	2	2003	02.05.2017	02.05.2017
ku8tgxa4…	7	X637XX72	2	A6	KMHBT31GP3U013758	6	2007	04.04.2017	04.04.2017
rs5b1c3m…	1	E703HP72	6	CAYENNE S	VF1LB0K0525551701	7	2003	15.03.2017	15.03.2017
landmku9…	3	M344OT77	3	M3	JHLRE48577C415490	4	2004	09.03.2017	09.03.2017
s2h4qhi…	6	P992TO72	8	2170	ZFA18800004473122	3	2016	17.02.2017	02.05.2017
qcvd921g…	6	E584OT72	7	PASSAT	KL1UF756E6B195928	5	2003	12.02.2017	12.02.2017
eywmapf7…	8	M001MM72	3	135i	4USBT53544LT26841	1	2015	03.01.2017	03.01.2017

fl

guid	*family*	*name*	*otch*	*birth_date*	*birth_year*	*inn*	*sex_id*	*birth_country_txt*	*birth_region_txt*	*birth_nasp_txt*	*snils*	*married_id*	*religion_id*	*create_date*	*modif_date*
f82e01b7…	ДУБОВА	ИРИНА	СЕРГЕЕВНА	01.09.1996	1996		2	РОССИЯ	ТЮМЕНЬ	ТЮМЕНЬ	579992315494	3	3	05.05.2017	05.05.2017
432d2bdb…	АЛЕКСАНДРОВ	ИГОРЬ	СЕРГЕЕВИЧ	11.02.1967	1967	660200852033	1	АРМЕНИЯ			999537518967	4	2	01.05.2017	01.05.2017
a72m6quf…	ГАРАНИН	ИВАН		14.01.1975	1975	781085030552	1	РОССИЯ	СМОЛЕНСК	ДЕСНОГОРСК	779561237728	1	3	16.04.2017	16.04.2017
3j8afqte…	ДУБРОВ	АНТОН	ИГОРЕВИЧ	01.09.1991	1991			РОССИЯ	МОСКВА	ПУШКИНО		1	3	25.03.2017	05.05.2017
39gj425t…	ДУБРОВ	ЕВГЕНИЙ	ИГОРЕВИЧ	03.10.1986	1986	780514587636	1	ГРУЗИЯ			649871235459	2	3	15.03.2017	15.03.2017
516g15ny…	ИВАНИЩЕВ	ГРИГОРИЙ		17.04.1995	1995		1	УКРАИНА		КИЕВ		2	3	05.02.2017	05.02.2017
30f779db…	КАЛИНИН	АРТУР	ПЕТРОВИЧ	23.10.1984	1984	715839976289	1	РОССИЯ	ТЮМЕНЬ	ИШИМ	562428656257	1	3	22.01.2017	22.01.2017

comm

guid	*type_id*	*account*	*url*		*create_date*	*modif_date*
jc51m7bg…	4	oleo26@hotmail.com			07.05.2017	07.05.2017
7kbhevsl…	6	236499746			08.04.2017	08.04.2017
095eefv9…	5	rimski	vk.com/rimskii		23.03.2017	23.03.2017
n7dzqram…	4	tinto@live.com			13.03.2017	13.03.2017
f7gatttf…	4	obs12@yahoo.com			08.02.2017	08.02.2017
4708y46x…	2	89220011598			04.02.2017	07.05.2017
ihqaq52d…	2	89051040301			17.01.2017	17.01.2017

link

guid	*type_id*	*type_rem*	*in_fl*	*in_transp*	*in_comm*	*out_fl*	*out_transp*	*out_comm*	*beg_date*	*end_date*	*create_date*	*modif_date*
s2dd1qbw…	3		3j8afqte…					ihqaq52d…			17.01.2017	17.01.2017
0mi0em69…	4				4708y46x…	516g15ny…					04.02.2017	04.02.2017
gqj14t9v…	2		30f779db…					095eefv9…			23.03.2017	23.03.2017
2s8ikft4…	1		255lu3rb…					n7dzqram…			13.03.2017	13.03.2017
cvymziy1…	10		432d2bdb…				qcvd921g…				12.02.2017	12.02.2017
ly33omsq…	9			s2h4qhi…		a72m6quf…					17.02.2017	02.05.2017
26351ax9…	8		39gj425t…				eywmapf7…				03.01.2017	03.01.2017
3z9jkghp…	7		516g15ny…				i1khaghr…				02.05.2017	02.05.2017
vlwd55sd…	6			ku8tgxa4…		516g15ny…					04.04.2017	04.04.2017
6tr4db4j…	5		30f779db…				landmku9…				09.03.2017	09.03.2017

Учебное издание

Панюшин Денис Борисович,
Сазонов Михаил Михайлович

SQL ДЛЯ СОТРУДНИКОВ ПОДРАЗДЕЛЕНИЙ ОПЕРАТИВНО-РАЗЫСКНОЙ ИНФОРМАЦИИ

Учебное пособие

Корректура и редактирование: *Е.В. Шабанова*
Техническое редактирование: *Е.К. Булатова*

Подписано в печать 05.07.2017. Формат 60х84/16
Усл. п. л. 3,6. Тираж 100 экз. Заказ № 92.

Редакционно-издательское отделение
Тюменского института повышения квалификации
сотрудников МВД России
625049, г. Тюмень, ул. Амурская, 75.

9 785931 602455