ФГБОУ ВПО «Воронежский государственный
технический университет»

СПРАВОЧНИК МАГНИТНОГО ДИСКА
(кафедра систем автоматизированного проектирования
и информационных систем)

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам по теме «Основные конструкции языка SQL» по дисциплине «Базы данных» для студентов направления 09.03.01 «Информатика и вычислительная техника» очной формы обучения, по дисциплине «Управление данными» для студентов направления 09.03.02 «Информационные системы и технологии» очной формы обучения

Составители Яскевич Ольга Георгиевна
Иванов Денис Вячеславович

БД.SQL 1.0.pdf 398 Кбайт 16.09.2015 2,7 уч.-изд.л.
(наименование файла) (объем файла) (дата) (объем издания)

ФГБОУ ВПО «Воронежский государственный
технический университет»

Кафедра систем автоматизированного проектирования
и информационных систем

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам по теме «Основные конструкции языка SQL» по дисциплине «Базы данных» для студентов направления 09.03.01 «Информатика и вычислительная техника» очной формы обучения, по дисциплине «Управление данными» для студентов направления 09.03.02 «Информационные системы и технологии» очной формы обучения

Воронеж 2015

Составители: канд. техн. наук О.Г. Яскевич,
ст. преп. Д.В. Иванов

УДК 681.38+681.3

Методические указания к лабораторным работам по теме «Основные конструкции языка SQL» по дисциплине «Базы данных» для студентов направления 09.03.01 «Информатика и вычислительная техника» очной формы обучения, по дисциплине «Управление данными» для студентов направления 09.03.02 «Информационные системы и технологии» очной формы обучения / ФГБОУ ВПО «Воронежский государственный технический университет»; сост. О.Г. Яскевич, Д.В. Иванов. Воронеж, 2015. 44 с.

Методические указания содержат краткие теоретические и практические сведения об основных конструкциях языках SQL и их практического применения.

Методические указания подготовлены в электронном виде в текстовом редакторе MS Word 2003 и содержатся в файле БД.SQL 1.0.pdf.

Табл. 11. Библиогр.: 4 назв.

Рецензент д-р техн. наук, проф. К.А. Разинкин

Ответственный за выпуск зав. кафедрой
д-р техн. наук, проф. Я.Е. Львович

Издается по решению редакционно-издательского совета Воронежского государственного технического университета

Лабораторная работа № 1

«Основы языка SQL»

Цель работы

Изучение теоретических основ языка SQL для проектирования базы данных с помощью SQL-запросов.

Краткие теоретические сведения

1. Основы языка SQL

1.1. Язык SQL

Информационное пространство становится более унифицированным. Это привело к необходимости создания стандартного языка, который мог бы использоваться в большом количестве различных видов компьютерных сред. Стандартный язык позволит пользователям, знающим один набор команд, использовать их для создания, нахождения, изменения и передачи информации - независимо от того, работают ли они на персональном компьютере, сетевой рабочей станции, или на универсальной ЭВМ.

Стандарт SQL определяется ANSI (Американским Национальным Институтом Стандартов) и в данное время также принимается ISO (Международной Организацией по Стандартизации). Однако, большинство коммерческих программ баз данных расширяют SQL без уведомления ANSI, добавляя различные особенности в этот язык, которые, как они считают, будут весьма полезны. Иногда они несколько нарушают стандарт языка, хотя хорошие идеи имеют тенденцию развиваться и вскоре становиться стандартами "рынка" сами по себе в силу полезности своих качеств.

SQL (обычно произносимый как "СИКВЭЛ" или "ЭСКЮЭЛЬ") символизирует собой Структурированный Язык Запросов. Это - язык, который дает Вам возможность создавать и работать в реляционных базах данных, являющихся наборами связанной информации, сохраняемой в таблицах.

1.2. Преимущества языка SQL

Язык SQL является основой многих СУБД, т.к. отвечает за физическое структурирование и запись данных на диск, а также за чтение данных с диска, позволяет принимать SQL- запросы от других компонентов СУБД и пользовательских приложений. Таким образом, SQL – мощный инструмент, который обеспечивает пользователям, программам и вычислительным системам доступ к информации, содержащейся в реляционных базах данных.

Основные достоинства языка SQL заключаются в следующем:

- стандартность;
- независимость от конкретных;
- возможность переноса с одной вычислительной системы на другую;
- реляционная основа;
- возможность создания интерактивных запросов;
- возможность программного доступа к БД;
- обеспечение различного представления данных;
- возможность динамического изменения и расширения структуры БД;
- поддержка архитектуры клиент-сервер – SQL.

1.3. Состав языка SQL

В язык SQL в качестве составных частей входят:

– язык определения данных (Data Definition Language, DDL);

– язык манипулирования данными (Data Manipulation Language, DML);

– язык запросов (Data Query Language, DQL);

– язык управления данными (Data Control Language, DCL).

Указанные языки не являются отдельными языками, а являются различными командами одного языка. Представленное деление приведено для выделения функциональных особенностей языка SQL.

1.3.1. Язык DDL

Язык определения данных используется для создания и изменения структуры базы данных и ее составных частей - таблиц, индексов, представлений (виртуальных таблиц), а также триггеров и сохраненных процедур. Основными его командами являются (табл. 1).

Таблица 1

Основные команды языка DDL

Команда	Описание
CREATE DATABASE	создать базу данных
CREATE TABLE	создать таблицу
CREATE VIEW	создать виртуальную таблицу
CREATE INDEX	создать индекс
CREATE TRIGGER	создать триггер
CREATE PROCEDURE	создать сохраненную процедуру
ALTER DATABASE	модифицировать базу данных
ALTER TABLE	модифицировать таблицу
ALTER VIEW	модифицировать виртуальную таблицу
ALTER INDEX	модифицировать индекс
ALTER TRIGGER	модифицировать триггер
ALTER PROCEDURE	модифицировать сохраненную процедуру
DROP DATABASE	удалить базу данных
DROP TABLE	удалить таблицу
DROP VIEW	удалить виртуальную таблицу

Продолжение табл. 1

DROP INDEX	удалить индекс
DROP TRIGGER	удалить триггер
DROP PROCEDURE	удалить сохраненную процедуру

1.3.2. Язык DML

Язык манипулирования данными используется, как это следует из его названия, для манипулирования данными в таблицах баз данных. Он состоит из 3 основных команд:

INSERT(вставить)
UPDATE (обновить)
DELETE (удалить)

1.3.3. Язык DQL

Язык запросов DQL наиболее известен пользователям реляционной базы данных, несмотря на то, что он включает всего одну команду SELECT. Эта команда вместе со своими многочисленными опциями и предложениями используется для формирования запросов к реляционной базе данных.

1.3.4. Язык DCL

Язык управления данными используется для управления правами доступа к данным и выполнением процедур в многопользовательской среде. Более точно его можно назвать "язык управления доступом". Он состоит из двух основных команд:

GRANT (дать права)
REVOKE (забрать права)

2. SQL-запросы определения данных

2.1. Запрос CREATE

Оператор CREATE DATABASE служит для создания новой базы данных и имеет следующий формат:

CREATE DATABASE <Имя базы данных>

Оператор CREATE TABLE служит для создания новой таблицы базы данных и имеет следующий формат:

```
CREATE TABLE <Имя таблицы>
<Имя поля> <Тип данных>,
…
<Имя поля> <Тип данных>);
```

В этом операторе обязательно указание хотя бы одного имени поля и его типа данных.

Приведем пример создания простой таблицы:

```
CREATE TABLE MyTable (
  Number INTEGER,
  Name CHAR(20),
  Surname CHAR(20)
);
```

При этом в каталоге текущей базы данных создастся новая таблица myTable, состоящая из полей Number, Name и Surname. Первое поле имеет целочисленный тип (INTEGER), остальные поля - символьного типа и ограничены длиной в 20 символов.

Если при выполнении этого запроса выяснится, что таблица с таким именем уже существует, будет сгенерирована исключительная ситуация.

Следующий SQL-запрос определит ключевое поле
для MySQL:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
PRIMARY KEY (P_Id)
)
```

для SQL Server:

```
CREATE TABLE Persons
(
P_Id int NOT NULL PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

Следующий SQL-запрос добавит внешний ключ P_Id для поля P_Id таблицы Persons

для MySQL:

```
CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
PRIMARY KEY (O_Id),
FOREIGN KEY (P_Id) REFERENCES Persons(P_Id)
)
```

для SQL Server

```
CREATE TABLE Orders
(
```

```
O_Id int NOT NULL PRIMARY KEY,
OrderNo int NOT NULL,
P_Id int FOREIGN KEY REFERENCES Persons(P_Id)
)
```

Для создания инкремента в ключевом поле, то есть, чтобы счетчик самостоятельно увеличивался на определенное значение, необходимо использовать атрибут IDENTITY(seed, increment) с параметрам, где seed – значение, присваиваемое самой первой строке, загружаемой в таблицу, increment - значение приращения, которое прибавляется к значению идентификатора предыдущей загруженной строки. В большинстве случаев требуется задавать:

```
IDENTITY(1, 1)
```

Таким образом, результирующий SQL-запрос для SQL Server будет выглядеть следующим образом:

```
CREATE TABLE Orders
(
O_Id int NOT NULL PRIMARY KEY IDENTITY(1, 1),
OrderNo int NOT NULL,
P_Id int FOREIGN KEY REFERENCES Persons(P_Id)
)
```

2.2. Запрос ALTER

Оператор ALTER DATABASE предназначен для изменения атрибутов базы данных, например:

```
ALTER DATABASE `test` DEFAULT CHARACTER SET utf8;
```

Оператор ALTER TABLE предназначен для добавления или удаления полей действующей таблицы базы данных. Во

время действия этого оператора никакие другие приложения не должны обращаться к таблице. Этот оператор имеет следующий формат:

```
ALTER TABLE <Имя таблицы>
ADD <Имя поля> <Тип данных>,
DROP <Имя поля>,
…
ADD <Имя поля> <Тип данных>,
DROP <Имя поля>;
```

При этом операнд ADD добавляет к таблице новое поле, а оператор DROP удаляет из таблицы существующее поле. Операнды могут располагаться внутри оператора ALTER TABLE произвольно:

```
ALTER TABLE MyTable
ADD Telefon INTEGER,
ADD Address CHAR (50),
DROP Number;
```

При выполнении данного примера в таблицу MyTable добавятся два поля: felefon и Address, целочисленного и символьного типа соответственно. Кроме того, будет удалено поле Number.

2.3. Запрос DROP

Оператор DROP DATABASE <имя БД> служит для удаления имеющейся базы данных.

```
DROP TABLE MyDB.
```

Оператор DROP TABLE <имя таблицы> служит для удаления имеющейся таблицы. Если таблицы с таким именем не существует, будет сгенерирована исключительная ситуация:

DROP TABLE MyTable.

3. SQL-запросы манипулирования данными

3.1. Запрос INSERT

Команда INSERT INTO предназначена для добавления одной или нескольких записей в конец таблицы. Возможны 2 варианта использования этой команды. Первый вариант добавляет одну запись в таблицу, а второй вариант добавляет записи из одной таблицы в другую.

Синтаксис первого варианта:

INSERT INTO ТаблицаНазначения [(Поля)] VALUES (Значения);

Синтаксис второго варианта:

INSERT INTO ТаблицаНазначения [(Поля)] [IN БазаДанных] SELECT [Таблица.]Поля FROM Таблица;

«ТаблицаНазначения» - таблица, в которую добавляются записи; «Поля» - названия полей; «Таблица» - имя таблицы, источника данных; «База данных» - путь и имя внешней базы данных, в которой содержатся таблицы. Если таблицы находятся в текущей базе данных, то этот аргумент необязателен. «Значения» - значения полей добавляемой записи.

Все поля записи и соответствующие им значения должны быть определены, иначе им будут присвоены значения Null.

Если таблица, в которую добавляются записи, имеет ключевое поле, то в него должны добавляться уникальные, непустые значения. Иначе запись не будет добавлена:

INSERT INTO Orders (ID, Name, Email, Order) VALUES (12, 'Иван Иванов', 'ivan@ivanov.ru', 'Pentium II 450 MHz');

Добавляется новая запись, в которой полям ID, Name, Email, Order соответствуют значения 12, 'Иван Иванов', 'ivan@ivanov.ru', 'Pentium II 450 MHz'.

INSERT INTO Orders2001 (ID, Name, Email, Order) SELECT ID, Name, Email, Order FROM Orders2000;

Этот запрос добавляет все записи из таблицы Orders2000 в таблицу Orders2001.

3.2. Запрос UPDATE

Команда UPDATE посылает запрос на изменение записи:

UPDATE Таблица SET НовоеЗначение WHERE ...;

«Таблица» - имена одной или нескольких таблиц, в которых изменяются записи; «НовоеЗначение» - новые значения для полей записи.

Команду UPDATE удобно использовать, если изменяется сразу большое число записей или если изменяемые записи находятся в разных таблицах. Новые значения указываются через запятую для каждого поля. Использование предложения WHERE аналогично его использованию в команде SELECT:

UPDATE Buyers SET Order='Ничего' WHERE ID=7;

Устанавливаем значение поля покупки 'Ничего' у покупателя, номер которого равен 7.

UPDATE Заказы SET СуммаЗаказа = СуммаЗаказа * 1.2, СтоимостьДоставки = СтоимостьДоставки * 1.1 WHERE Страна='Россия';

Этот запрос немного сложнее. Он повышает сумму заказа на 20% и стоимость доставки на 10% для покупателей из Россия.

3.3. Запрос DELETE

Команда DELETE посылает запрос на удаление записей из таблицы:

DELETE [Таблица.*] FROM Таблица WHERE ...;

Таблица - имя таблицы, из которой удаляются записи.

Использование предложения WHERE аналогично его использованию в команде SELECT.

Аргумент команды DELETE можно не указывать, поскольку он фактически дублируется в предложении FROM:

DELETE FROM Buyers WHERE ID=8;

Этот запрос удаляет из таблицы Buyers запись, в которой ID равно 8.

Для удаления не всей записи, а только ее поля, следует воспользоваться запросом на изменение записи (команда UPDATE) и поменять значения нужных полей на Null.

3.4. Запрос TRUNCATE

Запрос TRUNCATE TABLE удаляет все строки в таблице, не записывая в журнал удаление отдельных строк. Инструкция TRUNCATE TABLE похожа на инструкцию DELETE без предложения WHERE, однако TRUNCATE TABLE выполняется

быстрее и требует меньших ресурсов системы и журналов транзакций.

```
TRUNCATE MyTable;
```

Инструкция TRUNCATE TABLE удаляет все строки таблицы, но структура таблицы и ее столбцы, ограничения, индексы и т. п. сохраняются. Чтобы удалить не только данные таблицы, но и ее определение, следует использовать инструкцию DROP TABLE.

Если таблица содержит столбец идентификаторов, счетчик этого столбца сбрасывается до начального значения, определенного для этого столбца. Если начальное значение не задано, используется значение по умолчанию, равное 1. Чтобы сохранить столбец идентификаторов, используйте инструкцию DELETE.

Технология выполнения лабораторной работы

1. Запустите MySQL сервер.

Чтобы создать запрос вам нужно запустите Microsoft SQL Server Management Studio, далее, просто нажать кнопку New Querty (Новый запрос) на панели инструментов (рис. 1).

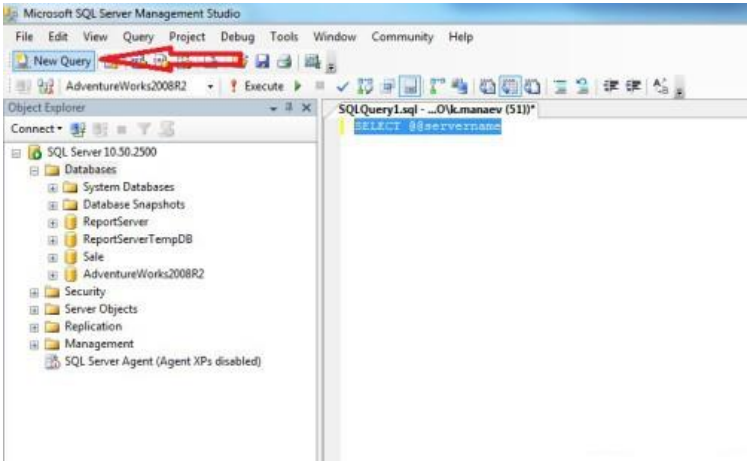

Рис. 1. Создание нового SQL запроса

2. В окне SQL запроса в верхнем полу пишется текст запроса, а в нижнем - выводится таблица результата выполнения запроса (рис. 2). Чтобы запустить запрос на выполнение нужно нажать кнопку Execute на панели инструментов.

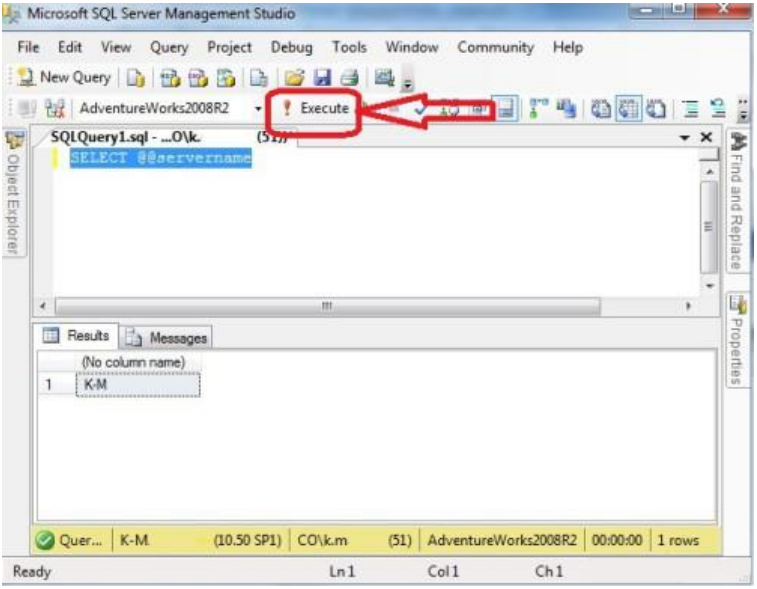

Рис. 2. Окно SQL-запроса

3. Выполните запросы своего варианта для построения базы данных и манипуляции данных.

Задание на лабораторную работу

1. Запустить SQL сервер.
2. Создать базу данных с помощью SQL-запросов.
3. Изменить структуру базы данных с помощью запросов определения данных.
4. Записать некоторые данные в таблицы базы данных.
5. Выполнить запросы манипулирования данными.

6. Показать результаты работы преподавателю.
7. Подготовить отчет по лабораторной работе.

Лабораторная работа № 2

«SQL-запросы выбора данных»

Цель работы

Изучение теоретических основ и получение практических навыков в области выбора и фильтрации данных.

Краткие теоретические сведения

1. SQL-запрос SELECT

Оператор SELECT образует основу каждого вопроса, который вы задаете базе данных. Когда создается и выполняется оператор SELECT, то вы “обращаетесь с запросом к базе данных” (надеемся, что все читатели единодушны в этом вопросе). Фактически многие программы СУРБД позволяют сохранить оператор SELECT как запрос, представление или хранимую процедуру. Когда кто-то собирается обратиться с запросом к базе данных, вы должны понимать, что предполагается выполнить некий оператор SQL. В зависимости от программы СУРБД операторы SELECT могут выполняться по-разному: непосредственно из окна командной строки, из таблицы интерактивного запроса с использованием примера (Query By Example, QBE) или из блока программного кода. Независимо от того, каким образом вы решили определить и выполнить его, синтаксис оператора SELECT всегда один и тот же.

2. Структура SQL-запроса SELECT

Команда SELECT - наиболее часто употребляемая команда из всех восьми. Она используется для выборки данных из базы данных. Её синтаксис:

SELECT [Предикат] Поля FROM Таблицы [IN БазаДанных] [WHERE ...] [GROUP BY ...] [HAVING ...] [ORDER BY ...];

Необязательные аргументы заключены в [].

Предикат - одно из четырёх слов ALL, DISTINCT, DISTINCTROW, TOP. Если предикат не указан, то устанавливается ALL. Предикат ALL позволяет отобрать все записи. При использовании предиката DISTINCT, записи, которые содержат повторяющиеся значения в выбранных в запросе полях, исключаются. Предикат DISTINCTROW исключает из выборки записи, если повторяется вся запись, а не одно из полей. Предикат TOP позволяет отобрать определённое количество записей.

Минимальный синтаксис запроса на выборку выглядит так:

SELECT поле FROM Таблица.

Поместите звездочку сразу после условия SELECT, когда нужно определить все столбцы из исходной таблицы в условии FROM. Например, так выглядит предшествующий оператор SELECT при использовании сокращения:

SELECT * FROM Subjects

3. Исключение дубликатов строк (DISTINCT)

При работе с операторами SELECT вы неизбежно столкнетесь в наборах результатов с повторяющимися строками. Поэтому используйте ключевое слово DISTINCT в своем операторе SELECT, и набор результатов не будет содержать повторяющихся строк:

SELECT DISTINCT City FROM Bowlers

Ключевое слово DISTINCT также можно использовать для составных столбцов. Изменим предыдущий пример, запросив из таблицы Bowlers как штат, так и город. Наш новый оператор SELECT будет выглядеть так:

```
SELECT DISTINCT State, City FROM Bowlers
```

Этот оператор SELECT возвращает набор результатов, который содержит уникальные записи.

4. Сортировка информации (ORDER BY)

Операцию SELECT можно разделить на три более мелкие операции: оператор SELECT, выражение SELECT и запрос SELECT. Можно объединять эти операции различными способами, чтобы ответить на сложные запросы. Также необходимо объединять эти операции для сортировки строк в наборе результатов.

По определению строки набора результатов, возвращенные оператором SELECT, не упорядочены. Последовательность, в которой они появляются, основывается на их физическом расположении в таблице.

Условие ORDER BY запроса SELECT позволяет определить последовательность строк в окончательном наборе результатов. Из последующих глав вы узнаете, что для ответа на очень сложные вопросы можно вложить оператор SELECT в другой оператор SELECT или в выражение SELECT. Однако запрос SELECT невозможно вложить в какой-либо другой уровень.

Условие ORDER BY позволяет упорядочить набор результатов указанного оператора SELECT по одному или нескольким столбцам, а также содержит опцию упорядочивания по возрастанию или убыванию для каждого столбца. В условии ORDER BY можно использовать только те столбцы, которые в настоящий момент перечислены в условии SELECT. Хотя это

требование указано в стандарте SQL, некоторые реализации его полностью игнорируют (однако во всех примерах, используемых в данной книге, это требование нами соблюдается). Когда в условии ORDER BY используются два или более столбцов, каждый столбец отделяется запятой. Как только сортировка завершается, запрос SELECT возвращает окончательный набор:

```
SELECT Category FROM Classes ORDER BY Category
```

В данном примере можно предположить, что категория будет использоваться для сортировки, поскольку это единственный столбец, указанный в запросе. Также можно предположить, что сортировка должна производиться в порядке возрастания, потому что в запросе не указано противоположное. Это безопасное предположение.

В следующем запросе столбец, необходимый для сортировки, определяется более явно:

```
SELECT VendName, VendZipCode
FROM Vendors
ORDER BY VendZipCode
```

В соответствие со стандартом SQL, если не определен порядок сортировки, то автоматически предполагается сортировка в порядке возрастания. Однако, если вы хотите все указать явно, вставьте ASC (от Ascending — по возрастанию) после Category в условии ORDER BY.

SELECT EmpLastName, EmpFirstName, EmpPhoneNumber, EmployeelD FROM Employees ORDER BY EmpLastName DESC, EmpFirstName ASC

Если нужно отобразить набор результатов в обратном порядке, вставьте ключевое слово DESC (от Descending — по

убыванию) после соответствующего столбца в условии ORDER BY. Посмотрите, как изменяется оператор SELECT из предыдущего примера, когда требуется представить информацию, отсортированную по почтовому индексу в порядке убывания:

```
SELECT VendName, VendZipCode FROM Vendors
ORDER BY VendZipCode DESC
```

Пример совместного использования сортировки по возрастанию и по убыванию:

SELECT TourneyDate, TourneyLocation FROM Tournaments ORDER BY TourneyDate DESC, TourneyLocation ASC

5. Фильтрация данных

5.1. Условие WHERE

Условие WHERE в операторе SELECT используется для фильтрации данных, которые оператор извлекает из таблицы. WHERE содержит условие поиска, применяемое оператором в качестве фильтра. Именно это условие поиска обеспечивает механизм, необходимый для выбора только требуемых строк или для удаления ненужных. СУБД применяет это условие поиска к каждой строке в логической таблице, определенной условием FROM.

Условие поиска содержит один или более предикатов, каждый из которых является выражением, которое тестирует одно или несколько типизированных выражений и возвращает в ответе True, False или Unknown. Несколько предикатов можно объединять в условие поиска, используя булевы операторы AND или OR. Когда условие поиска оказывается равным True для конкретной строки, эта строка будет присутствовать в окончательном наборе результата. Когда условие поиска содержит только один предикат, термины “условие поиска” и “предикат” являются синонимами.

Типизированное выражение может содержать имена столбца, значения литерала, функции или другие типизированные выражения. При построении предиката обычно включается по крайней мере одно типизированное выражение, которое указывает столбец из таблиц, определенных в условии FROM.

Самый простой и, возможно, наиболее распространенный предикат сравнивает одно типизированное выражение (столбец) с другим (литерал). Например, если нужны только те строки из таблицы Customers, в которых значение столбца фамилии клиента — “Smith”, то записывается предикат, который сравнивает столбец фамилии со значением литерала “Smith”.

```
SELECT CustLastName FROM Customers
WHERE CustLastName = ‘Smith’
```

Предикат в условии WHERE эквивалентен обращению с вопросом к каждой строке в таблице Customers: “Совпадает ли фамилия клиента со ’’Smith"?" Когда ответ положительный (True) для произвольной строки в таблице Customers, эта строка появляется в наборе результатов.

5.2. Базовые предикаты

Стандарт SQL определяет пять базовых предикатов: сравнение, BETWEEN, IN, LIKE и IS NULL.

Таблица 2

Базовые предикаты

Сравнение	Для сравнения одного типизированного выражения с другим используется один из шести операторов сравнения: (= , о, <, >, < = , >=).
Диапазон	Предикат BETWEEN позволяет тестировать, попадает ли указанное типизиро-

	ванное выражение в указанный диапазон значений. Диапазон определяется с помощью двух типизированных выражений, разделенных ключевым словом AND.
Принадлежность	Используя предикат IN, можно проверить, совпадает ли значение указанного типизированного выражения с элементом заданного списка значений.

Продолжение табл. 2

Поиск по шаблону	Предикат LIKE позволяет проверить, совпадает ли выражение типа “символьная строка” с указанным образцом символьной строки.
Null	Используйте предикат IS NULL для определения, равно ли типизированное выражение Null.

Пример сравнения:

```
SELECT ProductName
FROM Products
WHERE RetailPrice <= 50
```

Примеры диапазона:

```
SELECT FirstName, LastName FROM Staff WHERE DateHired BETWEEN '2015-01-01’ AND '2015-02-01'
```

```
SELECT StudLastName, StudFirstName, StudPhoneNumber FROM Students WHERE StudLastName BETWEEN ‘b’ AND ‘bz’
```

Примеры принадлежности множеству:

```
SELECT TourneyLocation FROM Tournaments WHERE TourneyDate IN (‘2014-06-05’, ‘2014-07-03', '2014-08-07’)
```

SELECT EntStageName FROM Entertainers WHERE EntCity IN (‘Seattle1, ‘Redmond’, ‘Bothell’)

5.3. Совпадение по шаблону (LIKE)

Условие совпадения с образцом является полезным, когда нужно найти значения, подобные указанной строке-образцу, или когда только неполная часть информации используется как критерий поиска.

В этом условии берется значение типизированного выражения и используется предикат LIKE для проверки, совпадает ли это значение с заданным образцом строки. Образец строки может состоять из любой логической комбинации обычных строк символов и двух специальных групповых символов: процента «%» и подчеркивания «_». Символ процента представляет ни одного или несколько произвольных обычных символов, а символ подчеркивания представляет собой единственный произвольный обычный символ. Способ определения образца строки устанавливает, какие значения извлекаются из типизированного выражения:

```
SELECT CustLastName, CustFirstName FROM Customers
WHERE CustLastName LIKE 'Mar%'

SELECT VendName FROM Vendors
WHERE VendStreetAddress LIKE ‘%Forest%’
```

5.4. Значение Null

Null не представляет собой нули, символьную строку из одного или нескольких пробелов или символьную строку нулевой длины (т. е. не содержащую в себе символов), потому что каждый из этих элементов может быть значащим во множестве

различных ситуаций. Null представляет собой пропущенное или неизвестное значение. Пример использования NULL:

```
SELECT CustFirstName FROM Customers WHERE
CustCounty IS NULL
```

5.5. Исключение строк (NOT)

Исключить строки из набора результатов можно с помощью оператора NOT. При включении оператора NOT оператор SELECT проигнорирует все строки, которые удовлетворяют условию, указанному любым из этих предикатов. В наборе результатов будут присутствовать те строки, которые не удовлетворяют условию.

Приведенные далее примеры показывают, как можно использовать NOT в качестве части условия поиска:

SELECT OrderlD, OrderDate FROM Orders WHERE OrderDate NOT BETWEEN ‘2014-07-01' AND ‘2014-07-31’

```
SELECT StaffID, Title FROM Faculty
WHERE Title NOT IN ('Professor', 'Associate Professor’)
```

Технология выполнения лабораторной работы

1. Запустите MySQL сервер.

Чтобы создать запрос вам нужно запустите Microsoft SQL Server Management Studio, далее, просто нажать кнопку New Querty (Новый запрос) на панели инструментов (рис. 1).

2. В окне SQL запроса в верхнем полу пишется текст запроса, а в нижнем - выводится таблица результата выполнения запроса (рис. 2). Чтобы запустить запрос на выполнение нужно нажать кнопку Execute на панели инструментов.

3. Выполните запросы своего варианта для построения базы данных и манипуляции данных.

Задание на лабораторную работу

1. Запустить SQL сервер.
2. Сформировать серию различных SQL-запросов выбора на основе своего варианта базы данных, с использованием:
- стандартной структуры запроса SELECT;
- исключения дубликатов DISTINCT;
- сортировки информации с помощью ORDER BY;
- условия WHERE;
- базовых предикатов фильтрации данных;
- работу со значением NULL;
- исключения строк с помощью NOT.
3. Показать результаты работы преподавателю.
4. Подготовить отчет по лабораторной работе.

Лабораторная работа № 3

«Работа с объединениями данных»

Цель работы

Изучение теоретических основ и получение практических навыков в области объединения данных.

Краткие теоретические сведения

1. Корреляционные имена AS

Стандарт SQL определяет способ назначения псевдонима (алиаса) — называемого в стандарте корреляционным именем — любой таблице, перечисленной в условии FROM. Эта возможность может быть очень удобной при построении сложных запросов с использованием таблиц, имеющих длинные описательные имена. Таблице можно назначить короткое корреляционное имя, чтобы облегчить явное указание столбцов в таблице с длинным именем.

Для того чтобы назначить корреляционное имя таблице, укажите после ее имени необязательное ключевое слово AS, а затем корреляционное имя, которое нужно присвоить. После назначения таблице корреляционного имени это имя используется вместо ее исходного имени во всех других условиях, включая условие SELECT, условия поиска в условиях ON и WHERE и условие ORDER BY. Это может немного сбить с толку, потому что обычно имеется склонность записывать условие SELECT до того, как записывается условие FROM. Если планируется назначить таблице алиас в условии FROM, то этот алиас следует использовать при уточнении имен столбцов в условии SELECT.

```
SELECT Name AS N1 FROM Person
```

В итоге получим таблицу 1:

Таблица 1

N1
Иванов
Петров
Сидоров

Как и для всех необязательных ключевых слов, AS рекомендуется использовать для того, чтобы облегчить чтение и понимание запроса.

2. SQL-запрос UNION

UNION позволяет выбрать с помощью SELECT **строки** из двух (или более) подобных наборов результата и объединить их в один набор. Запрос UNION чередует строки из одного набора результатов со строками из другого набора результатов. Каждый набор результатов определяется записью оператора SELECT. Затем они соединяются с ключевым словом UNION.

Пусть дана табл. 2 (payments2015):

Таблица 2

Сотрудник	Оклад
Иванов	5000
Петров	3000
Сидоров	7000

И дана табл. 3 (payments2014):

Таблица 3

Сотрудник	Оклад
Иванов	7000
Петров	3000
Сидоров	7000
Орлов	4000
Соколов	6000

При выполнении запроса

```
(SELECT * FROM payments2015)
UNION
(SELECT * FROM payments2014);
```

результат выполнения будет следующим (табл. 4):

Таблица 4

Сотрудник	Оклад
Иванов	5000
Иванов	7000
Петров	3000
Сидоров	7000
Орлов	4000
Соколов	6000

В результате отобразятся две строки с Ивановым, потому что эти строки различаются значениями в столбцах. При этом все остальные записи остались в прежнем виде, поскольку значения в столбцах полностью совпадают, повторения с одинаковыми значениями отсутствуют.

При добавлении ключевого слова ALL:

```
(SELECT * FROM payments2015)
UNION ALL
(SELECT * FROM payments2014);
```

будут возвращены все записи двух таблиц с повторениями (табл. 5):

Таблица 5

Сотрудник	Оклад
Иванов	5000
Иванов	3000
Петров	3000
Петров	3000
Сидоров	7000
Сидоров	7000
Орлов	4000
Соколов	6000

Имя столбцов наследуется от столбцов первой таблицы, выбранной для включения в выражение SELECT. Вместо появления их друг за другом данные из этих двух столбцов чередуются по вертикали.

Для выполнения UNION два набора результатов должны удовлетворять определенным требованиям. В первую очередь каждый из двух операторов SELECT, связываемых в UNION, должен иметь одинаковое количество выходных столбцов, определенных после ключевого слова SELECT, так что набор результата будет иметь то же самое количество столбцов. Во-вторых, каждый соответствующий столбец должен обладать свойством, которое в стандарте SQL называется “допускающим сопоставление”.

Допускается сравнение символьных значений только с символьными значениями, цифровых значений с цифровыми значениями, а значений дата/время со значениями дата/время.

Что касается сортировки результата UNION, то во многих СУБД полученный набор результатов уже отсортирован по выходным столбцам слева направо. Например, в UNION трех таблиц, только что построенном в предыдущем разделе, строки появятся в следующей последовательности: имя, улица и т. д.

Для сортировки строк по почтовому индексу можно добавить условие ORDER BY, но хитрость состоит в том, что это условие должно располагаться в самом конце последнего оператора SELECT. Условие ORDER BY применяется к результату UNION, а не к последнему оператору SELECT.

3. SQL-запрос JOIN

JOIN является операцией пересечения, но она отличается от остальных, потому что при ее выполнении от системы базы данных требуется присоединения только указанных столбцов. Таким образом, JOIN позволяет получить пересечение двух очень разнородных таблиц на основе совпадения значений столбцов. Например, можно воспользоваться JOIN для связывания Customers (клиенты) с их Orders (заказы) путем сопоставления CustomerID из таблиц в Customers с CustomerID в таблице Orders.

Запрос JOIN рассматривается как часть условия FROM в операторе SQL. JOIN определяет “логическую таблицу”, которая является результатом связывания двух таблиц или наборов результатов. Помещение JOIN в условие FROM определяет порядок связывания таблиц, из которых запрос извлекает окончательный набор результатов. Другими словами, JOIN заменяет имя отдельной таблицы. Можно также определить несколько операций JOIN для создания сложного набора результатов на основе более чем двух таблиц.

3.1. Запрос INNER JOIN

Стандарт SQL определяет несколько способов выполнения операций JOIN, наиболее обычным из которых является INNER JOIN. Предположим, вы связываете студентов и курсы лекций, на которые они записались. Может случиться так, что некоторые студенты, которые уже были приняты, еще не записались на какие-либо лекции. А может быть и так, что на некоторые курсы лекций, которые имеются в расписании, еще не записался ни один студент.

Операция INNER JOIN между таблицей Students и таблицей Classes возвращает строки Student, связанные с соответствующими строками Classes (через таблицу Student_Schedules) — но она не возвращает ни студентов, которые еще не зарегистрировались на какой-либо курс лекций, ни курсы лекций, на которые не записался ни один студент. INNER JOIN возвращает только те строки, где связывающие значения совпадают в обеих таблицах или в обоих наборах результатов.

Чаще всего связью, которую использует JOIN, является первичный ключ из одной таблицы и связанный внешний ключ из второй таблицы. Внешний ключ должен иметь тот же тип данных, что и связанный первичный ключ. Однако в JOIN также разрешается соединить две таблицы или два набора результатов по любым столбцам, которые имеют, как это называется в стандарте SQL, типы данных, “пригодные для соединения”.

В общем случае можно соединить один символьный столбец с другим символьным столбцом или выражением, столбец любого числового типа (например, целый) с любым столбцом другого числового типа (возможно, со значениями типа “с плавающей точкой”) и любой столбец типа “дата” с другим столбцом типа “дата”. Это позволит, например, соединять в JOIN строки из таблицы Customers со строками из таблицы Employees по столбцам City (Город) или Zip Code (Почтовый индекс) – возможно, чтобы узнать, кто из клиентов и сотрудников живут в одном и том же городе или в регионе с одинаковым почтовым индексом.

Рассмотрим запрос INNER JOIN на примере таблицы 6 – сотрудников (Person):

Таблица 6

IDPerson	Name	IDPost
1	Иванов	1
2	Петров	2
3	Сидоров	2
4	Попов	3

и таблицы 7 – должности (Post):

Таблица 7

IDPost	NamePost
1	Ведущий инженер
2	Инженер
3	Лаборант

При выполнении следующего запроса

```
SELECT * FROM Person
```

INNER JOIN Post ON Person.IDPost = Post.IDPost

в результате получится следующая таблица 8:

Таблица 8

IDPerson	Name	IDPost	IDPost	NamePost
1	Иванов	1	1	Ведущий инженер
2	Петров	2	2	Инженер
3	Сидоров	2	2	Инженер
4	Попов	3	3	Лаборант

Тело результата логически формируется следующим образом. Каждая строка одной таблицы сопоставляется с каждой строкой второй таблицы, после чего для полученной «соединённой» строки проверяется условие соединения (вычисляется предикат соединения). Если условие истинно, в таблицу-результат добавляется соответствующая «соединённая» строка.

Если совпадающие столбцы в двух таблицах имеют одинаковые имена и требуется только соединить равные значения, воспользуйтесь условием USING и перечислите имена столбцов. Решим предыдущую задачу снова, на этот раз с использованием USING:

```
SELECT * FROM Person
INNER JOIN Post USING (IDPost)
```

Некоторые системы баз данных пока еще не поддерживают использование USING. Однако всегда можно получить тот же результат в условии ON и условии сравнения на равенство.

СУБД логически создает комбинацию каждой строки из первой таблицы с каждой строкой из второй таблицы, а затем применяет условие ON или USING. Это воспринимается как большой объем дополнительной работы для базы данных: вначале построить все комбинации, а затем отфильтровать потенциально несколько строк, удовлетворяющих условиям.

3.2. Запрос OUTER JOIN

Стандарт SQL определяет несколько типов операций JOIN для связи двух или более таблиц или наборов результатов. При выполнении операции OUTER JOIN база данных должна возвратить не только строки, совпадающие с определенным критерием, но также строки, не удовлетворяющие данному критерию, из одного либо из обоих множеств, которые нужно связать.

Предположим, например, что нужно извлечь информацию о студентах и курсах лекций, на которые они записались, из базы данных расписания занятий. Операция INNER JOIN возвращает имена только тех студентов, которые записались на некоторый курс лекций, и список лекций, на которые записались студенты. Эта операция не возвращает имена студентов, которые уже приняты в колледж, но еще не записались на какие-либо курсы, и не возвращает курсы лекций, которые есть в расписании, но к которым пока еще студенты не проявили интереса.

А что если нужен список всех студентов и курсов лекций, на которые они записались, если таковые курсы имеются? Или наоборот, нужен список всех курсов лекций и имена студентов, которые записались на эти лекции, если имеются такие студенты. Для решения задач такого вида нужно использовать OUTER JOIN.

Обычно используется такой вид OUTER JOIN, в котором запрашиваются все строки из одной таблицы или набора ре-

зультатов и все строки, удовлетворяющие условию, из второй таблицы или набора результатов. Для этого определяется либо LEFT OUTER JOIN, либо RIGHT OUTER JOIN.

Рассмотрим OUTER JOIN на примерах. Вернемся к предыдущему примеру с сотрудниками. Предположим, что в таблицу был добавлен новый сотрудник, но должность ему еще не была назначена. Если мы использовали бы INNER JOIN, то новый сотрудник не был бы возвращен в результирующую таблицу. Однако, использование запроса LEFT OUTER JOIN предписывает компилятору выводить всё содержимое первой таблицы (в качестве первой таблицы мы укажем таблицу сотрудников):

```
SELECT * FROM Person
LEFT OUTER JOIN Post ON Person.IDPost = Post.IDPost
```

В результате получим следующую таблицу 9:

Таблица 9

IDPerson	Name	IDPost	IDPost	NamePost
1	Иванов	1	1	Ведущий инженер
2	Петров	2	2	Инженер
3	Сидоров	2	2	Инженер
4	Попов	4	4	Лаборант
5	Орлов	0	NULL	NULL

Рассмотрим другой пример. Предположим, мы ввели новую должность, но никому из сотрудников она еще не была назначена. Применяя следующий запрос:

```
SELECT * FROM Person
RIGHT OUTER JOIN Post ON Person.IDPost = Post.IDPost
```

получим результирующую таблицу 10:

Таблица 10

IDPerson	Name	IDPost	IDPost	NamePost
1	Иванов	1	1	Ведущий инженер
2	Петров	2	2	Инженер
3	Сидоров	2	2	Инженер
4	Попов	4	4	Лаборант
NULL	NULL	NULL	5	Рабочий

FULL OUTER JOIN не является ни “левым”, ни “правым” — оно является двусторонним! В него включаются все строки из обеих таблиц или наборов результатов, используемых в JOIN. Когда отсутствуют строки, выполняющие условие, с левой стороны JOIN, то в наборе результата “справа” будут присутствовать значения Null. НаоборсУт, когда отсутствуют строки, выполняющие условие, с правой стороны JOIN, то значения Null будут присутствовать в наборе результата “слева”.

```
SELECT * FROM Person
RIGHT OUTER JOIN Post ON Person.IDPost = Post.IDPost
```

получим результирующую таблицу 11:

Таблица 11

IDPerson	Name	IDPost	IDPost	NamePost
1	Иванов	1	1	Ведущий инженер
2	Петров	2	2	Инженер
3	Сидоров	2	2	Инженер
4	Попов	4	4	Лаборант
5	Орлов	0	NULL	NULL
NULL	NULL	NULL	5	Рабочий

При необходимости можно использовать тип соединения FULL OUTER JOIN вместе с другими операциями INNER и OUTER JOIN для получения правильного ответа. Были даны краткие пояснения варианта FULL OUTER JOIN — UNION JOIN.

С помощью OUTER JOIN можно решить самые разнообразные запросы. Мы привели почти дюжину примеров использования OUTER JOIN и показали логику, положенную в основу построения оператора, являющегося решением для каждого запроса.

Технология выполнения лабораторной работы

1. Запустите MySQL сервер.

Чтобы создать запрос вам нужно запустите Microsoft SQL Server Management Studio, далее, просто нажать кнопку New Querty (Новый запрос) на панели инструментов (рис. 1).

2. В окне SQL запроса в верхнем полу пишется текст запроса, а в нижнем - выводится таблица результата выполнения запроса (рис. 2). Чтобы запустить запрос на выполнение нужно нажать кнопку Execute на панели инструментов.

3. Выполните запросы своего варианта для построения базы данных и манипуляции данных.

Задание на лабораторную работу

1. Запустить SQL сервер.

2. Сформировать серию различных SQL-запросов выбора на основе своего варианта базы данных, с использованием:

- корреляционного имени AS;
- несколько запросов UNION;
- несколько запросов с использованием INNER JOIN;
- несколько запросов с использованием LEFT JOIN;
- несколько запросов с использованием RIGHT JOIN;
- несколько запросов с использованием FULL JOIN.

3. Показать результаты работы преподавателю.

4. Подготовить отчет по лабораторной работе.

Лабораторная работа № 4

«Подзапросы языка SQL»

Цель работы

Изучение теоретических основ и получение практических навыков в области создания подзапросов языка SQL.

Краткие теоретические сведения

1. Подзапросы

Говоря простым языком, подзапрос — это выражение SELECT, которое вложено в одно из условий оператора SELECT для образования оператора окончательного запроса.

Стандарт SQL определяет три типа подзапросов:

1. Строковый подзапрос — вложенное выражение SELECT, возвращающее больше одного столбца и только одну строку.
2. Табличный подзапрос — вложенное выражение SELECT, возвращающее один или несколько столбцов и ни одной или несколько строк.
3. Скалярный подзапрос — вложенное выражение SELECT, возвращающее только один столбец и не более одной строки.

2. Стандартные подзапросы

2.1. Скалярный подзапрос

Построим запрос, который выдает список названия и цены, где цена равна стоимости конкретного заголовка (в данном случае «Straight Talk About Computers»).

```
SELECT title, price
FROM titles
WHERE price =
  (SELECT price
```

```
FROM titles
WHERE title = "Straight Talk About Computers")
```

2.2. Табличный подзапрос

Теперь построим запрос, который выдает список заказов для конкретной даты и выбирает фамилию соответствующего клиента из таблицы Customers, используя подзапрос: «Выбрать номер заказа, дату заказа, дату отгрузки, (выбрать фамилию клиента из “Клиенты” и таблицы “Заказы”), где дата отгрузки = 24 декабря 2014».

```
SELECT Orders.OrderNumber, Orders.OrderDate,
Orders.ShippedDate,
(SELECT Customers.CustLastName FROM Customers
WHERE Customers.CustomerlD = Orders.CustomerlD)
FROM Orders
WHERE Orders.ShippedDate = ‘2014-12-24’
```

Нам потребовалось ограничить значение идентификатора клиента (Customer ID) в подзапросе значением идентификатора клиента в каждой строке, извлекаемой из таблицы Orders. В ином случае в подзапросе будут получены все строки из Customers. Помните, что это должен быть скалярный подзапрос, поэтому нужно что-то предпринять для ограничения результата, чтобы возвращалось не более одной строки.

Вспомнив INNER JOIN, вы можете спросить, почему бы не решить эту задачу так, как уже описано, вместо того чтобы объединять Orders в JOIN с Customers в условии FROM внешнего запроса. На самом деле именно сейчас наше внимание сосредоточивается на концепции использования подзапросов для создания столбца вывода в очень простом примере, и действительно, возможно, следует решить эту конкретную задачу, используя INNER JOIN:

SELECT Orders.OrderNumber, Orders.OrderDate, Orders.ShippedDate, Customers.CustLastName FROM Customers INNER JOIN Orders ON Customers.CustomerlD = Orders.OrderlD WHERE Orders.ShippedDate = ‘1999-12-24’

3. Агрегатные функции COUNT и MAX

Стандарт SQL определяет множество функций, которые вычисляют значения в запросе. Один из подклассов функций — агрегатные функции — позволяет вычислять одно значение для группы строк в наборе результатов. Например, агрегатной функцией можно воспользоваться для подсчета строк, поиска наибольшего или наименьшего значения для некоторого множества строк или для вычисления среднего или общего значения или выражения по набору результата.

Функция COUNT может использоваться для определения количества строк или отличных от Null значений в наборе результата. COUNT(*) используется для определения количества строк во всем множестве. Если в наборе результатов конкретный столбец определяется с использованием COUNT (column_name), то СУБД высчитывает количество строк, в которых значение этого столбца не равно Null. Можно также запросить подсчитать только уникальные значения, добавив ключевое слово DISTINCT.

Подобным образом можно найти наибольшее значение в столбце, используя функцию MAX. Если типизированное выражение является числовым, то наибольшее число будет получено из определенного вами столбца или выражения. Если же типизированное выражение возвращает данные символьного типа, то наибольшее значение будет зависеть от сортирующей последовательности, используемой системой базы данных.

Пример запроса (1): «Выбрать имя клиента, фамилию клиента (Выбрать количество (*) из “Заказы”, где идентификатор клиента = customerlD) из “Клиенты”»:

SELECT Customers.CustFirstName, Customers.CustLastName,
(SELECT C0UNT(*)
FROM Orders
WHERE Orders.CustomerlD = Customers.CustomerlD)
AS CountOfOrders FROM Customers

Пример запроса (2): «Выбрать имя клиента, фамилию клиента и (Выбрать шах(дата заказа) из “Заказы”, где идентификатор клиента = customerlD) из “Клиенты”»:

SELECT Customers.CustFirstName,
Customers.CustLastName, (SELECT MAX(OrderDate) FROM Orders
WHERE Orders.CustomerlD = Customers.CustomerlD) AS LastOrderDate FROM Customers

4. Множество IN

Ключевое слово IN используется в условии WHERE для сравнения столбца или выражения со списком значений. Каждое типизированное выражение в списке IN может быть скалярным подзапросом.

В случае использования подзапроса для формирования всего списка можно использовать табличный подзапрос, который возвращает один столбец и столько строк, сколько необходимо для построения списка.

SELECT SUM(Quantity) AS Qty FROM Sumproduct WHERE City IN (SELECT City FROM Sellers WHERE Country = 'Russia')

Приведенный выше пример демонстрирует суммирование некоторого числового поля Quantity по всем записям, полученным в результате применения подзапроса. Так как результат

подзапроса является табличным (в данном случае одно поле и несколько записей) следует использовать предикат IN.

5. Предикаты ALL / SOME / ANY

Предикат IN позволяет сравнивать столбец или выражение со списком, чтобы определить, содержится ли этот столбец или выражение в списке (IN). Другими словами, столбец или выражение равно одному из элементов списка. Если нужно установить, является ли столбец или выражение больше или меньше, чем любое, все или некоторые из элементов в списке, то можно использовать количественный предикат.

В данном случае выражение SELECT должно быть табличным подзапросом, который возвращает только один столбец и ни одной или несколько строк. Когда подзапрос возвращает больше одной строки, значения в строках составляют список. Этот предикат объединяет оператор сравнения с ключевым словом, которое указывает системе базы данных, как этот оператор применяется к элементам списка. При использовании ключевого слова ALL результат сравнения должен иметь значение True для всех значений, возвращенных подзапросом. При использовании ключевого слова SOME или ANY результат сравнения должен быть True хотя бы для одного значения в списке.

Когда подзапрос возвращает несколько строк, запрос для = ALL всегда будет False, если только все значения, возвращенные подзапросом, не являются одинаковыми и типизированное выражение в левой части сравнения равно каждому из них. Согласно этой же логике можно подумать, что < > ANY всегда будет False, если типизированное выражение в левой части всегда равно любому значению в списке.

На самом деле стандарт SQL интерпретирует SOME и ANY как одно и то же. Поэтому, если указать < > SOME или < > ANY, этот предикат будет True, если типизированное выражение слева не равно по крайней мере одному из значений в

списке. Другим смущающим моментом является то, что если подзапрос не возвращает строк, то любой предикат сравнения с ключевым словом ALL является True и любой предикат сравнения с ключевыми словами SOME или ANY является False.

Указание в запросе «>ALL» означает больше, чем любое значение или, что равносильно, больше максимальной величины. Например, «>ALL (1,2,3)» означает больше чем 3.

Указание в запросе «>ANY» означает больше, по крайней мере, одного значения или, что равносильно, больше минимальной величины. Поэтому «>ANY (1,2,3)» означает больше 1.

Квантор «=ALL» означает равенство каждому возвращаемому значению.

Квантор «<>ALL» эквивалентен условию NOT IN.

Квантор «=ANY» означает проверку существования, поэтому он эквивалентен условию IN.

Квантор «<>ANY» означает «не а или не в или не с». При этом условие NOT IN означает «не а и не в и не с».

```
SELECT title
FROM titles
WHERE advance > ALL
  (SELECT advance
   FROM publishers, titles
   WHERE titles.pub_id = publishers.pub_id
   AND pub_name = "New Age Books")
```

6. Существование EXISTS

Как принадлежность к множеству (IN), так и количественно определенные предикаты (SOME/ANY/ALL) выполняют сравнение с типизированным выражением — обычно это столбец из источника, определенного в условии FROM внешнего запроса. Иногда полезно просто знать, что интересующая

строка EXISTS (существует) в наборе результатов, возвращенном подзапросом.

Подзапрос не возвращает никаких данных, а вместо этого возвращает логическое значение TRUE (истина) или FALSE (ложь).

Следующий запрос находит названия всех издательств, которые публиковали книги по бизнесу:

```
SELECT pub_name
FROM publishers
WHERE exists
  (SELECT *
   FROM titles
   WHERE pub_id = publishers.pub_id
   AND type = "business")
```

7. Подзапросы как фильтры

Вернемся к предыдущему оператору SELECT и рассмотрим синтаксис построения запроса с простым предикатом сравнения в условии WHERE. Решим простую задачу, требующую сравнения со значением, возвращенным из подзапроса. В этом примере предполагается запросить все детали о заказах клиентов, но нас интересует только последний заказ каждого клиента.

Пример фильтрации.

Запрос: «Выбрать имя клиента, фамилию клиента, номер заказа, дату заказа, номер товара, наименование товара и заказанное количество из таблицы “Клиенты”, соединенной с таблицей “Заказы” по идентификатору клиента, затем соединенной с таблицей “Детали заказа” по номеру заказа, затем соединенной с таблицей “Товары” по номеру товара, где дата заказа равна максимальной дате заказа из таблицы “Заказы” для этого клиента».

Данный запрос можно уточнить следующим образом: «Выбрать имя клиента, фамилию клиента, номер заказа, дату заказа, номер товара, наименование товара, заказанное количество из “Клиенты”, соединенной с “Заказы” по идентификатору клиента, соединенной с “Детали заказа” по номеру заказа, соединенной с “Товары” по номеру товара где дата заказа = (Выбрать MAX(дата заказа) из “Заказы”, где orders.customerlD = customers.customerlD)».

Получим код SQL для данного запроса:

```
SELECT Customers.CustFirstName,
Customers.CustLastName, Orders.OrderNumber, Orders.OrderDate, Order_Details.ProductNumber, Products.ProductName,
Order_Details.QuantityOrdered
FROM ((Customers INNER JOIN Orders
ON Customers.CustomsrID = Orders.CustomerlD)
INNER JOIN Order.Details
ON Orders,OrderlD = Order_Details.OrderlD)
INNER JOIN Products
ON Products.ProductNumber =
Order_Details.ProductNumber WHERE Orders.OrderDate =
(SELECT MAX(OrderDate)
FROM Orders AS 02
WHERE 02.CustomerlD = Customers.CustomerlD)
```

Технология выполнения лабораторной работы

1. Запустите MySQL сервер.

Чтобы создать запрос вам нужно запустите Microsoft SQL Server Management Studio, далее, просто нажать кнопку New Querty (Новый запрос) на панели инструментов (рис. 1).

2. В окне SQL запроса в верхнем полу пишется текст запроса, а в нижнем - выводится таблица результата выполнения запроса (рис. 2). Чтобы запустить запрос на выполнение нужно нажать кнопку Execute на панели инструментов.

3. Выполните запросы своего варианта для построения базы данных и манипуляции данных.

Задание на лабораторную работу

1. Запустить SQL сервер.
2. Сформировать серию различных SQL-запросов выбора на основе своего варианта базы данных, с использованием:
- скалярного подзапроса;
- табличного подзапроса;
- агрегатных функций COUNT и MAX;
- условия IN;
- предикатов ALL и ANY;
- предиката EXISTS.
3. Показать результаты работы преподавателю.
4. Подготовить отчет по лабораторной работе.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Карабутов Н.Н. Методы построения и анализа информационного обеспечения в корпоративных системах. Введение в анализ данных: учеб. пособие / Н.Н. Карабутов; Сиб. гос. индустр. ун-т. – Новокузнецк: Изд. центр СибГИУ, 2011. – 377 с.

2. Зудилова Т.В. Создание запросов в Microsoft SQL Server 2008 / Т.В. Зудилова, Г.Ю. Шмелева. – СПб: НИУ ИТМО, 2013. – 149 с.

3. Майкл Дж. Хернандес, Джон Л. Вьескас. SQL-запросы для простых смертных: практическое руководство по манипулированию данными в SQL / пер. А. Головко, научн. ред. А. Киселева. М.: Изд-во «Лори», 2003. – 473 с.

4. Кузнецов М.В. Самоучитель MySQL 5 / М.В. Кузнецов, И.В. Симдянов. – СПб: БХВ-Петербург, 2007. – 560 с.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам № 1–4 по дисциплине «Базы данных» для студентов направления 09.03.01 «Информатика и вычислительная техника» очной формы обучения, по дисциплине «Управление данными» для студентов направления 09.03.02 «Информационные системы и технологии» очной формы обучения

Составители:
Яскевич Ольга Георгиевна
Иванов Денис Вячеславович

В авторской редакции

Компьютерный набор Д.В. Иванова

Подписано к изданию 16.09.2015.
Уч.- изд. л. 2,7.

ФГБОУ ВПО «Воронежский государственный
технический университет»
394026 Воронеж, Московский просп., 14