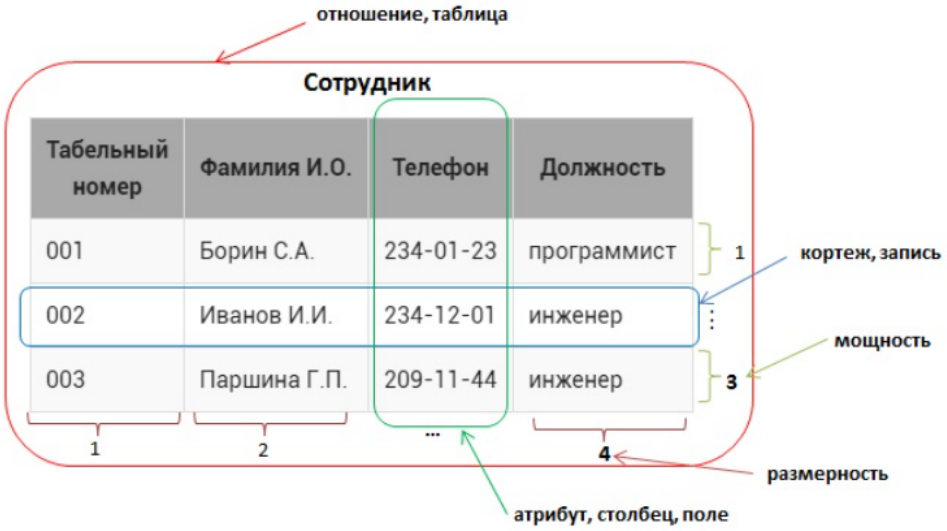
БД и SQL

Реляционная модель была разработана в конце 1960-х годов Е.Ф.Коддом . Она определяет способ представления данных (структуру данных), методы защиты данных (целостность данных), и операции, которые можно выполнять с данными (манипулирование данными). Эта модель лежит в основе всех реляционных баз данных до настоящего времени.

Основные принципы реляционных баз данных:

- все данные на концептуальном уровне представляются в виде объектов, заданных в виде строк и столбцов, называемых отношением, более распространенное название – таблица;
- в пересечение строки и столбца таблицы можно занести только одно значение;
- все операции выполняются над целыми отношениями и результатом этих операций является отношение.



На примере таблицы **Сотрудник** рассмотрим **терминологию реляционных баз данных:**

- ***отношение*** – это структура данных целиком, набор записей (в обычном понимании – таблица) , в примере –это `Сотрудник`;
- ***кортеж*** – это каждая строка , содержащая данные (более распространенный термин – запись), например, <00*1, Борин С.А, 234-01-23, программист*>, все кортежи в отношении должны быть различны;
- ***мощность*** – число кортежей в таблице (проще говоря, число записей), в данном случае 3, мощность отношения может быть любой (от 0 до бесконечности), порядок следования кортежей - неважен;
- ***атрибут*** – это столбец в таблице (более распространенный термин – поле), в примере – `Табельный номер, Фамилия И.О., Телефон, Должность`)
- ***размерность*** – это число атрибутов в таблице, в данном случае – 4;
- размерность отношения должна быть больше 0, порядок следования атрибутов существенен;
- ***домен атрибута*** – это допустимые значения (неповторяющиеся), которые можно занести в поле , например для атрибута `Должность` домен – {инженер, программист}.

Основные типы данных SQL:

Тип данных	Описание	Пример
`INT` `INTEGER`	Целое число, могут принимать значения от -2 147 483 648 до 2 147 483 647	-5671205
`DECIMAL` `NUMERIC`	Вещественное число, в скобках указывается максимальная длина числа (включает символы слева и справа от десятичной запятой)и количество знаков после запятой. Можно использовать оба этих типа, они эквивалентны, принимают значения в диапазоне -1038+1 до 1038-1.`DECIMAL(4,1) NUMERIC(6,3)`	34.6-3.294
`DATE`		

	Дата в формате ГГГГ-ММ-ДД *26 июля 2020 года3 января 2021 года*	2020-07-262021-01-03
`VARCHAR`	Строка длиной не более 255 символов, в скобках указывается максимальная длина строки, которая может храниться в поле `VARCHAR(10)(рассматриваются однобайтовые кодировки,для которых число в скобках соответствует максимальномуколичеству символов в строке)`	пример описание

Определим тип данных для каждого поля таблицы `book`:

book_id	title	author	price	amount
1	Мастер и Маргарита	Булгаков М.А.	670.99	3
2	Белая гвардия	Булгаков М.А.	540.50	5
3	Идиот	Достоевский Ф.М.	460	10
4	Братья Карамазовы	Достоевский Ф.М.	799.01	2

- `book_id` - ключевой столбец, целое число, которое должно генерироваться автоматически - `INT PRIMARY KEY AUTO_INCREMENT`;
- `title` - строка текста, ее длина выбирается в зависимости от данных, которые предполагается хранить в поле, предположим, что название книги не превышает 50 символов - `VARCHAR(50)`;
- `author` - строка текста - `VARCHAR(30)`;
- `price` - для описание денежного значения используется числовой тип данных с двумя знаками после запятой - `DECIMAL(8,2)`;
- `amount` - целое число - `INT`.

Создание таблицы

Для создания таблицы используется SQL-запрос. В нем указывается какая таблица создается, из каких атрибутов(полей) она состоит и какой тип данных

имеет каждое поле, при необходимости указывается описание полей (ключевое поле и т.д.). Его структура :

- ключевые слова : `CREATE TABLE`
- имя создаваемой таблицы;
- открывающая круглая скобка «(»;
- название поля и его описание, которое включает тип поля и другие необязательные характеристики;
- запятая;
- название поля и его описание;
- ...
- закрывающая скобка «)».

Пример. Создадим таблицу `genre` следующей структуры:

Поле	Тип, описание
genre_id	`INT PRIMARY KEY AUTO_INCREMENT`
name_genre	`VARCHAR(30)`

Запрос:

```
CREATE TABLE genre(
    genre_id INT PRIMARY KEY AUTO_INCREMENT,
    name_genre VARCHAR(30)
);
```

Созданная таблица - пустая.

БД

1. **Внутренний ключ** формируется при использовании суррогатного ключа - автоматически сгенерированного значения, никак не связанного с информационным содержанием записи, имеющего некий физический смысл только внутри БД

2. **Вторичный ключ** устанавливается по полям, которые часто используются при поиске или сортировке данных. Могут содержать не уникальные значения. Использование вторичных ключей, в большинстве случаев, увеличивает производительность СУБД.
3. **Первичный ключ** - поле, которое используется для обеспечения уникальности данных в таблице
4. **Внешний ключ** - это одно или несколько полей (атрибутов), которые являются первичными в другой таблице и значение которых заменяется значениями первичного ключа другой таблицы

Наиболее часто SQL используется для формирования выгрузок, витрин (с последующим построением отчетов на основе этих витрин) и администрирования баз данных.

Ссылка на :

Основные запросы SQL

Структура sql-запросов

Общая структура запроса выглядит следующим образом:

```
SELECT ('столбцы или * для выбора всех столбцов; обязательно')
FROM ('таблица; обязательно')
WHERE ('условие/фильтрация, например, city = 'Moscow'; необязательно')
GROUPBY ('столбец, по которому хотим сгруппировать данные; необязательно')
HAVING ('условие/фильтрация на уровне сгруппированных данных; необязательно')
ORDERBY ('столбец, по которому хотим отсортировать вывод; необязательно')
```

SELECT, FROM

SELECT, FROM — обязательные элементы запроса, которые определяют выбранные столбцы, их порядок и источник данных.

Выбрать все (обозначается как *) из таблицы Customers:

```
SELECT *FROM Customers
```

Выбрать столбцы CustomerID, CustomerName из таблицы Customers:

```
SELECT CustomerID, CustomerName.  FROM Customers
```

Если требуется получить только уникальные строки (скажем, нас интересуют только различные комбинации скорости процессора и объема памяти, а не характеристики всех имеющихся компьютеров), то можно использовать ключевое слово **DISTINCT**
:

SELECT DISTINCT speed**,** ram

FROM PC**;**

Помимо **DISTINCT**
может применяться также ключевое слово **ALL**
(все строки), которое принимается по умолчанию.

WHERE

WHERE — необязательный элемент запроса, который используется, когда нужно отфильтровать данные по нужному условию. Очень часто внутри элемента where используются IN / NOT IN для фильтрации столбца по нескольким значениям, AND / OR для фильтрации таблицы по нескольким столбцам.

Фильтрация по одному условию и одному значению:

```
select *from Customers
WHERE City = 'London'
```

Фильтрация по одному условию и нескольким значениям с применением IN (включение) или NOT IN (исключение):

```
select *from Customers
where CityIN ('London', 'Berlin')
```

```
select *from Customers
where CityNOTIN ('Madrid', 'Berlin','Bern')
```

Фильтрация по нескольким условиям с применением AND (выполняются все условия) или OR (выполняется хотя бы одно условие) и нескольким значениям:

```
select *from Customers
where Country = 'Germany'AND Citynotin ('Berlin', 'Aachen')AND CustomerID > 15
```

Предика́т — это утверждение, высказанное о субъекте. Субъектом высказывания называется то, о чём делается утверждение.

Горизонтальную выборку реализует предложение **WHERE** предикат, которое записывается после предложения **FROM**. При этом в результирующий набор попадут только те строки из источника записей, для каждой из которых значение предиката равно **TRUE**. То есть предикат проверяется для каждой записи. Например, запрос «получить информацию о частоте процессора и объеме оперативной памяти для компьютеров с ценой ниже $500» можно сформулировать следующим образом:

SELECT DISTINCT speed**,** ram

FROM PC

WHERE price **<** 500

ORDER BY 2 **DESC;**

speed	Ram
450	64
450	32

speed	Ram
500	32

Примеры простых предикатов сравнения:

предикат	**описание**
price < 1000	Цена меньше 1000
type = ‘laptop’	Типом продукции является портативный компьютер
cd = ‘24x’	24-скоростной CD-ROM
color <> ’y’	Не цветной принтер
ram – 128 > 0	Объем оперативной памяти свыше 128 Мбайт
Price <= speed*2	Цена не превышает удвоенной частоты процессора

GROUP BY

GROUP BY — необязательный элемент запроса, с помощью которого можно задать агрегацию по нужному столбцу (например, если нужно узнать какое количество клиентов живет в каждом из городов).

При использовании GROUP BY обязательно:

1. перечень столбцов, по которым делается разрез, был одинаковым внутри SELECT и внутри GROUP BY,
2. агрегатные функции (SUM, AVG, COUNT, MAX, MIN) должны быть также указаны внутри SELECT с указанием столбца, к которому такая функция применяется.

Группировка количества клиентов по городу:

```
select City,count(CustomerID)from Customers
GROUPBY City
```

Группировка продаж с фильтрацией исходной таблицы. В данном случае на выходе будет таблица с количеством клиентов по городам Германии:

```
select City,count(CustomerID)from Customers
WHERE Country = 'Germany'
GROUPBY City
```

Переименование столбца с агрегацией с помощью оператора AS. По умолчанию название столбца с агрегацией равно примененной агрегатной функции, что далее может быть не очень удобно для восприятия.

```
select City,count(CustomerID)AS Number_of_clientsfrom Customers
groupby City
```

HAVING

HAVING — необязательный элемент запроса, который отвечает за фильтрацию на уровне сгруппированных данных (по сути, WHERE, но только на уровень выше).

Фильтрация агрегированной таблицы с количеством клиентов по городам, в данном случае оставляем в выгрузке только те города, в которых не менее 5 клиентов:

```
select City,count(CustomerID)from Customers
groupby City
HAVINGcount(CustomerID) >= 5
```

В случае с переименованным столбцом внутри HAVING можно указать как и саму агрегирующую конструкцию count(CustomerID), так и новое название столбца number_of_clients:

```
select City,count(CustomerID)as number_of_clientsfrom Customers
groupby City
HAVING number_of_clients >= 5
```

ORDER BY

ORDER BY — необязательный элемент запроса, который отвечает за сортировку таблицы.

Простой пример сортировки по одному столбцу. В данном запросе осуществляется сортировка по городу, который указал клиент:

```
select *from Customers
ORDERBY City
```

Осуществлять сортировку можно и по нескольким столбцам, в этом случае сортировка происходит по порядку указанных столбцов:

```
select *from Customers
ORDERBY Country, City
```

Сортировку можно проводить по возрастанию (параметр **ASC** принимается по умолчанию) или по убыванию (параметр **DESC**).

По умолчанию сортировка происходит по возрастанию для чисел и в алфавитном порядке для текстовых значений.

Если нужна обратная сортировка, то в конструкции ORDER BY после названия столбца надо добавить DESC:

```
select *from Customers
orderby CustomerIDDESC
```

1. Использование функции CONVERT

При этом способе мы преобразуем дату к текстовому представлению в формате "mm-dd"

SELECT CONVERT(CHAR(5**), date,** 110**)** "mm-dd"

FROM Battles**;**

по которому и выполним сортировку:

SELECT date

FROM Battles

ORDER BY CONVERT(CHAR(5**),date,**110**);**

date
1941-05-25 00:00:00.000
1962-10-20 00:00:00.000
1962-10-25 00:00:00.000
1944-10-25 00:00:00.000
1942-11-15 00:00:00.000
1943-12-26 00:00:00.000

2. Использование функций MONTH и DAY

Здесь мы используем встроенные функции, которые возвращают компоненты даты - месяц (MONTH) и день (DAY) соответственно. По этим компонентам выполним сортировку:

SELECT date

FROM Battles

ORDER BY MONTH(date), DAY(date);

Что касается производительности, то вы можете выбрать любой вариант, т.к. оптимизатор строит для них идентичные планы.

В заключение представим последний запрос в более наглядном виде, добавив в него еще и "виновника торжества":

SELECT DAY(date) BD_day**, DATENAME(**mm**, date)** BD_month**,** name

FROM Battles

ORDER BY MONTH(date), DAY(date);

JOIN

JOIN — необязательный элемент, используется для объединения таблиц по ключу, который присутствует в обеих таблицах. Перед ключом ставится оператор ON.

Запрос, в котором соединяем таблицы Order и Customer по ключу CustomerID, при этом перед названиям столбца ключа добавляется название таблицы через точку:

```
select *from Orders
JOIN CustomersON Orders.CustomerID = Customers.CustomerID
```

Нередко может возникать ситуация, когда надо промэппить одну таблицу значениями из другой. В зависимости от задачи, могут использоваться разные типы присоединений. INNER JOIN — пересечение, RIGHT/LEFT JOIN для мэппинга одной таблицы знаениями из другой, FULL JOIN - полное соединение.

```
select *from Orders
join Customerson Orders.CustomerID = Customers.CustomerID
where Customers.CustomerID >10
```

Предикаты представляют собой выражения, принимающие истинностное значение. Они могут представлять собой как одно выражение, так и любую комбинацию из неограниченного количества выражений, построенную с помощью булевых операторов **AND**, **OR** или **NOT**. Кроме того, в этих комбинациях может использоваться SQL-оператор **IS**, а также круглые скобки для конкретизации порядка выполнения операций.

Предикат в языке SQL

может принимать одно из трех значений **TRUE** (истина), **FALSE** (ложь) или **UNKNOWN** (неизвестно).

Исключение, которые не могут принимать значение **UNKNOWN,** составляют следующие предикаты:

- **IS NULL** (отсутствие значения),
- **EXISTS** (существование),

- **UNIQUE** (уникальность)
- **MATCH** (совпадение).

Правила комбинирования всех трех истинностных значений легче запомнить, обозначив

- **TRUE** как **1**,
- **FALSE** как **0**
- **UNKNOWN** как **1/2** (где-то между истинным и ложным значениями).

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

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

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

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

- модели, не являющиеся ПК
- второй предикат ничего не меняет, т.к. он добавляет условие,
- уже учтенное в первом предикате

SELECT maker**,** model**, type**

FROM Product

WHERE NOT type='PC' **OR type=**'Printer'**;**

- модели производителя A, которые не являются ПК

SELECT maker**,** model**, type**

FROM Product

WHERE NOT type='PC' **AND** maker**=**'A'**;**

Поменять порядок выполнения логических операторов можно при помощи скобок:

- модели, не являющиеся ПК или принтером, т.е. модели ноутбуков в нашем случае

SELECT maker**,** model**, type**

FROM Product

WHERE NOT (type='PC' **OR type=**'Printer'**);**

- модели, которые не являются ПК, выпускаемыми производителем A

SELECT maker**,** model**, type**

FROM Product

WHERE NOT (type='PC' **AND** maker**=**'A'**);**

Следующий приоритет имеет оператор AND. Сравните результаты следующих запросов.

- модели ПК, выпускаемые производителем A, и любые модели производителя B

SELECT maker**,** model**, type**

FROM Product

WHERE type='PC' **AND** maker**=**'A' **OR** maker**=**'B'**;**

- модели ПК, выпускаемые производителем A или производителем B

SELECT maker**,** model**, type**

FROM Product

WHERE type='PC' **AND (**maker**=**'A' **OR** maker**=**'B'**);**

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

Если cond1 и cond2 являются простыми условиями, то ограничение по предикату cond1 **AND** cond2 эквивалентно пересечению ограничений по каждому из предикатов.

Ограничение по предикату cond1 **OR** cond2 эквивалентно объединению ограничений по каждому из предикатов .

Ограничение по предикату **NOT** cond1 эквивалентно взятию разности, когда от исходного отношения вычитается ограничение по предикату cond1.

Предикаты сравнения

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

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

Данные типа **CHARACTER STRING** (символьные строки) сравниваются в соответствии с их алфавитной последовательностью. Если a1a2…an и в1 в…вn — две последовательности символов, то первая «меньше» второй, если a1 < в1, или a1 = в1 и a2 < в2 и т. д. Считается также, что a1a2…an < в1в2…вm, если n < m и a1a2…an = в1в2…вn, то есть если первая строка является префиксом второй. Например, ‘folder’ < ‘for’, так как первые две буквы этих строк совпадают, а третья буква строки ‘folder’ предшествует третьей букве строки ‘for’. Также справедливо неравенство ‘bar’ < ‘barber’, поскольку первая строка является префиксом второй.

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

Предикат BETWEEN

Синтаксис:

BETWEEN::=

<Пр**о**веряе**мо**е выраже**н**ие**> [NOT] BETWEEN**

<Нача**ль**н**о**е выраже**н**ие**> AND <**К**он**еч**но**е выраже**н**ие**>**

Предикат **BETWEEN** проверяет, попадают ли значения проверяемого выражения в диапазон, задаваемый пограничными выражениями, соединяемыми служебным

словом **AND**. Естественно, как и для предиката сравнения, выражения в предикате **BETWEEN** должны быть совместимы по типам.

Предикат exp1 **BETWEEN** exp2 **AND** exp3 равносилен предикату exp1 **>=** exp2 **AND** exp1 **<=** exp3

А предикат exp1 **NOT BETWEEN** exp2 **AND** exp3 равносилен предикату **NOT (**exp1 **BETWEEN** exp2 **AND** exp3**)**

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

exp1 **>=** exp2 **AND** exp1 **<=** exp3

а второй как:

exp1 **>=** exp3 **AND** exp1 **<=** exp2

Предикат IN

Синтаксис:

IN::=

<Пр**о**веряе**мо**е выраже**н**ие**> [NOT] IN (<по**д**за**п**р**о**с**>)**

| (<выраже**н**ие д**л**я вычис**л**е**н**ия з**н**аче**н**ия**>,...)**

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

значение **FALSE**. Когда не соблюдается ни одно из упомянутых выше условий, значение предиката равно **UNKNOWN**.

Предикат LIKE

Синтаксис:

LIKE::=

<Выраже**н**ие д**л**я вычис**л**е**н**ия з**н**аче**н**ия стр**о**к**и>**

[NOT] LIKE <Выраже**н**ие д**л**я вычис**л**е**н**ия з**н**аче**н**ия стр**о**к**и>**

[ESCAPE <си**м**в**ол>]**

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

- символ подчеркивания (_), который можно применять вместо любого единичного символа в проверяемом значении;
- символ процента (%) заменяет последовательность любых символов (число символов в последовательности может быть от 0 и более) в проверяемом значении.

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

Образец	**Описание**
'abc%'	Любые строки, которые начинаются с букв «abc»
'abc_'	Строки длиной строго 4 символа, причем первыми символами строки должны быть «abc»
'%z'	Любая последовательность символов, которая обязательно заканчивается символом «z»
'%Rostov%'	Любая последовательность символов, содержащая слово «Rostov» в любой позиции строки

'% % %'	Текст, содержащий не менее 2-х пробелов, например, "World Wide Web"

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

SELECT *

FROM Ships

WHERE class LIKE '%o' **;**

Результатом выполнения запроса будет следующая таблица:

name	**class**	**launched**
Haruna	Kongo	1916
Hiei	Kongo	1914
Kirishima	Kongo	1915
Kongo	Kongo	1913
Musashi	Yamato	1942
Yamato	Yamato	1941

Если искомая строка содержит трафаретный символ, то следует задать управляющий символ в предложении **ESCAPE**. Этот управляющий символ должен использоваться в образце перед трафаретным символом, сообщая о том, что последний следует трактовать как обычный символ. Например, если в некотором поле следует отыскать все значения, содержащие символ «_», то шаблон ‘%_%’ приведет к тому, что будут возвращены все записи из таблицы. В данном случае шаблон следует записать следующим образом:

'%#_%' **ESCAPE** '#'

Для проверки значения на соответствие строке «25%» можно воспользоваться таким предикатом:

LIKE '25|%' **ESCAPE** '|'

Истинностное значение предиката **LIKE** присваивается в соответствии со следующими правилами:

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

- в противном случае, если проверяемое значение и образец имеют нулевую длину, истинностное значение равно **TRUE**;
- в противном случае, если проверяемое значение соответствует шаблону, то предикат **LIKE** равен **TRUE**;
- если не соблюдается ни одно из перечисленных выше условий, предикат **LIKE** равен **FALSE**.

Предикат LIKE и регулярные выражения

Предикат **LIKE** в его стандартной редакции не поддерживает регулярных выражений, хотя ряд реализаций (в частности, Oracle) допускает их использование, расширяя возможности стандарта.

В SQL Server 2005/2008 использование регулярных выражений возможно через CLR, т.е. посредством языков Visual Studio, которые могут использоваться для написания хранимых процедур и функций.

Однако в Transact-SQL, помимо стандартных символов-шаблонов ("%" и "_"), существует еще пара символов, которые делают этот предикат LIKE более гибким инструментом. Этими символами являются:

- **[ ]** - одиночный символ из набора символов (например, [zxy]) или диапазона ([a-z]), указанных в квадратных скобках. При этом можно перечислить сразу несколько диапазонов (например, [0-9a-z]);
- **^** - который в сочетании с квадратными скобками исключает из поискового образца символы из набора или диапазона.

Поясним использование этих символов на примерах.

```
SELECT * FROM
(
SELECT '5%' name    UNION ALL
SELECT '55'         UNION ALL
SELECT '5%%'        UNION ALL
```

```
SELECT '3%%'          UNION ALL
SELECT 'a5%%'         UNION ALL
SELECT 'abc'          UNION ALL
SELECT 'abc 5% cde' UNION ALL
SELECT '5c2e'         UNION ALL
SELECT 'C2H5OH'      UNION ALL
SELECT 'C25OH'        UNION ALL
SELECT 'C54OH'
) x
/* 1 */
--WHERE name LIKE'5%' -- начинается с 5
/* 2 */
--WHERE name LIKE '5[%]' -- 5%
/* 3 */
--WHERE name LIKE '5|%' ESCAPE '|'-- 5%
/* 4 */
--WHERE name LIKE '%5|%%' ESCAPE '|' -- 5% в любом месте строки
/* 5 */
--WHERE name LIKE '[0-9][a-zA-Z]%' -- первая цифра, вторая буква
/* 6 */
--WHERE name LIKE '[a-z][0-9]%' -- первая буква, вторая цифра
/* 7 */
--WHERE name LIKE '[^0-9]%' -- начинается не на цифру.
/* 8 */
--WHERE name LIKE '%[02468]%' -- содержит четную цифру.
/* 9 */
--WHERE name LIKE '%[02468][13579]%' -- комбинация четная-нечетная.
```

В данном запросе генерируются некоторые данные, для поиска в которых используется предикат LIKE. Девять примеров - это соответственно 9 закомментированных предложений WHERE. Для проверки результатов выполнения запроса на сайте предварительно снимите комментарий с одной из строк, начинающихся с WHERE. Тем, кто не может использовать сайт, приведу здесь результаты выполнения этих примеров.

1. Все строки, которые начинаются с 5:

name
5%
55
5%%
5c2e

2. Поиск строки '5%'. Символ в скобках воспринимается как обычный единичный символ:

name
5%

3. Другое решение, аналогичное второму, но использующее ESCAPE-символ, указывающий, что знак % следует воспринимать как обычный символ.

4. Поиск подстроки '5%', находящейся в любом месте строки:

name
5%
5%%
a5%%
abc 5% cde

5. Поиск строки, у которой первый символ является цифрой, а второй - буквой:

name
5c2e

6. Поиск строки, у которой первый символ является буквой, а второй - цифрой. Вариант для регистронезависимого сравнения:

name
a5%%
C2H5OH
C25OH
C54OH

7. Поиск строки, которая начинается не с цифры:

name
a5%%
abc
abc 5% cde
C2H5OH
C25OH
C54OH

8. Поиск строки, которая содержит четную цифру:

name
5c2e
C2H5OH
C25OH

C54OH

9. Поиск строки, которая содержит подряд идущие четную и нечетную цифры:

name

C25OH

Использование значения NULL в условиях поиска

Предикат:

IS [NOT] NULL

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

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

SELECT *

FROM PC

WHERE price **IS NULL;**

Характерной ошибкой является написание предиката в виде:

WHERE price **= NULL**

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

Аналогичной, но не такой очевидной ошибкой является сравнение с **NULL** в предложении **CASE**. Чтобы продемонстрировать эту ошибку, рассмотрим такую задачу: «Определить год спуска на воду кораблей из таблицы Outcomes. Если последний неизвестен, указать 1900».

Поскольку год спуска на воду (launched) находится в таблице Ships, нужно выполнить левое соединение:

SELECT ship**,** launched

FROM Outcomes o **LEFT JOIN**

Ships s **ON** o**.**ship **=** s**.**name**;**

Для кораблей, отсутствующих в Ships, столбец launched будет содержать **NULL**-значение. Теперь попробуем заменить это значение значением 1900 с помощью оператора **CASE**

SELECT ship**, CASE** launched

WHEN NULL

THEN 1900

ELSE launched

END 'year'

FROM Outcomes o **LEFT JOIN**

Ships s **ON** o**.**ship**=**s**.**name**;**

Однако ничего не изменилось. Почему? Потому что использованный оператор **CASE** эквивалентен следующему:

CASE

WHEN launched **= NULL**

THEN 1900

ELSE launched

END 'year'

А здесь мы получаем сравнение с **NULL**-значением, и в результате — **UNKNOWN**, что приводит к использованию ветви **ELSE**, и все остается, как и было. Правильным будет следующее написание:

CASE

WHEN launched **IS NULL THEN** 1900

ELSE launched

END 'year'

то есть проверка именно на присутствие **NULL**-значения.